
الشامل بلغة JAVA
أساسيات الجافا
البرمجة الموجهة OOP
هياكل البيانات

السيرة الذاتية (cv)

المعلومات الشخصية (Personal Information):

الاسم / عمار بن محمد عيسى الدبعي
الجنسية / يمني
محل و تاريخ الميلاد / الحديدة **1986**م

العنوان الدائم وبيانات الاتصال (Connection & Address):

جدة ، شارع الميناء ، موبايل **(00966538558880)** ، أيميل (**xpres@xpres-it.net**) موقع البرمجيات **www.xpres-it.net**.

المؤهلات العلمية (Education):

المؤهل والتخصص	سنة التخرج	الجهة المانحة
بك علوم وهندسة الحاسوب قسم / علوم الحاسب تقدير عام : جـ//	**2004-2008**م	كلية علوم وهندسة الحاسوب – جامعة الحديدة
دبلوم سكرتارية تقدير امتياز	**2002-2004**م	المعهد البريطاني
ثانوية عامة قسم علمي معدل **80,50**	**2001**م	مدرسة الثورة – الحديدة

المؤلفات العلمية المنشورة (Researches):

§ تم تأليف الكتب التالية:

* أصول البرمجة بلغة **c++** .
* هياكل البيانات بلغة **c++** .
* البرمجة الموجهة بلغة **Java** .
* **101** لإتقان البرمجة.
* هياكل البيانات بلغة **Java** .

وتم نشرها على شبكة الإنترنت **www.cb4a.com** .

الوظيفة الحالية:

§ مبرمج بمجموعة شركات محمود سعيد.
§ مدير مكتب برمجيات **XPres** في اليمن **www.xpres-it.net**.

الوظائف السابقة :

- أستاذ جامعي في جامعة الحديدة – قسم معلم حاسوب.
- ضابط حجز ومبيعات (الجزيرة العربية للسفريات والسياحة-**2002/2000**).
- مراجع حسابات بمؤسسة نجمة الهدف التجارية.

شهادات الشكر والتقدير:

- شهادة من معرض ابتكار **2013** الرياض.
- شهادة من شركة السحاب لخدمات شركة مايكرسوفت جدة.
- شهادة شكر وتقدير من جامعة البترا - عمان الأردن **2008**.
- شهادة شكر وتقدير من عميد مركز وأنظمة المعلومات –جامعة الحديدة **2008**.
- شهادة شكر وتقدير من مدير مكتب اليمنية للخطوط الجوية –الحديدة **2007**.
- شهادة شكر وتقدير من مؤسسة موانئ البحر الأحمر اليمنية **2009**.
- شهادة شكر وتقدير من رئيس جامعة الحديدة **2010**.
- شهادة شكر وتقدير من دفعة اليمن أولاً قسم معلم حاسوب –كلية التربية زبيد **2009-2010**.
- شهادة شكر وتقدير من دفعة سفراء البرمجة –كلية علوم وهندسة الحاسوب –الحديدة **2009-2010**.

الأعمال التي قمت بها :

- تم المشاركة في تاريخ **2013/12/2** في معرض ابتكار **2013** المقام في الرياض.
- تم المشاركة في تاريخ **2008/7/23** بمعرض تكنولوجيا المعلومات **ITSAF2008** ببرنامج إدارة المختبرات الحاصل على المرتبة السابعة على مستوى الوطن العربي المقام في عمان – الأردن.
- تم تصميم المواقع الإلكترونية(لمؤسسة موانئ البحر الأحمر اليمنية ليضم ميناء (الحديدة – المخاء – الصليف - ميدي) ،منظمة اليمن أولاً،جامعة الحديدة، ملتقى كلية التربية زبيد ،المعهد العالي للعلوم الصحية ، الجزيرة العربية للسفريات والسياحة.
- تم تصميم برنامج جرد المخزون بواسطة أجهزة Handheld وتم تركيبة بمجموعة شركات محمود سعيد.
- تم تصميم الموقع الالكتروني الخاص بالقسم التجاري لشركة محمود سعيد وتم ربطة على **37** فرع في المملكة **OnLineSalPoint**.
- تم تصميم برنامج إعارة الثلاجات للعملاء وتم تركيبة بمصنع المرطبات لشركة محمود سعيد.
- تم تصميم برنامج بركده الأصناف للمستودعات وتم تركيبة بمجموعة شركات محمود سعيد.
- تم تصميم المعونات الخيرية وتم تركيبة بمجموعة شركات محمود سعيد.
- تم تصميم برنامج تقارير **KPR** وتم تركيبة بمصنع المرطبات لشركة محمود سعيد.
- تم تصميم برنامج المحاسبي لمبروك أخوان للصناعات ودباغة الجلود .
- تم تصميم برنامج ضبط الحضور والانصراف لمجموعة شركات محمود سعيد.
- تم تصميم برنامج Gantt Chart Machine وتم تركيبة بمجموعة شركات محمود سعيد.
- تم تصميم برنامج Present Tracing, Pallets وتم تركيبة بمجموعة شركات محمود سعيد.
- تم تصميم برنامج Received Voucher. وتم تركيبة بمجموعة شركات محمود سعيد.
- تم تصميم برنامج المخازن للمعهد العالي للعلوم الصحية.
- تم تصميم برنامج الكنترول للمعهد العالي للعلوم الصحية.
- تم تصميم برنامج لإصدار بطائق دخول الميناء وتم تركيبة لمؤسسة موانئ البحر الأحمر اليمنية.
- تم تصميم برنامج للتذاكر المجانية للخطوط الجوية اليمنية فرع الحديدة.
- تم تصميم برنامج إدارة مقاهي الإنترنت وتم تسويقه على أكثر من عشرة مقاهي.
- تم تركيب شبكات لأكثر من عشرة أماكن مختلفة.
- تم تصميم برنامج لإدارة العيادات وتم تركيبة على (عيادة العمال لمؤسسة موانئ البحر الأحمر اليمنية-عيادة الدكتور عبد القادر العبسي).
- تم تصميم برنامج إدارة المختبرات للمستشفى العسكري فرع الحديدة. بشبكة لاسلكية.
- تم تصميم برنامج إدارة المختبرات وتم تركيب البرنامج على عدة مستشفيات (مستشفى الكويت التعاوني -مستشفى الحديدة التخصصي - مستشفى الثورة العام-- مستشفى الأمل التخصصي – المستشفى العسكري بالحديدة - مستشفى المختار- مختبر الغانم – مستوصف نجاه حجر- مختبر الفا الدولي بذمار- الهاشم بصنعاء –والعديد).

- تم تصميم برنامج لإدارة الصيدلية وتم تركيبة صيدلية (سهام – الجميل – مستشفى الشفاء – مستوصف نجاة حجر - والعديد).
- تم تصميم البرنامج المحاسبي وتم تركيبة (مؤسسة الصادق الطبية – المحيط للمستلزمات الطبية – مثلث عاهم للمستلزمات الطبية بحرض – مطعم الشرق الأوسط – محلات الأزرق للتجارة و الاستيراد - والعديد).
- تم تصميم برنامج لإدارة مستوصف وتم تركيبة لمستوصف (د. نجاه حجر – مستوصف تهامة الطبي بحرض).
- والعديد...

البرمجة وتحليل النظم:

- برمجة أجهزة **Handheld** بنظام تشغيل ويندوز موبايلي (جيد جدا).
- البرمجة بلغة **ORACEL**.(جيد جداً)
- البرمجة بلغة فيجيوال بيسك **6** و **NET.** (ممتاز).
  - البرمجة باستخدام دوال **API**.
  - برمجة الملتميديا.
  - برمجة الشبكات.
  - برمجة الدوال والخوارزميات.
  - برمجة قواعد البيانات (**ADO**)
- البرمجة بلغة **ASP .NET** (ممتاز)
- البرمجة بلغة **SQL** وتصميم وتنفيذ قواعد البيانات (ممتاز) .
- البرمجة بلغة **HTML, PHP, JAVA SCRIPT** (ممتاز).
- البرمجة بلغة التجميع (جيد).
- البرمجة بلغة جافا, بيسك (امتياز).
- البرمجة بلغة سي و سي ++ (امتياز).
- تحليل وتصميم النظم (جيد جيداً).
- سرعة تعلم لغات البرمجة وسرعة في اكتساب الخبرة فيها.

المهارات (**Skills**):

- سرعه في الإدراك وتنويع مصادر التعلم والتكيف مع التطوير.
- الإخلاص في العمل والأمانة في الأداء.
- الولاء للمؤسسة التي أعمل فيها مع عدم نشر أسرارها.
- سريع الاندماج مع فريق العمل.
- أستطيع العمل في الظروف الضاغطة.
- أحب التطور ، دقيق في الوقت ، هادئ المزاج ، صبور ، متعاون ، مبتهج ، حسن الخلق ، معتمد على الذات.

بسم الله الرحمن الرحيم

المقدمة INTRODUCTION

الحمد لله رب العالمين حمداً يوافي نعمة ويكافئ مزيدة الحمد لله الذي خلق الظلمات والنور وصلى الله على سيد الخلق معلم الناس الخير حبيبنا وشفيعنا محمد بن عبد الله النبي الأمي الذي بذا برسالته ب(اقرأ باسم ربك الذي خلق الإنسان من علق إقراء وربك الأكرم الذي علم بالقلم) وختمها ب(اليوم أكملت لكم دينكم وأتممت عليكم نعمتي ورضيت لكم الإسلام دينا) صدق الله العظيم وبلغ رسوله الكريم ونحن على ذلك شاهدين من اليوم إلى يوم الدين، أما بعد

نعيش اليوم قي عصر تتسارع به العلوم والتكنولوجيا ، وتتسارع معها العلوم والمعرفة الإنساني ة بشكل لم يسبق له مثيل في تاريخ الإنساني ة وتتضاعف المعرفة حاليا بشكل لوغارتمي مع مرور كل عقد من الزمن وأصبحت المعلومات تتسع والبحوث والدراسات تتعمق وتصبح أكثر تعقيداً في عصر توصل العقل البشري فيه إلى تسخير إمكانات الآلة ضمن نظام محكم وبرمجة فاعلة لتأمين خدمات نموه وتـ أور حياته نحو الأفضل.
ومع هذا التطور والنمو في حياة الإنسان بكل أوجهها برز الحاسب كأهم مؤشر لها النمو، بل أصبح الحاسب مقياساً لسمة عصر العلوم والتكنولوجيا وموجها لتقدم وحضارة الأمة، ولا يمكن لأمة من الأمم في عصرنا الحالي اللحاق بركب الحضارة التكنولوجية وتأمين حياة أفضل لشعوبها دون السيطرة والتحكم في هندسة الحاسب وعلوم الحاسب الالكتروني.

لمواجهة ذلك رأينا ضرورة تأليف هذا الكتاب فلقد أنشأنا بسبب قلة كتب هذه المادة بالغة العربية و تيسيراً لإخواننا الذين صعبت علي هم فهم هذه المادة وتبسيط لهم معنى هذه المادة والاستفادة الكاملة من الإمكانات التي توفرها لغة Java واعلموا أن كل ما كتب هو خلاصة ولب البرمجة بلغة Java وقد تميز هذا الكتاب

1. سهولة التعبير باللغة العربية.
2. إثبات المصطلحات الانجليزية حيثما لزم لتكون عونا للقارئ عند الحاجة للإطلاع على كتب أجنبية في مجال الكتاب.
3. أتباع الشمولية في أعداد محتوى الكتاب والتوضيح اللازم
4. الإكثار من الأمثلة المتنوعة المحلولة لما لها من أهمية في توضيح معاني المفاهيم التي تضمنها الكتاب الأمر الذي يؤدي استيعاب الـ موضوع استيعاب شاملاً.

فيتناول هذا الكتاب موضوعات متعددة لوصف

1. أساسيات اللغة.
2. البرمجة الموجهة بلغة OOP In Java .

3. هياكل البيانات.

ولقد وثقت هذه المواضيع ببرامج علمية طبقت جميعها للتأكد من صحتها وأيضاً وثقت بالرسوم البيانية لترسيخ الفكر في ذهن القارئ.

وأخير أ نسأل الله أن يحقق هذا الكتاب الهدف الذي كتب لأجله ويعلم الله أن غايتنا في هذا أن يعم الفائدة في أرض المسلمين وكل مسلم ومسلمة طالباً منكم دعوه صالحة في ظهر الغيب وان تصلوا وتسلموا على من علمنا وأنبانا نبينا محمد حبيب قلبنا ألف مليون صلاة وسلام من رب العبـ ـاد عدد تحرير السطور وعدد المخلوقات والمخلوق صلاه دائمة من اليوم إلى يوم النشور .

واسأل الله أن يبارك لنا ولكم في كل ما كتبناه و تعلمناه وتعلمتموه .

والحمد لله .....

لماذا هذا الكتاب ؟

يتميز هذا الكتاب عن غيرة من الكتب التي تتخذ من لغة Java أساساً لها في أنة يتمتع بالشمولية والتكاملية ، إذ ينذر أن تجد كتاباً يغطي معظم مواضيع اللغة ، أضف إلى ذلك أنة يتبع الطريقة العلمية في طرح للمواضيع وبشكل متكامل ، فهو لا يعتمد اعتماداً كبيراً على الشروح النظرية وإنما يركز كثيرا على شرح مميزات اللغة من خلال البرامج العملية بحيث يخرج القارئ بفائدتين : الأولى انه تعلم الفكرة ومارس تطبيقها العملي مباشرة ، والثانية انه استفاد من التقنية البرمجية المستخدمة في المثال.

يحتوي هذا الكتاب على ____ برنامجاً ، وسوف تجد هذه الأمثلة قيمة جداً عندما تدرس مكوناتها بشكل جيد، فقد شرحت معظم الإجراءات والتوابع و العبارات المكونة للبرامج بالتفصيل ، إضافة إلى ذلك أ، الكتاب يطرح العديد من المقترحات والأفكار والأخطاء الشائعة والعادات البرمجية الجيدة التي يمكن من خلالها زيادة مدى الاستفادة من الأمثلة .

وبعد أن تنتهي من قراءة الكتاب سوف تكون قد تعلمت جميع الصيغ الكتابية لهذت اللغة وكذلك قد امتلكت الخبرة الكافية ودرست الطرائق الصحيحة لكتابة البرامج التي تريد.

كيف نظم هذا الكتاب

لقد نظم هذا الكتاب وفق طريقة سهله المتابعة، فهو لا يفترض وجود خبرة برمجية مسبقة لدى القارئ ومع ذلك فهو يطرح مواضيع متقدمة تجعل من الكتاب حاجة للمبتدئين والمتقدمين.

ولقد رتبت مواضيع وفصول الكتاب لكي تسير بالقارئ بخطى ثابتة باتجاه تعلم لغة Java وبنا خبرات برمجية تعينه على كتابة برامجه لاحقاً.

الأهداف

يبدأ كل فصل بمجموعة من الأهداف التي تخبر القارئ عما يجب أن يعرفه في هذا الفصل كما إنها تعطيه الفرصة بعد قراءة الفصل بأن يحدد بنفسه فيما إذا كان قد توصل فعليا عن هذه الأهداف أم لا. يساعد تحديد هذه الأهداف على بناء الثقة الـ ذاتية للقارئ وتشكل له مصدرا إيجابيا للتقوية .

لمن هذا الكتاب

لقد اعد هذا الكتاب ليلبي حاجات فئات القراء الثلاث ومعلى كافة مستوياتهم :

1- فالمبتدئون الذين ليس لهم سابق عهد في البرمجة ، والذين يرغبون بالبدء بتعلم هذه اللغة ، فيتيح لهم هذا الكتاب فرصة تعلم قواعد البرمجة الصحيحة ، وتعلم هذه اللغة بالشكل الأمثل .

2- أما من لهم سابق في عهد البرمجة ولكن ليس لهم اطلاع على هذه اللغة فيأتي هذا الكتاب دليلاً متكاملاً يستطيعون المتابعة من خلاله بخطوات سريعة ، فتتشكل لديهم أرضية برمجية جيدة من خلال هذا الكتاب.

3- ولمن لهم سـ ابق عهد بلغة Java يقدم هذا الكتاب لهم من الأفكار والمواضيع المتقدمة ما ينذر أن يجدوه في أي كتاب ، وسيشعرون بالحاجة إلى الاحتفاظ بنسخة الكتاب قريبة منهم عند كتابة برامجهم.

الطريقة المثلى في قراءة الكتاب

يعتمد الكتاب في طرحه للمواضيع على عدة اعتبارات ينبغي توفرها عند القارئ لكي يستفيد استفادة مُثلى من الكتاب. إذ ينبغي أولاً قراءة المقدمة النظرية عن الموضوع ، ومن ثم تجريب المثال على الحاسب ضمن بيئة الJava ، وذلك من خلال قيم الدخل المقترحة ومراقبة الخرج الناتج ومن تم تتبع شرح البرنامج ضمن الكتاب .
ثانياً ينبغي الاستجابة للتوصيات المطروحة وإجراء التعديلات المقترحة وتجنب الأخطاء الشائعة .

القرص المرفق

لقد أرفقنا الكتاب بقرص يحوي ما يلي:
1- برامج الأمثلة المذكورة في الكتاب كاملة، ووضعت البرامج في أدلة تشير إلى رقم الفصل الحاوي للمثال + رقم المثال في ذلك الفصل ، فالملف chp1_1 يشير إلى الفصل الأول المثال الأول .

ختاماً

نرجو من القارئ الكريم أن لا يبخل علينا بدعوة صالحة في ظهر الغيب ونصائحه ومقترحاته ، فنحن نرحب بكل مقترح وتساؤل سعياً إلى تحقيق التواصل الأمثل .
وبعد أتمنى من الله نكون قد وفقنا في ما قمنا بة من عمل ، وأن يتقبله منا إنه سميع عليم.
المؤلفون

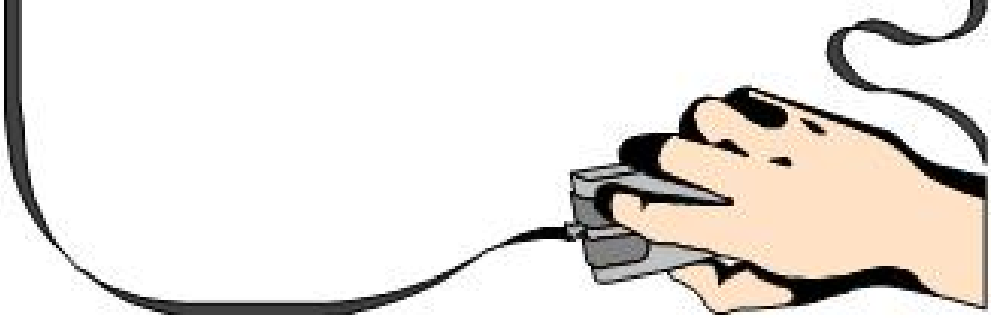
الفهرس

Chp1

مقدمة حول الحاسب الآلي والبرمجة:

<u>في نهاية هذا الفصل سوف تتعلم :</u>

- **مقدمة حول الحاسب.**
- **مقدمة على البرمجة.**
- **لغات البرمجة.**
- **خطوات حل المسألة.**



1.1 مقدمة

INTRODUCTION

الكمبيوتر أو الحاسب أو الحاسوب ترجمة حرفية للكلمة الانجليزية COMPUTER ، وقد شاع استخدام الكلمة الانجليزية التي اشتُقت من الفعل COMPUTE أي حَسَبَ...

تطلق كلمة الحاسب أو الكمبيوتر على كافة الأحجام و الأنواع من الحاسبات الآلية سواء أكان استعمالها للغرض الشخصي أو المنزلي أو أن يكون للاستخدام في مؤسسة أو شركة، أو أن يستخدم لأغراض بعينها في الصناعات الهندسية والطبية والفضائية وغيرها .

لقد تغلغل الكمبيوتر في مختلف نواحي الحياة ، حتى بات من نافلة القول أن الأمية لم تعد أمية القراءة والكتابة ، بل صار الجهل باستخدام الكمبيوتر هو الأمية الحقيقية في هذا العصر.

وهذا المدخل لدراسة أساسيات علوم الحاسب الآلي يلقي الضوء والشرح على عدد من الموضوعات التي تغطي الخلفية التاريخية مع شرح العلاقة بين كيفية عمل الكمبيوتر ومحاكاته لكيفية تشغيل الجسم البشري ، شرح المكونات الشيئية للكمبيوتر من وحدات للإدخال ووحدات للإخراج ، ووحدة للمعالجة المركزية ، وكيفية تخزين المعلومات والبيانات وتمثيل الأرقام والحروف داخل ذاكرة الكمبيوتر.

1.2 خصائص الحاسوب

PROPERTIES COMPUTER

1. سرعة إنجاز العمليات.
2. سرعة دخول البيانات و استرجاع المعلومات .
3. القدرة على تخزين المعلومات .
4. دقة النتائج و التي تتوقف أيضا على دقة المعلومات المدخلة للحاسوب .
5. تقليص دور العنصر البشري خاصة في المصانع التي تعمل آليا .
6. سرعة إجراء العمليات الحسابية و المنطقية المتشابكة .
7. إمكانية عمل الحاسوب و بشكل متواصل دون تعب .
8. تعدد البرمجيات و البرامج الجاهزة والتي تسهل استخدام الحاسوب دون الحاجة إلى دراسة علم الحاسوب و هندسة الحاسوب .
9. إمكانية اتخاذ القرارات وذلك بالبحث عن كافة الحلول لمسألة معينة و أن يقدم أفضلها وفقا للشروط الموضوعة والمتطلبات الخاصة بالمسألة المطروحة .
10. قابلية الربط و الاتصال من خلال شبكات الحاسوب حيث يمكن ربط أكثر من جهاز مع إمكانية التحاور ونقل البيانات والمعلومات فيما بينها .

1.3 تصنيف الحاسبات الالكترونية:

تصنف الحاسبات الالكترونية حسب :

1.3.1 من حيث قدرتها على التخزين و كفاءتها في إنجاز المهام:

وذلك عن طريق زيادة حجم الذاكرة التي تؤدي إلى زيادة سرعة وكفاءة الحاسوب في إنجاز العمل.

- الحاسوب الضخم (Super Computer) :

يعتبر الحاسوب الضخم أو العملاق من أكثر الحواسيب قوة و تستخدم الحواسيب العملاقة في المسائل التي تحتاج إلى عمليات حسابية معقدة جداً و تستعمل هذه الحواسيب في الجامعات، المؤسسات الحكومية و إدارة الأعمال الضخمة .

الشكل 1-1 الحاسوب العملاق

- الحاسوب الكبير(Main Frame Computers) :

يستطيع الحاسوب الكبير دعم ومساندة المئات أو الآلاف من المستخدمين بحيث يعالج الكثير من عمليات الإدخال و الإخراج و التخزين من المستخدمين لمعالجة البيانات، و يستخدم الحاسوب الكبير في الشركات الضخمة و المنظمات الكبيرة التي تضم الكثير من المستخدمين الذين يحتاجون إلى المشاركة في البيانات و البرامج .

الشكل 2-1 لحاسوب كبير

- الحاسوب المتوسط (Minicomputer) :

الحاسوب المتوسط أصغر من الحاسوب الكبير و لكنه أكبر من الحاسوب الصغير و يستعمل كمزود خدمة للشبكات و الانترنت Network servers, Internet servers.

الشكل 3-1 لحاسوب المتوسط

- الحاسوب الصغير (Microcomputer) :

من الشائع عن الكمبيوتر الصغير أنه الحاسوب الشخصي Personal Computer والذي يطلق عليه "PC"، و تندرج في إطار الحاسوب الشخصي الحواسيب المحمول (laptop computers (Notebook بحيث يستطيع المستخدمين حمله بكل سهولة و الإستفاده منه مثل PC.

الشكل 4-1 لحاسوب محمول

الشكل 5-1 لحاسوب صغير

1.3.2 من حيث طريقة العمل :

- الحاسبات الرقمية (Digital Computers):

هي أجهزة الكترونية تقوم بمعالجة البيانات المتقطعة و إجراء الحسابات باستعمال الأعداد ممثلة بصورة مباشرة بشكل رقمي وبسرعة فائقة، حيث يتم تمثيل قيم المتغيرات و الكميات بواسطة الأعداد (بالنظام الثنائي غالباً). وهذا النوع الأكثر شيوعاً و الأكثر دقة ويمكن برمجته واستخدام ه في كافة المجالات .

- الحاسبات التناظرية (Analogue Computers):

هي أجهزة الكترونية تعمل على أساس الموجات، ويختص بقياس التدفق المستمر للبيانات التي يمكن التعبير عنها في صورة كميات مادية مثل الضغط الجوي و درجة الحرارة و الجهد الكهربائي ويستخدم هذا النوع في المجالات العلمية و الهندسية ويعطي نتائج تقريبية .

- الحاسبات المهجنة (Hybrid Computers):
- وهي حواسب تجمع بين خواص النوعين السابقين (الرقمي و التناظري) وتستخدم في المجالات العلمية ، حيث أن الحاجة إلى معالجة بيانات من النوعين ضروري . ومن مميزات هذا النوع طريقة المعالجة الرقمية ، و القدرة على تخزين البيانات ، و الدقة المتناهية، و توليد الاقترانات الرياضية .

ومن مساوئ هذا النوع التكلفة العالية ،و الأخطاء الممكن حدوثها، و البرمجة المتداخلة .

1.3.3 من حيث طبيعة أغراض الاستعمال :

- حاسبات الأغراض العامة (General Purpose Computers):

يصمم هذا النوع من الحاسبات لأغراض متعددة، مثل تنظيم أجور و رواتب العمال و الموظفين، وتنظيم عمليات الخزن في المصانع و المؤسسات و تحليل المبيعات ، حيث تمتلك المرونة الكافية لتأمين الكفاءة في المجالات التجارية و العلمية والطبية والهندسية .

- حاسبات خاصة الاستعمال (Special Purpose Computers):

يصمم من أجل أداء وظيفة محددة، مثل أجهزة الآذ ذار المبكر و أجهزة الحاسوب المستخدمة في العمليات الصناعية وعادة ما تكون الحاسبات من النوع الحاسوب الصغير أو الحاسوب المتوسط .

1.4 تطور الحاسوب:

ارتكزت عملية تطوير الحواسيب على العناصر الأساسية التالية :

1. زيادة سرعة الحاسوب .
2. التقليل من حجم الحاسوب.
3. التقليل من تكلفة الحاسوب.
4. زيادة دقة النتائج .
5. زيادة القدرة التخزينية
6. تسهيل عملية الاستخدام والتشغيل.

1.5 أجيال الحاسب

Computer Generations

1.5.1 الجيل الأول (First Generation):

- بدأت حواسيب هذا الجيل في الظهور من الأربعينيات إلى منتصف الخمسينيات من القرن العشرين.
- الاعتماد على تكنولوجيا الصمامات المفرغة Vacuum tubes في بناء الدوائر المنطقية و دوائر الكترونية شبيهة بتلك المستخدمة في أجهزة الراديو في ذلك الوقت .
- استخدمت خطوط التأخير الزئبقية في بناء الذاكرة ، وفي نهاية هذا الجيل تم استخدام الحلقات المغناطيسية في بناء ذاكرة هذا الجيل .

- البطء النسبي ، وسرعة المتدنية نظراً لتدني سرعة الصمامات .

- كان حجم جهاز الكمبيوتر كبيراً ، بالإضافة إلى حاجة الجهاز إلى أجهزة التبريد نظراً لارتفاع درجة حرارة الصمامات .
- سعة الذاكرة متواضعة للغاية با لنسبة لحجم الأجهزة و بالنسبة للأجيال اللاحقة .

- الاعتماد على لغة الآلة Machine Language في برمجتها ، مما أدى إلى صعوبة التعامل مع الحاسوب و تشغيله.
- استخدمت البطاقات الورقية المثقبة لتخزين البيانات والتي طورت فيما بعد إلى الأشرطة المغناطيسية و الطبول المغناطيسية drums .
- كان أول حاسبات هذا الجيل هو الحاسب المسمى ENIAC تبعه EDVAC ثم EDSAC و أخيراً الحاسب المسمى UNIVAC.

1.5.2 الجيل الثاني (Generation Second):

- بدأت حواسيب هذا الجيل في الظهور من منتصف الخمسينيات إلى بداية الستينيات من القرن العشرين.
- الاعتماد على تكنولوجيا الترانزستور Transistor و دوائره التي تتميز بصغر الحجم و كفاءة التشغيل مما أدى إلى تصغير حجم الحاسب بدرجة ملحوظة و زيادة سرعة الحاسوب نظراً لما يمتاز به الترانزستور عن الصمام .

- استخدام الحلقات المغناطيسية في تركيب الذاكرة وقد ظهرت الأقراص المغناطيسية الصلبة Hard disk حيث استخدمت لتخزين البيانات من أجل الرجوع إليها لاحقاً .
- استحداث لغات برمجة جديدة ذات المستوى العالي(مثل لغة فورتران) التي يمكن باستخدام ها تسهيل التعامل البشري مع الحاسب وبرمجته.

1.5.3 الجيل الثالث (Generation Third):

- بدأت حواسيب هذا الجيل في الظهور من فترة الستينيات من القرن العشرين .
- الاعتماد على تكنولوجيا الدوائر المتكاملة صغيرة المجال Small Scale Integrated و تبعتها الدوائر المتكاملة المتوسطة Medium Scale Integrated مما أدى إلى تصغير الحجم بدرجة كبيرة مع زيادة هائلة في سعة الذاكرة و دقة الأداء .
- زيادة سرعة الأداء عن الأجيال السابقة بشكل كبير .
- بدأ ظهور الحاسبات الصغيرةMinicomputer، بالإضافة إلى تعدد المعا لجات Multiprocessors.
- تطورت برامج نظم التشغيلOperating System مما أدى إلى زيادة فاعلية وكفاءة الأداء ومن أمثلتها نظام البرمجة التعددية Multiprogramming .
- ظهور لغات برمجة راقية جديدة مثل لغة Basic و Pascal .
- ظهرت وحدات إدخال و إخراج جديدة مثل أجهزة القراءة الضوئية والشاشات الملونة .

1.5.4 الجيل الرابع (Generation Fourth):

- بدأت حواسيب هذا الجيل في الظهور من فترة السبعينيات و الثمانينيات من القرن العشرين .
- استخدمت أشباه الموصلات في تطوير الدوائر المتكاملة الكبيرة Large Scale Integrated حيث استخدمت في تصنيع دوائر الحاسوب وذاكرته ، وتطورت الدوائر المتكاملة الكبيرة إلى الدوائر المتكاملة الكبيرة جداً Very Large Scale Integrated و التي سميت بالمعالجات الميكروية (الدقيقة) microprocessors.

- ازدادت سرعة أداء حاسبات هذا الـ جيل عن الأجيال السابقة .
- بدأ ظهور الحاسبات المصغرة الشخصية و المنزلية Personal and Home Computers، Microcomputer .
- تم تطوير برامج و نظم التشغيل و انتشرت أنظمة التشغيل اللحظية Real time systems.
- ظهور الأقراص المغناطيسية المرنة .

1.5.5 الجيل الخامس (Generation Fifty):

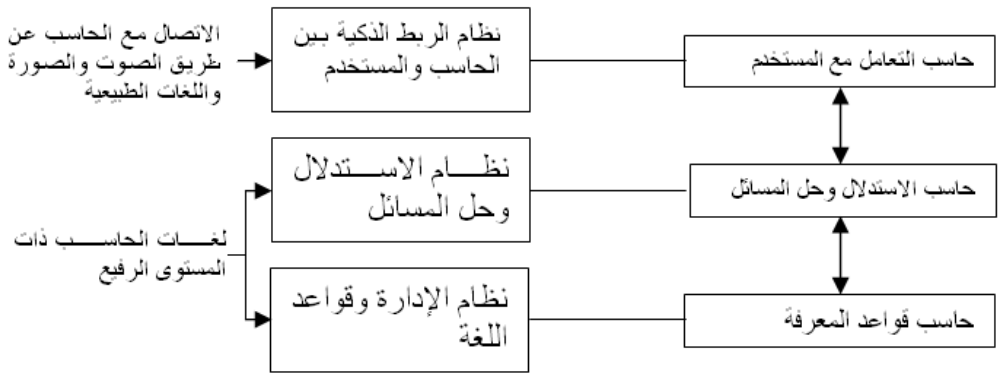
في أكتوبر سنة 1981 عقد مؤتمر في اليابان لمناقشة إنتاج جيل من الحاسبات تستخدم فيه جميع التطورات التي حدثت في مجال الحاسبات وعلى الأخص ما يسمى بالذكاء الصناعي وسمى هذا الجيل بالجيل الخامس . وفي عام 1985 أعلنت اليابان من المرحلة الأولى من جيل الحاسبات الخامس،

ولقد أعتبر برنامج الجيـ ل الخامس من الحاسبات موضوع استخدام وحدات المعالجة التي تعمل على التوازي هي الأساس لتصميم هذا النوع من الحاسبات. وكذلك لتلاءم الكثير من التطبيقات المختلفة وخاصة الغير عددية منها . مثل تطبيقات الذكاء الاصطناعي Artificial Intelligence.

لذلك فقد ظهرت حاجة ماسة إلى التفكير في تغيرات جذرية في تصميم الحاسبات لكي تتمشى مع التطبيقات المستوفية وقد وضع اليابانيون تصوراتهم في جيل جديد من الحاسبات سموه بالجيل الخامس.

ويكون حاسب الجيل ا لخامس أساساً من مجموعة من الحاسبات يتم التنسيق فيما بينها بواسطة نظام تشغيل عام كما هو مبين بشكل 6-1 ومن هذه الحاسبات :

1- حاسب التعامل مع المستخدم.
2- حاسب الاستدلال وحل المسائل.
3- حاسب قواعد المعرفة.



شكل 6-1 حاسبات الجيل الخامس

1.6 الكمبيوتر يحاكي الإنسان ..كيف ؟

جون فون نيومان JOHN VON NEWMAN أول من قام بدراسة التركيب الوظيفي للإنسان عن طريق ملاحظة كيفية حله للمشكلات ، ووجد أن الإنسان لكي يحل مشكلة معينة فإنه يقوم بتوظيف حواسه في جمع عناصر المشكلة ومعلوماتها ، ثم يلي ذلك تخزين هذه ا لمعلومات في الذاكرة ، ثم يقوم العقل بتحليل المشكلة ومن ثم إيجاد الحل المناسب من واقع خبرته التي تعلمها ، وينتهي الأمر باتخاذ قرار معين حيث تصدر الأوامر إلى العضلات المختلفة في عضو من أعضاء للتنفيذ وتقوم الأعصاب بدور الناقل في جميع مراحل هذه العملية.

ومن واقع دراسة نيومان نجد أن الإنسان ينقسم وظيفيا إلى الوحدات الأساسية التالية :

وحدات لإدخال واستقبال المعلومات والبيانات وتتمثل في الحواس الخمس : السمع والبصر والتذوق والشم واللمس ؛ ووحدة لتخزين البيانات تتمثل في الذاكرة ، ووحدة لمعالجة البيانات وهي الـ عقل، ووحدة إخراج وتنفيذ البيانات وتتمثل في العضلات، والناقلات بين الوحدات وهي الأعصاب .

و لما كانت ذاكرة الإنسان عاجزة عن استيعاب المعلومات والبيانات إلى ما لا نهاية فقد استعان الإنسان بوسائط مساعدة يقوم بتخزين المعلومات عليها ثم استدعائها في أي وقت ، وذل ك عن طريق أيٍ من وحدات الإدخال الخاصة به ؛ لذلك فقد اختُرع الكتابُ المقروء والصوتُ المسجَّل والفيديو المرئي وغيرها من وسائل حفظ البيانات والمعلومات .

ولبيان كيفية تمثيل ما سبق شرحه نضرب مثالا بعملية بسيطة : كأن يقوم إنسان بتوجيه سؤال إلى إنسان آخر ولتكن ع ملية ضرب رقم في رقم ، فيقوم الأول بتوجيه السؤال صوتيا ، وتقوم وحدة الإدخال في الثاني وهي الأذن باستقبال السؤال الذي تنقله الأعصاب إلى وحدة المعالجة المركزية وهي العقل الذي يستدعي ما يخزنه في الذاكرة عن كيفية إجراء العمليات الحسابية قسم جدول الضرب ويتولى حـ ساب النتيجة ويبث الناتج عبر الأعصاب إلى عضلات الفم

واللسان الذي يترجمه إلى جواب صوتي يسمعه السائل .
إن ما لاحظه جون فون نيومان هو ما تم تطبيقه عمليا عند تصميم جهاز الكمبيوتر؛ فوحدات إدخال تؤدي للكمبيوتر ما تؤديه الحواس الخمس، فنجد أن هناك عدة أنواع من أجهزة أو وحدات الإدخال ، مثل لوحة المفاتيح KEYBOARD والفأرة (MOUSE) والماسح الضوئي (SCANNER) ، وقلم القراءة الضوئي ، وهكذا .
تماما مثل الإنسان ، يحتاج الكمبيوتر إلى ذاكرة أساسية داخلية MAIN MEMORY ولا يمكن للجهاز أن يعمل بدونها وهي دائمة الاتصال بوحدة المعالجة المركزية ، والتقسيم المنطقي للذاكرة مكون من مجموعة من الحجرات تسع كل منها لثمانية بتّات BITS ، والبت هو أساس العمل في الكمبيوتر وكل ثمانية بتات BITS تمثل BYTE بايت واحد الذي يمثل بدوره حرف هجائي أو رقمي واحد .
والذاكرة الأساسية المرتبط بوحدة المعالجة المركزية محدودة في قدرتها التخزينية ؛ لذلك تم استحداث عدد من وسائل التخزين المساعدة تماما مثل الوضع في حالة الإنسان ، فنجد مثل الوسائط الممغنطة مثل الأسطوانات اللينة (FLOPPY DISKS) ، والأسطوانات الصلبة (HARD DISKS) والشرائط الممغنطة MAGNETIC TAP ، وكذلك الأسطوانات الضوئية OPTICAL DISKS ثم الأسطوانات المليزرة على اختلاف أنواعها .

ويلخص الجدول 1-1 مقارنة بين الحاسب الآلي والعقل البشري للخصائص المختلفة.

جدول 1-1 مقارنة بين الحاسب الآلي والعقل البشري

الخواص	الحاسب الآلي	العقل البشري
القدرة على التفكير	√	X
وضع البيانات اللازمة لحل المشكلة	يحتاج لوضع البيانات	يقوم بوضعها وجمعها
سرعة التشغيل والتداول	تصل إلى جزئ من المليون من الثانية	أقل قدرة
العرض للخطاء	لا يخطئ طالما كانت برمجته سليمة	عرضة للخطاء
الإجهاد	لا يتأثر	قدرة محدودة
إدخال البيانات	وسائل الإدخال المختلفة	العين – الأذن
إخراج البيانات	وسائل الإخراج المختلفة	الكتابة – النطق
الذاكرة	محدودة	غير محدودة

1.7 مكونات نظام الحاسوب الرقمي:

1. وحدة المعالجة المركزي (CPU) Central processing Unit :
   - وحدة الحساب و المنطق (ALU& Arithmetic) logical unit.
   - وحدة التحكم (CU) Control unit .

2. الذاكرة الرئيسية (MM) Main Memory :وترتبط وحدة المعالجة المركزية مع الذاكرة عن طريق مجموعة من النواقل buses تتألف من مجموعة من الخطوط (الأسلاك).

الناقلات Buses:تعرف الناقلات على إنها مجموعة من الأسلاك تربط وحدات الحاسوب المختلفة وذلك لتمرير وتبادل المعلومات بين هذه الوحدات ويمكن تصنيف الناقلات حسب:

1.طريقة نقل البيانات:

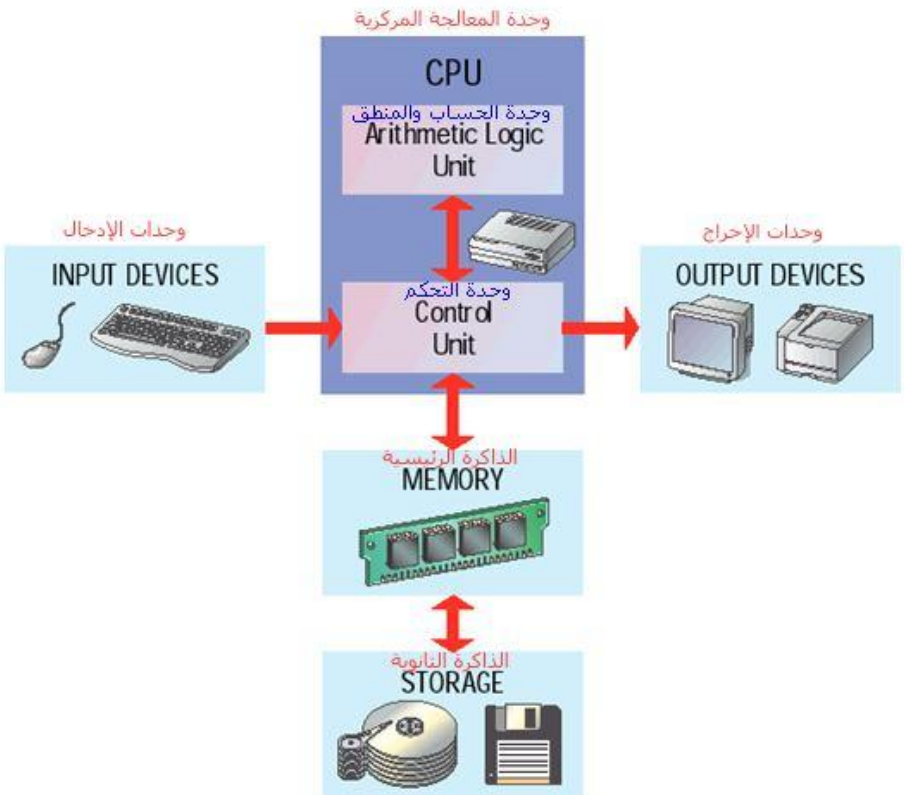
- ناقلات على التوالي: حيث يلزم سلك واحد لنقل البيانات بحيث تنقل بت تلو الآخر.
- ناقلات على التوازي: حيث يلزم عدد من الأسلاك مساو لعدد خانات الكلمة المراد تمريرها بحيث تنقل هذه الخانات دفعة واحدة وبشكل متوازي.

2.حسب طبيعة البيانات:

وتنقسم هذه النواقل إلى ثلاث أنواع:

- ناقل البيانات Data bus : هي مجموعة الخطوط المخصصة لتبادل ونقل المعلومات و البيانات بين وحدة المعالجة المركزية والذاكرة الرئيسية ، وتنقل البيانات في الاتجاهين من وحدة المعالجة المركزية إلى الذاكرة و العكس.
- ناقل العنوان Address bus :هي مجموعة الخطوط المخصصة لنقل العناوين من وحدة المعالجة المركزية إلى الذاكرة وتنقل العناوين باتجاه واحد من وحدة المعالجة المركزية إلى الذاكرة .
- ناقل التحكم Control bus :هي مجموعة الخطوط المخصصة لنقل أشارات التحكم بين وحدة المعالجة المركزية والذاكرة وتنقل أشارات التحكم في الاتجاهين من وحدة المعالجة المركزية إلى الذاكرة و بالعكس .

3. وحدات التداول :
   - وحدات إدخال البيانات والتعليمات Input unit
   - وحدات إخراج المعلومات Output unit
   - وحدات التخزين المساعدة Backing storage



الشكل 7-1 العناصر المادية لنظام الحاسوب

1.7.1 وحدة المعالجة المركزية CPU:

تعتبر وحدة المعالجة المركزية العقل المدبر للحاسوب فهي المسئولة عن تنفيذ كافة العمليات الخاصة بالمعالجة ومنها العمليات الحسابية و المنطقية وترتبط هذه الوحدة بالذاكرة حيث تستقبل منها البيانات و التعليمات الخاصة بالمعالجة.

وتتكون وحدة المعالجة المركزية من وحدتين هما وحدة التحكم ووحدة الحساب والمنطق، وبذلك تؤدي وظيفتين أساسيتين:

- تنفيذ البرنامج المخزن في الذاكرة الرئيسية وفق سياق أوامر و تعليمات البرنامج وضبط المعدات لتؤدي الوظائف المطلوبة.
- إجراء العمليات الحسابية و المنطقية.

أن وحدة المعالجة المركزية عبارة عن مجموعة من المسجلات Registers التي تستخدم في تخزين البيانات المدخلة إلى الدوائر الالكترونية لإجراء العمليات الحسابية ، وما يتم تخزينه في المسجلات فهو تخزين مؤقت حتى تتم معالجة البيانات و تنقل النتائج إلى الذاكرة الرئيسية ، فالتخزين في الذاكرة تخزين مؤجل طويل الأمد نسبياً حتى تستدعي بياناتها ومحتوياتها للمعالجة، بينما التخزين ع لى مسجلات وحدة المعالجة الرئيسية تخزين عاجل للمعالجة فقط.

وتخضع المسجلات للسيطرة المباشرة لوحدة التحكم التي تراقب وصول البيانات إلى المسجلات ، بعدها تعطي وحدة التحكم أوامرها إلى الدوائر الالكترونية الحسابية و المنطقية للعمل ثم تراقب حركة و تخزين البيانات في مسجلات أخرى مخصصة للمخرجات ، وقد تستخدم مسجلات المدخلات في الاحتفاظ بمخرجات العمليات الحسابية .

- **العلاقة بين وحدة المعالجة المركزية و الذاكرة:**

يتم انتقال البيانات بين الذاكرة ووحدة المعالجة المركزية بطبع أو قراءة نسخه من محتوى خلايا التخزين من الذاكرة إلى المسجلات المناسبة في وحدة المعالجة المركزية عبر مجموعة من نواقل البيانات، وعبر نواقل البيانات فإن وحدة المعالجة المركزية تقدر على استخلاص وقراءة البيانات أو إيعازات البرامج من الذاكرة بإرسال إشارة ق راءة من وحدة التحكم عبر نواقل التحكم تشمل إرسال عنوان خلية الذاكرة المطلوبة عبر ناقل العنوان من وحدة المعالجة المركزية إلى الذاكرة، وعلى نفس المنوال يمكن لوحدة المعالجة المركزية كتابة بيانات في خلايا الذاكرة.

1.7.2 وحدة الحساب و المنطق ALU:

تعتبر من أهم مكونات CPU حيث تقوم بتنفيذ كافة العمليات الحسابية و المنطقية وعمليات المقارنة حيث تتألف هذه الوحدة من مجموعة من الدوائر المنطقية.

ومن مكونات هذه الوحدة :

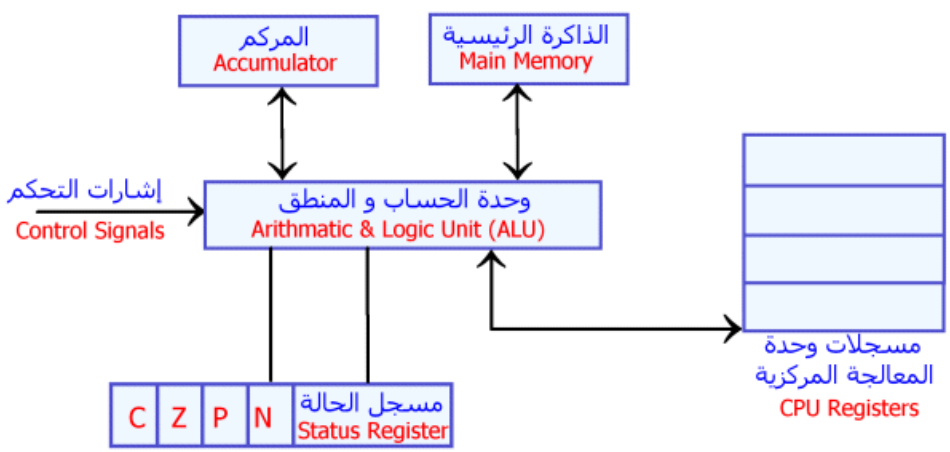
1. دارة الجامع التام الذي يقوم بجمع 3 خانات ثنائية و دارة الجامع النصفي الذي يقوم بجمع خانتين ثنائيتين.
2. دارة العكس Invertors المستخدمة للحصول على المكمل لواحد أو لاثنين للرقم الثنائي.
3. المركم Accumulator وهو مجموعة خلايا ثنائية تسمى المسجل و تستخدم عادة للاحتفاظ بنتائج العمليات المنفذة مؤقتاً لحين نقلها إلى الذاكرة أو إلى وحدات أخرى.

4. مجموعة من الخلايا الثنائية تبين حالة العملية المنفذة و تسمى هذه المجموعة مسجل الحالة Status Register أو Flag Register حيث تخصص كل خلية لمتابعة حالة معينة.

أن وحدة الحساب و المنطق تحتوي على مسجل المركم الذي يستخدم لتخزين نتائج العمليات الحسابية الآنية كما يستطيع القيام بإجراء بعض العمليات الحسابية و المنطقية ، ويمكن لوحدة الحساب و المنطق استقبال البيانات عن طريق المركم أو الذاكرة أو مسجلات وحدة المعالجة المركزية وتقوم باستقبال إشارة التحكم من وحد ة التحكم حيث تحدد هذه الإشارة نوع العملية المراد تنفيذها في وحدة الحساب و المنطق.

كما وتتصل وحدة الحساب و النطق مع مسجل الحالة Status Register والذي يخزن مجموعة من الأرقام الثنائية التي تستخدم للتحكم بمعالجة البيانات ومن أهم هذه الأرقام:

1. بت الحمل Carry bit حيث يحتوي هذا البت على واحد إذا حدث فائض بعد إجراء العملية الحسابية
2. بت الصفر Zero bit حيث يحتوي هذا البت على واحد إذا كانت القيمة الناتجة في المركم مساوية للصفر.
3. بت الإشارة الموجبة Positive bit حيث يحتوي على واحد إذا كان محتوى المركم موجباً.
4. بت الإشارة السالبة Negative bit حيث يحتوي على واحد إذا كان محتوى المركم سالباً.



الشكل 8-1 وحدة الحساب والنطق

- **عمليات وحدة الحساب و المنطق:**

تصنف عمليات وحدة الحساب و المنطق كما يلي :

1. العمليات ذات المعامل الواحد ومن أهم هذه العمليات

- تصفير محتوى مسجل ما. clear
- إيجاد المكمل) المعكوس) لمحتوى المركم.
- زيادة محتوى مسجل ما بمقدار واحد. Increment
- طرح واحد من محتوى مسجل ما. Decrement
- إزاحة محتوى مسجل إلى اليسار أو اليمين.

2. العمليات ذات المعاملين ومن أهمها :

- الجمع : جمع محتوى المركم مع محتوى مسجل ما.
- الطرح: طرح محتوى مسجل ما من محتوى المركم.
- المقارنة : حيث تشبه هذه العملية الطرح أو الجمع إلا أن النتيجة لا تخزن في المركم بل يخزن 1 أو صفر اعتماداً على نتيجة المقارنة في بت المقارنة.
- العملية المنطقية(OR) إجراء عملية الجمع المنطقي بين محتوى المركم ومحتوى مسجل ما حيث تخزن النتيجة في المركم.
- العملية المنطقية (AND) إجراء عملية الضرب المنطقي لمحتوى المركم و مسجل ما وتخزين النتيجة في المركم.

- **دوائر وحدة الحساب و المنطق:**

تتألف وحدة الحساب والمنطق من عدة دوائر تقوم بالعمليات الحسابية والمنطقية ومن أهم هذه الدوائر:

1. **الجامع النصفي:Half Adder :**

عبارة عن دائرة الكترونية مؤلفة من بوابات منطقية تقوم بجمع رقمين ثنائيين مكون كل منهما من بت واحد. ولهذه الدائرة مدخلان ومخرجان يوضع الرقمان المراد جمعهما على المداخل أما المخارج فالأول يمثل نتيجة الفيض. Carry ويبين الشكل 9-1 دائرة الجامع النصفي.

الشكل 9-1 دائرة الجامع النصفي

ويمكن توضيح عمل نصف الجامع من خلال الجدول المنطقي التالي:

جدول 2-1 الجدول المنطقي للجامع النصفي

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

من خلال هذا الجدول يمكن تمثيل معادلات النتيجة و الفيض كدوال بالنسبة للمدخلات A,B
فلو أخذنا الدالة S فإن قيمتها تكون مساوية للواحد في حالتين:
1. عندما تكون A=0 و B=1 وبهذا تؤلف هذه القيم الحد الأول AB
2. عندما تكون A=1 و B=0 وتؤلف هذه القيم الحد الثانيAB
وبهذا تصبح S كما يلي : :

$$S= A.B + A.B$$

أما بالنسبة لمعادلة الفيض فإنها تحتوي على حد واحد (لاحظ قيمة C تكون مساوية للواحد في حالة واحد وهي عندما تكون A=1 و B=1 وبهذا فإن C تصبح كما يلي:
C=A.B

2. الجامع التام: Full Adder

يعرف الجامع التام على أنه دائرة الكترونية لها ثلاثة مداخل و مخرجان

حيث يستخدم لجمع ثلاثة أرقام كل منها مؤلف من خانة واحدة بت ويمثل الجامع التام حسب الشكل التالي:

الشكل 10-1 دائرة الجامع النصفي

ويمكن فهم عمل هذا الجامع من خلال الجدول المنطقي التالي:

جدول 3-1 الجدول المنطقي للجامع التام

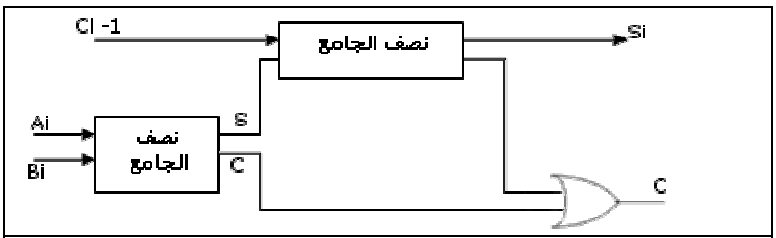
Ai	Bi	Ci-1	Si	Ci
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

من خلال هذا الجدول يمكن استنتاج معادلة S , C

$$Si = \overline{Ai}\,\overline{Bi}\,Ci\text{-}1 + \overline{Ai}\,Bi\,\overline{Ci}\text{-}1 + Ai\,\overline{Bi}\,\overline{Ci}\text{-}1 + Ai\,Bi\,Ci\text{-}1$$
$$= Ai + Bi + Ci\text{-}1$$

$$Ci = \overline{Ai}\,Bi\,Ci\text{-}1 + Ai\,\overline{Bi}\,Ci\text{-}1 + Ai\,Bi\,\overline{Ci}\text{-}1 + Ai\,Bi\,Ci\text{-}1$$
$$= Ai\,Ci\text{-}1 + Ai\,Bi + Bi\,Ci\text{-}1$$

ومن خلال هذه المعادلات يمكن بناء دائرة الجامع التام و التي تأخذ الشكل 1-11 :



الشكل 11-1 دائرة الجامع النصفي

1.7.3 وحدة التحكم (Control Unit):

تعتبر الجهاز العصبي للحاسوب حيث تقوم هذه الوحدة بالتحكم بكافة العمليات المنفذة وتشرف على تسلسل تنفيذ التعليمات وتنسيق العمليات وتبادل المعلومات بين وحدة الحساب والمنطق و الذاكرة الرئيسة.ويمكن إيجاز وظائف هذه الوحدة بما يلي:

1. تنشيط موقع التعليمة المراد تنفيذها والإشراف على نقل التعليمة من الذاكرة إلى مسجل التعليمة.
2. تحليل شفيرة العملية لتحديد نوع العملية المراد تنفيذها و إرسال إشارات التحكم الضرورية لوحدة الحساب والمنطق.
3. تنشيط مواقع البيانات في الذاكرة والإشراف على نقل هذه البيانات إلى مسجلات وحدة المعالجة المركزية CPU .
4. إخبار وحدة الحساب والمنطق بنوع العملية المراد تنفيذها.
5. زيادة عداد البرنامج بقيمة مساوية لطول التعليمة لتحديد عنوان التعليمة اللاحقة في الذاكرة الأساسية.
6. الإشراف على تمرير النتائج إلى الذاكرة الرئيسة.
7. إصدار إشارات التحكم اللازمة للقراءة من الذاكرة أو الكتابة فيها وإشارات التحكم بعمل وحدة الحساب والمنطق.
8. إجراء عملية التوقيت اللازمة لتنفيذ ما سبق بشكل متسلسل.

بهذا فان وحدة التحكم يمكن تعريفها على إنها وحدة الكترونية مؤلفة من مجموعة من الدارات المنطقة "الرقمية".

- **مكونات وحدة التحكم:**
  1. مسجل العنوان المستخدم لتخزين موقع الذاكرة المطلوب.
  2. مسجل التعليمة لتخزين التعليمة المراد تنفيذها.
  3. مسجل التعليمة لتحديد نوع العملية المراد تنفيذها.
  4. مسجل عداد البرنامج الذي يخزن عنوان التعليمة المراد تنفيذها لاحقاً.

5. دارات التوقيت لتحقيق عملية التسلسل في تنفيذ التعليمة.

- **حالات وحدة التحكم:**

ترتبط وحدة التحكم مع عدة وحدات وتتبادل معها المعلومات وأثناء تنفيذ هذه الوحدة لمهامها قد تقع في إحدى الحالات التالية:

1. حالة الإدخال Input Mode أي عندما تقوم بقراءة تعليمات وبيانات من ناقل البيانات.
2. حالة الإخراج Mode Output أي عندما تقوم بإرسال البيانات عبر ناقل البيانات.
3. حالة بداية عملية جديدة Beginning new Operation .
4. حالة القراءة من الذاكرة Memory Read.
5. حالة الكتابة في الذاكرة Memory Write.
6. حالة البحث عن تعليمه Instruction Fetch.
7. حالة القراءة من وحدة الإدخال I/O Read.
8. حالة الكتابة في وحدة الإخراج I/O Write.

- **التعليمات Instructions :**

يتكون البرنامج المراد تنفيذه من مجموعة من التعليمات تكون مخزنة في الذاكرة الرئيسة بالنظام الثنائي علماً بأن التعليمة مجموعة من الخلايا الثنائية مقسمة إلى حقل أو أكثر كما هو مبين في الشكل الشكل 12-1.

شفره العملية Operation Code	عناوين المعاملات Operands Address
التعليمة	

الشكل 12-1 قسيم التعليمات

مع ملاحظة أن شفره العملية تحدد نوع العملية المطلوب تنفيذها مثل الجمع،الطرح، .. الخ.
ومن الواضح أن عدد خانات شفره العملية يحدد عدد العمليات الممكن تنفيذها:

$$m = 2^n$$

حيث m عدد التعليمات، n عدد خانات شفره التعليمة أما حقل عناوين المعاملات مكون من عدة حقول مستخدمة لتخزين المعاملات أو عناوينها عنوان التعليمة الثنائية، عنوان النتيجة.

هذا وتخزن كل تعليمة من التعليمات في موقع في الذاكرة الرئيسة تحت عنوان محدد بحيث يتم حفظ عنوان التعليمة في مسجل خاص في وحدة التحكم يسمى عداد البرنامج Program Counter حيث يشير إلى عنوان التعليمة التالية، وبعد البحث عن التعليمة وإيجادها تخزن التعليمة في مسجل خاص في وحدة التحكم يسمى بمسجل التعليمة Instruction Register .

كيف ينفذ الحاسوب التعليمة (عمل وحدة التحكم) ؟

تقوم وحدة التحكم بتحديد عنوان التعليمة في الذاكرة الرئيسية وتشرف على تمرير التعليمة من الذاكرة إلى وحدة التحكم حيث تخزن في مسجل التعليمة ثم تأخذ دارات شفره التعليمة وتحللها لتحديد نوعها، بعد هذا تأخذ عناوين البيانات من مسجل التعليمة ويتم تنشيط هذه العناوين وتنقل البيانات اللازمة إلى سجلات CPU (مسجلات البيانات) ومن ثم تقوم وحدة التحكم بإخبار وحدة الحساب والمنطق عن نوع العملية المراد تنفيذها على البيانات المحددة.

- **تصنيف التعليمات:**

1. تصنف التعليمات حسب عدد العناوين المستخدمة في حقل العناوين إلى:
   - التعليمات ذات الأربع عناوين :
     حيث تحتوي هذه التعليمات على حقل شفره العملية إضافة إلى أربعة حقول تمثل أربعة عناوين.

OPC	A1	A2	A3	A4

حيث أن:
A1 , A2 عناوين المعاملين الأول والثاني.
A3 عنوان ناتج تنفيذ العملية.
A4 عنوان التعليمة التالية
OPC شفره العملية.

   - التعليمات ذات الثلاثة عناوين:
     إذا خزنت التعليمات في الذاكرة بشكل متسلسل فلا داعي للحقل A4.
     وللانتقال إلى التعليمة التالية يكفي زيادة عداد البرنامج بمقدار 1 وتمثل التعليمة ذات ثلاثة العناوين كما يلي:

OPC	A1	A2	A3

وحسب هذه التعليمة فانه يتم جلب المعاملات من المواقع A1,A2 وإجراء العملية المحدد نوعها في شفره العملية أما النتيجة فتخزن في الموقع A3.

- التعليمات ذات العنوانين:
تحتوي هذه التعليمات على الحقول OPC,A1,A2 وحسب هذه التعليمة فان المعاملات تجلب من المواقع A1,A2 وتنفذ عليها العملية المحدد نوعها قي شفره العملية أما النتيجة فتخزن في أحد مسجلات وحدة المعالجة المركزية.

- التعليمات ذات العنوان الواحد:
تحتوي هذه التعليمات على شفره العملية بالإضافة لعنوان معامل واحد أما المعامل الآخر فيتم استحضاره من المر كم.

2. تصنيف التعليمات حسب عدد المواقع اللازمة لتخزينها (حسب الطول):

- التعليمات أحادية البايت :
طول هذه التعليمة 8 بت "خانات" أو 1بايت تستخدم هذه الخانات لتخزين شفره العملية.
- التعليمات ثنائية البايت:يستخدم البايت الأول لشفره العلمية أما البايت الثاني فيستخدم لتحديد عنوان المعامل.
- التعليمات ثلاثية البايت:طول التعليمة من هذا النوع 3 بايت يستخدم الأول لتخزين شفره العملية أما الثاني والثالث فيستخدم لتخزين عناوين المعاملين الأول والثاني.

3.تصنيف التعليمات حسب نوعها:

- التعليمات الحسابية كتعليمات الجمع و الطرح الخ.
- التعليمات المنطقية كتعليمات AND, OR وتعليمات الإزاحة لليسار أو اليمين.
- تعليمات الإدخال والإخراج والمخصصة لتنفيذ عملية الإدخال أو الإخراج.
- تعليمات التكرار والمخصصة لتكرار تنفيذ عملية.
- تعليمات نقل التحكم والمخصصة لنقل التنفيذ إلى تعليمة لا تلي التعليمة التالية وذلك اعتمادا على شرط معين أو بدون شرط معين.

- **دورة التعليمة:**

إن الذاكرة الرئيسة ووحدة الحساب والمنطق ووحدة التحكم ترتبط معاً وتتعاون في تنفيذ التعليمة من خلال تنفيذ مجموعة من التعليمات يطلق عليها دورة التعليمة وتقسم دورة التعليمة إلى مرحلتين:

1. مرحلة البحث:
   - تبدأ هذه المرحلة بتمرير محتوى عداد البرامج إلى مسجل العنوان.
   - بعد تحديد عنوان التعليمة تقوم دوائر التحكم بإصدار الإشارات اللازمة لقراءة التعليمة من الذاكرة الرئيسية حسب العنوان المسجل في مسجل العنوان.
   - تمرر التعليمة إلى مسجل التعليمة في وحدة التحكم وعنوان المعامل إلى مسجل العنوان.
   - يقوم محلل التعليمة باستقبال التعليمة من مسجل التعليمة لتحليلها ومعرفة نوع العملية وإصدار الإشارات اللازمة لوحدة الحساب والمنطق لتنفيذها.
   - تتم زيادة عداد البرنامج بمقدار 1 للإشارة للتعليمة التالية في البرنامج.

2. مرحلة التنفيذ:

بعد تحديد نوع العملية وتحديد عناوين المعاملات في المرحلة الأولى تبدأ المرحلة الثانية حيث يمكن إيجاز هذه المرحلة في الخطوات التالية:

- تمرر المعاملات من الذاكرة الرئيسية إلى وحدة الحساب والمنطق (قراءة المعاملات).
- تقوم وحدة الحساب والمنطق بإجراء العملية المطلوبة على المعاملات.
- تقوم وحدة التحكم بإصدار الإشارة الخاصة لوحدة الذاكرة لاستقبال النتيجة في العنوان المحدد في التعليمة.

- **محلل التعليمة Instruction Decoder :**

قلنا انه لتنفيذ العملية لا بد من تحليل شفره التعليمة لتحديد نوعها وتنفيذها ويقوم بهذه المهمة وحدة خاصة داخل وحدة التحكم تسمى بمحلل التعليمة، ويلعب ط ول شفره التعليمة دورًا في تحديد تركيب محلل التعليمة حيث أن شفره التعليمة تحدد عدد الأوامر التي يمكن تنفيذها فلو كان طول الشفيرة 8 فإن اكبر عدد من الأوامر يمكن تنفيذه يساوي 256 أمرًا.

وبهذا يمكن تعريف محلل التعليمة على أنه دائرة الكترونية لها عدد من المداخل مساو لطول شفره التعليمة وعدد من المخارج مساو لعدد عمليات الأوامر الممكن تنفيذها.

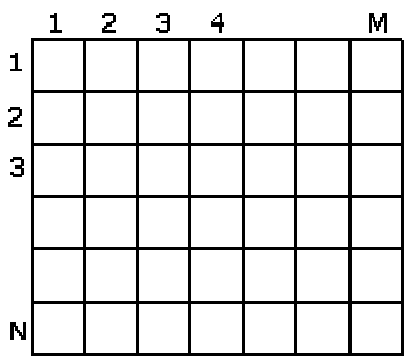
- **الذاكرة الرئيسية Main Memory:**

لتنفيذ العمل لابد من التواجد هذا العمل (البرنامج) و البيانات اللازمة في الذاكرة الرئيسية بصورة مؤقتة حتى تتم عملية المعالجة ويتم نقل النتائج إلى وحدات الإخراج وكما نعلم أن الذاكرة الرئيسية تتألف من مجموعة خلايا ثنائية على شكل مصفوفة وتحمل الخلايا في السطر الواحد نفس العنوان أو الموقع، ومسجل بيانات مؤقت يسمي مسجل الكلمة Word Register ووحدة تحكم محلية و ترتبط الذاكرة مع وحدة التحكم عن طريق مسجل العنوان (باستخدام خطوط العنوان) وبعض إشارات التحكم لتحديد عملية القراءة أو الكتابة.

1.7.4 تركيب الذاكرة الرئيسية:

يجب تمثيل رموز البيانات بالنظام الثنائي حتى يستطيع الحاسوب معالجة هذه الرموز لهذا فإن الذاكرة الرئيسية تتكون الخلايا Cells قادرة على تمثيل الأرقام الثنائية (صفر أو واحد) لذا يمكن لهذه الخلايا أن تمثل بالمفاتيح حيث أن حالة المفتاح المغلق تمثل الواحد و المفتوح تمثل الصفر أو يمكن أن تمثل بواسطة الخلايا المغناطيسية التي يسري فيها التيار الكهربائي. فعند سريان التيار الكهربائي في الخلية المغناطيسية فأنه سوف يولد بها مجالاً مغناطيسياً و اتجاه هذا المجال يحدد الحالة التي تقع فيها الخلية. فإذا كان اتجاه المجال مع عقا رب الساعة فإن الخلية تقع في حالة الواحد و في حالة الصفر إذا كان المجال بعكس عقارب الساعة.

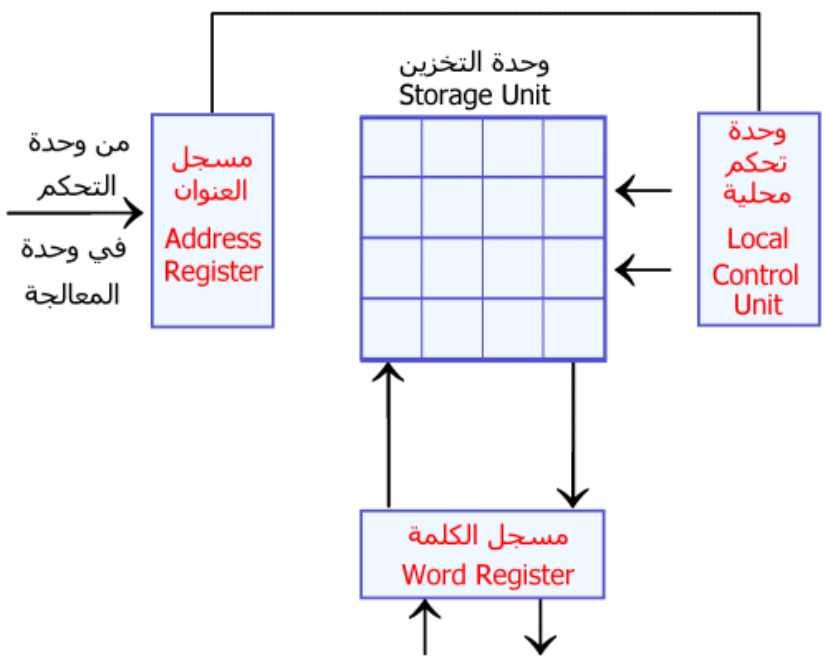
ومهما كان نوع خلايا الذاكرة فإنه يمكن اعتبار الذاكرة لوحة الكترونية مكونة من خلايا بحيث تشكل هذه الخلايا المصفوفة والشكل 13-1 يوضح هذا :



تمثل أسطر هذه المصفوفة المواقع في الذاكرة أما شكل 13-1
الأعمدة فتمثل الكلمات ويختلف طول الكلمة (عدد الخلايا في السطر) من حاسوب لآخر فقد تحتوي الكلمة على أربعة خلايا "بت"أو ثمانية أو ست عشرة.

ولتحديد كلمة ما في الذاكرة لابد من تحديد موقعها Address الممثل بالسطر فالكلمة الثالثة يـ تم الوصول إليها عن طريق العنوان 3 وهكذا . ومن هنا يتبين لنا تركيب الذاكرة الرئيسية حيث تضم هذه الذاكرة الأجزاء الرئيسية التالية:

1. مسجل العنوان address register حيث يخزن في المسجل عنوان الكلمة المراد الوصول إليها (رقم السطر في المصفوفة).
2. مسجل الكلمة word register حيث يسجل في المسجل الكلمة المراد الوصول إليها (الأعمدة المشار إليها بالعنوان المسجل في مسجل العنوان).
3. وحدة تحكم محلية Control Unit للإشراف على عمليات الوصول إلى الكلمات(القراءة والكتابة).
4. وحدة التخزين Storage Unit والممثلة بالمصفوفة نفسها (خلايا الذاكرة). و الشكل 1-14 يوضح تركيب الذاكرة الرئيسية:



الشكل 1-14 يوضح تركيب الذاكرة الرئيسية

يلعب مسجل العنوان دورًا في تحديد سعة الذاكرة حيث أن طول هذا المسجل (عدد الخلايا) يحدد عدد المواقع Addresses التي يمكن الوصول إليها فلو كان طول هذا المسجل 8 بت فإن عدد المواقع يساوي أي 256 موقعاً. أما مسجل الكلمة فيحدد طول الكلمة التي يمكن تخزينها في الذاكرة.

مثال: إذا علمت أن طول مسجل العنوان يساوي 8 خلايا وطول مسجل الكلمة 4 خلايا فاحسب حجم الذاكرة:

الحل:

عدد المواقع(Addresses) = 2^8 = 256 موقعاً

طول الكلمة = 4 بت

حجم الذاكرة = 256 × 4 = 1024 بت

حجم الذاكرة بالكلمات = 256 كلمة (على اعتبار أن الموقع الواحد يخزن كلمة)

حجم الذاكرة بالبايت = $\frac{1024}{8}$ = 128 بايت

(البايت = 8 بت)

من أهم العمليات التي يمكن أجراؤها على البيانات هي عملية القراءة و الكتابة حيث تتم عملية القراءة كما يلي:

1. يؤخذ العنوان من وحدة المعالجة المركزية (وحدة التحكم) ويخزن في مسجل العنوان.
2. تقوم وحدة التحكم المحلية بالإشراف على عملية البحث عن الكلمة المحدد عنوانها في مسجل العنوان .
3. عند إيجاد الكلمة المعينة تحت العنوان المحدد يتم نقلها إلى مسجل الكلمة وبعدها تنقل إلى وحدة المعالجة.

أما عملية الكتابة فتتم حسب الخطوات التالية:

1. يحدد العنوان المراد تسجيل الكلمة فيه بوضع هذا العنوان في مسجل العنوان.
2. تتم عملية البحث عن المواقع بإشراف وحدة التحكم المحلية.

3. تسجل الكلمة في مسجل الكلمة وبعد تحديد الموقع تنقل من هذا المسجل إلى الموقع المحدد.

- **طرق عنونة الذاكرة Addressing Methods :**

يستخدم حقل العنوان في التعليمة لتحديد عنوان موقع الذاكرة أو أحد مسجلات وحدة المعالجة المركزية بهدف الحصول على المعامل المطلوب أجراء العملية عليه.

تسمى الطريقة التي يتم فيها الحصول على المعامل بطريقة العنونة. والعنوان الذي يظهر مباشرة في التعليمة يسمى بالعنوان المبين Stated Address وعنوان موقع الذاكرة الذي يحتوي على المعامل يسمى بالعنوان الفعلي . Effective Address

قبل أن نستعرض طرق العنونة يجب أن نتذكر ما يلي:

- أن ذاكرة الحاسوب تخزن كل من التعليمات و البيانات:

الشكل 15-1 يوضح تمثيل التعليمات في الذاكرة

- لعنونة التعليمة يستخدم مسجل خاص في وحدة التحكم يسمى بعداد البرنامج Program Counter ولعنونة البيانات يستخدم غالباً مسجل آخر يسمى بعداد البيانات .Data Counter

تستخدم في الحاسوب طرق عنونة متعددة منها:

1. العنونة الفورية Immediate Addressing :

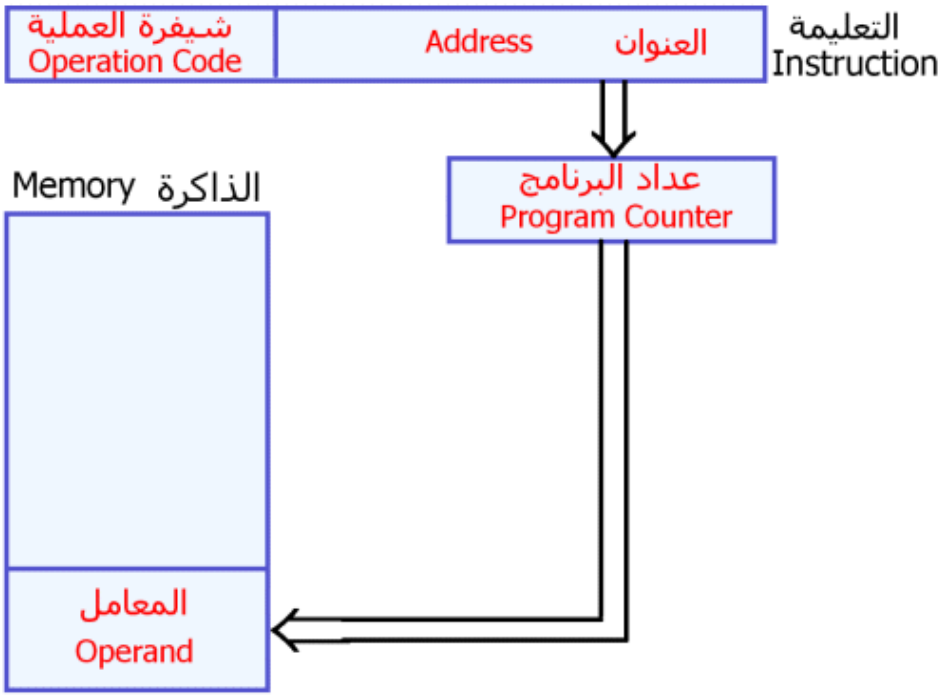
عند استخدام هذه الطريقة فإن التعليمة تحتوي على المعامل كجزء منها. أي أن حقل العنوان يحتوي على المعامل نفسه. وفي الحواسيب الصغيرة والميكروبية ، أي عند تخزين التعليمة في أكثر من موقع ذاكرة ، فإنه يمكن الحصول على المعامل بواسطة قراءة محتوى موقع الذاكرة الذي يلي الموقع الذي تخزن فيه التعليمة.
من الأمثلة على هذه الطريقة:

الشكل 16-1

2. العنونة المباشرة Direct Addressing :

وهي أكثر طرق العنونة انتشاراً ويتم فيها الحصول على العنوان الفعلي من العنوان المبين في التعليمة مباشرة. أي أن حقل العنوان يحتوي على عنوان موقع الذاكرة الذي يخزن المعامل .

الشكل 17-1 يبين طريقة الحصول على المعامل في حالة العنونة المباشرة.



الشكل 17-1 طريقة الحصول على المعامل في العنونة المباشرة

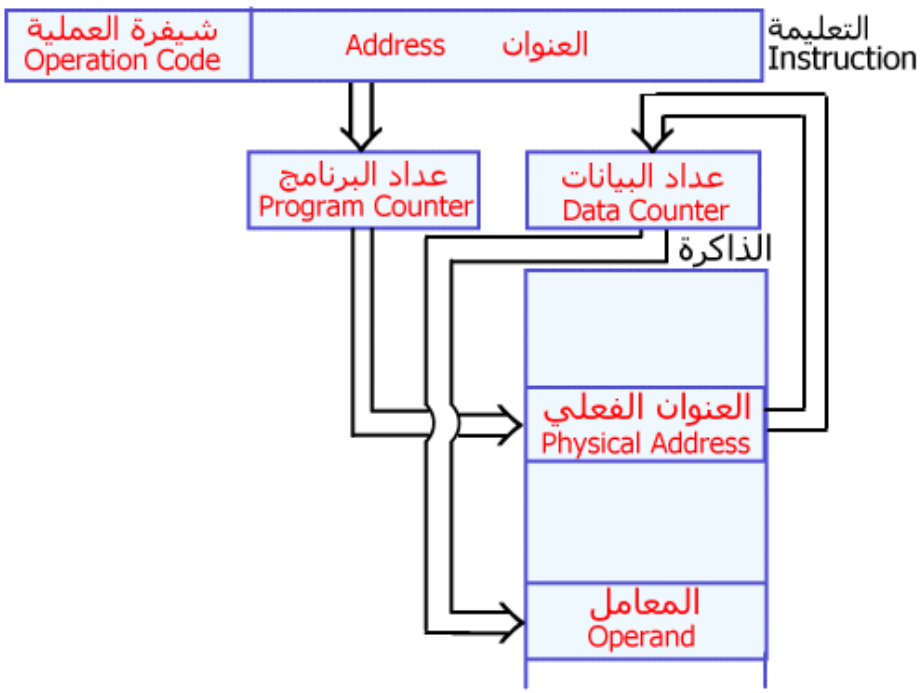
من الأمثلة على هذه الطريقة Load Register r LDR : وتبعاً لهذه التعليمة تقرأ محتويات موقع الذاكرة المعنون بواسطة التعليمة ويجلب ليحمل في المسجل r.
ومن مساوئ العنونة المباشرة أن عدد مواقع الذاكرة المعنوية محدود ، فإذا كان حقل العناوين يتكون من مواقع ثنائية عددها n فإنه يمكن عنونة موقعاً فقط.
ويمكن حل هذه المشكلة بعدة طرق منها:

1. زيادة قياس حقل العناوين وبالتالي قياس التعليمة، إلا أن هذا يتطلب زيادة قياس موقع الذاكرة.
2. تخصيص أكثر من موقع ذاكرة لتخزين التعليمة , وتستخدم هذه الطريقة في الحواسيب الميكروبية.
3. استخدام طرق عنونة أخرى.

3. العنونة غير المباشرة Indirect Addressing :

العنوان الفعلي في هذه الطريقة هو محتوى موقع الذاكرة المعنون بواسطة التعليمة. أي أن موقع الذاكرة المعنون بواسطة العنوان المبين في التعليمة يحتوي على عنوان موقع الذاكرة الذي يخزن فيه المعامل.

والشكل 18-1 يبين طريقة الحصول على المعامل في حالة العنونة غير المباشرة.



الشكل 18-1 يبين طريقة الحصول على المعامل في حالة العنونة غير المباشرة

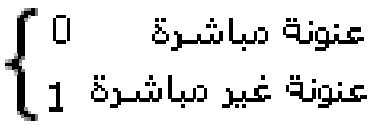
كما هو واضح من الشكل 18-1 فإنه للحصول على المعامل لابد من الرجوع إلى الذاكرة مرتين :

في الأول: تقرأ محتويات موقع الذاكرة المعنون بواسطة العنوان المبين في التعليمة وتجلب إلى وحدة المعالجة المركزية (إلى عداد البيانات).

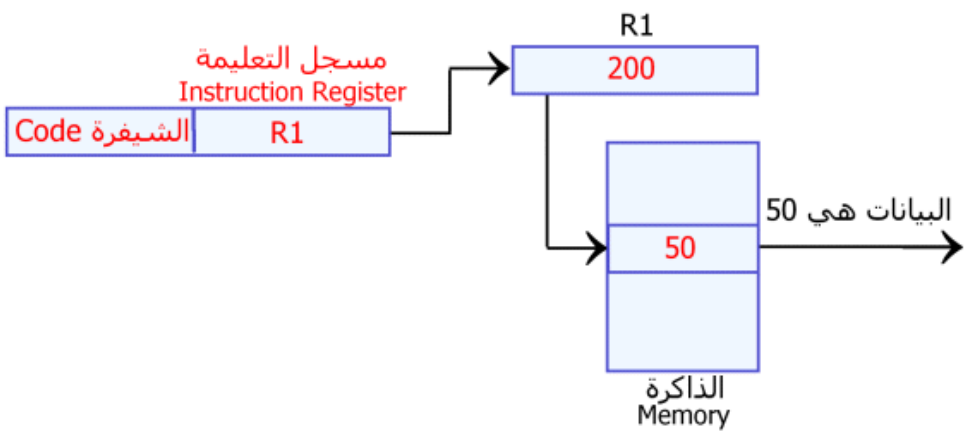
في الثانية: تقرأ محتويات موقع الذاكرة المعنون بواسطة عداد البيانات بهدف الحصول على المعامل.

لتميز نوع العنونة: هل هي مباشرة أم غير مباشرة يستخدم عادة بت خاص، فإذا كان محتواه 1 تكون طريقة العنونة المستخدمة غير مباشرة وإذا كان محتواه 0 تستخدم طريقة العنونة المباشرة.

التعليمة



توجد أنواع أخرى من طرق العنونة غير المباشرة . فمثلا يمكن أن تشير التعليمة أن زوج من مسجلات وحدة المعالجة المركزية يجب أن يستخدم لعنونة الذاكرة للحصول على المعامل.

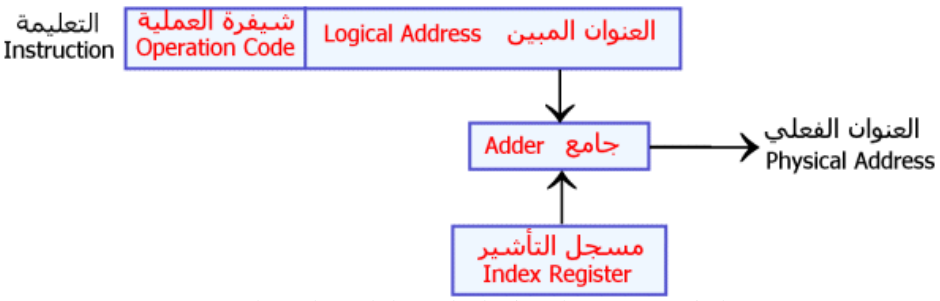


الشكل 19-1 العنونة غير المباشرة باستخدام مسجلات وحدة التحكم

من مساوئ العنونة غير المباشرة ضرورة الرجوع إلى الذاكرة مرتين مما يؤدي إلى تدني السرعة.

4. العنونة المؤشرة Indexed Addressing :

حسب طريقة العنونة المؤشرة , للحصول على العنوان الفعلي يجب جمع العنوان المبين في التعليمة مع محتوى مسجل خاص يسمى بمسجل التأشير Index Register ويبين الشكل 20-1 طريقة الحصول على العنوان الفعلي.



الشكل 20-1 طريقة الحصول على العنوان الفعلي في العنونة المؤشرة

عند التعامل مع الجداول المخزونة في مواقع متتالية في الذاكرة يمكن زيادة أو تنقيص محتوى مسجل التأشير بمقدار 1 للحصول على العنوان الفعلي، وتسمى هذه الطريقة بالتأشير الذاتي Auto Indexing .

ومن مساوئ استخدام العنونة المؤشرة ضرورة إجراء عملية الجمع للحصول على العنوان الفعلي.

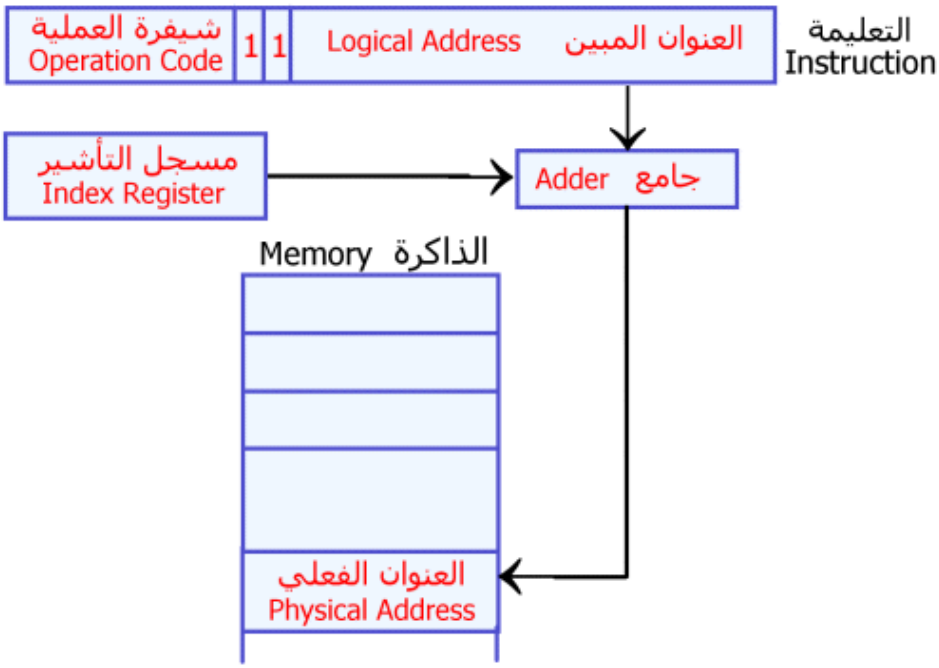
ومن مزايا هذه الطريقة بالمقارنة مع العنونة غير المباشرة , الرجوع إلى الذاكرة مرة واحدة بدلاً من مرتين.

في كثير من الحالات تستخدم طريقتا العنونة غير المباشرة والمؤشرة معاً، وفي هذه الحالة يجب أن تحتوي التعليمة على بت أضافي يدل على نوع العنونة المستخدمة. ويبين الجدول التالي طريقة العنونة المستخدمة بالاعتماد على محتوى الخانتين الإضافيتين المستخدمين لتحديد طريقة العنونة.

الخانة 1	الخانة 2	طريقة العنونة
0	0	عنونة مباشرة
1	0	عنونة غير مباشرة
0	1	عنونة مؤشرة
1	1	عنونة غير مباشرة - مؤشرة

شيفرة العملية	خانة 1	خانة 2	العنوان

ويبين الشكل 21-1 طريقة الحصول على العنوان الفعلي في العنونة غير المباشرة_المؤشرة.

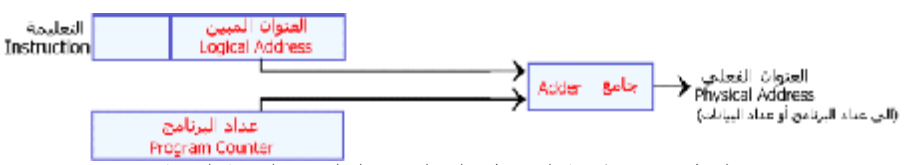


الشكل 1-21 طريقة الحصول على العنوان الفعلي في العنونة غير المباشرة - المؤشرة

5. العنونة النسبية : Relative Addressing

سنقوم بتسمية محتوى عداد البرنامج بالعنوان القاعدي .Base Address للحصول على العنوان الفعلي في هذه الطريقة يجب جمع العنوان المبين في التعليمة مع العنوان القاعدي.

ويبين الشكل 1-22 طريقة الحصول على العنوان الفعلي في العنونة النسبية.



الشكل 1-22 طريقة الحصول على العنوان الفعلي في العنونة النسبية

كما هو واضح، بزيادة أو تنقيص العنوان المبين (مع تثبيت محتوى عداد البرنامج) يمكن

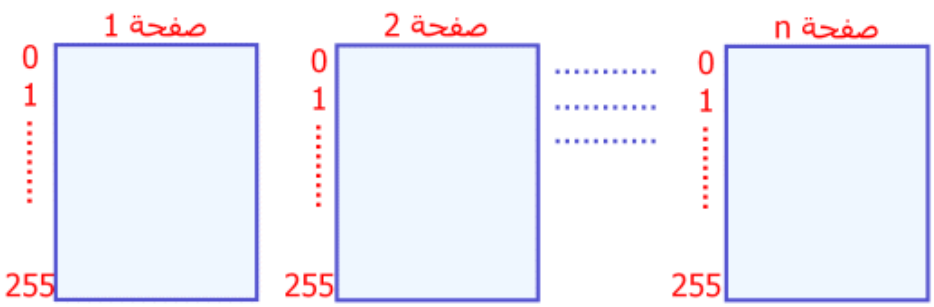
التفرع إلى الأمام أو الخلف. لهذا السبب يسمى العنوان المبين في التعليمة بقيمة الإزاحة Displacement وعادة توضع إشارة لقيمة الإزاحة .

6. العنونة الصفحية Page Addressing:

كما هو معروف فإن قياس موقع الذاكرة محدود. فمثلاً يبلغ قياس موقع الذاكرة في أغلب الحواسيب الميكروبية بايت واحد فقط. هذا الوضع يؤدي إلى تحديد قياس حقل العناوين في التعليمة.

من جهة أخرى، إذا كان قياس حقل العناوين في العناوين في التعليمة بايت واحد، فإن عدد مواقع الذاكرة الممكن عنونتها لا يزيد على 256 موقعاً، وواضح أن هذه السعة لا تسد الاحتياجات المطلوبة. وإحدى طرق حل هذه المشكلة هي زيادة سعة الذاكرة حسب الحاجة و تقسيم الذاكرة إلى صفحات Pages تحتوي كل منها على عدد متساوي من المواقع. وقياس الصفحة (عدد المواقع التي تحتويها) عادة يساوي عدد المواقع التي يمكن عنونتها بواسطة حقل العنوان في التعليمة.

ويبين الشكل 23-1 ذاكرة مقسمة إلى n صفحة (قياس كل صفحة 256 موقعاً).



الشكل 23-1 ذاكرة مقسمة إلى n صفحة تحتوي على 256 موقعاً

يتكون عداد البرنامج في العنونة الصفحية من جزئيين، حيث يحدد الجزء الأول رقم الصفحة ويحدد الجزء الثاني عنوان الموقع داخل الصفحة. أي أن العنوان الفعلي هو عنوان موقع الذاكرة بالنسبة للصفحة الحالية.

بمعني آخر فإنه يجب جمع عنوان أول موقع في الصفحة الحالية مع العنوان ا لمبين للحصول على العنوان الفعلي.

من مزايا العنونة الصفحية أمكانية زيادة سعة الذاكرة رغم القيود على قياس العنوان.
عند استخدام العنونة الصفحية توجد عدة مشاكل أهمها: ماذا يحدث إذا خرجت قيمة العنوان الذي يحدد الموقع في الصفحة الحالية عن مجال القيم المسموح بها، أي إذا وصلت قيمة هذا العنوان إلى 1....111؟

ولحل هذه المشكلة يمكن زيادة هذا العنوان بمقدار 1 للحصول على 0....00 وفيض يساوي 1 واستخدام أحدى الطرق التالية:

1- إهمال الفيض وبهذا فإنه تتم عنونة الموقع الأول من الصفحة الحالية.
2- جمع الفيض مع الجزء الثاني في عداد البرنامج والذي يحدد عنوان الصفحة و بهذا نكون قد انتقلنا إلى الموقع الأول في الصفحة التالية.

- تصنيف وحدة الذاكرة:

تصنف حسب العوامل التالية:

1- حسب الوسط الفيزيائي المستخدم لتخزين المعلومات.
2- حسب طريقة الوصول إلى المعلومات المخزنة.
3- حسب قابلية التطاير.
4- حسب القابلية للمسح والبرمجة.

- تصنيف وحدة الذاكرة حسب الوسط الفيزيائي المستخدم لتخزين المعلومات:
هناك أنواع كثيرة من الأوساط التي تستخدم لتخزين البيانات نذكر منها ما يلي:

1- الأوساط المغناطيسية: مثل ذاكرة الحلقة الممغنطة والأشرطة المغناطيسية والأقراص المغناطيسية.
2- الأوساط الإلكترونية: وتستخدم هذه الأوساط الدوائر الإلكترونية المصممة من الترانزستورات وكان هذا النوع مستخدما في الحواسيب القديمة وتستخدم الآن الدوائر المتكاملة خصوصًا في الذاكرة الرئيسية كوسط الكتروني لتخزين المعلومات.
3- الأوساط الورقية : تستخدم كأوسط تخزين في الذاكرة المساعدة مثل البطاقات المثقبة والأشرطة الو رقية المثقبة ، ولم تعد هذه الأوساط تستخدم في وقتنا الحالي .

- تصنيف وحدة الذاكرة حسب طريقة الوصول للمعلومات:
يوجد طريقتان للوصول إلى البيانات المخزنة في الذاكرة هما:

1. الوصول التتابعي: للوصول إلى سجل معين يجب قراءة جميع السجلات أو المعلومات التي تسبقه، كما هو الحال في الشريط المغناطيسي.
2. الوصول المباشر: في هذه الطريقة يتم الوصول مباشرة إلى المعلومات دون الحاجة إلى قراءة جميع المعلومات السابقة. هذا يعتبر زمن الوصول المباشر أقصر بكثير من الوصول التتابعي.

- تصنيف الذاكرة حسب قابلية التطاير:

نعني بالتطاير فقدان الذاكرة للمعلومات عند انقطاع التيار الكهربائي.

1. الذاكرة المتطايرة: وهي الذاكرة التي تفقد المعلومات المخزونة فيها عند انقطاع مصدر التغذية الكهربائية كما هو الحال في الذاكرة الرئيسية.
2. الذاكرة الغير متطايرة: وهي الذاكرة التي تحتفظ بالمعلومات المخزونة فيها فترة طويلة عند انقطاع التيار الكهربائي كما هو الحال في الذاكرة المساعدة.

- تصنيف الذاكرة حسب القابلية للبرمجة والمسح:

1. الذاكرة القابلة للمسح Erasable Memory حيث يمكن مسح الذاكرة وإعادة الكتابة عليها ومن أهم الأمثلة على هذه الذاكرة ذاكرة الحلقات الممغنطة والأقراص والأشرطة الممغنطة ومن الجدير بالذكر أن الذاكرة الرئيسية يطلق عليه RAM (Random Access Memory) حيث يمكن القراءة منها والكتابة عليها .
2. الذاكرة الثابتة: حيث تبقى المعلومات ثابتة على أوساط التخزين كما هو الحال في البطاقات المثقبة والأشرطة الورقية المثقبة . وهناك نوع من الذاكرة الثابتة يطلق عليها ذاكرة القراءة فقط ROM Read Only Memory و يتم برمجة هذا النوع من الذاكرة في مرحلة التصنيع ولا يمكن ا لكتابة عليها وتتم قراءتها عند الحاجة إليها حيث يخزن في الذاكرة برامج نظام التشغيل و الاقترانات المكتبية وغيرها .

ويبن الجدول 4-1 أهم الفروقات بين ROM RAM:

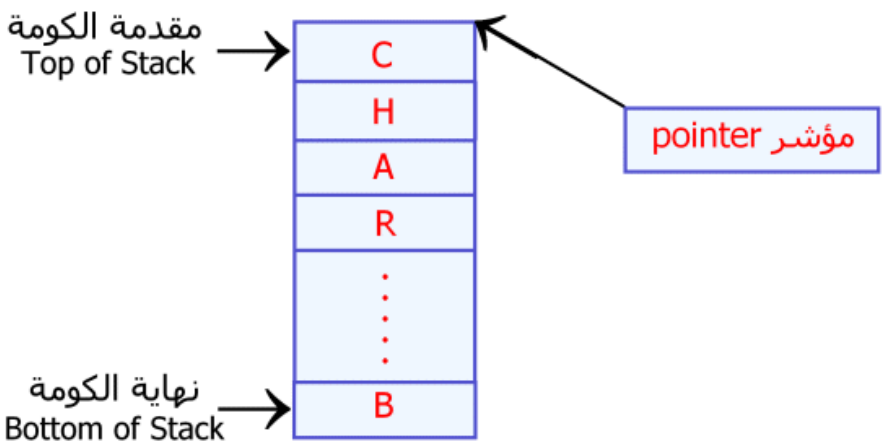
جدول 4-1

وجه المقارنة	ROM	RAM
عملية الكتابة	ممنوعة	مسموحة

تأثرها بالتيار	غير متطايرة	متطايرة
أمكانية البرمجة	مرة واحدة	حسب الحاجة
استخدام ها	لتخرين بعض البرامج اللازمة للتشغيل	لتخزين نظم التشغيل و برامج المستخدم

3. ذاكرة الكومة Stack Memory : تعتبر جزءًا من ذاكرة ROM وتستخدم لأغراض محددة وذلك لحفظ بعض القيم لاسترجاعها لاحقًاكما وتستخدم هذه الذاكرة بفاعلية عند استخدام البرامج الفرعية .

تتكون ذاكرة الكومة من مجموعة من المواقع تستخرج محتوياتها حسب مبدأ LIFO وذلك باستخدام مؤشر يستخدم لأغراض خاصة يعرف باسم مؤشر الكومة Stack Pointer وهو مسجل يشير إلى الموقع الذي يمكن التعامل معه . تقبل ذاكرة الكومة عملية القراءة (الاسترجاع pop) وعملية الكتابة (الإضافة push) كما هو مبين في الشكل 24-1 :



الشكل 24-1 ذاكرة الكومة

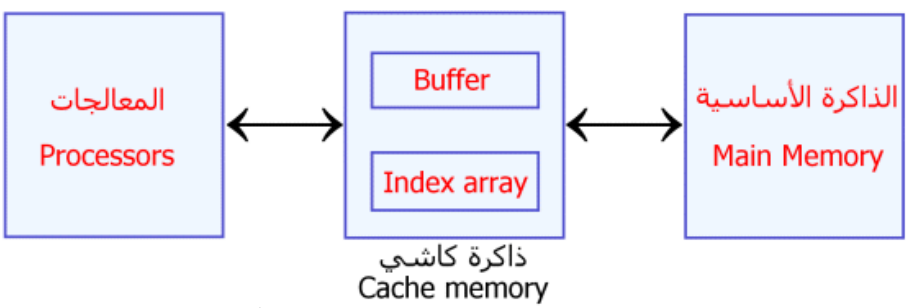
وتتم عملية التخزين في هذه الذاكرة دائمًا في المقدمة INTRODUCTION بحيث يكون في المقدمة INTRODUCTION بحيث يكون في المقدمة INTRODUCTION آخر عنصر تم إدخاله. LIFO last In First Out (الداخل آخرًا الخارج أولاً) .

إن عملية إضافة عنصر إلى مقدمة INTRODUCTION الكومة تسمى عملية دفع PUSH وعملية أخذ عنصر تسمى عملية سحب pop .

ودائماً يحتوي المؤشر على عنوان مقدمة INTRODUCTION الكومة Stack وتستخدم ذاكرة الكومة المنتظمة عند تنفيذ العمليات الحسابية وتستخدم أيضاً عند تنفيذ البرامج الفرعية لحفظ العنوان عند التفرع. وفي حالات الاعتراض عند توقف تنفيذ البرامج وانتقال السيطرة إلى تنفيذ برنامج أو عمل آخر.

4. الذاكرة الخفية Cache Memory:

الذاكرة الخفية عبارة عن مسجلات بالغة السرعة (أسرع من مسجلات الذاكرة الرئيسية) وأقل سرعة من مسجلات وحدة المعالجة المركزية CPU Registers ، وتقع الذاكرة المؤقتة(الخفية) بين المعالجات CPU Processors والذاكرة الأساسية، وفيها يتم تخزين الإيعازات المنتظرة والبيانات المرتبطة بها مما يحقق تقليل زمن الاستدعاء بنسبة 90% تقريباً.
عند استدعاء مجموعة من البيانات من الذاكرة الأساسية إلى المعالج، يتم فحص الذاكرة المؤقتة، فيما إذا كانت تحتوي على البيانات المطلوبة أم لا. فإذا كانت لا تحتوي على البيانات المطلوبة يتم إحضار البيانات من الذاكرة الأساسية إلى المواقع الفارغة في الذاكرة المؤقتة(الخفية).



الشكل 25-1 يبين موقع الذاكرة المؤقتة بالنسبة لكل من الذاكرة الأساسية و المعالجات

1.7.5 وحدات الإدخال Input Unite:

- لوحة المفاتيح KEYBOARD
- الفأرة MOUSE
- الماسحات الضوئية SCANNERS
- القارئ الكودي PARCODE READER
- القلم الضوئي LIGHT PEN

- **لوحة المفاتيح Key board:**

تتألف لوحة المفاتيح من مجموعة من المفاتيح بشكل مصفوفة (كما يبين في الشكل 26-1).

الشكل 26-1 مصفوفة لوحة المفاتيح

وتستخدم هذه المفاتيح لإدخال الرموز المكونة من الأحرف والأرقام والإشارات الخاصة وتصنف المفاتيح إلى:

1- مفاتيح الرموز والتي تستخدم لإدخال الرموز.
2- مفاتيح الحركة والتي تستخدم للتأثير على مؤشر الشاشة Cursor لتحريكه إلى اليسار أو اليمين أو إلى أسفل أو إلى أعلى.
3- مفاتيح التحكم.
4- المفاتيح الوظيفية والمخصصة لأداء وظائف محددة.

وعادة ما ترتبط لوحة المفاتيح مع الشاشة حيث تظهر الرموز المدخلة عن طريق اللوحة على الشاشة.

تحتوي لوحة المفاتيح بالإضافة إلى مصفوفة المفاتيح على وحدة تحكم محلية ومسجل خاص عن طريقه ترتبط لوحة المفاتيح بالحاسوب حيث يستخدم هذا المسجل لتخزين الرمز المدخل مؤقتاً ويبين الشكل التالي مكونات لوحة المفاتيح.

تستخدم وحدة التحكم المحلية لتنفيذ الوظائف الآتية:

1. تحديد المفتاح المضغوط وذلك بتحديد أحداثه (رقم السطر والعمود).
2. إيجاد شفره اسكي للرمز المناظر للمفتاح المضغوط.
3. إرسال شفره اسكي للرمز إلى مسجل الإدخال.

- **الشاشة Monitor Display:**

تستخدم الشاشة لعرض البيانات بصورة مرئية ويطلق عليها أحياناً أنبوبة الأشعة المهبطية CRT((Cathode Ray Tube أو الاسم Screen. ومهما كانت التسمية تعتبر من وحدات الإخراج الشائعة الاستخدام وتضم الشاشة أنبوبة الأشعة المهبطية المبينة في الشكل 1-27 .

الشكل 1-27الشاشة

ووحدة تحكم محلية تتحكم بموقع المؤشر لإظهار الرمز وتعمل على توليد الألوان وتوليد الإلكترونات اللازم قذفها للشاشة لإظهار الرموز الرسومات.

تستخدم الشاشة عادة لإظهار الرموز وبعضها يمكن استخدام ه لإظهار الرسومات والصور وعادة ما تقاس كفاءة الشاشة بالأمور الآتية:

1. إمكانية عرض الرسومات والصور إضافة لعرض الرموز.
2. الألوان المتوفرة.
3. دقة الشاشة Resolution وتقاس عادة بعدد النقاط Pixels التي يمكن التحكم بها أثناء عملية الرسم وإظهار الصور مثلاً لو كانت دقة الشاشة 640×480 فهذا يعني توفر 480 سطراً في الشاشة كل سطر مؤلف من 640نقطة.
4. الذاكرة الموقتة والتي تستخدم لتخزين النصوص أو الرسومات. تقسم الشاشة عادة عند استخدام ها في حالة النصوص (إظهار الرموز) إلى 24أو 25ويمكن عرض 80 رمزاً في السطر الواحد ويسمى الرقم 24× 80حرفاً بالصفحة.

أشرنا سابقاً إلى الأمور المحددة لكفاءة الشاشة وعادة ما تعتمد هذه الكفاءة على لوحة التحكم الخاصة بالشاشة Display Card.

وتعتمد عملية اختيار اللوحة المعينة على الدقة المراد الحصول عليها، عدد الألوان ومجال استخدام الشاشة للرسومات والنصوص ومن أهم أنواع اللوحات المتوفرة.

- لوحة الرسومات الملونة Color Graphic Adapter: CGA.
- لوحة الرسومات المحسنة Enhanced Graphic Adapter: EGA.
- لوحة الفيديو Video Graphic Adapter: VGA.
- لوحة الفيديو الفائقة Super VGA: SVG .

- **أنواع الشاشات من حيث اللون**

1. الشاشات أحادية اللون Monochrome : حيث يكون اللونان الأبيض والأسود هما المستخدمين .
2. الشاشات الملونة Colored : وتتراوح عدد الألوان المستخدمة من 16 إلى 256 إلى 16 مليون لون ، وذلك يعتمد على نوع بطاقة العرض المستخدمة ، والذي تتحدد معه أيضا كثافة عرض النقاط الضوئية الظاهرة على الشاشة ، Resolution .

و هناك نسبة مقارنة بين كثافة العرض وعدد الألوان المتاحة ؛ فكلما زادت الكثافة قلّ عدد الألوان المتاحة .

- **أنواع الشاشات من حيث كثافة العرض : RESOLUTION**

1. شاشات COLOR GRAPHICS ADAPTER (CGA) : وقد تم تطويرها عام 1981 حيث بلغت كثافة العرض فيها 200 X 320 نقطة ، وعدد الألوان 16 لون .
2. شاشات ENHANCED GRAPHICS ADAPTOR (EGA) : تم تطويرها عام 1984 وبلغت الكثافة النقطية 200 X 320 أو 350 X640 .
3. شاشات المسح العديدة : MULTI SCANNING : ظهرت عام 1985 بواسطة شركة NEC .
4. شاشات VIDIO GRAPHICS ADAPTOR (VGA) : طورته شركة IBM عام 1987 ، والذي يمكنه عرض كثافة نقطية 640 X 480 مع 16 لونا .
5. شاشات SUPER VIDIO GRAPHICS ADAPTOR (SVGA) : وتبلغ كثافته النقطية 800 X 600 مع 16 لونا .
6. شاشات EXTENDED GRAOHICS ADAPTOR (XVGA) : وتبلغ كثافته النقطية 1024 X 768 ، مع 256 لونا.

وتوفر كل لوحة من هذه اللوحات خصائص محددة مثل:

1. الدقة .
2. عدد الألوان.
3. حجم الذاكرة المؤقتة.

- **الطابعة Printer :**

تعتبر الطابعة وحدة من وحدات الإخراج المهمة حيث تخصص لإخراج النصوص و الرسومات وتحدد عادة الطابعة بالخصائص الآتية:

1. سرعة الطباعة وتقاس عادة بعدد الرموز التي يمكن طباعتها في الوحدة الزمنية.
2. دقة الطباعة وتقاس بعدد النقاط في الآنش الواحد والمخصصة لطباعة الرمز.
3. وجود الذاكرة المؤقتة والمخصصة لحفظ النصوص أو الرسومات المراد طباعتها.
4. عرض الورقة المستخدمة في الطباعة حيث تتوفر طابعات تستخدم الورق A4أو A3 (عرض80 حرف أو عرض132 حرفاً).

تتكون الطابعة عادة من:
1. وحدات ميكانيكية لتنفيذ الحركات اللازمة للورق أو شريط التحبير أو رؤوس الطباعة.
2. ذاكرة مؤقتة.
3. وحدات لتثبيت الورقة.
4. وحدات التحبير.
5. رؤوس الطباعة في بعض الطابعات.
6. شريط الأحرف والمطارق في بعض الطابعات.

ويبين الشكل 28-1 نموذجاً لأحد الطابعات:

الشكل 28-1 نموذج لأحدى الطابعات

تتوفر الآن مجموعة كبيرة من الطابعات وتصنف عادة إلى أصناف متعددة أهمها:

- تصنيف الطابعات حسب طريقة الطباعة ومنها:

1. الطابعات المطرقية التي تستخدم المطارق أو رؤوس الطباعة وعادة ما تحدث صوتاً ومن أهم أنواع هذه الطابعات الطابعات النقطية Dot Matrix أو الطابعات الخطية Line Printers.
2. الطابعات اللامطرقية وهي تستخدم في طرق مختلفة في الطباعة مثل طابعات الليزر، طابعات الـ نفث الحبري، الطابعات الحرارية، الطابعات الكهروستاتيكية وهذه الطابعات مريحة جداً لإنها لا تزعج بأصواتها كما في الطابعات المطرقية.

- تصنف الطابعات حسب السرعة إلى:

1. الطابعات البطيئة ومن الأمثلة عليها الطابعات النقطية وتقاس سرعتها عادة بعدد الأحرف المطبوعة في الثانية Character Per Second: CPS وقد تصل سرعة هذه الطابعات إلى أكثر من 300 حرف\ثانية.
2. الطابعات المتوسطة مثل الطابعات الخطية وتقاس سرعة هذه الطابعات بعدد الأسطر المطبوعة في الدقيقة الواحدة وقد تصل سرعة بعض أنواع هذه الطابعات إلى أكثر من 600 سطر\دقيقة.
3. الطابعات السريعة مثل طابعات الليزر والتي تحتوي على معالج يساعد في الحصول على طباعة رفيعة ا لمستوى حيث أن المعالج يفسر إشارات الحاسوب ويترجمها إلى أوامر تتحكم بحركة إشعاعات الليزر مما يؤدي إلى التحكم الجيد بحركة الورق وعملية طبع الصور على الورق. أو طابعات النفث الحبري وتقاس سرعة هذه الطابعات بعدد الصفحات المطبوعة في الدقيقة الواحدة وتحتاج هذه الأ نواع من الطابعات إلى ذاكرة عالية وقد تصل سرعتها إلى أكثر من 8 صفحات في الدقيقة الواحدة وتمتاز هذه الطابعات أيضاً بدقة طباعة عالية قد تتعدى 300 نقطة في الآنش الواحد.

- حسب اللون: يتوفر نوعان من الطابعات:

- الطابعات الملونة.

- الطابعات الغير ملونة.

وهناك عدة أنواع من الطباعات ، نذكر منها :

1- الطابعات النقطية أو الإبرية : DOT MATRIX PRINTERS .

2- الطابعات التي تعمل بأشعة الليزر : LASER PRINTERS .

3- الطابعات نفاثة الحبر : INK JET PRINTERS .

الطابعات النقطية أو الإبرية DOT MATRIX PRINTERS

انتشر استخدام هذه الطابعات بسبب رخص أسعارها واختلاف أحجامها وسرعاتها بما يتناسب وكل مستفيد .

في هذه النوع من الطابعات يتم تشكيل الحرف آليا بواسطة رؤوس معينة داخل الطابعة حيث تبرز الرؤوس الممثلة للحرف الواحد وتضغط بدورها على شريط مشبع ، ومنه إلى سطح الورق المثبت للطباعة .

تختلف سرعات الطابعات ويتم قياس السرعة بمعدل عدد

الحروف المطبوعة في الثانية CHARACTERS PER SECOND .

طابعات الليزر LASER PRINTERS

تعتمد هذه الطابعات في عملها على شعاع الليزر الذي يتم تشكيله بناء على البيانات المحولة من الحاسب ، ويسقط شعاع الليزر على مادة حساسة للضوء مشكلا الحرف أو الشكل المطلوب حيث تقوم هذه المادة بجذب نقاط الحبر وبثها فوق ورقة الطباعة ، بعد ذلك تمرر الورقة على نظام للتسخين من أجل تثبيت الحبر المطبوع على سطح ورقة الطباعة .

الطابعات نفاثة الحبر : INK JET PRINTERS

يعتمد نظام الطباعة في هذا النوع من الطابعات على ضخّ أو قذف الحبر في اتجاه الورق حيث يتم تكوين الصورة أو الشكل المطلوب طباعته ، وتمثيل جزء معين يتم ضخ الحبر الخاص به من وعية مثبتة يحتوي كل وعاء على لون معين .

- **الفأرة Mouse:**

الفأرة والتي تستخدم لتنفيذ الاختبار اللازم للتعليمات من نوافذ الشاشة أو لنقل وتحريك المؤشر على الشاشة أنظر للشكل 29-1.

كيفية عمل الفأرة:

- تقوم الكرة المطاط ية الموجودة أسفل الفأرة بتحريك العجلة التي تتحكم بالحركة العمودية والأخرى تتحكم بالحركة الأفقية للمؤشر حيث كل عجلة مرتبطة بمشعر.
- على حوافر المشعر يوجد قطع معدنية تولد إشارات كهر بائية كلما تلامست القطع المعدنية وعند ازدياد هذه الإشارات تتحرك الفأرة لمسافة أكبر.
- من خلال كيبل الفأرة يرسل السرعة المطلوبة وعدد الإ شارات إلى المؤشر على الشاشة حيث تقوم بضغط أي مفتاح للفأرة ليتم تمرير المعلومة وتنفيذها.

الشكل 29-1الفأرة

- **الراسمات والماسحات** Scanner:

تستخدم الراسمة في أعمال التصاميم أما الماسحة فتستخدم في إدخال الصور إلى الحاسوب ومن أهم الماسحات Scanner:

- يدوية والتي تمرر فوق الورقة.
- الوسادة المسطحة والتي تشبه آلة التصوير.
- ماسح التعليم الميكانيكي والذي يعمل على ترجمة مستويات الفوليتة إلى قيم رقمية.

الشكل 30-1 الماسحات الضوئية

- **وسائل الجمع الآلي للبيانات:**

تستغرق عملية إدخال البيانات عن طريق لوحة المفاتيح وقتاً طويلاً وجهداً كبيراً خاصة عندما يتعاظم حجم البيانات لذلك لاحت فكرة أتمتة إدخال البيانات آلياً دون جهد كبير مما استدعى ابتكار عدة وسائل أبرزها:

مميز حروف الحبر المغناطيسي Magnetic _Ink Character Recognition:
وتسمى اختصاراً MICR ويمكن لآلة المدخلات قراءة الحروف المكتوبة بواسطة حبر مغناطيسي ويوضح الشكل 31-1 شكل الحروف المستخدمة، وتستخدم هذه الآلات غالباً مع شيكات البنوك.

الشكل 31-1 حروف الحبر المغناطيسي

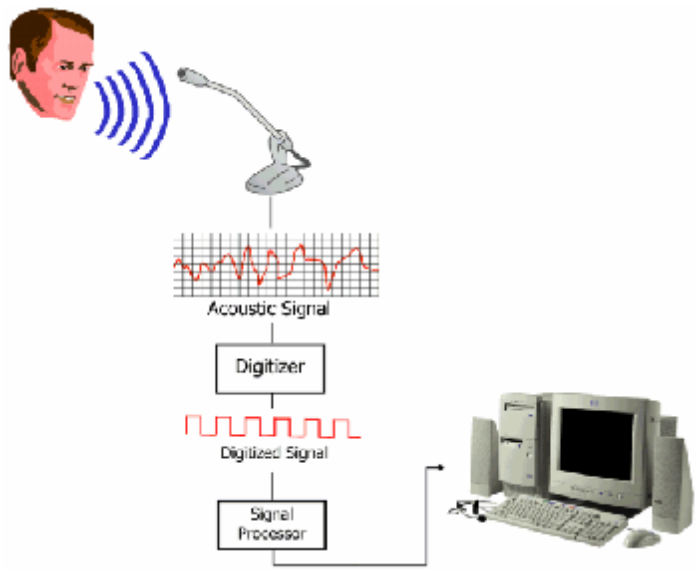
- **المميز الضوئي للحروف: Optical Character Recognition**

ويسمى اختصاراً, OCR وفيها تقوم الآلة على مسح المستند ضوئياً وتحويل كلماته إلى نبضات كهربية وفق نظام التشفير المتبع في الحاسوب وترسل هذه النبضات إلى الحاسوب لمعالجتها، ويوجد من أجهزة المميز الضوئي للحروف أجهزة يمكنها قراءة مدخلات العلامات الضوئية ويستخدم هذا الجهاز في قراءة علامات خاصة، لذلك تعتبر هذه الأجهزة مفيدة في تصحيح أوراق أسئلة الامتحانات، كما توجد أجهزة التميز الضوئي للحروف OCR التي يمكن للإنسان قراءتها و يمكن كذلك للآلة وتحويلها إلى نبضات كهربية و إرسالها للحاسوب وإن كانت هذه الحروف ترسم بشكل هندسي أقرب منه إلى الشكل الجمالي حتى لا تخطأ أجهزة OCR فيما يوضحه الشكل 32-1 وهناك أجهزة أخرى يمكنها التعامل مع الكتابة اليدوية باللغة الانجليزية و كذلك أجهزة أو قلم ضوئي لقراءة الشفرة الخطية Bar Code والتي تضعها الشركات على منتجاتها بحيث يكون لكل م نتج شفرة خاصة تدل على اسمه و اسم منتجه.

الشكل 32-1 حروف المميز الضوئي للحروف

- **أجهزة التميز الصوتي Speech Recognition Devices:**

يستخدم الصوت كمدخل من مدخلات البيانات حيث تتولى الأجهزة تحويل النبرات الصوتية عبر الميكروفون وتحويلها إلى الشفرة الثنائية المناظرة فيما يبدو من الشكل 33-1 .



الشكل 33-1 استخدام الصوت للاتصال بالحاسب

وتعتبر أجهزة التميز الصوتي فتحاً جديداً في تبسيط التعامل مع الحاسبات رغم صعوبتها البالغة حتى الآن بالنسبة لمختلف اللغات الحية.

- **الكمبيوترات المحمولة**

يعتبر الكمبيوتر المحمول (أو الجيبي) مهماً للذين تتطلب أعمالهم استخدام الحاسب الآلي خارج المكتب أو المنزل ، مثل رجال الأعمال والمتعاملين بالبورصات ؛ فهو لا يزيد وزنه على 400 جم ، ويمكن حمله في الجيب بسهولة ، ويعمل لعدة شهور ببطاريتين .

وهو مزود بلوحة مفاتيح والعديد منها مزود بقلم يستخدم كـ(ماوس) للتحريك والكتابة على الشاشة ، وفي حالة عدم وجود لوحة المفاتيح توجد برامج تتيح تتعرف على خط اليد من خلال الكتابة على الشاشة .

ليس ذلك فحسب ؛ بل إضافة إلى الوظائف الاعتيادية للمفكرات الرقمية مثل تخزين العناوين والتليفونات والمواعيد ، تتيح الموديلات الحديثة من الكمبيوتر المحمول تطبيقات عملية مثل معالجة الكلمات والجداول والتعامل مع البريد الإلكتروني وشبكة الانترنت ، وتعمل على بعض منها نسخة مصغرة من ويندوز 95 .

1.7.6 الكتابة بالقلم العربي الإلكتروني

تمتاز تقنية القلم العربي الإلكتروني بإنها تغلبت على واحدة من أبرز المشاكل التي واجهت تعامل الكمبيوتر مع النص العربي بصفة عامة ، وهي المشكلة المتعلقة بالخط اليدوي المتصف أصلاً بعدم انتظامه وتغيره الشديد . فبتقنية القلم ا لضوئي العربي أصبح الكمبيوتر يستطيع أن يقرأ ما كتبه الإنسان بخط يده باللغة العربية وتحويلها إلى نص قابل للتحرير ، بدون إدخالها بالنقر على لوحة المفاتيح حرفا حرفا كما هي الطريقة التقليدية ، بل باستخدام الماسح الضوئي صفحة صفحة.

و تمثل هذه التقنية نقلة كبيرة في عالم الكمبيوتر عند العرب ، حيث أصبح بالإمكان إدخال العديد من النصوص العربية المطبوعة أو المكتوبة على ورق مع توفير وقت وجهد وأجور موظفي الصف على لوحة المفاتيح .

وغالبا لا تحتاج هذه التقنية إلى أي تدريب مسبق على خط يد الكاتب ، وكل ما في الأمر أن الكاتب نفسه يحتاج إلى شيء من الألفة مع التقنية نفسها في زمن لا يتجاوز عشر دقائق . أي كي يألف التعامل مع هذه التقنية بشكل مريح .

والمطلوب من مستخدم هذه التقنية بخط يده أن يكتب بالحد الأدنى من المقبول من الكتابة اليدوية الواضحة للإنسان ، والحفاظ على التناسب الطبيعي بين أحجام الحروف العربية ؛ فلا يخلط بين " سـ " و" لللـ " وعلى المسافات المعتادة بين الحروف .

و لكن قد يعيب هذه التقنية حتى الآن إنها تتوافق مع بعض الخطوط دون غيرها، وقد تجهل بعض الرموز المستعملة في النصوص العربية، مثل بعض الأقواس .

1.8 الكمبيوتر ومتاعب المهنة

توجد قائمة غير قصيرة من الأمراض والأعراض المرضية التي يتسبب فيها الجلوس أمام جهاز الكمبيوتر ، من الصداع إلى آلام الظهر مرورا بالتهاب العينين وتخدر الذراعين وآلام المفاصل ، ولكن بشيء من الحرص واتباع التعليمات الطبية يمكن أن تتفادى هذه الأخطار .

فقد رصدت مراكز بحوث في أوروبا الغربية وأمريكا الشمالية واليابان انتشارا واسعا لما أسموه سي آر دي (Computer Related Disorders CRD) أي أمراض الجلوس أمام الكمبيوتر ، فقد وجدوا أن نصف الجالسين طويلاً أمام الكمبيوتر يعانون آلاما في الظهر أو الذراعين واليدين ، وأكثر من ثلثهم يعانون متاعب في العينين ، ومثلهم يعانون من الصداع .

و وصل الأمر بهذه الأمراض إلى أن اعتُبرت في كثير من دول منها الولايات المتحدة ضمن الأمراض المرتبطة بالوظيفة والتي يستحق المصابون بها بدلات إصابة عمل ، حيث يعجز المصاب بمثل هذه الأمراض عن تقليب صفحات كتاب أو الإمساك بفنجان شاي . وأحيانا ينجم عن الجلسة الخاطئة أمام الكمبيوتر عدة أعراض سيئة ، منها تشنج العضلات وصداع ناجم عن اضطراب جريان الدم في الرقبة ، ودوالي الرجلين بسبب اضطراب جريان الدم فيهما .

و لأن الوقاية خير من العلاج فلابد أ ن تراعي عدة أمور أثناء جلوسك أمام جهاز الكمبيوتر ؛ فينبغي أن يكون ارتفاع الكرسي مناسباً بحيث تكون القدمان مستويتين على الأرض ، والفخذان مستقرين على مقعد الكرسي ، ويستند الظهر قائما على مسند الكرسي وتسترخي الذراعان بحيث تكون اليدين مستقرتين بصورة مريحة على لوحة المفاتيح .

و قد بدأت شركات تصميم وإنتاج الأثاث المكتبي تراعي البعد الصحي في تصميم أثاثات مكاتب الكمبيوتر ، وأصبح ذلك من الدلائل التي تشير شركات الأثاث إليها في الدعاية عن منتجاتها . وكذلك تعددت أشكال وتصميمات لوحات المفاتيح في الآونة الأخيرة ، ويعود ذلك بالدرجة الأولى إلى الحرص على مراعاة التعليمات الطبية في هذا الشأن ، خصوصا بعد أن رفعت لدى المحاكم دعاوى ضد شركات الكمبيوتر لعدم اعتبارها هذه التعليمات .

و تشكل الشاشات أيضا مصدرا مهما من مصادر أمراض الكمبيوتر بسبب الضوء المنبعث منها وقرب المستخدم من الشاشة وعدم استخدام فلتر أمامها ، لكن آلام العين التي يشكو منها الجالس أمام الكمبيوتر لا تكون الشاشة سببها دائما ؛ فقد يكون السبب هو سوء إضاءة الحجرة مثلاً ؛ لذا ينصح باختيار مكان للشاشة بحيث لا ينعكس عليه ضوء النافذة وأن تكون المسافة بين المستخدم والشاشة حوالي 50 سنتيمتر .

1.9 لغات البرمجيات Programming

الحاسب الآلي بدون برمجيات كإنسان بلا روح ، كما أن التطور الذي يحدث في أجهزة الحاسب الآلي ومكوناته يصاحبه أيضا تطور وتحديث دائم في عالم البرمجيات .

والبرمجيات بصفة عامة هي عبارة عن مجموعة من الأوامر المرتبة منطقيا ، ويتم تنفيذها بواسطة وحدة المعالجة المركزية للحاسب الآلي ، ويختلف مستوى ونوع البرمجيات طبقا لعلاقاته وقربه من الحاسب الآلي من ناحية ، أو من قربه وعلاقته بالمستخدم من ناحية أخرى ، فنجد أن نظ ام التشغيل OPERING SYSTEMS بشكله الأولي هو الملتصق مباشرة بوحدة المعالجة المركزية CPU بينما نجد على الطرف الآخر ، التطبيقات البرمجية APPLICATIONS هي الأكثر قربا وسهولة بالنسبة للمستخدم.

إن القاسم المشترك بين جميع لغات البرمجة هي العناصر التالية:

* محرر النصوص أو ال(editor).
* المبرمج أو ال(compiler).

* نظام التشغيل أو ال(operating system).

فماذا نعني بالبرمجة ؟

البرمجة تعني : مجموعة من الأوامر والتعليمات التي تعطى للحاسب في صورة برنامج مكتوب بلغة برمجة معينة بواسطة معالج نصوص ويتكون مصدر البرنامج من عدة سطور وكل سطر يعتبر جملة ويتعامل الحاسب مع كل جملة بترتيب معين لإنجاز الأمر الذي صمم البرنامج لتحقيقه .

و كان للسيطرة على الكومبيوتر نكتب ما نطلق عليه اسم (برنامج) و هو يحتوي على عدد من الأوامر الموجهة للكومبيوتر ليحل مسألة معينة.

كان المبرمجون الأوائل (كان الله في عونهم) يكتبون البرامج بهذه اللغة! يكون شكل البرنامج هكذا:

10010001010100101
10100010000010011
10000000000000001
00000011111010010

و كما ترون فإنه من الصعب جدّاً فهم شيئاً كالذي ترونه في الأعلى بعقلنا البشري. لذا فقد كان من الصعب أيضاً إيجاد الأخطاء و تصحيحها. ولكن تلك التي تعمل، فإنها تعمل بسرعة شديدة جدّاً نظراً لأنها مكتوبة بلغة تفهمها الآلة بصورة مباشرة. و تسمّى البرامج المكتوبة بلغة الآلة ببرامج الجيل الأول.

لم يستطع أجدادنا المبرمجون الاستمرار على هذ ا الحال، فقد كان ف علاً صعباً، لذا قاموا باختراع لغة التجميع (Assembly) و هي لغة أبسط من لغة الآلة نظراً لاحتوائها على أوامر مثل (ADD و MOV) ، و هكذا كما ترون فقد ابتعدنا قليلاً عن لغة الآلة التي سبق و ركّزت على إنها لا تفهم سوى الصفر والواحد. و هنا ظهرت الحاجة الماسّة إلى المفسّر (Interpreter) و هو برنامج يقوم بتحويل الأوامر بالتتابع من لغة ال Assembly إلى لغة الآلة، و يوقف البرنامج فوراً في حالة مواجهة خطأ في البرنامج و لا يقوم بالنظر إلى بقية البرنامج (حتى لجبر الخاطر) .

نلاحظ هنا أننا حصلنا على برنامج تسهل كتابته نسبياً و يمكن بسهولة إيجاد الأخطاء فيه و تصحيحها(بدلاً من النظر إلى 0,1).

و هكذا أصبح التحويل من لغة التجميع إلى لغة الآلة، يأخذ جزء من وقت البرنامج لكن أبطأ قليلاً، و حين أقول قليلاً، فأنا أعني جزء من آلاف الأجزاء من الثانية الواحدة. و تعتبر لغة التجميع من الجيل الثاني (الذي يحتوي على هذه اللغة فقط).

المبرمجون لم يروق حالهم (كما لم يروق حالنا حين درسنا هذه اللغة) لذا أرادوا لغات تكون قريبة من لغة الإنسان، يفهمها بسهولة و ليست معقدة مثل لغة التجميع. لذا قاموا بوضع العديد من لغات البرمجة التي تقوم على أوامر مباشرة بلغتنا البشرية التي نفهمها بسهولة، ففي لغة الJava مثلاً إذا أردنا طباعة كلمة ("Word") على الشاشة نكتب الأمر التالي ضمن البرنامج:

```
System.out.print("mama");
```

و هذا كفيل بطباعة الكلمة على الشاشة

و لكننا هنا ابتعدنا عن لغة الآلة كثيراً، لذا توجّب علينا استخدام المترجم بدلاً من المفسّر، لتحويل البرنامج إلى لغة الآلة.

و هذه اللغات هي لغات الجيل الثالث و منها:Pascal, COBOL, ForTran, Basic, و منها أيضاً لغة ال C و التي تعتبر أسرع لغات الجيل الثالث على الإطلاق، بل إنها في كثير من الأحيان تتصرّف كما لو كنت من الجيل الثاني (حيث يقوم بعض المصنّفون تصنيفها ضمن الجيل الثاني لهذا السبب).

1.9.1 نبذة تاريخية عن C و ++C

جرى تطوير لغة C انطلاقاً من لغتين سابقتين لها هما: BCPL و B. بنيت لغة BCPL في عام 1967 من قبل Martin Richards لتكون لغة برمجة لكتابة نظم استثمار ومترجمات. بعدها أضاف Ken Thompson العديد من الإمكانات إلى لغته B التي طورها انطلاقاً من نسخة مطابقة من لغة BCPL وذلك من أجل كتابة النسخ الأولى لنظام التشغيل UNIX في مخابر شركة BELL سنة 1970 على المعالج DEC PDP-7. كانت اللغتان BCPLو B خاليتين من الأنماط (Typeless Languages) حيث يشغل كل عنصر من المعطيات كلمة حاسوب ‘‘Word’’ واحدة

في الذاكرة، ويقع من ثَم عبء معالجة عناصر المعطيات، على أنها أرقام كاملة مثلاً أو أعداد حقيقية، على عاتق المبرمج.
لقد طور Dennis Ritchie تطوير لغة البرمجة C اعتماداً على اللغة B في مخابر شركة Bell وتم بناؤها بالأصل من أجل حواسيب تستخدم المعالج DEC PDP-11 سنة 1972.
تستخدم لغة C العديد من المفاهيم الهامة التي كانت موجودة في اللغتين BCPL و B بالإضافة إلى إضافة أنماط للمعطيات مع العديد من الإمكانيات الأخرى الهامة.
في البداية، عُرفت لغة C على أنها لغة التطوير الخاصة بنظام التشغيل UNIX. أما حالياً فإننا نستطيع القول إن جميع نظم التشغيل الرئيسية هي نظم مكتوبة بلغة C و/أو لغة C. لقد أصبحت لغة C متاحة على معظم الحواسيب خلال العقدين الماضيين وهي تتصف بكونها لغة مستقلة عن البنية الصلبة للحواسيب، إضافةً إلى كونها ذات تصميم دقيق ويمكن استخدامها لكتابة برامج قابلة للنقل بين معظم الحواسيب.
في أواخر السبعينيّات، أضحت لغة C معروفة باسم لغة C التقليدية أو لغة C الخاصة بـالمؤلفين Ritchie و Kernighan. ولقد ساعد نشر كتاب لغة البرمجة C (The C Programming Language) للمؤلفين (دار النشر Prentice-Hall) سنة 1978 على جلب انتباه واسع إلى هذه اللغة، ليصبح بعدها هذا الكتاب أحد أشهر الكتب وأنجحها على الإطلاق من بين جميع الكتب في مجال المعلوماتية.

ولسوء الحظ، أدى الاستخدام الواسع للغة C على العديد من الأنواع المختلفة للحواسيب (منصات مادية مختلفة hardware platforms) إلى بعض الاختلافات بين نسخ لغة C التي يمكن أن تكون متشابهة ولكنها غالباً غير متوافقة فيما بينها. وهذا ما أنشأ مشكلة عويصة لمطوري البرامج الذين يحتاجون إلى كتابة برامج يمكن أن تعمل على عدة أنواع من الحواسيب. وظهرت بنتيجة ذلك الحاجة إلى تعريف نسخة معيارية من لغة C. أنشئت في سنة 1983 اللجنة التقنية X3J11 تحت إشراف اللجنة الوطنية الأميركية للمعيرة الخاصة بالحواسيب ومعالجة المعلومات (ANSI/X3) من أجل "تحديد تعريف غير مبهم ومستقل عن الحواسيب المُستخدمة للغة البرمجة C". وجرى في سنة 1989 التصديق على المعيرة التي أُصدرت. بعدها قامت منظمة الـ ANSI بالتعاون مع المنظمة العالمية للمعيرة (ISO) بتعميم المعايرة المذكورة في مختلف أنحاء العالم. و أُصدر نص المعايرة سنة 1990 تحت الرقم :ANSI/ ISO 9899 1990 ويمكن طلب نسخ منها من المعهد ANSI. تعكس الطبعة الثانية من كتاب Ritchie و Kernighan المنشور عام 1988 الأفكار المتعلقة بهذه النسخة المسماة بـ ANSI C والتي هي حالياً في قيد الاستخدام على مستوى العالم أجمع.

و مع تطور البرامج و زيادة عدد الأسطر في كل برنامج بدأ المبرمجون يفكرون في تطوير للغة السي .... نحن بحاجة للغة تعطيني القدرة على إعادة استخدام أكوادنا

القديمة ، طريقة تسمح باشتراك مجموعة كاملة من المبرمجين في مشروع واحد مع الحفاظ على السرية و الأمان ، كما أننا بحاجة لطريقة برمـ جية تكون قريبة من التفكير البشري ===> هنا ظهرت الحاجة للغة C++ .

C++

فأ خترعت هذه اللغة على يد المبرمج Bjarne Stroustrup في عام 1979 في معامل بيل في نيوجرسي . في البداية ،أطلق على هذه اللغة الاسم " C with Classes" ثم تم اعتماد اسم C++ في عام 1983.

لغة C++ تحتوي على جميع خصائص لغة C التقليدية (بالإضافة للمزيد من الخصائص) . بعد ذلك تم تطوير الصيغة القياسية المعتمدة لهذه اللغة: حيث كانت أول محالة لذلك في عام 1994 .

1.9.2 نبذة تاريخية عن FORTRAN , COBOL , Pascal and Ada

طورت مئات لغات البرمجة العالية لكن القليل منها اكتسبت قوة وشهرة واسعة مثل الفورتران . FORTRAN (FORmula TRANslator) طورت عن طريق شركات الأي بي إم في اواسط 1950 م لتستخدم في التطبيقات العملية والهندسية التي تحتاج إلى تطبيقات حسابية معقدة. فتستخدم بشكل واسع في التطبيقات الهندسية.

OCBOL (COmmon Business Oriented Language)

طورت في نهايات 1950 م عن طريق معامل الكمبيوتر وحكومة الولايات المتحدة ومعاهد استخدام الكمبيوتر ، كوبول تستخدم لغرض التطبيقات التجارية التي تحتاج إلى تسعيرات ومعالجات فعالة لاعداد كبيرة من البيانات ، معظم البرامج التجارية بقيت تُبرمج عن طريق الكوبول.

إلى نهايات اعوام 1960 م حاول كثير من مطوري البرامج تطوير وتعزيز الخادمات الصعبة وتوزيع البرامج كان عادة ما يتاخر بالإضافة إلى الاسعار التي تجاوزت الحد المعقول والمبيعات كانت تنتهي بسرعه لذلك بدأ الناس يدركون أن تطوير البرامح أكثر تعقيدا مما يتصورون.

توصلت الابحاث في الـ 1960 م إلى أن تطور البرمجة المركبة تقترب من كتابة البرامج السهلة والمفهومة للإختبار والتجربة وأسهل في التحديد من مبيعات البرمج الكبيرة مع مختلف التقنيات ..

واحدة من العديد من النتائج الحقيقة من هذه الأبحاث طانت تطوير لغة البرمجة Pascal عن طريق البروفيسور Niclaus Wirth وسماها باسم العالم الرياضي والفيلسوف باسكال قبل سبعة عشر قرن .

Pascal

باسكال صممت لتدريس البرامج المركبة في البيئات الاكاديمية وبسرعة أصبحت لغة البرمجة المفضلة لدى معظم الكليات . تفتقر باسكال للعديد من من الهيئات التي تحتاجها لجعلها أكثر استخداما في التطبيقات التجارية والحكومية وفي المعاهد لذلك ليست واسعة الإستخدام في هذه البيئات ..

Ada

لغة البرمجة Ada طورت تحت كفالة اقسام الولايات المتحدة للحماية(DOD) طوال عام 1970 وبدايات 1980م .. المئات من اللغات المختلفة بدات في استخدام مبيعات DOD التي سيطرت بضخامة وتحكمت في انظمة البرامج.. أرادت ال DOD لغة منفردة تلبي معظم احتياجاتها.

سميت Ada بهذا الإسم نشبة إلى السيدة Ada ابنة الشاعر Loard Byron ، الشيدة بدأت ك تابة لاول برنامج كمبيوتر عالمي بدايات 1800 م لتحليل تصميم أجهزة الكمبيوتر المحركة عن طريق تشارلز بابوج . واحدة من أهم القدرات ل Ada تسمى multitasking أي تسمح للمبرمجين أن يقرروا ما إذا كان يسمح لهم بكتابة البرامج بتفاعل متماثل أم لا ..

1.9.3 لغات Visual Basic و Visual C++ و#C و BASIC

BASIC(Beginners All-Purpose Symbolic Instruction Code)

طورت في أواسط الستينات في كلية Dartmouth كمعنى لكتابة البرامج البسيطة ، كانت البيسك في البدايات سببا لتعود المبتدئين على تقنية البرمجة.

لغة المايكروسفت فيجوال بيسك انتجت في التسعينيات تطويرا لتطبيقات المايكروسفت ويندوز وأصبحت واحدة من اهم لغات البرمجة في العالم .

تطوير أدوات المايكروسفت هو جزء من استارتيجية مشتركة وواسعة لتوحيد الإنترنت والوب لتطبيقات الكمبيوتر. هذه الإستراتيجية نفذت في مايكروسوفت . نت التي تعطي المطورين مقدرة على إنشاء وتشغيل التطبيقات في الكمبيوتر المنفذ عبر الإنترنت. هناك 3 لغات برمجة اوليه في مايكروسفت وهي VB.Net (اقتبست من

البيسك الأصليه) و ++VC(أقتبست من ++C) والسي شارب #C (لغة جديدة اقتبست من C++,Java التي طورت خصيصا لدوت نت)

استخدم المطورون دوت نت ليصبح باستطاعتهم كتابة برامج تجمع بين أغراضهم الشخصية مع ما كتب بأي لغة من الدوت نت.

إلى هنا و الكلام عند اللغات و المبرمجين سهل و بسيط، تسألهم ما هو البرنامج فيجيبون فوراً:

مجموعة من الأوامر لتأدية وظيفة معيّنة.

مُدخلات ، تحصل عليها -------> عمليات -----> تنتج منها ، مُخرجات

الإنسان بطبعه لا يفكّر في العمليات التي تحدث دون أن يفكّر في المعلومات أو الأشياء التي حدثت عليها أو صدرت منها هذه الأفعال. لا يمكن أن تكون الجملة الفعلية صحيحة دون فاعل أو مفعول.

إذاً ما زالت طريقة البرمجة بطريقة سلسلة العمليات المتلاحقة بعيدة عنّا و عن تفكيرنا.

لذا قام المبرمجون بإيجاد الحل المناسب و هو الجيل الرابع من لغات البرمجة.

- **لغات الجيل الرابع** :

تحدّثنا عن لغات البرمجة التي سبقت لغات الجيل الرابع، و قلنا أيضاً إنها جميعاً تشترك في صفة إنها تقوم على مبدأ التحدّث بالطريقة التي يفهمها الكومبيوتر.

و يجذر بالذكر أن جميع لغات الأجيال الثلاثة السابقة إذا تمّت ترجمتها على نظام تشغيل، لأنها لا تعمل إلا على هذا النظام.

مثلا إذا قمنا بكتابة برنامج بلغة C و قمنا بعمل ترجمة له (Compilation) على نظام التشغيل ويندوز، فان هذا البرنامج من المستحيل بأي حال من الأحوال أن يعمل على نظام الماكنتوش. ففي هذا الحالة أننا نأخذ البرنامج و نضعه على جهاز ماكنتوش و نقوم بترجمة مرة أخرى.

إذاً أي برنامج يتصرّف هكذا ؟

البرنامج --------> المفسّر/المترجم --------> لغة الآلة

ولأن المترجم متصل مباشرة كما ترون بلغة الآلة فإنه يعتمد عليها. و يختلف من نظام تشغيل إلى آخر.

لذا قال المبرمجون أنهم سيتخيّلون وجود آلة! تكون هذه الآلة محددة و أطلقوا عليها اسم (Virtual machine) أو الآلة التخيّلية. و هم يكتبون برامجهم حسب هذه الآلة و ليس حسب نظام تشغيل محدد هكذا:

البرنامج --------> المفسّر/المترجم --------> الآلة التخيلية --------> لغة الآلة

نلاحظ الآن أن المترجم ابتعد عن لغة الآلة، و أصبح يترجم بناءً على آلة تخيلية.

فما الذي نستفيده ؟
البرنامج سيعمل على أي جهاز عليه الآلة التخيلية مهما اختلف نظام التشغيل دون الحاجة لإعادة الترجمة

من لغات الجيل الرابع: C++ and Java و Oracle و Visual Basic و غيرها، وهذه اللغات تعرف بإنها لغات تعتمد على البرمجة الشيئية.

- **أنواع البرمجيات** Types Programming
  - نظم التشغيل OPERATTING SYSTEMS
  - المترجمات COMPILERS
  - التطبيقات APPLICATIONS
  - نظم إدارة قواعد البيانات DATA BASE MANAGEMENT SYSTEM

1.10 نظم التشغيل OPERTING SYSTEMS

عبارة عن مجموعة متكاملة من البرامج التي تنتجها شركات الحاسبات بهدف إخضاع الكيان الآلي لتنفيذ برامج المستخدم دون تدخل مباشر في أداء الآلات، وهي نظم ضرورية للحاسبات الكبيرة والصغيرة سواء بسواء.

وتعتبر نظم التشغيل بمثابة الروح للحاسب فدونها يستحيل إجراء أي مع الجات أو قيام الكيان الآلي منفردًا بأي عمل ذا فائدة على الإطلاق، وقد يظن بعض الذين لم يسبق لهم التعامل مع

الحاسبات أن نظم التشغيل ليست شيئا جوهريا في الحاسبات خاصة من يتعاملون مع حاسبات الجيب الصغيرة الدقيقة ، يظنون لا تحتوي على نظم تشغيل ، وهذا صحيح إلى حد ما وفق المفاهيم الحديثة لنظم التشغيل لكن داخل الذاكرة ROM مسجل بعض البرامج الصغيرة التي تؤدي الوظائف المنوط بها حاسب الجيب .

1.10.1 أنواع نظم التشغيل

- **نظم تشغيل الحاسبات الشخصية**

1- نظام تشغيل المعالج 280 ، والمعروف باسم
CP/M' CONTROL PROGRAM FOR MICRO COMPUTERS' .

2- نظام تشغيل الحاسبات المتوافقة مع IBM المعروف باسم MS DOS ، ' MICROSOFT DISK OPERATING SYSTEM ' , وهذا النظام شاع استخدام ه لسهولته وتطويره الدائم بواسطة شركة ميكروسوفت بدأ بالإصدار رقم 1 حتى الإصدار رقم 6 .

3- نظام التشغيل للحاسبات الشخصية من شركة أي بي إم ، والمعروف باسم OS/2 " OPERATING SYSTEM ، والذي ابتكرته شركة IBM للعمل على أجهزتها وهو متوافق مع نظام التشغيل MS-DOS .

4- نظام التشغيل لأجهزة آبل مكنتوش MAC OPERTING SYSTEM ، والذي قامت بتصميمه شركة أبل لتشغيل أجهزتها والذي تم تطويره على مراحل آخرها نظام التشغيل 8 .

- **نظم التشغيل للحاسبات المتوسطة**

أهم ما يميز الحاسبات المتوسطة هي قدرتها على التعامل مع أكثر من مستخدم في نفس الوقت ، وكذلك التعامل مع أكثر من جهاز ، ومن ثم فإن العلاقة بينهما هي جهاز أساسي يسمى HOST ، وجهاز فرعي يسمى TERMINE .
ولقد كان نظام التشغيل يونيكس UNIX الذي صممته معامل بل BELL LAB التابع لشركة AT&T الأمريكية أول نظام تشغيل لحاسبات متوسطة الحجم ، وقد قامت شركات عديدة بالاعتماد على هذه النظام في إصدار وتطوير عدة إصدارات من نظام يونيكس للعمل على أجهزتها مثل NCR ، DATA GENERAL، ICL ، DIGITAL، UNSIS .

- **نظام تشغيل الحاسبات العملاقة**

يتضح من اسم وحجم الحاسبات العملاقة حاجتها إلى نظام تشغيل قوي يتناسب وعدد المستخدمين الذي يتعدى الآلاف في بعض الأحوال ، وكذلك إدارته لحجم ضخم من البيانات والمعلومات ، وكانت شركة IBM من أوائل الشركات التي عملت في هذا المجال .
إن أهم خصائص نظم تشغيل الحاسبات العملاقة هو :
1- العدد الكبير جدا من المستخدمين والنهايات الطرفية .

2- تعدد استخدام وتنفيذ البرامج في نفس الوقت MULTI TASKING.
3- الذاكرة الافتراضية VIRTUAL MEMORY ، وذلك لعدم كفاية الذاكرة الرئيسية على مواجهة الحجم الضخم من العمليات والتعليمات سواء بالنسبة للمستفيدين أو بالنسبة للبرامج ونظام التشغيل .

1.10.2 مهام نظام التشغيل:

يتولى نظام التشغيل ما يلي:

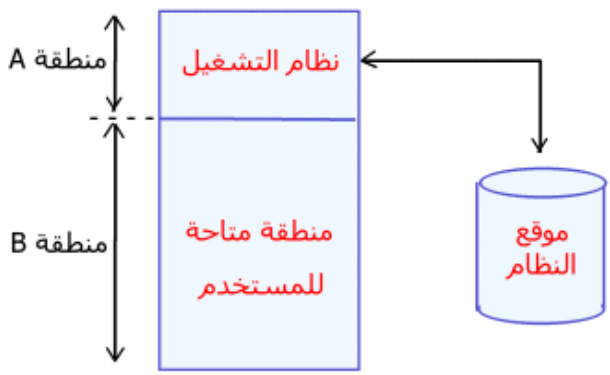
1. متابعة ومراقبة الموارد الآلية والبرمجية للنظام.
2. يشرف ويوزع الموارد على المهام .
3. يتابع تنفيذ البرامج والتنسيق بين الموارد المختلفة.
4. يستعيد الموارد متى أتم الحاسب تنفيذ المهمة.
5. تنظيم وتحميل البرامج إلي الحاسب لضمان الاستغلال الأمثل للموارد وضمان الرد .
السريع على تساؤلات المستخدم.
6. يفرض سيطرته على معدات الإدخال و الإخراج ويختار منها ما يناسب الإيعاز المحدد في البرنامج.
7. يحمي البرامج والمعدات والبيانات من التدخل الخاطئ لمستخدم ليس له الصلاحية في التعامل مع البيانات.
8. يستدعي إلى الذاكرة الأساسية البرامج والروتينات المكلفة بإجراء العمليات الحسابية.
9. يقدم لمستخدم الحاسب رسائل إرشادية أو إنذارية ويحدد الخطأ.
10. يرصد أداء الحاسب ويقدم تقريرًا شاملا عن كل الأحداث التي جرت على النظام أثناء فترة معالجة البيانات وتنفيذ البرامج.
11. يتيح الاتصال المباشر بين الحاسب والمستخدم من خلاله أوامر محدد .

1.10.3 موقع نظام التشغيل في الحاسب:

1. فور تشغيل الحاسب تنتقل برامج نظام التشغيل حوالي 80 % إلى الذاكرة الأساسية وتبقى بها طالما الحاسب يعمل أما الجزء الباقي فيبقى على الأقراص حتى يستدعي للعمل، ويسمى الجزء المنقول إلى الذاكرة الأساسية البرنامج المنفذ Kernel Executive Supervisor وهو يمثل برامج السيطرة والقيادة لجميع موارد الحاسب. في حين يسمى هذا الجزء في الحاسبات الصغيرة والمنزلية Monitor وفي نظام دوس يسمى الأوامر الداخلية .

2. يقوم على نقل برامج نظام التشغيل من على الأقراص المغناطيسية إلى الحاسب برنامج صغير يسمى الشاحن المبدئي وذلك في حالة الحاسبات الكبيرة ويسمى Boot Strapفي حاسبات PC.
3. عادة وفي حالة عدم تشغيل الحاسب تتواجد برامج نظم التشغيل على الأقراص المغناطيسية على النحو التالي :
   - على الأقراص الصلبة في PC المزود بها .
   - على مجموعة أقراص في الحاسبات الكبيرة .
   - على قرص مرن في حاسبات PC الغير مزودة بأقراص صلبة.

فيما يوضحه الشكل 34-1 موقع نظام التشغيل في الذاكرة.



موقع نظام التشغيل في الذاكرة
الشكل 34-1 موقع نظام التشغيل في الذاكرة

1.11 خطوات حل مسألة باستخدام الحاسب:

عند حل أي مسألة باستعمال الحاسب الالكتروني تتم المعالجة بإتباع خطوات نبينها بإيجاز فيما يلي:

1. **تعريف وتحليل المسألة:**

إن تعريف المسألة هو عبارة عن دقة التعبير في تطبيق المسألة بحيث يجعلها معروفة ومفهومة بصورة واضحة وبدون أي غموض لجميع الأشخاص العاملين ضمن مجال الاختصاص الذي تخضع له المسألة.

أما تحليل المسألة ووضع طريقة الحل فهو أصعب المصاعب و أشق الخطوات، و من أجل الوصول إلى حل صحيح فإن كثير من القوانين والطرق الرياضية المناسبة لحل المسألة يجب أن تستعمل و لربما تحتاج أيضاً إلى تطوير هذه القوانين والطرق لنجعلها تناسب الحل في كثير من الأحيان ففي هذه الخطوة يجب تحديد:

- طبيعة المخرجات(النتائج) و تنظيم كتابتها.
- المدخلات (البيانات أو المعلومات) و تحديد نوعها و تنظيم إدخالها إلى الحاسب الالكتروني.
- طرق الحل المناسبة و تقييمها بما يتلاءم مع كيفية تنفيذها بالحاسب الالكتروني و في ضوء ذلك يتم اختيار الحل الأفضل.
- برمجة الحل خطياً:

بعد اختيار طريقة الحل المثالية و تحديد كل ما تشمله من علاقات رياضية، يتم التعبير عنها على شكل خطوات متسلسلة ومترابطة منطقياً، دقيقة الوصف تؤدي إلى الوصول إلى حل المسألة. و هذه الخطوات المتسلسلة تعرف بخوارزمية المسألة Algorithm of the Problem و يمكن تمثيل هذه الخوارزمية بعد إيضاح جميع التعليمات والأوامر المتسلسلة التي يراد تنفيذها في كل خطوة بمخطط وصفي تسلسلي يدعى بمخطط سير العمليات Flowchart وذلك باستخدام مجموعة من الأشكال الاصطلاحية الرمزية. إن كلمة Algorithm مشتقة نسبة إلى العالم العربي المشهور الخوارزمي الذي قام بوضع أسس حل المسائل بشكل تتابعي.

- برمجة الحل باستخدام إحدى لغات البرمجة:

إن مخطط سير العمليات هو عبارة عن تخطيط تصوري مساعد سهل الملاحظة بالنسبة للمبرمج و لكنه غير مفهوم عند الحاسب الالكتروني، لذلك فإن طريقة الحل الممثلة بمخطط سير العمليات يجب أن تكتب بإحدى لغات الحاسب التي يفهمها و التي تتلاءم مع حل المسألة.

و يلي ذلك كتابة البرنامج على الوسط الخارجي المناسب و إدخال البرنامج إلى الحاسب و البرنامج الناتج من هذه الخطوة و المكتوبة بإحدى اللغات الرمزية أو الراقية يسمى بالبرنامج المصدري source program.

2. تتمل في دور الحاسب نفسه في حل المسألة وتتكون من عدة خطوات:

- إدخال البرنامج المصدري إلى الحاسب عن طريق وحدة الإدخال.
- ترجمة البرنامج المصدري:

بعد من كتابة البرنامج المصدري يتعين إدخاله إلى الحاسب للتأكد من صحة كتابته من جهة، ثم لترجمته إلى لغة الآلة بواسطة برنامج الترجمة الخاص في حالة عدم وجود أخطاء في البرنامج المصدري. و تمر عملية الترجمة في المراحل الآتية:

1- مرحلة التحليل المعجمي Lexical analysis:

في هذه المرحلة يتم مطابقة مفردات برنامج المصدر والعلاقات و الأسماء مع تلك المسموح بها في اللغة و اكتشاف أي أخطاء فيها.

2. مرحلة التحليل اللغوي والنحوي Syntax analysis:

في هذه المرحلة تجري عملية مطابقة تعليمات البرنامج المصدري مع القواعد اللغوية المستخدمة، و اكتشاف أي أخطاء فيها، بالإضافة إلى عملية تحويل

البرنامج المصدري إلى تعليمات و أوامر رمزية بلغة التجميع.

3.مرحلة ترجمة البرنامج إلى لغة الآلة:

في هذه المرحلة نحصل على البرنامج الهدفي object program و الذي بموجبه يمكن البدء في عملية التنفيذ، وتنتج من هذه المرحلة خطوتين أساسيتين

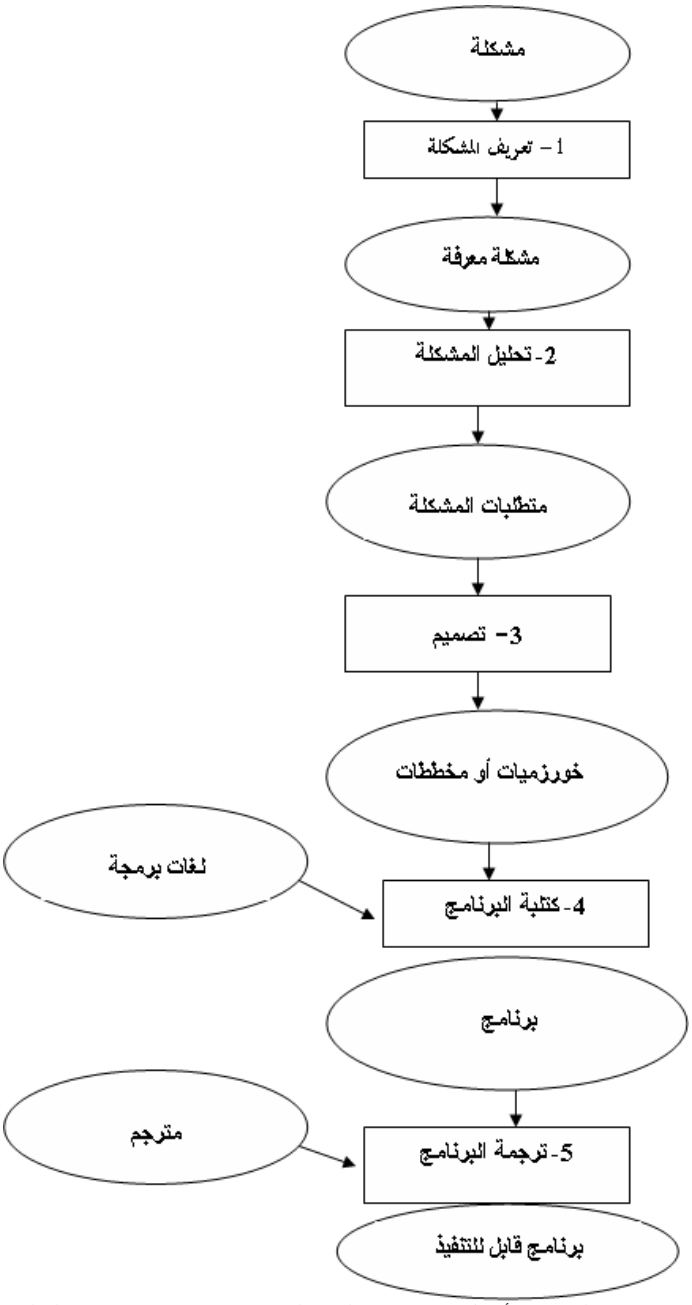
1. اكتشاف الأخطاء المعجمية واللغوية.

2. ترجمة البرنامج المصدري وتحويله إلى لغة الآلة.

- بعد تصحيح الأخطاء (Debugging) و من ترجمة البرنامج يقوم أحد برامج التحكم والمراقبة بالربط بين أجزاء البرنامج الهدفي ونخص بالذكر البرنامج الرئيسي بالبرامج الفرعية المرتبطة به.
- في حالة نجاح الخطوات السابقة يتم خزن البرنامج الهدفي وخزنة في موضعه في الحاسب. ليصبح جاهز للتنفيذ.
- تجربة البرنامج و تنفيذه :

بعد الحصول على البرنامج الهدفي، تتم تجربته للتأكد من صحته منطقياً وذلك باستخدام عينة من المعطيات الاختيارية Test Data فإذا ثبت صحة طريقة الحل بمطابقة النتائج الخارجة من الحاسب مع النتائج التي تم الحصول عليها يدوياً على سبيل المثال، يمكن تنفيذ البرنامج على المعطيات الحقيقية.

بعد من المرحلة الثانية تكب الملاحظات والتعليقات المختلفة في المناسبة داخل البرنامج وذلك لإغراض التوثيق والشكل 35-1 يبين جميع الخطوات المشروحة في هذا البند.



شكل 35-1 يبين الرموز والأشكال الاصطلاحية العامة المستخدمة في رسم خرائط سير العمليات لحل مسألة معينة

تمارين الفصل

1. ما هي أهم مزايا الحاسب من غيرة من وسائل معالجة البيانات.
2. ما هي الفرق بين لغات البرمجة الراقية والدنيا ؟
3. أكمل الفراغات الآتية :

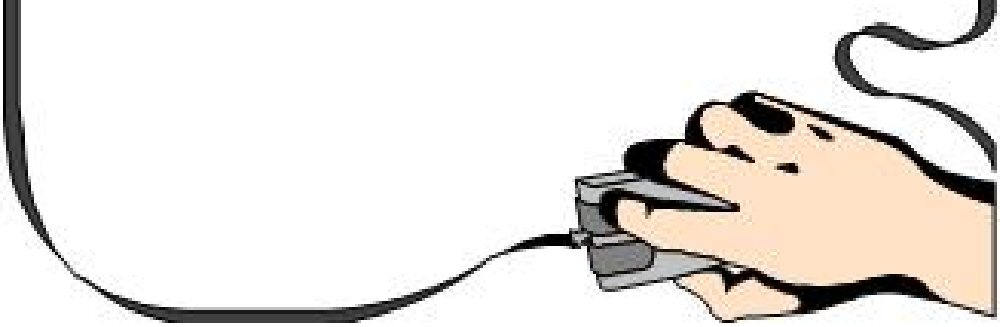
- دور المبرمج يتلخص في .....
- البرنامج المكتوب بـ..... يجب ترجمته أولاً إلى لغة الآلة.
- هناك نوعان من الأخطاء في البرنامج ينبغي التعرف عليها وتصحيحها وهي ..... ، .....
- المرحلة التي يتم فيها ترجمة البرنامج المصدري إلى برنامج هدفي تسمى .....

Chp2

الخوارزميات وخرائط سير العمليات

في نهاية هذا الفصل سوف تتعلم :

- **الخوارزميات.**
- **خصائص الخوارزميات.**
- **أنواع خرائط سير العمليات.**
- **خرائط سير البرامج.**



2.1 مقدمة

NTRODUCTION

إن المشكلات التي يمكن حلها بواسطة الحاسوب كثيرة ومتعددة الأنواع. فمنها تطبيقات رياضيه تؤدي حسابات رقمية دقيقة والأخرى تقوم بمعالجة الصور و الرسومات والأشكال مثل النقاط و الخطوط و أخرى تعتمد على معالجة النصوص و تستعمل الحروف كوحدة أساسية في معالجتها.

والسؤال المطروح هنا هو كيفية استعمال الحاسبات لحل هذه المشكلات بأنواعها وأهدافها المختلفة. فكما ذكرنا في الوحدة السابقة أن الحاسوب يقوم بإجراء العمليات التي يقوم المبرمج بإدخالها. فلا بد من وضع حل للمسألة يترجم لاحقا إلى برنامج يقوم الحاسوب بتنفيذه. ويتميز هذا الحل بأنه يتكون من مجموعة من الخطوات والعمليات المتسلسلة والمرتبة التي تؤدي في النهاية إلى حل لتلك المسألة. ويطلق على هذه الخطوات المتسلسلة اسم الخوارزمية.

2.2 الخوارزميات:

Algorithm

الخوارزمية عبارة على مجموعة من التعليمات التي تعبر عن المعالجة بطريقة متسلسلة مضمون نهايتها بعد عدد معين من الخطوات. مع الالتزام بالنتيجة الصحيحة لكل توقع من التوقعات التي يمكن أن تنتج. فأول خطوة حقيقية لوضع أسس نظرية الخوارزميات الحديثة جاءت من نظرية في المنطق أثبتها العالم الألماني كورث هيدل

عام 1931 كان مضمون هذه النظرية أن بعض المعضلات الرياضية لا يمكن حلها من بفئة معينة .ومن هنا جاءت فكرة البحث عن ما إذا كان هذا الشيء ينطبق على الفئات الأخرى وأعطت هذه النظرية دفعا كبيرا لبحث وتحليل وإعادة صياغة مفهوم الخوارزمية .

الجدير بالذكر أن الأعمال الأساسية في هذا المجال كانت منشورة في منتصف 1930 من قبل العلماء تيورنج وتشرتش وبوست.

شكلت هذه النماذج التي وضعها هؤلاء العلماء (آلة تيورنج-الة بوست – ونماذج الدوال التكرارية لتشرتش) التصورات الأولى لمفهوم الخوارزمية كانت الفرضية المصاغة التي هولاء العلماء الثلاثة كل على حدة قد أثبتت تكافؤ هذه النماذج فيما كان أهم نتائج من هذه الأعمال هو إثبات استحالة وجود حل للكثير من المسائل .

في عام 1950 ادخل العالم كولماجوروف إضافات جديدة على نظرية الخوارزميات مستفيدا من أعمال مواطنه ماركوف(عالمان روسيان).

ففي مجال الحاسب، الخوارزميات تعبر عن طريقة معالجة يمكن تنفيذها بواسطة الحاسبات عوضا على الإنسان. والخوارزميات لفظ يمكن أن يطلق على الخطوات المكتوبة باللغة العربية أو الانجليزية أو باستخدام رسوم توضيحية تمثل الخطوات وتسلسلها أو يمكن أيضا أن تطلق على البرامج التي تكون مكتوبة بلغة من لغات الحاسوب.

2.2.1 خصائص الخوارزميات

يهدف تصميم الخوارزميات إلى حل جميع جوانب المشكلة. ويمكن تصميم عدة خوارزميات لحل مشكلة واحدة. وتتميز الخوارزميات ببعض الصفات من بينها:

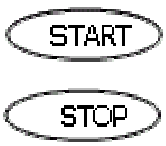
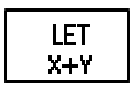
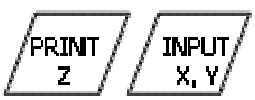
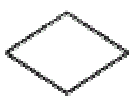
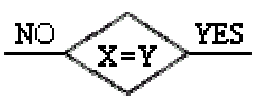
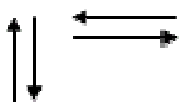
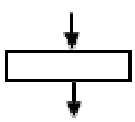
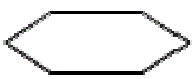
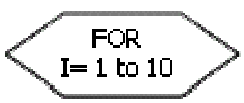
- الدقة (خطوات الخوارزمية يجب أن تكون معرفة وواضحة)
- الفعالية (الوصول إلى حل صحيح للمشكلة)
- منتهية (الوصول إلى حل صحيح للمشكلة بعد عدد معين ومحدود من الخطوات)

بعض الخوارزميات تتمكن من الوصول إلى الحل في زمن قصير ويمكن للبعض الآخر أن يأخذ زمنا أطول. والخوارزمية التي لا تنتهي إلى حل فلا تعتبر خوارزميه، مثلا طباعة الأعداد الحقيقية واحد تلو الأخر ليس بخوارزمية.

2.3 مفهوم خرائط سير العمليات:

هو تمثيل مصور للخوارزمية يخفي تفاصيل الكتابة لإعطاء صورة عامة. يوضح كيفية حل المشكلة من البداية إلى النهاية. وغالباً ما تكون استخراج الخوارزمية من خريطة سير العمليات أسهل بكثير من كتابة الخوارزمية مباشرة.

و عند رسم خريطة سير العمليات لمسألة معينة فإننا نسـ تخدم مجموعة من الأشكال الرمزية الاصطلاحية المبينة في الشكل 1-2:

الرمز	الحدث الذي يمثله	مثال
	حدث طرفي Terminal لبيان بدء (Start) أو انتهاء (Stop) خريطة سير العمليات	
	عملية حسابية (Process)	
	إدخال/ إخراج INPUT \ OUTPUT لبيان إدخال/ إخراج معلومات من/ إلى الحاسب	
	اتخاذ قرار Decision	
	اتجاه تدفق (سريان) Flow line	
	تكرار أو دوران Loop	

شكل 1-2 أهم رسومات المستعملة في المخططات

2.3.1 فوائد استخدام خرائط سير العمليات

من أهم فوائد استخدام خرائط سير العمليات قبل كتابة البرنامج لمسألة ما، ما يأتي:

1. تمكن المبرمج من الإلمام الكامل بالمسألة المراد حلها و السيطرة على كل أجزائها بحيث تساعده على اكتشاف الأخطاء المنطقية (Logic Error) و التي تعتبر من أهم الأخطاء التي تجهد المبرمج.
2. تساعد بيسر و سهولة على تعديل البرامج الموضوعة بمجرد النظر.
3. يعتبر الاحتفاظ برسوم خرائط سير العمليات لحلول مسائل معينة أمراً مهماً إذ يكون مرجعاً عند إجراء تعديلات عليها أو استخدام ها لحل مسائل أخرى مشابهة دون الحاجة إلى الرجوع إلى المبرمج الأول باعتبار أن الحلول الأولى قد صيغت في خطوات واضحة بسيطة و مفهومة.
4. توفير وسيلة مناسبة ومساعدة في كتابة البرامج ذات التفرعات الكثيرة.

2.4 التحكم في المعالجة وأنواعها

ترتيب الخطوات مهم جدا لان الاختلال بترتيب خطوات الخوارزمية لا يؤدي إلى النتيجة المطلوبة. كذلك ليس لكل التوقعات نفس الخطوات و الترتيب. ففي بعض الحالات يتم الاختيار بين مجموعتين من الخطوات وأحيانا أخرى تكون مجموعة من الخطوات مكررة. فهناك ثلاثة أنواع من التحكم : التسلسل، الاختيار و التكرار

أ- التسلسل (Sequence): تنفذ الخطوات بشكل متتالي، الواحدة بعد الأخرى، حسب ظهورها في الخوارزمية مثل خوارزمية حساب المعدل.

ب- الاختيار (Selection) : يمكن في هذا النوع الاختيار بين مجموعتين مختلفتين من الخطوات حيث انه لا يمكن تنفيذ إلا واحدة منهما فقط. مثلا حساب القيمة المطلقة لعدد ما.

إذا كانت س أكبر من 0 أحسب
القيمة المطلقة = س
وإلا نفذ
القيمة المطلقة =- س

في هذا المثال لا يمكن تنفيذ الخطوتين الأولى والثانية معا بل يمكن تنفيذ إحداهما فقط (الخطوة الأولى أو الثانية حسب قيمة س).

جـ- التكرار (Iteration) هذا النوع من التحكم يقوم بتكرار مجموعة من الخطوات عددا من المرات معتمدا على قيد أو شرط معين. مثال حساب المعدل لجميع طلبة الشعبة وعددهم 20.

أبدا بالطالب رقم = 1

بينما (رقم الطالب < 20) نفذ

(

خطوة 1. إدخال الدرجات

خطوة 2. حساب عدد الدرجات = ن

خطوة 3. حساب المجموع الدرجات = مجموع

خطوة 4. حساب المعدل = مجموع/ن

خطوة 5. إخراج المعدل

انتقل للطالب التالي

)

هناك أربع أنواع من العمليات:

- إدخال البيانات في الذاكرة
- عمليات معالجة البيانات
- عمليات التحكم
- إخراج البيانات من الذاكرة

كما يمكن استعمال المخططات لتصميم البرامج عوضا على الخوارزميات.

2.5 البرامج و أنواعها :

الخوارزميات القابلة للتنفيذ بواسطة الحاسوب تسمى برامج و تكتب بواسطة لغة برمجة. وتنقسم البرامج بصفة عامة إلى نوعان هي : برامج النظم ، برامج تطبيقية تكتب هذه البرامج بواسطة لغات برمجة و مترجمات

جدول 2-1

برامج نظم (System Programs)		برامج تطبيقية (Applications Programs)		
نظم التشغيل (OS)	نظم إدارة قواعد بيانات (DBMS)	برامج تحكم في الاتصالات (CCP)	برامج عامة (Generic Applications)	برامج خاصة (Specific Applications)
• DOS • WINDOWS 95, 98, 2000 • Unix • Linux • MacOS	• Oracle • Access • DB2 • ngres,...	• TCP/IP • HTTP • FTP • TELNET • ...	MS OFFICE Applications	حساب المعدل، حساب الرواتب

2.6 تصنيف خرائط سير العمليات:

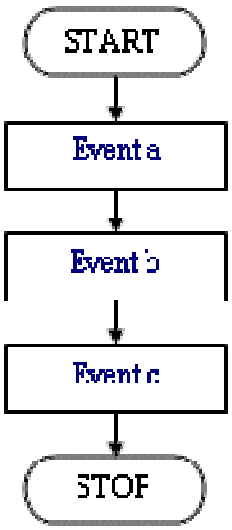
- خرائط الدوران البسيط (Loop Flowchart).
- خرائط التتابع البسيط (Simple sequential Flowchart).
- خرائط التفرع (Branched Flowchart).
- العداد Counter:

2.6.1 خرائط الدورانات المتداخلة (Nested).

و يمكن للبرنامج الواحد أن يشتمل على أكثر من نوع واحد من هذه الأنواع. و سنتناول فيما يأتي شرح هذه الأنواع بشيء من التفصيل.

2.6.2 خرائط التتابع البسيط:

يخلو هذا النوع من التفرعات Branches و الدورانات loops ، و يكون الشكل العام لهذا النوع كما هو مبين في الشكل 2-2:



الشكل 2-2 خرائط التتابع البسيط

و كلمة Event الواردة في شكل 2-2 تعني الحدث أو العملية المطلوب تنفيذها.

مثال : أرسم خريطة سير العمليات لإيجاد مساحة و محيط دائرة نصف قطرها معلوم R.

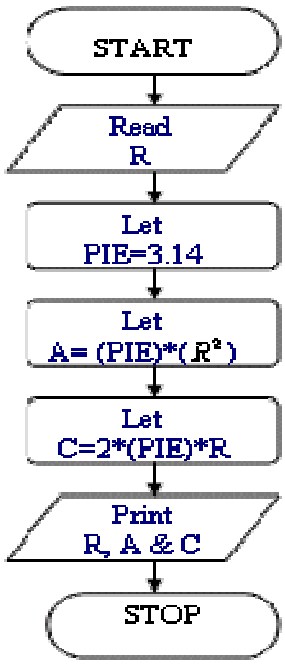
مساحة الدائرة = πR^2

محيط الدائرة = $2\pi R$

حيث π = النسبة التقريبية

وقيمتها العددية ثابتة و تساوي 3.14 بينما R متغير.

وتكون خطوات الحل المبينة في الشكل 3-2 كما يلي:



الشكل 3-2

1.ابدأ.

2.اقرأ قيمةR .

3. ضع قيمة 3.14=PIE

4. احسب المساحة(A) من المعادلة A =(PIE)*R*R.

5. احسب المحيط (C) من المعادلة C =2*(PIE)*R.

6. اطبع قيم كل من C, A, R.

7. توقف.

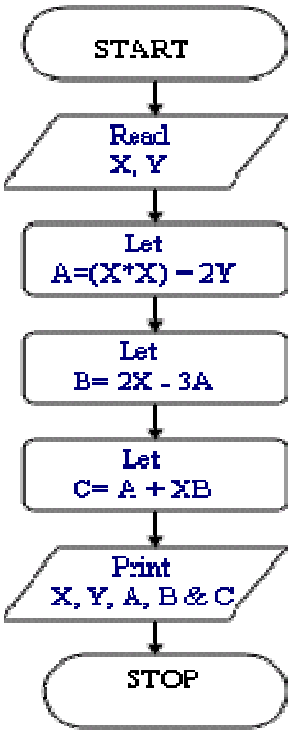
مثال: ارسم خريطة سير العمليات لحساب قيمة كل من المتغيرات C, B, A في المعادلة الآتية:

$$A = X^2 + 2Y \ldots (1)$$
$$B = 2X - 3A \ldots (2)$$
$$C = A^2 + XB \ldots (3)$$

إذا علمت أن قيم كل من X, Y معطاة (معلومة) ، ثم أطبع قيم كل من X, Y, A, B, C.

الحل: من الواضح أنه يمكننا من حساب قيمة المتغير A في المعادلة(1) لمعرفتنا بقيم المعطيات الأولية X, Y، ويمكننا من حساب قيمة المتغير B في المعادلة (2) بالاعتماد على قيمة X المعلومة لدينا وقيمة المتغير A المحسوبة في الخطوة السابقة، أما قيمة المتغير C في المعادلة (3) بالاعتماد على قيم كل من المتغيرات X, A, B وكلها معلومة.

وتكون خطوات حل المسألة كما هو مبين في الشكل 4-2 كما يلي:



الشكل 2-4

1.ابدأ.

2.اقرأ قيمة كل من X, Y.

3.احسب قيمة A من المعادلة (1).

4.احسب قيمة B من المعادلة (2).

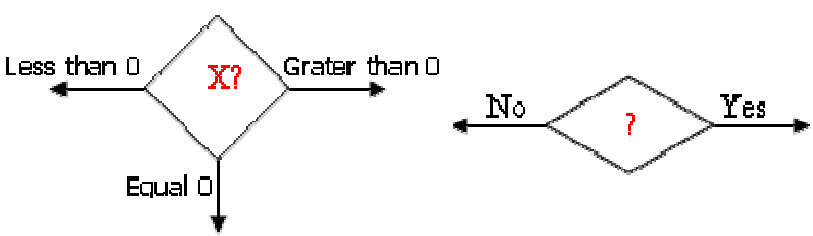
5.احسب قيمة C من المعادلة (3).

6.أطبع قيمة كل من X, Y, A, B, C.

7.توقف.

2.6.3 خرائط التفرع:

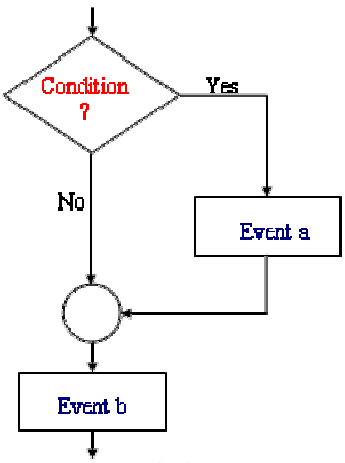
ويحدث التفرع في البرامج بسبب الحاجة لاتخاذ قرار أو مفاضلة بين اختيارين أو أكثر، وهناك أسلوبان في تنفيذ القرار(انظر شكل 2-5).



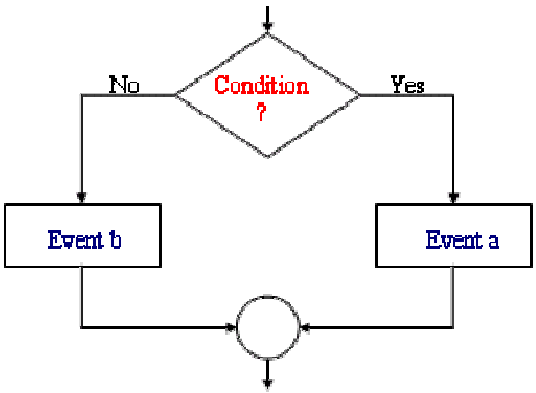
قرار ذو ثلاثة تفرعات قرار ذو تفرعين

الشكل 2-5

وبشكل عام فإن خرائط التفرع يمكن أن تأخذ إحدى الصورتين الآتيتين (انظر شكل 2-6 و الشكل 2-7).



الشكل 2-6



الشكل 2-7

يمكننا ملاحظة أن شكل 2-7 يبين أنه إذا كان جواب الشرط (Condition) YSE فإن الحدث التالي في التنفيذ يكون الحدث (a) أما إذا كان الجواب NO فإن الحدث التالي يكون الحدث(b) كما يمكننا أن نلاحظ في الشكل 2-6 أنه إذا كان جواب الشرط YSE فإن الحدث التالي في التنفيذ يكون الحدث (a) ثم يتبعه الحدث (b) أما إذا كان جواب الشرط NO فإن الحدث التالي يكون الحدث (b) مباشرة.

مثال:ارسم خريطة سير العمليات لإيجاد قيمة الاقتران F(x) المعرف حسب القاعدة التالية:

F(X) =I XI	X if X>=0
	-X if X<0

حيث كلمة (if) هنا تعني عندما.

خطوات الحل المبينة في الشكل 2-8 تكون:

1. ابدأ

2. اقرأ قيمة المتغير X .

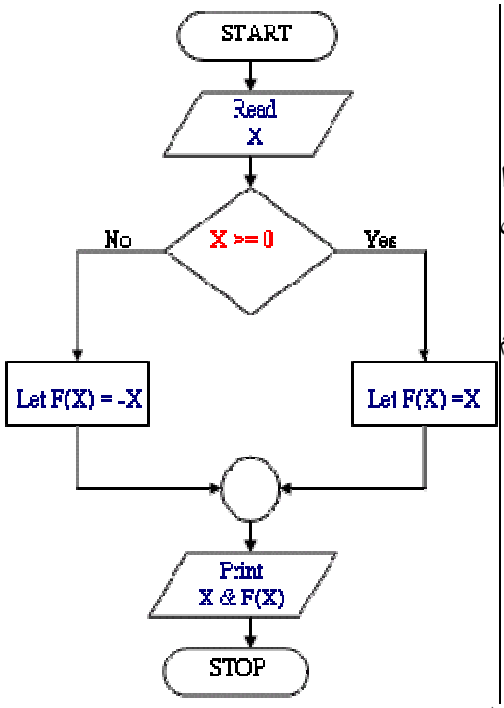
3. إذا كانت X أكبر أو تساوي صفرًا اذهب إلى خطوة(4) وإلا فأذهب إلى الخطوة(5).

4. احسب قيمة الاقتران من F(X)=X ثم اذهب إلى الخطوة(6).

5. احسب قيمة الاقتران من F(x)= -X.

6. أطبع قيمة كل من X، F(x).

7. توقف.



الشكل 2-8

مثال:ارسم خريطة سير العمليات لحساب قيمة W طبقًا للمعادلات الآتية علمًا بأن قيمة المتغير X معطاة معلومة:

$$W=\begin{cases} X^2+1 ; X>0 \\ X+5 ; X=0 \\ 2X^3-1 ; X<0 \end{cases}$$

خطوات الحل كما هي مبينة في الشكل 2-9 :

1. ابدأ.

2. اقرأ قيمة المتغير X.

3. إذا كانت X أكبر من صفر فاذهب إلى الخطوة 4 أما إذا كانت ليست أكبر من فاذهب إلى خطوة 5.

4. احسب W من المعادلة (1) ثم اذهب إلى الخطوة 8.

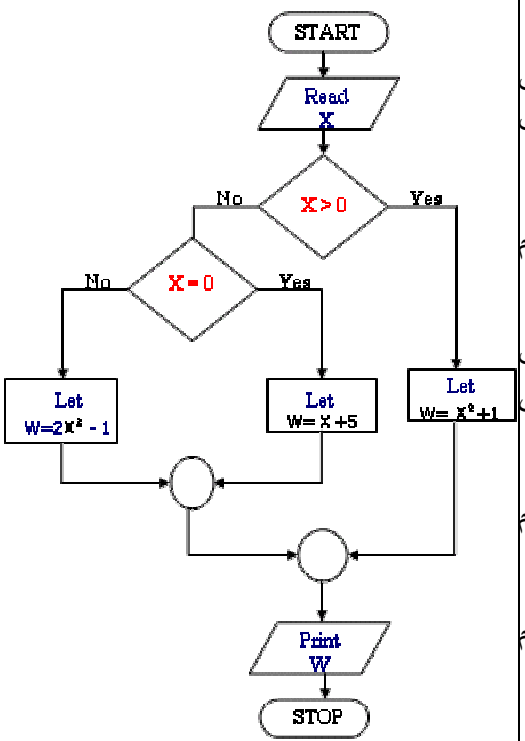
5. إذا كانت X تساوي صفر فاذهب إلى الخطوة 6 وإلا فاذهب إلى الخطوة 7.

6. احسب W من المعادلة (2) ثم اذهب إلى الخطوة 8.

7. احسب W من المعادلة (3) ثم اذهب إلى الخطوة 9.

8. أطبع قيمة W.

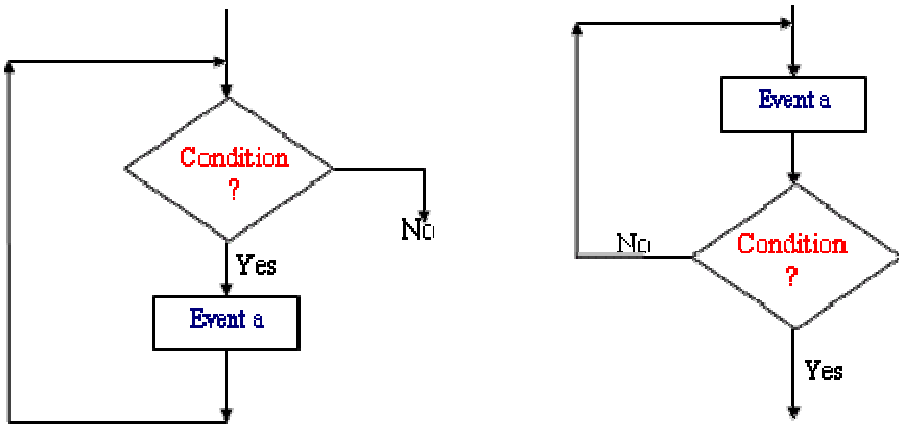
9. توقف.



الشكل 9-2

2.6.4 خرائط الدوران (التكرار) البسيط:

وهذه الخرائط نحتاج إليها لإعادة عملية أو مجموعة من العمليات في البرنامج عددًا محدودًا أو غير محدود من المرات، ويكون الشكل العام لمثل هذه الخرائط كما يلي (انظر الشكل 2-10).



الحدث (a) يتكرر تنفيذه في كل دوره حتى يصبح جواب الشرط YES.

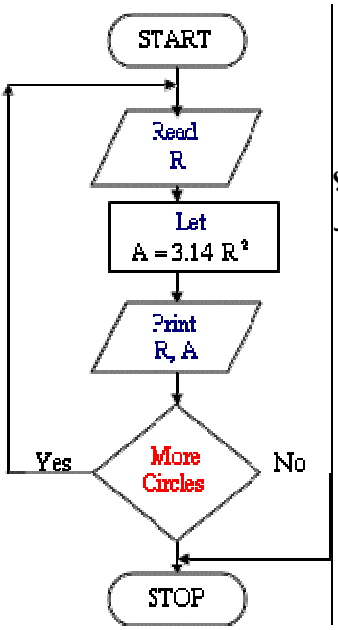
الحدث (a) يتكرر تنفيذه في كل دورة طالما كان جواب الشرط YES.

الشكل 2-10

مثال: ارسم خريطة سير العمليات لإيجاد مساحة مجموعة من الدوائر أنصاف أقطارها معلومة:

تكون خطوات الحل المبينة في الشكل 2-11 كما يلي:

1. ابدأ.
2. اقرأ نصف قطر الدائرة (R).
3. أوجد مساحة الدائرة (A).
4. اطبع قيم كل من A, R.
5. هل هناك مزيد من الدوائر؟ فإن كان نعم فعد إلى الخطوة(2) وإن كان لا فعد إلى الخطوة (6).
6. توقف.



الشكل 2-11

2.6.5 العداد (Counter):

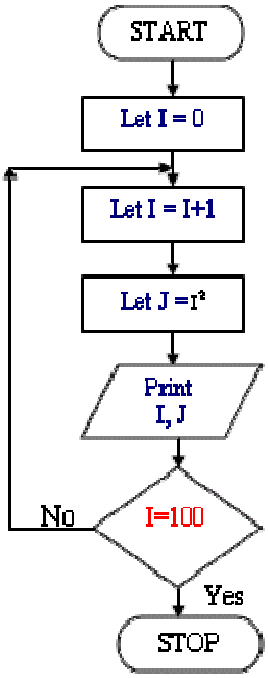
في كثير من الأحيان نحتاج في برامج الحاسب الالكتروني إلى العد Counting ، فقد نريد مثلاً أن نعد عدد كل من الطلاب والطالبات ضمن الشعبة، وقد تكون هذه العملية سهلة للإنسان لإ نها أصبحت ضمن قدراته العقلية التي يكتسبها من الطفولة، إلا أن الحاسب يحتاج إلى تصميم خوارزمية للعد Counting Algorithm تتضمن خطوات معينة إذا اتبعتها استطاع أن يعد.

ويمكن تحديد الخطوات التي يتبعها الحاسب حتى يتمكن من العد في الخطوات الأساسية:

1. اجعل العداد مساويًا للصفر.
2. اجعل القيمة الجديدة للعداد تساوي القيمة القديمة لها زائد واحد، أي أن:
   قيمة العداد (الجديدة)= قيمة العداد (القديمة)+1
3. كرر الخطوات ابتداء من الخطوة 2.

مثال: ارسم خريطة سير العمليات التي يتبعها الحاسب لطباعة الأعداد الطبيعية من 1 إلى 100 ومربعاتها.

الحل: خطوات الحل مبينة في الشكل 12-2 هي:



1. ابدأ.
2. اجعل I=0.
3. اجعل I=I+1.
4. اجعل $J = I^2$.
5. اطبع I ,J.
6. إذا كانت I=100 اذهب إلى الخطوة 7 وإلا اذهب إلى الخطوة 3.
7. توقف .

الشكل12-2

ý المجاميع الإجمالية:

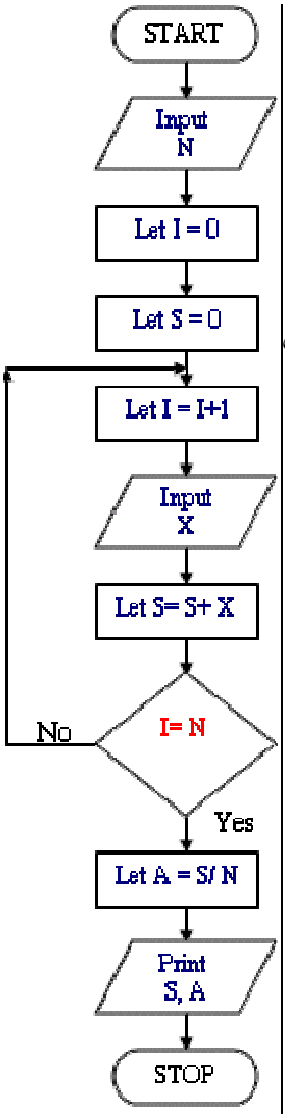
في كثير من الأحيان نحتاج في برامج الحاسب الإلكتروني إلى جمع مجموعة كبيرة من الأعداد التي تمثل معطيات ظاهرة معينة، فمثلاً قد نرغب في إيجاد الوسط الحسابي لأعمار طلاب الجامعة، ولتحقيق هذا أولاً يجب أن نحسب مجموع أعمار الطلاب، وطبعًا ليس عمليًا إعطاء رمز أبجدي لكل عمر طالب فقد تحتاج لأكثر من عشرة الآلاف رمز، في مثل هذه الحالات نصمم خوارزمية معينة للتجميع تسمى خوارزمية التجميع summers Algorithm تتضمن خطوات محددة إذا اتبعها الحاسب استطاع أن يجمع أي كمية من البيانات باستخدام متغيرين اثنين إحداهما هو المتغير الذي نجمعه والآخر هو الجمع الإجمالي (المجمع) ، ويمكن تحديد الخطوات التي يجب أن يتبعها الحاسب لتحقيق ذلك في أربع خطوات هي:

1. اجعل المجمع مساويًا الصفر.
2. ادخل قيمة واحدة للمتغير.
3. اجعل القيمة الجديدة للمجمع تساوي القيمة القديمة له زائد القيمة المدخلة للمتغير، أي أن:

قيمة المجمع الجديدة=قيمة المجمع القديمة + آخر قيمة مدخلة للمتغير.
4. كرر ابتداءًا من الخطوة الثانية.

مثال: ارسم خريطة سير العمليات لإيجاد الوسط الحسابي لأعمار طلاب شعبتك.
الحل: نفترض أن إجمالي عدد الطلاب =N و نستخدم عددًا لرقم كل طالب ونرمز له بالرمز I ونرمز لعمر الطالب بX ونستخدم مجمعًا لأعمار الطلبة ونرمز له بالرمزS ونستخدم الرمز A ليدل على معدل أعمار الطلبة.

وتكون خطوات الحل كما هو مبين في الشكل 13-2 هي:



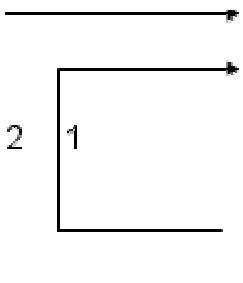
1. ابدأ.
2. ادخل إجمالي عدد الطلاب (N).
3. اجعل I=0.
4. اجعل S=0.
5. اجعل I=I+1.
6. ادخل X.
7. اجعل S=S+X.
8. إذا كانت I=N اذهب إلى الخطوة 9 وإلا اذهب إلى الخطوة 5.
9. اجعل A=S/N.
10. توقف.

الشكل 13-2

2.6.6 خرائط الدورانات المتداخلة:

في هذه الحالة تكون الدورانات داخل بعضها البعض بحيث لا تتقاطع فإذا كان لدينا مثلاً دورانان من هذا النوع (انظر شكل 14-2 فيسمى الدوران قم (1) دورانًا داخليًا (Inner Loop) بينما الدوران رقم (2) دورانًا خارجيًا (Outer Loop) ويتم التناسق في عملي مثل

هذين الدورانين بحيث: تكون أولوية التنفيذ للدوران الداخلي.



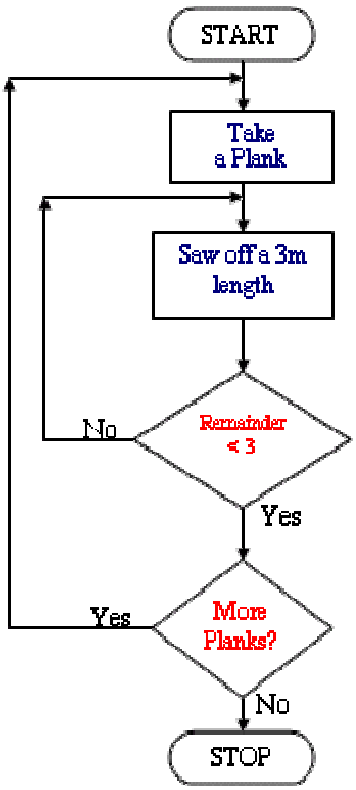
الشكل 14-2

مثال: يرغب نجار في تقطيع مجموعة من القطع الخشبية طول كل منها يزيد عن3 متر إلى قطع صغيرة طول الواحدة منها يساوي 3 متر.

ارسم خريطة سير العمليات.

خطوات الحل المبينة في شكل 15-2 هي:

ملحوظة: يلاحظ من الشكل 215 أن الدوران الداخلي يتضمن تقطيع القطعة الواحدة إلى قطع متعددة طول كل منها 3 متر بينما يمثل الدوران الخارجي تناول قطعة واحدة جديدة لتنفذ عليها إجراءات الدوران الداخلي.

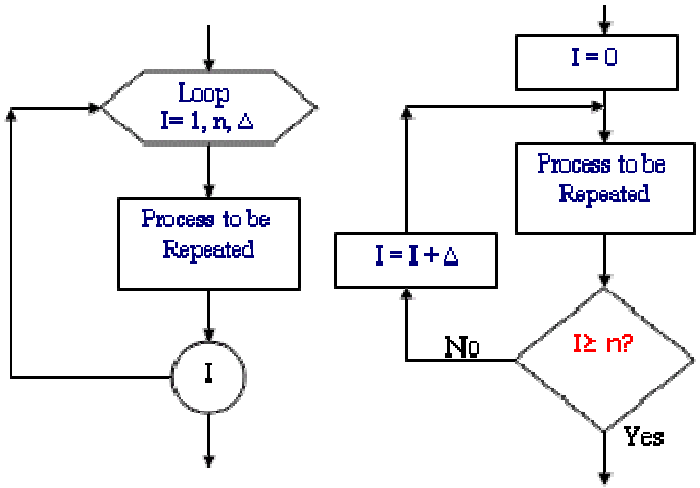


1. ابدأ.
2. خذ قطعة.
3. اقطع منها قطعة طولها 3 متر.
4. هل المتبقي يزيد ع ن 3 متر؟
   إذا كان الجواب نعم فاذهب إلى الخطوة(3).
   وإذا كان الجواب لا فاذهب إلى الخطوة (5).
5. هل هناك مزيد من القطع المراد تقطيعها ؟ إن كان الجواب نعم فاذهب إلى الخطوة(2) وإن كان لا فاذهب إلى الخطوة(6).
6. توقف.

الشكل 2-15

- **صيغة الدوران باستعمال الشكل الاصطلاحي:**

لقد عرفنا في الفقرتين السابقتين مفهوم الدوران البسيط والدورانات الضمنية ويمكننا الآن استخدم الشكل الاصطلاحي للدوران والوارد على النحو التالي:



الشكل2-16

نلاحظ في الشكل 16-2 أننا نحتاج إلى العناصر الآتية:

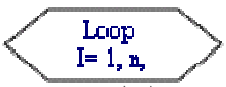
- القيمة الأولية للعداد I (هنا I=1).
- القيمة النهائية للعداد I (هنا I=1).
- القيمة النهائية للعداد I (هنا n).
- قيمة الزيادة عند نهاية كل دورة Δ .

نلاحظ في الشكل 16-2 إن إجراءات الدوران كانت تتم طبقاً للخطوات الآتية والمفصلة من قبل المبرمج:

1. أعط I قيمة أولية.
2. أتم الإجراءات المطلوب إعادتها.
3. (تقرير) إذا كانت قيمة العداد 1 وصلت إلى القيمة النهائية n اخرج إلى الخطوة التالية في البرنامج وإلا فاذهب إلى الخطوة (4).
4. زد I بمقدار الزيادة Δ .
5. عد إلى الخطوة (2).

يمكننا استبدال الخطوات المفصلة(5,4,3,2,1) في الشكل 16-2 بخ طوة مجملة واحدة مبينة في الشكل الاصطلاحي للدوران ، حيث تنفذ هذه الخطوات بصورة أوتوماتيكية من قبل الحاسب، وهذا من شأنه تسهيل عملية البرمجة واختصار عدد التعليمات في البرنامج وتجنب بعض الأخطاء.

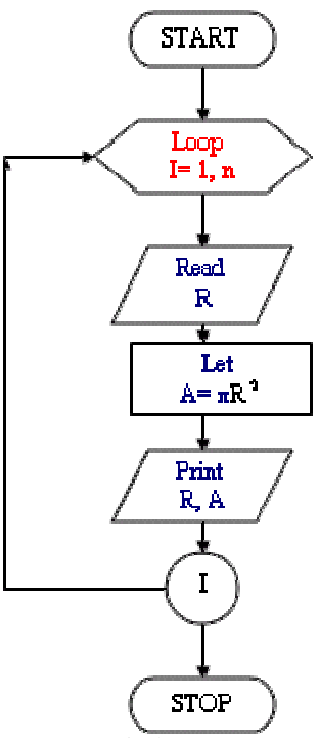
ملحوظة: تعتبر قيمة Δ تساوي 1 دائمًا إذا لم تعط قيمة أخرى بخلاف ذلك ، وفي حالة عدم ذكر قيمة Δ يصبح الشكل الاصطلاحي الوارد في الشكل 2-16 كما يلي حيث تكون قيمة Δ تساوي 1وبصورة أوتوماتيكية.



الشكل2-17

مثال: أعد حل مثال الموضح في الشكل 2-10 لإيجاد مساحة n من الدوائر باستخدام الشكل الاصطلاحي للدوران.

خطوات الحل كما هي مبينة في الشكل 2-18.

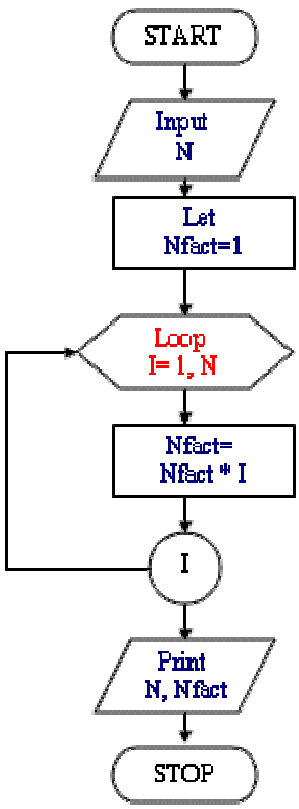


الشكل2-18

ارسم خريطة سير العمليات لإيجاد!N.

الحل:

.N!=N (N-1) (N-2) …3*2*1
فخطوات الحل كما يلي هي مبينة في الشكل 2-19 :



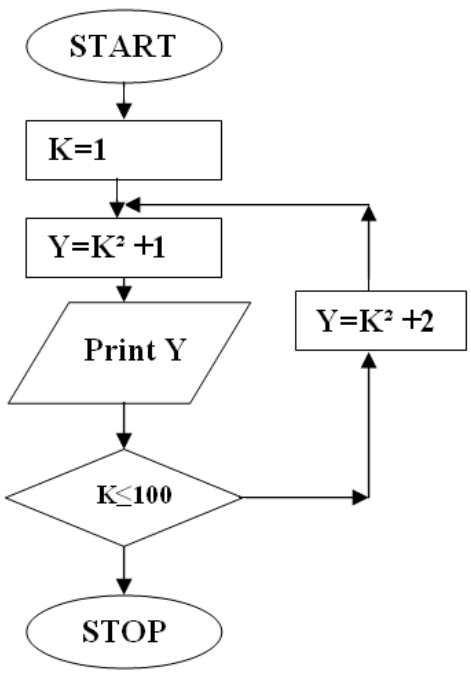
الشكل 2-19

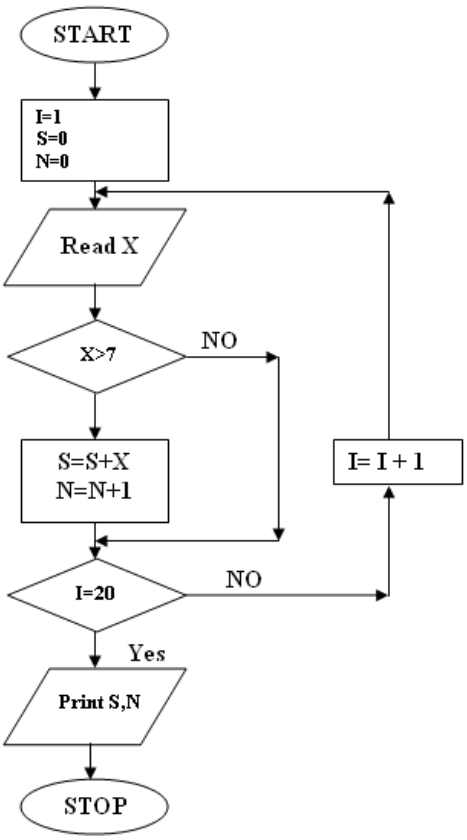
تمارين الفصل:

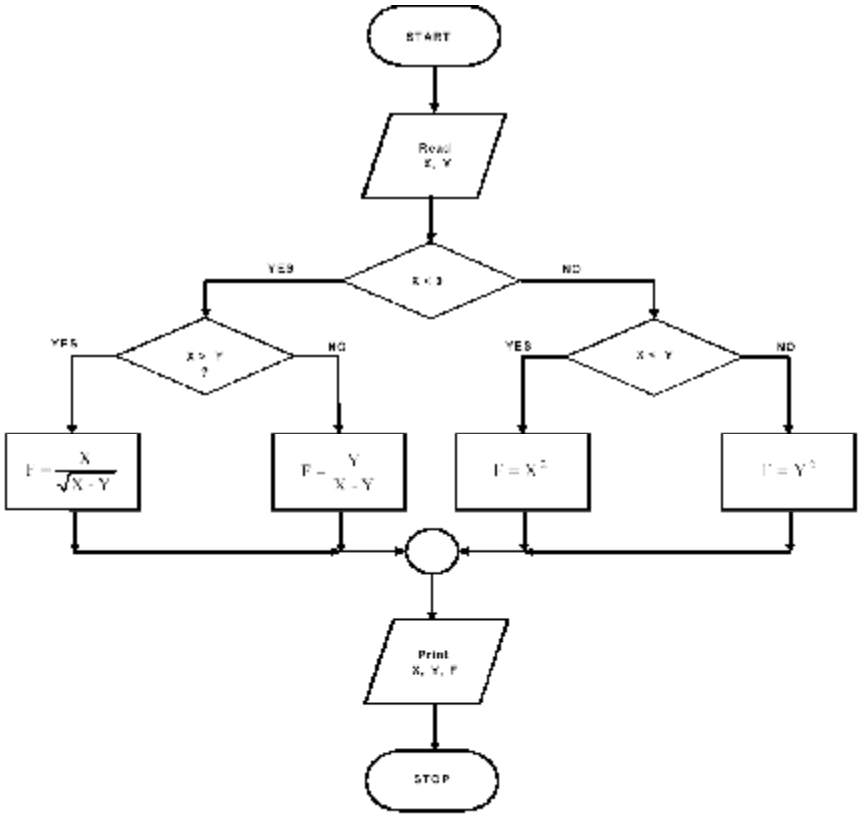
1- ما دور الإنسان في حل مسألة باستخدام الحاسب ؟
2- ما الفرق بين الخوارزمية وخريطة سير العمليات ؟
3- ما أهمية برنامج الترجمة في حل مسألة باستخدام الحاسب ؟
4- أكتب خوارزمية وأرسم خريطة سير العمليات لبرنامج يحاكي لعبة النرد؟
5- اكتب خرائط سير العمليات التي تقوم بالعمليات التالية:

- كتابة الأعداد الفردية بين 100,1 ومربعاتها ؟
- كتابة الأعداد التي تقبل القسمة على 7 وتقع بين 100,10.
- الأعداد الصحيحة من 1 إلى 15.
- حاصل ضرب عددين.
- القاسم المشترك الأعظم لعددين.
- المضاعف المشترك الأصغر لعددين .

6- صف عمل المخططات التالية مبيننا أهداف كل مخطط والنواتج النهائية التي سيطبعها الحاسب عند تنفيذ التعليمات المبينة في كل مخطط Y=5,X=2:




START
I=1
S=0
N=0
Read X
X>7
NO
S=S+X
N=N+1
I= I + 1
I=20
NO
Yes
Print S,N
STOP

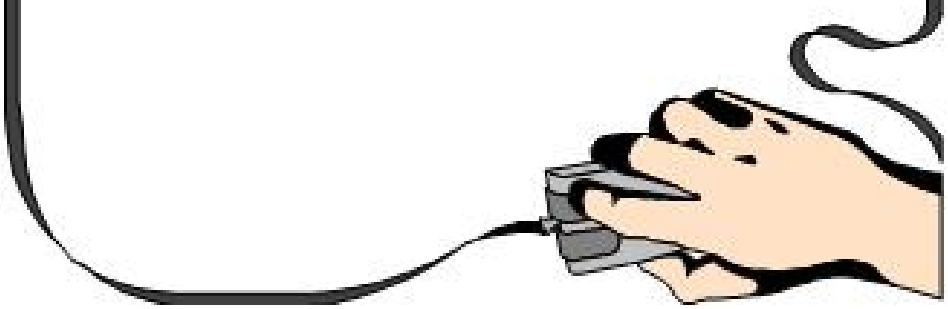

START
Read
X, Y
YES
NO
YES
NO
YES
NO
Print
X, Y, F
STOP

Chp3

أساسيات لغة Java

في نهاية هذا الفصل سوف تتعلم :

- **التعرف على لغة Java وتاريخها.**
- **العلاقة بين لغة java و(لغة c++ ، Java Script).**
- **التعرف على مترجمات java.**
- **تحميل مترجم java.**
- **صنع وترجمة وتنفيذ برامج java بسيط .**
- **عرض الخرج الناتج عن تطبيق ما باستخدام الشاشة .**



3.1 مقدمة

INTRODUCTION

فلنبدأ بالرقم صفر فهو نقطة الانطلاق الطبيعي للغة Java بدء بالعدد من رقم صفر. فصفر يرمز بلغة Java إلى إجابة غير صحيحة ، وعكس صفر يرمز إلى الإجابة الصحيحة .
فالصفر هو اصفر رقم دليل الصفيف (Array) ، والصفر هو الرمز الذي يشير إلى بداية حقل أو مجموعة أحرف (string) ، وهو يشير إلى غياب المعلومات ،وسيحاول هذا الكتاب شرح خبايا اللغة ولذة البرمجة بلغة Java .

فكثيراً ما نسمع في هذه الأيام عن لغة Java حتى يبدو وكإنها قي كل مكان حتى في المكتبات لو تمعنت النظر لوجدت كتب المكتبة مليئة بكتب Java حتى الصحف الضخمة و المجلات تجد فيها العديد من المقالات التي تتحدث عن Java .
كل هذا يجعلك تتساءل عن سبب انتشار هذه اللغة والجواب ببساطة إنها تتيح للمستخدمين إمكانية تطوير تطبيقات تعمل على الويب من اجل المخدمات والأجهزة الصغيرة كالهاتف النقال وغيرة . والآن لنبحر سوياً قي تاريخ Java .

ففي أوائل التسعينيات من القرن العشرين 1991 اخترعت لغة Java شركة Sun Microsystems ولهذا الاختراع قصة عجيبة حيث أن الشركة كانت قبل ذلك قد كلفت المهندس James Gosling.

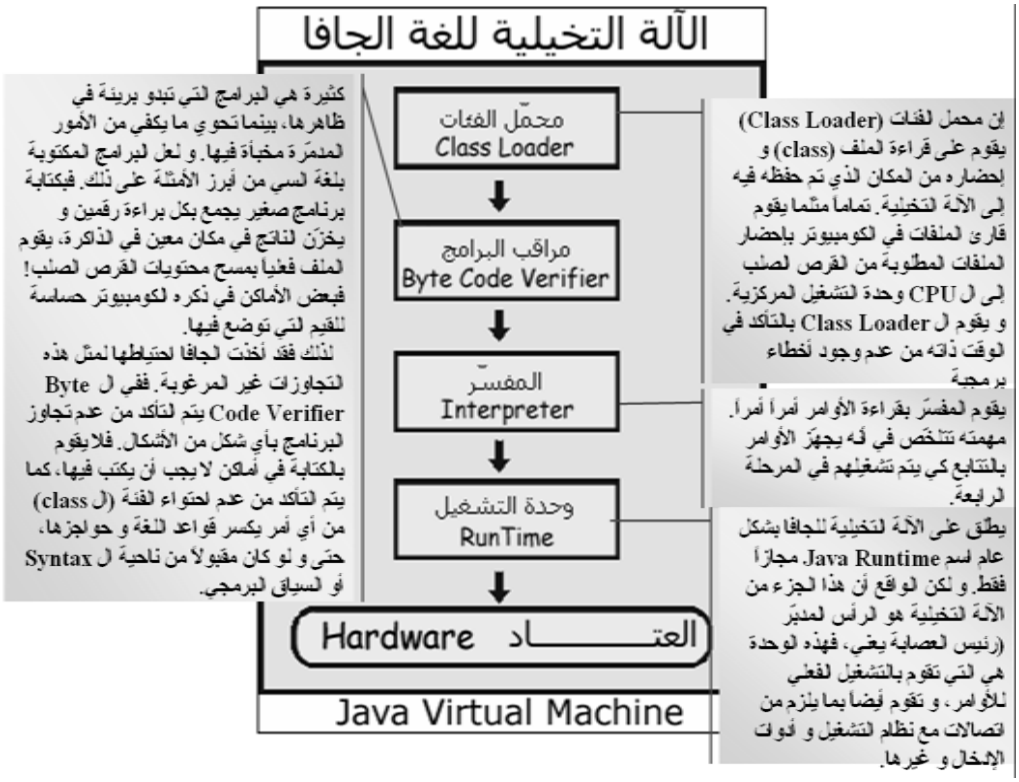

المهندس James Gosling

بوضع برامج لتشغيل الأجهزة التطبيقية الذكية مثل التليفزيون التفاعلي باستخدام لغة C++ وحينها وجد James Gosling صعوبة في التعامل مع هذه اللغة فقام هو وفريق العمل المساعد له بتطوير هذه اللغة فولدت لغة جديدة تتوافق مع احتياجاته فكانت لغة Java وقد خططت شركة Sun في تلك الأيام لاستغلال هذه اللغة الوليدة في التليفزيون التفاعلي لكي تربح المليارات وحدث نوع من البطء في مشروع التلفزيون التفاعلي ربما عن قصد من الشركات الأخرى المنافسة ـ ونتيجة لذلك فكرت شركة Sun في توقيف مشروع تطوير هذه اللغة الوليدة وتسريح العاملين في هذا المشروع أو نقلهم إلى قسم آخر ولكن حدث ما لم يكن في الحسبان حيث أنه في هذه الفترة كانت الانترنت قد بدأت في الآنتشار بسرعة مذهلة مع نزول نظام الويندوز للأسواق وحيث أن لغة Javaالوليدة التي اخترعت أصلا لبرمجة الأجهزة التطبيقية فيها من السمات ما يجعلها أكثر توافقا مع الشبكة العنكبوتية الدولية ـ الانترنت ـ فقد كان لها السبق وأضافت الكثير إلى الانترنت الذي كان قبلها مقصورا على تبادل النصوص ولكن المطورين بشركة صن ابتكروا طريقة تجعل برامج Java تعمل بسهولة في صفحات

الانترنت وغيروا الاسم الذي كان قد أطلقه عليه مبتكرها من Oak - شجرة السنديان - إلى Java ومن هنا أصبحتJava مرتبطة في شهرتها بالانترنت حيث أن برنامج جافا صغير يوضع في صفحة من صفحات موقع على الشبكة الدولية يراه الملايين في جميع أنحاء العالم في نفس الوقت وقد كان هذا لا يتوفر إلا مع Java مما أعطاها شهرة واسعة أكبر من شهرة نجوم هوليود ولحسن حظ شركة Sun أن لغة Java أكدت نفسها في المجال الذي طورت له أصلا فقد بدأ الآن التليفزيون التفاعلي في الآنتشار وما يسمى سينما المنزل والمشاهدة حسب الطلب وليس هذا فقط بل أنتشر ما هو أكثر فائدة لشركة Sun وهو الهاتف المحمول و للJava أكبر دور في برمجة البرامج التي يعمل بها في أجياله السابقة واللاحقة ولا نستغرب أن يحدث نوع من الغيرة بين شركة ميكروسوفت وشركة Sun مما دفع ميكروسوفت إلى أن تحذف ماكينة Java الافتراضية من الإصدارة الأولى للويندوز اكس بي وهذه الماكينة الافتراضية مسئولة عن عرض برامج Java على الانترنت ولكن ميكروسوفت تراجعت أمام طلب ملايين المستخدمين حول العالم فوضعتها مرة ثانية في الإصدارات اللاحقة وقد كانت قضية مشهورة تناولتها الصحف والمجلات .

وقد ساهم في شهرة Java أيضا برامجها العلمية التفاعلية التي تصلح لمعظم المناهج التعليمية في جميع مراحل التعليم، والتعامل مع الصور ، الأحياء ، الصوت ،الشبكات والحوسبة التعاونية ،وبالتالي فان لها دورا كبيرا في التعليم الالكتروني والتعليم عن بعد والفصول الافتراضية .

3.2 الآلة التخيلية للـ (JVM Java) :

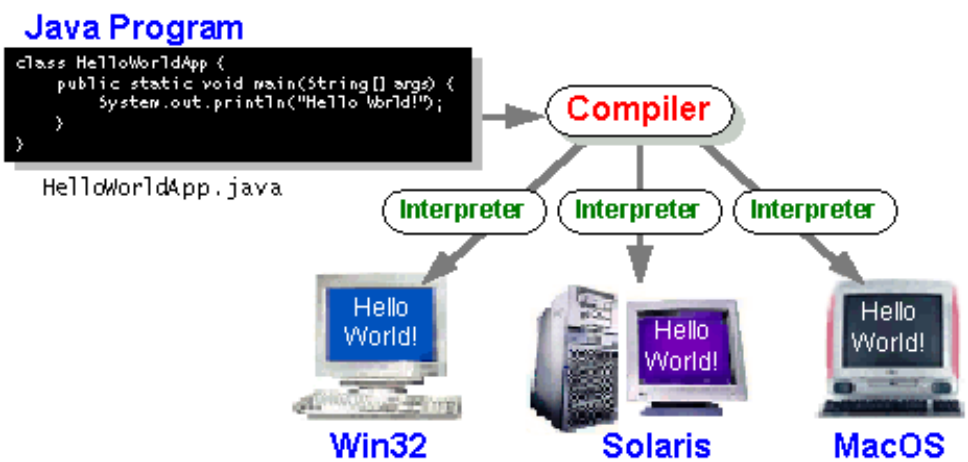


تعتبر JVM الجزء الوحيد من بيئة البرمجة الجافية الذي يعرف ما هو نظام التشغيل الذي تعمل عليه البرامج المختلفة. فالفئات كما ذكرت، تعرف إنها يجب أن تعمل لحساب الآلة التخيلية. و هي واحدة في كل مكان مهما اختلف نظام التشغيل و نوع الكومبيوتر. أما الآلة التخيلية نفسها، فهي العضو الذي يتصل بالكومبيوتر.. و يقوم بما يلزم من عرض على الشاشة، أو قراءة من الكيبورد، إذا لابد أن تعرف ال JVM عن نظام التشغيل الذي تعمل عليه.

3.3 مميزات Java

1. لغة سهلة التعلم و كبيرة الإمكانيات وبدون تعقيدات كا C++ .
2. لغة برمجية تعمل بواسطة الأهداف OOP حيث وفرت كثير من الجهد الذي كان يبذل باستخدام البرمجة التقليدية ، حيث كانت البرمجة التقليدية توفر للمبرمج مكتبة من الدوالي إضافة إلى تركيب تقليدي للبرنامج وعلى المبرمج أن يستعمل الدوالي مع تركيب البرنامج لإنشاء التطبيقات مما يضطره لكتابة السطور الكثيرة أكثر من مرة ؛

3. لقد كانت وحدة بناء البرنامج هي الدالة .. في حين أتت البرمجة بواسطة الأهداف بفكرة جديدة هي إنشاء عناصر متكاملة تحتوي على بيانات ودوالي هي أساس إنشاء البرنامج وبالتالي أصبحت وحدة بناء البرنامج وحدة كبيرة هي الفصيلة أو العنصر Objectمما سهل واختصر الكثير.
4. لغة أمنة من ناحية البرامج التي تنفذ في الحاسب فإنها لا تؤدي نظام التشغيل .
5. لها بيئة تشغيل خاصة بها JVM التي تقوم بترجمة البرنامج للغة الآلة وبالتالي فإن لغة الجافا غير مرتبطة بنظام التشغيل.
6. لها مكتبة فصائل Class Libraries توفر معظم أو كل الفصائل المطلوبة للإعمال مثل التعامل مع الملفات وقواعد البيانات والشبكات و الرسومات المجسمة والحركة وكذلك التعامل مع الانترنت.
7. تقوم على لغة C , C++ فعندما تم إنشاء لغة الجافا كان أساس بنائها لغة من أشهر وأقوى اللغات وهي C,C++ وبالتالي فهي لم تبدأ من حيث بدأ الآخرون بل من حيث انتهى الآخرون وهي لغة C++ و ثم إضافة الجديد في لغة الجافا وسميت CLink .
8. يمكن لأي برنامج معمول بلغة Java أن يعمل بشكل مباشر على أي framework بمعنى إن البرنامج يمكن أن يعمل على Windows Xp أو Linux أو Mac على عكس إمكانيات لغات البرمجة الأخرى مثل C++ آو حتى C# .
9. تأخذ تلقائياً البيئة التي تعمل ضمنها وتدعى هذه التقنية تقنية سوينغ أي انك عندما تقوم بتطوير جافا فان هذه التطبيقات عندما تعمل ضمن ويندوز فان عناصرها المختلفة تأخذ شكل ويندوز وعندما تعمل ضمن بيئة الماكنتوش فإنها تأخذ تلقائياً شكل واجهات الماكنتوش وهذه ناحية هامه جداً للمستخدم وللمبرمج أنظر شكل 1-3.



شكل 1- 3 يوضح تقنية سوينغ

ولكن ما هو سر هذه القوة ؟
لقد كان في فكر مخترع هذه اللغة هو اختراع لغة تستطيع أن تركز بها في وصف المشكلة التي تريد حلها بعيدا عن تفاصيل نظام التشغيل. هذه التفاصيل مثل:

كيفية كتابة ملف على القرص الصلب
كيف أكتب معلومات في ذاكرة الجهاز وكيف أعيد قراءتها
كيف أخلق المعلومات في صورة Object ومتى أقوم بحذفها

فمثلا إذا كنت أريد أن أكتب برنامج لإدارة المدارس وقلت لك أن تأخذ التفاصيل السابقة معك وأن تفكر في المشكلة فلن تصل لحل المشكلة مثل شخص آخر يضع كل تفكيره وتركيزه في وصف نظام إدارة المدرسة مثل من له حق استخدام النظام و ما هي المعلومات المطلوبة عن المدرسين و الطلبة و المناهج و ما هي السيناريوهات USE CASES المختلفة للنظام.
إذن فلغة Java هي تقريبا مثل أي لغة طبيعية كالعربية والانجليزية نستخدمها لنعبر عن أفكارنا ومشاكلنا ونتواصل بها مع الآخرين .

وقد أضيفت الكثير من المميزات للغة جافا في الإصدار الأخير لهذه الغه ومنها ارتفع عدد الحزم من 8 إلى 23 حزمه ومن هذه الحزم هيكلية حبيبات جافا والتي هي عبارة عن عناصر شكلت هذه الحزمة دعما قويا لجافا والتي أصبحت بفضلها لغة حقيقية للتطوير بالعناصر بحيث يمكن مزج وإعادة استخدام العناصر وتوزيع التطبيقات الناتجة وخلافه .

الأصناف الداخلية وهي إضافة تقنيه تمكن المبرمجين من تعريف عناصر جافا ضمن أصناف أخرى وهذا مهم لأن العناصر في برمجيات جافا يجب أن تكون هيكليه لتكون مرنه في معالجة المشكلة المراد حلها .
الخواص الدولية في جافا حث أصبح بإمكان عناصر جافا في هذا الإصدار تمييز موقع الجهاز الذي تعمل ضمنه والتصرف حسب ذلك حيث صار بإمكان برمجيات جافا تحديد موقعها وتحديد الوقت والتاريخ واعتماد المقاييس المستخدمة في ذلك الموقع وهذه إضافات هامه إذا أخذنا بعين الاعتبار إن لغة جافا صممت لوضع تطبيقات تنتقل عبر الانترنت ويجب على هذه التطبيقات أن تحتوي على مثال هذه الخاصية .

في عدة التطوير في جافا 1,1 تم تعزيز عناصر الإدخال والإخراج 1/0 بحيث تدعم أخراج وإدخال الحروف وهذه ميزه هامه حيث أن العديد من البرمجيات المصممة بغير اللغة الآنجليزيه تستخدم أساليب مختلفة لتشفير الحروف مثل الصينية واليابانية والعربية حيث تستخدم هذه لوحات محارف آسكي وغير تلك المستخدمة للغة الآنجليزيه وقد فتح هذا التطور الطريق لدعم اللغة العربية في جافا .

في جافا تم إضافة دعم التواصل مع قواعد البيانات بإضافة jdbc وتتكون job من مجموعه من العناصر ألازمه للتحكم بقواعد البيانات وتستخدم لغة البحث البنيوية sql وهي لغة خاصة للتحكم بالبيانات في قواعد البيانات .

وآخر تحديث في جافا هو أهمها كان تضمين تقنية تشغيل العمليات عن بعد وهي تقنيه يمكن بواسطتها للبرمجيات وخصوصا تلك المكتوبة بلغة جافا التواصل سويه حتى ولو كانت تعمل ضمن بيئات مختلفة وفي مواقع مختلفة وقد أدت هذه الإضافات إلى دعم جافا بشكل كبير كلغة لتطوير التطبيقات الموزعة حيث يمكن لتطبيق جافا يعمل ضمن نضام في الرياض مثلا أن يتعامل مع تطبيق جافا أخر يعمل على نضام في لندن دون أن يحتاج أي من هذين التطبيقين لمعرفة تفاصيل النظام الذي يعمل ضمنه التطبيق الأخر .

وبفضل هذه اللغة تولد لدينا لغتان برمجه تستعملان في برمجة صفحات الوب وقام بابتكار هذه اللغتان مبرمجون من شركة نافيجيتور والغتان هم java script and jscript .

3.4 الفرق بين لغة Java ولغة C++

جدول 1-3 بين الفرق بين Java و C++

Java	C++
تستخدم الأهداف بشكل أساسي في كتابة البرامج ولا يوجد برنامج خالي من الأهداف class	لا تستخدم الأهداف في معظم برامجها
أبعدت شيء أسمة المؤشرات بسبب تعقيدها وكثرة أخطائها إن لم تعامل صحيحاً .	تدعم المؤشرات
لا تدعم الوراثة المتعددة وقد أبدلتها بشيء اسمه الواجهات interface	تدعم الوراثة المتعددة
لا يوجد مدمرات Destructor function في الوراثة : ويتم تحرير الذاكرة بواسطة ما يسمة (garbage collector) الذي يعمل بالخلفية ويقوم باستدعاء المنهج (finalize) قبل التحرير، مع العلم أنه لا يوجد أي ضمان لهذا الاستدعاء ولضمان ذلك عليك القيام به بنفسك من خلال تعريف مناهجك الخاصة والتأكد من استدعائها.	يوجد في الوراثة

تعمل على أي Operating System	عكس السابق
البرامج المكتوبة بالـ java و مع الأسف محكوم عليها بالبطئ الشديد كحال أشهر البرامج في العالم برامج الإدارة في (Oracle) و (Builder) و (JDeveloper) في أحد المرات سئل James Gosling مخترع Java عن سبب بطي Java فأجاب : إن الديكتاتورية أسرع دائما من الديمقراطية فعلا إجابة نموذجية ، السرعة ليست كل شيء .	سريعة جدا بمعدل 10-20 مرة من Java و البرامج المكتوبة بها تكون سريعة جدا كذلك
لا تدعم البرمجة الجينية أي template ويمكنك استخدام الحاويات (containers) من الصنف Object.	اللغة الوحيدة التي دعمت هذه الخاصية
تدعم التجريد abstraction	لا تدعم التجريد abstraction
لا يوجد شيء أسمة تجاهل العمليات operator	تدعم هذه الخاصية
تدعم آلية التزامن synchronized	لا تدعم إلية التزامن
لا تسمح بتمرير المعطيات بواسطة المرجع reference	تدعم هذه الخاصية
لا تستطيع التعامل مع مواقع ذاكريه أو مع المنافذ فلا تستطيع كتابة كود ليشغل قطع .	تعتبر من اللغات المشهورة بالتعامل مع الهاردوير
تملك دعم خيالي للويب فهي لغة الويب والشبكات	تدعم بشكل بسيط عن نظيرها
يجب حفظ البرنامج باسم Class الرئيسي	ليس مهماً
يتم إدارة العمليات في الذاكرة تلقائيا فتكون أكثر أمانا .	يتم يدوياً .

3.5 شبكة الويب العالمية وما وراء لغة Java

3.5.1 تاريخ الانترنت

ARPANET (التابعة لإدارة مشاريع البحوث المتطورة) ولدت في عام 1969 وكانت هذه أول ملامح الشبكة العالمية استخدمت أربانت بروتوكول NCP (Network Control Protocol) ولد Internet Protocol و Transmission Control Protocol في السبعينات

في عام 1983 انشقت MILNET عن ARPANET

في عام 1983 انشأت مؤسسة العلوم الوطنية NSFNET

تلاشت ARPANET رسميا في عام 1989

3.5.2 وصف الانترنت

- الانترنت بنية رقمية عالمية توصل فيما بين ملايين من أجهزة الكمبيوتر وعشرات الملايين من البشر
- المعلومات على شبكة الانترنت متوفرة بأشكال عديدة نصوص، صور، مواد مسموعة، أفلام فيديو، ... الخ
- الآنترانت بنية محلية/عالمية توصل كيان ما (شركة أو مؤسسة) داخليا
  - محمية من المستخدمين من خارجها
  - معظم شبكات الآنترانت توضع خلف جدار الحماية(Firewall)
  - تتضمن معلومات ذات ملكية خاصة
  - تستخدم بعضها شبكة الانترنت للوصول إلى مناطق أوسع
- الإكسترانت :Extranet مزيج من الانترنت والآنترانت تستخدم فيما بين الشركات التجارية.

3.5.3 الشبكة ومكوناتها

- المضيف هو جهاز كمبيوتر مكلف بوظيفة العمل على الشبكة
- أي مجموعة من الأجهزة العاملة كمضيف متصلة بطريقة يستطيع اثنان منها تبادل الرسائل فيما بينهما تسمى شبكة
- البروتوكول طريقة يتفق بها جهازان على التخاطب
- العنوان أسماء الكمبيوترات التي يمكن التأشير إليها بمرجع

3.5.3.1 تعريف الشبكة العالمية WWW

- نظام لاسترجاع المعلومات واسع النطاق ومتشعب الوسائط يتيح الوصول إلى عالم واسع من الوثائق
- خدمة تتمتع بقاعدة شعبية واسعة
  - لإنها سهلت للغاية ايجاد المعلومات على الانترنت
  - وضعت طريقة موحدة للتوصل إلى المعلومات ومشاهدتها
  - لأن محتواها يزين بالرسوم والصور بازدياد
  - لأن متصفح الشبكة سهل التشغيل

3.5.3.2 المكونات الرئيسية للشبكة العالمية WWW

- نظام المستفيد/الخادم.
  - المتصفح يتخاطب مع خادم الشبكة.
- نظام المخاطبة العنوان الإلكتروني على الشبكة(URL).
  - لتحديد مكان الوثائق والملفات والبرامج ... الخ.
- بروتوكول الشبكة.
  - بروتوكول ارسال النص المتشعب(HTTP) .
  - يستخدم للانتقال بسرعة من صفحة إلى أخرى.
- لغة ترميز النصوص التشعبية(HTML) .
  - مجموعة من الأوامر التي تصف شكل الوثيقة.
  - لغة الترميز المستخدمة للشبكة.
  - يفك المتصفح رموزها ويظهرها.

3.5.3.3 تطبيقات الانترنت

- المتصفحات: لاستعراض صفحات الانترنت
- البريد الإلكتروني :Email لارسال واستقبال الرسائل الإلكترونية
- بروتكول ارسال الملفات :FTP لتنزيل وتحميل مختلف أنواع الملفات
- يوزنت :(Usenet (Newsgroups يتم من خلالها مشاطرة المعلومات ومتاحة لأي شخص
- :Telnet طريقة للاتصال كانت تستخدم قبل ظهور الشبكة العالمية
- Gopher : أول تطبيق للانترنت، نادرا ما يستخدم حاليا ولكنه كان له أثر كبير في تطور الانترنت.

فتعتبر الانترنت البنية التحتية للويب ، ومنذ إطلاق لغة Java أصبحت صفحات الويب جذابة وذلك لأنة يمكن تنفيذ برامج Java من خلال مستعرض الويب فتنتج واجهة رسومية حديثة مع جميع عناصرها للتفاعل مع مستخدمين الويب ومعالجة طلباتهم حيث أنشقت لغة سميت بلغة Java Script والجدول 2-3 بين الفرق Java و Java Script .

3.6 الفرق بين Java و Java Script

جدول 2-3 بين الفرق Java و Java Script

Java	Java Script
انتجتها شركة صن مايكروسيستمز	أنتجتها شركة نتسكيب
Object Oriented Language يعني تدعم مفهوم الكائنات و الوراثة	Object Based Language تدعم مفهوم الكائنات و لا تدعم مفهوم الوراثة
تستطيع الجافا أن تكتب على الملفات في السيرفر كما تستطيع استخدام خدمات السيرفر الأخرى	لاتستطيع الجافا سيكريبت أن تكتب على الملفات في السيرفر ولا تستطيع استخدام خدمات السيرفر الأخرى
برامج الجافا تخزّن كملف منفصل و يوضع لها أمر HTML في ال لقراءتها مثل أمر الصورة	أوامر الجافا سيكريبت تكون مضمّنة في الهتملHTML
لا يمكن الحصول على البرنامج لأنه يكون في برنامج خاص Class .	يمكن نقل البرنامج حتى و لو كان في ملف آخر و لا يمكن حفظ الحقوق بأي صورة
أول أربع حروف جافا	أول أربع حروف جافا
تكتب بها برامج منفصلة تعمل على الكومبيوتر بانفصال عن المتصفّح	لابد من وجود متصفّح لتعمل

3.7 مترجم للجافا

توجد أكثر من طريقة لكتابة برامج Java وترجمتها منها :

(1) استعمال المكتبة JDK مباشرة مع استعمال أي محرر سطور :

تعتبر هذه الطريقة التقليدية هي استعمال أدوات JDK التي أنتجتها شركة SUN مع أي محرر سطور لإعداد البرنامج وهي الطريقة المتبعة عند شرح أجزاء لغة جافا ونبدأ كما يلي :

الأدوات المطلوبة لاستعمال هذه الطريقة :

1. محرر سطور وليكن Notepad " المفكرة " الموجود مع ويندوز .
2. مجموعة JDK: ويمكنك الحصول على مكتبة JDK من موقع SUN .من هذا الرابط http://java.sun.com/j2se/downloads.html
3. أدوات المجموعة JDK:

- الملف Javac : وهو الملف التنفيذي المستعمل في ترجمة الملف المصدر إلى الصورة التنفيذية .
- الملف Java: هو البرنامج المسئول عن تنفيذ برامج Java التنفيذية بعد تحويلها .
- الملف Applet Viewer: لعرض برنامج Applet للإخبار .

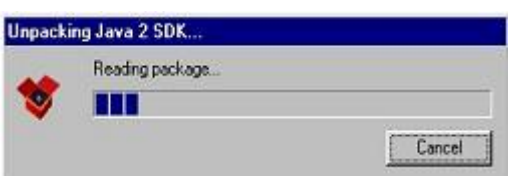
(2) استعمال برامج وسيطة مثل JCreator وأنا أفضلة واستخدمه و قد تم أرفاقه في القرص المرفق .
(3) استعمال البرامج المعدّة للغة Java مثل : Forte JBuilder : يمكنك إنزالها من هنا
http://www.borland.com/jbuilder/personal
http://www.jinfonet.com/download/Forte

3.8 تنصيب برنامج Java

لقد تم أرفاق في القرص المدمج محرر (JCreator™ 2.50) الذي يعمل على عدة أنظمة التشغيل ، وثم أرفاق مجلد (Java(TM) 2 SDK v2.0) الذي من خلاله يهيئ نظام التشغيل لأوامر الجافا.
فإذا كان الجهاز الذي تعمل عليه اقل من بينتيوم 4 فيتم تحميل البرنامج jdk1-2-2 فقط أما إذا كان بينتيوم 4 فيتم تحميل البرنامجين jdk1-2-2 ثم hotsport1-0-1

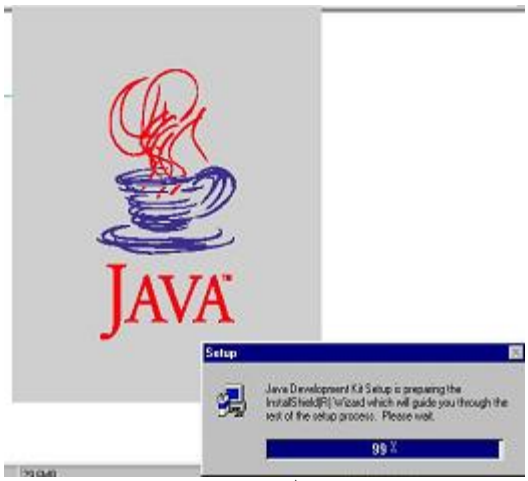
- **خطوات تحميل لغة Java**

1- عند الضغط على الملف التنفيذي ستظهر نافذة كما في الشكل 2-3.



شكل رقم 2-3

2- وتظهر أيضاً نافذة فيها شعار الشركة sun كما في الشكل 3-3.



شكل رقم 3-3

3- ستظهر نافذة فيها زرين أضغط next كما في الشكل 3-4

شكل رقم 3-4

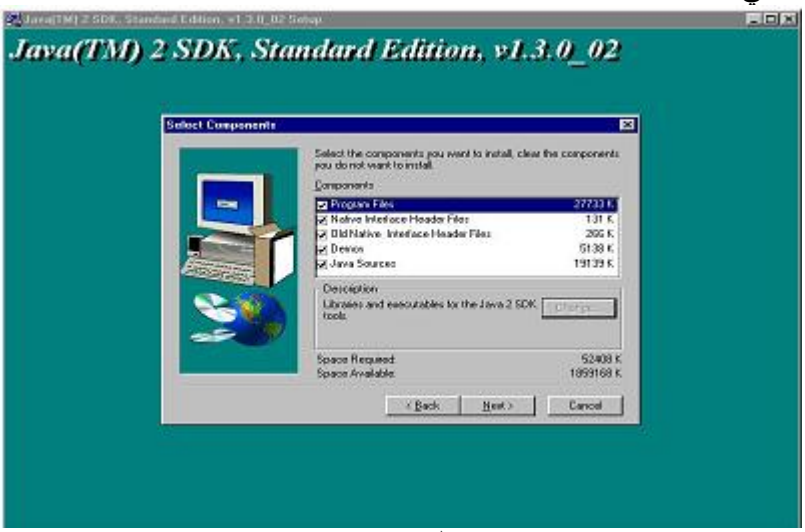
4- تظهر نافذة تخبرك باتفاقية الشركة أضغط زر yes كما في الشكل 3-5.

شكل رقم 3-5

5- تظهر نافذة تخبرك باسم المجلد الذي سيتم تحميل الملفات به ومكان هذا المجلد ، ويفضل عدم تغير المسار . أضغط زر next كما في الشكل 3-6.

شكل رقم 3-6

6- تظهر نافذة تخبرك بالملفات والمجموعات التي سيتم إنزالها على الجهاز ، أضغط next كما في الشكل 3-7.



شكل رقم 3-7

7- ستظهر نافذة توضح مدى تقدم عملية التحميل ، أنتظر حتى تصل إلى النهاية. كما في الشكل 3-8.

شكل رقم 3-8

8- أخيراً تظهر نافذة و بها زر finish أضغطة كما في الشكل 3-9.

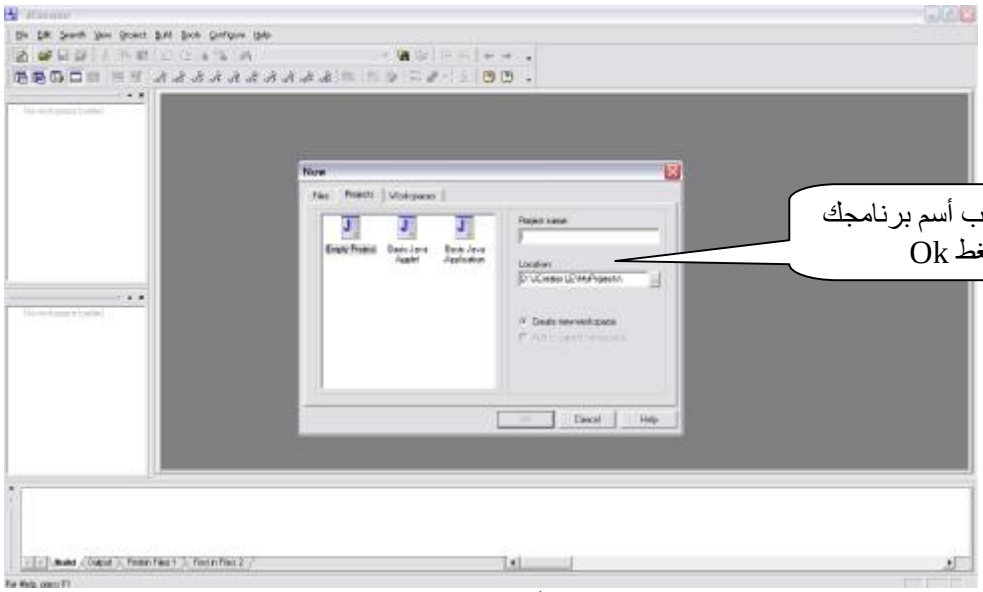
شكل رقم 3-9

3.9 إنشاء برنامج بسيط :-

Creating Sample Program

دعنا نبدأ بكتابة برنامج بسيط يقوم بعرض كلمتي (Hello World)

افتح محرر (JCreator™ 2.50) ومن قائمة File أختر new فيظهر مربع حوار كما في الشكل 3-10.

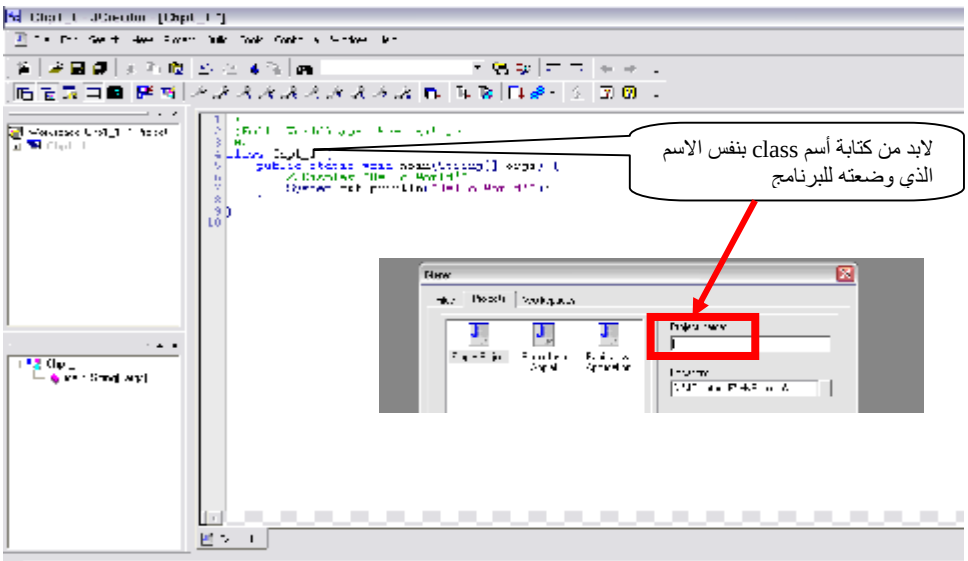


شكل رقم 3-10

أكتب هذا الكود

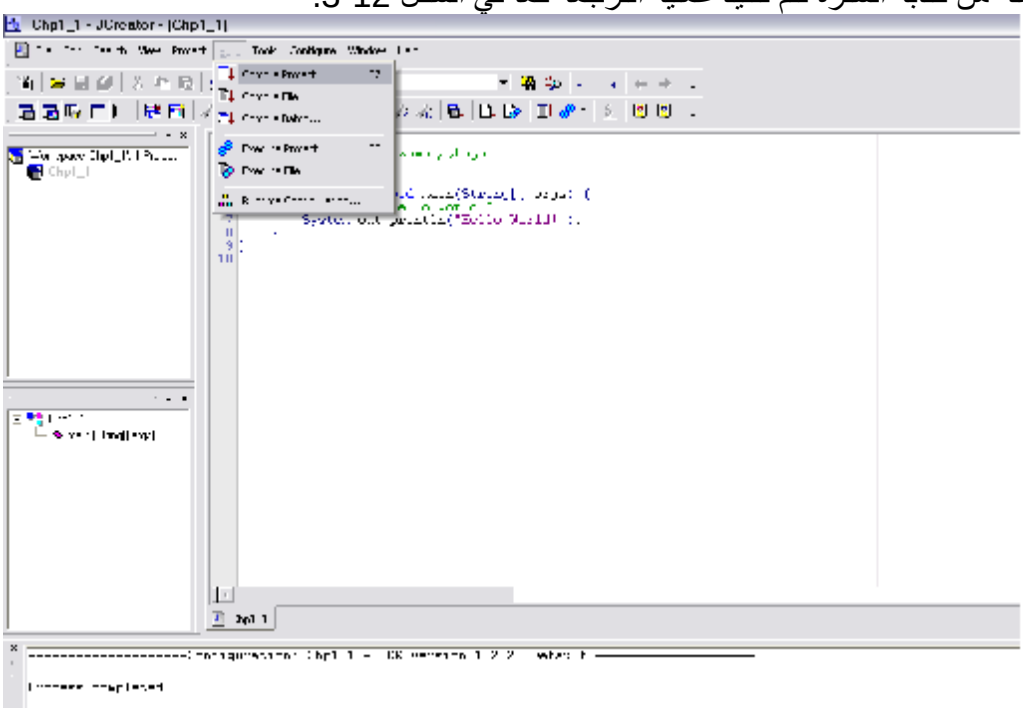
```
/*
 (Hello World) برنامج بسيط يعرض
 */
public class Chp3_1 {
    public static void main(String[] args) {
        //Display "Hello World!"
        System.out.println("Hello World!");
    }
}
```

كما في الشكل 3-11.



شكل رقم 3-11

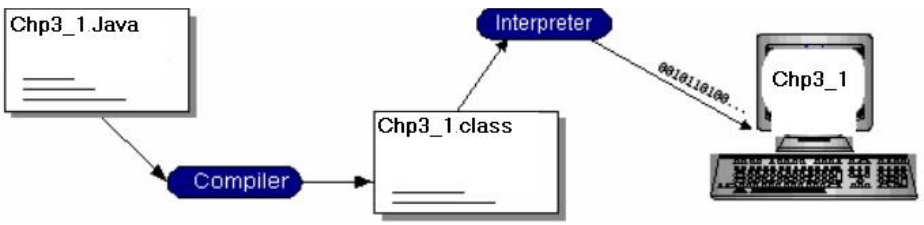
بعد من كتابة الشفرة تتم تنفيذ عملية الترجمة كما في الشكل 3-12.



شكل رقم 3-12

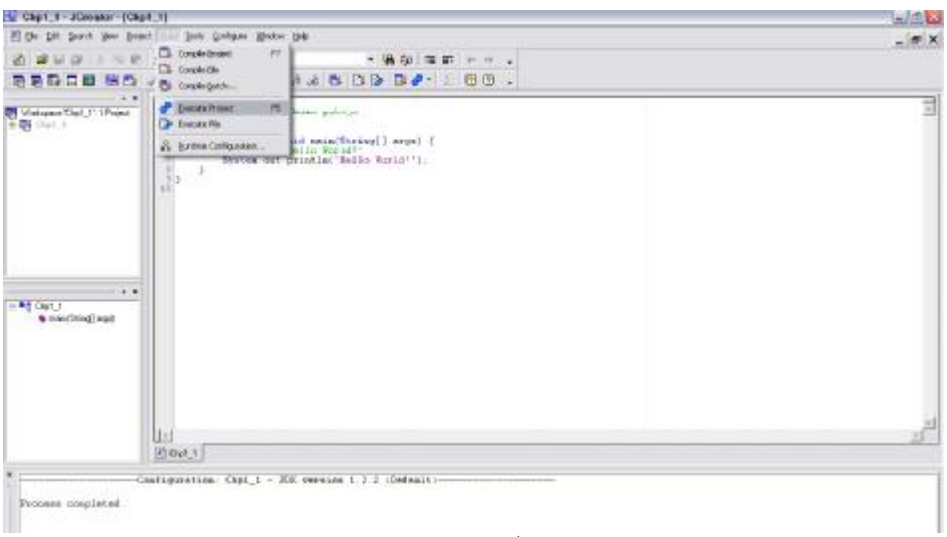
إذا كان الكود يوجد به خطاء فإن المحرر يعرض بالسطر الذي فيه الخطاء .

ستلاحظ أن المحرر صنع ملف جديد بامتداد class. كما في الشكل 3-13 .



شكل رقم 3-13

بقي أخر خطوة وهي إظهار الناتج على الشاشة كما في الشكل 3-14.



شكل رقم 3-14

فتظهر شاشة فيها خرج البرنامج كما في الشكل 3-15.

شكل رقم 15-3

شرح البرنامج :

في الأسطر (1 - 3) تعليق يضعه المبرمج ليشرح اسطر الشفرة .

السطر 4 يحتوي على تعريف الفئة ما تسمى ترويسة الصنف (class) . وكما قلنا أنه شرط في كل برامج جافا أن الملف الرئيسي يجب أن يطابق اسمه اسم الكائن (class) الرئيسي الموجود فيه.

و نحن هنا سنعرّف فئة جديدة اسمها .Chp3_1 و قد اعتاد مبرمجو لغة الجافا على اعتماد تسمية الفئات بطريقة معيّنة (Naming Convention)

كلمة Public تسمى مُغيّر دخول (Access Modifier) و هي تسبق تعريف الفئات classesوالأعضاء Objects والمتغيّرات Variables و الوظائف .Methods و سنبين ذلك بشكل أوسع في الفصول اللاحقة .

و مغيّر Public يعني أن هذه الفئة عامة و يمكن لأي فئة أخرى موجودة في الآلة التخيلية (virtual machine)أن تستخرج أعضاء منها، أو أن تقوم على تشغيلها .

و ملف الجافا قد يحتوي على تعريف لأكثر من فئة. و لكن فئة واحدة فقط يمكنها public وهي التي يتم تسمية الملف عليها .

فمثلاً، يمكنني أن أقول التالي :

```
public class FirstClass {
. .
. .
                    }

class SecondClass {
. .
```

. .

}

في المثال السابق يحتوي على تعريف فئتين. و نلاحظ أن أحداهما فقط هي التي تم تعريفها لتكون public وعلى هذه الأساس، سيكون اسم الملف على هذه الفئة، وفي مثالنا سيكون : Chp3_1.java.

نلاحظ أن السطر الرابع ينتهي برمز ({) و هو يعني أن تعريف الفئة (class) سيبدأ عند هذه النقطة. و بما أن لكل شيء بداية و نهاية، فإننا نتوقع أن يتوقف تعريف البرنامج عند العلامة المقابلة (}) و هذا يعني أن تعريف الفئات يكون دائماً محصوراً بين رمزي المجموعة .

السطر الخامس :

```
public static void main(String[] args) {
```

في هذا السطر يأتي لتعريف وظيفة (method) من الوظائف. سنتعرّف على الوظائف بشكل مفصّل في الفصول القادمة ان شاء الله. لكن ما يهمنا الآن هو الوظيفة الموجودة بين أيدينا .

هذه الوظيفة هي ذات نوعية خاصة، إنها ال main method و هي نقطة البداية لأي برنامج يتم تشغيله من مشغّل الدوس (Dos prompt) . فهي نقطة البداية التي تتحدّث عنها الآلة التخيلية للجافا، في حالة قمنا بتشغيل أي ملف جافا من محرك الدوس. و هي في الواقع فئة تتبع للفئة بشكل عام و لا تخص الأعضاء المنتمين لهذه الفئة .

فأي برنامج جافا، قد يتكون من عدة ملفات (بحيث يكون كل ملف محتوياً لتعريف فئة)، و يجب أن تحتوي فئة واحدة على نقطة البداية، التي تتمثّل في وظيفة main .

السطر السابع :

```
System.out.println("Hello World!");
```

إن هذا السطر في الواقع يقوم باستدعاء و ظيفة println التي تقوم بكتابة ما يمرر لها في الأقواس و في حالتنا هو ("Hello World ") .ما سبق هذه الوظيفة هو عنوان العضو و الفئة التي توجد فيها هذه الوظيفة، حتى تقوم الآلة التخيلية بتشغيلها بشكل صحيح . فالكلمة الأولى و هي System هي في الواقع اسم لفئة. و في هذه الفئة توجد صفة اسمها out و هي عضو في فئة PrintStream و يستطيع أعضاء هذه الفئة استخدام وظيفة الـ println ( ) .

لغة Java لغة حساسة للأحرف كالسي أي يجب الآنتباه لحالة الأحرف كهذه HelloWorld ≠ helloworld .

```
System.out.println("Input has " + count + " chars.")
```

عدم وضع فاصلة نهاية الفقرة خطاء شائعاً


لابد من الآنتباه إلى بعض الكلمات المحجوزة من ناحية شكل حروفها فبعض الكلمات تبدأ بحرف كبتل مثل (System) .

وفي السطور الأخيرة نقوم بغلق الأقواس، و نلاحظ أن تعريف الفئة قد تضمّن تعريف الوظيفة main و نلاحظ أننا نغلق القوس الداخلي أولاً ثم الخارجي و هكذا .

3.10 بنية البرامج بلغة Java

Java Program Structure

1. تعليقات (Comments).
2. كلمات محجوزة (Reserved Word) .
3. دوال – مناهج (Method) .
4. عبارات (Statements) .
5. كتل (Blocks) .
6. أصناف (Classes) .
7. معدلات الوصول (Modifiers) .
8. الدالة الرئيسية (main) .

فلكي تتمكن من فهم البرامج بشكل جيد يجب معرفة العناصر المذكورة سابقاً بشكل جيد ، فسنقوم بشرحها بالتفصيل .

3.10.1 التعليقات (Comments).

التعليقات هي عبارة عن جمل يكتبها المبرمج عند كتابة البرنامج لشرح نقطة معينة أو وصف البرنامج وهذه الجمل يتجاهلها برنامج المترجم.
فتستطيع كتابة التعليقات بأي لغة تريدها سواء الانجليزية أو العربية ، لأن التعليقات سوف يتجاهلها البرنامج، لذلك مثلا تستطيع كتابة أسمك .

- **أنواع التعليقات في الجافا:**

- التعليق بسطر واحد :

```
// make sure we don't go over total
if(current > 100)
 current = 100;
```

شكل 3-16 يبين شكل التعليق السطري

كما هو موضح في الشكل 3-16، فإن هذا النوع من التعليق يتم بوضع علامتي (//) قبل السطر المراد تعليقه. و هذا يعني أن يتجاهله المترجم تماماً. و لكنه بالطبع يقوم بتوضيح بعض الأمور للمبرمج. فأنت مثلاً و آنت تنظر للصورة تعرف أن الرقم 100 الموجود في الأسفل يمثل ال total الذي يتحدث عنه المبرمج. و هذا بالطبع يعني شيئاً ما للبرنامج .

```
g.fillArc(0, 0,      // start
capWidth, height,    // size
90, 180);            // angle
```

شكل 3-17 يبين شكل التعليق السطري

أما في هذا الشكل 3-17، فقد تم إضافة التعليق في نهاية بعض الأسطر. و هذا يسهّل الأمور. لأن المترجم سيقوم بتنفيذ البرنامج حتى يصل لعلامتي (//) عندها سيتجاهل بقية السطر، و سينتقل لتنفيذ السطر التالي . وسيقلل من زمن الترجمة .

- التعليق بعدة أسطر :

```
ARectangle orgClass = new ARectangle(100, 100, 102, 102);
ARectangle newClass = null;

boolean serialize = false;
boolean deserialize = false;

/*
   see if we are serializing or deserializing.
   The ability to deserialize or serialize allows
   us to see the bidirectional readability and writeability
 */

if (args.length == 1) {
```

شكل 3-18 يبين شكل التعليق

نلاحظ هنا أنه تم حجز عدد من الأسطر بين (*/) و (/*). و هذا يعني أن هذه الأسطر هي عبارة عن تعليق. و لن يتم تنفيذها في البرنامج، و لكنها وضعت للتوضيح .كما في الشكل 3-18.

إن رؤية المترجم لـ (*/) تجعله يتجاهل كل ما يقابله حتى يصل لعلامة (/*) و يقوم بتنفيذ ما يليها .

- التعليق بهدف إضافة معلومات لملفات المساعدة :

توفر الـ SDK أداة رائعة لإنشاء ملفات المساعدة و هي .javadoc و ان استدعاء هذا الأمر على أي ملف جافا، يقوم بإنشاء عدد من ملفات المساعدة بصيغة HTML و ما يهمنا الآن هو أن ملفات المساعدة هذه سيتم إنشاؤها بناءاً على المعلومات الموجودة و التي قمت أنت

بتوفيرها في برنامجك. لذا فقد فلابد من كتابة هذا النوع من التعليقات بصيغة محددة، حتى يتم تضمينها ضمن ملفات المساعدة .

```
/**
 * Prints out the usage
 */
static void usage() {
```

شكل 19-3

كما ترى في الشكل 19-3 فقد تم حصر التعليق بين (**/) و (/*) و هذا يعني أن هذه العبارة ستكون خاصة بالتعليق الذي سيظهر في ملفات المساعدة .
نلاحظ في هذا النوع من التعليقات انه يجب أن تسبق بعض المكونات المحددة في البرنامج، و إنها توضح بعض الإمكانات لها .

- **فوائد التعليقات**

من فوائد التعليقات أنه حينما تكتب برنامجا لك فقط دون أن يراه الغير ، فإنك ربما تقول لنفسك بأنك لن تحتاج إلى كتابة أي تعليق لأنك أنت صاحب البرنامج لذلك تستطيع فهمه دون أن تكون هنالك تعليقات تشرح البرنامج ، ولكني أسألك سؤال ، تخيل بأنك كتبت البرنامج ومرت عليه شهور ولم تراه ثم بعدها نظرت إلى برنامجك فهل سوف تتذكره مباشرة؟
ربما إذا كان برنامجا سهلا فلن تحتاج لأي شيء يذكرك به ، ولكن تخيل بأنه كان برنامجا معقدا ويحتوي على دوال كثيرة فإنك سوف تضيع وقتك في فهم البرنامج مرة أخرى.
لذلك فمن المهم جدا أن تقوم بإضافة تعليقات في البرنامج تشرح فيه الهدف من البرنامج وكيف يعمل ، وتضع تعليقات بجانب الأوامر المهمة ، وكذلك تضع تعليقات بجانب النقاط التي واجهتك صعوبة فيها ولم تستطع إكمال البرنامج.
لذلك من الضروري جدا كتابة التعليقات لنفسك ، وكذلك لغيرك ، فإنك ربما تتبادل هذا البرنامج مع غيرك ، فهل تتخيل نفسك تشرح في كل مرة البرنامج لكل من سوف ترسل له هذا البرنامج ، طبعا في هذا تعب لك ، لذلك وجدت التعليقات لكي تشرح فيها البرنامج ومراحل عمل البرنامج مرحلة مرحلة.

فالبرامج الناجحة هي البرامج التي تحتوي على تعليقات ! وسوف تكتشف هذا إذا أرسل لك يوما برنامجا بدون أي تعليق ، فانظر كيف سيضيع وقتك وأنت تحاول أن تفهم الغرض من البرنامج.

تستطيع إضافة التعليقات في أي مكان ترغب به في البرنامج ولكن هنالك نقطة مهمة يجب أن تنتبه لها . فأنت لا تستطيع إضافة تعليق في وسط الأوامر مثلا

```
Syste/*hi……..programmer*/m.out.print();
```

فمن الخطأ أن تضيف البرنامج في وسط الأوامر بل تستطيع إضافتها بعد أو قبل الجمل والأوامر الرئيسية.

- **من طرائف التعليقات:**

طلب أستاذ لغة Java من طلابه كتابة برامج معينة ، وطبعا منهم المجتهد ومنهم الكسول وكان هنالك صديقان يكتبان البرامج معا فهم يقسمان العمل بينهما ، وكل شخص منهم كتب برنامجان فأصبح مجموع البرامج هي أربعة وعند تسليم البرامج للأستاذ وضع لكل شخص منهم نصف الدرجة بالرغم من أن البرامج مكتملة وتعمل بطريقة صحيحة ، فاندهشوا من ذلك وسألوا الأستاذ لماذا نصف الدرجة، ولم يخطر في ذهن أي شخص منهم بأن الأستاذ قد كشف لعبتهم هذه في المشاركة في كتابة الواجب فكيف أكتشف الأستاذ طريقتهم؟
بكل بساطة حينما كتب كل شخص منهم البرنامجين ، كتب كل شخص منهم في أعلى البرنامج اسمه ، ظن منهما بأن كل واحد سوف ينتبه لذلك ويغير الاسم لأسمه ، ولكن للأسف لم ينتبها لذلك فكان سر اكتشاف الأستاذ لهم هو التعليقات!!!

خلاصة القول أن التعليقات في أي برنامج تساهم في توضيحه و تجعل قراءته أسهل. و إذا كنت تنوي أن تكون مبرمجاً محترفاً، فلا تنسى التعليقات أبداً. بل و لابد أن تجعل كودك أسهل و أسهل بجعل الكود يتكلم عن نفسه بأن تستخدم أسماء واضحة للمتغيرات و الفئات .
خاتمة القول، استخدم التعليق كلما بدا ذلك ضرورياً. و لا تنسى أن شيئاً يبدو واضحاً و بديهياً الآن قد لا يبدو كذلك بعد أربعة أو خمسة أسابيع.

3.10.2 الكلمات المحجوزة (Reserved Word) :

الكلمات المحجوزة أو الكلمات ألمفتاحيه هي كلمات ذات معنى محدد بالنسبة للمترجم ولا يمكن أستخدمها لأهداف أخرى في البرنامج . فعلى سبيل المثال يصادف المترجم كلمة class فأنة يعرف أن الكلمة التي تأتي بعدها هي أسم الصنف. الكلمات ألمفتاحيه الأخرى الموجودة في مثال Chp3_1 هي public وسيتم التطرق لها لاحقاً.

3.10.3 المناهج – الدوال (Method) :

الدوال أو الروتينان الفرعية وهي أجزاء من البرنامج مثل الدالة الرئيسية تقوم بعمل مهمة معينة تتكرر في برنامجك أو تستخدمها في برامج أخرى أو حتى إذا كانت لا تتكرر من فوائدها

- تقسيم البرنامج إلى أجزاء صغيرة تستدعى وقت اللزوم .
- عدم ازدحام الدالة الرئيسية للبرنامج بأوامر كثيرة .

- تقسيم البرنامج إلى أجزاء يمكن اختبارها منفصلة لسرعة تحديد الخلل بالبرنامج .
- توفير المجهود والوقت والتفكير بعمل مكتبة خاصة بك تعيد استخدام الدوال التي استخدمتها في برنامج وذلك في برنامج آخر عند اللزوم .
- تقسيم العمل بين المبرمجين عن العمل في مشروع جماعي لإنتاج برنامج كبير .
- تبادل الخبرات بين مطوري البرامج بنشر أجزاء يستخدمها الآخرون في برامجهم .

فماذا نعني بالعبارة (System.out.print) ؟ إن (System.out) معرفة على إنها كائن خرج قياسي و print هو منهج في هذا الكائن مؤلف من مجموعة من العبارات هدفها إنجاز سلسله من العمليات لأظهر رسالة في الشاشة .

3.10.4 عبارات (Statements) :

تمثل العبارات عملاً أو سلسلة من الأعمال . فمثلاً العبارة
(System.out.println("Hello World!");) في المثال Chp1_1 هي رسالة ترحيب لعرض "Hello World!".

3.10.5 كتل (Blocks) :

تشكل الأقواس المتعرجة في البرنامج الكتلة التي تجمع عناصر البرنامج. تبدأ كل كتلة في Java بقوس متعرج مفتوح ({) وتنتهي (}) وكل صنف أو داله ينطبق نفس الآلية . والشكل 3-20 يبن ذلك.

شكل 3-20 يبين شكل الكتل

3.10.6 أصناف (Classes):

الأصناف تمثل البنية الأساسية في لغة Java حيث تميزها وتعطيها أدوات قوية جدا ، وسيتم التطرق لها في الفصول القادمة بشكل واسع جداً .

3.10.7 معدلات الوصول (Modifiers) :

تستخدم Java بعض الكلمات المفتاحية لتحديد خصائص البيانات والدوال والفئات وكيفية استخدام ها ففي المثال chp3_1 تم استخدام الكلمة public ، static وسيتم التطرق في الفصول القادمة إن شاء الله .

3.10.8 الدالة الرئيسية main :

وتعتبر أهم جزئ في البرنامج حيث لا يوجد برنامج يخلي من الدالة main فتمنح هذه الدالة القدرة على التحكم بتدفق البرنامج .

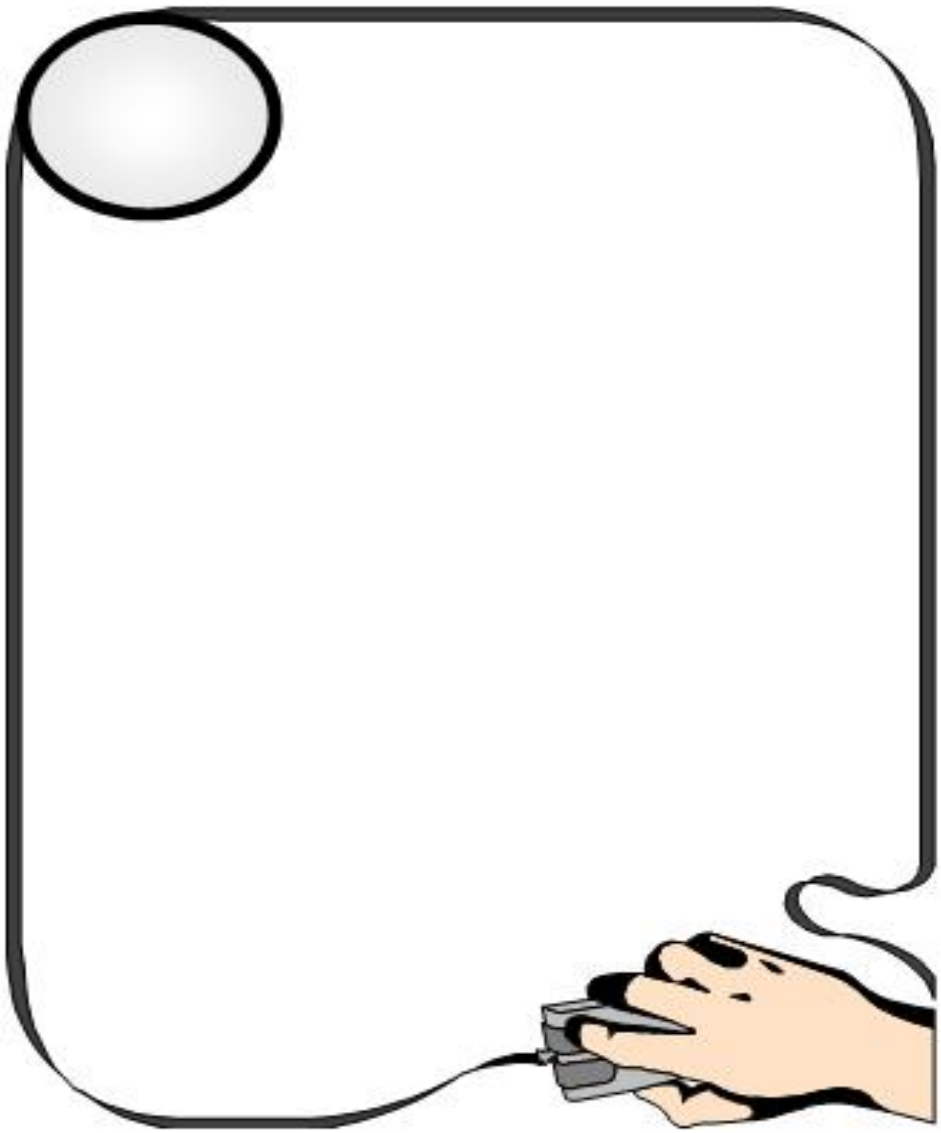
يجب أن يحتوي برنامج Java على المنهج main ، حيث يمثل هذا المنهج نقطة الدخول التي يبدأ منها البرنامج عند تنفيذه .

تمارين الفصل:

1. أكتب تعليمة Java واحدة للقيام بالأعمال التالية:
   1. أطبع الرسالة "Enter tow number".
   2. أطبع تربيع عدداً ما.
   3. أطبع تكعيب عدد ما.
2. أكتب برنامج يقوم بطباعة الأعداد من 1 إلى 10 على أن تكون الطباعة على نفس السطر وكل رقمين يفصل بينهما فراغ.
3. وضح الفرق بين Java,C++,Java Script؟
4. ما هو السبب الذي جعل لغة Java تتفوق على سائر اللغات؟
5. ماذا تطبع التعليمية التالية :

(System.out.print("*\n**\n***\n****\n*****\n")).

6. أكتب برنامج يطبع مجموع الأعداد من 1 إلى 10.
7. إذا كان لدينا المعادلة y=ax3+7 فأي من العبارات التالية تحقق هذه المعادلة:
   1. y=a*x*x*x+7 .
   2. y=(a*x)*X*X+7.
   3. ولا واحدة مما سبق.
8. أرسم الأشكال التالية:



4.1 مقدمة NTRODUCTION

نحن نحتاج في برامجنا في تخزين عناصر متنوعة من المعلومات، ومعظم هذه العناصر تتغير قيمتها أثناء تنفيذ البرنامج . لذلك يجب على البرنامج أن يزودنا بطريقة لحفظ وتخزين هذه القيم تدعى هذه الطريقة باستخدام المتحولات أو المتغيرات.

تتضمن لغة Java أنواع متنوعة وغنية جعلت منها لغة سهلة في تمثيل أنواع المعطيات المختلفة وبأفضل طريقة ، بالإضافة إلى إنها تمكن من إنشاء أنواع المعطيات التي نريد وحسب حاجة البرنامج .

وفي هذا الفصل سوف نشرح أنواع المعطيات البسيطة والتي مكنت لغة Java المبرمجين من استخدامها.

4.2 المتغيرات - المتحولات Variables

هي أنواع من المعلومات التي يمكن تخزينها في موقع خاص في البرنامج ونستطيع تغييرها أثناء عمل البرنامج ، فهي الوسيلة التي يتذكر بها الحاسب القيم خلال تشغيل البرامج .

فيمكن تخيل المتحولات وأماكن تخزينها كصناديق البريد. إذ يمثل اسم المتغير عنوان صندوق البريد ، وتمثل قيمة المتحول الرسالة أو محتويات صندوق البريد.

4.2.1 أسماء المتغيرات Naming Variables

أسماء المتغيرات في Java يمكن أن تكون طويلة جداً ، و لكن يجب أن تنتبه إلى أن بعض المحولات تفرض حداً معيناً لطول أسم المتغير .

عندما تريد أن تعطي أسما لمتغير ما يجب أن تراعي الأمور التالية

- الإشارات المسموح بها هي A - Z، a - z، 0 - 9 و إشارة الخط السفلي " _ " .
- الأحرف الصغيرة و الكبيرة لها دور مهم ، فمثلاً الأسماء amount، Amount ، AMOUNT هي أسماء لثلاثة متغيرات مختلفة .
- البدء برقم غير مسموح به ، مثال : 7days .

- لا يجوز بدء أسم متغير بخط سفلي " _ " .

- **أمثله على بعض أسماء متغيرات صحيحة**

amount
xyz
switch_8
A_very_long_Name
CustomersAmount

- **القيود المفروضة على الأسماء** **Restrictions on Name**

توجد بعض الكلمات التي لا يمكن استخدام ها كمميزات لغةJava , حيث تدعى هذه الكلمات بالكلمات المحجوزة (reserved keyword). فعلى سبيل المثال (int ، switch ، إلخ) أنظر الملحق B . ولكن تستطيع أن تستعملها كجزء من أسم متغير (مثلاً first_int_variable) .


لغة Java حساسة لحالة الأحرف و بالتالي فأن X,x هما مختلفان .

4.2.2 التصريح عن المتحولات Declaring a Variable

تعتبر لغة Java لغة شديدة الارتباط بأنواع المعطيات ، هذا يعني أن المترجم يجب أن يعرف نوع المعطيات الواجب تخزينها كمتحولات قبل يستخدمها قبل البرنامج، وبالإضافة إلى ذلك يضع المترجم قيوداً على العمليات المسموحه على أنواع المعطيات المختلفة . فمن أجل أعطاء كل متحول نوع المعطيات يجب التصريح (Declare) عم هذا المتحول أولاً ، فعندما نصرح عن متحول ، فأننا نعطي لهذا المتحول اسماً ونحدد له نوعاً، وبنفس الطريقة يمكن التصريح عن المتحولات .

" نوع المعطيات " < أسم المتحول >, < اسم المتحول > ;

4.2.3 أنواع المتحولات Type Variables

1. الأعداد الحقيقية Point Numbers Floating

كما يدل اسمها تختص الأعداد الحقيقية بـالأعداد التي تتألف من جزئيين ، الأول صحيح والثاني يأتي بعد فاصله . (مثال ...3,14159262 PI).

يستخدم الكمبيوتر طريقه أخرى للتخزين الداخلي لهذه الأعداد تختلف عن طريقته في تخزين الأعداد الصحيحة ، فعند تخزين الأعداد الصحيحة يقوم الكمبيوتر بتخزين هذه الأعداد رقماً برقم ، و بما أن الأعداد الحقيقية يمكن أن تكون طويلة جداً فإن اعتماد الكمبيوتر طريقة التخزين السابقة يمكن أن تكلفه الكثير من الذاكرة (Memory) . لذا يعتمد الكمبيوتر طريقه أخرى لتخزين الأعداد الحقيقية ، فهو يقوم بتقسيم العدد الحقيقي إلى جزئيين الأول يحوي الأرقام المكونة لهذا العدد دون فاصله ، و الثاني هو القوه أو الأس التي نستعملها مع 10 (مثال 10^3) لنحصل على العدد الحقيقي .

مثال :

3,14159 <== 0.314159 * 10^1 = 3,14159

و لكن من عيوب هذه الطريقة أننا قد نخسر بعضاً من دقة العدد الحقيقي و ذلك لمحدودية الخانات التي يخزن فيها الكمبيوتر العدد .

مثال :

نفرض أن الكمبيوتر يخزن الأعداد الحقيقية في 5 خانات بعد الفاصلة عندما نعطي الكمبيوتر العدد 3,14159 ، فإن الكمبيوتر يقوم بتخزين هذا العدد على الشكل التالي :

0,31415 ثم يقوم بضربه ب 10^1 فنحصل على العدد 3,14150 ، (لاحظ أن الرقم 9 قد اختفت).

و لهذا يجب أن نحدد مسبقاً الدقة التي نحتاجها في الحالات المختلفة ، و ذلك باستعمال أنواع المتغيرات التالية التي تقدمها لغة Java لنا :

جدول 1-4

Floating Point Primitive Data Types		
Type	Size	Range
float	32 bits	-3.4E+38 to +3.4E+38

double	64 bits	-1.7E+308 to 1.7E+308

العمود (الدقة) يخبرنا كم خانه على الأقل (قبل و بعد الفاصلة) ، يمكن أن يتم إظهارها بدقه . و لهذا فليس فقط كبر العدد الحقيقي هو الذي يلعب دوراً مهماً في اختيارنا لنوع المتغير ، بل أيضاً الدقة التي نحتاجها .

البرنامج التالي يوضح مرة أخرى الفرق في الدقة بين float و double :

```
1. /*
2. برنامج بسيط يجمع عددين من نمط الأعداد الحقيقية
3.  */
4. public class Chp4_1 {
5.     public static void main(String[] args) {
6.
7.     float    a;
8.     double   b;
9.
10.      a=10.12345678901234567890;
11.      b=10.12345678901234567890;
12.
13.      System.out.println("a ="+a);
14.      System.out.println("b = "+b);
15.
16.      System.out.println("a + b = "+(a+b));
17.
18.    }
19. }
```

فيكون خرج البرنامج

```
a =10.123457
b = 10.123456789012346
a + b = 20.2469137439684
```

الأسطر 10,11 : في هذه الأسطر نعطي متغير float a و متغير double b القيمة 10.12345678901234567890

عدد الخانات في كلا الحالتين كبير جداً بحيث يجب أن يتم اقتطاع عدد من الخانات .


عند تعريف متغير من نوع float يجب وضع الحرف f في نهاية العدد .


تعريف متغير من نوع float وعدم وضع الحرف f يسبب ظهور خطاء في زمن التنفيذ (Explicit cast needed to convert double to float).

2. متغيرات العدد الصحيح The Integer Variables

نستخدم متغيرات العدد الصحيح لتخزين أعداد صحيحة أي أعداد بدون فاصله ،

مثال :
1234 أو 0 أو 100.
و بما أن العدد الصحيح يمكن أن يأخذ قيماً كبيره كعدد سكان بلد ما أو قيماً صغيره كعمر إنسان ما ، تقدم لغة Java أنواعاً مختلفة من متغيرات العدد الصحيح .
الفرق بين هذه الأنواع يكمن في كبر أو صغر العدد الذي نريد تخزينه فيها .

جدول 2-4

Integer Primitive Data Types		
Type	Size	Range
byte	8 bits	-128 to +127
short	16 bits	-32,768 to +32,767
Int	32 bits	(about)-2 billion to +2 billion
long	64 bits	(about)-10E18 to +10E18

مثال تطبيقي على إشهار وإعطاء قيم للمتغيرات في الجافا:

البرنامج التالي مكتوب بلغة Java. و قد قمنا فيه بإنشاء بعض المتغيرات و من ثم طباعتها.

```
1. /*
2. مثال تطبيقي على اشهار واعطاء قيم لأنواع المتغيرات في الجافا
3.  */
4. public class Chp4_2 {
5.     public static void main(String[] args) {
6.         //الأدعداد الصحيحة
7.     byte     b;
8.     short          sh=200;
9.     int number=12;
10.      long lg=3094040;
11.     //الأدعداد الحقيقية
12.     float    balance=14.4f;
13.     double   d=10.3;
14.     //الأدعداد الحرفية
15.     char ch='a';
16.     String name="Java";
17.     //الأدعداد المنطقية
18.     boolean falg=true;
19.     //طباعة القيم
20.     // System.out.println(b);//مشكلة
21.      System.out.println(sh);
22.     //System.out.println(number);//مشكلة
23.      System.out.println(lg);
24.      System.out.println(balance);
25.      System.out.println(d);
26.
27.   }
28. }
```

في السطر 8 فقمنا بإنشاء متغير من نوع short و أعطيناه قيمة مبدئية هي 200 .

بينما في السطر 7 قمنا بإنشاء متغير من نوع byte و لم نعطه إي قيمة مبدئية .

عند تشغيل البرنامج السابق ستواجه المشكلة التالية :

```
C:\myJava\DeclareVars.java:24: variable b might not have been initialized
       System.out.println(b);           // مشكلة!
                          ^
C:\myJava\DeclareVars.java:26: variable number might not have been initialized
       System.out.println(number);      // مشكلة!
                          ^
2 errors
```

إن هذه المشكلة ظهرت تحديداً عندما حاولنا طباعة المتغيرين b و .number لاحظ إنها لم تظهر عندما قمنا بإشهارهما. و لكنها ظهرت عند محاولتنا لاستخدام هما.

هل تستطيع أن تعطي سبباً لذلك؟

في الواقع أن السبب هو أننا لم نعطهما قيمة مبدئية! ثم حاولنا طباعتهما. فحاولنا طباعة قيمة لا نعرفها! لذا سيتوقف مترجم Java و يعطيك الخطأ السابق .

3. المتغيرات ذات الأساس الرمزي Character-Based Type

تسمح لنا لغة Java بتمثيل المعطيات الغير رقمية، إذ يوجد نوع معطيات مسبق التعريف لتمثيل الرموز ، ويوجد أخر مسبق التعريف لتمثيل السلاسل.

- **Char :**

وهي بداية كلمة character و يخزّن في حجم من الذاكرة مقداره 2 بايت. و هذا المتغيّر يخزّن حرفاً واحداً فقط. و يكتب هكذا محصوراً بين علا متي تنصيص مفردتين.

فإذا أردت إنشاء متغيرا تضع فيه حرفا أكتب الجملة التالية في مصدر البرنامج :

```
char key = 'u';
```

ونلاحظ أنه حين نستعمل متغيرا لاحتواء قيم الحرف يجب وضع علامات اقتباس مفردة على جانبي قيمة الحرف المراد تعيينه كقيمة للمتغير.

- **متغيرات السلاسل String**

يستعمل هذا النوع لتخزين سلاسل الحروف ، حيث يمكننا التصريح عن سلاسل رمزية بدون تحديد طول هذه السلسلة، والتصريح التالي يشرح لنا كيفية ذلك.

```
String studentName = "ema";
```

ونلاحظ هنا أنه في حالة النوع الثاني تحاط سلسلة الحروف بعلامات اقتباس مزدوجة ويوجد تفرد في هذا النوع من المتغيرات حيث ينبغي كتابة الحرف ألأول كبيرا وذلك على غير العادة في باقي المتغيرات .

فالفرق بين String وبين char ؟

char كما عرفنا سابقا هو حرف أبجدي واحد أو رقم عشري واحد أو علامة ترقيم أو أي رمز من الرموز الأخرى المعروفة وفي لغة Java يعتبر الحرف من المعلومات التي يمكننا تخزينها في متغير char .

ثوابت الأحرف:
كما نعلم نضع ثوابت الأحرف بين إشارتي (' ') مثال 'A' . لا تخلط بين 'A' و "A" ! في الحالة الأولى نقصد ثابت حرف واحد بينما في الحالة الثانية نقصد متسلسلة .

إلى جانب ثوابت الحرف الواحد هناك أيضاً ثوابت الحرفين ('ab' ، 'XY' إلخ) ! هذه الثوابت الخاصة تعبر عن قيم صحيحة . و لكن من المفضل أن لا نستعملها ، لكي نتجنب مشاكل تتعلق بصلاحية تنفيذ البرنامج على نظم أخرى.من المفضل هنا استعمال طريقة الكتابة السادسة عشر (hexadecimal) .

- **متعاقبات الهروب Escape-Sequences**

متعاقبات الهروب هي نوع خاص من ثوابت الأحرف . نستعملها للتعبير عن الإشارات الغير قابله للإظهار . تبدأ هذه الإشارات دائماً بخط مائل \ (Backslash)
يليه حرف أو قيمه ثمانية (octal) أو سداسية عشر (hexadecimal). و لكنها تخزن (بفتح الزاي) داخلياً على شكل حرف واحد من نوع char .

جدول 3-4	
" \n "	سطر جديد (new line) .
" \r "	عوده إلى بداية السطر(Carriage Return) .
" \t "	مسافة بمقدار 8 خانات فراغ أفقياً (Tabulator) .
" \b "	رجوع بمقدار خانه واحده (Backspace) .
" \f "	صفحه جديدة (Formfeed) .
" \\ "	خط مائل (Backslash) .
" \' "	نقطه مبتدئه (Apostroph) .
" \" "	إشارة حصر .

4. المتغيرات المنطقية Logical Variables

وتضم نوع واحد فقط:

Boolean : سمي هذا المتغير نسبة للعالم الرياضي الانجليزي جورج بوول الذي أخترع الجبر البوليني الذي يعتبر الأساس لعلوم البرمجة وخاصة لغة الماكينة والاليكترونيات الرقمية والبوابات المنطقية وعلم المنطق.

و هذا النوع يخزّن نوعاً مميزاً من القيم .انه يخزّن قيمة من اثنتين فقط هما true, false :

ويستخدم في المقارنات المنطقية التي سيأتي تفصيلها فيما بعد إن شاء الله .

5. المتغيرات التي يعرفها المبرمج Aggregate Data Types

و هي المتغيرات التي يقوم المبرمج بتعريف نوعها. مثل ال classes ا لتي نقوم بكتابتها في ملفات و برامج الجافا.


المتغير هو مكان لتخزين القيم التي يحتاج المبرمج لحفظها بشكل مؤقت. و قد تتغير من آن إلى آخر. و يتم تحديد نوع القيمة التي يخزنها أي متغير .


أنواع المتغيرات لها حجم محدد للتخزين في الذاكرة .


يستطيع المبرمج أن يعرف أنواعاً لا عدد لها من المتغيرات باستخدام ال classes.

4.3 الثوابت Constants

بالإضافة إلى المتغيرات التي تعرفنا عليها حتى الآن ، كثيراً ما نحتاج أيضاً إلى استعمال الثوابت . فالقيمة 512 مثلاً في المثال x = y+ 512 هي ثابت دائم لا نستطيع تغيره .

4.3.1 ثوابت الأعداد الصحيحة Constants Integers Numbers

يمكن لثوابت الأعداد الصحيحة أن تتخذ الصيغ التالية مثلاً:

6815 (عشري)

0x1A9F (سداسي عشر)

عندما نبدأ العدد ب 0x أو (0X) فإن المقصود هو صيغة العدد السادس ة عشر (hexadecimal) ، وعندما نبدأ العدد (0) فإن المقصود هو صيغة العدد الثماني (Octal) .

final datatype Constantname =value;

فالكلمة final هي كلمة مفتاحيه في لغة Java وتعني أن الثابت لا يمكن أن يتغير.

وهذا الكود يبين ذلك

```
1. //برنامج يستخدم متغير من نوع ثابت
2. class Chp4_3
3. {
4.   public static void main ( String[] arg )
5.   {
6.     final double DURABLE = 0.045;
7.     final double NONDURABLE = 0.038;
8.
9.     System.out.println(DURABLE + 10);
10.
11.     NONDURABLE++;//سينتج خطاء بسبب محاولة تغير قيمتة وهو من نوع ثابت
12.
13.     System.out.println(NONDURABLE);
14.   }
15. }
```

4.3.2 ثوابت الأعداد الحقيقية Constants Real Numbers

يتم التعبير في Java عن هذه الثوابت بطرق مختلفة .

جدول 4-4

القيمة	طريقة التعبير
8,0	8.
-0,4	-.4
-2,4	-24.0e-1

-04.E+2	-400,0
18E-2	0,18
1.434E1	14,34

الثوابت التي يتم إعطاءها بهذه الطريقة تكون بشكل أوتوماتيكي من النوع double . كالأعداد الصحيحة نستطيع أن نحدد نوع الأعداد الحقيقية أيضاً و ذلك بكتابة حرف ملحق بعد العدد مباشرة. f مثلاً تعني float.


يجب التصريح عن الثابت وتهيئته قبل التمكن من استخدام ه .


اصطلاحياً تتم تسمية الثوابت بأحرف كبيرة.


من فوائد الثوابت

- التخلص من تكرار كتابه نفس القيمة.
- يمكن تغير القيمة في مكان واحد (عند الضرورة).
- يصبح البرنامج سهل القراءة.


إذا كتبنا رموزاً غير رقمية خطاء بينما كان البرنامج يتطلب منا قيماً حقيقية ، فإن البرنامج سوف يتوقف وينتج لدينا خطأ في التنفيذ .


إعطاء قيمة لمتحول قد تتجاوز الحدود وهذا ما يعرف (overflow) . فإذا حصل لدينا فيض زائد سينتج رسالة خطاء (Integer literal out of range).

4.4 أنواع المتغيرات من ناحية الوصول

- متغيرات من نوع static / هذه المتغيرات نستطيع أن نقول إنها نوع ثابت أي ليس بمعنى أننا لا نستطيع تغير قيمتها ولكن قد تكون بداخل Class فنتعامل معها مباشرة أي كأنها عامة فعند ترجمة البرنامج يتعرف عليه المترجم مباشرة قبل الدخول إلى الدالة الرئيسية .

- متغيرات محلية local / وهي المتغيرات المعرفة على مستوى Block أي المقطع ولا نستطيع الوصول إليها من خارج Block أي {} فعندما نخرج من Block فإنها تدمر من الذاكرة وتنتهي حياتها .

- معاملات / وهي التي تم تعريفها في رأس الدالة فتبدءا حياتها عند تنشيط الدالة وتنتهي حياتها عند انتهاء الدالة .

4.5 مجال تغطية المتغيرات The Scope and Lifetime of Variables

وهو الجزء من البرنامج الذي نستطيع من خلاله الوصول إلى المتغير . فتسمى المتغيرات المعرفة داخل المنهج بالمتغيرات المحلية .

4.6 المؤثرات Operators

المؤثرات هي الرموز التي تربط بين المتغيرات والثوابت لإنشاء علامة أو معادلة تختلف أنواع المؤثرات باختلاف وظيفة كل مؤثر .

وكما قلنا سابقا تطلب البرمجة عموما قدرات رياضية معينة وفكر رياضي في المبرمج وعلى الرغم من قيام الحاسب بكل العمليات الرياضية الا أنه يريد تعليمات وأوامر من المبرمج حتى يقوم بهكذا عمليات وتسمى الأوامر التي تعطيها للحاسب والتي تحتاج إلى عمليات رياضية تعابير ويمكننا استخدام هذه التعابير للقيام بعدة مهام منها تغيير قيمة متغير واستعمال المعادلات في البرنامج وتسجيل عدد مرات حدوث عمل ما في البرنامج وتستعمل هذه التعابير الجمع والطرح والضرب والقسمة وباقي القسمة

4.6.1 المؤثرات الحسابية Arithmetic operators

يمكننا استخدام المعاملات الحسابية العادية على القيم، مثل الجمع والطرح وغيرة كما في الجدول 5-4. وتستخدم مع المتغيرات والثوابت الرقمية .

جدول 5-4

Java operation	Arithmetic operator	Algebraic expression	Java expression

جدول 4-5			
Java operation	**Arithmetic operator**	**Algebraic expression**	**Java expression**
Addition	+	f+7	f + 7
Subtraction	-	pc	p - c
Multiplication	*	bm	b * m
Division	/	$x / y \ or \ \frac{x}{y} \ or \ x \div y$	x / y
Remainder	%	r mod s	r % s

4.6.2 مؤثرات المقارنة Relational operators

يمكننا استخدام مؤثر المقارنة مع المتغيرات والقيم لمقارنة قيمتين، والجدول 4-6 يبين عمليات المقارنة.

جدول 4-6			
المؤثر	الرمز	مثال	النتيجة
أكبر من greater than	>	10 > 8	1
أصفر من less than	<	10 < 8	0
يساوى equal to	==	10==8	0
لا يساوى not equal to	! =	10!=8	1
أقل من أو يساوى less than or equal to	<=	10<=8	0
أكبر من أو يساوى greater than or equal to	>=	10 > = 8	0

4.6.3 المؤثرات المنطقية Logical operator

يمكننا استخدام مؤثرات المقارنة مع نوع القيم المنطقية مما تعطينا نتائج True, False . ويستعرض الجدول 4-7 جميع العمليات للمؤثرات المنطقية.

جدول 7-4			
النتيجة	مثال	**الرمز**	المؤثر
1	10 > 8 && 9 > 7	&&	AND
1	1 0 < 8 || 7 < 8	||	OR
1	! (10 = = 8)	!	NOT
		~	Bitwise unary NOT
		&	Bitwise AND
		|	Bitwise OR
		^	Bitwise exclusive OR
		>>	Shift right
		>>>	Shift right zero fill
		<<	Shift left
		&=	Bitwise AND
		&=	Bitwise AND assignment
		|=	Bitwise OR assignment
		^=	Bitwise exclusive OR assignmentJAVA
		>>=	Shift right assignment
		>>>=	Shift right zero fill assignment
		<<=	Shift left assignment

وهذا مثال شامل لما سبق:

```
1. // Demonstrate the bitwise logical operators.
2. class Chp4_4 {
3.         public static void main(String args[]) {
4.                 //BitLogic
5.                 System.out.println("\nBitLogic");
6.
7.                 String binary[] = {
8.                 "0000", "0001", "0010", "0011", "0100", "0101", "0110", "0111",
9.                 "1000", "1001", "1010", "1011", "1100", "1101", "1110", "1111"
10.                };
11.                int a = 3; // 0 + 2 + 1 or 0011 in binary
12.                int b = 6; // 4 + 2 + 0 or 0110 in binary
13.                int c = a | b;
14.                int d = a & b;
15.                int e = a ^ b;
16.                int f = (~a & b) | (a & ~b);
17.                int g = ~a & 0x0f;
18.                System.out.println(" a = " + binary[a]);
```

```
19.                 System.out.println(" b = " + binary[b]);
20.                 System.out.println(" a|b = " + binary[c]);
21.                 System.out.println(" a&b = " + binary[d]);
22.                 System.out.println(" a^b = " + binary[e]);
23.                 System.out.println("~a&b|a&~b = " + binary[f]);
24.                 System.out.println(" ~a = " + binary[g]);
25.
26.                 // Left shifting a byte value.
27.                 System.out.println("\nLeft shifting a byte value.");
28.                 byte aa = 64, bb;
29.                 int i;
30.                 i = aa << 2;
31.                 bb = (byte) (aa << 2);
32.                 System.out.println("Original value of aa: " + aa);
33.                 System.out.println("i and bb: " + i + " " + bb);
34.
35.                 // Left shifting as a quick way to multiply by 2.
36.                 System.out.println("\nLeft shifting as a quick way to multiply by 2.");

37.                 int num = 0xFFFFFFE;
38.                 for(i=0; i<4; i++) {
39.                 num = num << 1;
40.                 System.out.println(num);
41.                                                         }
42.                 // Unsigned shifting a byte value.
43.                 System.out.println("\nUnsigned shifting a byte value.");

44.                 char hex[] = {
45.                 '0', '1', '2', '3', '4', '5', '6', '7',
46.                 '8', '9', 'a', 'b', 'c', 'd', 'e', 'f'
47.                 };
48.                 bb = (byte) 0xf1;
49.                 byte cc = (byte) (bb >> 4);
50.                 byte dd = (byte) (bb >>> 4);
51.                 byte ee = (byte) ((bb & 0xff) >> 4);
52.                 System.out.println(" bb = 0x"+ hex[(bb >> 4) & 0x0f] + hex[bb &
0x0f]);
53.                 System.out.println(" bb >> 4 = 0x"+ hex[(cc >> 4) & 0x0f] + hex[cc
& 0x0f]);
54.                 System.out.println(" bb >>> 4 = 0x"+ hex[(dd >> 4) & 0x0f] +
hex[dd & 0x0f]);
55.                 System.out.println("(bb & 0xff) >> 4 = 0x"+ hex[(ee >> 4) & 0x0f] +
hex[ee & 0x0f]);
56.
57.                 //OpBitEquals
58.                 System.out.println("\nOpBitEquals");
59.                 a = 1;
60.                 b = 2;
61.                 c = 3;
62.                 a |= 4;
63.                 b >>= 1;
```

```
64.             c <<= 1;
65.             a ^= c;
66.             System.out.println("a = " + a);
67.             System.out.println("b = " + b);
68.             System.out.println("c = " + c);
69.
        }
70. }
```

فيكون ناتج تنفيذ البرنامج كالتالي:

```
BitLogic
 a = 0011
 b = 0110
 a|b = 0111
 a&b = 0010
 a^b = 0101
~a&b|a&~b = 0101
 ~a = 1100

Left shifting a byte value.
Original value of aa: 64
i and bb: 256 0

Left shifting as a quick way to multiply by 2.
536870908
1073741816
2147483632
-32

Unsigned shifting a byte value.
 bb = 0xf1
 bb >> 4 = 0xff
 bb >>> 4 = 0xff
(bb & 0xff) >> 4 = 0x0f

OpBitEquals
a = 3
b = 1
c = 6
```

4.6.4 مؤثرات التخصيص Assignment Operators

يأخذ معامل الإلحاق (=) حداً واحداً، متحولاً كان أو قيمة ، ويلحق قيمة هذا الحد بمتحول . العنصر اليساري في عبارة الإلحاق يجب أن يكون متحولاً . وبشكل عام يجب أن يكون المتحول والحد من نفس نوع المعطيات .

في هذه العملية يجب أن يتوافق نمط معطيات المتحول الموجود على اليسار مع نمط معطيات القيمة الموجودة على اليمين ، فعلى سبيل المثال: ستكون العبارة int x=1.0 غير نظامية وذلك لأن نمط معطيات المتحول x هو int في حين أنة لا يمكن إسناد قيمة من نوع double(1.0) إلى متحول من نوع int بدون القيام بعملية تحويل للنمط (type casting). سنتعرف على عمليات تحويل النمط لاحقاً.

جدول 4-8

المؤثر	النتيجة	الطريقة الحديثة	التخصيص التقليدي
+ = addition assignment operators	11	A + = 5	A = a +5
Subtraition assignment operators	1	A - = 5	A = a – 5
Multipication assibnment operators	30	A * = 5	A = a + 5
Division assignment operators	2	A / = 3	A = a / 3

4.6.5 مؤثرات الزيادة والنقصان Increment& Decrement

توفر لغة Java عملية الزيادة بواحد الأحادية (++) وعملية الإنقاص بواحد الأحادية (--)، اللتان قمنا بتلخيصهما في الجدول 4-9.

جدول 4-9

مؤثر زيادة واحد	7	A ++	A = a + 1
مؤثر نقصان واحد	5	A --	A = a – 1
Example 1		Example 2	
B=3; A=++B; *// A is 4, B is 4*		B=3; A=B++; *// A is 3, B is 4*	

4.6.6 مؤثر باقي خارج القسمة %

يستخدم لمعرفة باقي القسمة (لتحديد هل الأرقام الموجودة في المتغير زوجية أو فردية فمثلا إذا كانت قيمة a = 5 وكتب C = a % 2 يكون باقي الرقم 2 / 5 = 1

4.7 أسبقية العوامل وترتيب الحدود Operator precedence

تنفذ معظم البرامج عمليات حسابية ويلخص الجدول 4-10 جميع العمليات الحسابية المستخدمة ، حيث تنفذ حسب ترتيب معين محدد تبعاً لقواعد الأولوية بين العمليات التي تماثل قواعد الأولوية المستخدمة في الجبر. حيث تظهر العمليات في الشكل حسب تسلسل تناقص درجة أولوياتها من الأعلى إلى الأسفل. يوضح العمود الرابع (Associativity) طريقة تجميع العمليات المذكورة في العمود الثاني.

جدول 4-10 Operator precedence

Priority	Operator	Description	Associativity
1	() [ ] -> . sizeof		Left
2	++ --	increment/decrement	Right
	~	Complement to one (bitwise)	
	!	unary NOT	
	& *	Reference and Dereference (pointers)	
	(*type*)	Type casting	
	+ -	Unary less sign	
3	* / %	arithmetical operations	Left
4	+ -	arithmetical operations	Left
5	<< >>	bit shifting (bitwise)	Left
6	< <= > >=	Relational operators	Left
7	== !=	Relational operators	Left
8	& ^ |	Bitwise operators	Left
9	&& ||	Logic operators	Left
10	?:	Conditional	Right
11	= += -= *= /= %= >>= <<= &=	Assignation	Right

	^= |=		
12	,	Comma, Separator	Left

يتم تقيم الحدود بشكل كامل من اليسار إلى اليمين ، فإذا تساوت أولوية العوامل التي تملك نفس الأسبقية وراء بعضها فان المترجم ينفذ العوامل من اليمين إلى اليسار .

```
1. // برنامج يوضح الاختلاف بين العوامل وأولوية كلاً منها
2. class Chp4_5 {
3.   public static void main(String[] args) {
4.         String binary[] = {
5.         "0000", "0001", "0010", "0011", "0100", "0101", "0110", "0111",
6.         "1000", "1001", "1010", "1011", "1100", "1101", "1110", "1111"};
7.         int a = 3; // 0 + 2 + 1 or 0011 in binary
8.         int b = 6; // 4 + 2 + 0 or 0110 in binary
9.         int c = a | b;
10.        int d = a & b;
11.        int e = a ^ b;
12.        int f = (~a & b) | (a & ~b);
13.        int g = ~a & 0x0f;
14.        System.out.println(" a = " + binary[a]);
15.        System.out.println(" b = " + binary[b]);
16.        System.out.println(" a|b = " + binary[c]);
17.        System.out.println(" a&b = " + binary[d]);
18.        System.out.println(" a^b = " + binary[e]);
19.        System.out.println("~a&b|a&~b = " + binary[f]);
20.        System.out.println(" ~a = " + binary[g]);
21.
22.        System.out.println(++a);//هنا يتم اولاً الزيادة ثم الطباعة بحسب الأولوية
23.        System.out.println(a);
24.        System.out.println((a++)+(++b));
25.
26.        System.out.println(a);
27.        System.out.println(b);
28.
29.        System.out.println(b*a%2-a);
30.                          }
31. }
```

فيكون ناتج البرنامج السابق كما يلي

```
 a  = 0011
 b  = 0110
 a|b  = 0111
 a&b  = 0010
 a^b  = 0101
~a&b|a&~b  = 0101
 ~a  = 1100
4
4
11
5
7
-4
```

المتغير المعلن برأس الحلقة for يمكن استخدام ه داخل الحلقة فقط بخلاف لغة C حيث بعد تعريفه و يمكن استخدام ه خارج الحلقة فهذا الكود من الأخطاء الشائعة

```
for(int i=0;i<4;i++){}
System.out.println(i);
```

هذا الكود من الأخطاء الشائعة والسبب أن I لم يتم التعرف علية فتصحيح هذا الخطاء نجعل I من نوع Static static int I=507;

```
1. //Can't make a static reference to nonstatic variable i in class Chp4_6.
2. class Chp4_6{
3.   int i=507;//Error
4.   public static void main(String[] args) {
5.     int b = i;
6.     System.out.println(b);
7.                   }
8. }
```

4.8 التعبير Expression

التعابير هي أساس إي شفرة برمجية ، بالتعاون مع الأساسيات الأخرى للغة جافا نستخدم التعابير لحساب قيم المتغيرات وتحليل النتيجة وذلك حتى نستطيع التحكم في طريقة سير وعمل البرنامج. ويتم ذلك عن طريق حساب القيمة وإرجاعها للكمبيوتر للقيام بفعل معين.

باختصار التعابير هي عبارة عن مجموعة متغيرات ومشغلات وأوامر لحساب قيمة معينة.

4.8.1 الفرق بين الجملة والتعبير Statement & Expressions

قلنا سابقا أن برامج الحاسوب هي مجموعة من الأوامر تحدد للحاسوب ما ينبغي عمله وكل أمر من هذه الأوامر يسمى جملة ويمكن تجميع عدد من الجمل بواسطة الحاصرات لتكوين تكتلات وحين تتضمن الجملة عملية رياضية أو تعبير رياضي فإن هذه الجملة تسمى تعبيرا .

4.9 التحويلات في الأنماط العددية Conversion and Casts

بمعنى التبديل بين أنواع البيانات مثل القناع وأحيانا يُجبر المبرمج في استخدام هذه العملية.

- عند تحويل نمط صغير إلى كبير فان المترجم تلقائياً يقوم بهذه العملية .

```
Byte i=10;
Long k=20;
K=I;
```

- عند تحويل نمط كبير إلى صغير فأنة من الضروري عمل casting أي قناع مثل

```
Int i=256;
Byte b=(byte)I;
```

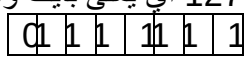
هنا يتم أخذ 8 بت فقط ويتم إسنادها إلى b فتكون قيمة b=0 .

فكيف تمت العملية ؟

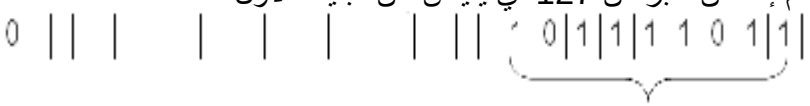
إليكم هذه الطريقة التي بسطتها :

```
{
{
 If (i<=127) b=I;
Else
If (i>127&bit [8].I=0) b= (sum bit [1] ->bit [7]). I
Else
If (i>127&bit [8].I=1) b= ((sum bit [1] ->bit [7]). I) -128
}
If (I<0) b*=-1;
}
```

إذا كان i اقل من 127 أي يملئ بايت واحد فان b يأخذ القيمة مباشرتاً

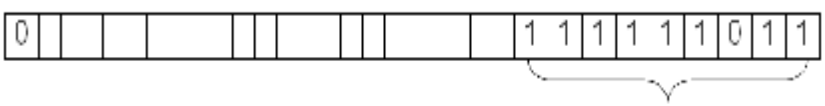


أم إذا كان اكبر من 127 أي يفيض عن البايت الأول



و البت 8 =0 أي غير مؤشر
فإننا نقوم بجمع محتوى البت الأول إلى السابع أي تحويل من ثنائي إلى عشري فنحصل على العدد 123 فيكون b=123

أم إذا كان اكبر من 127 أي يفيض عن البايت الأول و البت 8 = 1 أي مؤشر



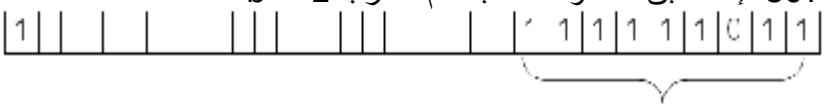
محتوى I بالعشري يساوي 507 وعند تحول البايت الأول إلى السابع من الثنائي إلى العشري فإننا نحصل على العدد 123 وبطرحة من 128 فيكون b=-5
واليكم هذا المثال

```
1. //برنامج يعمل على تحويل أنماط البيانات
2. public class Chp4_7{
3.     public static void main(String[] args) {
4.     int i=507;
5. //تتم عملية التحويل
6.     byte b=(byte)i;
7.     System.out.println(b);
8.     }
9. }
```

فيكون ناتج تنفيذ البرنامج:

−5

جميع الشروط التي ذكرناها في حالة القيم الموجبة أم إذا كانت قيمة -=I
507 فإننا نطبق الخطوات السابقة تم نضرب b*=-1



عندما يكون لديك عدد مؤشر أي سالب وتريد تمثيله بالثنائي فما عليك إلى أن تعتبر العدد موجب و تحول العدد إلى الثنائي تم تجد المتمم الأحادي له تم تضيف له واحد ويصبح عدد مؤشر بالثنائي ، وخلاصة القول يمكن أن تحول العدد إلى المتمم الثنائي مباشرة .
محتوى I بالعشري يساوي 507- وعند تحول البايت الأول إلى السابع من الثنائي إلى العشري فإننا نحصل على العدد 123 وبطرحة من 128 فيكون b = -5 وبضرب b*=-1 فتكون قيمة B=5 .

في حالة تحويل نمط من float إلى int فإننا نبعد الكسور فقط

```
float i=256.6f;
int b=(int)i;
```

فتكون قيمة b=256.

في هذا الكود سيقوم المترجم بإصدار خطاء !

```
int b=200;
short i=b;
```

والسبب إن b مكون من 4 بايت و I من 2 بايت وهذه العملية تحتاج إلى تحويل .
وهذا كود شامل لما سبق

```
1. //كود شامل لعملية التحويل
2. class Chp4_8 {
3.         public static void main(String args[]) {
4.                 byte b;
5.                 int i = 257;
6.                 double d = 323.142;
7.                 System.out.println("\nConversion of int to byte.");
8.                 b = (byte) i;
```

```
9.              System.out.println("i and b " + i + " " + b);
10.             System.out.println("\nConversion of double to int.");
11.             i = (int) d;
12.             System.out.println("d and i " + d + " " + i);
13.             System.out.println("\nConversion of double to byte.");
14.             b = (byte) d;
15.             System.out.println("d and b " + d + " " + b);
16.             }
17. }
```

فيكون الناتج كما يلي

```
Conversion of int to byte.
i and b 257 1

Conversion of double to int.
d and i 323.142 323

Conversion of double to byte.
d and b 323.142 67
```

مثال أخر:

```
1. //كود لعملية التحويل بين الأنماط
2. class Chp4_9 {
3.      public static void main(String args[]){
4.              byte b = 42;
5.              char c = 'a';
6.              short s = 1024;
7.              int i = 50000;
8.              float f = 5.67f;
9.              double d = .1234;
10.             double result = (f * b) + (i / c) - (d * s);
11.             System.out.println((f * b) + " + " + (i / c) + " - " + (d * s));
12.             System.out.println("result = " + result);
13.             }
14. }
```

وهذا ناتجة

```
238.14 + 515 - 126.3616
result = 626.7784146484375
```

أي عملية على byte يجب عمل تحويل وإلا ينتج خطاء.

فهذا الكود خاطئ ! وتصحيح هذا الخطاء نعمل Casting

```
b=(byte) (b+5)
```

```
1. //كود يوضح الأخطاء الشائعة لتجاهل عملية التحويل
2. // Incompatible type for =. Explicit cast needed to convert int to byte.
3. class Chp4_10{
4.         public static void main(String args[]){
5.                 byte b = 42;
6.                 //جصول خطأ
7.                 b=b+5;//Error
8.                 System.out.println(b);
9.                   }
10. }
```

لا تؤدي عملية تحويل النمط إلى تغير المتحول الذي طبق علية (أي يحافظ المتحول على نمطه وعلى قيمته).

مثال:

```
Double d=4.4;
Int i=(int)d;//لا يحصل تغير
```

تجاهل عملية التحويل في حال إسناد قيمة ما إلى متحول ينتمي إلى مجال أصغر (مثل إسناد قيمة من النوع double إلى متحول من النمط int) يحدث خطا أثناء الترجمة.

4.10 إظهار نص في صندوق الحوار Show Text From Dialog Box

بالرغم من إظهار النص السابق في نافذة الأوامر ، إلا أن كثير من التطبيقات الجافا تستخدم صناديق الحوار لإظهار النصوص بدلاً من نافذة الأوامر ، معظم البرامج وبخاصة متصفحات الانترنت تستخدم صناديق الحوار في كثير من التطبيقات.

وصناديق الحوار هي عبارة عن نافذة سيتم إظهار الرسائل المهمة الموجهة للمستخدم فيها ، أو التي تعطي خرجاً من البرنامج والـ class المسمى JOptionPane بالمناهج التي تساعدنا في إظهار صناديق الحوار المختلفة.

مثال لإظهار عبارة الترحيب "Welcome to Java Programming "

```
1. /* Fig Chp4_11
2. Printing multiple lines in a dialog box
```

```
3.
4. */ Java extension packages
5. import javax.swing.JOptionPane;  // import class JOptionPane
6. public class Chp4_11 {
7.
8. // main method begins execution of Java application
9. public static void main( String args[ ] )
10.  {
11.  JOptionPane.showMessageDialog(
12.  null, "Welcome\nto\nJava\nProgramming!" );
13.
14.  System.exit( 0 );  // terminate application
15.  }  // end method main
16.  }  // end class Chp4_11
```

شرح المثال
السطر5 يتم من خلاله استدعاء الحزمة الخاصة بعرض صندوق الحوار لكي يتم تعريف وتحميل المناهج التي سنستخدمها داخل البرنامج.

الأسطر 12,11 يشير إلى استدعاء المنهج Show.Messeg.Dialog من الكائن المسمى JOptione وهذه المنهج تتطلب برمترات الأول دائماً يكون null وهو يحدد المكان الذي سيظهر صندوق الحوار، في حالة null سيظهر صندوق الحوار بمنتصف الشاشة، أم الثاني فهو النص المراد إظهاره .
ويكون ناتج تنفيذ البرنامج

شكل 1-4 يوضح صندوق الأخراج

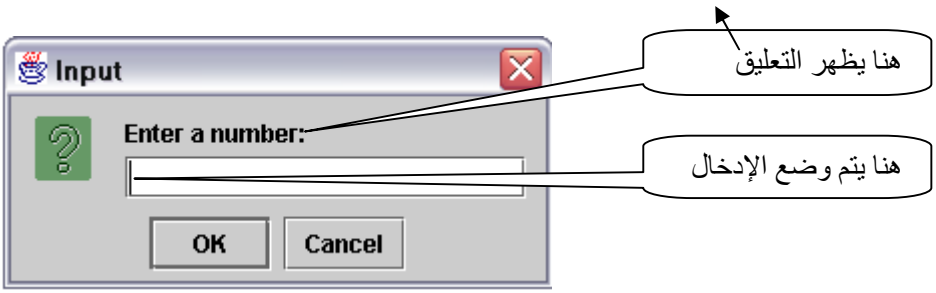
4.11 الدخل بواسطة صناديق الحوار Show Input Dialog Boxes

غالباً ما يحتاج المبرمج من إدخال بعض البيانات من لوحة المفاتيح ،وهذا يحتاج إلى بعض الخطوات الأولية لأعداد وتهيئة مترجم ال Java .

- يتم استدعاء مكتبة الإدخال التي تعرض صندوق الحوار (import javax.swing.*) في بداية البرنامج .
- يتم استدعاء منهج الإدخال

JOptionPane.showInputDialog("Enter a number:").

ويظهر مربع حوار كما في الشكل 4-1.



شكل 4-2 يوضح صندوق الإدخال

وتستطيع اختيار الأيقونة المناسبة لصندوق الحوار كما في الشكل 4-3.

نوع رسالة صندوق الحوار	الرمز	الوصف
JOptionPane. ERROR_ MESSAGE		عرض صندوق حوار يبين رسالة خطأ للمستخدم
JOptionPane. INFORMATION_ MESSAGE		عرض صندوق حوار مع رسالة للمستخدم
JOptionPane. WARNING_ MESSAGE		رسالة تحذيرية للمستخدم
JOptionPane. QUESTION_ MESSAGE	?	سؤال للمستخدم يجب الإجابة عليه بنعم أو لا
JOptionPane. PLAIN_ MESSAGE	لا يوجد رمز	يظهر رسالة في الصندوق بدون رموز

شكل (4-3) الرموز التي تظهر مع صندوق الحوار

- يستقبل القيمة المدخلة بواسطة متغير من نوع String .

```
1. //برنامج يدخل قيمة بواسطة مربع الإدخال ويقوم بطباعته
2. import javax.swing.*;
3. class Chp4_12 {
4.         public static void main(String args[]){
5.                 int b;
6.                 String s;
7.                 s=JOptionPane.showInputDialog("Enter a number:");
8.                 b=Integer.parseInt(s);
9.                 System.out.print("The Number "+b);
10.                } // end main
11. }//end class
```

دالة تقوم بتحويل سلسلة رقمية إلى عددية

يعاد الدخل من صندوق الحوار كسلسلة نصية (أي من نوع String) أي إذا أدخلت 12 فإنها تعاد "12" ولتتم العمليات الحسابية على العنصر المدخل فأننا استخدمنا دوال التحويل (Integer.parseInt) . ويمكنك تحويل إلى عدة أنماط ، ستتعرف عليها في الفصول اللاحقة .

4.12 الأخطاء البرمجية Programmatic Errors

أثناء تصحيح البرامج النصية من جانب المحرر الخاص بك قد تواجه أنواع عديدة من الأخطاء. قد تتسبب بعض هذه الأخطاء في التنفيذ الخاطئ للبرامج النصية الخاصة بك، أو إيقاف تنفيذ للبرنامج، أو إعادة نتائج غير صحيحة. فمهما كنت مبرمج محترف فلا تجنب هذه الأخطاء فعلى المبرمج أن يكون ملماً بجميع أنواع الأخطاء.

4.12.1 أنواع الأخطاء البرمجية Programmatic Errors Types

1. أخطاء قواعدية

خطأ بناء الجملة هو خطأ شائع ينتج عن بناء جملة برمجية غير صحيحة. على سبيل المثال، الأمر الذي به خطأ إملائي أو تمرير عدد غير صحيح من الوسائط إلى دالة

يصدران خطأ. كما يمكن أن تمنع الأخطاء الموجودة في بناء الجملة للبرنامج النصي هذا البرنامج من التشغيل. فعلى سبيل المثال يمكن كتابة برنامج فيه جملة خاطئة من حيث ترتيب الأحرف أو تنقيص بعض الأحرف .

```
1. // برنامج يوضح الخطاء القواعدي
2. // Error
3. public class Chp4_13{
4.         public static void main(String args[]){
5.                 int a=19;
6.                 //خطاء
7.                 system.out.println(a/0);
8.                         }
9. }
```

كما هو مبين في الشكل 3-4 فأن المترجم سيقوم بكشف الخطاء ، وسيبين رقم سطر الخطاء كما في الشكل التالي.

```
Undefined variable, class, or package name: system
                system.out.println(a/0);
                ^
1 error
```

شكل 4-4

2. أخطاء وقت التشغيل Runtime Errors

تحدث أخطاء وقت التشغيل بعد أن يبدأ البرنامج النصي التنفيذ وتنشأ عن الإرشادات البرمجية التي تحاول أداء إجراءات مستحيلة وقد تؤدي أحياناً إلى إغلاق البرنامج بشكل غير طبيعي. على سبيل المثال يدخل المستخدم قيمة نصية من نوع (String) والبرنامج يطلب قيمة عددية من نوع (Int) مثلاً فيعجز المحرر عن معالجتها ولحل هذا الخطاء ينبغي على المبرمج عرض رسالة إرشادية تبين للمستخدم نوعية الإدخال. و يحتوي البرنامج النصي التالي على دالة تقسم المتغير على صفر (وهي عملية حسابية غير صحيحة) وينشأ عنها خطأ وقت التشغيل ما يسمى (Runtime).

```
1. //برنامج يوضح الخطاء الذي قي زمن التنفيذ
2. // ArithmeticException: / by zero
3. public class Chp4_14{
4.         public static void main(String args[]){
5.                 int a=19;
6.                 //خطاء Run Time
7.                 System.out.println(a/0);
8.                         }
```

```
9. }
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Chp2_13.main(Chp2_13.java:7)
```

يجب تصحيح الأخطاء التي تنتج في وقت التشغيل للبرنامج النصي الخاص بك ليتم تنفيذها دون مقاطعة.

3. أخطاء منطقية Logical Errors

قد يكون الخطأ المنطقي هو أصعب الأخطاء في إمكانية الكشف عنها. مع الأخطاء المنطقية، والتي تحدث بسبب أخطاء في الكتابة أو أخطاء في المنطق البرمجي، يتم تشغيل البرنامج النصي بنجاح، ولكن ينتج عنه نتائج غير صحيحة. على سبيل المثال، قد يُرجع برنامج نصي من جانب المحرر تم إنشاؤه ليقوم بفرز قائمة من القيم ترتيب غير دقيق في حالة احتواء البرنامج النصي على علامة > (أكبر من) لمقارنة القيم، في حين أنه يجب استخدام علامة <(أصغر من). فعلى سبيل المثال يراد منك التحقق من العدد المدخل هل يقع بين 1-100 أم لا.

كما في هذا المثال .

```
1.  //برنامج يوضح نوع الخطاء المنطقي
2.  import javax.swing.*;//
3.
4.  public class Chp4_15{
5.          public static void main(String args[]){
6.                  String s;
7.
8.                  int b;
9.
10.                 s=JOptionPane.showInputDialog("Enter a number:");
11.
12.                 b=Integer.parseInt(s);
13.                 //              |هنا حصل الخطاء
14.                 System.out.println((b<=1)&&(b<=100));
15.
16.                         }
17. }
```

4.12.2 طرق تصحيح الأخطاء – التنقيح Debugging

يمكنك استخدام طرق تصحيح مختلفة لتحديد موقع مصدر الأخطاء ولاختبار التطبيقات الخاصة بك.

1. التصحيح عند الحاجة (JIT)

عند مقاطعة خطأ وقت التشغيل تنفيذ برنامج المحرر فيبدأ بتشغيل مصحح البرامج النصية تلقائياً، ويعرض عبارة يشير إلى السطر الذي سبب الخطأ، وينشئ رسالة خطأ. مع هذا النوع من التصحيح أحياناً ، يوقف جهاز الكمبيوتر الخاص بك أي تنفيذ آخر للبرنامج بشكل مرحلي. حتى يتم تصحيح الأخطاء.

2. التصحيح باستخدام نقطة الإيقاف

عند حدوث خطأ ولا يمكنك تحديد مصدر الخطأ بسهولة، فمن المفيد أحياناً أن تقوم بإجراء إعداد مسبق لنقطة الإيقاف. تقوم نقطة الإيقاف بإيقاف التنفيذ عند سطر محدد بشكل مرحلي في البرنامج النصي. يمكنك تعيين نقطة إيقاف أو أكثر مختلفة قبل السطر المشكوك في صحته للبرنامج النصي ثم استخدم المصحح لاختبار قيم المتغيرات أو الخصائص الـ تي تم تعيينها في البرنامج النصي. بعد تصحيح الخطأ، يمكنك مسح نقاط الإيقاف حتى يمكن تشغيل البرنامج النصي دون مقاطعة. وهذا لا يتواجد إلا في المحررات الحديثة مثل (Jbuilder).

4.12.3 تلميحات عن تصحيح البرامج النصية

بعيداً عن مصحح البرامج النصية، يمكن أن تسهم مجموعة جيدة من تلميحات البرامج النصية بشكل كبير في تقليل الفترة الزمنية التي تقضيها في التحقق من مصدر أخطاء البرنامج النصي. بالرغم من أن معظم الأخطاء تنتج من مصادر واضحة، أو الأوامر التي بها أخطاء إملائية أو متغيرات مفقودة، وقد يتعذر العثور على أنواع معينة من الأ خطاء المنطقية والتنفيذية.

تمارين الفصل:

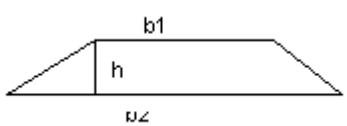
1. أذكر السبب الذي يجعل كل من الثوابت التالية غير مقبولة:

4,22
10+20
$33

2. أكتب برنامج يقراء ثلاثة أعداد ويطبع المتوسط الحسابي لهما؟
3. أكتب برنامج يعمل على تبديل قيم متغيرين بدون استخدام متغير ثالث؟
4. أكتب برنامج يقوم بقراءة نصف قطر دائرة من المستخدم ، ثم يقوم بعد ذلك بطباعة قطر الدائرة ، المحيط والمساحة؟ //(أفترض أن الثابت الطبيعي 3.14159= π)
// ملاحظة: يمكنك استخدام الثابت Math.PI وذلك للثابت الطبيعي وهذه القيمة تعتبر أدق من القيمة 3.14159.
5. أكتب برنامج لتحويل درجة الحرارة 136.36- بالفهرنتي إلى مئوي ؟
// القانون : C=5/9(F-32)
6. إذا كانت J=0 ,I=6 ,B=-2 ,A=3 . جد قيمة كل التعابير الحسابية التالية

- A*2/B
- B-I*J+(2*B)
- J+(B*(A-2)-7)

7. توفي رجل وترك مبلغاً قيمتة X وترك عددا من الأبناء الذكور (N1) وعدد من الآناث (N2) وكذلك زوجه . أكتب برنامج لتوزيع الميراث عليهم تبعاً للقواعد الألهية التالية:
   - (الزوجه لها الثمن مما ترك).
   - (للذكر مثل حظ النثيين).
8. أفترض أن سيارة تحركت من السكون والتسارع بمقدار a لعدد من الثواني t وأن السرعة النهائية هي v والمسافة المقطوعة هي d.استخدم القوانين التالية:
   - d=1/2 * a * t ^ 2
   - v=a * t
9. أكتب برنامج لحساب مساحة شبة المنحرف التالي:



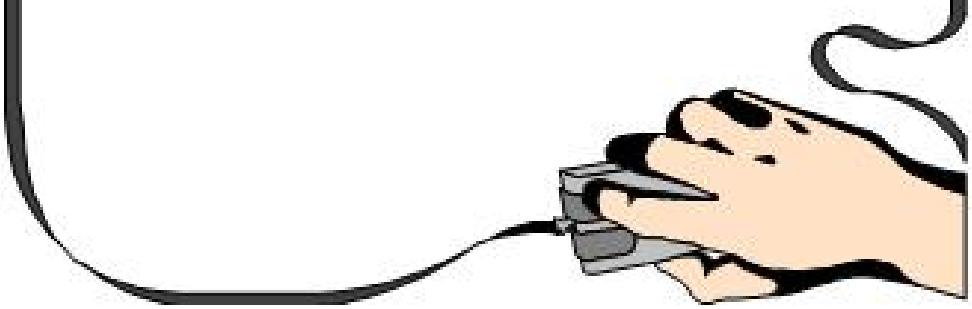
حيث أن مساحة شبة المنحرف = (b1+b2)*h/2

Chp5

عبارات التحكم

في نهاية هذا الفصل سوف تتعلم :

- التعرف على مفهوم التحكم بالبرنامج.
- استخدام عبارات الاختيار للتحكم في تنفيذ البرنامج .
- استخدام عبارات الحلقة للتحكم بتكرار العبارات .
- تنفيذ عملية التحكم بالبرنامج باستخدام **break** ، **continue** .
- التعرف على التقنيات لحل المسائل
- امتلاك القدرة على تطوير الخوارزميات .
- امتلاك المقدرة على استخدام البني **if** ، **if/else** ، **switch** .
- امتلاك القدرة على استخدام البني **while/do** ، **for** لتكرار تنفيذ مجموعة من التعليمات .



5.1 مقدمة INTRODUCTION

قبل كتابة أي برنامج لحل مسألة ما، يجب أن يتوفر لدينا فهماً شاملاً للمسألة المطروحة وأسلوباً منهجياً للحل . وعند كتابة البرامج ، يجب علينا فهم طبيعة وأنواع الأقسام المؤلفة له كما يجب علينا استخدام مبادئ وطرق مبرهن عليها مسبقاً. وسوف نقوم في هذا الفصل بمناقشة جميع القضايا المتعلقة بالمبادئ والطرق الخاصة بالبرمجة المهيكلة . يمكن استخدام التقنيات التي سوف تتعلمها في هذا الفصل .

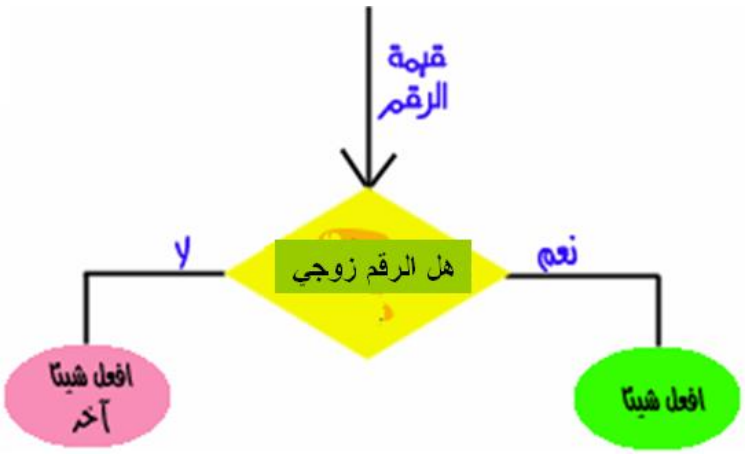
5.2 بني التحكم Control Structures

تمتلك لغة Java عدة جمل للتحكم في يسير البرنامج فتقسم إلى قسمين:

1- بني اختيار (selection construct): وفيها يتم تنفيذ أعمال جزئية معينة إذا تحقق الشرط الموجود قي هذه البنية .
2- بني تكرار (iteration construct): ويجري من خلالها تكرار أعمال معينة عدداً محدداً من المرات ، أو حتى يتحقق شرط معين.

5.2.1 العبارة الشرطية (IF Statement)

قد نحتاج في برامجنا تنفيذ عمل معين في حال تحقق شرط ما أو الاستمرار في سير البرنامج في حال عدم تحقق هذا الشرط ، فمثلاً إذا كان لدينا برنامج يبحث يتحقق هل العدد زوجي أم فردي . فلن نستطيع معرفة العدد إلا بواسطة عبارة IF فهي الوسيلة الوحيدة التي توجه البرنامج اعتمادا على القرار والشروط والشكل 1-5 يبين سير العملية لهذه العبارة.



الشكل 1-5 المخطط الصندوقي لبنية if

إن هذا الشكل يمثل حالة اتخاذ قرار. لو نظرت للشكل المعيّن في وسط الصورة ستجد أن لدينا فيه سؤالاً له جواباً من اثنين إما أن يكون نعم (صح)، أو لا (خطأ). و في الشكل السابق، كان السؤال عن قيمة خاله رقم محدد) هل زوجي أم فردي ؟ (و هذا هو السؤال. فإذا كانت الإجابة نعم فإننا سنفعل شيئاً. و ننفذ ما في الدائرة على اليمين، أما إذا كانت الإجابة لا، فسنقوم بفعل شيء آخر و تنفيذ ما في الدائرة على اليسار .

و لتطبيق ذلك في البرمجة، لننظر للجمل التالية :

```
if( Boolean Expression ) {

    Do something here
    ..
    ..

} else {

    Do something
    ..
    ..
}
```

حيث نبدأ بالكلمة (if) و بعدها نفتح قوسين دائريين نضع في داخلها شرطاً تكون قيمته إما true أو false. بعدها نفتح قوس مجموعة نضع في داخله ما نريد تنفيذه في حال كانت الجملة الشرطية صحيحة .(true) و هذا ما نسميه) جملة إذا (أو .(If Statement) و هي أول نوع من الجمل الشرطية .

و الآن لنتأمّل معاً الكود التالي :

```
1.  برنامج يتحقق من العدد المدخل بواسطة جمل القرار//
2.  import javax.swing.*;//
3.
4.  public class Chp5_1{
5.  public static void main(String args[]){
6.   String s;
7.
8.   int b;
9.
10.   s=JOptionPane.showInputDialog("Enter a number:");
11.
12.   b=Integer.parseInt(s);
13.
14.   if(b%2==0)
15.    {
16.        System.out.println("Even");
17.    }
18.   else
19.    {
20.      System.out.println("Odd");
21.    }
22.  }
23.  }
```

إننا في هذه الأسطر نقوم بتطبيق جملة إذا. بحيث نلاحظ في السطر 14 أننا وضعنا الجملة المنطقية (الرقم 2 Mod). فإذا كانت العبارة صحيحة فسنقوم بتنفيذ الكود في الجزء المحصور بين قوسي المجموعة الذي يلي الجملة المنطقية مباشرة. أما إذا كانت العبارة خاطئة، فسنقوم بتنفيذ الجزء الذي يلي كلمة else و المحصور أيضاً بين قوسي مجموعة .

لاحظ الأسطر 20,16 أننا قمنا بكتابة التعليمات على بعد مسافة جدولة واحدة عن if. وهي طريقة أختيارية ولكن ينصح بها بقوة لإنها تدعم البرمجة المهيكلة فنقترح أن تكون حجم المسافة ما تعادل ثلاثة فراغات أي ¼ إنش. هذا مع العلم أن مترجم لغة Java يقوم بتجاهل الفراغات وحروف الجدوله taps وحرف الأسطر الجديدة.


يمكن الاستغناء عن أقواس المجموعة في حالة كانت الجملة المراد تنفيذها جملة واحدة فقط. كما في المثال نفسه. و لكن إذا أردت تنفيذ مجموعة من الجمل، فيجب أن تحصرها بين قوسي مجموعة حتى تضمن أن يتم تنفيذها حسب ما تشاء .


يمكنك الاستغناء عن القسم else إذا أردت عمل شيء في حالة كان الجملة المنطقية صحيحة، و لا ترغب بعمل شيء إذا كانت الجملة المنطقية خاطئة.

اكتب البرنامج و قم بتشغيله، و حاول تغيير ا لشرط حسب ما ترى. جرّب أن تضع شروطاً منطقية أخرى.


يعتبر كتابة فاصلة منقوطة في نهاية الشرط if خطاء شائعاً.

مثلاً

```
If (i<1);

B=100;
```

من الصعب الكشف عن هذا الخطاء وذلك لأنة لا يسبب خطأ في الترجمة ولا أثناء التنفيذ ، بل هو خطاء منطقي .


تنظيم الأقواس في جمل التحكم من العادات البرمجية الجيدة فإنها تجعل من البرنامج اكتر قابلية للقراءة.


يفضل معظم المبرمجين كتابة قواسي البداية والنهاية قبل البدئ بكتابة كل من التعليمات ، مما يجنبهم نسيان أحد أو كلا القوسين.

- **المؤثر الثلاثي:**

تمتلك لغة Java على ما يسمى بالعملية الشرطية المختصرة (:?) conditional operator
وهي أسرع بالتنفيذ من الجملة if/else فهي تشبهها .
حيث يشكل المعامل الأول للشرط والمعامل الثاني الجزء الذي يتم تنفيذه إذا كان الشرط محققاً ويشكل المعامل الثالث الجزء الذي يتم تنفيذه إذا كان الشرط غير محققاً .
فعلى سبيل المثال يتضمن التعبير التالي :

```
System.out.println(a >= 60 ? "passed" : "failed");
```

أي ما تكافئ

```
If(a>=60)
        System.out.println("failed");
Else
System.out.println("passed");
```

لاحظ الفرق بين الجملتين فأيهما أسهل برأيك ؟
وهذا المثال يعزز ذلك

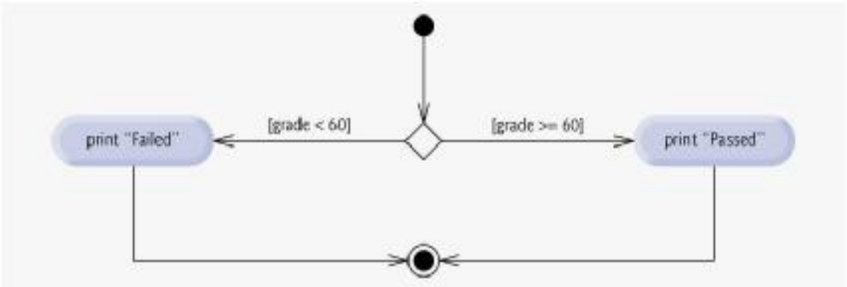
```
1. //برنامج يوضح العملية الشرطية المختصرة
2. class Chp5_2{
3.  public static void main (String arg[]){
4.          int b=10;
5.    int a=b >= 15 ? 20 : 40;
6.    //if(b>=15)a=20; else a=40;
7.  System.out.println(a);
8. }
9. }
```

5.2.2 العبارات الشرطية المتداخلة (if/else)

في بعض الأحيان نريد من الحاسب انجاز عمل ما إذا كان الشرط صحيحا والقيام بعمل آخر إذا كان هذا الشرط غير صحيح ،فسنستعمل الجملة else مع الجملة if

```
if (hour < 12)
        System.out.println("Good morning.\n");
else if (hour < 17)
        System.out.println("Good afternoon.\n");
else
        System.out.println("Good evening.\n");
```

يوضح الشكل 2-5 مخطط النشاط للبنية if/else حيث نلاحظ أن الرموز المستخدمة ضمنة هي فقط المستطيلات (من أجل الأفعال) والمعينات (من أجل القرارات) بالإضافة إلى الدوائر الصغيرة والأسهم.



الشكل 2-5 المخطط الصندوقي لبنية if/else

وهذا المثال لذلك

```
1.  //برنامج يطبع تقدير الطالب بواسطة جمل القرار المتداخلة
2.  import javax.swing.*;
3.   public class Chp5_3{
4.  public static void main(String args[]){
5.   String s;
6.
7.   int grade;
8.
9.   s=JOptionPane.showInputDialog("Enter a number:");
10.
11.   grade=Integer.parseInt(s);
12.
13.        if ( grade >= 90)
14.                  System.out.println("A");
15.        else if ( grade >= 80)
16.                  System.out.println("B");
17.        else if ( grade >= 70)
18.                  System.out.println("C");
19.        else if ( grade >= 60)
20.                  System.out.println("D");
21.        else
22.                  System.out.println("F");
23. System.exit(1); 23.                    }
24. }
```

شرح المثال

يقوم البرنامج بطباعة القيمة "A" إذا كانت علامة الفحص أكبر أو تساوي 90 أو القيمة "B" إذا كانت هذه العلامة محصورة بين القيمتين 89,80 أو القيمة "C" إذا كانت هذه العلامة محصورة بين القيمتين 79,70 أو القيمة "D" إذا كانت هذه العلامة محصورة بين القيمتين 69,60 أو القيمة F إذا كانت هذه العلامة تساوي أي قيمة أصغر تماماً من 60.


يمكن أن يجري تنفيذ البنية if/else المتداخلة بشكل أسرع من التنفيذ المتتالي لعدة بني if وحيدة الأختيار ، لأنة يمكن الخروج مبكراً بعد تحقيق أحد الشروط.


كتابة عمليات المقارنة (==، =! ، =< ,=>) وبينهما فراغ (= =) يعتبر من الأخطاء الشائعة .


كتابة أحد عمليات المقارنة معكوسة (=> , =< , =!) يعتبر من الأخطاء الشائعة .


عند نسيان أحد الأقواس فأنة ينتج خطأ قواعدي يتم أظهار المحرر ، أم في حالة نسيان قوسين لا يتم كشفة المحرر ، ويكون خطاء منطقي .

5.2.3 بنية الاختيار المتعددة (Switch Statement)

تمتلك لغة Java نوعاً أخراً من بني الاختيار ، وهو نوع مفيد جداً عندما يكون ناتج الفحص قيمة عددية ، أو عندما يكون لنتيجة الفحص احتمالات متعددة . فمثلاً : عندما نريد أن نفحص حرفاً فيما إذا كان حرفاً صوتياً وذلك باستخدام عبارة if سيكون الفحص طويلاً كالتالي:

```
If (a=='a'
||a=='s'
||a=='q'
||a=='e'
||a=='t'
||a=='r'
||a=='y') .....
```

لاحظ كم كان ذلك شاقاً وطويلاً ، فقد جعلت لغة Java الأمر أكثر سهولة وأيسر باستخدام جملة (switch).

فتتألف جملة (switch) من سلسلة من الأجزاء التي تبدأ بكلمة case ومن جزء اختياري default . يستخدم البرنامج التالي تحديد أسم الشهر.

يقوم المستخدم بإدخال رقم الشهر فتقوم جملة التحكم بإيجاد اسم الشهر.

```
1. //برنامج يوضح العملية الشرطية المتعددة
2. class Chp5_4{
3.  public static void main (String arg[]){
4.
5.        int month=10;
6.
7.        switch (month) {
8.
```

```
9.          case (1):
10.         System.out.print("January");
11.         break;
12.
13.         case (2):
14.         System.out.print("February");
15.         break;
16.
17.         case (3):
18.         System.out.print("March");
19.         break;
20.
21.         case (4):
22.         System.out.print("April");
23.         break;
24.
25.         case (5):
26.         System.out.print("May");
27.         break;
28.
29.         case (6):
30.         System.out.print("June");
31.         break;
32.
33.         case (7):
34.         System.out.print("July");
35.         break;
36.
37.         case (8):
38.         System.out.print("August");
39.         break;
40.
41.         case (9):
42.         System.out.print("September");
43.         break;
44.
45.         case (10):
46.         System.out.print("October");
47.         break;
48.
49.         case (11):
50.         System.out.print("November");
51.         break;
52.
53.         case (12):
54.         System.out.print("December");
55.         }
56. }
57. }
```

تراقب العبارة switch القواعد التالية :

- يجب أن يولد التعبير switch-expression قيمة من النمط int ، char ، byte ، short ، ويجب أن يتم دائماً تضمينها في أقواس.
- الكلمة ألمفتاحيه break هي اختيارية لكن يجب استخدام ها عند نهاية كل حالة case وذلك لإنها بقية العبارة switch . إذا لم تكن العبارة break موجودة عندها يتم تنفيذ عبارة case التالية .
- الحالة default هي اختياريه ، ومن الممكن استخدام ها لإنجاز أعمال أخرى عندما لا تتوافق أية عبارة case مع التعبير switch-expression.
- يتم تنفيذ عبارة case وفق ترتيب تسلسلي لكن ترتيب الحالات ليس مهماً (بما فيها الحالة default). على أي حال للحصول على نموذج برمجي جيد يفضل أتباع التسلسل المنطقي للحالات ووضع الحالة default في النهاية .

لتجنب الأخطاء البرمجية ولتحسين إمكانية تصحيح الكود فإنه من الجيد وضع تعليق في العبارة case إذا تم إلغاء العبرة break لهدف معين .

يسبب نسيان كتابة التعليمة break عندما يكون هناك حاجة لوجودها ضمن أقسام البنية switch ، خطاء منطقياً.

يمكن أن يسبب حذف الفراغ ما بين الكلمة case والقيمة المفحوصة خطاء منطقياً.على سبيل المثال ، تسبب الكتابة case3 بدلاً من كتابة case 3: خلق بطاقة عنوان label غير مستخدمة .

الحرص على كتابة القسم default ضمن البنية switch لأنة بدونه يتم تجاهل إمكانيات الفحص الأخرى التي لا تقوم أقسام البينية بفحصها. ويجب الآنتباه إلى أن هناك حالات لا يوجد نعها أي حالة افتراضية default تحتاج إلى معالجة .

إذا تم ذكر default في الجزء الأخير من الجملة switch ، عندها لا يوجد أي حاجة لكتابة التعليمة break . لكن يفضل بعض المبرمجين كتابتها على الرغم من ذلك من أجل المحافظة على الوضوح والتناظر بين الأقسام المؤلفة للبنية.

يسبب استخدام نفس القيمة لدى عدة حالات case ضمن بنية switch . يسبب خطاء قواعدياً.

إليكم هذا البرنامج بعنوان كم الساعة وهذا البرنامج يتضمن كل ما سبق

```
1. import java.util.*;
2. class Chp5_5 {
3. public static void main(String[] arguments) {
4. // اجلب الوقت الحالي والتاريخ
5. GregorianCalendar now = new GregorianCalendar();
6. int hour = now.get(Calendar.HOUR_OF_DAY);
7. int minute = now.get(Calendar.MINUTE);
8. int month = now.get(Calendar.MONTH) + 1;
9. int day = now.get(Calendar.DAY_OF_MONTH);
10. int year = now.get(Calendar.YEAR);
11.
12. // إعرض التحية
13. if (hour < 12)
14. System.out.println("Good morning.\n");
15. else if (hour < 17)
16. System.out.println("Good afternoon.\n");
17. else
18. System.out.println("Good evening.\n");
19.
20. // ابدأ الرسالة بعرض الدقائق
21. System.out.print("It's");
22. if (minute != 0) {
23. System.out.print(" " + minute + " ");
24. System.out.print( (minute != 1) ? "minutes" : "minute");
25. System.out.print(" past");
26. }
27.
28. // اعرض الساعة
29. System.out.print(" ");
30. System.out.print( (hour > 12) ? (hour - 12) : hour );
31. System.out.print(" o'clock on ");
32.
33. // عرض اسم الشهر
34. switch (month) {
35. case (1):
36. System.out.print("January");
37. break;
38. case (2):
39. System.out.print("February");
40. break;
41. case (3):
42. System.out.print("March");
```

```
43. break;
44. case (4):
45. System.out.print("April");
46. break;
47. case (5):
48. System.out.print("May");
49. break;
50. case (6):
51. System.out.print("June");
52. break;
53. case (7):
54. System.out.print("July");
55. break;
56. case (8):
57. System.out.print("August");
58. break;
59. case (9):
60. System.out.print("September");
61. break;
62. case (10):
63. System.out.print("October");
64. break;
65. case (11):
66. System.out.print("November");
67. break;
68. case (12):
69. System.out.print("December");
70. }
71.
72. // عرض التاريخ والسنة
73. System.out.println(" " + day + ", " + year + ".");
74. }
75. }
```

- **جمل switch المتداخلة**

يمكن أن تكون العبارة switch عبارة بسيطة أو مركبة أي عبارة عن switch داخلية أخرى فتمكنك لغة Java من بناء عبارات switch متداخلة فيما بينها. تبين الشفرة التالية كيفيية بناء switch متداخلة

```
switch(count) {
                case 1:
                        switch(target) { // nested switch
                                        case 0:
                                                System.out.println("target is zero");
                                                break;
                                        case 1: // no conflicts with outer switch
                                                System.out.println("target is one");
```

```
                    break;
                }
            break;
        case 2: // ...
```

5.3 التكرار(Loop)

قد يصادفك حينما تكتب برنامج في Java احتياجك إلى إعادة جزء أو عدة أجزاء من البرنامج مرات عديدة ،لذلك وجد التكرار وهو يقوم بإعادة أجزاء من البرنامج عدة مرات ، بعداد أو حتى يتحقق شرط معين تكون قد وضعته للبرنامج.

فيا ترى من الذي سيقوم بهذا التكرار ؟
هنالك ثلاثة أنواع من حلقات التكرار تقدم لك لغه Java هي:

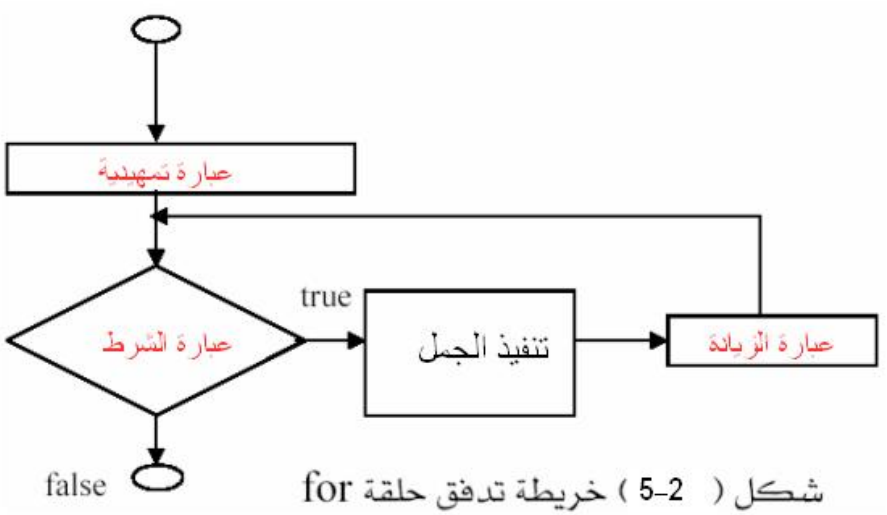
1- تكرار.for
2- تكرار.while
3- تكرار.do …while

وسوف نقوم نحن بتعليمك الأنواع الثلاثة وعليك أنت باختيار التكرار المناسب لبرنامجك.

5.3.1 تكرار (For)

وهو يسمح بتكرار جزء معين من البرنامج عدة مرات، حيث نعلم مسبقاً عدد المرات التي نريد فيها تكرار العمل. والقاعدة الأساسية لهذا التكرار هي:

```
for( عبارة تمهيدية؛عبارة الشرط؛ عبارة الزيادة)
{
الأوامر;
}
```



شكل (5-2) خريطة تدفق حلقة for

ثم دعنا نأخذ المثال الآتي وذلك حتى تكون عملية الفهم أسرع.

```
for(x=0;x<100;x++)
{
System.out.print ("This is my best site to learn Cn");
System.out.print ("Tell all your friends about this siten");
}
```

دعنا نتعرف معا بالتفصيل على هذه القاعدة:

1- في السطر الأول نكتب كلمة for ثم نفتح قوس ونكتب في أول خانة عبارة التمهيد وهي x=0 تنفذ مرة واحدة عند بداية التكرار كما في المثال السابق .

2 - في الخانة الأخرى نكتب الشرط الذي سوف يمثل استمرار التكرار كما في المثال السابق x < 100 بحيث يختبر هذا الشرط في كل مرة يقوم بالتكرار ، فإذا كان الشرط صحيحا فإن التكرار سيستمر أما إذا كان خاطئا فسيتوقف التكرار.

3 - أما في الخانة الأخيرة نكتب عبارة الزيادة وهي تنفذ بعد كل تكرار كما في المثال السابق x++ وبذلك سوف تقوم بزيادة قيمة المتغير وسوف يقوم بزيادة قيمة المتغير بمقدار واحد.

ثم يأتي القوس { ومن بعده نكتب الأوامر التي نريد تنفيذها كما في المثال السابق، وهي سوف تتكرر مائة مرة ، طبعا نستطيع وضع أكثر من عبارة ، وأخيرا كل ما علينا هو أن نغلقها بالقوس }

إذا كنا نريد تكرار الأمر مرة واحدة فقط ، فنحن بإمكاننا التخلي عن الأقواس { و } ، أما في حالة كتابة أكثر من أمر فيجب وضع الأقواس.


أن استخدام القيمة النهائية في شرط التكرار في بنية التكرار مع استخدام عملية المقارنة =< يساعدان على تجنب أخطاء **الإنهاء** قبل بمرة . فعلى سبيل المثال:

```
For(i=1;i<=10;i++);
```

الحالة المستخدمة لطباعة القيم من 1-10 أم إذا استبدلت i<10 فأن **الإنهاء** يكون قبل بمرة .


استخدام متغير داخل الحلقة دون التعريف على هذا المتغير.


استخدام متحول الحلقة خارج بلوك التكرار، كهذا المثال لن ينفذ إطلاقاً.

```
1. //برنامج يوضح الخطاء الشائع في الحلقة
2. class Chp5_6{
3.  public static void main (String arg[]){
4.
5. for(int i=0;i<10;i++)
6.  System.out.println(i);
7.  //خطاء
8.  System.out.println(i);
9. }
10. }
```


استخدام الفواصل بدلاً من الفواصل المنقوطة في الجزء الرأسي للبنية for .


يسبب وضع الفاصلة المنقوطة على يمين القوس للجزء الرأسي للبنية for ، أن يصبح جسم هذه البنية مؤلفاً من التعليمية الفارغة ويعتبر ذلك عادة خطاء منطقي. كهذا الكود

```
1. //برنامج يوضح االخطاء الشائع في الحلقة
2. class Chp5_7{
3.  public static void main (String arg[]){
4.          int i;
5. for(i=0;i<10;i++);
6.  System.out.println(i);
7.  }
8. }
```


عند كتابة جملة الحلقة for بدون أي شروط ، أو عداد فإنها تتولد حلقة لا نهائية وقد يغلق البرنامج. كهذا المثال لن يتوقف أبداً.

```
1. //برنامج ي لا نهائي أشبة بالفيروس
2. class Chp5_8{
3.  public static void main (String arg[]){
4.          int k=1;
5. for(;;)
6.          System.out.println(k++);
7.                                          }
8. }
```

وهكذا نكون قد انتهينا من التكرار for بكل بساطة . والآن هيا بنا ننطلق لتعلم الأنواع الأخرى من التكرار.


لابد أن تطبق كل برنامج مكتبو لأنك لن تتعلم البرنامج ولن تستوعبه حتى ترى النتيجة بعينيك.


يمكن تهيئة الدوارة بمحارف من نوع (char) كهذا المثال:

```
1. // استخدام  محارف الحروف في الدورة
2. class Chp5_9{
3.  public static void main (String arg[]){
4.          char i;
5. for(i='a';i<='z';i++)
6.  System.out.println(i);
7.  }
8. }
```


تمكنك لغة الجافا من استخدام عدة متحولات داخل الجملة for فتتمكن من تصغير حجم البرنامج والتقليل من كثرة المتحولات كهذه الشفرة

```
int a, b;
b = 4;
for(a=1; a<b; a++) {
            System.out.println("a = " + a);
            System.out.println("b = " + b);
            b--;
                        }
```

يمكن اختصارها بهذه الشفرة

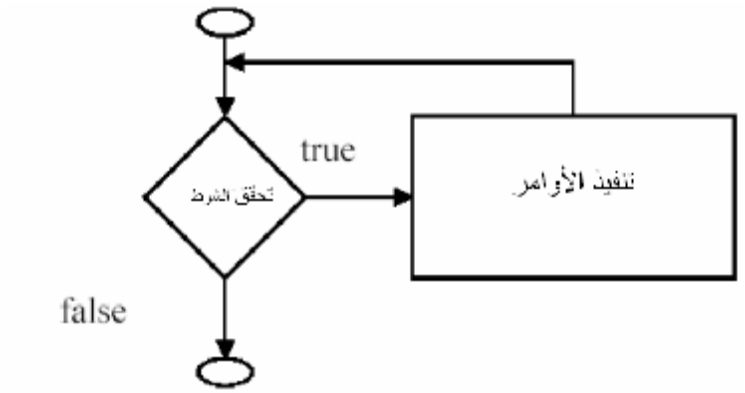
```
int a, b;
for(a=1, b=4; a<b; a++, b--) {
            System.out.println("a = " + a);
            System.out.println("b = " + b);
                                    }
```

5.3.2 بنية التكرار الحذر (while)

وهذا التكرار يقوم بتكرار غير محدد من المرات حتى يتحقق شرط معين كما في الشكل 3-5، والقاعدة الأساسية لهذا التكرار هي:

while(الشرط)

{
.....;
.....الأوامر;
}



شكل (53) خريطة تدفق حلقة while

أساسيات حلقة التكرار ذات العداد:

1- تعريف لمتحول التحكم بالحلقة .
2- تحديد القيمة الابتدائية لمتحول التحكم بالحلقة.
3- تحديد أسلوب الزيادة أو النقصان الذي يتم من خلاله تغير قيمة متحول التحكم بالحلقة في كل مرة نمر فيها.
4- تحديد فيما الشرط الذي من خلاله نقوم بفحص النتيجة النهائية لمتحول التحكم بالحلقة (حتى نحدد فيما إذا كان من الممكن معودة الحلقة).

أن عدم وضع شرط التوقف في جسم بنية while ينتج عن ذلك حلقة لا نهائية (infinite loop).

دعنا نأخذ المثال التالي:

```
while(num<99)
{
```

```
System.out.println("This is a correct numbern");
num++;
}
```

دعنا نتكلم بالتفصيل عن هذا التكرار:

1- عند كتابة الشرط فإن البرنامج سوف يقوم بتكرار الأوامر ما دام الشرط صحيحا ، أما إذا كان الشرط خاطئا فإن التكرار سوف يتوقف ، ففي المثال السابق نستطيع القول بأنه ما دام الرقم اقل من 99 فإن العبارة سوف تطبع.

2- بعد طباعة الجملة سوف ينتقل إلى العبارة الأخرى وهي num++ وبذلك سوف يزيد الرقم.

3- وهكذا سوف تتم تكرار العبارة إلى إن تصل إلى الرقم 99 ومن بعدها سوف يتوقف التكرار.

- **الفرق بين (for ،while)**

يكمن الفرق بين التعليمتين السابقتين أن جملة for تحدد نقطة البداية والنهاية أي تكون على علم متى ستنتهي الدوارة. أم جملة while تأخذ فقط شرط توقف وتستعمل في البرامج كالقاسم المشترك بين عددين.

5.3.3 حلقات التكرار do…while

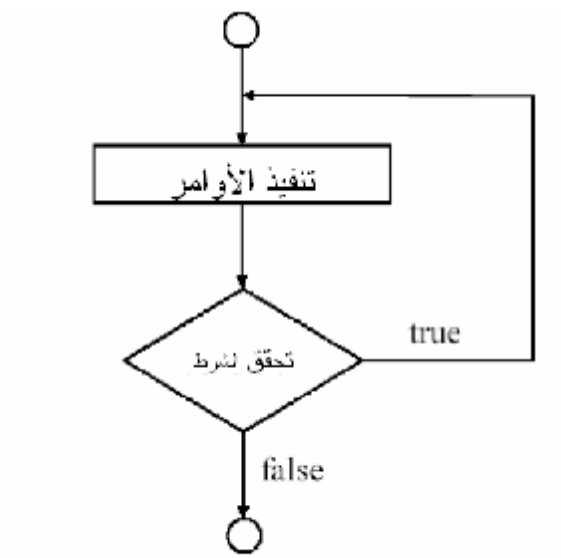
لكي ندرك الفرق بين الحلقتين إليك هذا المثال :
لنفرض مثلا أن الآنسة أروى تريد الذهاب إلى الكوافير فان أمامها طريقتان :
الأولى أن تذهب إلى الكوافير ثم تخبر والدتها .
الثانية أن تستأذن أولا من والدتها قبل أن تذهب إلى الكوافير .

الطريقة الأولى تشبه حلقة التكرار do...while بمعنى أنه سوف يتم تحقق الشرط لمرة واحدة قبل أن يختبر يعني أن الآنسة أروى سوف تذهب لمرة واحدة إلى الكوافير سواء وافقت والدتها أم لم توافق . ومن هنا يتضح لنا أنه في حلقة التكرار while…do يتم تنفيذ الشرط أولا قبل الـ تحقق من صحة الشرط موضع الاختبار فعندما يصل البرنامج خلال عمله للمرة الأولى إلى حلقة التكرار do فانه ينفذ الجمل الموجودة بين الجملة do والجملة while بشكل

تلقائي ، ثم يجري بعد ذلك التحقق من صحة شرط الجملة while فإذا كان صحيحا فأن البرنامج يعاود تكرار حلقة الـ تكرار مرة أخرى أما إذا كان الشرط غير صحيح فان حلقة التكرار سوف تتوقف.

- **القاعدة الأساسية لهذا التكرار هي :**

```
do{
……..;
……..;
الأوامر;
}while(الشرط)
```



شكل (5 4) خريطة تدفق حلقة do/while

دعنا نأخذ المثال التالي:

```
1. //(DO...while)برنامج يوضح عمل الجمله
2.  import javax.swing.JOptionPane ;
3.  public class Chp5_10 {
4.  public static void main ( String [ ] args )
5.  {
```

```
6.  String input ;
7.  int data ;
8.  int sum = 0 ;
9.  do
10.  {
11.  input = JOptionPane.showInputDialog ( " enter number " ) ;
12.  data =Integer.parseInt ( input ) ;
13.  sum += data ;
14.  } while ( data != 0 ) ;
15.  JOptionPane.showMessageDialog ( null , " the sum is " + sum ) ;
16.  System.exit ( 0 ) ;//(end) جملة الخروج من البرنامج
17.  }
18.  }
```

شرح المثال :

هذا البرنامج يقوم بقراءة أرقام من المستخدم عن طريق توجيه رسالة له عن طريق صندوق الحوار ثم إذا أدخل المستخدم رقم صفر يقوم البرنامج بطباعة حاصل جمع الأرقام المدخلة في صندوق رسالة .

لاحظ استخدام جملة do/while في السطور من 9 إلى 14.

داخل الحلقة تم توجيه رسالة للمستخدم ليدخل رقماً أو يدخل صفراً عند .

السطر رقم 14

} while (data!=0);

هو عبارة عن إغلاق القوس { ثم يأتي بعد ذلك اختبار شرط استمرار الحلقة وهو هل قيمة الرقم المدخل لا تساوي صفراً فإذا كان الجواب بنعم يتم إعادة تكرار الحلقة وإن كان الجواب بلا فتتوقف الحلقة فوراً وننتقل إلى السطر التالي.

السطر رقم 15

JOptionPane.showMessageDialog (null , “ the sum is “ + sum) ;

هو جملة طباعة في صندوق رسالة يتم فيها طباعة المتغير sum وهي حاصل جمع الأعداد المدخلة من قبل المستخدم<

السطر رقم 16

هو عبارة عن جملة إنها البرنامج ليتم الخروج من البرنامج دون تعليق شاشة الخرج كما في البرامج السابقة التي تحتوي على صناديق حوار الإدخال .

5.4 تجاوز قوانين الحلقات Getting Around The Looping Rules

تملك لغة Java عدة روتينان مسبقة التعريف تمكن من القيام بأعمال خاصة أثناء تنفيذ الحلقات . وهذه الإجراءات هي break, continue ووظيفة هذه الإجراءات تغير انسياب التحكم (flow of control) ضمن البرنامج .

5.4.1 جملة (break)

تعمل هذه التعليمية في داخل الحلقة لإيقاف الدوارة من الدوران والخروج منها ومتابعة تنفيذ أسطر البرنامج . حيث تستخدم غالباً تكون الحلقة بدون شرط أي لا نهائية مثل

While(true){…..}

واليكم هذا المثال يقوم بطباعة الأعداد الأولية :

```
1. // برنامج يطبع الأعداد الأولية
2. class Chp5_11{
3.  public static void main (String arg[]){
4.        int i,j,k=1;
5.        for(i=1;i<100;i++){
6.                for(j=2;j<i;j++)
7.                        if(i%j==0){
8.                                                k=0;
9.                                                break;
10.                                          }
11.       if(k==1)System.out.println(i);
12.       k=1;
13.                                          }
14.                                        }
15. }
```

نلاحظ أن هذه التعليمية تعمل على إيقاف الدوارة الداخلية للبرنامج ، بمجرد تحقق الشرط.

5.4.2 جملة (goto)

إن الكلمة goto هي كلمة مفتاحية محجوزة للغة C++ لكنها غير مستخدمة حالياً في Java مع العلم أن البرامج المكتوبة بدون استخدام جملة goto هي برامج تحدي العصرالتي تواجه المبرمجين في أسلوب كتابتهم لبرامجهم.

تقوم هذه العبارة بنقل التحكم إلى أي عبارة (ضمن البرنامج) معلمة باستخدام اللافتة وتنفذها بدون تميز ما يسمى بالقفز الغير مشروط أو بعبارة التفرع الغير مهيكل.

ويمكن للمبرمج التحايل في برنامجة وتطبيق عمل الجملة goto باستخدام الجملة break وتطبيقها مع الحلقات كهذا المثال

```
1. // Using break as a civilized form of goto.
2. class Chp5_12 {
3.         public static void main(String args[]) {
4.                 boolean t = true;
5.                 first: {
6.                         second: {
7.                                 third: {
8.                                         System.out.println("Before the break.");
9.                                         if(t) break second; // break out of second block
10.                                        System.out.println("This won't execute");
11.                                                 }
12.                        System.out.println("This won't execute");
13.                                         }
14.                System.out.println("This is after second block.");
15.                                 }
16.
        }
17. }
```

فيكون ناتج التنفيذ:

```
Before the break.
This is after second block.
```

شرح المثال

في السطر 8 تم تنفيذ جملة الطباعة ، وفي السطر 9 جملة الشرط if تتحقق وتعمل على إيقاف اللافتة second حيث تم تجاهل محتوى اللافتة third في السطر 10 وبقية محتوى اللافتة second كالسطر 12 وأنتقل مباشرة إلى السطر 14 وتابع تنفيذ البرنامج.

استخدام الجملة break لإيقاف لافتة خارج نطاق block كهذا المثال ينتج عنه رسالة خطاء في وقت الترجمة

```
1. // This program contains an error.
2. class Chp5_13 {
3.         public static void main(String args[]) {
4.                 one: for(int i=0; i<3; i++) {
5.                         System.out.print("Pass " + i + ": ");
6.                                                 }
7.                 for(int j=0; j<100; j++) {
8.                         if(j == 10) break one; // WRONG undefined label: one
9.                                 System.out.print(j + " ");
10.                                                 }
11.                   }
12. }
```

5.4.3 جملة (continue)

وهو أقل هذه الإجراءات تمزيقاً للحلقة ، إذ يسبب هذان الإجراء بداية تنفيذ التكرار التالي للحلقة مباشرة . وتجاوز ما تبقى من الإجراءات في جسم التكرار والمتابعة مع المرور التالي في الحلقة .

وهذا المثال يوضح ذلك:

```
1. //برنامج يوضع عمل متتاليات الهروب
2. public class Chp5_14 {
3.   public static void main(String[] args) {
4.     int i = 0;
5.     outer: // Can't have statements here:
6.     for(; true ;) { // infinite loop
7.       inner: // Can't have statements here
8.       for(; i < 10; i++) {
9.         System.out.println("i = " + i);
10.          if(i == 2) {
11.            System.out.println("continue");
12.            continue;
13.                                  }
14.          if(i == 3) {
15.            System.out.println("break");
16.            i++; // Otherwise i never
17.            //:زيادة عداد الدوارة بمقدار واحد أي تخطي خطوة من تنفيذ الأوامر
18.                 // gets incremented.
19.            break;
20.                                  }
21.          if(i == 7) {
22.            System.out.println("continue outer");
23.            i++; // Otherwise i never
24.                 // gets incremented.
25.            continue outer;
26.                                  }
27.          if(i == 8) {
28.            System.out.println("break outer");
29.            break outer;// توقف نقطةBlockمن التنفيذ
30.                                  }
31.          for(int k = 0; k < 5; k++) {
32.            if(k == 3) {
33.              System.out.println("continue inner");
34.              continue inner;// تخطي نقطةBlockمن التنفيذ
35.                                        }
36.                                                                  }
37.                                                        }
38.                                        }
39.   }
40. }
```

فيكون خرج البرنامج

```
i = 0
continue inner
i = 1
continue inner
i = 2
continue
i = 3
break
i = 4
continue inner
i = 5
continue inner
i = 6
continue inner
i = 7
continue outer
i = 8
break outer
```

تمتلك بعض لغات البرمجة العبارة goto . فلغة Java لا توجد بها هذه العبارة فيمكن استبدالها باللافتة كما في برنامجنا السابق.

يؤدي الاستخدام الصحيح للتعليمتين continue , break إلى تسريع تنفيذ البرنامج أكثر من تلك التقنيات المكافئة لهما حسب أسس ومبادئ البرمجة المهيكلة .

5.4.4 جملة (return)

تستطيع إيقاف تنفيذ البرنامج بهذه التعليمة return فهي تعمل على إيقاف الدالة الرئيسية main وهذا المثال يبين ذلك:

```
1. // Demonstrate return.
2. class Chp5_15 {
3.         public static void main(String args[]) {
4.                 int i;
5.                 for(i=1;i<5;i++){
6.                         System.out.println(i+"- Before the return.");
7.                         if(i==3) return; // return to caller
8.                                 }
```

```
9.                  System.out.println("This won't execute.");
10.                                                     }
11. }
```

حيث يكون ناتج تنفيذ البرنامج:

5.5 كيفية اختيار الحلقة المناسبة

كما تعلمنا من السابق ولاحظنا أن لغة (Java) تمكننا من استخدام ثلاثة أنواع من الحلقات (for , while , do...while) وكليهم متكافئات من الناحية التعبيرية بحيث يمكنك من كتابة أي منهم . فعلى سبيل المثال يمكن كتابة برنامج يطبع الأعداد من 1-10 بأي حلقة تريدها

```
For(i=1;i<11;i++)
        System.out.printle(i);
```

ويمكن كتابته

```
Int i=1
While(i<11)
        System.out.printle(i++);
```

ويمكن كتابته

```
int i=1;
 do
   System.out.println(i++);
 while(i<=10);
```

من الأفضل استخدام الحلقة التي تجدها أكثر سهوله ، وخلاصة القول استخدم حلقة for إذا كان عدد التكرار معروف كطباعة الأعداد من 1-100 ، و استخدام الحلقة while إذا كان عدد التكرار مجهول كقراءة عدد حتى نصل إلى القيمة 0 كدخل ، و استخدام الحلقة do..while إذا كان لابد من تنفيذ جسم الحلقة مرة واحدة على الأقل .

تمارين الفصل:

1. أكتب الجمل التي تقوم بالمهام التالية:
   - جمع الأعداد الفردية من 1 إلى 99 باستخدام الحلقة for.
   - طباعة الأعداد الصحيحة من 1 إلى 22 باستخدام الحلقة while.
   - طباعة مجموع الأعداد الفردية السالبة من بين مجموعة من الأعداد.
2. أكتب برنامج يقوم بضرب عددين بدون استخدام علامة الضرب (*) ؟
3. أكتب برنامج يقوم بقسمة عددين بدون استخدام علامة القسمة (/) ؟
4. أكتب برنامج لإيجاد المتواليات التالية

- 2 4 6 9 12 17 20 25……N
- 1 3 7 15 31 63 127 ……N
- 1 5 9 17 3 36 4 20 81……N
- 1 5 9 13 17 21 25 ……N
- 1 3 6 18 24 30 36……N
- 1 2 3 5 7 11 13 17……N
- 1 2 4 8 16 32 64 128 ……N
- -1 2 -3 5 -11 13 -17……N
- 1 + N/1 – N/3 + N/ 5 –N/7 ….N

5. أكتب برنامج يطبع N من الأعداد بشرط ألا يطبع الأعداد الأولية ؟
6. أكتب برنامج لإيجاد القاسم المشترك والمضاعف المشترك N من الأعداد ؟
7. أكتب برنامج يرتب العدد المدخل ؟ // ß 1526 ß 1256
8. أكتب برنامج يعرف هل مراتب العدد متساوية أم لا ؟
9. أكتب برنامج يقوم بطرح عددين باستخدام الطريقة التقليدية ؟
10. أكتب برنامج يقوم بعكس عدد مدخل ؟ // ß 123 ß 321
11. أكتب برنامج يقوم بحذف الأعداد المتكررة من مرات العدد المدخل ؟ // ß 1324324 ß 1243
12. أكتب برنامج يعرف هل بداية الرقم تساوي نهايته ؟
13. أكتب برنامج يستطيع أن يحدد أصغر قيمة من بين مجموعة أرقام ؟
14. تقوم شركة بدفع أجور موظفيها على شكل رواتب أسبوعية، وأجور عمالها حسب عدد ساعات عملهم (تعتبر أول 40 ساعة عمل أسبوعية عبارة عن ساعات عمل عادية ذات أجر عادي أم ما يزيد عن ذلك فكل ساعة تعتبر معادلة لـ 1.5 ساعة عمل عادية من ؟
15. أكتب البرامج التي تولد الأشكال التالية

******** ******* ****** **** *** ** *	* ** *** **** *** ** *	* *** ***** ******* ***** *** *	* ** *** **** ***** ****** *******	* *** ***** ******* ***** *** *
********** ********* ******** ******* ***** *** *	********** ***** ***** **** ***** ** ** * * **********	1 212 32123 4321234 543212345 65432123456	1111111111 1222222221 1233333321 1234444321 1234554321 1234554321 1234444321 1233333321 1111111111	1111111111111 12222222221 123333321 1234321 12321 121 1
1 121 12321 1234321 123454321	1234567 23456 345 4	1 121 12221 1234321 122221 121 1	1 121 12321 1234321 12321 121 1	1234567 1230567 1200067 1000007

16. أوجد ناتج تنفيذ البرامج التالية:

A)

```
1 public class Mystery2 {
2
3 public static void main ( string args [ ] )
4 {
5        int count = 1;
6
7        while ( count <= 10 ) {
8                System.out.println (
9                count % 2 = = 1 ? "****" : "++++++++" );
10               ++count;
11                                }
12 }
13 }
```

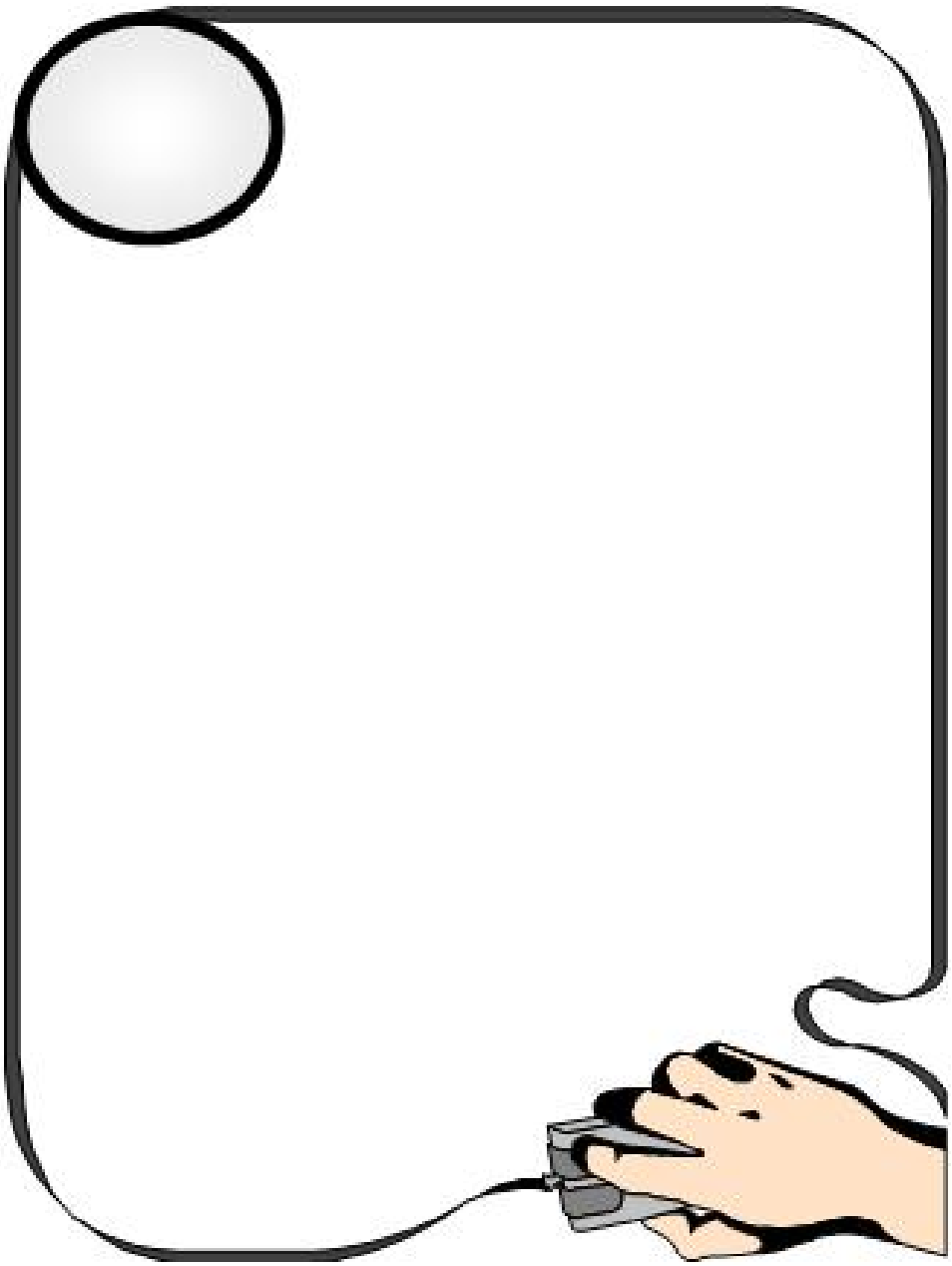
B)

```
1 public class Mystery3 {
2
3 public static void main ( String args [ ] )
```

```
4 {
5         int row = 10, column;
6
7       while ( row >= 1 ) {
8                         column = 1;
9
10                        while ( column <= 10 ) {
11                                System.out.print (row % 2 = = 1 ? "<" : ">" );
12                                ++column;
13                                                  }
14
15      --row;
16      System.out.println( );
17                                }
18 }
19 }
```

C)

```
1 class BreakLoop4 {
2 public static void main(String args[]) {
3 outer: for(int i=0; i<3; i++) {
4       System.out.print("Pass " + i + ": ");
5       for(int j=0; j<100; j++) {
6               if(j == 10) break outer; // exit both loops
7       System.out.print(j + " ");
8                                   }
9       System.out.println("This will not print");
10                                  }
11 System.out.println("Loops complete.");
12                                                              }
13 }
```



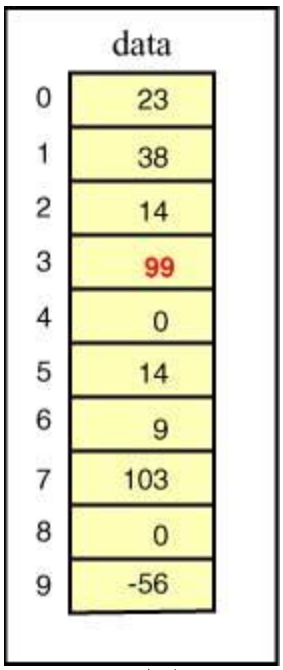
6.1 مقدمة INTRODUCTION

قد تحتاج في بعض الأحيان تخزين عدد كبير من الأرقام في الذاكرة خلال تنفيذ البرنامج . فلا يمكن أن تقوم بتعريف متغيرات من أجل كل رقم ، فلغة Java وضعت حداً لهذه المشكلة بما يدعى المصفوفات Array. فيعني هذا الفصل في عرض مقدمة حول أحد بني المعطيات الهامة وهي المصفوفات Array .

فما المقصود بالمصفوفات ؟

قبل أن نتعرف على تعريف المصفوفات من الناحية البرمجية ، وأقصد هنا تعريفها ضمن سياق البرمجة ، حيث نعرفها مسبقا من دراسة الرياضيات . أود أن أشير إلى أننا في الفصول السابقة قد تعرفنا ع لى الطريقة الأساسية لتخزين المعلومات في برامجنا السابقة وهي للتذكرة وضع هذه المعلومات في متغير ولكن هذه الطريقة تناسب البرامج البسيطة ذات البعد الواحد . ولكن إذا أردت أن تصمم برنامجا للقيام بأعمال الكنترول في مدرستك ، فإنك ستكون في حاجة إلى وسيلة معقدة للتعامل مع 1000 طالب في صفوف دراسية مختلفة. هذه الطريقة هي المصفوفات (Array).

إذاً التعريف المحدد للمصفوفات ضمن سياقنا هو ؟



الشكل 1-6

مجموعة من المتغيرات المتماثلة في النوع والمرتبطة معا أو بعبارة أخرى هي عبارة عن مجموعة من المتغيرات باسم واحد ، موضوعه داخل الذاكرة بشكل متتالي حيث أول موقع بداخل المصفوفة تكون قيمته 0 . كما في هذا الشكل 1-6.


من المهم أن نلاحظ الفرق بين العنصر الأول في المصفوفة ، وعبارة الموضع الأول في المصفوفة . فأن قيمة دليل المصفوفة تبدأ بالرقم 0 . فعبارة العنصر الأول من عناصر المصفوفة تدل على العنصر ذو الدليل 0 . فيجب الانتباه إلى هذه النقطة لأنها تسبب أخطاء في تحديد العناصر.

6.2 التصريح عن المصفوفات

لتتمكن من استخدام المصفوفة يجب عليك التصريح عن متحول كمرجع لهذه المصفوفة ، بالأضافه إلى تحديد النمط لهذه المصفوفة . وفيما يلي الصيغة القواعدية للتصريح عن متحول من نوع مصفوفة.

```
datatype[] namearray;
```

أو

```
datatype namearray[];
```

هذا التصريح لم يحجز مساحة على الذاكرة بل قيمته null.

بعد التصريح عن متحول المصفوفة تأتي بعدها حجز مقدار المواقع في الذاكرة لتخزن القيم بداخلها. في العبارة التالية:

```
namearray = new datatype[arraysiza];
```

فتقوم العبارة السابقة بعمليتين الأولى صنع مصفوفة باستخدام (namearray = new datatype[arraysiza];) ، والثانية إسناد مرجع المصفوفة المصنوعة حديثاً إلى المتحول namearray.

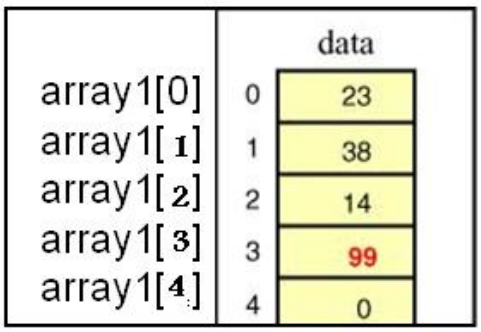

نم الممكن دمج الخطوات السابقة بعملية واحدة كما يلي:

datatype[] namearray=new datatype[arraysiza];

وهذا مثال للعبارة السابقة:

int [] array1=new int(5);

فهنا تم حجز 5 مواقع في الذاكرة مخصصة لتخزين قيم من نوع int ومسندة للمتحول array1. كما في الشكل 2-6.



شكل 2-6

فيتم تمثيل كل عنصر داخل البرنامج باستخدام الصيغة التالية :

array1[0];

يمكن تهيئة المصفوفة بقيم بدائية (initializes) كما يلي:

int [] array1={1,2,3,5};
array1=new int[4];

من الخطاء أعطاء قيم ابتدائية تزيد في عددها عن عدد عناصر المصفوفة . مما يسبب خطاء قواعدي .

6.3 أنواع المصفوفات The Array Types

- المصفوفات أحادية البعد(One-Dimensional Arrays).
- المصفوفات متعددة الأبعاد (Multidimensional Arrays).
- المصفوفات الغير منتظمة ().

6.3.1 المصفوفات أحادية البعد One-Dimensional Arrays

المصفوفة أحادية البعد هي المصفوفة التي لها بعد واحد فقط ، سطر واحد من الأرقام التي تتراوح بين الصفر ورقم العنصر الأعلى المحدد للمصفوفة . ونقصد بعناصر المصفوفة عدد البنود التي يمكن تخزينها في المصفوفة فكل بند في المصفوفة يسمى عنصرا .

وسنقوم بإنشاء مصفوفة من الأرقام الصحيحة تسمى الأعداد الأولية ، وتحتوي هذه المصفوفة على 300 عنصرا يمكننا استعمالها لتخزين 300 عدد أولي .

```
1. //برنامج بواسطة المصفوفات الأحادية يدخل 300 عنصر أولي بداخلها
2. public class Chp6_1 {
3.   public static void main(String[] args) {
4. int [] array1=new int[300];
5.         int i,f,j,k;
6.         k=i=f=j=0;
7.
8.         array1[i++]=1;
9.         while(i<300)
10.                              { k++;
11.                                       for(j=2;j<k;j++){
12.                                                if(k%j==0){
13.                                                                                    f=1;
14.                                                                                    break;
15.                                                                          }
16.                                                                 }
17.                                       if(f==0)array1[i++]=k;
18.                                       f=0;
19.                              }
20.        for(j=0;j<i;j++)
21.                  System.out.println(array1[j]);
22.
23.  }
24. }
```

يقوم البرنامج بالتصريح عن مصفوفة مؤلفة من 300 عدد صحيح . فيقوم من السطر 9-19 بتوليد سلسلة من الأعداد الأولية وإدخالها في المصفوفة كما في السطر رقم 17.
والسطر رقم 21 يقوم بطباعة عناصر المصفوفة .

إسناد قيم خارج نطاق المصفوفة يسبب خطاء في وقت التنفيذ مما يصدر المحرر رسالة :

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException
        at Chp4_1.main(Chp4_1.java:17)
```

تدعم لغة Java تعليمة (namearray.length) يتم بواسطتها معرفة طول حجم المصفوفة . كهذا المثال يقوم بإيجاد أصغر قيمة من داخل المصفوفة:

```
1. //برنامج يقوم بإيجاد أصغر قيمة من داخل المصفوفة الأحادية
2. class Chp6_2
3. {
4.
5.   public static void main ( String[] args )
6.   {
7.
8.     int[] array =  { -20, 19, 1, 5, -1, 27, 19, 5 } ;
9.     int   min;
10.
11.     // initialize the current minimum
12.     min = array[ 0 ];
13.
14.     // scan the array
15.     for ( int index=0; index < array.length; index++ )
16.     {
17.       if ( array[ index ] < min )
18.
19.         min = array[ index ] ;
20.
21.     }
22.
23.     System.out.println("The minimum of this array is: " + min );
24.   }
25. }
```

فيكون ناتج البرنامج كالتالي:

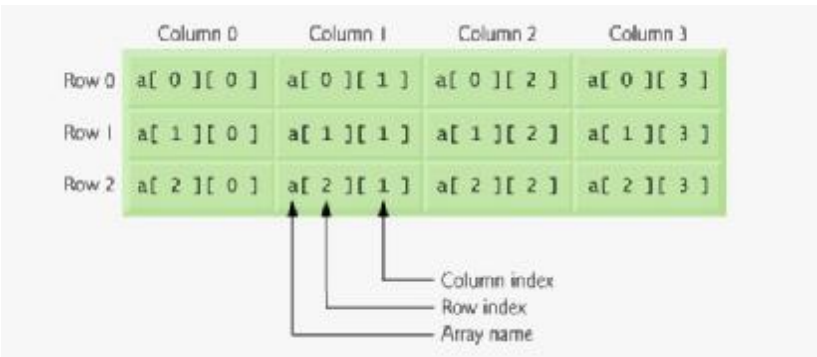
```
The minimum of this array is: -20
```

6.3.2 المصفوفات ثنائية البعد Tow-Dimensional Arrays

يمكن للمصفوفات في لغة Java أن تأخذ عدة أبعاد. ومن بين الاستخدامات الشائعة لهذا النوع من الشائعة لهذا النوع من المصفوفات نجد الجدول Tables التي هي عبارة عن مجموعة من القيم المرتبة ضمن مجموعة من الأسطر rows والأعمدة columns. وبالتالي من أجل الوصول إلى عنصر ما من ضمن مجموعة من عناصرها يجب أن تحدد دليلين هما : رقم

السطر ورقم العمود اللذان ينتمي إليهما العنصر . فنسمي المصفوفات أو الجداول التي تستخدم دليلين لتحديد أي عنصر من عناصرها بالمصفوفات ذات بعدين . يظهر في الشكل التالي مصفوفة ذات بعدين يحتوي على ثلاثة صفوف ، و أربعة أعمدة . لذلك نسمي هذه المصفوفة إنها مصفوفة 3*4 حيث نسمي بشك عام المصفوفات المؤلفة من m سطر و n عمود بالمصفوفات m*n.

يمكن تحديد كل عنصر من عناصر المصفوفة a في الشكل 3-6 بأن نكتب a[i][j] حيث a هو عبارة عن أسم المصفوفة و i,j هما الدليلان المحددان للعنصر المطلوب .



شكل 3-6

لاحظ أن أسماء العناصر الذي تنتمي إلى السطر الأول من أسطر المصفوفة a تأخذ القيمة 0 للدليل الأول (I) وأن أسماء العناصر في العمود الرابع من نفس المصفوفة تأخذ القيمة 3 للدليل الثاني .

من الأخطاء البرمجية الشائعة تحديد عنصر من عناصر المصفوفة ثنائية البعد a[i][j] بالتعبير التالي a[I,j] .

- **إعطاء قيم ابتدائية للمصفوفة ذات البعدين**

```
1. //برنامج باستخدام  مصفوفة ذات بعدين
2. class Chp6_3
3. {
4.   public static void main( String[] arg )
5.   {
6.     // declare and construct a 2D array
```

```
7.    int[][] gradeTable =
8.       {  {99, 42, 74, 83, 100},
9.          {90, 91, 72, 88, 95},
10.           {88, 61, 74, 89, 96},
11.           {61, 89, 82, 98, 93},
12.           {93, 73, 75, 78, 99},
13.           {50, 65, 92, 87, 94},
14.           {43, 98, 78, 56, 99}  };
15.
16.    System.out.println("grade 0,0: " + gradeTable[0][0]);
17.    System.out.println("grade 2,4: " + gradeTable[2][4]);
18.
19.    gradeTable[5][3] = 99 ;
20.
21.    int sum = gradeTable[0][1] + gradeTable[0][2] ;
22.    System.out.println( "sum: " + sum );
23.  }
24. }
```

فيكون ناتج البرنامج كما يلي

```
grade 0,0: 99
grade 2,4: 96
sum: 116
```

يقوم البرنامج السابق بإعطاء قيم ابتدائية للمصفوفة، و بطباعة بعض عناصر المصفوفة . وفي السطر 19 تم تغير القيمة ذات الدليل [3][5] السطر الخامس في العمود الثالث بقيمة 99.

- **الحصول على طول المصفوفة ذات البعدين**

تتألف المصفوفة ثنائية البعد من مصفوفة من العناصر وكل عنصر هو مصفوفة أحادية البعد، فيمكن الحصول على أبعاد المصفوفة باستخدام الصيغة التالية:

arryaname.length // لطول الصفوف
arrayname[0].length// لطول الأعمدة

6.3.3 المصفوفات ذات البعد الثلاثي Three-Dimensional Arrays

```
1. // Demonstrate a three-dimensional array.
2. class Chp6_4 {
3.         public static void main(String args[]) {
4.                 int threeD[][][] = new int[3][4][5];
5.                 int i, j, k;
6.                 for(i=0; i<3; i++)
7.                         for(j=0; j<4; j++)
8.                                 for(k=0; k<5; k++)
```

```
9.                                                  threeD[i][j][k] = i * j * k;
10.             for(i=0; i<3; i++) {
11.                     for(j=0; j<4; j++) {
12.                             for(k=0; k<5; k++)
13.                                     System.out.print(threeD[i][j][k] + " ");
14.                             System.out.println();
15.                                                             }
16.                     System.out.println();
17.                                                     }
18.                                             }
19. }
```

فيكون ناتج التنفيذ كما يلي

```
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0

0 0 0 0 0
0 1 2 3 4
0 2 4 6 8
0 3 6 9 12

0 0 0 0 0
0 2 4 6 8
0 4 8 12 16
0 6 12 18 24
```

6.3.4 المصفوفات غير المنتظمة

يمكن لصفوف المصفوفة امتلاك أطوال مختلفة ، كهذا التعبير

```
int[][] uneven =
        { { 1, 9, 4 },
          { 0, 2},
    { 0, 1, 2, 3, 4 } };
```

يمثل بداخل الذاكرة كما في الشكل 5-6.

Row	Col				
	0	1	2	3	4
0		1	9	4	
1		0	2		

2 0 1 2 3 4

uneven

شكل 5-6

وهذا المثال يصنع مصفوفة غير منتظمة :

```
1. //برنامج يبين شكل المصفوفة الغير منتظمة
2. class Chp6_5
3. {
4.   public static void main( String[] arg )
5.   {
6.     // declare and construct a 2D array
7.     int[][] uneven =
8.         { { 1, 9, 4 },
9.           { 0, 2},
10.            { 0, 1, 2, 3, 4 } };
11.
12.     System.out.println("uneven[0][2] is " + uneven[0][2] ) ; // OK
13.     System.out.println("uneven[1][1] is " + uneven[1][1] ) ; // OK
14.     System.out.println("uneven[1][2] is " + uneven[1][2] ) ; // WRONG!
15.
16.     uneven[2][4] = 97;  // OK
17.     uneven[1][4] = 97;  // WRONG!
18.
19.     int val = uneven[0][2] ;  // OK
20.     int sum = uneven[1][2] ;  // WRONG!
21.   }
22. }
```

فيكون ناتج البرنامج:

```
uneven[0][2] is 4
uneven[1][1] is 2
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsExceptior
        at Chp4_5.main(Chp4_5.java:14)
```

نلاحظ في السطر 14 حصول خطاء ناتج من الوصول إلى عنصر ليس موجود بمحتوى المصفوفة .

- **طباعة المصفوفة الغير منتظمة**

في المصفوفة المنتظمة كنا نستخدم الصيغة التالية

```
For(i=0;i<arrayname.length;i++)
        For(j=0;j<arrayname[0].length;j++)
```

لكن في المصفوفة الغير منتظمة يختلف الأمر وأن استخدمنا العبارة السابقة فأن المترجم سيطبع رسالة خطاء كما في المثال رقم (Chp6_6) وهو الوصول إلى عنصر غير موجود بمحتوى المصفوفة ، فكيف سنتعامل مع ذلك ؟

الجواب في البرنامج التالي:

```
1. //برنامج يطبع مصفوفة ثنائية البعد غير منتظمة
2. class Chp6_6
3. {
4.   public static void main( String[] arg )
5.   {
6.     // declare and construct a 2D array
7.     int[][] uneven =
8.         { { 1, 9, 4 },
9.           { 0, 2},
10.            { 0, 1, 2, 3, 4 } };
11.
12.    // print out the array
13.    for ( int row=0; row < uneven.length; row++ )
14.    {
15.      System.out.print("Row " + row + ": ");
16.      for ( int col=0; col < uneven[row].length; col++ )
17.        System.out.print( uneven[row][col] + " ");
18.      System.out.println();
19.    }
20.
21.  }
22. }
```

ناتج البرنامج :

```
Row 0: 1 9 4
Row 1: 0 2
Row 2: 0 1 2 3 4
```

6.4 نسخ المصفوفات Arrays Copying

غالباً ما تحتاج إلى إنشاء عدة نسخ من المصفوفة ، وسنقوم بإنشاء برنامج يقوم بنسخ مصفوفة إلى مصفوفة أخرى ،ويكون التخزين بداخل المصفوفة الثانية معكوس.

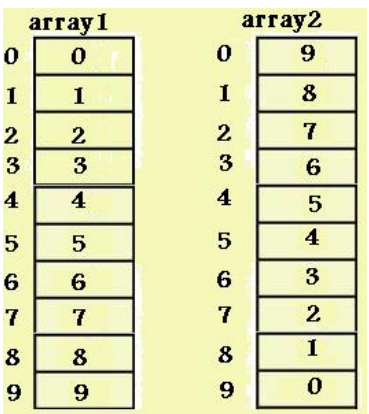
```
1. //برنامج بواسطة المصفوفات الأحادية يينسخ مصفوفة إلى مصفوفة أخرى معكوساً
2. public class Chp6_7 {
3.   public static void main(String[] args) {
4. int [] array1=new int[10];
5. int [] array2=new int[10];
```

```
6.      int j;
7.
8.      for(j=0;j<10;j++)
9.              array1[j]=j;
10.
11.     for(j=0;j<10;j++)
12.             array2[j]=array1[9-j];
13.
14.     for(j=0;j<10;j++)
15.             System.out.print(array1[j]+" ");
16.
17.     System.out.println();
18.
19.     for(j=0;j<10;j++)
20.             System.out.print(array2[j]+" ");
21.
22.     System.out.println();
23.  }
24. }
```

فيكون ناتج البرنامج كما يلي:

```
0 1 2 3 4 5 6 7 8 9
9 8 7 6 5 4 3 2 1 0
```

نلاحظ في برنامجنا السابق أننا حجزنا للمصفوفة array2 مواقع بحجم المصفوفة array1. كما في الشكل 6-6.

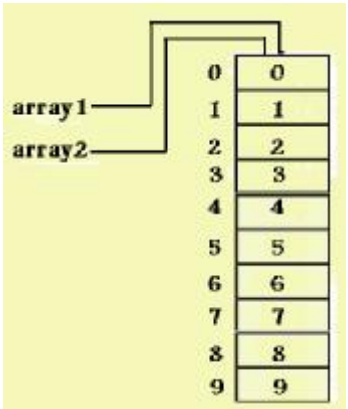


الشكل 6-6

ونظرا لكبر البرنامج يمكننا اختصار حجم البرنامج و مواقع الذاكرة بهذه الخطوة:

aArray2=array1;

حيث تكون المصفوفة array2 تؤشر إلى نفس المصفوفة array1 كما في الشكل 6-7:



الشكل 7-6

وبهذا نكون قد اختصرنا من البرنامج ، وقللنا من المساحة المحتجزة بداخل الذاكرة.

6.5 خوارزميات ترتيب المصفوفات Sort Algorithms

إحدى أبرز المشاكل في عالم الكمبيوتر هي مشاكل الترتيب. أي كيفية ترتيب قائمة من العناصر الغير مرتبة . فهنالك العديد من الحلول لهذه المشاكل تعرف باسم " خوارزميات الترتيب"Sort Algorithms ، بعض من هذه الخوارزميات سهلة وبديهية، بينما بعضها الآخر معقد، في النهاية كلاً منهم يعطي نتائج مذهلة وللمبرمج حرية الاختيار.

أشهر سبعة خوارزميات للترتيب هي:

1. خوارزم الفقاعة Bubble
2. خوارزم الحشر Insertion
3. خوارزم التحديد Selection
4. خوارزم التكويم Heap
5. خوارزم الدمج Merge
6. خوارزم السرعة Quick
7. خوارزم الهيكل Shell

سنتعرف في هذا الكتاب على كل 4 خوارزميات، نشرحها ونكتب خرائط التدفق flowchartsالخاصة بها ونقوم بتطبيق برمجي coding عليها..

س/ أين توجد هذه البيانات ؟

ج/ تكون:

1. مخزنة داخلياً في الحاسوب Internal في الذاكرة مثلاً .Memory
2. أو مخزنة على وسط تخزين خارجي External في ملف مثلاً.File

س/ ما هي أنواع الترتيب؟!

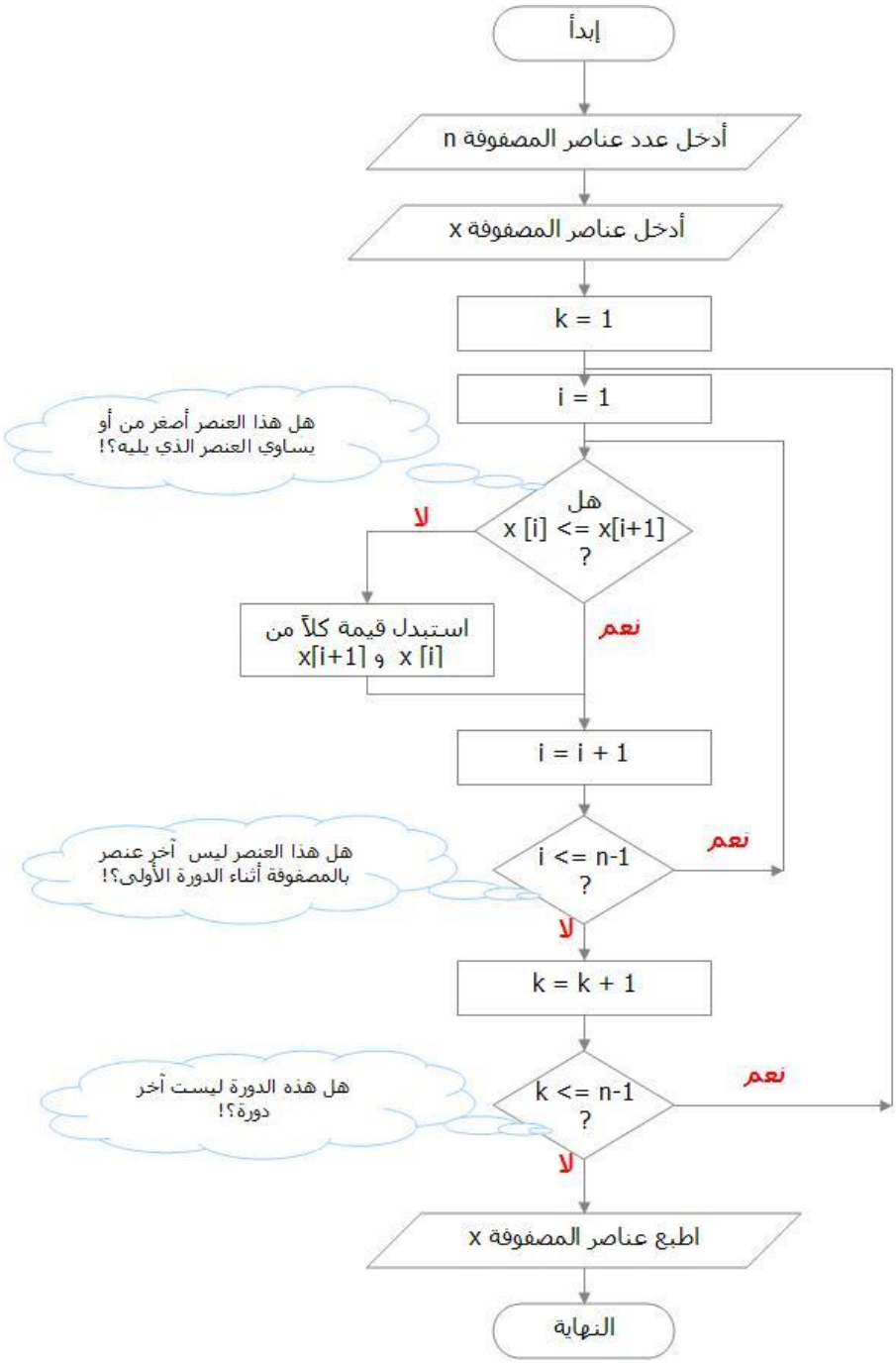
ج/ يكون الترتيب إما تصاعدياً أو أبجدياً Ascending(A->Z) , أو تنازلياً Descending (Z->A)

6.5.1 خوارزم للترتيب الفقاعي Bubble Sort Algorithm

تعتبر هذه الخوارزمية الأبطأ من بين أقرانه، حيث تقوم هذه الخوارزمية بترتيب عناصر المصفوفة عنصراً عنصراً عن طريق مقارنته بالعنصر الذي بعده، ويقوم بالتبديل في موقعهما في المصفوفة متى ما استلزم الأمر. تكرر هذه الخوارزمية هذه الدورة على جميع العناصر عنصراً خلف الآخر إلى أن يصل إلى دورة لا يتغير فيها موقع أي عنصر، حينها يتوقف وتكون العناصر قد ترتبت!

بعبارة أخرى:

تعمل هذه الخوارزمية على ترتيب عناصر مصفوفة ليست مرتبة العناصر، وتتم عملية الترتيب على مراحل أو دورات phases كل مرحلة تتضمن عملية مقارنة زوجين متجاورين من العناصر واستبدال موضعهما إذا كان ترتيبهما غير صحيح... وهكذا، انظر إلى الشكل رقم 8-6:


إبدأ
أدخل عدد عناصر المصفوفة n
أدخل عناصر المصفوفة x
k = 1
i = 1
هل هذا العنصر أصغر من أو يساوي العنصر الذي يليه؟!
هل x [i] <= x[i+1] ?
لا
نعم
استبدل قيمة كلاً من x [i] و x[i+1]
i = i + 1
هل هذا العنصر ليس آخر عنصر بالمصفوفة أثناء الدورة الأولى؟!
i <= n-1 ?
نعم
لا
k = k + 1
هل هذه الدورة ليست آخر دورة؟!
k <= n-1 ?
نعم
لا
اطبع عناصر المصفوفة x
النهاية

شكل 8-6

مثال:

لنفرض أننا نريد ترتيب مصفوفة تحتوي س من العناصر تصاعدياً، بعد دورته الأولى في المصفوفة سيكون أكبر عنصر موضوع في مكانه الصحيح (آخر المصفوفة بمعنى[س-1]) ، وبعد دورته الثانية سيكون ثاني أكبر عنصر في المصفوفة في مكانه الصحيح ([س-2 , ...]وهكذا، تتوقف الخوارزمية متى ما أتم دورة دون تبديل في أماكن العناصر.

تقلب الآية لو كان الترتيب تنازلياً كما في الشكل 9-6.

الشكل 9-6

والآن قد تتساءل كيف نكتب هذا الكود برمجياً؟

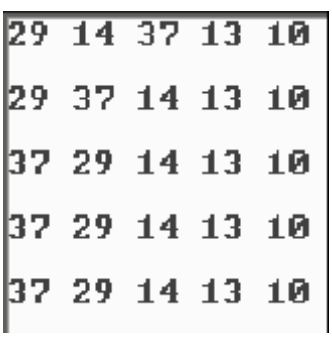
الإجابة سهلة، يمكنك تمثيل الخوارزمية السابقة بلغة Java ، كالتالي:

- **برنامج "خوارزميات الترتيب":**

```
1. // A bubble sort for Strings.
2. class Chp6_8 {
3. public static void main(String args[]) {
4.
```

```
5. int arr[] = {29,10,14,37,13};
6.
7. for(int j =arr.length-1; j >=0 ; j--) {
8.       for(int i = 1; i <=j ; i++) {
9.                if(arr[i-1]< arr[i]) {
10.                                        int t = arr[i-1];
11.                                        arr[i-1] = arr[i];
12.                                        arr[i] = t;
13.                                      }
14.                                    }
15.     for(int i =0;i<arr.length; i++)
16.                System.out.print(arr[i]+" ");
17.
18.    System.out.println("\n");
19.                                         }
20. }
21. }
```

والنتيجة أثناء استخدام برنامجنا وتطبيق هذا الكود فيه:



من المؤكد أنك لاحظت بأن المصفوفة ترتبت بعد الدورة الثانية، لكن الخوارزمية أكملت حتى الدورة الخامسة ولم تتوقف فوراً، هذا صحيح، لذلك تعتبر هذه الطريقة من أبسط وأبطأ طرق الترتيب!

ربما تتساءل منذ البداية لماذا هذا الاسم الغريب ؟

سميت بالفقاعة لأن العنصر الأصغر يصعد كالفقاعة إلى بداية المصفوفة .

معلومة إضافية متقدمة:

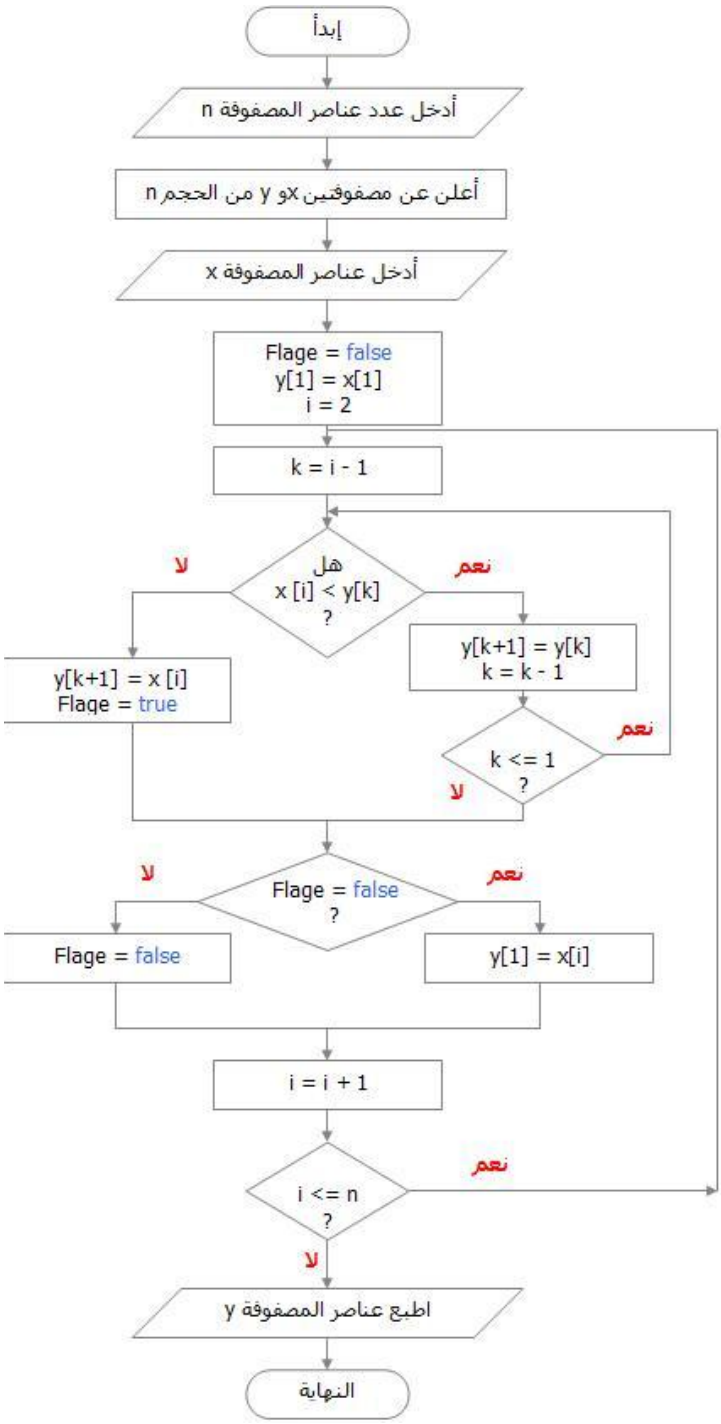
لو استخدمنا مفهوم Big-Oh Notation لتحليل هذه الخوارزمية لاستنتجنا أن أسوأ حالات تطبيق هذه الخوارزمية هي عندما يكون(On^2) وهو نفسه درجة فعالية هذه الخوارزمية في الحالة القياسية.

6.5.2 خوارزم الحشر Insertion Sort Algorithm

تعمل هذه الطريقة على اعتبار أن هناك مصفوفتين، في البداية تحتوي إحداهما على البيانات المطلوب ترتيبها بمعنى الغير مرتبة unsorted والأخرى فارغة. وفي النهاية ستحتوي المصفوفة الفارغة على بيانات المصفوفة الأولى ولكن مرتبه إما تصاعدياً أو تنازلياً!

الفكرة الأساسية في هذه الطريقة هي تحريك العناصر من المصفوفة الأولى (الغير مرتبة العناصر) إلى المصفوفة الثانية عنصراً كل مرة مع مراعاة وضع العنصر المتحرك في مكانه الصحيح، ولكي نضع العنصر في مكانه الصحيحة في المصفوفة الثانية المرتبة العناصر بالنسبة لجميع العناصر التي سبقت هذا العنصر المتحرك، فإن هذه العملية قد تتطلب حشر هذا العنصر الجديد بين عنصرين مرتبين مسبقاً ليأخذ مكانه الصحيح، من هنا أتى اسم هذه الخوارزمية .

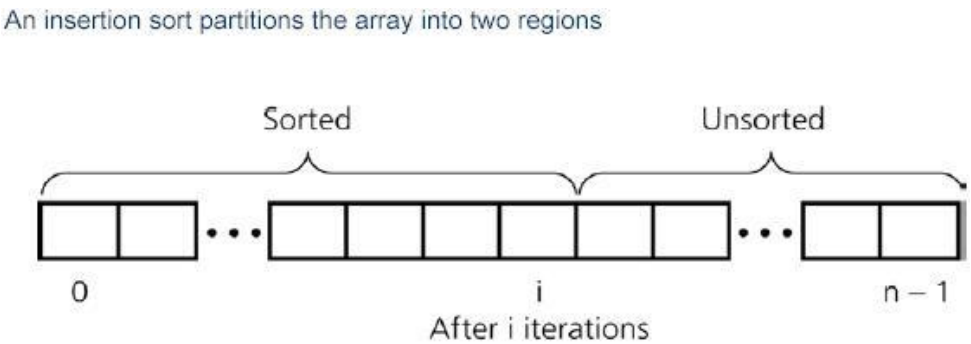
انظر إلى الشكل 10-6 والذي يحاكي العملية:


إبدأ
أدخل عدد عناصر المصفوفة n
أعلن عن مصفوفتين xو y من الحجم n
أدخل عناصر المصفوفة x
Flage = false
y[1] = x[1]
i = 2
k = i - 1
هل
x [i] < y[k]
?
لا
نعم
y[k+1] = x [i]
Flage = true
y[k+1] = y[k]
k = k - 1
k <= 1
?
نعم
لا
Flage = false
?
لا
نعم
Flage = false
y[1] = x[i]
i = i + 1
i <= n
?
نعم
لا
اطبع عناصر المصفوفة y
النهاية

الشكل 6-10

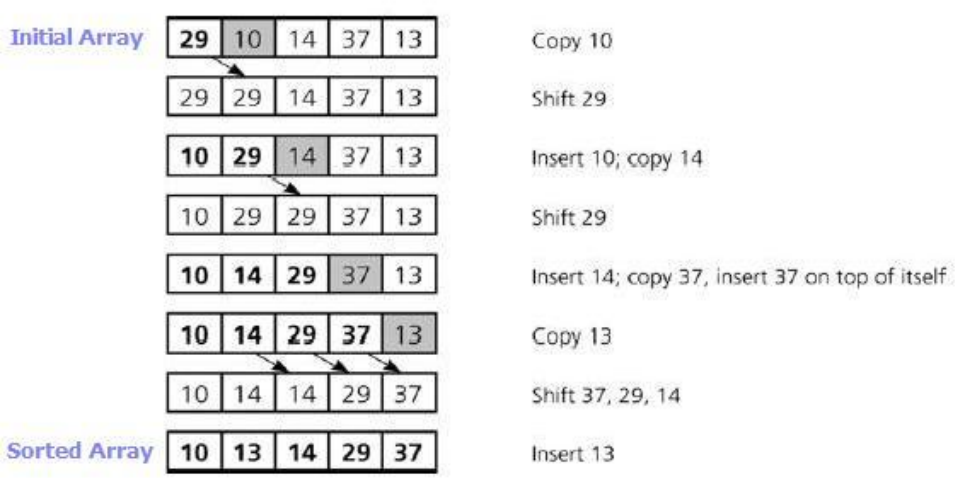
هذا أبسط تمثيل لهذه الخوارزمية باستخدام مصفوفتين !

نعم وكما يتبادر إلى ذهنكم الآن يمكننا أن نمثل هذه الخوارزمية باستخدام مصفوفة واحدة عن طريق حشر العنصر في مكانه الصحيح فيها ثم عمل shift لبقية العناصر وهكذا ستتقسم المصفوفة الواحدة إلى قسمين، قسم به العناصر المرتبة وقسم به العناصر الغير مرتبة كما في الشكل 6-11:



شكل 6-11

لنرى مثال على ذلك كما في الشكل 6-12:



الشكل 6-12

لنمثله الآن برمجياً, وباستخدام مصفوفة واحدة. كما يلي:

```
1. // array of integers to hold values
2. class Chp6_9 {
3. public static void main(String args[]) {
4. // Insertion Sort Algorithm
5. int arr[] = {29,10,14,37,13};
6.
7.   int i;
8.   int j;
9.   int index;
10.
11.   for( i = 1; i < arr.length; i++ )
12.   {
13.     index = arr[i];
14.     j = i;
15.
16.     while( (j > 0) && (arr[j-1] > index) )
17.     {
18.       arr[j] = arr[j-1];
19.       j = j - 1;
20.     }
21.
22.     arr[j] = index;
23.
24.      for(j =0;j<arr.length; j++)
25.                System.out.print(arr[j]+" ");
26.
27.     System.out.println("\n");
28.
29.   }
30. }
31. }
```

وكما رأيتم فهي من حيث السرعة مناسبة لتطبق على مصفوفة ذات ألف عنصر أو أقل. هذه الخوارزمية تعتبر أسرع مرتين من طريقة الفقاعة Bubble كما أنه أسرع 40% كذلك من خوارزمية التحديد (Selection) التي سنتعرف عليها فيما بعد .

هذه هي النتيجة باستخدام برنامجنا:

```
29 10 14 37 13
29 14 10 37 13
37 29 14 10 13
37 29 14 13 10
```

رائع، أليس كذلك، هذه الخوارزميات تعطينا خيال برمجي كبير جداً .

معلومة إضافية متقدمة:

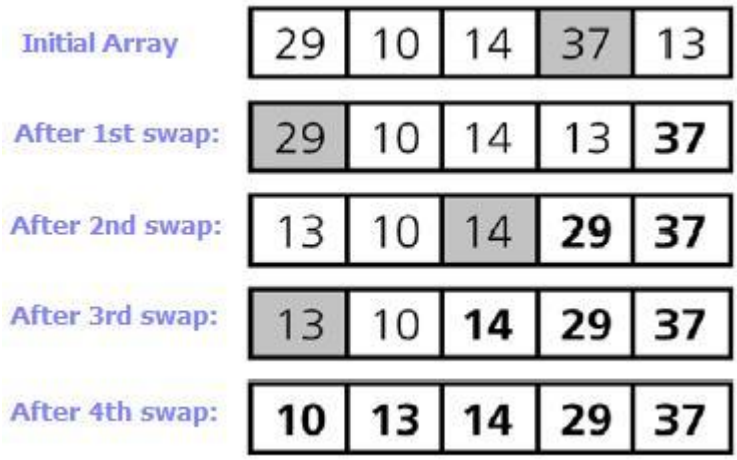
لو استخدمنا مفهوم Big-Oh Notation لتحليل هذه الخوارزمية لاستنتجنا أن أسوأ حالات تطبيق هذه الخوارزمية هي عندما يكون(On^2) وهو نفسه درجة فعالية هذه الخوارزمية في الحالة القياسية.

6.5.3 خوارزم التحديد أو الاختيار Selection Sort Algorithm

تعمل هذه الطريقة عن طريق تحديد أو اختيار أكبر (أو أصغر) عنصر غير مرتب في المصفوفة. ومن ثم تحريكه كي يشغل الحيز المتاح له في آخرها، وهكذا لكل عنصر. فتنتهي عملية الترتيب عندما ننتهي من تحريك جميع العناصر.

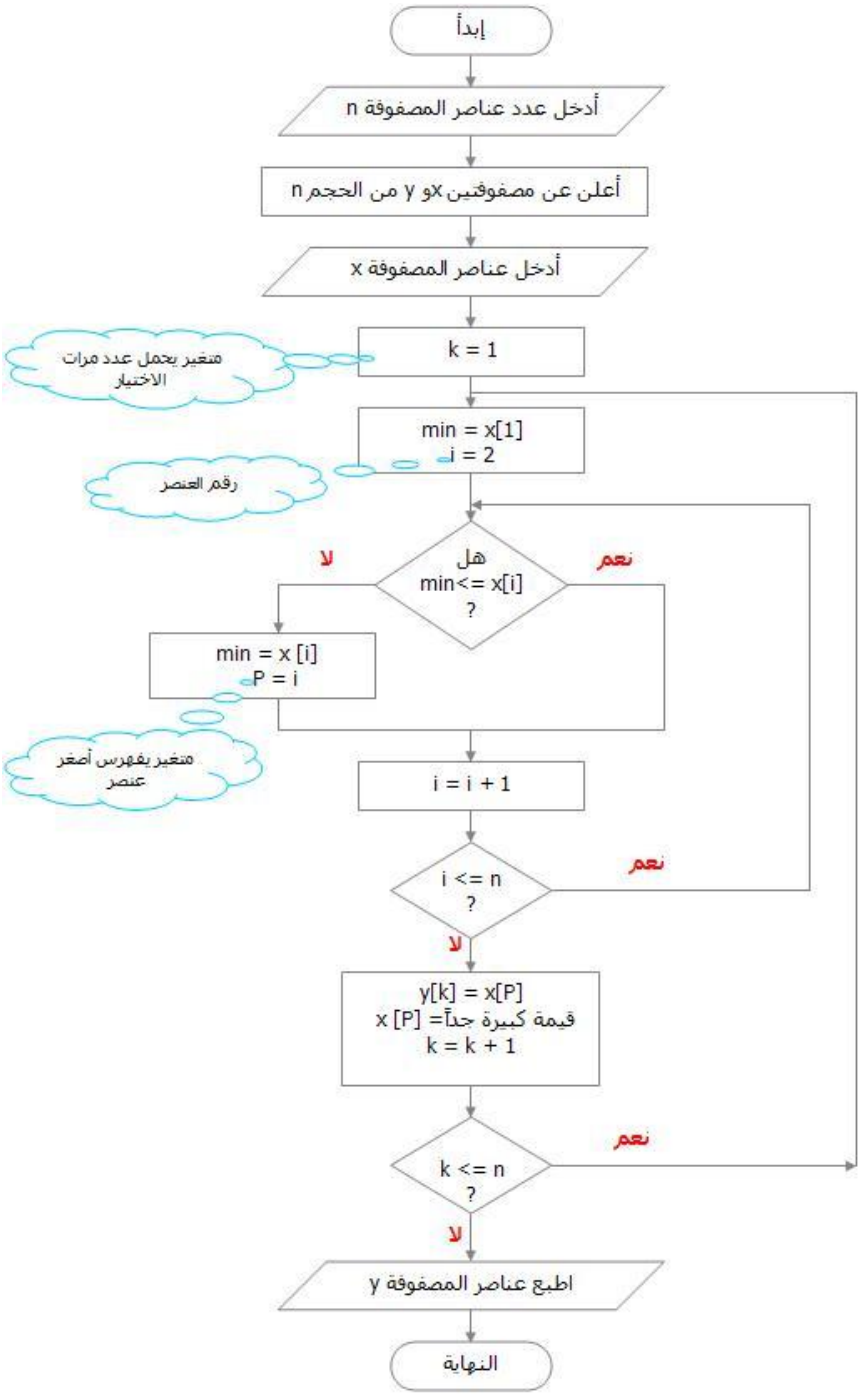
مثال:

في الشكل 13-6: العناصر المظللة بالرمادي هي العناصر المختارة أو المحددة، بينما العناصر المكتوبة بالبنط العريض bold هي العناصر المرتبة في مكانها الصحيح:



الشكل 13-6

من الممكن أن نستخدم مصفوفتين لعمل ذلك، بحيث تحوي الأولى العناصر غير المرتبة ويتم تخزين هذه العناصر بترتيب تصاعدي أو تنازلي في المصفوفة الثانية، سنقوم بكتابة flowchart باستخدام مصفوفتين كما في الشكل 14-6 ، ثم نكتب الكود باستخدام مصفوفة واحدة:


إبدأ
أدخل عدد عناصر المصفوفة n
أعلن عن مصفوفتين xو y من الحجم n
أدخل عناصر المصفوفة x
k = 1
متغير يحمل عدد مرات الاختيار
min = x[1]
i = 2
رقم العنصر
هل min<= x[i] ?
لا
نعم
min = x [i]
P = i
متغير يفهرس أصغر عنصر
i = i + 1
i <= n ?
نعم
لا
y[k] = x[P]
قيمة كبيرة جداً= x [P]
k = k + 1
k <= n ?
نعم
لا
اطبع عناصر المصفوفة y
النهاية

الشكل 14-6

وهادا هو تمثيل الخوارزمية باستخدام مصفوفة واحدة:

```
1. // array of integers to hold values
2. // Selection Sort Algorithm
3. class Chp6_10 {
4. public static void main(String args[]) {
5. int arr[] = {29,10,14,37,13};
6.
7.   int i, j;
8.   int min, temp;
9.
10.   for( i = 0; i < arr.length-1; i++ )
11.   {
12.     min = i;
13.
14.     for( j = i+1; j < arr.length; j++ )
15.     {
16.       if( arr[j] < arr[min] )
17.       {
18.         min = j;
19.       }
20.     }
21.
22.     temp = arr[i];
23.     arr[i] = arr[min];
24.     arr[min] = temp;
25.
26.
27.      for(j =0;j<arr.length; j++)
28.                System.out.print(arr[j]+" ");
29.
30.     System.out.println("\n");
31.
32.   }
33. }
34. }
```

في الحقيقة تعتبر هذه الخوارزمية أفضل من خوارزم الفقاعة Bubble بقل يل، لكنه ليس أفضل من خوارزم الحشرInsertion ، استخدمه إن كان عدد عناصر مصفوفتك لا يتجاوز الألف ونيفاً!

لنطبقه على برنامجنا لنرى كيف ستكون النتيجة:

معلومة إضافية متقدمة:

لو استخدمنا مفهوم Big-Oh Notation لتحليل هذه الخوارزمية لاستنتجنا أن أسوأ حالات تطبيق هذه الخوارزمية هي عندما يكون(On^2) وهو نفسه درجة فعالية هذه الخوارزمية في الحالة القياسية.

6.5.4 خوارزم التكويم Heap Sort Algorithm

هذه الخوارزمية لا يتطلب أكثر من مصفوفة كي يرتب عناصر مصفوفة ما، ولا يحتاج كذلك لعمليات التكرار والإعادة العديدةmassive recursion ، لذلك يعتبر مناسب لترتيب مصفوفات تحوي ملايين العناصر.

طريقة عمله كما يوحي لك اسمه، يبدأ ترتيبه ببناء كومة خارجية مكدسة من مجموعة بيانات المصفوفة (لنقل أنه يحفظها على القرص(Hard Disk ، ثم ينقل أكبر عنصر من هذه الكومة ويكومه في آخر المصفوفة المرتبة، وبعد إخراج أكبر عنصر من الكومة heap يعيد بناء الكومة من جديد ويخرج منها أكبر عنصر متبقي فيها وينقله إلى المكان ما قبل الأخير في المصفوفة المرتبة . وهكذا حتى يكتمل ترتيب المصفوفة المرتبة ولا يبقى بيان واحد في الكومة Heap !

البناء الأولي لهذه الخوارزمية برمجياً يتطلب وجود مصفوفتين، الأولى تعمل كالكومة والثانية تعمل لحفظ البيانات المرتبة، ولكننا سنمثل هذه الخوارزمية بخدعة ونستخدم مصفوفة واحدة

تعمل كالمكوم ويتم ترتيب العناصر فيها في نفس الوقت، فعندما نخرج أي عنصر أو بيان من هذه المصفوفة نقوم بتحرير مكان خالي في آخر المصفوفة كي يحتوي هذا العنصر!

إليكم تمثيل هذه الخوارزمية:

```
1. // Heap Sort Algorithm
2. class Chp6_11 {
3. // array of integers to hold values
4. static int arr[] = {29,10,14,37,13};
5. public static void main(String args[]) {
6.   int i,j;
7.   int temp;
8.
9.   for( i = (arr.length/2)-1; i >= 0; i-- )
10.  {
11.    siftDown( i, arr.length );
12.  }
13.
14.  for( i = arr.length-1; i >= 1; i-- )
15.  {
16.    temp = arr[0];
17.    arr[0] = arr[i];
18.    arr[i] = temp;
19.    siftDown( 0, i-1 );
20.
21.     for(j =0;j<arr.length; j++)
22.               System.out.print(arr[j]+" ");
23.
24.    System.out.println("\n");
25.  }
26.
27. }
28.
29. static public void siftDown( int root, int bottom )
30. {
31.  boolean  done = false;
32.  int maxChild;
33.  int temp;
34.
35.  while( (root*2 <= bottom) && (!done) )
36.  {
```

```
37.     if( root*2 == bottom )
38.       maxChild = root * 2;
39.     else if( arr[root * 2] > arr[root * 2 + 1] )
40.       maxChild = root * 2;
41.     else
42.       maxChild = root * 2 + 1;
43.
44.     if( arr[root] < arr[maxChild] )
45.     {
46.       temp = arr[root];
47.       arr[root] = arr[maxChild];
48.       arr[maxChild] = temp;
49.       root = maxChild;
50.     }
51.     else
52.     {
53.       done = true;
54.     }
55.   }
56. }
57. }
```

استخدمنا دالة باسم (siftDown) وهي جزء برمجي يتم استدعاءه كلما ذكر أسم الدالة. سيأتي شرح الدوال في الفصل التالي.

لنطبق البرنامج ونرى النتيجة كالتالي:

```
29 14 13 10 37

14 13 10 29 37

13 10 14 29 37

10 13 14 29 37
```

كما لاحظت، استطعنا أن نكوم بعض العناصر في طرف المصفوفة ونرتب العناصر الأخرى في طرفها الآخر.

Searching Algorithms	6.6 خوارزميات البحث

عادةً يقوم المبرمج بالتعامل مع مصفوفات كبيرة الحجم وبالتالي لتحديد أي عنصر معين موجود في مصفوفة لابد من استخدام طرق البحث ، ومن الضروري أن يستخدم إستراتيجية معينه يحدد له ما إذا كان العنصر الذي يبحث عنه key ينتمي إلى هذه المصفوفة أم لا! هذه الإستراتيجية يطلق عليها "البحث" وله عدة أنواع. ومن خلال هذا الدرس سوف نتعلم طريقتين من طرق البحث وهما:

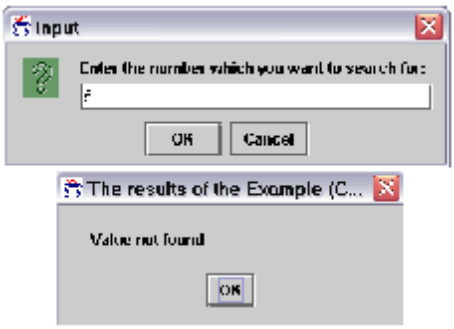
- البحث الخطي (linear Search) .
- البحث الثنائي (Binary Search).

6.6.1 البحث الخطي (linear Search)

يستخدم البحث الخطي للبحث عن عنصر معين داخل المصفوفة غير المرتبة أو المصفوفة المرتبة ، وفي هذه الطريقة يتم مقارنة جميع محتويات المصفوفة مع القيمة المراد البحث عنها وبشكل متسلسل من بداية المصفوفة إلى نهايتها.
وهذا البرنامج لهذه العملية :

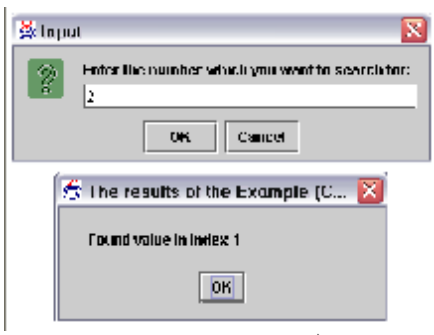
```
1. //برنامج البحث الخطي
2.  import javax.swing.*;
3.  class Chp6_12{
4.          public static void main(String args[]){
5.                  int n[] = new int[10];
6.                  int num, k=-1;
7.                  String title="The results of the Example (Chp4_11)";
8.                  String s, output="";
9.                  for(int i=0; i<n.length; i++)
10.                         n[i]=i*2;
11.                 s=JOptionPane.showInputDialog("Enter the number which you want to search for:");
12.                 num=Integer.parseInt(s);
13.                 for(int i=1; i<n.length; i++)
14.                         if(n[i]==num){
15.                                                 k=i;
16.                                                 break;
17.                                                  }
18.                 if(k!=-1)
19.                         output+="Found value in index "+k;
20.                         else
21.                         output+="Value not found";
22.                 JOptionPane.showMessageDialog(null, output, title,JOptionPane.PLAIN_MESSAGE);
23.                 System.exit(0);
24.                                                                                         }
25. }
```

حيث يكون خرج البرنامج في حالة أدخلنا الرقم 5 كما في الشكل 15-6:



شكل 6-15

أم في حالة أدخلنا الرقم 2 كما في الشكل 6-16:



شكل 6-16

شرح المثال:

في هذا المثال تم البحث داخل مصفوفة مرتبة عن رقم معين يتم إدخاله عن طريق لوحة المفاتيح . حيث تقوم الأسطر (13-17) بمقارنة الرقم المراد البحث عنه والمخزن في المتغير num مع جميع محتويات المصفوفة. وفي حالة تم العثور على هذا الرقم في المصفوفة يتم تخزين موقعة في المتغير k، وفي حالة عدم العثور على الرقم في المصفوفة يبقى محتوى المتغير k كما هو -1 . ومن خلال الأسطر (18-21) يتم طباعة رسالة بعدم وجود الرقم المراد البحث عنة في المصفوفة إذا كانت قيمة k لم تتغير وبقيت -1 ، بينما إذا تغيرت قيمة k فهذا يعني بأن الرقم المراد البحث عنه موجود داخل المصفوفة وفي الموقع k وسوف يتم طباعة رسالة بذلك . والشكل رقم ــ يبين ناتج تنفيذ البرنامج السابق في حالة العثور على الرقم 2 في المصفوفة .

6.6.2 طريقة البحث الثنائي Binary Search

ولكي نطبق أحد خوارزميات الـ Binary Search على مصفوفة ما نتبع الخطوات البسيطة التالية:

1. الخطوة الأولى والأهم والتي لا يمكن تطبيق الـ Binary Search هي ترتيب المصفوفة تصاعدياً أو تنازلياً أو أبجدياً على حسب نوع البيانات المخزنة فيها.
2. تحديد أول عنصر في المصفوفة ولنسمه i ، وآخر عنصر فيها ولنسمه مثلاً j .
3. تحديد العنصر الذي يقع في منتصف هذه المصفوفة ولنسمه k .

بعد ذلك يمكننا تطبيق إستراتيجية البحث الثنائي على المصفوفة، وهناك عدة خوارزميات للبحث الثنائي، أولهما :

خوارزم البحث الثنائي Binary Search Algorithm

تقوم فكرة البحث الثنائي على تقسيم المصفوفة إلى نصفين واستبعاد النصف الذي لا ينتمي إليه المفتاح key الذي نبحث عنه، كيف ذلك؟

عن طريق تحديد العنصر الذي يقع في منتصف هذه المصفوفة، ثم نقارن هذا العنصر مع المفتاح الذي نبحث عنه كالتالي) تذكر أن مصفوفتنا مرتبة تصاعدياً أو تنازلياً.

1. إذا كان يساويه نكون قد وجدنا العنصر الذي نبحث عنه.
2. إذا كانت قيمة المفتاح أقل من قيمة العنصر الأوسط في المصفوفة، إذن نحتاج أن نبحث فقط في نصف المصفوفة الأول ونستبعد البحث في نصفها الثاني.
3. وفيما عدا ذلك: إذا كانت قيمة المفتاح أكبر من قيمة العنصر الأوسط في المصفوفة، إذن نحتاج أن نبحث فقط في نصف المصفوفة الثاني ونستبعد البحث في نصفها الأول.
4. بعد ذلك: نعتبر النصف الذي حددنا لأنفسنا البحث فيه مصفوفة بحد ذاتها، نحدد فيها الـ(i, & k) j أي نقوم بتقسيمها إلى قسمين. ونطبق نفس الخطوات من 1 إلى 3 فيها، ثم نقارن المفتاح مع العنصر الأوسط الجديد، بنفس الترتيب الذي ذكر في الخطوات 1 إلى 3 السابقة.

البرنامج التابع لهذه الخوارزمية :

قبل البدء في هذه الخوارزمية يجب ترتيب المصفوفة بإحدى الخوارزميات التي ذكرت سابقاً .

```
1. //1 برنامج البحث الثنائي
2.  import javax.swing.*;
3.  class Chp6_13{
4.  public static void main(String args[]){
5.
6.  int NumArray[]={0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28};
7.
8.  int result,key,max_size=15;
9.
10.  String s;
11.  s=JOptionPane.showInputDialog("Enter the number which you want to search for:");
12.  key=Integer.parseInt(s);
13.
14.  int i=0, j=max_size-1, k=(i+j)/2;
15.
16.  while(i<j&&i!=k){
17.       if (key == NumArray[k]){
18.
19.                 System.out.println("we found the key "+ key);
20.                 System.exit(0);
21.                                                   }
22.       else{
23.                 if (key < NumArray[k]){
24.                           j=k;
25.                           k=(i+j)/2;
26.                                                             }
27.                 if (key > NumArray[k]){
28.                           i=k;
29.                           k=(i+j)/2;
30.                                                             }
31.                 }
32.                                   }
33. System.out.println("we not found the key "+key);
34. System.exit(0);
35. }
36. }
```

أرجو أن يكون هذا المثال البسيط جداً واضحاً.

عدد مرات البحث في أي مصفوفة عن عنصر محدد باستخدام الـ:Binary Search

لو تساءلنا عن أقصى عدد من مرات البحث باستخدام الـ Binary Search في أي مصفوفة، لوجدنا أنه يُعطى من إيجاد القوة التي يرفع إليها رقم 2 كي يعطينا العدد الذي يزيد عن عناصر المصفوفة بواحد. أي أنه أول قوة لـ 2 والتي تُعطي رقم أكبر من عدد عناصر المصفوفة بواحد.

ففي مثالنا: استخدمنا مصفوفة من 15 عنصر، نلاحظ أن العدد الذي يزيد على عدد عناصر المصفوفة بواحد، أي العدد 16 ينتج من القوة الرابعة لرقم 2 (2^4=16) وذلك يعني أننا نحتاج على الأكثر لأربع مرات مقارنة في الـ Binary Search حتى نجد العنصر الذي نبحث عنه! فمن الممكن أن نجده من أول مرة في المقارنة، ومن الممكن أن نجده في ثاني مرة، أو ثالث مرة أو رابع مرة. أو أن يكون غير موجود في المصفوفة!

وفي مثال آخر: لو بحثنا في مصفوفة تحوي 1024 عنصر، سنحتاج إلى 10 مرات للمقارنة كحد أقصى، ونعرف ذلك بتكرار قسمة عدد العناصر على رقم 2 إلى أن نصل إلى العدد واحد في خارج القسمة (وسبب ذلك هو أننا بعد كل مقارنة نقوم بإلغاء نصف عناصر المصفوفة من الاعتبار) ، فبتكرار قسمة 1024 على رقم 2 نحصل على القيم التالية على الترتيب:2,4,8,16,32,64 ,128,256,512 ، ورقم 1. نلاحظ أن العدد 1024(2^10) قسم على رقم 2 عشر مرات حتى حصلنا على العدد.1
نستنتج من ذلك، أن القسمة على اثنين تقابل مرة واحدة من المقارنة في الـ Binary Search Algorithm. فمصفوفة بـ 1048576 (2^20) عنصر تستلزم على الأكثر 20 مرة من المقارنة حتى نجد العنصر الذي نبحث عنه، ومصفوفة تحوي بليون عنصر، تستلزم على الأكثر إلى 30 مرة من المقارنة حتى نجد العنصر المطلوب فيها!

ترى، كم يوفر لنا هذه الإستراتيجية من الوقت في البحث ؟ فقط 30 مرة من البحث بين بليون عنصر لنجد ضالتنا! إنها الإستراتيجية عبقرية فعلاً.

يعتبر البحث الخطي من الطرق الجيدة والمفيدة لإيجاد العنصر في مصفوفة صغيرة أو في مصفوفة غير مرتبة ، لكنة غير فعال مع المصفوفات الكبيرة حيث يكون البحث الثنائي أكثر فعالية ، لكن يحتاج البحث الثنائي إلى مصفوفة مرتبة.

بما أنه يتم استخدام البحث الثنائي بشكل متكرر في البرمجة فإن لغة Java تقدم لنا مناهج جاهزة من أجل البحث عن مفتاح ما في مصفوفة من النمط (float , int, char, long, double, short) . والترتيب أيضاً. فعلى سبيل المثال يقوم البرنامج التالي بالبحث عن المفتاح في مصفوفة:

```
1. // برنامج يوضح مناهج المصفوفات
2. import java.util.*;
3. class Chp6_14 {
4.         public static void main(String args[]) {
5.
```

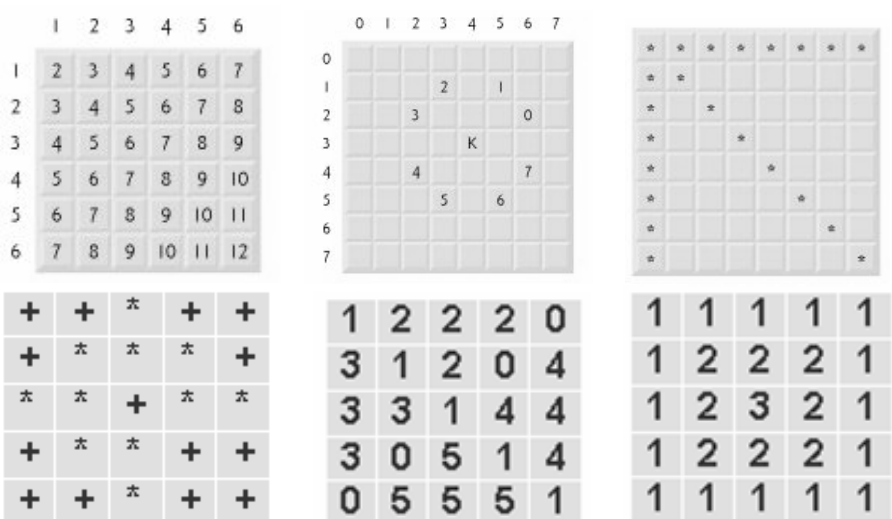
```
6.                  // allocate and initialize array
7.                  int array[] = new int[10];
8.
9.                  for(int i = 0; i < 10; i++)
10.                 array[i] = -3 * i;
11.
12.                 // display, sort, display
13.                 System.out.print("Original contents: ");
14.                 display(array);
15.                 Arrays.sort(array);
16.                 System.out.print("Sorted: ");
17.                 display(array);
18.
19.                 // binary search for -9
20.                 System.out.print("The value -9 is at location ");
21.                 int index =Arrays.binarySearch(array, -9);
22.                 System.out.println(index);
23.
        }
24.
25. static void display(int array[]) {
26.     for(int i = 0; i < array.length; i++)
27.                 System.out.print(array[i] + " ");
28.     System.out.println("");
29.                                                                 }
30. }
```

فيكون ناتج البرنامج

```
Original contents: 0 -3 -6 -9 -12 -15 -18 -21 -24 -27
Sorted: -27 -24 -21 -18 -15 -12 -9 -6 -3 0
The value -9 is at location 6
```

تمارين الفصل:

1. أكتب برنامج لقراءة مصفوفة مكونة من 20 عنصر وتخزين الأعداد الفردية في مصفوفة والأعداد الزوجية في مصفوفة أخرى؟
2. أكتب برنامج ينشئ جدول الضرب في مصفوفة ذات بعدين؟
3. أكتب برنامج يرتب مصفوفة ثنائية البعد بشرط أن يكون الترتيب في وقت الأدخال؟
4. أكتب برنامج يطبع العدد الأكثر تكراراً داخل المصفوفة ، ويطبع عدد مرات التكرار؟
5. أكتب برنامج يعمل على تبديل العنصر الأكبر مع العنصر الأصغر ، وطباعة موقع كلاً منها؟
6. أكتب برنامج يعكس القطر الرئيسي لمصفوفة ثنائية البعد؟
7. أكتب برنامج يأخذ أكبر قيمة من كل صف ويقوم بتبديله مع القطر الرئيسي ، ويأخذ أصغر قيمة من كل صف ويقوم بتبديله مع القطر الثانوي؟
8. أكتب برنامج يطبع العناصر الواقعة تحت القطر الرئيسي فقط؟
9. أكتب برنامج يعمل على إزاحة مصفوفة ثنائية البعد إلى اليمين؟
10. أكتب برنامج يدخل N من الأعداد إلى مصفوفة دون تكرار أي عدد، أي لا تقبل الأعداد المتشابهة؟
11. مصفوفة بها 1000 عنصر مرتبة ترتيباً تصاعدياً ابحث عن عنصر ما باستخدام :
    - مبدأ المقارنة.
    - مفهوم البحث الخطي.
12. أكتب برنامج يعمل على إنشاء المصفوفات التالية:



Chp7

المناهج - الدوال (Function)

في نهاية هذا الفصل سوف تتعلم :

- التعرف على فوائد المناهج (method).
- التعرف على كيفية صنع و استدعاء المناهج وعلى كيفية تمرير البرمترات لها.
- استخدام المناهج ذات التحميل الزائد والتعرف على غموض التحميل الزائد.
- تحديد مجال التغطية للمتحولات المحلية.
- التعرف على مفهوم تجريد المناهج.
- التعرف على كيفية استخدام المناهج الموجودة في الصنف math.
- تطوير البرامج باستخدام فرق – تسد (تقسيم المهام الضخمة الى مهام اصغر).
- التعرف على مفهوم الاستدعاء الذاتي.

7.1 مقدمة INTRODUCTION

لقد برهنت التجربة على أن أفضل طريقة لتطوير وصيانة برنامج كبير ، تتمثل في عملية تجزئته إلى قطع وأجزاء أصغر (modules) يمكن التحكم بها بسهولة أبر من البرنامج الأصلي. تسمى هذه الطريقة بطريقة فرق – تسد . يعرض هذا الفصل عدة إمكانيات للغة Java مساعدة في عملية تطوير وتصميم وتشغيل وصيانة البرامج الكبيرة.

7.2 المناهج Function

- وتسمى الطرق أو الدوال أو التوابع وهى التي من وضع المبرمج ، حيث تجرى كتابة التعليمات المعرفة للمنهج مرة واحدة فقط وتبقى بعدها هذه التعليمات معزولة ومخبئة بالنسبة لبقية المناهج.

- الهدف منها : انه عند تكرار مجموعة من سطور الأوامر أكثر من مرة في مواضع مختلفة فإن أوامر التكرار لن تكون ذات منفعة . ولذلك يتم كتابة هذه السطور منفصلة عن البرنامج الأساسي. حيث تستدعى من خلال استدعائه باسم المنهج المستدعى والمعلومات اللازمة له للقيام بعملة. تشبه هذه الطريقة في البرمجة بطريقة الغدارة ذات التسلسل الهرمي حيث يطلب رب العمل (الذي يقوم باستدعاء المنهج) من عاملة (المنهج) القيام بمهمة محددة تم يعيد الناتج التي حصل عليها.

7.3 مزايا استخدام المناهج

1- عدم تكرار التعليمات داخل البرنامج : حيث يتم إنشاء الدالة مرة واحدة ثم يتم استدعائها أكثر من مرة عند الحاجة إليها .
2- تجعل البرنامج أكثر وضوحاً.
3- عدم ازدحام الدالة الرئيسية للبرنامج بأوامر كثيرة.
4- تقسيم البرنامج إلى أجزاء يمكن اختبارها منفصلة لسرعة تحديد الخلل بالبرنامج
5- توفير المجهود والوقت والتفكير بعمل مكتبة خاصة بك تعيد استخدام الدوال التي استخدمتها في برنامج وذلك في برنامج آخر عند اللزوم .

6- تقسيم العمل بين المبرمجين عن العمل في مشروع جماعي لإنتاج برنامج كبير .

7- تبادل الخبرات بين مطوري البرامج بنشر اجزاء يستخدمها الآخرون في برامجهم .

7.4 الشكل العام للمنهج

وفيه نخبر compiler بشكل المنهج ومحتوياتها كما في الجدول 1-7

جدول 1-7 يشرح الشكل العام للمنهج

الصيغة	التفسير
public	عبارة عن Access modifier و يمكن أن يحل محلها أي من الكلمات التي أعلى المربع ، ومن الممكن ألا تكون موجودة أبداً.
static	من أجل جميع المناهج ، وإخبار المترجم على أن هذه الدالة من نوع ثابت أي أن المترجم يتعرف عليها قبل الدخول إلى الدالة الرئيسية .
int	تعني نوع القيمة التي يرجعها المنهج. فإذا كانت void فهذا يعني أن الوظيفة لا تعيد قيمة، و إنما تقوم بعمل محدد فقط، أما إذا كانت غير ذلك، فإن هذا يعني أن الوظيفة تعيد قيمة محددة، و لابد من استخدام كلمة return داخل الوظيفة في هذه الحالة. و من الممكن أن يكون نوع القيمة التي ترجعها الوظيفة أي واحدة من الكلمات التي في المربع بأعلاها. و نلاحظ إنها قد تكون أيضاً عبارة عن منهج من الناهج .

max	الاسم الذي نعطيه للمنهج ، و يفضّل أن يكون فعلاً، يبدأ بحرف صغير.و معبرا قليلا عما سيفعله المنهج .
(	قوس لاحتواء سلسلة المتغيرات المدخلة للمنهج (parameters list) . التي يتم تحديد نوعها، و إعطائها اسماً لاستخدام ه داخل المنهج ، و يفصل بينها باستخدام الفاصلة. و من الممكن أن لا تكون هناك أي متغيرات مدخلة للمنهج ، و يكون القوس فارغاً.
int	parameters التي ستمرر للدالة وهنا اخترنا int .
)	قوس إغلاق لمجموعة parameters .

{
statements
}
	كود المنهج يوضع بين القوسين.

المتغيرات الموجودة في سلسلة المتغيرات المدخلة (parameters list) تكون صالحة للاستخدام داخل المنهج فقط، ، و لا يمكن استخدام ها خارج الوظيفة أبداً.

7.5 أنواع المناهج

Functions Types

نستطيع تقسيم المناهج إلى نوعين حسب الإعادة . فبعض المناهج التي يتم تحديد نوع القيمة المرجعة، تقوم بإرجاع قيمة عن طريق استخدام الكلمة return . بينما لا ترجع المناهج من نوع void شيئاً والأمثلة التالية تبين ذلك.

```
public String getDate(){
 string
=day+"/"month+"/"year;
  return str;
                    }
```

```
    public void setDate(int d, int m,
srt int y){
      day=d;
      month=m;
      year=y;
     }
```

كما يمكننا تقسيم المناهج حسب وضعية الوظيفة في الفئة إلى نوعين، مناهج خاصة بالفئة، ومناهج خاصة بالعضو. و يتم في النوع الأول كتابة كلمة (static) في توقيع الوظيفة (method signature). و هكذا نكون قد جعلنا هذه الوظيفة هي خاصة بالفئة بشكل عام و ليست خاصة لعضو من الأعضاء. و بإمكان أي عضو استخدام ها من الفئة مباشرة دون الحاجة
لإنشاء عضو من الفئة. و سيأتي شرح ذلك بالتفصيل لاحقاً.

وهذا مثال بسيط بواسطة الدوال يقوم بتربيع عدد ما:

```
1. //برنامج بواسطة منهاج يطبع تربيع عدد ما
2. class Chp7_1 {
3.
4.         static int sqr (int b){ return b*b;}
5.
6.         public static void main(String[] args) {
7.                 System.out.println(sqr(3));
8.                                                 }
9. }
```

فيكون خرج البرنامج كما يلي:

شرح المثال:
يحتوي هذا البرنامج على المنهج main وعلى المنهج sqr حيث المنهج main هو مثل بقية المناهج إلا أنه يستخدم من قبل المفسر Java.
ففي السطر 7 يقوم المحرر بحفظ نقطة استدعاء المنهج في مكدس ليتم تنفيذ النقطة التي بعدها بعد تنفيذ المنهج.
فعند استدعاء المنهج sqr في السطر 4 تم تمرير قيمة بواسطة المتحول b ، وتم إعادة ناتج العملية (b*b) بواسطة التعليمية return إلى منهاج main . حيث تم طباعة مردود المنهج في المنهج الرئيسي كما هو في السطر 7.

يسبب عدم تحديد نمط النتيجة المعادة من قبل المنهج أثناء عملية تعريفه، خطاء قواعدي إذا كان نموذج ذلك المنهج يحدد نمطاً لتلك النتيجة.

تسبب عملية إعادة قيمة للمنهج تم التصريح عنة أنه من النمط void ، خطاء قواعدي.

يسبب وضع فاصلة منقوطة بعد القوس اليميني الذي يغلق قائمة تعريف وسطاء المنهج ، حدوث خطاء قواعدي.

ليس من الخطاء استخدام نفس المتغيرات أو وسطاء عند استدعاء المنهج وعند تعريفة أيضاً . بل على العكس يمكن لذلك أن يساعد على إزالة الكثير من الغموض.

7.6 تمرير القيم إلى المناهج

يبين الجدول 2-7 كيفية تمرير القيم إلى المناهج بالطرق الصحيحة:

جدول 2-7			
الاستدعاءات الصحيحة		الاستدعاءات الخاطئة	
يمكن أن تمرر قيمة صحيحة	Method(30)	هناك زيادة بعدد المتحولات الممررة	Method(myint,10)
يمكن للمتحول يمرر على أنة وسيط	Method(myint)	يجب أن تكون قيمة هذا المتحول صحيحة	Method(1.2)
يمكن أن تمرر عبارة صحيحة	Method(2*2)	نقص في عدد المتحولات	Method(myint)
يمكن لمتحول سلسلة رمزية أن يمرر	Method(mystr,myint)	خطاء في ترتيب المتحولات	Method(myint,mychar)
يمكن لسلسلة رمزية أن تمرر	Method("mystring")		

7.7 فترة حياة المتغيرات (Variable Life Time) داخل المنهج :

وهو الجزء من البرنامج الذي نستطيع من خلاله الوصول إلى المتغير. فبالنسبة لمتغيرات النسخة و الطرق نستطيع الوصول إليها داخل الصنف إي من بداية تعريف الصنف وحتى نهاية تعريفة. أما المتغيرات المحلية فإمكانية الوصول إليها داخل المقطع (block) الذي عرفت به فقط. أما بالنسبة للمتغيرات المحلية على مستوى الطريقة والمعاملا ت فتكون إمكانية الوصول إليها داخل تلك الطريقة فقط.

```
1. // VariableScope
2.  public class Chp7_2{
3.          static int i; //instance variable
```

```
4.
5.        public static void main(String args[]){
6.        int x = 5, y = 6; //local variables
7.        i = 10;
8.        System.out.println("i =  "+ i);
9.        i = method1(x, y);
10.
11.       System.out.println("i = " + i);
12.       i = method2(x, y);
13.
14.       System.out.println("i = " + i);
15.                                                    } //end main
16.
17.  static int method1(int arg11 ,int arg12 ){
18.                 double num11 ,num12;
19.                 for(int counter = 0; counter <= 5; counter++){
20.                 i+= counter;
21.                                                          }
22.                 //end of for counter loop
23.
24.                 return i+arg11+arg12;
25.                                                 } //end method1
26.
27.  static int method2 (int arg21, int arg22){
28.                 int num21, num22, i=0; //local variables
29.                 {
30.                  String s; //local variable
31.                 }
32.                 return i+arg21+arg22;
33.                 } //end method1
34.
35. } //end of class
```

شرح المثال :

من خلال المثال سوف نتعرف على أن المجال المتغير (Variable Scope) يؤثر على المكان الممكن استخدام هذا المتغير فيه.

في السطر (3) تم تعريف المتغير I ليكون مرئي على مستوى الصنف كاملاً، حيث تكون فترة حياة هذا المتغير من بداية تحميل الصنف إلى نهايته، وبما أن هذا الصنف يحتوي على الطريقة main فإنه يعتبر الصنف الرئيسي لتنفيذ البرنامج ، وبذلك تكون فترة حياة المتغير I من بداية البرنامج إلى نهايته.

في السطر 4 تم تعريف المتغيرين x , y كمتغيرات محلية (local variable) يمكن رؤيتها داخل الطريقة main فقط، وفترة حياتها تمتد من بداية الطريقة main إلى نهايتها. في السطر 17 المعاملان arg11 ، arg12 الخاصين بالطريقة method1 فيكونان مرئيان فقط داخل هذه الطريقة، وفترة حياتهما تبدأ من لحظة استدعاء الطريقة ولغاية من هذه الطريقة والخروج منها .

في السطر 18 المتغيران num11,num12 هما متغيران محليان ويكونان مرئيان داخل الطريقة method1 فقط، وتبدأ فترة حياتهما باستدعاء الطريقة وتنتهي بالخروج منها.
في السطر 19 تم تعريف المتغير counter ليكون مرئي داخل جملة الدوران for ففقط، وتمتد فترة حياته هذا المتغير من لحظة الدخول إلى جملة الدوران وتستمر حتى نهاية المقطع (block) الخاص بهذه الجملة.
وفي السطر 28 تم تعريف متغيرات محلية للطريقة method2 ومن هذه المتغيرات متغير أسمة I ، ونلاحظ أن اسم هذا المتغير يتطابق مع اسم المتغير المعرف على مستوى الصنف في السطر 3 ، وهذا التعريف يلغي رؤية المتغير I المعرف على مستوى الصنف داخل هذه الطريقة . وعند استخدام المتغير داخل الطريقة فهذا يعني الرجوع للمتغير المعرف على مستوى الطريقة فقط.
وفي السطر 30 تم تعريف المتغير s على مستوى المقطع (block) الذي يبدأ من السطر 29 وينتهي بالسطر 31 ، وبهذا يكون مجال رؤية هذا المتغير داخل هذا المقطع فقط وفترة حياته تبدأ من بداية المقطع وتنتهي بنهاية المقطع.

فيكون ناتج تنفيذ البرنامج كما يلي:

```
i =  10
i = 36
i = 11
```


تعريف متغير من نوع static داخل الدالة الرئيسية يعتبر خطاء شائع .

فهذا الكود لن ينفذ إطلاقا .

```
1. // Statement expected.
2. class Chp7_3 {
3.         public static void main(String args[]){
4.
5.                 static int i=0;//Error
6.                 System.out.println(i);
7.                                         }
8. }
```


من الأخطاء الشائعة استخدام قيمة متغير عادي إلى متغير من نوع static .

كهذا الكود خاطئ:

```
1. // Can't make a static reference to nonstatic variable b in class Chp7_4.
2. class Chp7_4 {
3.
4.          int b=10;
5.          static int i=b;//Error
6.
7.          public static void main(String args[]){
8.
9.                    System.out.println(i);
10.                                                  }
11. }
```

من الأخطاء الشائعة إعادة قيمة من نوع عادي بواسطة دالة من نوع static مثل هذا المثال:

```
1. // Can't make a static reference to nonstatic variable a in class Chp7_5.
2. class Chp7_5 {
3.
4.          int a=10;
5.          static int b(){
6.                              return a;//Error
7.                    }
8.
9.          public static void main(String args[]){
10.                   Chp5_5 c =new Chp5_5();
11.                   System.out.println(c.b());
12.                                                  }
13. }
```

شرح المثال:
لتصحيح هذا المثال توجد طرقتين :
الأولى : جعل a من نوع static int a .
الثانية : إبعاد جملة static من الدالة .

عدم تعريف المنهج من نوع static ومحاولة الوصول إليها مباشرة دون أطلاق الصنف.

7.8 أنواع تمرير البيانات

7.8.1 التمرير بالقيمة

عندما تستدعي منهجاً ما مع بارامتراته الموافقة فإنه يتم أخذ نسخة من البارامترات الفعلية وتمريرها إلى المنهج. فيطلق على هذه الطريقة التمرير بالقيمة (pass by value) حيث لا تتأثر البارامتر الفعلي الموجود خارج المنهج بالتغيرات التي تحدث على البارامتر الشكلي الموجود داخل المنهج.ولتوضيح أكثر فلنتتبع البرنامج التالي:

```
1. // برنامج بواسطة المنهج يمرر قية إلى المنهج
2. class Chp7_6{
3.
4.  static void  fun (int i){
5.                              i++;
6.                              System.out.println(i);
7.                             }
8.
9.  public static void main(String args[]){
10.       int i=10;
11.       System.out.println(i);
12.       fun(i);
13.       System.out.println(i);
14.                                        }
15. }
```

فيكون ناتج البرنامج كما في الشكل التالي

في السطر 10 أعطينا المتحول I القيمة 10 ، وتم طباعة قيمته في السطر 11، في السطر 12 تم استدعاء المنهج fun وإرسال لها قيمة المتحول I .
في المنهج fun السطر 5 تم تزيد المتحول المستقبل من المنهج وطباعته.
في السطر13 تم طباعة قيمة المتحول وكان ناتج الطباعة نفس ما نتج في السطر 11، رغم أننا غيرنا قيمة المتحول بداخل المنهج .
ما نلاحظه أن أثناء تغير قيمة المتحول بداخل المنهج لا يؤثر إطلاقاً على المتغير بداخل المنهج الرئيسي main . أي أخذ المنهج نسخة من قيمة المتحول.

7.8.2 التمرير بالمرجعية (Pass-By-Reference)

التمرير باستخدام العنوان (Pass-By-Reference): وفي هذا النوع يتم إرسال عنوان المتغير (المعامل الفعلي) إلى المعامل الشكلي المقابل له في الطريقة (Method)، ليصبح المعامل الشكلي والمعامل الفعلي يؤشران إلى نفس العنوان في الذاكرة الرئيسية .

وفي هذا النوع أي تغيير يتم على المعامل الشكلي يؤثر وفي نفس الوقت على المعامل الفعلي الذي تم أرسالة للطريقة عند الاستدعاء . وهذا النوع من تمرير البيانات يتم تطبيقه بشكل تلقائي عندما تكون المعاملات الفعلية من أحد الكائنات (Objects) مثل: المصفوفات Array, String .... الخ.

```
1. // Passing_Parameters
2.  class Chp7_7{
3.          public static void main(String args[]){
4.                  int x;
5.                  int a[] = {1, 2, 3, 4};
6.                  x = a[1];
7.                  System.out.println("The value of x before change is" + x) ;
8.                  System.out.println("The value of a elements before change is: ");
9.                  printArray(a);
10.                 change(a, x);
11.                 System.out.println("The value of x after change is" + x);
12.                 System.out.println("The value of a elements after change is: ");
13.                 printArray(a);
14.                                                 } //end of main
15.         static void change(int b[], int i){
16.                 i *= 2 ;
17.                 for (int index=0; index < b.length; index++)
18.                 b[index]*= 2;
19.                                         } //end of method change
20.         static void printArray(int c[]){
21.                 for (int index=0; index < c.length; index++)
22.                 System.out.print(c[index] + "\t");
23.                 System.out.println();
24.                                     } //end of method print
25. } //end of class Passing_Parametres
```

فيكون ناتج تنفيذ البرنامج

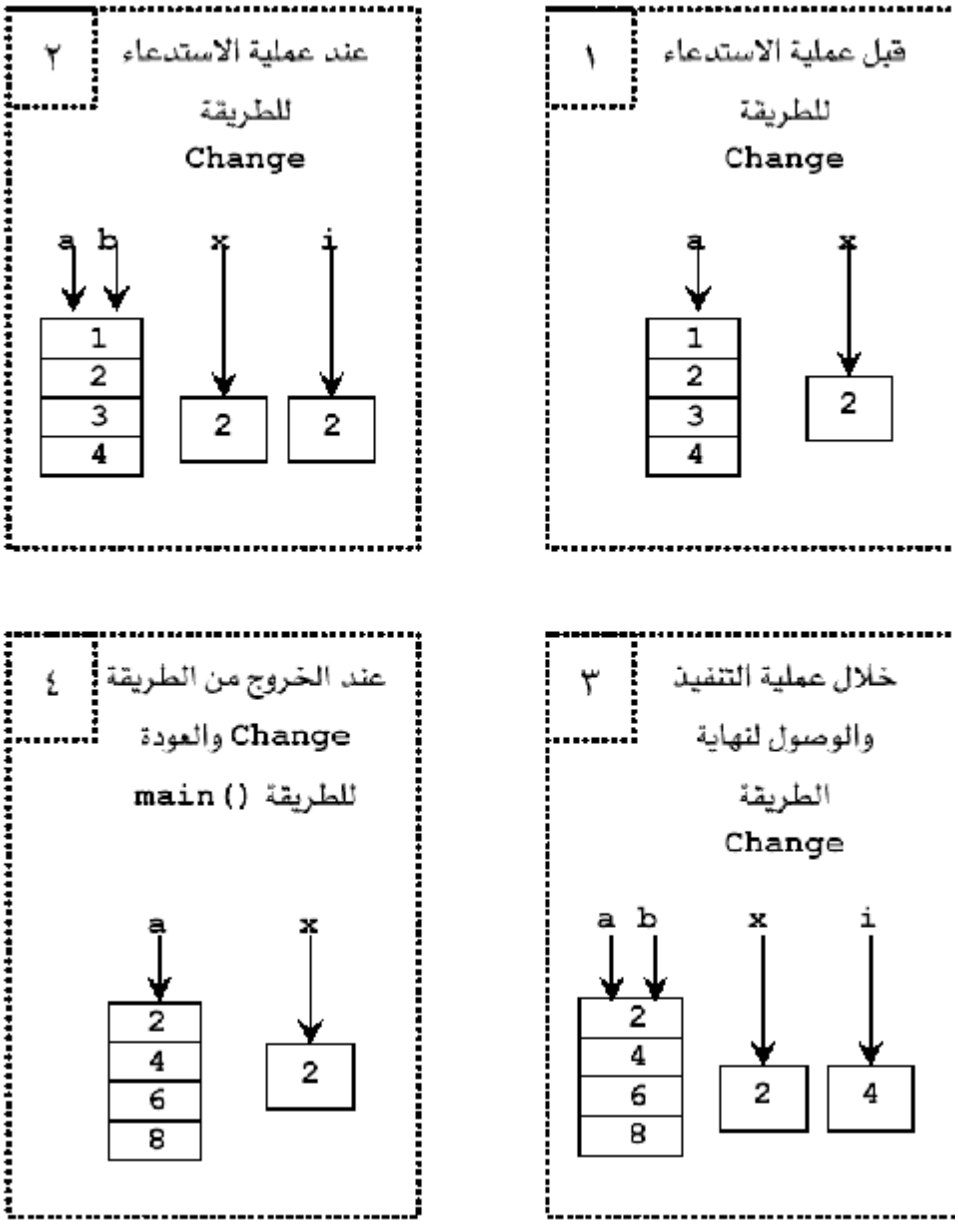
```
The value of x before change is2
The value of a elements before change is:
1       2       3       4
The value of x after change is2
The value of a elements after change is:
2       4       6       8
```

شرح المثال:

هذا المثال يبين كيفية الاستدعاء باستخدام القيمة والاستدعاء باستخدام العنوان . في السطر رقم 9 تمت عملية استدعاء للطريقة Change بإرسال x (متغير يشير إلى قيمة من نوع int) و a (متغير يشير إلى مصفوفة من نوع int) . وكما ذكرنا سابقاً فإن عملية تمرير المتغيرات التي تشير إلى أنواع بيانات بدائية (primitive Data Type) تكون باستخدام

القيمة حيث سيتم إرسال نسخة من قيمة المعامل الفعلي x إلى المعامل الشكلي I الموجود في تعريف هذه الطريقة. وبما أنه قد تم إرسال نسخة من قيمه المعامل الفعلي إلى المعامل الشكلي الثاني a والذي يرسل عند استدعاء الطريقة Change فهو عبارة عن مصفوفة. وكما نعرف بأن المصفوفة هي عبارة عن كائن إذن سيتم إرسال عنوان هذه المصفوفة إلى المعامل الشكلي الثاني b، أي سيكون المعامل الفعلي a والمعامل الشكلي b يشيران إلى نفس الموقع (المصفوفة) في الذاكرة.

والشكل 1-7 يوضح الفرق بين عملية التمرير باستخدام القيمة وعملي التمرير باستخدام العنوان وذلك عند استدعاء الطريقة change الموجود في السطر 9 في المثال السابق.


١
قبل عملية الاستدعاء
للطريقة
Change
a
x
1
2
3
4
2
٢
عند عملية الاستدعاء
للطريقة
Change
a b
x
i
1
2
3
4
2
2
٣
خلال عملية التنفيذ
والوصول لنهاية
الطريقة
Change
a b
x
i
2
4
6
8
2
4
٤
عند الخروج من الطريقة
Change والعودة
للطريقة main ()
a
x
2
4
6
8
2

شكل 7-1


كتابة منهج بداخل منهج ، ينتج عن ذلك خطاء.


من الممكن استخدام عبارة الإعادة (return) في المنهج الذي لا يملك نمط إعادة وذلك من أجل إنها المنهج والعودة إلى مستدعي المنهج، وتعتبر هذه الطريقة مفيدة من أجل التحايل على تدفق التحكم المشروط وهذا المثال يبين ذلك.

```
1. // إيقاف تنفيذ الدالة والخروج منها
2. class Chp7_8 {
3.         //function static
4.         static void b(){
5.                                         if(true){
6.                                                         System.out.println("1");
7.                                                         //Stop function
8.                                                         return;
9.                                                 }
10.                                        System.out.println("2");
11.                         }
12.        public static void main(String args[]){
13.                         b();
14.                         System.out.println("3");
15.                                                 }
16. }
```

وهذا ناتج التنفيذ

```
1
3
```

7.9 الاستدعاء الذاتي – العودية Recursion

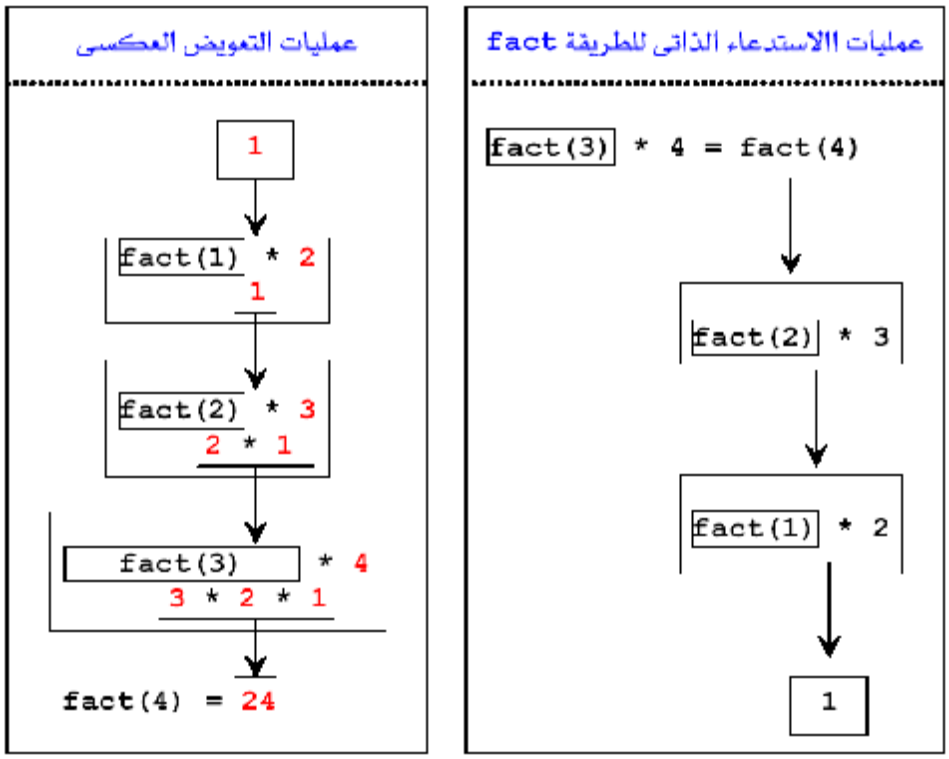
والمقصود بالاستدعاء الذاتي هو أن تقوم الطريقة باستدعاء نفسها بنفسها، فهنالك الكثير من المسائل التي يمكن حلها باستخدام الاستدعاء الذاتي وبهذه الطريقة يمكن توفير الكثير من جمل التكرار فنستطيع الاستعاضة عن جمل التكرار بعملية استدعاء الطريقة لنفسها. فمثلاً لإيجاد المضروب (factorial) لعدد معين يمكن كتابة البرنامج على الشكل التالي:

```
1. // factorial
2.  import javax.swing.JOptionPane;
3.  class Chp7_9 {
4.          public static void main (String args[ ]){
5.                  String snum1;
```

```
6.                  int num1, fac_of_num1;
7.                  snum1 = JOptionPane.showInputDialog("Enter num1:") ;
8.                  num1 = Integer.parseInt(snum1);
9.                  fac_of_num1 = fact(num1);
10.                 JOptionPane.showMessageDialog(null, num1 + "! = " +fac_of_num1);
11.                                         } //end of main
12.
13.       static int fact(int n){
14.                 if (n == 0 || n ==1)
15.                         return 1;
16.                         else
17.                         return n * fact(n-1);
18.                           } //end of fact method
19.  } //end of class Chp7_9
```

شرح المثال :

في السطر 7 يتم إدخال الرقم المراد حساب قيمة المضروب (Factorial) له. ثم في السطر 8 يتم استدعاء الطريقة Fact والتي تقوم بعملية حساب قيمة المضروب وتعيدها ليتم تخزينها في المتغير fac_of_num1 والآن لنرى ماذا سيحدث عند استدعاء الطريقة fact: في السطر 14 تتم عملية السؤال عن قيمة الرقم المرسل فإذا كان مساوياً للواحد أو الصفر (حسب التعريف الرياضي لعملية إيجاد المضروب) تتوقف هذه الطريقة وترجع واحد. ويعتبر هذا الشرط هو شرط التوقف الرياضي لعملية إيجاد المضروب) تتوقف هذه الطريقة وترجع واحد. ويعتبر هذا الشرط هو شرط التوقف الوحيد للاستدعاء الذاتي لهذه الطريقة ، حيث أنه لابد من وجود شرط توقف داخل طرق الاستدعاء الذاتي وإلا استمرت عملية الاستدعاء الذاتي إلى مالا نهاية . وفي السطر 17 تتم إرجاع العدد مضروباً بعملية استدعاء أخرى لنفس الطريقة ولكن هذه المرة بإرسال العدد السابق مضروباً بعملية استدعاء أخرى لنفس الطريقة ولكن هذه المرة بإرسال العدد السابق مطروحاً منه واحد (n-1) ، وتستمر العملية حتى يتحقق شرط الخروج من الطريقة وتوقيف عملية الاستدعاء الذاتي وتبدأ الآن عملية التعويض العكسي للقيم المرجعة من عمليات الاستدعاء . والشكل 2-7 يوضح عمليات الاستدعاء وعملية التعويض العكسي عند إدخال الرقم 4.



شكل 2-7

7.9.1 انسياب الاستدعاء التعاودي The Flow of a Recursive Call

لقد لاحظنا في البرنامج السابق العلاقة بين استدعاء الروتين وإكماله ، حيث إن أخر روتين يستدعى هو أول روتين ينهى، وأول روتين يستدعى يكون أخر روتين ينهى .وأن وسطاء المناهج تمرر بقيمها أي أن كل إجراء يحتفظ بقيمته الخاصة به.

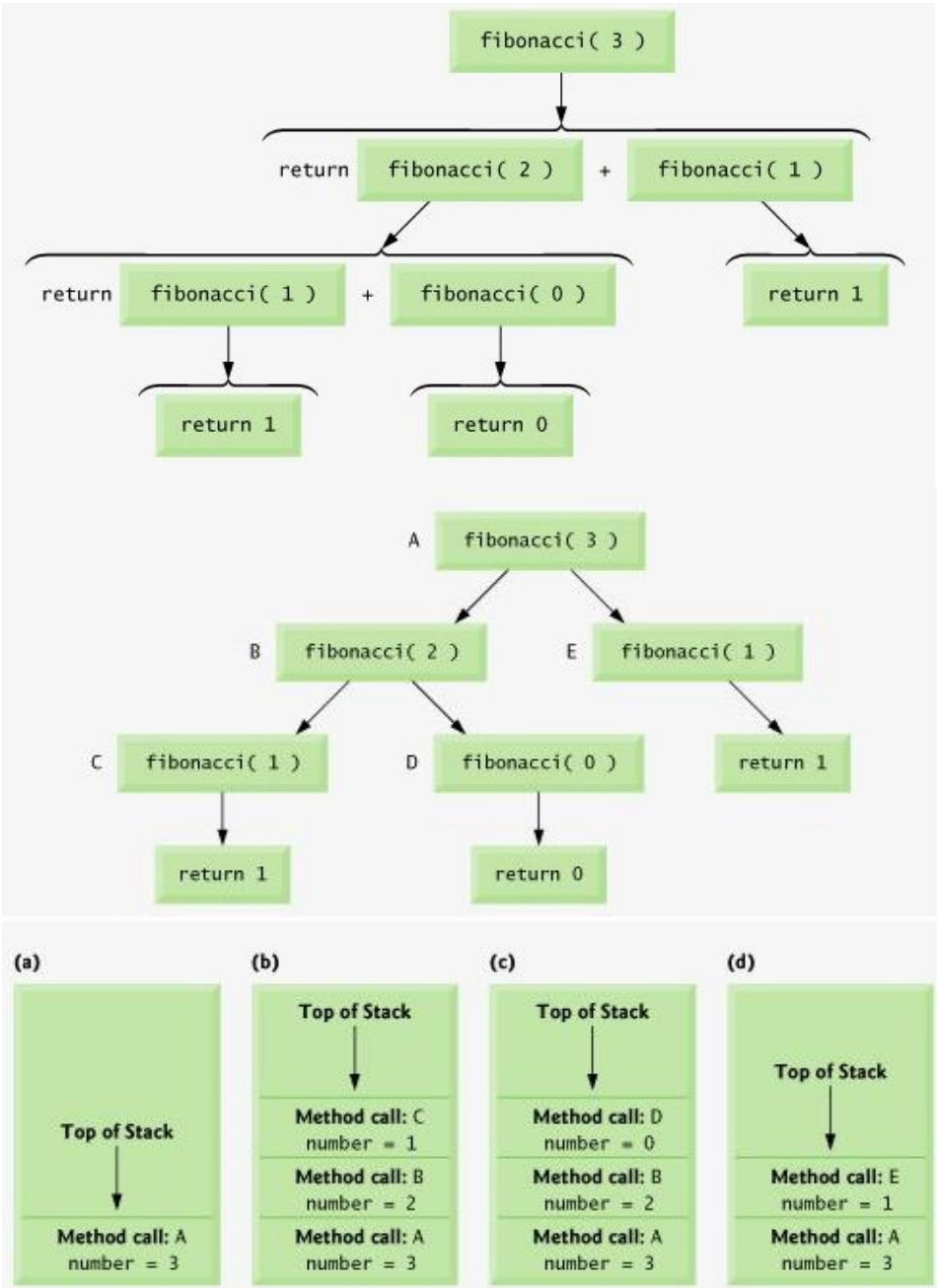
وهذا مثال على سلسلة(Fibonacci):
تبدأ سلاسل Fibonacci التالية

0,1,1,2,3,5,8,13,21,……

من القيمتين 1,0 وتتمتع بالخاصية التالية : يساوي كل عدد من أعدادها مجموع العددين السابقين له. وسميت هذه السلسلة بهذا الاسم باسم مكتشفها Leonardo Fibonacci (عالم رياضي من القرون الوسطى ، قام بإيجاد هذه السلسلة لحل و نمذجه مشكلة توالد الأرانب ، ويمكن تطبيق هذه السلسلة في تحسين العمليات العددية وفي مجالات عديدة أخرى).

```
1. // Recursive fibonacci method.
2.
3.  public class Chp7_10
4.  {
5.     // recursive declaration of method fibonacci
6.     static long fibonacci( long number )
7.     {
8.        if ( ( number == 0 ) || ( number == 1 ) ) // base cases
9.           return number;
10.       else // recursion step
11.          return fibonacci( number - 1 ) + fibonacci( number - 2 );
12.    } // end method fibonacci
13.
14.    static void displayFibonacci()
15.    {
16.       for ( int counter = 0; counter <= 10; counter++ )
17.          System.out.println( "Fibonacci of "+ counter + "\t"+fibonacci( counter ) );
18.    } // end method displayFibonacci
19.
20.   public static void main(String[] args) {
21.    displayFibonacci();
22.                                  }
23.  }
```

يقوم هذا البرنامج بحساب سلسلة Fibonacci عودياً باستخدام المنهج Fibonacci . لاحظ أن سلسلة Fibonacci تميل إلى التزايد بصورة سريعة . لذلك تم استخدام متحول من نوع long . لا يمثل الاستدعاء التابع Fibonacci داخل المنهج الرئيسي main استدعاءاً عودياً لكن الاستدعاءات التالية له هي استدعاءات عوديه . ففي كل مرة يستدعى التابع Fibonacci ، يتم مباشرة التحقق فيما إذا كانت قيمة number أكبر من 1 يتولد استدعائين يتناول كل منها مسألة أبسط من المسألة التي يتناولها الاستدعاء الأصلي للمنهج Fibonacci . ويبين الشكل 3-7 سير البرنامج.


fibonacci(3)
return
fibonacci(2)
+
fibonacci(1)
return
fibonacci(1)
+
fibonacci(0)
return 1
return 1
return 0
A
fibonacci(3)
B
fibonacci(2)
E
fibonacci(1)
C
fibonacci(1)
D
fibonacci(0)
return 1
return 1
return 0
(a)
Top of Stack
Method call: A
number = 3
(b)
Top of Stack
Method call: C
number = 1
Method call: B
number = 2
Method call: A
number = 3
(c)
Top of Stack
Method call: D
number = 0
Method call: B
number = 2
Method call: A
number = 3
(d)
Top of Stack
Method call: E
number = 1
Method call: A
number = 3

شكل 7-3

يثير الشكل 3-7 نقاطاً هامة لها علاقة بترتيب تقييم معاملات العمليات المؤلفة لتعليمات الاستدعاء الذاتي. يظهر لدينا أنه من أجل حساب Fibonacci(3) يجب القيام باستدعائين Fibonacci(2)، Fibonacci(1) .
لكن ما هو الترتيب الذي يتم وفقه القيام بالاستدعائين السابقين ؟
يفترض معظم المبرمجين أن حساب قيم المعاملات يتم وفق ترتيب متسلسل من اليسار إلى اليمين.
لكن في الواقع ، لم تحدد لغة Java الترتيب الواجب اتباعية لحساب قيم المعاملات لمعظم العمليات (بما في ذلك العملية +) . لذلك لا يمكن للمبرمج أن يفترض وجود أي ترتيب لتنفيذ الاستدعاءات الذاتية السابقة حيث يمكن أن ينفذ الاستدعاء Fibonacci(2) أولاً ثم الاستدعاء Fibonacci(1) ثانياً . لكن البرنامج السابق يقوم بإعادة النتيجة الصحيحة في كلا الحالتين مثله مثل غيرة من البرامج العودية المماثلة .

7.9.2 مقارنة بين الاستدعاء الذاتي والتكرار

درسنا في الفقرات السابقة مثالين يمكن كتابة برامج لهما سواء كانت عوديه أم تكرارية. سوف نحاول في هذه الفقرة المقارنة بين الأسلوبين وسوف نناقش لماذا يقوم المبرمج أحياناً باختيار أحداهما دون الأخر في بعض الحالات.
يعتمد الأسلوبان العودي والتكراري على بني التحكم : فيستخدم التكرار بني التكرار . أما الأسلوب العودي فيعتمد لتحقيق التكرار عندما يصبح شرط استمرار التكرار غير صحيحاً .
أما العودية فتنتهي عند الوصول إلى الحالة أو الحالات الابتدائية base case. تتشابه طريقة التكرار ذات العداد مع الطريقة العودية من ناحية أسلوب التوقف : فالتكرار يعتمد على تغير قيمة عداد وصولاً إلى قيمة تجعل من شرط استمرار التكرار غير محققاً ، أما العودية فإنها تعتمد على تناول مسائل أصغر وأبسط من المسألة الأصلية حتى تصل إلى الحالة الابتدائية التي نعرف حلها . لكن يمكن لكلا الأسلوبين الوقوع في حلقات لا نهائية : إذا بقي شرط استمرار التكرار صحيحاً دائما فأن عملية التكرار لن تتوقف ، أما إذا لم تنجح الخطوة العودية في تقسيم المسألة المتناولة إلى مسائل جزئية أصغر منها فإن عمليه التقارب نحو الحالة الابتدائية لن تحدث مما يسبب الدخول في حلقة لإنهائية.
تعاني الطريقة العودية من عدة مساوئ : فهي تعتمد على الاستدعاءات العودية أي إنها تعاني من ضياع الوقت والذاكرة الناتج عن استدعاء التوابع يسبب كل استدعاء عودي نسخ التابع (متحولاته فقط في واقع الأمر) مما يسبب استهلاكاً كبيراً للذاكرة ، فتعتمد على الإضافة التراكمية (overhead) ، حيث في كل مرة يستدعى فيها البرنامج المنهج التعاودي يجب على النظام حجز مساحة من أجل جميع البارامترات والمتحولات المحلية التابعة لهذا المنهج .في حين لا تفعل الطريقة التكرارية ذلك لإنها تتجاوز مشكلتي ضياع الوقت واستهلاك الذاكرة .

مما يدفعنا للسؤال لماذا يتم أحيانا اختيار الأسلوب العودي لحل بعض المسائل؟

وللإجابة :

- الحل العودي قادر على التعبير عن المسألة ونتائجها على شكل برنامج سهل الفهم والفحص وصغير.
- عدم وضوح الحل التكراري للمسألة المدروسة.

7.10 التحميل الزائد للمناهج Method Overloading

تسمح لغة Java بتعريف عدة مناهج لهما نفس الاسم بشرط أن تتعامل مع وسطاء لها أنماط مختلفة . تسمى هذه الإمكانية التحميل الزائد للتوابع (Method Overloading) . فهي عبارة عن دالتين أو أكثر تحمل نفس الاسم إلا إنها تختلف في عدد الوسائط أو الأنماط.

أي مثلاً أنت عندك ثلاثة مسدسات ربع ونص وكامل فكل مسدس لدية رصاص خاص به وكلهم يحملان نفس الاسم ويميز الأول عن الثاني عن الثالث بالرصاص وحجمهم فقط وعند تسليمك رصاص من نوع صغير أنت تلقائياً ستفهم أن هذا الرصاص خاص بالمسدس الربع وستقوم بشحن الرصاص بالمسدس . هذ ا هو المترجم عندما يكون في البرنامج عدة دوال بنفس الاسم الأولى تأخذ من نوع int والثانية تأخذ من نوع char والثالثة من نوع String فأنة عند إرسال قيمة من نوع int فان المترجم سيستخدم الدالة التي من نفس نوع القيمة المرسلة .

يساعد التحميل الزائد للمناهج على أذ جاز مهام متقاربة وعلى جعل البرنامج أكثر قابلية للقراءة والفهم.

يستخدم الشكل ــ تحميل زائد للمنهج sum من أجل عدة عمليات حسابية .

```
1. //برنامج التحميل الزائد للمناهج
2. class Chp7_11{
3.  public static void main(String args[]){
4.          sum();
5.          sum(100,3);
6.          System.out.println("sum= " + sum(8.5, 4));
7.          System.out.println("sum= " + sum(10, 4.2));
8.          System.out.println("sum= " + sum(8, 9, 4));
9.                                                  }
10.         static void sum(){int num1 = 10, num2 = 5;
11.                                 System.out.println("sum = " + (num1 + num2));
12.                                }
13.         static void sum(int num1, int num2){
```

```
14.                                 System.out.println("sum = " + (num1 + num2));
15.                                 }
16.     static double sum(double num1, int num2){return (double)(num1 + num2);}
17.
18.     static double sum(int num2 ,double num1){return (double)(num1 + num2);}
19.
20.     static int sum(int num1, int num2, int num3){return num1+num2+num3;}
21. }
```

فيكون خرج البرنامج

```
sum = 15
sum = 103
sum= 12.5
sum= 14.2
sum= 21
```

شرح المثال:

يوضح هذا المثال مفهوم التحميل الزائد . نلاحظ في هذا المثال وجود 5 طرق تحمل نفس الاسم لكن لابد لهذه الطرق أن تختلف عن بعضها في واحد أو أكثر من مكونات التواقيع (يكون الاختلاف بنوع البارامترات ، عددهم ، ترتيبهم) ففي السطر 10 تم تعريف المنهج حيث أنة ليس لها مع املات شكلية ولا ترجع قيمة من أي نوع. وهكذا لباقي المناهج الأربع تختلف الإحدى عن الأخرى.

إنشاء مناهج بنفس الاسم لها نفي البارامترات ولكن تختلف عن بعضها البعض بأنماط النتائج المعادة.

التحميل الزائد يجعل البرنامج واضحاً وأكثر قابلية للقراءة ويجب إعطاء نفس الاسم للدوال التي تقوم بانجاز عملها .

لا يمكن الحصول على دوال ذات تحميل زائد من خلال نمط القيمة المرتجعة من الدالة فهذا خطاء شائعاً.

في بعض الأحيان قد يحصل توافق بالبارامترات عند الدوال ذات التحميل الزائد فيفشل المترجم اختيار الدالة المناسبة بما معنى انه غموض فهذا المثال خاطئ:

```
1. //برنامج غموض التحميل الزائد
```

```
2. class Chp7_12{
3.
4.  static void  fun (int i){
5.                                i++;
6.                                System.out.println(i);
7.                               }
8.
9.  public static void main(String args[]){
10.        int i=10;
11.        System.out.println(i);
12.        fun(i);
13.        System.out.println(i);
14.                                                  }
15. }
```

يمكن اختيار max(int num1,double num2) او max(double num1, int num2) ليتوافق مع max(3.1,10) حيث لا يوجد منهم محدداً أكثر من الأخر وبالتالي هذا الاستدعاء غامض .

7.11 مكتبة التوابع الرياضية

تسمح توابع المكتبة الرياضية للمبرمج بالقيام بالكثير من الحسابات الرياضية الشائعة. سوف نقوم في هذه الفقرة باستخدام عددا من التوابع الرياضية لتوضيح المفاهيم المتعلقة بالتوابع وطريقة استخدام ها .

7.11.1 الصنف Math

يتم استدعاء التوابع عادة بأن نكتب المكتبة متبوعاً بنقطة متبوعاً أسمها متبوعاً بقوس يساري مفتوح ووسيط (أو قائمة من الوسطاء يفصل بينهما بواسطة فواصل) ثم نضع أخيراً قوس يميني .
الهيكل العام لهذه التوابع:

```
ClassName.methodName( arguments )
```

على سبيل المثال:
يستطيع مبرمج في حساب وطباعة قيمة الجدر التربيعي للقيمة 9000 أن يكتب التعليمة التالية:

```
System.out.println( Math.sqrt( 900.0 ) );
```

عند تنفيذ هذه التعليمة ، يجري استدعاء التابع الرياضي sqrt لحساب الجذر التربيعي للرقم المعطى بين القوسين .
يمكن لوسطاء التابع أن تكون ثوابت، متحولات ، أو حتى تعابير. فإذا كان لدينا مثلاً c = 13.0, d = 3.0 , f = 4.0 فالتعليمة:

```
System.out.println( Math.sqrt( c + d * f ) );
```

تقوم بحساب وطباعة الجذر التربيعي لناتج العملية التي بداخل الأقواس.
يلخص الجدول 3-7 بعضاً من توابع المكتبة الرياضية حيث أن المتحولين y , x هما من النمط double.

جدول 3-7

التابع	وصفه	مثال
abs(x)	تابع القيمة المطلقة	abs(23.7) is 23.7 abs(0.0) is 0.0 abs(-23.7) is 23.7
ceil(x)	تقريب إلى أصغر عدد صحيح	ceil(9.2) is 10.0 ceil(-9.8) is -9.0
cos(x)	تابع التجيب (قيمة مقدرة بالراديان)	cos(0.0) is 1.0
exp(x)	التابع الأسي	exp(1.0) is 2.71828 exp(2.0) is 7.38906
floor(x)	تابع التقريب إلى أكبر عدد صحيح أصغر من x	floor(9.2) is 9.0 floor(-9.8) is -10.0
log(x)	تابع اللغوريتم الطبيعي لـ x ذو القاعدة	log(Math.E) is 1.0 log(Math.E * Math.E) is 2.0
max(x, y)	تابع إيجاد القيمة الكبرى بين x ,y	max(2.3, 12.7) is 12.7 max(-2.3, -12.7) is -2.3
min(x, y)	تابع إيجاد القيمة الصغرى بين x ,y	min(2.3, 12.7) is 2.3 min(-2.3, -12.7) is -

جدول 3-7		
التابع	وصفه	مثال
		12.7
pow(x, y)	X مرفوع إلى القوة y (i.e., x^y)	pow(2.0, 7.0) is 128.0 pow(9.0, 0.5) is 3.0
sin(x)	تابع الجيب لـ x (x قيمة مقدرة بالراديان)	sin(0.0) is 0.0
sqrt(x)	تابع الجذر التربيعي	sqrt(900.0) is 30.0
tan(x)	تابع الظل لـ x (x قيمة مقدرة بالراديان)	tan(0.0) is 0.
random	تابع توليد أعداد بفاصلة عائمة من النمط double أكبر أو يساوي 0.0 و أصغر من 1.0	(0< Math.random()<1.0)

تمارين الفصل

1. أفترض أن X=-3 , Y=4 فما قيمة كل من
   - ABS(X+Y)
   - LOG(X)
   - SQRT(X-Y)
   - SIN(X)
3. أكتب منهج يقوم بإيجاد تكعيب عدداً ما ؟
4. أكتب منهج يقوم بقراءة عناصر مصفوفة وتعيد لنا مجموع العنار السالبة ؟
5. وضح باستخدام الأمثلة ما المقصود بما يلي:
   - التمرير باستخدام القيمة Pass-By-Value.
   - التمرير باستخدام العنوان Pass-By-Reference.
   - Method Overloading.
6. بواسطة الأستدعاء الذاتي أكتب المناهج التي تقوم بإيجاد:
   - القاسم المشترك .
   - المضاعف المشترك.
   - عوامل العدد.
7. أكتب ناتج تنفيذ البرامج التالية:

A)
```
public class checkupper {
 public static void main (String args[]){
  char c1 = 'f', c2 = 'T';
  System.out.println("Is "+c1 +" in uppercase ? " +
isUpperCase(c1));
  System.out.println("Is "+c2 +" in uppercase ? " +
isUpperCase(c2));
 }
 static boolean isUpperCase(char testChar) {
  return ((testChar>='A') && (testChar<='Z'));
  }
}
```

B)
```
public class validateAddress {
 public static void main (String args[]){
  String My_email = "java_doc@java.net";
  if (validate(My_email) == true)
```

```
   System.out.println("this a valid email address");
  else
   System.out.println("this an invalid email
address");
 }
 static boolean validate(String email) {
  String name;
  String domain;
int index;
  if (( index = email.indexOf( '@')) == -1) {
   return false;
  }
  name = email. substring(0, index);
  domain=email.substring(index+1,email.length());
  System.out.println(" Name: " + name);
  System.out.println(" Domain: " + domain);
  return true;
 }
}

C)
public class primenumbers{
 public static void main(String [] args) {
  System.out.println("The Prime numbers between 1 and
100 are");
  for (int i = 0; i < 100; i++)
   if (isPrime(i))
    System. out. print(i + " ");
 }
 static boolean isPrime(int test) {
  if (test < 2)
   return false;
  if (test == 2)
   return true;
  for (int i = 2; i < test; i++)
if (( test % i) == 0)
        return false;
  return true;
 }
}
```

D)

```
public class SwapArray{
 public static void main(String [] args) {
  int values[]={1, 2, 3, 4, 5, 6, 7, 8};
  System.out.println("values before swap");
  printArray(values);
  swap(values);
  System.out.println ("values after swap");
  printArray(values);
 }
 static void swap(int a[]){
  int length = a.length, temp;
  for (int i = 0; i <= (length/2); i++){
   temp = a[ length - i - 1];
   a[length - i - 1] = a[ i];
   a[ i] = temp;
  }
}
 static void printArray(int a[]){
  for (int i =0;i<a.length;i++){
   System.out.print (a[i]+"\t");
  }
System.out.println( );
 }
}
```

Chp8

النصوص

في نهاية هذا الفصل سوف تتعلم :

- استخدام الصنف String لمعالجة السلاسل.
- استخدام الصنف Character لمعالجة الحروف.
- استخدام توابع الخاصة بالسلاسل.

8.1 مقدمة INTRODUCTION

سوف نتعرف في هذا الفصل بعضاً من توابع المكتبة المعيارية التي تستخدم للتعامل مع سلاسل الحروف . تشكل التقنيات التي سوف نقدمها البداية التي لا بد منها لبناء محررات نصوص text editor ومعالجات لها word processors وغيرها من برامج الإخراج الطباعي وبرامج معالجة النصوص.

8.2 أساسيات الحروف وسلاسلها

تشكل الحروف البنية الأساسية المؤلفة لكتل البرامج المكتوبة بلغة Java يتألف كل برنامج من سلسلة من الحروف التي يشكل تجميعها مع بعضها البعض نصاً ذو معنى ويمكن تفسيره بواسطة الحاسب على أنة سلسلة من التعليمات اللازمة لإتمام مهمة معينة. وفي معظم لغات البرمجة تكون السلاسل على شكل مصفوفة من الرموز characters ، ولكن في Java تكون السلاسل الرمزية على شكل كائن منفصل من الصنف String .

يمكن للبرنامج أن يتضمن حروف ثابتة التي هي عبارة عن القيم الصحيحة المكافئة لتمثيل أي حرف منها بين فتحتين (') منفردتين . على سبيل المثال ، تمثل 'z' القيمة الصحيحة لـ z (القيمة 22 حسب جدول الحروف ASCII الملحق C) و '\n' القيمة الصحيحة للحرف سطر جديد أي القيمة 10 جدول الحروف ASCII .

تتألف سلاسل الحروف من مجموعة من الحروف التي تعامل على إنها وحدة وهي قد تتضمن حروفاً عادية أرقاماً وبعض من الحروف الخاصة special characters مثل (, : , - , + !.... $, &) وغيرها . تكتب سلاسل الحروف الثابتة في لغة Java بين إشارتي (") على الشكل التالي:

"Ammar aldopaee" (اسم)
"00967711850000" (رقم هاتف)

8.3 الخواص البنيوية للسلاسل الرمزية في لغة Java:

لا بد أولاً من تذكير الأخوة المبرمجين القادمين إلى Java من بعض اللغات الأخرى (وخاصة C++) ، أن نوع السلاسل الرمزية String يُشكل أحد أصناف لغة Java. مما يعني أن هذا الصنف وكسائر الأصناف الأخرى يمتلك مجموعة متنوعة من الطرق التي تحدد سلوكه.

هذا يعني أن تلك السلاسل ليست مجرد مصفوفة من الرموز كما هي الحال في بعض اللغات الأخرى كلغة C++.

أولى الخواص التي يمكن الحديث عنها في بنية الكائنات من الصنف String ، أنها ثابتة وغير قابلة للتعديل بعد إتمام إنشائها.

فمثلاً عند كتابة العبارة

```
String S = "Hello" + YourName;
```

في أحد برامج اللغة، فإن المتحول S يشير إلى الكائن المنشأ لاحتضان السلسلة:

```
"Hello" + YourName
```

(حيث يشير المتحول YourName إلى سلسلة رمزية جزئية). وبعد إنشاء ذلك الكائن يصبح من غير الممكن تعديل السلسلة التي يضمها ضمن أي جزءٍ من أجزاء البرنامج بشكلٍ عام.

يمكن استخدام المتحول S للإشارة إلى كائنٍ آخر أو لسلسلة أخرى حتى ولو كانت تلك السلسلة مشتقة من السلسلة الأصلية "Hello" + YourName كما في العبارة التالية:

```
S = S.trim( );
```

ولكن ماذا عن بقية المناهج، كالمنهج ()toUpperCase التي تؤدي إلى تغيير حالة الرموز؟

حتى هذه المناهج، لا تؤدي إلى تعديل السلسلة الرمزية. حيث تقوم هذه المناهج بإنشاء كائنات جديدة من السلاسل لاحتضان السلاسل الجديدة بعد إجراء التعديلات المطلوبة عليها ضمن هذه المناهج.

8.4 الصنف من نوع String

ينمذج الصنف java.lang.String سلسلة من المحارف كسلسلة نصية. حيث يملك هذا الصنف أحد عشر بانياً وأكثر من أربعين منهجاً من أجل فحص المحارف المنفصلة في السلسلة ومقارنة السلاسل النصية والبحث في السلاسل النصية الفرعية واستخراج السلاسل النصية الفرعية وصنع نسخة من السلسلة النصية ثم تحويل جميع المحارف الموجودة في هذه السلاسل النصية إلى شكلها الكبير أو إلى شكلها الصغير.

كما يتيح الصنف String أيضاً إمكانية الوصول إلى أحد رموز السلسلة عبر الـ منهج ()charAt حيث يمكننا هنا أن نلاحظ أن تلك الـمناهج لم تسمَ (وكما هو معهود في بقية الأصناف) بالاسم getCharAt وذلك لعدم وجود الـمنهج المناظر setCharAt (والسبب هو عدم القدرة على تعديل السلسلة المضمنة في أي من كائنات هذا الصنف بعد إنشائه).

يعرض الجدول 1-8 أكثر المناهج استخداماً.

8.5 المناهج الخاصة بالسلاسل

جدول 1-8

التابع	الوظيفة
s.length()	ترجع طول السلسلة النصية
s.compareTo(t)	تقوم بمقارنة السلسلة الرمزية s مع السلسلة الرمزية t If s<t then output =(-) Else If s>t then output=(+) Else If s=t then output=(0)
s.compareToIgnoreCase(t)	نفس التابع ((compareTo(t) إلا أن الاختلاف في هذا التابع لا يفرق بين الحروف الكبيرة والصغيرة .
s. equals(t)	if s=t then يعيد هذا التابع true output=
s. equalsIgnoreCase(t)	نفس التابع ((equals(t) إلا أن الاختلاف في هذا التابع لا يفرق بين الحروف الكبيرة والصغيرة .
s. startsWith(t)	يعيد هذا التابع true إذا كان s يبدأ بالسلسلة الرمزية t.
s.startsWith(t, i)	تعيد true إذا كانت السلسلة الرمزية t موجودة في s بدءاً من الموقع i.
s.endsWith(t)	تعيد true إذا كان s تنتهي ب t.
عمليات البحث : كل التوابع ()indexOf تقوم بأرجاع 1- إذا كانت العنصر المراد البحث عنة غير موجود.	
s.indexOf(t)	ترجع أول موقع يوجد فيه t داخل السلسلة الرمزية s. قد يكون t حرف char أو سلسلة String.
s.indexOf(t , i)	ترجع موقع أول مكان توجد فيه t داخل

السلسلة الرمزية s بعد الموقع i. قد يكون t حرف char أو سلسلة String.	
يرجع موقع آخر مكان يوجد فيه (الحرف – السلسلة) المخزن في المتغير c داخل السلسة الرمزية s.	s.lastIndexOf(c)
يرجع الحرف الموجود في الموقع I داخل السلسلة الرمزية s.	s.charAt(i)
ترجع جزئ من السلسلة الرمزية s بدءاً من الموقع I وحتى النهاية.	s. substring(i)
ترجع جزئ من السلسلة الرمزية s بدءاً من الموقع I وحتى الموقع j-1.	s. substring(I,j)
تحويل سلسلة المتغير s إلى حروف صغيرة.	s.toLowerCase()
تحويل سلسلة المتغير s إلى حروف كبيرة.	s.toUpperCase()
أبعاد الفراغات من السلسلة الموجودة في s.	s.trim()
إنشاء سلسلة رمزية جديدة من السلسلة الرمزية s بعد تبديل كل c1 بـ c2 ، وهما من النوع char.	s.replace(c1, c2)
if s=t then output= true يرجع	s.matches(t)
إنشاء مصفوفة تحتوي على أجزاء من سلسلة الرمزية s مقسمة حسب ظهور regexStr.	s.split(regexStr)
كما في split() لكن مع تحديد عدد التقسيم.	s. split(regexStr, count)
يجمع هذه السلسلة مع سلسلة أخرى.	s.concat(t)
يحول سلسلة نصية إلى قيمة عددية .	s.toSteing()
يعيد نص يمثل قيمة من الأنماط المحددة .	s.valueOf(d : double , int , float , long)

لنعطي مثالاً يعمل على أخذ جزء من سلسلة ما:

```
1. // برنامج يعمل على أخذ جزء من سلسلة ما
2. class Chp8_1 {
3.
4.         public static void main(String[] av) {
5.                 String a = "Java is great.";
6.                 System.out.println(a);
7.                 String b = a.substring(5);
8.                 // b is the String "is great."
9.
```

```
10.             System.out.println(b);
11.             String c = a.substring(5,7);// c is the String "is"
12.             System.out.println(c);
13.             String d = a.substring(5,a.length());// d is "is great."
14.             System.out.println(d);
15.                                     }
16. }
```

```
General Output
1 ---------------
2 Java is great.
3 is great.
4 is
5 is great.
```

شرح المثال:
في السطر 7 يقوم المنهج substring() ببناء كائنٍ جديدٍ من الصنف String وبحيث يضم هذا الكائن سلسلة متوالية من الرموز المضمنة في مكان ما ضمن السلسلة الأم التي استخدمت فيها هذا المنهج.

لا بد أن أحدنا يتساءل عن عدم تطبيق العرف المستخدم في تسمية الطرق ضمن Java، والذي يقوم على استخدام الحروف الكبيرة مع بداية الكلمات المستخدمة في تسمية تلك المناهج.

يمكنك أن تعلم عزيزي القارئ بأن Java لا تلتزم التزاماً دقيقاً بهذا العرف. مما يعني وجود بعض الحالات المخالفة لهذا العرف، ومنها هذه الحالة substring().

يضم الصنف String شكلين مختلفين للمنهج substring:

يقبل الشكل الأول وسيطاً وحيداً يتألف من رقمٍ يمثل ترتيب الرمز الأول في السلسلة الجديدة المراد إنشائها ضمن السلسلة الأم. وهذا يعني أن الجديدة تضم إضافة إلى هذا الرمز كافة الرموز التالية حتى نهاية السلسلة الأصلية كما في السطر 7.

أما الشكل الثاني لهذه الطريقة فيأخذ وسيطين، وهما: رقم ترتيب البداية ورقم ترتيب النهاية. أي أنه يضم ترتيب رمزي البداية والنهاية في السلسلة الأصلية كما في السطر 11.

لا بد أن نشير هنا إلى الاختلاف الكامن بين هذا الشكل وبين نظرائه في لغات أخرى. ففي بعض اللغات يمثل الوسيط الثاني طول السلسلة الجزئية المطلوبة.

يمكن الحصول على ترتيب أي من رموز السلسلة باستخدام المنهجين ,lasttindexof( ) indexof() من الصنف String.

تذكر أخي القارئ بأن رقم ترتيب الرمز الأخير ضمن السلسلة الرمزية الجزئية يكون دائماً أقل بواحد من قيمة الرقم المحدد كوسيط ثانٍ في استدعاء المنهج substring (رقم ترتيب رمز النهاية).

8.6 خزن السلاسل

تخزن السلاسل في لغة Java بأشكال مختلفة فيمكن استخدام الحالة التالية:

```
String namestring = new String("Literal");
```

ويمكن استخدام الصيغة التالية:

```
String namestring = "Literal";
```

وكل حالة لها شكل معين يتم تخصيصها في الذاكرة .

ففي الحالة الأولى يتم أخبار المترجم بأن يقوم بإنشاء سلسلة جديدة وتخصيص القيمة التي بين علامة (") اقتباس ، أم الحالة الثانية فيتم إسناد القيمة التي بين الحاصرتين – الاقتباس إلى المتغير المعرف.

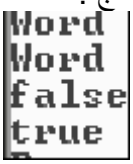
لاحظ المثال التالي

```
1. // برنامج يبين كيفية تخزين السلاسل بين المتحولات
2. class Chp8_2 {
3.         public static void main(String args[]) {
4.                 String s1 = new String("Word");
5.                 String s2 = "Word";
6.
7.                 System.out.println(s1);
8.                 System.out.println(s2);
9.
10.                System.out.println(s1==s2);
11.
12.                System.out.println(s1.equals(s2));
13.                                         }
14. }
```

شرح المثال:

في السطر 4 تم إنشاء سلسلة جديدة باسم s1 وإعطائها الكلمة "Word"
، وفي السطر 5 تم إعطاء المتحول s2 نفس الكلمة "Word" ، وتمت طباعة قيمة المتحولات في الأسطر 8,7 .

ما يهمنا السطر 10 حيث تم السؤال هل s1==s2 ، فكانت الإجابة false رغم أن قيمتهما متساويتين !
ما يفهمه المترجم في السطر 10 هو هل موقع المتغير s1 نفس موقع المتغير s2 .
في السطر 12 استخدمنا تابع من توابع المكتبة String ، فكان السؤال هل قيمة المتغير s1 تساوي قيمة المتغير s2. فكانت الإجابة true. والشكل التالي يبين ناتج تنفيذ البرنامج :



نستطيع أيضاً صنع سلسلة باستخدام مصفوفة من المحارف. على سبيل المثال:

```
1. // Construct one String from another.
2. class Chp8_3 {
3.         public static void main(String args[]) {
4.                 char c[] = {'J', 'a', 'v', 'a'};
5.                 String s1 = new String(c);
6.                 String s2 = new String(s1);
7.                 System.out.println(s1);
8.                 System.out.println(s2);
9.
        }
10. }
```

شرح المثال:
في السطر 4 تم تعريف مصفوفة من المحارف باسم c ، والسطر 5 تم إنشاء سلسلة جديدة تحتوي على جميع محارف المصفوفة في جملة واحدة مخزنة في المتحول s1 ، والسطر 6 نفس ما تم على المتحول s1 طبق على المتحول s2.
فيكون ناتج تنفيذ البرنامج كما يلي:

- السؤال هنا هل ناتج تنفيذ هذا الكود true ولماذا ؟

```
1. // Exp
2. class Chp8_4 {
3.         public static void main(String args[]) {
4.                 char c[] = {'J', 'a', 'v', 'a'};
5.                 String s1 = new String(c);
6.                 String s2 = new String(s1);
7.                 System.out.println(s1);
8.                 System.out.println(s2);
9.                                         }
10. }
```

تدعم لغة c++ ميزة الوصول إلى السلاسل وكإنها مصفوفة ، بينما لغة Java لا تدعم هذه الميزة مباشرة بل أعطت التابع charAt() الذي يمكننا من الوصول إلى أحرف هذه السلسلة . كما في المثال التالي:

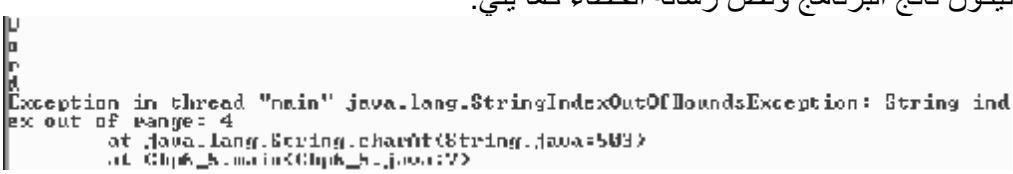
```
1. //برنامج يبين كيفية الوصول إلى احرف سلسلة ما
2. class Chp8_5 {
3.         public static void main(String args[]) {
4.                 String s1 = new String("Word");
5.
6.                 for(int i= 0; i< s1.length();i++)
7.                         System.out.println(s1.charAt(i));
8.                                                 }
9. }
```

فيكون ناتج تنفيذ البرنامج كما يلي:

تعتبر محاولة الوصول إلى محارف السلسلة s1 باستخدام دليل يقع خارج نطاق هذه السلسلة من الأخطاء الشائعة، كهذا البرنامج يعطي رسالة خطاء في زمن التنفيذ :

```
1. //برنامج يبين كيفية الوصول إلى احرف سلسلة ما
2. class Chp8_6 {
3.         public static void main(String args[]) {
4.                 String s1 = new String("Word");
5.
6.                 for(int i= 0; i<= s1.length();i++)
7.                         System.out.println(s1.charAt(i));
8.                                                 }
9. }
```

فيكون ناتج البرنامج ونص رسالة الخطاء كما يلي:



فيجب إنقاص قيمة الدليل بواحد 1-()length.

يؤدي استخدام عوامل المقارنة مثل (< , =< , > , =>) من أجل المقارنة بين النصوص إلى خطاء قواعدي حيث يجب استخدام إحدى صيغ المقارنة المبينة في الجدول ــ السابق.

8.7 التحويل التلقائي

يمكن تحويل مصفوفة أرقام تحوي على شفرات المحارف إلى سلسلة نصية مكونة من محارف ، كالمثال التالي:

```
1. // Construct string from subset of char array.
2. class Chp8_7{
3.         public static void main(String args[]) {
4.                 byte ascii[] = {65, 66, 67, 68, 69, 70 };
5.                 String s1 = new String(ascii);
6.                 System.out.println(s1);
7.                 String s2 = new String(ascii, 2, 3);
8.                 System.out.println(s2);
9.                                                 }
10. }
```

فيكون ناتج تنفيذ البرنامج كما يلي:

```
ABCDEF
CDE
```

من الممكن تحويل السلسلة إلى مصفوفة والعكس بالعكس، فلكي نحول سلسلة إلى مصفوفة من المحارف نستخدم التابع toCharArray. كهذا المثال:

```
Char c ="Java".toCharArray();
```

إليك هذا المثال الذي هو خلاصة المكتبة String ، لما يحتوي على الكثير من التوابع الخاصة بالسلاسل :

```
1. // Strings.java
2.  public class Chp8_8{
3.  public static void main(String args[]){
4.  String s0="Well Come to Java World!" ,
5.  s = "hello", t = "HELLO", s1, s2[], s3;
6.  char c;
7.  boolean b;
8.  int i;
9.  System.out.println();
```

```
10.  i = s0.length();
11.  System.out.println(" The length of " + "\"" + s0 + "\"" + " = " + i +"\n");
12.  i = s.compareTo(t);
13.  if (i == 0)
14.  System.out.println(" \"" + s + "\"" + " is == " + "\"" + t + "\"\n");
15.  else if (i<0)
16.  System.out.println(" \"" + s + "\"" + " is < " + "\"" + t + "\"\n");
17.  else
18.  System.out.println(" \"" + s + "\"" + " is > " + "\"" + t +"\"\n");
19.  i = s.compareToIgnoreCase(t);
20.  System.out.print(" Ignoring case: ");
21.  if (i == 0)
22.  System.out.println(" \"" + s + "\"" + " is == " + "\"" + t + "\"\n");
23.  else if (i<0)
24.  System.out.println(" \"" + s + "\"" + " is < " + "\"" + t + "\"\n");
25.  else
26.  System.out.println(" \"" + s + "\"" + " is > " + "\"" + t +"\"\n");
27.  b = s.equals(t);
28.  System.out.println(" Is " + "\"" + s + "\"" + " equals to " + "\"" + t+ "\"" + " ? " + b +"\n");
29.  b = s.equalsIgnoreCase(t);
30.  System.out.print(" Is " + "\"" + s + "\"" + " equals to ");
31.  System.out.println("\"" + t + "\"" + " (ignoring case)? " + b +"\n");
32.  b = s.startsWith("H");
33.  System.out.println(" Is " + "\"" + s + "\"" + " starts with \"H\"? " +b + "\n");
34.  b = s.startsWith("l", 3);
35.  System.out.print(" Is " + "\"" + s + "\"" + " starts with \"l\" ");
36.  System.out.println("from position 3 ? " + b + "\n");
37.  b = s.endsWith("lo");
38.  System.out.print (" Is " + "\"" + s + "\"" + " ends with \"lo\"");
39.  System.out.println(" ,from position 3 ? " + b + "\n");
40.  i = s0.indexOf("Java");
41.  System.out.print (" Java is at position ");
42.  System.out.println( i + " of " + "\"" + s0 + "\"\n");
43.  i = s0.indexOf("java", 4);
44.  System.out.print(" java is at position" + i +"of ");
45.  System.out.println(" \"" + s0 + "\"" + ", starting from position 4\n");
46.  i = s0.indexOf('e');
47.  System.out.print (" \'e\' is at position ");
48.  System.out.println(i + " of " + "\"" + s0 + "\"\n");
49.  i = s0.indexOf('e', 4);
50.  System.out.print (" \'e\' is at position " + i + " of ");
51.  System.out.println("\"" + s0 + "\"" + " starting from position 4\n");
52.  i = s0.lastIndexOf('e');
53.  System.out.print(" Last occurrence of \'e\' in ");
54.  System.out.println("\"" + s0 + "\"" + " is at "+ i + "\n");
55.  i = s0.lastIndexOf(t);
56.  System.out.print(" Last occurrence of 'rl' in ");
57.  System.out.println("\"" + s0 + "\"" + " is at " + i + "\n");
58.  c = s0.charAt(3);
59.  System.out.print(" The character at position 3 in");
60.  System.out.println("\"" + s0 + "\"" + " is " + c + "\n");
```

```
61.  s3 = s0.substring(6);
62.  System.out.print (" The substring of ");
63.  System.out.println("\"" + s0 + "\"" + " starting from 6 is\n" +"\t\t\t\"" + s3 + "\"\n");
64.  s1 = s0.substring(6, 10);
65.  System.out.print(" Substring of " + "\"" + s0 + "\"" + " starting ");
66.  System.out.println("from 6 to 10 is:" + "\"" + s1 + "\"\n");
67.  System.out.print (" \"" + s0 + "\"" + " in lowercase is ");
68.  System.out.println("\"" + s0.toLowerCase() + "\"\n");
69.  System.out.print("\""+ s0 + "\"" + "in uppercase ");
70.  System.out.println("\""+ s0.toUpperCase() + "\"\n");
71.  System.out.println(" \"" + s0 + "\"" + " with replacing all spaces ");
72.  }
73.  }
```

وهذا ناتج التنفيذ:

```
The length of "Well Come to Java World!" = 24

"hello" is > "HELLO"

Ignoring case:  "hello" is -- "HELLO"

Is "hello" equals to "HELLO" ? false

Is "hello" equals to "HELLO" (ignoring case)? true

Is "hello" starts with "H"? false

Is "hello" starts with "l" from position 2 ? true

Is "hello" ends with "lo" ,from position 2 ? true

Java is at position 13 of "Well Come to Java World!"

java is at position-1of  "Well Come to Java World!", starting from position 4

'e' is at position 1 of "Well Come to Java World!"

'e' is at position 8 of "Well Come to Java World!" starting from position 4

Last occurrence of 'e' in "Well Come to Java World!" is at 8

Last occurrence of 'rl' in "Well Come to Java World!" is at -1

The character at position 3 in"Well Come to Java World!" is l

The substring of "Well Come to Java World!" starting from 6 is
                "come to Java World!"

Substring of "Well Come to Java World!" starting from 6 to 10 is:"come "

"Well Come to Java World!" in lowercase is "well come to java world!"

Well Come to Java World!"in uppercase "WELL COME TO JAVA WORLD!"

 "Well Come to Java World!" with replacing all spaces
Press any key to continue...
```

8.8 لصق السلاسل

تمكننا لغة Java من لصق السلاسل بسهولة تامة ، حيث يتم استخدام علامة (+) لعملية لصق سلسلتين . كهذا المثال:

```
1. //برنامج يوضح عملية لصق السلاسل
2. class Chp8_9 {
3.         public static void main(String args[]) {
4.                 String s1 = new String("Word");
5.                 String s2= s1+"Top";
6.
7.                 System.out.println(s2);
8.
9.                 System.out.println(s2+2+2);
10.
11.                System.out.println(s2+(2+2));
12.
        }
13. }
```

شرح المثال:
في السطر 5 تم دمج سلستين مع بعضها وخزنها في المتحول s2.
وفي السطر 9 يبين لنا كيف يقوم مترجم Java بجمع ما بداخل () القوسين لجمله الطباعة ، وعرضها على الشاشة كقيمة نصية ،
فإذا أردنا دمج ناتج العملية مع المتحول s2 ينبغي علينا وضع () للعملية الحسابية ، كما في السطر 11.
وهذا ناتج تنفيذ البرنامج:

8.9 الصنف من نوع Character

يقدم هذا الصنف العديد من التوابع للتعامل مع الأحرف. وسنستعرض أهم هذه التوابع في الجدول 2-8.

جدول 2-8

التابع	الوظيفة
isDigit(t)	ترجع true إذا كان t هو رقم.
isLetter(t)	ترجع true إذا كان t هو محرف.

isLetterOrDigit(t)	ترجع true إذا كان t هو رقم أو محرف.
toLowerCase(t)	ترجع true إذا كان t هو حرف صغير.
isUpperCase(t)	ترجع true إذا كان t هو حرف كبير.

8.10 الصنف من نوع StringBuffer

لو أردنا إجراء بعض التعديلات على سلاسلنا الرمزية يتمثل الحل في استخدام الصنف StringBuffer بدلاً من الصنف String (والذي يمكن أن يُهيأ بالقيمة الأولية للسلسلة الرمزية عند إنشاء أي كائن من كائناته). وعند إجراء كافة المعالجات والتغيرات المطلوبة على السلسلة ضمن أحد كائنات هذا الصنف، يتم تحويل القيمة النهائية لهذه السلسلة إلى أحد كائنات الصنف String باستخدام المنهج ()toString المشمولة في أي من الأصناف القياسية للغة Java.

يمكننا أن نؤكد لك أخي القارئ عدم إمكانية إضافة المنهج ()setCharAt أو أي من مناهج التعديل الأخرى إلى الصنف String في أي من الإصدارات المستقبلية للغة Java.

ولكن كيف يمكننا أن نتأكد من ذلك؟

يعود سبب تأكيد هذا إلى قاعدة بسيطة تقول:(" إن ثبات السلاسل وعدم إمكانية تعديلها يعد من أهم الأسس التي تقوم عليها آلة Java الافتراضية").

لنوضح ذلك:

نحن نعلم أن مفهومي: مسالك التنفيذ المتعددة (multiprocessing) و(Threads) والأمن (Security)، هما من أهم المفاهيم المطروحة ضمن الآلة الافتراضية للغة Java.

بناء على ذلك يمكننا أن نناقش المثال التالي حول البريمجات في لغة Java:

من المعروف أن استخدام البريمجات في عالم Java ، يُحاط عادة ببعض القيود التي تفيد في مسألة حماية وأمن المعلومات على طرف الز بون، حيث يُمنع البريمج في الحالة العامة، من الوصول إلى العديد من الموارد المحلية على طرف الزبون.

لنفترض أننا في أحد البريمجات وضمن هذا البريمج يقوم المسلك A بفتح مسلك آخر يدعى B لنفترض أيضاً أن المسلك A يقوم بإنشاء سلسلة رمزية S تضم اسم أحد الملفات.

كما يقوم أيضاً بحفظ مؤشر إلى هذا الملف ضمن السلسلة S2.

يمرر هذا المسلك السلسلة S إلى إحدى الطرق التي تتطلب درجات معينة من سماحيات الوصول.

سوف تقوم هذه الطريقة بالتأكيد باستدعاء أحد كائنات الصنف SecurityManager الذي يمثل أحد أصناف الآلة الافتراضية للغة Java، وذلك في حال إنشاء وتثبيت هذا الكائن ضمن بيئة العمل (وهذا ما يحدث ضمن بيئة العمل الخاصة بالبريمجات).

ماذا يحدث لو قام المسلك B بتعديل السلسلة S2 التي تشير إلى الملف المطلوب، بحيث تشير بعد هذا التعديل إلى أحد ملفات النظام، لا يُسمح للبريمجات بالوصول إليها، وذلك خلال الفاصل الزمني القصير الذي يفصل بين إعطاء الكائن SecurityManager الموافقة على الوصول إلى الملف المشار إليه، وبين قيام نظام الدخل والخرج بالإجراءات اللازمة لفتح هذا الملف.

لا شك أن مثل هذه الحالات تشكّل ضربة قاضية لمفهوم الأمن والحماية ضمن لغة Java. من هنا جاءت الحاجة إلى منع أي عمليات تعديل على السلاسل الرمزية بعد إنشاء كائناتها. لا شك أن بإمكاننا إسناد أي سلسلة رمزية بديلة إلى المتحول المرجعي S ، إلا أن ذلك لا يُعد على الإطلاق تعديلاً للسلسلة الرمزية التي كان يُشار إليها باستخدام ذلك المتحول. أي أن هذا الإسناد ليس له أي تأثير على السلسلة الرمزية الأولى.

إلا أن هذا الإسناد قد يؤدي إلى ضياع السلسلة الأولى من خلال برنامج جامع النفايات (garbage collection) الذي يُعد أحد مكونات الآلة الافتراضية للغة Java ، وذلك في حال عدم وجود أي متحول مرجعي يشير إلى تلك السلسلة بعد عملية الإسناد السابقة.

تعد السلاسل الرمزية في لغة Java أحد أهم الأنواع الأساسية في هذه اللغة. فبخلاف معظم الأصناف الأخرى التي تشكّل واجهة اللغة الخاصة بالمبرمجين، تتميز هذه السلاسل بسلوك ثابت غير قابل للتعديل وذلك من خلال استخدام الكلمة Final في تعريف هذا الصنف.

هذا يعني أن إمكانية الاشتقاق من هذا الصنف غير موجودة. والسبب في ذلك يعود بنا إلى ذات السبب في منع عملية تعديل السلاسل بعد إنشائها ألا وهو الأمن والحماية. فعند الاشتقاق يمكن للمبرمج إضافة الطريقة SetCharAt() المثيرة للجدل المناقش سابقاً.

سوف نقدم الآن شفرة بسيطة يمكنك أخي القارئ اختباره على حاسبك للتأكد من كل ما ذكرناه حول سلوك هذا الصنف.

```
/**
 * If this class could be compiled, Java security would be a myth.
 */
public class WolfInStringsClothing extends java.lang.String {
    public void setCharAt(int index, char newChar) {
        // The implementation of this method
        // would be left as an exercise for the reader.
        // Hint: compile this code exactly as-is before bothering!
```

```
    }
}
```

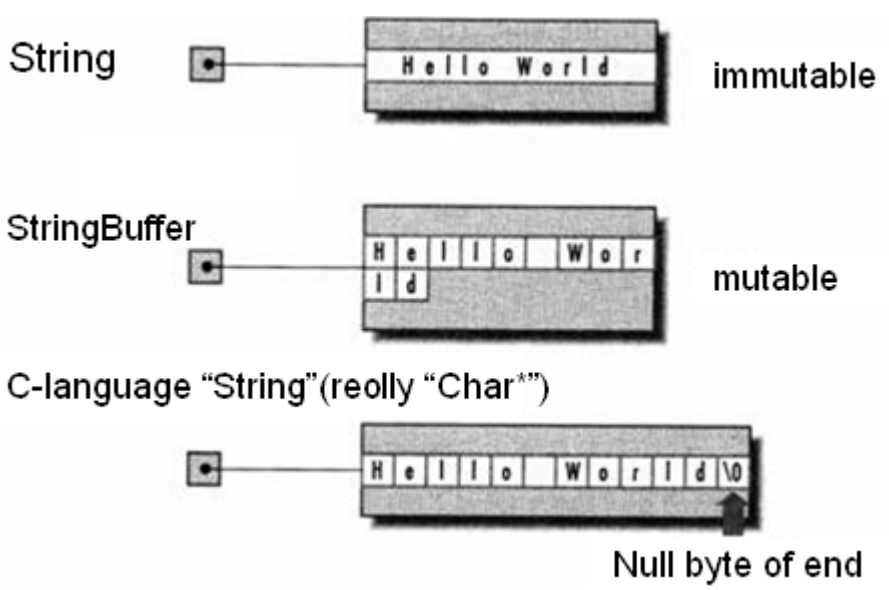
ولكن ما العمل؟
نحن بحاجة إلى منهج عملي يمكننا من إجراء بعض التعديلات على سلاسلنا الرمزية!

كما ذكرنا سابقاً فإن الصنف StringBuffer يقدم الحل لهذه المشكلة حيث يتعامل هذا الصنف مع كل من الرموز والسلاسل ويقدم لنا عدداً من المناهج التي تؤمن إجراء معظم التعديلات المطلوبة على السلسلة المحتواة في كل كائنٍ من هذا الصنف.

بالإضافة إلى هذه المناهج فإن العودة إلى الصنف String انطلاقاً من هذا الصنف يُعد عملية سهلة باستخدام الطريقة toString().

نذكّر مرة أخرى مبرمجي باقي اللغات كلغة C ، بأن السلاسل الرمزية في لغة Java ليست مجرد مصفوفة من الرموز. وبالتالي فإن المبرمج هنا مضطر إلى استخدام المناهج المعرَّفة ضمن الأصناف لإجراء بعض عمليات المعالجة، كتلك التي تهتم بمعالجة رموز السلسلة كل على حدة.

يوضح الشكل 1-8 رسم توضيحي لبنية كل من الصنف String والصنف StringBuffer بالإضافة إلى توضيح بنية هذه السلاسل في لغة C.



الشكل 1-8 رسم توضيحي لبنية كل من الصنف String والصنف StringBuffer

نستعرض الآن بعض إمكانيات الصنف (StringBuffer) الذي بواسطته نستطيع من (حشر ، عكس ، حذف ، استبدال) سلسلة.

8.10.1 الإضافة إلى السلسلة (insert)

مثال الإضافة إلى السلسلة (insert):

```
1. // Demonstrate insert().
2. class Chp8_10 {
3.         public static void main(String args[]) {
4.                 StringBuffer sb = new StringBuffer("I Java!");
5.                 sb.insert(2, "like ");
6.                 System.out.println(sb);
7.                                                 }
8. }
```

شرح المثال:
في السطر رقم 5 تم حشر الكلمة (like) في المتحول sb من الموقع الثاني لها ، فيكون ناتج التنفيذ كما يلي:

8.10.2 عكس السلسلة (reverse)

يمكننا ببساطة استخدام المنهج reverse() ضمن الصنف StringBuffer. حيث يـ في هذا الـ منهج بالغرض المطلوب، دون الحاجة إلى إضافة إجراءات لتحقيق الغرض المطلوب. نقدم فيما يلي مثالاً بسيطاً يوضح استخدام الطريقة السابقة.

```
1. // Using reverse() to reverse a StringBuffer.
2. class Chp8_11{
3.         public static void main(String args[]) {
4.                 StringBuffer s = new StringBuffer("FCGDAEB");
5.                 System.out.println(s);
6.                 s.reverse();
7.                 System.out.println(s);
8.
        }
9. }
```

شرح المثال:

حيث تمثل السلسلة الرمزية في المثال السابق ترتيب التواقيع الخاصة بالمفاتيح الموسيقية الحادة في الموسيقى الغربية. وبالتالي فإن عكس ترتيب تلك السلسلة يقدم لنا ترتيب المفاتيح المسطحة في تلك الموسيقى.

فيكون ناتج التنفيذ :

```
General Output
--------
FCGDAEB
BEADGCF
```

لا بد أن ننوه انه يمكننا إجراء مهمة عكس الرموز السابقة، دون استخدام الطريقة السابقة. حيث يمكننا معالجة كل رمز على حدة، ووضعه في المكان المناسب.

8.10.3 الحذف من السلسلة (delete)

مثال حذف حرف من سلسلة (delete):

```
1. // Demonstrate delete() and deleteCharAt()
2. class Chp8_12 {
3.         public static void main(String args[]) {
4.                 StringBuffer sb = new StringBuffer("This is a test.");
5.                 sb.delete(4, 7);
6.                 System.out.println("After delete: " + sb);
7.                 sb.deleteCharAt(0);
8.                 System.out.println("After deleteCharAt: " + sb);
9.                                         }
10. }
```

فيكون ناتج التنفيذ :

```
After delete: This a test.
After deleteCharAt: his a test.
```

8.10.4 استبدال من السلسلة (replace)

مثال استبدال حرف من سلسلة (replace):

```
1. // Demonstrate replace()
2. class Chp8_13 {
3.         public static void main(String args[]) {
4.                 StringBuffer sb = new StringBuffer("This is a test.");
5.                 sb.replace(5, 7, "was");
6.                 System.out.println("After replace: " + sb);
7.                                                         }
8. }
```

فيكون ناتج التنفيذ :

```
After replace: This was a test.
```

8.11 الصنف من نوع StringTokenizer:

يعتبر الصنف StringTokenizer أحد الأصناف المفيدة والمتعلقة بمعالجة النصوص. يستخدم هذا الصنف من أجل تقسيم النص إلى قطع صغيرة بحيث يصبح من الممكن استعادة ومعالجة المعلومات المخزنة فيها. والجدول 3-8 يقدم العديد من مناهج الصنف StringTokenizer من أجل معالجة الأجزاء المنفصلة في النص.

جدول 3-8

المنهج	الوصف
StringTokenizer(str: String)	يبني كائن جديد من أجل السلسلة النصية المحددة
StringTokenizer(str:String , delim:String)	يبني كائن جديد من أجل السلسلة النصية المحددة مع تحديد الفواصل
StringTokenizer(str:String , delim:String , ruternDelims: Boolean)	يبني كائن جديد من أجل السلسلة النصية المحددة مع تحديد الفواصل وتحديد قيمة الإعادة المنطقية
hasMoreTokens():boolean	يعيد القيمة true إذا كان لا يزال باقي أجزاء
countTokenz() :Boolean	يعيد عدد الأجزاء المتبقية
nextToken(): String	يعيد الجزء التالي
nextToken(delim: String)	يعيد الجزء التالي مع استخدام الفواصل

ويقدم هذا المثال تخزين السلسلة باستخدام الصنف StringTokenizer:

```
1. import java.util.*;
2. /**
3.  * Simple StringTokenizer demo program.
4.  */
5. public class Chp8_14 {
6.
```

```
7.      public static void main(String[] argv) {
8.              StringTokenizer st = new StringTokenizer("Hello World of Java");
9.              while (st.hasMoreTokens())
10.                     System.out.println("Token: " + st.nextToken());
11.                                                     }
12. }
```

```
General Output
1 ---------------
2 Token: Hello
3 Token: World
4 Token: of
5 Token: Java
```

شرح المثال:

السطر 8 يقوم التصنيف StringTokenizer بتقسيم السلسلة الرمزية إلى كلمات أو موصفات وفق ما يمكن تسميته بحدود الكلمات (فواصل الكلمات) ضمن اللغات الأوروبية، والذي يكون عادة عبارة عن فراغ (Space) يفصل بين تلك الكلمات.

ولكن ماذا لو أردنا تحقيق التقسيم السابق باستخدام رمز اختياري مختلف ؟

لا مشكلة يتيح لنا هذا الصنف بناء الكائنات مع تحديد الرمز أو مجموعة الرموز التي يتم بواسطتها تجزيء السلسلة الرمزية. حيث يتم تمرير هذه الرموز على شكل سلسلة رمزية كوسيط ثانٍ إلى عملية البناء الخاصة بهذا الصنف حيث تسمى هذه الرموز بالمحددات .ومن البرنامج السابق نستبدل السطر 8 بهذا السطر إذا أردنا عملية التجزئة باستخدام الرمز " | ".

```
StringTokenizer st = new StringTokenizer("Hello, World|of|Java", ", |");
```

وهذا مثال أخر يقوم بإضافة المعالجة الضرورية بهدف الحفاظ على الحقول الفارغة وعدم إظهار المحددات كجزءٍ من العرض أو كجزءٍ من مجموعة الموصفات (Tokens).

```
1. import java.util.*;
2. /** Show using a StringTokenizer including getting the delimiters back */
3. public class Chp8_15 {
4.      public final static int MAXFIELDS = 5;
5.              public final static String DELIM = "|";
6.              /** Processes one String, returns it as an array of fields */
7.              public static String[] process(String line) {
8.                      String[] results = new String[MAXFIELDS];
9.                      // Unless you ask StringTokenizer to give you the tokens,
10.                     // it silently discards multiple null tokens.
11.                    StringTokenizer st = new StringTokenizer(line, DELIM, true);
```

```
12.                         int i = 0;
13.                         // stuff each token into the current slot in the array.
14.                         while (st.hasMoreTokens()) {
15.                                 String s = st.nextToken();
16.                                 if (s.equals(DELIM)) {
17.                                         if (i++>=MAXFIELDS)
18.                                 // This is messy: See StrTokDemo4b which uses
19.                                 // a Vector to allow any number of fields.
20.                                 throw new IllegalArgumentException("Input line "
+
21.                                         line + " has too many fields");
22.                                 continue;
23.                                                 }
24.                                 results[i] = s;
25.                                                 }
26.                         return results;
27.                                                 }
28.         public static void printResults(String input, String[] outputs) {
29.                 System.out.println("Input: " + input);
30.                 for (int i=0; i<outputs.length; i++)
31.                    System.out.println("Output " + i + " was: " + outputs[i]);
32.                                                                 }
33.         public static void main(String[] a) {
34.                 printResults("A|B|C|D", process("A|B|C|D"));
35.                 printResults("A||C|D", process("A||C|D"));
36.                 printResults("A|||D|E", process("A|||D|E"));
37.                                         }
38. }
```

```
General Output
 1  ------------------
 2  Input: A|B|C|D
 3  Output 0 was: A
 4  Output 1 was: B
 5  Output 2 was: C
 6  Output 3 was: D
 7  Output 4 was: null
 8  Input: A||C|D
 9  Output 0 was: A
10  Output 1 was: null
11  Output 2 was: C
12  Output 3 was: D
13  Output 4 was: null
14  Input: A|||D|E
15  Output 0 was: A
16  Output 1 was: null
17  Output 2 was: null
18  Output 3 was: D
19  Output 4 was: E
```

شرح المثال:

يقوم هذا المثال أولاً بتحديد العدد الأعظمي للحقول المطلوبة انطلاقاً من السلسلة الرمزية الأم، وذلك بسبب اعتمادنا على المصفوفة العادية (محدودة الطول) في تخزين نتائج المعالجة النهائية.

يمكننا بالطبع تجاوز هذه الخطوة عند اعتمادنا على صنف الشعاع Vector في تخزين تلك النتائج. حيث يعمل هذا الصنف كمصفوفة ديناميكية يمكن زيادة طولها عند الحاجة إلى تخزين عناصر جديدة فيها (مصفوفة غير محدودة الطول).

يتم ضمن المنهج Process بناء النموذج

التكراري ً، في معالجة كل عنصر من مجموعة موصفات الكائن StringTokenizer وذلك اع تماداً على الـ منهجين hasMoreTokens وnextToken. حيث يتم فحص كل عنصر من المجموعة واستبعاد العنصر المحدد لفصل الموصفات داخل الكائن يتم في نهاية المعالجة تقديم حقول المعلومات المطلوبة على شكل مصفوفة من السلاسل الرمزية.

يعمل المنهج PrintResults على تقديم نتائج المعالجة السابقة بشكل أكثر وضوحاً للقارئ، وذلك من خلال إضافة بعض النصوص التوضيحية إلى حقول المعلومات المستحصلة.

يمكن بوجود المنهج الرئيسي main اختبار صحة المعالجة حيث قمنا كما هو واضح، بوضع أمثلة مختلفة لسلسلة المعلومات المحتملة.

8.12 إقحام الرموز غير القابلة للطباعة ضمن السلاسل الرمزية:

قد نحتاج أحياناً إلى إضافة بعض الرموز التي لا تدخل في عملية الطباعة إلى السلاسل الرمزية. حيث ينحصر دور هذه الرموز في الإشارة إلى وجود بعض المعطيات الخاصة التي لا يمكن إضافتها بشكل مباشر إلى تلك السلاسل.

تقدم لغة Java مـ جموعة من الرموز التي تدعى رموز الهروب والتي تضاف إلى السلسلة الرمزية بعد إضافة الرمز Backslash (\) في المكان الذي يراد له أن يحوي المعطيات الخاصة. حيث تلبي تلك الرموز مباشرة قيم المعطيات المطلوبة.

يقدم الجدول 4-8 مجموعة رموز الهروب ضمن لغة Java ووظائفها.

جدول 4-8

رموز الهروب	وظيفة الرمز	ملاحظات
\t	لإضافة الرمز Tab	
\n	للانتقال إلى بداية سطر جديد (السطر التالي)	العبارة System.getProperty("line.separator") تعيد رمز نهاية السطر المستخدم في النظام الحالي
\r	العودة إلى بداية السطر الحالي	
\f	لتمرير الصفحات في عملية الطباعة	
\b	لإضافة الرمز Backspace (القيام بدوره)	
\'	لإضافة علامة التنصيص المفردة	
\"	لإضافة علامة التنصيص المزدوجة	
\uNNNN	لإضافة شيفرة الرمز وفق الصيغة	

	الموحدة للرموز (Unicode)	
\NNN	لإضافة شيفرة الرموز وفق النظام الثماني للعد	
\\	لإضافة الرمز \ (Backslash) إلى السلسلة	

كما يقدم المثال التالي استخدام الرموز السابقة.

```
1. /**
2.  * Chp8_16.java - show string escapes.
3.  * Note that they may not print correctly on all platforms.
4.  */
5. public class Chp8_16 {
6.       public static void main(String[] argv) {
7.            System.out.println("Java Strings in action:");
8.            // System.out.println("An alarm or alert: \a"); // not supported
9.            System.out.println("An alarm entered in Octal: \007");
10.             System.out.println("A tab key: \t(what comes after)");
11.             System.out.println("A newline: \n(what comes after)");
12.             System.out.println("A UniCode character: \u0207");
13.             System.out.println("A backslash character: \\");
14.                        }
15. }
```

تمارين الفصل

1. أكتب برنامج لإيجاد عدد مرات تكرار الرمز E في الكلمة SENTENCE ؟
2. أكتب برنامج يدخل سلسلة من الكلمات ويقوم بطباعة كل كلمة بسطر ؟
3. أكتب برنامج يقوم بتحويل حروف العلة من صغير إلى كبير ؟
4. أكتب برنامج يقوم بطباعة أكبر كلمة من بين كلمات مدخلة وطباعة عدد حروفها ؟
5. أكتب برنامج يعكس كل كلمة داخل سلسلة من الكلمات ؟
6. أكتب برنامج يطبع أوسط حرف من كل كلمة ؟
7. أكتب البرامج التي تقوم برسم الأشكال التالية عند إدخال كلمة ما:

```
   A                    M
  MAM         M        MMA
 MMAMM       MMA      AMMAR
AMMAMMA     AMMAR      MMA
RAMMAMMAR                M
```

8. أكتب البرامج التي تقوم برسم الأشكال التالية عند إدخال سلسلة ما:

à AMMAR ALI AMIN SAEED

à ALIE
à A AL AMI SAEE
à SAEED AMIN ALI AMMAR
à A ALI A SAEED
à AMMA A A S
à AAASR ALI MMIN AAEED
à AMMARALIAMINSAEED
à AMR LI N SED
à AmMaR aLi AmIn SaEeD
à MAMRA LIA MANI ASEDE
à AMMAR ILA AMAEN DEEAS
à AR AL AN SD
à
```
D  N   I   R
E  I   L   A
E  M   A   M
A  A       M
S          A
```

9. أكتب برنامج تدخل سلسلة من الكلمات ثم تدخل رقم الكلمة فيقوم بعكس الكلمة ؟

10. إذا كانت s1="NAME" ، s2="Java" فما ناتج التعابير الأتية:

- s1.equals(s2)
- s1 += s2;
- s1.length()

11.

Chp9

الكائنات والأصناف

في نهاية هذا الفصل سوف تتعلم :

- التعرف إلى البرمجة الموجهة OOP.
- التعرف على الكائنات والأصناف والعلاقة بينهما.
- التعرف على كيفية تعريف الصنف وكيفية صنع كائن من صنف.
- التعرف على قواعد البانيات(constructors).
- تحديد مجال تغطية المتحولات .
- التصريح عن الأصناف الداخلية.

9.1 مقدمة INTRODUCTION

البرنامج هو مجموعة من تعليمات مرتبة ومتسلسلة منطقيا تؤدي في النهاية لبلوغ هدفاً ما هذا عن البرامج عامة وهذا هو التعريف المبسط للبرنامج وبالتالي يمكن تعريف البرمجة بأنها عملية وضع هذه التعليمات بالطريقة المرتبة والمتسلسلة منطقياً بحيث تؤدي في النهاية لبلوغ الهدف . ولما كانت البرمجة للحاسب الآلي مثلها في ذلك مثل باقي النشاطات الإنسانية قد مرت بمراحل تطور كان ينبغي علينا أن نلمح هنا لأهم هذه المراحل حتى تستطيع أن تدرك معنى عبارة البرمجة الموجهة أو object oriented والتي يطلق عليها اختصارا ل (O.O.P) .

ماذا نعني بالبرمجة الكائنية المنحى - البرمجة الموجهة بالكائنات

؟ OOP(Object Orientation Programming)

في البداية كانت طريقة البرمجة السائدة هي البرمجة الخطية حيث كان البرنامج يكتب كله في ملف واحد وترتب أوامر وتعليمات البرنامج كلها في هذا الملف بحيث يبدأ ا لحاسب في تنفيذ هذه الأوامر والتعليمات من أعلى وكلما انتهى الحاسب من تنفيذ احد هذه التعليمات ينتقل للتي تليه وهكذا حتى ينتهي البرنامج وكانت عملية اختبار البرنامج أصعب ما يمكن وكذلك كانت صعوبة في تصحيح الأخطاء . بعد ذلك ظهرت طريقة البرمجة الإجرائية أو التركيبية وهي التي تعتمد على تقسيم البرنامج لمجموعة منفصلة من الإجراءات والوظائف كل منها يؤدي جزءاً محدداً من البرنامج وبالتالي أصبح البرنامج شكله أفضل مقسم لمجموعة من الأقسام يمكن صيانة وتجربة كل جزء على حداه كما يمكن استخدام نفس الجزء في برنامج أخر. فعى سبيل المثال يتكون البرنامج في فيجوال بيسك من مجموعة من الوحدات (modules) موديول . فمثلاً النافذة الرئيسية التي أنشأها فيجوال بيسك لك بأي مشروع جديد ويسميها form1 وهي وحدة موديول وهذه الوحدة لها عدة مواصفات مثل مكإنها في الشاشة وأبعادها ولون الخلفية وغير ذلك من ا لخصائص وكذلك يمكن أن يحدث لها عدة أشياء مثل أن ينقر بالفارة فوقها أو يتم تغير أبعادها وكذلك فان هذا الشيء يستجيب لتلك الأشياء بردود أفعال مختلفة مثل أن يختفي أو يظهر وذلك ضمن إجراء مرتبط بوقوع حدث ما يمكن أن يحتوي form1 بدوره على مجموعة أخرى من الأشياء مثل مفتاح زر أو صندوق نص text box كل منهما عبارة عن object له خصائصه properties وأحداث event وأفعال mesot وقد تطورت الفكرة أكثر فظهرت البرمجة الشيئية .

حيث يتكون البرامج فيها من عدة أشياء بدلا من إجراءات ووظائف . والأشياء قريبة جدا من أذهاننا حيث يمتلئ العالم من حولنا بالأشياء فكل شي له اسم فهو شي ولا مانع أن يكون جزء منه شي أخر وهكذا وإذا نظرنا للأشياء حولنا OBJECT فسوف نجد أن الأشياء أنواع .

وكذلك نلاحظ أن لكل شي مجموعة خواص وصفات تحدد شكله وسلوكه كما سنجد أن خصائص بعض الأشياء ليس لها وجود في نوع أخر من الأشياء .

وقبل أن ندخل في تفاصيل البرمجة الموجهة سنشرح البرمجة المهيكلة لكي ترسخ الفكرة وتستوعب معنى البرمجة الموجهة.

9.1.1 البرمجة المهيكلة

أولا، لنختبر بصورة سريعة كيف يتم تصميم الأنظمة البرمجية باستخدام الاتجاه المهيكل (أحيانا يُسمّى وظائفي Functional).

في البرمجة المهيكلة Structured Programming، الطريقة المتّبعة عامة هي النظر إلى المسألة، ثم تصميم مجموعة من الوظيفيات functions التي يمكنها انجاز المهام المطلوبة لحلها. إذا تضخّمت هذه الوظيفيات ، يتم تجزئتها حتى تصير صغيرة بالحدّ الذي يتيسّر فيه مناولتها و فهمها. هذه العملية تدعى التفكيك الوظائفي functional decomposition.

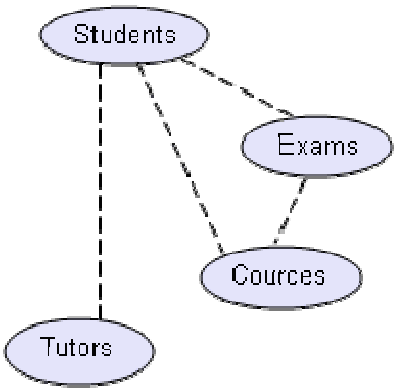
معظم الوظيفيات ستحتاج إلى بيانات من نوع ما لتعمل عليها. البيانات في الآنظمة الوظائفية عادة ما يحتفظ بها في قاعدة بيانات من نوع ما (أو قد يحتفظ بها في الذاكرة كمتغيّرات شاملة global variables) .

لنأخذ مثالا بسيطا، تخيّل منظومة لإدارة معهد، هذه المنظومة تحتفظ ببيانات الطلبة و المدرّبين في المعهد إضافة للمعلومات حول الدورات المتوفرة، كذلك تقوم المنظومة بتتبع كل طالب و الفصول التي التحق بها.

التصميم الوظائفي المحتمل سيتضمّن كتابة الوظيفيات functions التالية:

- إضافة طالب add_student 5
- دخول امتحان enter_for_exam
- فحص علامات امتحان check_exam_marks
- إصدار شهادة issue_certificate
- طرد طالب expel_student

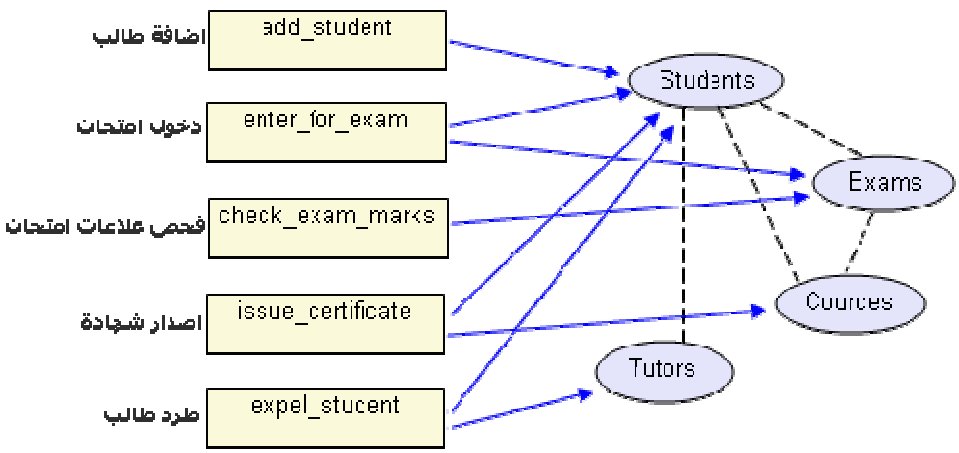
سوف نحتاج أيضا إلى نموذج بيانات data model ليمثّل هذه الوظيفيات. نحتاج لتخزين معلومات عن الطلبة، و المدربين و الامتحانات و الدورات، لذا يجب علينا تصميم مخطط قاعدة بيانات database schema للاحتفاظ بهذه البيانات كما في الشكل 1-9.



شكل 1-9 مخطط قاعدة بيانات بسيط. الخطوط المنقطة تشير إلى اعتمادية مصفوفة بيانات على أخرى، مثلا، كل طالب يتم تدريبه بواسطة عدة مدرّبين.

الآن بدأ واضحا أن الوظيفيات functions التي حدّدناها سابقا سوف تعتمد على هذه المصفوفة من البيانات. مثلا، وظيفة "add_student" (اضافة طالب) ستقوم بتغيير محتويات "Students" (طلبة) ، و وظيفة "issue_certificate" (إصدار شهادة) تحتاج إلى الوصول إلى بيانات طالب Student (لمعرفة تفاصيل الطالب الذي يحتاج للشهادة) و ستحتاج الوظيفة أيضا إلي بيانات الامتحانات Exam .

المخطط diagram 9-2 عبارة عن رسم لكل الوظيفيات، مجتمعة رفق البيانات، و رسمت الخطوط فيها حيثما وجدت اعتمادية dependency .



شكل 2-9: خريطة الوظائف، و البيانات و الاعتماديات.

المشكلة مع هذه المقاربة أن المسألة التي نتعامل معها إذا ما تعقدت أكثر ستزداد صعوبة المحافظة على المنظومة و صيانتها. فلو أخذنا المثال أعلاه، ماذا سيحدث لو تغيّرت المتطلبات requirement بطريقة تؤدّي إلى تغيير أسلوب مناولة بيانات الطالب Student.

كمثال، لنتخيّل أن منظومتنا تعمل على أكمل ما يكون، لكننا اكتشفنا أن تخزين تاريخ ميلاد الطالب على شكل عدد ذو خانتين كي يمثل السنة كانت فكرة سيّئة، الحل المبدأ هنا هو أن نقوم بتغيير حقل تاريخ الميلاد في جدول الطلبة Students من خانتين إلى أربع خانات لرقم السنة.

المشكلة الجدية لهذا التغيير تنبع من أنه قد يسبّب في ظهور أثار جانبية غير متوقعة. فبيانات جدول الامتحانات Exam و جدول الدورات Courses و جدول المدرّبين Tutors كلها (بطريقة أو بأخرى) تعتمد على بيانات جدول الطالب Students ، لذا قد نتسبب في كسر بعض العمليات بـ تغييرنا البسيط هذا، و إعاقة وظيفيات add_student و enter_for_exams و issue_certificate و expel_student ، فوظيفة add_student لن تعمل بالتأكيد لإنها تتوقع أن تكون المعلومة الخاصة بسنة الميلاد على شكل رقم بخانتين بدلا من أربع.

إذا، لدينا معدل كبير من المشاكل المحتملة، و الأسوأ إننا لن نستطيع بسهولة تعيين أماكن الاعتمادية في التوليف code التي ستتأثر بهذا التغيير.

كم من مرّة قمت بتعديل سطر في التوليف و بكل براءة دون أن تعي انك سبّبت عن غير قصد في كسر عمليات أخرى قد تبدو لا علاقة لها في الظاهر؟

كإشكالية عام 2000 (ثغرة الألفية) ذات التكلفة العالية كان سببها بالضبط هذه المشكلة. فحتى لو أن حلها يفترض به أن يكون بسيطا (تغيير كل حقل سنة من خانتين إلى أربع) فان التداعيات المحتملة لهذا التغيير البسيط يجب التحقق منها و فحصها بدقة.

ويمكن تلخيص خواص برمجة الإجراءات كالأتي:

1. تركز على المعالجة (الخوارزميات).
2. يقسم البرنامج الكبير إلى مجموعة برامج صغيرة تسمى الدوال.
3. أغلب الدوال تتقاسم المتغيرات الشاملة global.
4. تنتقل البيانات بصورة مفتوحة حول النظام من دالة إلى دالة أخرى.
5. تستخدم طريقة من أعلى إلى أسفل في تصميم البرامج.

9.1.2 العلاقة بين كائنات العالم الحقيقي وكائنات البرمجة

عندما فكر مخترعو البرمجة الكائنة بهذا المفهوم الجديد كل ما كان لديهم في ذلك الوقت هو تسهيل البرمجة بأكبر فرصة لتصبح مشابهة للتصرفات على الواقع تماما فكر المخترعون على طريقة لإبعاد المبرمج كليا عن طريقة عمل كائن ما في البرمجة بحيث يركز عمله فقط على كيفية استعماله !!

لتركيز هذا المفهوم في الواقع خذ عندك مثالا: لعبة رجل ألي يلعب بها طفل ويحركها بيديه ويضغط فيها أزرارا لتصدر بعض الأصوات والحركات وتنفذ بطاريتها فتتوقف عن اللعب ويرميها في الأرض فتتحطم !!

الآن هذا الطفل لن يعرف مطلقا كيف يعمل هذا الرجل الآلي كيف يتحرك إذا ضغطنا هذا الزر كيف يصدر صوتا إذا ضغطنا ذلك الزر !!

هذا مشابه تماما لما يريدنا مخترعو ال OOP الوصول إليه أن نتحكم بالكائنات بكل سهولة دون الدخول في تفاصيل طريقة عملها ومن هنا بزغ فجر مفهومين جديدين للبرمجة " صانعو الفئات "ومستخدمو الفئات .

فالكائنات Objects في عالم الواقع يمكن تمييزها بشيئين : كل كائن في عالم الواقع لديه بيانات data و سلوك behaviour . فمثلا جهاز التلفاز هو كائن و يعالج بيانات بطريقة تجعلها تتضبط من خلال قناة محددة، معدّل المسح يتم تحديده إلى قيمة معيّنة، كذلك معدّل التباين و شدّة الإضاءة و هكذا. التلفاز أيضا يمكنه "يقوم" بأشياء، التلفاز يمكنه التشغيل أو الإقفال، القنوات يمكن تغييرها، و هكذا. على المنوال نفسه إذاً، فان "كائنات" العالم الحقيقي بالإمكان قولبتها بطريقة مشابهة للقوالب البرمجية.

لهذا السبب، نسمّي هذه القوالب بالكائنات Objects و منها جاء مصطلح البرمجة / التصميم بالمنحى للكائن Object Oriented Design/Programming.

تصنف الكائنات إلى صنفين:

- كائنات نشطة حية (Animate Objects) وهي التي نحس فيها فنجد لها حركة ونشاط.
- كائنات غير نشطة غير حية (Inanimate Objects) هي التي لا نلاحظ لها نشاط أو حركة أو وقع أينما وجدت .

وجميع الكائنات بصنفيها لها:

1. خصائص Attribute مثل :الحجم، اللون، الوزن، الشكل...ألخ .
2. سلوك Behavior فمثلاً: الطفل (كائن) يبكي، وينام، ويمشي، ويأكل (سلوكيات) .

الإنسان وخصوصاً المبرمج يتعلم عن الكائنات بمعرفة خصائصها، وملاحظة (تجربة) سلوكها، فمن الممكن أن يكون لكائنات مختلفة نفس الخصائص وسلوك متقارب.
حيث أن نظمنا البرمجية تقدّم حلولا لمشاكل حقيقية في واقعنا (سواء كان ذلك نظام تسجيل في معهد، أو نظام إدارة مخازن ، أو نظام توجيه صواريخ) ، يمكننا تحديد الكائنات في العالم الواقعي و بسهولة نقوم بتحويلها إلى كائنات برمجية.

وصناع الفئات هم كما في لعبة الرجل الآلي الشركة المصنعة لهذه اللعبة والمستخدمون هم الأطفال الذين يلعبون بها ولا يعلمون شيئا عن طريقة عملها الداخلية فقط يصدر الصانعون Manual لطريقة الاستخدام لكي يعرف الطفل كيف يستمتع بها وهو تماما ما يحدث في كائنات ال OOP .
الآن هل يمكن فعلا أن تكون البرمجة بهذه السهولة ؟
أقول نعم إذا ركز كل على عمله ، مصنعو الفئات سيكون بالطبع عليهم العبء الأكبر المستخدمون قد يكون عليهم عبء وقد يكونون في قمة حالات الاستمتاع بهذا الكائن ، حالات الاستمتاع في الواقع كما لدينا الطفل الذي يلعب بالكائن الآلي الكامل وهي أخر مراحل استخدام الكائن لأن الطفل لن يستخدم الكائن ليطوره لكائن أخر " إلا إذا كنا في عالم ال Matrix ونحن لا نعلم ! فقط سيكتفي باللعب به ، أما لو كنا في مثال أخر لو كان الكائن الحالي لدينا هو عبارة عن محرك سيكون هناك بعض العبء على مستخدم الكائن الذي سيقوم بتركيبه مع عدة كائنات أخرى ليكون في النهاية كائنا جديدا . هنا نحن لم ننتهي من سلسلة التطوير لهذا الكائن بعد فيمكن اعتبار المستخدمين مطورين بهذا المفهوم مطورو المحركات سيبيعونها لمصنعين أخريين وبهذا التكامل نبني واقعنا في الحياة نفس المفهوم تماما موجود في عالم البرمجة OOP لكن من يستطيع الوصول لهذه المراحل من التطوير من قال أنه لا يوجد لو دخلت ورأيت برمجة الألعاب ستجد العجب العجاب .
ولو اضطلعت على نماذج محاكاة الواقع الافتراضي فهي القمة في استخدام الكائنات لإنها تبنى أساسا على محاولة محاكاة كائن في الطبيعة بشكل حقيقي تماما ليعمل على الكمبيوتر بنفس طريقة عمله في الطبيعة مثلا متتابعات الأشعة ومحاكاة حركة الرياح والأعاصير محاكيات أحوال الطقس محاكيات التفاعلات الكيميائية وغيرها .
مثلا في محاكيات التفاعلات الكيميائية سيكون المطورون بداية كائن هو عبارة عن ذرة بالألكتروناتها ونواتها وبوزوتروناتها وبروتوناتها وكل محتوياتها هذا الكائن سيدمج في كائن أكبر منه وهو الجزيء سيتكون من عدة كائنات ذرة ثم نتدرج حتى نصل إلى المادة الكيميائية ويكون مبرمجو الكائنات السفلية قد اطلعوا على كيفية تفاعل الجزيئات مع بعضها بشكل تام ثم يبدؤون بكتابة " الدوال ، التي ستقوم بعملية الالتحام الدمج بين الجزيئات ويملإنها بكل تفاصيل التفاعلات في النهاية فقط ما على مستخدم الكائن النهائي وهو " كائن بيانات المحلول " إن ندخل له بيانات المحلول الأول والثاني ونطلب منه أن يتفاعل بينهما وننتظر نتيجة التفاعل !
هذه الأشياء بالطبع تحتاج لكمبيوترات عملاقة سريعة لتنفذ كل هذا الكم من التعليمات .

لكن يمكن التدرج وصولا لمستويات مبرمجي الألعاب حيث يقومون ببناء ألعابهم على أساس الكائنات مثلا خذ عندك لعبة بلياردو وهو مثال أوضح نوعا ما .
ما على مطوري اللعبة إلا استخدام كائنات كرة بلياردو " لإنها الجزء الأصعب " كائن البلياردو هذا سيتعامل كما في الحياة الواقعية تماما سيكون الكائن عبارة عن جسم كروي له كتلة بافتراض أن الجاذبية الأرضية 9.8 سنعطيه أيضا مكان لتخزين معلومات طاقته الحركية وطاقته الكامنة فكل ما علينا هو كتابة دالة لتقوم بعملية التصادم بحيث أن كل كرة عندما تصطدم بكرة أخرى ستستمد طاقة حركية وطاقة كامنة داخلها بهذا المبدأ يمكن أن نحرك كراتنا وننسى تماما كيفية تصادمها وانعكاسها !
أردت أن أبين هذه الأمور لإنها الأشياء التي عقدت الكثير في فهما بالشكل الصحيح لم لأجد كتابا يتحدث عنها بالشكل المفروض كل الكتب تعطي أمثلة سطحية سريعة مباشرة لا تعبر عن الاستخدام الأمثل للكائنات .
فمثلا لو قلت لك مثال مصعد كهربائي هو عبارة عن كائن ستقول لي يمكن أكتب برنامجا كهذا دون ا لدخول في تفاصيل الكائنات باستخدام لغة إجرائية بسيطة ! فيصبح المبرمج المبتدئ الذي سيكون ضيق الأفق في البداية مشوشا لا يعرف الاستخدام الأمثل لهذه التقنية .
أخيرا قبل أن أنتهي من هذه المقدمة الفلسفية أقول أن المستقبل سيحمل فقط لغات كائنيه المنحى من لم يرد ا لدخول في ذلك سيسقط وما عليه إلا بانتظار قدره . فأمر الكائنات ليس معقدا بل مفهومه مختلف فقط و لنفرض أن لدينا مبرمج يريد إنشاء لعبة تصويب ثلاثية الأبعاد مثل Quake3 مثلا بها شخصيات وأناس يتحركون ويتصرفون بشكل ذكي وكأن لهم عقول يفكرون بها الآن انظر إلى حال أ حد المبرمجين القابعين أمام أجهزتهم كل الوقت وهو يكتب كود تكامل اللعبة مع بعضها لولا الكائنات في البرمجة لظل هذا المبرمج 6 سنوات وهو يحاول أن يتكامل بين ألاف الأجزاء في مقابل أن يجلس سنتين فقط وهو يستعمل كود للكائنات .
"الألعاب الكبيرة تستغرق فترة متوسطها سنتين " .
الآن لنفرض مثلا أن هذا المبرمج لا يستخدم كود كائني سيضطر في كل frame أن يتكفل بتحريك كل شخصية في اللعبة ويقلق بشأن تصرفها هل هو سليم أم لا هل تعدى الكائن الأخر حدود المشهد أم لا هل اصطدم شخصين مع بعضهما في المشهد أم لا سيجن جنونه وهو يحاول ملاحقة هذه الاحتمالات وكل تعديل طفيف سيأخذ منه وقتا كبيرا وكل تعديل كبير يمكن أ ن يؤدي بالمشروع إلى الهاوية هذا بالنسبة لحال مبرمج واحد فما بالك إذا تشارك فريق لتطوير اللعبة يجب عليهم أولا أن يتواجدوا في مكان واحد واحتمال تضارب الأكواد بينهم كبير لدرجة تجعل من المستحيل تنفيذ المشروع .
في المقابل افرض أن مبرمجنا يستخدم كود كائني المذ حى سيتم تقسيم أعضاء المشروع إلى فرق كل فريق له مهمة واضحة محددة كالشمس .
مثلا الفريق الذي سيهتم بكتابة كود الشخصيات سيقوم بكتابة فئة تعرف الشخصية ويضع كل الاحتمالات الممكنة لهذه الشخصية الحركة التخاطب الأصوات حدود المشهد التصادم بين

الشخصيات ماذا لو اصطد مت الشخصية بأخرى قد ترتد وتصدر صوتا مثلا أو غيرها من الاستجابات .
حتى الآن هذا الكائن بدأ يتجسد بالطبع بعد مكاملة فريق رسم الشخصية مع المبرمج يبقى أمر مهم بث الحياة في هذه الشخصية ! كيف يمكن بث الحياة فيها بعد تعريف الفئة وتعريف كل المتغيرات الضرورية فيه ا والدوال التي ستقوم بالأمور المهمة تبقى الدالة الأب التي هي في الواقع كإنها العقل البشري الذي يحدد ما يجب فعله حسب التغيرات الخارجية " لا يمكن بالطبع جعلها تتصرف كالعقل البشري " مثلا لنفرض أن الشخصية ستكون حارس لبوابة وكل من يقترب من هذه البوابة سيتم التصدي له .
سنكتب دالة اسمها UpDate يتم استدعاءها كل Frame مثلا بحيث يتم مسح دائرة نصف قطرها 8 أمتار من الشخصية وإذا وجدت شخصية أخرى في هذا المدى تستدعى دالة أخرى لتحفيز القتال ! دالة تحفيز القتال ستستدعي دالة لتغير وضعية الشخصية الرسوميه ثم تستدعي دالة الهج وم وهكذا بسلسلة كهذه من الاحتمالات الأساسية يكون لدينا في النهاية مقاتل صنديد يتصرف بتلقائية وبالشكل المطلوب .
الآن فلنعد لمبرمجنا الذي كان سيقضي 6 سنوات وكأنه يقضيها في السجن ونعطيه فئة الشخصية وأنواع الشخصيات الأخرى سيكون سعيدا جدا لأنه لن يفعل شيئا في كل Frame إلا أنه سيستدعي الدالة UpDate كل مرة وينتهي الأمر ! لأن الدالة هي التي ستجعل الكائن يتصرف هكذا يمكن لكل عضو في الفريق أن يركز فقط على عمله وبشكل مدهش. وأن يعملوا مع بعضهم بشكل فعال حتى لو كان بينهم ألاف الأميال !
الآن هذه الفئة فئة الشخصية حجمها قد يكون كبير لكن مبرمج اللعبة لن يقلق بشإنها فليس له أي علاقة بحجمها فقط كل ما عليه هو أن يضعها ويقرأ طريقة استخدام ها وينسى كل شيء ، ويعتمد على أن مبرمج الفئة قد أتقن عمله فعلا هنا تقريبا يكمن العبء الأكبر على مبرمج الفئة حيث يجب أن يكون حذرا ويتأكد بشكل كبير من عمل الفئة بالشكل الصحيح .

9.1.3 خواص البرمجة الموجهة

ويمكن تلخيص خواص البرمجة الموجهة:

1- تركز على البيانات فضلاً على الإجراءات.
2- البرامج تقسم إلى ما يسمى Objects (أشياء).
3- هياكل البيانات تصمم بحيث تعكس خواص OOP.
4- الدوال تترابط مع بعضها البعض في هياكل البيانات.
5- البيانات مخفية hidden ولا يمكن الوصول إليها من قبل دالة خارجية.
6- الأشياء Objects يمكن أن تتصل مع بعضها ، من خلال مخاطبة مسلكياتها.
7- البيانات الجديدة والدوال يمكن أن تضاف بسهولة عند الحاجة.
8- تستخدم طريقة الأسفل إلى الأعلى bottom-up في تصميم البرامج.

9- طريقة التفكير تختلف عن الاتجاه المهيكل.
10- نقوم بالجمع بين البيانات و التصرفات ذات العلاقة داخل أصناف.

لماذا الكائنات مهمة جدا ؟

هناك الكثير من الأسباب ، دعني أعطيك بعضاها :
1- قدرتك على معرفة مكان الخطأ بسهولة إذا حصل.
2- القدرة على تتطوير البرنامج بسهولة مع الوقت.
3- القدرة على إعادة استخدام الكثير من أجزاء البرنامج لتطوير برامج أخرى.
4- عدم الحاجة لإعادة كتابة الشفرة البرمجية عند كل إصدار جديد للبرنامج.
5- سهولة تحويل الشفرة البرمجية للغة مختلفة.
6- القدرة على توزيع العمل في برنامج واحد ضخم على أكثر من مبرمج بسهولة ويسر.

9.1.4 إستراتجية المنحى الكائني

بالرغم من أن هذا الفصل قد لمس باختصار فوائد المنحى للكائن (مثل: منظومات أكثر ثباتا، تمثيل أفضل للواقع) ، إلا أننا تركنا بعض الأسئلة بدون إجابة. كيف نميّز الكائنات التي نحتاجها عند تصميمنا لمنظومة ما؟
ما هي السمات attributes المفترض وجودها؟ ما هو الحجم المناسب للصنف؟ و غيرها من الأسئلة.

أحد أهم نقاط ضعف المنحى للكائن في الماضي هو إنها في الوقت الذي تتميّز فيه بإنها قوية على مستوى الصنف/الكائن، إلا إنها ضعيفة عند التعبير عن سلوك المنظومة ككلّ. النظر من خلال الأصناف شيء جيّد، لكن الأصناف في حد ذاتها هي كينونات على مستوى منخفض و لا يمكن لها أن تصف ما تقوم به المنظومة ككل. باستخدام الأصنا ف فقط فإن الأمر يشبه محاولة فهم كيفية عمل الحاسوب من خلال فحص مكونات اللوحة الأم!

الاتجاه الحديث و المدعوم بقوة من قبل UML هو نسيان كلّ ما يتعلّق بالكائنات و الأصناف في المراحل المبكّرة للمشروع، و التركيز بدل ذلك على ما يجب أن تكون المنظومة قادرة على القيام به. بعد ذلك، و مع تقدّم العمل في المشروع يتم تدريجيا بناء الأصناف لتجسيد النواحي الوظيفية للمنظومة المطلوبة.

9.2 أنواع البيانات التجريدية (ADT) Abstract Data Type

في جميع لغات البرمجة توجد أنواع معرفة مسبقاً من قبل مترجم اللغة. مثلاً int يعتبر نوع نقوم باستخدام ه لمعالجة Manipulate الأرقام الصحيحة ، والمقصود بكلمة معالجة هنا هو إجراء العمليات المعتادة على هذا النوع مثل "- + * /" وهكذا ، لأن الجمع والطرح والضرب عمليات تجرى عادة على الأرقام.

ADT هي قاعدة الأساس في البرمجة بطريقة OOP ، وهي الخطوة الأولى التي يتم فيها تصميم وتعريف أنواع جديدة. هذه الأنواع الجديدة يتم تعريفها وترجمتها من أوصاف وأفعال النوع نفسه، فالأوصاف يتم التعبير عنها بمتغيرات Variables ، و الأفعال يتم التعبير عنها بدوال Functionsكما في الجدول 1-9.

جدول 1-9

الشئ / الكائن	أوصاف	أفعال
سيارة	لون ، عدد الأبواب ، لون ، سرعة ، نوع ، أسم ،..	تشغيل ، إيقاف ، تغيير سرعات ، ...
صالون	لون ، نوع القماش ،عدد الكراسي ، ...	تركيب ، تنظيف ، فك...

الأوصاف والأفعال التي يتم تعريفها لهذه الأنواع الجديدة تختلف من برنامج إلى أخر ، فمثلاً لو أردنا أن نقوم بتعريف نوع سيارة لاستخدام ه في برنامج تحكم بالسرعات فإن لون السيارة سيكون غير مهم بالنسبة لنا.

يسمى هذا النوع من البيانات بالبيانات التجريدية وذلك لإنها لازالت تحتاج إلى تعريف محدد لاستخدام ها من قبل لغات البرمجة ، فنوع القماش مثلاً عبارة عن معلومة يجب ترجمتها حتى يمكن التعبير عنها ، فمثلاً :

```
int Fabric;  // 1=Blue, 2=Red….
```

أو

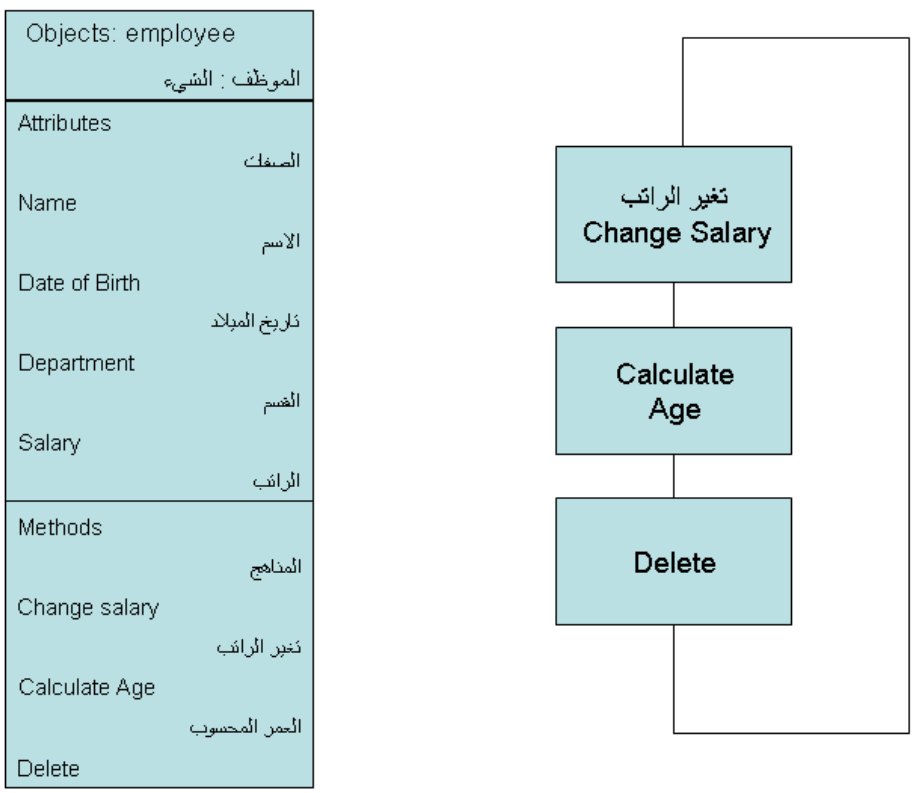
```
char Fabric; // B=blue, R=red, …
```

9.3 الأشياء (Objects)

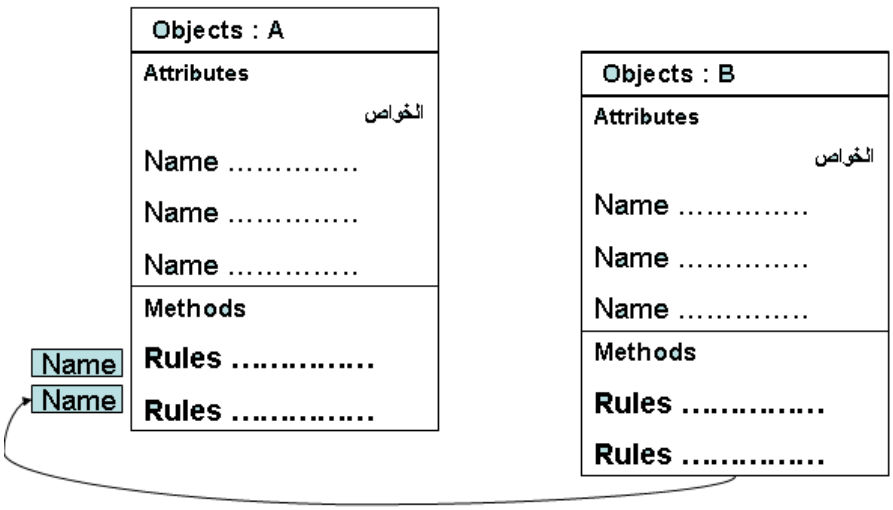
تعد الأشياء الوحدة الأساسية في نظام البرمجة الشيئية OOP system حيث يمكن أن نمثل شخص ، مكان ، حساب في مصرف ، أو يمكن أن نمثل بيانات تعرف من قبل المستخدم مثل المتجهات vectors، القوائم lists وتأخذ الأشياء مساحة في الذاكرة ولها عنوان مثل القيود Records . لذلك نستطيع أن نقول أن الكلمة شيء Object معنى محدد في البرمجة

الشيئية OOP فهو يمتلك خواص ومنهجية، وهوية فبالنسبة للخواص Attribute تشير إلى البيانات التي يحفظها كل شيء خاص به، أما منهجية Methods فتشير إلى القواعد التي تحكم سلوك الشيء Object Behavior كاستجابة الرسائل حيث يمثل إرسال الرسالة Message هو العمل الذي يؤديه الشيء كطلب من شيء أخر. كما يحتاج كل شيء إلى هوية Identity فريدة لتميزه عن كل الأشياء الأخرى.

ويمكن رسم الأشياء كصندوق له حافات دائرية له اسم الشيء وتقع خواصه ومنهجيته داخل الصندوق والشكل 3-9 يمثل رسمين الأول (a) يمثل طرقتين لتمثيل شيء وهو أما الرسم الثاني (b) فيمثل شيئيين أحداهما يرسل رسالة للأخر.



شكل 3-9 (a) طريقة لتمثيل الشيء



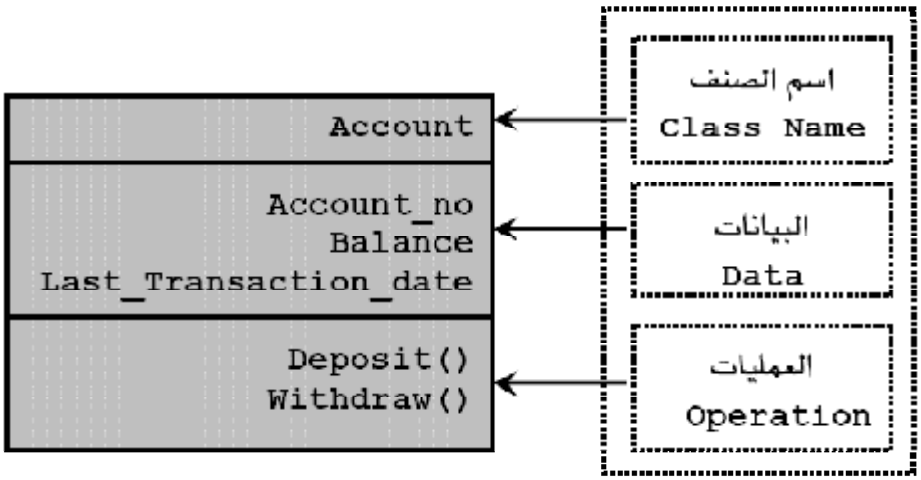
شكل 3-9 (b) يوضح صندوقين لشيئين ترسل أحداهما رسالة للأخر

9.4 الصنف class

أن مجموعة البيانات والبرامج لشيء ما يمكن أن تكون نوع من البيانات المعرفة من قبل المستخدم وذلك باستخدام الصنف class ، لذلك فإن الصنف يعد فكرة OOP التي تغلف البيانات والإجراءات المتطلبة لوصف المحتوى والسلوك لوجود العالم الحقيقي . وحالما يعرف Class يمكن أن تولد عدد من الأشياء Object التي تنتمي إلى Class . وفي ضوء ذلك فإن الصنف Class يعد مجموعة من الأشياء Object المتشابه النوع فعلى سبيل المثال فإن صنف الفاكهة يضم أنواعاً عديدة مثل التفاح والبرتقال و المنجة وغيرها.

ويمكن القول أن شيئان إلى صنف واحد متى ما قدمت نفس السطح البيني للعام الخارجي، فضلاً عن استخدام هما نفس هيكل الخواص في تخزين بيا ناتها الداخلية ونفس المنهجية في الاستخدام للرسائل.

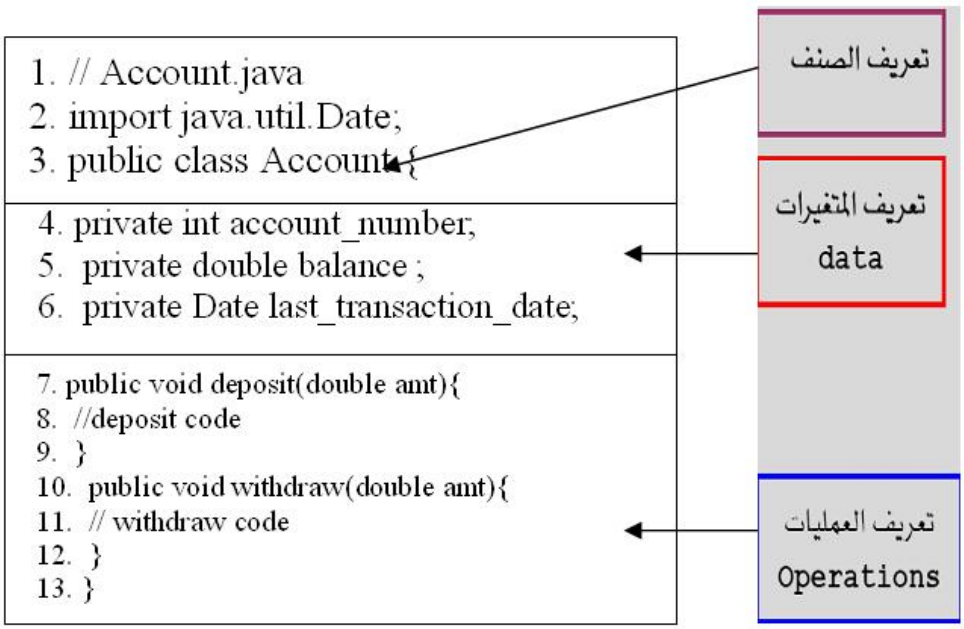
ويضم الصنف Class سطح بيني Interface وهيكل خواص ومنهجية للأشياء التي تنتمي لها. حيث يبين شكل 4-9 بسيط جدا من صنف الحساب البنكي ، والذي يحوي على البيانات والعمليات :



شكل 4-9 يبين صنف الحساب البنكي

9.4.1 إنشاء الأهداف

يتم تعريف الأصناف في لغة Java عن طريق الكلمة المحجوزة class، حيث يتبعها اسم الصنف، وعند اختيار اسم للصنف لا بد من تطبيق القواعد الخاصة بالأسماء . والمثال Account.Java يبين كيفية تعريف الصنف Account ، لكن دون وجود جمل تنفيذية لأنه للتوضيح فقط.



شرح المثال:

كما نلاحظ في المثال فإن عملية تعريف الأصناف تكون بالطريقة الآتية:

- نبدأ باسم الصنف (class name) ويمكن أن يكون مسبوقاً بكلمة public (وتعني عام) وهذا يعني أنة يمكن لأي صنف آخر أن يقوم بإنشاء نسخ من هذا الصنف ، أما إذا لم توضع الكلمة public في عملية التعريف فأن الأصناف داخل نفس الحزمة التي يوجد بها هذا الصنف هي وحدها تستطيع إنشاء نسخ من هذا الصنف.
- ثم بعد ذلك نبدأ بتعريف المتغيرات كما في الأسطر(4-6) وكما نلاحظ فإن المتغيرات مسبوقة بكلمة private (وتعني خاص) وهذا يعني أن المتغيرات يمكن التعامل معا داخل هذا الصنف فقط، أما إذا كانت مسبوقة بالكلمة public فإن جميع الأصناف يمكنها التعامل مع هذه المتغيرات بعد إنشاء نسخة من هذا الصنف، أما إذا لم ن ضع شيء فإن الأصناف داخل نفس الحزمة التي يوجد بها هذا الصنف هي وحدها تستطيع التعامل مع هذه المتغيرات .
- وفي الأسطر (7-12) تم تعريف العمليات (المناهج) على الصنف.

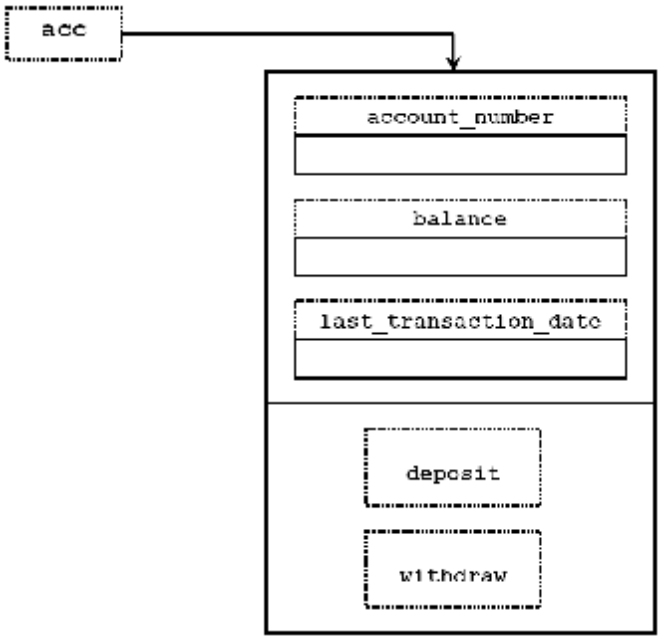
9.4.2 إنشاء الأصناف والوصول لمكوناته

والآن بعد ما لاحظنا كيف يتم تعريف الأصناف لنرى كيف يتم استخدام هذه الأصناف: تتم عملية استخدام الأصناف وذلك عن طريق إنشاء كائنات (Objects) تكون على شكل نسخ من الصنف ، وبالتالي يتم التعامل نه هذه الكائنات (النسخ) ، وتتم عملية إنشاء النسخ على النحو التالي

- تعريف متغير من نوع الصنف المراد استخدام ه والذي تم تعريفه مسبقاً.
- إنشاء كائن حقيقي من نفس الصنف وذلك باستخدام كلمة new متبوعة بإحدى البانيات (constructors) .
- ثم بعد ذلك يتم التعامل مع الكائن باستخدام اسم المتغير الذي يشير إليه متبوعاً بنقطة ثم بأحد المتغيرات أو المناهج حسب أمكانية الوصول(public , private , protected , default) . والمثال التالي يبين كيفية إنشاء كائن من الصنف Account الذي تم تعريفه في المثال Account.

```
1. // Bank
2.  public class Chp9_1 {
3.  public static void main(String[] args) {
4.  Account acc =new Account();
5.  acc.deposit(1000);
6.                          }
7.  }
```

شرح المثال:
في السطر4 تم تعريف وإنشاء المتغير acc ليصبح كائن من نوع account وهذا يعني أن المتغير acc يشير إلى كائن من نوع new تقوم بإنشاء هذا الكائن بعد استدعاء احدى البانيات الخاصة بالصنف Account . وفي السطر 5 تمت استدعاء الطريقة deposit في داخل الكائن acc ، وذلك بكتابة اسم الكائن متبوعا ياسم المنهج يفصل بينها نقطة. والشكل 9-5 يبين محتوى الكائن acc.



شكل 5-9 يبين محتوى الكائن acc

- استخدامات الكلمة ألمفتاحيه (new)

1) يتم إنشاء هدف من الفصيلة المعلن عنها .
2) يتم حجز جزء من الذاكرة لهذا الهدف .
3) يتم استدعاء دالة البناء الخاصة بهذه الفصيلة .

9.4.3 دوال البناء Constructor

وهو عبارة عن طريقة التكوين التي يتم بها إنشاء العضو من الفصيلة فتأخذ نفس اسم الفصيلة وتنفذ عند إنشاء الفصيلة و من التنويهات لها

 تستطيع إعطاء قيم ابتدائية لمتغيرات فصيلة الهدف .

 يجب أن تملك دالة البناء نفس اسم Class .


لا تملك البانيات نمط إرجاع ولا حتى void .


يتم استدعاء البانيات باستخدام العامل new عند إنشاء الكائن .


إذا لم يعرف Class الرئيسي أي باني بشكل صريح فأنة تلقائياً يتم تعريف باني افتراضي وهذا الكود يبين ذلك.

```
1. class Bird {}
2. class Chp9_2 {
3.   public static void main(String[] args) {
4.     Bird nc = new Bird();
5.                   }
6. }
```


من الأخطاء المرتكبة وضع كلمة void or int قبل اسم الباني فيصبح طريقة وليس بانياً .


يمكن تعريف ال Constructor بحيث يكون فارغاً من الكود لأسباب .


إذا قمت بتعريف constructor خاص بك، فأنك تفقد الـ constructor الافتراضي، فإذا أردت أن تحتفظ به، عليك أن تقوم بكتابته يدوياً .


يمكن أن يكون لنفس الفئة أكثر من Constructor يختلفون في أعداد أو أنواع المتغيرات في سلسلة المتغيرات الممررة لهم، أو كلاهما .


شكل الـ Constructor قد يشبه شكل الوظيفة، و لكن تذكّروا دائماً. اسم الـ Constructor هو نفس اسم الفئة، و لا يوجد له نوع بعكس الوظيفة .


إن عمل new هو إنشاء العضو من الـ constructor المناسب. فإذا لم يوجد constructor في الفئة تقوم new بـ استخدام الـ constructor الافتراضي .

```
1. // برنامج الباني الأفتراضي
2. class Chp9_3{
3.         public static void main(String args[]){
4.                 new Chp9_3();
5.                 System.out.println("by");
6.                 new Chp9_3(10);
7.                                                 }
8. Chp9_3( ){ System.out.println("yes");}
9. Chp9_3(int b){ System.out.println(b+b);}
10. }
```

```
yes
by
20
```

شرح المثال:

نلاحظ من كودنا السابق انه عندما تريد تفعيل دوال بناء Class نفسه يتم بـ new تم اسم دالة البناء أو بإنشاء هدف من نوع Class ;Chp9_3() m=new Chp9_3 .


دوال البناء تسبق أي جمل مكتوبة بداخل الكود فينفذ البرنامج أولا داله البناء الأولى ثم الثانية ثم جملة الطباعة .

وفي كودنا الأتي يختلف الأمر

```
1. //برنامج يوضح أسبقية تنفيذ دوال البناء
2. class Chp9_4 {
3.         //الكتل المعشعشة
4.         {System.out.println("by");}
5.
6. public static void main(String args[]){
7.         new Chp9_4();
8.     new Chp9_4(10);
9.                                                 }
10.
11. Chp9_4( ){System.out.println("yes");}
12.
13. Chp9_4( int b){System.out.println(b+b);}
14. }
```

```
by
yes
by
20
```

شرح المثال:

نلاحظ أنة تم تنفيذ جملة الطباعة التي خارج الدالة الرئيسية التي بداخل Block ات المسماة الكتل المعشعشة في السطر 4 مرتان مرة عند تنفيذ دالة البناء الأولى ومرة عند تنفيذ دالة البناء الثانية 0

ومن الملاحظ أن في المثال السابق Chp9_3 كانت دالة البناء تسبق أي جمل مكتوبة ولا كن هنا الأمر يختلف الآن فأي جمل مكتوبة بداخل بلوكات وموقعها خارج الدالة الرئيسية فإنها تسبق تنفيذ دالة البناء .

جملة الطباعة السابقة في السطر 3 لا تنفذ إطلاقا إلى إذا حدث إطلاق للفئة.

```
1. //عدم تنفيذ دالة البناء
2. class Chp9_5 {
3.         {System.out.println("by");}
4. public static void main(String args[]){
5.     System.out.println("start");
6. }
7. }
```

```
start
```

في هذا الكود أيضا يختلف الأمر

```
1. //أسبقية تنفيذ الجمل الاستاتيكية
2. class Chp9_6 {
3. public static void main(String args[]){
4.          new Chp9_6();
5. }
6.
7. Chp9_6( ){      System.out.println("yes");}
8.            {System.out.println("by");}
9.
10.             static
11.                     {System.out.println("start");}
12. }
```

```
start
by
yes
```

شرح المثال:

نلاحظ أن المحرر نفذ محتوى static كما في السطر 10 ثم محتوى الكتلة المعشعشة في السطر 8 ثم دالة البناء في السطر 7 .

دائماً يتم تنفيذ محتوى static قبل كل شيء حتى وان لم يتم إطلاق Class كما في المثال التالي:

```
1. // تنفيذ الجمل الاستاتيكة
2. class Chp9_7 {
3. public static void main(String args[]){
4.                     System.out.println("by");
5.                                         }
6.
7.          static{System.out.println("start");}
8. }
```

9.4.4 تمرير البارامترات إلى الكائنات

لقد تعلمنا مسبقاً كيف نمرر البارامترات التي تنتمي إلى الآ ماط الأولية والمصفوفات إلى المناهج ، يمكنك أيضا تمرير البارامترات إلى الكائنات ، يقوم المثال التالي بتمرير قيمة متحول إلى الكائن عبر دالة البناء.

```
1. //بتمرير قيمة متحول إلى الكائن عبر دالة البناء
2. // this used implicitly and explicitly to refer to members of an object.
3.
4.  public class Chp9_8
5.  {
6.     public static void main( String args[] )
7.     {
8.        SimpleTime time = new SimpleTime( 15, 30, 19 );
9.        System.out.println( time.buildString() );
10.     } // end main
11.  } // end class ThisTest
12.
13.  // class SimpleTime demonstrates the "this" reference
14.  class SimpleTime
15.  {
16.     private int hour;   // 0-23
17.     private int minute; // 0-59
18.     private int second; // 0-59
19.
20.     // if the constructor uses parameter names identical to
21.     // instance variable names the "this" reference is
22.     // required to distinguish between names
23.     public SimpleTime( int hour, int minute, int second )
24.     {
25.        this.hour = hour;     // set "this" object's hour
26.        this.minute = minute; // set "this" object's minute
27.        this.second = second; // set "this" object's second
28.     } // end SimpleTime constructor
29.
30.     // use explicit and implicit "this" to call toUniversalString
31.     public String buildString()
32.     { String s="this.toUniversalString()\t"+ this.toUniversalString()+
33.          "\ntoUniversalString()\t\t"+ toUniversalString() ;
34.        return s;
35.
36.     } // end method buildString
37.
38.     // convert to String in universal-time format (HH:MM:SS)
39.     public String toUniversalString()
```

```
40.    {
41.       // "this" is not required here to access instance variables,
42.       // because method does not have local variables with same
43.       // names as instance variables
44.       String s=this.hour+":"+ this.minute+":"+ this.second;
45.       return s;
46.
47.    } // end method toUniversalString
48. } // end class SimpleTime
```

فيكون ناتج التنفيذ كما يلي:

```
this.toUniversalString()          15:30:19
toUniversalString()               15:30:19
```

9.4.5 مقارن الأهداف

في فصل أنواع البيانات تعرفنا إلى موضوع المؤثرات Operator (=> , < , > , =<) ، وهذه المؤثرات تعمل فقط مع الأنواع المبينة في اللغة وإذا حاولت استخدام ها مع الأهداف فسيقوم المترجم بإصدار رسائل خطاء.

ولكن يوجد استثناء من هذه المؤثرات وهما اثنان (== ، =!) فالأول يقوم بفحص المساواة والثاني يقوم بنفي المساواة ، ولكن في لأهداف يختلف عملها في الأهداف كما في المثال التالي:

```
1. // برنامج مقارنة الأهداف
2. class ammar{ int i ; ammar(int a){i=a;}}
3.
4. class Chp9_9 {
5.         public static void main(String args[]){
6.                   ammar a=new ammar(10);
7.                   ammar a1=new ammar(100);
8.                   System.out.println(a==a1);
9.
10.                  a=new ammar(100);
11.                  System.out.println(a==a1);
12.
13.                  a=a1;
14.                  System.out.println(a==a1);
15.
16.                  a.i=1000;
17.                  System.out.println(a1.i);
18.                                          }
19. }
```

```
false
false
true
1000
```

شرح المثال:

السطر 8 ليس بمعنى أن محتوى Class الأول يساوي محتوى Class الثاني لا ولكن هل موقع Class الأول يساوي موقع Class الثاني في الذاكرة .

السطر 13 رغم إننا ساوينا محتوى الكلاسين إلا إننا نجد ناتج أمر الطباعة false والسبب ذكرناه سابقاً.

السؤال هنا كيف تغيرت قيمة a1.i ونحن غيرنا قيمة a.i ؟
الإجابة هو عندما جعلنا a =a1 فأنة أصبح a يؤشر إلى موقع a1 وأي تغير في a.i تتغير قيمة a1.i والعكس كما في السطر 14.


هذا الكود أيضا به قاعدة :

```
1. // برنامج يوضح تغير الموقع الذاكري للفئة
2. class Chp9__10{ int i;  Chp9__10(int a){i=a;} }
3.
4. class Chp9_10 {
5.         public static void main(String args[]){
6.                 Chp9__10 a=new Chp9__10(10);
7.                 Chp9__10 a1= a;
8.                 System.out.println(a==a1);
9.                 System.out.println(a1.i);
10.
11.                a=new Chp9__10(100);
12.
13.                System.out.println(a==a1);
14.                System.out.println(a.i);
15.                System.out.println(a1.i);
16.                                        }
17. }
```

```
true
10
false
100
10
```

شرح المثال:

السطر 11 تم إعادة إطلاق الهدف وإرسال لـ I القيمة 100 بما معنى أن موقعة ألذاكري قد تغيـر.

9.4.6 تمرير الكائنات إلى المناهج

لقد تعلمنا مسبقاً كيف نمرر البارامترات التي تنتمي إلى الآذ ماط الأولية والمصفوفات إلى المناهج ، يمكنك أيضا تمرير الكائن إلى المنهج ، يقوم المثال التالي بتمرير الكائن كبارامترإلى المنهج (fun) .

```
1. // this used implicitly and explicitly to refer to members of an object.
2.  public class Chp9_11
3.  {
4.        static void fun(Chp9__11 x){
5.                                          System.out.println( x.Simple_fun() );
6.                                      }//end function fun
7.
8.    public static void main( String args[] )
9.    {
10.       Chp9__11 exp = new Chp9__11();
11.       fun(exp);
12.    } // end main
13. } // end class Chp9_11
14.
15. // class Simple
16. class Chp9__11
17. {
18.    public String Simple_fun()
19.    {
20.       return "Simpe";
21.    } // end SimpleTime constructor
22. } // end class Chp9__11
```

شرح المثال:

يستدعي المنهج main المنهج fun حيث يمرر له الكائن (exp) كبرامتر كما في السطر 11 ، فيقوم المنهج fun باستدعاء المنهج Simple_fun الذي بداخل الصنف Chp9__11 ، حيث تعيد قيمة رمزية من نوع String كما في السطر 20.
فيكون ناتج تنفيذ البرنامج :

Simpe

9.4.7 مصفوفة من الكائنات

والقصد هنا تكوين عدة كائنات من Class واحد وتتم بخطوتين

1) تعريف عدد الكائنات المراد تكوينها .
2) اشتقاق الكائنات .

كما في المثال التالي:

```
1. // برنامج مصفوفة الفئات
2. class Chp9__12 {static int count;
3.                                          int j;
4.              Chp9__12() {
5.                                          j=count;
6.                                          count++;
7.                                  }
8.                      }
9.
10. class Chp9_12 {
11.       public static void main(String[] args) {
12.                       int i=10,j;
13.                       // Create Objeact
14.          Chp9__12 p[] =new Chp9__12[i];
15.
16.          for(j=0;j<i;j++)
17.                       p[j]=new Chp9__12();
18.
19.          System.out.println("j"+" "+"count");
20.
21.          for(j=0;j<i;j++)
22.                       System.out.println(p[j].j+" "+p[j].count);
23.                                                  }
24. }
```

```
j count
0 10
1 10
2 10
3 10
4 10
5 10
6 10
7 10
8 10
9 10
```

شرح المثال:

السطر 14 تم تحديد عدد الفئات المراد تكوينها .

الأسطر 17 , 16 تم اشتقاق الفئات .

المتحول count في السطر 2 متحول عام لجميع الفئات المشتقة أي إي تغير لة في أي فئة تتغير قيمته في جميع الفئات بسبب كونه من نوع static.

9.4.8 الكلمة ألمفتاحيه (this):

استخدامات الكلمة ألمفتاحيه (this) / له دور كبير مع دوال البناء وعموماً هذه الكلمة تشير إلى Class نفسه ولها استخدامات .

1. تستدعي دوال البناء كما في المثال التالي:

```
1. // برنامج يستدعي دالة البناء
2. class Chp9__13{
3.                         Chp9__13(){System.out.println("start");}
4.                         Chp9__13(int b)
5.                         {this();System.out.println(b*b);}
6.                 }
7.
8. class Chp9_13 {
9.         public static void main(String args[]){
10.                 Chp9__13 b=new Chp9__13(100);
11.                                                 }
12. }
```

```
start
10000
```

عند استدعاء باني يجب وضع الكلمة ألمفتاحيه this قبل أي جملة وإلا المترجم سيصدر خطاء .

. (Constructor invocation must be the first thing in a method this())

وتعتبر من الأخطاء الشائعة.

فهذا الكود لن ينفذ إطلاقا .

```
1. //Error This
2. class Chp9__14{
3.          Chp9__14(){System.out.println("start");}
4.         Chp9__14(int b)
5.                      {System.out.println(b*b);
6.                       //Error
7.                       this();
8.                      }
9.           }
10.
11. class Chp9_14 {
12.         public static void main(String args[]){
13.                      Chp9__14 b=new Chp9__14(100);
14.                       }
15. }
```

2. تحل مشكلة أسماء الأعضاء المتشابهة .

```
1. // برنامج يحل مشكلة تشابة الأعضاء
2. class Chp9__15 {int b;
3.                          Chp9__15 (int b){ this.b=b;}
4.                        }
5.
6. class Chp9_15 {
7.       public static void main(String[] args) {
8.          Chp9__15 a = new Chp9__15(10);
9.          System.out.println(a.b);
10.                                                  }
11. }
```

```
10
```

شرح المثال:

في بعض الأحيان قد يتشابه اسم البارمتر المرسل للدالة كما في السطر 3 واسم المتغير في داخل Class كما في السطر 4 فجملة this تعني المتغير الخاص بالفئة نفسها.

3- تعيد قيمة من نوع Class نفسه مثل الدالة عندما تعيد قيمة.

وهذا المثال يبين ذلك :

```
1. // برنامج يعيد قيمة من نوع الكلاس
2. class Chp9__16{
3.                         int b=0;
4.                         Chp9__16 (){}
5.                         // Return This Object
6.                         Chp9__16 a(){b++;return this;}
7.                         void print(){System.out.println(b);}
8.                 }
9.
10. class Chp9_16 {
11.       public static void main(String args[]){
12.                 Chp9__16 q=new Chp9__16();
13.                 q.a().a().a().a().a().print();
14.                         }
15. }
```

5

لا نستطيع استخدام الكلمة this في استدعاء أكثر من دالة بناء .

9.4.9 الكلمة (final) : -

بما معنى ثابت أي لا نستطيع تغيره ولها عدة استعمالات فان ذكرت مع

- Final class / فإننا لا نستطيع توريث Class .
- Final function / فإننا لا نستطيع عمل override .

- Final variable / فإننا لا نستطيع تغير محتوى المتغيرات

وهذا مثال على ذلك:

```
1. // final instance variable in a class.
2.  class Increment
3.   {
4.      private int total = 0; // total of all increments
5.      private final int INCREMENT; // constant variable (uninitialized)
6.
7.      // constructor initializes final instance variable INCREMENT
8.      public Increment( int incrementValue )
9.      {
10.        INCREMENT = incrementValue; // initialize constant variable (once)
11.     } // end Increment constructor
12.
13.     // add INCREMENT to total
14.     public void addIncrementToTotal()
15.     {
16.       total += INCREMENT;
17.     } // end method addIncrementToTotal
18.
19.     // return String representation of an Increment object's data
20.     public String toString()
21.     {  String s="total = "+total;
22.       return s;
23.     } // end method toIncrementString
24.  } // end class Increment
25.
26.  // final variable initialized with a constructor argument.
27.   public class Chp9_17
28.   {
29.     public static void main( String args[] )
30.     {
31.       Increment value = new Increment( 5 );
32.
33.       System.out.println( "Before incrementing: \n\n"+ value );
34.
35.       for ( int i = 1; i <= 3; i++ )
36.       {
37.         value.addIncrementToTotal();
38.         System.out.println( "After increment :\n"+ i+" "+ value );
39.       } // end for
40.     } // end main
41.  } // end class IncrementTest
```

```
Before incrementing:

total = 0
After increment :
1 total = 5
After increment :
2 total = 10
After increment :
3 total = 15
```


كهذا الكود خاطئ .

```
1. //Error Class
2. public class Chp9_18 {
3.   public static void main(String[] args) {
4.         ammar++;//error
5.         System.out.println(ammar);
6.                                                }
7.   static final int ammar=20;
8. }
```

وهذا ناتج التنفيذ:

```
Can't assign a value to a final variable: ammar
```


هذا الكود خاطئ لماذا ؟

```
1. // Syntax Error
2. class Chp9_19 {
3.         int i=10;
4.         public static void main(String args[]){
5.                   int j=i;
6.                   System.out.println(j);
7.                   System.out.println(i);
8.                                                  }
9. }
```

```
Can't make a static reference to nonstatic variable i in class Chp9_19.
```

شرح المثال:

وهو استخدام متغير محجوب عن المترجم فلن يقدر أن يصل إلى I الذي في السطر 3 إلا إذا أطلقنا هدف من نوع Class أو نجعل I من نوع static.

عند التصريح على متغير من نوع static داخل Class فهذا يعني أن هذا المتغير مشترك لجميع الفئات المشتقة من Class وهذا الكود يشرح ذلك

```
1. // برنامج التصريح على متغير من نوع ستاتك داخل الكلاس
2. class Chp9__20{
3.                         static int i;
4.                         int j;
5.                         Chp9__20(int i,int j){this.i=i;this.j=j;}
6.                       }
7.
8. class Chp9_20 {
9.         public static void main(String args[]){
10.
11.                 Chp9__20 c =new Chp9__20(10,20),d=new Chp9__20(30,40);
12.                 System.out.println(c.i+" "+c.j);
13.                 System.out.println(d.i+" "+d.j);
14.                                                 }
15. }
```

```
30 20
30 40
```

شرح المثال:

نلاحظ انه عندما أرسلنا القيمة 30 إلى I للفئة d فان قيمة I قد تغيرت من 20 إلى 30 تبعاً للقيمة الأخيرة فهذا يعني أن أي تغير للمتغيرات من نوع static فأنة تتغير جميع متغيرات الفئات التي من نوع static كما في السطر 11.

هذا المثال يعتبر غريب نوعاً ما !!

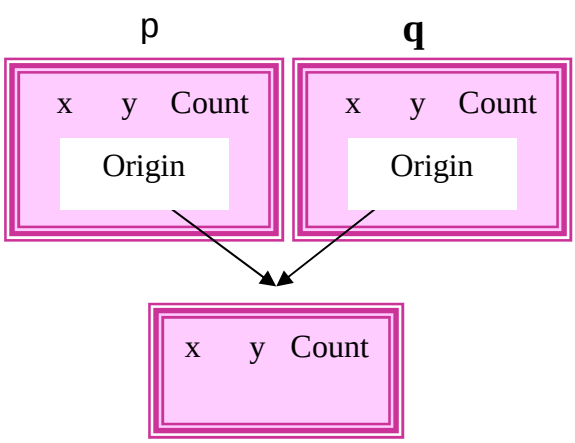
```
1. // برنامج يبين الأختلاف بين المتغيرات المحلية والعامة للكلاس
2. class Chp9__21 {
3.                             int x, y, Count;
4.                             Chp9__21(int x, int y) { this.x = x; this.y = y; }
5.                             static Chp9__21 origin = new Chp9__21(0, 0);
6.                               }
7. class Chp9_21 {
8.       public static void main(String[] args) {
9.          Chp9__21 p = new Chp9__21(1,1);
10.           Chp9__21 q = new Chp9__21(2,2);
11.           p.Count++; p.origin.Count++;
12.           System.out.println( p.x + "," + p.y );
13.           System.out.println(q.Count);
```

```
14.         System.out.println(q.origin == Chp9__21.origin);
15.         System.out.println(q.origin.Count);
16. }
17. }
```

```
1,1
0
true
1
```

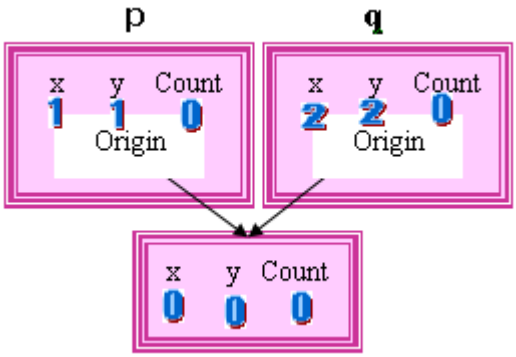
شرح المثال:

عند تتبع أي برنامج فعليك أن ترسم هيكل Class كما في الشكل 9-6 لكي يسهل عليك إيجاد خرج البرنامج :



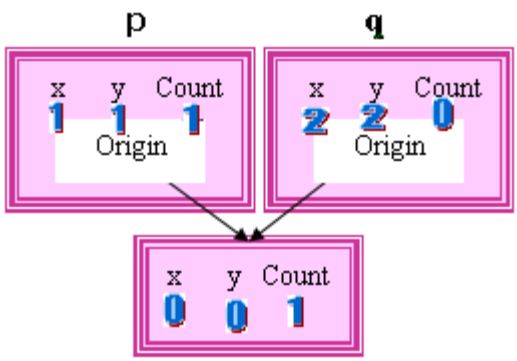
شكل 9-6

فمن الملاحظ إننا لدينا فئتان p , q وتم إسناد لهم (1,1) و (2,2) وداخل كل فئة فئة داخلية لها هيكل بنفس هيكل الفئة الخارجية فبعد اشتقاق الفئات وإرسال القيم لها كما في الأسطر (10 - 9) تكون الفئات كما في الشكل 9-7 :



شكل 9-7

بعد تنفيذ السطر 11 تصبح أشكال الفئات وقيمها داخل الذاكرة كما في الشكل 9-8:



شكل 9-8

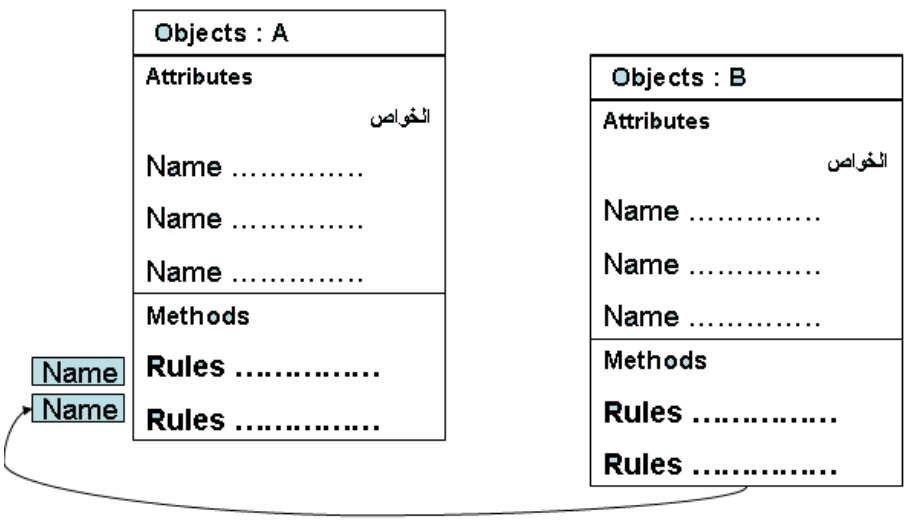
ومن خلال الإشكال السابقة يتضح خرج البرنامج .

9.5 الكبسلة Encapsulation.

إن عملية تغليف أو لف البيانات والدوال في وحدة واحدة تسمى التغليف (Encapsulation) ، وبذلك فإن خواص الشيء ومنهجيته تكون داخل الصندوق وبشكل حزمة واحدة ويتضح شكل أعلاه أن الصندوق مقسم إلى جزيئين:

الجزء العلوي يضم الخواص وفي حين أن الجزء السفلي يضم منهجيته. ومن الجدير بالذكر أنة لا يمكن الوصول إلى البيانات من المحيط الخارجي ولكن فقط الدوال الموجودة داخل الصنف class يمكن أن تصل إلية ، إن هذه الدوال تجهز بواسطة اتصال مابين بيانات الشيء والبرامج ، والطريقة الوحيدة للحصول على معلومات من الشيء تكون عن طريق إرسال رسالة إلى هذا الشيء وتكون المعلومات والقو اعد التي يستخدمها الشيء كاستجابة الرسالة مخفية عن العالم الخارجي information hiding.

ويتضح ومن الشكل 9-9 أن أسماء منهجية الشيء تكون بشكل مستطيل قد تتحدى حدود الصندوق (اجتيازها) لذلك فهي يمكن أن تمثل بأزرار button على الشيء ويتصل كل واحد منها بمنهجية الشيء بالضبط . ويشمل الأزرار الموجودة على الشيء السطح البيني للشيء Object Interface وكما يلاحظ من خارج الشيء .



شكل 9-9

قد تكون عزيزي القارئ عدنك خبرة سابقة في البرمجة وتقول أننا في البرمجة المهيكلة نستطيع أن نخفي ونغلف البيانات في الدالة بجعلها خاصة private بهذه الدالة فقط ومعك حق في ذلك ولكن ماذا لو أردنا جعل هذه البيانات Global وهذا صحيح ولكن ما الحل إذا أردنا جعل تلك البيانات متاحة لبعض الدوال وغير متاحة للأخرى والحل أن يكون عندنا Multiple Scope وهذا ما تؤديه لنا عملية Encapsulation فمن خلال الفصلية استطيع أن اجعل البيانات Global للدوال التي أريد أن تتوصل Access لهذه البيانات بان أضعهم سويا في فصيلة واحدة .

- **إخفاء البيانات Data Hiding و معدلات الرؤية والوصول :**

لقد خلق المبدع والأول والآخر سبحانه الclass الإنسان وجعل له صفات (members = variables) من عينين وأنف وأذن و قلب و الكثير من الصفات الأخرى وقد جعل الله هذه الصفات من الممكن أن نصل لها ونمسكها فنستطيع أن نمسك أعيننا و هناك الطبيب الجراح الذي يمسك القلب بيديه فهذه الصفات يستطيع أن يصل لها أي شيء فهي عامة (public) ، وخلق الله أيضاً صفات (members) في الإنسان مثل الروح ولكننا لا نستطيع أن نصل لها و نمسك بها. يقول تعإلى (و يسألونك عن الروح قل الروح من أمر ربي) فهي صفات خاصة (private) ممنوع أن تصل لها أي (class) أخرى . و تخيل لو كانت الروح من الممكن أن نصل لها ونمسكها مثل العين فكما إنه هناك من تمرض عي نه فيقوم بنقل عين إنسان آخر فلو كانت الروح من الممكن أن نصل لها (public) لوجدنا من لا تعجبه روحه فينقل ويستبدل روح إنسان آخر ! إذن فهناك (members) في ال class تكون public وأخرى private لا تستطيع class أخرى أن تصل لها مباشرة.
ولكن قد يقوم صانع ال class بعمل methods تؤثر في هذه ال private members ودون أن تصل لها. فمثلا جعل الله الخالق العظيم methods في الإنسان يستطيع بها أن يؤثر في الروح ومن هذه ال methods يصلي و يزكي ويصوم .
وخلق الخالق سبحانه وتعإلى أيضا جينات نمتلكها من أبوينا ولا يستطيع فرد خارج العائلة أن يمتلكها أو أن يحصل عليها مثل النخاع ...
في Java يستطيع ال class creator أن يحدد قواعد الوصول (access control) لل members في ال class من variables و methods. هذه access control هي : (public و private و protected و default) .

- Public / وهي عامة أي تستطيع الوصول إليها من خارج Class ومن خارج البرنامج أيضا بواسطة الحزم أو الواجهات وهذه مواضيع سيتم شرحها بالتفصيل في الفصول القادمة
- Private / أي بمعنى مخفية تستطيع الوصول له من داخل Class فقط ولا يمكن أن توصل له من خارج Class إطلاقا إلا عن طريق حيلة سيتم ذكرها لاحقاً وتستخدم في حالات
  1- عندما تكون الفصائل الأخرى لا تحتاج إلى استخدام تلك المتغيرات .
  2- عندما تخشى فصيلة خارجية من ألعبت بالبيانات .
- Protected / أي بمعنى محمي أي انك تستطيع الوصول لـ ه من داخل Class أو من خارج Class إذا كان Class يرث منة ولا تستطيع إطلاقا الوصول لهذا النوع من خارج الفصيلة دون وراثة وسيتم ذكرها في فصل الوراثة .

إذا لم يتم ذكر أي محدد من التي ذكرت فان لغة Java تعتبر المتغيرات والدوال و Class عام في داخل الحزمة فقط بخلاف لغة السي تعتبرها من نوع private .

جدول 2-9

مكان الوصول	public	default	Protected	Private
من داخل الفصيلة	ü	ü	ü	ü
من أي فصيلة داخل الحزمة	ü	ü	ü	û
من أي فصيلة خارج الحزمة	ü	û	û	û
من أي فصيلة فرعية أي مورثة داخل الحزمة	ü	ü	ü	û
من أي فصيلة فرعية خارج الحزمة	ü	û	ü	û

وهذا المثال يشرح ما سبق:

```
1. // برنامج يوضح محددات الوصول
2. class Chp9__22 {
3.       private int x, y;
4.       public int Count;
5.       Chp9__22(int x, int y) { this.x = x; this.y = y; }
6.       int get_x(){return x;}
7.       private int get_y(){return y;}
8.       void show(){System.out.println(get_y());}
9.                  }
10.
11. class Chp9_22 {
12.      public static void main(String[] args) {
13.          Chp9__22 p = new Chp9__22(1,10);
14.          p.Count++;
15.          //p.x++;//error
16.          //p.y++;//error
17.          //System.out.println( p.x + "," + p.y );//error
18.          System.out.println(p.get_x());
19.          //System.out.println(p.get_y());//error
20.          System.out.println(p.Count);
21.          p.show();
22. }
23. }
```


على الرغم من أنه يمكن استخدم المحددين public , private أكثر من مرة ، فينصح بذكر المتغيرات العامة أولاً ثم المتغيرات الخاصة ثانياً.

هذا المثال توجد به داله بناء من نوع محمي private ومطلوب منك تنفيذ الكود دون المساس بمحددات الوصول فكيف ستعالج هذه المشكلة ؟

```
1. // Invalid method declaration; return type required.
2. class Chp9__23 {
3.       private Chp9__23() { System.out.println("Start");}
4.         }
5. class Chp9_23 {
6.       public static void main(String[] args) {
7.           // Error No constructor matching Chp9__22() found in class Chp9__23.
8.          Chp9__23 p = new Chp9__23();
9.          System.out.println("End");
10.
11. }
12. }
```

حل المثال:

لديك طريقتين

الأولى: أن نصنع دالة من نوع static تعيد لنا قيمة من نوع دالة البناء المحمية كالسطر رقم 4 من هذا المثال:

```
1. // Chp9_23 حل برنامج
2. class Chp9__24 {
3.       private Chp9__24() { System.out.println("Start");}
4.       static Chp9__24 dop(){return new Chp9__24();}
5.           }
6.
7. class Chp9_24 {
8.       public static void main(String[] args) {
9.          Chp9__24 p =  Chp9__24.dop();
10.          System.out.println("End");
11.                                              }
12. }
```

وعند إنشاء الهدف نذكر اسم Class. اسم الدالة الاصطناعية كما في السطر 9.

نلاحظ أن هذه الطريقة مربكة نوعاً ما وطويلة بعض الشيء .

فإليكم هذه الطريقة الثانية والبسيطة:

نصنع دالة اصطناعية من نوع static وبداخلها إطلاق لدالة البناء المحمية . كم في السطر 5 من هذا المثال:

```
1. // حل االمثال بطريقة أخرى
2. class Chp9__25 {
3.             // Constructors
4.        private Chp9__25() { System.out.println("Start");}
5.        static void dop(){ new Chp9__25();}
6.          }
7. class Chp9_25 {
8.       public static void main(String[] args) {
9.           Chp9__25 p =null;
10.           p.dop();
11.           System.out.println("End");
12.                                                  }
13. }
```

ثم عند إنشاء الفئة نجعلها تساوي null كما في السطر 9. فبهذا الشكل لن يتم تنفيذ دالة البناء إطلاقا.

ثم نذكر اسم الفئة. اسم الدالة الاصطناعية ، كما في السطر 10.

فالسؤال هنا أيهما أفضل المثال Chp9_24 أم المثال Chp9_25 ولماذا ؟

9.6 مجال تغطية المتغيرات بشكل أوسع

بشكل عام لكي تفهم معنى مجال تغطية المتغيرات تتبع هذا المثال:

```
1. // برنامج يوضح مجال تغطية المتحولات
2. class Chp9_26 {
3.          static int i=10;
4.     static void print(int i){System.out.println(i);
5.                                    System.out.println(Chp9_26.i);
6.                                  }
7. public static void main(String args[]){
```

```
8.        int i=100;
9.        System.out.println(Chp9_26.i);
10.       System.out.println(i);
11.       print(1000);
12.                                   }
13. }
```

```
10
100
1000
10
```

شرح المثال:

جملة الطباعة الموجودة في السطر 4 تشير إلى البارامتر الداخل للدالة ()print.

جملة الطباعة الموجودة في السطر (5,9) تشير إلى المتحول الموجود في السطر 3.

جملة الطباعة الموجودة في السطر 10 تشير إلى المتحول الموجود في السطر 8.

أم الآن وبعد أن تعلمنا الأساسيات فما عليك الآن صديقي القارئ إلا أن تركز في هذا الكود فهو مهم جدا!

```
1. // مثال تطبيقي
2. class Bowl {
3.   Bowl(int marker) { System.out.println("Bowl(" + marker + ")"); }
4.   void f(int marker) {  System.out.println("f(" + marker + ")"); }
5. }
6. class Table {
7.   static Bowl b1 = new Bowl(1);
8.   Table() { System.out.println("Table()"); b2.f(1);
9.   }
10.   void f2(int marker) { System.out.println("f2(" + marker + ")"); }
11.   static Bowl b2 = new Bowl(2);
12. }
13. class Cupboard {
14.   Bowl b3 = new Bowl(3);
15.   static Bowl b4 = new Bowl(4);
16.   Cupboard() { System.out.println("Cupboard()");  b4.f(2); }
17.   void f3(int marker) { System.out.println("f3(" + marker + ")");}
18.   static Bowl b5 = new Bowl(5);
19. }
20. public class Chp9_27 {
```

```
21.  public static void main(String[] args) {
22.    System.out.println("Creating new Cupboard() in main");
23.    new Cupboard();
24.    System.out.println( "Creating new Cupboard() in main");
25.    new Cupboard();
26.    t2.f2(1);
27.    t3.f3(1);
28.  }
29.  static Table t2 = new Table();
30.  static Cupboard t3 = new Cupboard();
31. }
```

شرح المثال:

1) يتم تنفيذ الجمل الاستاتيكية في الدالة الرئيسي كما في الأسطر (29,30).
2) بداخل كل Class كتل استاتيكي يقوم بتنفيذه أولاً كما في الأسطر (7,11,15,18) ثم الجمل الغير استاتيكية مثل جمل إنشاء الفئات كما في السطر 14 .
3) يقوم بتنفيذ دوال البناء داخل كل Class .
4) أن عاود استدعاء Class مرة أخرى فأنة لا ينفذ الكتل الاستاتيكية بل ينفذ
   1) الكتل العادية .
   2) دوال البناء .

وهذا ناتج تنفيذ البرنامج:

```
Bowl(1)
Bowl(2)
Table()
f(1)
Bowl(4)
Bowl(5)
Bowl(3)
Cupboard()
f(2)
Creating new Cupboard() in main
Bowl(3)
Cupboard()
f(2)
Creating new Cupboard() in main
Bowl(3)
Cupboard()
f(2)
f2(1)
f3(1)
```

- **الأصناف الداخلية**

الصنف الداخلي او الصنف المعشش هو الصنف الذي يتم تعريفة ضمن مجال التغطية التابع لصنف أخر . واليك المثال التالي عن الصنف الداخلي:

```
1. // الأصناف الداخلية
2. class Chp9__28{
3.                 Chp9__28(){System.out.println("Star Chp9__28");}
4.                 // local class
5.                 static class Chp9___28{
6.                                         Chp9___28(){System.out.println("Star
Chp9___28local");}
7.                                                 }
8.                  }
9.
10. class Chp9_28 {
11.   public static void main(String[] args) {
12.    new Chp9__28.Chp9___28();
13.                                                 }
14.    static{System.out.println("Star Chp9_28");}
15. }
```

```
Star Chp7_28
Star Chp7___28local
```

إذا تم إبعاد جملة static فماذا يكون خرج البرنامج ؟

من الممكن الصنف الداخلي الحصول على مرجع البيانات والمناهج الموجودة في الصنف الخارجي الذي يعشش فيه . لذلك لا تحتاج إلى تمرير مرجع للصنف الخارجي ضمن باني الصنف الداخلي . كالمثال التالي:

```
1. // حصول الكلاس الداخلي على قيم متحولات الكلاس الخارجي
2. class Chp9__29{static int i=20;
3.                 Chp9__29(){System.out.println("Star Chp9__29");}
4.
5.                 static class Chp9___29{static int i=40;
6.                                         Chp9___29(){int i=50;
7.                                         System.out.println(i+" "+this.i);
8.                                         System.out.println(Chp9___29.i+" "+Chp9__29.i);
```

```
9.                                                  }
10.                                             }
11.             }
12. public class Chp9_29 {
13.   public static void main(String[] args) {
14.    new Chp9__29.Chp9___29();
15.                                 }
16. }
```

```
50 40
40 20
```


استخدام الأصناف الداخلية تجعل البرنامج بسيطاً ومختصراً.


من الممكن التصريح عن الصنف الداخلي على أنه (public , private , protected) حيث يخضع لنفس قواعد الرؤية المطبقة على عناصر الصنف.


من ا لممكن التصريح على عناصر الصنف الداخلي على أنه static حيث من الممكن الوصول إلى الصنف الاستاتيكي بـاستخدام الصنف الخارجي. فستخدم الصيغة التالية:

```
Otherclass.innerclass innerobject =new Otherclass.innerclass();
```


محاولة الوصول إلى العناصر الستاتيكية التابعة للصنف الخارجي من الصنف الداخلي الاستاتيكي يعتبر خطاء شائعاً.

للقارئ النبيل فقط ما خرج هذا الكود ولماذا ؟

```
1. // مثال للقارئ النبيل
2. class Chp9__30{
3.              Chp9__30(){System.out.println("Star Chp9__30");}
4.
```

```
5.              static class Chp9___30{
6.                                  Chp9___30(){System.out.println("Star Chp9___30");}
7.                                  }
8.              Chp9___30 b= new Chp9___30();
9.             }
10. public class Chp9_30 {
11.   public static void main(String[] args) {
12.    new Chp9__30.Chp9___30();
13.                                  }
14.   static{System.out.println("Star Chp9_30");}
15. }
```

وهل هذا الكود يشبه الكود السابق؟

```
1. // مثال للقارئ النبيل
2. class Chp9__31{
3.              Chp9__31(){System.out.println("Star Chp9__31");}
4.              static class Chp9___31{
5.                                  Chp9___31(){System.out.println("Star Chp9___31");}
6.                                  }
7.              Chp9___31 b= new Chp9___31();
8.              {Chp9__31 b= new Chp9__31();}
9.             }
10. public class Chp9_31 {
11.   public static void main(String[] args) {
12.    new Chp9__31.Chp9___31();
13.   }
14.   static{System.out.println("Star Chp9_31");}
15. }
```

هل ناتج هذا المثال صحيح؟

```
1. // مثال للقارئ النبيل
2. class Chp9__32{
3.              Chp9__32(){System.out.println("Star Chp9__32");}
4.              static class Chp9___32{
5.                          Chp9___32(){System.out.println("Star Chp9___32");}
6.                          static void show(){System.out.println("show Chp9___32");}
7.                                  }
8.              static{Chp9__32 b= new Chp9__32();}
9.             }
10. public class Chp9_32 {
11.   public static void main(String[] args) {
12.    Chp9__32.Chp9___32.show();
13.   }
14. }
```

```
Star ammar
show ammarlocal
```

إن قلت ناتج الكود صحيح فقد أخطأت !!

والسبب :

صحيح إننا أخبرناك بأن المترجم ينفذ الجمل static أولاً، ولكن في حاله إطلاق الهدف.

ففي مثالنا السابق وصلنا وصول مباشر Class الداخلي ولم ننشط داله البناء Class الخارجي فهذا هو السبب.

تمارين الفصل:

1. املأ الفراغات التالية:
   - تقوم العملية ..... بعمليات الحجز الديناميكي للذاكرة من أجل إنشاء صنف ما من نمط معين.
   - تمثل المعطيات الأعضاء التي هي من صف التخزين ..... المعلومات المتوفرة على مستوى الصنف بأكمله.
   - لا يمكن الوصول إلى أعضاء الصنف ..... إلا من خلال مناهج الأعضاء المرتبطة بالصنف.
   - يعتبر التابع .... أحد التوابع الأعضاء التي تستخدم لإعطاء قيم ابتدائية إلى المعطيات الأعضاء المرتبطة بالصنف.
   - يمكن الوصول إلى الأعضاء .... المرتبطة بصنف من أي نقطة يمكن فيها إنشاء صنف تابع لهذا الصنف.
2. ماهو الهدف من عملية تحديد مجال الرؤية ؟
3. اكتشف الخطاء في هذا البرنامج واجعل ناتج تنفيذه نفس المثال Chp9_32 دون المساس بمحدد الوصول ؟

```
1. // EXP 1
2. class Chp9__33{static int i=20;
3.          Chp9__33(){System.out.println("Star Chp9__33");}
4.
5.          private static class Chp9___33{static int i=40;
6.                                  Chp9___33(){int i=50;
7.                                  System.out.println(i+" "+this.i);
8.                                  System.out.println(Chp9___33.i+" "+Chp9__33.i);
9.                                                  }
10.                                                                 }
11.                 }
12. public class Chp9_33 {
13.   public static void main(String[] args) {
14.    new Chp9__33.Chp9___33();
15.                                                 }
16. }
```

4. اكتشف الخطاء في هذا البرنامج :

```
1. // EXP 2
2. class Chp9__34{int i=20;
3.          Chp9__34(){new Chp9___34();}
4.          static class Chp9___34{static int i=40;
5.                                  Chp9___34(){ int i=50;
6.                                  System.out.println(i+" "+this.i);
7.                                  System.out.println(Chp9___34.i+" "+Chp9__34.i);
```

```
8.                                                  }
9.                                          }
10.                      }
11. public class Chp9_34 {
12.   public static void main(String[] args) {
13.    new Chp9__34();
14.                                                 }
15. }
```

5. ما خرج هذا البرنامج :

```
1. // EXP 3
2. class Chp9__35{
3.            static int a(){return j;}
4.            static int i=a();
5.            static int j=1;
6.         }
7. class Chp9_35{
8.   public static void main(String args[]) {
9.               System.out.println(Chp9__35.i);
10.                                                    }
11. }
```

6. أنشئ فئة تمثل مربع وسمه Rectengle ، بحيث تحتوي على المتغيرات التالية: length، width يأخذ كل واحد منهم قيمة ابتدائية تساوي 1، ويحتوي على منهجين ، يقوم المنهج الأول بحساب مساحة المربع، بينما يقوم المنهج الثاني بحساب محيط المربع. ثم أكتب برنامج لتطبيق هذه الفئه.

Chp10

الوراثة وتعددية الأشكال

في نهاية هذا الفصل سوف تتعلم :

- كيفية صنع صنف أبن (subcalss) من صنف أب(supclass) باستخدام الوراثة.
- تجاوز (override) المناهج في صنف الابن.
- استدعاء بانيات الصنف الأب ومناهجه باستخدام الكلمة ألمفتاحيه super.
- التعرف على تعددية الأشكال والربط الديناميكي .

10.1 مقدمة INTRODUCTION

سوف نناقش في هذا الفصل أمكانية من الإمكانيات الهامة التي تتيحها البرمجة غرضيه التوجه : وهي الوراثة . فتعتبر أحد أشكال إعادة استخدام النصوص البرمجية حيث يتم وفقاً لها إنشاء صفوف جديدة انطلاقاً من صفوف موجودة مسبقاً . وتساعد أيضاً على تقليل الإشكاليات الناتجة عن وضع البرمجيات قيد الاستخدام الفعلي.

10.2 الوراثة Inheritance

يوجد في علوم الأحياء علم أسمه التصنيف العلمي (Scientific Classification) وفيه يقوم العلماء بتصنيف الكائنات الحية وترتيبها طبقاً للخواص المشتركة. أول نظام للتنصيف قام به أرسطو الذي صنفها على أساس بيئتها ، وقد ترجم ابن رشد تصنيف أرسطو في كتاب مفقود وبقيت الترجمة اللاتينية لكتاب ابن رشد. و التصنيف الحديث تعود جذوره إلى نظام كارلوس لينيوس، الذي صنف الأنواع طبقا للخواص الفيزيائية المشتركة.
يبدأ التصنيف الرئيسي بتسلسل مملكه ، شعبة ، طائفة ، رتبة ، عائلة ، جنس ، نوع . بعد ذلك أضيف فوق رتبة ، تحت رتبة ، فوق طائفة ، تحت طائفة ، قبيلة . وتصنيفات أخرى.
فمثلا الإنسان ينتمي لمملكة الحيوان (Animalia) ومن صفات هذه المملكة أن الكائنات فيها متعددة الخلايا multicultural أي إنها كائنات تتكون من أكثر من خلية و إذا تدرجنا في شجرة الحياة للإنسان فسنجده ينتمي إلى طائفة (class) أسمها الثديات (Mammilla) والتي ينتمي إليها كل الحيوانات و من صفات هذه الطائفة أن الكائنات التي تنتمي إليها تلد صغاراً و ترضعهم أمهاتهم اللبن عن طريق الثدي. وينتمي الإنسان أيضاً إلى تحت طائفة (subclass) أسمها placentalia و التي من صفاتها أن الطفل في مرحلة الحمل يـ تغذى عن طريق المشيمة.
فـ الإنسان يرث صفات وسلوك ال تحت طائفة (subclass) التي تسمى placentalia وبالتالي يرث صفات وسلوك الطائفة (class) التي أسمها Mammaila وبالتالي يرث صفات وسلوك المملكة Animalia. وكما نري فإن التصنيفات العليا أي التي تتجه ناحية جذر الشجرة مثـ ل المملكة تحتوي على صفات عامة و كلما تدرجنا ناحية فرع الشجرة كلما كانت الصفات و السلوك أكثر تخصصاً مثل صفات وسلوك الإنسان.
مفهوم ال Inheritance في OOP ينطبق عليه نفس الكلام السابق. فال class من الممكن أن ترث صفات و سلوك class أخرى. وال class التي ترث نسميها subclass و الclass التي يورث منها تسمى superclass وكما نلاحظ فهذه المسميات جاءت من علم التصنيف.

وفي Java من الممكن أن ترث ال class مباشرة من class واحدة فقط ولكنها تستطيع أن ترث من أكثر من class بطريقة غير مباشرة.
وكما نرى فإن ال subclass ليست محدودة بصفات ال superclass التي ترث منها بل تزيد عليها صفات وسلوك . في Java فإن ال superclass العليا في شجرة الوراثة أي الجذر هي ال class التي تسمى Object.
ال class Object ليس لها علاقة بمعنى object الذي تكلمنا عنه بل هي class وإسمها Object. وطالما أن جذر الشجرة في Java هي Object class فإن كل ال classes ترث سلوك وصفات هذه ال class مثل السلوك (toString method) و التي ترجع String (تتابع من الحروف) والتي تصف ال class.
لو رجعنا لعلم التصنيف و أردنا أن نعرف class تكون هي الجذر لكل الأشياء سواء حية أو غير حية فما هي ال methods التي ستكون في هذه ال class؟
أعتقد إنها تُسَبِّحُ.
يقول تعالى (تسَبِّحُ لَهُ السَّمَاوَاتُ السَّبْعُ وَالأَرْضُ وَمَن فِيهِنَّ وَإِن مِّن شَيْءٍ إِلاَّ يُسَبِّحُ بِحَمْدَهِ وَلَـكِن لاَّ تَفْقَهُونَ تَسْبِيحَهُمْ إِنَّهُ كَانَ حَلِيماً غَفُوراً)
إن مميزات الوراثة هي:
ال subclasses نستخدمها لنحصل على سلوك وصفات أكثر تخصصا من ال superclass التي سلوكها عاما
في البرامج الكبيرة يشترك أكثر من مبرمج في كتابة البرنامج ومن الممكن أن يكتب أحد المبرمجين abstract classes أي classes تجريدية بمعنى إنها تشتمل على سلوك عام وعلى المبرمجين الآخرين أن يقوموا في ال subclasses بكتابة الكود الخاص بهذه السلوك (methods) فمثلا في برنامج صناع الحياة قام المبرمج عمرو خالد بعمل ال abstract class التي أسمها نهضة و ال methods التي تنتمي لها هذه ال class هي (مثلا) : تنهض بالأب و تتفوق و تبدع و تصمم و تنتج
وعلى المبرمجين المشتركين في برنامج النهضة أن يكتبوا subclasses ترث من ال class نهضة و يقوموا بكتابة ال الكود الخاص بال methods التي تنتمي لل class نهضة. فمنهم من يكتب ال class زراعة الأسطح ومنهم من يكتب class النهضة الصحية وهكذا...
فالنعود لمثال الشخصية الـ تي ذكرت في بداية شرح oop صفحة 32 لنقول أن الشركة المطورة للعبة طورت سابقا لعبة بها شخصيات أيضا لكن كتاب كائن الشخصية لم يطوروها بالشكل المطلوب فقط اكتفوا بجعلها تتصرف التصرفات الطبيعية المشتركة بين الشخصيات الطبيعة الحقيقية مثلا التنفس الحركة الطبيعة وحدود القطع في المشهد والتصادم ثم " اتكنسل المشروع " وتم إغلاقه وإعلان فشله وبعد 5 سنوات قام مشروع جديد وهو مشروع مبرمجنا الصنديد وتم تغيير أصناف المبرمجين في هذه الفترة لكن الوثائق القديمة والأدوات والفئات مازالت موجودة فكل ما سيفعله مدير المشروع هو أن يهرع بجلب الفئة التي عرفت سابق ويضيفها للمشروع بحال كائن أب ويشتق فريق تطوير شخصية المحارب منها ليكتسب كل

الصفات الأساسية في لمح البصر ويصبو مجهودهم على الإضافات فقط كالقتال وغيره !
يمكن لفرق الشخصيات إنتاج مئات الشخصيات بسرعة خرافية لأنهم سيركزون على
ا لإضافات فقط مثلا شخصية طباخ وشخصية لاعب كرة وشخصية وحش " إذا كان يتنفس
أيضا !! " كل هذه الشخصيات ستتطور بسرعة كبيرة بالطبع مبرمجنا سيكون سعيدا للغاية
وهو يحتسي قدحا من القهوة وهو يراقب هذه الفرق وهي تعمل فهو بلا عمل لأنه أنجز كل
عمله !فإعادة الاستخدام ميز ة رائعة جدا لكن تتطلب بعض الحذر واتساع الأفق وبعد النظر
ومحاولة جعل التطور يكون في أكبر عدد من المستويات لكل مستوى فئة ترث من التي قبلها
ليسهل في أي لحظة الوصول للفئة الأقرب للحاجة .

10.3 الكلمة ألمفتاحيه (extends)

وهي أساس الوراثة فعند وضع هذه الكلمة بجانب اسم Class هذا يعني أن هذا Class
يرث من class أخر

```
1. // A simple example of inheritance.
2. // Create Chp10_1 superclass.
3. class Chp10__1 {static int i=10;
4.                    static private int j=20;
5.                    static protected int c=100;
6.                   static int r(){return j;}
7.                 }
8. // Create Chp10_1 subclass by extending class Chp10__1.
9. class Chp10_1 extends Chp10__1 {
10.   public static void main(String[] args) {
11.     System.out.println(i);
12.     System.out.println(c);
13.     System.out.println(r());
14.                                          }
15. }
```

```
10
100
20
```

شرح المثال:

السطر 9 عملية توريث Chp10_1 Class من Chp10__1 Class فأصبح Class الوارث Chp10_1 يستطيع أن يصل لجميع الدوال والمتغيرات التي من نوع public, protected في Chp10__1 Class.

محاولة الوصول إلى المتحولات الخاصة Private مباشرة في الصنف الابن ينتج عن ذلك خطاء كهذا المثال:

```
1. // Error : Variable j in class Chp10__2 not accessible from class Chp10_2.
2. // Create Chp10__2 superclass.
3. class Chp10__2 {
4.                         static private int j=20;
5.                         static protected int c=100;
6.                 static int r(){return j;}
7.               }
8. // Create Chp10_2 subclass by extending class Chp10__2.
9. class Chp10_2 extends Chp10__2 {
10.   public static void main(String[] args) {
11.     System.out.println(Chp10__2.j);//Error
12.     System.out.println(c);
13.     System.out.println(r());
14.                                         }
15. }
```

محاولة الوصول إلى المتحولات العادية الموجودة في الصنف الابن ينتج عن ذلك خطاء في زمن التنفيذ كهذا المثال:

```
1. // Error : Can't make a static reference to nonstatic variable i in class Chp10__3..
2. // Create Chp10__3 superclass.
3. class Chp10__3 {int i=10;
4.                         static private int j=20;
5.                         static protected int c=100;
6.                 static int r(){return j;}
7.                }
8. // Create Chp10_3 subclass by extending class Chp10__3.
9. class Chp10_3 extends Chp10__3 {
10.   public static void main(String[] args) {
11.     System.out.println(i);//Error
12.     System.out.println(c);
13.     System.out.println(r());
14.                                         }
15. }
```

10.4 الكلمة ألمفتاحيه (Super)

ذكرنا سابقاً أن الكلمة this تشير إلى Class نفسه فالكلمة super تشير إلى نفس صنف الأب واقصد بالأب أي Class المورث أي موقعة في مثالنا السابق Chp10__1 ولها استخدام ان

10.4.1 تستدعي باني الصنف الأب .

```
1. // Chp10__4 now uses super to initialize its Chp10_4 attributes.
2. class Chp10__4{
3.                     Chp10__4(){System.out.println("Star Chp10__4");}
4.                    }
5.
6. class Chp10_4 extends Chp10__4{
7.   public static void main(String[] args) {
8.    new Chp10_4(10);
9.    System.out.println("End");
10.                                         }
11.    // constructor for Chp10_4
12.    Chp10_4(int i){
13.                                     super();// call superclass constructor
14.                                     System.out.println(i*i);
15.                                    }
16. }
```

شرح المثال:

السطر 13 تم استدعاء باني الصنف Chp10_4 أولاً ثم ينفذ بقية أسطر البرنامج.

وعند عملية الاستدعاء يجب مرعاه الآتي:

عند استدعاء دالة بناء الصنف الأب فيجب وضع الكلمة super في بداية دالة البناء كما في السطر 13.

10.4.2 تستدعي دوال ومتغيرات الأب .

```
1. // Using super And This.
2. class Chp10___5{protected int i=10;}
3.
4. class Chp10__5 extends Chp10___5{protected int i=20;
5.
6.                              Chp10__5(){}//default constructor
7.
8.                              void test(int a){
9.                                                  System.out.println(super.i);
10.                                                 System.out.println(this.i);
11.                                                 System.out.println(i);
12.                                                }
13.                                                  }
14. class Chp10_5 {static int i=30;
15.   public static void main(String[] args) {
16.      Chp10__5 b=new Chp10__5();// default
17.      b.test(40);
18.                                                 }
19. }
```

```
10
20
20
```

شرح المثال:

السطر 9 يتم استدعاء قيمة المتحول من الصنف الابن Chp10___5 بواسطة التعليمة super.

السطر 10 يتم استدعاء قيمة المتحول الموجودة داخل الصنف Chp10__5 بواسطة التعليمة this.

السطر 16 يتم إنشاء فئة مشتقة من الصنف Chp10__5 و استخدام المنهج test الموجود بداخل الصنف.

وهذا مثال أخر :

```
1. // Using super And This.
2.
3. class Chp10___6{int i=10;}
4.
5. class Chp10__6 extends Chp10___6{
6.                                    int i=20;
7.                                    Chp10__6(){System.out.println(i+" "+super.i);}
8.                                  }
9. class Chp10_6 {
10.  static int i=30;
11.  public static void main(String[] args) {
12.   Chp10__6 b=new Chp10__6();
13.   int i=40;
14.   System.out.println(b.i+" "+i);
15.   System.out.println(Chp10_6.i+" "+((Chp10___6)b).i);
16.                                    }
17. }
```

شرح المثال:

لا يوجد أي شيء غامض إلا السطر 15 { ((Chp10___6)b).i } والغرض منها / هو الوصول إلى متغير الصنف Chp10___6 عبر الصنف Chp10__6 أي الوصول إلى أب الصنف Chp10__6 وتعتبر مفيدة جداً.

وهذا ناتج تنفيذ المثال:

```
20 10
20 40
30 10
```

10.5 أسبقية استدعاء دوال البناء للصنف الموروث

عند تنشيط صنف وهو يرث من صنف أخر فأنة أولا يتم تنفيذ دالة بناء صنف الأب ثم دالة بناء صنف المنشط وهذا المثال يبين ذلك

```
1. // Using super to overcome name hiding.
2. class Chp10____7{
3.                 Chp10____7(){System.out.println("star Chp10____7");}
```

```
4.                         }
5.
6. class Chp10___7 extends Chp10____7{
7.                                 Chp10___7(){System.out.println("star Chp10___7");}
8.                                                 }
9.
10. class Chp10__7 extends Chp10___7{
11.                                                 Chp10__7(){System.out.println("star Chp10__7");}
12.                                                         }
13. class Chp10_7 {
14.   public static void main(String[] args) {
15.    Chp10__7 b=new Chp10__7();
16.                                                 }
17. }
```

```
star Chp8____7
star Chp8___7
star Chp8__7
```

أم إذا تواجدت دوال تنشيط أي إنشاء هدف فيختلف الأمر هكذا المثال:

```
1. // اسبقية التنفيذ في حالة اطلاق صنف
2. class Chp10___8 {
3.             Chp10___8() {System.out.println("Chp10___8");}
4.                         }
5.
6. class Chp10__8{
7.                         Chp10__8() {System.out.println("Chp10__8");}
8.                         }
9.
10. class Chp10_8 extends Chp10___8 {
11.
12.   Chp10__8 a=new Chp10__8();
13.   public static void main(String[] args) {
14.    new Chp10_8();
15.                                                 }
16.   Chp10_8() {System.out.println("Chp10_8");}
17. }
```

شرح المثال:

خطوات تنفيذ المثال :

1- يتم إطلاق الصنف الرئيسي كما في السطر 14.
2- يتم الآنتقال لصنف الأب Chp10___8 وينفذ دالة البناء الخاصة به كما في السطر 3.

3- يتم تنفيذ جمل المتغيرات لتنشيط الصنف كما في السطر 12 ، وتنفيذ دالة البناء الخاصة به كما في السطر 7.
4- ينفذ داله البناء للصنف الرئيسي Chp10_8.
وهذا ناتج التنفيذ:

```
Chp8____8
Chp8___8
Chp8_8
```

نلاحظ في المثال Chp10_8 لم نذكر الكلمة ألمفتاحيه super وكما قلنا إنها تعمل على استدعاء دوال البناء فعندما تذكر هذه الكلمة فإنها تستدعي دالة البناء الخاصة بنفس نوع super كهذا المثال:

```
1. // استدعاء جملة باني الصنف بنوع المعطيات
2. class Chp10____9{
3.                 Chp10____9(){System.out.println("star Chp10____9");}
4.                 Chp10____9(int i){System.out.println("Chp10____9 ="+i*i);}
5.                       }
6. class Chp10___9 extends Chp10____9{
7.                              Chp10___9(){
8.                                                  super(10);
9.                                                  System.out.println("star Chp10___9");
10.                                    }
11.                                                }
12. class Chp10__9 extends Chp10___9{
13.                                          Chp10__9(){System.out.println("star Chp10__9");}
14.                                                   }
15. class Chp10_9 {
16.   public static void main(String[] args) {
17.    Chp10__9 b=new Chp10__9();
18.    System.out.println("star Chp10_9");
19.                                                 }
20. }
```

```
Chp8____9 =100
star Chp8___9
star Chp8__9
star Chp8_9
```

في حال تنشيط Class وهذا Class يرث من Class أخر وفي داخل Class دوال تنشيط أي كتل بناء استاتيكية فان أسبقية التنفيذ لكتل البناء الاستاتيكية ثم دوال البناء وهذا المثال يبين ذلك:

```
1. // اسبقية تنفيذ الجمل الاستاتيكية
2. class Chp10___10 {
3.                     Chp10___10(){System.out.println("star Chp10___10");}
4.                     Chp10___10(int i){System.out.println("Chp10___10 ="+i*i);}
5.                   }
6. class Chp10__10 extends Chp10___10{
7.                                       Chp10__10(){System.out.println("star Chp10__10");}
8.                                       static{Chp10___10 m= new Chp10___10(10);
9.                                             }
10.                                                                   }
11. class Chp10_10 {
12.   public static void main(String[] args) {
13.    Chp10__10 b=new Chp10__10();
14.                                        }
15. }
```

ناتج تنفيذ المثال:

```
Chp8___10 =100
star Chp8___10
star Chp8__10
```

10.6 وراثة الصنف الداخلي

تستطيع وراثة صنف داخل الصنف الرئيسي مما يسهل عليك الكثير من الصعوبات ، كالمثال التالي:

```
1. // وراثة الصنف الداخلي
2. class Chp10___11 {
3.             Chp10___11(){System.out.println("Star Chp10___11");}
4.             // local class
5.             class Chp10____11{
6.                     Chp10____11(){System.out.println("Star Chp10____11");}
7.                                   }
8.                       }
9. class Chp10__11 extends Chp10___11.Chp10____11{
10.                                   Chp10__11(){
11.                                                   new Chp10___11().super();
```

```
12.                                                 System.out.println("star Chp10__11");
13.                                             }
14.                                     }
15. class Chp10_11{
16.       public static void main(String args[]) {
17.                 Chp10__11 d=new Chp10__11();
18.                 System.out.println("End main");
19.                                             }
20. }
```

```
Star Chp8___11
Star Chp8____11
star Chp8__11
End main
```

شرح المثال:

يتم الوصول للصنف الداخلي بذكر اسم الصنف الخارجي ثم نقطة ثم اسم المنهج أو الكلمة super كما في السطر 11.

وعلى الرغم من إننا لم نحفز دالة البناء للصنف Chp10___11 ولكن نجد من إنها تم تنفيذها فالسؤال هنا إذا تم إبعاد دالة البناء للصنف Chp10___11 فهل سينفذ البرنامج وما خرج البرنامج ولماذا ؟

في هذا المثال أولوية بالتنفيذ فتتبعه بتمعن:

```
1. // اسبقية تنفيذ الاصناف الخارجية ولاداخلية
2. class Chp10___12 {
3.                 Chp10___12(){System.out.println("Star Chp10___12");}
4.                 Chp10___12(int g){System.out.println("Chp10___12 ="+g);}
5.           //local class
6.           class Chp10____12{
7.                  Chp10____12(){System.out.println("Star Chp10____12");}
8.                  Chp10___12 s=new Chp10___12(10);
9.                 }
10.          }
11. class Chp10__12 extends Chp10___12{
12.                         Chp10__12(){System.out.println("star Chp10__12");}
13.                         Chp10____12 d=new Chp10____12();
14.                         static {System.out.println("star Chp10__12 static");}
15.                                 }
16. class Chp10_12{
```

```
17.     public static void main(String args[]) {
18.             Chp10__12 d=new Chp10__12();
19.             System.out.println("End main");
20.                                             }
21. }
```

شرح المثال:

أولاً يتم تنفيذ السطر 14.

ثانياً يتم تنفيذ السطر 3.

ثالثاً يتم تنفيذ السطر 13.

رابعاً يتم تنفيذ السطر 8.

خامساً يتم تنفيذ السطر 7.

سادساً يتم تنفيذ السطر 12.

سابعاً يتم تنفيذ السطر 19.

وهذا خرج البرنامج:

```
star Chp8__12 static
Star Chp8___12
Chp8___12 =10
Star Chp8____12
star Chp8__12
End main
```

10.7 تجاهل دوال صنف الأب وعمل عملية Method Overriding

والقصد بة وجود دالة موجودة بصنف الأب وموجودة بصنف الابن بنفس الاسم ففي هذه الحالة يحصل عملية تعديل فعند استدعاء هذه الدالة فيأخذ المترجم دالة الابن بدل دالة الأب

وهذا المثال يبين ذلك:

```
1. // Methods with differing type signatures are overloaded – not
2. // overridden.
3. class Chp10___13 {
4.                         Chp10___13(){}
5.                         void show(){System.out.println("halo Chp10___13");}
6.                       }
7.
8. class Chp10__13 extends Chp10___13{
9.                         Chp10__13(){}
10.                        // display Chp10__13 this overrides show() in Chp10___13.
11.                       void show(){System.out.println("halo Chp10__13");}
12.                                          }
13. class Chp10_13{
14.     public static void main(String args[]) {
15.               Chp10__13 d=new Chp10__13();
16.               d.show();// this calls show() in Chp10__13
17.               System.out.println("End main");
18.                                                  }
19. }
```

```
halo Chp8__13
End main
```

شرح المثال:

نجد في المثال انه تجاهل الدالة الموجودة في Class الأب Chp10___13 ، السؤال هنا كيف يمكننا استخدام دالة Class الأب أي بمعنى كيف يمكننا أن نصل إلى ألداله الموجودة في Class الأب Chp10___13 ؟

الجواب كما يلي:

ر اجع المثال Chp10_6 فمن ذلك المثال يتضح حل السؤال .

هذا المثال يوضح أكثر الاختلاف بين عمليه التحوير أي التعطيل في الدوال static والعادية

```
1. // Method overriding and Desebald
2. class Chp10___14 {
3.                         Chp10___14(){}
4.                         static String show1(){return "Ammar";}
5.                         String show2(){return "Mohammed";}
6.                       }
7. class Chp10__14 extends Chp10___14{
```

```
8.                                          Chp10__14(){}
9.                                           static String show1(){return "Sand";}
10.                                        String show2(){return "Alopaee";}
11.                                    }
12. class Chp10_14{
13.       public static void main(String args[]) {
14.                 Chp10___14 d=new Chp10__14();
15.                 System.out.println(d.show1()+"  "+d.show2());
16.                                                  }
17. }
```

```
Ammar  Alopaee
```

شرح المثال:

في السطر 14 لا يتم استبدال الدوال التي تحمل نفس الاسم بل يذهب إلى دالة الأب وينفذها ويترك حق الابن أي لا توجد Overload.

10.8 الأصناف النهائية

عندما تريد من أن يصبح الصنف غير قابلة للتوريث ، يمكنك استخدام المعدل final للدلالة على أن هذا الصنف نهائي ولا يمكن أن يصبح صنف أب.

10.9 الاستبدال أو التعطيل Override

وهي عملية تعديل صفة من الصفات الموروثة من الاب واخذ جميع صفات class الأب وإضافة صفات نحتاجها فوق صفات class الأب وهذا المثال يبين ذلك:

```
1. // برنامج الأستبدال والتعطيل
2. class Chp10___15 {   int i;
3.                                Chp10___15(int i){this.i=i;}
4.                                int get_i(){return i;}
5.                           }
6. class Chp10__15 extends Chp10___15{
7.                                int a,b;
8.                      Chp10__15(int a,int b,int c){
9.                                                                  super(c);
10.                                                                        this.a=a;
11.                                                                        this.b=b;
12.                                                                  }
13.                               void show(){
```

```
14.                                         System.out.println(super.get_i()+" "+a+" "+b);
15.                                 }
16.                         }
17. class Chp10_15{
18.         public static void main(String args[]) {
19.                 Chp10__15 d=new Chp10__15(10,20,30);
20.                 d.show();
21.                                         }
22. }
```

وهذا ناتج البرنامج:

```
30 10 20
```

شرح المثال:

في الأسطر (8 - 12) تمت عملية التعديل أي استخدام الصنف Chp10___15 وكأن الصنف Chp10__15 صنف واحد.

هل هذا المثال ناتج تنفيذه هكذا؟

```
star Chp8___16 =1
star Chp8___16 =2
star Chp8___16
star Chp8__16
star Chp8_16
get a=10
```

```
1. //Exp
2. class Chp10___16 {
3.                  Chp10___16(){System.out.println("star Chp10___16");}
4.                  Chp10___16(int x){System.out.println("star Chp10___16 ="+x);}
5.                  void get(int x){System.out.println("get a="+x);}
6.                      }
7. class Chp10__16 extends Chp10___16{static Chp10___16 b1,b2;
8.                          Chp10__16(){System.out.println("star Chp10__16");}
9.                          static{b1=new Chp10___16(1);b2=new Chp10___16(2);}
10.                                         }
11. class Chp10_16{
12.         public static void main(String args[]) {
13.                 Chp10__16 d=new Chp10__16();
14.                 System.out.println("star Chp10_16");
15.
16.                 Chp10__16.b1.get(10);
17.                                         }
```

```
18. }
```

شرح البرنامج:

إن قلت صح فقد أخطئت بسبب أننا في السطر 16 وصلنا وصول مباشر لذلك تم تجاهل دالة البناء الأب والابن أي لا توجد تنشيط لهما .

والسطر 9 تعتبر هذه الحالة بصنف داخل صنف.

محددات الوصل التي تم شرحها سابقاً يرجى الآذ تباه منها في الوراثة كهذا المثال يوجد بة خطاء قم بإيجاده ؟

```
1. //Exp
2. class Chp10___17 {
3.                         int i;
4.                         private int j;
5.                         void setij(int x, int y) {i = x;j = y;}
6.                       }
7. class Chp10__17 extends Chp10___17 {
8.                                                 int total;
9.                                                 void sum() {total = i + j; }
10.                                          }
11. class Chp10_17 {
12.       public static void main(String args[]) {
13.                 Chp10__17 subOb = new Chp10__17();
14.                 subOb.setij(10, 12);
15.                 subOb.sum();
16.                 System.out.println("Total is " + subOb.total);
17.                                            }
18. }
```

قبل أن ننتقل إلى الميزة الثالثة في الوراثة أهديكم هذا المثال الرائع :

```
1. // أسبقية التنفيذ في الوراثة
2. class Chp10___18 {
3.                         Chp10___18(){System.out.println("6");}
4.                           {System.out.println("5");}
5.                           static {System.out.println("3");}
6.                 }
7. class Chp10__18 extends Chp10___18 {
8.                                                 Chp10__18(){System.out.println("8");}
9.                                                 {System.out.println("7");}
```

```
10.                                         static {System.out.println("4");}
11.                                     }
12. class Chp10_18 {
13.       public static void main(String args[]) {
14.                 new Chp10_18();
15.                                         }
16.    Chp10_18(){new Chp10__18();}
17.
18.    {System.out.println("2");}
19.
20.    static {System.out.println("1");}
21. }
```

```
1
2
3
4
5
6
7
8
```

شرح المثال:

خطوات التنفيذ:

1- ينفذ الكتل الاستاتيكية في البرنامج الرئيسي كالسطر 20.
2- ينفذ الكتل الغير استاتيكية كالسطر 18.
3- ينفذ الكتل الاستاتيكية للصنف الأب كالسطر 5.
4- ينفذ الكتل الاستاتيكية للصنف الابن كالسطر 10.
5- ينفذ الكتل الغير استاتيكية للصنف الأب كالسطر 4 .
6- ثم دالة البناء للصنف الأب كالسطر 3.
7- ينفذ الكتل الغير استاتيكية للصنف الابن كالسطر 9.
8- ثم دالة البناء للصنف الابن كالسطر 8.

استخدام الجملة final مع Overriding ينتج خطاء في زمن التنفيذ
يصدر المترجم رسالة (The method void meth() declared in class Chp10_19_B cannot override the final method of the same signature declared in class Chp10_19_A. Final methods cannot be overridden.) وهذا مثال لذلك:

```
1. // Using final to Prevent Overriding
```

```
2. class Chp10_19_A {
3.                 final void meth() {
4.                                                 System.out.println("This is a final method.");
5.                                         }
6.                         }
7. class Chp10_19_B extends Chp10_19_A {
8.         void meth() { // ERROR! Can't override.
9.         System.out.println("Illegal!");
10.                                                 }
11. }
```

10.10 إرسال الطرق ديناميكياً Dynamic method dispatch

ويقصد بة إرسال الطرق ديناميكياً في حالة overriding أي التحكم في الوصول للدوال وهذا

مثال لذلك:

```
1. //Dynamic method dispatch
2. class Chp10____20 {
3.         void callme() {System.out.println("Inside Chp10____20's callme method");}
4.                         }
5. class Chp10___20 extends Chp10____20 {// override callme()
6.                         void callme() {System.out.println("Inside Chp10___20's callme method");}
7.                                                 }
8. class Chp10__20 extends Chp10____20 {// override callme()
9.                 void callme() {System.out.println("Inside Chp10__20's callme method");}
10.                                                 }
11. class Chp10_20 {
12.         public static void main(String args[]) {
13.                         Chp10____20 a = new Chp10____20(); // object of type Chp10____20
14.                         Chp10___20 b = new Chp10___20(); // object of type Chp10___20
15.                         Chp10__20 c = new Chp10__20(); // object of type Chp10__20
16.                         Chp10____20 r; // obtain a reference of type Chp10____20
17.                         r = a; // r refers to an Chp10____20 object
18.                         r.callme(); // calls Chp10____20's version of callme
19.                         r = b; // r refers to a Chp10___20 object
20.                         r.callme(); // calls Chp10___20's version of callme
21.                         r = c; // r refers to a Chp10__20 object
22.                         r.callme(); // calls Chp10__20's version of callme
23.                                                 }
24. }
```

شرح المثال:

```
Inside Chp8____19's callme method
Inside Chp8___19's callme method
Inside Chp8__19's callme method
```

10.11 تعددية التشكل Polymorphism

تعود هذه الكلمة إلى كلمة يونانية تعني تعدد الأشكال (many forms). فتسمح لنا بكتابة برنامجنا في صورة قابلة لتغيير واسع النطاق، سواء كان التغيير لفئات موجودة مسبقاً أو تغيير مستقبلي لإنتاج برامج جديدة. هذه الخاصية تسهل علينا توسيع قدرات نظامنا .

وكما ذكرنا في السابق أن الفئات الجديدة تسمى فئة فرعية -subclass ترث صفات الفئات التي أُنتجت وتكونت منها تسمى الفئة الأب -superclass كما يرث الطفل جينات أبويه. وهذه الفئة الجديدة والتي تعتبرsubclass ، من الممكن أن تكون superclass لفئات جديدة أخرى ينشئها المبرمج. وهكذا تمتد لدينا سلسلة من الوراثة بين الفئات extends ، يحكمها قانون" الوراثة المفردة "Single Inheritance حيث ينص هذا القانون على:

تنشأ أي فئة فرعية من فئة أم واحدة، في Java لا تدعم التوارث المتعدد multiple inheritance كالسي++ و لكنها تدعم مفهوم الواجهات Interfaces . على سبيل المثال كل دائرة هي كائن ولكن ليس كل كائن هو دائرة ، لذلك لا نستطيع دائماً تمرير الكائنات التي تنتمي إلى الصنف الابن كبارامتر من نمط الصنف الأب للصنف الابن الذي تم صنع هذا الكائن منه.

لنأخذ المثال التالي لتوضيح الفكرة:

```
1. // Polymorphism in Java
2. class Chp10_Shape {
3.                         void draw() {}
4.                         void erase() {}
5.                        }
6. class Chp10_Circle extends Chp10_Shape {
7.                                                 void draw() {
8.
System.out.println("Chp10_Circle.draw()");
9.                                                         }
10.                                                void erase() {
11.                                          System.out.println("Chp10_Circle.erase()");
12.                                                        }
13.                                               }
14. class Chp10_Square extends Chp10_Shape {
15.                                                void draw() {
16.                                          System.out.println("Chp10_Square.draw()");
17.                                                        }
18.                                                void erase() {
19.                                          System.out.println("Chp10_Square.erase()");
20.                                                        }
```

```
21.                                                     }
22. class Chp10_Triangle extends Chp10_Shape {
23.                                                      void draw() {
24.                                                 System.out.println("Chp10_Triangle.draw()");
25.                                                                 }
26.                                                      void erase() {
27.                                                 System.out.println("Chp10_Triangle.erase()");
28.                                                                 }
29.                                                     }
30. public class Chp10_21 {
31.   public static Chp10_Shape randChp10_Shape() {
32.       switch((int)(Math.random() * 3)) {
33.       default: // To quiet the compiler
34.       case 0: return new Chp10_Circle();
35.       case 1: return new Chp10_Square();
36.       case 2: return new Chp10_Triangle();
37.                                                     }
38.                                                      }
39.   public static void main(String[] args) {
40.     Chp10_Shape[] s = new Chp10_Shape[9];
41.     // Fill up the array with Chp10_21:
42.     for(int i = 0; i < s.length; i++)
43.       s[i] = randChp10_Shape();
44.     // Make polymorphic method calls:
45.     for(int i = 0; i < s.length; i++)
46.       s[i].draw();
47.                                                   }
48. }
```

```
Chp8_Circle.draw()
Chp8_Square.draw()
Chp8_Circle.draw()
Chp8_Circle.draw()
Chp8_Square.draw()
Chp8_Square.draw()
Chp8_Circle.draw()
Chp8_Circle.draw()
Chp8_Triangle.draw()
```

تذكر أن أي كائن ينتمي إلى فئة فرعية فهو ينتمي إلى الفئة الأب لهذه الفئة الفرعية ويحمل خصائصهما وسلوكهما.

10.12 الربط المتغير (الربط المتأخر) Dynamic Binding (Late Binding)

والمقصود هو أننا من الممكن استخدام الكائن الذي ينتمي إلى الصنف الابن من خلال أي شفرة برمجية معدة للعمل مع كائن ينتمي إلى الصنف الأب. عند استدعاء المنهج مع كائن ما فأنة يتم تحديد تنفيذ المنهج الذي سيتم استخدام ه من قبل آلة Java الافتراضية (JVM) بشكل ديناميكي وأثناء زمن التنفيذ (Runtime).

تمارين الفصل

1. ملأ الفراغات بالكلمات المناسبة ضمن التعابير التالية:
   - تعتبر ..... شكلاً من أشكال استخدام البرمجيات حيت تقوم وفقاً لها الأصناف الجديدة بأخذ المعطيات ومناهج أصناف موجودة مسبقاً وإضافة قدرات جديدة عليها.
   - يمكن الوصول إلى الأعضاء ..... التابعة لصنف أساسي من خلال التعريفات الواردة في الصنف الأساسي والأصناف المشتقة عنة فقط.
   - تتمتع الأعضاء المحمية التابعة لصنف أساسي بمستوى حماية يقع بين مستوى الحماية المصاحب للأعضاء من النوع public ومستوى الحماية المصاحب للأعضاء من النوع .....
   - عند اشتقاق صنف من صنف أساسي باستخدام الوراثة ، تصبح الأعضاء العامة المرتبطة بالصنف الأساسي أعضاءً... .... ضمن الصنف المشتق وتصبح الأعضاء المحمية المرتبطة بالصف الأساسي أعضاء ..... ضمن الصنف المشتق.
   - إذا تضمن أحد الأصناف منهجاً ظاهرياً أو أكثر، فيمكن أعتبارة .....
2. يفضل بعض المبرمجين عدم استخدام طريقة الوصول protected لإنها تضر بمفهوم تضمين المعطيات المرافق للصنف الأساسي. حدد الميزات الحسنة لاستخدام طريقة الوصول المحمية protected مقارنة من الطريقة الخاصة private ضمن الأصناف الأساسية ؟
3. كيف يم كن لتعددية الأشكال أن تساعد في تطوير برامج عامة بدلاً من أن تكون موجهة لمعالجة وضع محدد. بين فوائد البرمجة العامة ؟
4. كيف تسمح تعددية الأشكال بتمديد التطبيق وتوسيعه ؟
5. أوجد ناتج تنفيذ المثال التالي:

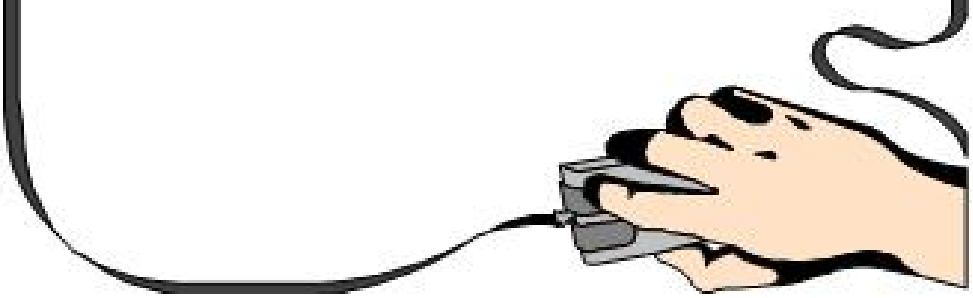
```
class B {
         B(){System.out.println("B 6");}
         {System.out.println("B 5");}
         static {System.out.println("B 3");}

        }
class A extends B {
                    A(){System.out.println("A 8");}
                    {System.out.println("A 7");}
                    static {System.out.println("A 4");}
                  }
class aldopaee {
public static void main(String args[]) {
new aldopaee();
}
    aldopaee(){new A();}
    {System.out.println("ammar 2");}
    static {System.out.println("ammar 1");}
    A b=new A();
    {B a=new B();}
}
```

Chp11

Java (Java Packages) حزم

<u>في نهاية هذا الفصل سوف تتعلم :</u>

- ما هي حزم.
- لماذا نحتاج الحزم .
- إنشاء الحزم .



11.1 مقدمة INTRODUCTION

تخيل أن عندك مكتبة كبيرة فيها عدد ضخم من الكتب . إذا أردت البحث على كتاب معين فأنك ستبدل مجهوداً ضخماً إذا لم يكن هناك تصفيف وترتيب معين للكتب. إذن لتسهيل عملية البحث على كتاب فأنك تضع مجموعة الكتب المتشابهة مع بعضها البعض.
بالمثل في لغة Java فعندنا عدد ضخم من الفئات Classes المتشابهة مع بعضها البعض في حزمة واحدة . وفي هذا الفصل ستتعرف معنى الحزم وكيفية تنظيم وترتيب برامجك في مجلدات مرتبة.

11.2 تعريف الحزم

هي مجموعة من الفئات المترابطة، متصلة في ما بينها بشجرة هرمية نعبر عنها بوضع نقطة X.Y أي الفرع Y من الأصل X نستطيع تحميل هذه الحزم ب import ثم اسم الحزمة ويمكنك تحميل كل الحزم المتفرعة بوضع * مكان الفروع مثلاً import java.applet*. و كل مجموعة من الفئات تنظم تحت حزمة معينة لأجل تحديد الهوية .

11.3 مكونات الحزمة

11.3.1 حزم فرعية تحت الحزمة الأب .

11.3.2 مجموعة من الفئات المتعلقة بالحزمة الأب .

بعض الأمثلة: الحزمة Java تحتوي على حزم فرعية منها :-

applet
awt
io
lang
net
util

و لو أخذنا الحزمة الفرعية Java.awt لحصلنا على حزمة فرعية من awt مثل image و يكون الامتداد لها Java.awt.image

11.4 سبب استخدام الحزم

مبرمجي Java يعتمدون على الحزم لتكوين فئات مترابطة داخل هذه الحزم و الأسباب هي:

- العثور على الفئات بشكل سريع و استخدام ها بالبرامج .
- تنحدر الفئات تحت الحزم لكي لا تتعارض أسماء الفئات مع بعضها البعض .
- للتحكم بالفئات بشكل كامل .

11.5 مسميات الحزم و الحزم الفرعية و الفئات

الحزمة تتكون من حزم فرعية و فئات متفرعة، لكن لا نستطيع تسمية الحزمة أو الحزم الفرعية أو إحدى الفئات باسم واحد. و مثال على ذلك: الحزمة java.awt لديها حزمة فرعية بالاسم .image لكن لا نستطيع تسمية إحدى الفئات بالاسم image ، لأن الاسم محجوز للحزمة الفرعية و العكس صحيح.

11.6 إنشاء الحزم

لنفترض الآن إننا سننشئ كلاس به دالة اسمها ;()show مهمتها طباعة رسالة على الشاشة ونريد استخدام هذه الدالة في كلاس أخر .

أولاً

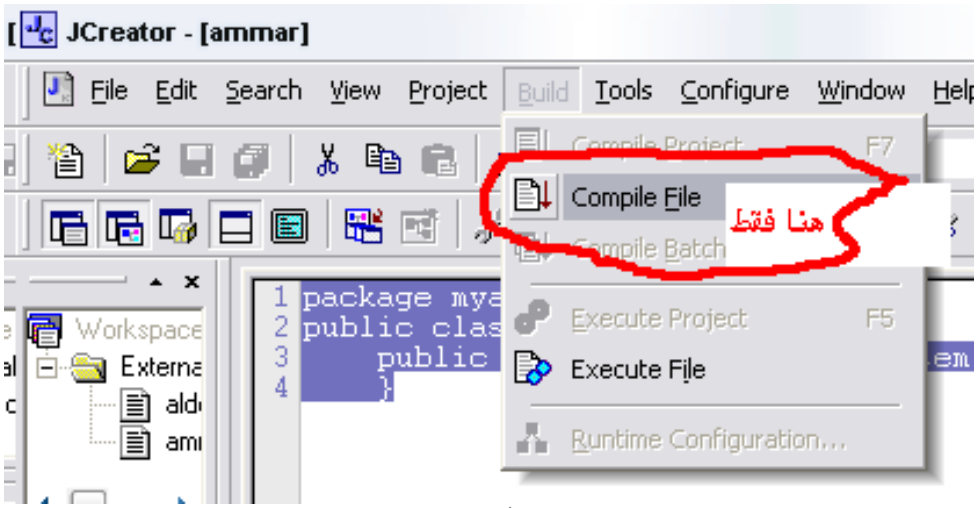
ننشئ Class وبداخلة دالة الطباعة وفي بداية Class نكتب (اسم الحزمة package)

```
1. package Chp11;
2. public class Chp11_1{
3.       public static void show(){System.out.println("my class");}
4.       }
```

إنشاء مناهج من نوع خاص private واستدعائها بواسطة الحزم ينتج خطاء في زمن التنفيذ . فيجب جعل نوع المناهج والمتحولات من نوع public كما في المثال Chp11-1 في السطر 3.

ثانياً

نعمل عملية ترجمة وهنا تتم الترجمة على حسب المترجم المستخدم فان كنت تستخدم JCreator فعمل كما في الشكل 1-11 فقط.



شكل 1-11

وان كنت على dos اكتب الأتي javac –d . Chp11_1.java

لنرى لو لدينا مجموعة من الفئات و التي نستطيع وضعها في حزمة معينة. نفترض إننا كتبنا فئات عن النقاط و الدائرة و المستطيل و المربع.

الآن نود أن نضع هذه الفئات مع بعضها البعض في حزمة لعدة أسباب:

- نستطيع نحن و المبرمجين الآخرين أن نجد هذه الفئات لإنها مترابطة .
- نستطيع نحن و المبرمجين الآخرين أن نعرف كيف نجد هذه الفئات لإنها دوال رسم مترابطة .
- أسماء الفئات السابقة لن تتعارض مع أسماء الفئات من الحزم الأخرى لإنها سوف تكون تحت حزمة جديدة من إنشائك، مثال على ذلك :

```
1. package Chp11;
2. public class Chp11_2 {
3.                         int x_coord;
4.                         int y_coord;
5.
```

```
6.    public Chp11_2() {
7.       x_coord = 0;
8.       y_coord = 0;
9.                          }
10.    public Chp11_2(int x ,int y) {
11.       x_coord = x;
12.       y_coord = y;
13.    }
14. }
```

```
1. package Chp11;
2. public class Chp11_3 extends Chp11_2
3. {
4.    double width;
5.    double height;
6.
7.    public Chp11_3(int x ,int y ,double w, double h)
8.    {
9.       super(x, y);
10.       width = w;
11.       height = h;
12.    }
13. }
```

```
1. package Chp11;
2. public class Chp11_4 extends Chp11_2
3. {
4.    double radius;
5.
6.    public Chp11_4(int x ,int y ,double r)
7.    {
8.       super(x, y);
9.       radius = r;
10.    }
11. }
```

```
1. package Chp11;
2. public class Chp11_5 extends Chp11_2
3. {
4.    double edge;
5.
6.    public Chp11_5(int x ,int y ,double e)
7.    {
8.       edge = e;
9.    }
10. }
```

نلاحظ هنا أننا أضفنا السطر package Chp11 في كل الفئات) كل فئة توجد في ملف مستقل). لكن لو فرضنا أننا نريد استخدام الفئة Ch11_3 موجودة بالحزمة java.awt مع الفئة الموجود بالحزمة Chp11 بنفس البرنامج الذي نريد كتابته، فماذا نفعل ؟

11.7 استدعاء فئتين بنفس الأسم

نستطيع ذلك باستخدام fully qualified name و هو كتابة المسار الكامل للفئة، مثال على ذلك:

```
java.awt.Chp11_3 rec1 = new java.awt.Chp11_3  (...); // استخدمنا المسار الكامل للفئة
Chp11.Chp11_3 rec2 = new Chp11. Chp11 _3(...); // هنا ايضاً و
```

11.8 استدعاء فئة معينة من الحزمة

تستطيع استدعاء الفئات من الحزم عن طريق ثلاث طرق:

استدعائها عن طريق كتابة المسار الكامل (كما المثال السابق) .
استدعائها فقط عن طريق الحزمةjava.awt. Chp11_3
استدعاء الحزمة كاملة بما فيها من فئات أخرى .;*.java.awt


النجمة (*) تدل على استدعاء الحزم الفرعية و الفئات الموجودة تحت هذه الحزمة.


عندما تنشئ حزمة وتريد الوراثة منها أي بالذي بداخلها فيجب عليك جعل اسم Class والدوال التي ستستخدمها ودوال البناء من نوع public , protected .

11.9 محددات الوصول للحزم

يبين الجدول 1-11 محددات الوصول داخل وخارج الحزمة.

جدول 1-11

	Private	No modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Class Member Access

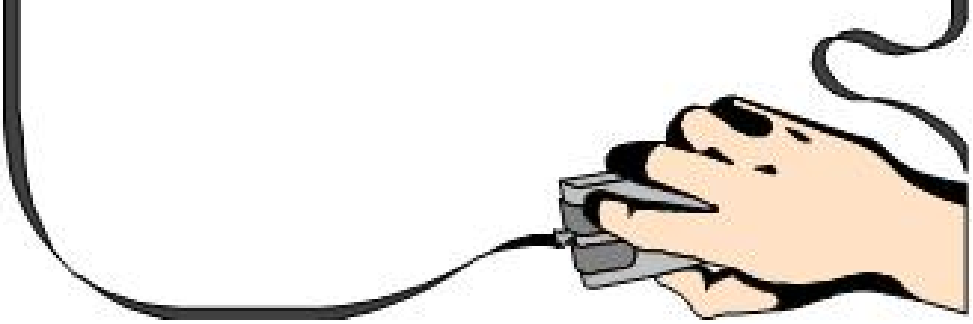
تمارين الفصل

Chp12

الأصناف المجردة والواجهات

في نهاية هذا الفصل سوف تتعلم :

- تصميم واستخدام الأصناف المجردة (abstract classes).
- التصريح عن الواجهات بهدف نمذجه العلاقات الوراثية الضعيفة.



12.1 مقدمة INTRODUCTION

تصبح الأصناف أكثر تحديداً وخصوصية كلما انتقلنا إلى صنف فرعي جديد ضمن السلسلة الوراثية، لكن إذا انتقلنا بالعكس (من الصنف الفرعي إلى الصنف الأب) عندها تصبح الأصناف أكثر عمومية وأقل خصوصية . يجب أن نتحقق أثناء مرحلة تصميم الصنف من احتواء الصنف الأب على الصفات العامة للأصناف الأبناء التابعة له، في بعض الأحيان تكون الأصناف الأب مجردة لا تستطيع الحصول على أيه كائن مخصصة، تدعى مثل هذه الأصناف بالأصناف المجردة (abstract class).

قد نحتاج في بعض الأحيان الوراثة من أكثر من صنف ما يسمى بالوراثة المتعددة (multiple inheritance)، لكن Java لا تدعم هذه الميزة حيث لا يستطيع الصنف الوراثة من عدة أصناف ، لكن تستطيع باستخدام الواجهات (interfaces) الحصول على نفس أثر الوراثة المتعددة.

وسنتعرف في هذا الفصل على الأصناف المجردة والواجهات.

12.2 الأصناف المجردة Abstract classes

والمعنى منة الفئات الخالية من الكود أي تحتوي على توقيع فقط أي يبقى بدون تنفيذ أي نستطيع أن نقول النوع المجرد من البيانات ليس له وجود في الواقع إنما هو في الحقيقة مفهوم أو فكرة للأصناف الأخرى التي تتشابه. وما يمز الصنف المجرد هو الكلمه (abstract) الموضوعة قبل كلمة class. فيكون التعريف الخاص بالأصناف المجردة كما يلي:

```
abstract class ClassName
{

  . . . . . // definitions of methods and variables

}
```

والتعريف الخاص بالمناهج كما يلي :

```
abstract class Card
{
 String recipient;              // name of who gets the card
 public abstract void greeting();  // abstract greeting() method
```

}

مثال :

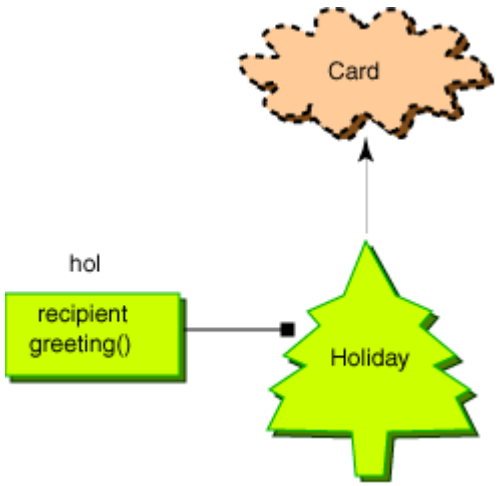
```
1. // Chp12_Card Simple demonstration of abstract.
2. abstract class Chp12_Card
3. {
4.   String recipient;
5.   public abstract void Chp12_greeting();
6. }
7.
8. class Chp12_Holiday extends Chp12_Card
9. {
10.   public Chp12_Holiday( String r )
11.   {
12.     recipient = r;
13.   }
14.
15.   public void Chp12_greeting()
16.   {
17.     System.out.println("Dear " + recipient + ",\n");
18.     System.out.println("Season's Chp12_greetings!\n\n");
19.   }
20. }
21.
22. public class Chp12_1
23. {
24.   public static void main ( String[] args )
25.   {
26.     Chp12_Holiday hol = new Chp12_Holiday("Santa");
27.     hol.Chp12_greeting();
28.   }
29. }
```

شرح المثال:

نلاحظ أن الصنف Chp12_Card وجدت في بداية الكلمة (abstract) ما يعني أنه مجرد ،

ويحتوي على توقيع المنهج Chp12_greeting الذي أيضاً مجرد ، فهي خالية من

التعليمات. والصنف Chp12_Holiday يرث من الصنف Chp12_Card حيث قام بتعديل المنهج Chp10_greeting وإدخال تعليمات إلى جسم المنهج . والشكل 12-1 يوضح ذلك :



شكل 12-1

ويكون ناتج تنفيذ هذا البرنامج:

```
Dear Santa,

Season's Chp10_greetings!
```

مثال أخر:

```
1. // Simple abstract.
2. abstract class Chp11_B {int i;
3.                         Chp11_B(int s){i=s;}
4.                         abstract void show();
5.                        }
6. class Chp11_A extends Chp11_B {
7.                                 Chp11_A(int s){super(s);}
8.                                 void show(){System.out.println(super.i*.10);}
9.                               }
10. class Chp11_C extends Chp11_B {
11.                                Chp11_C(int s){super(s);}
12.                                void show(){System.out.println(super.i*.15);}
```

```
13.                                     }
14. class Chp11_2 {
15.         public static void main(String args[]) {
16.                 Chp11_A a=new Chp11_A(100);
17.                 Chp11_C b=new Chp11_C(100);
18.                 a.show();
19.                 b.show();
20.                                             }
21. }
```

```
10.0
15.0
```

شرح المثال:

نلاحظ أن Chp12_B Class من نوع مجرد والدالة show من نوع مجرد فجميع الأصناف التي ترث Chp12_B class تعيد تعريف الدالة show كما تشاء ففي Chp12_A Class جعلناها تطبع نسبة الربح مثلاً 10. وفي Chp12_C Class جعلناها تطبع نسبة الربح 15. فهنا يتبين فائدة الأصناف المجردة أي تتيح للعملاء وضع الكود الخاص بهم .

12.2.1 قواعد حول التجريد

وضع مناهج مجردة بداخل صنف غير مجرد يسبب خطاء في زمن التنفيذ .(Abstract and native methods can't have a body: void show())

كهذا المثال :

```
class B {int i;
        B(int s){i=s;}
        abstract void show();
       }
```

يجب على الصنف الغير مجرد الذي يرث من صنف مجرد ، تنفيذ جميع المناهج حتى ولو لم يستخدمها كهذا الشفرة لن ينفذ إطلاقاً.

```
1. // class Chp12_3 must be declared abstract. It does not define void show2() from class
Chp12_3_B.
2. abstract class Chp12_3_B {int i;
3.                                  Chp12_3_B(int s){i=s;}
4.                                  abstract void show();
5.                                  abstract void show2();
6.                                 }
7. class Chp12_3 extends Chp12_3_B {
8.                                                  Chp12_3(int s){super(s);}
9.                                                  void show(){System.out.println(super.i*.10);}
10.                                          }
```

لا يمكن الحصول على فئات مشتقة من الصنف المجرد باستخدام الكلمة new كهذا المثال خاطئ .

```
1. // class Chp12_4_B is an abstract class. It can't be instantiated.
2. abstract class Chp12_4_B {int i;
3.                                  Chp12_4_B(int s){i=s;}
4.                                    abstract void show();
5.                                   }
6. class Chp12_4 {
7.          public static void main(String args[]) {
8.                     Chp12_4_B a=new Chp12_4_B(100);//Error
9.                                                          }
10. }
```

من الممكن التصريح على صنف مجرد يحتوي على مناهج غير مجردة كالمثال التالي:

```
1. // برنامج يستخدم صنف مجرد يحتوي على مناهج غير مجردة
2. class Chp12_5_B {
3.                          Chp12_5_B(){System.out.println("Chp12_5_B");}
4.                          void show(){}
5.                }
```

```
6. abstract class Chp12_5_A extends Chp12_5_B {int i;
7.                                                      Chp12_5_A(int s){super();}
8.                                                      abstract void show();
9.                                                  }
10. class Chp12_5_C extends Chp12_5_A {
11.                                                 Chp12_5_C(int s){super(s);}
12.                                                 void show(){System.out.println(super.i*.15);}
13.                                                  }
14. class Chp12_5 {
15.         public static void main(String args[]) {
16.                     Chp12_5_C a=new Chp12_5_C(100);
17.                                                 }
18. }
```

Chp10_5_B

من الممكن إن يكون الصنف الابن مجرداً حتى ولو كان الصنف الأب غير مجرد كالمثال التالي :

```
1. // اشتقاق صنف منجرد من اب غير مجرد
2. class Chp12_5_B {int i;
3.                     Chp12_5_B(int s){i=s;}
4.                     void show(){System.out.println(i*.15);}
5.                    }
6. abstract class Chp12_5_A extends Chp12_5_B {
7.                                                  Chp12_5_A(int s){super(s);}
8.                                                  void show(){System.out.println(super.i*.10);}
9.                                                      }
```

تسبب محاوله اشتقاق فئة من صنف مجرد ، حدوث أخطاء أثناء زمن الترجمة.

12.3 الواجهات Interfaces

هي عبارة عن بيئة مشابهة للصنف وتحتوي فقط على المناهج المجردة وبمعنى أخر يمكن القول أن الواجهات مشابهة للأصناف المجردة .

12.4ما تحتويه الواجهات

من الممكن أن يحتوي Interfaces على متغيرات ومناهج ولكن كل المتغيرات تكون فيها نهائية بمعنى أنه لا يمكنك تغير قيمتها الأبتدائية وأيضاً كل المناهج تكون abstract .

الهيكل العام للواجهات

```
Modifier interface InterfaceName
        {
          constant definitions
          method declarations (without implementations.)

          }
```

وعند وراثه الصنف من الواجهة نستخدم الكلمة implements:

```
public class BigClass extends Parent
   implements  interfaceName, interfaceName, interfaceName
{

  ordinary class definition body

}
```

مثال :

```
1. // implementation of Chp12__7_a.
2. interface Chp12__7_a{int i=100;
3.                      int get();
4.                     }
5. // Implement Chp12__7_a's interface
6. class Chp12__7 implements Chp12__7_a {
7.                                          Chp12__7(){}
8.                                        public int get(){return i;}
9.                                      }
10. class Chp12_7 {
11.       public static void main(String args[]) {
12.                                      Chp12__7 t=new Chp12__7();
13.                                      System.out.println(t.get());
14.                                        }
15. }
```

100

12.4.1 قواعد حول الواجهات

من الممكن أن تحتوي الواجهة على متغيرات ومناهج مثله مثل الصنف ولكن كل المتغيرات فيه تكون نهائية أي final لا يمكن تغير قيمتها ، وجميع المناهج من نوع abstract حتى وإن لم تذكر .

تتألف جميع المناهج في الواجهات من التوقيع فقط ولا تحتوي على التنفيذ .

لابد من إعادة تعريف جميع الدوال المعرفة بداخل الواجهة في الصنف وإلا اعتبر هذا الصنف من نوع مجرد فلا يمكن اشتقاق هدف منة 0

تسمح لغة Java بالوراثة المتعددة بواسطة الواجهات . حيث تنفذ جميع المناهج المجردة المعرفة في الواجهات كالمثال التالي :

```
1. // One interface can extend another.
2. interface Chp12_8_a{int i=100;
3.                     int get();
4.                     }
5. interface Chp12_8_b{int j=10000;
6.                      int Get();
7.                     }
8. class Chp12__8 implements Chp12_8_a,Chp12_8_b {
9.                                                  Chp12__8(){}
10.                                                 public int
get(){return i;}
11.                                                 public int
Get(){return j;}
12.                                             }
13. class Chp12_8 {
14.       public static void main(String args[]) {
15.                                           Chp12__8 t=new Chp12__8();
16.                                           System.out.println(t.get()+" "+t.Get());
17.                                            }
```

```
18. }
```

```
100 10000
```

تستطيع الواجهة وراثة واجهات أخرى فقط ولا تستطيع وراثة أصناف كهذا المثال:

```
1. // One interface can extend another.
2. interface Chp10_9_a{int i=100;
3.                     int get();
4.                     }
5. interface Chp10_9_b extends Chp10_9_a{int j=10000;
6.                                       int Get();
7.                                     }
8. class Chp10__9 implements Chp10_9_b {
9.                                         Chp10__9(){}
10.                                       public int get(){return i;}
11.                                       public int Get(){return j;}
12.                                     }
13. class Chp10_9 {
14.       public static void main(String args[]) {
15.                                       Chp10__9 t=new Chp10__9();
16.                                       System.out.println(t.get()+" "+t.Get());
17.                                         }
18. }
```

```
100 10000
```

وبعد هذه المقدمة وهذا التوصيف لعالم الـ OOP نلاحظ أن كلّ التركيز في هذا النوع من البرمجة يقع على الفئات Classes ، فالمبرمج يستخدم الفئات المبنية مسبقاً في اللغة مع الفئات التي يبنيها هو كي ينتج برنامجاً بالـ Java ، ربما يفسر هذا الاسم OOP .

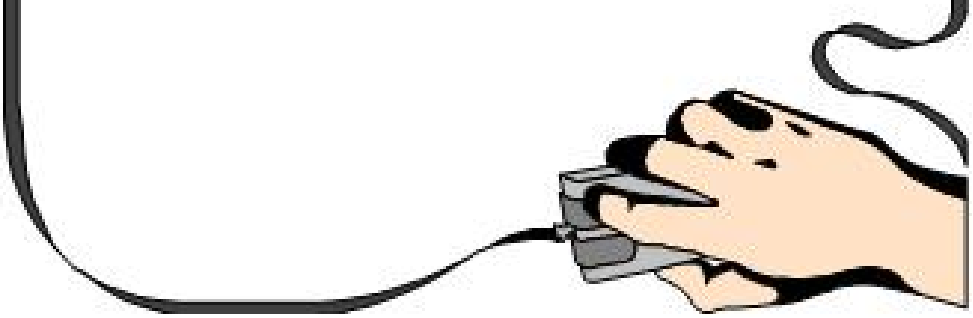
12.5 تمارين الفصل:

Chp13

الاستثناءات

في نهاية هذا الفصل سوف تتعلم :

- التعرف على مفهوم الاستثناءات.
- التصريح عن الاستثناءات في ترويسه المنهج.
- رمي الاستثناءات خارج المنهج.
- كتابة الكتلة try-catch من أجل معالجة الاستثناءات.
- شرح كيف يتم توليد الاستثناء.
- إنشاء استثناء خاص بك.



13.1 مقدمة

INTRODUCTION

نعرض في هذا الفصل واحدة من الإضافات الهامة بلغة Java، ألا وهي الاستثناءات (Exception Handling). يمكن لتوسعة لغة Java أن تزيد من عدد الأخطاء التي يمكن أن تحدث حيث يضيف كل صف جديداً عدداً من الأخطاء الممكنة المرتبطة به. تساعد التقنيات المعروض في هذا الفصل على كتابة برامج أكثر وضوحاً ومتانة ومقاومة للأعطال. لقد تم تطوير نظم الاستثمار الحديثة (مثل نظام windows NT من windows) باستخدام تقنيات مشابهة للتقنيات المعروضة في هذا الفصل ولقد أعطت نتائج أجابيه. سنبين في هذا الفصل أيضاً متى يجب معالجة الاستثناءات.

تجنب استخدام معالجة الاستثناءات من أجل غايات مختلفة عن غاية معالجة الخطاء لأن ذلك يقل من درجة وضوح البرنامج.

13.2 معنى الاستثناءات

الاستثناء هو مؤشر لحدوث خطاء أثناء عملية تنفيذ البرنامج مما يؤدي إلى تعطيل التسلسل الطبيعي لتعليمات البرنامج وقد تعلمنا في الفصل السابق أن الوراثة في Java تعطيها صفة الامتدادية وهذه الصفة يمكن أن تزيد من عدد ونوع الأخطاء التي يمكن أن تحدث حيث إن كل فصيلة جديدة تضاف إلى البرنامج يمكن أن تضيف مصدراً من مصادر الأخطاء الاستثناءات في البرنامج . إذاً نستطيع القول أن الاستثناءات هو حدوث خطاء ما وهذا الخطاء ليس خطاء في بناء الجملة syntax error ولكنه قد يكون له العديد من المصادر مثل القسمة على صفر ومعاملات غير متاحة للمناهج و الإشارة إلى عنصر في المصفوفة خارج نطاقها.

عند حدوث استثناء يحتاج البرنامج إلى معالجة هذا الاستثناء لكي يستمر تنفيذ البرنامج بصورة طبيعية وسابقاً قبل عام 1990 كانت معالجة هذا الاستثناء تتم باختبار قيم صحيحة تعود بدلائل مثل القيمة صفر تدل على نجاح والقيمة السالبة تدل على نوع من نوع الاستثناءات وهذه القيم أصبحت تعرف بشفرات الأخطاء Error codes وقد تم اكتشاف ان استخدام هذا النوع من معالجة الأخطاء يتسبب في ثلاث مشاكل:

1- غالباً تهمل شفرة الخطاء.
2- اختيار شفرة الأخطاء تعرض التدفق الطبيعي للبرنامج مما يصعب تتبه المستخدم للبرنامج.
3- اختبار شفرة الأخطاء يزيد من حجم البرنامج.

13.3 أساسيات معالجة الاستثناء في لغة Java

لقد أدت مشاكل استخدام شفرة الخطاء Error codes إلى تطوير أليه جديدة لمعالجة الاستثناءات قي لغه Java تعتمد على كائنات مما أدى إلى برامج سهل القراءة والتتبع وكذلك برامج أكثر مرونة.

وفي هذا النموذج عند حدوث استثناء أثناء تشغيل برنامج Java إما البرنامج program أو اله لغة Java الافتراضي Jvm تنشئ كائن لوصف الاستثناء ويشمل هذا الكائن قيم المتغيرات في لحظة حدوث الاستثناء.

إذا تم إنشاء الكائن من البرنامج فإن البرنامج يمرر ذلك الكائن إلى اله Java الافتراضية JVM وعند استقبال الكائن تبحث في البرنامج عن معالج الاستثناء الذي يمكن أن يعالج الاستثناء الموصوف بالكائن. إذا وجد المعالج يتم تمرير الكائن لمعالج الاستثناء الذي يقوم باستخدام محتويات الكائن لمعالجة الاستثناء . إذ لم يوجد معالج الاستثناء يتوقف البرنامج عن التنفيذ.

13.4 مصطلحات لابد منها :

يمكن أن ينظر لمعالجة الاستثناء كبنية تحكم غير محلية. عندما يرفع التابع استثناء فإن المستدعي له يجب أن يحدد فيما إذا كان من الممكن أن يمسك الاستثناء أم لا. إذا استطاع المنهج المستدعى أن يمسك الاستثناء فعليه أن يسيطر عليه ويستمر الـ شخص المستدعي في الطلب أما إذا لم يستطع المنهج المستدعى إمساك الاستثناء فعندها يتوقف المستدعي عن طلبه.

تستمر هذه المعالجة حتى يتم إمساك الاستثناء أو يتم الوصول إلى قمة أو أسفل مكدس الاستدعاء (ذلك حسب ما تنظر إليه) و ينتهي التطبيق لأن الاستثناء لم يُمسك.

لنأخذ هذا المثال البسيط .

```
String str = "x";
Int I = Integer.parseInt(str);
System.out.print(str);
```

في السطر الأول أنشأت متغير من نوع نصي ووضعت فيه القيمة x

في السطر الثاني أنشأت متغير من نوع رقم صحيح وقرأت القيمة الرقمية من المتغير النصي. طبعاً في الحالات العادية من المفترض أن تكون القيمة الموجودة في النص رقم ..مثلاً "123" ولكن في حالتنا كانت حرف وليس رقماً.. لذا عملية القراءة ستتسبب حدوث exception كما يلي :

```
Exception in thread "main" java.lang.NumberFormatException: x
        at java.lang.Integer.parseInt(Integer.java:405)
        at java.lang.Integer.parseInt(Integer.java:454)
        at TryException.main(TryException.java:9)
Press any key to continue . . .
```

ما حدث في هذه الحالة نسميه .Exception وما ترونه في الصورة الأخيرة هي وسيلة الآلة التخيلية لإخبارنا إنها واجهت مشكلة، ولا تعرف كيف يمكن أن تحلها. وفي الحقيقة في الصورة ستجدون اسم الـ Exception وهو :

NumberFormatException.

وفي الواقع مثالنا هذا

```
1. // Here is the output generated when this example is executed.
2. public class Chp13_1{
3.         public static void main(String args[]){
4.                 int i=0;
5.                 i=i/i;
6.                         }
7. }
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at aldopaee.main(aldopaee.java:4)
```

نلاحظ أن تم إطلاق خطاء وهو القسمة على صفر فأنت في هذه الحالة قد تعالج هذا الخطاء بالمعالجة المناسبة كطباعة رسالة للمستخدم بأنه حصل خطاء وتعاود إدخال القيم مرة أخرى دون توقف البرنامج ففي مثالنا السابق يتم إيقاف البرنامج كلياً فلهذا وجدت الاستثناءات .

13.5 أنواع الاستثناءات Exception type

من الأمثل السابقة تبين أن هناك العديد من أنواع الاستثناءات ومن معرفتنا للغة Java بإنها تتكون من فصائل فإن الاستثناءات في Java هي فصائل classes وكل فصيلة تختص بنوع من الاستثناءات وجميع هذه الفصائل ثرت الفصيلة العليا Throwable وتوجد فصيلتان فرعيتان ترثان هذه الفصيلة وهما Error , Exception وهذه الفصائل موجودة في الحزمة Java.lang وهذه الفصائل الفرعية تصنف الاستثناءات هي ذات علاقة بالبرنامج أم هي ذات علاقة بآلة Java الافتراضية JVM.

13.5.1 استثناءات الحزمة java.lang :

تحتوي الحزمة java.lang على الكثير من تعليمات لغة البرمجة java الأساسية. لذلك لا تعرف الاستثناءات المأخوذة من RuntimeException ضمن البند throw الخاص بالمنهج. تعتبر هذه الاستثناءات عادية وتقريباً أي منهج يمكن أن يرفعها. يبين الجدول 1-13 الاستثناءات القابلة للاستعادة من الحزمة java.lang .
ويبين الجدول 2-13 الاستثناءات غير القابلة للاستعادة من الحزمة java.lang .

الجدول 1-13: استثناءات الحزمة java.lang

الاستثناء	السبب
ArithmeticException	خطأ حسابي، على سبيل المثال القسمة على صفر
ArrayIndexOutOfBoundsException	دليل المصفوفة إما أقل من الصفر أو أكبر من الحجم الفعلي للمصفوفة
ArrayStoreException	وجود لبس بين نوع الكائن والكائن الذي يُخزن في المصفوفة
ClassCastException	تحويل نمط الكائن إلى نمط غير ملائم
ClassNotFoundException	عدم القدرة على تحميل الصنف المطلوب
CloneNotSupportedException	الكائن لا ينفذ الواجهة Cloneable
Exception	الصنف الرئيسي للتسلسل الهرمي للاستثناء
IllegalAccessException	لا يمكن الوصول للصنف
IllegalArgumentException	المنهج يستقبل بارامتراً غير مناسب
IllegalMonitorStateException	حالة المراقب غير صحيحة(مسلك متزامن)
IllegalThreadStateException	المسلك Thread هو حالة غير صحيحة من أجل العملية المطلوبة
IndexOutOfBoundsException	الدليل هو خارج الحدود
InstantiationException	محاولة لإنشاء كائن من صنف مجرد abstract class
InterruptedException	مقاطعة المسلك
NegativeArraySizeException	حجم المصفوفة أقل من الصفر
NoSuchMethodException	غير قادر على تصميم المنهج
NullPointerException	الكائن غير موجود وبالتالي محاولة الوصول إلى القيمة Null
NumberFormatException	غير قادر على أن يحول النمط string إلى عدد
RuntimeException	صنف أساسي من أجل استثناءات الصنف java.lang
SecurityException	لا تسمح إعدادات الأمن بالعملية
StringIndexOutOfBoundsException	الدليل سالب أو أكبر من الحجم المخصص للصنف string

الجدول 2-13: أخطاء الحزمة java.lang

الخطأ	السبب
AbsractMethodError	محاولة استدعاء منهج مجرد abstract method
ClassCircularityError	هذا الخطأ لَمْ يَعُدْ مستعملاً
ClassFormatError	صيغة الصنف الثنائية غير صحيحة

Error	الصنف الرئيسي للتسلسل الهرمي للخطأ
IllegalAccessError	محاولة الوصول إلى صنف متعذر الوصول إليه
IncompatibleClassChangeError	الاستعمال غير صحيح للصنف
InstantiationError	عدم القدرة على صنع كائن (أو حالة)
InternalError	خطأ في المترجم
LinkageError	خطأ في ارتباطات الصنفِ
NoClassDefFoundError	غير قادر على إيجاد تعريف الصنف
NosuchFieldError	غير قادر على إيجاد الحقل المطلوب
NosuchMethodError	غير قادر على إيجاد المنهج المطلوب
OutOfMemoryError	خارج الذاكرة
StackOverflowError	طفحان في المكدس
ThreadDeath	يشير إلى أن المسلك thread سوف ينتهي. قد يمسك لينجز عملية تنظيف.(إذا أمسك يجب أن يعاد رميه مرة ثانية)
UnKnownError	خطأ غير معروف في الآلة الافتراضية
UnsatisfiedLinkError	يوجد روابط غير محللة (resolved) في الصنفِ المحمّل
VerifyError	خطأ في التعرف على الشيفرة التي تكون على شكل بايتات
VirtualMachineError	صنف رئيسي من أجل أخطاء الآلة الافتراضية

13.5.2 استثناءات الحزمة java.io

ترفع الأصناف في الحزمة java.io مجموعة منوعة من الاستثناءات كما هو مبين في الجدول 3-13. إن الأصناف التي تعمل مع أجهزة الدخل/ الخرج (I/O) مرشحة لترفع استثناءات قابلة للاستعادة. على سبيل المثال النشاطات المختلفة مثل فتح الملفات أو الكتابة إلى الملفات من المحتمل أن تفشل من وقت لآخر. إن الأصناف في الحزمة java.io لا ترفع أخطاء على الإطلاق.

الجدول 3-13:استثناءات الحزمة java.lang :

الاستثناء	السبب
IOException	الصنف الجذر من أجل استثناءات الدخل/الخرج
EOFException	نهاية الملف
FileNotFoundException	غير قادر على تخصيص الملف
InterruptedIOException	عملية دخل/خرج تمت مقاطعتها. تتضمن العضو bytesTransferred الذي يشير إلى عدد البايتات التي نُقلت قبل أن تتم عملية المقاطعة للعملية
UTFDataFormatException	نص خاطئ من التنسيق UTF-8

13.5.3 استثناءات الحزمة java.net :

تعالج الحزمة java.net اتصالات الشبكة. ترمي صفوفها استثناءاتها لتشير إلى فشل الاتصال وما شابه. يبين الجدول 4-13 الاستثناءات القابلة للاستعادة من الحزمة java.net . إن الأصناف في الحزمة java.net لا ترفع أخطاء على الإطلاق.

الجدول 4-13: استثناءات الحزمة java.lang:	
الاستثناء	**السبب**
MalformedURLException	غير قادر على مقاطعة محدد موقع المعلومات (URL)
ProtocolException	خطأ نظام للصنفِ Socket (خطأ في المقبس)
SocketException	استثناء للصنف Socket
UnKnownHostException	غير قادر أن يفسر اسم المضيف Host Name
UnKnownServiceException	لا يدعم الاتصال الخدمة

13.5.4 استثناءات الحزمة java.awt :

تملك أصناف الحزمة java.awt أعضاء ترمي استثناء واحداً وخطأ واحداً:

AWTException(استثناء في الحزمة AWT)
AWTError(خطأ في الحزمة AWT)

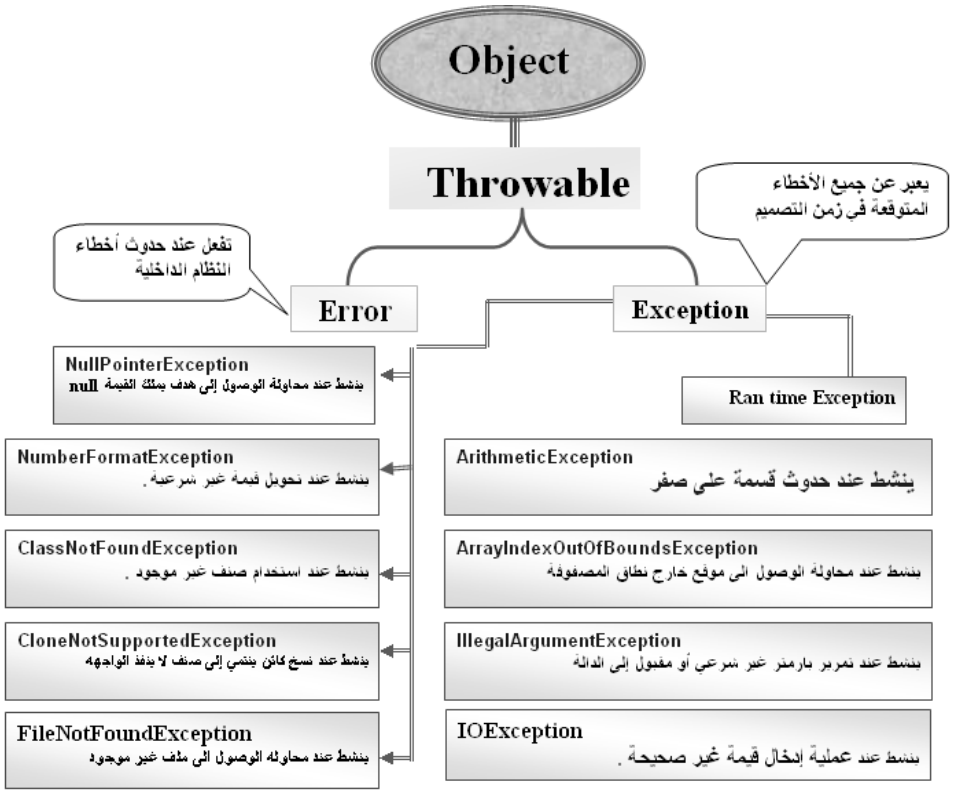
13.5.5 استثناءات الحزمة java.util :

ترمي أصناف الحزمة java.util الاستثناءات التالية:

- EmptyStackException(لا يوجد كائنات في المكدس)
- NoSuchElementException(لا يوجد كائنات أكثر في المجموعة)

13.5.6 الفصيلة (Exception)

هي الفصيلة الجذرية root class لجميع الفصائل التي تصف جميع الاستثناءات ذات العلاقة ببرنامج ويبين الشكل التالي بعض الفصائل الفرعية للفصيلة Exception ووصف كل منها.



شكل 13-1

13.6 خطوات إنشاء الاستثناءات

1- تعريف الاستثناء أو التصريح عنة

public void mymethod() throws نوع الاستثناء

2- دفع الاستثناء

throw new ----------

3- جلب الاستثناء ومعالجته

catch(-------)

13.7 إيعازات الاستثناء

1- try / يستخدم لتحديد المنطقة التي ستقع فيها الخطاء في البرنامج .

2- Catch / تأتي بعد التعليمية السابقة مباشرة لتحدد نوع الخطاء المتوقع وقد يأتي أكثر من catch في البرنامج .

3- Finally / هي اختيارية تستخدم لتنفيذ كود معين ينفذه المفسر إجباريا سوى حدث استثناء أم لم يحدث .

4- Throw / إعلان حالة الطوارئ هي الوسيلة التي تستخدمها الآلة التخ يلية في Java للإعلان عن وجود مشكلة أو خطأ في تشغيل البرنامج .

5- Throws / التصريح على الاستثناء .

13.8 الهيكل العام للاستثناءات

```
try{
---------------
جمل مشابهه للخطاء المتوقع
---------------
}catch( نوع الاستثناء اسم له )المعالجة المناسبة{
}catch(نوع أخر اسم له )المعالجة المناسبة{
}finally{ شيء يراد تنفيذه إجباريا }
```

لنحاول إضافة الاستثناء في كود القسمة على صفر

```
1. // ArithmeticException generated by the division-by-zero error:
2. class Chp13_2 {
3.     public static void main(String args[]) {
4.         int d, a;
5.         try { // monitor a block of code.
6.             d = 0;
7.             a = 42 / d;
8.             System.out.println("This will not be printed.");
9.         } catch (ArithmeticException e) { // catch divide-by-zero error
10.            System.out.println("Division by zero.");
11.        }
12.        System.out.println("After catch statement.");
13.    }
14. }
```

```
Division by zero.
After catch statement.
```

شرح المثال:

نلاحظ أنة بعد حصول الخطاء تم تجاهل أمر الطباعة وذهب مباشرة ليبحث عن catch المناسب ليعالج هذا الخطاء ومن خلال الشكل يتضح البرنامج .


إذا حصل الاستثناء المعالجة المناسبة في catch الأول أو الثاني فأنة ينفذ محتواة ثم يذهب إلى finally مباشرة ثم يخرج من try ولا ينفذ catch أخر إطلاقاً .

13.9 معالجة الاستثناءات عن طريق الكلمة المحجوزة (throws)

```
1. // Handle an exception and move on.
2. class Chp13_3 {
3.
4.   static void compute (int a[]) throws ArrayIndexOutOfBoundsException
5.         {  System.out.println(a[2]);
6.            System.out.println("Normal Exit");
7.        }
8.   public static void main(String args[]) {
9.         int a[]={1,2};
10.     try{
11.        compute(a);
12.          }catch(ArithmeticException e){System.out.println("div by 0");}
13.          catch(ArrayIndexOutOfBoundsException e){System.out.println("Caught "+e);}
14.
15.       finally{System.out.println("finally");}
16.                                                    }
17. }
```

```
Caught java.lang.ArrayIndexOutOfBoundsException
finally
```

في السطر 4 تم التصريح برأس الدالة عن الخطاء المتوقع .
نلاحظ في السطر 9 تم تعريف مصفوفة من موقعين فقط وفي السطر 11 يتم الآنتقال إلى كود الدالة ويتم في السطر 5 طباعة محتوى المصفوفة ذات الموقع الثالث وبما أن المصفوفة مكونة ن موقعين إذن فأنة يحصل استثناء فينتقل التنفيذ إلى السطر 12 ويسأل هل نوع catch من نوع الخطاء فأنة هنا لا يوافق الخطاء فينتقل إلى السطر 13 ويحصل تطابق فينفذ محتوى catch ويخرج من try .

وهذا مثال أخر:

```
1. // Displaying a Description of an Exception.
2. class Chp13_4 {
3.   public static void main (String arg[]){
4.     int denom[] = {2, 0, 0, 4};
5.         try {
```

```
6.          for (int i =0; i< 5; i++) {
7.                                      try {
8.                                  System.out.println( i+"/"+ denom[i]+"is "+i/denom[i]);
9.                                          } catch (ArithmeticException e){
10.                                            System.out.println("Can't divide by ZERO!");
11.                                                                          }
12.                                              finally{System.out.println(i+"finally by 1");}
13.                          }
14.         } catch (ArrayIndexOutOfBoundsException ex) {
15.                                              System.out.println("No matching element
found.");
16.                                                                  }
17.         finally{System.out.println("finally by 2\n");}
18.                                      }
19. }
```

شرح المثال:

نلاحظ هنا انه يوجد try بداخل try وسينفذ try الداخلي بعدد مرات اللوب وان تحقق طبع الخطاء الموضح، كما في الأسطر (7- 8)، ثم يطبع finally by 1 بعدد مرات اللوب ثم ينتهي بحصول خطاء المصفوفة فيطبع رسالة الخطاء للمصفوفة ثم يطبع finally الخارجي .وهذا ناتج تنفيذ البرنامج:

```
0/2is 0
0finally by 1
Can't divide by ZERO!
1finally by 1
Can't divide by ZERO!
2finally by 1
3/4is 0
3finally by 1
4finally by 1
No matching element found.
finally by 2
```

13.10 إطلاق الاستثناء

```
1. // Demonstrate multiple catch statements.
2. class Chp13_5 {
3.   public static void main (String arg[]){
4.          try {
5.                int i =0;
6.                      if(i<=0)throw new Exception("Throw an Error");
7.                      System.out.println("Main-");
8.           } catch (Exception e) {
9.                              System.out.println(e);
```

```
10.                                     }
11.          finally{System.out.println("Finally Block!");}
12.         System.out.println("Finally Block!");
13.                                                 }
14. }
```

```
java.lang.Exception: Throw an Error
Finally Block!
Finally Block!
```

نلاحظ في السطر 6 انه تم تنشيط أو إطلاق استثناء من نوع أب أي عام وثم الآنتقال إلى السطر 8 لمعالجة هذا الاستثناء ثم تنفيذ محتوى finally و تنفيذ بقية الكود .

13.11 إعادة إطلاق الاستثناء

```
1. // rethrow an Exception.
2. class Chp13_6 {
3.          public static void method1() throws Exception{
4.                    try {
5.                              System.out.println("method1+");
6.                              throw new Exception();
7.                       } catch (Exception e) {
8.                                                  System.out.println("rethrow an exception");
9.                                                  throw e;
10.                                               }
11.                   finally {System.out.println("finally block1");}
12.                                                                 }
13.
14.         public static void main (String arg[]){
15.                   try {
16.                             System.out.println("main+");
17.                             method1();
18.                             System.out.println("main-");
19.                        } catch (Exception e) {System.out.println("caught from main");}
20.                   finally {System.out.println("finally block2");}
21.                   System.out.println("fnished");
22.                                                    }
23. }
```

```
main+
method1+
rethrow an exception
finally block1
caught from main
finally block2
fnished
```

شرح المثال:
يبدأ تنفيذ البرنامج من السطر 17 – 15 ثم ينتقل إلى الدالة في السطر 3 فينفذ محتواها وفي السطر 6 يتم إطلاق استثناء وتتم معالجته في السطر 7 وفي السطر 9 يعاد إطلاق استثناء من نفس النوع السابق فينفذ محتوى finally الداخلي ثم ينتقل للسطر 19 ليعالج هذا الاستثناء و فينفذ محتوى finally الخارجي ثم بقية كود البرنامج .

عند إطلاق استثناء تم تأتي return فإنها تبطل عمل throw وتخرج من الطريق وهذا المثال شبيه السابق ولا كن الفرق كلمة return .

```
1. // rethrow an Exception.
2. class Chp13_7 {
3.         public static void method1() throws Exception{
4.                 try {
5.                         System.out.println("method1+");
6.                         throw new Exception();
7.                     } catch (Exception e) {
8.                                                 System.out.println("rethrow an exception");
9.                                                 throw e;
10.                                               }
11.                 finally {System.out.println("finally block1");return;}
12.                                                                 }
13.
14.         public static void main (String arg[]){
15.                 try {
16.                         System.out.println("main+");
17.                         method1();
18.                         System.out.println("main-");
19.                     } catch (Exception e) {System.out.println("caught from main");}
20.                 finally {System.out.println("finally block2");}
21.                 System.out.println("fnished");
22.                                                 }
23. }
```

نلاحظ انه بعد تنفيذ السطر 11 وجد أمر return فتم تجاهل الاستثناء الذي أطلق في السطر 9 فانتقل إلى الدالة الرئيسية وتابع تنفيذ بقية الكود بشكل طبيعي .
وهذا ناتج تنفيذ البرنامج:

```
main+
method1+
rethrow an exception
finally block1
main-
finally block2
fnished
```


إذا استخدمت الكلمة return في الدالة الرئيسية فهذا يعني إنها البرنامج كلياً بعد تنفيذ محتوى finally وهذا المثال يبين ذلك :

```
1. // Exit Programs Exception to Return.
2. class Chp13_8 {
3.         public static void main (String arg[]){
4.                   try {
5.                         int r2=10/0;
6.                      } catch (ArithmeticException e) {
7.                                                                System.out.println("Calculation
Error");
8.                                                                 return;
9.                                                               }
10.                              catch (Exception e){System.out.println("Genreal Exption");
11.                                                          }
12.                              finally{System.out.println("Finally block");}
13.
14.                              System.out.println("fnished");
15.                                                          }
16. }
```

نلاحظ في السطر 8 وجد المترجم return فانتقل إلى السطر 12 لينفذ محتوى finally وينهى البرنامج وهذا ناتج تنفيذ البرنامج:

```
Calculation Error
Finally block
```


من الأخطاء البرمجية الشائعة تقديم catch يحتوي على نوع من أنواع الاستثناء أب Exception على استثناء ابن NumberFormatException كهذا الكود لن ينفذ إطلاقاً .

```
1. // catch not reached.
2. class Chp13_9 {
3.         public static void main (String arg[]){
4.                   try {
5.                                System.out.println("main+");
6.                              int i = 0;
7.                              i=i/1;
8.                      } catch (Exception e) {System.out.println(e);
9.                      } catch (NumberFormatException e) {System.out.println(e);}//Error
```

```
10.                                                    }
11. }
```

هل كل منهج ينبغي عليه أن يمسك كل إستثناء؟

تصور ماذا سيحدث لو وجد منهج يستدعي منهجاً آخر يرفع استثناء ما لكن يختار ألا يمسكه. في المثال التالي يستدعي المنهج ()main المنهج ()foo الذي بدوره يستدعي المنهج ()bar ويسجل المنهج ()bar الصنف Exception في بنود رماية الأخطاء، بما أن المنهج ()bar يرفع استثناء ولا يمسك به فإن المنهج ()foo يملك الفرصة ليمسك به لكن لا يملك المنهج ()foo كتلة catch لذلك فهو لا يستطيع أن يمسك الاستثناء. في هذه الحالة يتكاثر الاستثناء ضمن مكدس الاستدعاء الخاص بمستدعي المنهج ()foo ألا وهو المنهج ()main ولن يتم التنفيذ.

```
1. // Statement not reached
2. import java.lang.Exception;
3. public class Chp13_10{;
4.       public static void main ( String[] args) {
5.        try {
6.              foo();
7.             }
8.        catch( Exception e)
9.           {
10.          System.out.println( "Caught exception –" +e.getMessage() );
11.           }
12.    }
13.   static void foo() throws Exception {
14.                throw new Exception ("Who cares");
15.                                        bar();// Error
16.                                           }
17.   static void bar() throws Exception {
18.                throw new Exception ("Who cares");
19.                                           }
20. }
```

قواعد مهمة

إذا وجد المترجم في try الخارجي خطاء والمعالجة المناسبة في try الداخلي فأنة لا يدخل إلى try الداخلي إطلاقا ولا حتى finally تبعة بل يذهب مباشرة إلى catch الخاص بة و ينفذ المعالجة المناسبة كهذا المثال:

```
1. // Local Try
```

```
2. class Chp13_11 {
3.         public static void main (String arg[]){
4.                 try {
5.                     int r2 = 10/0;
6.                         try{
7.                             //------
8.                         } catch (ArithmeticException e){
9.                                 System.out.println("ArithmeticException");
10.                        }finally {System.out.println("Finally block local");}
11.
12.                    } catch (NullPointerException e)
13.                        {System.out.println("NullPointerException");
14.                        } finally {System.out.println("Finally block");}
15.                System.out.println("Finished");
16.                                                 }
17. }
```

```
Finally block
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Chp11_10.main(Chp11_10.java:5)
```

إن لم يجد المعالجة المناسبة فان لغة Java تتكفل بالمسئولية وتطبع نوع الخطاء والاستثناء المناسب لها ومثالنا Chp13_11 يبين ذلك .

إذا وجد المترجم في try الداخلي خطاء ولم يلقي المعالجة المناسبة فأنة ينفذ فقط finally الخاص به ثم يخرج إلى try الخارجي ليبحث المعالجة الناسبة وينفذ finally ل try الخارجي .

```
1. // Local Try
2. class Chp13_12 {
3.         public static void main (String arg[]){
4.                 try {
5.
6.                         try{
7.                             int r2 = 10/0;
8.                            } catch (NullPointerException e)
9.                          {System.out.println("NullPointerException");
10.                        } finally {System.out.println("Finally block local");}
11.
12.                    } catch (ArithmeticException e){
13.                                 System.out.println("ArithmeticException");
14.                }finally {System.out.println("Finally block ");}
15.                System.out.println("Finished");
16.                                                 }
```

```
17. }
```

```
Finally block local
ArithmeticException
Finally block
Finished
```

خلاصة القول try الداخلي يخرج إلى الخارجي وليس العكس .

يراد منك استثناء يعرف هل للعدد جذر صحيح أم لا فان كان للعدد المدخل جذر عشري فأنة يطلق استثناء وان لا تأتي بجذر العدد المدخل ؟

```
1. // Asc Root Number Integer
2. import java.io.*;
3. class Chp13_13{
4.         //Function Check
5.         static void check(int a)throws Exception{
6.                                 int c=0;
7.                                 for (int i = 1; i <= a / 2; i++)
8.                                         if (i * i == a)c=i;
9.                                         if(c==0)throw new Exception();
10.                                        System.out.println("SQRT = "+c);
11.                                                                 }
12.

13.        public static void main(String k[])throws IOException{
14.                        int a;
15.                        BufferedReader b = new BufferedReader(new
InputStreamReader(System.in));
16.                        a=Integer.parseInt(b.readLine());
17.                        try{
18.                            check(a);
19.                           }catch(Exception e){System.out.println("NOT SQRT NUMBER");}
20.                        finally{System.out.println("BY");}
21.                                                                                 }
22. }
```

```
5
NOT SQRT NUMBER
BY
9
SQRT = 3
BY
```


عندما المترجم يوجه كلمه System.exit(); فأنة يخرج من البرنامج ولا ينفذ شيء حتى finally وهذا المثال يبين ذلك :

```
1. // Exit Of System.exit(1).
2. class Chp13_14{
3.         public static void main(String k[]){
4.                 int i=3;
5.                 System.out.println("star");
6.                 try{
7.                     if(i%2!=0){System.exit(1);}
8.                    }catch(Exception e){System.out.println("Chp13_14");
9.                    }finally{System.out.println("BY");}
10.                System.out.println("Finsh");
11.                                         }
12. }
```

```
star
```

13.12 متى نستخدم الاستثناءات

تفصل معالجة الاستثناء شفرة معالجة الخطاء عن مهام البرمجة العادية ، بحيث يصبح البرامج سهله القراءة والتعديل. كن حذراً عند استخدام ك للاستثناءات ، حيث تحتاج هادة معالجة الاستثناءات إلى زمن ومصادر أكثر مما تطلبه البرمجة العادية ، إذ تتطلب صنع كائن جديد من أجل الاستثناء واستعادة مكدس الاستدعاء ونشر الاستثناء في المناهج ، فإذا كنت ترغب في أن تتم معالجة الاستثناء بواسطة مستدعية عندها يجب صنع كائن من الاستثناء ورمية، أما إذا كنت تستطيع التعامل مع الاستثناء في المنهج الذي حدث فيه فإنك لا تحتاج إلى رمية.
بشكل عام يتم ترشيح (رمي) الاستثناء العامة التي من الممكن أن تحدث ضمن أصناف متعددة في المشروع إلى أصناف الاستثناء ، ومن الأفضل معالجة الأخطاء البسيطة التي من الممكن أن تحدث ضمن مناهج منفصلة بشكل محلي وبدون رمي الاستثناءات.
أين يجب عليك استخدام الكتلة try-catch في الشفرة ؟
تستخدم في الأوضاع التي تكون فيها الأخطاء غير متوقعة .

السطر 2 عمل توريث للصنف من صنف الاستثناء .
السطر 5 دالة خاصة بمكتبة الاستثناء تنفذ تلقائياً حال حصول تنشيط للاستثناء

13.13 إنشاء استثناء خاص بك

```
1. // Creating Your Own Exception Subclasses.
2. class Chp13_MyException extends Exception {
3.         private int detail;
4.         Chp13_MyException(int a) {detail = a;}
5.         public String toString() {return "MyException[" + detail + "]";}
6.                                                                 }
7. class Chp13_15 {
8.         static void compute(int a) throws Chp13_MyException {
9.                   System.out.println("Called compute(" + a + ")");
10.                  if(a > 10) throw new Chp13_MyException(a);
11.                  System.out.println("Normal exit");
12.                                                                         }
13.        public static void main(String args[]) {
14.                  try {
15.                            compute(1);
16.                            compute(20);
17.                            } catch (Chp13_MyException e) {
18.                            System.out.println("Caught " + e);
19.                                                                  }
20.                                                       }
21. }
```

```
Called compute(1)
Normal exit
Called compute(20)
Caught MyException[20]
```

عند حصول خطاء داخل catch فإن المترجم لا يطلع ولا ينزل للبحث عن المعالجة المناسبة بل يطبع نوع الخطاء من داخل لغة Java وينفذ محتوى Fanlly وينهي البرنامج وهذا المثال يبين ذلك .

```
1. // Demonstrate finally.
2. class Chp13_16{
3.         public static void main(String k[]){
4.                   int i=0,b[]={2,3};
5.                   System.out.println("star");
6.                   try{
7.                             i/=i;
8.                   }catch(ArrayIndexOutOfBoundsException
e){System.out.println("Chp13_16_1");
9.                   }catch(ArithmeticException e){System.out.println("Chp13_16_2");b[2]=i;
10.                  }catch(Exception e){System.out.println("Chp13_16_3");
11.                  }finally{System.out.println("BY");}
12.                  System.out.println("Finsh");
13.                                                       }
14. }
```

```
star
Chp11_15_2
BY
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException
        at Chp11_15.main(Chp11_15.java:9)
```

إذا أنشأت استثناء خاص بك واحتوى على داله بناء فان إذا حصل إطلاق لهذا الاستثناء داخل البرنامج فان محتوى دالة البناء للاستثناء تنفذ أولاً ثم ما بداخل catch ثانياً وهذا المثال يبين ذلك:

```
1. // Exp
2. class Chp13_TestException extends Exception {
3.                     Chp13_TestException(){System.out.println("TestExce");}
4.                                                                     }
5. class Chp13_17 {
6.       public static void main(String[] args) {
7.                     String arg="t";
8.                     System.out.println(arg);
9.        try {
10.                     thrower(arg);
11.             System.out.println("Test ");
12.           }catch (Exception e) {System.out.println("Test 1");}
13.         }
14.      static void thrower(String s) throws Chp13_TestException {
15.      try {
16.           if (s.equals("d")) {int i = 0; i/=i;}
17.           if (s.equals("t"))throw new Chp13_TestException();
18.       }catch (Chp13_TestException e) {System.out.println("Test 2");}
19.        }
20. }
```

```
t
TestExce
Test 2
Test
d
Test 1
```

شرح المثال:

السطر 17 إذا حصل إطلاق للاستثناء الخاص بك فأنة يذهب إلى الاستثناء وينفذ محتواة ثم ينفذ محتوى catch .

نلاحظ عند عدم إطلاق الاستثناء الخاص بك فأنة يبحث عن catch المناسب في البرنامج الرئيسي ويعالجه ولا علاقة له بالصنف الخاص بك وما بداخلة من دوال بناء وغيرة .

سؤال للقارئ النبيل فقط ؟

```
1. // سؤال للقارئ النبيل
2. class Chp13_18 {
3.       public static void main(String[] args) {
4.                   String arg="ammar";
5.        try {
6.              thrower(arg);
7.            }catch (Exception e) {
8.                   System.out.println("ammar 1");}
9.                                }
10. static void thrower(String s) throws Exception {
11.  try {
12.         if (s.equals("ammar")){int i = 0; i/=i;}
13.       }catch (Exception e){
14.                  System.out.println("ammar 2");throw  e;}
15.
16.    B: System.out.println("ammar aldopaee");
17.                                                  }
18. }
```

```
ammar 2
ammar 1
```

من المعروف انه عند حصول إطلاق للاستثناء داخل ألداله فان المترجم يوقف ايعازات محتوى الدالة ويذهب مباشرة للبحث عن catch المناسب خارج الدالة مثل هذا المثال .
السؤال هنا أنا أريد تنفيذ محتوى ألداله كاملاً سوى حصل إطلاق للاستثناء أم لا ففي هذا الكود سيتم تجاهل تنفيذ أمر الطباعة الموجود في السطر B فما الحل ؟
بشرط بدون استخدام finally
فيكون الناتج هكذا:

```
ammar 2
ammar aldopaee
ammar 1
```

13.14الخلاصة:

تسمح آلية معالجة الاستثناء في لغة البرمجة java لمناهجك أن تقدم تقريراً عن الأخطاء في أسلوب لا يمكن أن يتم تجاهله، وكل استثناء يُرفع يجب أن يتم مسكه أو سينتهي التطبيق. إن الاستثناءات هي فعلياً أغراض من أصناف تنشأ من الصنف Trowable .

لذلك تجمع الاستثناءات بين المعطيات والمناهج. يتضمن كائن الاستثناء عادة عبارة من الصنف تشرح ما هو الخطأ.

تساعدك معالجة الاستثناءات أن تدمج معالجة الخطأ في مكان واحد.

تمارين الفصل:

1. عدد شروط حصول الاستثناءات التي قمنا باستخدام ها ضمن برامج هذا الفصل ؟
2. أذكر بعض الاستثناءات الشائعة ؟
3. إذا توفر ضمن البرنامج التعليمة:

Throw

فأين تتوقع أن ترد؟ وماذا يحدث إذا ظهرت هذه التعليمة ضمن جزء أخر مختلف من أجزاء البرنامج ؟

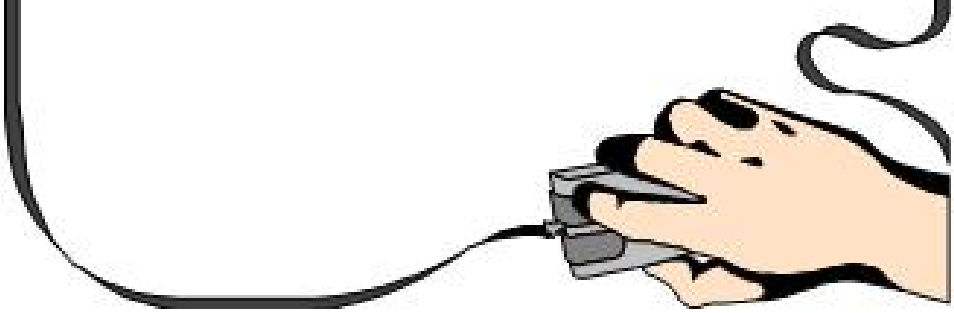
4. عدد فوائد معالجة الاستثناءات مقارنة مع الوسائل التقليدية لمعالجة الأخطاء ؟
5. لنفرض أنه لدينا برنامج وقد تم إلقاء استثناء ضمنة وتجري حالياً عملية المعالجة له. فيفترض أن معالج الاستثناء نفسه قام بإلقاء الاستثناء نفسه ، هل سيؤدي ذلك إلى الدخول في حلقة غير منتهية ؟ اكتب برنامج للتحقق من وجة نظرك.
6. اكتب برنامج لتوضيح أهمية ترتيب معالجات الاستثناء حيث يتم دوماً تنفيذ أول معالج مطابق. قم بترجمة وتشغيل برنامجك بطرقتين مختلفتين لتوضيح فكرة واستخدام معالجين مختلفين في كلا الحالتين.
7. اكتب برنامج لتوضيح أنه لا يجب على الكتلة try المرتبطة بتابع أن تقوم بالتقاط كافة الأخطاء الممكنة الحاصلة ضمنها وإنما يمكن ترك بعضاً منها ليتم معالجتها ضمن كتل أخرى.
8. أكتب برنامج لتوضيح كيفية إعادة إلقاء الاستثناء ؟
9. أكتب برنامج يجري إلقاء استثناء ضمن منهج جرى استدعاؤه ضمن منهج أخر وبقي على الرغم من ذلك المعالج catch الموجود بعد الكتلة try المتضمنة لسلسلة الاستدعاءات قادراً على التعامل مع الاستثناء ؟

Chp14

الدخل والخرج

في نهاية هذا الفصل سوف تتعلم :

- اكتشاف خصائص الملف باستخدام File.
- التعرف على مجاري (Streams) الدخل والخرج وكيفيه صنعهم.
- الفرق بين مجاري البايتات ومجاري المحارف.
- الكتابة والقراءة على الملفات الخارجية .
- استخدام الصنف RandomAccessFile من أجل القراءة والكتابة .
- عرض صناديق الحوار المتعلقة بفتح الملف وحفظة باستخدام الصنف JFileChooser.



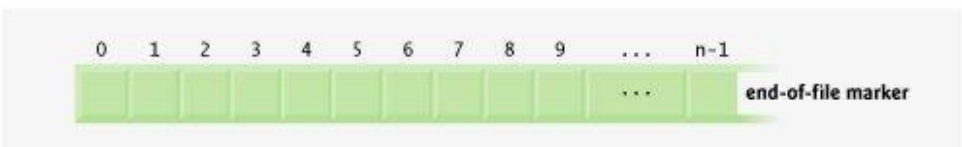
14.1 مقدمة INTRODUCTION

كثيراً ما يستصعب مبتدئو البرمجة مواضع التعامل مع الملفات ، والأمر ليس لصعوبة الموضوع لحد ذاته بل إلى عرضة والطريقة التي يحاول فيها المبتدئ التعامل مع الموضوع فهو ربما إنها أصعب مواضيع البرمجة مبدئياً ، وربما في احد الأيام أراد تطوير برنامجه ليكون قادراً على التعامل مع الملفات وحتى يفعل ذلك فأنه لا يأخذ هذا الموضوع بشكل جدي ويتجاوز أساسياته ليذهب بعيداً كـ ي يتعامل مع المواضيع المتقدمة نسبياً والنتيجة لا شيء عدا إضاعة الوقت فيما لا يجدي وحتى تكون قادراً على فهم هذه الوحدة فأرجو منك أن تتعامل معها على إنها وحدة متكاملة لها أساسياتها الأولية وما إلى ذلك ولا تتعامل معها على إنها وحدة أمثلة تطبيقية فحسب .

يتم تـ خزين البيانات ضمن المتحولات والمصفوفات بشكل مؤقت وتستخدم الملفات لتخزين كميات كبيرة من البيانات بشكل دائم . يقوم الحاسب بتخزين الملفات بدورة ضمن وحدات التخزين الثانوية مثل الأقراص المغناطيسية ، الأقراص الضوئية ووحدة التخزين الخارجية flash disc . سوف نشرح في هذا الفصل كيف يمكن إنشاء ملف معطيات والتعامل معة من خلال برامج مكتوبة بلغه Java . وسوف نتعرض لتقنيات إدخال وإخراج البيانات بواسطة الصنف File من أجل الحصول على خصائص الملفات وحذفها وإعادة تسميتها.

14.2 الملفات ومجاري الدخل والخرج I/O Streame

تنظر لغة Java للملفات على إنها مجاري تسلسلية من البايتات (الشكل1-14).
ينتهي كل ملف بإشارة نهاية الملف end-of-file masker أو عند رقم بايت محدد ومسجل ضمن بنية للبيانات مدارة من قبل النظام . عند فتح ملف ، يتم إنشاء غرض ومجرى متصل مع هذا الغرض . فتوجد عدة أغرض لإنشاء الاتصال بين الملفات.
فالمجاري هي عبارة عن كائنات تملك عدة مناهج من أجل قراءة البيانات وكتابتها أو مسح المجرى وإغلاقه وحساب عدد البايتات في المجرى.



شكل 1-14

14.2.1 المجرى (Stream)

يغطي نظام (الدخل/الخرج) في لغة البرمجة Java جميع احتياجات المبرمج للتفاعل مع المحيط الخارجي، بما في ذلك الوصول إلى الملفات أو إلى واجهة سطر الأوامر (Console) و حتى عند الاتصال عبر الشبكة.

إضافة إلى ذلك فإنه يدعم طرق التخاطب المختلفة مع تلك الوسائط، ابتداءً بالوصول التسلسلي إلى المعطيات، أو الوصول العشوائي، أو معالجة المعطيات الثنائية، أو المحارف، أو الوصول بالسطر، أو بالكلمة... الخ.

و يتم ذلك عن طريق المكتبة IO الغنية بالأصناف المختصة بـ معالجة كل عملية من العمليات السابقة.

14.3 أصناف المجاري و الأصناف القارئة /الكاتبة

يتعامل الجزء Stream من الحزمة java.io (انظر الشكل 2-14) مع (قراءة أو كتابة) بايت من المعطيات (من المعلوم أن البايت مؤلف من ثمانية بتات في معظم الحواسب الحديثة).

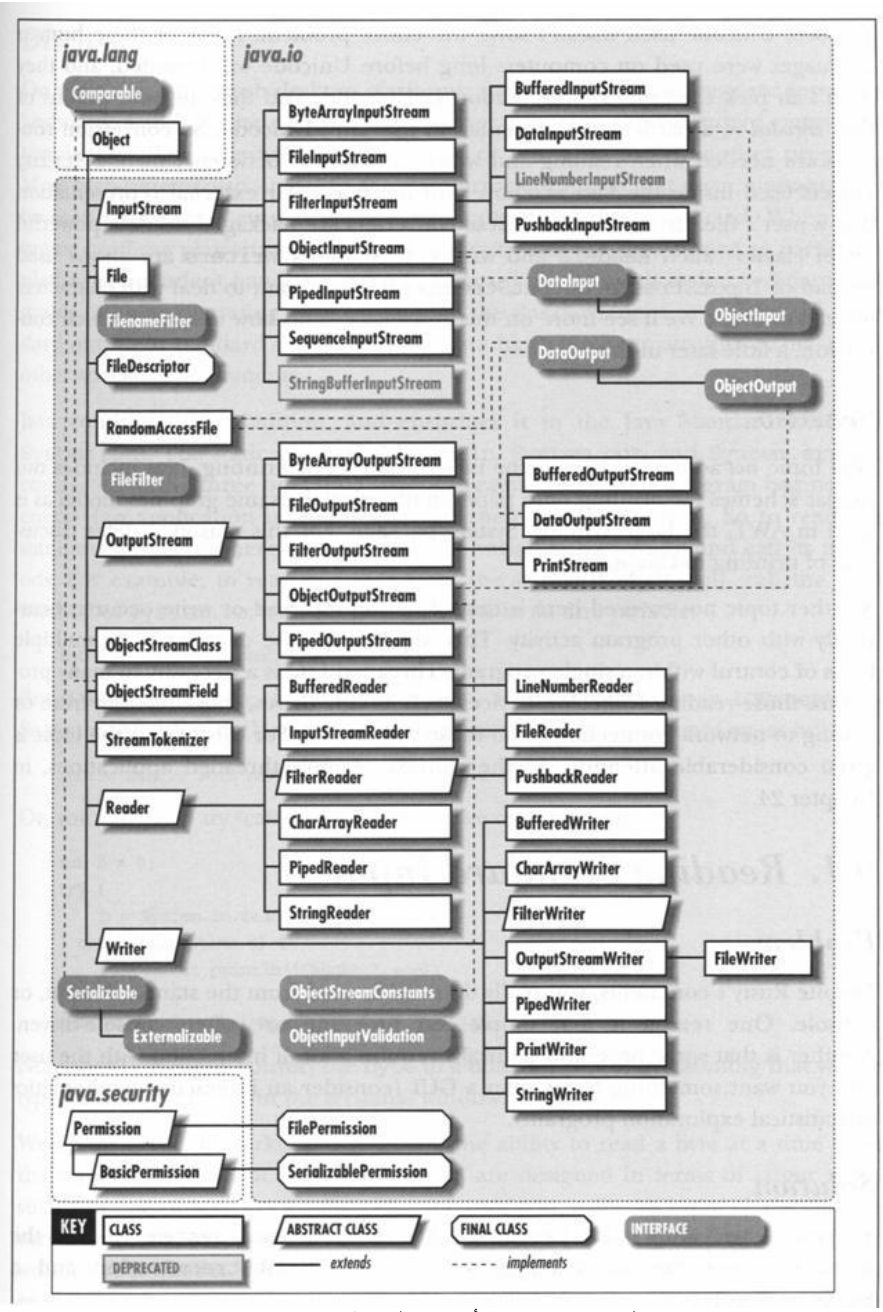
قد يمثل هذا البايت محرفاً (Character) أو رقماً أو أي محرف لغوي آخر. على أية حال، صممت اللغة Java لتكون لغة برمجة عالمية، لذلك فإن ثمانية بتات غير كافية لمعالجة المحارف المختلفة في لغات العالم الأخرى (كاللغات التي تعتمد على الكتابة النصية (Scrip-based Language) كاللغة العربية والهندية) إضافة إلى اللغات المنقوشة (المرسومة) كالصينية واليابانية والكورية والتي تحتوي أبجديتها أكثر من 256 حرف، في حين أن العدد الأعظمي للمحارف الممثلة بثمانية بتات هو 256 ، لذلك جرى توحيد مجموعات شفرات المحارف لهذه اللغات لنحصل على ما يسمى شفرة Unicode أو الشيفرة العالمية والتي سرعان ما شاع استخدامها.

بالطبع تستخدم اللغة Java ترميز Unicode ، وبذلك تسمح بقراءة وكتابة النصوص لجميع لغات العالم، ويتم ذلك باستخدام الأصناف القارئة Reader و الأصناف الكاتبة Writer.

لكن الترميز Unicode لم يحل كامل المشكلة ، حيث يوجد العديد من اللغات والتي استخدمت في الحواسب قبل ظهور الترميز Unicode ولم يتم تمثيلها باستخدام الترميز Unicode. حيث تمتلك هذه اللغات عدداً هائل من الملفات المشفرة بتمثيل خاص (مغاير للترميز Unicode).

لذلك ظهرت الحاجة إلى روتينات تحويل عند القراءة أو الكتابة للتحويل بين أغراض سلاسل الترميز Unicode والمستخدمة داخل آلة Java الافتراضية إلى التمثيل الخارجي الخاص بملفات المستخدم (الوسائط التي تتم إليها عملية القراءة أو الكتابة).

تم وضع روتينات التحويل المذ كورة في مجموعة من الأصناف تدعى بالأصناف القارئة Readers و الأصناف الكاتبة Writers والتي تستخدم عند التعامل مع المحارف وليس مع البايتات. ستتعرض هذا الفصل إلى الكثير من هذه الروتينات وكيفية تحديد الروتين المراد استخدامه وكيفية استخدامه.



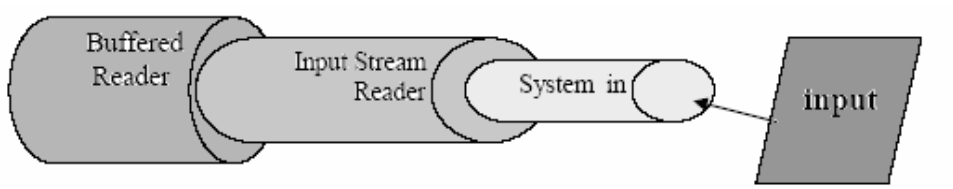
شكل 2-14 : يوضح أصناف الحزمة java.io

تتعامل الفئات inputstream , outputstream والفئات المشتقة منها مع مجاري دخل البايت . و تتعامل الفئات Write , Reader والفئات المشتقة منها مع مجاري دخل من المحارف .

14.4 الدخل Input

نلاحظ لغة Java تختلف عن بقية اللغات المشهورة C++ بالإدخال ففي C++ نجد سهولة تامة باستخدام أوامر الإدخال دون تعقيد ولكن في Java تعد الإدخال في مراحل شبة معقدة .

تقدم لغة Java مجاري مؤقتة (buffered) كما في الشكل 14-3 التي تستخدم مصفوفة مؤلفة من البايتات أو من المحارف على حسب طلب المبرمج



شكل 14-3

ومن خلال الشكل 14-3 السابق نلاحظ أن الإدخال بلغة Java يتكون من ثلاثة أنابيب

System.in

وهو الآنبوب الأول ويعمل على قراءة بايت واحد في كل مرة

Input StreamReader

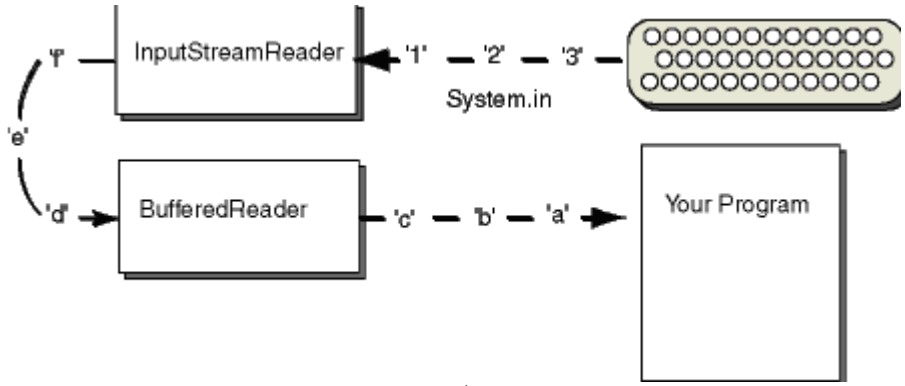
ويعمل على تحويل كل 2 بايت إلى حرف أو رمز

BufferedReader

ويعمل على تجميع هذه الحروف أو الرموز في الذاكرة المؤقتة لعمل منها سلسلة

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

هنا القارئ br هو من نوع BufferedReader Class وBufferedReader Class لة المعامل InputStreamReader وInputStreamReader Class له المعامل System.in وهو كلاس للقراءة بايت من لوحة المفاتيح والشكل 14-4 يبين ذلك .



شكل 14-4

1. وأول ما نقوم به عند أي عملية الإدخال نستدعي مكتبة الإدخال في بداية البرنامج

Import java.io.*;

2. نكتب إلية الإدخال التي نريدها وليكن إدخال عدة بايتات

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

3. في دالة main نكتب استثناء الإدخال

public static void main(String args[]) throws IOException

وننوه إلى القراء أنة دا ئماً يتم قراءة البيانات من لوحة المفاتيح بصيغة أسكي (Ascii) ،إذا كانت القراءة بايت واحد ، أم بصيغة سلسلة إذا كانت القراءة بعدة بايتات ، وعلى المبرمج تحويل من سلسلة نصية إلى أرقام عددية int باستخدام الدوال الخاصة بالسلاسل.

وهذا كود يبين كيفية الإدخال باستخدام بايت واحد

```
1. // برنامج يدخل محرف من لوحة المفاتيح
2. import java.io.*;
3. class Chp14_1{
4.         public static void min(String args[])throws IOException{
5.                 int b;
6.                 b=System.in.read();// يتم خزن النحرف المدخل في هذا المتغير
7.
8.                 System.out.println(b);
```

```
9.              }
10.     }
```

شرح المثال:
نلاحظ في المثال السابق أن عملية الإدخال تأخذ بايت واحد من لوحة المفاتيح وعند الطباعة كما في السطر 8 ينتج شفرة المحرف المدخل .

وإذا أردنا طباعة المحرف المدخل كما أدخل نعمل عملية تحويل المعطيات كما ذكرناه سابقاً .

وهذا المثال يبين ذلك :

```
1. // Use a BufferedReader to read characters from the console.
2. import java.io.*;
3. class Chp14_2 {
4.         public static void main(String args[])throws IOException{
5.                 char c;
6.                 BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
7.                 // read characters
8.                         c = (char) br.read();
9.                         System.out.println(c);
10.                     }
11. }
```

شرح المثال:
السطر 8 تتم عملية تحويل نمط العدد من أسكي إلى محرف وقد بينا ذلك سابقاً أنة يتم القراءة بشكل أسكي إذا كان بايت واحد .

14.4.1 قراءة سلسلة من لوحة المفاتيح

يبين هذا المثال كيفية إدخال سلسلة مكونة من عدة بايتات:

```
1. // Read a string from console using a BufferedReader.
2. import java.io.*;
3. class Chp14_3 {
4.         public static void main(String args[])throws IOException{
5.                 String c;
6.                 BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
7.                 // read string
8.                         c = br.readLine();
9.                         System.out.println(c);
10.                     }
11. }
```

شرح المثال:
نلاحظ الاختلاف هو عندما يقرءا بايت فإننا نعرف المتغير من نوع char وجملة القراءة read().

أم عندما نقرأ عدة بايتات فإننا نعرف المتغير من نوع String وجملة القراءة readLine(). أي سطر كامل .

14.5 الملفات Files

الملف في المكتب هو وعاء لحفظ وتخزين المعلومات التي تخص موضوعا معينا وتحت اسم خاص به ومن ثم يتم إجراء عدد من العمليات من إضافة مستند أو تعديل بيان أو حتى إلغاء الملف بالكامل - كذلك الحال بالنسبة للحاسبات الآلية ؛ فملف الحاسب الآلي هو وسيلة لحفظ البيانات والمعلومات من البرامج والنصوص ، وهذه الملفات يتم حفظها في إحدى أنواع الذاكرات الخاصة بالحاسب الآلي الرئيسية منها أو الثانوية . ويقوم الحاسب الآلي بالتعامل مع هذه الملفات وفق قواعد معينة حسب نوع كل ملف.

14.5.1 أنواع الملفات

- ملفات البرامج PROGRAM FILES
- ملفات البيانات DATA FILES
- ملفات النظام SYSTEM FILES
- ملفات النصوص TEXT FILES
- ملفات الصور IMAGE FILES
- ملفات الصوت AUDIO FILES

1. **ملفات البرامج PROGRAM FILES**

وتختص بحفظ البرامج التي تم صياغتها بواسطة إحدى لغات البرمجة وتحولت بواسطة المترجمات إلى لغة الآلة ، ومثال على ذلك كتابة برامج بلغة COBOL ، لحساب المرتبات لشركة معينة أو كتابة برنامج بلغة C لتصميم وتنفيذ ألعاب على الحاسب الآلي .

2. **ملفات البيانات DATA FILES**

و تحتوي كما يتضح من تسميتها على البيانات والمعلومات التي تتم معالجتها بواسطة الحاسب الآلي : إضافة ، إلغاء ، تعديل ، اطلاع. وغالبا ما يتم حفظ هذه البيانات داخل الملفات في إطار قواعد بيانات DATA BASE ويقوم نظام إدارة

قواعد البيانات بالتعامل معها وربطها ببعضها للاستفادة منها إما في صورة مباشرة INTERACTIVE أو من خلال برنامج يتم تصميمها بواسطة المستخدمين .

3. **ملفات النظام SYSTEM FILES**

هذه الملفا ت تحتوي على أوامر التشغيل الخاصة بالحاسب الآلي ومكوناته . ومن ثم يطلق عليها نظام التشغيل OPERATING SYSTEM ، وهي أربعة أنواع من الأوامر : النوع الأول مثبت في وحدة خاصة من الذاكرة الأساسية " ROM MEMORY ONLY BASIC ، وتحتوي على أوامر التشغيل الأساسية الخاصة بالتحكم في المدخلات والمخرجات إلى وحدة المعالجة المركزية ، أما النوع الثاني فيحتوي على أوامر التشغيل التي تتعامل مع أسطوانات التخزين DISK COMMANDS ، والنوع الثالث يختص بأوامر الملفات FILE COMMANDS ، والفهارس هنا هي أسلوب لترتيب وتنظيم الملفات المخزنة في وسائط التخزين .

4. **ملفات النصوص TEXT FILES**

تحتوي هذه الملفات على بيانات نصية يمكن الاطلاع عليها مباشرة بواسطة المستخدم بإحدى برامج معالجة النصوص WORD PROCESSORS وعادة ما يتم إجراء بعض الزيادات بواسطة هذه المعالجات خا صة في حالة النماذج متعددة للخطوط أو الربط مع الصور والرسومات .

5. **ملفات الصور IMAGE FILES**

و تحتوي على التمثيل الرقمي للصور و الرسومات ، و يتم إيصال هذه الصور إلى الحاسب بواسطة الماسح الضوئي Scanner . و التي تقوم بتحويل مكونات الصورة و درجاتها إلى شكل رقمي و ه ناك عديد من أنواع التمثيل الرقمي و أساليب الحفظ لهذا النوع من الملفات مثل TIFF, PCX

6. **ملفات الصوت AUDIO FILES**

تحتوي هذه الملفات على التمثيل الرقمي للصوت الذي يتم إدخاله إلى الحاسب الآلي عن طريق ميكروفون MIC متصل بمحول أو كارت صوت مثبت بإحدى المقسمات SLATES

14.6 البنية الهرمية للملفات

يتم التعامل مع كافه المعطيات المخزنة ضمن الحاسب على إنها مجموعة من الأصفار والواحدات. يحدث ذلك لأنه أسلوب بسيط واقتصادي لبناء أجهزة إلكترونية ذات حالتي استقرار (الحالة 0 والحالة 1). لاحظ أن أكثر التوابع تأثيراً وإدهاشاً هي التوابع التي تتضمن عمليات أساسية على الأصفار والواحدات. تعتبر الخانة bit أصغر عنصر تخزين في الحاسب ويمكن لها أن تتضمن القيمة 0 أو القيمة 1.

تستطيع دارات الحاسب القيام بالعديد من الأعمال البسيطة على الخانات مثل فحص قيمة خانة أو إعطاء قيمة لخانة أو عكس قيمة لخانة (من 0 إلى 1 أو من 1 إلى 0).
ليس بإمكان المبرمجين التعامل مع المعطيات على مستول الخانات الذي يعتبر أدنى مستوى التعامل مع المعطيات . وإنما يفضلون التعامل مع المعطيات من خال أشكال لها مثل الأعداد العشرية (0-9) والحروف (z-a أو Z-A) والرموز الخاصة (!,@,#,$,%,%,....) وجميعها يعرف باسم Characters. وعلى اعتبار أن الحاسب يتعامل مع الصفر والواحد فقط، فأنه يتم تحويل كل حرف من حروف مجموعة الحروف إلى نموذج من الأصفار والواحدات (تسمى بالبايت Byte). يبلغ طول البايت الواحد ثمان خانات . يقوم المبرمجون بإنشاء برامج وعناصر معطيات باستخدام الحروف ويتعامل الحاسب معها على إنها نماذج من الخانات.

14.6.1 أنواع الملفات حسب طريقة الوصول

تنقسم الملفات حسب طريقة الوصول للبيانات فيها إلى:

- ملفات الوصول ألتتابعي Sequential Access Files

حيث يتم الوصول إلى البيانات من بداية الملف حتى نصل إلى السجل المطلوب أو نهايته ، وكذلك عند الكتابة على الملف.

- ملفات الوصول العشوائي Random Access Files

حيث يمكننا الوصول إلى السجل المطلوب مباشرة دون الحاجة للمرور على السجلات التي قبلة.

كما يمكن تقسيم الملفات حسب نوع البيانات إلى:

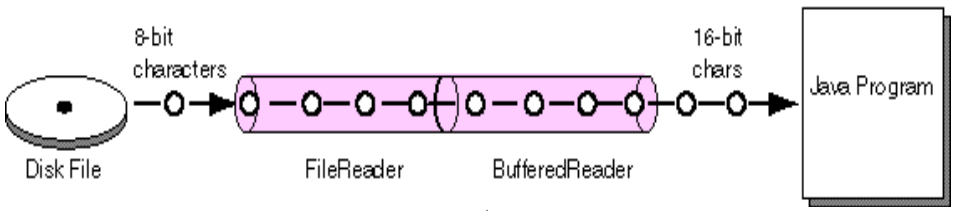
1- ملفات نصية Text Fils : حيث تتم القراءة والكتابة من الملف على شكل محارف Characters.
2- ملفات ثنائية Binary Files : حيث يتم التعامل مع البيانات على هيئة بايتات Bytes.

14.6.2 عمليات الملفات

من العمليات التي تجرى على الملفات عملية القراءة والكتابة فأفضل الطرق التي نفضلها وهي:

1. القراءة من الملف

```
1-  BufferedReader br = new BufferedReader(new FileReader("c:\\ammar.txt"));
```

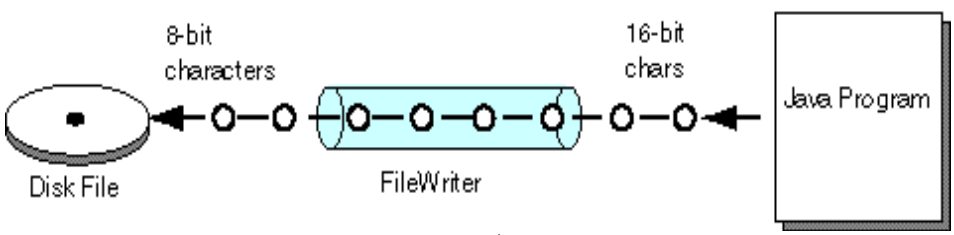


شكل 14-5

2- FileReader br=new FileReader("c:\\ammar.txt"));
3- FileInputStream br=new FileInputStream("c:\\ammar.txt"));

2. الكتابة على الملف

1- PrintWriter br=new PrintWriter(new FileWrite("c:\\ammar.txt"));
Br.println("ammar");
2- FileOutputStream br=new FileOutputStream("c:\\ammar.txt"));
Br.write("ammar");
3- FileWriter br= new FileWriter("c:\\ammar.txt"));
Br.write("ammar");



شكل 14-6

وكما بينا سابقا أن عملية القراءة تتمثل في كيفية كتابة تعريف القراءة كالقراءة على بايت أو على سلسلة محارف .

Int c=(char)br.read(); ← **على حرف حرف**
String c=br.readLine(); ← **على سطر سطر**

وهذا المثال يبين كيفية خزن عشره أعداد عشوائية في ملف

```
1. // insert ten number ranmodm
2. import java.io.*;
3. class Chp14_4 {
```

```
4.      public static void main(String[] args){
5.              java.util.Random a=new java.util.Random();
6.              try{// create File
7.              PrintWriter     out=new     PrintWriter(new     FileWriter("c:\\
aldopaee.txt",false));
8.              for (int i=0;i<10;i++)
9.                      out.println(a.nextInt(100));
10.             out.close();//close file
11.             }catch(IOException e){System.out.println("File Not Found ");}
12.             System.out.println("completed insert number");
13.                }
14. }
```

شرح المثال:

الآن بعد تنفيذ الكود اذهب إلى القرص C وستجد ملف باسم aldopaee افتح الملف وستجد العشرة الأعداد التي ولدتها بواسطة الدالة random موجودة داخل الملف .
السطر (7) قمنا بتعريف اسم الكائن out وهو من نوع PrintWrite يستخدم للكتابة على الملف . لاحظ أن استخدام نا للملفات قد يتسبب في حدوث بعض الاستثناءات مثل ملف غير موجود أو خطاء في إدخال البيانات وغيرة لذلك فيجب علينا تحديد أن المنهج main قد يطلق استثناء .
بعد من الكتابة على الملف وجب علينا إغلاق الملف كما في السطر 10.

في السطر 7 لو استبدلنا بدل الجملة false الجملة true فإننا سنضيف فوق الملف أي عملية إضافة Update.

عند عملية قراءة من ملف أو إضافة فيجب استخدام الاستثناءات .
هذا المثال يقرءاه محتويات الملف السابق بصيغه محارف

```
1. //Display aldopaee text file.
2. import java.io.*;
3. public class Chp14_5 {
4.     public static void main(String[] args)throws IOException {
5.         BufferedReader fi;
6.         String s;
7.                 try {
8.                         fi = new BufferedReader(new FileReader ("c:\\ aldopaee.txt"));
9.                         // read characters until EOF is encountered
10.                        while ((s=fi.readLine()) != null)
11.                        System.out.println(s);
12.                        fi.close();//close file
13.                       }
```

```
14.                     catch (Exception e) { System.err.println("File Not Found ");}
15.                      }
16. }
```

```
40
90
3
37
1
60
89
71
10
74
```

شرح المثال :

نلاحظ انه تم قراءه محتويات الملف بصيغة سلسلة وان أردت إجراء بعض العمليات الحسابية كمجموع أو اكبر قيمة أو ما شابة ذلك فيجب تحويل من صيغة سلسلة رقمية إلى أرقام بواسطة الدالة Integer.parsInt() وقد تم شرحها سابقاً .

وهذا كود لقراءة الملف السابق على حرف حرف ومعرفة عدد السطور الملف:

```
1. // يقرة الملف على حرف حرف ويعد عدد الأسطر بداخل الملف
2. import java.io.*;
3. class Chp14_6 {
4.         public static void main(String args[])throws IOException{
5.                 int i,g=0;String s;
6.                 FileInputStream fin;
7.                 try {
8.                  fin = new FileInputStream("c:\\ aldopaee.txt");
9.                 } catch(FileNotFoundException e){System.out.println("File Not Found");
10.                return;
11.
        }
12.
13.                do{
14.                i = fin.read();
15.                if(i!=-1)System.out.print((char) i);
16.                if(i==10)g++;
17.                 }while(i != -1);
18.                System.out.println("The Number Of line  : "+g);
19.                fin.close();
20.                   }
21. }
```

```
40
90
3
37
1
60
89
71
10
74
The Number Of line  : 10
```

شرح المثال:
السطر 17 يستفسر هل وصلنا إلى نهاية الملف أم لا ليستمر.
السطر 16 إذا كان I= شفرة enter فإننا انتقلنا إلى سطر جديد.

في مثالنا السابق بإمكاننا قراءة الملف على سطر سطر باستبدال الصيغة التالية

```
while((s = fin.readLine())!=null)
```

وهذا المثال يفتح ملف وينسخه إلى ملف أخر بدون مسح محتويات الملف الثاني أي إضافة:

```
1. /* Copy a text file.
2. To use this program, specify the name
3. of the source file and the destination file.
4. For example, to copy a file called FIRST.TXT
5. to a file called SECOND.TXT, use the following
6. command line.
7. java CopyFile FIRST.TXT SECOND.TXT
8. */
9. import java.io.*;
10. class Chp14_7 {
11.       public static void main(String args[])throws IOException{
12.                 int g=0;
13.                 FileInputStream fin;
14.                 // open output file
15.                 FileOutputStream fin2=new FileOutputStream("c:\\ out.txt",true);
16.                 try {// open input file
17.                  fin = new FileInputStream("c:\\ aldopaee.txt");
18.           }catch(FileNotFoundException e)
                                                {System.out.println("File Not Found");return;}
19.                 // Copy File
20.                 while(g != -1){
21.                                  g = fin.read();
22.                                  if(g!=-1)fin2.write((char)g);
23.                                 }
24.                 fin.close();fin2.close();
25.                 System.out.println("Copy File Good");
26.                   }
27. }
```

```
Copy File Good
```

شرح المثال:

يقرأ البرنامج بشكل مستمر البيانات من المجرى fin.read ويرسلها إلى المجرى fin2.Write إلى أن تصل قيمة المتحول -1 فنكون قد انتهينا من قراءة الملف ووصلنا إلى نهاية الملف كما في الأسطر (20 - 23).


يؤدي عدم إغلاق الملفات إلى بعض الأخطاء البرمجية.


عادة يتم إغلاق الملفات ضمن عبارة finally.

14.6.3 الملفات الثنائية	Binary Files

لا تستطيع الملفات النصية التعامل مع حالات أخرى أكثر تعقيداً من مجرد نصوص ، فحينما تقوم بإنشاء أنظمة أو برامج أكثر تعقيداً فإننا نحتاج للتعامل معها على صورتها الحقيقية وليس على أنها جميعها متغيرات محرفية ، وهذا ما تقوم به الملفات الثنائية.

ف هى عبارة عن سلسلة من البتات (0-1) صفر و واحد ويتم التعرف عليها وفهمها بواسطة التطبيقات التي أنشأتها . ونلاحظ ذلك في عدم القدرة على قراءة ملف ما إلا بواسطة التطبيق الذي صنع لأجله فمثلا لو أرت فتح مستند مكتوب بواسطة برنامج معالج كلمات ما ولنفرض Word Perfect على برنامج Microsoft Word فان الملف لم يفتح أو أنة يفتح بشكل غير مناسب وذلك لأنة لم يعد للعمل على هذا التطبيق بالاختلاف انه صمم لكي يناسب العمل على تطبيق أخر . ولحسن الحظ فان معظم تطبيقات معالجة الكلمات اليوم تحتوى على محولات تستطيع فتح وقراءة الملفات التي تم إنشاءه على معالجات أخرى .

- **ميزات الملفات الثنائية**

أن الميزة الجيدة للملفات الثنائية هي سهولة فهم الشفرات الثنائية من قبل الحاسب ، بما أن بنية الحاسب هي أصلا بنية ثنائية تعتمد على الوا حدات والأصفار فان قراءة الملفات الثنائية ستكون أسرع من قراءة هيئات الملفات الأخرى .

- **عيوب الملفات الثنائية**

السيئة الوحيدة هي عدم القدرة على فتح تطبيق تم بناءة على تطبيق أخر . وقد يصل الأمر إلى عدم إمكانية فتح وقراءة الملف في نفس التطبيق ولكن ضمن منصة تشغيل Platform مختلفة أو ضمن إصدارة سابقة لنفس التطبيق.

ونظراً لأن الملفات الثنائية تختلف عن ا لملفات النصية فقد تجد دوالاً أخرى هنا تختلف عن الملفات النصية مع عدم الاختلاف في الأساسيات.
ونعرض الآن مثالين الأول يكتب على ملف ثنائي والمثال الثاني يقرءا من ملف ثنائي.

a. الكتابة على الملف

```
1. //الكتابة على الملف الثنائي
2. import java.io.*;
3. class Chp14_8
4. {
5.
6.  public static void main ( String[] args )
7.  {
8.    String fileName = "intData.dat" ;
```

```
9.
10.    int value0   =   0, value1  = 1,
11.        value255 = 255, valueM1 = -1;
12.
13.    try
14.    {
15.      DataOutputStream out = new DataOutputStream(
16.          new FileOutputStream( fileName  ) );
17.
18.      out.writeInt( value0 );
19.      out.writeInt( value1 );
20.      out.writeInt( value255 );
21.      out.writeInt( valueM1 );
22.      out.close();
23.    }
24.    catch ( IOException iox )
25.    {
26.      System.out.println("Problem writing " + fileName );
27.    }
28.  }
29. }
```

b. القراءة من الملف

```
1. // لبقراءة من الملف الثنائي
2. import java.io.*;
3. class Chp14_9
4. {
5.  public static void main ( String[] args )
6.  {
7.    String fileName = "intData.dat" ;  long sum = 0;
8.
9.    try
10.    {
11.      DataInputStream instr =
12.        new DataInputStream(
13.          new BufferedInputStream(
14.            new FileInputStream( fileName  ) ) );
15.
16.      sum += instr.readInt();
17.      sum += instr.readInt();
18.      sum += instr.readInt();
19.      sum += instr.readInt();
20.
21.      System.out.println( "The sum is: " + sum );
22.      instr.close();
23.    }
24.    catch ( IOException iox )
25.    {
```

```
26.     System.out.println("Problem reading " + fileName );
27.   }
28.  }
29. }
```

14.6.4 إنشاء نسخة مماثلة لمعطيات مجرى أثناء كتابتها:

قد تحتاج لشيء ما تتم كتابته من خلال مجرى وليكن مجرى الدخل المعياري System.out، أو مجرى الخطأ المعياري System.err. بحيث تستطيع تسجيله في ملف مع إبقاء إمكانية ظهوره في المكان المخصص له (بالنسبة لمجرى الخرج المعياري والخطأ سينتقل محتواهما إلى أداة الخرج القياسية)، أي الحصول على نسخة عن بيانات المجرى دون إعادة توجيهه.

لحل هو اشتقاق الصنف Chp14_10 في المثال التالي من الصنف الأب PrintStream بحيث نضيف له بعض الوظائف الإضافية. ومن ثم سنستخدمه كما يلي:

System.setErr(new TeePrintStream (System.err, "err.Log));

حـ يث من المعلوم أن المنهج System.SetErr يقوم بتحديد وجهة إخراج النص المحدد بـالمجرى System.err.سيقوم السطر السابق من الشيفرة بتوجيه خرج مجرى الخطأ المعياري إلى الملف err. Log مع الإبقاء على الوجهة التي كان الخرج مجرى الخطأ موجهاً إليه مسبقاً.

من المهم الذكر بأن هذه التقنية صالحة لأن تستخدم مع أي منهج يستخدم PrintStream كما يمكن استخدامها من أجل كل من الأصناف BufferedInputStream وPrintWriter وBufferedReader وغيرها من الأصناف.
و يعرض المثال التالي طريقة اشتقاق الصنف Chp14_10:

```
1. import java.io.*;
2.
3. public class Chp14_10 extends PrintStream {
4.         protected PrintStream parent;
5.         protected String fileName;
6.
7.         /** A simple test case. */
8.         public static void main(String[] args) throws IOException {
9.                 Chp14_10 ts = new Chp14_10(System.err, "err.log");
10.                 System.setErr(ts);
11.                 System.err.println("An imitation error message");
12.                 ts.close();
13.          }
14.
15.          /** Construct a Chp14_10 given an existing PrintStream,
```

```
16.         * an opened OutputStream, and a boolean to control auto-flush.
17.         * This is the main constructor, to which others delegate via "this".
18.         */
19.        public Chp14_10(PrintStream orig, OutputStream os, boolean flush)
20.        throws IOException {
21.                super(os, true);
22.                fileName = "(c://random.dat)";
23.                parent = orig;
24.        }
25.
26.        /** Construct a Chp14_10 given an existing PrintStream and
27.         * an opened OutputStream.
28.         */
29.        public Chp14_10(PrintStream orig, OutputStream os)
30.        throws IOException {
31.                this(orig, os, true);
32.        }
33.
34.        /* Construct a Chp14_10 given an existing Stream and a filename.
35.         */
36.        public Chp14_10(PrintStream os, String fn) throws IOException {
37.                this(os, fn, true);
38.        }
39.
40.        /* Construct a Chp14_10 given an existing Stream, a filename,
41.         * and a boolean to control the flush operation.
42.         */
43.        public Chp14_10(PrintStream orig, String fn, boolean flush)
44.        throws IOException {
45.                this(new FileOutputStream(fn), flush);
46.        }
47.
48.        /** Return true if either stream has an error. */
49.        public boolean checkError() {
50.                return parent.checkError() || super.checkError();
51.        }
52.
53.        /** override write(). This is the actual "tee" operation. */
54.        public void write(int x) {
55.                parent.write(x); // "write once;
56.                super.write(x);          // write somewhere else."
57.        }
58.
59.        /** override write(). This is the actual "tee" operation. */
60.        public void write(byte[] x, int o, int l) {
61.                parent.write(x, o, l);    // "write once;
```

```
62.             super.write(x, o, l);     // write somewhere else."
63.         }
64.
65.         /** Close both streams. */
66.         public void close() {
67.             parent.close();
68.             super.close();
69.         }
70.
71.         /** Flush both streams. */
72.         public void flush() {
73.             parent.flush();
74.             super.flush();
75.         }
76. }
```

14.6.5 الصنف من نوع StreamTokenizer

يقدم الصنف StreamTokenizer إمكانيات أوسع لعملية مسح ملف ما، حيث يمكنه قراءة المحارف و تجميعها في كلمات أو تشكيلات (tokens) و من ثم إعادتها مع تحديد نوع هذه التشكيلات، سيكون نوع التشكيلة أحد الأنماط التالية:

(Tokenizer.TTWORD, TT_NUMBER, TT_WORD, TT_EOL)

النمط TT_EOL يدل على نهاية السطر، أو يمكن أن يعيدها على شكل محارف ASCII.

كما يحتو ي هذا الصنف على مجموعة من المناهج ا لمساعدة، فالمنهج(ordinaryCharacter يسمح بتحديد الشكل الذي سيتم تجميع المحارف و فقه،حيث يستخدم لإهمال أية أحرف وعدم تضمينها في التشكلية التي تقوم بإنشائها. بينما يسمح المنهج SlashslashComment() بتفعيل أو تعطيل تلك المزايا.

يوضح المثال التالي كيفية استخدام StreamTokenizer في تطبيق بسيط لآلة حاسبة حيث ستعمل و فق الأسلوب:

2 2 + =
4
22 7 / =
3.141592857

```
1. /**
2. * StringTokenizer إنشاء آلة حاسبة بسيطة باستخدام الصنف
3. */
4. import java.io.*;
```

```
5. import java.net.*;
6. import java.util.*;
7. public class Chp14_11 {
8.         /** The StreamTokenizer */
9.         protected  StreamTokenizer tf;
10.          protected String variable;
11.          /** التصريح عن المكدس*/
12.          protected Stack s;
13.
14.          /* Driver - main program */
15.          public static void main(String[] av) throws IOException {
16.                 if (av.length == 0)
17.                    new Chp14_11(new InputStreamReader(System.in)).doCalc();
18.          }
19.
20.          /** باستخدام قارىء موجود Chp14_11 بناء كائن من الصنف  */
21.          public Chp14_11(Reader rdr) throws IOException {
22.                 tf = new StreamTokenizer(rdr);
23.                 //معالجة مجموعة محارف الإدخال
24.                 tf.slashSlashComments(true);       // treat "//" as comments
25.                 tf.ordinaryChar('-');                  // used for subtraction
26.                 tf.ordinaryChar('/');                 // used for division
27.
28.                 s = new Stack();
29.          }
30.
31.          protected void doCalc() throws IOException {
32.                 int iType;
33.                 double tmp;
34.
35.                 while ((iType = tf.nextToken()) != tf.TT_EOF) {
36.                    switch(iType) {
37.                    case StreamTokenizer.TT_NUMBER: // سحب الرقم وإدخاله إلى المكدس
38.                           push(tf.nval);
39.                           break;
40.                       case StreamTokenizer.TT_WORD:
41.                           // Found a variable, save its name. Not used here. */
42.                           variable = tf.sval;
43.                           break;
44.                       case '+':
45.                           //التعرف على معامل الجمع وتنفيذ العملية مباشرة
46.                           push(pop() + pop());
47.                           break;
48.                       case '-':
49.                           //التعرف على معامل الطرح وتنفيذ العملية حسب ترتيب الورود
50.                           tmp = pop();
```

```
51.                         push(pop() - tmp);
52.                         break;
53.                 case '*':
54.                         // التعرف على معامل الضرب وتنفيذ العملية مباشرة
55.                         push(pop() * pop());
56.                         break;
57.                 case '/':
58.                         // التعرف على معامل القسمة وتنفيذ العملية مباشرة
59.                         tmp = pop();
60.                         push(pop() / tmp);
61.                         break;
62.                 case '=':
63.                         System.out.println(peek());
64.                         break;
65.                 default:
66.                         System.out.println("What's this? iType = " + iType);
67.                 }
68.             }
69.         }
70.         void push(double val) {
71.             s.push(new Double(val));
72.         }
73.         double pop() {
74.             return ((Double)s.pop()).doubleValue();
75.         }
76.         double peek() {
77.             return ((Double)s.peek()).doubleValue();
78.         }
79.         void clearStack() {
80.             s.removeAllElements();
81.         }
82. }
```

بشكل عام فإن استخدام StreamTokenizer مفيد لحد ما، لكنه محدود بعدد التشكيلات المختلفة التي يستطيع التعرف عليها كما أنه لا يستطيع تحديد فيما إذا كانت تشكيلة ما (token) يجب أن تظهر بترتيب محدد.

لذلك يمكن استخدام بعض الأدوات الأخرى مثل الأداة JavaCC و المستخدمة بشكل واسع (تطور هذه الأداة من قبل شركة Web Gain) لكتابة قو اعد عامة لمختلف أنواع البرامج،

ابتداءً بتطبيقات الآلات الحاسبة كالتطبيق الذي تم استعراضه فيما سبق مروراً بـتطبيقات CORBA/IDL و HTML و صولاً إلى المترجمات المختلفة (حيث يمكنك بهذه الأداة كتابة مترجم للغة Java أو لغة C++ أو أي لغة أخرى قد تصنعها بنفسك).

و خلاصة القول يمكن استخدام StringTokenizer للقيام بمسح سطري (Line-at-a time) ، في حين يستخدم الصنف StreamTokenizer للقيام بمسح أكثر دقة باستخدام التشكيلات (token) كما يمكن استخدام الأداة JavaCC لإجراء مسح قو اعدي (grammar-based).

14.6.6 الملفات العشوائية RandomAccessFile

هي مجموعة من السجلات المرتبطة منطقياً، وكل سجل داخل الملف يعنون بدليل أو مفتاح يستخدم للتميز بين السجلات المختلفة بالملف، بمعني أن هذا الدليل يستخدم كمعرف للسجل.

ويمكن تعريف السجل داخل الملف على أنه مجموعة من مفردات البيانات Data Items يسمى كل منها حقل، و أحد هذه الحقول يستخدم كدليل أو مفتاح للسجل Record Key أما الحقل فهو عبارة عن مجموعة من الرموز قد تكون هجائية أو رقمية أو رقمية هجائية أو علامات خاصة، تنتظم معاً لتعطي معني منطقياً.

وكمثال على الملفات: ملف الطلبة Student File فهو تجميع لسجلات الطلبة Records، كل سجل منها مخصص لطالب، ويحوي مفردات البيانات Data Items للطالب كرقم الطالب، اسمه، عنوانه، جنسه، تاريخ ولادته، وكل واحد من هذه المفردات يسمى حقلاً وكل حقل مكون من مجموعة من الحروف.

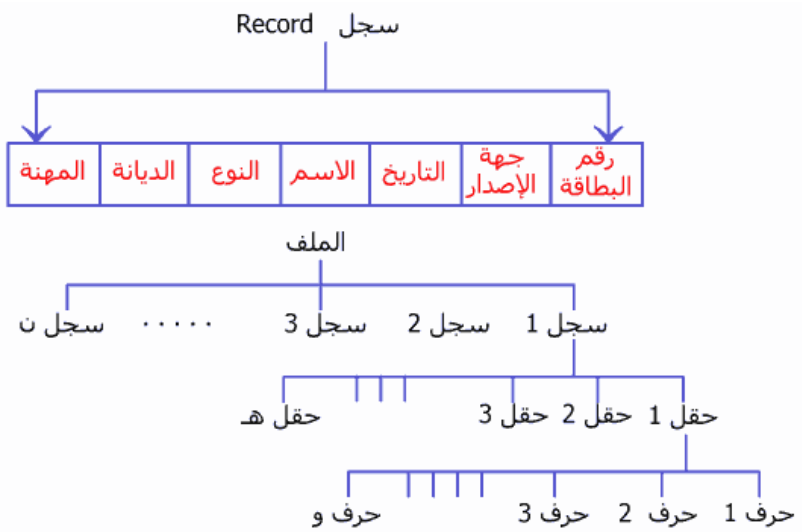
ويمكن تلخيص ما سبق:

الحرف: هو أصغر مكون منطقي في الملف، وقد يكون أبجدي أو رقمي

الحقل: وهو أصغر وحدة بيانات ويتشكل من مجموعة متناغمة من الحروف.

السجل: مجموعة مترابطة منطقياً مع حقول توصف كيان بذات توصيفاً يتلاءم مع طبيعة ومتطلبات الملف، وقد تسمى الحقول بالموصفات للكيان Attributes .

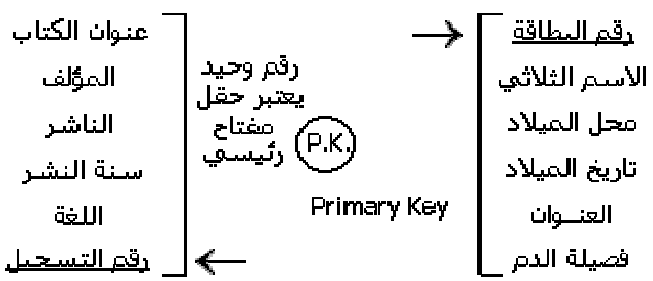
الكيان Entity : هو شيء محدد مثل الإنسان- السلعة- التاجر- الموظف- أو نظام المعلومات، أو هو شيء نهتم بتسجيل بيانات عنه.



شكل يوضح مكونات الملف العشوائي

- **أنواع حقول السجل Record Field Kinds :**

إذا تدارسنا بيانات البطاقة الشخصية أو بيانات تسجيل كتاب بأحد المكتبات لوجدنا أن موصفات صاحب البطاقة أو موصفات الكتاب كما هي موضحة في الشكل ـــ.



شكل يوضح بيانات البطاقة الشخصية

وسوف نلاحظ أن هناك نوعين من الحقول حقل وحيد له قيمة لا تتكرر على مستوى ا لسجل وعلى مستوى الملف كله مثل البطاقة / الرقم القومي / رقم تسجيل الكتاب ، وكلاهما رقم فريد Unique ولذلك إذا سألت عن بيانات صاحب البطاقة رقم (كذا) أو ما هو عنوان واسم مؤلف الكتاب الذي رقم تسجيله

(كذا) فما أيسر الوصول إلى البيانات المطلوبة لأن هذا النوع من ا لحقول يشير مباشرة إلى باقي حقول السجل ويحدد الكيان المعبر عنه تحديدًا دقيقًا.
مثل هذه الحقول الخاصة تسمى حقل مفتاح رئيسي Primary Key ولا ينفي وجود هذا النوع من الحقول وجود حقول أخرى تصلح للإشارة للسجل ويطلق عليها المفاتيح الثانوية Secondary Key ، كما يمك ن الدمج بين أكثر من حقل غير مفتاحي لتكوين حقل مفتاحي Unique ، أما باقي الحقول فهي حقول يحتمل تكرارها في أي عدد من السجلات في الملف الواحد ، ففي ملف البطاقات الشخصية قد يتواجد عدد من السجلات لها نفس أسماء أصحاب البطاقات ، وربما نفس محل الميلاد ويحتمل نفس تا ريخ الميلاد ولا تنفي احتمال وجود ولو عدد محدود جدًا من الأشخاص لهم نفس العنوان وهكذا.

الرقم القومي الاسم محل الميلاد تاريخ الميلاد العنوان فصيلة الدم

والحقل الفريد أو المفتاح الرئيسي هو أحد الأدوات المنطقية في استرجاع البيانات من الملفات .

- **الصنف من نوع RandomAccessFile**

قد تحتاج إلى القراءة أو الكتابة إلى موضع محدد في ملف ما، كما في الملفات المفهرسة. لغة Java أعطتك الحل بـ استخدام الصنف RandomAccessFile يمكنك نقل موضع (القراءة/الكتابة) ضمن ملف معين، إذ يمكنك الانتقال إلى آخر الملف، مما يسهل التعامل مع الملفات المفهرسة، و حتى مع ملفات قواعد البيانات الأخرى.

في حين يقوم المنهج void skipBytes (int howmany) بنقل موضع (القراءة/الكتابة) الحالي إلى الموضع المحدد بالمتحول howmany (ستتم عملية النقل إلى الأمام)، أما المنهج long getFilePointer() فيعيد مكان المؤشر الحالي.

المثال التالي يـ وضح كيفية استخدام الصنف RandomAccessFile ، والتنقل ضمن الملف Random.dat، ومن ثم طباعة سطر الموضع الحالي.

```
1. import java.io.*;
2. /**
3.  * Read a file containing an offset, and a String at that offset.
4.  */
5. public class Chp14_12 {
6.                     // يتم تحديد مسار ملف مؤقت
7.         final static String FILENAME = "random.dat";// "C://random.dat"
8.         protected String fileName;
```

```
9.          protected RandomAccessFile seeker;
10.
11.         public static void main(String[] argv) throws IOException {
12.               Chp14_12 r = new Chp14_12(FILENAME);
13.
14.               System.out.println("Offset is " + r.readOffset());
15.               System.out.println("Message is \"" + r.readMessage() +
"\".");
16.         }
17.
18.         /** Constructor: save filename, construct RandomAccessFile */
19.         public Chp14_12(String fname) throws IOException {
20.               fileName = fname;
21.               seeker = new RandomAccessFile(fname, "r");
22.         }
23.
24.         /** Read the Offset field, defined to be at location 0 in the file. */
25.         public int readOffset() throws IOException {
26.               seeker.seek(0);                        // move to very beginning
27.               return seeker.readInt();  // and read the offset
28.         }
29.
30.         /** Read the message at the given offset */
31.         public String readMessage() throws IOException {
32.               seeker.seek(readOffset());         // move to where
33.               return seeker.readLine(); // and read the String
34.         }
35. }
```

14.6.7 تخزين واستعادة الكائنات بشكل تسلسلي

دعنا نعرف عملية التحويل التسلسلي (Serialization) أولاً:

هي عبارة عن تقنية ممتازة تسمح لك بأخذ أي كائن ينفذ الواجهة Serializable وتقوم بتحويله إلى سلسلة بايتات يمكن إرجاعها فيما بعد إلى الكائن الأصلي. تستخدم هذه التقنية بكثرة عبر الشبكة بحيث تقوم وبشكل تلقائي بإجراء عملية التوافق بين مختلف أنظمة التشغيل. هذا يعني إن باستطاعتك إنشاء كائن على حاسب Windows ثم تحويله تسلسلياً وإرساله ع بر الشبكة إلى حاسب Unix حيث تتم إعادة بناءه بشكل صحيح، لذلك لم يعد هنالك أي داع للقلق بشأن أشكال تمثيل المعطيات على الأجهزة المختلفة.

لقد تم استخدام تقنية التحويل التسلسلي للكائنات Object Serialization لدعم تقنتين رئيسيتين:

الأولى تقنية RMI(Remote Method Invocation) والتي تسمح للكائنات التي تتواجد على حواسب أخرى بالتصرف وكأنها موجودة على حاسبك. أما التقنية الثانية، فهي java Beans. فعندما يتم استخدام beans، فإن معلومات الحالة له يتم توصيفها في وقت التصميم، ويجب أن يتم تخزين معلومات الحالة هذه واستردادها فيما بعد عند تشغيل البرنامج، حيث تقوم تقنية التحويل التسلسلي بإنجاز هذه المهمة.

تعتبر عملية تحويل الكائن بشكل تسلسلي بسيطة، حيث تم تغيير العديد من أصناف مكتبة Java لتصبح قابلة للتحويل التسلسلي. فمن أجل تحويل كائن بشكل تسلسلي، يجب إنشاء كائن من الصنف OutputStream وتغليفه بغرض من الصنف ObjectOutPutStream. تحتاج عند هذه النقطة إلى استدعاء المنهج Writeobject() فقط، يتم بعدها تحويل العنصر تسلسلياً وإرساله إلى مجرى الخرج OutputStream.

في البرنامج التالي سنقوم بتمرير الكائن MyData(بعد أن نصنع الصنف الخاص به) إلى ا لمنهج Writeobject وذلك باستخدام كائن من الصنف ArrayList يحتوي على مجموعة من معطيات الكائنات أولها من النمط java.util.Dataالكائنات المتبقية من النمط MyData سيلحظ هذا المنهج بأن أحد حقول الكائن MyData هو كائن أيضاً من النمط String ، لذلك سيتم تحويل معطيات تلك الحقول بشكل تسلسلي أيضاً.

بالطبع لكي تستطيع استخدام عملية التحويل التسلسلي عليك التصريح عن الأصناف التي ستصنعها بحيث تنفذ الواجهة Serializable ، قد تلاحظ أيضاً استخدام الكلمة المحجوزة Transient لمنع إمكانية حدوث عملية التحويل التسلسلي عل كائن ما (وذلك لأغراض الأمان) حيث استخدمت في هذا البرنامج لمنع تخزين كلمات السر غير المشفرة.

```
1. package ioarticles;
2. import java.io.*;
3. import java.util.*;
4. /** Demonstrate use of Serialization. */
5.
6. public class Chp14_13 {
7.
```

```
8.      protected static final String FILENAME = "random.dat";
9.                              public static void main(String[] s) throws
IOException,ClassNotFoundException {
10.
11.             new Chp14_13().save();
12.             new Chp14_13().dump();
13.     }
14.
15.     /** The save method in an appliction */
16.     public void save() throws IOException {
17.             ArrayList v = new ArrayList();
18.             // Gather the data
19.             MyData u1 = new MyData("raynet", "java_article");
20.             v.add(new Date());
21.             v.add(u1);
22.             v.add(new MyData("Abby Brant", "dujordian"));
23.             write(v);
24.     }
25.     /** Does the actual serialization */
26.     public void write(Object theGraph) throws IOException {
27.
28.             // Save the data to disk.
29.             ObjectOutputStream os = new ObjectOutputStream(
30.                     new BufferedOutputStream(
31.                             new FileOutputStream(FILENAME)));
32.             os.writeObject(theGraph);
33.             os.close();
34.     }
35.     public void dump() throws IOException,ClassNotFoundException
36. {
37.             ObjectInputStream is = new ObjectInputStream(
38.                     new FileInputStream(FILENAME));
39.             System.out.println(is.readObject());
40.             is.close();
41.     }
42. }
43.
44. /** Simple data class to be serialized. */
45. class MyData implements Serializable {
46.     String userName;
47.     String passwordCypher;
48.     transient String passwordClear;
49.     public MyData(String name, String clear) {
50.             userName = name;
51.             // Save the clear text p/w in the object, it won't get serialized
52.             passwordClear = clear;
```

```
53.             // So we must save the encryption! Encryption not shown here.
54.             passwordCypher = DES.encrypt(passwordClear);
55.     }
56. }
57.
58. /** More of the demo; this just generates a String; Strings are serializable */
59. class DES {
60.     // Obviously just a placeholder.
61.     public static String encrypt(String s) {
62.             return s;
63.     }
64. }
```

14.6.8 تحويل النص إلى النموذج PostScript

البوست سكريبت (Post Script) هو الصيغة التي يتم تحويل الملفات التي يريد المستخدم طباعتها من صيغتها الأصلية إلى الصيغة التي تفهمها الطابعة في حالة الطابعة الليزر ، ويتم ذلك باستخدام برمجيات خاصة ، تسمى محركات الطابعة Printer Drivers .و تكون هذه المحركات مدرجة ضمن نظام التشغيل .

و يحتوي محرك الطابعة على محول بوست سكر يبت يقوم بتحويل الملف من صيغته الأصلية إلى البوست سكريبت . ويمكن اعتبار هذا المحول كفلتر ؛ إذا يقوم باستقبال الملفات في صيغتها القياسية المختلفة سواء كانت نصية أو رسومات أو غير ذلك ، ثم يحولها إلى ملفات البوست سـ كريبت ، و من الممكن برمجة محرك الطابعة بحيث تصلح تعليمات البوست سكريبت التي ينشئها للتشغيل على كافة الأجهزة . لكن هذا قد يؤدي إلى إهمال الوظائف والمزايا الخاصة بالطابعة المستخدمة .

و بإدراج محركات الطابعة تتاح لك العديد من المزايا ؛ حيث يمكن لكل البرامج ا لمثبتة على الكمبيوتر أن تستخدم نفس محرك الطابعة ، وعند تحديث المحرك تستفيد كل البرامج من ذلك ، مما يعني أنه مهما تطورت تقنية الطابعات فإن تلك البرامج تتمكن من خلال محرك الطابعة الحديث من استخدام تلك الطابعات دون الحاجة إلى إعادة كتابتها أو تحديثها.

هنالك العديد من الطرق للطباعة في لغة Java بحيث تلبي احتياجات المبرمج وذلك عند تطوير تطبيقات الواجهة الرسومية GUI. لذلك فإن من الضروري تحويل النص إلى نموذج آخر أكثر ملائمة.

سيعرض النص البرمجي التالي كيفية القراءة من ملف نصي ومن ثم إخراج محتواه وفق النموذج Postscript ، وبسبب طبيعة النموذج Postscript سيتم حذف

بعض المحارف، حيث يتم معالجة ذلك ضمن المنهج toPsString() والذي يستدعى من قبل المنهج doLine() كما تم كتابة بعض الشيفرة والتي تعمل على ملاحقة موضع المؤشر الحالي ضمن الصفحة الواحدة ولإظهار على الخرج عند انتهاء الصفحة.

يمكن إرسال خرج هذا البرنامج مباشرة على طابعة تدعم النموذج Postscript.

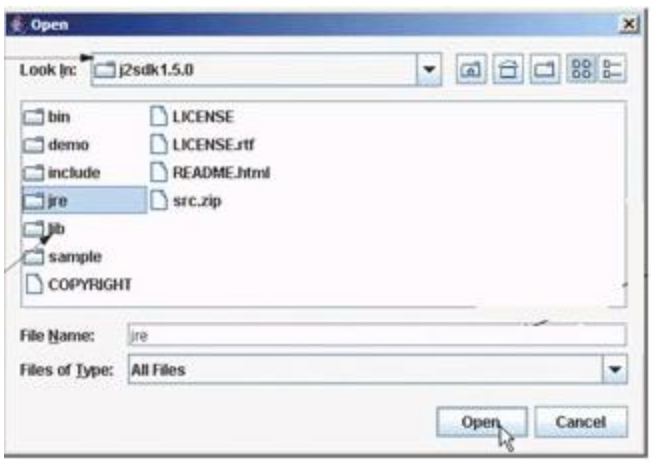
```
1. import java.io.*;
2. /** Text to PS */
3. public class Chp14_14 {
4.
5.       /** The current input source */
6.       protected BufferedReader br;
7.       /** The current page number */
8.       protected int pageNum;
9.       /** The current X and Y on the page */
10.        protected int curX, curY;
11.        /** The current line number on page */
12.        protected int lineNum;
13.        /** The current tab setting */
14.        protected int tabPos = 0;
15.
16.        public static final int INCH = 72; // PS constant: 72 pts/inch
17.        // Page parameters
18.        /** The left margin indent */
19.        protected int leftMargin = 50;
20.        /** The top of page indent */
21.        protected int topMargin = 750;
22.        /** The bottom of page indent */
23.        protected int botMargin = 50;
24.        // FORMATTING PARAMETERS
25.        protected int points = 12;
26.        protected int leading = 14;
27.
28.        public static void main(String[] av) throws IOException {
29.              if (av.length == 0)
30.                    new Chp14_14(
31.                          new InputStreamReader(System.in)).process();
32.              else for (int i = 0; i < av.length; i++) {
33.                    new Chp14_14(av[i]).process();
34.              }
35.        }
36.
37.        public Chp14_14(String fileName) throws IOException {
```

```
38.             br = new BufferedReader(new FileReader(fileName));
39.         }
40.
41.
42.         public Chp14_14(Reader in) throws IOException {
43.             if (in instanceof BufferedReader)
44.                 br = (BufferedReader)in;
45.             else
46.                 br = new BufferedReader(in);
47.         }
48.
49.         /** Main processing of the current input source. */
50.         protected void process() throws IOException {
51.             String line;
52.             prologue();             // emit PostScript prologue, once.
53.             startPage();        // emit top-of-page (ending previous)
54.
55.             while ((line = br.readLine()) != null) {
56.                 if (line.startsWith("\f") || line.trim().equals(".bp")) {
57.                     startPage();
58.                     continue;
59.                 }
60.                 doLine(line);
61.             }
62.
63.             // finish last page, if not already done.
64.             if (lineNum != 0)
65.                 println("showpage");
66.         }
67.
68.         /** Handle start of page details. */
69.         protected void startPage() {
70.             if (pageNum++ > 0)
71.                 println("showpage");
72.             lineNum = 0;
73.             moveTo(leftMargin, topMargin);
74.         }
75.
76.         /** Process one line from the current input */
77.         protected void doLine(String line) {
78.             tabPos = 0;
79.             // count leading (not imbedded) tabs.
80.             for (int i=0; i<line.length(); i++) {
81.                 if (line.charAt(i)=='\t')
82.                     tabPos++;
83.                 else
```

```
84.                     break;
85.             }
86.             String l = line.trim(); // removes spaces AND tabs
87.             if (l.length() == 0) {
88.                 ++lineNum;
89.                 return;
90.             }
91.             moveTo(leftMargin + (tabPos * INCH),
92.                 topMargin-(lineNum++ * leading));
93.             println('(' + toPSString(l)+ ") show");
94.
95.             // If we just hit the bottom, start a new page
96.             if (curY <= botMargin)
97.                 startPage();
98.         }
99.
100.        protected String toPSString(String o) {
101.            StringBuffer sb = new StringBuffer();
102.            for (int i=0; i<o.length(); i++) {
103.                char c = o.charAt(i);
104.                switch(c) {
105.                    case '(':   sb.append("\\("); break;
106.                    case ')':   sb.append("\\)"); break;
107.                    default:    sb.append(c); break;
108.                }
109.            }
110.            return sb.toString();
111.        }
112.
113.        protected void println(String s) {
114.            System.out.println(s);
115.        }
116.
117.        protected void moveTo(int x, int y) {
118.
119.            curX = x;
120.            curY = y;
121.            println(x + " " + y + " " + "moveto");
122.        }
123.
124.         void prologue() {
125.            println("%!PS-Adobe");
126.            println("/Courier findfont " + points + " scalefont setfont ");
127.        }
128. }
```

14.6.9 صناديق حوار الملفات

تقدم الحزمة swing الصنف JFileChooser والذي يعرض صندوق حوار يستطيع المستخدم من خلاله التنقل ضمن نظام ملفات القرص واختيار الملف الذي يرغب في تحميلها أو حفظها. كما يبين شكل 7-14 صندوق حوار من أجل فتح ملف.



شكل 7-14

توجد العديد من الطرق لعرض صناديق الحوار ، ولكن أبسطها هو استخدام الباني JFileChooser .

وهذا مثال شامل لما سبق ويعرض أيضاً عنصر التحكم JTextArea.

```
1. //  Demonstrate using JFileDialog to display
2. // file dialog boxes for opening and saving files
3. import java.awt.*;
4. import java.awt.event.*;
5. import java.io.*;
6. import javax.swing.*;
7.
8. public class Chp14_15 extends JFrame
9.   implements ActionListener {
10.   // Menu items Open, Save, exit, and About
11.   private JMenuItem jmiOpen, jmiSave,jmiExit, jmiAbout;
12.
13.   // Text area for displaying and editing text files
14.   private JTextArea jta = new JTextArea();
```

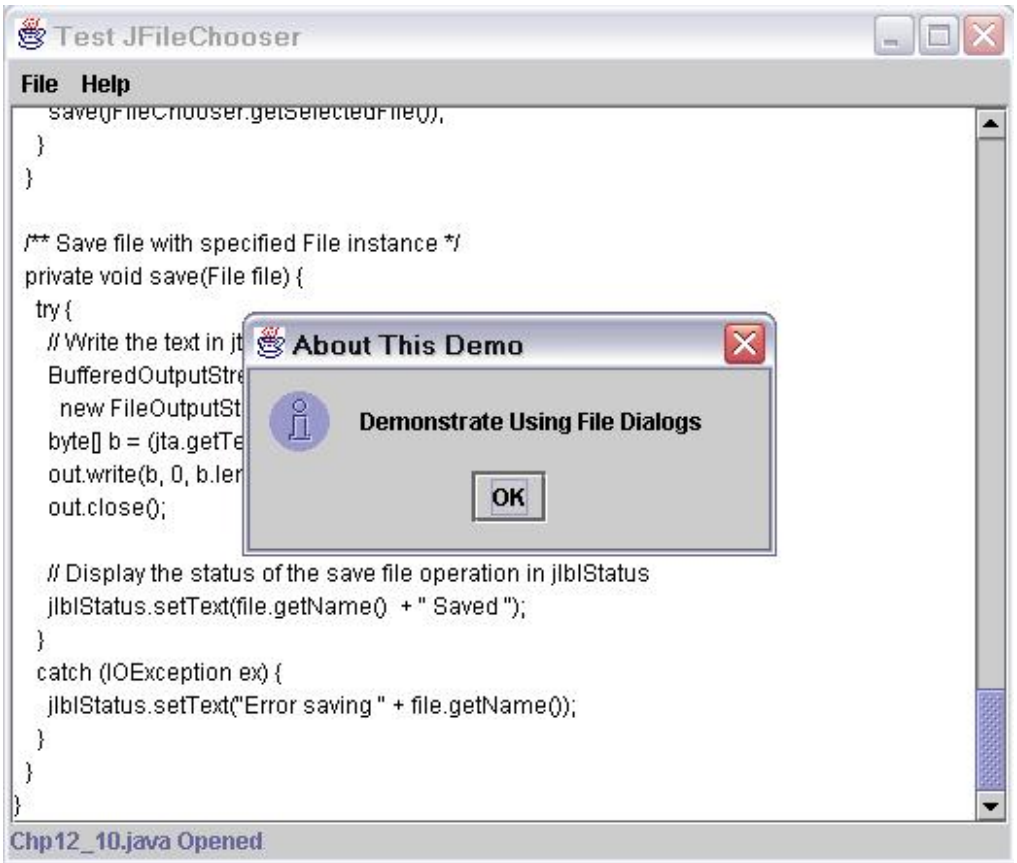
```
15.
16.  // Status label for displaying operation status
17.  private JLabel jlblStatus = new JLabel();
18.
19.  // File dialog box
20.  private JFileChooser jFileChooser = new JFileChooser();
21.
22.  /** Main method */
23.  public static void main(String[] args) {
24.    Chp14_15 frame = new Chp14_15();
25.    frame.setSize(300, 150);
26.    frame.setVisible(true);
27.  }
28.
29.  public Chp14_15() {
30.    setTitle("Test JFileChooser");
31.
32.    // Create a menu bar mb and attach to the frame
33.    JMenuBar mb = new JMenuBar();
34.    setJMenuBar(mb);
35.
36.    // Add a "File" menu in mb
37.    JMenu fileMenu = new JMenu("File");
38.    mb.add(fileMenu);
39.
40.    //add a "Help" menu in mb
41.    JMenu helpMenu = new JMenu("Help");
42.    mb.add(helpMenu);
43.    // Create and add menu items to the menu
44.    fileMenu.add(jmiOpen = new JMenuItem("Open"));
45.    fileMenu.add(jmiSave = new JMenuItem("Save"));
46.    fileMenu.addSeparator();
47.    fileMenu.add(jmiExit = new JMenuItem("Exit"));
48.    helpMenu.add(jmiAbout = new JMenuItem("About"));
49.
50.    // Set default directory to the current directory
51.    jFileChooser.setCurrentDirectory(new File("."));
52.
53.    // Set BorderLayout for the frame
54.    getContentPane().add(new JScrollPane(jta),
55.      BorderLayout.CENTER);
56.    getContentPane().add(jlblStatus, BorderLayout.SOUTH);
57.
58.    // Register listeners
59.    jmiOpen.addActionListener(this);
60.    jmiSave.addActionListener(this);
61.    jmiAbout.addActionListener(this);
62.    jmiExit.addActionListener(this);
63.  }
64.  /** Handle ActionEvent for menu items */
65.  public void actionPerformed(ActionEvent e) {
```

```
66.    String actionCommand = e.getActionCommand();
67.
68.    if (e.getSource() instanceof JMenuItem) {
69.      if ("Open".equals(actionCommand))
70.        open();
71.      else if ("Save".equals(actionCommand))
72.        save();
73.      else if ("About".equals(actionCommand))
74.        JOptionPane.showMessageDialog(this,
75.          "Demonstrate Using File Dialogs",
76.          "About This Demo",
77.          JOptionPane.INFORMATION_MESSAGE);
78.      else if ("Exit".equals(actionCommand))
79.        System.exit(0);
80.    }
81.  }
82.  /** Open file */
83.  private void open() {
84.    if (jFileChooser.showOpenDialog(this) ==
85.      JFileChooser.APPROVE_OPTION) {
86.      open(jFileChooser.getSelectedFile());
87.    }
88.  }
89.  /** Open file with the specified File instance */
90.  private void open(File file) {
91.    try {
92.      // Read from the specified file and store it in jta
93.      BufferedInputStream in = new BufferedInputStream(
94.        new FileInputStream(file));
95.      byte[] b = new byte[in.available()];
96.      in.read(b, 0, b.length);
97.      jta.append(new String(b, 0, b.length));
98.      in.close();
99.
100.       // Display the status of the Open file operation in jlblStatus
101.       jlblStatus.setText(file.getName() + " Opened");
102.     }
103.     catch (IOException ex) {
104.       jlblStatus.setText("Error opening " + file.getName());
105.     }
106.   }
107.   /** Save file */
108.   private void save() {
109.     if (jFileChooser.showSaveDialog(this) ==
110.       JFileChooser.APPROVE_OPTION) {
111.       save(jFileChooser.getSelectedFile());
112.     }
113.   }
114.   /** Save file with specified File instance */
115.   private void save(File file) {
116.     try {
```

```
117.      // Write the text in jta to the specified file
118.      BufferedOutputStream out = new BufferedOutputStream(
119.        new FileOutputStream(file));
120.      byte[] b = (jta.getText()).getBytes();
121.      out.write(b, 0, b.length);
122.      out.close();
123.
124.      // Display the status of the save file operation in jlblStatus
125.      jlblStatus.setText(file.getName()  + " Saved ");
126.    }
127.    catch (IOException ex) {
128.      jlblStatus.setText("Error saving " + file.getName());
129.    }
130.  }
131. }
```

شرح المثال:

يصنع البرنامج القائمتين File,Help ،تحتوي القائمة File على الأوامر التالية: Open من أجل التحميل Save من أجل الحفظ ، Exit من أجل إنهاء البرنامج. بينما تحتوي القائمة Help على الأمر About لعرض الرسالة التي حول البرنامج كما في الشكل 8-14:

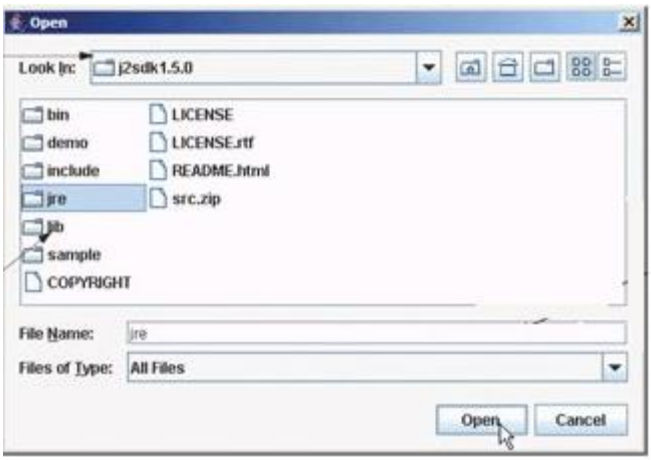


شكل 14-8

ويتم صنع الكائن jFileChooser (التابع للصنف JFileChooser) بهدف عرض صندوق حوار الملف من أجل الحصول على أمكانية فتح وحفظ الملفات كما في السطر 20.يتم استخدام المنهج setCurrentdirectory السطر 51 من أجل تحديد الدليل الحالي على أنة الدليل الذي سيتم فيه حفظ الملف.

يتم استدعاء المنهج Open السطر 70 عندما يختار المستخدم الأمر Open من القائمة.

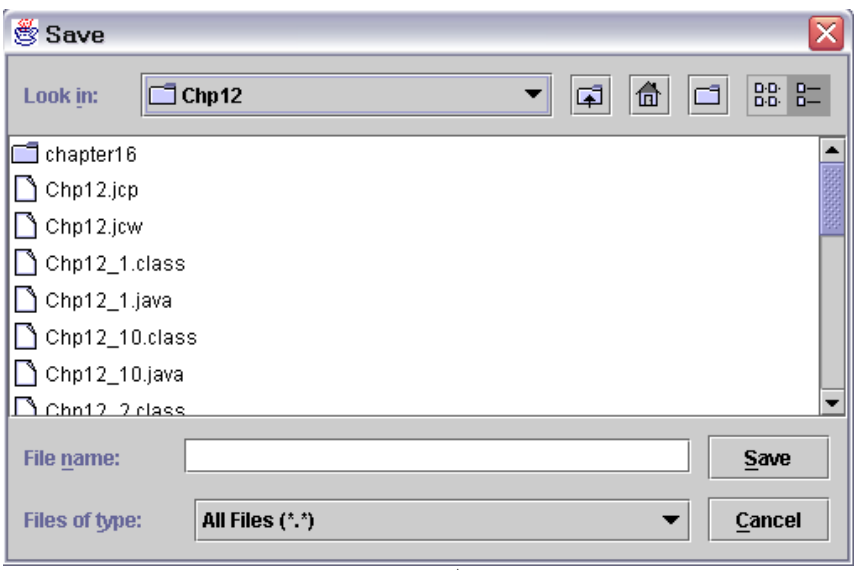
يعرض المنهج ShowOpenDialog السطر 84 صندوق الحوار Open كما في الشكل 14-9:



شكل 14-9

يتم استدعاء المنهج Open(file) عند استقبال الملف الذي تم أختيارة في السطر 84 وذلك من أجل تحميل الملف إلى العنصر JTexArea يستخدم لإنجاز هذه المهمة الصنف bufferedInputStream المغلف للصنف FileInputStream.

عندما ينقر المستخدم على الأمر Save يتم استدعاء المنهج ()Save السطر 72. يعرض المنهج ()ShowSavedialog السطر 109 صندوق حوار الحفظ كما في الشكل 14-10:



شكل 14-10

يتم استدعاء المنهج Save(file) السطر 111 عند استقبال الملف الذي تم أختيارة وذلك من أجل حفظ المحتويات من المساحة النصية (text area) إلى الملف. يستخدم لإنجاز هذه المهمة الصنف BufferedOutputStream المغلف بالصنف FileOutputStream.

14.7 مزايا نظام الملفات:

1. الملفات المتتالية تستهلك من حيز التخزين أدنى قدر متاح خصوصاً إذا كانت مكتلة ، لكن الاسترجاع المتتالي يبرز عدم مرونة هذا التنظيم.

2. تمتاز الملفات المباشرة بأقصى سرعة استرجاع لكنها غير مناسبة من حيث الاسترجاع المتتالي.

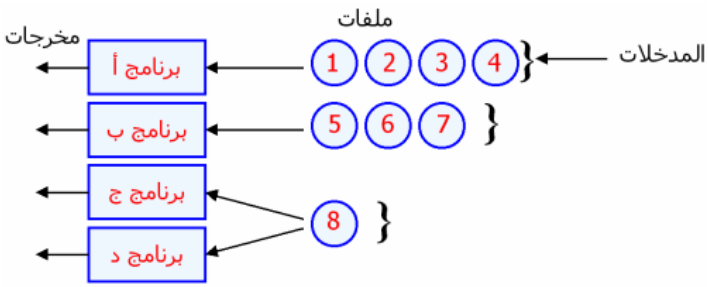
3. التنظيم المفهرس يتيح للمستخدم الاسترجاع المتتالي و المباشر رغم أن الأخير ليس بالسرعة المناسبة وهنا يتفوق التنظيم المتتالي المفهرس.

4. معظم نظم الملفات تلبي متطلبات النظم غير النشطة مثل أنظمة المرتبات والمخزون مما يستدعي استخدام تكلفة إضافية.

5. من أبسط الأساليب في إدارة البيانات رغم العيوب الكثيرة التي تعاني منها.

14.8 عيوب نظم الملفات:

حتى السبعينات ونظراً للقصور الكبير في معدات الحاسبات صممت معظم التطبيقات على مبدأ الارتباط المباشر بين البرنامج والبيانات الخاصة بها فيما عرف باسم Data Program Dependence ، فعندما كان يكتب برنامج بلغة Cobol فإنه يجري إنشاء ملف يضم البيانات اللازمة لهذا البرنامج ، وعندما يكتب برنامج بلغة Java ينشأ معه ملف آخر تطابق صياغة بياناته برنامج Java وهكذا.



شكل 14-11

حقيقة أن كل برنامج يمكنه التعامل مع أكثر من ملف لكنه يتعامل معها تتابعياً لأنه لا يمكنه التعامل مع أكثر من ملف واحد في الوقت الواحد ، شريطة أن تكون الملفات مكتوب بياناتها بطريقة تلاءم لغة البرمجة المستخدمة .

هذه الحقيقة كان لها نتائجها السلبية التي نلخصها على النحو:

1. حدت من مرونة النظام.
2. سببت نقص كفاءة النظام.
3. أتاحت وجود تكرارية في البيانات الواحدة.
4. رغم الارتباط المباشر بين البيانات بلغة البرمجة فليس متاحاً استخدام بيانات برنامج لبرنامج آخر رغم أنهما مكتوبان بنفس لغة البرمجة مما حتم استخدام نسخ جديدة من ملف البيانات ليلاءم البرنامج الآخر.
5. تسببت تكرارية البيانات في عدم تحقيق التكاملية بينها لأن تحديث أحد الملفات لا يعني بالضرورة تحديث الآخر.
6. انعكس كل هذا على زيادة كبيرة في وسائط تخزين البرامج والبيانات.

كل هذه المشاكل قادت إلى التفكير نحـ و إيجاد حلول مناسبة لأحداث تكاملية بين مختلف الملفات وإنهاء مشكلة تكرارية البيانات...الخ.

وقد سميت هذه الفكرة تكاملية الملفات Integrated Files ثم شاع مسمى قواعد البيانات Data Base وتتلخص الفكرة في وضع البيانات ضمن إطار موحد ونطلق عليه للتبسيط حوض البيانات Data Pool أو قاعدة بيانات Data Base بحيث يستطيع مختلف مستخدمي النظام التعامل معها بشكل سهل ومبسط.

تمارين الفصل

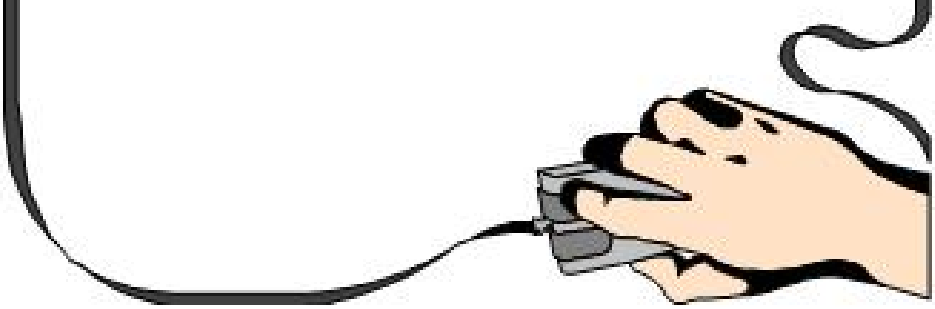
1. ما هي فائدة الملفات ؟
2. أملأ الفراغات التالية بالكلمات المناسبة:
   - تخزن الحواسب كميات كبيرة من المعطيات ضمن وحدات التخزين الثانوية مثل .....
   - يتألف .... من عدة حقول.
   - يتم تخزين المعطيات الضخمة ضمن الحواسب داخل ما نسميه .....
   - تقسم الملفات بحسب طريقة الوصول للبيانات بـ.....،.....
3. أكتب برنامج يقوم بتخزين مجموعة كلمات ثم يقوم بعرض هذه الكلمات بشرط أن تكون الكلمة تحتوي على حروف متكررة بداخل كل كلمة ؟
4. أكتب برنامج يقوم بتعديل ملف ؟
5. ما هي عيوب نظام الملفات ؟

Chp15

هياكل البيانات

في نهاية هذا الفصل سوف تتعلم :

- **مقدمة إلى هياكل البيانات**
- **أنواع هياكل البيانات.**
- **التعرف على المكدس ، الطابور ، القوائم ، الأشجار.**
- **تمثيل البيانات الديناميكية.**



15.1 مقدمة INTRODUCTION

من المألوف قبل الشروع إلى فهم شيء يجب معرفة توابعه وتفاعل هذه العناصر مع بيئتها المتوافرة فيها والعمليات التي يمكن أن تحدث على هذه العناصر التي تؤلف فيما بينها وحدة متناغمة مترابطة لكي توصل الفكرة إلى العقل بشكل جيد وسريع.

ومن الأشياء التي يجب تحديدها كمتخصصين في علم الحاسب الآلي هي البيانات DATA و المعلومات INFORMATION لإنها هي أساس تعاملنا مع الحاسب.

فما هي البيانات وما هي المعلومات وكيف يمكن لنا أن نحددها ؟

فالبيانات / هي مجموعة من الحقائق أو الأفكار قد تكون حروف أو أرقام أو صوراً أو خليطٌ مما سبق.

والمعلومات /مجموعة من الحقائق والأفكار عن شيء ما تمت معالجتها .

إذن فالبيانات هي المادة الخام للمعلومات أو أن المعلومات هي البيانات بعد معالجتها ومن الواضح آن الفرق الأساسي بينهم هي المعالجة .

- أنواع المعالجة على البيانات

1. الإضافة.
2. الحذف أو الإلغاء .
3. الدمج .
4. الفرز .
5. التحليل والتركيب باستخدام العمليات الحسابية (+ ،*،- ،/) والعمليات المنطقية (= ،=! ،=>, =< ,>, <) .
6. النسخ الالكتروني (SAVE).
7. الحماية والفك .
8. الاسترجاع والتعديل والتخزين .

إذن لكي نحصل على معلومات لابد من الحصول على بيانات أولاً ثم القيام بمعالجة هذه البيانات معالجة صحيحة.

- التركيب الفيزيائي والتركيب المنطقي للبيانات

نجد أن سرعة معالجة البيانات تعتمد كثيراً على عدة عوامل أضافه إلى العامل الزمني اللازم للمعالجة ومن أهميتها

- ü عوامل تحدد من الذاكرة الرئيسية.

- عوامل تحدد من وحدات الإدخال والإخراج.
- عوامل تحدد من تفاعل الإدخال والإخراج مع الذاكرة الرئيسية(تبادل المعلومات بين هذه الوحدات أي عملي مقايضة.

نجد أن العامل الأول يتطلب وجود برنامج للمعالجة والنسبة للعامل التأني فأنة يتطلب سرعة للوصول إلى المعلومات لإحضارها من الوحدات الإدخال ، والذي يهمنا هو الإقلال من عملية المقايضة بـ ين الذاكرة الرئيسية ووحدات الإدخال والإخراج ولهذا لابد من الإشارة إلى التركيب الفيزيائي والمنطقي للبيانات حيت نجد تعريف الاثنين باختصار

التركيب المنطقي / هو وجه نظر المبرمج في سير البرنامج أي آن هذا ترتيب معلومات البرنامج بشكل معين حتى يتم تنفيذ هذا البرنامج بطريقة صحيحة .

التركيب الفيزيائي / وهو يعني كيفية ترتيب البيانات على أوساط التخزين مثل الشريط المغناطيسي والقرص المغناطيسي حيت تخزن البيانات على القرص المغناطيسي بطريقة مباشرة أو تتابعيه أي تسلسلية مفهرسة.

هياكل البيانات وتراكيب البيانات وبني المعطيات لهما نفس المعنى / عبارة عن آليات وخوارزميات معينة توضع لبرامج بحيث تطبق بشكل جيد فهي مفيد جدا في برمجه قواعد البيانات و تساعد على تنفيذ مهام وتسهيل مهام من مهام الكمبيوتر ومن استغلال مواقع الذاكرة بشكل جيد وم نظم ويجب على المبرمج تطيق هذه الآلي ات بشكل جيد وإلا خلت من معنى الخوارزمية .

وبتعريف منطقي له / هي طريقه ترابط و ترص البيانات مع بعضها البعض في الذاكرة بحيث هذه البيانات تتخذ شكلا وهيكلاً معيناً في الذاكرة فتعتبر بنية عضوية لمجموعة من عناصر البيانات المتطابقة نوعاً

وشكلاً والتي تنظم في نسق واحد لتؤدي غرضاً محدداً.

15.2 فوائد هياكل البيانات

- التحكم في توزيع البيانات و التعرف إلى طبيعتها وبنائها الأساسي بنسق معين في الذاكرة.
- بناء برامج قوية ومتماسكة من حيت البناء والمنطق.
- تمكين المبرمج من أبداع طرق مبتكرة في كتابة البرامج المختلفة.
- اختصار زمن التخزين واسترجاع البيانات من الذاكرة .

15.3 أنواع هياكل البيانات

- هياكل بيانات ثابتة ساكنة (STATIC INFORMATION).

كالمتجهات والجداول والسجلات وعند التصريح عنها فيجب تحدي حجم هذه البيانات فلا تقبل الإضافة فوق حجمها المحدد .....

- هياكل بيانات شبكية.
- هياكل بيانات ديناميكية إي متحركة متغيرة

وينقسم هذا النوع إلى نوعين

1)هياكل بيانات خطية متغيرة / وهي التي تنظم في خط متتالي

- الملفات .
- القوائم .
- الطوابير .
- المكدسات .
- الأبجديات .
- المجموعات .

2) هياكل بيانات متشعبة إي بشكل عشوائي مخزنة في الذاكرة / مثل الأشجار ، الخرائط

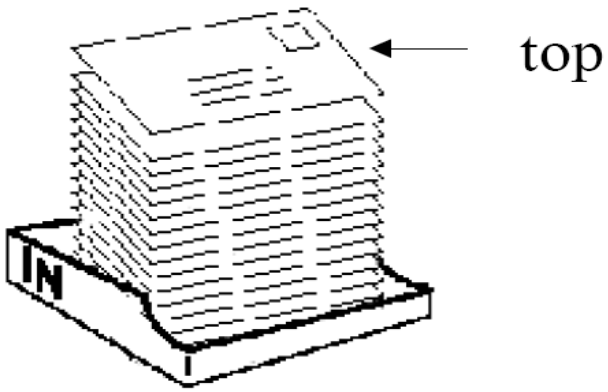
وسنتكلم عن هذه المواضيع مبدئياً بالهياكل الاستاتيكية بواسطة المتجهات .....

15.4 المكدس (Stack)

وهو عبارة عن نموذج خاص لتخزين البيانات بالية ثابتة وإخراجها باليه ثابتة بشكل مؤقت وهو عبارة عن صندوق توضع به البيانات بالية الداخل أولا الخارج أخرا والداخل أخرا الخارج أولا LIFO (LAST INPUT FIRST OUTPUT)

وكمثال بسيط أيضا نشبه عملة بقشطه المسدس الـ رصاصة الأولى تخرج أخر شيء والرصاصة الأخيرة تخرج أولاً ولهذا فان الإضافة تتم من الأعلى والحذف و يوجد مؤشر واحد يسمى top .

والقراءة أيضا يتم من الأعلى إي من طرف واحد عن طريق top كما في الشكل 1-15.



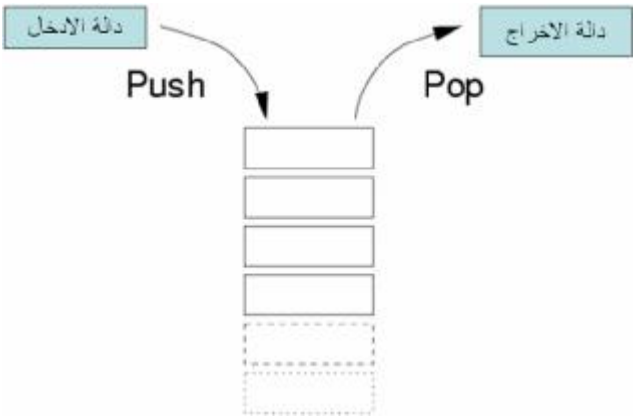
شكل 15-1

15.4.1 فوائد المكدس

- ü إيجاد قيم التعابير الحسابية
- ü يستخدم لغايات الاستدعاء الذاتي
- ü في عمليات الاعتراض والمقاطعة المستخدمة بالويندوز
- ü استدعاء البرامج الفرعية

ومن الإشكال السابقة نجد أن المكدس لا يحتوي إلا على مؤشر واحد فقط TOP

فعندما يكون المكدس فارغاً فأن TOP=-1 وعند إدخال أول قيمة فإننا نزيد من قيمة TOP ++ وكل ما أدخلنا قيمة فان المؤشر يزيد بمقدار واحد إلى أن يمتلئ المكدس والشكل 15-2 يبين ذلك.



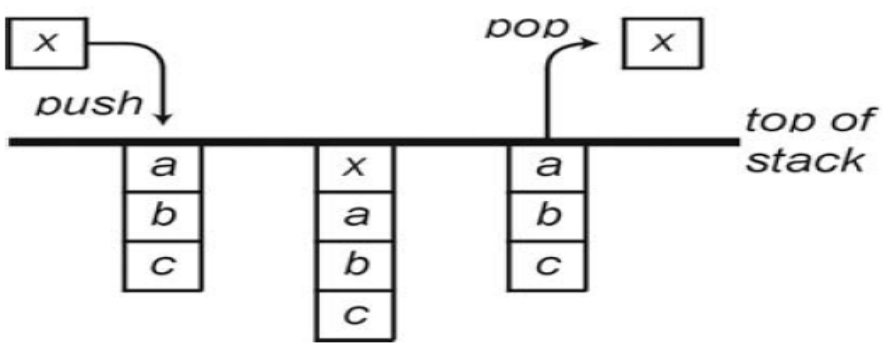
شكل 15-2

وعملية أخراج القيم من المكدس فإننا ننقص المؤشر بمقدار واحد أيضا إلى أن يصل قيمة المؤشر = -1 أو NULL فبهاكدا يكون المكدس فارغاً.

15.4.2 طرق تمثيل المكدس وتخزين عناصره في الذاكرة وتامين عملية بلوغها

- التمثيل المترابط الحلقي لعناصر المكدس على شكل لائحة إي على شكل قائمة .
- التمثيل المتراص (COMPACT) للعناصر في الذاكرة إي على شكل مصفوفة أحادية .

وكمثال على تمثيل المكدس بالمتجهات لننظر إلى الشكل 15-3



شكل 15-3

ولنفترض أن لدينا مصفوفة حجم (4) وأدخلنا أخر قيمة (X) وإذا أردنا إخراج قيمة فان المكدس سيعطي لنا أخر قيمة دخلت وهي (x) كما في الشكل 3-15
وهذا أول برنامج له يعمل على إدخال قيم داخل المكدس ثم يقوم بطباعته

```
1. //ArrayStack
2. import java.io.*;
3. class Chp15_1 {
4.   static final int CAPACITY = 5;
5.   static int[] Stack1 = new int[CAPACITY];
6.   static int top = -1;
7.
8.   static boolean isEmpty(){return (top < 0);}
9.
10.   static boolean isFull() {return (top+1== CAPACITY);}
11.
12.   static void push(int element){
13.     if (isFull())
14.                                 System.out.println("Stack is full.");
15.                                 else
16.                                 Stack1[++top] = element;
17.                           }
18.
19.   static int pop() {
20.
21.     if (isEmpty()){
22.                                  System.out.println("Stack is empty.");
23.                                  System.exit( 0 );
24.                                 }
25.     return Stack1[top--];
26.                 }
27.
28. public static void main(String args[])throws IOException {
29.         String num;
30.         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
31.         System.out.println( "Enter first integer" );
32.         while(!isFull())
33.                                                 {num=br.readLine();
34.                                                         push(Integer.parseInt(num));
35.                                                 }
36.
37.         while(!isEmpty())System.out.println(pop()+" ");
38.
39. }
40. }
```

شرح المثال:

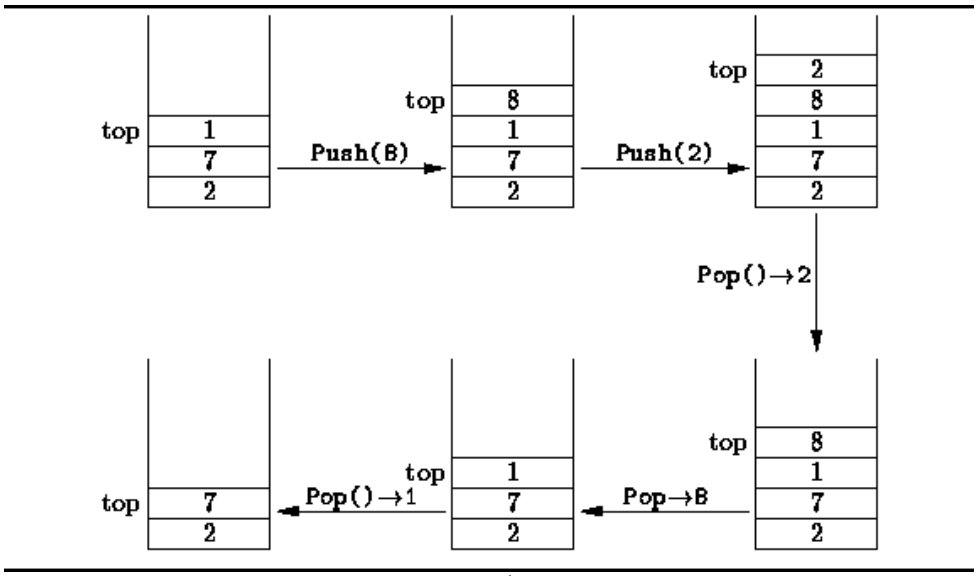
في السطر 6 تم تعريف Top ويسمى ذيل المكدس وهو متغير عام تستطيع الدوال الوصول إلية وتغير قيمته .
السطر 19 دالة أخراج البيانات من المكدس
السطر 12 دالة إدخال البيانات من المكدس وهي تأخذ بارا متر من نوع مصفوفة ومن نوع أنتجر

نعلم أن المكدس ممتلئ عندما يكون مؤشر الذيل أي top = حجم المصفوفة -1
وإلا سنزيد من المؤشر بواحد وسنضع القيمة بداخل المصفوفة بداخل الموقع الذي تكون قيمته top

STACK TOP ,PUSH,POP كل هذه التعابير عبارة عن أسماء متغيرات وليس من الضروري التقيد بهذه الأسماء فهي ليست دوال .

نعلم أن المكدس أصبح فارغاً عندما تكون قيمة المؤشر top =-1 أي اقل من الصفر
وإلا سنخرج القيمة من داخل المصفوفة التي تحمل عنوان قيمة top وسنطرح قيمة المؤشر بواحد .

فيكون ناتج تنفيذ البرنامج كالتالي:



شكل 15-4

ومن المثال السابق والشكل 15-4 يتبين لنا إليه عمل المكدس فيا أحبابي لا يخيفكم هذا المصطلح الغريب STACK مصفوفة ولا يختلف عنها إلى بشيء واحد ألا وهي إليه عملة التي ذكرناها سابقاً.

وكمثال أخر سنقوم باستخراج اكبر قيمة بالمكدس فقط سيكون التغير في دالة ألإخراج pop

```
1. //استخراج اكبر عدد من المكدس
2. import java.io.*;
3. class Chp15_2 {
4.   static final int CAPACITY = 5;
5.   static int[] Stack1 = new int[CAPACITY];
6.   static int top = -1;
7.
8.   static boolean isEmpty(){return (top < 0);}
9.
10.   static boolean isFull() {return (top+1== CAPACITY);}
11.
12.   static void push(int element){
13.     if (isFull())
14.                                 System.out.println("Stack is full.");
15.                                 else
16.                                 Stack1[++top] = element;
17.                           }
18.
19.   static int pop() {
```

```
20.
21.    if (isEmpty()){
22.                                System.out.println("Stack is empty.");
23.                                System.exit( 0 );
24.                               }
25.    return Stack1[top--];
26.                }
27.   static int max(){ int temp=pop(),temp2;
28.                                                  while(!isEmpty()){
29.                                                                        temp2=pop();
30.
if(temp<temp2)temp=temp2;
31.                                                                      }
32.                                                   return temp;
33.                                                  }
34.
35. public static void main(String args[])throws IOException {
36.        String num;
37.        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
38.        System.out.println( "Enter first integer" );
39.        while(!isFull())
40.                                          {num=br.readLine();
41.                                                  push(Integer.parseInt(num));
42.                                          }
43.
44.        System.out.println(max());
45.
46. }
47. }
```

شرح المثال:
اعتقد أن المثال واضح وبسيط ولا يوجد بة إي تعقيد فقط الاختلاف بينة وبين البرنامج السابق هو إضافة منهاج max التي تعيد لنا أكبر قيمة بداخل المكدس.

إلا الآن اعتقد قد تبثث فكرة المكدس والية عملة في ذهنك ، الآن إذا طلب منك إن تدخل بيانات إلى المكدس وتعكس المكدس فكيف ذلك سيكون فكر قليلاً وتذكر إلية عمل المكدس ولا تقول نقوم بطباعة المصفوفة من البداية فهذا ليس صحيحاً فقد خليت من عملة .....
ها هل أتت الفكرة بعقلك حاول ولا تستعجل.....
يبدو لي أن الفكرة لم تأتي إليك إذن صلي على نبيك وتتبع البرنامج التالي بهدوء

```
1. // برنامج يعمل على عكس مكدس
2. import java.io.*;
3. class Chp15_3 {
4.   static final int CAPACITY = 5;
5.   static int[] Stack1 = new int[CAPACITY];
6.   static int top = -1;
```

```
7.
8.  static boolean isEmpty(){return (top < 0);}
9.
10.  static boolean isFull() {return (top+1== CAPACITY);}
11.
12.  static void push(int element){
13.   if (isFull())
14.                              System.out.println("Stack is full.");
15.                              else
16.                              Stack1[++top] = element;
17.                         }
18.   static int pop() {
19.
20.    if (isEmpty()){
21.                               System.out.println("Stack is empty.");
22.                               System.exit( 0 );
23.                              }
24.    return Stack1[top--];
25.                }
26.  static void rev(){
27.                                                    int[] Stack2 = new int[CAPACITY];
28.                                                    int[] Stack3 = new int[CAPACITY];
29.                                                    int top2=-1,top3=-1;
30.
31.                                                    while(top>=0)Stack2[++top2]=Stack1[top--];
32.                                                    while(top2>=0)Stack3[++top3]=Stack2[top2--];
33.                                                    while(top3>=0)Stack1[++top]=Stack3[top3--];
34.                                          }
35.
36. public static void main(String args[])throws IOException {
37.       String num;
38.       BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
39.       System.out.println( "Enter first integer" );
40.       while(!isFull())
41.                                          {num=br.readLine();
42.                                                 push(Integer.parseInt(num));
43.                                          }
44.       rev();
45.       while(!isEmpty())System.out.println(pop());
46. }
47. }
```

شرح المثال:

فكرة البرنامج هو استخدام مكدسين آخرين لعملية نقل البيانات فعندما نقلنا البيانات من المكدس الأول إلى المكدس الثاني فان البيانات قد انعكست ولكننا استخدمنا مكدس ثالث لكي نعيد البيانات إلى المكدس الأصلي بالطريقة التي طلبت منا وبالية عمل المكدس .

وهذا الكود يعمل على حذف إي قيمة من المكدس ؟

```
1. // حذف أي عدد من دتخل المكدس
2. import java.io.*;
3. class Chp15_4 {
4.   static final int CAPACITY = 5;
5.   static int[] Stack1 = new int[CAPACITY];
6.   static int top = -1;
7.
8.   static boolean isEmpty(){return (top < 0);}
9.
10.   static boolean isFull() {return (top+1== CAPACITY);}
11.
12.   static void push(int element){
13.     if (isFull())
14.                                 System.out.println("Stack is full.");
15.                                 else
16.                                 Stack1[++top] = element;
17.                           }
18.
19.   static int pop() {
20.
21.     if (isEmpty()){
22.                                  System.out.println("Stack is empty.");
23.                                  System.exit( 0 );
24.                                 }
25.     return Stack1[top--];
26.                 }
27.   static void delete(int number){ int top2=-1,temp;
28.                                                 int[] Stack2 = new int[CAPACITY];
29.                                                  while(!isEmpty()){
30.                                                                 temp=pop();
31.
            if(number!=temp)Stack2[++top2]=temp;
32.                                                                                 }
33.                                                   while(top2>=0)push(Stack2[top2--]);
34.                                               }
35.
36. public static void main(String args[])throws IOException {
```

```
37.     String num;
38.     BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
39.     System.out.println( "Enter first integer" );
40.     while(!isFull())
41.                         {num=br.readLine();
42.                          push(Integer.parseInt(num));
43.                         }
44.
45.     System.out.println("Enter number delete");
46.     num=br.readLine();
47.     delete(Integer.parseInt(num));
48.     System.out.println();
49.     while(!isEmpty())System.out.println(pop());
50.
51. }
52. }
```

15.4.3 الصنف من نوع Stack

لقد بينا في السابق طرق التعامل مع المكدس ، حيث قمنا بإنشاء جميع المناهج المتعلقة بعمليات المكدس يدوياً.

لغه Java توفر لك الصنف java.util.Stack الذي يمكنك من التعامل مع حميع عمليات المكدس من حذف،استعادة،حشر إي عنصر من قمة المكدس. ويقدم العديد من المناهج من أجل تقديم بنية معطيات تحقق القاعدة التالية : الداخل أولاً الخارج أخراً LIFI . والجدول 15-1 يبين مناهج هذا الصنف:

جدول 15-1

empty()	يعيد القيمة true إذا كان المكدس فارغاً
peek()	يعيد العنصر الموجود في قمة المكدس دون أن يحذفه
push()	يضيف عنصر إلى قمه المكدس
search()	يعيد مكان العنصر المحدد ضمن المكدس
remove()	تحذف موقع عنصر ما

ونعرض الآن مثال يستخدم جميع المناهج التي ذكرت في الجدول السابق:

```
1. // استخدام  المكتبة الخاصة بالمكدس
2. import java.io.*;
3. class Chp15_5 {
4. public static void main(String args[])throws IOException {
5.     String num;
6.     int i;
7.     BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
8.     java.util.Stack stack= new java.util.Stack();
9.     System.out.println( "Enter first integer" );
```

```
10.
11.     for(i=0;i<5;i++)
12.                         {num=br.readLine();
13.                          stack.push(new Integer(Integer.parseInt(num)));
14.                         }
15.
16.     System.out.println("Enter number Select");
17.     System.out.println("1- Search");
18.     System.out.println("2- Remove");
19.     System.out.println("3- Desply");
20.     num=br.readLine();
21.
22.     switch(Integer.parseInt(num)){
23.             case 1:
24.                                 System.out.println("Enter number Search");
25.                                 num=br.readLine();
26.                                 System.out.println("The Pos In "+
27.                                 stack.search(new Integer(Integer.parseInt(num))));
28.                                 if(stack.search(new Integer(Integer.parseInt(num)))<0)
29.                                         System.out.println("Not Found ");
30.                                 break;
31.             case 2:
32.                                 System.out.println("Enter Pos number delete");
33.                                 num=br.readLine();
34.                                 stack.remove(Integer.parseInt(num));
35.                                 while(!stack.empty())System.out.print(stack.pop()+" ");
36.                                 break;
37.             case 3:
38.                                 while(!stack.empty())System.out.print(stack.pop()+" ");
39.                                 break;
40.                                                                         }
41.
42. }
43. }
```

شرح المثال:

في السطر 8 تم اشتقاق صنف جديد باسم stack من المكتبة الخاصة بالتعامل مع المكدس.

وفي السطر 13 تم إدخال إلى المكدس العناصر بواسطة التعليمة push ، وتلاحظ أننا قمنا بعملية التحوير new Integer لنحدد نوع البيانات التي سيخزنها المكدس.

في السطر 27 نفس قمنا بالبحث عن عنصر . ونلاحظ أن التعليمة search تعطي لنا موقع العنصر إن وجد وإلا تعيد القيمة -1 في حالة إنها لم تجده.

في السطر 34 قمنا بحذف قيمة عنصر بواسطة قيمة الموقع.


من الأخطاء الشائعة إفراغ المكدس بواسطة التعليمة pop ومن ثم نقوم البحث عن عنصر أو حذف عنصر. مما يسبب لنا خطاء في زمن التنفيذ.


عدم استخدام عملية تحديد المعطيات أثناء إدخال القيم للمكدس أو بحث عن قيم ، مما يسبب لنا خطاء قواعدي.

15.5 الطوابير (Queues) أو سجلات الآنتظار

وهي عبارة نوع من هياكل البيانات الخطية ويشبه المكدس لتخزين المعلومات بشكل مؤقت مع فارق يكمن في أن التنظيم المتبع لإدخال المعلومات وإخراجها هو FIFO (First Input First Output) إي الداخل أولا الخارج أولا إي تكون عملية الإضافة من النهاية والحذف من الأمام إي يـ وجد للطابور مؤشرين مؤشر الرأس ويسمى head or front ومؤشر الذيل ويسمى tail or rear وعند الإضافة فإننا نزيد من قيمة الذيل بواحد وعند الحذف فإننا نزيد قيمة الرأس بواحد أيضا فتكون البيانات مرتصة بشكل نتتإلى ومتقاربة على شكل خط وليست على مواقع متفرقة بالذاكرة إي أشبة بالطابور المدرسي فأول طالب حاضر هو أول طالب داخل للفصل .
فهو يشبه طابور الآنتظار للإفراد عند المؤسسة أو المستشفى كما في الشكل 5-15.

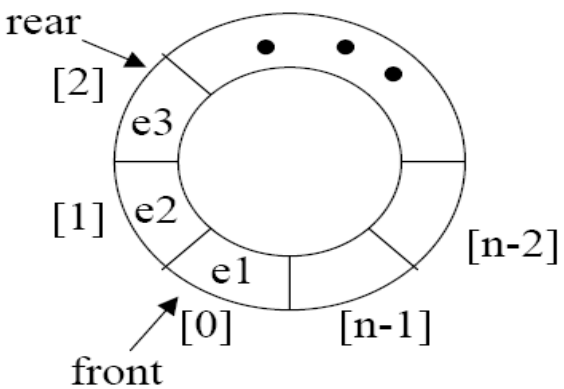
15.5.1 أنواع الطوابير

طابور خطي

وهو له حجم محدود وشرط امتلائه أن تكون قيمة الذيل تساوي حجم المصفوفة .

شكل 5-15

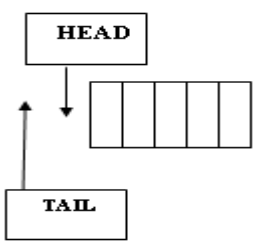
2) طابور دائري / نفس تعريف السابق إلى أن شرط الامتلاء يختلف عن السابق الرأس =1 و الذيل = حجم المتجه أو الرأس=الذيل+1 كما في الشكل 15-6.



شكل 15-6

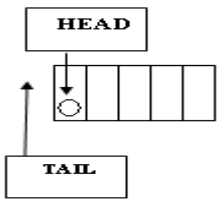
وسنبدأ بالتحدث إلى الطابور الخطي

نجد في البداية يكون الرأس والذيل لا يؤشران لأي موقع ولمعرفة أن الطابور لم تدخل إلية إي قيمة عندما يكون if(tail==-1||head==-1) والشكل 15-7 يوضح ذلك.



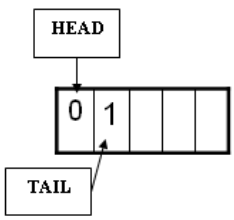
شكل 15-7

وعند إدخال أول قيمة يصبح قيمة الرأس والذيل = 0 كما في الشكل 15-8.



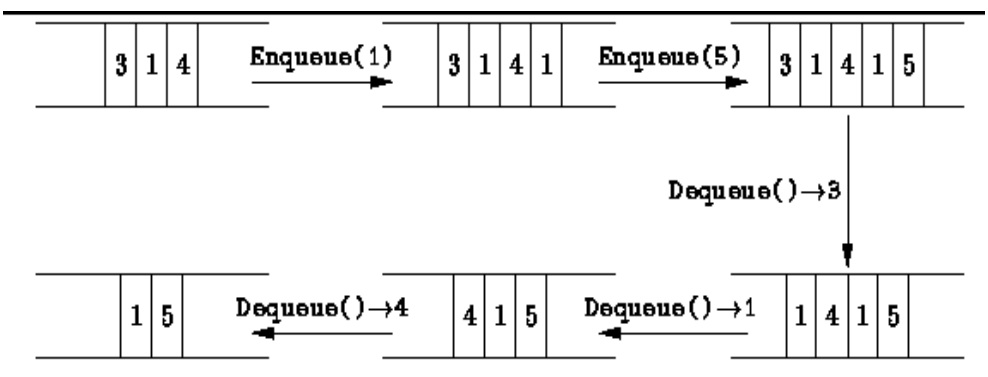
شكل 15-8

وعند إدخال ثاني قيمة نزيد من قيمة الذيل بواحد فقط أم الرأس يبقى كما هو ، كما في الشكل 15-9.

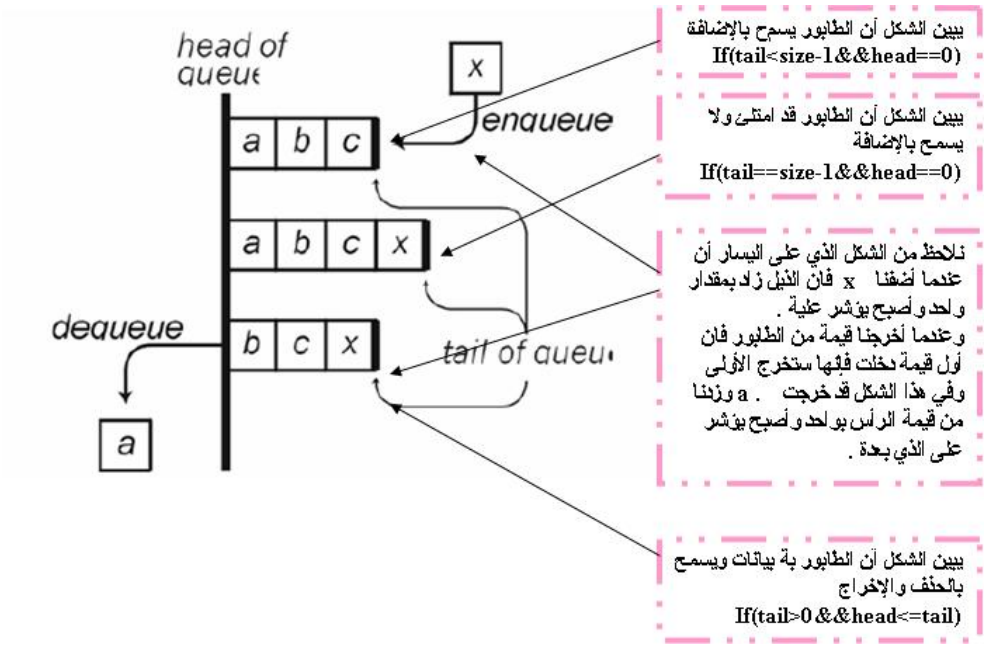


شكل 15-9

وعملية الحذف عكس السابق إي يكون الذيل تابت والرأس يزيد في كل عملية حذف بمقدار واحد
مع عمل إزاحة للمتجه لليسار في كل عملية حذف إن أردت .والأشكال 15-10,15-11 تبين عملية الإدخال و الإخراج من داخل الطابور.



شكل 15-10



شكل 11-15

ومن خلال الإشكال السابقة سنورد أول برنامج للطابور

```
1. // برنامج الطابور
2. import java.io.*;
3. class Chp15_6 {
4.   static final int CAPACITY = 5;
5.   static int[] Queue = new int[CAPACITY];
6.   static int tail=-1,head=-1;;
7.
8.   static boolean isEmpty(){return (tail < 0||head>tail);}
9.
10.   static boolean isFull() {return (tail== CAPACITY-1);}
11.
12.   static void add_Queue(int element){
13.     if (isFull())
14.                         System.out.println("Is FULL Queue");
15.                         else
16.                                 {
17.                                  if(tail==-1){head=tail=0;Queue[tail]=element;}
18.                                         else
19.                                         Queue[++tail]=element;
20.                                 }
21.                                         }
```

```
22.
23.   static int De_Queue() {
24.
25.    if (isEmpty()){
26.                                System.out.println("Queue is empty.");
27.                                System.exit( 0 );
28.           }
29.    return Queue[head++];
30.                                 }
31.
32. public static void main(String args[])throws IOException {
33.        String num;
34.        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
35.        System.out.println( "Enter first integer" );
36.        while(!isFull())
37.                                          {num=br.readLine();
38.                                                  add_Queue(Integer.parseInt(num));
39.                                          }
40.
41.        System.out.println();
42.        while(!isEmpty())System.out.print(De_Queue()+" ");
43.
44.                                                                          }
45. }
```

15.5.2 العمليات على الطابور

*)الإضافة ADD

والكود التابع لهذه العملية هو نفس الكود السابق

*)الحذف DEL

وهذا الكود لهذه العملية

```java
// برنامج الحذف من الطابور
import java.io.*;
class Chp13_2 {
  static final int CAPACITY = 5;
  static int[] Queue = new int[CAPACITY];
  static int tail=-1,head=-1;

  static boolean isEmpty(){return (tail < 0||head>tail);}

  static boolean isFull() {return (tail== CAPACITY-1);}

  static void add_Queue(int element){
    if (isFull())
                System.out.println("Is FULL Queue");
                else
                {
                 if(tail==-1){head=tail=0;Queue[tail]=element;}
                   else
                   Queue[++tail]=element;
                }
                              }

  static int De_Queue() {

    if (isEmpty()){
                   System.out.println("Queue is empty.");
                   System.exit( 0 );
                  }
    return Queue[head++];
                   }

```

```
32    static void del_Queue(int element){
33      int[] Queue2 = new int[CAPACITY];
34      int tail2=-1,head2=-1;
35      if(isEmpty()){
36                      System.out.println("Queue is empty.");
37                      System.exit( 0 );
38                    }
39      else
40      while(head<=tail){
41                      if(Queue[head]!=element){
42                      if(tail2==-1){head2=tail2=0;Queue2[tail2
43                       else
44                       Queue2[++tail2]=Queue[head];
45                                                 }
46                      head++;
47                     }
48      head=tail=-1;
49      while(tail2>=head2){
50                     if(tail==-1){head=tail=0;Queue[tail]=Queu
51                     else
52                     Queue[++tail]=Queue2[head2];
53      head2++;              }
54                                      }
55
56  public static void main(String args[])throws IOException {
57      String num;
58      BufferedReader br = new BufferedReader(new InputStreamR
59      System.out.println( "Enter first integer" );
60      while(!isFull())
61                      {num=br.readLine();
62                       add_Queue(Integer.parseInt(num));
63                      }
64      System.out.println("Enter number delete");
65      num=br.readLine();
66      del_Queue(Integer.parseInt(num));
67
68      System.out.println();
69      while(!isEmpty())System.out.print(De_Queue()+" ");
70
71  }
72  }
```

شرح المثال:

الفكرة المستخدمة بالبرنامج هو خلق طابور جديد وإدخال جميع القيم ماعدا القيمة التي تساوي القيمة المراد حذفها تم نقل الطابور الجديد للقديم كما في الأسطر من 40 إلى 53.

*) البحث

نفس البرنامج السابق إلى إننا لا نقوم بخلق طابور ونقل بل إننا نبحث علية إن وجد نطبعه وإلا نطبع إننا لم نحصل علية . كما يلي:

```
1. // برنامج البحث عن عنصر بداخل الطابور
2. import java.io.*;
3. class Chp15_8 {
```

```
4.  static final int CAPACITY = 5;
5.  static int[] Queue = new int[CAPACITY];
6.  static int tail=-1,head=-1;
7.
8.  static boolean isEmpty(){return (tail < 0||head>tail);}
9.
10.  static boolean isFull() {return (tail== CAPACITY-1);}
11.
12.  static void add_Queue(int element){
13.    if (isFull())
14.                              System.out.println("Is FULL Queue");
15.                              else
16.                                        {
17.                                         if(tail==-1){head=tail=0;Queue[tail]=element;}
18.                                                  else
19.                                                  Queue[++tail]=element;
20.                                        }
21.                         }
22.
23.  static int De_Queue() {
24.
25.    if (isEmpty()){
26.                          System.out.println("Queue is empty.");
27.                          System.exit( 0 );
28.              }
29.    return Queue[head++];
30.                                    }
31.
32.  static void F_Queue(int element)
33.                    {int y=0,temp;
34.                     if (isEmpty()){
35.                              System.out.println("Queue is empty.");
36.                              System.exit( 0 );
37.                          }
38.                     else
39.                     while(!isEmpty()){temp=De_Queue();
40.                                                       if(temp==element)
41.                                                              {y=1;
42.                                                               System.out.println("FOUND
"+temp);
43.                                                               break;
44.                                                               }
45.                                        }
46.                    if(y==0)System.out.println("NOT FOUND "+element);
47.                    }
48.
49.
50. public static void main(String args[])throws IOException {
51.        String num;
52.        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
53.        System.out.println( "Enter first integer" );
```

```
54.     while(!isFull())
55.                                     {num=br.readLine();
56.                                             add_Queue(Integer.parseInt(num));
57.                                     }
58.     System.out.println("Enter number Search");
59.     num=br.readLine();
60.     F_Queue(Integer.parseInt(num));
61.                                                                     }
62. }
```

*) دمج طابورين

سأذكر فكرة البرنامج وعلى القارئ أن يحل الكود

أولا يجب التأكد من أن الطابور الثاني بوجد فيه مساحة كافية لاستيعاب قيم الطابور الأول ومن العلاقة الآتية

(عدد العناصر الموجودة في الأساسي- عدد القيم)=><(عدد العناصر بداخلة - حجم الطابور المستضيف) ثم بعد ذلك نضيف الطابور الثاني في الأول .

الطابور الدائري

نفس البرامج التي ذكرناها سابقاً والاختلاف سيكون في أوامر الشرط

يكون الطابور فارغاً if(head==tail+1) .

يكون الطابور غير ممتلئ if(tail!=size&&head=1) .

يكون الطابور ممتلئا if(head==1&&tail==size).

ونفس البرامج التي ذكرتها بالمكدس تطبق على الطوابير بالية الطابور.

وإلى هنا يجب على القارئ أن يكون قد اتضحت فكرة الطابور .

15.6 القوائم (List)

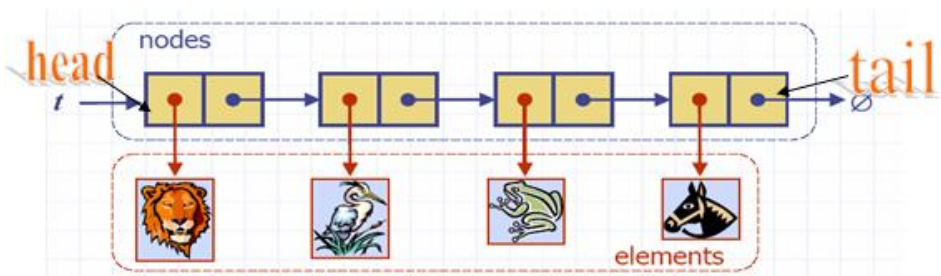
وهي نوع من هياكل البيانات الخطية تتألف من مجموعة من الخلايا المرابطة فيما بينها وكل عنصر فيها يسمى عقدة وهذه العقدة فيها حقلين حقل للقيم وحقل يؤشر لعنوان للعقدة الذي بعدها أو قبلها أو NULL وتستعمل هذه الكلمة للدلالة إلى نهاية اللائحة , ومن الممكن أن تتأ لف العقدة على أكثر من مؤشر ومعلومات إي قيم ، فتكون ضمن مجموعة (block) أو كتلة ، ولا بد من مؤشر يؤشر إلى أول عقدة ومؤشر يؤشر إلى أخر عقدة إي مثل الطابور .

أنواع القوائم

- القوائم الأحادية.
- القوائم المذبلة أي الثنائية.
- القوائم الدائرية.

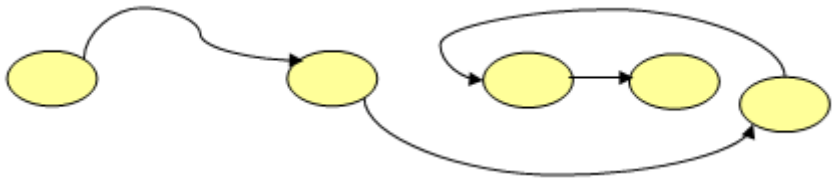
15.6.1 القوائم الأحادية

تشبه حبل الغسيل تعلق علية البيانات تتالياً إذا كان الإدخال في نفس الوقت ويوجد عنوان راسي يؤشر إلى أول عنصر من اللائحة ويسمى head ويوجد عنوان نهائي يؤشر إلى أخر عنصر من اللائحة ويسمى tail وكل عقدة تؤشر إلى العقدة التالية وأخر عقدة تكون قيمة المؤشر لها NULL و تكون كما في الشكل 12-15



شكل 12-15

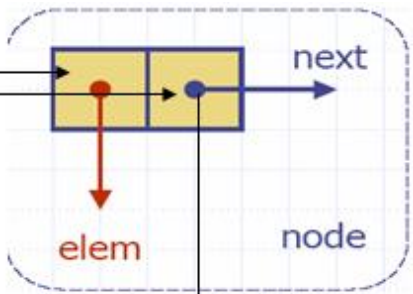
وليس من الضروري أن تكون العقد مرتبة بشكل متتالي في الذاكرة فهي تكون مبعثرة في الذاكرة لان الجهاز الذي يحجزها في الذاكرة وليس اليوزر لكنها متصلة فيما بينها بواسطة المؤشرات و الشكل 13-15 يبين كيف تكون شكلها .



شكل 13-15

- كيفية تعريف الهيكل العام للعقد

```
class Node {
  element;
  Node next;
  Node( s) {
              element = s;
              next = null;
             }
getElement( ) { return element; }
void setElement(newElem) { element = newElem; }
}
```



شكل 14-15

ما معنى هذا الحقل next ؟

معناه مؤشر من نوع الصنف Node نفسه ، أي يؤشر إلى صنف Node أخر من نفس النوع ، أي نستطيع الوصول إلى Node أخر وأخر و هكدا إلى مالا نهاية. ويكون بداخل العقدة موقع العقدة التي بعدها أو قبلها.

أي لنتذكر أحبائي علبة الحليب حيث أن بداخلها نفس صورة العلبة نفسها و بداخل الصورة نفس الصورة العلبة ووو إلا مالا نهاية . والعقدة هنا نفس الشيء حيث أن داخلة عقدة وبداخل العقدة عقدة وهكذا .

المنهج getElement() :

تعطي لنا قيمة العقدة.

المنهج setElement () :

نخزن قيمة مرسلة للعقدة.

وللإضافة عدة أنواع

- الإضافة من اليمين
- الإضافة من اليسار

- الإضافة من أي مكان

وهذا أول مثال لهذه القوائم وهو الإضافة من اليمين للقائمة:

```
1. // برنامج الأضافة من اليمين للقائمة الأحادية
2. class Chp15_9{
3. public static void main(String args[]){
4.   Node head=null;                    // head node of the list
5.   Node tail=null;                    // tail node of the list
6.   Node node=null;
7.    int size=5;
8.
9.    tail=head=node=new Node(0);
10.  for (int i=1;i<size;i++)
11. {
12.   node=new Node(i);
13.   tail.setNext(node);
14.   tail=node;
15. }
16.
17.  node=head;
18.
19.  while(node!=null)
20. {
21.   System.out.print(node.getElement()+" ");
22.   node=node.getNext();
23. }
24.  System.out.println();
25. }
26. }
27.
28. /** Node of a singly linked list of ints. */
29. class Node {
30.   private int element;    // we assume elements are character ints
31.   private Node next;
32.   /** Creates a node with the given element and next node. */
33.   public Node(int s) {
34.     element = s;
35.     next = null;
36.   }
37.   /** Returns the element of this node. */
38.   public int getElement() { return element; }
39.   /** Returns the next node of this node. */
40.   public Node getNext() { return next; }
41.   // Modifier methods:
42.   /** Sets the element of this node. */
43.   public void setElement(int newElem) { element = newElem; }
44.   /** Sets the next node of this node. */
45.   public void setNext(Node newNext) { next = newNext; }
```

46. }

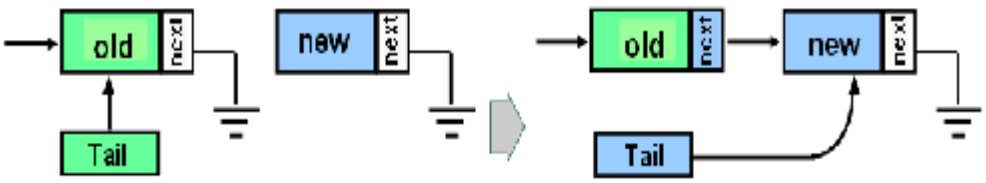
شرح المثال:
سنبدأ بشرح المناهج المضافة للصنف Node :
السطر 38 منهج يعيد لنا قيمة العقدة.
السطر 40 منهج يعيد لنا موقع العقدة التالية.
السطر 43 منهج يخزن قيمة مرسلة للعقدة في المتحول element.
السطر 45 منهج يخزن موقع عقدة مرسلة للعقدة في المتحول next.

الأسطر (4 - 6) هنا عرفنا head من نوع Node مؤشر لعقدة وهو الرأس و tail من نوع Node مؤشر لعقدة وهو الذيل و node سنستخدمه كمتغير لإدخال بيانات العقد .


قم بإسناد القيمة null إلى التوابع الخاصة بالعقدة لأنة يجب إعطاء قيم ابتدائية لها قبل التعامل معها.

السطر 9 تم إنشاء أول عقدة وإرسال إليها القيمة 0 ، وجعل العقدة تساوي head , tail , node حيث تعتبر هذه أهم مرحلة عند إنشاء العقد.


عند إنشاء العقد يجب أن تنشئ أول عقدة بمفردها حتى يتم مساواة الرأس والذيل بها.
الأسطر (10 - 15) هنا سنكوِّن 4 عقد أضافية بجانب الأولى فيكون لدينا 5 عقد .
السطر 13 يبين أن حقل العقدة الأولى يساوي العقدة الجديدة . فبهذه الحالة تمت عملية الربط بين العقدتين بقي علينا نقل الذيل إلى العقدة الجديدة tail=node كما في السطر 14 .
وهكذا بباقي العقد إلى أن ينتهي عمل اللوب ويمكنك تكوين مئات العقد بهذه الطريقة والشكل 15-15 يبن الشرح .
الأسطر (17 - 23) في عملي طباعة العقد أول شي يجب أن تعمله هو الوصول لأول عقدة فكيف ستعمل لو تتذكر قليل أن أول ما أنشانا أول عقدة ساوينا الرأس والذيل بها وبعد ذلك كان كل ما أضفنا عقدة جديدة تحرك معانا الذيل وأصبح الذيل بمؤخرة العقد والرأس في بـ داية العقد .إذن node= head فنكون وصلنا إلى أول عقدة كما في السطر17 .
بقي علينا طباعة العقد والتنقل إلى العقدة التالية كما في السطر 22. بمعنى أن العقدة التي واقفين عليها تساوي حقل العقدة نفسها التي داخلة موقع العقدة التالية فبذلك نكون قد انتقلنا إلى العقدة التالية وتستمر هذه العملية إلى أن تساوي العقدة NULL فينتهي عمل اللوب .



لشكل 15-15

أم الإضافة من اليسار نفس المثال السابق إلا أن الاختلاف فقط بعملية إدخال العقد الثانية وما بعدها

```
for (int i=1;i<size;i++)
{
 node=new Node(i);
 node.setNext(head);
 head=node;
}
```

الإضافة من اليمين يكون الرأس متحرك والإضافة من اليمن يكون الذيل هو المتحرك .

وهذا مثال على إضافة عقدة بعد قيمة عقدة يريدها المستخدم :

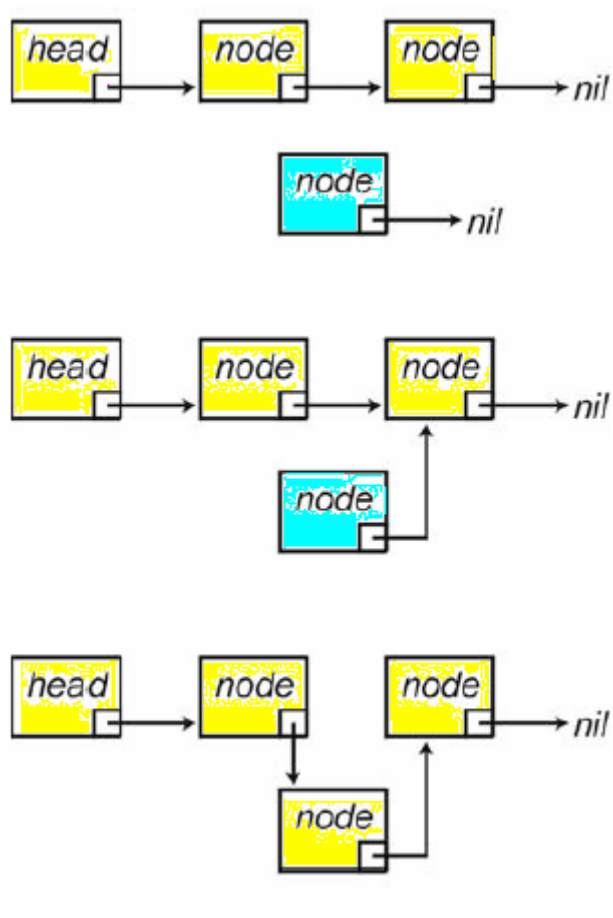
```
1. // برنامج الأضافة بعد قيمة عقدة معينة
2. import javax.swing.JOptionPane;
3. class Chp15_10 extends  Node{
4. public static void main(String args[]){
5.   Node head=null;                    // head node of the list
6.   Node tail=null;                    // tail node of the list
7.   Node node=null;
8.    int size=5;
9.
10.  tail=head=node=new Node(0);
11.  for (int i=1;i<size;i++)
12.  {
13.   node=new Node(i);
14.   node.setNext(head);
15.   head=node;
16.  }
17.
18.  node=head;
```

```
19.
20. while(node!=null)
21. {
22.  System.out.print(node.getElement()+" ");
23.  node=node.getNext();
24. }
25. System.out.println();
26.
27.    String snum1;
28.    int num1;
29.    snum1 = JOptionPane.showInputDialog("Enter num1:") ;
30.    num1 = Integer.parseInt(snum1);
31.
32. node=head;
33. while(node!=null)
34. {
35.    if(node.getElement()==num1)
36.                          {
37.                           Node temp;
38.                           snum1=JOptionPane.showInputDialog
                                                    ("Enter Value  node:") ;
39.                                  num1 = Integer.parseInt(snum1);
40.                           temp=new Node(num1);
41.                           temp.setNext(node.getNext());
42.                           node.setNext(temp);
43.                           break;
44.                          }
45.    node=node.getNext();
46. }
47. node=head;
48. while(node!=null)
49. {
50.  System.out.print(node.getElement()+" ");
51.  node=node.getNext();
52. }
53. System.out.println();
54. System.exit(0);
55. }
56. }
```

شرح المثال:

بعد إدخال العقد طلبنا من المستخدم إدخال قيمة فإذا وجدت هذه القيمة بالقائمة سنضع العقدة الجديدة بعدها مباشرة .

فعملنا عملية بحث عن العنصر إذا وجد فإننا سنعمل على إنشاء عقدة جديدة وسندخل قيمة العقدة وسنربط حقل مؤشر العقدة الجديدة بالعقدة التي بعد العنصر والعقدة التي مازلنا واقفين عليها تم ربط مؤشرها بالعقدة الجديدة كما في الأسطر (35 - 44) والشكل 16-15 يبين هذه العملية .



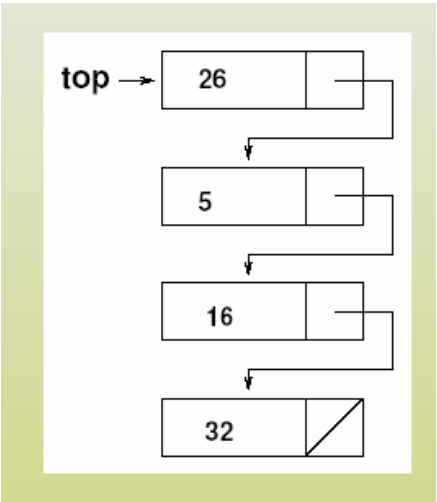
شكل 15-16

15.6.2 صنع المكدسات و الطوابير ديناميكياً

تحدثنا عن الهياكل الإستاتيكيه أي الثابتة وتكلمنا عن المكدسات والطوابير لنعمل على تطبيق تلك الخوارزميات بالقوائم الأحادية ونجعلها متغيرة أي ديناميكية ونتخلص من شيء أسمة المكدس قد امتلئ أو الطابور قد امتلئ والآن سنورد مثال عن المكدس باستخدام القوائم الأحادية والية الإدخال والإخراج قد تكلمنا عنها في السابق .
واليكم الكود :

```
1. // برنامج مكدس بواسطة القوائم
2. class Chp15_11 extends  Node{
3. public static void main(String args[]){
4.    int size=5;
5.    stack stack1=new stack();
6.    for (int i=1;i<size;i++)
7.       stack1.push(i);
8.
9.    while(!stack1.isEmpty())
10.   System.out.print(stack1.pop()+" ");
11.
12.  System.out.println();
13. }
14. }
15.
16. /** A linked Stack. */
17. class stack extends Node{
18.
19.    public Node Stack1=null,top =null ;
20.    //* Return whether the stack is empty.
21.    public boolean isEmpty(){return (top == null);}
22.    //*Insert an element at the top of the stack.
23.    public void push(int element){
24.  if(top==null)
25.                    {
26.                     top=Stack1=new Node(element);
27.                    }
28.                    else
29.                    {
30.                     Stack1=new Node(element);
31.                     Stack1.setNext(top);
32.                     top=Stack1;
33.                              }
34.                                                  }
35.     /**
36.     * Remove the top element from the stack.
37.     * @return element removed.
38.     * @exception EmptyStackException if the stack is empty.
39.     */
40.    public int pop() {
41.
42.     if (isEmpty()){
43.                                      System.out.println("Stack is empty.");
44.                                      System.exit( 0 );
45.                                     }
46.                                      int temp=top.getElement();
47.     top = top.getNext();
48.     return temp;
49.                                       }
50. }
```

الشكل 15-17 يبين شكل العقد للمكدس
وجميع ما ذكرناه من تطبيقات المكدس والطوابير على القارئ أن يطبق تلك الأمثلة بالقائمة الأحادية.



شكل 15-17

وبعد أن تكلمت على عملية الإضافة بجميع أنواعها بالقائمة الأحادية يبقى لنا الآن أن نتكلم عن عملية الحذف وما يدور من تطبيقات حولها.

- الحذف (DELETE)

هو عملية بسيطة في لائحة الوصل الخطية ، فتوصل وصلة العنصر الذي يأتي قبل العنصر المراد حذفه بعنوان العنصر الذي ير اد حذفه فتعتبر عملية عكسية لعملية الإضافة .

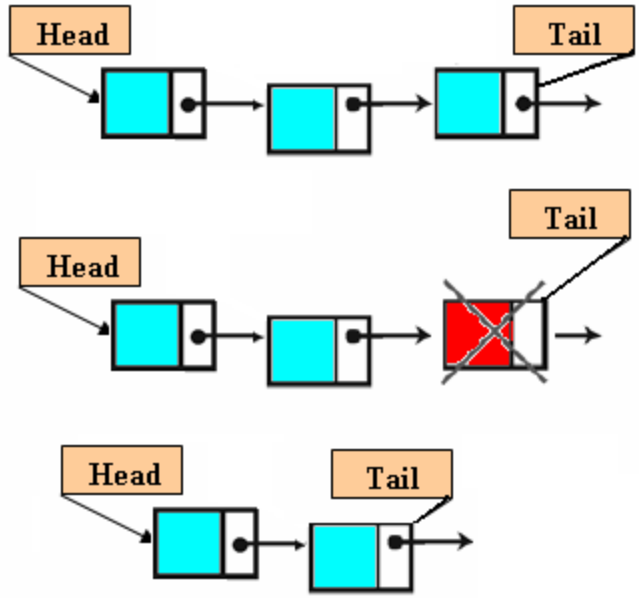
وللحذف عدة أنواع
- الحذف من الرأس
- الحذف من الذيل
- الحذف من أي مكان

وسنرى أول مثال لهذه القوائم وهو الحذف من النهاية للقائمة أي أخر عقدة.
ويتم ذلك جعل العقدة قبل الأخيرة في القائمة مساوية NULL تم ننقل الذيل إلى وراءه بمقدار واحد أي العقدة التي قبل الأخير ثم نحذف العقدة الأخيرة . كما في الشكل 15-18.

وهذا المثال لهذه العملية:

```
1. // برنامج لحذف عقدة من نهاية القائمة الأحادية
2. class Chp15_12 extends  Node{
3. public static void main(String args[]){
4.   Node head=null;                    // head node of the list
5.   Node tail=null;                    // tail node of the list
6.   Node node=null;
7.    int size=5;
8.
9.    tail=head=node=new Node(0);
10.  for (int i=1;i<size;i++)
11.  {
12.   node=new Node(i);
13.   tail.setNext(node);
14.   tail=node;
15.  }
16.
17.  node=head;
```

```
18.
19.  while(node!=null)
20.  {
21.   System.out.print(node.getElement()+" ");
22.   node=node.getNext();
23.  }
24.  System.out.println();
25.
26.  node=head;
27.  while(node!=null)
28.  {
29.     if(node.getNext()==tail)
30.                                  {
31.                                   tail=null;
32.                                   node.setNext(null);
33.                                   tail=node;
34.                                   break;
35.                                  }
36.     node=node.getNext();
37.  }
38.  node=head;
39.  while(node!=null)
40.  {
41.   System.out.print(node.getElement()+" ");
42.   node=node.getNext();
43.  }
44.  System.out.println();
45. }
46. }
```



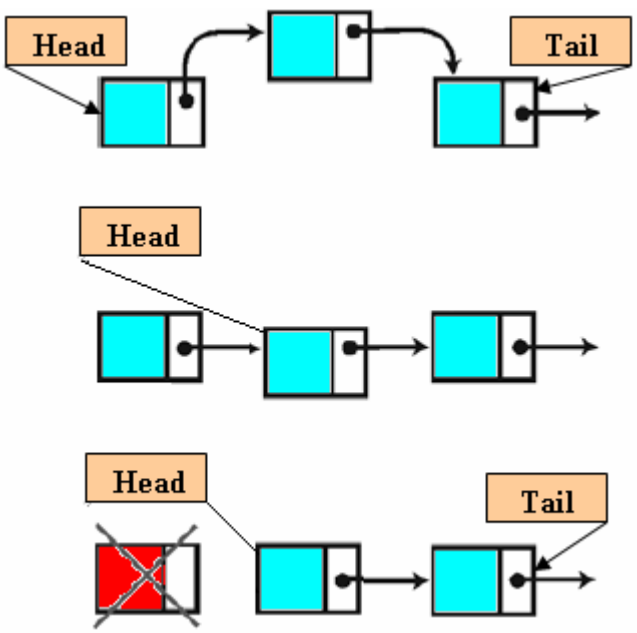
شكل 15-18

واليكم هذا الكود لعملية
الحذف من الإمام أي أول عقدة .
سننقل الرأس إلى الإمام بمقدار واحد أي للعقدة التي بعدها ثم نحذف أول عقدة .
كما في الشكل 15-19.

وهذا المثال لهذه العملية:

```
1. // برنامج لحذف أول عقدو من داخل القائم الأحادية
2. class Chp15_13 extends  Node{
3. public static void main(String args[]){
4.   Node head=null;                    // head node of the list
5.   Node tail=null;                    // tail node of the list
6.   Node node=null;
7.   int size=5;
8.
9.   tail=head=node=new Node(0);
10.  for (int i=1;i<size;i++)
11.  {
12.   node=new Node(i);
13.   tail.setNext(node);
14.   tail=node;
15.  }
16.
17.  node=head;
18.
19.  while(node!=null)
20.  {
```

```
21.   System.out.print(node.getElement()+" ");
22.   node=node.getNext();
23.  }
24.  System.out.println();
25.
26.  node=head;
27.  head=node.getNext();
28.  node=null;
29.
30.  node=head;
31.  while(node!=null)
32.  {
33.   System.out.print(node.getElement()+" ");
34.   node=node.getNext();
35.  }
36.  System.out.println();
37. }
38. }
```



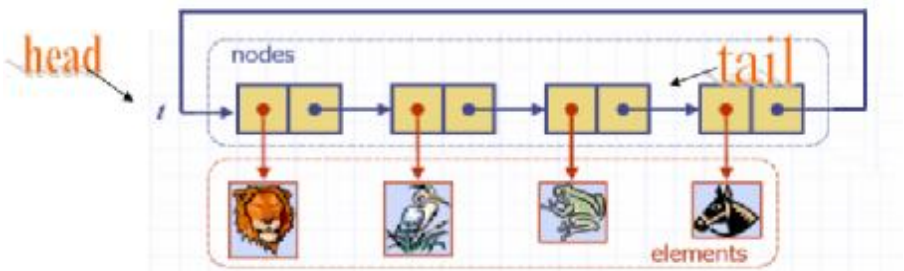
شكل 15-19

وعمليه الحذف من الوسط يتم البحث عن العقدة المراد حذفها تم نغير حقل المؤشر للعقدة الذي قبلها بالعقدة التي بعدها ويتم ذلك بجعل مؤشر يمشي ورآنا بمقدار واحد بـ عملية جعل متغير وهذا المتغير يا خد قيمه العقدة تم ننتقل للعقدة التي بعدها .

15.6.3 القوائم الأحادية المتصلة

حيث يشير مؤشر العقدة الأخيرة إلى العقدة الأولى أي مؤشر الذيل سيؤشر إلى الرأس والشكل 15-20 يوضح ذلك.

وتستعمل هده القوائم المتصلة كثيراً في أنظمة إدارة بنوك المعطيات وفي البرمجة إذ تسمح بربط العناصر التي تتمتع بنفس الخصائص فيما بينها .



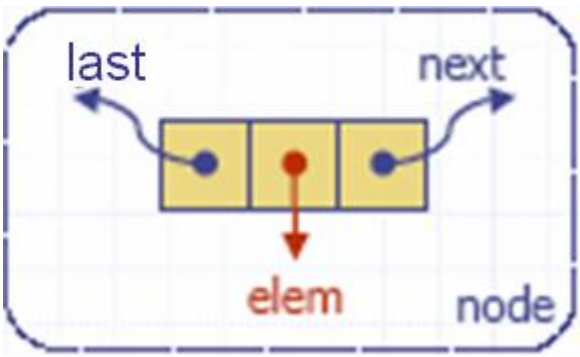
شكل 20-15

وعندما تريد تحويل القائمة الدائرية إلى قائمة الخطية نجعل مؤشر أي عقدة في القائمة مساوياً إلى (NULL) فتتحول إلى قائمة متصلة .

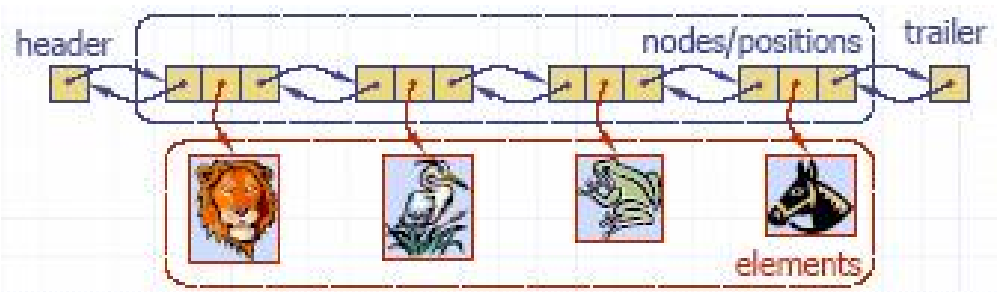
15.6.4 القوائم المذبلة الثنائية

تعتبر القوائم الثنائية قوائم أحاديه ولكن ليس العكس حيث أن القوائم المذبلة لها مؤشرين مؤشر يؤشر إلى العقدة التالية ويسمى next ومؤشر يشير إلى العقدة السابقة يسمى last .

وتستعمل هذه القوائم عندما نحتاج للرجوع إلى وراء لجلب معلومات معينة ولنتذكر برنامج معالجة النصوص حيث أنة يستطيع العودة إلى الورى لتعديل حرف مثلاً.
ويكون الهيكل العام لها كما في الشكل 15-21 والشكل 15-22 يبين الشكل العام للقوائم المذبلة.



شكل 15-21



شكل 15-22

ومن الشكل 15-22 يتضح لنا شكل هذه القوائم . وسنقوم الآن بإنشاء صنف يمثل القوائم المذبلة:

```
1. /**
2.  * Class binary tree by storing references to
3.  * an element, a parent node, a left node, and a right node.
4.  */
5. public class BTNode{
```

```
6.  private int element;                         // element stored at this node
7.  private BTNode left, right;  // adjacent nodes
8.  /** Main constructor */
9.  public BTNode(){}
10.
11.   public BTNode(int element) {
12.     setElement(element);
13.     setLeft(null);
14.     setRight(null);
15.                                                     }
16.  /** Returns the element stored at this position */
17.  public int element() { return element; }
18.  /** Sets the element stored at this position */
19.  public void setElement(int o) { element=o; }
20.  /** Returns the left child of this position */
21.  public BTNode getLeft() { return left; }
22.  /** Sets the left child of this position */
23.  public void setLeft(BTNode v) { left=v; }
24.  /** Returns the right child of this position */
25.  public BTNode getRight() { return right; }
26.  /** Sets the right child of this position */
27.  public void setRight(BTNode v) { right=v; }
28. }
```

شرح المثال:

جميع المناهج في الصنف DNODE هي نفسها في صنف NODE في القوائم الأحادية.
الاختلاف فقط هو المتحول last وهو نفس المتحول next الذي يؤشر إلى العقدة السابقة.

نفس العمليات التي طبقت على القوائم الأحادية ستطبق على القوائم المذبلة.

15.6.5 العمليات على القوائم المذبلة

ونبدأ بأول عمليه ألا وهي عملية الإضافة من اليمين.
وهذا المثال لهذه العملية :

```
1. // برنامج الأضافة من اليمين للقائمة الثنائية
2. class Chp15_14 extends DNODE {
3. public static void main(String args[]){
4.   DNODE head=null;              // head DNODE of the list
5.   DNODE tail=null;              // tail DNODE of the list
6.   DNODE node=null;
7.       int size=5;
8.
9.       tail=head=node=new DNODE(0);
10.      for (int i=1;i<size;i++)
```

```
11.    {
12.     node=new DNODE(i);
13.     tail.setNext(node);
14.     node.setPrev(tail);
15.     tail=node;
16.    }
17.
18.    node=head;
19.
20.    while(node!=null)
21.    {
22.     System.out.print(node.getElement()+" ");
23.     node=node.getNext();
24.    }
25.    System.out.println();
26.
27.    node=tail;
28.    while(node!=null)
29.    {
30.     System.out.print(node.getElement()+" ");
31.     node=node.getPrev();
32.    }
33.    System.out.println();
34. }
35. }
```

شرح المثال:

الأسطر (4-5) عرفنا head من نوع DNODE مؤشر لسجل وهو الرأس و tail من نوع DNODE مؤشر لسجل وهو الذيل و node سنستخدمه كمتغير لإدخال بيانات العقد .

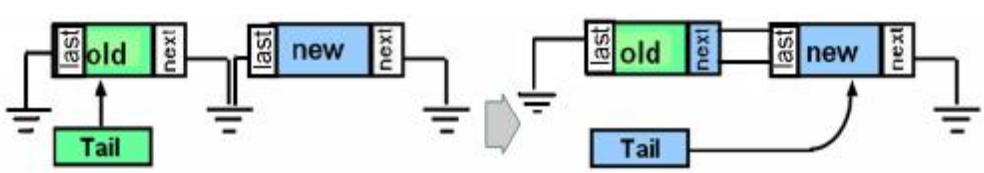
السطر 9 خطوة ضرورية ولابد أن تكون منفردة عن أخواتها لكي نساوي الرأس والذيل بأول عقدة .

الأسطر (16 - 10) سنكون 4 عقد أضافية بجانب الأولى فيكون لدينا 5 عقد .

السطر 13 تحويل حقل الذيل من null إلى عنوان العقدة الجديدة.

السطر 14 حقل العقدة الثانية يساوي العقدة القديمة إي الذيل.

فبهذه الحالة تمت عملية الربط بين العقدتين بقي علينا نقل الذيل إلى العقدة الجديدة tail=node كما في السطر 15 وهكذا بباقي العقد إلى أن ينتهي عمل اللوب ويمكنك تكوين مئات العقد بهذه الطريقة والشكل 23-15 يبن الشرح .



شكل 15-23

أم الإضافة من اليسار نفس السابق إلا أن الاختلاف فقط بعملية إدخال العقد الثانية وما بعدها كهذه الشفرة :

```
for (int i=1;i<size;i++)
{
 node=new DNODE(i);
 node.setNext(head);
 head.setPrev(node);
 head=node;
}
```

الإضافة من اليمين يكون الرأس متحركا والإضافة من اليمن يكون الذيل هو المتحرك.

ونفس البرامج التي ذكرناها في القوائم الأحادية تطبق على القوائم الثنائية فلاختلاف فقط هو زيادة المؤشر الخلفي وربطة بالعقدة الجديدة.

وهذا المثال لعملية ترتيب قائمة ثنائية

```
1. // برنامج ترتيب لقائمة الثنائية
2. class Chp15_15 extends DNODE {
3. public static void main(String args[]){
4.   DNODE head=null;                  // head DNODE of the list
5.   DNODE tail=null;                  // tail DNODE of the list
6.   DNODE node=null;
7.       int size=5;
8.
9.       tail=head=node=new DNODE(0);
10.      for (int i=1;i<size;i++)
11.      {
12.       node=new DNODE(i);
13.       tail.setNext(node);
14.       node.setPrev(tail);
15.       tail=node;
16.      }
17.
18.      node=head;
19.
20.      while(node!=null)
```

```
21.    {
22.     System.out.print(node.getElement()+" ");
23.     node=node.getNext();
24.    }
25.    System.out.println();
26.
27.    DNODE temp,temp2;
28.
29.    for(temp=head;temp!=null;temp=temp.getNext())
30.      for(temp2=head;temp2!=null;temp2=temp2.getNext())
31.                  if(temp.getElement()>temp2.getElement())
32.                              {
33.                               int j;
34.                               j=temp2.getElement();
35.                               temp2.setElement(temp.getElement());
36.                               temp.setElement(j);
37.                              }
38.
39.    node=head;
40.
41.    while(node!=null)
42.    {
43.     System.out.print(node.getElement()+" ");
44.     node=node.getNext();
45.    }
46.    System.out.println();
47. }
48. }
```

شرح المثال:

السطر 27 عرفنا متغيرين من نوع DNODE وتعاملنا بعملية الترتيب كترتيب مصفوفة وهذه الخوارزمية معروفة ولا جديد فيها.

ينبغي على القارئ حل هذا المثال بدون أن ينضر للكود المكتوب

- اكتب برنامج يعمل على إدخال الأعداد الفردية من اليسار و الزوجية من اليمين؟

هل اكتشفت فكرة البرنامج فهي سهلة جداً ولا تحتاج إلى جهد وضياع للوقت !

إن لم تتضح لك الفكرة يا عزيزي فصلي على معدن الأسرار ومنبع الأذوار سيدنا محمد وعلى آله وصحبة الأطهار وتتبع هذا الكود.

```
1. /*برنامج الأضافة من اليمين للقائمة الثنائية
2. الأعداد الزوجية ومن اليسار الأعداد الفردية
3. */
4. class Chp15_16 extends DNODE {
5. public static void main(String args[]){
6.   DNODE head=null;              // head DNODE of the list
7.   DNODE tail=null;              // tail DNODE of the list
```

```
8.  DNODE node=null;
9.      int size=5;
10.
11.     tail=head=node=new DNODE(0);
12.     for (int i=1;i<size;i++)
13.     {
14.      if(i%2==0){//if number evn
15.                             node=new DNODE(i);
16.                             tail.setNext(node);
17.                             node.setPrev(tail);
18.                             tail=node;
19.                     }
20.                     else
21.                     {//if number add
22.                             node=new DNODE(i);
23.                             node.setNext(head);
24.                             head.setPrev(node);
25.                             head=node;
26.                     }
27.     }
28.     //print DNODE
29.     node=head;
30.     while(node!=null)
31.     {
32.      System.out.print(node.getElement()+" ");
33.      node=node.getNext();
34.     }
35.     System.out.println();
36.
37. }
38. }
```

شرح المثال:

فكرة البرنامج هي بعد إنشاء أول عقدة يتم إنشاء ثاني عقدة ويتم تفحص القيمة فإذا كانت زوجيه فان الإضافة ستكون من اليمين وإلا ستكون الإضافة من اليسار. كما في الأسطر (26 - 14).

عمليات الحذف

كما نفذت عملية الحذف في اللوائح الأحادية، هي نفسها تنفذ في اللوائح الثنائية .

الشيء الذي نريد توضيحه هو عندما يراد منك حذف عقدة من أي مكان مع الاحتفاظ برأس اللائحة وذيل اللائحة .

هذا مثال يعمل على حذف إي عقدة بالقائمة

```
1. // برنامج حذف عقدة من القائمة الثنائية
```

```
2. import javax.swing.JOptionPane;
3. class Chp15_17 extends DNODE {
4. public static void main(String args[]){
5.   DNODE head=null;                  // head DNODE of the list
6.   DNODE tail=null;                  // tail DNODE of the list
7.   DNODE node=null;
8.       int size=5;
9.
10.     tail=head=node=new DNODE(0);
11.     for (int i=1;i<size;i++)
12.     {
13.      node=new DNODE(i);
14.      tail.setNext(node);
15.      node.setPrev(tail);
16.      tail=node;
17.     }
18.
19.     //print DNODE
20.     node=head;
21.     while(node!=null)
22.     {
23.      System.out.print(node.getElement()+" ");
24.      node=node.getNext();
25.     }
26.     System.out.println();
27.
28.     String snum1;
29.     int num1;boolean flag=false;
30.     snum1 = JOptionPane.showInputDialog("Enter Number Delete:") ;
31.     num1 = Integer.parseInt(snum1);
32.     node=head;
33.     while(node!=null)
34.     {
35.       if(node.getElement()==num1)
36.                                       {
37.                                        if(node==head)
38.                                                                {
39.                                                                 node.getNext().setPrev(null);
40.                                                                 head=node.getNext();
41.                                                                 node = null;
42.                                                                }
43.                                                                else
44.                                        if(node==tail)
45.                                                                {
46.                                                                 node.getPrev().setNext(null);
47.                                                                 tail=node.getPrev();
48.                                                                 node = null;
49.                                                                }
50.                                                                else
51.                                                                {
52.                                       node.getPrev().setNext(node.getNext());
```

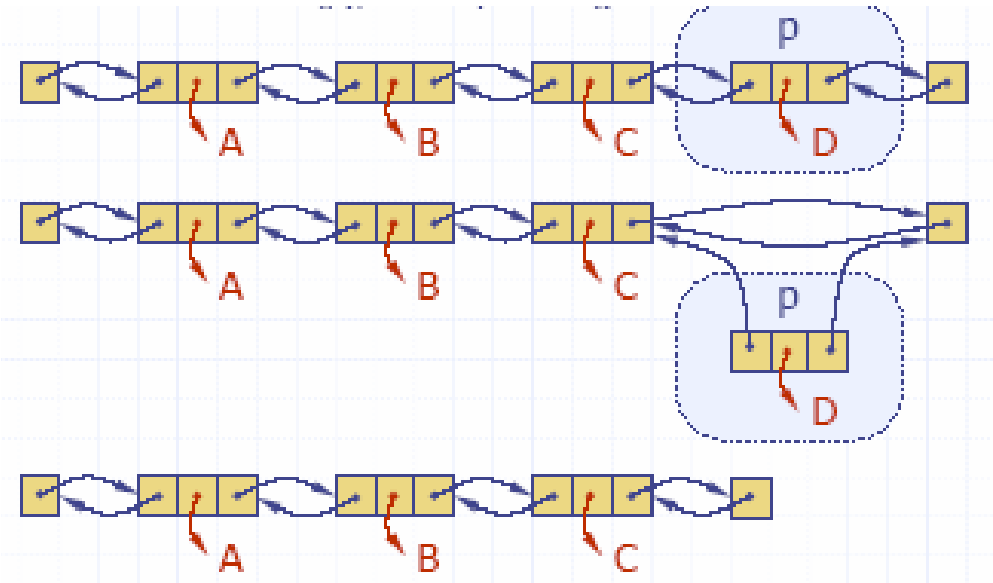
```
53.                                     node.getNext().getNext().setPrev(node.getPrev());
54.                                                         node= null;
55.                                                        }
56.                                      flag=true;
57.                                      System.out.println("The Found Nuber And Deleted");
58.                                      break;
59.                                     }
60.       node=node.getNext();
61.     }
62.
63.     if(!flag)System.out.println("Not Found Nuber");
64.
65.     //print DNODE
66.     node=head;
67.     while(node!=null)
68.     {
69.      System.out.print(node.getElement()+" ");
70.      node=node.getNext();
71.     }
72.     System.out.println();
73.     System.exit(0);
74. }
75. }
```

شرح المثال:

في السطر 37 نستفسر إذا كانت العقدة هي الرأس فإنها حالة خاصة إي الحذف من البداية وسبق وان تكلمنا عن هذه الحالة وفائدة هذا الشرط هو الحفاظ على مكان الرأس وهو نقلة بمقدار واحد للإمام و بعد ذلك حذف العقدة.

وفي السطر44 نستفسر إذا كانت العقدة هي الذيل فإنها حالة خاصة أيضاً إي الحذف من النهاية وسبق وان تكلمنا عن هذه الحالة وفائدة هذا الشرط هو الحفاظ على مكان الذيل وهو نقلة بمقدار واحد للخلف و بعد ذلك حذف العقدة.

وإلا ستكون العقدة بين الرأس والذيل فإنها حالة خاصة أيضاً ، فيتم ربط مؤشر العقدة السابقة مع العقدة التالية وربط مؤشر العقدة التالية مع العقدة السابقة كـ ما في الأسطر (51 - 55) والشكل 24-15 يبن ذلك .



لشكل 24-15

دمج القوائم الثنائية:
بعمل هذا المثال على دمج قائمتين ثنائيتين:

```
1. //برنامج دمج قائمتين ثنائيتين
2. class Chp15_18 extends DNODE {
3. public static void main(String args[]){
4.   DNODE head=null;                  // head DNODE of the list
5.   DNODE tail=null;                  // tail DNODE of the list
6.   DNODE node=null;
7.
8.   DNODE head2=null;   // head2 DNODE of the list
9.   DNODE tail2=null;                 // tail2 DNODE of the list
10.   DNODE node2=null;
11.     int size=5;
12.     //insert DList1
13.     tail=head=node=new DNODE(0);
14.     for (int i=1;i<size;i++)
15.     {
16.      node=new DNODE(i);
17.      tail.setNext(node);
18.      node.setPrev(tail);
19.      tail=node;
20.     }
```

```
21.     //insert DList2
22.     tail2=head2=node2=new DNODE(10);
23.     for (int i=11;i<size+10;i++)
24.     {
25.      node2=new DNODE(i);
26.      tail2.setNext(node2);
27.      node2.setPrev(tail2);
28.      tail2=node2;
29.     }
30.
31.     //Print DList1
32.     node=head;
33.     while(node!=null)
34.     {
35.      System.out.print(node.getElement()+" ");
36.      node=node.getNext();
37.     }
38.     System.out.println("\nDList1");
39.
40.     //Print DList2
41.     node2=head2;
42.     while(node2!=null)
43.     {
44.      System.out.print(node2.getElement()+" ");
45.      node2=node2.getNext();
46.     }
47.     System.out.println("\nDList2");
48.
49.     //node+node2
50.     tail.setNext(head2);
51.     head2.setPrev(tail);
52.     tail=tail2;
53.     tail2=head2=node2=null;
54.
55.     //Print DList1 + DList2
56.     node=head;
57.     while(node!=null)
58.     {
59.      System.out.print(node.getElement()+" ");
60.      node=node.getNext();
61.     }
62.     System.out.println("\nDList1 + DList2");
63.
64. }
65. }
```

شرح المثال:

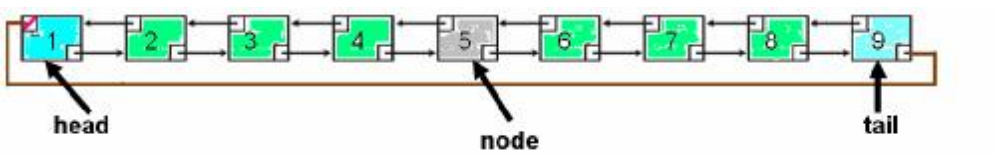
الأسطر (53 - 50) تمت دمج اللائحتين مع بعضها البعض عن طريق جعل ذيل اللائحة الأولى يؤشر إلى رأس اللائحة الثانية كما في السطر 50. و جعل رأس اللائحة الثانية يؤشر

إلى ذيل اللائحة الأولى كما في السطر 51 . وأخر عملية هي مساواة مؤشر اللائحة الأولى بمؤشر اللائحة الثانية كما في السطر 52.
السطر 53 يعمل على حذف المؤشرات الزائدة من عملية الدمج.

إلى هنا قد اتضحت عمل القوائم المذبلة وهذه الأمثلة التي كتبت إذا فهمها القارئ فأننا نضمن له أن إي سؤال سيواجه سيعرف إجابته بلا تعب أو مجهود.

مثال مهم :

لنفترض أن لدينا لائحة ثنائية د ائرية بداخلها هذه كما في الشكل 15-25



شكل 15-25

ولدينا ثلاثة مؤشرات node , tail , head
وكان أمر الطباعة

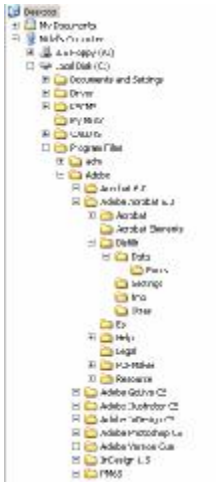
```
System.out.println(node.getElement());

System.out.println(node.getNext().getNext().getElement());

System.out.println(node.getNext().getPrev().getElement());

System.out.println(node.getNext().getNext().getNext().getNext().getNext().getNext().g
etElement());
System.out.println(head.getElement());

System.out.println(head.getNext().getElement());

System.out.println(head.getPrev().getPrev().getElement());

System.out.println(tail.getPrev().getPrev().getNext().getElement());

System.out.println(tail.getNext().getPrev().getElement());
```

فما هو ناتج تنفيذ البرنامج؟

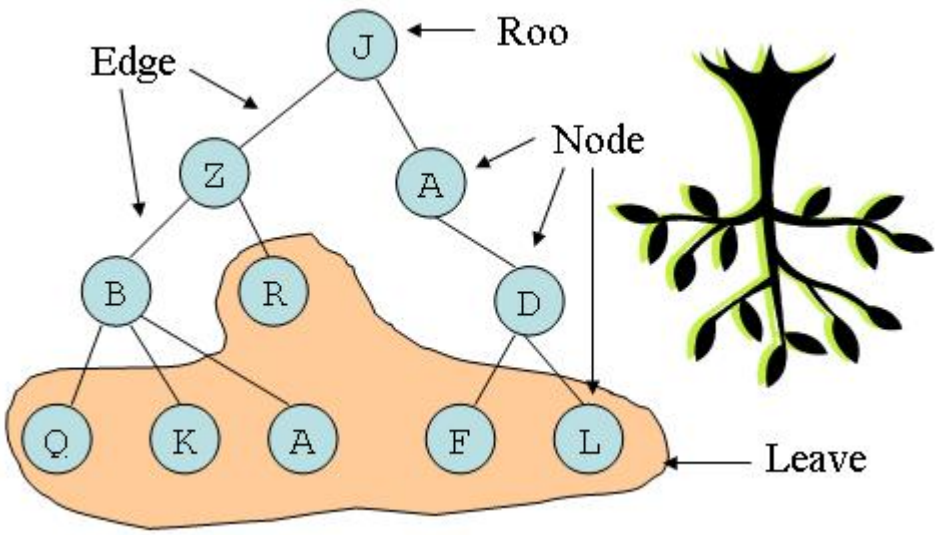
بإمكاننا أن نقدر ثمن خوارزم معالجة اللوائح بواسطة :

- الحجم المشغول في الذاكرة .
- عدد المؤشرات التي من الواجب عبورها أو استعمالها.

15.7 الأشجار (Trees)



الشجرة هي مكان غير خطي لتخزين المعلومات ، وتستخدم الأشجار لتمثيل المعلومات التي لها علاقات تشعبيه بحيث أن كل عنصر "معلومة" في الشجرة له أب واحد وقد يكون له صفر أو أكثر من الأبناء.
الجذر Root هو العنصر الوحيد الذي لا يوجد له أب كما في الشكل 26-15.



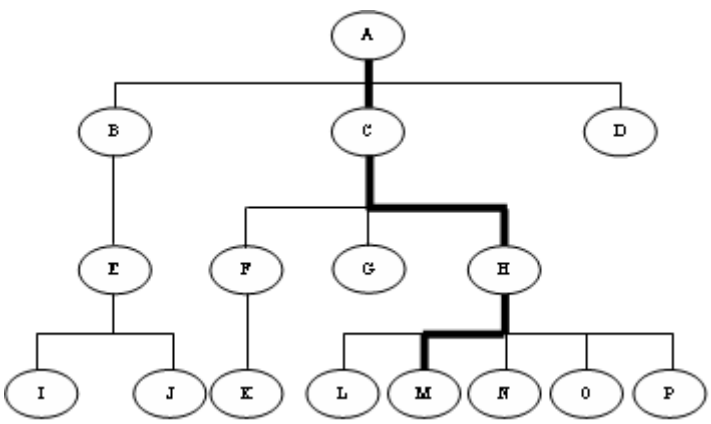
شكل 26-15

15.7.1 مصطلحات الأشجار

عناصر الأشجار تسمى خلايا Nodes ، وكل خلية لها مسار Path واحد فقط يوصلها بالجذر Root.

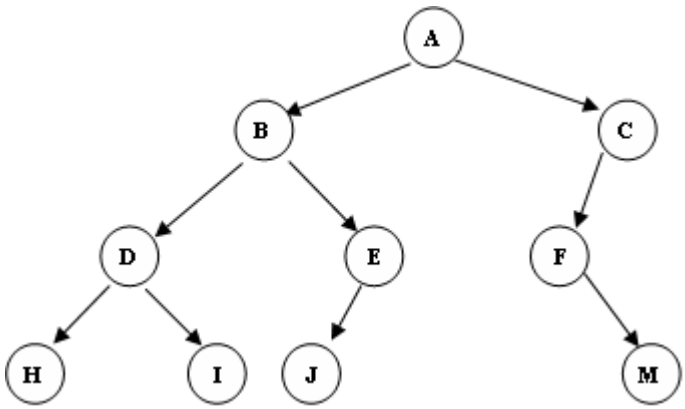
والمسار هو عبارة عن مجموعة خلايا متتابعة للوصول إلى خلية معينة. طول المسار Path Length هو عبارة عن عدد الوصلات من الجذر إلى الخلية المراد معرفة طول مسارها والذي يساوي عدد الخلايا ناقص واحد.
في الشجرة التالية المسار (M,H,C,A) يوصل الخلية M بالجذر A طوله 3.

عمق Depth الخلية هو طول مسارها إلى الجذر ، مثلاً الخلية E عمقها 2. الجذر A عمقه 0.
المستوى Level هو كل الخلايا التي لها نفس العمق. المستوى الثاني عبارة عن {E,F,G,H}.
إرتفاع Height الشجرة هو أكبر عمق موجود للشجرة، وفي الشجرة السابقة يساوي 3.
الشجرة التي يوجد بها خلية واحدة فقط إرتفاعها يساوي 0. والشجرة التي لا تحتوي على أي خلية يعرف إرتفاعها (-1).
درجة الخلية Degree هو عدد أبنائها الخلية H درجتها 5.
الورقة Leaf هي الخلية التي درجتها صفر أي لايوجد لها أبناء.
ومن الشكل 27-15 يتضح جميع ما سبق.



شكل 27-15

15.7.2 الأشجار الثنائية (Binary Trees)



شكل 15-28

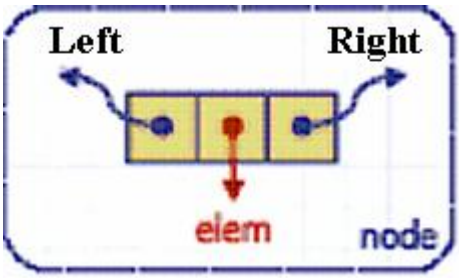
الأشجار الثنائية هي أشجار كسابقتها في التعريف إلا أن عدد الأبناء لأي خلية لا يتجاوز الاثنان

طرق تمثيل الشجرة الثنائية

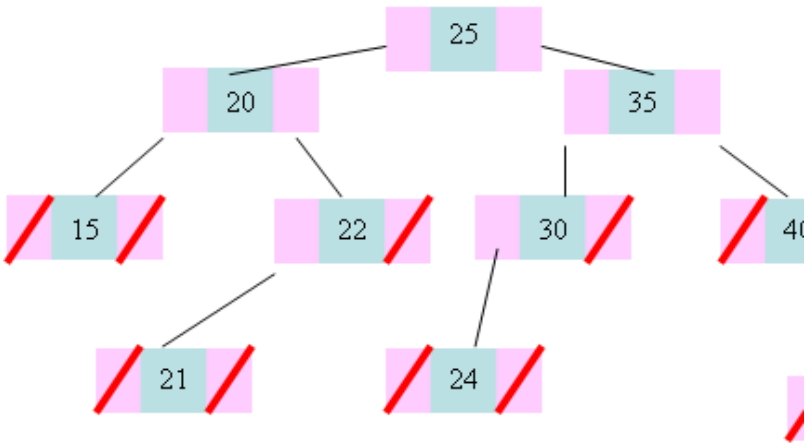
أن الشجرة تمثل بعدة طرق بواسطة القوائم، وتمثل بالمتجهات وسندرس فيما يلي هذه الطرق:

- إذا كانت الشجرة ثنائية منتظمة بعمق (H) فإن عدد عناصر هذه الشجرة يساوي (2^(H+1)-1) وعلية فإنها تمثل بمصفوفة أحادية البعد و عدد عناصر المتجه الذي يمثل الشجرة يساوي (2^(H+1)-1) . ومميزات هذه الطريقة
  - السهولة فإذا أعطيت موقع العقدة الابن فمن السهل تحديد موقع الأب بالنسبة لها. فلو كانت العقدة الابن في الموقع n من المصفوفة فإن موقع الأب يكون صحيحاً(n/2).
  - تطبق بسهولة في لغات البرمجة ، مثل بيسك وفورتران حيث تكون مواقع الذاكرة الثابتة متوفرة مباشرة .

  وعيوب هذه الطريقة
  - عملية الإضافة والحذف تؤدي إلى تحريك البيانات إلى أعلى وأسفل في المصفوفة وهذا يضيع وقت المعالجة .
  - تكون هنالك مواقع ذاكرة غير مستغلة
- **تمثيل الشجرة بواسطة القوائم :**



شكل 15-29



شكل 15-30

ونلاحظ من الشكل السابق أن كلما اتجهنا إلى اليمين يكون العدد اكبر من السابق وكل ما اتجهنا إلى اليسار كان العدد اصغر من الأب أو العقدة السابقة وهكذا تكون الأعداد مرتبة تصاعدياً وتنازلياً .

ونكرر أن الأعداد لا تتكرر في الشجرة الثنائية.

ونلاحظ أن من عيوب هذه الطريقة

- تحتوي على فراغ في فضاء الذاكرة غير مستعمل نتيجة لاستخدام مؤشر صفرية NULL.
- خوارزمية التطبيق لها أصعب اللغات في اللغات التي لا تعطي تقنية ذاكرة متحركة (ديناميكية) .

15.7.3 تطبيقات الأشجار الثنائية

i. شجرة Huffman لضغط البيانات

لشرح كيفية عمل طريقة Huffman نفرض انه يوجد لدينا ملف وحجمه byte1000 يحتوي على الحروف (a,b,c,d,e,f,g,i,j) .

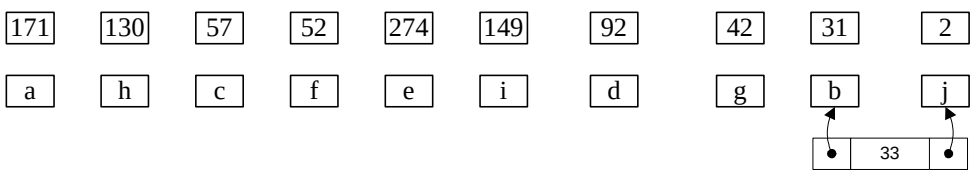
1- نقوم بعمل إحصائية عن الملف المراد تقليص حجمه وذلك بعدّ تكرار كل حرف.

جدول 15-2

a	b	c	d	e	f	g	h	i	J
171	31	57	92	274	52	42	130	149	2

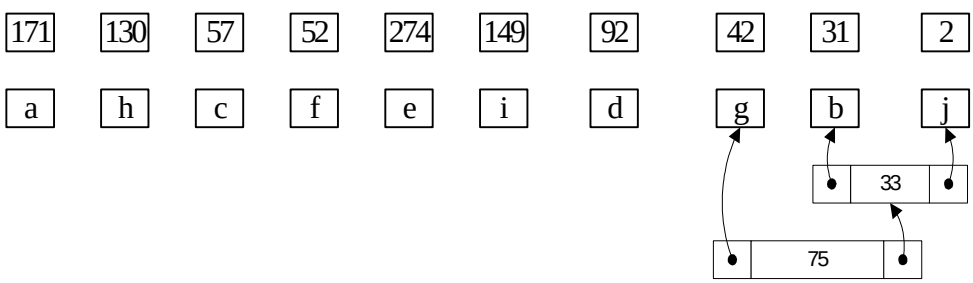
2- نقوم ببناء Binary Tree وذلك عن طريق اختيار الحروف ذات الأقل تكرار وتوصيلها ببعض.

في هذا المثال حرف j وحرف b يمثلان أقل تكرار ، نقوم بتوصيلهما مع وضع مجموع تكرارهما كما في شكل 15-31.



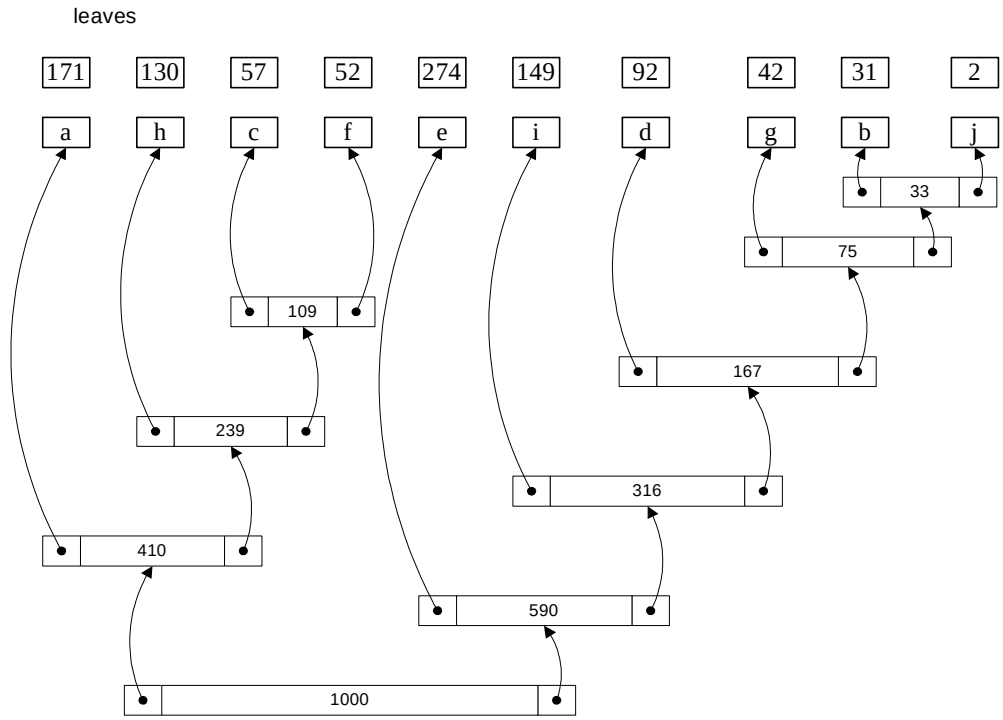
شكل 15-31

الآن يمثل مجموع حرفي b,j وحرف g الأقل تكرارا نقوم بتوصيلهما بنفس الخطوات السابقة



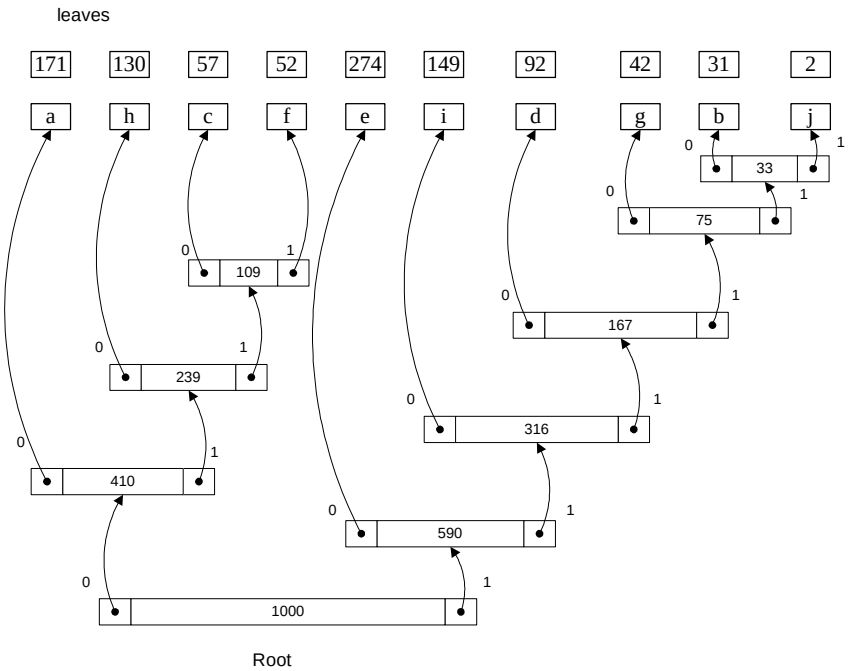
شكل 15-29

عند من توصيل جميع الحروف تكون Binary Tree كالشكل 32-15:



شكل 33-15

3- نقوم بتوزيع 1 على كل فرع من الفروع اليمنى ، و 0 على الفروع اليسرى.



شكل 15-33

4- نقوم بتسجيل المسارات التي توصل إلى كل حرف من الحروف الأصلية للملف ، وذلك بتتبع مسارات Binary Tree من الجذر Root إلى الأوراق Leaves.

جدول 15-3

a	b	c	d	e	f	g	h	i	J
00	111110	0110	1110	10	0111	11110	010	110	111111

عند تكوين الملف المضغوط نستبدل الحروف الأصلية بمساراتها المحسوبة في الخطوة رقم 4.

ونستطيع حساب حجم الملف الجديد ونسبة تقليصه بضرب طول مسار كل حرف في تكراره في الملف

جدول 15-4

a	b	c	d	e	f	g	h	i	j	
00	111110	0110	1110	10	0111	11110	010	110	111111	
2	7	4	4	2	4	5	3	3	6	
171	31	57	92	274	52	42	130	149	2	
342	217	228	368	548	208	210	390	447	12	2970

أي أن الملف الجديد سيكون حجمه 2970/8= 372 بايت ، بمعنى أن الحجم الجديد يعادل 37.2% من الحجم الأصلي.
فكرة عمل هذه الطريقة تتلخص في إنها تستغل الحروف الموجودة بكثرة في الملف وتضعها في أقصر مسار في Binary Tree مما يعني إنها (الحروف) سيتم أعطائها أقل حيز تخزيني ممكن.

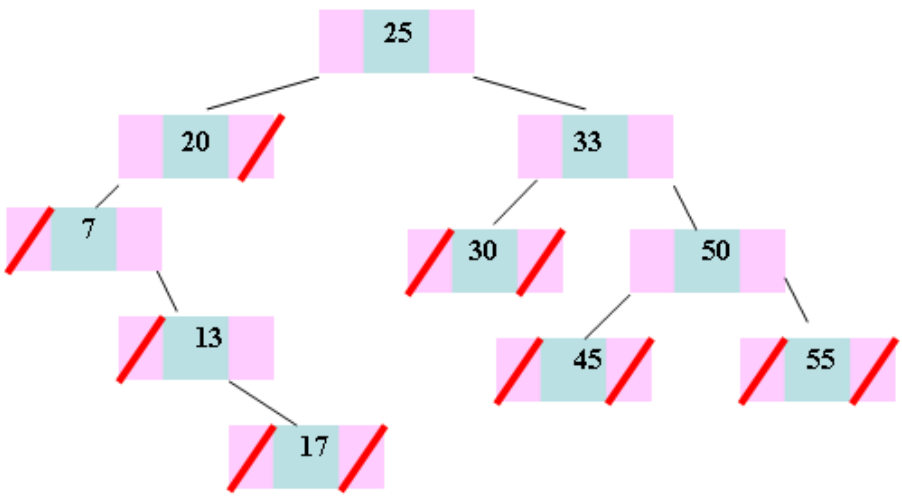
15.7.4 خوارزمية بناء الشجرة الثنائية

أن بناء الشجرة الثنائية يعتمد على طريقة عبورها، وسابقاً قد سقنا عدة طرق لعبور الشجرة بالاعتماد على المؤشرات وفي الخوارزميات التالية جميعها سنفترض الشجرة الثنائية ذات مؤشر الإباء بالأبناء (Father Link) . وعملية إضافة عقدة في الشجرة الثنائية تعتمد على قيمة تلك العقدة ، فإن كانت القيمة اكبر من الجذر اتجهنا إلى اليمين والعكس نتجه إلى اليسار

وخلاصة هذه الخوارزمية تتلخص بالأتي:

- ý ضع العنصر الأول على أساس أنة العقدة الأول في الهيكل (الجذر) .
- ý والعدد التالي إذا كان العنصر المراد إدخاله اكبر من الجذر سنضعه على يمين الجذر .
- ý وإلا على يسار الجذر .

والشكل 34-15 يبين ذلك . فإن أدخلنا هذه القيم (25,20,7,13,33,50,45,17,30,55) ستكون الشجرة بهذا الشكل



شكل 34-15

لـ

قد بدئنا بالجدر (25) ثم انتقلنا إلى اليسار بالعدد 20 لأنة اصغر من 25 ثم انتقلنا إلى يسار 25 و 20 بالعدد 7 لأنة أصغر من 20 ثم انتقلنا بالعدد 13 إلى اليسار من 25 وهكذا إلى نهاية الأعداد .

ونعرض الآن صنف الأشجار الثنائية :

```
1. /**
2.  * Class binary tree by storing references to
3.  * an element, a parent node, a left node, and a right node.
4.  */
5. public class BTNode{
6.   private int element;                                    // element stored at this node
7.   private BTNode left, right;  // adjacent nodes
8.   /** Main constructor */
9.   public BTNode(){}
10.
11.   public BTNode(int element) {
12.     setElement(element);
13.     setLeft(null);
14.     setRight(null);
15.                                                                    }
16.   /** Returns the element stored at this position */
17.   public int element() { return element; }
18.   /** Sets the element stored at this position */
19.   public void setElement(int o) { element=o; }
20.   /** Returns the left child of this position */
21.   public BTNode getLeft() { return left; }
22.   /** Sets the left child of this position */
23.   public void setLeft(BTNode v) { left=v; }
24.   /** Returns the right child of this position */
25.   public BTNode getRight() { return right; }
26.   /** Sets the right child of this position */
27.   public void setRight(BTNode v) { right=v; }
28. }
```

وفيما يلي البرنامج الذي ينفذ جميع ما سبق :

```
1. //برنامج الشجرة الثنائية
2. class Chp15_19 extends BTNode {
3. public static void main(String args[]){
4.   BTNode root=null;                          // root BTNode of the BTree
5.   BTNode right=null;              // right BTNode of the BTree
6.   BTNode left=null;                          // left BTNode of the BTree
7.   BTNode node=null;
8.
9.       int Arr[]={5,6,2,8,4,10,18,9,0};
10.     //insert BTree
11.     for (int i=0;i<Arr.length;i++)
12.     {
13.      if(root==null)
14.                                {
15.                                        root=node=new BTNode(Arr[0]);
16.                                }
17.                                else
18.                                {
```

```
19.                                     node=new BTNode(Arr[i]);
20.                                     BTNode s,p;
21.                                     p=s=root;
22.                                     while(s!=null)
23.                                                     {
24.                                                       p=s;
25.                                                       if(node.element()>s.element())
26.                                                               s=s.getRight();
27.                                                               else
28.                                                               s=s.getLeft();
29.                                                     }
30.                                     if(node.element()>p.element())
31.                                                               p.setRight(node);
32.                                                               else
33.                                                               p.setLeft(node);
34.
35.                             }
36.     }
37.
38.     //Print DBTree
39.     Print(root);
40.     System.out.println("\nDBTree");
41. }
42. static void Print(BTNode node){
43.     if(node!=null){
44.                                     System.out.print(node.element()+" ");
45.                                     Print(node.getLeft());
46.                                     Print(node.getRight());
47.                             }
48.                                                       }
49. }
```

شرح المثال:

الأسطر (18 - 35) قمنا بعمليه البحث عن الموقع التي سنضع العقدة حسب خوارزمية الأشجار فإذا كانت القيمة اكبر اتجهنا يساراً وإلا اتجهنا يميناً إلى أن يصل s = null كما في السطر 22.

فائدة المتحول p هو عبارة مؤشر يؤشر بمقدار واحد للخلف لنحتفظ بموقع أخر عقدة الناتجة من عملية البحث .

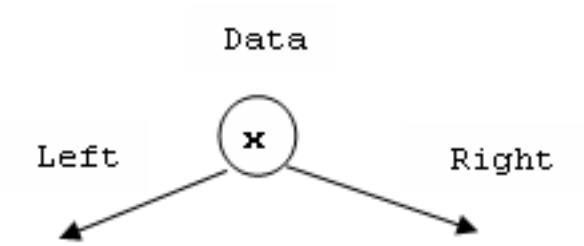
بعد من بحث الموقع نقوم بعمليه استفسار فإذا كانت القيمة اكبر من العقدة التي عثرنا عليها بواسطة p فأننا نضع العقدة على اليمين وإلا نضعها على اليسار وهكذا لباقي العقد كما في الأسطر (30 – 34) .

السطر 39 تم استدعاء منهج Print ليقوم بطباعة الشجرة . و هنا تكون عملية الطباعة بالاستدعاء الذاتي فهي أسهل .

15.7.5 خوارزم إسترجاع المعلومات من الأشجار الثنائية

Binary Tree Traversal

والمقصود هنا هو زيارة كل خلية واسترجاع معلوماتها " للطباعة مثلاً" مرة واحد فقط.
ولنفرض أن تمثيل الشجرة الثنائية كما في الشكل 35-15 :



شكل 35-15

فمن الشكل السابق يمكننا استرجاع البيانات من الشجرة الثنائية بست طرق وهي :

Left Right Data
Left Data Right
Right Left Data
Right Data Left

Data Left Right

Data Right Left

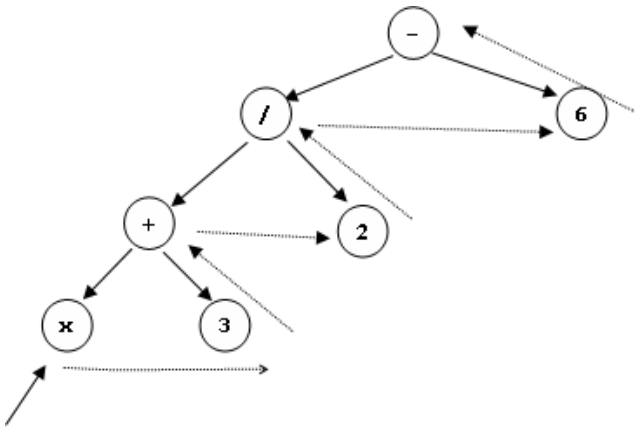
وباستبعاد الطرق التي توجد بها Right قبل Left يتبقى لنا ثلاث طرق وهي :

الأولى وتسمى Post Order وتمثل Left Right Data
الثانية وتسمى Pre Order وتمثل Data Left Right
والثالثة وتمسى In Order وتمثل Left Data Right

وسميت هذه الطرق نسبة إلى موقع الـ Data ، ففي الطريقة الأولى تقع Data الأخيرة ...وهكذا..

1. طريقة Post Order .

```
static void PostOrder(Node ptr)
{
  if(ptr!=null){
    PostOrder(ptr.getLeft());
    PostOrder(ptr.getRight());
    System.out.print(ptr.element()+" ");
        }
}
```



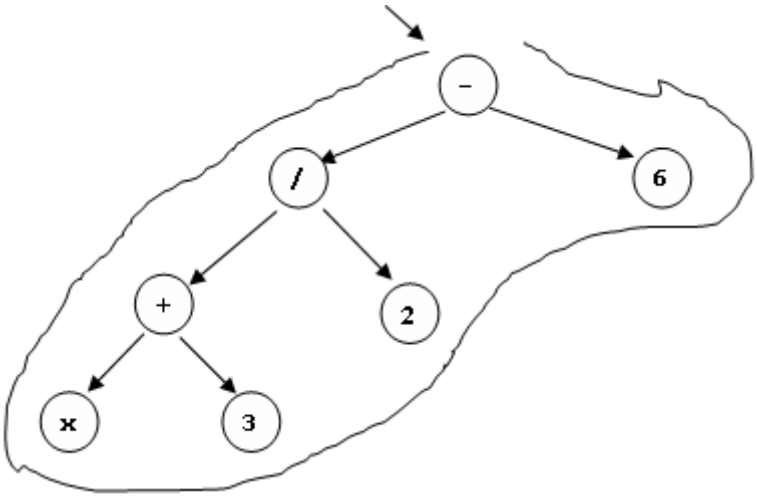
شكل 36-15

بالنظر إلى عمل هذه الطريقة نجدها تسترجع البيانات من المستويات الأعمق أولاً ، ثم التي تليها وهكذا، مع أولوية الجهة اليسرى للشجرة.

2. طريقة Pre Order

```
static void PreOrder(Node ptr)
{
  if(ptr!=null){
    System.out.print(ptr.element()+" ");
    PreOrder(ptr.getLeft());
    PreOrder(ptr.getRight());
          }
}
```

يمكن إسترجاع البيانات بالرسم وذلك برسم محيط حول الشجرة بادية من الجذر ، مع أولوية الجهة اليسرى للشجرة.

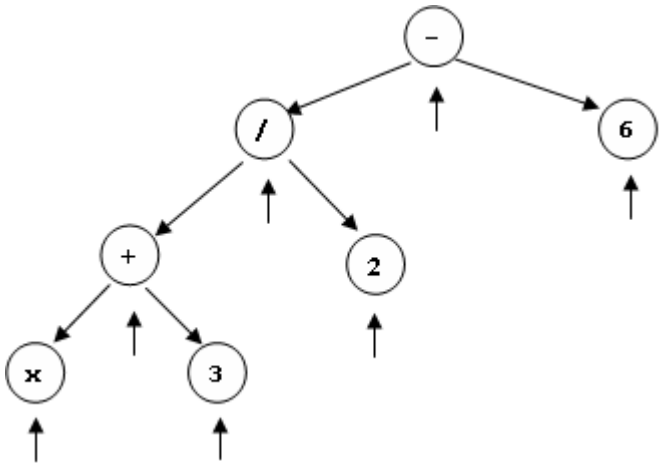


شكل 15-37

c. طريقة In Order

```
static void InOrder (Node ptr)
{
  if(ptr!=null){
    InOrder (ptr.getLeft());
    InOrder (ptr.getRight());
    System.out.print(ptr.element()+" ");
        }
}
```

يمكن إسترجاع البيانات من الرسم وذلك بإسترجاع البيانات الموجودة في أقصى اليسار ، بغض النظر عن المستوى ... وهكذا.

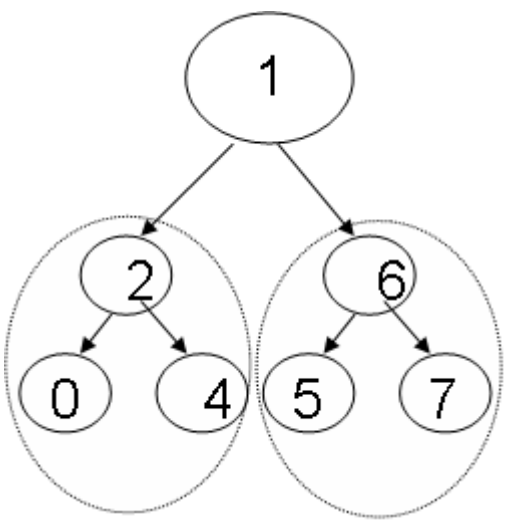


شكل 15-38

15.7.6 خوارزمية عد عقد الأشجار :

إن حجم الشجرة يساوي عدد العقد في الشجرة الفرعية اليمنى مضافاً إليها عدد العقد في الشجرة الفرعية اليسرى مضافاً إليها عقدة الجذر.

طباعة أب وأخ العدد المدخل كما في الشكل 15-39



شكل 15-39

فلو أدخلنا الرقم 6 فأنة سوف يطبع لنا الأب وهو 1 و يطبع لنا الأخ وهو 2 .
وهذا كود البرنامج:

```
1. //برنامج يطبع أخ و أب عدد مدخل بواسطة الشجرة الثنائية
2. import javax.swing.JOptionPane;
3. class Chp15_20 extends BTNode {
4. public static void main(String args[]){
5.   BTNode root=null;                    // root BTNode of the BTree
6.   BTNode right=null;                    // right BTNode of the BTree
7.   BTNode left=null;                    // left BTNode of the BTree
8.   BTNode node=null;
9.
10.       int Arr[]={5,6,2,8,4,10,18,9,0};
11.       //insert BTree
12.       for (int i=0;i<Arr.length;i++)
13.       {
14.        if(root==null)
15.                                    {
16.                                          root=node=new BTNode(Arr[0]);
17.                                    }
18.                                    else
19.                                    {
20.                                          node=new BTNode(Arr[i]);
21.                                          BTNode s,p;
22.                                          p=s=root;
23.                                          while(s!=null)
24.                                                {
25.                                                 p=s;
26.                                                 if(node.element()>s.element())
27.                                                    s=s.getRight();
28.                                                    else
29.                                                    s=s.getLeft();
30.                                                }
31.                                          if(node.element()>p.element())
32.                                                      p.setRight(node);
33.                                                      else
34.                                                      p.setLeft(node);
35.
36.                                    }
37.       }
38.
39.       //Print DBTree
40.       Print(root);
41.       System.out.println("\nDBTree");
42.
43.       String snum1;
44.       int num1;boolean flag=false;
45.       snum1 = JOptionPane.showInputDialog("Enter Number:") ;
46.       num1 = Integer.parseInt(snum1);
```

```
47.
48.     if(root.element()==num1)
49.                                             {
50.                                              System.out.println
51.                                              ("The Number Not Prather And Father");
52.                                              System.exit(0);
53.                                             }
54.     //Sertch The Number In Tree
55.     BTNode p=null;
56.     node=root;
57.     boolean f=true;
58.     while(f)
59.                             {
60.                              p=node;
61.                              if(node.element()>num1)
62.                                      node=node.getLeft();
63.                                      else
64.                                      node=node.getRight();
65.
66.                              if(node==null)break;
67.
68.                              if(node.element()==num1){
69.                                                              f=false;
70.                                                              break;
71.                                                      }
72.                             }
73.     if(f)
74.                      {
75.                              System.out.println("Not Found Number");
76.                              System.exit(0);
77.                      }
78.
79.     System.out.println(p.element()+" Father");
80.
81.     if(p.element()<num1)
82.                      {
83.                      if(p.getLeft()!=null)
84.                                      System.out.println(p.getLeft().element()+" Prather");
85.                                      else
86.                                      System.out.println("The Number Not Prather");
87.                      }
88.                      else
89.                      {
90.                      if(p.getRight()!=null)
91.                                      System.out.println(p.getRight().element()+" Prather");
92.                                      else
93.                                      System.out.println("The Number Not Prather");
94.                      }
95.     System.exit(0);
96.
97. }
```

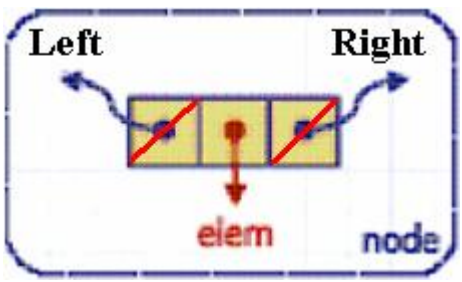
```
98. static void Print(BTNode node){
99.       if(node!=null){
100.                        System.out.print(node.element()+" ");
101.                        Print(node.getLeft());
102.                        Print(node.getRight());
103.                  }
104.                              }
105. }
```

15.7.7 خوارزمية حساب عدد أوراق الشجرة الثنائية

إن الورقة في الشجرة هي العقدة التي ليس لها أبناء كما قلنا سابقاً ، ولحساب عدد أوراق الشجرة ، لابد أولاً من التحقق من أن العقدة هي ورقة أم لا وذلك بهذا الشرط

if(node.getLeft()==null&&node.getRight()==null)

فيكون شكل الورقة كما في الشكل 40-15.



شكل 40-15

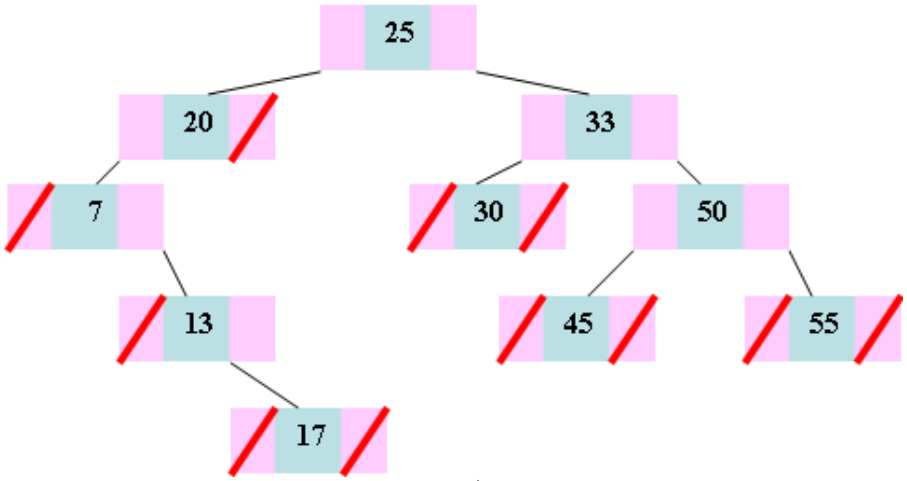
15.7.8 خوارزمية حذف عقد أوراق الشجرة الثنائية

- إذا كان العدد المراد حذفه ورقة لا يحتوي على أبناء فالأمر سهل ولا توجد مشقة

1. نبحث عن العدد وقبل التنقل إلى اليمين أو إلى الشمال نخزن موقعنا في متغير ثم ننتقل وهذا المتغير سيكون يمشي ورآنا بمقدار واحد للخلف إي يكون أب الموقع الحالي .
2. إثناء عمليه التنقل نستفسر عن نوع العقدة فإذا كانت ورقة أم لا بهذا الشرط :

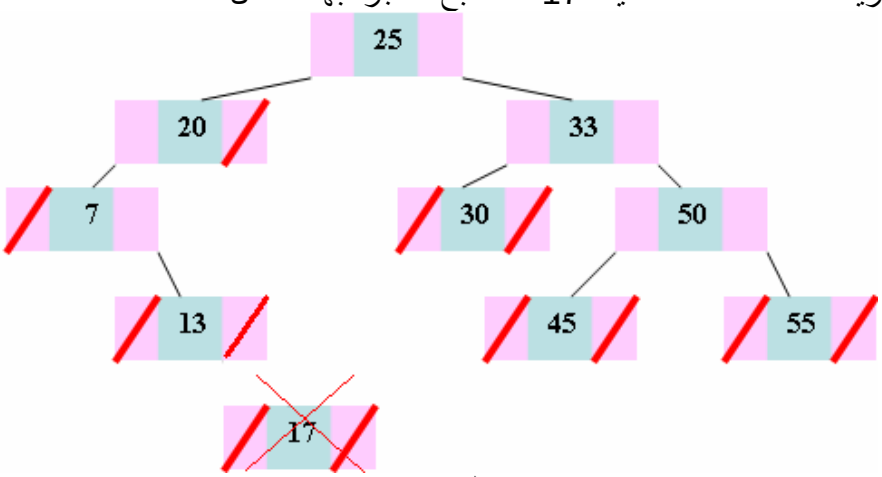
if(node.getLeft()==null&&node.getRight()==null)

بعد العثور على العدد بقي علينا أن نتأكد هل هو على يسار الأب أم على يمينه فأن كان على يساره نجعل حقل left=NULL فنكون عزلنا العقدة من الشجرة ثم نحذف العقدة ومن الشكل 41-15 سيوضح ذلك .



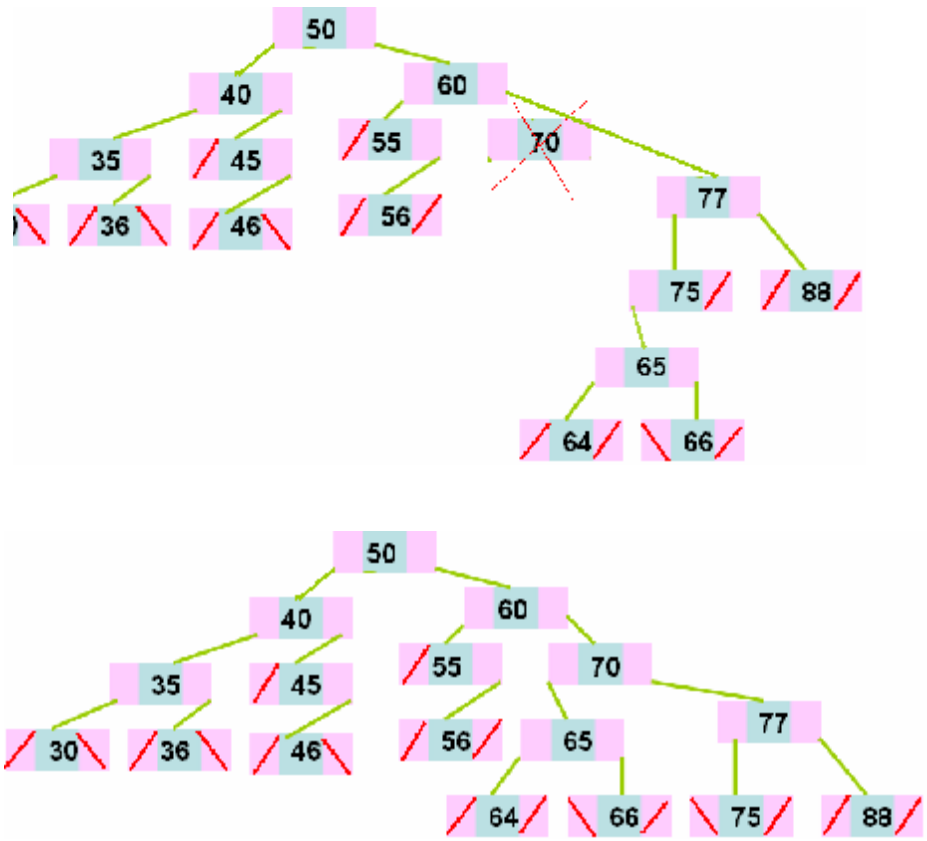
شكل 15-41

نريد حذف العقدة ذات القيمة 17 فستصبح الشجرة بهذا الشكل



شكل 15-42

- إذا كان العدد المراد حذفه يحتوي على شجرة فرعية مثل العدد 70



شكل 43-15

1. نحدد موقع العدد هل هو على يسار الأب أم على يمينه if(60>70).
2. هل العدد يوجد على يمينه فرضاً أعداد وشجرة فرعية if(node->right!=NULL) فإذا تحقق الشرط فأننا نربط يمين الأب بيمين الابن فنكون في هذه الحالة عزلنا 70 بقي علينا ربط
أجزاء الابن وهو نصل لأصغر قيمة في العقدة 77 ونربط يسارها بالعقد التي كانت متصلة بالعدد المراد حذفه فتصير الشجرة بهذا الشكل .

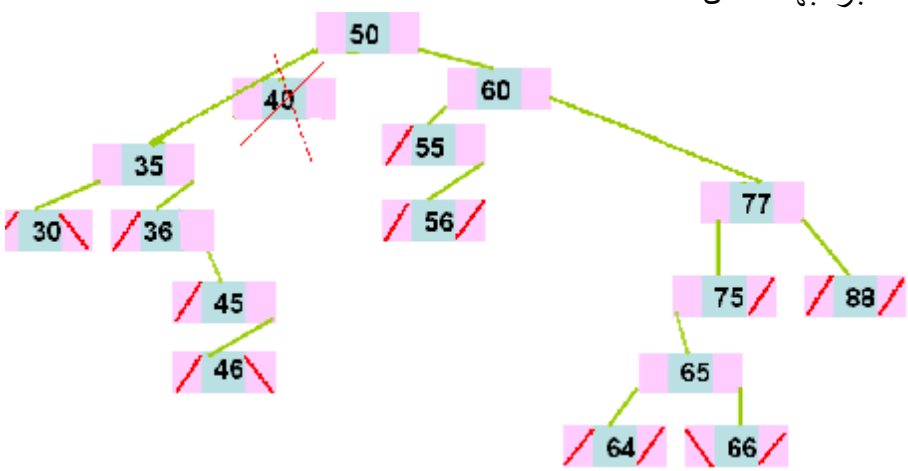
شكل 44-15

3

- تم نحذف العقدة .

هذا إن وجدت على يمين العدد عقد وان لم توجد نربط يمين الأب بيسار الابن مباشرة

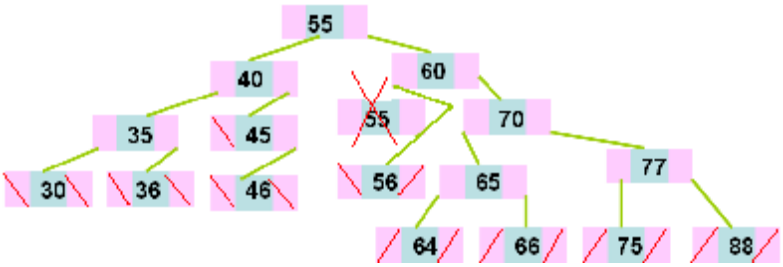
- **v** إذا كان العدد اصغر من الأب نفس الخطوات السابقة .
- **v** إذا كان العدد يوجد بيساره تفرع تربط يسار الأب بيسار الابن ونصل لأكبر عدد داخل التفرع ونربطه بيمين العدد مثل العدد 40 في الشكل السابق فتصير الشجرة بهذا الشكل



شكل 45-15

وإلا نربط يسار الأب بيمين الابن ، ثم نحذف العقدة .

- **v** بقي علينا حالة وهي أن أراد حذف الجذر هي بعض الشيء مربكة إلا إنها بسيطة
  - **ý** نبحث أولا عن اصغر عقدة في الجذع الأيمن للجذر ونطبق جميع الشروط التي ذكرناها سابقاً .
  - **ý** ونأخذ قيمة العقدة ونساوي قيمة الجذر بها ثم نحذف العقدة التي بحثنا عنها وبهذا نكون أحللنا قيمة الجذر إي بمثابة حذفنا الجذر وهذا الشكل سينتج .



شكل 46-15

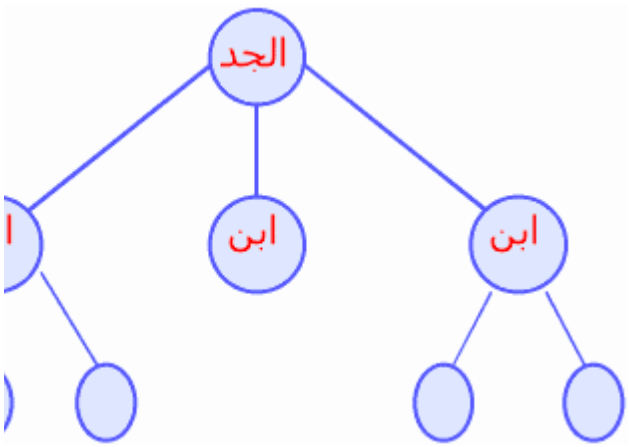
- وإن لم يوجد تفرع يمين للجذر فإننا ننقل الجذر بمقدار واحد لليسار ونحذف العقدة .

وهذه كل الخطوات يمكن دمجها ببرنامج شامل يستطيع أن يحذف من إي مكان .

إلى هنا ينتهي حديثنا عن الأشجار .

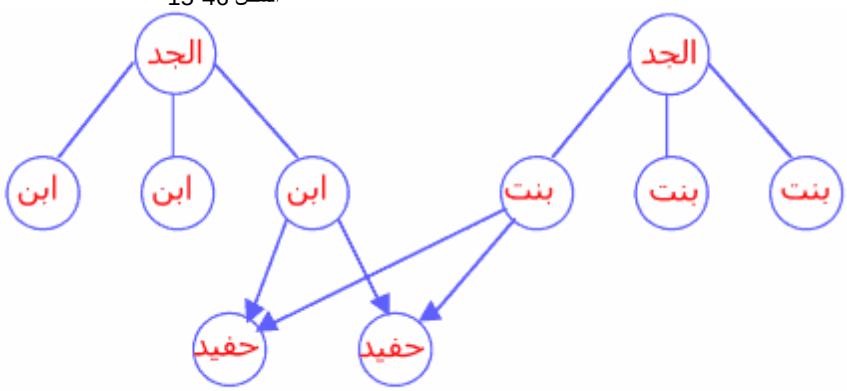
15.8 هياكل البيانات الشبكية (المترابطة)

وتعنى Plex الشبكات (الرسوم Graphs) كما تسمى بيانات مضفرة. إذا اتصل أي عنصر بيان في المستوى الأدنى من هياكل البيانات الشجرية بأكثر من عنصر في مستوى أعلى فيطلق عليه اسم هياكل بيانات شبكية، حتى



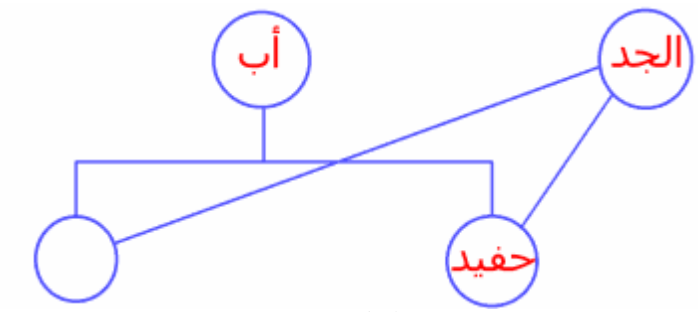
الشكل 15-46

شجرة العائلة من النوع الشبكي وليست من النوع الهرمي فالشكل15-46 يوضح العلاقات الهرمية لأسرة تتكون من جد، أبناء، أحفاد، شكل غير حقيقي لأننا أهملنا الأمهات منذ زمن بعيد لكن الشكل 15-47 وهي علاقة توضحيها أكثر كما في شكل 15-48 و الشكل 15-49 يبين التكافئ الشجري لمثال ما.

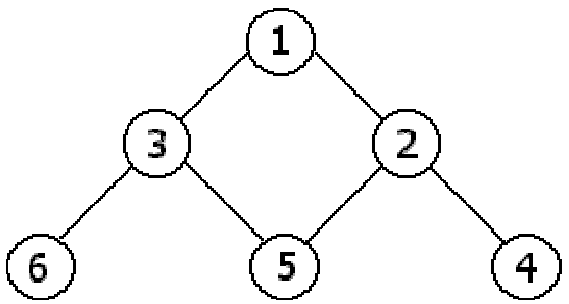


العلاقة الهرمية هي في الأساس علاقة شبكية

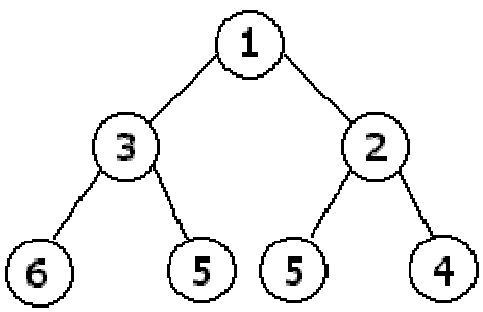
الشكل 15-47



الشكل 15-48

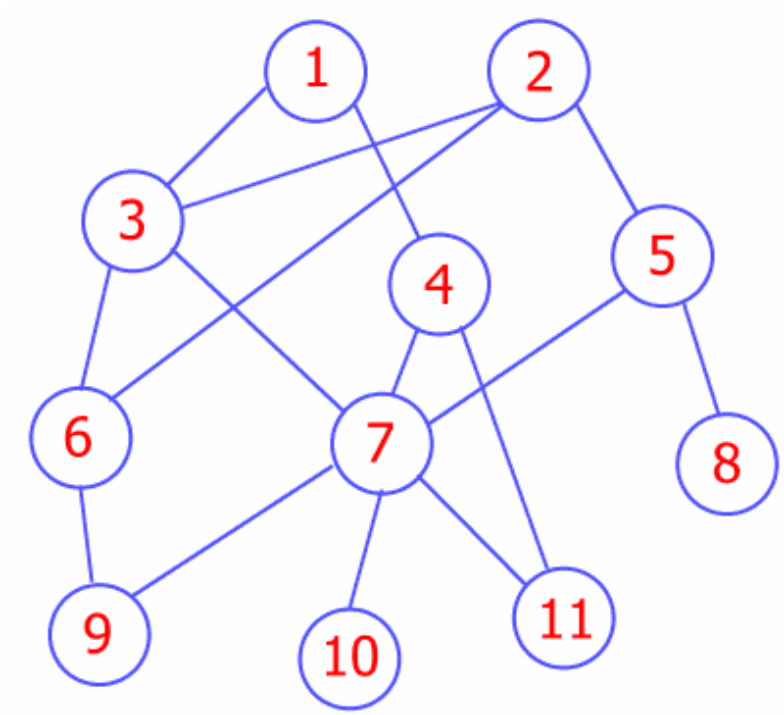


تكافئ الشكل الشجري التالي



الشكل 15-49

وتنقسم هياكل البيانات الشجرية إلى نوعين، بسيط ومعقد، ففي النوع البسيط يمكن تحديد مستويات الهيكل البنائي للبيانات أما النوع المعقد فيصعب ذلك كثيراً كما في الشكل 15-50 .



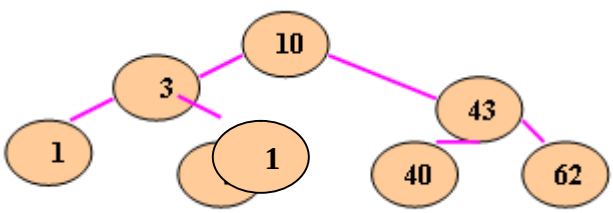
الشكل 50-15

تمارين الفصل

1. اكتب برنامج يقوم باستخراج الأعداد المتكررة من المكدس إي بحذف الأعداد المتكررة ؟
2. اكتب برنامج يقوم بعملية ترتيب مكدس ؟
3. اكتب برنامج يقوم بعملية إزاحة يمين وشمال للأعداد حسب الإزاحة المطلوبة من المستخدم؟
4. اكتب برنامج يقوم بدمج مكدسين ويعمل جميع العمليات المنطقية (تقاطع، اتحاد، طرح) ؟
5. اكتب برنامج يقوم بتحويل الأعداد الزوجية بمكدس والفردية بمكدس أخر ؟
6. اكتب برنامج يقوم بحذف الأعداد الأولية من المكدس ؟
7. اكتب برنامج يقوم بعكس طابور ؟
8. اكتب برنامج يقوم بفصل طابور من مكان محدد من قبل المستخدم ؟
9. اكتب برنامج يقوم بحذف المواقع الزوجية بالطابور ؟
10. اكتب برنامج يقوم بتدوير طابور بالاتجاهين يمن وشمال ؟
11. اكتب برنامج يقوم بنسخ مكدس إلى طابور ؟// ملاحظة يوجد فرق بين النسخ والقص//
12. لديك طابور بة أشخاص وهم في المستشفى ومرقمين من 1 إلى 5 وهم في صف الآذ تظار فجأة الشخص رقم 3 أصيب بوجع قوي ولا يستحمل الآذ تظار فأمر الطبيب بإدخاله لأنة حالة طارئة فكيف تعالج هذه المشكلة بتطبيق إلية وخوارزمية الطابور الأتي أولاً للطبيب الداخل أولاً ؟
13. اكتب برنامج يقوم بترتيب قائمة أحادية ؟
14. اكتب برنامج يقوم بقلب أي عكس قائمة أحادية ؟
15. اكتب برنامج يقوم بقلب المواقع الزوجية مع الفردية أي swap ؟
16. اكتب برنامج يقوم بحذف المواقع الزوجية بالقائمة الأحادية ؟
17. اكتب برنامج يقوم بدمج قائمتين أحاديتين وعمل جميع العلاقات الرياضية (تقاطع ، اتحاد ، طرح) ؟
18. اكتب برنامج يقوم بطباعة ثالث اكبر قيمة من القائمة ؟
19. اكتب برنامج يقوم بترتيب طابور بواسطة القائمة الأحادية ؟
20. اكتب برنامج بواسطة الدوال يقوم بطباعة ثالث مجموع اكبر القيم وطباعة الأعداد بالقائمة الأحادية مثل
21. (1,2,7,9,5,6) فيطبع 21 والأعداد 7,9,5 ؟
22. اكتب برنامج يقوم بتمثيل بيانات القائمة ألأحادية في الذاكرة بهذا الشكل ؟
23. هذا السؤال للقارئ النبيل ----- بواسطة القائمة الأحادية اكتب برنامج يعمل عمل القائمة الثنائية إي يكون الهيكل العام لقائمه مكون من متغيرين فقط إي next , int

لا يوجد مؤشر يؤشر للخلف مثل last فيستطيع العودة للخلف مرة ومرتين ..... الخ ؟

24. بواسطة القوائم ألثنائيه اكتب برنامج يقوم حذف المواقع الأولية من هذه القائمة ومعروف إن الأعداد الأولية (1,2,3,5,7,9,11,13،.....) ؟
25. بواسطة ألعوديه أو التكرار اكتب برنامج يعمل على طباعة عدد العقد الثنائية ؟
26. جميع الأسئلة التي ذكرت في الفصول السابقة ينبغي على القارئ تطبيقها بالقوائم الثنائية ؟
27. اكتب برنامج يقوم بتمثيل بيانات هذا الشكل في الذاكرة عم مراعاة نوع القوائم ؟
28. اكتب برنامج بواسطة الأشجار الثنائية يقوم بطباعة الشجرة بدون استخدام الاستدعاء الذاتي ؟
29. اكتب برنامج بواسطة الأشجار الثنائية يطبع مجموع ما تحت العدد المدخل وكم أعداد اكبر منة وكم أعداد اصغر منة ؟
30. اكتب برنامج بواسطة الأشجار الثنائية يقوم بطباعة الشجرة مرتبة تصاعدي ومرة تنازلي ؟
31. اكتب برنامج بواسطة الأشجار الثنائية يطبع كم عدد الآباء الذين لديهم أبناء اثنان وكم الذين لديهم ابن واحد وكم الذين ليس لهم أبناء ؟
32. ليكن لديك البيانات التالية (10,3,43,62, 5,1,40)



33. المطلوب منك طباعه هذه الشجرة أو إي شجرة بالشكل الأتي

```
            62
      43
            40
10
            5
      3
            1
```

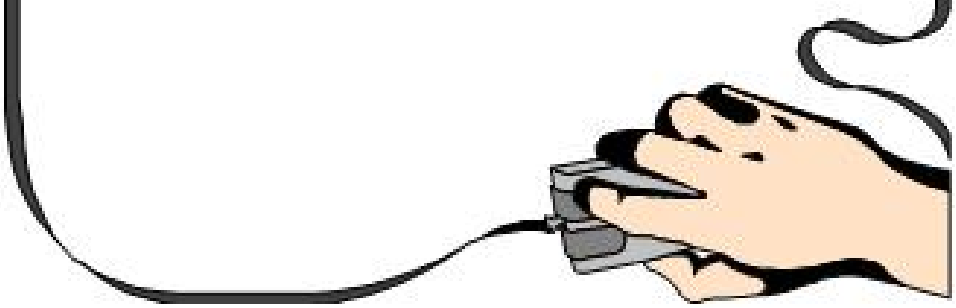
34. اكتب برنامج يقوم بتحويل شجرة ثنائية إلى طابور بشرط أن تدخل البيانات مرتبة للطابور وبدن استخدام إي خوارزمية ترتيب ؟
35. اكتب برنامج يقوم بتحويل طابور إلى شجرة ثنائية علماً أن البيانات في الطابور متكررة والشجرة الثنائية لا تقبل القيم المتكررة ؟
36. لديك الكلمات التالية (SAMI , AMMAR , AHMED , BASSAM , (SANAD , MOSTAFA , READ , ALI , KAMAL ,AMIN) المطلوب تكوين شجرة ثنائية مثل المثال الذي تكلمنا علية سابقاً ؟
37. اكتب برنامج يقوم بحذف الأعداد الأولية من الشجرة الثنائية ؟

Chp16

الرسم بالحاسوب

في نهاية هذا الفصل سوف تتعلم :

- مجالات الرسم بالحاسوب Computer Graphics Fields.
- الرسوم ثنائية الأبعاد Two-Dimensional والرسوم ثلاثية الأبعاد -Three Dimensional.
- مقدمة للرسم بالحاسوب في Introduction to Computer Graphics in Java
- أهم نظم الألوان Major Color Systems .
- نظم الألوان والأجهزة الطرفيةDevice-Dependent Color .
- جمعية اللون العالميةInternational Color Consortium .
- نظام الألوانRGB .
- نظم الألوان والألوان في Color Control & colors in Java .
- رسم الخطوط، المستطيلات، والدوائر Drawing Lines, Rectangles, & Ovals.
- رسم الأقواس Drawing Arcs .
- رسم المضلعات Drawing Polygons & Polylines .



16.1 مقدمة INTRODUCTION

لو سألنا أنفسنا عن مكونات أي صورة رسمت باستخدام الحاسوب، لو جدنا أن مكونات هذه الصورة ماهية إلا عناصر أولية هندسية، فالصورة ليست سوى مجموعة من الكائنات المركبة، هذه الكائنات أو العناصر الأولية Geometric Primitives من الممكن أن تكون:

- نقطة Pixel
- خط مستقيم Straight Line
- دائرة Circle
- مخروط Conic
- منحنى Spleen Curves
- مضلع Polygon
- رموز Character
- غير ذلك !!

إذن، نستطيع توضيح مفهوم تكوين الصورة باستخدام الحاسوب بالشكل 1-16:

شكل 1-16

ونستنتج من الصورة أن الرسم باستخدام الحاسوب ما هو إلا برامج Software يبرمجها المبرمج بهدف الرسم على الشاشة!! ولكن يجب أن نتنبه أن هذه البرامج لابد من أن تنفذ على مكونات صلبة Hardware خاصة، ففي C لابد من تهيئة بيئة للرسم قبل الشروع فيه، أما Java فبحكم كونها GUI فالبيئة فيها مهيأة، وسنتعرف في هذا الفصل على كيفية الرسم باللغة Java.

16.2 مجالات الرسم بالحاسوب Computer Graphics Fields

للرسم باستخدام الحاسوب مجالات عدة، نذكر منها على سبيل المثال أن الرسوم تستخدم في:

1. التعليم والتدريب Education & Training .
2. الواجهات الرسومية للبرامج "GUI" وهي اختصار لـ Graphical User Interfaces.
3. العمليات على الصور Image Processing .
4. الرؤية باستخدام الحاسوب Visualization .
5. تمثيل البيانات بالرسوم Representation Graphics.
6. الرسم والفن باستخدام الحاسوب Computer Art.
7. الترفيه Entertainment .
8. وأهم استخدام للرسم بالحاسوب هو في التصميم باستخدام الحاسوب أو ما يسمى "CAD" وهي اختصار لـ Computer Aided Design.
9. وغير ذلك الكثير الكثير.

فأشهر البرامج المستخدمة في الرسوم والتصاميم مثل برنامج أدوبي فوتوشوب، أو المستخدمة في التأثيرات والملتيميديا مثل برامج الفلاش، حتى أن الخطوط التي نستخدمها في محررات الكلمات ما هي إلا رسوم بالحاسوب برمجت باستخدام مفاهيم الرسم بالحاسوب المتعددة كما سندرسها إن شاء الله.

16.3 الرسوم ثنائية الأبعاد والرسوم ثلاثية الأبعاد Two- Three -Dimensional Dimensional and

لا يخفى على أي متعلم مفهوم الأشكال الثنائية البعد (كالمستطيل، المربع والمثلث.. ألخ) والأشكال الثلاثية البعد) كالمكعب، المخروط والأسطوانة.. ألخ) ، كما لا يخفى عليه أيضاً أهمية ومدى تأثير هذه الأشكال بنوعيها في الحياة التي نعيشها.
الأشكال ثنائية البعد تنشأ عن طريق ترتيب مقاسات مكونات الشكل واتجاهه، والتأثيرات الثنائية البعد تنشأ عن تحريك المنظور على الشكل طوال مدة التأثير. هذا الترتيب وهذه التأثيرات تسمى بـ"التحويلات الهندسية ثنائية الأبعاد Two-Dimensional Geometric Transformations"أي تغير إحداثيات الشكل على المحاور في المستوى.
كذلك الأشكال ثلاثية الأبعاد والتي تتكون من تركيب مجموعة أشكا ل ثنائية الأبعاد، أو بتأثيرات دقيقة على بعض الأشكال ثنائية البعد كي تنتج أشكالاً ثلاثية البعد.


تذكر أن الرسم بالحاسوب سواء رسم لأشكال ثنائية البعد أو لأشكال ثلاثية البعد أو حتى عمل بعض التأثيرات عليهما يعتمد على: استنتاج رياضي، فهم فكرة البرمجة، والتطبيق العملي!

16.4 مقدمة للرسم بالحاسوب في الجافا Introduction to Computer Graphics in Java

من المعلوم أن برامج Java تنقسم بصورة أساسية إلى قسمين:

- تطبيق Java Application: هو تطبيق يشبه التطبيقات المنشأة بجميع لغات البرمجة الأخرى يعمل مع نظام التشغيل بعيداً عن شبكة الانترنت والمشهور عن لغة جافا إنها تعدّ برامج للإنترنت ولكن غير المشهور أيضاً إنها توفر كثير من نقاط القوة في إعداد أي تطبيق سواء مكتبي DISKTOP أو خاص بالشبكات CLIENT\SERVER.
- برامج Java Applet: هو برنامج جافا يتم تشغيله على أداه تسمى *appletviewer* أو أي متصفح للإنترنت.

واشتهر الرسم في Java باندراجه تحت الـ Java Applet وفي الحقيقة أنه بإمكاننا الرسم Java أياً كان نوع البرمجة التي نستخدمها وسنعرف السبب لاحقاً في هذا الفصل بإذن الله.

وحيث أن Java تندرج تحت البرمجة الشيئية (Object Oriented Programming OOP) فلابد من أن يكون لدى القارئ خلفية عن هذا النوع من البرمجة وبإمكانه مراجعة الفصل التاسع.

16.5 مقدمة للرسم بالحاسوب في Java Introduction to Computer Graphics in Java2D

توفر Java إمكانيات رسم رفيعة المستوى كجزء من Java2D API ، والشكل 16-2 يوضح الفئات classes الأساسية للرسم بـJava.

حيث أن:

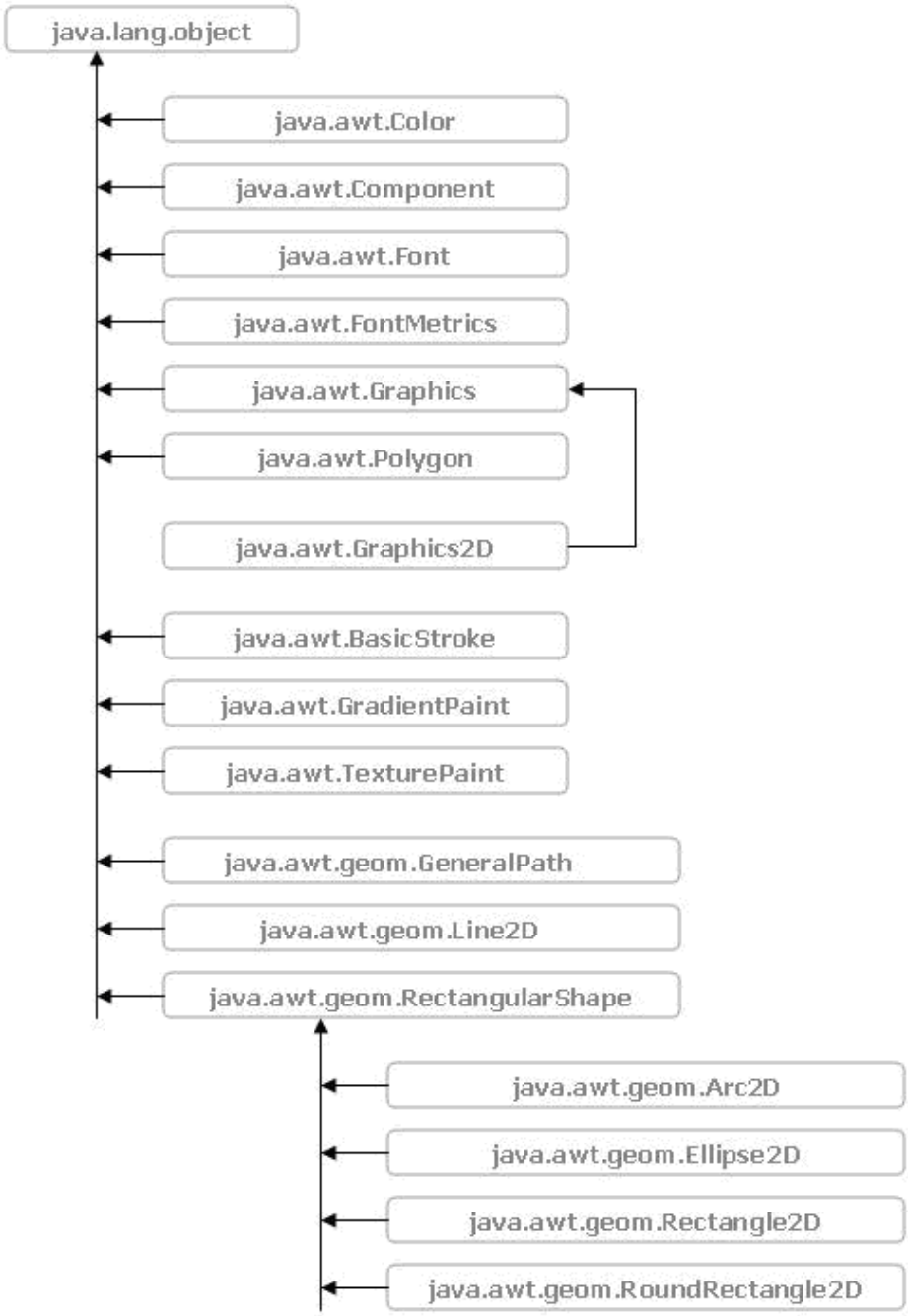
- Color Class: يحتوي الطرق methods والثوابت constants التي تمثل الألوان بـJava.
- Font Class: يحتوي الطرق methods والثوابت constants التي تمثل الخطوط في Java.
- FontMtrics Class:يحتوي الطرق methods التي تمثل معلومات الخطوط.
- Polygon Class:يحتوي طرق methods رسم المضلعات.

- Graphics Class : يحتوي طرق methods رسم الجُمل، الخطوط، المستطيلات والأشكال الأخرى.
- BasicStroke Class:يساعد على تحديد خصائص رسم الخطوط.
- GradientPaint & TexturePaint Classes: تساعد على تحديد خصائص ملء الأشكال بالألوان والنقوش.
- GeneralPath, Arc2D, Ellipse2D, Line2D, Rectangle2D & RoundRectangle2D Classes: تعرّف مختلف الأشكال ثنائية الأبعاد في Java Java2D Shapes.

كي نبدأ بالرسم بـJava، لابد أولاً من أن نفهم نظام محاور الإحداثيات في هذه اللغة!

من المعلوم لدينا أن الشاشة ما هي إلا نقط تعرّف كل نقطة بالبكسلPixel ، النقطة الواقعة في الركن الشمالي الغربي (الركن العلوي الأيسر) في الشاشة تمثل نقطة الأصل (0،0) في Java

النصوص والأشكال تعرض على الشاشة بتحديد مواضع إحداثياتها، حيث أن وحدة القياس في المحاور على الشاشة هي البكسل، تذكر ذلك جيداً فلن تستطيع الرسم على الشاشة إن نسيته .


java.lang.object
java.awt.Color
java.awt.Component
java.awt.Font
java.awt.FontMetrics
java.awt.Graphics
java.awt.Polygon
java.awt.Graphics2D
java.awt.BasicStroke
java.awt.GradientPaint
java.awt.TexturePaint
java.awt.geom.GeneralPath
java.awt.geom.Line2D
java.awt.geom.RectangularShape
java.awt.geom.Arc2D
java.awt.geom.Ellipse2D
java.awt.geom.Rectangle2D
java.awt.geom.RoundRectangle2D

شكل 2-16

16.6 مكونات الجرافيكس وكائناته

Graphics Contexts & Graphics Object

مكونات الرسم بـJava تتيح للمبرمج التحكم برسم المعلومات على الشاشة، كائنات فئة الجرافيكس أو Graphics class تحوي طرقاً methods للرسم وتكوين الخطوط والألوان وما إلى ذلك. أي Applet نراه يرسم على الشاشة فهو يستخدم كائن الـ Graphics المعروف "g" والذي يمثل متغير فيpaint method ، كذلك الحال بالنسبة لأي Applicationوسيأتي شرح ذلك بالتفصيل:

فئة Graphics تعتبر كباقي فئات Java فئة مجردةabstract class ، وهي تنتمي إلى awt class وفيها أي زر سيبدو مثل زر ويندوز على جهاز ويندوز، زر ماكنتوش على كمبيوتر ماكنتوش، وهكذا.

في الواقع فإن فئة المكونات Component Class هي الفئة الأم لأكثر الفئات في مكتبة java.awt.ويوجد بها الـ(paint method مربط الفرس) والتي تأخذ كائن الـ Graphics كمتغير argument لها.

بداية الـ paint method أو ما يطلق عليه الـ header كالتالي:

```
public void paint (Graphics g)
```

حيث أن كائن الجرافيكس g يعتبر كائن لفئة Graphics التي تتناغم مع أي نظام تشغيل كما ذكرنا. هذا الـ header نستخدمه للرسم مع الـ Java Application والـ Java Applet على السواء.

لماذا؟

لأن فئة Component هي فئة أساسية غير مباشرة (فئة أم) لفئة JApplet وبالتالي فإن كل إمكانيات فئة Component ورثت إلى فئةJApplet ، والـ paint method معرفة في فئة Component وهي التي نتحدث عنها. من المؤكد أنه اتضح للقارئ الكريم سبب إمكانية الرسم في الـ Java Applicationوالـ Java Applet بنفس الأدوات الآن .

ولمسح جميع الرسوم الموجودة على الشاشة للبدء بالرسم من جديد، هناك دالة تقوم بهذا الغرض هي:

public void repaint ( )

والتي بدورها تستدعي دالة أخرى تقوم فعلياً بمسح الشاشة هي:

public void update (Graphics g)

16.7 نظم الألوان، دقة اللون، والتحكم باللون ودوال التعامل في Java

Color Accuracy, Color Systems, & Color Control

16.7.1 أهم نظم الألوان Major Color Systems

نظم الألوان في العالم الرقمي تنقسم إلى قسمين رئيسيين، وفيما يلي تعداد لأهم نظم الألوان والحد الأقصى من عدد الألوان الذي يوفره كل نظام:

1. نظم تقوم على فكرة حبس الضوء: مثل نظام CMYK وهذه الأحرف اختصار لـ (cyan, magenta, yellow, and black سماوي، بنفسجي، أصفر، وأسود)، تستخدم في طباعة الألوان. شدة اللون في هذا النظام هي 4 بايت ويسمح هذا النظام بتخزين 4,294,967,296 لون.
2. نظم تقوم على فكرة إطلاق الضوء: مثل نظام RGB وهي اختصار لـ red, green, and blue، تستخدم كنظام ألوان أساسي في التلفاز وفي الشاشات. شدة اللون في هذا النظام هي 3 بايت ويسمح هذا النظام بتخزين16,777,216 لون.

وتقاس دقة اللون في كلا النظامين بجهاز يسمى colorimeter وهو يحاكي استجابة الإنسان للألوان ويُعني بقياس كثافة الضوء .

16.7.2 نظم الألوان والأجهزة الطرفية Device-Dependent Color: -

تعتمد نظم ألوان محددة لكل جهاز طرفي، وفيما يلي تحديد لنظم الألوان المستخدمة في كل جهاز:

- يعتمد نظام الألوان RGB في الشاشات وأجهزة المسح الضوئية والكاميرات الرقمية.
- يعتمد نظام الألوان CMYK في الطابعات.

إن كنت تنوي فصل الصورة لطباعتها في مطبعة تجارية، فالأفضل أن تستخدم نظام الألوان CMYK.

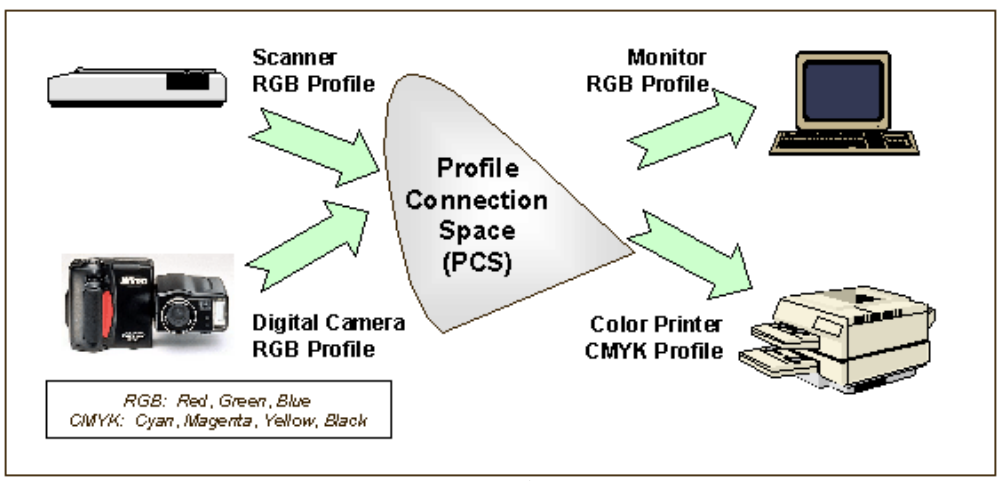
International Color Consortium

16.7.3 جمعية اللون العالمية

أنشئت هذه الجمعية في أوائل التسعينات كي تعطي تحديد وتوصيف لنظم الألوان المستخدمة في كل جهاز طرفي، الوظيفة الأساسية لهذا التوصيف هي إيجاد ملف خاص لأي جهاز إدخال أو إخراج Profile يحوي التحولات اللازمة لأي صورة عند انتقالها من جهاز طرفي إلى جهاز طرفي آخر -أي عند عرضها على جهاز طرفي آخر - قد يستخدم نفس نظام الألوان أو يستخدم نظام ألوان مختلف! ، يبلغ عدد أعضاء هذه الجمعية أكثر من خمسين عضو ما بين شركات مصنعة ومستخدمي الأجهزة التي تتعامل مع الصور.

اقترحت هذه الجمعية بيئة وسيطة تسمى (profile connection space (PCS, تحوي توصيف واضح للتحويلات اللازمة بين نظم الألوان المعتمدة في الأجهزة الطرفية المختلفة، وأنشئت لغات خاصة لهذا الغرض كما اعتمدت نوعين من هذه المساحة سمتهما CIEXYZ. PCS and CIELAB PCS

والصورة التالية توضح التحولات في نظم الألوان من جهازي إدخال إلى جهازي إخراج باستخدام الـ profileالخاص بكل جهاز والتعاون مع الـPCS :



شكل 3-16

وسنكمل حديثنا في هذا الدرس عن نظام الألوان الأكثر شهرة واستخدام وهو نظام RGB.

16.7.4 نظام الألوان RGB والألوان في Java:-

يعتمد هذا النظام في كل من الشاشات والماسحات الضوئية والكاميرات الرقمية، كما أن جميع البرامج تقريباً وبلا استثناء تستخدم هذا النظام، وسنحاول فيما يلي شرحه .

تشير الأحرف الثلاث RGB اختصار Red, Green & Blue إلى نظام الألوان المستخدم في العرض الكمبيوتري، تخلط هذه الثلاث الألوان بنسب مختلفة للحصول على أي لون من ألوان الطيف الضوئي، ولكل لون نطاق يبدأ من الصفر ويصل إلى الـ255 (بمعنى 256 حالة لكل لون)، وهو يقابل:

- نطاق عشري Floating يبدأ من 0،0 ويصل إلى 1،0.
- نطاق ثنائي Binary يبدأ من 00000000 إلى 11111111.
- والذي بدوره يقابل في النظام السادس عشر نطاق يبدأ من 00 ويصل إلى.FF

عدد الألوان المتاحة في هذا النظام هي 256 x 256 x256 وتساوي 16,777,216 لون.

في لغة HTML مثلاً، تمثل الألوان في هذا النظام عن طريق استخدام النظام السادس عشر، بتخصيص 6 خانات لكل لون: الخانتين الأولى والثانية للأحمر والخانتين الثالثة والرابعة للأخضر والأخيرتان للأزرق. ويمثل هذا النظام في بقية البرامج واللغات بنفس الشكل تقريباً. عدم وجود أي من هذه الألوان الثلاثة يولد اللون الأسود، أما وجود الألوان الثلاثة جميعاً بنسبة 100% - باستخدام أي نظام لتمثيلها- فيولد اللون الأبيض، و الجدول 1-16 يوضح قيم RGBلبعض الألوان والثوابت التي توفرها Java للألوان:

جدول 1-16

اللونColor	قيمتهRGB Value	ثابت اللون في اللغة Color Constant
أحمر	255,0,0	public final static color red
أخضر	0,255,0	public final static color green
أزرق	0,0,255	public final static color blue
برتقالي	255,200,0	public final static color orange

public final static color pink	255,175,175	وردي
public final static color cyan	0,255,255	سماوي
public final static color magenta	255,0,255	بنفسجي
public final static color yellow	255,255,0	أصفر
public final static color black	0,0,0	أسود
public final static color white	255,255,255	أبيض
public final static color gray	128,128,128	رمادي
public final static color lightGray	192,192,192	رمادي باهت
public final static color darkGray	64,64,64	رمادي غامق

ومن الممكن أن تعطى من المعادلات البسيطة التالية

جدول 2-16

اللونColor	أحمرRed	أخضرGreen	أزرقBlue
Black = 0	0	0	0
Blue = B	0	0	1
Green = G	0	1	0
Cyan = G + B	0	1	1
Red = R	1	0	0

Magenta = R + B	1	0	1
Yellow = R + G	1	1	0
White = R + G + B	1	1	1

16.7.5 دوال للتعامل مع الألوان في Java

الجدول 3-15 يوضح بعض الدوال الجاهزة في Java للتعامل مع الألوان,

جدول 3-16	
التوصيفDescription	الدالةMethod
إنشاء لون من نظام RGB يعبر عن هذا اللون بالأرقام الصحيحة من 0 إلى 255.	public color (int r, int g, int b)
نشاء لون من نظام RGB يعبر عن هذا اللون بقيمة عشرية من 0,0 إلى 1,0.	public color (float r, float g, float b)
يعيد قيمة بين 0 و 255 تمثل قيمة خانة اللون الأحمر.	public int getRed ( ) // Color class
يعيد قيمة بين 0 و 255 تمثل قيمة خانة اللون الأخضر.	public int getGreen ( ) // Color class
يعيد قيمة بين 0 و 255 تمثل قيمة خانة اللون الأزرق.	public int getBlue ( ) // Color class
يعيد كائن Color يمثل اللون الحالي في أي محتوى رسوم.	public color getColor ( ) // Graphics class
تحديد اللون المطلوب لاستخدام ه في الرسوم.	public void setColor (Color c) // Graphics class

سؤال :لماذا هذه الثلاثة ألوان بالذات (أحمر، أخضر، أزرق)؟

عوضاً عن أمكانية تكوين أي لون من ألوان الطيف باستخدام هذه ألوان الثلاثة، فإن تقنيات أجهزة العرض صممت لتتعامل مع هذه الثلاثة ألوان، بإمكانك المرور سريعاً على هاذان

الدرسان عن شاشات الكمبيوتر لتتعرف أكثر الارتباط الوثيق بين الـ software و الـ hardwareعن طريق هذه الألوان الثلاث:

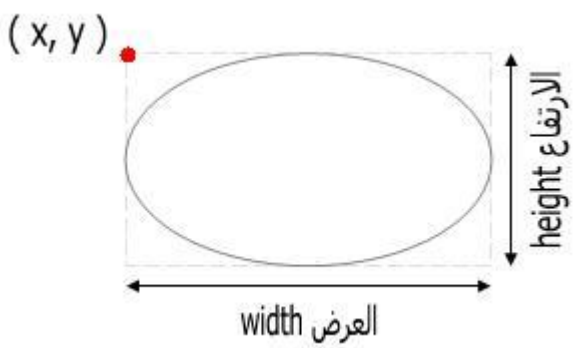
- شاشات الكمبيوتر[1]: شاشات :CRT كيفية العمل ودليل الشراء
- شاشات الكمبيوتر[2]: شاشات LCD ومقارنة بينها وبين شاشاتCRT

16.7.6 رسم الخطوط، المستطيلات، والدوائر

Drawing Lines, Rectangles, & Ovals

سنناقش في هذا الجزء دوال الجرافيكس Graphics method المتوفرة في Java لرسم الخطوط، المستطيلات والدوائر، حيث أننا:

- نرسم الخط بتحديد إحداثيات نقطتين عليه، إحداثيات هذه النقاط لابد من أن تكون أعداد صحيحة فلا يوجد شيئ اسمه "بكسل ونصف" على الشاشة (:
- نرسم المستطيل بتحديد إحداثيات الركن العلوي الأيسر) الشمالي الغربي) فيه، ثم بتحديد عرض وارتفاع المستطيل.
- نرسم الشكل البيضاوي كذلك بتحديد إحداثيات الركن العلوي الأيسر (الشمالي الغربي) فيه، ثم بتحديد عرض وارتفاع المستطيل كما يوضح الشكل 4-16:



شكل 4-16

16.7.7 الصنف من نوع Graphics

الصنف Graphics هو صنف مجرد ويستخدم من أجل عرض الأشكال والصور على الشاشة ضمن منصات مختلفة.فيتم تنفيذه ضمن المنصة الأم في آلة Java الافتراضية.

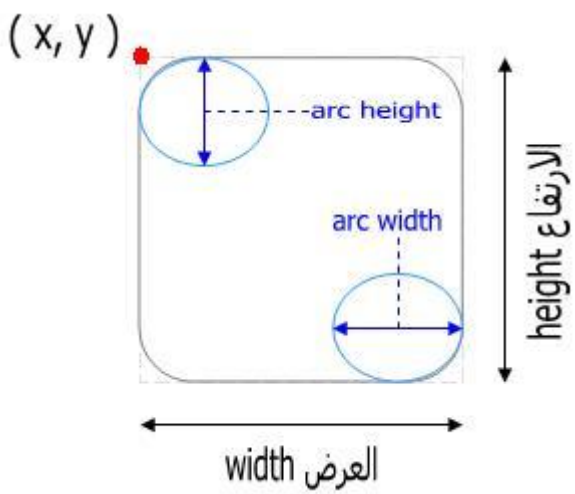
يوضح الجدول 4-16 المناهج التابعة للصنف Graphics التي تقوم بالرسم:

جدول 4-16

الدالةMethod	الوصفDescription
puplic void drawLine (int x1, int y1, int x2, int y2)	
	لرسم خط يصل بين النقطتين (x1, y1)وَ (x2, y2).
puplic void drawRect (int x, int y, int width, int height)	
	لرسم مستطيل أجوف بالعرض المعطى والارتفاع المعطى، إحداثيات الركن العلوي الأيسر هي (x, y).
puplic void fillRect (int x, int y, int width, int height)	
	لرسم مستطيل معبأ باللون بالعرض المعطى والارتفاع المعطى، إحداثيات الركن العلوي الأيسر هي (x, y).
puplic void clearRect (int x, int y, int width, int height)	
	لرسم مستطيل بالعرض المعطى والارتفاع المعطى معبأ بلون الخلفية، إحداثيات الركن العلوي الأيسر هي (x, y).
puplic void drawRoundRect (int x, int y, int width, int height, int arcWidth, int arcHeight)	
	لرسم مستطيل أجوف ذو حواف دائرية بالعرض المعطى والارتفاع المعطى، المتغيران arcWidth, arcHeight & يحددان مقدار انحناء الحواف كما

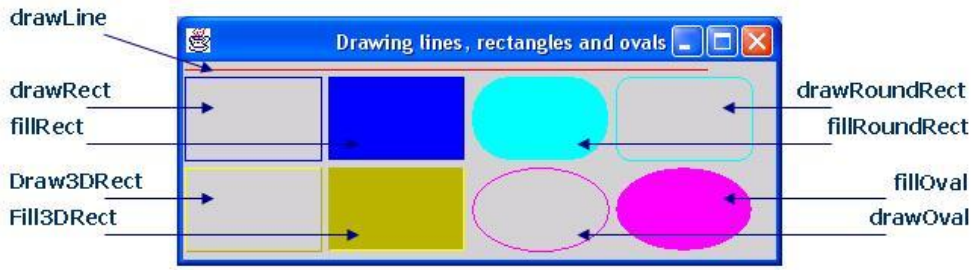
توضح الصورة (2)، إحداثيات الركن العلوي الأيسر هي (x, y).	
puplic void fillRoundRect (int x, int y, int width, int height, int arcWidth, int arcHeight)	
لرسم مستطيل ذو حواف دائرية معبأ باللون بالعرض المعطى والارتفاع المعطى، المتغيران arcWidth, arcHeight & يحددان مقدار انحناء الحواف كما توضح الصورة (2) ، إحداثيات الركن العلوي الأيسر هي (x, y).	
puplic void draw3DRect (int x, int y, int width, int height, boolean b)	
لرسم مستطيل ثلاثي الأبعاد أجوف بالعرض المعطى والارتفاع المعطى وحدوده باللون الحالي، إحداثيات الركن العلوي الأيسر هي(x, y) ، المستطيل يظهر بارزاً raised عندما نعطي المتغير b قيمة true ويكون مجوفاً lowered عندما نعطي المتغير b قيمة false.	
puplic void fill3DRect (int x, int y, int width, int height, boolean b)	
لرسم مستطيل ثلاثي الأبعاد معبأ باللون بالعرض المعطى والارتفاع المعطى واللون الحالي، إحداثيات الركن العلوي الأيسر هي ((x, y ، المستطيل يظهر بارزاً raised عندما نعطي المتغير b قيمة true ويكون مجوفاً lowered عندما نعطي المتغير b قيمة false.	
puplic void drawOval (int x, int y, int width, int height)	
لرسم شكل بيضاوي أجوف باللون الحالي بالعرض المعطى والارتفاع المعطى، إحداثيات الركن العلوي	

	الأيسر للمستطيل الذي يحيط بالشكل البيضاوي كما توضح الصورة (1) هي (x, y). الشكل البيضاوي يلامس المستطيل في أربع نقاط تقع كل نقطة في منتصف كل ضلع للمستطيل. المستطيل طبعاً لن يظهر على الشاشة.
puplic void fillOval (int x, int y, int width, int height)	
	لرسم شكل بيضاوي معبأ باللون الحالي بالعرض المعطى والارتفاع المعطى، إحداثيات الركن العلوي الأيسر للمستطيل الذي يحيط بالشكل البيضاوي كما توضح الصورة (1) هي (x, y). الشكل البيضاوي يلامس المستطيل في أربع نقاط تقع كل نقطة في منتصف كل ضلع للمستطيل. المستطيل طبعاً لن يظهر على الشاشة.



شكل 5-16

سنستخدم جميع هذه الدوال تقريباً في برنامجنا التالي، مخرجات برنامجنا يوضحها الشكل 16-6.



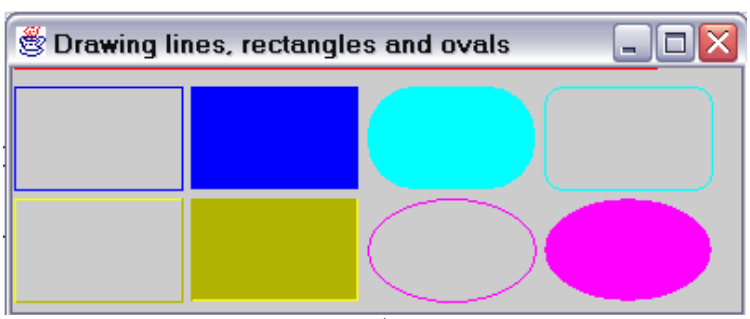
شكل 16-6

وهذا البرنامج:

```
1. // Drawing lines, rectangles and ovals
2. import java.awt.*;
3. import java.awt.event.*;
4. import javax.swing.*;
5.
6. public class Chp16_1 extends JFrame {
7.         private String s = "Using drawString!";
8.
9.         public Chp16_1()
10.                 {
11.                  super( "Drawing lines, rectangles and ovals" );
12.
13.                  setSize( 400, 165 );
14.                  show();
15.                 }
16.
17.        public void paint( Graphics g )
18.                 {
19.                  g.setColor( Color.red );
20.                  g.drawLine( 5, 30, 350, 30 );
21.
22.                  g.setColor( Color.blue );
23.                  g.drawRect( 5, 40, 90, 55 );
24.                  g.fillRect( 100, 40, 90, 55 );
25.
26.                  g.setColor( Color.cyan );
27.                  g.fillRoundRect( 195, 40, 90, 55, 50, 50 );
28.                  g.drawRoundRect( 290, 40, 90, 55, 20, 20 );
29.
```

```
30.                     g.setColor( Color.yellow );
31.                     g.draw3DRect( 5, 100, 90, 55, true );
32.                     g.fill3DRect( 100, 100, 90, 55, false );
33.
34.                     g.setColor( Color.magenta );
35.                     g.drawOval( 195, 100, 90, 55 );
36.                     g.fillOval( 290, 100, 90, 55 );
37.                   }
38.
39. public static void main( String args[] )
40.  {
41.        Chp16_1 app = new Chp16_1();
42.
43.        app.addWindowListener(    new WindowAdapter() {
44.                    public void windowClosing( WindowEvent e )
45.        {
46.         System.exit( 0 );
47.        }
48.
                    }
49.                                                              );
50.  }
51. }
```

فيكون الخرج:



شكل 16-7

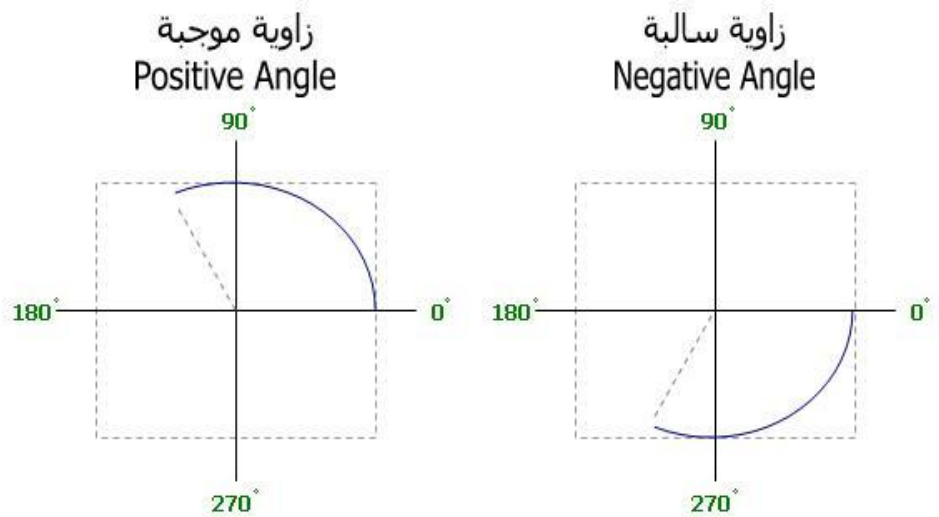
شرح المثال:

- كما رأينا في الصورة؛ فإن المستطيل ثلاثي الأبعاد يتم رسمه عن طريق اللعب بلون الحواف لإظهار الشكل بارز أو مجوّف، فتظهر حافتان باللون الذي حددناه بينما ترسم الحافتان الأخيرتان بنفس اللون ولكن بدرجة أغمق قليلاً، بعض الألوان يصعب رؤية الأشكال الثلاثية البعد التي ترسم بها!

- إذا أردت رسم دائرة، اجعل قيمة العرض والارتفاع واحدة في أحد الدالتين drawOval أوfillOval .

16.7.8 رسم الأقواس Drawing Arcs

القوس هو جزء من الشكل البيضاوي أو الدائرة، تقاس زاوية التقوس بالدرجة، لرسم القوس يزاح القوس من نقطة البداية إلى أن يصل إلى زاوية التقوس. الشكل 16-8 يوضح قوسين، القوس الأيسر أزيح من زاوية البداية (0) إلى تقريباً 110 درجة وهي درجة موجبة !بينما القوس الأيمن أزيح باتجاه عقارب الساعة من زاوية البداية (0) إلى تقريباً -110 درجة وهي درجة سالبة!



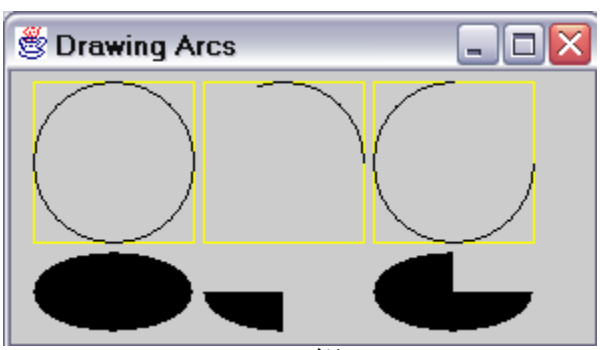
شكل 16-8

تذكر أن المستطيل المنقط الظاهر في الصور يمثل في الحقيقة كيفية رسم القوس، حيث أن القوس هو جزء من الشكل الدائري أو البيضاوي، وقد استخدمنا المستطيل في رسم الأشكال البيضاوية وتعلمنا ذلك في الدرس السابق!

توفر فئة الـ Graphics دالتين لرسم الأقواس، هما drawArc و fillArc نوضحهما في الجدول 16-5.

جدول 16-5	
Method الدالة	Description الوصف
puplic void drawArc (int x, int y, int width, int height, int startAngle, int arcAngle)	
لرسم قوس يقترب نسبياً من المستطيل الذي نحدد إحداثيات الركن العلوي الأيسر فيه بـ (x, y)ونرسمه بالطول المعطى والعرض المعطى. القوس الدائري يبدأ رسمه من زاوية البداية startAngle ويزاح إلى أن يصل إلى درجة التقوّس arcAngle.	
puplic void fillArc (int x, int y, int width, int height, int startAngle, int arcAngle)	
لرسم قوس معبأ باللون solid يقترب نسبياً من المستطيل الذي نحدد إحداثيات الركن العلوي الأيسر فيه بـ (x, y)ونرسمه بالطول المعطى والعرض المعطى. القوس الدائري يبدأ رسمه من زاوية البداية startAngle ويزاح إلى أن يصل إلى درجة التقوّس arcAngle.	

والآن، سنجرب هذه الدوال في تطبيق ! سنقوم برسم ثلاثة أقواس باستخدام الدالة الأولى وثلاثة أقواس معبأة باستخدام الدالة الثانية، كذلك سنضع الأقواس الثلاث الأولى داخل مربعات صفراء اللون باستخدام دالة drawRect التي درسناها من قبل وسنعطيها نفس إحداثيات الأقواس x, y, width & height حتى تظهر الأقواس داخل المربعات! مخرجات برنامجنا يوضحها الشكل 16-9.



شكل 16-8

طبّق البرنامج حالاً:

```
1. // Drawing Arcs – Auther:
2. import java.awt.*;
```

```
3. import javax.swing.*;
4. import java.awt.event.*;
5. public class Chp16_2 extends JFrame {
6.         public Chp16_2()
7.         {
8.         super( "Drawing Arcs" );
9.
10.        setSize( 300, 170 );
11.        show();
12.    }
13.
14.        public void paint( Graphics g )
15.        {
16.        // start at 0 and sweep 360 degrees
17.        g.setColor( Color.yellow );
18.        g.drawRect( 15, 35, 80, 80 );
19.        g.setColor( Color.black );
20.        g.drawArc( 15, 35, 80, 80, 0, 360 );
21.
22.        // start at 0 and sweep 110 degrees
23.        g.setColor( Color.yellow );
24.        g.drawRect( 100, 35, 80, 80 );
25.        g.setColor( Color.black );
26.        g.drawArc( 100, 35, 80, 80, 0, 110 );
27.
28.        // start at 0 and sweep -270 degrees
29.        g.setColor( Color.yellow );
30.        g.drawRect( 185, 35, 80, 80 );
31.        g.setColor( Color.black );
32.        g.drawArc( 185, 35, 80, 80, 0, -270 );
33.
34.        // start at 0 and sweep 360 degrees
35.        g.fillArc( 15, 120, 80, 40, 0, 360 );
36.
37.        // start at 270 and sweep -90 degrees
38.        g.fillArc( 100, 120, 80, 40, 270, -90 );
39.
40.        // start at 0 and sweep -270 degrees
41.        g.fillArc( 185, 120, 80, 40, 0, -270 );
42.        }
43.
44.        public static void main( String args[] )
45.        {
46.        Chp16_2 app = new Chp16_2();
47.
48.        app.addWindowListener(
49.        new WindowAdapter() {
50.        public void windowClosing( WindowEvent e )
51.                {
52.                System.exit( 0 );
53.                }
```

```
54.                                     }
55.                                     );
56.         }
57. }
```

شرح المثال:

- من السطر 17 - 20 : رسمنا دائرة داخل مربع.
- من السطر 23 - 26 : رسمنا قوس يبدأ من الزاوية صفر إلى 110 درجة- زاوية موجبة- داخل مربع أصفر بنفس الإحداثيات.
- من السطر 32 - 29 : رسمنا قوس يبدأ من زاوية الصفر إلى الزاوية -270 درجة -زاوية سالبة- داخل مربع أصفر بنفس الإحداثيات.
- السطر 35 :رسمنا شكل بيضاوي معبأ باللون باستخدام دالة رسم القوس.fillArc
- السطر 38 : رسمنا قوس معبأ باللون يبدأ من زاوية 270 إلى الزاوية -90 درجة. سنحصل على نفس النتيجة لو رسمنا القوس من الزاوية 180 إلى 90 درجة.
- السطر 41 : رسمنا قوس معبأ من الزاوية صفر وأزيح 270 درجة سالبة.

تمارين الفصل

.1

Chp17

برمجة الوسائط المتعددة

في نهاية هذا الفصل سوف تتعلم :

-

17.1 مقدمة INTRODUCTION

Chp18

قواعد البيانات

في نهاية هذا الفصل سوف تتعلم :

- معنى قاعدة البيانات.
- تركيب قواعد البيانات:
- أنواع قواعد البيانات.
- لغة الاستعلام المركّبة SQL.
- استعلامات التحديد Selection Queries.
- إجراء العمليّات على خانات الحقول.
- ربط الجداول SQL Joins.
- استعلامات الأداء Action Queries.
- إنشاء اتصال بين قاعدة البيانات و برنامج Java .
- استعلام محتويات الجداول الموجودة بداخل قاعدة البيانات.
- التعديل والحذف والإضافة على الجداول.

18.1 مقدمة INTRODUCTION

تعتبر قواعد البيانات أحد الحلول الجيدة والمكلفة لمشاكل بيئة نظم الملفات وإبرازها مشكلة تكرارية البيانات وانعكاساتها السلبية على استخدام وسائط التخزين وتضارب المعلومات وما يستتبع ذلك من تكاليف لحفظها وتشغيلها وصيانتها ناهيك عن أن تحديث أي عنصر بيان لا يعني تحديثه على مستوى النظام بل يقتصر على الملف المعني بهذا التحديث مما يسبب عدم تكاملية البيانات وعدم إمكانية فرض إدارة مركزية وسيطرة أمنية تقي المعلومات من مخاطر التدخل فيها أوالإخلال بها أو سرقتها.

وفي هذا الفصل سوف تتعلم ذلك بالتفصيل.

18.2 معنى قاعدة البيانات

قاعدة البيانات Database هي تركيبٌ تستطيع من خلاله تخزين كميّةٌ ضخمةٌ من البياناتِ التي تربطها علاقات معيّنة، مثل الجداول.. وحقيقةُ الأمر، فإنّ قاعدة البيانات ما هي إلا ملفّ File أو مجموعة ملفات تُخزّن على القرص الصلب Hard Disk ككلّ الملفّات العاديّة، ولكنّ ما يميّزها عن باقي الملفّات، هو قدرتك على كتابة البيانات فيها وقراءتها منها، باستخدام الوسائل التي تمنحها لك برامج قواعد البيانات، دون الحاجة إلى التعامل مع الملفّ مباشرةً، وهو ما يجعل الأمر أكثر سهولةً وتنظيمًا بالنسبة لك.

ولا مانعَ أن تخزّنَ بياناتك في ملفّ خاصّ بك بالطريقة التي تريحك، ولكنّ استخدام قواعد البيانات يوفّر لك الكثير من الوقتِ والجهد.

وهناك أنواع كثيرة من قواعد البيانات، تبعا للشركة التي تنتجها والبرامج التي تنشئها، فهناك قواعد بيانات برنامج Access وبرنامج SQL Server وهما من إنتاج ميكروسوفت، وهناك عشرات البرامج غيرهما من إنتاج شركات أخرى.

وتسمّى هذه البرامج " أنظمة إدارة قواعد البيانات" Database Management Systems (DBMS) ، ومهمّتها الأساسيّة أن تمنحك الوسائل والأدواتِ اللازمة لإنشاءِ قواعدِ البياناتِ والتعاملِ مع ها، بأسهلِ طريقةٍ وأفضلِ إمكانيّات، بحيثُ تقوم بدور الوسيط بينك وبينَ البيانات المخزّنة في ملفّ قاعدة البيانات.

ومهما كانَ نوع قاعدة البيانات التي تتعامل معها، ومهما كانت طريقة تخزينها في الملفّ، فإنَّ كلّ قواعد البيانات تتبع قواعدَ أساسيّة وتحقّق شروطا معيّنة متعارفا عليها دوليّا، كما إنها كلّها بلا استثناء تستخدم " لغة الاستعلام المركّبة" Structured Query Language (SQL)، وهي لغة خاصّة لحفظ واسترجاع وتحديث البيانات في قواعد البيانات.

18.3 مميزات قواعد البيانات: -

1. إمكانية إضافة ملفات جديدة .
2. إضافة بيانات جديدة على الملفات الموجودة في القاعدة.
3. استرجاع بيانات من الملفات المكونة لقاعدة البيانات.
4. تحديث البيانات .
5. حذف البيانات من الملفات .
6. إزاحة ملفات خالة أو مكتوب عليها مسجلات .
7. يمكن تعديل البرامج دون تعديل البيانات والعكس صحيح .
8. يمكن للمستخدم النظر إليها على إنها ملفات متكاملة .
9. تلبي حاجات كافة المستخدمين للبيانات .
10. يمكن فرض قيود التأمين والسرية على بعض البيانات الهامة .
11. تحقق المرجعية على الملفات.
12. إمكانية تخليق بيانات جديدة من البيانات الموجودة على الملفات.

وبذلك تتلافى معظم عيوب بيئة نظم المعلومات المرتكنة على الملفات.

18.4 مكونات نظام قاعدة البيانات

يتكون نظام قاعدة البيانات من أربعة مكونات أساسية هي:

18.4.1 البيانات

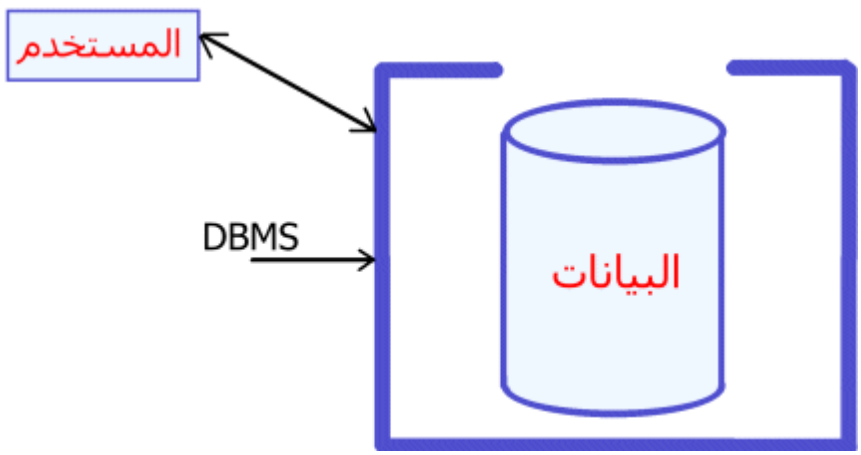
تتوافر قواعد البيانات على الحاسبات الصغيرة والشخصية كما تتوافر على الحاسبات الكبيرة وتعتمد كفاءة النظام على قدرة وإمكانيات الكيان الآلي للحاسب، فالحاسبات الشخصية توفر قاعدة بيانات لمستخدم واحد بينما الحاسبات الكبيرة توفر قاعدة بيانات لعدد من المستخدمين يشا ركون على البيانات المتاحة،يجب أن تتصف البيانات بالتكاملية و عدم التكرارية وإمكانية المشاركة عليها،و عموماً دون بيانات دقيقة ومنطقية وصحيحة فلا معنى لقاعدة البيانات.

18.4.2 المعدات

ترتكن قواعد البيانات على الأقراص المغناطيسية ارتكاناً كبيراً علاوةً على وحدات من الشرائط الكثيفة كوحداتBACK UP احتياطية لتخزين البيانات للظروف الطارئة.

18.4.3 البرامج

وهي الطبقة الوسيطة بين البيانات المخزنة في الملفات أعلى الأقراص و بين مستخدمي قاعدة البيانات، وأبرز هذه البرامج برنامج مدير قاعدة البيانات DBMS وهو عبارة عن برنامج بالغ التعقيد باهظ الثمن ويحتاج من مختص قاعدة البيانات قدراً علياً و معرفياً كبيراً للتعامل إذ أنه يتولى السيطرة على العناصر الآلية والبرمجية للقاعدة بالتعاون مع نظام التشغيل فيما هو مبين بالشكل 1-18.



الشكل1-18

18.5 مستخدمو قواعد البيانات

وينقسمون إلى ثلاث فئات:

- مخططو البرامج الذين يكتب برامجهم و يستخدمون إمكانيات قاعدة البيانات.
- مختصو قواعد البيانات ، و هم المسئولون عن صيانة و تشغيل قاعدة البيانات.
- المستخدمون لقواعد البيانات الذين يتعاملون مع قاعدة البيانات عبر النهايات الطرفية.

18.6 تركيب قواعد البيانات

تتركّب قاعدة البيانات من مجموعةٍ من الجداول، هي التي يتم تخزين البيانات بها.. وطبعا أنت تعرف أنّ الجدول يتكوّن من أعمدة Columns (وأحيانا تسمّى حقول Fields) وصفوف Rows (وأحيانا تسمّى سجلات Records).. فمثلا، لو لديك قاعدة بيانات لمكتبة، فلا بدّ إنها ستحتوي على جدولٍ يحتوى على عناوين الكتب الموجودة في المكتبة (وليكن اسمه

Titles)، وجدولٍ آخر يحتوى على أسماء مؤلّفي هذه الكتب Authors ، وجدولا ثالثا يحتوى على أسماء الشركات التي نشرت هذه الكتب Publishers.

وفي جدول عناوين الكتب، يمكن إنشاء عدّة أعمدة لتخزي ن بعض المعلومات عن كلّ كتاب، مثل اسمه، وعدد صفحاته، وموضوعه... إلخ.

ولكّنا لن نضع اسم مؤلّف الكتاب في جدول عناوين الكتب، وذلك لسبب جوهريّ: ذلك أنّه من الممكن أن يحتوى الجدول على أكثر من كتابٍ لنفس المؤلّف.. في هذه الحالة سيتمّ تكرار اسم المؤلّف أكثر من مرّة، ممّا سيعمل على زيادة حجم قاعدة البيانات، وإبطاء عمليّات البحث..

إذن ما الحلّ؟

الحلّ هو تخصيص جدولٍ للمؤلّفين، هذا الجدول يحتوي على رقم مسلسل ID لكلّ مؤلّف.. وفي جدول الكتب ننشئ عمودا نضع فيه رقم مؤلّف الكتاب.. فمثلا لو كان رقم (توفيق الحكيم) في جدول المؤلّفين هو 100 ، فإنّ خانة رقم المؤلّف في جدول العناوين لا بدّ أن تكون 100 لكلّ من الكتب التالية "شهرزاد" و"الأيدي الناعمة" و"أرني الله" و"شجرة الحكم".

ولكي تستوعب مقدار التوفير في الحجم، يجب أن تعرف أنّ كلّ حرف في النصّ يخزّن في وحدة واحدة Byte ، ممّا يعن ي أن النصّ " توفيق الحكيم" يحتاجُ إلى 12 وحدة تخزين، وتكراره أربع مرّات يعني أنّه يحتاج إلى 48 وحدة!.. بينما تخزين الأرقام رقم يصل إلى 32000 لن يحتاج إلا إلى وحدتين فقط، ممّا يعني أنّ تخزين رقم 100 أربع مرّات يحتاج إلى 8 وحدات فقط.. طبعا الفرق واضح.

وليتَ الأمر يقتصر على هذا.. إنّ إنشاء عمود يحتوي على أسماء المؤلّفين يقتضي تصميمه ليستوعب أطول اسم من هذه الأسماء.. افترض أنّ أطول اسم لمؤلّف هو 20 حرفا، في هذه الحالة لو كان جدول الكتب يحتوي على 30 ألف كتاب، فإنّ هذا معناه أن عمود أسماء المؤّلفينَ يحتاج إلى 30000 × 20 = 600 ألف وحدة، أي 600 كيلو بايت!

بينما لو استعضنا عن ذلك بعمود أرقام المؤلّفين، فإنّه سيحتاج إلى 60 ألف وحدة، أي 60 كيلو بايت فقط.

طبعا ستتساءل: ولكنّنا في كلّ الأحوال سننشئ عمودا لأسماء المؤلّفين بالإضافة لعمود رقم المؤلّف في جدول المؤلّفين.. نعم، ولكنّ عدد المؤّلفين قد لا يزيد عن 3 آلاف مؤلّف (باعتبار أنّ كلّ مؤلّف له عشر كتب)، وهو ما سيحتاج إلى 3000 × (20 + 2) = 66 كيلو بايت.

إذن فـ إنّ المجموع الكلّيَّ لمساحة التخزين في حالة تقسيم المعلومات على جدولين = 60 + 66 = 126 كيلو بايت، أي حوالي خمس المساحة المستهلكة عند وضع أسماء المؤلّفين في نفس جدول الكتب!

هذا بالإضافة إلى سرعة عمليّة البحث، فالكمبيوتر يكونُ أسرع بكثير عند البحث عن رقم 100 عنه عند البحث عن النصّ "توفيق الحكيم".

ليس هذا فحسب.. افترض أنّ الذي يدخل أسماء المؤلّفين أخطأ وكتب اسم (توفيق الحكيم) كالتالي: " تفيق الحكيم".. في هذه الحالة ما عليه إلا أن يعدّل الخطأ مرّة واحدة في جدول المؤلّفين.. أمّا لو كانت أسماء المؤلّفين في نفس جدو ل الكتب، فإنّ تكرار كتابة اسم المؤلّف

يجعل احتمالات الخطإ أكبر، بالإضافة إلى تضييع الوقت والجهد في كتابتها، وصعوبة تعديلها كلّها!
طبعا نفس هذا الكلام ينطبق على جدول الناشرين.

قاعدة هامّة:
كلّما كان ممكنا، قم بتقسيم البيانات التي تتكرّر أكثر من مرّة على أكـ ثر من جدول، خاصّةً إذا كانت هذه البيانات تستهلك مساحة تخزين كبيرة.. إنّ هذه العمليّة تسمّى التطبيع Normalization، ولها الكثير من القواعد، وإن كان من الممكن استنتاجها منطقيّا.

18.7 أنواع قواعد البيانات

توجد أنواع كثيرة من قواعد البيانات لكن أبرزها وأظهرها على الساحة ثلاثة أنواع رئيسية هي:

- قواعد بيانات علائقية RELATIONAL DATA BASE .
- قواعد بيانات شبكية NETWORK DATA BASE .
- قواعد بيانات هرمية التركيب HIERARCHICL DATA BASE .

النوع الأخير أصبح الأكثر استخدام اً و شيوعاً وتعتبر قاعدة البيانات،DBIV+DB
111 أقرب قواعد البيانات الكبيرة، و سوف نركز دراستنا على هذا النوع.

18.7.1 قواعد البيانات العلاقية

تعتبر قواعد البيانات العلاقية من أهم قواعد البيانات وتتصف بـإنها قاعدة بيانات يستقبلها المستخدمون على هيئة جداول وليس شيئاً آخر سوى الجداول فيما يوضحه الجدول(18-1)الملف المعبر عن بيانات الأشخاص المسموح لهم بالاستعارة (ملف الاستعارة).

جدول 18-1

رقم الهوية	الاسم الأول	اسم الوالد	اسم العائلة	تاريخ الميلاد
2131314	محمد	محمود	عليوه	1940-01-04
1435466	أحمد	عبدالله	سويلم	1976-07-17
4536436	إبراهيم	خليل	سعد	1970-07-13
8768686	إيهاب	سعدي	صبح	1976-11-13

ملحوظة: البيانات في الجدول السابق بيانات افتراضية وليس لها أصل من الواقع.
ملف أوعية مكتبية:

جدول 2-18				
رقم الإيداع	المؤلف	اسم الكتاب	الناشر	سنة النشر
434234234	جمال الغيطاني	الأدب والثقافة	الأهرام	1973
434443	أنيس منصور	حول العالم	النهضة	1962

و يفرض أن شخصاً له سجل في ملف البطاقات الموضح في الجدول الأول استعار كتاباً من مقتنيات المكتبة فإن العلاقة الناشئة عن هذا الإجراء يمكن توضيحها في الشكل 2-18.

شكل 2-18

و تعتبر قواعد البيانات العلاقية هذه العلاقة ملف جديد - جدول جديد- موصفاته أي حقوله كالتالي:

- رقم المستعير(الرقم القومي).
- رقم إيداع الكتاب.
- تاريخ الاستعارة.

يمكن الاستغناء عن حقل تاريخ الاستعارة في هذا الملف.
و فيما يلي بعض الملاحظات:

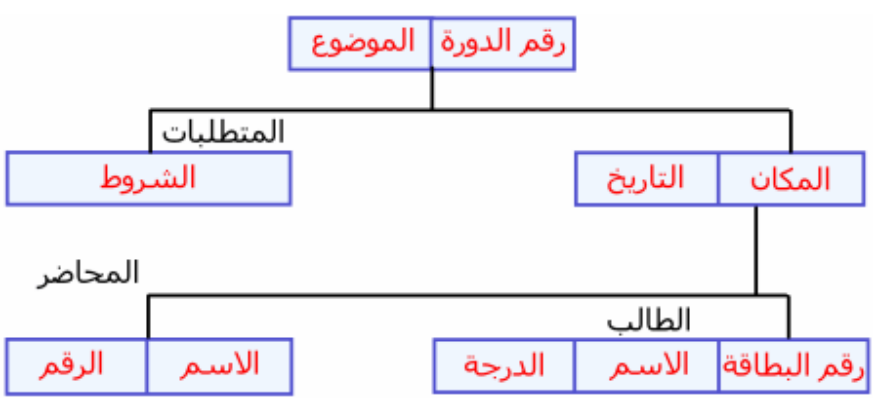
- الجدول في قواعد البيانات العلائقية يعادل الملف.
- الأعمدة تناظر الحقول.
- السطر يعادل السجل.
- لكل جدول مسمى وحيد.
- أن الجدول المسمى الاعارات نشأ بين العلاقة بين الفرد والكتاب.
- أن لكل مسجل في جدول المستعيرين حقل مفتاح(الرقم القومي) كذلك في جدول الكتب فإن حقل المفتاح هو(رقم التسجيل)
- أن رقم المستعير و رقم تسجيل الكتاب يمكن استخدام هما كحقل مفتاحي في الجدول(الملف) المسمى الإعارات.
- أن كل البيانات في الجداول الثلاث بيانات ذرية لا يمكن تفتيتها لأدنى من ذلك.
- كل القيم معبر عنها صراحة و ليس ضمناً.
- لا تستخدم مؤشرات إحالة فيما بين الملفات.

- **خصائص قواعد البيانات العلاقية**

1. كل ملف في قاعدة البيانات العلاقية يضم نوع واحد متكرر من السجلات.
2. ليس هناك ترتيب محدد للحقول.
3. ليس هناك ترتيب محدد للسجلات- سيان في قمة الجدول أو في أي مكان آخر منه.
4. لكل حقل قيمة واحدة فقط (لا تكرارية).
5. لكل سجل حقل مفتاح.
6. أوامر التعامل مع قاعدة البيانات لا تقتصر على الأربعة (اختار- حدث- احذف- ادخل) إنما تستخدم تعليمات أخرى مثل JOIN "صل" "اربط" وله شروط وأساس رياضي في التعامل مع هذا الأمر.
7. المنظر VIEW ليس ملفاً مخزناً على وسائط التخزين بل هو ملف وهمي يتخلق من الملفات الكائنة ولا يمكن تحديثه أو إنشاء فهارس عليه (الحقول) CREATE VIEW VIEW NAME (الحقول) AS SELECT (اسم الجدول) FORM (شروط) WHERE.

18.7.2 قواعد البيانات الهرمية HIERARCHICAL D.B

هي عبارة عن مجموعة مرتبة ومتكررة من نوع واحد من السجلات المركبة على هيئة شجرة، أي أن لكل سجل جذر ROOT واحد أي سجل واحد تتفرع منه هذه الفروع إلى سجلات و هكذا فيما يمكن تشبيهه بشجرة العائلة (الجد- الابن- الأبناء) ولا يسمح في قواعد البيانات الهرمية بأن يكون لأي سجل أكثر من واحد مثال قاعدة بيانات مركز تدريب فيما يوضحه الشكل(3-18).



الشكل 3-18 قواعد البيانات الهرمية

ونلحظ في هذا النوع من قواعد البيانات أنبعض البيانات التي يمكن إبرازها

في قواعد البيانات العلاقية باستخدام حقول مدمجة تظهر في هذه القاعدة بالروابط بين الأب والابن أي المستوى الأعلى و المستوى الأدنى .

ويختلف تحريك أو تداول معالجة البيانات في هذه القواعد عن العلاقية في أن تداولها يتطلب:

1. تسجيل جذر السجل.
2. وظيفة لتحريك البيانات من شجرة إلى أخرى.
3. معامل للحركة بين كل سجلات هذا التكون الشجري.
4. وظيفة لإضافة السجلات.
5. وظيفة لحذف السجلات.

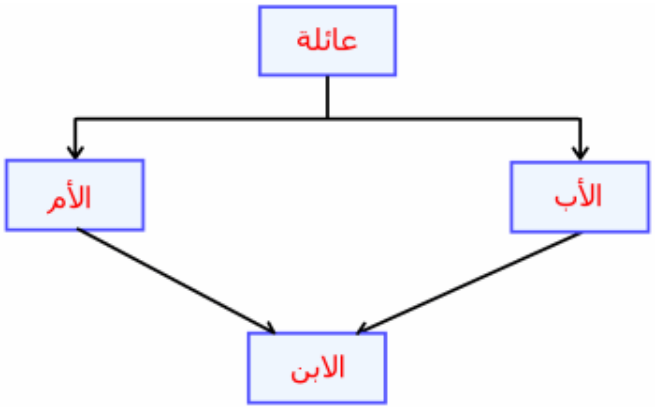
مثال عن أسلوب توصيف البيانات :

1- DBD NAME = EDUCP DBD
2- SEG NAME = COURSE , BYTES = 36
3- FIELD NAME = COURSE#, QES , BYTES = 3
4-FIELD NAME = TITLE , BYTES = 33 START 4
5- SEG

ويبدأ في توصيف سجل جديد و هكذا.

18.7.3قواعد البيانات الشبكية NET WORK

وهي برامج DBMS نتعامل مع السجلات ذات الارتباط المتعدد وهي أقرب قواعد البيانات للواقع إذ أنه من الصعب أن تكون العلاقات الطبيعية في الحياة على النظام فقط ويمكن تلخيص هذا النوع من العلاقات بأن المستوى الأدنى قد يكون له أكثر من اتصال بالمستوى الأعلى فيما يوضحه الشكل.(4-18)

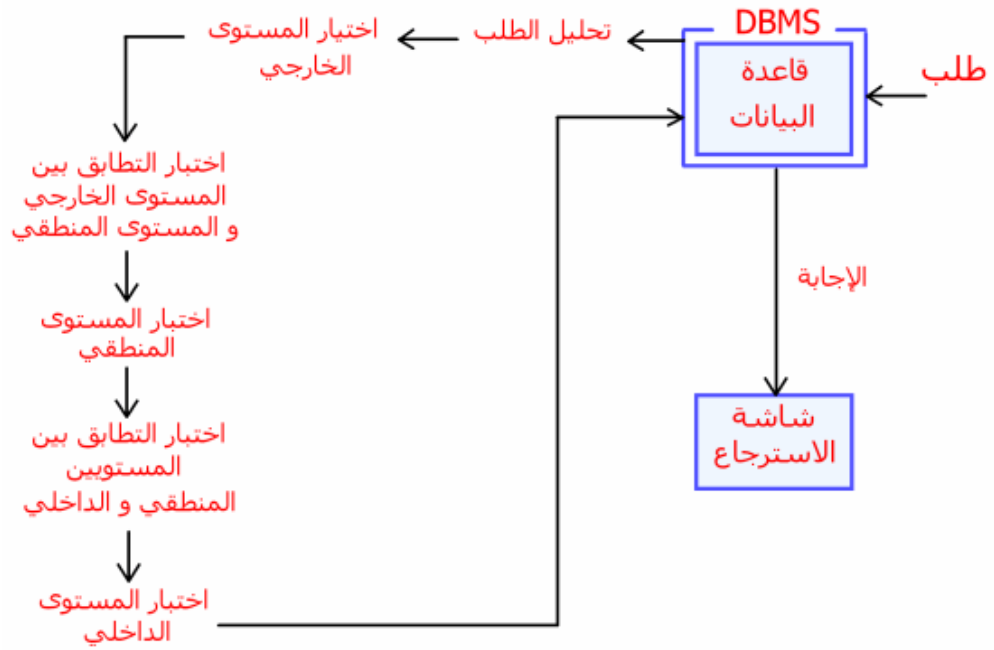


الشكل 18-4 قواعد البيانات الشبكية

18.8 دور DBMS عند طلب الاسترجاع

سبق و عرفنا أن DBMS أنه عبارة عن مجموعة برامج أو حزمة برامج يتم إعدادها و كتابتها بواسطة الشركات المنتجة للحاسبات أو شركات كتابة الكيان البرمجي للحاسبات و تعفي مستخدمي قواعد البيانات من مهام معقدة للولوج إلى البيانات و توصيفها و تحريكها وتخليق المستويات المختلفة.

ويتولى مدير قواعد البيانات وتحت إشرافه إدارة خطوات كثيرة و معقدة فيما يوضحه الشكل 18-5).



الشكل 5-18 يوضح خطوات الدور الذي يقوم به برنامج إدارة قاعدة البيانات

18.9 لغة الاستعلام المركّبة (SQL) Structured Query Language

ما هي لغة الاستعلام المركّبة؟

مهما كان نوع تطبيق قاعدة البيانات الذي تتعامل معه، فإنّه يستخدم لغة خاصّة بقواعد البيانات، متّفق عليها دوليّا، هي SQL.. صحيح أنّ هناك اختلافات في تركيب هذه اللغة ما بينَ تطبيق وآخر، ولكنّها اختلافات طفيفة لا تكفي للادعاء بإنها نسخ مختلفة تمام الاختلاف.
وبالمقارنة بلغات البرمجة المألوفة، فإنّ SQL لغة غير إجرائيّة Nonprocedural، بمعنى إنها لا تحتوي على تركيبات لغويةٍ مثل جمل الشرط وجمل التكرار وما شابهها.. إنّ SQL تعتبر لغة برمجة عالية المستوي، حيث يمكنها في سطر واحد إجراء عمليّا ت بليغة التعقيد على قاعدة البيانات.

18.9.1 أقسام جملة SQL

تنقسم جمل لغة SQL إلى طائفتين رئيسيّتين، تعتبر كلّ منهما لغة منفصلة:

1. لغة التعامل مع البيانات Data Manipulation Language (DML)

وتختصّ باسترجاع أو تحديث أو إضافة أو حذف السجلات التي تحقق شروطا معيّنة، لهذا تسمّى جمل هذه اللغة " استعلامات" Queries.. وتسمّى جمل استرجاع البيانات بـ "استعلامات التحديد" Selection Queries ، بينما تسمّى جمل الحذف والتعديل والإضافة بـ "استعلامات الفعل" Action Queries.

2. لغة تعريف البيانات Data Definition Language (DDL)

وتختصّ بإنشاء أو حذف كائنات قاعدة البيانات، مثل الجداول وما تحتويه من أعمدة وفهارس، ومثل العلاقات والقيود Constrains.

18.9.2 استعلامات التحديد Selection Queries

1. جملة التحديد SELECT Statement

تمكّنك جملة SELECT من استرجاع جزء من الجدول.. وأبسط صور هذه الجملة كالتالي:

SELECT أسماء الأعمدة
FROM أسماء الجداول

1- أسماء الأعمدة والجداول يفصل بينها علامة ",".
2- يمكن كتابة اسم مختصر للجدول بعد اسمه الحقيقيّ مباشرةً، و استخدام ه في باقي الجملة للتسهيل.
3- إذا كان اسم الجدول يحتوي على أيّ رموز غير مقبولة برمجيّا ـ مثل المسافات أو العلامات ":" أو "+" أو "=" ... إلخ ـ فيجب وضعه بين قوسين مضلّعين [ ].
4- إذا تشابهت بعض أسماء الحقول في أكثر من جدول، فيجب التمييز بينها بكتابة اسم كلّ حقل على الصيغة: (اسم الجدول.اسم الحقل).. وإذا كان اسم الحقل يحتوي على رموز غير مقبولة، فيجب استخدام الصيغة: (اسم الجدول.[اسم الحقل]).
5- ستظهر الحقول في السجلات المعادة بالترتيب الذي كتبت أسماءها به في جملة SELECT.
6- جمل SQL غير حسّاسة لحالة الأحرف، ولكن تُعُورفَ على كتابة الكلمات الرئيسيّة فيها بحروف كبيرة لسهولة تمييزها.

7- يمكن كتابة جملة SQL على أكثر من سطر، فهي ليست كأوامر VB تنتهي بنهاية السطر، ولكنّها كأوامر C++ تنتهي بالفاصلة المنقوطة ";" ، وإن كان من الممكن عدم كتابتها كاختصار، لأنّك في الغالب لن تكتب إلا جملة واحدة.
8- معظم الأخطاء التي تحدث في كتابة جملة SQL تنتج عن الخطإ في كتابة أسماء الحقول والجداول، فانتبه لذلك جيّدا.

فمثلا: لتحديد كلّ سجلات جدول المؤلفين يمكنك استخدام جملة كالتالية:

```
SELECT ID, Author
FROM Authors
```

ولكن ماذا لو كان الجدول يحتوي على عشرين عمودا، نريد استرجاعها كلّها.. هل سنكتب أسماء عشرين عمودا؟؟!!
طبعا لا.. بل إنّ الأمر سيكونُ أسهل من كتابة اسم عمود واحد، حيث سنستخدم العلامة "*" (أو كلمة ALL) كالتالي:

```
SELECT *
FROM Authors
```

ولو أردت استرجاع كلّ حقول جدول ما، مع حقل أو اثنين من جدول آخر، فاستخدم الصيغة التالية:

```
SELECT Authors.*, Books.Book
FROM Authors, Books
```

ويمكن اختصار أسماء الجداول في المثال السابق كالتالي:

```
SELECT A.*, B.Book
FROM Authors A, Books B
```

لو جرّبت هذا المثال فستفاجأ بنتائج غريبة، حيث سيتم تكرار كلّ سجلّ من سجلات المؤلفين مع كلّ أسماء الكتب، ممّا يبدو معه أنّ كلّ مؤلف قد ألّف كلّ الكتب!.. إنّ هذا خطأ متوقع، لأنّنا لم نضع أيّ شرط يربط بين حقول الجدولين، وهو ما سنتعلّمه حالا.

2. استخلاص السجلات التي تحقق شروطا معيّنة

- **فقرة "حيثُ" WHERE Clause:**

لم تخترع قواعد البيانات لكي تقرأ كلّ سجلات الجدول كما أدخلتها!!.. إنّ الأهمّ من ذلك قدرتك على حساب بعض النتائج واستخلاص المعلومات من هذه السجلات.. هنا يبرز الدور الحيويّ لفقرة WHERE ، فهي تسمح لك بتحديد الشروط التي سيتمّ على أساسها استرجاع السجلات، بحيث لا تسترجع منها إلا ما يحقق هذه الشروط.

ولن تكون صيغة هذه الفقرة صعبةً عليك، فستكب صيغة الشرط مماثلةً للشروط التي تعوّدت كتابتها في جملة If في VB.
تعال نسترجع كلّ الكتب التي يحمل مؤلّفها رقم 1 أو 3:

```
SELECT Book
FROM Books
WHERE AuthorID = 1 OR
        AuthorID = 3
```

ولكنّ من هو المؤلّف رقم 1؟
إنّ هذه الطريقة قد تكون عديمة الجدوى، فهي غير عمليّة في البرمجة.
إنّك تريد أن تعرف الكتب التي ألّفها (توفيق الحكيم) مباشرةً، دونَ أن تبحث عن رقمه.. بسيطة.. استخدم الجملة التالية:

```
SELECT Book
FROM Books, Authors
WHERE Author = 'توفيق الحكيم' AND
        AuthorID = Authors.ID
```

لاحظْ أنّنا كتبنا اسم جدول المؤلّفين في فقرة FROM رغم أنّ أيا من حقوله لن يظهر في النتيجة، وذلك لأنّنا سنستخدم حقول هذا الجدول في فقرة WHERE.
والآن هذا ما سيحدث: سيتمّ اختيار السجلات التي تحمل اسم (توفيق الحكيم) في جدول المؤلّفين.. ثمّ سيتمّ اختيار السجلات من جدول الكتب، التي يتساوى فيها رقم المؤلّف مع رقم المؤلّف في السجلات المختارة من جدول المؤلّفين.. ونظرا لأنّه لن يوجد سوى رقم (توفيق الحكيم) فحسب، فإنّ كتبه فقط هي التي سيتمّ عرضها.
وكان من الممكن تبديل ترتيب جملتي الشرط، فالترتيب ليس مهمّا، حيث يقوم تطبيق قاعدة البيانات بتنفيذ الشروط على حسب أولويّة تنفيذها، وليس على حسب ترتيب كتابتها.
والآن تعال نطوّر المثال السابق، لنحصل على كلّ الكتب التي ألّفها (توفيق الحكيم) أو (عباس العقاد).. في هذه الحالة لا بدّ من عرض اسم المؤلّف في النتيجة، وإلا لتعذّر معرفة مؤلّف كلّ كتاب:

```
SELECT Book, Author
FROM Books, Authors
WHERE (Author = 'توفيق الحكيم' OR Author = 'عباس العقاد') AND
        AuthorID = Authors.ID
```

لاحظ استخدام كلمة OR بين اسمي المؤلّفين، ولو استخدمت كلمة AND لما حصلت على أيّ نتائج، فلا يمكن أن يوجد جدول به سجلّ يحتوي على

خانة واحدة لها قيمتان!!.. ولكن يمكن أن يكون للخانة القيمة كذا أو القيمة كذا.. يمكن كذلك أن تكون لإحدى خانات السجلّ القيمة كذا و لخانة أخرى القيمة كذا.. فمثلا، لو كان لدينا في جدول الكتب كتاب اسمه "حياتي"، وحدث أن تكرّر اسمه في أكثر من سجلّ، نتيجة لأنّ هناك أكثر من مؤلّف له كتاب بهذا الاسم، فإن بإمكانك استرجاع الكتاب الذي يحقّق الشرط التالي:

```
SELECT ID
FROM Books, Authors
WHERE Author = 'كوتموتو' AND
        AuthorID = Authors.ID AND
        Book = 'جافا'
```

تلاحظ طبعا أنّ الهدف من الاستعلام قد اختلف، فعندما تبحث عن كتاب تعرف اسمه ورقم مؤلّفه، فلا بدّ أنّك لسبب أو لآخر تريد أن تعرف رقم الكتاب ID.

ولكن ماذا لو أردنا أن نعرض كلّ أسماء الكتب مع ما يناظرها من مؤلّفين؟

سيكون الأمر في منتهى البساطة، فسيقتصر الشرط في هذه الحالة على تساوي رقم المؤلّف في الجدولين:

```
SELECT Book, Author
FROM Books, Authors
WHERE AuthorID = Authors.ID
```

ملحوظة1:

يمكنك عرض الجدول فارغا بوضع شرط خاطئ أو مستحيل في فقرة WHERE (مثل 1 = 2)!!، كالتالي:

```
SELECT *
FROM Books
WHERE 1 = 2
```

ملحوظة2:

لكي تنشئ فقرة WHERE بباني الاستعلام، أضف للاستعلام الحقل الذي سيظهر في هذه الفقرة (بضغط خانة الاختيار المجاورة لاسمه في الجدول).. سيضاف هذا الحقل إلى جدول الحقول.. اضغط هذا الحقل ـ في جدول الحقول ـ بزرّ الفأرة الأيمن، ومن القائمة الموضعيّة اضغط الأمر Group By.. سيظهر عمود جديد يحمل اسم Group By.. اضغط الخانة المناظرة للحقل في هذا العمود.. سيظهر لك زرّ إسدال القائمة.. اضغطه بالفأرة، ومن القائمة الموضعيّ ة اختر WHERE.. وفي العمود التالي مباشرة ـ ذلك الذي يحمل اسم Criteria ـ اكتب الشرط الذي تريده، بدون

كتابة اسم الحقل (كأن تكتب 5 < مثلا).. فإذا أردت أن تكتب شرطا آخر مرتبط بالشرط الأوّل بالمعامل OR ، فانتقل إلى العمود التالي.. ستجده يحمل عنوان OR.. اكتب الشرط الذي تريده في هذا العمود.. فإذا كان هناك المزيد من الشروط، فلديك المزيد من الأعمدة التي تحمل العنوان OR.

ولكن ماذا لو أردت أن تكتب شرطا يرتبط بالشروط السابقة بالمعامل AND؟

في هذه الحالة يجب أن يظهر هذا الشرط في صفّ جديد.. طبعا لن تجد مشكلة إذا كان الشرط الجديد ينطبق على حقل مختلف.. ولكنّ المشكلة هي أنّ يظهر نفس الحقل في أكثر من شرط يربطها المعامل AND!

بسيطة.. اضغط بالفأرة في أيّ خانة فارغة في العمود Column ، ومن القائمة المنسدلة اختر نفس اسم الحقل.. هذه الطريقة تمكّنك من تكرار اسم الحقل أكثر من مرّة في جدو ل الحقول.. في الصفّ الجديد اختر كلمة WHERE في العمود Group By، واكتب الشرط في خانة Criteria.

ولكن ماذا لو كان هناك شرطان كلّ منهما على حقل مختلف، ويربطهما المعامل OR؟

في هذه الحالة يجب أن تكتب الشرط الثاني في خانة OR ، مع ترك خانة Criteria فارغة.

وكتلخيص: ا لشروط التي تظهر في خانة Criteria في صفوف مختلفة يربطها المعامل AND ، بينما الشروط التي تظهر في خانة OR ، حتّى لو كانت في صفوف مختلفة، يربطها المعامل OR.

- **المعاملات < > , =, <, >**

كأيّ جملة شرط عاديّة، يمكنك استخدام معاملات المقارنة التقليديّة < > , =, <, >.. فمثلا للحصول على كلّ الكتب ما عدا ذلك الذي يحمل رقم 9، استخدم الجملة التالية:

```
SELECT ID, Book
FROM Books
WHERE ID <> 9
```

وللحصول على الكتب التي تسبق كتاب "حائرة في الحب" فالترتيب الهجائيّ، استخدم الجملة التالية:

```
SELECT ID, Book
FROM Books
WHERE Book < Java'
```

وللحصول على الكتب التي تحمل أرقاما أكبر من 12 بالإضافة إلى الكتاب رقم 2 استخدم الجملة التالية:

```
SELECT ID, Book
FROM Books
WHERE ID = 2 OR
          ID > 12
```

- **بين BETWEEN**

يسمح لك هذا المعامل بتحديد المجال الذي ينتمي إليه الحقل.. فمثلا، يمكننا استخدام الجملة التالية للحصول على الكتب التي ينحصر رقمها بين 3 و 10:

```
SELECT ID, Book
FROM Books
WHERE ID BETWEEN 3 AND 10
```

كما يمكننا استخدام NOT قبل هذا المعامل للحصول على قيم الحقل التي لا تنتمي للمجال المحدّد.. والجملة التالية تعيد الكتب التي لا تنحصر بين 3 و 10:

```
SELECT ID, Book
FROM Books
WHERE ID NOT BETWEEN 3 AND 10
```

- **في IN**

أحيانا لا يحلّ المعامل BETWEEN كلّ مشاكلنا، فماذا لو أردنا أن نختار قيما متفرّقة للحقل؟.. في هذه الحالة ستكون جملة AND طويلة وبطيئة التنفيذ.

في هذه الحالة يمكن استخدام المعامل IN ووضع كلّ القيم بين قوسين مفصولةً بعلامة , .. والجملة التالية تعيد لك الكتب التي تحمل الأرقام 2 و 6 و 7 و 10:

```
SELECT ID, Book
FROM Books
WHERE ID IN (2,6,7,10)
```

ويمكن نفي الجملة السابقة للحصول على باقي الكتب (التي لا تحمل الأرقام المذكورة):

```
SELECT ID, Book
FROM Books
WHERE ID NOT IN (2,6,7,10)
```

ولا يقتصر الأمر على الأرقام، فالجملة التالية تعيد كلّ الكتب التي ألفها (توفيق الحكيم) و(نبيل فاروق) و(محمد حمدي غانم):

```
SELECT Book, Author
FROM Books, Authors
WHERE AuthorID = Authors.ID AND
          Author IN ('توفيق الحكيم', 'نبيل فاروق' , 'محمد حمدي غانم')
```

ولا مانع من تكوين جملة شرط معقّدة تحتوي على مزيج من كلّ هذه المعاملات معا.. المهمّ أن تصل لهدفك من أقصر طريق.

v المعامل "يشبه" LIKE Operator

بعض الاختلافات الطفيفة ستدركها عند التعرّف على العلامات الخاصّة التي يستخدمها هذا المعامل، وهي في الجدول 3-18

جدول 3-18

علامة SQL	الوظيفة
%	تعبّر عن أيّ عدد من الحروف (يمكن أن يكون هذا العدد صفرا)، مهما كانت هذه الحروف. مثال: استخدم الجملة التالية للحصول على جميع أسماء المؤلّفين التي تحتوي على حرف الميم في أيّ موضع: SELECT Author FROM Authors WHERE Author LIKE '%م%' ولو أردت أن تبحث عن السجلات التي تبدأ بحرف الميم، فاستخدم الصيغة 'م%'.
_ (الشرطة المنخفضة (Underscore	تعبّر عن حرف واحد فقط يجب أن يكون موجودا في الخانة السجلّ في الموضع المناظر لهذا العلامة في الصيغة، مهما كان هذا الحرف1. مثال: استخدم الجملة التالية للحصول على جميع أسماء المؤلّفين التي يكون حرف الميم فيها هو ثالث حرف: SELECT Author FROM Authors WHERE Author LIKE "__م%' حيث استخدمنا علامتي "_" لتحفظ موضع حرفين (أيّ حرفين) يليهما الحرف الثالث وهو الميم، يليه العلامة % لتدّل على أنّ أيّ عدد من الحروف مهما كانت يمكن أن يأتي بعد حرف الميم، بما في ذلك أن يكون حرف الميم هو آخر حرف في النصّ ولا يليه أية حروف.
#	تعبّر عن رقم واحد فقط من 0 إلى 9.
[ ]	تعبّر عن حرفٍ واحد من الحروف الموجودة بين القوسين.. هذه الحروف إمّا أن تُكتب متتالية مثل [ACdF] ، وإمّا أن تُكتب على صورة مجال، مثل [g-y] حيث تعبّر هذه الصيغة عن

	الحروف من g إلى y. مثال: استخدم الجملة التالية للحصول على جميع أسماء المؤلفين التي يكون حرف الميم أو النون أو الواو فيها هو ثالث حرف: SELECT Author FROM Authors WHERE Author LIKE "__من%' ولو أردت أن تبحث عن السجلات التي تبدأ بأحد الحروف المحصورة بين الفاء والياء وتنتهي بحرف السين، فاستخدم الصيغة '[ف-ي]%س'.
[^]	تعبّر عن أيّ حرفٍ غير ذلك الموجود في القوسين (سواء من الحروف المذكورة صراحة أو من الحروف التي تقع في المجال المحدّد).

ملحوظة:
إذا أردت البحث عن أيّ علامة من هذه العلامات الخاصّة في النصّ، فضعها بين قوسين مضلّعين.. فمثلا، يمكنك استخدام الصيغة '%[%]50%' للبحث عن النصّ "50%" في أيّ موضع من الحقل.

- **المقارنة بحقل من جدول**

من الإمكانيّات التي تمنحها لك فقرة WHERE ، قدرتك على مقارنة قيمة أحد الحقول، بقيمة أيّ حقل من جدول آخر، ناتج عن جملة SELECT كاملة أخرى!
انظر للمثال التالي، وفيه نحصل على كلّ الكتب التي ألّفها (عباس العقاد) و(توفيق الحكيم):

```
SELECT *
From Books
WHERE AuthorID = (
                    SELECT ID
                    FROM Authors
                    WHERE Author = 'عباس العقاد' OR
                          Author = 'توفيق الحكيم'
                  )
```

حيث تقوم تعيد SELECT الفرعيّة حقلا به رقمي هذين المؤلفين، لتقوم جملة SELECT الخارجيّة بعرض الحقول التي تحتوي على أيّ من هذين الرقمين في حقل AuthorID.

v كلمة AS

يمكنك استخدام كلمة AS لوضع الاسم المستعار للحقول التي تريدها، وذلك كالتالي:

```
SELECT Book AS [اسم الكتاب], Author AS [مؤلفه]
FROM Books, Authors
WHERE AuthorID = Authors.ID
```

الآن ستظهر نفس النتائج السابقة، ولكنّ العمودين سيحملان اسمين عربيّين هذه المرّة.

ومن الاستخدامات الطريفة للاسم المستعار، استخدامه لتسميّة أحد الحقول الناتجة عن دمج حقلين معا.. ما موضوع هذا الدمج؟

افترض أنّنا نريد عرض حقل يحتوي على اسم الكتاب متبوعا بشرطة متبوعة باسم المؤلف.. كلّ ما علينا فعله إذ ذاك، هو توصيل concatenate الحقلين، تماما كما نفعل مع المتغيّرات النصيّة في VB ، مع منح الحقل الناتج اسما مستعارا باستخدام كلمة AS، كالتالي:

```
SELECT Book + '_' + Author AS [اسم الكتاب]
FROM Books, Authors
WHERE AuthorID = Authors.ID
```

ملحوظة:

عند تنفيذ الجملة السابقة في باني الاستعلام سيتم وضع التعبير Book + '_' + Author كحقل في جدول الحقول، مع منحه الاسم المستعار "اسم الكتاب".. هذه هي الطريقة التي تضيف بها الحقول المدمجة في باني الاستعلام.. اكتب العمليّة التي تجمع بين الحقلين في أيّ خانة فارغة في العمود Column ، وامنح هذا الحقل الجديد الاسم المستعار المناسب.

وليست الحقول النصيّة فقط هي ما نستطيع دمجه، حيث يمكننا أن نجرى عمليّات الجمع والطرح والضرب والقسمة على الحقول الرقميّة كما يحلو لنا.

v الكلمة TOP

يمكنك استخدام هذه الكلمة إذا كان عدد السجلات الناتجة عن تنفيذ الاستعلام كبيرا، بينما ينحصر اهتمامك في مجموعة قليلة منها فقط.. فمثلا، لكي تعرض أوّل خمس سجلات من أسماء الكتب ومؤلّفيها استخدم الجملة التالية:

```
SELECT TOP 5 Book, Author
FROM Books, Authors
WHERE AuthorID = Authors.ID
```

وإذا لم تكن تعرف بالضبط عدد السجلات المطلوبة، فيمكنك استخدام النسبة المئويّة كالتالي:

```
SELECT TOP 5 PERCENT Book, Author
FROM Books, Authors
WHERE AuthorID = Authors.ID
```

- **الكلمة "منفصل" DISTINCT**

استخدم هذه الكلمة لإزالة السجلات المكرّرة.. فمثلا لو عرضنا جدولا بأسماء الكتب بدون مؤلّفيها، فإنّ كتاب "حياتي" سيتكرّر 3 مرّات.. هنا يمكن تلافي هذا التكرار باستخدام الجملة التالية:

```
SELECT DISTINCT Book
FROM Books
```

- **القيم المنعدمة Null Values**

هناك اختلاف جوهريٌّ بينَ أن يكونَ النص فارغا "" أو القيمة العدديّة صفرا، وبينَ أن تكون لهما القيمة Null ، حيثُ إنها تشيرُ في هذه الحالة إلى أنّ هذه الخانة لن توضع بها أيّ قيمة من أساسِه.. إنّ مثل هذه الحالة تؤدّي إلى تبعاتٍ غير مرغوبٍ فيها، ففقرة WHERE مثلا لا تسترجع أيّ خانة فارغة، لأنّ القيمة NULL تنتج False بطريقة دائمة في أيّ عمليّة مقارنة.. هذا بالإضافة إلى أنّ دخول هذه القيمة في أيّ عمليّة حسابيّة يؤدّى لحدوث خطإ في البرنامج.

إذا ما الحل؟

الحلّ هو أن تعامل هذه القيمة بحذر، حيث يمكنك أن تختبر وجودها باستخدام التعبير IS NULL أو IS NOT NULL.. وكمثال، استخدم الاستعلام التالي للبحث عن أيّ خانة في حقل المؤلّفين تمّ تركها فارغة:

```
SELECT * FROM Authors WHERE Author IS NULL
```

عامّةً يمكن تلافي الكثير من احتمالات الخطأ بوضع القيمة False في خاصيّة "السماح بطول صفريّ" Allow Nulls عند تصميم الحقول.

- **فقرة الترتيب ORDER BY**

يمكنك ترتيب السجلات الناتجة من الاستعلام تبعا لحقلٍ أو أكثر.. وفي المثال التالي سنعرض أسماء الكتب ومؤلّفيها مرتّبة باسم الكتاب:

```
SELECT Book, Author
FROM Books, Authors
ORDER BY Book
```

```
WHERE AuthorID = Authors.ID
```

ولكنّ هناك ثلاث كتب تحمل اسم "حياتي" فكيف سيتمّ ترتيبها؟

في هذه الحالة يمكنك أن تحدّد حقل المؤلّفين كمفتاح ثانٍ للترتيب، بحيث لو تشابهت أسماء الكتب، يتم ترتيب الكتب المتشابهة على حسب أسماء مؤلّفيها:

```
SELECT Book, Author
FROM Books, Authors
ORDER BY Book, Author
WHERE AuthorID = Authors.ID
```

ولكنّ هذا الترتيب تنازليّ في الوضع الافتراضيّ.. ماذا لو أردت أن يتمّ الترتيب تصاعديّا؟

في هذه الحالة أضف الكلمة DESC بعد اسم أيّ حقلٍ تريد ترتيبه تنازليّا، والكلمة ASC بعد ذلك الذي تريد ترتيبه تصاعديّا (وهي افتراضيّة ويمكن عدم كتابتها)، كالتالي:

```
SELECT Book, Author
FROM Books, Authors
ORDER BY Book DESC, Author ASC
WHERE AuthorID = Authors.ID
```

ملحوظة:

لتحديد كيفيّة الترتيب في باني الاستعلام، اضغط الحقل الذي تريد أن يتمّ الترتيب على أساسه بزرّ الفأرة الأيمن، ومن القائمة الموضعيّة اضغط الأمر Sort Ascending إذا كنت تريد الترتيب تصاعديّا، أو الأمر Sort Descending إذا كنت تريد الترتيب تنازليّا.. ستجد أنّ أيقونة تمثّل نوع الترتيب قد ظهرت بجوار اسم الحقل.. ولو أردت إزالة الترتيب، فاضغط نفس الأمر من القائمة الموضعيّة مرّةً أخرى.. كما أنّ بإمكانك فعل ذلك مع أكثر من حقل ومن أيّ جدول، حيث ستظهر في جملة SQL بالترتيب الذي أضفتها به.

كما يمكن اختيار نوع الترتيب بطريقة أخرى، وذلك باستخدام جدول الحقول Grid Pane ، حيث يمكنك تغيير قيمة هذه الخاصيّة لأيّ حقل من العمود الذي يحمل العنوان Sort Order.

18.9.3 إجراء العمليّات على خانات الحقول

- **دوال التجميع Aggregate functions**

تمنحك SQL بعض الدوال الجاهزة لحساب وتلخيص بعض النتائج.. ويجب أن تلاحظ هنا ما يلي:

- أنّ هذه الدوال تقبل معاملا واحدا فقط، هو أحد أعمدة الجدول، أو أيّ عمود جديد ناتج عن إجراء إحدى العمليّات الحسابيّة (كالجمع والطرح والضرب والقسمة) على واحد أو أكثر من الأعمدة.
- أنّ هذه الدوال تعيد قيمة واحدة فقط (رقم).. أيّ أنّ الناتج منها هو عمود يحتوي على خانة واحدة فقط.. ولو كنت ستعرض هذه النتيجة، فاستخدم تعبير AS لـ منح هذا العمود الجديد اسما مناسبا، وإلا فإنّ SQL ستمنحه اسما افتراضيّا من لديها.
- لا مانع من استخدام فقرة WHERE لتحديد السجلات التي سيتمّ عليها العمليّة الحسابيّة.

وهذه الدوال هي

COUNT() لحساب عدد الخانات في العمود المرسل كمعامل.. ويمكن تطبيقها على أيّ نوع من البيانات.

مثال: الجملة التالية تحسب عدد الكتب التي تنتمي لتوفيق الحكيم:

```
SELECT COUNT(Book) AS [عدد الكتب المتاحة]
FROM Books, Authors
WHERE Author = 'توفيق الحكيم' AND
      AuthorID = Authors.ID
```

لاحظْ أن الحقول التي تحتوي على القيمة NULL لا يتمّ عدّها ضمنَ السجلات.. ولو أردت أن تفعل العكس، فعليك أن ترسل الرمز (*) كمعامل لهذه الدالة، حتّى تأخذ هذه السجلات في اعتبارها:

```
SELECT COUNT(*) AS [عدد المؤلفين]
From Authors
```

SUM() لحساب مجموع الخانات في العمود المرسل كمعامل.. ويمكن تطبيقها على الأعمدة الرقميّة فقط.

مثال: لسبب غير واضح، قرر شخصٌ ما مهووسٌ بالأرقام، أن يجمع عمود رقم الكتاب على عمود رقم المؤلف في جدول الكتب، وأن يحسب المجموع الكلّي للعمود الجديد.. هذا هو ما فعله بالضبط:

```
SELECT SUM(ID + AuthorID) AS [مجموع كلّ أرقام الجدول]
From Books
```

AVG() لحساب المتوسّط الحسابيّ (المجموع ÷ العدد) لخانات العمود المرسل كمعامل.. ويمكن تطبيقها على الأعمدة الرقميّة فقط.

وقد قام نفس الشخص المهووس بحساب متوسّط الأرقام، بوضع الدالة AVG بدلا من SUM في المثال السابق!

MIN() لحساب أصغر قيمة في العمود المرسل كمعامل.. ويمكن تطبيقها على

الأرقام والنصوص، حيث في حالة النصوص تعيد أصغر نصّ في الترتيب الأبجديّ.

MAX() لحساب أكبر قيمة في العمود المرسل كمعامل.. ويمكن تطبيقها على الأرقام والنصوص، حيث في حالة النصوص تعيد أكبر نصّ في الترتيب الأبجديّ.

الجدير بالذكر أنّ بإمكانك استخدام القيم المعادة من هذه الدوال في شروط فقرة WHERE.. والمثال التالي يريك كيف نحسب عدد الكتب التي ألّفها أوّل مؤلّف في الترتيب الأبجديّ:

```
SELECT COUNT(Book) AS [عدد كتب المؤلّف الأول في الترتيب الهجائيّ]
FROM Books, Authors
WHERE AuthorID = Authors.ID AND
        Author = (SELECT MIN(Author) From Authors)
```

حيث استخدمنا جملة SELECT كاملة تعيد ناتج دالة القيمة الصغرى في شرط فقرة WHERE.

عامّة نفس هذه الجملة لها صيغة ثانيةٌ، باستخدام الربط الداخليّ INNER JOIN الذي سنتعرّف عليه لاحقا.

ولكن ماذا لو أردت أن تطبّق هذه الدوال على حقل به قيم مكرّرة، وأردت ألا تأخذ التكرار في اعتبارك؟

في هذه الحالة يجب أن تستخدم كلمة DISTINCT للحصول على حقل ليس به أيّ تكرار.. ولكن أين سيكون موضع هذه الكلمة؟

ستكون بالطبع قبل اسم الحقل مباشرةً، أي داخل قوس الدالة.. والمثال التالي يريك كيف نحسب عدد المؤلّفين بحساب عدد خانات الحقل AuthorID في جدول الكتب بدون تكرار:

```
SELECT COUNT (DISTINCT AuthorID) AS [عدد المؤلّفين]
From Books
```

ملحوظة:

لاستخدام دوال التجميع في باني الاستعلام، أضف للاستعلام الحقل الذي تريد تطبيق الدالة عليه (بضغط مربّع الاختيار المجاور له في الجدول).. سيظهر اسم هذا الحقل في جدول الحقول.. اضغطه بزرّ الفأرة الأيمن، ومن القائمة الموضعيّة اضغط الأمر Group By.. سيظهر عمود جديد في جدول الحقول اسمه Group By.. اضغط الخانة المناظرة للحقل في هذا العمود بالفأرة.. سيظهر لك زرّ إسدال القائمة.. اضغطه بالفأرة.. ستظهر لك قائمة منسدلة بها أسماء الدوال التي يمكنك استخدام ها.. اختر منها ما تريد.

- **تجميع السجلات باستخدام تعبير GROUP BY**

رأينا كيف نجري بعض العمليّات الحسابيّة باستخدام دوال التجميع.. ولكن ماذا لو أردنا أن نحل على سجلّ يحتوي على عدد الكتب التي ألفها كلّ مؤلف مثلا؟
في هذه الحالة لن تسعفنا التعبيرات التي تعلمتاها حتّى الآن.
هنا ستبرز أهمّية التعبير GROUP BY ، حيث يؤدّي إلى ضمّ كلّ الصفوف ذات القيم المتشابهة في الحقول المحدّدة معا، بحيث تعمل دوال التجميع على كلّ منها كإنها جدول مستقلّ، وبهذا يكون ناتج دوال التجميع في هذه الحالة عمودا يحتوي على مجموعة من الصفوف، وليس صفّا واحدا كما ألفنا من قبل.
هذه هي الجملة التي تحسب عدد كتب كلّ مؤلف:

```
SELECT AuthorID, COUNT(AuthorID) AS [عدد الكتب]
FROM Books
GROUP BY AuthorID
ORDER BY [عدد الكتب]
```

لاحظ قدرتنا على استخدام الاسم المستعار للعمود الناتج في جملة SQL.
ولكن.. الجدول الناتج من الاستعلام السابق يحتوي على أرقام المؤلفين وليس أسماءهم.. فما العمل؟
إنّ هناك قاعدةً تقضي بأنّه عند استخدام فقرة GROUB BY يجب أن تكون كلّ الحقول التي تظهر في فقرة SELECT مستخدمة إمّا في دالة التجميع أو في فقرة GROUB BY.. إنّ هذا يعني أنّنا لا نستطيع استخدام الحقل Author مباشرةً..
إذن ما العمل؟
بسيطة.. أليس ناتج الاستعلام السابق جدولا؟. تعال نعامله كما عاملنا جدول الكتب من قبل، حيث سنستخدم جملة WHERE لربط رقم المؤلف باسمه بين هذا الجدول المشتقّ، وبين جدول المؤلفين.
ولكن كيف سنتعامل مع هذا الجدول الوهميّ الجديد؟
ربما ستستغرب ذلك، ولكنّه غاية في البساطة.. ضع كلّ جملة الاستعلام السابقة التي تُنتج هذا الجدول بين قوسين، وضعها في جملة FROM كإنها اسم جدول!
شيءٌ واحد فقط سنحتاج لتعديله قبل أن يعمل هذا الأمر، هو حتميّة كتابة فقرة TOP في تعبير مثل هذا الجدول المشتقّ.. وكحيلة بسيطة، دعنا نعرض 100% من الجدول.. انظر للجملة التالية:

```
SELECT Author, [عدد الكتب]
FROM Authors, (
SELECT TOP 100 PERCENT
AuthorID, COUNT(AuthorID) AS [عدد الكتب]
FROM Books
GROUP BY AuthorID
ORDER BY [عدد الكتب]
```

)
WHERE AuthorID = Authors.ID

إنّ هذه الإمكانيّة الرائعة تسمح لك بكتابة جمل SELECT متداخلة، لأداء استعلامات غاية في التعقيد.

عامّةً هناك طريقة أخرى لحلّ المشكلة السابقة، تعتبر أبسط وأسهل، وذلك بوضع الحقل Author في فقرة GROUP BY ولكن بعد إضافة فقرة WHERE، كالتالي:

```
SELECT Author, COUNT(AuthorID) AS [عدد الكتب]
FROM Authors, Books
WHERE AuthorID = Authors.ID
GROUP BY Author
ORDER BY [عدد الكتب]
```

فكأنّنا كوّنا جدول الكتب وأسماء مؤلّفيها أولا، ثمّ استخدمنا فقرة GROUP BY على الحقل Author لحساب عدد مرات تكراره لكلّ مؤلّف.

ولا غبار عليك لو استخدمت أكثر من دالة تجميع مع فقرة GROUP BY.

ملحوظة:

عرفنا كيف نستخدم دوال التجميع في باني الاستعلام.. ورغم أنّنا اخترنا الدالة من العمود الذي يحمل العنوان Group By، إلا إنّ فقرة Group By لم تظهر في جملة SQL.. كيف إذن نضيف هذه الفقرة؟

يمكن أن تفعل ذلك بنفس طريقة إضافة دوال التجميع، ولكن مع ترك قيمة الخانة كما هي Group By.

ولكن ماذا لو أردنا أن نضيف دالة تجميع وفقرة Group By لنفس الحقل؟

بسيطة.. بمجرّد اختيار دالة التجميع للحقل، ستجد أنّ علامة الاختيار المجاورة لاسمه في جدوله الأصليّ قد اختفت.. ضعها مرّة أخرى.. ستجد أنّ اسم الحقل قد ظهر مرّة أخرى (مكرّرا) في جدول الحقول، وستجد أنّ قيمة العمود Group By هي Group By.. اتركها كما هي، وبهذا تحلّ المشكلة!.. ويمكنك أن تكرّر اسم نفس الحقل في جدول الحقول بطريقة أخرى.. اضغط خانة فارغة في عمود الحقول.. سيظهر لك زرّ إسدال القائمة.. اضغطه بالفأرة، ومن القائمة اختر اسم الحقل.

- **استخدام فقرة HAVING**

ولكن ماذا لو أردنا عرض أسماء المؤلّفين الذين تزيد كتبهم عن كتاب واحد؟

في هذه الحالة يمكننا استخدام فقرة HAVING التي تسمح باستخدام دوال التجميع، كالتالي:

```
SELECT Author, COUNT(AuthorID) AS [عدد الكتب]
```

```
FROM Authors, Books
WHERE AuthorID = Authors.ID
GROUP BY Author
HAVING COUNT(AuthorID) > 1
ORDER BY [عدد الكتب]
```

3. ربط الجداول SQL Joins

تحدّد عمليّات ربط الجداول Joins كيفيّة توصيل أكثر من جدول معا في استعلام واحد.. وهناك أربعة أنواع من عمليّات الربط:

- **الربط الأيسر Left Joins**

هذه العمليّة تعرض كلّ سجلات الجدول الأيسر (الموجود في بداية الصيغة) ، مع بعض سجلات الجدول الأيمن، التي تحقق شرطا معيّنا.
وكمثال: يمكننا أن نعرض كلّ الكتب من جدول الكتب، مع أسماء المؤلّفين المناظرة لهذه الكتب، كالتالي:

```
SELECT Book, Author
FROM Authors LEFT JOIN Books
ON AuthorID = Authors.ID
```

ملحوظة1:
لكي تشعر بوجود اختلاف عن ناتج جملة WHERE ، يجب أن يكون هناك بعض المؤلّفين الذين لا توجد لهم كتب مناظرة في جدول الكتب، حيث سيتمّ عرض أسمائهم أيضا في هذه الحالة، مع ترك خانة الكتب فارغة NULL.. هذا بالإضافة لعرض باقي الكتب وما يناظرها من مؤلّفين.

ملحوظة2:
لإنشاء الجملة السابقة بـ استخدام باني الاستعلام، حدّد الحقل Book من جدول الكتب، والحقل Author من جدول المؤلّفين، ثمّ اضغط بزرّ الفأرة الأيمن على علامة الربط في منتصف الخطّ الواصل بينَ الجدولين، ومن القائمة الموضعيّة اضغط Property Pages.. ستظهر لك الصفحة التالية:

اختر الاختيار All Rows From Books، و لا تختر الاختيار All Rows From Authors.. أغلق النافذة.. انتهى!

- **الربط الأيمن Right Joins**

هذا النوع مماثل للنوع السابق، إلا إنّ كلّ سجلات الجدول الأيمن سيتمّ عرضها بالكامل، مع عرض سجلات الجدول الأيسر التي تحقّق شرطا معيّنا.

```
SELECT Book, Author
FROM Authors RIGHT JOIN Books
ON AuthorID = Authors.ID
```

ملحوظة1:

لكي تشعر بوجود اختلاف عن ناتج جملة WHERE ، يجب أن يكون هناك بعض الكتب التي لا يوجد لها مؤلف مناظر في جدول المؤلفين، حيث سيتمّ عرض أسمائها أيضا في هذه الحالة، مع ترك خانة المؤلف فارغة NULL.. هذا بالإضافة لعرض باقي الكتب وما يناظرها من مؤلفين.

حريٌّ بي أن أذكّرك، أنّنا عندما أنشأنا الحقل AuthorID أزلنا علامة الاختيار من خاصيّة Allow Nulls ممّا لن يسمح لك بترك خانة رقم المؤلف فارغة.. عامّة يمكنك أن تغيّر قيمة هذه الخاصيّة وتجرّب إدخال كتاب بدون مؤلف، لترى تأثير عمليّة الربط الأيمن.

ملحوظة2:

لإنشاء الجملة السابقة باستخدام باني الاستعلام، اتبع نفس الخطوات التي اتبعناها في إنشاء الربط الأيسر، ولكن أزل الاختيار All Rows From Books، واختر الاختيار All Rows From Authors.

- **الربط الكامل Full Joins**

هذا النوع هو مزيج من النوعين السابقين، وفيه يتمّ عرض كلّ بيانات الجدولين:

```
SELECT Book, Author
FROM Authors FULL JOIN Books
ON AuthorID = Authors.ID
```

ملحوظة1:

لكي تشعر بوجود اختلاف عن ناتج جملة WHERE وعن ناتج العمليّتين السابقتين، يجب أن يكون هناك بعض المؤلفين الذين لا توجد لهم كتب، وبعض الكتب التي لا يوجد لها مؤلف، حيث سيتمّ عرض أسماء هؤلاء المؤلفين وهذه الكتب، مع ترك الخانة الأخرى فارغة NULL.. هذا بالإضافة لعرض باقي الكتب وما يناظرها من مؤلفين.

ملحوظة2:
لإنشاء الجملة السابقة باستخدام باني الاستعلام، اتبع نفس الخطوات التي اتبعناها في إنشاء الربط الأيسر، ولكن اختر كلا الاختيارين All Rows From Books، وAll Rows From Authors.

- **الربط الداخليّ Inner Joins**

هذه العمليّة مماثلة لفقرة WHERE ، حيث يتمّ عرض السجلات المتوافقة فقط من الجدولين.

```
SELECT Book, Author
FROM Authors INNER JOIN Books
ON AuthorID = Authors.ID
```

ملحوظة:
لإنشاء الجملة السابقة باستخدام باني الاستعلام، اتبع نفس الخطوات التي اتبعناها في إنشاء الربط الأيسر، ولكن أزل كلا الاختيارين All Rows From Books، وAll Rows From Authors.

4. استعلامات الأداء Action Queries

يعتبر هذا النوع من الاستعلامات أبسط من استعلامات التحديد، حيث لا يقوم باسترجاع أيّ سجلات.. ولكنّه في المقابل يقوم بالتغيير في محتويات الجدول، سواء بتحديث قيم السجلات أو إضافة سجلات جديدة أو حذف بعض السجلات الموجودة.
هذا وتعيد هذه الاستعلامات عدد السجلات التي تأثرت (وليس السجلات نفسها).

ملحوظة:
لن يمكنك تجريب هذه الاستعلامات في باني الاستعلام، فهو لا يتعامل إلا مع استعلامات التحديد.

ويمكنك أن تؤدّى بهذه الاستعلامات العمليّات التالية:

- **حذف الصفوف Deleting Rows**

يمكنك أن تحذف أيّ عدد تريده من الصفوف تبعا لل شرط الذي تحدّده في مقطع WHERE.. والجملة التالية تريك كيف يمكن حذف كلّ الكتب التي تحمل رقم 3:

```
DELETE Books
WHERE ID = 3
```

والجملة التالية تحذف كلّ الكتب التي كتبها (عباس العقاد):

```
DELETE Books
WHERE AuthorID = (
SELECT ID
FROM Authors
WHERE Author = 'عباس العقاد'
)
```

طبعا لاحظت استخدام نا لاستعلام التحديد في جملة الشرط.. هذه الإمكانيّة تمنحك قدرات بلا حدود، لحذف السجلات التي تنطبق عليها المواصفات التي تحدّدها.

- **إدراج سجلات جديدة Inserting New Rows**

لجملة الإدراج الصيغة التالية:

INSERT INTO اسم الجدول (أسماء الحقول) VALUES (قيم الحقول)

حيث ستضاف القيم للحقول تبعا لترتيب كتابة أسمائها.. ويمكن ألا تكتب كلّ أسماء الحقول (لو أردت أن تترك بعض الحقول فارغة) ، ولكن يجب أن يكون عدد القيم مساويا لعدد أسماء الحقول.

فإذا أردت أن تدخل قيما لكلّ الحقول، فيمكن ألا تكتب أسماءها، على أن يكتب القيم مرتّبة تبعا لترتيب الحقول الأصليّ في الجدول، تبعا للصيغة التالية:

INSERT INTO اسم الجدول VALUES (قيم الحقول)

مثال: لإضافة حقل جديد لجدول الكتب، استخدم الجملة التالية:

```
INSERT INTO Books (Book, AuthorID)
VALUES ('نجم الخيال الضاحك', 6)
```

لاحظ أنّنا لم نضع قيمة للحقل ID لأنّه يُولّد تلقائيّا، حيث إنّنا جعلناه ترقيما تلقائيّا عند إنشائه.

ويمكن أن تنسخ مجموعة من السجلات من جدول لآخر (على أن يكون لهما لحقولهما نفس نوع البيانات)، باستبدال فقرة VALUES بجملة SELECT.

افترض أنّ عندنا جدولا مما ثلا لجدول الكتب، سنضع فيه بعض الكتب مؤقتا لأيّ سبب، هذا الجدول اسمه TempBooks.. في هذه الحالة يمكن أن ننسخ فيه كتب (توفيق الحكيم) بالجملة التالية:

```
INSERT INTO TempBooks
SELECT Books.*
FROM Authors, Books
WHERE Author = 'توفيق الحكيم' AND
        AuthorID = Authors.ID
```

- **تحرير السجلات الموجودة Editing Existing Rows**

لتغيير قيم بعض أو كلّ سجلات أحد الجداول، استخدم جملة UPDATE ، التي لها الصيغة التالية:

UPDATE اسم الجدول SET الحقل1 = القيمة1, الحقل2 = القيمة2, ...
WHERE شرط

مثال: استخدم الجملة التالية لتغيير اسم المؤلف (منصور) إلى (عايش):

UPDATE Authors SET Author = 'عايش'
WHERE Author = 'منصور'

18.10 عيوب قواعد البيانات

إن القائمة الطويلة التي عرضناها عبر هذا الفصل عن مزايا قواعد البيانات لا ينفي بحال من الأحوال أن إنشاء قواعد البيانات له عدة عيوب ، نلخصها في النقاط التالية:

1. الحيز:

حتى تقدم قواعد البيانات كل خدماتها المميزة إلى مختلف مستخدميها فإن هذا يتطلب حيز هائل من وسائط التخزين الثانوية وذاكرة أساسية ذات حيز ضخم فيما يضيف تكلفة مادية إضافية إلى جانب جهد صيانة و تعديل وتحديث الملفات كما تتطلب معدات إضافية كثيرة.

2. مشاكل الكيان البرمجي للقاعدة:

يتصف برنامج مدير قاعدة البيانات بأنه برنامج معقد يتطلب جهداً كبيراً في استيعابه وفهمه من مسئولي نظام المعلومات حتى يستفاد بما عرضناه من مزايا.

3. التكلفة :

يعتبر برنامج مدير قاعدة البيانات مكلف في حد ذاته فإذا أضفنا تكلفته إلى باقي عناصر التكلفة (وسائط التخزين- الذاكرة- تدريب مسئولي النظام ...) لاتضح ارتفاع ثمن مكونات قاعدة البيانات.

4. توقف قاعدة البيانات:

عن العمل نتيجة مشاكل الكيان إلى أو البرمجي لها فإنها تؤثر ع لى قطاع عريض من المستخدمين لا يسهل درأ أضراره المادية والمعنوية.

5. برامج التأمين و الدعم و استعادة التشغيل:

برامج مكلفة مادياً وصعبة التصميم وتحتاج أفراداً على مستوى تعليمي وتدريبي مرتفع.

18.11 الاتصال بقواعد البيانات باستخدام JDBC

تأتي عملية الاتصال بقاعدة البيانات في مقدمة أولويات تطبيق JDBC.
و سنبدأ بشرح مفهوم مشغلات JDBC ، و الاختلاف الهيكلي بين النماذج الأربعة لمشغلات JDBC، ومتى يفضل استخدام كل منهما، بعد ذلك سدنتقل لموضوع تسجيل وإدارة المشغلات باستخدام كائنات DriverManager ، أخيراً سأبين كيفية تأسيس الاتصال مع القاعدة باستخدام كائنات DriverManager من خلال نصوص برمجية تعليمية وقياسية بآن واحد.

18.11.1 مفهوم مشغلات JDBC

كما أن لكل بلد لغته الخاصة والتي يتخاطب بها سكانه، يمتلك نظام قاعدة البيانات لغة خاصة أو بروتوكول خاص للتخاطب بين المخدم والزبائن.
تعمل معظم أنظمة قواعد البيانات الحديثة وفق منطق زبون- مخدم، باستخدام بنى أحادية أو ثنائية أو ثلاثية الطبقة.
عندما يعمل التطبيق الزبون على نفس مضيف المخدم، نقول عن النظام بأنه نظام أحادي الطبقة. أما في الأنظمة ثنائية الطبقة فيتوضع الزبون في مكان مختلف عن المخدم، ويتخاطبان مع بعضهما عبر الشبكة باستخدام بروتوكولهما الخاص.

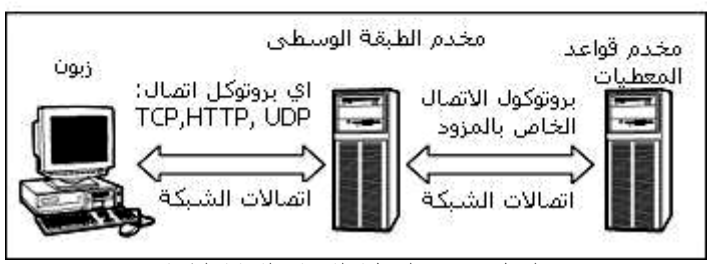
تتبع معظم الأنظمة الحالية هذه الهيكلية، كما أن إصدارات Oracle وSQLServer القياسية تستخدم الهيكلية ثنائية الطبقة.

أما الأنظمة ثلاثية الطبقة فتستخدم مخدماً آخر، يدعى في الغالب بمخدم العمل (application Server) ويتوضع بين الزبون ونظام إدارة قواعد البيانات DBMS. لذلك يقال عنه بأنه الطبقة الوسطى. يستخدم المطورون مخدم العمل ليضعوا عليه تطبيقات الأعمال للتقليل من حجم الشيفرة البرمجية لتطبيقات الزبون، ولجعل صيانة شيفرة الأعمال أ سهل في المستقبل ومستقلة عن التطبيقات الزبونة.

فعندما يحتاج المطور إجراء تعديل ما، فإنه سيجريه على مخدم العمل بدون أن يؤثر على إعدادات التطبيقات الزبونة. تلعب تطبيقات J2EE ومخدمات الويب دور الطبقة الوسطى في أغلب مجتمعات الويب.

الهيكلية ثنائية الطبقة لمنطق زبون - مخدم

الهيكلية ثلاثية الطبقة لمنطق مخدم زبون



الشكل 6-18: الهيكلية الثنائية والثلاثية الطبقة

بغض النظر عن هيكلية الطبقات، فإن كل أنظمة زبون- مخدم تؤدي وظائفها الأساسية بأسلوب متماثل.

حيث يقوم الزبون بإرسال عبارات SQL أو أوامر قاعدة البيانات إلى نظام DBMS للتنفيذ وذلك باستخدام لغة معينة أو بروتوكول خاص يقبله كل من الزبون ونظام إدارة قاعدة البيانات. وعندما ينتهي المخدم من معالجة الطلب، فإنه سيرسل الرد إلى الزبون باستخدام نفس البروتوكول أو اللغة المستخدمة سابقاً. قد يحتوي الرد على معطيات تقابل استعلام SQL أرسله الزبون، أو يحتوي على معلومات بنجاح أو إخفاق الطلب.

بشكل عام: عندما تريد كتابة تطبيق قواعد بيانات زبون فيجب عليك أن تتصل مع المخدم. يمكنك استخدام إحدى هاتين الطريقتين لتحقيق هذه الغاية:

1- إنشاء تصريحات خاصة من أجل بروتوكول الاتصال مع DBMS. يحتاج القيام بذلك إلى دراية واسعة بتفاصيل بروتوكول الاتصال المستخدم من قبل قاعدة البيانات، كما تحتاج لتضمين أي استدعاء ستقوم به إلى قاعدة البيانات في تصريحاتك. وتحتاج لآلية لمعالجة الرد المحتمل واستخلاص المعطيات منه.

يتطلب هذا الخيار جهداً كبيراً، لأن التصريح عن كل وظائف البروتوكول ليس بالأمر السهل، ويحتاج لجهد برمجي هائل، لذلك يبقى هذا الخيار كحل تستخدمه المؤسسات وليس الأفراد ولكن كان لابد من ذكره.

2- استخدام تصريحات مزود قاعدة البيانات لبروتوكول الاتصال. تسمى هذه التصريحات بالمشغلات أو واجهات برمجة تطبيقات (APIs) ، وهي الأكثر استخداماً.

تعرّف واجهة برمجة التطبيقات الخاصة بقاعدة البيانات المناهج والخصائص التي تمكنك من إرسال واستعادة والحصول على معلومات الحالة لقاعدة البيانات، بالإضافة إلى إمكانية استخراج المعطيات منها. يمكنك الحصول على مشغلات قواعد البيانات من مصادر متعددة، أهمها إصدارات قاعدة البيانات نفسها. مع ملاحظة أن بعض أنظمة قواعد البيانات تخضع لحقوق توزيع في حين أنَّ بعضها الآخر مفتوح المصدر.

قدمت شركة Sun Microsystems واجهة برمجة تطبيقات معيارية تختص بكل تقنيات الاتصال مع قواعد البيانات، وأطلقت عليها اسم JDBC. حيث يمثل الاسم علامة تجارية مسجلة لشركة Sun ، تعرّف تقنيات اللغة Java للتخاطب مع قواعد البيانات بالكامل. وبذلك استطاعت تخليصك من حقوق الملكية التي فرضتها بعض الشركات على مزوداتها، كما خلصتك من احتمال إنشاء تصريحاتك الخاصة للاتصال بقاعدة البيانات.

ما هي مشغلات JDBC؟

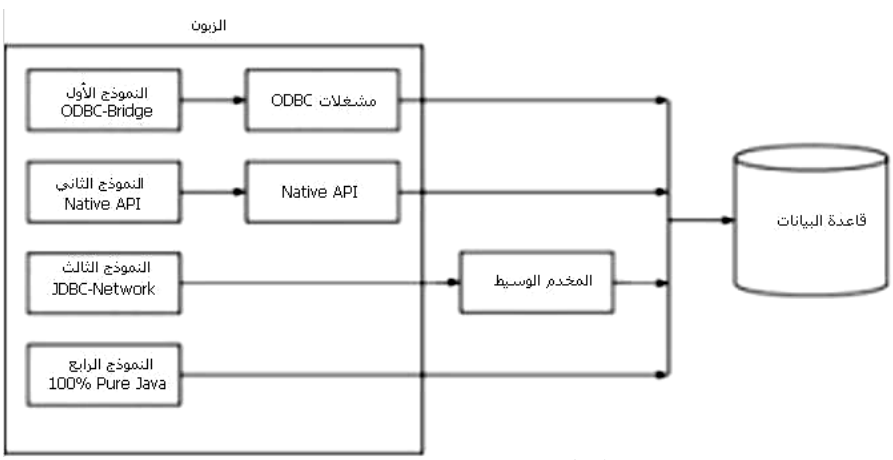
تنفّذ مشغلاتJDBC الواجهات المعرَّفة في JDBC API من أجل التخاطب مع مخدم قاعدة البيانات فباستخدام مشغلات JDBC يمكنك أن تؤسس اتصالاً مع قاعدة البيانات، وتتخاطب معها وذلك بإرسال عبارات SQL أو أوامر قاعدة بيانات، ومن ثم تستقبل النتائج وكل ذلك ضمن إطار عمل اللغة Java.

وكما ذكرت سابقاً يمكنك الحصول على مشغلات JDBC من مزود قاعدة بياناتك، أو من إصدار قاعدة البيانات نفسه، ولكن قبل أن تستطيع الاستفادة من المشغل عليك القيام بتسجيله، وإعداد الخاصية JDBC URL بشكل صحيح، لتتمكن من تأسيس اتصال إلى قاعدة البيانات (سنناقش هذه النقطة لاحقاً).

18.11.2 نماذج مشغلات JDBC

هنالك نماذج متعددة لتصريحات مشغلات JDBC ، ويعود هذا التنوع بسبب منصات العمل وأنظمة التشغيل التي تعمل عليها اللغة Java.

لذلك قامت شركة Sun بتقسيم تصريحات المشغلات إلى أربع نماذج رئيسية: النموذج الأول والثاني والثالث والرابع، ويتفرع عن كل نموذج الكثير من النماذج الفرعية الأخرى. يوضح الشكل 7-18 النماذج الرئيسية للمشغلات.



الشكل 7-18: نماذج مشغلات JDBC

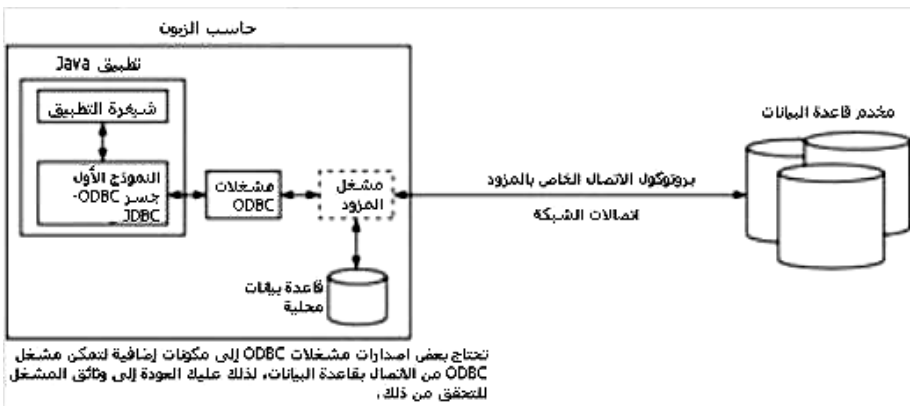
يعتمد النموذج الأول والثاني على برمجيات ثانوية لتستطيع التواصل مع قاعدة البيانات. وغالباً ما تكون هذه البرمجيات ملفات DLL مصممة باستخدام اللغة C++/C ومتواجدة على الحاسب الزبون، حيث تستخدمها كل من Java و JDBC للتخاطب مع قاعدة البيانات.

أما بالنسبة للنموذجين الثالث والرابع فهما عبارة عن تصريحات Java خالصة. ولا تحتاج لأية برمجيات وسيطة، فيما عدا مشغل JDBC المناسب.

النموذج الأول: الجسر JDBC-ODBC

يعمل هذا النموذج مع مشغلات ODBC المقدمة من قبل مزود قاعدة بياناتك، وعلى الرغم من أنك قد تجد مشغلات ODBC خاصة بنظام التشغيل Unix، إلا أنها تستخدم غالباً مع نظام التشغيل Microsoft Windows. ينبغي عليك الحصول على مشغل ODBC المنا سب لقاعدة بياناتك أولاً لتستطيع استخدام الجسر JDBC-ODBC، وربما تحتاج لبرمجيات ثانوية أخرى.

يوضح الشكل 8-18 كيفية تخاطب الزبون مع قاعدة البيانات باستخدام الجسر JDBC-ODBC.



الشكل 8-18: الأول للمشغلات(الجسر JDBC-ODBC)

يتطلب استخدامODBC إعداد اسم مصدر البيانات DSN، أو (Data Source Name) والذي يتضمن معلومات عن قاعدة البيانات، كاسمها وموضعها....

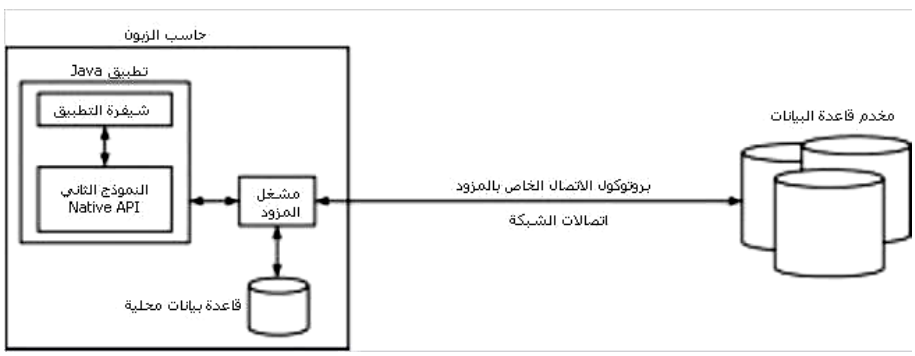
يمكنك استخدام هذا النموذج مع تطبيقات قواعد البيانات المحلية مثل التطبيقات التي تستخدم قاعدة بياناتMicrosoft Access، وبذلك ستوفر الكثير من التكلفة المتعلقة بتطوير التطبيق ونشره.

كما يمكنك استخدام هذا النموذج من أجل تطبيقات قواعد البيانات العاملة على نظام التشغيل Windows والثنائية الطبقة أو ثلاثية الطبقة، والمحدودة بعدد معين من الزبائن الذين يمكنهم التواصل مع قاعدة البيانات. أما استخدام هذا النموذج مع تطبيقات Java التي تحتاج لتحميل متغيرات (ODBC DNS أو أية مكونات أخرى) في وقت التنفيذ فيعتبر أمراً غير مألوفاً، وقد يحتاج لـ كثير من الجهد والذي يغنيك عنه استخدامك لنماذج أخرى.

النموذج الثاني: JDBC-native API

يحتاج هذا النموذج إلى واجهة برمجة تطبيقات خاصة بنظام التشغيل الذي يعمل عليه الزبون، لتتولى عملية الاتصال مع قاعدة البيانات. وغالباً ما تكون هذه الواجهة مكتوبة باللغة C++/C ، لذلك ستكون مناسبة للبيئة التي يستخدمها الزبون.

وبغض النظر عن تصريحات الواجهة المستخدمة فإنها ستتولى عملية التخاطب مع قاعدة البيانات. يوضح الشكل 9-18 كيفية إعداد المشغلات التي تستخدم هذا النموذج.



الشكل 9-18: النموذج الثاني (Native API)

وكما تلاحظ من الشكل فإن النموذج الثاني يشبه النموذج الأول، حيث تقوم الواجهة NativeAPI بتحويل استدعاءات JDBC إلى الصيغة التي تتعامل معها قاعدة البيانات الهدف.

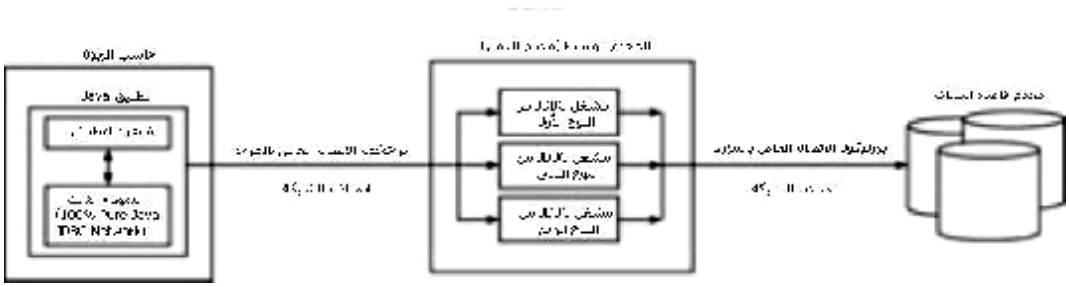
كما ستلاحظ أثناء استخدام هذا النموذج سرعة في الأداء عن النموذج الأول، يعود ذلك إلى إقصاء دور مشغل ODBC هنا.

ولكنك ستبقى مقيداً - كما في النموذج الأول- بأعداد المزود Native API على كل حسابات الزبائن.

النموذج الثالث: Pure Java%100, JDBC-network

يستخدم هذا النموذج الطبقة الثالثة (مخدم العمل) للوصول إلى قاعدة البيانات، أي أنه يستخدم نفس هيكلية الانتشار المتبعة في إطار العمل J2EE.

حيث يستخدم الزبائن مقابس الشبكة المعيارية للاتصال مع مخدم العمل. بعد ذلك يقوم مخدم العمل بترجمة معلومات المقابس إلى صيغة الاستدعاء المطلوبة من قبل النظام DBMS ، وبعدها تنتقل هذه الاستدعاءات إلى مخدم قاعدة البيانات. يوضح الشكل 10-18 الإعداد النموذجي لنظام ثلاثي الطبقة، والذي يستخدم النموذج الثالث للاتصال بقاعدة البيانات.



الشكل 10-18: النموذج الثالث (100% Pure Java, JDBC-Network)

يمكنك تصور مخدم العمل كوكيل (JDBC Proxy) ، أي انه يقوم بتنفيذ مجموعة من الاستدعاءات للتطبيق الزبون. على العموم ستحتاج إلى بعض المعرفة حول كيفية إعداد مخدم العمل لكي تستخدم هذا النموذج من المشغلات بشكل فعّال.

فمثلاً قد يستخدم مخدم العمل أحد نماذج المشغلات الأول أو الثاني أو الرابع، للاتصال بقاعدة البيانات. لذلك فإن إدراك الفرق بين هذه النماذج سيساعدك كثيراً.

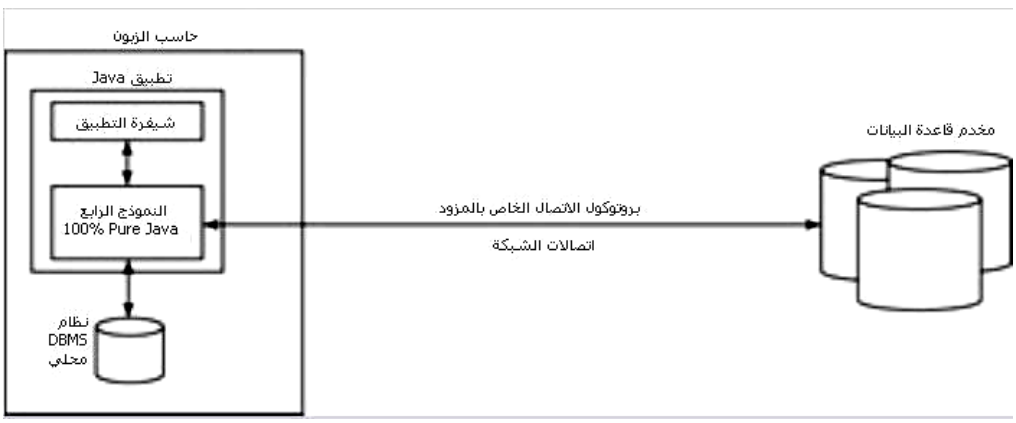
إن ما يزيد من مرونة هذا النموذج هو الدعم الذي يقدمه أغلب مزودي مخدمات العمل لميزة التخاطب المتعدد مع قواعد البيانات (multiple dataebase bak-ends). حيث تمكنك هذه الميزة من كتابة شيفرة برمجيـ ة واحدة للتخاطب مع عدد كبير من قواعد البيانات المختلفة، لأن مخدم العمل سيقوم بمعالجة

الفروقات الموجودة بين قواعد البيانات من حيث الصياغة القواعدية لعبارات SQL والاختلاف بين أنواع المعطيات.

بشكل عام: يجعل هذا النموذج عملية النشر أسهل بكثير من النموذجين السابقين. لأن الزبون في هذه الحالة لن يحتاج إلى أية برمجيات ثانوية، بل سيستخدم التطبيق الزبون فقط للتخاطب مع قاعدة البيانات عبر مخدم العمل.

النموذج الرابع: Java 100 %

تصمم هذه المشغلات كلياً باستخدام Java. حيث تغلف مهام بروتوكولات الاتصال مع قواعد البيانات وذلك لتحقيق الاتصال المباشر مع النظام DBMS، وكما في النموذج الثالث لا يحتاج الزبائن إلى أية برمجيات ثانوية.
يقدم غالبية مزودي قواعد البيانات مشغلات هذا النموذج. يوضح الشكل 18-11 كيف تعمل مشغلات النموذج الرابع في حاسب الزبون.



الشكل 18-11 : النموذج الرابع 100% Pure Java

وهكذا فإن كل نموذج من النماذج الأربعة السابقة له محاسنه وله مساوئه، لذلك فأنت بحاجة لتفهم هذه النماذج لتستطيع اتخاذ القرار المناسب لنظامك.

يقدم الجدول 18-4 بعض محاسن ومساوئ كل نموذج من النماذج الأربعة السابقة.

الجدول 18-4 يعرض محاسن ومساوئ كل نموذج من النماذج الأربعة السابقة.

نموذج المشغل	محاسنه	مساوئه

الجسر JDBC-ODBC	- يمكنك من الوصول إلى قواعد البيانات المحلية مثل Microsoft Access وFoxPro. - لا يتطلب قواعد بيانات تدعم المستوى المؤسساتي. - جيد لاختبار ميزات JDBC الأساسية على حاسبات تشغّل النظام Windows بشكل مستقل (دون شبكة اتصال).	- يتطلب إعداد مصادر بيانات ODBC والحفاظ عليها بشكل مستمر، مما يقلل من مرونة التطبيق. - بطيء لأنه طبقة ODBC الإضافية تحتاج لمرحلة معالجة إضافية. - قد يحتاج إلى برمجيات زبونة ثانوية مثل عناصر الاتصال الخاصة بقواعد البيانات عبر الشبكة.
النموذج Native API	- أسرع من النموذج الأول، ذلك لاستبعاد الطبقة ODBC. - تصريحات مشغلاته مناسبة تماماً لكل من إطار العمل ونظام DBMS المستخدمين.	- يتطلب إعداد واجهة المزود الخاصة (Specific-vender API) على حاسب الزبون. - غير مناسب للاستخدام من أجل نشر تطبيق زبون سيعمل على منصات عمل مختلفة، لأنه يحتاج إلى إعدادات خاصة لحاسب الزبون ويشترك في هذه النقطة مع النموذج الأول.
النموذج JDBC-Net Work	- لا يحتاج إلى أية برمجيات ثانوية. - يمكن أن يجعل مخدم العمل قادراً على الوصول إلى أنواع مختلفة من قواعد البيانات.	- يتطلب استخدام برمجيات وسيطة أو مخدم عمل. - قد يتطلب إعدادات إضافية للاستخدام عبر شبكة الإنترنت.
النموذج 100% Pure Java	- لا يحتاج إلى برمجيات ثانوية، يتصل مباشرة مع مخدم قاعدة البيانات.	- قد يتطلب إعدادات إضافية للاستخدام عبر شبكة الإنترنت.

18.11.3 اختيار نموذج المشغل المناسب

قبل أن تختار نموذج المشغل المناسب، عليك أن تعرف التوجه المراد إتباعه لنشر تطبيق، وبذلك ستستقر على النموذج المناسب مباشرة. كما يتوجب عليك الإجابة على السؤال التالي:

هل عليك أن تتحكم بالحاسب الذي يحتاج للتواصل مع قاعدة البيانات؟ وهل هذا الحاسب هو حاسب زبون أم مخدم عمل؟

إذا كان جوابك لا، فأنت مقيد بأحد النموذجين الثالث والرابع، وبالطبع فهذا التقييد ليس سيئاً أبداً، لأن هذه المشغلات ستقودك لاستخدام تصريحاتهما مباشرة ودون الحاجة لإعداد حاسب الزبون. فهما لا يتطلبان أية برمجيات ثانوية مثل مشغلات ODBC أو برمجيات DBMS.

إذا كان تطبيقك يتطلب الوصول إلى قاعدة البيانات عبر الإنترنت فسيصبح استخدام أي نموذج من نماذج المشغلات الأربعة معقداً جداً.

لأن مخدمات قواعد البيانات لا تتواصل مع زبائنها باستخدام بروتوكولات الإنترنت المعيارية مثل البروتوكول HTTP وFTP. بل تستخدم بروتوكولات اتصال خاصة بمزوديها (تدعم أمن المعلومات وأنظمة الحماية الأخرى مثل الجدران النارية).

أما إذا كنت مصرّاً على التخاطب مع قاعدة بياناتك عبر الإنترنت، فالأفضل أن تُضَمِّن برمجة (JDBC في صفحات JSP(Java Server Page وتستخدم تقنية Servlets للوصول إلى القاعدة.

18.11.4 مزودي مشغلات JDBC

يقدم معظم مزودي أنظمة إدارة قواعد البيانات مشغلات خاصة لتقنية JDBC. فعلى سبيل المثال يقدم Oracle مشغلاً خاصاً بكل إصدار من إصداراته، في حين

لا تقدم شركة Microsoft مشغلاً خاصاً بقاعدة البيانات SQL Server، لذلك ستضطر لشراء هذا المشغل.
كما يتوافر الكثير من مشغلات JDBC مفتوحة المصدر من أجل قواعد البيانات مفتوحة المصدر مثلmSQL وPostgreSql.
على كل الأحوال يمكنك العودة إلى موقع Sun على الشبكة العالمية للحصول http://industry.java.sun.com/product/jdbc/drivers على معلومات عن مزودي مشغلات JDBC. كما يحتوي الموقع على الكثير من التفاصيل المتعلقة بنماذج وميزات مشغلات JDBC. ويقدم ارتباطات إلى مواقع مزودي المشغلات لطلب معلومات إضافية كالسعر والتوافقية.

18.11.5 استخدام مشغل JDBC

سنوضح في هذه الفقرة كيف يمكنك استخدام مشغلات JDBC في تطبيقاتك.
يفضل قبل البدء بالعمل أن تتحقق من اسم الصنف المشغل الذي حصلت عليه من المزود. يتبع المزودون قواعد تسمية الحزم في Java في أغلب الأحيان عند تسمية مشغلاتهم.
فمثلاً مشغل Oracle: oracle.jdbc.driver.OracleDriver.

18.11.6 الصنفين من نوع Driver ، DriverManager:

تمثل كل من الواجهة java.sql.Driver والصنف java.sql.DriverManager أدوات التعامل مع مشغلات JDBC ومن المعرف أن أي مشغل JDBC يجب أن ينفّذ الواجهة Driver ، ولكن واقع العمل يقول بأن التعامل مع هذه الواجهة غير مجدي، فعلى الرغم من أنها تزودك بمناهج للحصول على الاتصال، فإن استخدام الصنف DriverManager يعطيك الكثير من أساليب الاتصال العملية.

بالإضافة إلى أنَّ الكائن DriverManager يمنحك القدرة على إدارة المشغلات. فباستخدامه ستستطيع تسجيل المشغل وتحديده أو إزالة مشغل ما من تطبيقك، لذلك ستصبح قادراً على تخزين عدد هائل من المشغلات في تطبيقك برمجياً، ومن ثم تختار المشغل المطلوب أثناء زمن التنفيذ. الأمر الذي يعطي تطبيقك مرونة عالية في التعامل مع قواعد البيانات المختلفة.

18.11.7 تسجيل مشغلات JDBC

لكي تستطيع استخدام مشغل JDBC ينبغي عليك تسجيله مع كائن من الصنف DriverManager، والذي يمتلك منهجاً خاصاً لتسجيل المشغلات.

على العموم هنالك عدّة طرق لتسجيل مشغلات JDBC:

- استخدام المنهج
Class.forName(String drivername).newInstance ( ).

- استخدام المنهج
DriverManager.registerDriver(Driver diverName).

أكثر هذه الطرق شيوعاً هي طريقة استخدام المنهج
Class.forName(String drivername).newInstance.( )
فبالإضافة إلى إنشائه لمرجع إلى كائن JDBC Driver جديد، فإنه يسمح للكائن أن يسجل نفسه ذاتياً مع DriverManager.

ستتسائل هنا: كيف تمت عملية التسجيل بدون إنشاء كائن من الصنف DriverManager؟

تتطلب مواصفات المشغل JDBC كائنات Driver لكي تسجيل نفسها مع الصنف DriverManager ، تتم هذه العملية بمساعدة بادئ ستاتيكي يقوم باستدعاء المنهج DriverManager.registerDriver().

إن إتباع هذه الطريقة يمنحك المقدرة على تسجيل المشغل ديناميكياً في زمن التنفيذ وبغض النظر عن كيفية حصول عملية التسجيل.

- **إنشاء اتصال بين مشغل قاعدة البيانات ODBC و برنامج Java**

سوف نقوم بعملية ربط Java بقاعدة البيانات Access سوف نستخدم Microsoft Office Access و Windows XP.

أولاً

نقوم بفتح لوحة التحكم/ أدوات أدارية / مصادر البيانات (ODBC Data Source Administrator) كما في الأشكال 12-18,13-18.

شكل 12-18

شكل 13-18

ثانياً

نقوم بإضافة User Data Sources عن طريق الزر إضافة(Add) ، سوف تجد لائحة أختر (Microsoft Office Access). ثم أضغط على موافق Finish كما في الشكل 14-18.

شكل 14-18

ثالثاً

سوف تظهر نافذة (ODBC Microsoft Access) و منها نقوم بكتابة Data Source Name و اسم المستخدم User ID كما في الشكل 15-18.

شكل 15-18

قم بتحديد قاعدة البيانات المطلوبة من الزر (تحديد).
و أخيراً قم بالضغط على موافق ثم موافق للموافقة على الدرايفر.

18.11.8 تحديد وإلغاء مسجلات JDBC

ستحتاج في مرحلة من مراحل تطوير تطبيق JDBC إلى تحديد مشغل ما من قائمة DriverManager أو إزالته من القائمة، ستدرك مدى أهمية هاتين العمليتين عند تطوير تطبيقات باستخدام نموذج التصميم (Factory Design patterns)Factory والتي تستخدم العديد من أنواع قواعد البيانات.

نموذج التصميمFactory: هو عبارة عن تصميم (مجموعة من الأصناف) يتيح لك إمكانية دعم أنواع مختلفة من قواعد البيانات، وذلك بتقديم نموذج تصميم خاص بكل نوع من أنواع قواعد البيانات المستخدمة.
حيث يمكنك ضمن الصنف Factory استخدام الكائن DriverManager لتسجيل مشغلات JDBC الخاصة بقواعد الـ بيانات المستخدمة وذلك لإمكان إجراء الاتصال معها. فعندما يحتاج التطبيق لإنشاء كائنات الاتصال (الكائنات Connection) سيقوم الصنف Factory بتحديد المشغلات المطلوبة للاتصال - والمعدَّة مسبقاً- في

الكائن DriverManager وبعدها يقوم بإنشاء كائنات الاتصال المطلوبة.وبهذا الأسلوب سيكون باستطاعتك جعل التطبيق يدعم العديد من أنواع قواعد البيانات في الوقت نفسه، ويصبح قادراً على إنشاء كائنات الاتصال حسب طلب التطبيق الزبون.

يُستخدم المنهجان

DriverManager.getDriver()
DriverManager.getDrivers( )

من أجل تحديد المشغلات حيث يعيد المنهج الأول كائناً من النمط Driver وذلك عندما تمرر له البارمتر JDBC URL المعرَّف لقاعدة بياناتك (سنقوم بتغطية موضوع JDBC URL في الفقرة التالية).

أما المنهج الثاني فيعيد كائناً من النمط Enumeration والذي تستطيع من خلاله التجول في قائمة الكائنات Driver حتى تجد المشغل المطلوب. تبين الشيفرة التالية كيفية استخدام المنهج DriverManager.getDrivers().

```
//Assume valid JDBC drivers have been registered
//Enumeration object needed to obtain registered drivers
Enumeration driverEnum = DriverManager.getDrivers();
//List the drivers.
while(driverEnum.hasMoreElements()) {
//Cast to a Driver object
Driver driver = (Driver)driverEnum.nextElement();
String str = "Driver name is: " + driver.getClass().getName();
System.out.println(str);
}
```

عند تنفيذ الشيفرة السابقة سيتم طباعة كل أسماء المشغلات المستخدمة والمسجلة في الكائن DriverManager. لكن ليس هذا هو هدف الشيفرة فقط، فهذه الشيفرة تمكنك من استخدام الكائن Driver والمعاد من الكائن Enumeration من أجل تأسيس اتصال مع قاعدة البيانات أو لتستخدمه مع المنهج DriverManager.deregisterDrivers() لإلغاء تسجيل مشغل JDBC ما من القائمة الداخلية للكائن DriverManager.

قد تضطر لسبب ما إلى إلغاء دعم قاعدة بيانات معينة عندها يمكنك استدعاء المنهج DriverManager.deregisterDrivers() من أجل إلغاء تسجيل مشغل JDBC الخاص بقاعدة البيانات من قائمة مشغلات الكائن DriverManager.

يشبه هذا المنهج المنهج registerDriver() فهما يأخذان بارمتراً من النمط Driver.

18.11.9 التعامل مع الكائنات Connection

في عالم JDBC يمثل مرجع الكائن Connection اتصالاً فيزيائياً مع قاعدة البيانات، حيث يقوم المنهج Driver.connect() بتزويدك بكائن الاتصال. يمكنك استخدام الكائن Driver مباشرة لتأسيس الاتصال، ولكن كائنات DriverManager تغلف ذلك الاستدعاء بمنهجها getConnection() .

يفضّل ولكثير من الأسباب استخدام كائنات DriverManager لتأسيس اتصال مع قاعدة البيانات. وأحد هذه الأسباب هو أن الكائن DriverManager سيتحقق من مشغل JDBC والمسجل عنده المطلوب لتأسيس الاتصال، كما يزدك المنهج getConnection() التابع للصنف DriverManager بأساليب متنوعة لتأسيس اتصال مع قاعدة البيانات.

يقبل المنهج DriverManager.getConnection() الكثير من البارمترات ولكن أهم هذه البارمترات هو البارمتر JDBC URL والذي سنتعرف عليه في الفقرة التالية.

18.11.10 مفهوم JDBC URL

يحتاج JDBC لنظام تسمية ليتمكن من الاتصال مع قاعدة البيانات، ويمثل JDBC URL هذا النظام وفق البنية العامة التالية:

jdbc:<subprotocol><subname>

لذلك يتوجب عند إنشاء الصيغة JDBC URL أن تحدد القيمتين <subprotocol> و<subname> تعرّف القيمة <subprotocol> البروتوكول الخاص الذي يستخدمه المزود عند الاتصال بقاعدة البيانات مع العلم بأن بعض أنظمة DBMS يستخدم أكثر من بروتوكول للاتصال مع مخدم قاعدة البيانات.

أما القيمة <subname> فتعرّف مصدر المعطيات أو قاعدة البيانات المراد الاتصال بها. أيضاً تحتوي بعض المخدمات على أكثر من قاعدة بيانات لذلك تستخدم أسماء منطقية لتمثل كل واحدة منهم. إذاً تمثل القيمة <subname> الاسم المنطقي لقاعدة البيانات على المخدم.

تعتمد القيمتين <subprotocol> و <subname> على مشغل JDBC المستخدم حيث تختلف المشغلات المقدمة من نفس المزود بالقيمة <subprotocol>. فلا يوجد صيغة معيارية لكل قيمة. لذلك عليك العودة إلى وثائق مشغل JDBC للحصول على الصيغة المعيارية.

لنأخذ المثالين التاليين:

- مثال عن JDBC-ODBC

```
String url = "jdbc:odbc:MyD";
```

تقرأ هذه السلسلة على الشكل التالي:
استخدم المشغل ODBC للاتصال بقاعدة البيانات يدل عليها اسم مصدر المعطيات MyDB. القيمة <subname> تذكر بأنه للاتصال باستخدام مشغل ODBC يجب إعداد اسم مصدر المعطيات (ODBC DSN).

- مثال Oralcle:

```
String url = "jdbc:oracle:thin:@dbServername:1521:ORCL";
```

يمثل القسم oracle:thin القيمة <Subprotocol>. يمثل الجزء oracle معياراً لمشغل Oracle أما الجزء thin فـ يشير إلى آلية الاتصال الخاصة بمشغلOracle والمطلوب استخدامها. وكما ذكرت سابقاً فإن بعض المزودين يقومون بتغليف العديد من بروتوكولات الاتصال مع قواعد بياناتهم عبر الشبكة في مشغلهم، وهذا ما يفعله مزود Oracle مع مشغله التابع للنموذج الرابع من المشغلات.
يدل القسم dbServername:1521:ORCL@ على القيمة <Subname> والتي تعبر عن اسم مضيف مخدم قاعدة البيانات ورقم منفذه بالإضافة إلى اسم قاعدة البيانات المطلوب الاتصال بها.

يجدر الذكر بأن القيمة <Subprotocol> هي قيمة فريدة وخاصة بكل مشغل لأن المزودين يقومون بتسجيل القيمة <Subprotocol> الخاصة بكل منهم في شركة Sun والتي تعتبر السجل الرسمي لهذه القيم.

18.11.11 تأسيس الاتصال مع قاعدة البيانات

سنعود الآن للتعليق على المناهج DriverManager.getConnection() ذات التحميل الزائد (overloaded) ، اخترت من هذه المجموعة ثلاثة مناهج رئيسية لأنها تستخدم بكثرة وهي:

1- المنهج getConnection(String url).
2- المنهج getConnection(String url, Properties prob).
3- المنهج
getConnection(String url, String User, String Password).

يتطلب المنهج الأول بارمتراً واحداً فقط وهو JDBC URL لتأسيس الاتصال مع قاعدة البيانات.
من الملاحظ بأن هذا المنهج يفتقر إلى أمن المعلومات، على كل الأحوال يمكنك استخدام هذا المنهج إذا كانت قاعدة بياناتك لا تدعم خدمات التحقق من صلاحيات المستخدم بشكل مباشر. كما تفترض بعض أنظمة قواعد البيانات بأن المستخدم يمتلك صلاحيات الاتصال معها طالما أنَّه استطاع تسجيل الدخول إلى التطبيق الزبون.

يأخذ المنهج الثاني بارمتراً إضافياً وهو كائن من النمط Properties إلى جانب السلسلة المحرفية url. يمكنك استخدام هذا المنهج عند الحاجة إلى تمرير معلومات معينة إلى قاعدة بيانات أثناء إجراء الاتصال. للقيام بذلك ما عليك سوى إعداد الزوج (اسم – قيمة) التابع للكائن ومن ثم تمريره كبارمتر عند استدعاء المنهج.

أما المنهج الثالث فهو المنهج الأكثر استخداماً والأكثر فاعلية وواضح أنَّه يتطلب ثلاثة بارامترات السلسلة url والسلسلة user والتي تعبر عن ا سم المستخدم والسلسلة password والتي تعبر عن كلمة المرور الخاصة بالمستخدم.

بشكل عام عندما تقوم باستدعاء المنهج getConnection() مع الكائن DriverManager فإنه يعيد كائناً من النمط Connection ولكن كيف يتم ذلك؟

يمرر الكائن DriverManager بارمترات المنهج getConnection() إلى كل الكائنات Driver المسجلة في قائمته، وهكذا تبدأ العملية بمشغل JDBC الأول حيث يمرر إليه البارمترات ويطلب منه محاولة الاتصال فإذا فشل سينتقل إلى المشغل التالي وهكذا يكرر العملية حتى يجد مشغلاً يستطيع الاتصال بقاعدة البيانات الموصَّفة بالبارمتر JDBC URL.

وفي حال أخفق تأسيس الاتصال مع قاعدة البيانات عندها يرمي الكائن DriverManager الاستثناء SQLException الذي يحتوي على رسالة خطأ يوضح فيها سبب الإخفاق لذلك لابد لك من التآلف مع هذه الرسائل لتستطيع تحديد موطن الخلل بسرعة.

قد يتساءل القارئ:
ماذا سيحدث إذا كان هنالك مشغلان مسجلان يستطيعان الاتصال بقاعدة البيانات وفقاً للبارمترات الممررة؟
الجواب بسيط جداً:
سيختار الكائن DriverManager المشغل الأول ويستخدمه لتأسيس اتصال ناجح.

توضح الشفرة التالية الطريقة القياسية لتأسيس اتصال مع قاعدة البيانات باستخدام الكائن DriverManager.

```
String url = "jdbc:oracle:thin:@myServer:1521:PROD";
String user = "boss";
String pwd = "bosspwd";
Connection conn = DriverManager.getConnection(url,user,pwd);
```

يمكن تأسيس الاتصال بطريقة أخرى. فمن الممكن استعادة المشغل الهدف من قائمة الكائن DriverManager وبالتالي استخدامه بشكل مباشر لتأسيس الاتصال وذلك باستدعاء المنهج Driver.Connection().

أو يمكنك إنشاء مرجع من الكائن Driver بشكل مباشر واستخدماه للاتصال. تمثل الشيفرة التالية كيفية تأسيس الاتصال باستخدام مرجع جديد للكائن Driver.

```
Driver drv = new sun.jdbc.odbc.JdbcOdbcDriver();
Properties prop = new Properties ();
Connection conn = drv.connect("jdbc:odbc:authors",prop);
```

استخدمت المتحول prop لتضمينه مجموعة من البارمترات، مثل اسم المستخدم وكلمة المرور المطلوبة للاتصال بقاعدة البيانات، ولكنه بقي فارغاً في الشيفرة السابقة.

18.11.12 إنشاء أول تطبيق JDBC بواسطة مشغل ODBC:

بعد أن حصلت على مشغل JDBC وتعلمت كيفية إعداده وتكونت لديك صورة مبدئية عن مكونات العمل في تطبيق JDBC ، دعنا نستعرض مثالاً لإنشاء تطبيق JDBC بسيط والذي سترى من خلاله الآلية المستخدمة لتأسيس اتصال مع قاعدة البيانات وتنفيذ استعلام وعرض ناتج الاستعلام.

يمكنك اعتبار هذا التطبيق نموذجاً عاماً يمكنك العودة إليه عند الحاجة لإنشاء تطبيقات JDBC في المستقبل.

ستكتشف من خلال هذا المثال بأنَّ برمجة JDBC قائمة على إتباع مجموعة من الخطوات الثابتة، ولإجراء كل منها ما عليك سوى إتباع أسلوب متشابه في كل مرة تقوم بذلك.

ففي كل تطبيقات JDBC يجب أن تحمّل مشغل JDBC ومن ثم تقوم بتأسيس اتصال مع قاعدة البيانات ومن ثم تنفذ استعلامك، وكل مرحلة لها نفس الخطوات المستقلة عن طبيعة التطبيق الذي تقوم بإنشائه لذلك قد تلجأ في المستقبل لبناء أصنافك وواجهاتك المختصة بكل مرحلة من هذه المراحل.

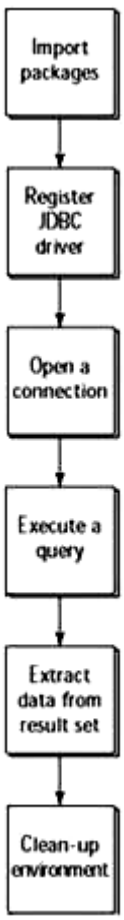
سأستخدم من أجل هذا التطبيق مخطط قاعدة البيانات الموضح في الشكل 16-18.

SSN	NAME	SALARY	HIREDATE
١١١١١١١١١	Todd	٥٠٠٠,٠٠	١٩٩٥-٠٩-١٦
٤١٩٨٧٦٥٤١	Larry	١٥٠٠,٧٥	٢٠٠١-٠٢-٠٥
٣١٢٦٥٤٩٨٧	Lori	٢٠٠٠,٩٥	١٩٩٩-٠١-١١
١٢٣٤٥٦٧٨٩	Jimmy	٣٠٨٠,٠٥	١٩٩٧-٠٩-٠٧
٩٨٧٦٥٤٣٢١	John	٤٢٥١,٢٧	١٩٩٦-١٢-٣١

شكل 16-18: قاعدة البيانات Employee

خطوات العمل

يوضح (الشكل17-18) الخطوات الست المتبعة لإنشاء هذا التطبيق والتي سأستعرضها خطوة بخطوة:



شكل 17-18: الخطوات المتبعة في بناء تطبيق JDBC.

1. تضمين الحزم المطلوبة (Import Packages)
يحتاج هذا التطبيق لتضمين الحزم المحتوية على أصناف وواجهات JDBC التي سنستخدمها هنا. غالباً ما تستخدم التعليمة ;*.import java.sql. بدلاً من سرد ستة تعليمات Import،كما فعلنا هنا.

2. تسجيل مشغل JDBC(Register The JDBC driver)
بعد القيام بإعداد المشغل لابد من تسجيله للتحقق من إمكانية فتح قناة اتصال مع قاعدة البيانات.

3. فتح الاتصال (Open Connection)
سنقوم باستخدام المنهج DriverManager.getConnection() لإنشاء كائن الاتصال. والذي يمثل اتصالاً فيزيائياً مع قاعدة البيانات.

4. تنفيذ استعلام (execute a query)
سأستخدم كائناً من النمط Statement من أجل بناء وإرسال عبارات (استعلامات) SQL إلى قاعدة البيانات.

5. استخراج البيانات (Extract The data Form ResultSet)
سأستخدم مجموعة المناهج ResultSet.getXXX() لاستعادة البيانات.

6. تنظيف بيئة العمل (Clean Up environment)
سأقوم بإغلاق كل موارد قاعدة البيانات من اتصالات وغيرها.

سنلقي الضوء في الفقرة التالية على كل خطوة من هذه الخطوات على حدا، ولكن في البداية دعنا نلقي نظرة سريعة على هذا التطبيق والمهام التي يؤديها.

ينشأ التطبيق اتصالاً مع قاعدة البيانات Employee، ومن ثم يرسل استعلام SQL يطلب من خلاله معلومات من جدول الموظفين، ثم يقوم بعرض هذه المعلومات وإغلاق الاتصال مع قاعدة البيانات.

ستلاحظ بأن كامل الشيفرة مغلفة بالكتلة try…..finally لأن أغلب مناهج JDBC تلقي الاستثناء SQLException في حال الإخفاق، لذلك لابد من مراعاة هذا الأمر.

كما ستلاحظ أيضاً بأننا قمنا بإنشاء كائن الاتصال خارج الكتلة try…finally لكي يتسنى الوصول إليه ضمن عبارة finally للتحقق من أنه سيتم إغلاقه بشكل مؤكد.

```
1. // برنامج يعرض محتويات جدول الموظفين
2. package Article1;
3. //STEP 1. Import packages
4. import java.sql.DriverManager;
5. import java.sql.Connection;
6. import java.sql.Statement;
7. import java.sql.ResultSet;
8. import java.sql.Date;
9. import java.sql.SQLException;
10.
11. public class Chp18_1 {
12.   public static void main(String[] args) {
13.     //Define Connection variable
14.     Connection conn = null;
15.
16.     //Begin standard error handling
17.     try{
18.       //STEP 2: Register JDBC driver
19.       String driver = "sun.jdbc.odbc.JdbcOdbcDriver";
20.       Class.forName(driver);
21.
22.       //STEP 3: Open a connection
23.       System.out.println("Connecting to database...");
24.       conn = DriverManager.getConnection("jdbc:odbc:DBJava","scott","tiger");
25.
26.       //STEP 4: Execute a query
27.       Statement stmt = conn.createStatement();
28.       String sql;
29.       sql = "SELECT SSN, NAME, SALARY, HIREDATE FROM Employee";
30.       ResultSet rs = stmt.executeQuery(sql);
31.
32.       //STEP 5: Extract data from result set
33.       while(rs.next()){
34.         //Retrieve by column name
35.         int ssn= rs.getInt("ssn");
```

```
36.         String name = rs.getString("name");
37.         //Retrieve by column index as an example
38.         double salary = rs.getDouble(3);
39.         Date date = rs.getDate(4);
40.
41.         //Display values
42.         System.out.print("SSN: " + ssn);
43.         System.out.print(", Name: " + name);
44.         System.out.print(", Salary: $" + salary);
45.         System.out.println(", HireDate: " + date);
46.       }
47.
48.       //STEP 6: Clean-up environment
49.       rs.close();
50.       stmt.close();
51.       conn.close();
52.     }catch(SQLException se){
53.
54.       //Handle errors for JDBC
55.       se.printStackTrace();
56.     }catch(Exception e){
57.
58.       //Handle errors for Class.forName
59.       e.printStackTrace();
60.     }finally{
61.
62.         //finally block used to close resources
63.         try{
64.           if(conn!=null)
65.             conn.close();
66.         }catch(SQLException se){
67.           se.printStackTrace();
68.     }//end finally try
69.   }//end try
70.   System.out.println("Goodbye!");
71.  }//end main
72. }//end Chp18_1
```

```
General Output
2 Connecting to database...
3 SSN: 1, Name: Ammar, Salary: $12099.0, HireDate: 1999-09-09
4 SSN: 2, Name: Aldopaee, Salary: $100000.0, HireDate: 2007-07-07
5 SSN: 3, Name: Ali, Salary: $168.0, HireDate: 2000-02-02
6 Goodbye!
```

وبالإطلاع على الخرج ستجد بأن الاستعلام قد أعاد كل معلومات الجدول Employee.

شرح المثال بشكل مفصل
الخطوة الأولى: تضمين الحزم المطلوب .
كما هي العادة في كل تطبيقات Java يجب تضمين الحزم المطلوبة في بداية الصنف للوصول إلى الأصناف الموجودة ضمنها. ولحسن الحظ فإن كل الواجهات والأصناف الخاصة بالواجهة JDBC موجودة في الحزمتين java.sql وjavax.sql ، ولأننا سنتعامل مع مبادئ JDBC الأساسية، اكتفينا بتضمين مكونات من الحزمة java.sql فقط كما في الأسطر 4-9.
في حين ستقوم بتضمين الحزمة javax.sql عندما تبدأ بتطوير تطبيقات مؤسساتية وتحتاج لاستخدام ميزات JDBC المتقدمة.

سنقوم باستعراض الواجهات والأصناف المضمنة في التطبيق Chp18_1 مع تعليق بسيط على كل منها:

- java.sql.DriverManager: تتحكم بمشغلات JDBC. تتحكم هذه الواجهة بالكائنات التابعة للواجهة Driver ، يمكنك استخدامها في عملية إنشاء الاتصال مع قاعدة البيانات حسب طبيعة المشغل المستخدم.

- java.sql.Connection: تمثل الاتصال الفيزيائي مع قاعدة البيانات، وتتحكم كائناتها بمستويات المناقلة، وبنوع ناتج العبارة الذي سيتم إنشاؤه بعد تنفيذ الاستعلام.

- java.sql.Statement: ترسل استعلامات SQL إلى قاعدة البيانات. تمكنك هذه الواجهة من إرسال استعلامات ساكنة (استعلامات تحديد Select). في حين يمكنك استخدام الواجهة java.sql.PreparedStatement من أجل إرسال استعلامات تقبل بارمترات مثل (Select Where = [ ]).

- java.sql.ResultSet: تحتفظ بمعطيات ناتج عبارة SQL، وتزودك بممرر (iterator) يسمح باستخدام معطيات الكائن ResultSet.

- java.sql.Date: عبارة عن معرِّف نوع معطيات خاص بالواجهة JDBC، يسمح بالتعامل مع نوع المعطيات SQL DATE في Java .

- java.sql.SQLException: يعالج أخطاء قاعدة البيانات، واستثناءات JDBC البرمجية.

الخطوة الثانية: تسجيل مشغل JDBC.
يعبر مشغل JDBC عن الطريقة التي سيتم بها الاتصال مع قاعدة البيانات أو ما يسمى "Know-how". لذلك يتوجب تسجيل المشغل الذي يتعامل مع JDBC أثناء كتابة التطبيق.

هنالك طريقتان للقيام بذلك:

1. استخدام المنهج:
DriverManager.registerDriver (Driver driverClassName).

2. استخدام المنهج: Class.forName(String driverClassName)
يعتبر استخدام المنهج Class.forName() أفضل، لأنه يقبل بارامتراً من النمط String لتمثيل اسم الصنف المشغل. وبذلك ستتمكن من تمرير البارامتر الخاص بمشغلك في زمن التنفيذ من خلال سطر الأوامر (Console) أو من خلال جلبه من ملف.

أما استخدام المنهج DriverManager.register Driver() فيتطلب نوع المشغل كبارمتر. لذاك يتوجب عليك إنشاء كائن من النمط Diver وتمريره إلى المنهج.

لا ننصحك باستخدام هذا الأسلوب لأنه سيحد من إمكانية تطوير تطبيقك مستقبلاً، فعندما تحتاج لتطوير تطبيق يتعامل مع نوعين أو أكثر من قواعد البيانات ستكون حكيماً بإتباعك للأسلوب الأول لأنه سيجعل التطبيق أكثر مرونة.

بشكل عام إن هذه النصيحة تعبر عن وجهة نظر شخصية وذلك لأن الأسلوبين السابقين متماثلين في الأداء تماماً.

```
//STEP 2: Register JDBC driver.
String driver = "sun.jdbc.odbc.JdbcOdbcDriver";
Class.forName(driver);
```

استخدمت من أجل هذا التطبيق "sun.jdbc.odbc.JdbcOdbcDriver" حيث تمثل السلسلة المحرفية Diver الاسم الكامل للصنف المشغل، وبتمريرها إلى المنهج Class.forName() ستتم عملية التسجيل والأسطر 20-18تبين ذلك.

سند عود للقول بأنَّ أداء الأسلوبين السابقين متماثل تماماً، لأنك إذا قمت بتفحص الكائنات في الذاكرة سترى كائناً من DriverManager حتى لو كنت تستخدم المنهج Class.forName(). يعود ذلك إلى القاعدة التالية والتي تشكل جزءاً من مواصفات JDBC:

" ستسجل كل الكائنات التي تنفذ الواجهة Driver ذاتياً مع الواجهة DriverManager".

يلقي المنهج Class.forName() الاستثناء ClassNotFoundException، إذا لم يكن قادراً على إيجاد المشغل المحدد بالبارمتر الممرر إليه أثناء تنفيذ التطبيق. لذلك قمت بمعالجة هذا الاستثناء بالتقاط الاستثناء القياسي (Standard Exception):

```
Catch(Exception e } (

e.printStackTrace ();
```

أما المنهج DriverManager.registerDriver() فيلقي الاستثناء SQLException في حالة وقوع أي خطأ أثناء عملية التسجيل.

الخطوة الثالثة: فتح الاتصال مع قاعدة البيانات.

تعمل معظم تطبيقات قواعد البيانات في بيئة زبون- مخدم. تعتبر تطبيقات JDBC تطبيقات زبونة تتخاطب مع المخدم عن طريق تأسيس اتصال فيزيائي به. حتى ولو كانت هذه التطبيقات تعمل على مضيف قاعدة البيانات نفسه.

وكما ذكر نا سابقاً فإن كائنات Connection تمثل اتصالاً فيزيائياً مع قاعدة البيانات، أو بمعنى آخر: تمثل القناة التي يتفاعل عن طريقها تطبيق JDBC الزبون مع مخدم قاعدة البيانات.

دعنا نأخذ مثالاً توضيحياً:

تريد تنفيذ استعلام ما، بالطبع لن تستطيع استعمال كائن Connection مباشرة لتنفيذ الاستعلام، بل ستقوم بإنشاء كائن ينفذ الواجهة Statement وتستدعي المنهج Statement.excuteQurey() التابع للكائن Connection؟

تقول الإجابة: يمثل الكائن Connection مولداً لإنشاء كائنات Statement والتي ستستخدمه لتقانة إرسال الاستعلام إلى قاعدة البيانات، وبعدها ستستخدمه كآلية لاستعادة ناتج الاستعلام. ومن ثم تقوم كائنات Statement بنقل ناتج العبارة إلى كائنات ResultSet المختصة بذلك.

بشكل عام: تحتاج من أجل استدعاء المنهج DriverManager.getConnection() إلى تأسيس الاتصال مع قاعدة البيانات. يقبل هذا المنهج العديد من البارامترات والتي تطلبها قاعدة البيانات لقبول تأسيس الاتصال.

أكثر هذه البارامترات استخداماً هي:

- JDBC URL: يحدد هذا البارامتر معلومات عن موضع قاعدة البيانات وعن إعدادات المشغل المستخدم. تعتمد الصيغة المستخدمة في تمثيل هذا البارمتر على مزود قاعدة البيانات بالإضافة إلى متطلبات خاصة بالمشغل التي تم أنشائة سابقاً ODBC .

- Username: يمثل هذا البارامتر اسم المست خدم الذي ستسجل الدخول إلى حسابه في قاعدة البيانات (يحدد اسم المستخدم صلاحيات المستخدم وهذا ما يسمى بحسابه scott).

- Password: يمثل هذا البارامتر كلمة المرور الخاصة باسم المستخدم المحدد في البارمتر السابق.

والسطر 24 يبين ذلك

```
conn = DriverManager.getConnection("jdbc:odbc:DBJava","scott","tiger");
```

بعد تجميع هذه البارامترات تمرر إلى المنهج DriverManager.getConnection() ليقوم بإنشاء اتصال فيزيائي مع قاعدة البيانات في حال كون هذه البارامترات صحيحة أو يلقي الاستثناء SQLException في حال حدوث أي خطأ.

الخطوة الرابعة: تنفيذ استعلام SQL.

بعد أن قمنا بفتح الاتصال مع قاعدة البيانات، أصبح من الممكن تنفيذ أي استعلام SQL. يمكنك JDBC من استخدام اللغة DDL (لغة تعريف البيانات) لإنشاء

قاعدة البيانات وبنيتها الداخلية، كما يمكنك أيضاً من استخدام اللغة DML(لغة معالجة البيانات) لمعالجة البيانات واستعادتها.

ولكن عليك قبل ذلك أن تتأكد من الصلاحيات المتاحة لك لتنفيذ استعلام ما. لأن تجاوز الصلاحيات سيتسبب برمي الاستثناء SQLException.

تحتاج عملية استعلام قاعدة البيانات إلى كائنين، بحيث ينفذ الكائن الأول إحدى الواجهات التالية
(Statement،PreparedStatement،CallableStatement).

يوضح الجدول 5-18 وظيفة كل واجهة من هذه الواجهات.

جدول 5-18

الواجهة	الوصف
java.sql.Statement	تمكنك هذه الواجهة من إرسال استعلامات SQL ساكنة (ستاتيكية) إلى قاعدة البيانات.
java.sql.PrepareStatement	تعمل هذه الواجهة مع استعلامات SQL التي تقبل البارمترات، وتحتفظ بالبارمترات الممررة في الاستعلام مما يسرع تنفيذ الاستعلام في المرة التالية.
java.sql.CallableStatement	تمكنك هذه الواجهة من الوصول إلى الإجرائيات المخزنة في قاعدة البيانات وتنفيذها.

أما الكائن الثاني في السطر 30 والذي تحتاجه لتنفيذ الاستعلام فهو كائن من النمط ResultSet، سيحتفظ هذا الكائن بناتج الاستعلام كما أنه يقدم ممرراً تستطيع من خلاله الحصول على المعطيات الناتجة عن الاستعلام وتقوم بإظهارها.

استخدمنا في السطر 27 كائن العبارة الأساسي (Statement)، لإرسال استعلام (Select) بسيط حيث استخدمنا في الأسطر 30-26 هذه الشيفرة:

```
//STEP 4: Execute a query
    Statement stmt = conn.createStatement();
    String sql;
    sql = "SELECT SSN, NAME, SALARY, HIREDATE FROM Employee";
    ResultSet rs = stmt.executeQuery(sql);
```

استدعينا المنهج Conn.createStatement() لإنشاء الكائن Statement. لأنه كما ذكر نا في الفقرة السابقة بأن الكائن Connection يمثل الواجهة الرئيسية لقا عدة البيانات، أما الكائنات الأخرى المستخدمة للتخاطب مع القاعدة فتشتق منه والسطر 27 يوضح ذلك.

بعد ذلك قمنا بتعريف سلسلة محرفية (السلسلة SQL) ، مهمتها احتواء عبارة SQL. أخيراً، استدعيت المنهج Statement.excuteQuery() لتنفيذ عبارة SQL، وخزنت نتيجتها ضمن الكائن rs الذي ينتمي إلى الواجهة ResultSet كما في الأسطر 29-30 .


سيتم إلقاء الاستثناء SQLException في حال وقوع أي خطأ أثناء تنفيذ الشيفرة السابقة.

الخطوة الخامسة: إظهار معطيات ناتج العبارة.
تعتبر هذه الخطوة أهم مراحل بناء تطبيق قواعد البيانات، لذلك سنستعرضها بشيء من التفصيل.
السطر 30 يـ تمثل مهمة الكائن ResultSet بالاحتفاظ بالمعطيات الناتجة عن عبارة SQL. فهو يخزن المعطيات على شكل جدول مؤلف من صفوف وأعمدة.

وفي مثالنا تمثل صفوف هذا الجدول المعطيات المتعلقة بكل الموظفين، ويعود ذلك إلى طبيعة الاستعلام المنفذ. فمثلاً لو أضفنا العبارة التالية إلى الاستعلام " Where Name = Thomas" ، ستقتصر معطيات الصفوف على المعلومات المتعلقة بالموظف Thomas فقط.

أما أعمدة الكائن ResultSet فتابعة للواصفات المحددة في القسم Select من عبارة SQL وفي مثالنا اخترنا واصفات الرقم والاسم وتاريخ المباشرة والراتب.

ويجدر بالذكر بأن نوع معطيات أعمدة الكائن ResultSet هي نفسها نوع معطيات الأعمدة الموجودة على المخدم، لذلك يستخدم المنهج ResultSet.get() لاستعادة معطيات الأعمدة من قاعدة البيانات وفق نوع معطيات مقابل ومتوافق مع اللغة Java.

يستخدم الكائن ResultSet مؤشراً للإشارة إلى صفوف المعطيات في ناتج العبارة. لذلك يتوجب التنقل باستخدام المؤشر من صف إلى آخر.

تعتبر عملية التنقل بسيطة جداً، فما عليك سوى استدعاء المنهج ResultSet.next() والذي سينقل المؤشر إلى الصف التالي.

هنالك موضعين مميزين للمؤشر، الموضع قبل الصف الأول BFR (Before first Row) والموضع بعد آخر صف ALR(After Last Row) ، وواضح أنّ صفوف هذين الموضعين لا تحتوي على أية معطيات، بل إن محاولة استعادة المعطيات منهما ستولد استثناء SQLException.

يتوضع المؤشر بعد تهيئة الكائن ResultSet بالموضع BFR ، لذلك يجب نقله إلى الموضع التالي الحاوي على معطيات قبل البدء بعملية استعادة المعطيات.

غالباً ما يستخدم المنهج ResultSet.next() لنقل موضع المؤشر إلى الصف التالي.
سنقل الآن للتعليق على كيفية استخدام الكائن ResultSet في مثالانا ، حيث يوضح السرد التالي الجزء من الشيفرة والخاص بالخطوة الخامسة كما في الأسطر 32-46:

```
//STEP 5: Extract data from result set
while(rs.next()){
  //Retrieve by column name
  int ssn= rs.getInt("ssn");
  String name = rs.getString("name");
  //Retrieve by column index as an example
  double salary = rs.getDouble(3);
  Date date = rs.getDate(4);

  //Display values
  System.out.print("SSN: " + ssn);
  System.out.print(", Name: " + name);
  System.out.print(", Salary: $" + salary);
  System.out.println(", HireDate: " + date);
}
```

استخدمنا حلقة While من الشرط rs.next() للتحقق من المرور على كل سجلات الجدول. حيث سيعيد المنهج rs.next() القيمة true عند الانتقال إلى سجل مقبول (حاوي على معطيات) أو يعيد القيمة false بخلاف ذلك.

في كل مرور بالحلقة While سند قوم بمسح كل خلايا السجل الحالي، واستخراج القيم منها بعد تحويلها إلى أنواع معطيات متوافقة مع اللغة java وذلك باستخدام المنهج ()getXXX. من ثم سند خزن هذه القيم في متحولات وسيطة (SSN، Name، Salary، date) لاستخدامها في عملية الخرج.

يقبل المنهج ()ResultSet.getXXX بارمتراً وحيداً. إما سلسلة محرفية تمثل اسم العمود أو عدد صحيح يمثل رقم العمود المطلوب لاستخراج معطياته.

يختلف نظام الترقيم المستخدم لتحديد رقم العمود عن نظام ترقيم المصفوفات في اللغة java. فالترقيم هنا يبدأ من الرقم 1.... وهكذا، وليس من الرقم 0 كما هي العادة.

الخطوة السادسة: تنظيف بيئة العمل.
يتم أغلاق الكائنات. ستلاحظ بأننا قمنا بإغلاق الكائنات ResultSet،Statement،Connection على الترتيب. مع العلم بأن أغلب تصريحات مزودي مشغلات JDBC تكتفي بإغلاق الكائن Connection والذي سيغلق بقية الكائنات بشكل تلقائي.

نؤكد على إتباع هذه الطريقة لأنها الأكثر سلامة، والأكثر تفضيلاً من قبل المبرمجين.

18.11.13 إغلاق اتصالات JDBC

لقد قمنا في الشيفرة السابقة بإغلاق الاتصال مع قاعدة البيانات بشكل صريح، وذلك لإنهاء جلسة العمل، وبطبيعة الحال فإن مجمع النفايات الخاص بلغة Java سيتولى هذه العملية في حال تم تجاهلها من قبل المبرمج .
ولكن الاعتماد على مجمع النفايات لإغلاق الاتصال في برمجة قواعد البيانات ليس بالأمر المستحسن إطلاقاً .

لذلك عليك التفكير وبشكل مستمر بإغلاق كل الاتصالات في تطبيقك باستخدام المنهج .( ) Close يأتي هذا لعدّة أسباب :

فبإغلاقك الاتصال مع قاعدة البيانات ستتأكد من انتهاء جلسة عمل التطبيق الزبون على مخدم قاعدة البيانات ، تأتي أهمية هذا الإجراء لكون بعض أنواع قواعد البيانات تتحسس لانتهاء جلسة الزبون وبالتالي يمكن أن تنفذ إجراءاً تابعاً لذلك. كما أنَّ الفشل المفاجئ في الاتصال أو عدم إغلاق الاتصال بقاعدة البيانات سيفسر من قبل بعض أنواع قواعد البيانات على أنه فشل في الجلسة و بالتالي سيتم تجاهل التغيرات التي قام بها التطبيق الزبون أثناء جلسة العمل. فمثلاً يستخدم Oracle هذه الطريقة حيث يتم تجاهل كل التغييرات التي حصلت في الجلسة إذا لم يتم إنهائها بشكل واضح وصريح وهنا ما يسمى بالمناقلة Transaction ، كما أنَّ إغلاق الاتصال بشكل صحيح يتبعه سلوك وقائي من قبل نظام إدارة قاعدة البيانات يتمثل بتنظيف بيئة العمل التي استخدمها التطبيق الزبون.

إنَّ إغلاق اتصالات قاعدة البيانات يوّفر من موارد نظام DBMS ، وهذه نقطة هامة ولا يمكن لمبرمج جاد إغفالها. فالمحافظة على موارد النظام DBMS أمر هام ويشـ دد عليه مدراء قواعد البيانات لأن ترك الاتصالات مفتوحة سيشكل عبئاً إضافياً على مخدم قاعدة البيانات لأنه يستهلك قدراً من الذاكرة RAM ومن قدرات المعالج CPU.

ولا يقتصر هذا على المخدم بل على حاسب الزبون أيضاً. فإن إغلاق الاتصال سيوفر من موارد الزبون والمخدم في آن واحد.

ينصح باستخدام الكتلة Finally في الشيفرة البرمجية للتحقق من إغلاق الاتصال بشكل مؤكد. فتعليمات الكتلة ستنفذ سواءً حدث استثناء ما أو لم يحدث وبذلك ستتأكد من المحافظة على مواد قاعدة بياناتك لأنك ستجعل عملية إغلاق الاتصال عملية إجبارية.

تمثل الشيفرة التالية كيفية استخدام الكتلة Finally لعملية التحقق من إغلاق الاتصال بشكل مؤكد.

```
finally {
      try {
        if (connection!=null) {
                    String msg = "Closing connection from finally block.";
                    System.out.println(msg);
                    connection.close();
                          }
```

```
} catch(SQLException se) {
                    se.printStackTrace();
                }
}
```

18.11.14 بناء عبارات JDBC

تعرَّفنا في السابق (الاتصال بقواعد البيانات باستخدام JDBC) على كيفية الاتصال بقواعد البيانات باستخدام الكائنات DriverManager والكائنات Connection. وذكرنا مثالاً على ذلك أما الآن فسأنتقل لاستعراض الخطوة الثانية من خطوات بناء تطبيق JDBC ، حيث ستزودك في هذة الفقرة معرفة تفصيلية عن كيفية التخاطب مع قاعدة البيانات.

هنالك أساليب متنوعة للتخاطب مع قواعد البيانات، فمثلاً قد ترسل استعلام SQL إلى قاعدة البيانات لاستعادة مجموعة جز ئية من المعطيات تقابل الشروط الموصَّفة في الاستعلام، أو قد يكون استعلام SQL المرسل استعلاماً إنشائياً يضيف بنية معينة إلى مخطط قاعدة البيانات.

تصادفك في بعض الأحيان حالات لا تستطيع فيها تحديد قيم الحقول المطلوبة في الاستعلام. لذلك ستلجأ إلى الحصول على هذ ه القيم أثناء زمن التنفيذ. وذلك باستخدام استعلامات SQL التي تقبل البارامترات، أو باستخدام أوامر قاعدة البيانات. كما قد تواجه حالات أخرى تتطلب منك استدعاء الإجرائيات المخزنة في قاعدة البيانات للحصول على أداء أفضل.

بشكل عام، تزودك مجموعة الكائنات (Statement و PreparedStatement و CallableStatement) بالأدوات المناسبة لتنفيذ المهمة المطلوبة منك. سند قوم في هذه الفقرة بتغطية تفاصيل العمل مع هذه الكائنات، وسند ركز على تغطية مواضيع إنشاء هذه الكائنات واستخدامها لإرسال الأوامر، وذلك من خلال مناقشة عملية مدعومة بالأمثلة التوضيحية المناسبة.

18.11.15 استخدام عبارات JDBC وكائناتة

تعرِّف مجموعة الواجهات (Statement و PreparedStatement و CallableStatement) مناهج وخصائص تمكنك من إرسال الأوامر إلى قاعدة البيانات ومن ثم استعادة المعطيات منها، كما أنها تعرِّف مناهج تساعدك في

تجاوز الاختلاف القائم بين أنواع معطيات SQL وأنواع معطيات Java.

تعتبر مسألة الاختلاف مسألة هامة، فمثلاً لا يجوز تمثيل نوع المعطيات الأولي int بالقيمة NULL في اللغة Java ، في حين تستخدم قواعد المعطيات هذه القيمة وبشكل أساسي لتمثيل حقول المعطيات الخالية وخاصة حقول المعطيات الرقمية.
يعتبر نوع المعطيات الممثل للتاريخ والوقت مثالاً آخر عن هذا الاختلاف، فالصيغة التي تستخدمها Java لتمثيل هذا النو ع تختلف كلياً عن الصيغة المحددة بقواعد SQL-92 المعيارية لتمثيل هذا النوع، وكما ذكرت تعالج واجهات العبارة (Statement) هذا الاختلاف.

عندما تحتاج لاستعلام قاعدة بياناتك عليك استخدام إحدى واجهات العبارة الثلاث، ولكن أي واجهة بالتحديد؟ يعود ذلك إلى نوع الإجراء المراد تنفيذه، فمثلاً قد تحتاج لاستعادة معطيات من القاعدة وتقديمها للمستخدم، أو تحتاج لإجراء تحديث على معطيات القاعدة، أو لاستدعاء إجراء مخزن في القاعدة وغير ذلك من الاحتياجات التي يستخدم في كل منها واجهة معينة مع مجموعة من المناهج أو غيرها.

إذاً هنالك نقاط اختلاف بين واجهات العبارة الثلاث، ولكن هذا الاختلاف ليس اختلافاً كلياً.فهنالك الكثير من التشابه بين الواجهات.
يقدم مزودي المشغلات مشغلاتهم على شكل أصناف تنفذ هذه الواجهات، لذلك لا يمكنك استخدام ميزّات هذه الواجهات بدون استخدام مشغل JDBC.

سنقدم في الجدول 5-18 شرحاً أولياً لاستخدامات كل واجهة من واجهات العبارة.

جدول 5-18

الواجهة	الاستخدام المفضل
Statement	تستخدم لاحتياجات الوصول العامة إلى قاعدة المعطيات، من المفيد جداً استخدام هذه الواجهة عند استخدام عبارات SQL الستاتيكية في زمن التنفيذ. مع ملاحظة أنها لا تقبل أي بارامترات.
PreparedStatement	تستخدم عند الحاجة لاستخدام عبارات SQL بشكل متكرر. فهي أسرع من الكائنات Statement التي تعاد ترجمتها في كل استدعاء، كما أنها تقبل تمرير بارامترات في زمن التنفيذ.
	تستخدم عند الحاجة للوصول إلى الإجرائيات المخزنة

CallableStatement	في قاعدة البيانات، تقبل هذه الواجهة تمرير بارامترات في زمن التنفيذ أيضاً.

- **مدخل إلى استخدام الكائنات Statement**

سنركز في هذه الفقرة على الكائن Statement من حيث التأسيس الهيكلي للواجهة statement لأن ترسيخ مفهوم هذا الكائن سيساعدك على توضيح مفهوم الكائنين PreparedStatement وCallableStatement.

يمنحك الكائن Statement الإمكانيات الأساسية للتخاطب مع قاعدة البيانات. كما يقدم بعض الإمكانيات المتقدمة الأخرى. حيث يمكنك من استخدام كل أنواع عبارات DML و DDL ، أو أية أوامر خاصة بقاعدة بياناتك. ويدعم بالإضافة إلى ذلك التحديثات المتسلسلة(Patch Updating) والتي تمكنك من استخدام سلوك المناقلة في تطبيقك.

- **إنشاء كائنات Statement**

تعتبر عملية إنشاء مرجع لكائن Statement عملية منهجية، فإنشاء الكائن Statement مرتبط بإنشائك لاتصال فعّال مع قاعدة البيانات كما هو موضح في الشيفرة التالية:

```
Connection conn = DriverManager.getConnection(url, "toddt", "mypwd";

Statement stmt = conn.createStatement();
```

- **استخدام الكائن Statement**

يمتلك الكائن Statement ثلاثة مناهج تشكل صلة الوصل مع قاعدة البيانات، فباستخدامها يمكنك إرسال الاستعلامات واستعادة نتائجها، يستعرض الجدول 6-18 هذه المناهج موضحاً نقاط الاختلاف فيما بينها.

جدول 6-18

المنهج	الاستخدام المفضل
	يستخدم هذا المنهج لاستعلام قاعدة البيانات باستخدام عبارة

executeQuery	يستخدم هذا المنهج لتنفيذ عبارات SELECT، ويعيد هذا المنهج كائناً من النمط ResultSet.
executeUpdate	يستخدم هذا المنهج لتنفيذ عبارات (DELETE، UPDATE، INSERT) أو عبارات SQL DDL ، ويعيد عدد السجلات التي تأثرت بعبارات الإضافة أو التحديث أو الحذف، أو يعيد القيمة 0 للعبارات التي لا تعيد أي شيء قبل عبارات DDL.
execute	يستخدم هذا المنهج لمعالجة أية عبارات DML و DDL أو الأوامر الخاصة بقاعدة البيانات ويعيد كائناً أو أكثر من النمط ResultSet ، أو أنه يعيد عدد السجلات المتأثرة حسب نوع العبارة المنفذّة، كما أنه في بعض الأحيان يعيد ناتجاً مكوناً من النمطين السابقين. وعلى الرغم من سهولة استخدام هذا المنهج، إلا أنك قد تواجه بعض الصعوبات في استخراج النتائج منه.

يعيد المنهج executeQuery() كائناً من النمط ResultSet في حال نجاحه. أما الآن فسأقوم بتغطية مفاهيم استخدام المنهجين executeUpdate() و execute()، ولكن معظم هذه المفاهيم تنطبق على المنهج executeQuery().

18.11.16 الاستعلامات المرسلة إلى قاعدة البيانات بواسطة الكائن ResultSet

تمثل الكائنات Connection الاتصال الفيزيائي مع قاعدة البيانات، في حين تمكنك الكائنات Statement من تنفيذ استعلاماتك على قاعدة البيانات. أما هذه الكائنات فتشكل الأساس الذي يتيح لك إمكانية بناء مناظير (Views) إلى المعطيات في قاعدة البيانات.

يشير التعبير "result set" أو (ناتج العبارة) إلى معطيات صف و عمود محتوى في الكائن ResultSet ، وهذا ما يمثل منظاراً منطقياً لمعطيات العمود والصف في قاعدة البيانات والتي تقابل استعلام SQL المُرسل. يمكن أن يمتلك ناتج العبارة عدداً من الصفوف والأعمدة. حيث يعتمد هذا العدد على الاستعلام، فإذا كان استعلامك يضم شرطاً ضمن عبارة WHERE ، فسيكون لديك صف وحيد من المعطيات التي تقابل الشرط المطبق.

تعرّف الواجهة ResultSet مناهجاً تمكنك من التخاطب مع الـ معطيات المخزنة في قاعدة البيانات. حيث يحتوي مشغل JDBC صنفاً ResultSet ينفذ هذه الواجهة. فعندما يتم تنفيذ استعلام ما أو استعلام ذو بارامترات أو تنفيذ إجراء مخزن في قاعدة البيانات بنجاح. فسيتم إعادة كائن ResultSet.

يمكنك الكائن ResultSet القياسي من استعراض المعطيات في قاعدة البيانات والتي تقابل استعلام SQL المرسل، ولكن يمكنك إنشاء كائن ResultSet يقوم بتحديث معطيات صف ما تقوم باستعراضه، أو إضافة صف جديد إلى جدول ما، أو حتى حذف صف آخر. لذلك ستجد بأنَّ الكائن ResultSet سيمكنك من تنفيذ عبارات DML برمجياً ودون ا لتصريح بشكل مباشر عنها كعبارات INSERT أو UPDATE.

لا تدعم كل قواعد البيانات الإجرائيات المخزنة التي تعيد نواتج عبارة، لذلك

قد لا يعيد الكائن CallableStatement ناتج عبارة. عليك التحقق من ذلك بالعودة إلى التوثيق الخاص بمشغل JDBC المستخدم.

سن قوم في الفقر ات التالية بتقديم لمحة عن المفاهيم المطلوبة للتعامل مع الكائن ResultSet بشكل فعّال.

- **مدخل إلى مفاهيم ResultSet الأساسية**

على الرغم من أنَّ تقنية JDBC تعرّف العديد من أنواع نواتج العبارة لتلبية مختلف الاحتياجات البرمجية فإن هنالك عدّة مفاهيم أساسية تطبق على كل هذه الأنواع، لذلك قمنا بتخصيص الفقرات التالية لعرض هذه المفاهيم لتزودك بالأساسيات المساعدة للتمييز بين الأنواع المختلفة لنواتج العبارة.

مؤشرات ناتج العبارة (Result set cursors)

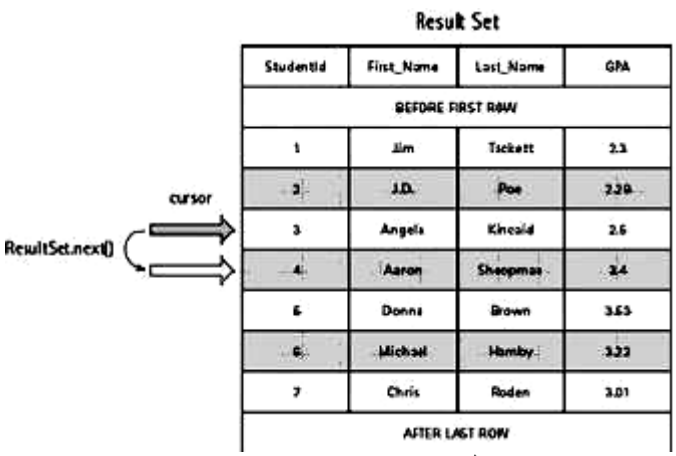
يحتوي ناتج العبارة على العديد من صفوف المعطيات، لكن يمكنك الوصول إلى صف واحد فقط في لحظة معينة. حيث يشير الكائن ResultSet إلى الصف الفعّال (active) باستخدام مؤشر (cursor) ، فإذا أردت الانتقال للإشارة إلى صف آخر،

عليك القيام بنقل المؤشر إلى هذا الصف باستخدام أحد مناهج الانتقال التابعة للكائن ResultSet.

يمثل الشكل 18-18 مؤشر ناتج العبارة وكيفية انتقاله عبر مجموعة المعطيات. ستلاحظ بأنَّ المؤشر يقف عند الصف الثالث من صفوف ناتج العبارة السبعة، فإذا استدعيت المنهج ResultSet.next() عندها سينقل المؤشر صفاً واحداً إلى الأمام ليقف عند الصف الرابع. لذلك عليك أن تتذكر دائماً بأنه عند التعامل مع معطيات ناتج العبارة فإنك تتعامل مع الصف الذي يقف عند المؤشر.

هنالك موضعين خاصيين للكائن ResultSet يوضحهما الشكل 18-18 أيضاً. الموضع الأول هو الموضع ما قبل الصف الأول BFR (Before First Row) والموضع الآخر هو الموضع ما بعد الصف الأخير ALR (After Last Row)..

لا يحتوي هذان الموصعان على أية معطيات، وإن استخدام أحد المناهج getXXX() أو updateXXX() معهما سيلقي الاستثناء SQLException.



شكل 18-18

عندما يتم تهيئة الكائن ResultSet لأول مرّة، سيكون المؤشر واقفاً عند الموضع BFR ، لذلك يفترض بك نقل المؤشر إلى أي موضع آخر يحتوي على معطيات قبل استدعاء أحد مناهج الوصول للمعطيات. كذلك فعند التنقل خلال الكائن ResultSet قد يصل المؤشر إلى الموضع ALR وعندها قد تتمكن من نقل موضع المؤشر مرة أخرى إلى موضع حاوي على معطيات وقد لا تستطيع القيام بذلك تبعاً لنوع ناتج العبارة المستخدم.

بشكل عام يمكنك إعادة بناء ناتج العبارة بتنفيذ استعلام SQL من جديد في حال لم تتمكن من نقل موضع المؤشر.

أنواع نواتج العبارة

يزودك JDBC بالعديد من الأنواع للكائن ResultSet، والتي تمكنّك من التخاطب مع قاعدة بياناتك بأساليب متنوعة. يمثل النوع الأول النوع القياسي، حيث يمتلك الحد الأدنى من الوظيفية التي تمكنك من التنقل في ناتج العبارة بالاتجاه الأمامي فقط، كما أنَّه لا يمكنك تحديث المعطيات في ناتج العبارة.

أما النوع الثاني يتيح لك إمكانية التنقل في ناتج العبارة بالاتجا ه الأمامي والخلفي، وحتى إلى أيّ صف ضمن ناتج العبارة، في حين يمكنك النوع الثالث من تحديث المعطيات المحتواة في ناتج العبارة.

عندما تقوم بإنشاء أحد الكائنات Statement أو PreparedStatement أو CallableStatement فإنك تعرّف نوع ناتج العبارة الذي ستستخدمه عن طريق تمرير البارامترات إلى المنهج التابع للكائن Connection والذي يقوم بإنشاء العبارة، حيث سينشأ ناتج عبارة من النوع الأول القياسي في حال عدم تمرير أي بارمتر.

يعرض الجدول -18 لمحة سريعة عن كل نوع من أنواع نواتج العبارة، حيث سنقوم بتغطية كل نوع منها على حدة.

جدول -18

ناتج العبارة	الوصف
Standard (القياسي)	يزودك بأساسيات الوصول إلى مجموعة معطيات ناتج العبارة. لا يقوم بنقل التغييرات التي تتم على معطياته إلى مخدم قاعدة البيانات. ينقل المؤشر باتجاه مفرد، الاتجاه الأمامي، لا يمكنه تحديث معطيات ناتج العبارة.
Scrollable (قابل للتنقل)	يزودك بقدرات تنقل مطورة لقدرات نوع ناتج العبارة الأساسي. ينقل المؤشر إلى الأمام وإلى الخلف وإلى أي صف

	معطيات محدد. يمكنه نقل التغييرات المطبقة على معطياته إلى مخدم قاعدة البيانات.
Updateable (قابل للتحديث)	يمكنك من تحديث المعطيات في قاعدة البيانات بتحديث معطيات ناتج العبارة. يتم استخدام هذا النوع بتحديد البارمتر ResultSetType عند إنشاء أحد الكائنات Statement، PreparedStatement، CallableStatement.

تتميز الكائنات التي تنتمي إلى النوعين Scrollable و Updateable بأنها ذات إقصاء تبادلي، فقد تجد كائناً من النوع Updateable لكنه ذو اتجاه تنقل أمامي فقط، أو تجد كائناً من النوع Scrollable ولكنه غير قابل للتحديث.

نواتج العبارة القياسية Standard

ناتج العبارة القياسي وحيد الاتجاه وغير قابل للتحديث. حيث تقتصر إمكانيات هذا النوع على نقل المؤشر بالاتجاه الأمامي فقط كما أنه لا يدعم تحديث معطيات ناتج العبارة أيضاً، وأي عملية نقل مؤشر معاكسة (للخلف مثلاً) أو عملية إضافة بيانات أو تعديلها سترمي الاستثناء SQLException.

لذلك يقتصر استخدام نواتج العبارة القياسية عند الحاجة لتنفيذ العمليات الأساسية على مجموعة معطيات ناتج العبارة، فمثلاً يمكنك استخدام ناتج العبارة القياسي من أجل تهيئة العنصر ListBox لكي يعرض قائمة بالمواد المتاحة في مخزن ما، أو يمكنك استخدام هذا النوع أيضاً من أجل المرور على كل صفوف معطيات الكائن ResultSet لمعرفة عدد الصفوف الموجودة في قاعدة البيانات والتي تقابل الشرط المحدد بعبارة WHERE في استعلام SQL المرسل.

نواتج العبارة القابلة للتنقل Scrollable

عليك استخدام النوع Scrollable من أنواع كائنات ناتج العبارة لكي تتمكن من التنقل الفعّال خلال معطيات الكائن ResultSet.

تمكنك هذه الكائنات من التنقل إلى الأمام أو إلى الخلف أو حتى القفز إلى موضع صف معين ضمن معطيات ناتج العبارة.

يعتبر استخدام هذا الكائن مفيداً، وخاصة عندما تحتاج لنقل المؤشر إلى مواضع مختلفة في ناتج العبارة، ورغبت بإ نشاء أحداث مرتبطة بهذا التنقل. كما ستستخدمه لبناء شبكة معطيات تمكّن المستخدم من التنقل خلال مختلف مواضع ناتج العبارة.

هناك ميزة هامة لهذا النوع من نواتج العبارة تتمثل بإمكانية إنشاء كائنات ResultSet حساسة لـ تغيرات معطياتها على المخدم. فعندما ينتقل المؤشر خلال ناتج العبارة سيقوم الكائن ResultSet بالتحقق فيما إذا تم تغيير المعطيات على المخدم، حيث سيقوم بهذه الحالة بتحديث معطياته مباشرة بالمعطيات الجديدة على المخدم.

ستدرك أهمية هذه الميزة عندما تحاول إبقاء المستخدم على إطلاع آني للمعلومات في تطبيقات الزمن الحقيقي مثل أنظمة الحجز في شركات الطيران.

لايفضّل استخدام الكائنات (Updateable ،Scrollable) إلا عند الحاجة إليها فقط. لأن استخدامها يتطلب الكثير من المعالجة الإضافية والتي تؤثر على الأداء بشكل عام. ولكن لا يمكن تجاهل فوائد هذه الكائنات عند الحاجة إليها.

نواتج العبارة القابلة للتحديث Updateable

يمكنك استخدام هذا النوع من نواتج العبارة من تحديث عمود معطيات في الصف الحالي لمجموعة معطيات الكائن ResultSet. وبذلك ستتمكن من إجراء تغييراتك بدون إرسال استعلامات SQL بشكل مباشر إلى قاعدة البيانات.

استعادة المعطيات من ناتج العبارة

تكون المعطيات المُعادة باستخدام ناتج العبارة ممثلة بأنواع معطيات JDBC، لذلك يتوجب تحويلها إلى أنواع معطيات Java لكي تستطيع استخدامها في تطبيقك.

يقدم الكائن ResultSet مجموعة المناهج getXXX() للقيام بعمليات التحويل المطلوبة. حيث يمثل الجزء XXX نوع معطيات اللغة Java المراد استعادة المعطيات وفقه. فمثلاً إذا أردت استعادة قيمة من عمود ما بالشكل Java int، ستستخدم المنهج getInt() . وهكذا فكل نوع معطيات JDBC له نوع معطيات مقابل في Java ، يتطلب استخدام المنهج getXXX() المناسب لإجراء عملية التحويل.

على كل الأحوال، يعطيك المنهج getXXX() الحرية الكاملة لقسر نوع معطيات JDBC إلى نوع معطيات Java غير مقابل له. فمثلاً يمكنك استخدام المنهج getString() لاستعادة أي نوع معطيات SQL رقمي، مثل INTEGER أو DOUBLE أو NUMBER. كما يمكنك أيضاً التحويل من نوع معطيات رقمي ذو دقة عالية إلى نوع معطيات آخر ذو دقة أقل، لكن هذا التحويل قد يتسبب بضياع المعلومات، فمثلاً يمكنك المنهج getInt() من استعادة قيمة من نوع معطيات JDBC DOUBLE ، وأثناء عملية التحويل هذه سيتم تجاهل الخانات الواقعة على يمين الفاصلة العشرية. وهذا مشابه لإسناد قيمة من النمط double لقيمة من النمط int.

يمتلك المنهج getXXX() شكلين متاحين، لتسهيل الإشارة إلى قيم الأعمدة المراد استعادة المعطيات منها. وهذان الشكلان هما:

```
ResultSet.getXXX(int columnIndex)
ResultSet.getXXX(String columnName)
```

يمكنك المنهج الأول من استعادة معطيات ناتج العبارة على أساس موضع ترتيب العمود. حيث يبدأ تعداد الأعمدة بالرقم واحد. أما المنهج الثاني فيستخدم اسم العمود لاستعادة المعطيات كما هو موضح في مقطع الشيفرة التالي:

```
//Assume a valid Connection, conn.
Statement stmt = conn.createStatement();
//Create a ResultSet object
String SQL = "SELECT Name FROM Employees";
ResultSet rset = stmt.executeQuery(SQL);
//Retrieve by ordinal column position
String byColumnNumber = rset.getString(1);
//Retrieve by column name
String byColumnName = rset.getString("name");
```

ينصح بإتباع هذه الطريقة دائماً لأن تحديد العمود في ناتج العبارة بالاعتماد على اسمه أسهل من تحديده باستخدام موضع ترتيبه.

يمتلك الكائن ResultSet أيضاً مجموعة مناهج ()getXXX، تستخدم للوصول إلى أنواع معطيات SQL3 التالية:
(CLOB ،BLOB ،ARRAY ،STRUCT ،REF ،DISTINCT).
حيث تمكنك هذه الطريقة من الوصول إلى المعطيات باستخدام ما يسمى SQL LOCATOR
وهو عبارة عن مؤشر منطقي يعمل على الزبون ويشير إلى المعطيات على المخدم.
وبنتيجة ذلك لن تحتاج لجلب أنواع المعطيات الكبيرة هذه إلى الزبون باستخدام المناهج ()getXXX ، بل ستقوم بأداء هذه المهمة لتتمكن من عرض تلك المعطيات على الزبون.

قد تحتاج لاستخدام مجرى دخل لجلب الأعمدة التي تحتوي معطيات ثنائية أو محرفية ذات حجم كبير، مثل أنواع المعطيات CLOB و BLOB. حيث يعيد المنهجان ()getBinaryStream و ()getAsciiStream كائنات من النمط InputStream.
لذلك يمكنك التحكم في المعطيات المُعَادة لتمنع القيم الكبيرة من استهلاك حيز كبير من الذاكرة.
وإليك هذة الشفرة التي تستخدم مجرى الدخل لاستعادة معلومات عمود معطيات:

```
//Assume a valid Statement object
Sting SQL = "SELECT Data FROM DataTable";
ResultSet rset = stmt.executeQuery (SQL);

//Loop through the result set
while (rset.next()){
        //Use an input stream to store the data
        InputStream is = rset.getBinaryStream (1);
        //Collect results from InputStream into a
        //ByteArrayOutputStream object
        int i;
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        while((i = is.read ()) != -1){
```

```
                bos.write(i);
                        }
                }
```

هنالك منهج آخر هو المنهج getObject() الذي يمكنك من استعادة أي نوع معطيات. حيث يعيد هذا المنهج كائناً ما، فإذا أردت نمطاً أكثر تحديداً فإنك ستقوم بقسره.

استخدام نواتج العبارة القياسية Standard

يمكنك الكائن ResultSet القياسي من استعادة واستعراض المعطيات من ناتج العبارة، حيث ستحصل على ناتج عبارة من هذا النوع بشكل افتراضي بمجرد عدم تمرير أي بارمتر إلى منهج إنشاء العبارة التابع للكائن Connection ، وقد يكون هذا المنهج createStatement( ) أو prepareStatement() أو prepareCall( ).و كما ذكرنا سابقاً فإن هذا النوع أحادي الاتجاه وغير قابل للتحديث.

كيف يقوم ناتج العبارة بتهيئة معطياته؟

يتم جلب معطيات ناتج العبارة بشكل تزايدي من أجل عبارات SELECT القياسية وغير الحاوية على توابع فرز أو تجميع وهذا يعني بأنه في كل مرة ينتقل فيها مؤشر ناتج العبارة فإن المعطيات ستعاد من المخدم وتوضع في حيز التطبيق الزبون. فإذا قام شخص آخر بتعديل معطيات تنتمي لناتج العبارة على مخدم قاعدة البيانات في اللحظة التي تسبق عرض هذه المعطيات أو في اللحظة التي كان المؤشر واقفاً في موضع مختلف عن موضع المعطيات الـ محدّثة، فإنك ستتلقى هذه التغييرات عندما تقوم بنقل المؤشر إلى موضع هذه المعطيات من خلال الكائن ResultSet. أما عند طلب استعلامات فرز أو تجميع فسيتم تلقي ناتج العبارة بشكل مباشر من قبل التطبيق الزبون.

بشكل عام تقدم الكائنات ResultSet القياسية أسرع طريقة للو صول إلى معطيات ناتج العبارة في حين تتطلب الأنواع الأخرى (Updateable ،Scrollable) معالجات إضافية للحصول على معطيات ناتج العبارة الأمر الذي سيقلل من الأداء.

إنشاء الكائن ResultSet القياسي

لا يمكنك إنشاء الكائن ResultSet بشكل مباشر لأن تقنية JDBC تعرّف واجهة وليس صنف ResultSet.

يعيد كل من Statement وCallableStatement وPreparedStatement كائناً من النمط ResultSet ، وذلك عندما تقوم بتنفيذ المنهج execute() أو المنهج executeUpdate() بنجاح، وكما ذكرنا في الفقرات السابقة فإن المنهج executeUpdate() يعيد قيمة عدَّاد التحديث وليس ناتج عبارة.

يمثل مقطع الشيفرة التالي كيفية إنشاء الكائن ResultSet باستخدام كل من الكائنين Statement وPreparedStatement .

```
//Assume a valid Connection, conn.
Statement stmt = conn.createStatement();

//Create ResultSet object with Statement
String sql1 = "SELECT Name, Salary FROM Employees";
ResultSet rset1 = stmt.executeQuery(sql1);

//Create ResultSet object with PreparedStatement
String sql2 = "SELECT Name, Salary FROM Employees WHERE Ssn = ?";
PreparedStatement pstmt= conn.prepareStatement(sql2);
pstmt.setInt(1,876349372);
ResultSet rset2 = pstmt.executeQuery();
```

وهكذا سيصنع المنهجان stmt.executeQurey() و pstmt.executeQuery() الكائنين res1 و res2 ، حيث يحتوي ناتج العبارة res1 على معلومات الاسم (Name) والراتب (Salary) من أجل كل الموظفين في الجدول Employees أما ناتج العبارة من أجلres2 فسيحتوي على نفس المعلومات ولكن من أجل الموظف الأول فقط. حيث استخدمت استعلام بارمتري ليقتصر عدد الصفوف المعادة إلى صف واحد فقط من أجل هذا المثال البسيط.

التنقل ضمن معطيات الكائن ResultSet القياسي

لن تتمكن من الحصول على كامل معطيات ناتج العبارة إذا لم تقم بالتنقل ضمن صفوف معطياته واستعادتها. فمن أجل ناتج العبارة القياسي تستطيع استخدام المنهج resultSet.next() لنقل المؤشر خلال صفوف معطيات ناتج العبارة. حيث يقوم هذا المنهج بنقل المؤشر من موضعه الحالي إلى الموضع التالي الحاوي على معطيات.

يجب التأكد عند استخدام هذا المنهج من أن المؤشر لن يصل إلى الموضع ALR (نهاية الصفوف). ولكن لحسن الحظ فإن هذا المنهج يقوم بالتحقق من ذلك تلقائياً ويعيد القيمة true في حال تم الانتقال إلى صف مقبول أو يعيد القيمة false في حال تم نقل المؤشر إلى الموضع ALR ، وبهذا السلوك ستتمكن من معالجة معطيات ناتج العبارة باستخدام الحلقة While مستخدماً المنهج resultSet.next() كشرط لاستمرار هذه الحلقة كما هو موضح في الشيفرة التالية:

```
//Assume a valid Statement object stmt
ResultSet rs = stmt.executeQuery("SELECT * from Employees);
while(rs.next()){
                    //process rs data
                  }
```

قد تحتاج أثناء التنقل خلال صفوف معطيات ناتج العبارة لمعرفة موضع المؤشر الحالي، مثلاً للتأكد من أن المؤشر يشير إلى الموضع ALR قبل استدعاء أحد المناهج getXXX() لاستعادة المعطيات. في هذه الحالة يمكنك استخدام مجموعة من المناهج والتي يقدمها الكائن ResultSet لتحديد موضع المؤشر.

يستعرض الجدول -18 هذه المناهج مع شرح بسيط لكل منها:

جدول -18

المنهج	الوصف
isBeforFirst ( )	يعيد هذا المنهج القيمة true إذا كان المؤشر في الموضع BFR.
isFirst ( )	يعيد هذا المنهج القيمة true إذا كان المؤشر في موضع الصف الأول.
isAfterLast ( )	يعيد هذا المنهج القيمة true إذا كان المؤشر في الموضع ALR.
isLast ( )	يعيد هذا المنهج القيمة true إذا كان المؤشر في موضع الصف الأخير.
getRow ( )	يعيد هذا المنهج قيمة من النمط int تحدد ترتيب الصف، حيث تكون هذه القيمة واحد بالنسبة للصف الأول، واثنان للصف الثاني. وهكذا.. ويعيد القيمة صفر في حال عدم وجود أي صف.

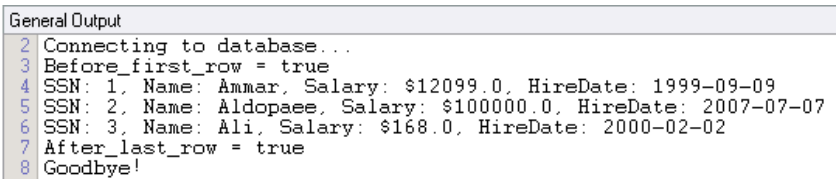
من الطبيعي أنك سترغب باستخدام المنهج getRow() أكثر من أي منهج لأن هذا المنهج سيعيد رقم الصف الحالي والذي يقف عنده مؤشر ناتج العبارة أو سيعيد القيمة صفر في حال كان المؤشر في احد الموضعين ALR أو BFR.

تمكنك مجموعة المناهج الأخرى من التحقق فيما إذا كان المؤشر موجود داخل ناتج العبارة أيضاً، حيث يعيد المنهجان isBeforFirst() و isAfterLast() القيمة true في حال انتقل المؤشر إلى أحد الموضعين BFR أو ALR، كما يمكنك معرفة فيما إذا كان المؤشر يتوضع في الصف الأول أو الأخير باستخدام المنهجين isFirst()و isLast ( ) حيث تعيد هذه المناهج القيمة true في حال وجود المؤشر في الموضع الملائم لكل منهج.

سنستعرض مثالاً يضم كل المفاهيم التي تمت مناقشتها في هذه الفقرة، حيث سنقوم في هذا المثال بتأسيس اتصال مع قاعدة البيانات، ومن ثم إنشاء الكائن Statement وإرسال استعلام لاستعادة قيم الأعمدة التالية (Name, SSN, Salary) من الجدول Employees، بعد ذلك سنستخدم العديد من مناهج تحديد موضع المؤشر.
سنستدعي في البداية المنهج ( ) ResultSet.isBeforFirst والذي سيعيد القيمة true لأننا لم نقم بنقل موضع المؤشر بعد، لذلك سيبقى في الموضع BFR بعدها سنتنقل خلال ناتج العبارة باستخدام المنهج ResultSet.next() ونقوم بطباعة القيم إلى الخرج. حيث ستلاحظ بأننا استخدمنا المنهج getRow() للحصول على رقم الصف الذي يقف عنده المؤشر وقبل إنهاء التطبيق سنقوم باستدعاء المنهج Result.isAfterLast() لتوضيح آلية عمله، حيث سيعيد هذا المنهج القيمة true وذلك لأن الاستدعاء الأخير للمنهج resultSet.next() قد نقل المؤشر إلى الموضع ALR.

```
1. // EXP Class Result
2. import java.sql.*;
3. public class Chp18_2 {
4.         public static void main(String[] args) {
5.
6.                 //Declare Connection, Statement, and ResultSet variables
7.                 Connection conn = null;
8.                 Statement stmt = null;
9.                 ResultSet rs = null;
10.
11.                //Begin standard error handling
12.                try{
13.                        //Register driver
14.                        String driver = "oracle.jdbc.driver.OracleDriver";
15.                        Class.forName(driver).newInstance();
16.
17.                        //Open database connection
18.                        System.out.println("Connecting to database...");
19.                        String jdbcUrl = "jdbc:oracle:thin:@localhost:1521:ORCL";
```

```
20.                     conn                                                    =
DriverManager.getConnection(jdbcUrl,"toddt","mypwd");
21.
22.
23.
24.                     //Create a Statement object and execute SQL query
25.                     stmt = conn.createStatement();
26.                     String sql = "SELECT SSN, NAME, SALARY, HIREDATE
FROM Employee";
27.                     rs = stmt.executeQuery(sql);
28.
29.                     //Variables to hold information
30.                     int ssn;
31.                     String name;
32.                     double salary;
33.             System.out.println("Before_first_row" + "= "+ rs.isBeforeFirst());
34.
35.                     while(rs.next()){
36.                             //Retrieve by column name
37.                             ssn= rs.getInt("SSN");
38.                             name = rs.getString("Name");
39.                             //Retrieve by column index
40.                             salary = rs.getDouble(3);
41.                             //Display values
42.                     System.out.print("Row Number=" + rs.getRow());
43.                             System.out.print(", SSN: " + ssn);
44.                             System.out.print(", Name: " + name);
45.                     System.out.println(", Salary: $" + salary);
46.                                                     }
47.             System.out.println("After_last_row"+" = " +rs.isAfterLast());
48.                     //Standard error handling
49.                     } catch(SQLException se) {
50.
51.                             //Handle errors for JDBC
52.                             se.printStackTrace();
53.                     } catch(Exception e) {
54.
55.                             //Handle errors for Class.forName
56.                             e.printStackTrace();
57.                     } finally {
58.                             try {
59.                                     if(conn!=null)
60.                                     conn.close();
61.                                     } catch(SQLException se) {
62.                                             se.printStackTrace();
63.                                     }//end finally try
64.                                       }//end try
65.                     System.out.println("Goodbye!");
66.                     }//end main
67. }//end Chp18_2 class
```



استخدام نواتج العبارة القابلة للتمرير Scrollable

تسمح نواتج العبارة القياسية بالانتقال وفق الاتجاه الأمامي فقط، قد يكون ذلك مفيداً في أغلب الأحيان ولكنك قد تحتاج في حالات معينة للانتقال خلال ناتج العبارة بأي اتجاه تشاء. ستجد أن نواتج العبارة القابلة للتمرير تتيح لك الانتقال إلى الأمام وإلى الخلف وإلى أي صف تحدده في ناتج العبارة. تعتبر هذه الميزة ميزة تسهيلية رائعة وخاصة عندما تطور تطبيقاً يسمح للمستخدم بالتخاطب مع معطيات ناتج العبارة. فمثلاً يحتاج مستخدم تطبيق إدارة مخزن للانتقال إلى ا لأمام وإلى الخلف خلال معطيات ناتج العبارة للتحقق من معطيات المخزن الحالية، وهذا ما تقدمه لهم نواتج العبارة القابلة للتمرير. كما تستطيع نواتج العبارة القابلة للتمرير من التعرف على التغيرات الحاصلة للمعطيات المقابلة في قاعدة البيانات في حال تم إعدادها لذلك لأن نواتج العبارة القابلة للتمرير الحساسة لتغيرات المعطيات في قاعدة البيانات ستطلعك بشكل مستمر على المعطيات المحدثة. سيجد نظام إدارة المخزن هذه الميزة مفيدة للغاية لأنه سيبقى على إطلاع مستمر بتغيرات معطيات المواد الموجودة في مخزنه، ولكن نواتج العبارة الحساسة لتغير المعطيات قد تسبب بعض الإزعاج في حال كان المطلوب هو الحصول على صورة آنية للمعطيات في القاعدة فقط.

سنقوم في الفقرة التالية باستعراض كيفية إنشاء كائنات نواتج العبارة القابلة للتمرير واستخدامها مع تقديم مثال توضيحي لذلك.

إنشاء نواتج العبارة القابلة للتمرير

كما ذكرنا سابقاً فإنه يجب عليك بتحديد نوع ناتج العبارة الذي سيعيده استعلامك أثناء إنشاء كائنات العبارة (Statement, PreparedStatement, CallableStatement).

تحتاج لتمرير بارمتر مُعرَّف مسبقاً إلى أحد مناهج الاتصال

(createStatement( ), prepareStatement( ),prepareCall())

لتتمكن من إنشاء ناتج عبارة من النوع القابل للتمرير، توضح الشيفرة التالية كيفية استخدام هذه المناهج:

```
CreateStatement (int resultSetType, int resultSetConcurrency);
PrepareStatement (String SQL, int resultSetType, int resultSetConcurrency)
PrepareCall (String sql, int resultSetType, int resultSetConcurrency)
```

يحدد البارمتر الأول نوع الانتقال المراد الحصول عليه، والحساسية تجاه تغير المعطيات للكائن ResultSet الذي سيصنعه المنهج.

يوضح الجدول -18 القيم المتاحة للبارمتر resultSetType والتي تتيح إمكانية إنشاء كائنات ResultSet قابلة للتمرير.

جدول -18

الثابت	التعليق
TYPE_SCROLL_INSENSITIVE	بتمرير هذه القيمة سينتج كائن ResultSet يستطيع مؤشره التنقل إلى الأمام وإلى الخلف وإلى أي صف آخر محدد. لكنه لا ينقل تغيرات معطيات قاعدة البيانات أثناء تنقل مؤشره خلال مجموعة المعطيات.*
TYPE_SCROLL_SENSITIVE	بتمرير هذه القيمة سينتج كائن ResultSet يستطيع مؤشره التنقل إلى الأمام وإلى الخلف وإلى أي صف آخر محدد. كما أنه يمثل منظاراً ديناميكياً للمعطيات على المخدم.
TPYE_FORWARD_ONLY	بتمرير هذه القيمة سينتج كائن ResultSet افتراضي. لا يمكنه نقل تغييرات معطيات قاعدة البيانات أثناء تنقل مؤشره خلال مجموعة المعطيات.*

يعتمد ذلك على نوع المعطيات المشار إليه، حيث تستخدم بعض أنواعٍ SQL3 مؤشرات منطقية (تدعى LOCATORs) إلى المعطيات في مخدم قاعدة البيانات، وباستخدام أنواع المعطيات هذه يمكنك مشاهدة التغييرات لأنه لا يتم هنا تجسيم المعطيات في الزبون.

إذاً تستطيع إنشاء نواتج عبارة قابلة للتمرير من خلال تمرير القيمتين TYPE_SCROLL_INSENSITIVE و TYPE_SCROLL_SENSITIVE للبارمتر resultSetType ، حيث سيصنع البارمتر TYPE_SCROLL_INSENSITIVE منظاراً ديناميكياً للمعطيات المقابلة على المخدم، وبذلك تستطيع تلقي تغييرات هذه المعطيات مباشرة أثناء التنقل بالمؤشر خلال معطيات ناتج العبارة في تطبيقك.

يصنع البارمتر TPYE_FORWARD_ONLY ناتج العبارة الافتراضي ذو اتجاه التنقل الأمامي فقط، لذلك يقتصر استخدا م هذه القيمة في حالات التحكم بإعدادات المطابقة الخاصة بالتنقل الأمامي فقط.

إن استخدام ناتج العبارة القابل للتمرير سيبطئ من أداء التطبيق بشكل عام لأن ناتج العبارة هذا سيقوم بعمل إضافي لكي يمكنك من التنقل خلال معطيات ناتج العبارة، وفي حال الرغبة في جعل ناتـ ج العبارة حساساً للتغيرات فإن ذلك سيزيد من العبء أيضاً لأنه سيقوم بتفحص قاعدة البيانات كلما تنقل المؤشر في مجموعة المعطيات. لذلك لا يفضل استخدام هذه الأنواع الا عند الحاجة إليها فقط.

التنقل ضمن معطيات كائن ناتج العبارة القابل للتمرير

يمتلك الكائن ResultSet القابل للتمرير مجموعة من المناهج والتي تمنحك القدرة على نقل المؤشر إلى الأمام أو إلى الخلف أو إلى أي صف تقوم بتحديده. تعمل مجموعة مناهج التنقل هذه بغض النظر فيما إذا كان ناتج العبارة حساساً لتغيرات المعطيات المقابلة أو لا.

يقدم الجدول -18 لمحة عن المناهج المستخدمة للتحكم بالمؤشر، حيث ستلاحظ بأن كل المناهج ستعيد القيمة true في حال أدت مهمتها بنجاح.

جدول -18

المنهج	الوصف
next ( )	ينقل المؤشر إلى الصف التالي.
previous ( )	ينقل المؤشر إلى الصف السابق.
beforFirst ( )	ينقل المؤشر إلى الموضع BFR (ما قبل الصف الأول)، بحيث أن استدعاء أحد المناهج getXXX( ) مباشرة بعد هذا المنهج سيولد

	الاستثناء SQLException.
afterLast ( )	ينقل المؤشر إلى الموضع ALR (ما بعد الصف الأخير)، بحيث سيولد استدعاء أحد المناهج ( )getXXX بعد هذا المنهج مباشرة الاستثناء SQLException.
first ( )	ينقل المؤشر إلى الصف الأول في مجموعة المعطيات.
last ( )	ينقل المؤشر إلى الصف الأخير في مجموعة المعطيات.
absolute ( )	ينقل المؤشر إلى الصف المحدد ابتداءً من صف مجموعة المعطيات الأول.
relative ( )	ينقل المؤشر إلى الصف المحدد ابتداءً من الصف الحالي.
moveToCurrentRow()*	ينقل المؤشر إلى الصف الذي كان واقفاً عنده قبل انتقاله إلى صف الإضافة insert row.
moveToInsertRow( )*	ينقل المؤشر إلى صف الإضافة insert row.

يستخدم هذا المنهجان من أجل كائنات ResultSet القابلة للتحديث (Updateable) ، سنتحدث عن كائنات ResultSet القابلة لتحديث في الفقرات القادمة.

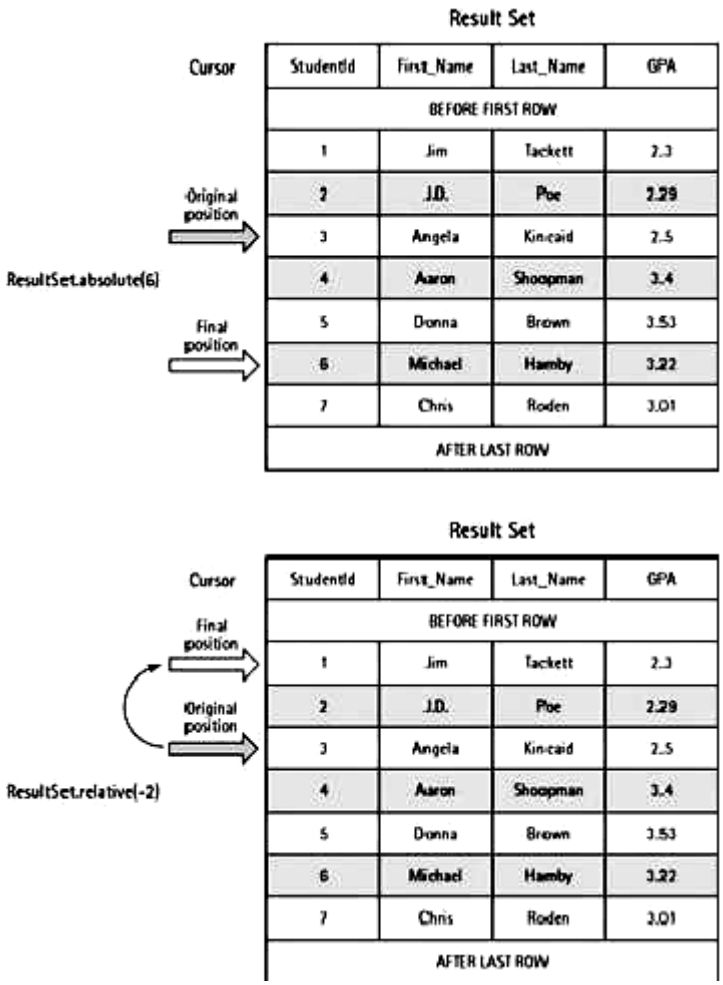
يقوم المنهجان ( ) next و ( )previous بنقل المؤشر إلى الصف الأمامي التالي أو إلى الصف الخلفي السابق. فإذا كان المؤشر في الصف الأول وقمت باستدعاء المنهج ( )previous فسيتم نقل المؤشر إلى الموضع BFR ، وبشكل مماثل، فإذا كان المؤشر في الصف الأخير وقمت باستدعاء المنهج ( )ResultSet.next فسيتم نقل المؤشر إلى الموضع ALR. على كل حال يمكنك استخدام المنهجين التاليين beforFirst( ) و ( )afterLast لنقل المؤشر إلى الموضعين BFR و ALR ، حيث يعتبر هذان الموضعان كنقطة بداية تمكنك من البدء بإجراء تنقلات أخرى، فمثلاً ستحتاج لنقل المؤشر إلى بداية أو نهاية مجموعة المعطيات قبل البدء باستعادتها باستخدام حلقة While.

يقوم المنهجان first( ) و last( ) التابعان للكائن Result بنقل المؤشر إلى الصف الأول والأخير من ناتج العبارة. حيث ستحصل على نفس النتيجة باستدعائها من أجل ناتج عبارة يحتوي على صف واحد فقط. كما يمكنك استدعاء هذين المنهجين من الموضعين BFR و ALR لنقل المؤشر إلى موضع ابتدائي.

يستخدم المنهجان

ResultSet.absolute(int n) ,ResultSet.relative(int n)

لنقل المؤشر إلى صف معين في ناتج العبارة. حيث ينقل المنهج absolute ( ) المؤشر عدداً من الصفوف بالمقدار n ابتداءً من الصف الأول، لذلك ستحصل على نتيجة متماثلة باستدعاء المنهجين ResultSet.first() و ResultSet.absolute(1) . بينما ينقل المنهج relative() المؤشر إلى الأمام أو إلى الخلف عدداً من الصفوف بالمقدار n ابتداءً من الصف الحالي (تنقل قيم n الموجبة المؤشر إلى الأمام في حين تنقل قيمه السالبة المؤشر إلى الخلف). يوضح الشكل -18 آلية عمل هذين المنهجين.



شكل -18

يقوم المثال التالي باختبار كل الموضوعات التي تمت مناقشتها في هذه الفقرة. سأقوم في هذا المثال بإنشاء الكائن ResultSet القابل للتمرير، واستخدام مناهج التنقل للوصول إلى مختلف المواضع في ناتج العبارة.

```
1. //
2. package Article3;
3. import java.sql.*;
4. public class Chp18_3 {
5.         public static void main(String[] args) {
6.                   //Declare Connection, Statement, and ResultSet variables
```

```
7.              Connection conn = null;
8.              Statement stmt = null;
9.              ResultSet rs = null;
10.
11.             //Begin standard error handling
12.             try{
13.                     //Register driver
14.                     String driver = "oracle.jdbc.driver.OracleDriver";
15.                     Class.forName(driver).newInstance();
16.
17.                     //Open database connection
18.                     System.out.println("Connecting to database...");
19.
20.                     String jdbcUrl =
"jdbc:oracle:thin:@localhost:1521:ORCL";
21.                     conn =
DriverManager.getConnection(jdbcUrl,"toddt","mypwd");
22.
23.                     //createStatement() method that specifies I want a
24.                     //scrollable result set that is insensitive to changes on
25.                     //the database. The result set is also READ_ONLY so
the I
26.                     //cannot use it to make changes.
27.
        stmt=conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
28.                     ResultSet.CONCUR_READ_ONLY);
29.
30.                     String sql = "SELECT ssn, name, salary FROM
EMPLOYEE";
31.                     rs = stmt.executeQuery(sql);
32.
33.                     System.out.println("List result set for reference....");
34.                     while(rs.next())
35.                             printRow(rs);
36.                     System.out.println();
37.
38.                     //Demonstrate afterLast() and beforeFirst()
39.                     System.out.println("Move to After_last_row"  +"position
with afterLast()");
40.
41.                     rs.afterLast();
42.
43.                     System.out.println("After_last_row = " +
rs.isAfterLast());
44.                     System.out.println();
45.
```

```
46.                     System.out.println("Move to Before?first?row" +
"position with beforeFirst()");
47.                     rs.beforeFirst();
48.
49.                     System.out.println("Before?first?row = "+
rs.isBeforeFirst());
50.                     System.out.println();
51.
52.                     //Demonstrate first() and last() methods.
53.                     System.out.println("Move to first row with first().");
54.                     System.out.println("The row is:");
55.                     rs.first();
56.                     printRow(rs);
57.                     System.out.println();
58.
59.                     System.out.println("Move last row with last().");
60.
61.                     System.out.println("The row is:");
62.                     rs.last();
63.                     printRow(rs);
64.                     System.out.println();
65.
66.                     //Demonstrate previous() and next() methods.
67.                     System.out.println("Move to previous row with
previous().");
68.                     System.out.println("The row is:");
69.                     rs.previous();
70.
71.                     printRow(rs);
72.                     System.out.println();
73.                     System.out.println("Moving to next row with next().");
74.                     System.out.println("The row is:");
75.                     rs.next();
76.                     printRow(rs);
77.                     System.out.println();
78.
79.                     //Demonstrate absolute() and relative()
80.                     System.out.println("Move to the 3rd row with
absolute(3).");
81.                     System.out.println("The row is:");
82.                     rs.absolute(3);
83.                     printRow(rs);
84.                     System.out.println();
85.
86.                     System.out.println("Move back 2 rows with
relative(?2).");
```

```
87.                     System.out.println("The row is:");
88.                     rs.relative(-2);
89.                     printRow(rs);
90.                     System.out.println();
91.                     //Standard error handling
92.             } catch(SQLException se) {
93.                     //Handle errors for JDBC
94.                     se.printStackTrace();
95.             } catch(Exception e) {
96.                     //Handle errors for Class.forName
97.                     e.printStackTrace();
98.             } finally {
99.                     try {
100.                            if(conn!=null)
101.                                    conn.close();
102.                    } catch(SQLException se) {
103.                                                    se.printStackTrace();
104.                    } //end finally try
105.                      } //end try
106.            System.out.println("Goodbye!");
107.            }//end main
108.
109.    public static void printRow(ResultSet rs) throws SQLException{
110.            //Field variables
111.            int ssn;
112.            String name;
113.            double salary;
114.            //Retrieve by column name
115.            ssn= rs.getInt("ssn");
116.            name = rs.getString("name");
117.            salary = rs.getDouble("salary");
118.            //Display values
119.            System.out.print("Row Number=" + rs.getRow());
120.            System.out.print(", SSN: " + ssn);
121.            System.out.print(", Name: " + name);
122.            System.out.println(", Salary: $" + salary);
123.            }//end printRow()
124. }//end Chp18_3 class
```

استعادة عدد الصفوف في ناتج العبارة

لا يقدم JDBC طريقة مباشرة لاستعادة عدد الصفوف المعادة بواسطة استعلام SQL، كما لا تعرف الواجهة ResultSet خاصية أو منهج صريح لاستعادة هذه القيمة. لذلك يجب علينا استخدام وسائل مساعدة لاستعادة هذه المعلومات، وتبعاً لذلك سأستعرض تقنيتين مساعدتين للحصول على قيمة الصفوف المعادة وهما:

1- استخدام متحول كعدّاد : تستخدم هذه التقنية عدّاداً لحساب عدد الصفوف في الجدول، وأثناء التنقل خلال ناتج العبارة يتم زيادة العداد في كل مرة بحيث تصبح قيمة المتحول العدّاد مساوية لعدد الصفوف في ناتج العبارة بعد انتهاء حلقة التنقل.

```
//Assume a valid ResultSet object rs
int count;
while(rs.next(){
count++;
}
```

2- استخدام المنهج ResultSet.getRow : ستفضل استخدام هذه الطريقة في حال كنت تتعامل مع كائن ResultSet قابلاً للتمرير. لا تستخدم هذه الطريقة حلقة للتنقل وإنما تستخدم المنهج ResultSet.last() بدلاً من ذلك، لأن الانتقال إلى الصف الأخير من ناتج العبارة سيهيئ عدّاد الصفوف الداخلي للكائن ResultSet. توضح الشيفرة التالية آلية العمل.

```
//Assume a valid ResultSet object rs
rs.last();
int count = rs.getRow();
```

استخدام نواتج العبارة القابلة للتحديث Updateable

تمنحك كائنات ResultSet القابلة للتحديث الحرية في معالجة معطيات ناتج العبارة بشكل مباشر، بحيث تغنيك عن تنفيذ استعلامات SQL إضافية لإجراء التحديث على قاعدة بياناتك. تمكنك هذه الكائنات من إجراء التغييرات المختلفة مثل تحديث عمود معطيات أو إضافة صفوف جديدة أو حذف صفوف. ولكن عند تحديث الأعمدة فإن التغييرات الجديدة لا تطبق على مخدم قاعدة البيانات مباشرة، وإنما يقوم المنهج ResultSet.updateXXX() بتغيير معطيات ناتج العبارة فقط، وليس المعطيات

على مخدم قاعدة البيانات، وباستدعاء المنهج ResultSet.updateRow() يتم حفظ التغييرات المحدثة إلى كل من ناتج العبارة ومخدم قاعدة البيانات.

يتطلب استخدام هذه الكائنات معالجة إضافية (مثل كائنات ResultSet) لذلك يفضل عدم استخدامها إلا عند الحاجة إذا أنه سينتج عن ميزة هذه الكائنات (إجراء التحديث المباشرة للمعطيات) بطء في استعراض معطيات ناتج العبارة.

بشكل عام، ستجد نفسك في بعض الأحيان غير قادر على تحديث ناتج العبارة حتى ولو قمت بتمرير البارمترات الصحيحة له عند إنشائه، فبعض المشغلات تلقي تحذير SQL يدل على عدم إمكانية إنشاء ناتج العبارة المطلوب، وتقوم بإنشاء ناتج عبارة لكنه لا يتمتع بالوظائفية المطلوبة. يعود السبب في ذلك إلى أن استعلام SQL المستخدم يجب أن يتوافق مع معايير محددة، لكي يتم إنشاء ناتج عبارة قابل للتحديث (Updateable). بشكل عام يتطلب استعلام SQL أن يتوافق مع القواعد التالية لكي تتمكن من إنشاء ناتج عبارة قابل للتحديث:

- أن لا يستخدم استعلام SQL عبارات تجميع أو فرز مثل GROUP BY أو ORDER BY.

- لا تستطيع استخدام نواتج العبارة القابلة للتحديث من أجل تحديث معطيات ناتج عبارة مصنوع باستخدام استعلام SQL يعيد معطيات من عدة جداول، لأن التغيير في ناتج العبارة هذا سيؤثر على عدة جداول وهذه ميزة غير متاحة في إصدار JDBC الحالي.

- في حال أردت إضافة صفوف جديدة إلى جدول ما فيجب أن تعيد عبارة SQL المستخدمة المفتاح الرئيسي للجدول وإلا سيحدث خطأ أثناء محاولتك لإضافة صف بدون تحديد قيم المفتاح الرئيسي.

على أي حال فإن بعض أنظمة إدارة قواعد البيانات (DBMS) تقوم تلقائياً بتوليد المفتاح الرئيسي. لذلك عليك مراجعة الوثائق الخاصة بالمشغل للتحقق من ذلك.

إنشاء نواتج العبارة القابلة للتحديث

يتم صنع كائنات ResultSet القابلة للتحديث بواسطة نفس مناهج الكائن Connection المستخدمة لإنشاء كائنات ResultSet القابلة للتمرير. تمثل المناهج الثلاثة التالية مناهج الكائن Connection المستخدمة لإنشاء كائنات ResultSet القابلة للتمرير:

```
createStatement(int resultSetType, int resultSetConcurrency);

prepareStatement(String sql,int resultSetType,int resultSetConcurrency);

prepareCall(String sql, int resultSetType, intresultSetConcurrency);
```

ستستخدم البارمتر resultSetType لتحديد نوع ناتج العبارة القابل للتمرير، ستلاحظ أنه ممن الممكن إنشاء ناتج عبارة ذو اتجاه أمامي (forword-only) وقابل للتحديث (Updatable) في آن واحد. كما ستلاحظ بأنه عليك تمرير البارمتر حتى ولو كنت ترغب في إنشاء ناتج عبارة افتراضي.

أما البارمتر الثاني (resultSetConcurrency) فيحدد مستوى التنافسية (Concurrency) الذي ترغب في أن يمتلكه الكائن ResultSet. يأخذ هنا البارمتر أحد القيمتين التاليتين:

- CONCUR_UPDATEABLE: والذي ينشأ ناتج عبارة قابل للتمرير.

- CONCUR_READ_ONLY: والذي ينشأ ناتج عبارة قابل للقراءة فقط، وهو ناتج العبارة الافتراضي.

يمثل مقطع الشيفرة التالي كيفية استخدام المنهج createStatement() لإنشاء كائن ناتج عبارة ذو اتجاه أمامي وقابل للتحديث:

```
//Assume a valid connection object.

Statement stmt = conn.createStatement(
        ResultSet.TYPE_FORWARD_ONLY،
        ResultSet.CONCUR_UPDATABLE);
```

يشبه الأثر الناتج عن كائنات ناتج العبارة القابلة للتحديث مثيله في نواتج العبارة القابلة للتمرير لأنها قد تقلل من الأداء أيضاً، حيث يتطلب هذا الكائن معالجة معلومات إضافية بالإضافة إلى إجراء استدعاءات أكثر عبر الشبكة الأمر الذي سيبطئ الاستجابة، ولكن الحسنات البرمجية لاستخدام هذا الكائن قد تفوق مساوئ زيادة الأداء هذه بشكل كبير.

ما المقصود بالتنافسية Concurrency؟

تعرّف التنافسية بالمقدرة على تحديث ومشاركة المعلومات في قاعدة البيانات مع بقية المستخدمين وفي الوقت نفسه. ستواجه هذه المسألة عندما تسمح للمستخدمين بتحديث قيم معطيات قاعدة البيانات. فعندما يريد مستخدم ما إجراء تحديث على المعطيات ستقوم قاعدة البيانات بقفل (lock) المعطيات لتمنع وصول الآخرين إليها وإجراء التعديل على نفس المعطيات ويستمر هذا القفل ريثما ينتهي المستخدم من إجراء تحديثه. يختلف مستوى القفل من قاعدة بيانات إلى أخرى، فبعض الأنظمة تقوم بقفل صف المعطيات المحدث بينما تقوم أنظمة أخرى بقفل كامل الجدول.

من الم مكن أن تتسبب التنافسية بمشاكل كبيرة في الأنظمة التي يعمل عليها أعداد كبير من المستخدمين. لذلك تم إيجاد نوعين للتنافسية: النوع المتشدد Pessimistic والنوع اليسير Optimistic.

يفترض النوع المتشدد بأن هنالك الكثير من الفعالية لذلك يقوم بقفل المعطيات الخاضعة للـ تحديث، الأمر الذي يمنع بقية المستخدمين من تحديث هذه المعطيات في نفس الوقت. أما النوع اليسير فيفترض بأن هنالك القليل من الفعالية، لذلك لا يقوم بقفل المعطيات، وفي حال وجود عمليتي تحديث للمعطيات تتمان في نفس الوقت فإنه سيتم إجراء مفاضلة بين هذه المناقلات (Transactions) قبل أن تتم عملية الحفظ النهائي، وفي أغلب الأحيان فإن المناقلة التي تتم في النهاية هي التي ستحدث التغيير.

تحديث المعطيات باستخدام ناتج العبارة القابل للتحديث

يتيح لك المنهج ResultSet.updateXXX() إمكانية تغيير المعلومات في قاعدة البيانات برمجياً. وبذلك ستتجنب تنفيذ استعلامات SQL إضافية باستخدامك لنواتج العبارة القابلة للتحديث.

يقوم استدعاء المنهج UpdateXXX() بتطبيق التغييرات إلى عمود معين في الصف الحالي لمعطيات ناتج العبارة حيث يتطلب استخدام هذا ا لمنهج بارمترين، يمثل البارمتر الأول رقم ترتيب العمود الذي ترغب في تحديثه (كما يمكنك استخدام سلسلة محرفية String لتمرير قيمة هذا البارامتر) أما البارمتر الثاني فيمثل القيمة الجديدة المراد إعطاؤها للعمود. لكي تستطيع استخدام المنهج UpdateXXX() بنجاح عليك إتباع الخطوات التالية:

أولاً: يتوجب عليك نقل المؤشر إلى الصف الذي ترغب بتحديثه. وعدم القيام بذلك سيؤدي إلى تحديث معطيات مغايرة للمعطيات المراد تحديثها.

ثانياً: استدعاء المنهج updateXXX() المناسب من أجل نوع معطيات Java الذي تستخدمه. كما ذكرنا سابقاً عند استخدام المنهج setXXX() فإن الرموز XXX تدل على أنواع معطيات Java البرمجية، فمثلاً عند العمل مع كائن من النمط String سوف تستخدم updateString() ، حيث سيقوم المشغل بتحويل المعطيات إلى نوع معطيات JDBC المقابل قبل إرسالها إلى قاعدة البيانات.

ثالثاً: استدعاء المنهج updateRow() لكي يقوم بحفظ التغييرات إلى قاعدة البيانات حالاً. وأي خطأ سيحدث خلال عملية الحفظ سيؤدي إلى تجاهل التغييرات. كما سيتم تجاهل التغيرات أيضاً في حال استدعيت المنهج updateXXX() ومن ثم قمت بنقل المؤشر. لذلك عليك الانتباه إلى أنَّ المنهج updateXXX() لا يقوم بحفظ التغييرات بشكل مباشر، وإنما يتم ذلك بالاستدعاء الصريح للمنهج ResultSet.updateRow().

يمكنك التراجع عن التغييرات التي تمت من خلال الكائن ResultSet، باستخدام المنهج ResultSet.cancelRowUpdate() حيث يمكنك هذا المنهج من التراجع عن كل استدعاءات المنهج updateXXX() للتأكد من أنه تم التراجع عن كل التغييرات.

تقدم الشيفرة التالية مثالاً عن كيفية استخدام ناتج العبارة القابل للتحديث، حيث ستقوم هذه الشيفرة بالتنقل خلال معلومات جدول الموظفين وتقوم بإضافة كلفة المعيشة إلى رواتب الموظفين سيتم تطبيق التغييرات أثنا ء التنقل خلال المعطيات، ومن ثم يتم نقل هذه التغيرات إلى قاعدة البيانات باستدعاء المنهج ResultSet.updateRow().

```
1. //
2. import java.sql.*;
3. public class Chp18_4 {
4.         public static void main(String[] args) {
5.                   //Declare Connection, Statement, and ResultSet variables
6.                   Connection conn = null;
7.                   Statement stmt = null;
8.                   ResultSet rs = null;
9.                   //Begin standard error handling
10.                  try{
11.                            //Register driver
12.                            String driver = "oracle.jdbc.driver.OracleDriver";
13.                            Class.forName(driver).newInstance();
14.                            //Open database connection
15.                            System.out.println("Connecting to database...");
16.                            String jdbcUrl =
"jdbc:oracle:thin:@localhost:1521:ORCL";
17.                            conn =
DriverManager.getConnection(jdbcUrl,"toddt","mypwd");
18.                            //Create a Statement object and execute SQL query
19.                            stmt = conn.createStatement();
20.                            //createStatement() method that specifies I want a
21.                            //scrollable and updateable result set that is insensitive
22.                            // to data changes on the database server.
23.                            stmt=conn.createStatement(
ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_UPDATABLE);
24.                            String sql = "SELECT ssn, name, salary FROM
EMPLOYEES";
25.
26.                            rs = stmt.executeQuery(sql);
27.
28.                            System.out.println("List result set for reference....");
29.                            printRs(rs);
30.
31.                            //Loop through result set and give a 5.3%
32.                            //cost of living adjustment
33.                            //Move to BFR postion so while?loop works properly
34.                            rs.beforeFirst();
35.
```

```
36.                         while(rs.next()){
37.                                 double newSalary =
rs.getDouble("salary")*1.053;
38.                                 rs.updateDouble("salary",newSalary);
39.                                 rs.updateRow();
40.                                                         }
41.                         System.out.println("List result set showing new
salaries");
42.                         printRs(rs);
43.                         //Standard error handling
44.                         } catch(SQLException se) {
45.                                 //Handle errors for JDBC
46.                                 se.printStackTrace();
47.                         } catch(Exception e) {
48.                                 //Handle errors for Class.forName
49.                                 e.printStackTrace();
50.                         } finally {
51.                                 try {
52.                                         if(conn!=null)
53.                                         conn.close();
54.                                         } catch(SQLException se) {
55.                                                 se.printStackTrace();
56.
                }//end finally try
57.                                         }//end try
58.                         System.out.println("Goodbye!");
59.                         }//end main
60.
61.     public static void printRs(ResultSet rs) throws SQLException{
62.             //Field variables
63.             int ssn;
64.             String name;
65.             double salary;
66.             //Ensure we start with first row
67.             rs.beforeFirst();
68.             while(rs.next()){
69.                     //Retrieve by column name
70.                     ssn= rs.getInt("ssn");
71.                     name = rs.getString("name");
72.                     salary = rs.getDouble("salary");
73.                     //Display values
74.                     System.out.print("Row Number=" + rs.getRow());
75.                     System.out.print(", SSN: " + ssn);
76.                     System.out.print(", Name: " + name);
77.                     System.out.println(", Salary: $" + salary);
78.                                         }
```

```
79.                 System.out.println();
80.                 }//end printRs()
81. }//end Chp18_4 class
```

إضافة المعطيات وحذفها باستخدام نواتج العبارة القابلة للتحديث

يمكنك أيضاً استخدام الكائن ResultSet لإضافة أو حذف صفوف المعطيلت برمجياً باستخدام المنهجين ()insertRow و ()deleteRow.

عند القيام بإضافة صف إلى معطيات ناتج العبارة، يفترض نقل المؤشر إلى موضع ا بتدائي يسمى بموضع صف الإضافة insert row ، حيث تمثل هذه المنطقة ذاكرة مؤقتة Buffer يتم تخزين المعطيات فيها ريثما يتم نقلها إلى قاعدة البيانات، وللقيام بعملية نقل المؤشر إلى موضع الإضافة ستستدعي المنهج ResultSet.moveToInsertRow(). بعد القيام بعملية نقل المؤشر ، ستستخدم المنهج updateXXX() لتحديث معطيات العمود. بالطبع أنت تقوم هنا بإضافة معطيات وليس تحديثها حيث أن استدعاء المنهج getXXX() بعد استدعاء المنهج updateXXX() سيعيد المعلومات التي تمت إضافتها في صف الإضافة، وبالطبع لتتمكن من إرسال المعطيات المضافة إلى قاع دة البيانات وحفظها ستستدعي المنهج insertRow() ليقوم بذلك. يمثل مقطع الشيفرة التالية كيفية إضافة صف معطيات جديد إلى قاعدة البيانات باستخدام الكائن ResultSetالقابل للتحديث.

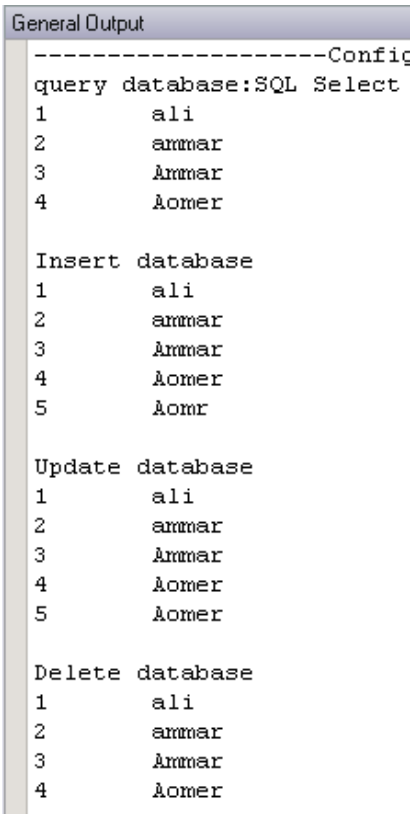
```
//Assume a valid Connection object conn
stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,

ResultSet.CONCUR_UPDATABLE);
//build SQL string
String SQL="SELECT ssn, name, salary FROM employees";
ResultSet rs = stmt.executeQuery(SQL);

//Move to insert row and add column data with updateXXX()
rs.moveToInsertRow();
rs.updateInt("SSN",5697529854);
rs.updateString("Name","Rebecca");
rs.updateDouble("Salary",45555.77);

//Commit row
rs.insertRow();
```

أما لحذف صف من ناتج العبارة ستحتاج لاستدعاء المنهج deleteRow()، حيث يقوم هذا المنهج بالتأثير على المعطيات في كل من ناتج العبارة وقاعدة البيانات بنفس الوقت، على خلاف كل مناهج معالجة المعطيات الأخرى. أما المؤشر فإنه ينتقل بعد استدعاء هذا المنهج إلى مواضع مختلفة تتعلق بتصريحات المشغل المستخدم. حيث تقوم بعض المشغلات بنقل المؤشر إلى الصف التالي، في حين يقو م بعضها الآخر بنقله إلى الصف السابق. لذلك تحتاج لمراجعة الوثائق المرفقة مع المشغل الذي تستخدمه قبل استخدام هذا المنهج.

وفيما يلي نستعرض مثال يطبق عمليات الأضافة والحذف بواسطة مشغل ODBS

```
1. //  برنامج يستخدم جميع العمليات
2. // local database using JDBC.
3. import java.sql.*;
4. class Chp18_5{
5.  public static void main(String[] args){
6.   Connection connection;
7.   Statement statement;
8.   ResultSet result;
9.   try{
10.                  // Load the driver (registers itself)
11.                  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
12.     }catch(Exception e){}
13.
14.   try{
15.       connection = DriverManager.getConnection("jdbc:odbc:DBJava","scott","tiger");
16.                  statement  = connection.createStatement();
17.                  //query database:SQL code
18.                  result  = statement.executeQuery("select * from emp");
19.
20.                  System.out.println("query database:SQL Select");
21.                  //print table
22.                  Print(result);
23.
24.                  // Insert to Table
25.                  statement.executeUpdate(" INSERT INTO emp ( num, name) VALUES (
5,'Aomr' )");
26.                  result  = statement.executeQuery("select * from emp");
27.
28.                  System.out.println("Insert database");
29.                  //print table
30.                  Print(result);
31.
32.                  //Update to Table
33.                  statement.executeUpdate("UPDATE emp SET name = 'Aomer' WHERE
num = 5");
```

```
34.                 result  = statement.executeQuery("select * from emp");
35.
36.                 System.out.println("Update database");
37.                 //print table
38.                 Print(result);
39.
40.                 // Delete to Table
41.                 statement.executeUpdate("DELETE FROM emp WHERE num = 5");
42.                 result  = statement.executeQuery("select * from emp");
43.
44.                 System.out.println("Delete database");
45.                 //print table
46.                 Print(result);
47.
48.                 }
49.                 catch(SQLException e){
50.                 System.out.println(e.toString());
51.                 }
52.                 finally{
53.                                     try{
54.                                             // Also closes Result
55.                                             result.close();
56.                                             statement.close();
57.                                             connection.close();
58.                                       }catch(SQLException se) {se.printStackTrace();}
59.                                     }//end finally
60.       }//end main
61.
62. static void Print(ResultSet recodr)
63.                                 {
64.                                 try{
65.                                             while(recodr.next()){
66.                                                     // Capitalization doesn't matter:
67.                                                     System.out.println(recodr.getString("num") + "\t\t"
+ recodr.getString("name"));
68.                                                                                 }
69.
70.                                     }catch(SQLException e)
71.                                                                         {
72.                                                                         System.out.println(e.toString());
73.                                                                         }
74.                                  System.out.println();
75.                                 }// end Print
76. }//end Chp18_5
```



18.11.17 استخدام JDBC للوصول إلى المعطيات الواصفة بواسطة الكائن DatabaseMetaData و ResultSetMetaData

يمكن تعريف المعطيات الواصفة وبعبارة بسيطة بأنها معطيات توصف معطيات موجودة في قاعدة البيانات، أي أنها معطيات توصف المعطيات المدارة من قبل قاعدة البيانات أو أنها توصف البنى والتطبيقات التي تضم المعطيات المدارة. على سبيل المثال: الوصف الملحق بالجداول أو بمواصفات الأعمدة أو ببنى المعطيات أو الوصف المتعلق بالمعطيات بحد ذاتها (مثل أنواع المعطيات).

تمكنك الواجهة JDBCAPI من الكشف عن المعطيات الواصفة لقاعدة البيانات واستعلام ناتج العبارة باستخدام كل من الواجهتين DatabaseMetaData و ResultSetMetaData.

بحيث تمكنك الواجهة الأولى من الحصول على معلومات عن واصفات قاعدة البيانات وبالتالي اتخاذ القرارات في زمن التنفيذ بناءً عليها، كما تزودك بالمعلومات الضرورية والتي تساعدك في كتابة تطبيقات إدارة قواعد البيانات.

أما الواجهة الثانية فتمكنك من التحقق من الواصفات (مثل عدد الأعمدة وأسمائها وأنواع معطياتها) التابعة لناتج العبارة. مما يتيح لك إمكانية استخدام هذه المعلومات لاستعادة ترويسات الأعمدة ضمن تقرير أو يساعدك في اختيار المنهج ResultSet.getXXX() المناسب من مجموعة المناهج هذه.

تتمحور هذه الفقرة حول كيفية الحصول على المعطيات الواصفة من قاعدة البيانات وناتج العبارة باستخدام الواجهتين السابقتين. حيث سد بدأ بالتقديم لكل من الواجهتين واستعراض أمثلة عنها تساعدك في التعرف على كيفية استخدامهما في تطبيقاتك بعدها سدننتقل لشرح الواجهة ResultSetMetaData ومن ثم الواجهة DatabaseMetaData وكل فقرة ستتضمن أمثلة مناسبة عن كيفية استخدام كل من الواجهتين.

- **واجهات JDBC المختصة بالمعطيات الواصفة**

كما سبق وذكر نا بأن JDBC يمتلك الواجهتين ResultSetMetaData و DatabaseMetaData اللتين تمكنانك من الوصول إلى المعطيات الواصفة، حيث تزودك الأولى بمعلومات عن الأعمدة الموجودة في ناتج العبارة مثل اسم العمود ونوع معطياته وطوله الأعظمي في حين تزودك الثانية بمعلومات من بنى قاعدة البيانات، مثل أسماء الجداول ومفاتيحها الرئيسية والثانوية وأنواع معطياتها.

يعتبر استخدام الواجهة ResultSetMetaData فعالاً عند الحاجة لبناء طريقة عملية تعالج منذ خلالها نواتج العبارة فباستخدام المعطيات الواصفة يمكنك التحقق من أنواع معطيات ناتج عبارتك واستدعاء المنهج getXXX() المناسب لاستعادة هذه المعطيات.

تعتبر التط بيقات المولدة للتقارير مثالاً حيوياً عن كيفية استخدام المعطيات الواصفة لبناء مناهج تعالج نواتج العبارة حيث تقوم هذه التطبيقات بإرسال استعلامات لاستعادة نواتج عبارة ومن ثم طباعة معطياتها، ويأتي هنا دور مناهج المعالجة لتقوم بسحب المعطيات من نواتج العبارة وتهـ يئتها للطباعة. فمثلاً يمكنك استخراج اسم عمود ما من الكائن ResultSetMetaData لإنشاء ترويسة عمود الخاصة بالتقرير.

بالطبع فإن معرفتك العميقة لبنية قاعدة البيانات ستساعدك في استخدام المعطيات بشكل أكثر فعالية فمثلاً لنفترض بأنك تقوم بتطوير أداة مساعدة لمدراء قواعد البيانات تمكنهم من تفحص قواعد بياناتهم أو إدارتها أو قد يستخدمها المطورون للتحري عن أنواع المعطيات المستخدمة في القاعدة (و للتحري عن بنية الجداول) ولاستعراض قائمة المستخدمين للنظام أو لمعرفة الميزات التي تدعمها القاعدة، وفي كل هذه الحالات يفترض بك إنشاء أداة إدارة قادرة على استعلام قاعدة البيانات للتحقق من المعلومات بشكل ديناميكي، بالإضافة إلى ذلك يمكنك استخدام الكائن ResultSetMetaData كمجس قاعدة بيانات يتحرى عن واصفاتها. حيث ستلجأ إلى هذا الأسلوب عندما تفتقر إلى المعرفة ببنية قاعدة البيانات والتي تريد التخاطب معها. كمثال عن ذلك، لنفترض بأنك تقوم بكتابة منهج إعداد يقوم

بإنشاء دعماً للجداول في قاعدة البيانات عندها يمكنك استخدام الكائن ResultSetMetaData لجمع المعلومات الضرورية مما يمكنك من إنشاء جداولك في قاعدة البيانات وفقاً لأنواع المعطيات المناسبة.

لقد قصدنا من عرض الأمثلة السابقة توضيح مدى أهمية استخدام المعطيات الواصفة والتطبيقات التي تعتمد عليها. على كل الأحوال، ستكون قادراً على إدراك ذلك حالما تصبح متآلفاً مع تلك الواجهات، وبالطبع ستجد عندها العديد من الأفكار الأخرى.

الواجهة ResultSetMetaData

تزو دك الواجهة ResultSetMetaData بمعلومات توضيحية واصفة للأعمدة التابعة لناتج العبارة، مثل عدد الأعمدة الحاوية على نوع معطيات معين ونوع المعطيات الخاص بكل عمود مثلاً وهكذا.

لا تزود هذه الواجهة أية معلومات تتعلق بقاعدة البيانات أو عن عدد الصفوف التابعة لناتج العبارة، وإنما تختص بناتج العبارة والذي تُربط معه فقط.

إنشاء الكائنات ResultSetMetaData

يتم إنشاء كائنات ResultSetMetaData من كائن ResultSet موجود مسبقاً ومتاح.
يوضح مقطع الشيفرة التالية كيفية إنشاء كائن ResultSetMetaData اسمه rsmd، والحاوي على عمود معطيات واصفة لجدول الموظفين.

```
//Assume a valid connection conn
Statement stmt = conn.createStatement();
//Create a result set
ResultSet rs = stmt.executeQuery("SELECT * FROM Employees");
//Obtain the result set metadata
ResultSetMetaData rsmd = rs.getMetaData();
```

استخدام الكائنات ResultSetMetaData

تزودك الواجهة ResultSetMetaData بعدد هائل من المناهج لاستعادة المعلومات المتعلقة بناتج العبارة. حيث سنقوم باستدعاء العديد من المناهج setter والمناهج getter لاستعادة المعطيات من الكائن ResultSetMetaData.

يمثل الجدول -18 قائمة المناهج الأكثر استخداماً (من وجهة نظرنا).

جدول -18

اسم المنهج	الشرح
getTableName( )	يعيد قيمة من النمط String تمثل اسم الجدول الذي تطلبه.
getColumnCount( )	يعيد قيمة من النمط int تمثل عدد الأعمدة الموجودة في ناتج العبارة.
getColumnName(intn)	يعيد قيمة من النمط String تمثل اسم العمود الموجود في الموضع n من ناتج العبارة.
getColumnType(intn)	يعيد قيمة من النمط int تمثل نوع معطيات JDBC للعمود في الموضع n لناتج العبارة.
getColumnTypeName(intn)	يعيد قيمة عن النمط String تمثل اسم نوع معطيات العمود (المحدد بالبارامتر الممرر n) كما تحدده قاعدة البيانات.

يعيد المنهج getColumnCount() عدد الأعمدة المعادة من قبل ناتج العبارة. حيث ستكون القيمة المعادة مساوية لعدد الأعمدة المطلوبة في الجزء SELECT من عبارة SQL. إذا قمنا بتحديد جميع الأعمدة باستخدام عبارة مثل العبارة التالية:

```
SELECT * FROM tableX
```

فإن المنهج getColumnCount() سيعيد عدد الأعمدة الموجودة في الجدول بالكامل.

ستجد فائدة استخدام هذا المنهج عندما ترغب في تحديد عدد الأعمدة الكلي الموجودة في ناتج العبارة والتي تريد استعادة معطياتها الواصفة. أما بالنسبة للمنهج getColumnName() فإنه يعيد اسم العمود ذو الترتيب المحدد بالبارامتر الممرر، قد يكون ترتيب العمود هو الترتيب ا لخاص بناتج العبارة أو قد يكون الترتيب الخاص بالجدول بأكمله في حال تمت استعادة جميع أعمدة الجدول. ستجد في الجدول السابق منهجين مفيدين وهما المنهج getColumnType() و getColumnTypeName() ، يعيد المنهج الأول قيمة من النمط int تمثل نوع معطيات JDBC المعرف في الحز مة java.sql.Types ، أما المنهج الثاني فإنه يزودك بنوع معطيات SQL الخاص بالعمود كما هو معرف في قاعدة البيانات.

من الضروري التذكير هنا بعدم الخلط بين أسماء أنواع معطيات قاعدة البيانات وأسماء أنواع معطيات JDBC، فهما مختلفان تماماً.

تستطيع باستخدام المنهج getColumnTypeName() إنشاء منهج عام (generic) يمكنك من استخراج معطيات ناتج العبارة عن طريق التحقق من النوع أو من اسم نوع معطيات العمود ومن ثم استخدام كتلة switch لاستدعاء المنهج getXXX() المناسب.

مثال عن استخدام الواجهة ResultSetMetaData

ستجد في الشيفرة ا لتالية مثالاً متكاملاً لاستخدام الكائن ResultSetMetaData ، حيث سـ نـ قوم باستعادة أنواع معطيات JDBC و SQL للأعمدة الموجودة في الجدولين: Employees و Location وعرضها في قائمة. بالإضافة إلى ذلك سنـ قوم بإنشاء منهج يمكنك من استعادة المعطيات من ناتج العبارة بناءً على نوع معطيات العمود الموجود في ناتج العبارة. يمكنك اعتبار هذا المنهج مثالاً عن إنشاء منهج عام يعالج ناتج العبارة باستخدام المعطيات الواصفة، في الحقيقة لقد استخدمنا هذا المنهج لعرض قائمة بمحتويات الجدولين Employees و Location.

```
1. //Specific imports
2. import java.sql.*;
3. import java.math.BigDecimal;
4.
5. public class Chp18_6 {
6.       //Global string buffer to buffer Strings before printing
7.       public static StringBuffer strbuf = new StringBuffer();
8.
9.       public static void main(String[] args) {
10.            //Declare Connection, Statement, and ResultSet variables
11.            Connection conn = null;
12.            Statement stmt = null;
13.            ResultSet rs = null;
14.            //Begin standard error handling
15.            try{
16.                  //Register driver
17.
18.                  String driver = "sun.jdbc.odbc.JdbcOdbcDriver";
19.                  Class.forName(driver).newInstance();
```

```
20.
21.                     //Open database connection
22.                     System.out.println("Connecting to database...");
23.                     conn =
DriverManager.getConnection("jdbc:odbc:DBJava","scott","tiger");
24.
25.                     //Create a Statement object
26.                     stmt = conn.createStatement();
27.                     rs = stmt.executeQuery("SELECT * FROM
Employee");
28.                     System.out.println("Meta data for Employees table:");
29.
30.                     //List column details
31.                     System.out.println("Column Information:");
32.                     printColumnInfo(rs);
33.                     System.out.println();
34.
35.                     //List result set data with generic method
36.                     System.out.println("List table data:");
37.                     printColumnNames(rs);
38.                     processRs(rs);
39.                     System.out.println();
40.
41.                     //Create a result set of data from the Location table
42.                     rs = stmt.executeQuery("SELECT * FROM Location");
43.                     System.out.println("Meta data for Location table:");
44.                     System.out.println("Column Information:");
45.                     printColumnInfo(rs);
46.                     System.out.println();
47.
48.                     //List table data with generic method
49.                     System.out.println("List table data:");
50.                     printColumnNames(rs);
51.                     processRs(rs);
52.                     System.out.println();
53.                     //Standard error handling
54.                     } catch(SQLException se) {
55.                             //Handle errors for JDBC
56.                             se.printStackTrace();
57.                     } catch(Exception e) {
58.                             //Handle errors for Class.forName
59.                             e.printStackTrace();
60.                     } finally {
61.                             try {
62.                                     if(conn!=null)
63.                                     conn.close();
```

```
64.                                     } catch(SQLException se) {
65.                                             se.printStackTrace();
66.                                     }//end finally try
67.                             }//end try
68.                             System.out.println("Goodbye!");
69.                     }//end main
70.
71.     //Generic method to retrieve data from a result set
72.     public static void processRs(ResultSet rs)throws SQLException{
73.             //Create ResultSetMetaData object
74.             ResultSetMetaData rmd = rs.getMetaData();
75.             //Loop through result set and retrieve column information.
76.             while(rs.next()){
77.                     for (int col=1;col<=rmd.getColumnCount();col++)
78.                             getData(rs,rmd.getColumnType(col),col);
79.                     System.out.println(strbuf.toString());
80.                     strbuf = new StringBuffer();
81.
82.                     }//end while
83.             }//end processRs()
84.
85.     //Prints column names as headings
86.     public static void printColumnNames(ResultSet rs)throws
SQLException{
87.             ResultSetMetaData rmd = rs.getMetaData();
88.             StringBuffer sb = new StringBuffer();
89.             for (int col=1;col<=rmd.getColumnCount();col++)
90.                     sb.append(rmd.getColumnName(col) + " ");
91.             System.out.println(sb.toString());
92.             }//end printColumnNames()
93.
94.     //Method to determine the data type of a result set column
95.     //then call the correct getXXX() method. For brevity, not all
96.     //JDBC data types are supported
97.     public static void getData(ResultSet rs, int type, int colIdx)throws
SQLException{
98.             //Temporary variable to hold value from result set as a string.
99.             String s;
100.            switch (type){
101.                    //Handle character related data types
102.                    case java.sql.Types.CHAR:
103.                    case java.sql.Types.VARCHAR:
104.                    s = rs.getString(colIdx);
105.                    strbuf.append(s + " ");
106.                    break;
107.                    //Handle the INTEGER data type
```

```
108.                    case java.sql.Types.INTEGER:
109.                    int i = rs.getInt(colIdx);
110.                    strbuf.append( i + " ");
111.                    break;
112.                    //Handle the NUMERIC data type
113.                    case java.sql.Types.NUMERIC:
114.                    BigDecimal bd = rs.getBigDecimal(colIdx);
115.                    s = bd.toString();
116.                    strbuf.append(s + " ");
117.                    break;
118.                    //Handle date related data types. I just want the date
119.                    //portion so I combine the two data types.
120.                    case java.sql.Types.TIMESTAMP:
121.                    case java.sql.Types.DATE:
122.                    java.sql.Date d = rs.getDate(colIdx);
123.                    s = d.toString();
124.                    strbuf.append(s + " ");
125.                    break;
126.                    }//end select
127.    }//end getData()
128.
129.    //Method to print column metadata.
130.    public static void printColumnInfo(ResultSet rs)throws SQLException{
131.            ResultSetMetaData rsmd = rs.getMetaData();
132.            System.out.println("Column, " + "JDBC_Type,     "+
"Database_Type_Name ");
133.            int cols = rsmd.getColumnCount();
134.            for (int colIdx=1;colIdx<=cols;colIdx++){
135.                    String name = rsmd.getColumnName(colIdx);
136.                    int type = rsmd.getColumnType(colIdx);
137.                    String typeName =
rsmd.getColumnTypeName(colIdx);
138.                    System.out.println(name + ", " + type + ", " +
typeName);
139.            }//end for
140.    }// end printColumnInfo
141. }// end Chp18_6
```

```
General Output
 1 --------------------Configuration: <Def
 2 Connecting to database...
 3 Meta data for Employees table:
 4 Column Information:
 5 Column, JDBC_Type,  Database_Type_Name
 6 num, 4, INTEGER
 7 name, 12, VARCHAR
 8 salary, 4, INTEGER
 9 phone, 12, VARCHAR
10 notes, 12, VARCHAR
11
12 List table data:
13 num name salary phone notes
14 1 ali 1010 10127524 null
15 2 ammar 101011 7527 null
16 3 Ammar 0 null null
17 4 Aomer 0 null null
18
19 Meta data for Location table:
20 Column Information:
21 Column, JDBC_Type,  Database_Type_Name
22 Num, 4, INTEGER
23 Name, 12, VARCHAR
24 JOP, 12, VARCHAR
25
26 List table data:
27 Num Name JOP
28 1 Msbil DBA
29 2 Nabil ADD
30
31 Goodbye!
```

الواجهة DatabaseMetaData

تمكنك الواجهة DatabaseMetaData من التحري عن البنى الموجودة في قاعدة البيانات. حيث تمتلك هذه الواجهة أكثر من مائة منهج تعيد كلها معلومات عن قاعدة البيانات. ستصادفك أثناء عملك كمبرمج لقواعد بيانات في لغة Java احتياجات تدفعك لاستخدام الواجهة DatabaseMetaData ، فمثلاً عند تطويرك لأداة قاعدة بيانات مثل منهج تنصيب (installation) أو أداة لإدارة قاعدة البيانات (DBA tool) أو مُوثق للكيانات (schema documenter، والكيان أو schema في قاعدة البيانات هو عبارة عن المستخدم الذي يملك كائنات) عندها ستقف عند هذه الواجهة لتقضي معها الكثير من الوقت.

إنشاء الكائن DatabaseMetaData

يمثل الكائن Connection الاتصال بقاعدة البيانات كما أنه يصنع كائناً من النمط DatabaseMetaData باستخدام المنهج ()getMetaData. يضم الكائن DatabaseMetaData معلومات عن قاعدة البيانات التي يتصل معها الكائن Connection.

يمثل مقطع الشيفرة التالي كيفية إنشاء الكائن DatabaseMetaData.

```
//Assume a valid Connection conn
DatabaseMetaData dmd = conn.getMetaData();
```

يتوجب عليك قبل إنشاء هذا الكائن تسجيل الدخول إلى قاعدة البيانات والحصول على اتصال معها.

استخدام كائنات DatabaseMetaData

يمتلك الكائن DatabaseMetaData الكثير من المناهج والخصائص التي تزودك بالكثير من المعلومات عن قاعدة البيانات. ستجد في بداية الأمر بأن التعامل مع هذه المناهج صعب نوعاً ما وذلك لكثرة عددها لذلك قمت بتقسيمها إلى قسمين رئيسين لأساعدك في تصنيفها. يتعامل القسم الأول مع خصائص قاعدة البيانات في حين يتعامل القسم الثاني مع بنى قاعدة البيانات.

المناهج المختصة بخصائص قاعدة البيانات

كما ذكرنا فإن استخدام الكائن DatabaseMetaData يمكنك من الحصول على الكثير من المعلومات المتعلقة بقاعدة بياناتك وخصائصها. فمثلاً يمكنك باستخدامه الإجابة عن كل هذه التساؤلات:

- هل تدعم قاعدة البيانات التحديثات التسلسلية (batch updates)؟
- ما هي كلمات SQL المفتاحية؟
- ما هو اسم المستخدم الذي استخدمته للاتصال؟
- ما هي أنواع المعطيات التي تدعمها قاعدة البيانات؟

يمكن تقسيم المناهج المختصة بخصائص قاعدة البيانات إلى عدة أقسام ثانوية أخرى:

قسم مختص بمعلومات قاعدة البيانات العامة وقسم يختص بحدود قاعدة البيانات وقسم يختص بالميزات التي تدعمها قاعدة البيانات. بالنسبة للـ مناهج التي تقدم معلومات عامة عن قاعدة البيانات فإنها تعيد قيم من النمط String. فمثلاً يمكنك الحصول عن طريقها على أسماء الكيانات (schema) أو أسماء قواعد البيانات أو كلمات SQL المفتاحية أو توابع قاعدة البيانات. تعيد هذه المناهج قيمة مفردة أو قيم متعددة.
فمثلاً يعيد المنهج getDatabaseName() سلسلة محرفية مفردة تمثل اسم قاعدة البيانات، في حين يعيد المنهج getSQLFunction() قائمة تحتوي على العديد من السلاسل المحرفية والتي تمثل أسماء توابع SQL الموجودة في قاعدة البيانات. أما بالنسبة للمناهج التي تختص بحدود قاعد ة البيانات فإنها تعيد قيمة من النمط int ، وكمثال عنها المناهج التي تعيد العدد الأعظمي لعدد الاتصال المسموح بها أو الطول الأعظمي لاسم العمود أو الحجم الأعظمي للصف المعاد في ناتج العبارة وغيرها.
بقي القسم الثالث والذي يختص بالتحقق من دعم ميزة معينة في قاعدة البيانات أم لا وبالطبع فإن مناهج هذا القسم تعيد قيمة من النمط Boolean.
فمثلاً يمكنك التحقق فيما إذا كانت قاعدة بياناتك تدعم المعالجة التسلسلية والمناقلات أم لا باستخدام المنهج supportBatchUpdate() والمنهج supportTransaction().

المعلومات البنيوية

يتيح لك ا ستخدام الكائن DatabaseMetaData إمكانية الحصول على معلومات عن الجداول أيضاً وعن الإجرائيات المخزنة وعن علاقات التكامل المرجعي وعن أنواع المعطيات أيضاً. حيث ستعيد المناهج المختصة بذلك كائنات من النمط ResultSet وبالطبع سيختلف ناتج عبارة عن الآخر تبعاً للمنهج المستخدم في استعلام قاعدة البيانات.

يتم استدعاء هذه المناهج بشكل مباشر، على أي حال فإن بعض هذه المناهج يمكنك من استخدام نماذج للسلاسل النصية (String patterns) كبارامترات بحيث تضع قيودك الخاصة على الصفوف التي ترغب باستعادتها. تمثل نماذج السلاسل النصية بشكل أساسي محارف البدل (wildcard characters) والتي تمكنك من تضييق مجال البحث، كما يمكنك استخدام محرف الخط السفلي (_)

(underscore) لمطابقة محرف مفرد (single) أو استخدام المحرف (%) لمطابقة 0 محرف أو أي عدد من المحارف.

على العموم ستستخدم نماذج السلاسل النصية لتحديد عدد الصفوف المعادة عن طريق مناهج الكائن DatabaseMetaData الخاصة. فمثلاً إذا قمت باستدعاء المنهج getColumn() لاستعادة معلومات عن أعمدة معينة في جدول ما في قاعدة بياناتك فإنك ستحصل على كل أسماء الأعمدة لكل عمود في جداول قاعدة البيانات وبذلك فإن ناتج العبارة لن يحتوي على معلومات الأعمدة للجدول المطلوب فقط بل سيحتوي معلومات عن جداول النظام والمستخدم الأخرى مما يجعل ناتج العبارة كبيراً بالإضافة إلى أنه يحوي على معلومات غير مهمة.

نماذج السلاسل النصية

تمكنك نماذج السلاسل النصية من ترشيح عدد الصفوف المعادة في ناتج العبارة وبالطبع يقتصر استخدام نماذج السلاسل النصية على مناهج الكائن DatabaseMetaData التي تدعم ذلك. يمكنك المحرفان الخاصان (% و _) من إنشاء نماذج السلاسل النصية، فباستخدام المحرف (%) يمكنك مطابقة 0 محرف أو أي عدد من المحارف. فمثلاً لتكن السلسلة EM% ستسمح هذه السلسلة للمنهج باستعادة المعطيات التي تبدأ بالمحرفين EM فقط. أما نموذج السلسلة التالي EM%S فسيعيد المعطيات التي تبدأ بالمحرفين EM وتنتهي بالمحرف S، وبالطبع يمكن أن تكون المحارف بينهما أي شيء وغير مقيدة بطول معين أو شكل معين.

يمكنك استخدام المحرف (_) لمطابقة محرف مفرد. فمثلاً النموذج EM_ يعني أنه يمكنك استعادة أية معطيات تبدأ بالمحرفين EM وتنتهي بمحرف واحد مهما يكن هذا المحرف، أي أن طول السلسلة بالكامل سيكون ثلاثة محارف فقط.

سنركز في بقية الفقرات على كيفية استخدام الكائن DatabaseMetaData لاستعادة المعلومات من الجداول والأعمدة وقواعد التكامل المرجعي وبالطبع يمكنك استخدامه لاستعادة الكثير من المعلومات الأخرى ولكننا اقتصرنا على ما سبق لتكرار استخدامها في تطبيقات قواعد البيانات.

المعطيات الواصفة للجدول والعمود

يمكنك الحصول على تفاصيل كل الجداول أو عن أعمدة جدول معين في حال كنت تمتلك صلاحيات للوصول إليه. فهذا يتضمن كيانات (schema) خاصة بالنظام والمستخدمين بالإضافة إلى الكيان الخاص بك، وكما ذكرنا سابقاً عليك استخدام نماذج السلاسل النصية لتجنب استعادة حجم كبير من المعطيات غير الضرورية.

يعيد المنهجان ()getTables و ()getColumns معلومات عن الجداول وعن أعمدة جدول معين في قاعدة البيانات، وستجد بأن هذين المنهجين يختلفان بعض الشيء عن بقية مناهج اللغة Java ، حيث يعود ذلك إلى أنهما يأخذان بارامترات على شكل سلاسل نصية منمذجة. يمثل السطر التالي تعريفاً للمنهج ()gettable:

```
public ResultSet getTables(String catalog,
            String schemaPattern,
            String tableNamePattern,
            String[] types);
```

عليك التفكير بهذه البارامترات على أنها فلاتر لناتج العبارة فكل واحد منه يمكنه تقليص عدد الأعمدة المعادة. ففي التعريف السابق يمثل البارامتر الأول دليل (Catalog ، يشبه الدليل فضاء الاسم "namespace" ويستخدم للفصل بين بنى البيانات "data structures") قاعدة البيانات والذي ترغب بالبحث عن الجداول ضمنه. يقوم البارامتران التاليان بترشيح مجموعة النتائج من خلال تحديد الكيان (schema) والجداول التي ترغب في أن تكون مجموعة البيانات محددة بهم فقط. لاحظ أنه يمكنك هنا استخدام نماذج السلاسل النصية للتحكم بالبيانات التي ستعاد مع مجموعة البيانات. أما البارامتر الأخير فهو عبارة عن مصفوفة سلاسل نصية تمثل "أنواع الجداول" والتي تريد استعادة المعلومات منها. لا يمثل هذا البارامتر نوع معطيات الجدول وإنما يمثل الفئة التي ينتمي إليها الجدول، حيث تعتمد انواع الجداول على قاعدة البيانات المستخدمة، يمكنك الحصول على أنواع الجداول المتوفرة من خلال المنهج ()getTableTypes (أمثلة عن أنواع الجداول المختلفة: TABLE، VIEW، SYSTEM.

يمثل مقطع الشيفرة التالي كيفية استخدام المنهج ()getTables:

```
//Assume a valid Connection object conn
DatabaseMetaData dmd = conn.getMetaData();
//Define parameters.
String catalog = null;
String schema = "TODDT";
String tableName = "E%";
String [] types = {"TABLE"}
```

قبل استدعاء المنهج getTables() قمنا بتمرير القيم لبارامترات المنهج. حيث أعطينا البارامتر catalog القيمة null والتي تطلب من المنهج تجاهل هذا البارامتر. في الواقع يمكنك أن تطلب من المنهج أن يتجاوز أية بارامتر آخر من خلال إسناد القيمة null إليه حيث تعادل هذه القيمة (null) النجمة (*) المستخدمة في عبارة SELECT.
أما البارامتر الثاني والذي يمثل اسم الكيان فأعطيناه القيمة "TODDT" لأننا نريد استعادة الجداول التابعة للكيان TODDT فقط.

لقد استخدمنا مع البارامتر الثالث نموذج سلسلة نصية لكي نختار جميع الجداول التي تبدأ بالحرف E فقط (لتحقيق هذه الغاية استخدمنا المحرف % للدلالة على قبول أي محرف يأتي بعد المحرف E). أخيراً استخدمنا مصفوفة سلاسل نصية من أجل البارامتر الأخير لتحديد فئة الجداول التي سيحتويها ناتج العبارة.

يحتوي ناتج العبارة الذي سيعيده المنهج gettable() على خمسة أعمدة وهي: catalogname، tablename، tabletype، remarks.

في هذا المثال ستكون كل الأعمدة من النمط string لذلك يمكن استخدام المنهج ResultSet.getString() لاستعادة معلوماتها. يمكنك استخدام المنهج getColumn() بأسلوب مماثل وتوضح الشيفرة التالية آلية ذلك:

```
//Assume a valid Connection object conn
DatabaseMetaData dmd = conn.getMetaData();
//Create a ResultSet object that holds the database table
//information.
String catalog = null;
String schema = "TODDT";
String tableName = "E%"
String columnName = null;
ResultSet rsCols=dmd.getColumns(null, schema, tableName, null)
```

من البارامترات الأربعة السابقة، تقبل البارامترات الثلاثة الأخيرة نماذج السلاسل النصية، فالبارامتر الأول هو نفسه بارامتر المنهج getTables() حيث يحدد الدليل الذي ترغب بالعمل معه. يحدد البارامتر الثاني الكيان، حيث قمنا بتحديد الكيان TODDT (والذي صنعناه) وذلك للحصول على الأعمدة التابعة للجداول الموجودة في هذا الكيان فقط.

استخدمنا من أجل البارامتر التالي نموذج سلسلة نصية لتحديد الجداول التابعة للكيان TODDT والتي تبدأ بالحرف E. أخيراً قمنا بتحديد البارامتر الأخير (والذي يسمح بتحديد اسم العمود عن طريق استخدام نموذج سلسلة نصية) بالقيمة null لأننا نريد كل الأعمدة.

أنواع المعطيات والمعطيات الواصفة

يمكنك التحقق من أنواع المعطيات التي تدعمها قاعدة بياناتك وذلك باستخدام المنهج getTypeInfo(). لكن ناتج العبارة الذي يعيده هذا المنهج معقد نوعاً ما فهو يحتوي على 18 عمود تزودك بمعلومات مثل اسم نوع المعطيات والحساسية تجاه الأحرف ونوع معطيات JDBC وفيما إذا كان العمود يحتوي على قيم خالية أم لا و...الخ.

قد تجد أن استخدام هذا المنهج مفيد عند كتابتك لتطبيق خدمي يتحقق من أنواع المعطيات في قاعدة البيانات.

يمثل مقطع الشيفرة التالي منهج يمكنه عرض قائمة بكل أنواع المعطيات في قاعدة البيانات، حيث يعرض اسم نوع المعطيات كما تعرفه قاعدة البيانات بالإضافة إلى قيمة صحيحة تمثل نوع معطيات JDBC المقابل.

```
//Assume a valid Connection object conn
DatabaseMetaData dmd = conn.getMetaData();
//Create result set
ResultSet rs = dmd.getTypeInfo();
//Loop through result set
while(rs.next(){
System.out.println(rs.getString("TYPE_NAME +" " + ("rs.getShort("DATA_TYPE"));
                }
```

كما يمكنك استخدام المنهج getUDT() للحصول على ناتج عبارة يحتوي على معلومات عن الأنواع المعرفة من قبل المستخدم والموجودة في مخدم قاعدة البيانات.

المعطيات الواصفة للمفتاحين الرئيسي والثانوي

يزودك الكائن DatabaseMetaData بالمعلومات المتعلقة بقواعد التكامل المرجعي من أجل قواعد البيانات أو مجموعة من الجداول، حيث يزودك المنهجان getPrimaryKeys() و getImportedKeys() بمعلومات عن المفتاح الرئيسي والثانوي.

مثال عن استخدام الكائن DatabaseMetaData

تقدم الشيفرة التالية العديد من الأمثلة عن استخدامات مناهج الواجهة DatabaseMetaData والتي شرحناها في الفقرات السابقة. يبدأ البرنامج بعرض قائمة تحتوي على اسم ونسخة قاعدة البيانات التي تم تأسيس الاتصال معها بعدها يقوم بعرض قائمة لكل الجداول الموجودة في الكيان TODDT، بالإضافة إلى أنواع المعطيات المستخدمة والمفاتيح الرئيسية والثانوية. أما القائمة الثالثة فتكتفي بأنواع المعطيات التي تدعمها قاعدة البيانات. ستلاحظ بأن المنهج getTypeInfo() ينشأ ناتج عبارة يزودك باسم نوع المعطيات في قاعدة البيانات وأنواع معطيات JDBC المقابلة. أخيراً يقوم البرنامج باستدعاء المناهج getNumericFunction() و getStringFunction() و getTimeDateFunction() لعرض قائمة بالكلمات SQL المفتاحية والتوابع المساعدة.

```
1. // Class DatabaseMetaData
2. //Specific imports
3. import java.sql.*;
4.
5. public class Chp18_7 {
6.
7.        public static void main(String[] args) {
8.                //Create Connection, Statement, and ResultSet objects
9.                Connection conn = null;
10.               Statement stmt = null;
11.               ResultSet rs = null;
12.
13.               //Begin standard error handling
14.               try{
15.                       //Load a driver
16.                       String driver = "oracle.jdbc.driver.OracleDriver";
17.                       Class.forName(driver).newInstance();
18.
19.                       //Obtain a Connection object
20.                       System.out.println("Connecting to database...");
21.                       System.out.println();
22.                       String jdbcUrl =
"jdbc:oracle:thin:@myserver:1521:ORCL";
23.                       String user = "TODDT";
24.                       String pwd = "mypwd";
25.                       conn =
DriverManager.getConnection(jdbcUrl,user,pwd);
```

```
26.
27.
28.                     //Initialize a DatabaseMetaData object
29.                     DatabaseMetaData dmd = conn.getMetaData();
30.
31.                     //Retrieve database name and version
32.                     String dbname = dmd.getDatabaseProductName();
33.                     dbname = dbname + " " +
dmd.getDatabaseProductVersion();
34.                     System.out.println("Database information:");
35.                     System.out.println(dbname);
36.                     System.out.println();
37.
38.                     //Retrieve a result set with table information
39.                     String [] types = {"TABLE"};
40.                     rs = dmd.getTables(null,user,null,types);
41.                     while(rs.next()){
42.                             String tableName = rs.getString(3);
43.                             System.out.println("Table Name: " +
tableName);
44.                             System.out.println(" Column, Data Type");
45.                             ResultSet
rsCols=dmd.getColumns(null,user,tableName,null);
46.                             while(rsCols.next()){
47.                                     System.out.println(""+
rsCols.getString("COLUMN_NAME")
48.                                             + ", " +
rsCols.getString("TYPE_NAME"));
49.                             }//end while rsCols
50.                             System.out.println();
51.
52.                             //Get primary keys for tables
53.                             ResultSet
rsPkey=dmd.getPrimaryKeys(null,user,tableName);
54.                             if(rsPkey.next()){
55.                                     System.out.println(" PK Name,
Column Name");
56.                                     do{
57.                                             System.out.println(" " +
rsPkey.getString("PK_NAME") + ", "
58.                                             +
rsPkey.getString("COLUMN_NAME"));
59.                                     }while(rsPkey.next());
60.                             System.out.println();
61.                             }//end primary key
62.
```

```
63.                         //Get foreign keys for tables
64.                         ResultSet
rsFkey=dmd.getImportedKeys(null,user,tableName);
65.                         if(rsFkey.next()){
66.                             System.out.println(" FK Name, FK
Table, Column Name");
67.                             do{
68.                                 System.out.println(" " +
rsFkey.getString("FK_NAME") + ", "+ rsFkey.getString("PKTABLE_NAME")
+ ", "+ rsFkey.getString("FKCOLUMN_NAME"));
69.                                 }while(rsFkey.next());
70.                             System.out.println();
71.                         }//end foreign key
72.                 }//end while for table information
73.
74.                 //Get supported data types
75.                 rs = dmd.getTypeInfo();
76.                 System.out.println("Supported data types:");
77.                 while(rs.next()){
78.                         System.out.println("Database Type Name:"
+ rs.getString("TYPE_NAME")+ " JDBC Type: "+
rs.getShort("DATA_TYPE"));
79.                 }//end while
80.                 System.out.println();
81.
82.                 //Retrieve SQL Keywords, numeric functions, string,
and
83.                 //time and date functions.
84.                 System.out.println("SQL Keywords:");
85.                 String sql = dmd.getSQLKeywords();
86.                 System.out.println(sql);
87.                 System.out.println();
88.                 System.out.println("Numeric Functions:");
89.                 String numeric = dmd.getNumericFunctions();
90.                 System.out.println(numeric);
91.                 System.out.println();
92.                 System.out.println("String Functions:");
93.                 String string = dmd.getStringFunctions();
94.                 System.out.println(string);
95.                 System.out.println();
96.                 System.out.println("Time and Date Functions:");
97.                 String time = dmd.getTimeDateFunctions();
98.                 System.out.println(time);
99.                 System.out.println();
100.
101.                //Standard error handling
```

```
102.                    } catch(SQLException se) {
103.                            //Handle errors for JDBC
104.                            se.printStackTrace();
105.                    } catch(Exception e) {
106.                            //Handle errors for Class.forName
107.                            e.printStackTrace();
108.                    } finally {
109.                            try {
110.                                    if(conn!=null)
111.                                            conn.close();
112.                                    } catch(SQLException se) {
113.                                            se.printStackTrace();
114.                                    }//end finally try
115.                            }//end try
116.                    System.out.println("Goodbye!");
117.                    }//end main
118.
119. }//end Chp18_7.java
```

18.11.18 إبداع اللمسات الفنية في الجداول بواسطة الصنف JTable

منذ مدة طويلة يحاول الكثير من المطوِّرين إيجاد الطرق السهلة والعملية لإنشاء وإضافة التأثيرات الفنية (كتغيير الألوان، اللمعان، الحواف ثلاثية الأبعاد،...) إلى عناصر واجهة المستخدم، بحيث تتعدل هذه التأثيرات وفق التغييرات التي تطرأ على المعطيات الموجودة ضمن تلك العناصر.

تم أخيراً التوصل إلى نموذجٍ بسيطٍ وفعالٍ لتصميم تلك المكوِّنات الفنية. يدعى هذا النموذج باسم النموذج Decorator. تعتمد فكرة هذا النموذج على الاستفادة من إمكانيات وقدرات المكوِّنات الفنية المعرَّفة مسبقاً في بناء مكوِّنات فنية جديدة مخصَّصة لإضافة تأثيرات جديد ة غير موجودة سابقاً إلى عناصر واجهة المستخدم. لا شك أن استخدام هذا النموذج يوفر الكثير من الجهد والوقت المُستغرق في بناء كل من تلك المكوِّنات. فالعديد من الطرق والإجراءات البرمجية أصبحت جاهزة الآن من خلال استخدام طرق وفعاليات المكوِّن الموجود والمعرَّف مسبقاً.

لا بد أن نشير أيضاً إلى الدور الهام الذي يلعبه التعريف الجيد لواجهات المستخدم التي سوف تحتضن العروض الحاوية على تلك التأثيرات الفنية. حيث تؤدي كل هذه الأشياء معاً إلى الحصول على البنى الأساسية الفعالة والمتقنة اللازمة لبناء التطبيق المطلوب.

التكوين الفني للجداول JTable

يمكن تعريف عملية التكوين الفني الافتراضي داخل لغة Java ، بأنها الآلية التي تتم داخل الحزمة swing بهدف رسم وتنسيق العناصر على واجهة المستخدم في التطبيق.

عند الانتقال إلى جداول الصنف JTable (الجداول المبنية باستخدام هذا الصنف) ، فإننا نجد مجموعة من المكونات الفنية الافتراضية التي تتحكم بعملية التكوين الفني لكل من خلايا تلك الجداول. حيث تتحكم تلك المكوِّنات بكل مما يلي:
تنسيق المعطيات داخل الخلية، الخط، الحواف الخارجية، بالإضافة إلى ألوان الخلفية والأمامية ضمن تلك الخلية.
كما يتم عادة التشارك على تلك المكوِّنات بين عدد من خلايا الجدول، بحيث تحمل كل من تلك الخلايا نفس التكوين الفني المخزن ضمن ذلك المكوِّن (المكوِّن المشترك).

يمكن بناء على ذلك تشبيه تلك المكوِّنات بالأختام المطاطية التي تُحمَّل بمواصفاتٍ فنية معينة تنقلها بدورها إلى جميع الخلايا التي تمر عليها ضمن الطراز الخاص بالجدول المطلوب (table model).
يمكن لهذه المكوِّنات أن ترتبط إما بأعمدة داخل الجدول أو بأنواع المعطيات المعرَّفة في لغةJava.

تقدم الحزمة swing مجموعة أولية من المكوِّنات الفنية الافتراضية لخلايا الجدول JTable. تدعم هذه المكوِّنات عرض وتنسيق أنواع المعطيات المعرَّفة في لغة Java. تشتمل هذه الأنواع على كل مما يلي: المعطيات البوليانية (Boolean)، الصور (Image) ، الأيقونات (Image Icon) ، التاريخ (Date) ، الأعداد المضاعفة (Double)، ا لأعداد الحقيقية ذات الفاصلة العائمة (Float)، الصنف العددي (Number) ، بالإضافة إلى الصنف Object الأب الافتراضي لجميع الأصناف المكتوبة من قبل المستخدم في لغة Java.

يترتب على المطوِّر إضافة أي شيفرة برمجية لاستخدام تلك المكوِّنات الافتراضية. حيث يتم تقديم هذه المكوِّنات بشكل افتراضي عند استخدام الصنف JTable مباشرة.

التحكم بألوان الأسطر داخل الجداول JTable

لا يُكلف المطوِّر في أغلب الأحيان مهمة التكوين الفني للعنصر المقدم داخل واجهة المستخدم كما قلنا سابقاً. حيث تتولى الحزمة swing هذه المهمة. إلا أن بعض المطوِّرين قد يرغبون ببناء تكوينات فنية مختلفة عن تلك المقدمة في الحزمة swing. يتطلب تحقيق ذلك كما قلنا سابقاً بناء أصناف برمجية تدعى المكوِّنات الفنية (Renderers) لإدارة مثل هذه التكوينات.

بالنسبة للجداول وعند بناء مثل هذه المكوِّ نات الخاصة، لا بد من إنشاء صنف يحقق الواجهة TableCellRenderer من الحزمة swing. حيث ينبغي على جميع المكوِّنات الفنية الخاصة بالجداول، تحقيق هذه الواجهة كخطوة أولى في عملية البناء.

تعيد الطريقة getTableCellRendererComponent من تلك الواجهة، أحد عناصر واجهة المستخدم (UI) الذي يمكن للمطوِّر أن يستخدمه في عرض قيمة ما. حيث تهدف هذه الطريقة إلى الحصول على هذا العنصر بعد تطبيق التأثيرات الفنية الخاصة عليه.

تقدم الحزمة swing تحقيقاً افتراضياً للواجهة TableCellRenderer من خلال الصنف DefaultTableCellRenderer المحقق لـ تلك الواجهة. يتفرع عن هذا الصنف، الصنف JTable كما تعيد الطريقة getTableCellRendererComponent في هذا التحقيق المؤشر this الذي يشير إلى ذات الكائن المستخدم والحاوي على تلك الطريقة.

يمكن للمطوِّر سلوك أحد الطريقتين التاليتين في بناء المكوِّنات المخصصة:

1- ا لاشتقاق من الصنف DefaultTableCellRenderer المحقق للواجهة TableCellRenderer.

2- التحقيق المباشر للواجهة TableCellRenderer.

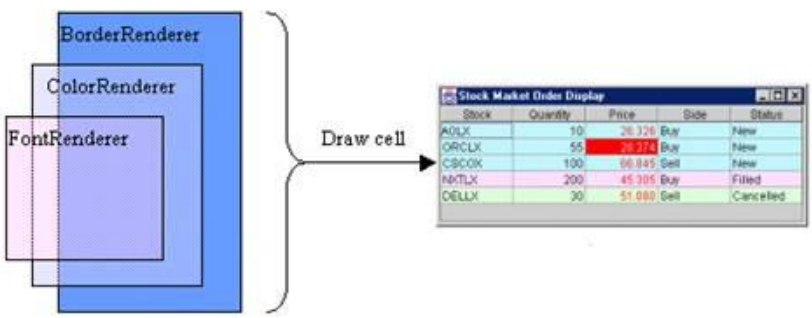
استخدام النموذج Decorator

كما قلنا سابقاً فإن المكوِّنات الفنية مسؤولة عن تحديد المظهر والانطباع العام عن العناصر المقدَّم. بالحديث عن الجداول، فإنه من الضروري أن تمتلك تلك المكوِّنات القدرة على معالجة كل مما يلي: تنسيق المعطيات داخل الخلايا، الخطوط، الألوان، الحواف الخارجية للخلية.

تبنى معظم المكوِّنات الفنية المخصصة عادة لحل مشكلة محددة ضمن عملية بناء واجهة المستخدم (UI) ، مما يعني أن هذه المكوِّنات ترتبط عادة بمشكلة محددة وبالتالي فإنها تحتاج إلى إعادة كتابتها من أجل كل عملية عرض. يمكن اتباع طريقة أخرى في بناء تلك المكوِّنات تتم ثل في تجزيء المكوِّن الكبير الشامل إلى مجموعة من المكوِّنات العامة الصغيرة التي يمكن استخدامها في جميع الجداول. تتحمل كل واحدة من هذه المكوِّنات مسؤولية إعداد أحد أوجه عملية التكوين الفني المتعلق بمظهر الخلية على واجهة المستخدم.

بناء على ذلك يمكن إنشاءالمكوِّنات
ColorRenderer،BorderRenderer،FontRenderer
للتحكم بكل من اللون والحدود الخارجية والخطوط على الترتيب.

يوضح الشكل -18 التجزيء السابق للمكوِّنات الفنية:



شكل -18 :

بالإضافة إلى ما سبق فإنه من الممكن تصميم تلك المكوِّنات الصغيرة العامة بحيث تعمل معاً وذلك باستخدام النموذج Decorator 2.

يمكن تحقيق هذه التقنية بجعل طريقة البناء الخاصة بالمكوِّن الحالي تقبل وسيطاً يمثل مكوِّناً فنياً آخر مسبق التعريف. يمكّن هذا الإجراء المكوِّن الفني من تغليف أو إحاطة مكوِّنٍ آخر مسبق التعريف (مكوِّنٍ فني مسبق الإنشاء كما هي الحال في المكوِّنات الافتراضية ا لمقدَّمة في الحزمة swing بشكل مترافق مع الصنف JTable).

تعني عملية الإحاطة هذه، أن يستفيد المكوِّن المحيط من فعاليات وقدرات المكوِّن المُحاط.

يوضح الجزء التالي من الشيفرة، طريقة البناء الخاصة بالصنف ColorRenderer:

```
public class ColorRenderer implements TableCellRenderer {
        protected TableCellRenderer delegate;
        public ColorRenderer(TableCellRenderer anotherRenderer) {
        this.delegate = anotherRenderer؛
        }
        ...
        };
```

نلاحظ خلال الشيفرة السابقة تهيئة الصنف delegate بالمكوِّن الفني anotherRenderer المُحرّر كوسيط. يعمل المكوِّن anotherRenderer كنائب (delegate) عن المكوِّن ColorRenderer حيث ينوب عنه في استدعاء الطرق الضرورية والتي كان من المفترض أن تُكتب ضمن المكوِّن ColorRenderer.

تعمل الطريقة TableCellRendererComponent في الصنف ColorRenderer على إعادة العنصر ا لمستهدف بعملية التكوين الفني على واجهة المستخدم، محمَّلاً بالتأثيرات الفنية الخاصة المطلوبة. يتم الحصول على هذا العنصر داخل تلك الطريقة من خلال استدعاء ذات الطريقة من المكوِّن النائب delegate.

تتمثل التأثيرات الخاصة المطلوبة في الطريقة السابقة من الصنف ColorRenderer في الحفاظ على ألوان الخلفية والأمامية للخلية في حال اختيارها بالمؤشر من قبل المستخدم، وتحميلها بألوانٍ خاصة (قرنفلي وأزرق للخلفية والأمامية على الترتيب) عند عدم اختيارها من قبل المستخدم. حيث يحدد العلم المنطقي ISSelected حالة الاختيار للخلية الهدف.

تعرض الشيفرة التالية بنية الطريقة getTableCellRendererComponent الخاصة بالصنف ColorRenderer:

```
public Component getTableCellRendererComponent(JTable table,
   Object value,
   boolean isSelected,
```

```
    boolean hasFocus,
    int row,
    int column) {

    Color bgrd = null;
    Color fgrd = null;

    if (isSelected) {
      // Preserve selection colors
      fgrd = table.getSelectionForeground();
      bgrd = table.getSelectionBackground();

    } else {
      // Set our colours
      fgrd = Color.pink;
      bgrd = Color.blue;

Component c =
            delegate.getTableCellRendererComponent(table, value, isSelected,
            hasFocus, row, column);
    // Set the component colours
    c.setBackground(bgrd);
    c.setForeground(fgrd);

    return c;
  }
```

منح الاستقلالية للمكوِّنات الفنية

لقد وجدنا سابقاً مقدار الفعالية الكبيرة التي يقدمها استخدام النموذج Decorator في بناء المكوِّنات الفنية (Renderers). إلا أن الألوان ضمن المكوِّن الفني تحد من قابلية الاستخدام المتكرر التي حاولنا سابقاً إضافتها إلى العنصر.

علينا أن نضع نصب أعيننا عندما نتعامل مع الجداول في هذه الفقرة، أن المعطيات داخل خلايا الجدول يجب أن تمتلك القدرة على التحكم بالألوان داخل تلك الخلايا (التحكم بأحداث التأثيرات).

يمكن تحقيق ذلك بسهولة من خلال توزيع المسؤوليات والمهام المطلوبة بين بنيتين منفصلتين. تمثل الأولى صنف المكوِّن الفني الذي يتحمل فقط مسؤولية تحقيق عملية التكوين الفني، دون التدخل في خصائص العنصر الهدف (يؤدي هذا الصنف الدور ذاته الذي تؤديه المكوِّنات الافتراضية في الحزمة swing). في حين يدعى

الثاني بالمزود (Provider) ويعد مسؤولاً عن تحديد الإعدادات الخاصة بالتطبيق من حيث المظهر والانطباع العام.

يؤدي هذا الفصل في المهام والمسؤوليات إلى احتفاظ المكوِّن الفني ببنيته العامة التي تمنحه خاصية الاستخدام المتكرر ضمن عدد من العناصر المختلفة. في حين يتولى المزود (Provider) مسؤولية تخصيص هذا الاستخدام بما يتوافق مع خواص وميزات العنصر الهدف ضمن واجهة المستخدم.

يمكن صياغة العلاقة السابقة بين المكوِّن والمزوِّد باستخدام مفهوم الواجهات في لغة Java. علينا أن نتصوَّر المزود بحيث يشكّل أداة الاتصـ ال التي تربط المكوِّن الفني بالعالم الخارجي. تقدم الشيفرة التالية تعريف المزود كواجهة ضمن لغة Java.

```
public interface ColorProvider{
 public Color getForeground(int row, int column);
 public Color getBackground(int row, int column);
        }
```

بناء على ذلك تأخذ بنية الصنف ColorRenderer شيئاً أكثر من التطور والتعقيد. حيث يتم تمرير الواجهة السابقة ColorProvider كوسيط لطريقة بناء الصنف السابق كما يلي:

```
public class ColorRenderer implements TableCellRenderer{
protected TableCellRenderer delegate;
protected ColorProvider provider;
public ColorRenderer(TableCellRenderer anotherRenderer, ColorProvider provider){
this.delegate = anotherRenderer;
this.provider = provider;
        }
        ...
        }
```

يتم استخدام الواجهة ColorProvider بعد ذلك ضمن الطريقة getTableCellRendererComponent لتحديد لون الخلية كما يلي:

```
public Component getTableCellRendererComponent(...) {
   Color bgrd = null;
   Color fgrd = null;
```

```
    if (isSelected) {
      fgrd = table.getSelectionForeground();
      bgrd = table.getSelectionBackground();
                    }
       else {
      // Adjust for columns moving around
      int mcol = table.convertColumnIndexToModel(column);
      // Get the colors from the provider
      fgrd = provider.getForeground(row, mcol);
      bgrd = provider.getBackground(row, mcol);
    }
Component c =delegate.getTableCellRendererComponent(table,   value, isSelected,
hasFocus, row, column);

    // Set the component colours
    c.setBackground(bgrd);
    c.setForeground(fgrd);
    return c;
  }
};
```

منح المكوِّنات الفنية القدرة على تحسس تغييرات المعطيات

تمثل الخطوة التالية في بناء النظام السابق، في تحقيق الواجهة ColorProvider بما يؤمن التحديد التلقائي للون الخلية تبعاً لقيمة المعطيات الموجودة ضمن الجدول.

لنفترض مثلاً أن لدينا كائناً من الصنف JTable يعمل على عرض الطلبات الواردة إلى وسيط عقاري (طلبات البيع والشراء). يضم كل سطر في الجدول إحدى هذه الطلبات ويضم كل من هذه الطلبات الأعمدة التالية: رمز الطلب، الكمية، السعر، جهة الطلب (البائع أو الشاري)، بالإضافة إلى عمود الحالة.

يتم تحديد لون السطر تبعاً لقيمة عمود الحالة في ذلك السطر. حيث يتم تحديد قيمة هذا العمود أثناء الزمن الحقيقي (زمن تنفيذ التطبيق). وبالتالي فإن أسطر الجدول يجب أن تتغير استجابة لأي تغير يطرأ على قيمة المعطيات في ذلك العمود.

يحتوي الصنف TableModel (وهو أحد الأصناف الخاصة بهذا التطبيق) على المعطيات التي تظهر ضمن JTable. حيث يتحكم هذا الصنف بالمعطيات وبعدد الأسطر وعدد الأعمدة وأسمائها بالإضافة إلى أنواع المعطيات في تلك الأعمدة.

يدير الصنف OrderTableModel مصفوفة الطلبات الواردة ويقوم بإبلاغ الجدول JTable بأي تحديث يطرأ على حالة أحد هذه الطلبات ضمن تلك المصفوفة وذلك باستخدام الطريقة UpdateOrderStatus().

ينقل البلاغ السابق العمل في هذه المرحلة إلى صنف المكوِّن الفني ColorRenderer الذي يحدد بدوره اللون الموافق للتغيير السابق.

يتطلَّب الانتقال إلى صنف المكوِّن الفني ColorRenderer تحقيق الواجهة ColorProvider التي تشكِّل (كما ذكرنا سابقاً) نقطة الاتصال مع صنف المكوِّن الفني. يتم داخل هذا التحقيق الاتصال بكائن الصنف OrderTableModel الذي يتولى إدارة مصفوفة الطلبات، وذلك للحصول على قيمة عمود الحالة (status) في السطر المطلوب.
يتم تحقيق الواجهة السابقة على شكل صنف داخلي عديم الاسم*(anonymous inner class) وبالتالي يتم تمرير هذا التحقيق كوسيط إلى طريقة بناء الصنف ColorRenderer عند تهيئة الكائنات المستنسخة عن هذا الصنف.

يعرض الشكل التالي الأسطر المطلوبة لهذا التحقيق:

```
OrderTableModel model = new OrderTableModel();
final static Color newColor = new Color(194, 245, 254);
final static Color filledColor = new Color(255, 221, 254);
final static Color cancelledColor = new Color(221, 254, 221);
// The provider here implemented as an anonymous inner class
ColorProvider provider = new ColorProvider() {
   public Color getForeground(int row, int column) {
     // Show the price column in red
     if (column == 2)
       return Color.red;
     else
       return Color.black;
   }
   public Color getBackground(int row, int column) {
     if (model.isNew(row))
       return Color.NewColor;
     else if (model.isFilled(row))
       return filledColor;
     else
       return cancelledColor;
   }
 };
```

يتم من خلال الشيفرة السابقة (ضمن تحقيق الواجهة ColorProvider) تلوين الأمامية في عمود الثمن باللون الأحمر، وذلك وفق التحقيق الخاص بالطريقةgetForeground. كما يتم تلوين خلفية أسطر الجدول بألوانٍ تُناسب نوع وحالة السطر باستخدام الطريقة getBackground.

تسجيل المكوِّنات الفنية المخصَّصة

لقد ذكرنا سابقاً أن الحزمة swing تضم المكوِّنات الفنية الافتراضية التي تعمل مع الصنف JTable. هذا يعني أن علينا توجيه هذا الصنف إلى استخدام المكوِّنات الفنية المخصصة التي قمنا ببنائها بدلاً من استخدام المكوِّنات الافتراضية في الحزمة swing. يتم هذا التوجيه بتسجيل مكوِّننا المخصَّص ملازماً للصنف JTable باستخدام الطريقة ()setDefaultRenderer، وهي إحدى طرق الصنف JTable.

إن نموذج Decorator المتبَّع في بناء المكوِّن الفني ColorRenderer الذي يعتمد على تغليف المكوِّن المخصَّص للمكوِّن المنشأ مسبقاً (للاستفادة من قدراته في عملية التنسيق) ، يعرض علينا إضافة بعض الخطوات إلى عملية التسجيل. وبالتالي فإننا بحاجة إلى كتابة طريقة إضافية للتحكم بهذه العملية. يجدر بنا الإشارة إلى أن عملية التسجيل هذه يجب أن تُعاد من أجل كل نوع من أنواع المعطيات المستخدمة ضمن الصنف OrderTableModel.

تقوم الطريقة registerRendererForClass بتسجيل المكوِّن الفني ColorRenderer كمكوِّن افتراضي للجدول المستخدم لعرض معطياتنا. تتحدد عملية التسجيل هذه بالصنف الممرر إلى هذه الطريقة كوسيط ثان.

يتم داخل هذه الطريقة أولاً استحصال المكوِّن الفني الافتراضي لنوع المعطيات المُمثَّل بالصنف الممرر إلى الطريقة. حيث يقوم المكوِّن المخصص فيما بعد بتغليف هذا المكوِّن (كما ذكرنا سابقاً). وفي النهاية يتم تسجيل المكوِّن المخصص كمكوِّن افتراضي باستخدام الطريقة SetDefaultRenderer.

توضّح الشيفرة التالية الطريقة registerRendererForClass:

```
private void registerRendererForClass(JTable table, Class klass) {
  // Get Default Renderer from the table
  DefaultTableCellRenderer defaultRenderer =
      DefaultTableCellRenderer)
  table.getDefaultRenderer(klass);
  // Wrap(Decorate) the color renderer around the default renderer
  TableCellRenderer colorRenderer = new
    ColorRenderer(defaultRenderer, provider);
  // Register the color Renderer with the JTable
  table.setDefaultRenderer(klass, colorRenderer);
}
```

يتمتع الصنف الداخلي عديم الاسم ببنية الصنف العادي غير أنه لا يملك اسماً، كما أنه يُعرف داخل أصنافٍ أخرى.

كما نعرض أيضاً الأسطر المطلوبة لاستخدام الطريقة registerRendererForClass في تسجيل الصنف ColorRenderer كمكوِّنٍ فني لجميع خلايا الجدول التي تحتوي قيماً تنتمي إلى نوعي المعطيات Double (الأعداد المضاعفة)، String (السلاسل النصيّة):

```
JTable table = new JTable (model);
registerRendererForClass (table, String.class);
registerRendererForClass (table, Double.class);
```

أخيراً وبعد تطبيق كل الخطوات السابقة لاستخدام المكوِّن ColorRenderer يظهر الشكل -18 للجدول حتى هذه المرحلة كما يلي:

شكل -18 :

إضافة اللمعان إلى الخلايا والسطور

يمكن الاستفادة من نموذج مكوِّن/ مزود المتَّبع في إنجاز المكوِّن ColorRenderer ، في بناء مكوِّنات أخرى تتحكم بكل من الخطوط والحواف الخارجية أو أي نوع آخر من التأثيرات المرئية التي تُضاف عادة إلى الخلايا والسطور داخل الجداول.

سنقوم في هذه الفقرة بإضافة تأثير اللمعان إلى أسطر المثال السابق (طلبات الوسيط التجاري) عند حدوث أ ي تغيير في قيمة العمود الحاوي على السعر لأي من أسطر الطلبات في ذلك الجدول.

لن تكون طريقة العمل في هذه الفقرة مختلفة عن الفقرة السابقة. حيث تقوم ببناء الصنف FlashColorRenderer بالإضافة إلى الواجهة FlashProvider. سنقوم أيضاً باستخدام النموذج Decorator في بناء الصنف FlashColorRenderer. سنقوم أولاً ببناء الواجهة FlashProvider كما يلي:

```
public interface FlashProvider {
  public boolean isFlashOn(int row, int column);

};
```

تقبل طريقة البناء (constructor) للصنف FlashColorRenderer وسيطين. يمثل الأول المكوِّ ن المسبق التعريف والمراد تغليفه بواسطة المكوِّن المخصص. في حين يمثل الثاني الواجهة FlashProvider. توضح الشيفرة التالية الجزء الأول من بنية المكوِّن FlashColorRenderer:

```
public class FlashColorRenderer implements TableCellRenderer {

protected TableCellRenderer delegate;
protected FlashProvider provider;
public FlashColorRenderer(TableCellRenderer anotherRenderer,
   FlashProvider provider) {
   this.delegate = anotherRenderer;
   this.provider = provider;
 }
...
};
```

يتم استخدام الواجهة FlashProvider ضمن الطريقة getTableCellRendererComponent لمعرفة حالة اللمعان المطبقة حالياً على الخلية، قبل إضافة حالة اللمعان إلى هذه الخلية. توضح الشيفرة التالية بنية هذه الطريقة:

```
public Component getTableCellRendererComponent(...) {
// Get the component from the delegate
Component c =delegate.getTableCellRendererComponent(table,
                    value, isSelected, hasFocus, row, column);
// Convert view column to model column
  int mcol = table.convertColumnIndexToModel(column);
  // invert the colours to flash
  if (provider.isFlashOn(row, mcol)) {
   Color bgrd = c.getForeground();
   Color fgrd = c.getBackground();
   c.setBackground(bgrd);
   c.setForeground(fgrd);
  }
 return c;
}
```

الخطوة الجديدة هنا هي تطبيق النموذج Decorator للمرة الثانية بهدف تغليف الصنف ColorRenderer بواسطة الصنف FlashColorRenderer. حيث يشكِّل كل غلاف يُضاف إلى النموذج السابق طبقة جديدة أكثر تطوراً تضيف سلوكها

الخاص مستفيدة من إمكانيات وسلوك الطبقة السابقة. يمكن بهذا الشكل إضافة عدد غير منته من الطبقات من خلال عملية التطوير المستمرة.

تعرض الطريقة registerRendererForClass عملية تسجيل المكوِّن الفني FlashRenderer كمكوِّنٍ افتراضي للجدول وذلك بعد تغليف هذا المكون للمكوِّن ColorRenderer المسبق التعريف. الطريقة السابقة موضحة في الشيفرة التالية:

```
private void registerRendererForClass(JTable table,Class klass) {
   // Get Default Renderer from the table
DefaultTableCellRenderer defaultRenderer =
  (DefaultTableCellRenderer)table.getDefaultRenderer(klass);
// Wrap the color renderer around the default renderer
  TableCellRenderer colorRenderer = new
  ColorRenderer(defaultRenderer, provider);
// Wrap the flash renderer around the colour renderer
  TableCellRenderer flashRenderer = new
    FlashColorRenderer(colorRenderer,flashProvider);
// Register our flash renderer with the table
  table.setDefaultRenderer(klass, flashRenderer);
  }
```

تتمثل الخطوة الأخيرة في عملية إضفاء ميزة اللمعان إلى خلايا الجدول، في إنشاء الصنف TableFlasher المحقّق للواجهة FlashProvider للتحكم بحالة اللمعان تلك ضمن خلايا الجدول الهدف. كما أن جزء الشيفرة المسؤول عن تحديث قيمة السعر داخل التطبيق يجب أن يتضمن عملية إعلام الصنف TableFlasher بهذا التحديث حال حصوله.

تعتبر الحزمةswing ذات مسلك تنفيذ وحيد (singlethreaded()).لذا لا بد من استخدام الفعالية invokeLater() عند تنفيذ الشيفرة التي تحتوي على أي اتصالٍ بعناصر واجهة المستخدم (UI) خارج سلك التنفيذ الخاص بالأحداث داخل الحزمة swing.

تعرض الشيفرة التالية الأسطر اللازمة داخل التطبيق، لاستخدام الصنف TableFlasher وإضافة حالة اللمعان إلى خلايا الجدول في الحالات المطلوبة.

```
TableFlasher flashProvider = new TableFlasher(table);
...
// In a background thread, pick a row
```

```
int row = (int) (Math.random() * model.getRowCount());
// Generate a new price
Double price = new Double(Math.random() * 100);
// Update the OrderTableModel
model.updateOrderPrice(row, price);
// Flash the price cell
flashProvider.flashCell(row, 2);
```

حيث يتم بداية تعريف كائنٍ جديد الصنف TableFlasher المسؤول عن إضافة حالة اللمعان ثم يتم في مسلك التنفيذ الخاص من الخلفية الانتقاء العشوائي للسطر الذي سيحتوي على التغيير في قيمة عمود الثمن الموجبة لإضافة حالة اللمعان. يتم بعد ذلك توليد القيمة الجديدة التي ستخزن في الخلية الحاوية على الثمن ضمن السطر السابق، ومن ثم تعديل قيمة هذه الخلية داخل الجدول باستخدام الطريقة UpdateOrderPrice.

وأخيراً يُستخدم كائن الصنف TableFlasher لإضافة حالة اللمعان إلى الخلية الهدف. يعرض الشكل -18 الشكل الناتج عن تطبيق المكوِّن السابق على جدول الطلبات التجارية:

شكل -18:

الحدود ثلاثية الأبعاد

سنقوم الآن بإضافة آخر تأثير في بحثنا هذا إلى الجدول JTable. يمكن منح الحدود الخارجية لخلايا هذا الجدول شكل ثلاثي الأبعاد بإنشاء المكوِّن الفني

FlashBorderRenderer. كما في المكوِّن السابق يستخدم هذا المكوِّن الواجهة FlashProvider لتحديد حالة اللمعان الحالية للخلية من خلال الشيفرة الخاصة بالطريقة getTableCellRendererComponent ، ومن ثم رسم الحدود الخارجية لتلك الخلية على شكل حدودٍ مرفوعة مما يعطيها شكلاً ثلاثي الأبعاد.

تعرض الأجزاء التالية من الشيفرة البنية الأساسية للصنف FlashBorderRenderer.

```
public class FlashBorderRenderer {
protected static final Border raised = BorderFactory.createRaisedBevelBorder();
protected static final Border noFocusBorder = new EmptyBorder(1, 1, 1, 1);
  protected static final Border focusBorder =
UIManager.getBorder("Table.focusCellHighlightBorder");
...
public Component getTableCellRendererComponent(...) {
// Get the component from the delegate
  Component c =  delegate.getTableCellRendererComponent(table, value,
  isSelected, hasFocus, row, column);
// Convert view column to model column
    int mcol = table.convertColumnIndexToModel(column);
// invert the colours to flash
    JComponent jc = (JComponent) c;
// If an abstract button e.g JCheckBox then set
// border painted to true
    if (jc instanceof AbstractButton) {
      ((AbstractButton) jc).setBorderPainted(true);
                                        }
    if (provider.isFlashOn(row, mcol)) {
                jc.setBorder(raised);
                                              }
    else {
      if (hasFocus)
        jc.setBorder(focusBorder);
      else
        jc.setBorder(noFocusBorder);
    }
  return c;
  }
```

يتم بداية تعريف الكائنات الحاوية على تنسيقات الحدود الخارجية المطلوبة في الحالات المختلفة التي تمر بها الخلية الهدف. يتم تعريف هذه الكائنات على أنها

صنفية (نسخة واحدة لكل كائنات الصنف) باستخدام الكلمة المحجوزة static، وغير قابلة للتغيير باستخدام الكلمة Final. يتم بعد ذلك الحصول على العنصر C المستهدف بالتكوين الفني باستخدام النائب delegate. يتم أيضاً الحصول على ترتيب العمود ضمن الطراز الخاص بالجداول، وذلك باستخدام الطريقة ConvertColumnIndexToModel. يعتبر هذا الرقم (الترتيب) المحدد الحقيقي لموقع العمود داخل ا لجدول بحيث يستخدم في تحديد موقع الخلية المطلوبة داخل الجدول.

يستخدم السطر التالي داخل الشيفرة السابقة لتحويل العنصر C المشتق من الصنف Component إلى العنصر jc المشتق من الصنف JComponent تمهيداً لنقل حالة الألوان إلى حالة اللمعان.

تُستخدم العبارة الشرطية if لتأكيد رسم الحدود الخارجية في الحالة التي يكون فيها العنصر jc مشتقاً من صنف الزر التجريدي AbstractButton الذي تتفرع عنه العديد من العناصر التي تستخدم في واجهة المستخدم ضمن الحزمة swing.

أخيراً يتم رسم الحدود الخارجية للخلية الهدف وفق التنسيق الموافق للحالة الحالية لهذه الخلية.

لا بد أن نذكر أخيراً بضرورة تسجيل المكوِّن السابق كمكوِّن افتراضي للجدول، وذلك بإنشاء الطريقةregisterRendererForClass بشكل مماثلٍ تماماً لطرق التسجيل المكتوبة في الفقرات السابقة.

18.11.19 حلول لمشاكل تطبيق JDBC المتوقعة

تصادفك الكثير من المشـ اكل أثناء برمجة تطبيقات JDBC ، ففي التطبيق Chp18_1 توقعنا حدوث الكثير من استثناءات SQLException أثناء تنفيذ أي خطوة من خطوات التطبيق. لذلك فقد تم دعم واجهة برمجة التطبيقاتJDBC بمقدرة عالية في معالجة الأخطاء. كما أن معظم تصريحات مشغلات JDBC لا تغفل هذه النقطة.

حتى تتفهم معالجة الأخطاء بشكل جيد، يجب أن تمتلك حدساً يمكنك من توقع مكان حدوث الخطأ. يأتي ذلك من فهم مكونات تطبيق JDBC ومراحل العمل المتبعة في تطويره.

بشكل عام ستمر بالمراحل التالية: ستنشأ اتصالك إلى مخدم قاعدة البيانات عبر الشبكة، ولا ن خفي عليك أن برمجة تطبيقات زبون- مخدم ليست بالسهلة وتحتوي على احتمالات كثيرة للإخفاق.

ستقوم بعدها بإرسال استعلامات SQL إلى قاعدة البيانات، الأمر الذي يتطلب منك تفهماً تاماً لمخطط قاعدة البيانات المُستَعْلَمَة، لتستطيع تحديد جدول الاستعلام المناسب، وتحديد أسماء الأعمدة المطلوبة أيضاً.

كما تحتاج لمعرفة أنواع معطيات الأعمدة لتستطيع استخدام المنهج ()getXXX المناسب لاستعادة المعطيات من الكائن ResultSet (أو البيانات الناتجة)، وأي مشكلة تحدث خلال أية عملية من العمليات السابقة يتم رمي الاستثناء SQLException . يقدم لك الجدول 7-18 بعض النصائح الأولية لمعالجة الأخطاء المتوقعة في تطبيقJDBC.

جدول 7-18

المشكلة	الحل
لا يمكن ترجمة أو تنفيذ التطبيق	- تأكد من أن مسار الأصناف يحتوي على مسار مشغل JDBC المستخدم. - تأكد من أنك قد قمت بكتابة اسم المشغل بالشكل الصحيح. فبعض أسماء المشغلات طويلة واحتمال الخطأ في كتابتها متوقع.
لا أستطيع الاتصال بالمخدم	- تأكد من اسم المخدم. - تأكد من أنك قمت بتأسيس اتصال الشبكة. - تأكد من أنك تستخدم صيغة JDBC URL الصحيحة من أجل مشغلك. - تأكد من اسم قاعدة البيانات ورقم المنفذ.
فشل في أوامر SQL	- تأكد من الصياغة القواعدية لعبارة SQL. - تأكد من أنك تستخدم اسم جدول صحيح. - تأكد من أسماء الأعمدة. - تأكد من أنك تمتلك صلاحيات لتنفيذ الاستعلام. - تأكد من أن عبارة SQL لا تتعرض لقيود تكامل

	المعطيات في قاعدة البيانات.

تمارين الفصل

1. عدد أنواع قواعد البيانات ؟
2. أشرح مكونات قاعدة البيانات ؟
3. عدد مميزات وعيوب قاعد البيانات ؟
4. ما هي أفضل التعامل مع البيانات هل الملفات أم قواعد البيانات . ولماذا ؟
5. لقد تعلمت في هذا الفصل كيفية ربط قاعدة البيانات أكسس ببرنامجك . الآن قم بربط مشغل البيانات من نوع اوراكل ببرنامجك ؟

.6

الملاحق

Appendix

A: أسبقية العوامل Operator Precedence :-

Operators are shown in decreasing order of precedence from top to bottom (Fig. A.1).

Figure A.1. Operator precedence chart.

Operator	Description	Associativity
++	unary postfix increment	right to left
--	unary postfix decrement	
++	unary prefix increment	right to left
--	unary prefix decrement	
+	unary plus	
-	unary minus	
!	unary logical negation	
~	unary bitwise complement	
(type)	unary cast	
*	Multiplication	left to right
/	Division	
%	Remainder	
+	addition or string concatenation	left to right
-	Subtraction	
<<	left shift	left to right
>>	signed right shift	
>>>	unsigned right shift	

Figure A.1. Operator precedence chart.		
Operator	Description	Associativity
<	less than	left to right
<=	less than or equal to	
>	greater than	
>=	greater than or equal to	
instanceof	type comparison	
==	is equal to	left to right
!=	is not equal to	
&	bitwise AND	left to right
	boolean logical AND	
^	bitwise exclusive OR	left to right
	boolean logical exclusive OR	
|	bitwise inclusive OR	left to right
	boolean logical inclusive OR	
&&	conditional AND	left to right
||	conditional OR	left to right
?:	Conditional	right to left
=	Assignment	right to left
+=	addition assignment	
-=	subtraction assignment	
*=	multiplication assignment	
/=	division assignment	
%=	remainder assignment	

Figure A.1. Operator precedence chart.		
Operator	Description	Associativity
&=	bitwise AND assignment	
^=	bitwise exclusive OR assignment	
|=	bitwise inclusive OR assignment	
<<=	bitwise left shift assignment	
>>=	bitwise signed-right-shift assignment	
>>>=	bitwise unsigned-right-shift assignment	

B: الكلمات المفتاحية Keywords and Reserved Word:-

Table Java keywords.				
Java Keywords				
abstract	assert	boolean	break	Byte
case	catch	char	class	Continue
default	do	double	else	Enum
extends	final	finally	float	For
if	implements	import	instanceof	Int
interface	long	native	new	Package
private	protected	public	return	Short
static	strictfp	super	switch	Synchronized
this	throw	throws	transient	Try
void	volatile	while		
Keywords that are not currently used				
const	goto			

إن الكلمتين goto, const هي كلمات مفتاحية محجوزة للغة C++ (لكنها غير مستخدمة حالياً) وموجودة في Java.

C : شفرة المسح ASCII Character Set

19.1.1 رموز الشفرة الأمريكية المعيارية لتبادل المعلومات ASCII Code

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

19.1.2 رموز الشفره الأمريكية الموسعة

128	Ç	144	É	160	á	176	▒	193	┴	209	╤	225	ß	241	±
129	ü	145	æ	161	í	177	▒	194	┬	210	╥	226	Γ	242	≥
130	é	146	Æ	162	ó	178	▓	195	├	211	╙	227	π	243	≤
131	â	147	ô	163	ú	179	│	196	─	212	╘	228	Σ	244	⌠
132	ä	148	ö	164	ñ	180	┤	197	┼	213	╒	229	σ	245	⌡
133	à	149	ò	165	Ñ	181	╡	198	╞	214	╓	230	µ	246	÷
134	å	150	û	166	ª	182	╢	199	╟	215	╫	231	τ	247	≈
135	ç	151	ù	167	º	183	╖	200	╚	216	╪	232	Φ	248	°
136	ê	152	_	168	¿	184	╕	201	╔	217	┘	233	Θ	249	·
137	ë	153	Ö	169	_	185	╣	202	╩	218	┌	234	Ω	250	·
138	è	154	Ü	170	¬	186	║	203	╦	219	▉	235	δ	251	√
139	ï	156	£	171	½	187	╗	204	╠	220	▄	236	∞	252	_
140	î	157	¥	172	¼	188	╝	205	═	221	▌	237	ϕ	253	²
141	ì	158	_	173	¡	189	╜	206	╬	222	▐	238	ε	254	■
142	Ä	159	ƒ	174	«	190	╛	207	╧	223	▀	239	∩	255	
143	Å	192	└	175	»	191	┐	208	╨	224	α	240	≡		

19.1.3 رموز شفرة EBCDIC

Dec	Hx	Oct	Char		Dec	Hx	Oct	Char	Dec	Hx	Oct	Char	Dec	Hx	Oct	Char
0	0	000	nul	(Null)	65	41	101		130	82	202	b	195	c3	303	C
1	1	001	soh	(Start of Heading)	66	42	102		131	83	203	c	196	c4	304	D
2	2	002	stx	(Start of Text)	67	43	103		132	84	204	d	197	c5	305	E
3	3	003	etx	(End of Text)	68	44	104		133	85	205	e	198	c6	306	F
4	4	004	pf	(Punch Off)	69	45	105		134	86	206	f	199	c7	307	G
5	5	005	ht	(Horizontal Tab)	70	46	106		135	87	207	g	200	c8	310	H
6	6	006	lc	(Lower Case)	71	47	107		136	88	210	h	201	c9	311	I
7	7	007	del	(Delete)	72	48	110		137	89	211	i	202	ca	312	
8	8	010	ge		73	49	111		138	8a	212		203	cb	313	
9	9	011	rlf		74	4a	112	¢	139	8b	213		204	cc	314	
10	a	012	smm	(Start of Manual Message)	75	4b	113	.	140	8c	214		205	cd	315	
11	b	013	vt	(Vertical Tab)	76	4c	114	>	141	8d	215		206	ce	316	
12	c	014	ff	(Form Feed)	77	4d	115	(	142	8e	216		207	cf	317	
13	d	015	cr	(Carriage Return)	78	4e	116	+	143	8f	217		208	d0	320	}
14	e	016	so	(Shift Out)	79	4f	117	|	144	90	220		209	d1	321	J
15	f	017	si	(Shift in)	80	50	120	&	145	91	221	j	210	d2	322	K
16	10	020	dle	(Data Link Escape)	81	51	121		146	92	222	k	211	d3	323	L
17	11	021	dc1	(Device Control 1)	82	52	122		147	93	223	l	212	d4	324	M
18	12	022	dc2	dc2 (Device Control 2)	83	53	123		148	94	224	m	213	d5	325	N
19	13	023	tm	(Tape Mark)	84	54	124		149	95	225	n	214	d6	326	O
20	14	024	res	(Restore)	85	55	125		150	96	226	o	215	d7	327	P
21	15	025	nl	(New Line)	86	56	126		151	97	227	p	216	d8	330	Q
22	16	026	bs	(Backspace)	87	57	127		152	98	230	q	217	d9	331	R
23	17	027	il	(Idle)	88	58	130		153	99	231	r	218	da	332	
24	18	030	can	(Cancel)	89	59	131		154	9a	232		219	db	333	
25	19	031	em	(End of Medium)	90	5a	132	!	155	9b	233		220	dc	334	
26	1a	032	cc	(Cursor Control)	91	5b	133	$	156	9c	234		221	dd	335	
27	1b	033	cu1	(Customer Use 1)	92	5c	134	*	157	9d	235		222	de	336	
28	1c	034	ifs	(Interchange File Separator)	93	5d	135	)	158	9e	236		223	df	337	
29	1d	035	igs	(Interchange Group Separator)	94	5e	136	;	159	9f	237		224	e0	340	\
30	1e	036	irs	(Interchange Record	95	5f	137		160	a0	240		225	e1	341	
31	1f	037	ius	(Interchange Unit Separator)	96	60	140	-	161	a1	241	~	226	e2	342	S
32	20	040	ds	(Digit Select)	97	61	141	/	162	a2	242	s	227	e3	343	T
33	21	041	sos	(Start of Significance)	98	62	142		163	a3	243	t	228	e4	344	U
34	22	042	fs	(Field Separator)	99	63	143		164	a4	244	u	229	e5	345	V
35	23	043			100	64	144		165	a5	245	v	230	e6	346	W
36	24	044	byp	(Bypass)	101	65	145		166	a6	246	w	231	e7	347	X
37	25	045	lf	(Line Feed)	102	66	146		167	a7	247	x	232	e8	350	Y
38	26	046	etb	(End of Transmission Block)	103	67	147		168	a8	250	y	233	e9	351	Z
39	27	047	esc	(Escape)	104	68	150		169	a9	251	z	234	ea	352	
40	28	050			105	69	151		170	aa	252		235	eb	353	
41	29	051			106	6a	152	|	171	ab	253		236	ec	354	
42	2a	052	sm	(Set Mode)	107	6b	153	,	172	ac	254		237	ed	355	
43	2b	053	cu2	(Customer Use 2)	108	6c	154	%	173	ad	255		238	ee	356	
44	2c	054			109	6d	155		174	ae	256		239	eF	357	
45	2d	055	enq	(Enquiry)	110	6e	156	<	175	af	257		240	f0	360	0
46	2e	056	ack	(Acknowledge)	111	6f	157	?	176	b0	260		241	f1	361	1
47	2f	057	bel	(Bell)	112	70	160		177	b1	261		242	f2	362	2
48	30	060			113	71	161		178	b2	262		243	f3	363	3
49	31	061			114	72	162		179	b3	263		244	f4	364	4
50	32	062	syn	(Synchronous Idle)	115	73	163		180	b4	264		245	f5	365	5
51	33	063			116	74	164		181	b5	265		246	f6	366	6
52	34	064	pn	(Punch On)	117	75	165		182	b6	266		247	f7	367	7
53	35	065	rs	(Reader Stop)	118	76	166		183	b7	267		248	f8	370	8
54	36	066	uc	(Upper Case)	119	77	167		184	b8	270		249	f9	371	9
55	37	067	eot	(End of Transmission)	120	78	170		185	b9	271		250	fa	372	|
56	38	070			121	79	171	`	186	ba	272		251	fb	373	
57	39	071			122	7a	172	:	187	bb	273		252	fc	374	
58	3a	072			123	7b	173	#	188	bc	274		253	fd	375	
59	3b	073	cu3	(Customer Use 3)	124	7c	174	@	189	bd	275		254	fe	376	
60	3c	074	dc4	(Device Control 4)	125	7d	175	'	190	be	276		255	ff	377	eo
61	3d	075	nak	(Negative Acknowledge)	126	7e	176	=	191	bf	277					
62	3e	076			127	7f	177	"	192	c0	300	{				
63	3f	077	sub	(Substitute)	128	80	200		193	c1	301	A				
64	40	100	Sp	(Space)	129	81	201	a	194	c2	302	B				

D: معدلات الوصول:

يتم استخدام المعدلات مع الأصناف ومع أعضاء الصنف (المناهج والمعطيات)، لكن لا يمكن استخدام المعدل final إلا مع المتحولات المحلية ضمن المناهج. يطلق على المعدلات التي من الممكن تزويدها للأصناف بمعدلات الصنف (class modifier) ، ويطلق على المعدلات التي من الممكن تزويدها إلى المناهج بمعدلات المنهج ، ويطلق على المعدلات التي من الممكن تزويدها إلى حقل البيانات بمعدلات البيانات.

يلخص الجدول التالي معدلات Java.

المعدل	صنف	منهج	بيانات	الوصف
Default	√	√	√	تكون الأصناف والمناهج وحقول البيانات قابلة للرؤية في هذه الحزمة.
Public	√	√	√	تكون الأصناف والمناهج وحقول البيانات قابلة للرؤية لجميع البرامج حتى الموجودة في حزم أخرى.
Private		√	√	تكون الأصناف أوحقول البيانات تكون قابلة للرؤية في هذا الصنف.
Protected		√	√	تكون المناهج وحقول البيانات قابلة للرؤية في هذه الحزمة. وفي الاصناف الأبناء لهذا الصنف والموجودة في أي حزمة.
Static		√	√	تعرف منهج صنف أو حقل بيانات.
Final	√	√	√	لا يمكن وراثة الصنف النهائي. ولا يمكن تعديل المنهج النهائي في الصنف الابن. إن حقل البيانات النهائي هو عبارة عن ثابت.
Abstract	√	√		يجب وراثة الصنف المجرد ويجب تنفيذ المنهج المجرد في صنف ابن ثابت.
Native			√	يدل على أنه يتم تنفيذ المنهج باساخدام لغة أخرى غير Java.
Synchronized		√		يمكن لمسلك واحد فقط في نفس اللحظة

				تنفيذ هذا المنهج.
Strictfp	√	√		استخدام حسابات الفاصلة العائمة القاطعة لتضمن الحصول على نفس نتيجة التقييم في جميع الات Java الإفتراضية.

E: أنواع البيانات Primitive Types

Figure Java primitive types.			
Type	Size in bits	Values	Standard
boolean		true or false	
[Note: A boolean's representation is specific to the Java Virtual Machine on each platform.]			
char	16	'\u0000' to '\uFFFF' (0 to 65535)	(ISO Unicode character set)
byte	8	128 to +127 (2^7 to 2^7 1)	
short	16	32,768 to +32,767 (2^{15} to 2^{15} 1)	
int	32	2,147,483,648 to +2,147,483,647 (2^{31} to 2^{31} 1)	
long	64	9,223,372,036,854,775,808 to +9,223,372,036,854,775,807 (2^{63} to 2^{63} 1)	
float	32	Negative range: 3.4028234663852886E+38 to 1.40129846432481707e45 Positive range: 1.40129846432481707e45 to 3.4028234663852886E+38	(IEEE 754 floating point)
double	64	Negative range: 1.7976931348623157E+308 to 4.94065645841246544e324 Positive range: 4.94065645841246544e324 to 1.7976931348623157E+308	(IEEE 754 floating point)

F: النظام العددي

توجد أنظمة عديدة ، وتختلف هذه الأنظمة عن بعضها باختلاف عدد الرموز التي تمثل الأعداد الشفرية فيها، ولقد استخدم الإنسان في معاملاته اليومية نظام الأعداد العشرية، وقد استخدمت الحاسبات الآلية هذا النظام في بدء نشأتها. أما الآن فمعظم الحاسبات تستخدم الأعداد الثنائية، الأمر الذي يجعل الحاسبات سهلة التصميم لسهولة وسرعة تنفيذ العمليات الحسابية بها.

ويتطرق هذا الملحق أنظمة الأعداد المعروفة:

- نظام العد العشري Decimal System.
- النظام الثنائي Binary System.
- النظام الثماني Octal System.
- النظام السادس عشر Hexa decimal System.

19.2 النظام العشري Decimal System :

يعتبر النظام العشري أكثر أنظمة العد استعمالاً من قبل الإنسان، وقد سمي بالعشري لأنه يتكون من عشرة أرقام هي (9 ..0)و التي بدورها تشكل أساس نظام العد العشري.
وبشكل عام يمكن القول أن أساس أي نظام عد Base يساوي عدد الأرقام المستعملة لتمثيل الأعداد فيه، وهو يساوي كذلك أكبر رقم في النظام مضافاً إليه واحد.
تمثل الأعداد في النظام العشري بواسطة قوى الأساس 10 وهذه تسمي بدورها أوزان خانات العدد ومثال ذلك العدد العشري :N=7129.45

حيث يمكن كتابته على النحو التالي :

$$N= 7x10^{3}+1x10^{2}+2x10^{1}+9 \times 10^{0}+4x10^{-1}+5x10^{-2}$$

19.3 النظام الثنائي Binary System :

إن الأساس المستعمل في النظام الثنائي هو 2 ويتكون هذا النظام من رقمين فقط هما 0 و1 ويسمى كل منهما رقماً ثنائياً. Binary Digit ولتمثيل كل من الرقمين 0 و 1 فأنه لا يلزم إلا خانة واحدة، ولهذا السبب أصبح من الشائع أطلاق اسم بت Bit على الخانة التي يحتلها الرقم داخل العدد الثنائي.

19.3.1 التحويل من النظام الثنائي إلى النظام العشري :

Binary to Decimal Conversion

لتحويل أي عدد ثنائي إلى مكافئه العشري فإنه يجب علينا استعمال قانون التمثيل الموضعي للأعداد. و ينطبق هذا القانون عندما يكون الرقم الثنائي صحيحاً أو كسراً مع مراعاة أن أساس نظام العد هنا هو 2.

$$N= a_nR^n + a_{n-1}R^{n-1}+....+ a_0R^0 + a_{-1}R^{-1}+....+ a_{-m}R^{-m}$$

مثال :

حول العدد الثنائي التالي إلى مكافئه العشري :

$$(11001.011)_2 \longrightarrow (?)_{10}$$

4	3	2	1	0		-1	-2	-3
1	1	0	0	1	.	0	1	1

$$N = 1x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0 + 0x2^{-1} + 1x2^{-2} + 1x2^{-3}$$

$$N = 1x16 + 1x8 + 1x1 + \frac{1}{4} + \frac{1}{8} = 25.625$$

$$(11001.011)_2 = (25.625)_{10}$$

19.3.2 تحويل الأعداد من النظام العشري إلى الثنائي :

Number Conversion from decimal to binary System

- تحويل الأعداد العشرية الصحيحة الموجبة:

لتحويل أي عدد صحيح موجب من النظام العشري إلى الثنائي نستعمل طريقة الباقي Remainder Method لموضحة كالآتي:

1. أقسم العدد العشري على الأساس 2.
2. أحسب باقي القسمة الذي يكون أما 1 أو 0.
3. أقسم ناتج القسمة السابق على الأساس 2 كما في خطوة (1).

4. أحسب باقي القسمة كما في خطوة (2).
5. استمر في عملية القسمة وتحديد الباقي حتى يصبح خارج القسمة الصحيح صفراً.
6. العدد الثنائي المطلوب يتكون من أرقام الباقي مقروءة من الباقي الأخير إلى الأول (لاحظ أن الباقي الأول يمثل LSD بينما يمثل الباقي الأخير MSD).

مثال :
لتحويل الرقم 12 من النظام العشري إلى الثنائي نتبع الآتي:

	ناتج القسمة	الباقي	
.1	12 ÷2 =6	0	الخانة الأدنى منزلة LSD
.2	6÷2 =3	0	
.3	3÷2 =1	1	
.4	1÷2 =0	1	الخانة الأعلى منزلة MSD

إنهاء القسمة

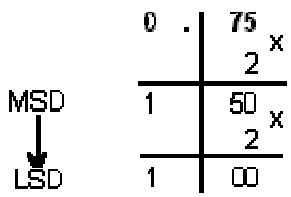
فيكون الناتج (من أسفل إلى أعلى ومن اليسار إلى اليمين): $(12)_{10} = (1100)_2$

- تحويل الكسر العشري إلى ثنائي:

لتحويل الكسر العشري إلى مكافئة الثنائي نضرب الكسر في الأساس 2 عدداً معيناً من المرات حتى نحصل على ناتج ضرب يساوي صفراً أو حتى نحصل على الدقة المطلوبة.

مثال :

لتحويل الكسر العشري $(0.75)_{10}$ إلى مكافئة الثنائي:

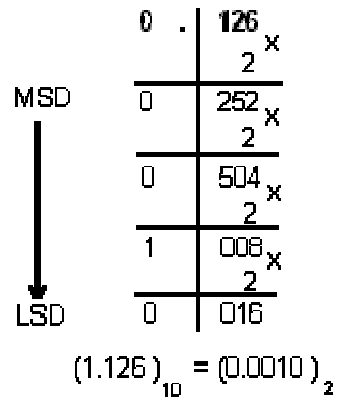


$(0.75)_{10} = (0.11)_2$

فيكون الناتج (من أعلى إلى أسفل ومن اليسار إلى اليمين) : (0.11).

مثال:

لتحويل الكسر العشري 0.126 إلى مكافئة الثنائي بدقة تصل إلى أربعة أرقام ثنائية:



فيكون الناتج (من أعلى إلى أسفل ومن اليسار إلى اليمين) (0.0010) .

- تحويل العدد العشري الكسرى:

يتم تحويل كل جزء على حدة ثم تضم النتائج مع بعض لتعطي النتيجة المطلوبة.

مثال:

تحويل العدد العشري 10.15 إلى مكافئة الثنائي:

الحل:

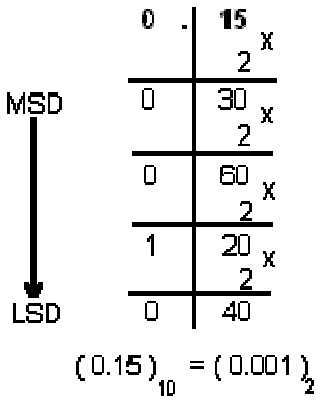
1. حول الجزء الصحيح إلى مكافئه الثنائي:

	ناتج القسمة	الباقي	
1.	10 ÷2 =5	0	الخانة الأدنى منزلة LSD
2.	5÷2 =2	1	
3.	2÷2 =1	0	
4.	1÷2 =0	1	الخانة الأعلى منزلة MSD

إنهاء القسمة

يكون الناتج (من أعلى إلى أسفل ومن اليسار إلى اليمين) : $(10)_{10} \longrightarrow (1010)_2$

2. ثم نحول الجزء الكسري كما يلي:



الناتج الكلي: $(10.15)_{10} = (1010.001)_2$

19.3.3 إجراء العمليات الحسابية على الأعداد الثنائية الموجبة:

يمكن إجراء العمليات الحسابية من جمع و طرح و ضرب وقسمة كما هو الحال في النظام العشري مع مراعاة أن أساس النظام المستعمل هنا هو 2.

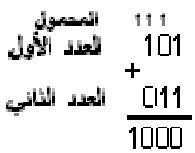
•عملية الجمع : لو أخذنا عددين ثنائيين A,B وكان كل منهما يتكون من خانة واحدة فقط Bit ، وبما أن كل خانة يمكن أن تكون أما 0 أو 1 فإنه يوجد للعددين معاً أربع احتمالات كالآتي:

A	B	المجموع S= A+B	الفيض Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

أما إذا كانت الأعداد الثنائية مكونة من أكثر من خانة واحدة فإن عملية الجمع تنفذ بنفس طريقة الجمع في النظام العشري مع مراعاة أن أساس النظام العد المستعمل هو 2.

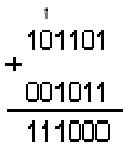
مثال(1):

أجمع العددين الثنائيين : $(101)_2 + (011)_2 = (\ ?\ )_2$



الناتج : $(101)_2 + (011)_2 = (1000)_2$

مثال(2): جمع العددين الثنائيين: $(101101)_2 + (1011)_2 = (\ ?\ )_2$

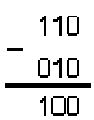


الناتج : $(101101)_2 + (1011)_2 = (111000)_2$

•عملية الطرح (إذا كان المطروح أقل من المطروح منه): لو أخذنا عددين ثنائيين A,B وكان كل منهما يتكون من خانة واحدة فقط، فإنه توجد الاحتمالات التالية لعملية الطرح تكون كالآتي:

A	B	الفرق D=A-B	المستقرض Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

مثال(1): اطرح العددين الثنائيين $(110)_2 - (010)_2 = (\ ?\ )_2$



الناتج : $(110)_2 - (010)_2 = (100)_2$

مثال(2): اطرح العددين الثنائيين $(1010)_2 - (111)_2 = (?)_2$

$$\begin{array}{r} 1010 \\ -\ 0111 \\ \hline 0011 \end{array}$$

الناتج : $(1010)_2 - (111)_2 = (011)_2$

•عملية الضرب:

مثال(1)ما هو ناتج ضرب العددين الثنائيين $(101)_2 \times (10)_2 = (?)_2$

$$\begin{array}{r} 101 \\ \times\ 10 \\ \hline 000 \\ 101 \\ \hline 1010 \end{array}$$

الناتج : $(101)_2 \times (10)_2 = (1010)_2$

• عملية القسمة

مثال(1)ما هو ناتج قسمة $(1001)_2$ على $(11)_2$

$$\begin{array}{r|l} & 11 \\ \hline 11 & 1001 \\ & 11 \\ \hline & 11 \\ & 11 \\ \hline & 00 \end{array}$$

الناتج :
$(1001)_2 \div (11)_2 = (11)_2$

19.4 النظام الثماني Octal System

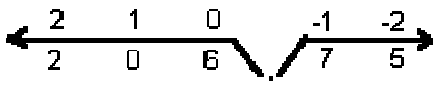
كما هو معروف فإن أساس النظام الثماني هو العدد 8. وتتكون رموز هذا النظام من الأرقام (7..... 2,1,0) .

19.4.1 التحويل من النظام الثماني إلى العشري

Octal to Decimal Conversion

للتحويل من النظام الثماني إلى النظام العشري يستعمل قانون التمثيل الموضعي للأعداد مع مراعاة أن أساس نظام العد هنا هو 8 .

مثال حول العدد الثماني (206.75) إلى مكافئه العشري ؟



$$N = 2x8^{2} + 0x8^{1} + 6x8^{0} + 7x8^{-1} + 5x8^{-2}$$

$$N = 2x64 + 6x1 + 7x\frac{1}{8} + 5x\frac{1}{64}$$

$$N = 128 + 6 + \frac{7}{8} + \frac{5}{64}$$

$$N = 134 + 0.875 + 0.078125$$

$$N = 134.953125$$

الناتج: $(206.75)_{8} = (134.95312)_{10}$

19.4.2 تحويل من النظام العشري إلى الثماني

Number Conversion from decimal to octal System

- تحويل الأعداد الصحيحة الموجبة:

لتحويل أي عدد صحيح موجب من النظام العشري إلى الثماني نستعمل طريقة الباقي المشروحة في النظام الثنائي مع مراعاة أن الأساس الجديد هو 8.

مثال حول العدد العشري 122إلى مكافئه الثماني؟

	ناتج القسمة	الباقي	
1.	122÷8= 15	2	الخانة الأدنى منزلة LSD

.2	15÷8= 1	7	
.3	1÷8= 0	1	الخانة الأعلى منزلة MSD

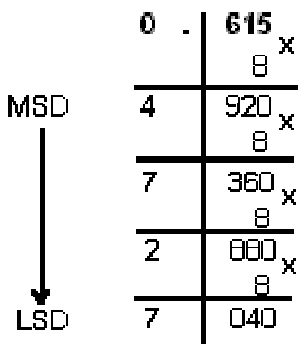
إنهاء القسمة

فيكون الناتج (من أسفل إلى أعلى ومن اليسار إلى اليمين): $(122)_{10} = (172)_8$

• تحويل الكسر العشري إلى مكافئه الثماني:

لتحويل الكسر العشري إلى مكافئه الثماني فإننا نضرب الكسر في الأساس 8 عدداً معيناً من المرات حتى نحصل على ناتج ضرب يساوي صفراً أو حتى نحصل على الدقة المطلوبة.

مثال حول الكسر العشري 0.615 إلى مكافئه الثماني المكون من 4 خانات فقط.



فيكون الناتج (من أعلى إلى أسفل ومن اليسار إلى اليمين): $(0.615)_{10} = (0.4727)_8$

• تحويل العدد العشري الكسري:

في هذه الحالة نحول كل جزء على انفراد، ثم نضم الناتج مع بعض للحصول على الجواب المطلوب.
مثال:

حول العدد العشري 982.42 إلى مكافئه الثماني؟

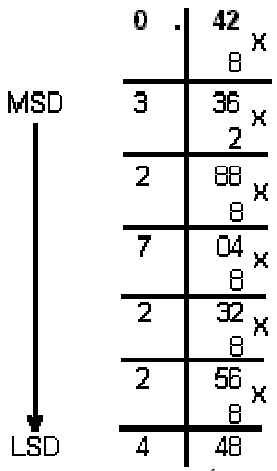
		ناتج القسمة	الباقي	
.1	982÷8=	122	6	الخانة الأدنى منزلة LSD

2 122÷8= 15 .2
7 15÷8= 1 .3
1 1÷8= 0 .4 الخانة الأعلى منزلة MSD

إنهاء القسمة

فيكون الناتج (من أسفل إلى أعلى ومن اليسار إلى اليمين):

$(982)_{10} = (1726)_8$



فيكون الناتج (من أعلى إلى أسفل ومن اليسار إلى اليمين):

$(0.42)_{10} = (0.327224)_8$

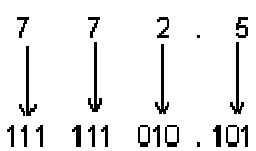
العدد المطلوب: $(982.42)_{10} = (1726.327224)_8$

19.4.3 التحويل من النظام الثماني إلى الثنائي

Number Conversion from Octal to Binary System

لتحويل أي عدد ثماني إلى مكافئه الثنائي نستبدل كل رقم من أرقام العدد الثماني بمكافئه الثنائي المكون من ثلاث خانات و بذلك ينتج لدينا العدد الثنائي المكافئ للعدد الثماني المطلوب تحويله.

مثال حول العدد الثماني $(772.5)_8$ إلى مكافئه الثنائي ؟



$(772.5)_8 = (111111010.101)_2$

19.4.4 التحويل من النظام الثنائي إلى الثماني

Number Conversion from Binary to Octal System

لتحويل الأعداد الثنائية الصحيحة إلى ثمانية نتبع الخطوات التالية:

1. نقسم العدد الثنائي إلى مجموعات كل منها مكون من ثلاث خانات، و يجب أن نبدأ التقسيم من الرقم الأقل أهمية(LSD) .
2. إذا كانت المجموعة الأخيرة غير مكتملة فإننا نضيف في نهايتها الرقم صفر حتى تصبح مكونة من ثلاث خانات ثنائية.
3. نضم الأرقام الثمانية معاً للحصول على العدد المطلوب.
4. في حالة الكسور الثنائية نبدأ بالتقسيم إلى مجموعات من الخانة القريبة على الفاصلة.

مثال: حول العدد الثنائي التالي $(1011011010.1011)_2 = (\ ?\ )_8$ إلى مكافئه الثماني؟

001 011 011 010 . 101 100

↓ ↓ ↓ ↓ ↓ ↓

1 3 3 2 . 5 4

$(1011011010.1011)_2 = (1332.54)_8$

19.4.5 جمع وطرح الأعداد الثمانية

- جمع الأعداد الثمانية:

عند جمع الأعداد الثمانية نتبع نفس الطريقة في حالة الأعداد العشرية مع مراعاة أن أساس نظام العد هو8.

مثال

اجمع العددين الثمانيين $(176.7)_8 + (52.2)_8 = (\ ?\ )_8$:

```
  111
  176.7
+
  052.2
-------
  251.1
```

الناتج: $(176.7)_8 + (52.2)_8 = (\ 251.1)_8$

- طرح الأعداد الثمانية

مثال (1)

اطرح العددين $(260)_8 - (123)_8 = (?)_8$:

$$\begin{array}{r} 260 \\ -\ 123 \\ \hline 135 \end{array}$$

الناتج: $(260)_8 - (123)_8 = (135)_8$

مثال (2):اطرح العددين

$$(2005)_8 - (756)_8 = (?)_8$$

$$\begin{array}{r} 2005 \\ -\ 756 \\ \hline 1027 \end{array}$$

الناتج : $(2005)_8 - (756)_8 = (1027)_8$

19.4.6 ضرب وقسمة الأعداد الثمانية

يمكن تلخيص حقائق الضرب في الجدول ضرب الأعداد الثمانية :

10	7	6	5	4	3	2	1	0	x
0	0	0	0	0	0	0	0	0	0
10	7	6	5	4	3	2	1	0	1
20	16	14	12	10	6	4	2	0	2
30	25	22	17	14	11	6	3	0	3
40	34	30	24	20	14	10	4	0	4
50	43	36	31	24	17	12	5	0	5
60	52	44	36	30	22	14	6	0	6
70	61	52	43	34	25	16	7	0	7
100	70	60	50	40	30	20	10	0	10
110	77	66	55	44	33	22	11	0	11
120	86	74	62	50	36	24	12	0	12

مثال:
أوجد حاصل الضرب :

$(726)_8 \times (3)_8 = (?)_8$

```
   1 2
   7 2 6
x
       3
-------
 2 6 0 2
```

$(726)_8 \times (3)_8 = (2602)_8$: الناتج

مثال:

أوجد ناتج عملية القسمة التالية:

$(2602)_8 \div (3)_8 = (?)_8$

```
     0 7 2 6
   ---------
 3 | 2 6 0 2
   - 2 5
   -------
     0 1 0
   -   6
   -------
       2 2
   -   2 2
   -------
       0 0
```

$(2602)_8 \div (3)_8 = (726)_8$: الناتج

ويمكن أجراء عملية الضرب أو القسمة بتحويل الأعداد المراد ضربها أو قسمتها إلى مكافئها الثنائي أو العشري وأجراء العملية المطلوبة ومن ثم تحويل الناتج إلى مكافئه الثماني.

19.5 النظام السادس عشر Hexadecimal System

إن أساس هذا النظام هو العدد 16 و الجدول التالي يبين رموز(أرقام) هذا النظام و الأعداد العشرية التي تكافؤها.

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	النظام السادس عشر

النظام العشري	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

19.5.1 التحويل من النظام السادس عشر إلى العشري

Number Conversion from Hexadecimal to Decimal System

للتحويل من النظام السادس عشر إلى العشري نستعمل قانون التمثيل الموضعي للأعداد مع مراعاة أن أساس هذا النظام هو 16.

مثال (1) حول العدد $(2AF3)_{16}$ إلى مكافئه العشري؟

$$N = 3x16^0 + Fx16^1 + Ax16^2 + 2x16^3$$
$$N = 3x16^0 + 15x16^1 + 10x16^2 + 2x16^3$$
$$N = 3 + 240 + 2560 + 4096$$
$$N = 6899$$

الناتج: $(2AF3)_{16} = (6899)_{10}$

مثال (2) حول العدد $(0.3A)_{16}$ إلى مكافئه العشري؟

$$N = 3x16^{-1} + Ax16^{-2}$$
$$N = 3x\frac{1}{16} + 10x\frac{1}{256}$$
$$N = 0.1875 + 0.0390625$$
$$N = 0.2265625$$

الناتج: $(0.3A)_{16} = (0.2265625)_{10}$

19.5.2 التحويل من النظام العشري إلى السادس عشر

Number Conversion from Decimal to Hexadecimal System

لتحويل الأعداد الصحيحة الموجبة من النظام العشري إلى السادس عشر: نستعمل طريقة الباقي و ذلك بالقسمة على الأساس16.

مثال (1) حول العدد العشري $(72)_{10}$ إلى مكافئه السادس عشر؟

	ناتج القسمة	الباقي	
1.	72÷16=4	8	MSD

2.	4÷16=0	4	LSD
	إنهاء القسمة		

الناتج: $(72)_{10} = (48)_{16}$

مثال (2) حول العدد العشري $(1256)_{10}$ إلى مكافئه السادس عشر؟

	ناتج القسمة	الباقي	
1.	1256÷16=781	8	MSD
2.	78 ÷16=4	14	
3.	4÷16=0	4	LSD
	إنهاء القسمة		

الناتج: $(1256)_{10} = (4E8)_{16}$

لتحويل الأعداد العشرية الكسرية : فإننا نضرب الكسر في الأساس 16 ثم نضرب الناتج في الأساس 16 و هكذا حتى نحصل على الدقة اللازمة.

مثال حول العدد العشري $(0.12)_{10}$ إلى مكافئه السادس عشر، على أن يكون الجواب مكوناً من 4 أرقام؟

	0 . 12	x 16
MSD ↓	1 92	x 16
	14 72	x 16
	11 52	x 16
LSD	8 32	

الناتج: $(0.12)_{10} = (0.1EB8)_{16}$

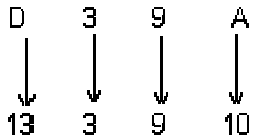
19.5.3 التحويل من النظام السادس عشر إلى الثنائي

Number Conversion from Hexadecimal to Binary System

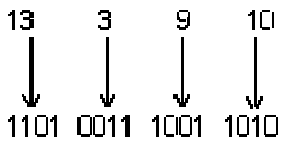
• لتحويل أي عدد من النظام السادس عشر إلى مكافئه الثنائي نتبع الآتي:

مثال حول العدد السادس عشر $(D39A)_{16}$ إلى مكافئه الثنائي؟

1. نستبدل الخانات المكتوبة بدلالة الحروف إن وجدت في العدد بالأعداد العشرية المكافئة لها.



2. نستبدل كل عدد عشري بمكافئه الثنائي المكون من أربعة خانات.



3. ثم نضم الأرقام الثنائية مع بعضها لنحصل على العدد المطلوب:

$$(D39A)_{16} = (1101001110011010)_2$$

19.5.4 التحويل من النظام الثنائي إلى السادس عشر:

Number Conversion from Binary to Hexadecimal System

- لتحويل أي عدد صحيح من النظام الثنائي إلى السادس عشر نتبع الآتي:

1. نقسم العدد الثنائي إلى مجموعات كل منها يتكون من 4 خانات مع مراعاة أن يبدأ التقسيم من الرقم الأقل أهمية (LSB) .
مثال العدد الثنائي التالي 1010011011011110011101 يصبح تقسيمه إلى مجموعات كالآتي:
1 0100 1101 1011 1100 1101
2. إذا كانت المجموعة الأخيرة غير مكتملة فإننا نضيف في نهايتها الصفر حتى تصبح مكونة من أربعة خانات :

0001	0100	1101	1011	1100	1101

3. نحول كل مجموعة ثنائية إلى مكافئها في النظام العشري:

0001	0100	1101	1011	1100	1101
1	4	13	11	12	13

4. نستبدل كل رقم عشري(من الخطوة السابقة) أكبر من9 بدلالة حروف النظام السادس عشر:

1	4	13	11	12	13
1	4	D	B	C	D

5. نضم الأرقام الناتجة مع بعضها لنحصل على الجواب المطلوب في النظام السادس عشر:
14DBCD

6. إذا كان العدد الثنائي كسراً نبدأ بالتقسيم إلى مجموعات من الخانة القريبة على الفاصلة ثم نتبع باقي الخطوات المشروحة سابقاً.

19.5.5 التحويل من النظام السادس عشر إلى الثماني

Number Conversion from Hexadecimal to Octal System

- لتحويل أي عدد من النظام السادس عشر إلى النظام الثماني: نقوم أولاً بتحويله إلى النظام الثنائي كما مر معنا سابقاً و ذلك باستبدال كل رقم من أرقام العدد السادس عشر إلى مكافئه الثنائي المكون من أربعة خانات، و بعد ضم الأرقام الثنائية إلى بعضها نقوم مرة أخرى بتقسيمها إلى مجموعات من ثلاثة خانات و نستبدل كل مجموعة برقم ثماني و بذلك نكون قد حصلنا على العدد الثماني المطلوب.

مثال حولي العدد السادس عشر $(B51.DF2)_{16}$ إلى مكافئه الثماني:

الحل:

1. نقوم بتحويل العدد السادس عشر إلى مكافئه الثنائي

B	5	1	.	D	F	2
11	5	1		13	15	2
1011	0101	0001	,	1101	1111	0010

2. ثم نعيد تقسيم العدد الثنائي إلى مجموعات كل منها يتكون من ثلاثة خانات ثنائية ثم نكتب العدد الثماني المكافيء لكل مجموعة :

101	101	010	001	.	110	111	110	010

5	5	2	1		6	7	6	2

الناتج: $(B51.DF2)_{16} = (5521.6762)_8$

19.5.6 التحويل من النظام الثماني إلى السادس عشر

Number Conversion from Octal to Hexadecimal System

لتحويل أي عدد ثماني إلى النظام السادس عشر: نقوم أولاً بتحويله من الثماني إلى الثنائي، ثم نقسم العدد الثنائي الناتج إلى مجموعات كل منها يتكون من أربعة خانات، و نقوم باستبدال كل مجموعة منها بما يكافئها في النظام السادس عشر.

مثال حول العدد الثماني $(163.45)_8$ إلى مكافئه السادس عشر

1 6 3 . 4 5

↓ ↓ ↓ ↓ ↓

001 110 011 . 100 101

001110011.10010100

↓ ↓ ↓ ↓

7 3 . 9 4

الناتج: $(163.45)_8 = (73.94)_{16}$

19.5.7 جمع و طرح الأعداد في النظام السادس عشر

عند جمع وطرح الأعداد في النظام السادس عشر نتبع نفس الأسلوب المستعمل في النظام العشري مع مراعاة أن أساس هذا النظام هو 16.

مثال(1) اجمع العددين التاليين: $(6AD)_{16} + (253)_{16} = (?)_{16}$

$$\begin{array}{r} 6AD \\ + \quad 253 \\ \hline 900 \end{array}$$

الناتج: $(6AD)_{16} + (253)_{16} = (900)_{16}$

مثال(2) اجمع العددين التاليين: $(F6F)_{16} + (ABA)_{16} = (?)_{16}$

```
  F6F
+ ABE
------
 1A2D
```

الناتج: $(F6F)_{16} + (ABA)_{16} = (1A2D)_{16}$

مثال(3)اطرح العددين التاليين: $(AED)_{16} - (826)_{16} = (?)_{16}$

```
  AED
- 826
------
 2C7
```

الناتج: $(AED)_{16} - (826)_{16} = (2C7)_{16}$

مثال(4)اطرح العددين التاليين: $(88E)_{16} - (70F)_{16} = (?)_{16}$

```
  8BE
- 7DF
------
 0DF
```

الناتج: $(88E)_{16} - (70F)_{16} = (DF)_{16}$

19.5.8 ضرب وقسمة الأعداد في النظام السادس عشر

يمكن تلخيص حقائق الضرب في الجدول ضرب الأعداد في النظام السادس عشر

10	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	X
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	1
20	1E	1C	1A	18	16	14	12	10	E	C	A	8	6	4	2	0	2
30	2D	2A	27	24	21	1E	1B	18	15	12	F	C	9	6	3	0	3
40	3C	38	34	30	2C	28	24	20	1C	18	14	10	C	8	4	0	4
50	4B	46	41	3C	37	32	2D	28	23	1E	19	14	F	A	5	0	5
60	5A	54	4E	48	42	3C	36	30	2A	24	1E	18	12	C	6	0	6
70	69	62	5B	54	4D	46	3F	38	31	2A	23	1C	15	E	7	0	7
80	78	70	68	60	58	50	48	40	38	30	28	20	18	10	8	0	8
90	87	7E	75	6C	63	5A	51	48	3F	36	2D	24	1B	12	9	0	9
A0	96	8C	82	78	6E	64	5A	50	46	3C	32	28	1E	14	A	0	A
B0	A5	9A	8F	84	79	6E	63	58	4D	42	37	2C	21	16	B	0	B
C0	B4	A8	9C	90	84	78	6C	60	54	48	3C	30	24	18	C	0	C
D0	C3	B6	A9	9C	8F	82	75	68	5B	4E	41	34	27	1A	D	0	D
E0	D2	C3	B6	A8	9A	8C	7E	70	62	54	46	38	2A	1C	E	0	E
F0	E1	D2	C3	B4	A5	96	87	78	69	5A	4B	3C	2D	1E	F	0	F
100	F0	E0	D0	C0	B0	A0	90	80	70	60	50	40	30	20	10	0	10

مثال:أوجد حاصل الضرب :

$(A14)_{16} \times (5)_{16} = (?)_{16}$

```
    1
  A 1 4
x
      5
-------
  3 2 6 4
```

الناتج : $(A14)_{16} \times (5)_{16} = (3264)_{16}$

مثال:أوجد ناتج عملية القسمة التالية:

$(3264)_{16} \div (5)_{16} = (?)_{16}$

```
    0 A 1 4
  ---------
5 | 3 2 6 4
  - 3 2
  -------
    0 0 6
  -   5
  -------
      1 4
  -   1 4
  -------
        0 0
```

الناتج : $(3264)_{16} \div (5)_{16} = (A14)_{16}$

ويمكن أجراء عملية الضرب أو القسمة بتحويل الأعداد المراد ضربها أو قسمتها إلى مكافئها الثنائي أو العشري وأجراء العملية المطلوبة ومن ثم تحويل الناتج إلى مكافئه السادس عشر.

19.6 تمثيل الأعداد السالبة Signed Numbers

في العمليات الرياضية العادية يسمى العدد سالباً إذا سبقته إشارة الناقص(-)، و يسمى موجباً إذا سبقته إشارة الزائد(+) أما في الحاسوب فتستعمل ثلاث طرق لتمثيل الأعداد السالبة و هي-:

1- التمثيل بواسطة الإشارة و المقدار Representation Signed-Magnitude.
2- التمثيل بواسطة العدد المكمل للأساس Representation Radixed-Complement.
3- التمثيل بواسطة العدد المكمل للأساس المصغر Diminished Radix Complement Representation.

19.6.1 التمثيل بواسطة الإشارة و المقدار:

لتمثيل الأعداد الثنائية داخل الحاسوب، اصطلح على استعمال الرقم "0" ليدل على الإشارة الموجبة و الرقم "1" ليدل على الإشارة السالبة. و يتكون العدد الممثل بهذه الطريقة من جزئين هما: الإشارة و المقدار.

مثل العددين +24 , -24 في كل من النظامين العشري و الثنائي بواسطة طريقة التمثيل بالإشارة و المقدار؟

الجواب:

في النظام العشري		في النظام الثنائي	
المقدار	الإشارة	المقدار	الإشارة
24	+	11000	0
24	-	11000	1

و عند التعامل مع الأعداد الثنائية الممثلة بالإشارة و المقدار، توضع عادة فاصلة بين خانة الإشارة و المقدار ويمكن كذلك وضع خط صغير تحت خانة الإشارة ، أو يمكن استعمال الفاصلة و الخط الصغير معاً.

19.6.2 التمثيل بواسطة المكمل للأساس Radixes-Complement Representation :

نفترض وجود العدد N ممثلاً بنظام عد أساسه R ، ونفترض كذلك أن هذا العدد يتكون من n خانة صحيحة و m خانة كسرية، و سنرمز

لمكمل العدد N على الأساس R، بالرمز $\bar{N}$ حيث يمكن حساب العدد $\bar{N}$ حسب العلاقة التالية:

$$\bar{N} = R^n - N \quad \ldots\ldots\ldots (1)$$

ويسمى العدد $\bar{N}$ في النظام العشري "بالمكمل لعشرة" (10's Complement)

و في النظام الثنائي "بالمكمل لاثنين.(2's Complement)"

مثال (1) جد المكمل لعشرة للعدد 320.52:

الحل:

$$\bar{N} = R^{n} - N$$
$$= 10^{3} - 320.52$$
$$= 1000 - 320.52$$
$$\bar{N} = 679.48$$

مثال (2) جد المكمل لاثنين للعدد الثنائي101.1:

الحل:

$$\bar{N} = 2^{3} - 101.1$$
$$= 1000 - 101.1$$
$$= 10.1$$

19.6.3 التمثيل بواسطة المكمل"للأساس الأصغر" Diminished Radix Complement Representation :

يسمى أساس نظام العد مصغراً إذا كان ينقص بمقدار واحد عن الأساس الأصلي. فمثلاً الأساس المصغر للنظام الثنائي هو 1 و كذلك الأساس المصغر للنظام العشري هو9. و يرمز للمكمل للأساس المصغر بالرمز $\bar{\bar{N}}$ حسب العلاقة التالية:

$$\bar{\bar{N}} = R^{n} - N - R^{-m} \quad \ldots\ldots\ldots\ldots (2)$$

حيث أن:
R:أساس نظام العد.
N:العدد المطلوب إيجاد مكمله للأساس المصغر.
n:عدد خانات الجزء الصحيح.
m:عدد خانات الجزء الكسري.
يسمى المكمل للأساس المصغر في النظام العشري"بالمكمل لتسعة"(9's Complement)
ويسمى في النظام الثنائي"بالمكمل لواحد" (1's Complement).

مثال(1) جد المكمل لتسعة للعدد320.53:

الحل:

$$\bar{\bar{N}} = 10^{3} - 320.52 - 10^{-2}$$
$$\bar{\bar{N}} = 1000 - 320.52 - 0.01$$
$$\bar{\bar{N}} = 679.47$$

مثال(2) جد المكمل لواحد للعدد الثنائي101.1:

الحل:

$$\bar{\bar{N}} = 2^{3} - 101.1 - 2^{-2}$$
$$\bar{\bar{N}} = 1000 - 101.1 - 0.1$$
$$\bar{\bar{N}} = 10.0$$

• المكمل لواحد 1's Complement:

بالإضافة إلى الطريقة المشروحة فيما سبق فإنه من الأسهل إتباع القاعدة التالية للحصول على المكمل لواحد لأي عدد ثنائي فإنه سالب:(للحصول على المكمل لواحد لأي عدد ثنائي فإنه يلزم أن نعكس خانات ذلك العدد بحيث نستبدل الواحد بالصفر والصفر بالواحد).

مثال جد المكمل لواحد للعدد الثنائي100.10:

الحل:

نعكس خانات العدد باستبدال الصفر بالواحد و الواحد بالصفر

الجواب هو: (011.01).

• المكمل لاثنين 2's Complement:

كذلك لإيجاد المكمل لاثنين لأي عدد ثنائي سالب يمكن إتباع القاعدة التالية:
[المكمل لاثنين=المكمل لواحد+1]
أي أننا نقوم أولاً باستخراج المكمل لواحد، ثم نضيف إليه العدد 1.

مثال أوجد المكمل لاثنين للعدد 100.10:
الحل:

المكمل لواحد هو011.01

```
  011.01
+
       1
-------
  011.10
```

المكمل لاثنين هو 011.10
و يمكن التأكد من الجواب لو طبقنا العلاقة الرياضية (1) المشروحة فيما سبق.

19.6.4 جمع وطرح الأعداد الثنائية باستعمال المكمل لواحد

Binary Addition and Subtraction using 1's complement

عند جمع وطرح الأعداد الثنائية باستخدام المكمل لواحد نقوم في البداية بتحويل العدد السالب إلى صيغة المكمل لواحد، ثم نجمع المكمل لواحد مع العدد الآخر الموجب و بذلك نكون قد حولنا عملية الطرح إلى جمع حسب القاعدة X+ (-Y).

و من الملاحظ هنا أن خانة الإشارة تشترك في عملية الجمع و قيمتها النهائية تقرر إشارة العدد الناتج، فإذا كانت خانة الإشارة للناتج صفراً فإن الناتج يكون موجباً و ممثلاً بطريقة الإشارة و المقدار. أما إذا كانت خانة الإشارة واحداً فإن الناتج يكون سالباً وممثلاً بواسطة المكمل لواحد. و لإيجاد القيمة الحقيقية للناتج يمكن تحويله مرة أخرى إلى المكمل لواحد.

لو افترضنا أن العددين المطلو ب جمعهما أو طرحهما هما X,Y فإنه يمكن الحصول على الحالات التالية لاحتمالات الجمع والطرح وهذه الحالات هي:

•الحالة الأولى: إذا كان X موجبة، Y موجبة:

في هذه الحالة لا توجد عملية طرح، بل نقوم بجمع العددين معاً كما هو الحال في الأعداد الموجبة الممثلة بالإشارة و المقدار. و يجب أن نلاحظ أنه قد تظهر حالة الفيض(Overflow) عند الجمع و لهذا السبب يجب إضافة خانة الصفر إلى يسار كل عدد لاستيعاب حالة الفيض.(الخانة المضافة يجب أن تكون في نهاية المقدار على يمين خانة الإشارة).

مثال (1) اجمع العددين X= +12 Y= +9 :

الحل:

+12	0.01100
+ 9	0.01001
+21	0.10101

•الحالة الثانية: إذا كانت Xموجبة، Y سالبة:

1. إذا كانت |Y|>|X|

مثال (2)اجمع العددين X= +12, Y= -9

الحل :

X= +1100 Y= -1001

المكمل لواحد للعدد1001- هو1.0110 الآن نجمع العددين معاً:

+12		0.1100
- 9		1.0110
+ 3	محمل مدور	10.0010
		1
		0.0011

نلاحظ أنه أثناء الجمع حدث محمل (Carry) في خانة الإشارة، و يسمى هذا المحمل بالمحمل المدور(End Around Carry) حيث تلزم إعادة جمعه مع الخانة الأولى في النتيجة.الجواب الناتج إشارته موجبة ويكون ممثلاً بالإشارة و المقدار.

أي أنه يساوي هنا (+3).

مثال(3) اجمع العددين: X=+9 ,Y= -12 :

الحل:

X=+1001 Y= -1100

المكمل لواحد للعدد 1100- هو 10011

+ 9	0.1001
- 12	1.0011
- 3	1.1011

نلاحظ أن الإشارة الناتجة سالبة و في هذه الحالة تكون النتيجة ممثلة بواسطة المكمل لواحد. ولإيجاد النتيجة الصحيحة نقوم بتحويل النتيجة إلى المكمل لواحد مرة أخرى. أي أن الجواب يساوي(3-).

• الحالة الثالثة:إذا كانتXسالبة، Yموجبة.

1. إذا كانت |X|>|Y|

مثال (4):

X=-12 -1100

Y=+9 +1001

نحول العدد السالب إلى المكمل لواحد ثم نجمع العددين.

المكمل لواحد للعدد 12-هو10011

- 12	1.0011
+ 9	0.1001
- 3	1.1100

إشارة النتيجة هنا سالبة و النتيجة ممثلة بواسطة المكمل لواحد. و لذلك نحولها مرة أخرى إلى المكمل لواحد. الجواب هو(0011-) و يساوي(3-).

مثال(5) : -1001 X=-9

+1100 Y=+12

المكمل للعدد 1001- هو 1.0110

- 9	1.0110
+12	0.1100
	1 0.0010
+ 3	1
	0.0011

-0 -12 النتيجة موجبة و ممثلة بطريقة الإشارة و المقدار أي أن الجواب هنا(+0011) و يساوي (+3).

1100

المكمل لواحد للعدد -01001 هو 1.10110

المكمل لواحد للعدد -01100 هو 1.10011

•الحالة الرابعة: إذا كانتXسالبة، Yسالبة. في هذه الحالة نحول كلاً منهما إلى المكمل لواحد ثم نجمعهما.

مثال (6):-1001 X=-9

-1100 Y=-12

في هذه الحالة و بسبب كون إشارتي العددين متشابهتين فإنه أثناء الجمع تنتج حالة فيض و من أجل استيعاب النتيجة و قبل أن نقوم بتحويل العددين إلى صيغة المكمل لواحد نضيف إلى يسار كل عدد خانة الصفر فيصبح كل منهما كما يلي:

و الآن نقوم بالجمع:

- 9	1.10110
- 12	1.10011
- 21	1 1.01001
	1
	1.01010

إشارة النتيجة سالبة و يلزم تحويل النتيجة إلى المكمل لواحد فيكون الجواب (-10101) أي (-21).

نلاحظ من خلال الحالات التي تكلمنا عنها و من خلال الأمثلة المحلولة أن المكمل لواحد لا

يحقق المعادلة الرياضية 0=(-n)+(+n). فعلى سبيل المثال لو كانت X=+5 ,Y=-5. فإنه عند جمعهما باستعمال المكمل لواحد ينتج:

+ 5	0.101
- 5	1.010
0	1.111

يلاحظ هنا أن جمع عددين متساويين في المقدار و مختلفين في الإشارة لا يعطي مباشرة الصفر بل يلزم تحويل النتيجة إلى المكمل لواحد، و يلاحظ كذلك أن إشارة الجواب سالبة أي (-0).

19.6.5 جمع و طرح الأعداد الثنائية باستعمال المكمل لاثنين

Binary Addition and Subtraction Using 2's Complement:

فإنه يجب جمعه مع الخانة الأولى للنتيجة، و هذه الخطوة تعتبر خطوة زائدة من شأنها أن تجعل عملية الطرح أو الجمع بطيئة.

و للتخلص من المحمل المدور هذا تستعمل في الحاسوب طريقة تمثيل الأعداد السالبة بواسطة المكمل لاثنين. و لجمع و طرح الأعداد بواسطة المكمل لاثنين نتبع الأسلوب التالي:

نقوم بتمثيل العدد السالب بواسطة المكمل لاثنين ثم نجمعه مع العدد الآخر و إذا حدث محمل في خانة الإشارة فإنه يهمل و لا تلزم إضافته إلى النتيجة.

و لتوضيح فكرة استعمال المكمل لاثنين فإننا نورد الحالات التالية للعددين الثنائيينX ,Y:

- الحالة الأولى: إذا كانت Xموجبة، Yسالبة.

نقوم في هذه الحالة بجمع الأعداد مباشرة و لا يلزم التحويل إلى المكمل لاثنين، و هذه الحالة تشبه الحالة الأولى التي ذكرناها في موضوع جمع و طرح الأعداد الثنائية باستعمال المكمل لواحد.

- الحالة الثانية: إذا كانتXموجبة، Yسالبة.

1. إذا كانت |X|>|Y|

في هذه الحالة نحول العدد السالب إلى المكمل لاثنين ثم نجمعه مع العدد الموجب، و إذا نتج

محمل في خانة الإشارة نهمله.

مثال(1):1100+ X=+12

-1001 Y=-9

المكمل لاثنين للعدد -9 هو 10111

+ 12	0.1100
- 9	1.0111
+ 3	⇒ 0.0011

النتيجة موجبة و هي (+0011) و تساوي (+3)

مثال(2) :1001+ X=+9

1100 Y=-12

المكمل لاثنين للعدد -12 هو 10100

+ 9	0.1001
- 12	1.0100
- 3	1.1101

إشارة النتيجة سالبة و هي بدلالة المكمل لاثنين، و للحصول على النتيجة الصحيحة يجب تحويلها مرة أخرى إلى المكمل لاثنين. أي أن النتيجة الصحيحة هي (-0011)أي (-3).

•الحالة الثالثة: إذا كانت X سالبة، Y موجبة
و هذه الحالة تشبه الحالة السابقة.

•الحالة الرابعة: إذا كانتX سالبة، Y سالبة
في هذه الحالة نحول كلاً من العددين إلى المكمل لاثنين ثم نجمعهما.

مثال(3) : -1001 X=-9

-1100 Y=-12

نضيف خانة خامسة قيمتها الصفر إلى كل من العددين و ذلك لاستيعاب حالة الفيض.

من مساوئ استخدام المكمل لواحد أنه عادةً إذا ظهر محمل مدور(End Around Carry)

-9 = -01001
-12= -01100

ثم نحول كل عدد إلى المكمل لاثنين:
المكمل لاثنين للعدد -9 هو 1.10111
المكمل لاثنين للعدد -12 هو 1.10100

- 9	1.10111
- 12	1.10100
- 21	1.01011

إشارة النتيجة سالبة و لذلك نحول النتيجة إلى المكمل لاثنين.
أي أن النتيجة الصحيحة هي (10101-)و تساوي (21-).

19.6.6 طرق ضرب الأعداد الثنائية

Methods of Binary Multiplication:

يمكن إجراء عملية الضرب في النظام الثنائي على الأعداد الممثلة بالإشارة و المقدار و كذلك الأعداد الممثلة بواسطة المكمل لواحد أو المكمل لاثنين. و لكن تعتبر طريقة الضرب باستخدام الأعداد الممثلة بالإشارة و المقدار الطريقة المثلى في حالتي الضرب والقسمة و ذلك لأن الإشارة السالبة يمكن التعامل معها بسهولة، حيث أن ضرب أي عددين مختلفين في الإشارة يعطي نتيجة سالبة الإشارة و كذلك قسمة عددين متشابهين في الإشارة تعطي أيضاً نتيجة موجبة الإشارة.

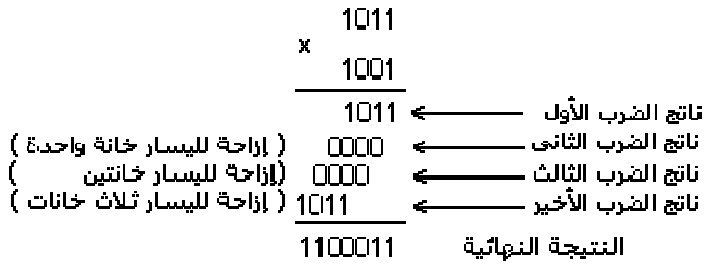
وطرق الضرب المستعملة في الحاسوب كثيرة و تختلف فيما بينها من حيث سرعة تنفيذها داخل الحاسوب. و للتبسيط سنقوم هنا بشرح الطريقة المعروفة" بطريقة الضرب بواسطة الجمع المتتالي و الإزاحة".

- الضرب بواسطة الجمع المتتالي و الإزاحة Multiplication by Successive Shifting & Addition:

سنستعرض في البداية الطريقة العادية المتبعة لتنفيذ عملية الضرب باستعمال القلم و الورقة من خلال المثال التالي:

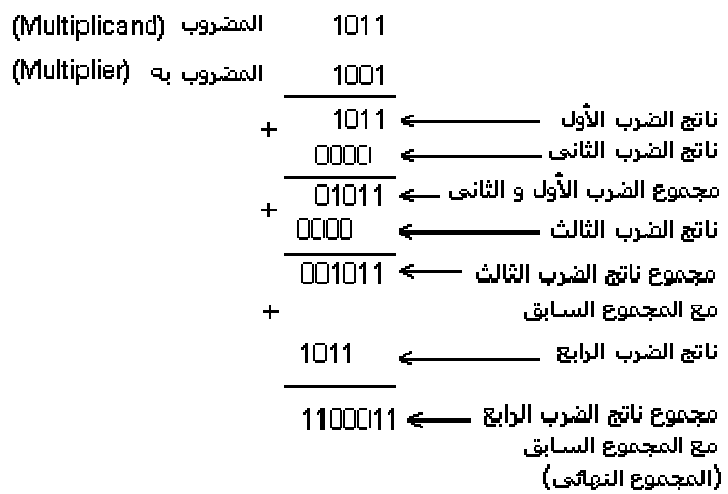
اضرب العددين الثنائيين: Y=1001, X=1011

الحل:

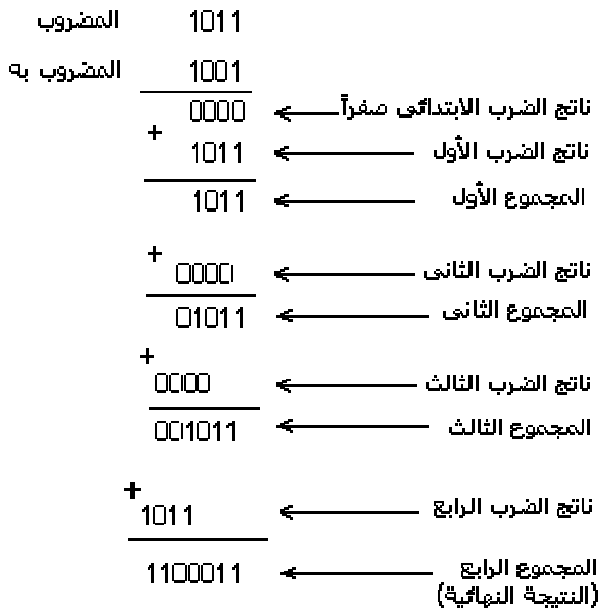


إن طريقة (خوارزمية) عملية الضرب المستعملة في هذا المثال، هي أننا ضربنا الخانة الأولى من المضروب به في المضروب ثم جمعنا إلى الناتج حاصل ضرب الخانة الثانية من المضروب به في المضروب و هكذا.

و يمكن توضيح طريقة الضرب هذه من خلال المثال التالي:



أما داخل الحاسوب فتستعمل الطريقة المعدلة التالية، و هي أن نعتبر أن ناتج الضرب الابتدائي يساوي صفراً ثم نجمع إليه حاصل الضرب الأول و هكذا:



و كما نلاحظ، لا تختلف هذه الطريقة عن سابقتها سوى في إضافة ناتج ضرب ابتدائي يساوي صفر، و يتضح من مثال هذه الطريقة فكرة الجمع المتتالي لناتج الضرب مع المجموع السابق.

19.6.7 طرق قسمة الأعداد الثنائية Binary Division

بينما تعتبر عملية الضرب سلسلة من عمليات الجمع المتتالي و الإزاحة، فإن عملية القسمة تعتبر سلسلة من عمليات الطرح المتتالي و الإزاحة.

و طرق تنفيذ عملية القسمة داخل الحاسوب متنوعة وكثيرة أيضاً و سنتكلم هنا عن أبسط هذه

الطرق و هي طريقة القسمة باستعمال الطرح المتتالي، وهي طريقة شبيهة بطريقة القسمة باستعمال الورقة والقلم، و تطبق عادةً على الأعداد الممثلة بالإشارة و المقدار و في حالة كون إشارتي المقسوم و المقسوم عليه مختلفين تكون إشارة الناتج سالبة.
و المثال التالي يوضح هذه الطريقة:

اقسم العدد 10110على 111

الحل:

```
      00011.001001
111 | 10110
      111
      ---
       1000
        111
      ------
      001000
       111
      ------
      001000
       111
      ------
         001
```

الجواب :11.001001

19.7 تمثيل الأعداد بواسطة النقطة العائمة

Representation of Numbers by Floating Point:

إن أي عدد عشري صحيح مثل 125يمكن كتابته على النحو التالي:

$$125 = .125 \times 10^{3} = 1.25 \times 10^{2} = 12.5 \times 10^{1}$$

و إذا رمزنا للأساس 10 بالرمز E فإن العدد السابق يصبح كما يلي:

125 = .125E3 = 1.25E2 = 12.5E1

أما إذا كان العدد كسرايا مثل.00127. فيمكن كتابته على النحو التالي:

$$.00127 = 12.7 \times 10^{-4} = 1.27 \times 10^{-3} = .0127 \times 10^{-1}$$

و إذا استبدلنا الأساس10 بالرمز E فإن تمثيل العدد يصبح كالآتي:

.00127=12.7E-4=1.27E-3=.127E-2=.0127E-1

يلاحظ مما سبق أن موقع النقطة داخل العدد عائم (غير ثابت) و يعتمد على الأس المرفوع له أساس نظام العد. و يمكن اعتبار أي عدد ممثل بواسطة النقطة العائمة منسجماً مع الشكل العام التالي: $\pm M \times E^{\pm P}$

M الجزء ألكسري من العدد (Mantissa or Fraction).
E أساس نظام العد.
P الأس (القوة) (Exponent or Characteristic).
يشترط في العدد الممثل بواسطة النقطة العائمة ألاّ يكتب على شكل عدد صحيح وألاّ يكون أول رقم فيه على يمين النقطة صفراً.

و يسمى هذا الشكل الموصوف بهذه الشروط بالشكل المعياري لـ لعدد الممثل بالنقطة العائمة. و مثال ذلك العدد الثنائي110.110 يمثل بالشكل المعياري بواسطة النقطة العائمة كما يلي:

$.110110 \times 2^{3}$

و عادة يكتب الشكل العام للعدد الممثل بالنقطة العائمة ضمن الكلمة(Word) داخل الحاسوب، و يخصص لكل جزء من أجزاء الكلمة عدد معين من الخانات بما في ذلك الجزء الخاص بالإشارة، و ذلك حسب طول الكلمة المستعملة في الحاسوب و الشكل التالي يبين كلمة حاسوب تستعمل فيه النقطة العائمة.

أشارة العدد Sign	الجزء ألكسري Mantissa	أشارة الأس Exponent Sign	الأس Exponent

إن الشكل العام لهذه الكلمة يمكن أن يختلف من حاسوب إلى آخر و خاصة فيما يتعلق بترتيب أجزاء الكلمة.

G: مصطلحات البرمجة

فهرس بالألفاظ

Index

O

P

R

S

K

L

M

N

References المصادر

المراجع العربية:

1. أصول البرمجة بلغة C++ تأليف م. عمار الدبعي (2005).
2. البرمجة الموجهة بلغة Java تأليف م. عمار الدبعي (2007).
3. مئة سؤال وسؤال لإتقان البرمجة تأليف م. عمار الدبعي (2006).
4. هياكل البيانات بلغة C++ تأليف م. عمار الدبعي (2006).
5. مبادئ الحاسوب و البرمجة بلغة بيسك تأليف د. مروان مصطفي ناعه (1997).
6. مبادئ الحاسوب الإلكتروني تأليف د. سعيد عساف (1997).
7. مقدمة في علم الحاسوب تأليف د. محمد كامل ،د. حسن طاهر،دار النهظة (1998)، جمهورية مصر العربية.
8. مقدمة إلى لغة C تأليف عامر بواب (1996).
9. نظم أعداد الميكروكمبيوتر تأليف د.مظهر طايل (1986).
10. تصميم و إنشاء و إدارة مواقع الويب تأليف ماري هاجارد (1998).
11. تعلم Java باستخدام بيئة تطوير JBulider تأليف جلال خرسانة (2006).
12. برمجة الحاسبات الالكترونية بلغة فوتران تأليف د. محمد الفيومي (1986).
13. الطريق إلى احتراف Java تأليف عزب محمد عزب.
14. كيف تبرمج بلغة C++ تأليف د. صلاح الدوه جي (2004).
15. المؤسسة العامة لتعليم الفني والتدريب المهني.
16. التحليل و التصميم بالمنحى للكائن باستخدام UML تأليف خالد شقروني (2006).

مواقع الويب العربية:

1. الموسوعة العربية http://www.c4arab.com.
2. http://www.javagirl.ws .
3. المدرس العربي http://www.deyaa.org.
4. منتديات الفريق العربي للبرمجة.

المراجع الانجليزية:

- Introduction to Computers Fourth Edition Peter Norton's (2000).
- Introduction to Computer Science.

- Digital Design Second Edition M.Morris Mano (1991).
- A Book on C Third Edition Al Kelley / Ira Pohl (1995).
- Advanced Programming in Pascal with Data Structures Larry Nyhoff / Sanford Leestma (1989).
- Osborne - Java 2--Complete Reference (5th Ed 2002)
- Thinking in Java, 2nd edition, Revision 12.
- *The Java Tutorial, Third Edition*.
- *The Java Tutorial Continued: The Rest of the JDK*.
- *The JFC Swing Tutorial: A Guide to Constructing GUIs*.
- Java™ How to Program, Sixth Edition(2004).
- Java™ Language Specification, Third Edition, The(2005).
- Object-Oriented Data Structures using Java(2001).
- Data Structures and Algorithms in Java(2005).
- Introduction to Computer Science using Java(2003).
- The Java Language Specification Third Edition(2005).
- Introduction to Programming Using Java Version (4.1, June 2004).

مواقع الويب الانجليزية:

- www.java.sun.com.
- http://www.rlg.org/visguides/visguide3.html.
- http://www.data-compression.com/lossless.html.
- http://splash.javasoft.com/jdbc.
- http://www.scism.sbu.ac.uk/jfl/index.html.