الجمهورية اليمنية

جامعة إب

لغة جافا و البرمجة الموجهة بالكائنات

JAVA Programming Language and OOP

للدكتور

نشوان المجمر

2016

محاضره (1)

مقدمة عن لغات البرمجه و المفاهيم البرمجيه

قبل أن نتحدث عن لغة جافا يجب أن نعطي مفهوماً عاماً عن البرمجة بالأهداف (**Object-Oriented Programming**) فلغة الجافا هي من لغات البرمجه بالأهداف.

Object-Oriented Programming (OOP)

1. مراحل تطور اللغات وظهور الكثير من المفاهيم.

من أوائل اللغات التي ظهرت هي لغة Assembly وهي أصعب من لغات المستوى العالي ثم تلي ذلك ظهور لغات تستخدم مفردات اللغة الانجليزية مثل Fortran فلغات البرمجة تنقسم إلى قسمين :

a) لغات المستوى العالي High Level Language مثل Basic

b) لغات المستوى الأدنى Low Level Language مثل Assembly

أدى تطور لغات البرمجة إلى ظهور مفاهيم برمجيه مختلفه:

- البرمجة التركيبية Structured programming: تعتمد البرمجة التركيبية على تقسيم البرنامج إلى أجزاء (دوال Functions) ويعطى لكل جزء(Function) أسم خاص به ثم يقوم بعد ذلك المبرمج باستدعاء هذه الأجزاء التي يقوم كلٌ منها بأداء مهمة محددة. يتم تقسيم البرنامج عاده إلى دالة رئيسية والى دوال فرعية و لاكن لا بد من وجود داله رئيسية main و التي يبدأ المترجم من عندها بتنفيذ البرنامج و هي الجزء الاساسي و الضروري من البرنامج و من دون وجود الداله الرئيسيه لا يتم تنفيذ أي جزء من البرنامج.

Main function

Other Functions

تعريف

البرنامج في البرمجة التركيبة : عباره عن مجموعه من الدوال(داله من هذه الدوال هي الداله الرئيسيه main) والدالة تتكون من مجموعة من التعليمات أو الأوامر المختلفه.

مع تطور البرمجة وزياد التعقيد في البرامج تطورت مفاهيم البرمجة فظهر مفهوم البرمجه بالأهداف Object-Oriented Programming (OOP) وتفيدنا البرمجة بالأهداف في إننا نستطيع تمثيل الاشياء المادية المحيطة بنا تمثيلا حقيقياَ.

- البرمجة بالأهداف (OOP) : تعتمد البرمجة بالأهداف على تقسيم البرنامج إلى أجزاء (فصائل classes) ويعطى لكل جزء(class) أسم خاص به. يقوم كلً من هذه الفصائل بأداء مهمة محددة. يتم تقسيم البرنامج عاده الي فصيله رئيسية و هي التي تحتوي علي الداله الرئيسيه (main) والى فصائل أخري.

تعريف:

البرنامج في OOP : عبارة عن مجموعة من الأهداف (العناصر) التي تتفاعل مع بعضها البعض ولكي نستطيع أن ننشى أي من العناصر لابد أن يكون لكل عنصر قالب حتى نستطيع بناء عناصر متشابهة في كل الصفات. هذا القالب يسمى في OOP بالكلاس اوالفصيلة, لابد من إنشاء القالب الفصيلة(class) قبل إنشاء أي عنصر(object).

نستطيع تمثيل الأشياء المحيطة بنا باستخدام الفصائل ونستطيع استخدامها من دون الحاجة إلى معرفة مكوناتها . فنحن نهتم بالوظيفة التي يؤديها ال class من دون أن نهتم بكيفية بناءه. بعد تعريف class نستطيع إنشاء أي عدد من العناصر objects من هذه الفصيله.

2. فوائد البرمجة بالأهداف

يمكن تلخيص الفوائد في النقاط الآتية:

- حماية البيانات Data hiding & protection: لا احد يستطيع تغيير البيانات الموجودة التي يعرفها class. فالمستخدم لـ class لايرى الكود ولكنة فقط يتعامل مع الدوال والبيانات التي يعرضها إلـ class
- الكبسلة Encapsulation: في OOP نقوم بإنشاء الدوال وتعبئة البيانات معا وتغليفها في إلـ class بحيث لايستطيع مستخدم ال class رؤية مكونات تلك الدوال والبيانات. فلكي نتعامل مع class يتم إنشاء أهداف objects تنتتمي إلي هذا الكلاس. و التي بدورها تتعامل مع الـ class. عملية إخفاء البيانات تخضع لقواعد يضعها المبرمج أثناء التصميم للفصيلة فابستطاعتة جعل بعض الدوال والبيانات قابلة لان يتعامل معها مستخدم الكلاس والبعض الأخر لايراه أولا يستطيع التوصل إليه

- الوراثة Inheritance: قبل ظهور OOP عند الحاجة إلى تطوير البرنامج بإدخال مزيد من الوظائف علية لدينا طريقتين لعمل ذلك:

الأولي ــ الحصول على الشفرة المصدرية للبرنامج ونقوم بالتغير فيها وهذا صعب جداً لأن من يصنع البرنامج لا يقوم بإعطاء المستخدمين الشفرة المصدرية.

الثانيه ـ القيام بإعادة كتابة البرنامج باكملة وهذا يؤدي إلى إضاعة الوقت والمال.

باستخدام مبادئ الوراثة نستطيع الوصول إلى البيانات أو الدوال الموجودة في الكلاسات الأخرى ونستطيع أن نكتب كلاس جديد نضيف إليه الدوال والبيانات الجديدة فقط مع استخدام الدوال والبيانات الموجودة في الكلاس القديم

تـــعـــريـــف ـ :

الوراثة هي العملية التي نقوم فيها بإنشاء كلاس جديد بحيث يأخذ (يرث) خصائص الكلاس القديم ثم نضيف خصائص جديدة للكلاس الجديد فيصبح الكلاس الجديد محتويا علي كل البيانات و الوال الموجوده في الكلاس القديم بالإضافه إلي بيانات و دوال جديده

- تعدد الاشكال :يعني اننا نستطيع ان نجعل دالة بأكثر من شكل لتؤدي اكثر من وظيفة طبقاً للهدف المرتبط بها .

3. مكونات الكلاس (الفصيلة) class

- الخصائص أو الصفات (Attributes) : وهي البيانات Data أو المتغيرات Member Variables
- السلوك او الوظيفة (Functions) : وهي الدوال الأعضاء Member Function

مثلا فالإنسان له خصائص :الطول ـ الوزن ـ لون العينين ــ لون الشعر... وله وظائف يقوم بها: يأكل –يشرب- ينام ...

بعد ان أعطينا مقدمه عن OOP سنتحدث عن الجافا

Programming language "Java"

1. مقدمة عن لغة java

هي لغة من اللغات الحديثة التي أنتجتها شركة Sun Microsystems في العام 1995م لتناسب التطبيقات الحديثة . وهي تناسب تطبيقات الانترنت وكذلك الأنواع المختلفة من التطبيقات:

Desktop Applications

Client-Server Applications

2. مزايا لغة Java

- لغة تلتزم بقواعد البرمجة بالأهداف: فلقد اتت البرمجة بالاهداف بفكرة جديدة هي انشاء عناصر متكاملة تحتوي على بيانات ودوال وهي الفصيله او الكلاس وهي وحدة بناء البرنامج.
- لغة لها بيئة تنفيذ خاصة بها: وهي بيئه خاصة قام بانشاءها مصممو Java ليجعلو من لغه جافا تعمل على جميع او معظم انظمة التشغيل وتقوم الفكرة على انشاء طبقة وسطية وكأنها برامج تشغيل للبرامج لكل نظام تشغيل. وبهذا كان من أهم مزايا لغة جافا انها تعمل على كثير من نظم التشغيل الموجودة بعد اعداد ماكينه جافا الخياليه Java Virtual Machine (JVM) الخاصة بنظم التشغيل. لتنفيذ أي برنامج مكتوب في لغة java لا بد من وجود JVM علي الجهاز.
- لغه تمتلك مكتبة كلاسات قوية : لغة جافا تحتوي على مكتبة كلاسات قوية محتويه علي الكثير من الكلاسات المطلوبة للتعامل مع الملفات وقواعد البيانات والشبكات والرسومات والحركة وكذلك التعامل مع الانترنت.
- لغة جافا لغة معتمده على c/c++ : عندما تم انشاء لغة جافا كان اساس بناءها لغة c++ وبالتالي لم تبدا من حيث بدأ الاخرون بل من حيث انتهى الاخرون .

3. أنواع التطبيقات في جافا

- Java Applet :وهو نوع من التطبيقات الذي يصمم خصيصاً للانترنت حيث يقوم المطور باعداد هذا البرنامج (Applet) ثم يتم استدعائه من خلال ملف html فيتم عرض هذا التطبيق من خلال صفحة الانترنت عندما يستدعي المستخدم هذاه الصفحة . والمشهور عن لغة جافا انها تعد برامج للانترنت.

- Java Application : هو تطبيق يشابه التطبيقات التي يتم انشاءها بجميع اللغات البرمجة ليعمل مع نظم التشغيل بعيداً عن شبكة الانترنت . فلغة الجافا توفر كثير من نقاط القوة في اعداد التطبيقات المكتبيه أو الخاصه بالشبكات مما شجع الشركات مثل شركة أوراكل على تبني هذه اللغة واستعمالها في اعداد التطبيقات من خلال الاداة وقاعدة بيانات oracle .

4. طرق كتابة برامج Java

هناك عدة طرق لكتابة برنامج جافا وترجمتها منها :

الطريقة الاولى : استعمال المكتبة JDK (Java Developing Kite) (مترجم جافا) والتي توفرها شركة SUN على موقعها مجاناً . باستخدام أي محرر للنصوص يمكننا كتابه برامج جافا و من ثم تنفيذ هذه البرامج. .يمكن تجميل (JDK) من موقع الشركة علي الانترنت.

أدوات المجموعة JDK

- الملف javac وهو الملف التنفيذي المستعمل في ترجمة الملف المصدر , كل ملفات جافا تمتلك الامتداد java مثلاً: program1.java

عند تنفيذ الامر javac program1.java يتم ترجمة البرنامج ونحصل علي ملف باسم program1.calss

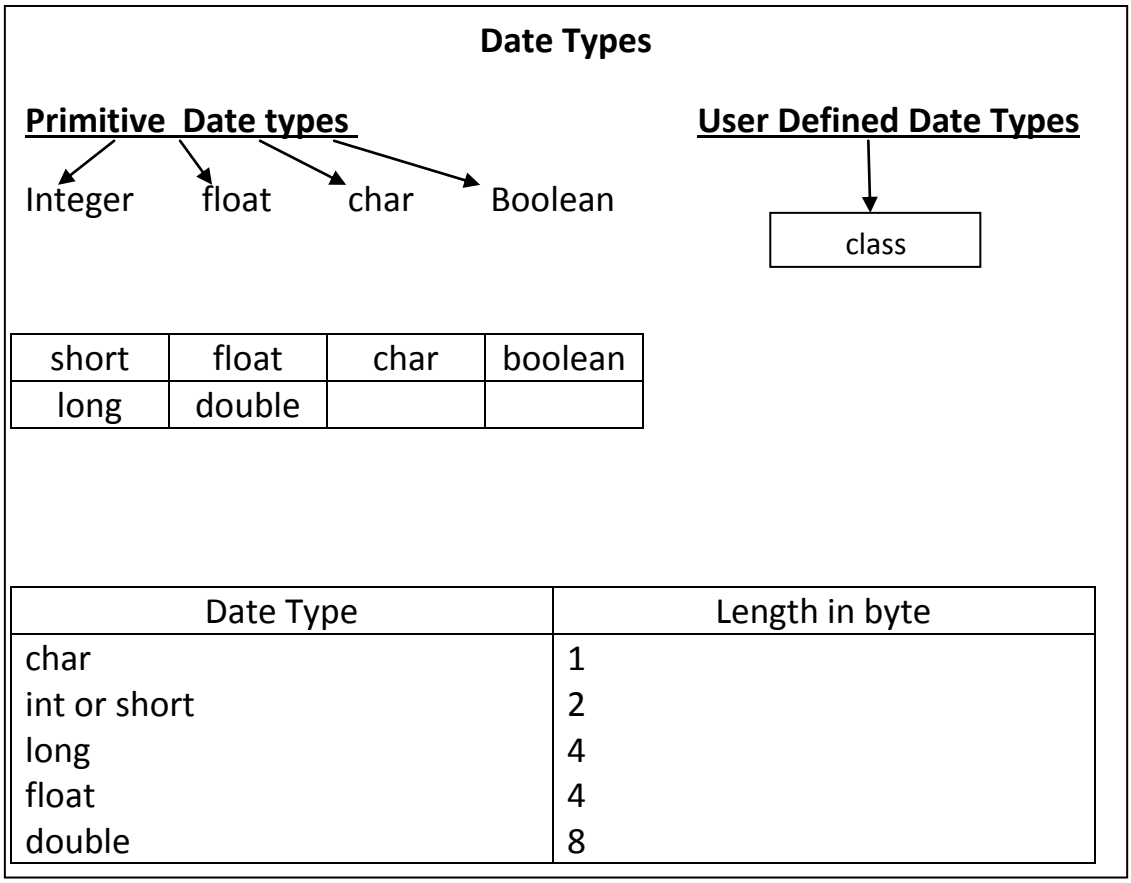
- الملف java - وهو البرنامج المسئول عن تنفيذ برامج java التي قد تم ترجمتها بواسطة الملف javac الي ملفات ذو امتداد class , لتنفيذ البرنامج نقوم بكتابة الامر التالي: Java program1.

الطريقة الثانية: استعمال برامج visual التي توفر جميع متطلبات إعداد تطبيقات java مثل البيئه السهله و المكتبه و المترجم و نستعمل هذه البرامج مثل استعمال أحد اللغات المشهوره مثل لغه visual basic من هذه البرامج المعده للغه جافا Forte , JDeveloper و غيرها.

محاضره (2)

أنواع البيانات و المؤثرات Data Types and Operators

1. أنواع البيانات Data types in java



short	float	char	boolean
long	double		

Date Type	Length in byte
char	1
int or short	2
long	4
float	4
double	8

2. المتغيرات Variables

المتغير عبارة عن عنوان لمكان في الذاكره نستطيع تخزين معلومة فيه أثناء عمل البرنامج و نستطيع تغيير القيمه المخزنه في هذا المتغير اثناء كتابه او نتفيذ البرنامج. حجم هذا المكان في الذاكره و القيمة المخزنة داخل هذا المكان تتوقف على نوع المتغير (Type).

يتم الإعلان عن المتغير بكتابة النوع (Type) ثم أسم المتغير مع إمكانية إعطاء قيمة ابتدائية ويشترط في الاسم أن لا يبدأ برقم ولا يكون بين حروفه مسافة أو قوس أو أحد الحروف الخاصة مثل # أو @ و غيرها من الشروط التي سنتطرق لها لاحقاً, يمكننا مثلا الإعلان عن المتغيرات بالطريقه التاليه :

```
int a;
float b;
double number=3.6;
boolean male=false;
char ch;
```

3. الجمل والتعبيرات Statements & Expressions

الجمل statements : هي أمر بسيط مكتوب بلغة برمجة (مثلاً java) يؤدي مهمة معينة

التعبير Expression : هي جملة تنتج قيمة (مثلا تنفيذ عملية حسابية ما)

```
int  x=5;       // Statement
int y=1;        //Statement
int z=x+y;      // Expression
```

4. الثوابت والتعليقات Constants &Comments

الثوابت Constant : وهي قيمة لا تتغير أثناء عمل البرنامج ولكي نعرف ثابت في لغة java نضيف كلمة final بالصورة التالية:

```
final  double pi=3.141536
```

التعليقات Comments: وهو نص لا يلتفت إلية المبرمج أثناء عملية الترجمة compilation فالتعليقات مهمة للغاية لأننا في بعض الأحيان ننسى كيفية عمل جزء معين من البرنامج أو يريد الآخرين فهم عمل برنامج قمنا بكتابته.

هناك نوعين من التعليقات:

A. تعليق السطر الواحد, لكي نكتب تعليقا مكونا من سكر واحد يجب كتابه علامة // و بعدها نكتب نص التعليق
مثلا // This is single line comments

B. تعليق مكون من عدة أسطر, لكي نكتب تعليقا مكونا من عدة اسطر لابد من حصر هذا التعليق بين الرمزين التاليين /* */ مثلا

```
/* This is multiple comment in my first  java
program*/
```

5. المؤثرات Operators

هي الرموز التي تربط بين المتغيرات والثوابت لإنشاء علاقة ما أو معادلة ما.

أنواع المؤثرات:

A. المؤثرات الحسابية Arithmetic Operators
(+،-،++،--، *، / ، %،...)

B. المؤثرات المنطقية Logical Operators
(&& , || , !،...)

C. المؤثرات العلائقيه Relational Operators
(==,<= , < , > ، >=،!=،...)

مثال 1

Example 1	العملية	النتيجة
	10>8	1
	10==8	0
المؤثرات العلائقية	10!=8	1
	10>=8	1

مثال 2

Example 2	العملية	النتيجة
	10>8 && 9>7	1 (True)
المؤثرات المنطقية	10<8 || 9>7	1(True)
	!(10!=8)	0 (false)

مثال 3 بفرض ان قيمة a =6

Example 2	العملية	النتيجة
	a=a/3 أو (a/=3)	2
المؤثرات الحسابية	a=a/3 ⇨a=2 a/=3 ⇨ a=2	
	a=a+1 أو (a++)	7
	a=a-1 أو (a--)	5

6. أسبقية التعامل مع المؤثرات

A. الحسابية

() ⇨ ++ ⇨ --⇨ * ⇨ / ⇨ % ⇨ + ⇨ -

Example int answer

answer= 3+5*3-10%4

1. 3 + 5 * 3 - 10 % 4

15

2. 3 + 5 * 3 - 10 % 4

2

3. 3 + 15 =18

4. 18 - 2 = 16

⇨ **The result is 16**

B. المنطقية و العلائقيه

! → <,>,<=,>=,== → && → ||

Example

1. **(3>5) && (2==1) || (7==7)**

   true

2. **(3>5) && (2==1) || (7==7)**

   false

3. **(3>5) && (2==1) || (7==7)**

   true

4. **true && false**

   false

5. **false || true**

   true

   ⇨ **The result is true**

محاضره (3)

مكونات لغة جافا Components of java

1. برامج جافا

نبحث في مكونات برامج جافا والتي تتكون من متغيرات وثوابت وكلمات محجوزة وعمليات حسابية ومنطقية وعلائقية وعمليات إسناد .

Variables, Constants, Reserved words, Arithmetic & Logical & and Relational operators and Assignment.

في المحاضرة السابقة تحدثنا عن المتغيرات والثوابت وكيفية الاعلان عنها.

الصيغة العامة للإعلان عن متغير:

Type variableName [=init-value];

Type variable1 [, variable 2, variable 3[=init value]…];

For example:

int number 1;

float number2 = 10.6;

الصيغة العامة للإعلان عن ثابت:

final Type constantName = constantValue;

For example:

final double pi= 3.14;

final int days = 7;

تحدثنا أيضاً عن أنواع البرامج المكتوبة في جافا .

1. **Application program** : وهي برامج مكتوبة بلغة *java* و تشبه البرامج المكتوبه باللغات الأخري. هذه البرامج يمكن تنفيذها مباشرة من خلال بيئة الجافا باستخدام مفسر جافا *java interpreter*
2. **Applet program** : وهذه البرامج يتم تنفيذها من خلال متصفحات الانترنت مثل *Netscape Navigator & Internet Explorer* وتنفذ على أي حاسب ومع أي متصفح مما يؤكد خاصية النقل أو الحمل لبرامج جافا.

حروف لغة جافا :

1. الحروف الهجائية الانجليزية الكبيرة والصغيرة مع العلم بأن المترجم يفرق بين الأحرف الكبيرة والأحرف الصغيرة a,b,c,...X,Y,Z,...A,B,C
2. الارقام العربية ...0,1,2,3,
3. الرموز الخاصة ..%, # ,@ ,!,$
4. المعاملات ...!,&&,=>,>,<,*,+

تعريف

المعرف Identifier :هو إسم لثابت او متغير او دالة مع مراعاة الشروط التاليه عند التسميه:

1. ان يتكون هذا الإسم من حروف كبيره أو صغيره أو من حروف وأرقام بشرط أن يبدأ بحرف أو بالشرطة التحتية under score
2. أن لا يحتوي علي أي رموز خاصة
3. ان لايكون ضمن الكلمات المحجوزة للغة جافا
4. يفضل ان يكون للمعرف إسما دالا علي وظيفته في البرنامج و واضحا

For example:

```
int theResult;
final double pi= 3.14;
 int adding(int number1,int number 2)
{
   //Function Body
}
```

بعض الكلمات المحجوزة في جافا :

public, return, main, boolean, break, for, if, short, static, void, package, new, class, private, protected, switch, while, …

ملاحظات

- الحروف الصغيرة لاتساوي الحرف الكبيرة في جافا
- عند حفظ البرنامج في جافا يجب ان يكون اسم الملف هو نفسه اسم الكائن متبوعاً "java"

2. البرنامج الاول في جافا

برنامج بسيط	الصيغة العامه لبرنامج جافا بسيط

//my first program in java " Welcom1.Java"
Public class Welcom1
{
Public static void main(string arg[])
{
System.out.println ("welcome to java .this is my first program")
}
}
	public class className
	{
	Data member
	Function member
	}

- كل تطبيق من تطبيقات جافا يجب أن يحتوي على الدالة main و تأخذ الشكل العام التالي:

```
public static void main (string args [])
{
    Main body
}
```

في حالة عدم وجود main في البرنامج لا ينفذ هذا البرنامج أي شيئ. كل دالة من دوال البرنامج تمتلك اسم بعد هذا الاسم تأتي (parameters list) ثم القوس } و الذي يشير إلي بداية جسم الداله وتنتهي بالقوس { و الذي يشير إلي نهايه هذه الدالة.

- الجملة System.out.println( ): تعرف بأنها جملة الخرج القياسية وهذه الجملة تقوم بإظهار كل ما داخل القوسين سواء كانت نصوص أو أي بيانات أخري علي شاشة الخرج وعندما تنهي هذه الداله عملها, ينتقل المؤشر الي بداية سطر جديد علي الشاشة.
- في نهاية كل جملة من جمل برنامج جافا توضع (;) الفاصلة المنقوطة وهذه الفاصلة المنقوطة تحدد نهاية الجملة

3. ترجمة وتنفيذ البرنامج الاول

هناك طريقتان لتنفيذ البرامج المكتوبه بلغة جافا:

الطريقه الاولي : تنفيذ البرنامج باستخدام نافذة الاوامر Command prompt

بفرض أننا نريد تنفيذ البرنامج welcome1 ,يجب تغيير الفهرس الى الفهرس الذي تم حفظ البرنامج فيه ثم كتابة الامر javac welcome1.java , إذا كان هناك أخطاء بنائيه(Syntax error) فإن هذه الأخطاء سوف تظهر في نافذة الأوامر موضحاً فيها رقم السطر ومكان الخطأ وتفسير محتمل للخطأ, في هذه الحالة يجب تصحيح هذه الاخطاء في البرنامج ثم اعادة الامر السابق مرة اخرى حتى اذا اصبح البرنامج بدون اخطاء في هذه الحالة يقوم المفسر(Interpretator) بانشاء وحفظ ملف جديد يسمى welcome1.class هذا الملف ينتج عن ترجمة جمل لغة جافا بواسطة المترجم إلي Byte code وهي صورة أخري للبرنامج وهي الصورة التنفيذية للبرنامج ولتنفيذ البرنامج من خلال نافذة الاوامر نكتب الامر java welcome1 في نافذة الاوامر واذا لم يتوفر الملف ذو الامتداد class. فإن المفسر لايستطيع تنفيذ البرنامج ويعطي رسالة خطأ .

الطريقه الثانيه : تنفيذ البرنامج من خلال بيئة العمل(Visual)

يتم كتابة البرنامج وترجمتة وتفيذة من خلال بيئة عمل مرئيه مثل: (forte) او (jdeveloper)

لنقوم بتغيير البرنامج الأول بحيث يتم اظهار الجملة welcome to java .this is my first program في سطر واحد باستخدام جملتين .

```
//welcome 2.java
Public class welcome2
{
  Public static void  main  (string  args  [])
    {
      System. out print ("welcome to java .") ;
      System .out println("This is my first program");
    }
}
```

الفرق بين print ,println

print تقوم بعملية الطباعه ثم تبقي المؤشر في نهاية السطر الذي طبع علي الشاشه

prinln تقوم بعملية الطباعه ثم تنقل المؤشر الي بدايه سطر جديد علي الشاشه

لنقوم بتغيير البرنامج السابق بحيث يظهر عدد من الاسطر باستخدام جملة واحدة

```
//welcome 3.java
Public  class welcome3
{
  Public   static void  main  (string   args [])
  {
    System .out .println("welcome \n  to  \n  java \n. This is  my first   program);
   }
}
```

لتنظيم عملية إخراج البيانات في دالة الإخراج تستعمل حروف الهروب Escape characters وتقوم بوظائف معينة عند أخراج البيانات على الشاشة ، هذا الحروف تكون مكتوبة بين علامتي تنصيص.

بعض هذه الحروف :

الحرف	الوصف
\n	الانتقال الي سطر جديد أي انه يضع المؤشر في بداية السطر التالي
\t	مسافة افقية tab أي تحريك المؤشر مسافة tab في السطر
\r	يضع المؤشر في بداية السطر الحالي
\\	اظهار \ شرطة خلفية في الخرج
\"	اظهار علامة تنصيص مزدوجة

4. برنامج حساب و اخراج عمر شخص بالأيام و الساعات و الواني.

```
//Example1:
Public  class Example1
{
  public  static void  main  (string   arg [])
  { //  declaration section
     short age_Inyears = 80;
```

```
    int age_Indays , age_Inhours, age_Inseconds;
    string name =" Ahmed";
    System.out.println("\n welcome  Mr: "+name);
    age_Indays =age_Inyears*365;
    System.out.println("\n your age in days ="+ age_Indays);
    age_Inhours  = age_Indays *24;
    age _Inseconds=(long) age_Inhours *3600;
    System. Out ("\n your  age in second =" + age _Inseconds);
  }
}
```

5. إظهار نص في صندوق الحوار

تحدثنا عن اظهار النصوص في نافذة الاوامر, إلا أن الكثير من التطبيقات تستخدم صناديق الحوار لاظهار النصوص وبخاصة متصفحات الانترنت.

صناديق الحوار: هي عبارة عن نافذة يتم فيها إظهارالرسائل الموجهة للمستخدم أو التي تعطي خرخا من البرنامج فالكلاس JOptionPane يمدنا بالدوال methods التي تساعدنا في إظهار صناديق الحوار.

مثال

```
//welcome4 .java
import.javax.swing.JOptionPane ;    //impot class JOptionPane
public  class   welcome4
{
 public static void main(  string args [])
 {
   JOptionPane.showMessageDialag (null,"welcome\n to\n java !");
   System .exit (0);  \\terminate application
  }
}
```

الداله showMessageDialog تاخذ معاملين **المعامل الاول** يشير الي المكان الذي سيظهر فيه صندوق الحوار فاذا كانت قيمة هذا المعامل null فان صندوق الحوار سيظهر في منتصف الشاشه

المعامل الثاني هو محتوي صندوق الحوار أي ما سيظهر في هذا الصندوق.

الداله (0) exit الموجوده في الكلاس System تستدعي لإنهاء التطبيق و القيمه (0) تعني أن التطبيق تم انهاؤه بنجاح. يجب استخدام هذه الداله في كل التطبيقات التي تستخدم GUI (Graphic Unit Interface).

من مميزات جافا احتوائها على الكثير من الفصائل الجاهزة التي يمكن للمبرمجين استخدامها, هذه الفصائل(classes) يتم تنظيمها وتجميع المتعلق بعضه ببعض في حزم(Package) ,هذه الحزم هي التي تكون مكتبة الجافا Java Application Programming Interface (Java API) .

يوجد نوعين من الحزم :

- الحزم الأساسيه (core packages) و سمها يبدأ بـ java
- حزم الامتداد (Extension packages) و إسمها يبدأ بـ javax

فمثلا الحزمه swing تحتوي علي الكثير من الفصائل الخاصه بالرسومات و التعامل مع المستخدم من خلال البيئه الرسوميه (GUI) و التي تسهل عمليه إدخال و إخراج البيانات من خلال مربعات الحوار.

مثال: برنامج يطلب من المستخدم ادخال العدد الأول ثم العدد الثاني عبر صناديق الحوار التي تظهر للمستخدم ثم يقوم هذا البرنامج بضرب هذين العددين و إظهار الناتج عبر صندوق حوار أيضا.

```
// Multiply.java.
// Java extension packages
import javax.swing.JOptionPane; // import class
public class Multiply
{
   // main method begins execution of Java application
   public static void main( String args[ ] )
   {
      String firstNumber;      // first string entered by user
      String secondNumber; // second string entered by user
      int number1;             // first number to add
      int number2;             // second number to add
      int result;              // number1*number2

      // read first number from user as a string
      firstNumber = JOptionPane.showInputDialog( "Enter first
         integer" );
```

```
        // read second number from user as a string
        secondNumber = JOptionPane.showInputDialog( "Enter second
           integer" );

        // convert numbers from type String to type int
        number1 = Integer.parseInt( firstNumber );
        number2 = Integer.parseInt( secondNumber );

        // multiply the numbers
        sum = number1 * number2;

        // display the results
        JOptionPane.showMessageDialog(null, "The multiply result is: "
           + result, "Results", JOptionPane.PLAIN_MESSAGE);

        // terminate application
        System.exit( 0 );

    }      // end method main
}          // end class Multiply
```

محاضره (4)

جمل التحكم وجمل التكرار Conditionals and Loops

1. جمل التحكم Control statements

في البرامج السابقة نفذت العمليات بطريقة متسلسلة بمعنى تعليمة بعد أخرى بطريقة متتابعة.

بواسطة التراكيب الشرطية نتمكن من التحكم بتنفيذ خطوات البرنامج .

❖ **التركيب الشرطي البسيط (if)**

تقوم if بتحقيق أو تنفيذ عمليات ما إذا كان الشرط صحيحاً فقط.

الصيغة العامة:

- if (condition) statement;

  أو
- if(condition)
  {
  block of statement
  }

For Example: if (x>y) System .out. println("x is greater than y");

❖ **التركيب الشرطي الكامل (if....else) إذا ....وإلا**

نستخدم هذه التركيب الشرطي لاختيار شرط معين, فإذا كان صحيح ينفذه ما بعد if وإلا ينفذ ما بعد else

الصيغة العامة:

- if (condition)
  statment1;
  else
  statement 2;

  أو
- if (condition)
  {
  Block of statements
  }
  else
  {
  Block of statements

```
}
```

For Example: If(x>y)

```
{
  System .out .println(" x is greater than  y");
  System .out.println("so ok"  );
 }
else
   System.out  .println( " x  is less  than  y");
```

- **تراكيب المؤثر الشرطي (Conditional operator ?)**

الصيغة العامة:

```
variableName = (condition)?  statement1 :   statement2;
```

عمل المؤثر الشرطي(؟) مشابه لعمل التركيب الشرطي if..else

❖ **التركيب الانتقائي Switch**

التراكيب الشرطية السابقة تتم فيها المقارنة بين قيمتين حيث تكون النتيجة أما صائبه أو خاطئة ولكن في بعض الأحيان علينا أن نقارن بين عدد من الحالات تبعاً لشروط مختلفة في هذه الحالة نستطيع استخدام التركيب (switch) والذي يقوم باختيار القيمة الصحيحة من عدد من القيم وحسب الشرط الموجود في الصيغة

الصيغة العامة:

```
switch (variableName)
{
case value _1: statement_1;
                  break;
case value_2:  statement_2;
                  break;
.
.
case value_N: statement_N;
                  break;
```

```
default :default  statement;
}
```

variableName: هو ذلك المتغير الذي يتم اختياره ويكون من النوع صحيح(int) أو الحرفي (char)

case : تمثل نوع الحالة المناسبة بعد احتساب المتغير

value: تمثل قيمة المتغير ويمكن أن تكون عدداً موجباً او سالباً من النوع الصحيح (int) أو أن تكون حرفا (char)

break: وهي عبارة توقف وتستعمل عند أخر كل مجموعة جمل من جمل الحالة (case) لتفادي الاستمرار في بقية الحالات وإذا لم تستعمل بعد أي حالة فان التعبير ينتقل إلى الحالة التالية لهذه الحالة

default: وتعني حالة إسقاط وهي اختيارية (يمكن عدم ذكرها في البرنامج) وتنفذ عندما لا تتوافق قيمة المتغير مع أي قيمة value1 , value2, …, valueN

For Example:

```
class  convNumbers
{
  String   convFunc (int  v)
  {
     switch  (v)
      {
         case 0 : return "zero";
         case 1 : return "one";
              :
              .
          case 9: return "nine";
          default : return " ";
        }
  }
  public  static void main (string args[ ])
  {
```

```
    convNumbers   CN= new   convNambers ();
    System .out .println (CN.confunc(2));
    String  word= CN.convFunc (3) + CN.ConvFunc (4) + CN.convFunc (5);
    System.out.println (word);
  }
}
```

2. الحـــلقـــات Loops(for, while ,do-while)

تعريف الحلقات **loops** : هي مجموعة من الجمل التي تستعمل لتكرار تنفيذ الأوامر أكثر من مرة وهي مهمة جدا لكتابه البرامج ً وهذه الجمل هي :

- ✓ جملة التكرار for
- ✓ جملة التكرار while
- ✓ جملة التكرار do- while

❖ جملة التكرار for

تستخدم for لتكرار تنفيذ عملية أو عمليات عدد محدد من المرات وتأخذ الصيغة العامة ل for الشكل التالي:

- for(variableName=init_value; condition; increment)
  statements;
  أو
- for(variableName=init_value; condition; increment)

```
{
 Statement_1;
 ….
 ….
 Statement_N;
 }
```

إذا كنا نريد تكرار عدة جمل فلابد من كتابتها بين القوسين {}

شرح الصيغه العامه:

حيث أن:

init-value : هي القيمة الابتدائية

condition : هو شرط التكرار

increment : هي قيمه أو جمله الزياده
for(..) :الجمله for تعني اعطي لمعامل الزياده القيمة الابتدائية (init-value) و
طالما ان الشرط (condition) صائب نفذ الجمل الموجوده في جسم الحلقه for ثم قم بزياده
أو نقصان قيمه معامل الزياده أي نفذ الجمله increment

يتم استعمال for مع المصفوفات بكثرة لتجنب التعامل مع كل عنصر من عناصر المصفوفه على حده

مثال: نعيد المثال السابق (طباعة اسماء اشخاص مخزنة في مصفوفة) باستخدام for

```
class  friends
{
  String [ ] Names ={"Omar", Alaa", ”Mohmomed", "Saad"};
  void  printNames()
  {
    for( int  i=0; i< Names.Length ;i++)
         System.out.println(Names[i]);
   }
   public  static void main (string  args[]);
   {
     friends f=new friends ( );
     f.printNames( );
    }
}
```

تغيير مقدار الزيادة في الحلقة for

في يعض الأحيان نحتاج إلي أن يكون مقدار الزيادة أكثر من واحد حينئذ يلزم تغيير هذا التعبير إلى الصورة المطلوبة مثلا اذا كنا نريد أن يكون مقدار الزياده (2)

مثلا: for(i=2; i<=10; i+=2)

تغيير معدل الزيادة بالسالب

أحياناً نحتاج إلى أن يكون معدل الزيادة بالسالب لحل بعض المسائل

مثلا: for(i=10; i>0;i--)

❖ جملة التكرار while

تستخدم الحلقة أو الدوارة (while) لتكرار تنفيذ جملة أو مجموعة من الجمل عدد غير معلوم من المرات يتوقف هذا العدد على شرط موجود بداخل الحلقة (while)

الصيغة العامة لـ (while):

```
while (condition)
{
  statements
}
```

سيتم تنفيذ العمليات ما دام ان الشرط صحيح.

مثال:

```
class  whileloop
  {
    public  static void  main (string args[])
      {
        int  [] array1={7,8,9,10,11,12};
        int  i=0;
        while (i < array1.length)
        {
            System.out .println(array1[i]);
            i++;
         }
    }
}
```

❖ جمله التكرار do.. while

التكرار باستعمال do – while : يختلف هذا التكرار عن while في انه يبدأ بتنفيذ الجمل ثم يختبر الشرط في نهاية التركيب فإذا كان صحيح يعاد التكرار مرة اخرى والا يتوقف وبالتالي يتم تنفيذ الجمل مرة واحدة على الاقل حتى لوكان الشرط خاطئاً وتاخذ الصورة التالية

الصيغه العامه:

```
do
{
  statements
 } while (condition);
```

مثال

```
class  doloop
 {
   public static void main (string  args [])
     {
        Int  x= 1;
          do
           {
            System .out.println("This is number is: "+x)
            x++;
           } while (x<=20);
       }
  }
```

محاضره (5)

المصفوفات Arrays

المصفوفات و جمل التحكم و جمل التكرار هذه الموضوعات الثلاثه لا تخلو منها لغة برمجة

المصفوفات Arrays

نعرف أن اسم المتغير يحمل قيمة واحدة فقط قابلة للتغيير اثناء تنفيذ البرنامج ,فطرق التعامل مع أسماء المتغيرات و الثوابت المختلفه التي وردت سابقا تعد صالحة للتعامل مع عدد محدود من هذه المتغيرات و الثوابت , سواء في عمليات الإدخال و الإخراج أو العمليات الحسابيه و المنطقيه. فعندما يصبح لدينا مجموعة كبيره من البيانات التي لها نفس النوع أي أن عدد القيم الثابته أو المتغيره التي نريد التعامل معها كبيرا نسبيا تصبح تلك الطرق غير عمليه. هذه المجموعة من القيم تحتاج إلى عملية تخزين في متغير واحد لكي يسهل التعامل مع هذه القيم لذالك يتم استخذام مفهوم المصفوفه.

تعريف المصفوفة : هي عبارة عن سلسلة من العناصر التي لها نفس النوع والموضوعة في الذاكرة بشكل متجاور والتي تحمل اسماً واحداً بحيث يسهل الرجوع إلى أي من هذه العناصر عن طريق رقم هذا العنصر index في هذه السلسله فالعنصر الأول في المصفوفه يمتلك الرقم صفر و العنصر الثاني يمتلك الرقم 1 و هكذا .., نستطيع التعامل مع عناصر المصفوفه عن طريق اسم المصفوفه و أرقام عناصرها في داخل هذه المصفوفة.

لانشاء مصفوفة لابد من تنفيذ 3 خطوات:

a) الاعلان عن متغير المصفوفة Array Variable
b) تعريف (انشاء) عنصر المصفوفة Array object
c) تخزين القيم او البيانات داخل المصفوفة .

تنقسم المصفوفات إلى قسمين هما:

1- مصفوفات ذات البعد الواحد one dimensional arrays
2- مصفوفات متعددة الابعاد multi-dimensional arrays

1. المصفوفات الاحادية :

a) الاعلان عن متغير مصفوفة يتم بطريقتين:

1. Type arrayName [ ];

   For example: string jobs[ ];

2. Type [ ] arrayName;

   For example: string [ ] jobs;

b) تعريف عنصر المصفوفة يتم بطريقتين:

الطريقه الاولي: استعمال كلمه new لانشاء عنصر المصفوفه, في هذه الحاله لابد من تحديد عدد العناصر

Type objectName = new Type [number of elements]

For example: string jobs =new string [10];

يمكن كتابه السطر السابق في سطرين بالشكل التالي:

string jobs;

jobs = new string [10];

الطريقة الثانية : بدون استعمال new وذلك باعطاء المصفوفة قيم ابتدائية

Type [ ] arrayName ={"value1",value2",…,"valueN"};

string [ ] jobs ={"Engineer","Doctor",Teacher"};

c) التعامل مع عناصر المصفوفة :

يتم التعامل مع عناصر المصفوفة بكتابة اسم المصفوفه و رقم العنصر في هذه المصفوفة arrayName[number] , مع ملاحظة ان ترتيب عناصر المصفوفة تبداء في القيمة صفر

مثلاً : int numbers [6];

هنا نعلن عن مصفوفة اسمها numbers عدد عناصرها = 6 هذه العناصر من النوع الصحيح int

الشكل التالي يظهر كيف يتم حجز مواقع في الذاكرة لعناصر المصفوفة numbers

0	1	2	3	4	5

هذه الارقام تبين ارقام عناصر هذه المصفوفه و ليس قيم هذه العناصر

حيث تحجز مساحة في الذاكرة للمصفوفة حسب نوع البيانات المخزنة (تكلمنا سابقا عن حجم البيانات في الذاكره بالبايت للأنواع المختلفه من البيانات) في المصفوفة فتحجز خانات متجاوره كل خانة من هذه الخانات ترقم برقم, الترقيم يبدأ من الصفر ويسمى رقم العنصر بالدليل (index) فالعنصر الاول من المصفوفة دليله (0) والعنصر الثاني من المصفوفة دليله (1) والعنصر الثالث من المصفوفة دليله (2) وهكذا... .

مثال :

String [ ] jobs= new string [10];

Jobs [10] ="program4"

في الجملة الثانية عند الترجمة يعطي المترجم رسالة خطاء لان عدد العناصر المعلن عنها =10 jobs[10]; تعني باننا نتعامل مع العنصر الحادي عشر.

لحساب عدد عناصر المصفوفة يمكن استعمال الدالة LENGTH هذه الداله تستدعي بالشكل التالي:

int variableName= arrayName.length;

For example: int number =jobs .length ;

مثال علي المصفوفه الاحاديه البعد

```
class friends
{
  String [] Names ={"Omar",Alaa",”Mohmomed","Saad"};
  void printNames()
  {
    int E=0;
    System.out.println (Names[E]);
    E++;
    System.out.println (Names[E]);
    E++;
    System.out.println (Names[E]);
    E++;
    System.out.println (Names[E]);
  }
```

```
  public  static void main (string  args[]);
  {
    friends f=new friends ( );
     f.print Names( );
   }
}
```

2. المصفوفات ثنائية الابعاد Two dimension Arrays

المصفوفات ذات البعدين (two dimensional arrays) تتكون من صفوف وأعمدة

الصيغة العامة للإعلان عن مصفوفة ذات بعدين

```
Type arrayName [][] = new Type [size1][size2]
```

حيث أن:

Type : نوع البيانات في المصفوفة

arrayName :اسم المصفوفة

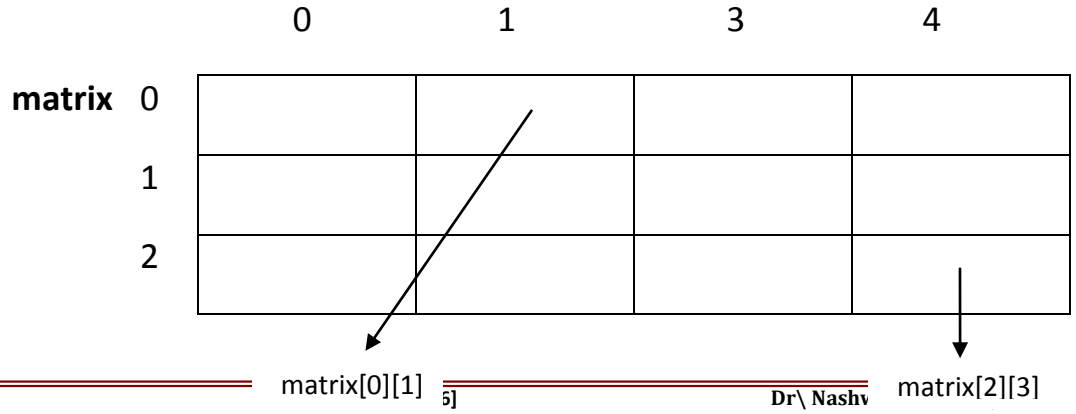
size1 : عدد الصفوف في المصفوفة

size2 : عدد الأعمدة في المصفوفة

مثال : `int  matrix [ ][ ] = new int [ 3][ 4];`

في هذا المثال تم الاعلان عن مصفوفة اسمها matrix والتي تتضمن 3 صفوف (اسطر) واربعة اعمدة أي أن عدد عناصر المصفوفة = 12 عنصراً يبدأ ترقيم عناصر المصفوفة من الرقم 0 ويسمى هذا الرقم الدليل (index) فالعنصر الاول في المصفوفة الثنائية يكون دليلة [0][0] والعنصر الواقع في السطر الأول وفي العمود الثاني يكون دليله [1][0] وهكذا

الشكل التالي يوضح دليل عناصر والمصفوفة الثنائية :



القيم الابتدائية للمصفوفة ذات البعدين (initial values)

يمكن تخصيص قيم ابتدائية لمصفوفة ذات بعدين بطريقتين

الاوليً : اثناء الاعلان عن المصفوفة و بدون استخدام new والصيغة العامة هي :

```
Type arrayName[ ][ ]={{list of first row elements},{list of second row
elements},...,{list of last row elements}};
```

مثال : لتكن لدينا المصفوفة :

```
int matrix [ ][ ] = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};
```

إذاً

matrix[o][0]=1 , matrix[0][1]=2, matrix[0][2]=3, matrix[0][3]=4

matrix[1][0]=5 , matrix[1][1]=6, matrix[1][2]=7, matrix[1][3]=8

matrix[2][0]=9 , matrix[2][1]=10, matrix[2][2]=11, matrix[2][3]=12

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

الثانية : في داخل البرنامج بكتابة اسم عنصر المصفوفه و إعطاؤه قيمه

مثلا: matrix [2][1] =80;

يتم التعامل مع المصفوفة ذات البعدين بتحديد رقم الصف ورقم العمود لعنصر المصفوفة المراد التعامل معة مع ملاحظة ان ترقيم العناصر في المصفوفة يبدا من الرقم صفر .

وللتعامل مع المصفوفة ذات البعدين نستخدم for المتداخلة

مثال: برنامج يقوم باعطاء قيم لمصفوفتين ذات بعدين ثم يقوم باخراج عناصر هذه المصفوفات علي الشاشة بشكل رياضي.

```
public class twoDemArray
{
   public static void main(String[] args)
   {
     int array1[][] = { { 1, 2, 3 }, { 4, 5, 6 } };
      int array2[][] = { { 1, 2 }, { 3 }, { 4, 5, 6 } };
      System.out.println("The contents of the array1 are:");
      for(int i=0; i<array1.length; i++)
      {
        for(int j=0; j<array1[i].length; j++)
        System.out.print("\t"+array1[i][j]+"\t");
        System.out.println();
       }
       System.out.println("The contents of the array2 are:");
       for(int i=0; i<array2.length; i++)
       {
          for(int j=0; j<array2[i].length; j++)
          System.out.print("\t"+array2[i][j]+"\t");
          System.out.println();
         }
   }// end main
 } // end class
```

بلوك الاومر Block of statements

بلوك الاوامر:عبارة عن مجموعة من الجمل او الأوامر المحصورة بين قوسين {} كما في الشكل التالي:

```
{
    Statement  1;
          :
    Statement     n;
 }
```

والفائدة من انشاء بلوك الاومر في برنامج ما هي امكانية الاعلان عن متغيرات داخل البلوك خاصة بة

محاضره (6)

الطرق Functions

1- ما هي الطرق (Methods)

الطريقة هي عبارة عن مجموعة من الجمل وتعرف بجسم الطريقة (Method body) حيث يكون لها اسم معين حيث يخضع هذا الإسم لشروط تسمية المعرفات و تؤدي كل دالة وظيفة محددة داخل البرنامج وتعتبر عضودالة في الكلاس الذي عرفت فيه. قد يوجد عدة دوال في داخل صنف ما لها نفس الإسم فيستطيع المترجم التفريق بين هذه الدوال من خلال التوقيع (signature) الخاص بها وهو عبارة عن اسم الطريقة ونوع المعاملات وترتيبها بالإضافة إلى نوع البيانات الراجعة منها .

2- صنف العمليات الحسابيه (math class)

يحتوي هذا الصنف على العديد من الطرق التي تقوم بالعمليات الحسابيه الشائعة , وتتم عملية استدعاء الطرق بكتابة اسم الصنف متبوعاً بنقطه بعدها اسم الطريقة ثم قائمة المعاملات داخل اقواس دائرية كما يلي :

```
Calss_Name.Method_Name (argument list)
```

مثال :

```
System.out.println(math.sqrt(4.0))
```

تقوم هذه الجملة باستدعا الطريقة (sqrt) الموجودة في الصنف (math) والتي تأخذ معامل واحد من نوع (double) مثلاً عند قيمة المعامل (4.0) نتيجة تنفيذ هذه الجملة ستكون طباعة (2.0)

الجدول التالي يحتوي على بعض الطرق الموجودة في الصنف (math) :

الطريقة	وصف الطريقة	مثال
abs (x)	القيمة المطلقة ل x	Math.abs(6.2)→6.2 Math.abs(2.4)→2.4

Math.ceil(5.1)→6 Math.ceil(-5.1)→-5	تقرب x الي اقل عدد صحيح ليس اقل من x	السقف ceil(x)
Math.Floor(5.1)→5 Math.Floor(-5.1)→-6	تقرب x إلى اكبر عدد صحيح ليس اكبر من x	الأرضية floor(x)
Math.max(6,7)→7	تعطي القيمة الاكبر من x,y	max(x,y)
Math.min(-7,-8)→-8	تعطي القيمة الأصغرمن x,y	min(x,y)
Math.pow(6,2)→36	x مرفوع للأس y	pow(x,y)
Math.sqrt(9)→3	الجذر التربيعي ل x	sqrt(x)
Math.random()→0.8732	تكون رقم عشوائي بين 0 و 1	random()

برنامج توضيحي للعمل مع الطرق الموجودة في الصنف math :

```
//useMath.Java

public class useMath {

  public static void main(String[ ] args) {

    System.out.print("the square root of
81="+Math.sqrt(81));

    System.out.print("the absolute value of 20
="+Math.abs(20));

    System.out.println("the absolute value of -10
="+Math.abs(-10));

    System.out.println("5 to power 3 ="+Math.pow(5, 3));

 }//end of main
```

3- فوائد استخدام الطرق :

1) ثبت عملياً أن أفضل طريقة لحل مسألة هي تقسيم هذه المسألة إلى وحدات صغيرة ويعرف هذا المبدأ بمبدأ فرق تسد (Divide and Conquer) مما يسهل علينا كتابة البرنامج وتتبعه وامكانية فهمه وصيانته بسهوله .
2) استخدام الطرق المعرفة مسبقاً يوفر علينا كتابة البرنامج وذلك من خلال إعادة استخدام هذه الطرق دون الحاجة إلى كتابتها مرة أخرى (Reusability)
3) تفادي تكرار كتابة الجمل في البرنامج من خلال كتابة الجمل التي نحتاج إلى تكرارها في البرنامج داخل طريقة ما (method) ومن ثم نقوم باستدعاء هذه الطريقة عن طريق اسمها في أكثر من موقع في البرنامج .

4- تعريف الطرق واستدعائها في Java

الشكل العام لتعريف الطرق :

```
Access_specifier static Return_type Method_Name (parameters)
{
Method body
}
```

Access_specifier	وهو محدد الوصول ويمكن أن يكون واحد من المحددات التالية: private, public, protected
Private	أي بمعنى خاص بحيث تكون الطريقة (Method) مرئية فقط داخل الصنف (class) الذي عرفت فيه
Public	أي بمعنى عام , و تكون الطريقة مرئية في أي مكان في البرنامج
Static	أي بمعنى ثابت وتستخدم لتعريف الطرق ليتم استخدامها داخل الصنف الذي عرفت فيه فقط
Return_type	وهذا يحدد نوع البيانات التي ترجعها الطريقة عند استدعائها ويمكن للطريقة أن لا ترجع أي قيمه , وفي هذه الحالة يجب أن يكون نوع البيانات المرجعة (void)
Method_Name	وهو اسم الطريقة ويجب مراعاة الشروط الخاصة بتحديد أسماء المعرفات عند اختيار اسم الطريقة
Parameters	وهي المعاملات وعند تعريف الطرق تسمى هذه المعاملات بالمعاملات الشكلية (Formal parameters)

	ويمكن استخدام هذه المعاملات في جسم الطريقة كمتغيرات بالإضافة إلى المتغيرات المحلية (Local Variables) التي تعرف داخل جسم الطريقة , وعند استدعاء الطريقة بإعطاء قيم لهذه المعاملات تسمى هذه المعاملات بالمعاملات الفعلية (Actual parameters)
Method body	وهو جسم الطريقة , ويجب أن يحتوي جسم الدالة على جملة (return) إذا كان نوع البيانات المرجعة من هذه الطريقة غير النوع (void) هذه الجملة ترجع قيمة من الدالة إلي مكان الاستدعاء .

مـــثـــــــال:

```
public int method1 ( int i , String s )
{
    //body
       String name = s;
       int j= i*i;
       return (j);
}
```

تتم عملية استدعاء الطرق عن طريق كتابة اسم الطريقة و إرسال قيم المعاملات إن وجدت. ويتم ذلك في المكان المراد تنفيذ عمل الطريقة فيه في داخل البرنامج. تستدعي الطرق بعضها البعض كما يمكن كذالك أن تستدعي الطريقة نفسها ذاتيا

الشكل العام لعملية استدعاء الطرق هو التالي :

```
Method_Name ([Parameters_list])
```

Method_Name	اسم الطريقة و عند استدعاء طريقه موجودة في صنف أخر لابد من كتابة اسم هذا الصنف قبل اسم هذا الصنف قبل اسم الطريقة بحيث تفصل بينهم بنقطة (.) مثلا : Math.pow (3.2) ;
Parameters_list	: قائمه المعاملات الفعلية (Actual parameters) وهي القيم الفعلية التي تستخدم في عملية استدعاء الطرق ويمكن أن

	تكون بالأشكال التالية: قيم ثابتة مثل Sum1(5 , 6) متغيرات مثل Sum1(x , y) استدعاء لطريقة أخرى Sum1(Sum2(2,4),y)

عند عملية الاستدعاء لابد من التركيز علي أن هناك نوعين من الطرق:

1) الطرق التي لا ترجع قيم و هي الطرق التي من النوع (void) وهذه الطرق لا يمكن إسنادها إلى متغير أو إستخدامها في تعبير .
2) الطرق التي تقوم بإرجاع قيم و في هذه الحالة يجب أن تستخدم في إحدى الحالات التالية :
   إسنادها إلى متغير.
   إستخدامها في تعبير.
   إستخدامها في عملية استدعاء لطريقة أخرى.

5- ماذا يحدث عند استدعاء الطريقة (method)

1) ننسخ المعاملات الفعلية (Actual Parameters) إلى المعاملات الشكلية (Formal parameter) أي تصبح المعاملات الفعلية قيماً إبتدائيه للمعاملات الشكليه وتعمل المعاملات الشكليه عمل المتغيرات المحلية في جسم الطريقة .
2) ينتقل تنفيذ البرنامج إلى بداية الطريقة المستدعاه .
3) عند إلانتها من تنفيذ الطريقة يستمر تنفيذ البرنامج من الجملة التالية لجملة الاستدعاء .

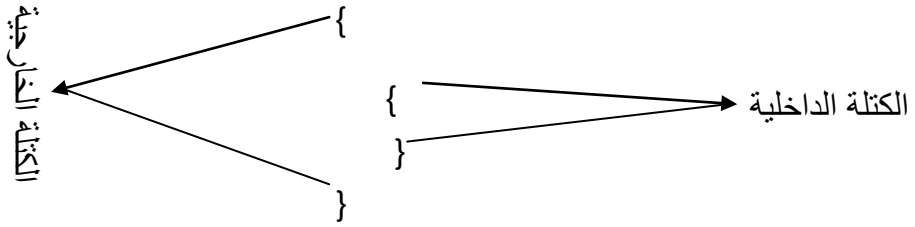
مثال توضيحي لاستدعاء الدوال

```
public class MathodCall {
    public static void main(String[ ] args) {
       int x=5, y=6, z=0, s=0;
       sum1(10,5);
       sum1(x,y);
       s = sum2(5,6);
       System.out.println("Sum="+ sum2(5,6));
       z=12+3 * sum2(x,10);
       sum1(sum2(3,4),5);

    }//end of amin
  static void sum1(int num1, int num2){
     int sum=0;
     sum = num1+num2;
     System.out.println("sum=" +sum);
    }//end of sum1
  static int sum2(int num1,int num2) {
     int sum=0;
    sum=num1+num2;
    return sum;
   }//end of sum2
}//end of class Mathod Call
```

6- الكتل والمجالات Blocks and Scops :

ضمن أي صنف أو دالة بالكود الموجود بين { } يدعى بالبلوك أو الكتل ويمكن إنشاء كتل متداخلة فالكتل الخارجية تغلف الكتلة الداخلية يمكن للمبرمج أن يستخدم نفس أسماء المتغيرات ضمن البرنامج الواحد أكثر من مره ولكن بشرط أن يقع كل من هذه المتغيرات في Scope (مجال الرؤيه) خاص به



مثال يوضح التصريح الصحيح:

```
public static void twoDeclaration()
{
   {//first block
      int var1=7;
      System.out.println(var1);
   }//end first block
   {
      //begin second block
      int var1=899;
      System.out.println(var1);
   }//end second block
}
```

مثال يوضح التصريح الخاطئ :

```
 public static void mathod wroning Declaration() {
   int value=30;
   System.out.println(value);
   int value=99;// خاطئة لأننا صرحنا عن متغير بنفس الإسم وفي نفس المجال
   {
      int value=55; // خاطئة لأن هذه الكتلة أصبحت داخل الكتلة الأولى فالكتلة الأولى غلفت هذه
الكتلة
    }
}// فكل المتحولات المعرفة في الكتلة الخارجية تصبح سارية المفعول بالنسبة للكتلة الداخلية
```

7- الاستدعاء الذاتي للطرق (Recursion)

المقصود بالستدعاء الذاتي هو أن تقوم الطريقة باستدعاء نفسها بنفسها فهناك الكثير من المسائل التي يمكن حلها باستخدام الاستدعاء الذاتي وبهذه الحالة يمكن توفير الكثير من جمل التكرار .

مثال : إيجاد المضروب (Factorial) لعدد معين

```
//Factorial.Java
import  javax.swing.JOptionPane;
public class Factorial {
   public static void main(String[ ] args) {
     String snum1;
     int num1,fact_of_num1;
     snum1=JOptionPane.showInputDialog("enter num1:");
     num1=Integer.parseInt(snum1);
     fact_of_num1=fact(num1);
     JOptionPane.showMessageDialog(null, num1+"!="+fact_of_num1);
    }
   static int fact(int n)
    {
      if(n==0||n==1)
        return  1;
      else
        return n*fact(n-1);
      }
}
```

عمليات الاستدعاء الذاتي للطريقة ( )Fact	عمليات التعويض العكسي
Fact(4) = 4 * Fact(3)	Fact(1) = 1
Fact(3) = 3 * Fact(2)	Fact(2) = 2 * 1 = 2
Fact(2) = 2 * Fact(1)	Fact(3) = 3 * 2 * 1 = 6
Fact(1) = 1	Fact(4) = 4 * 3 * 2 * 1 = 24

Fact(4)= 24

8- دالة الإدخال من لوحة المفاتيح ( )read :

الطريقة التي تسمح بالإدخال هي ()read باعتبار أن جهاز الإدخال هو لوحة المفاتيح والكلاس الذي يحتوي على هذه الطريقة هو (System) والغرض هو (in) شكل هذه الدالة هو:

System.in.read()

الطريقة ()read تستقبل كل ضربات لوحة المفاتيح وتعتبرها إشارات دخل إلى للبرنامج حيث تقبل دخلا من نوع (byte) وتعيده بنوع (int) إلى البرنامج .

مـثـــال :

```
public class DenmoInput
{
   public static void main(String[ ] args) throws Exception
   {
      char userIn;
      System.out.println("please enter a charater:");
      userIn=(char)System.in.read();
      System.out.println("you entered: "+userIn);
    }
 }
```

Throws Exeption : هي عبارة تعبر عن ضرورة مراعاة الأخطاء المحتمل حدوثها أثناء عملية الإدخال هذه الأخطاء مثل أن تكون لوحة المفاتيح معطلة أو أن يقوم المستخدم بإدخال قيم تختلف في نوعها عن نوع البيانات المعلن عنها في البرنامج

محاضره (7)

الفصائل classes في Java

تعتبر الاصناف classes و الكائنات objects مفهومين أساسيين في برمجه الكائنات, و تكمن الفائده في برمجه الكائنات في أن معظم برامج الحاسوب المستخدمه لحل المشاكل الحقيقيه تكون كبيره جدا و ثبت علميا أن أفضل طريقه لكتابه الرنامج هي تقسيمه إلي وحدات صغيره modules.
وحدات البرنامج في جافا هي الأصناف. عندما يقوم شخص بكتابة برنامج يقوم بضم الأصناف الجديده مع تلك المتوفره في مكتبه الأصناف في جافا API حيث توفؤ هذه المكتبه العديد من الأصناف التي تقوم بالعمليات الجسابية و معالجة النصوص و عمليات الادخال و الإخراج و الكثير من العمليات الأخري.
تتم عملية التخاطب بين هذه الأهداف عن طريق الرسائل.
الكائن object هو عباره عن شيء حقيقي في واقع الحياه العمليه مثل الطالب سعد مطيع فالطالب الجامعي سعد مطيع مثلا يعتبر كائن في البيئه الجامعيه.

1. تعريف الفصيلة defining class

الصنف class عباره عن قالب (مخطط) يحتوي (يمثل) الصفات للكائنات التي تنتمي لهذا الصنف, و يجب أن يحتوي هذا المخطط علي جميع صفات الكائنات التي تنتمي إليه و جميع الصفات الخاصه بإنشاء هذه الكائنات (النسخ) فمثلا الصنف طالب جامعي (UnivStudents) يمثل جميع الصفات لجميع الطلاب في البيئه الجامعية. و هذه الصفات أو البيانات Data يتم تمثيلها في الأصناف بالمتغيرات Data Members بينما العمليات Operations يتم تمثيلها بإستخدام الطرق Methods.

يتم انشاء فصيلة وتعريفها من خلال الكلمة ***class*** و يمكن أن يسبق هذه الكلمه كلمه public و التي معني عام. و هذا يعني أن كل صنف يمكن أن يقوم بإنشاء نسخ من هذا الصنف أما إذا لم نضع كلمة public في عملية التعريف فإن الأصناف داخل الحزمه package التي يوجد بها هذا الصنف هي وحدها تستطيع إنشاء نسخ instance من هذا الصنف, ثم يتبعها اسم الفصيلة ومن ثم جسم هذه الفصيلة المحصور بين القوسين {} , جسم هذه الفصيله يحتوي علي أعضاء بيانات و أعضاء دوال, اذا سبق تعريف الاعضاء البيانات الكلمه Private و التي تعني خاص فإن هذه البيانات يمكن التعامل معها فقط داخل هذا الصنف. أما لم نكتب شيء فإن الأصناف داخل الحزمه package التي يوجد بها هذا الصنف هي وحدها تستطيع التعامل مع هذه المتغيرات بعد إنشاء نسخه من هذا الصنف.

```
public class className
{
   Data   member
   Function  member
}
```

لغة جافا هي مجموعة من الحزم وكل حزمة تحتوي على عدد من الفصائل التي تخص موضوع محدد

مثل: (.... ,swing ,Applet ,network)

```
class  firstClass
 {
  int  x;
  final int y=  4;
  void  function ( )
   {
     System.out.println("This is my first   program ");
   }
 }
```

2. إنشاء الدوال Creating methods

الدوال هي التي تحدد سلوك الهدف object, لانشاء دالة ما لابد من القيام بالتالي :

1. تعريف نوع البيان الذي سترجعة الدوال Return type
2. تحديد اسم الدالة (كتابة اسم الدالة) Method name
3. كتابة المعاملات Parameters
4. كيان الدالة Method body

```
returnType   methodName (Type arg1, Type arge2,…)
{
  Body  of the method
 }
```

يمكن للداله ان ترجع مصفوفة ويكون شكلها كالتالي:

```
returnType [ ] methodName (par1[, …])
{
  Method body
}
```

Or

```
returnType  methodName (par1[, …]) [ ]
{
  Method body
}
```

مثال:

```
public class methodsClass
 {
   int makeArray (int lower, upper )[]
    {
      int arr [ ] =new int  [(upper -  lower )+1];
      for ( int  i = 0;  i< arr.length  ; i++)
       {
         arr[i] =lower ++;
      }
      return arr;
    }

     public static  void   main (string arge [ ])
      {
        int   myArray [ ];
        methodsClass M=new  methodsClass
        myArray =M. makeArray (1,10);
```

```
        System.out.print("the array is :[  ");
        for (int i=0; i<myArray.length; i++)
         {
          System.out.print (myArray [i]+ "  ");
          }
        System.out.println("]");
         }
}
```

3. دالة الفصيلة Class method

دالة الفصيلة هي دالة للفصيلة كلها فجميع الاهداف مهما تعددت تستطيع الوصول إلى هذه الدالة.

لبناء دالة من هذا النوع class method نكتب الكلمه static في بداية جمله التعريف لهذه الداله :

```
static returnType  methodName (parameter 1[,parameter  2,…])
{
   // Body  of the method
}
```

ميزة هذا النوع من الدوال اننا نستطيع استدعائها بدون اقترانها بأي هدف ولكي نستدعي هذة الدالة نكتب إسم الفصيلة ثم إسم الدالة

For Example:

```
class  squreNumber
{
    static  float   x=0;
    static  int  sqx (int y);
     {
         x=y*y;
         return   x;
       }
      public static  void main (string  args [  ])
```

```
    {
      squreNumber SQ=new   squreNumber ();
       System.out .println("the squre  of 100 ="+ squreNumber.sqx(100));
       System.out .println("the squer  of 100 ="+ SQ.sqx(100));
     }
}
```

أخر جملتين في البرنامج يقومان بنفس العمل و هو طباعه الجذر التربيعي للعدد 100 {

4. الفصائل المساعدة Helper classes

في البرامج الكبيرة ينقسم البرنامج الى فصائل متعددة وكل فصيلة لها عمل تؤديه, واحده من هذه الفصائل هي الفصيلة الرئسية و التي تحتوي على الدالة الرئسية ()main . تسمى الفصائل التي لاتحتوي على الدالة الريئسية و الموجودة ضمن التطبيق بالفصائل المساعدة. في كل التطبيقات ينفذ المترجم الدله ()main لذلك لكي يتم التعامل مع الفصائل المختلفه و الموجوده ضمن التطبيق يجب انشاء أهداف من هذه الفصائل داخل الداله الرئيسيه.

5. التحميل الزائد للدوال Overloading

في c++ و Java يمكن ان يوجد عدد من الدول تحمل نفس الاسم, **فالتحميل الزائد** : هو إنشاء دوال Methods تحمل نفس الاسم ولكن تختلف في عدد وانواع المعاملات arguments. عند التعامل مع مكتبة الفصائل الموجودة في لغة جافا نجد أن هناك فصائل بها الكثير من الدوال ذات الاسم الواحد ولكن كل واحدة من هذه الدوال تحتوي على انواع واعداد مختلفة من المعاملات فيتم التميز بين هذه الدوال من خلال بصمة الفصيلة class signature .

هذه البصمه تتكون من:

- عدد المعاملات التي تأخذها الداله
- أنواع هذه المعاملات

6. برنامج يوضح التحميل الزائد للدوال

```
import java .awt .point;
class   drawRect
{
```

```
int  x1=0, int y1=0, int  x2=0, int  y2=0;
drawRect buildRect (int  x1, int y1, int  x2,int y2)
   {
       this .x1=  x1;
       this .x2= x2;
       this . y1= y1;
       this . y2= y2;
       return    this;
    }
drawRect  buildRect (point  p1, point p2)
  {
      x1 = p1. x;
      y1 = p1 .y;
      x2 = p2.  x ;
      y2 = p2. y ;
      return this;
     }
 drawRect  buildRect (point  p1, int w ,int   h)
    {
     x1 =p1 .x;
     y1= p1.y;
     x2= x1+w;
     y2= y1+h;
     return  this;
    }
void  printRect ()
  {
```

```
        Systrm.out .print ("my Rect  :<"+x1+"," +y1);
        System.out .print("," + x1+"," +y2">");
       }
    public static void main (string args[])
      {
       drawRect  d= new drawRect ();
        System.out.println("calling  buildRect with coordinates 30,30,80,80");
        d. buildRect (30,30,80,80);
        d. printRect ();
        system .out . println("calling  buildRect with two points (5,5),(30,30));
        d . buildRect (new point (5,5),new point(30,30));
        d . printRect ();
        system .out . println("calling  buildRect with point (10,10) and h,w);
        d . buildRect ( new point (10,10),50,50);
        d . printRect ();
       }
} //  end of class
```

7. برنامج اخر يوضح التحميل الزائد للدوال Overloading

```
public class  demoOverload
 {
   Public   static   void main  (string   [  ] args )
    {
      int   month  =3,  day =24,  year =2010;
      overloadDate (month);
      overloadDate (month,day);
      overloadDate (month, day ,year);
```

```
    }
    public  static  void overloadDate (int   mm)
    {
      System .out .println("event  date: "+mm+"/1/2001")
    }
    public   static void overloadDate (int  mm, int   dd)
    {
      System .out .printlnj(" event date: "+mm+"/"+dd +"/2001")
     }
    public static  void   overloadDate (int   mm, int   dd , int yy)
    {
      System .out .println("event date: " +mm +"/"+dd+"/"+ yy );
     }
}  // end of the class
```

محاضره (8)

قواعد التوصل للفصائل Access Modifiers

1. المعدلات Modifiers

كلمة معدل modifier هي كلمة ترمز إلي عدة كلمات محجوزه Reserved word تستخدم في في تحديد التوصل Access إلي الفصيلة و دوالها و متغيراتها و هذه الكلمات هي:

static, public, private, protected, abstract,…

يستطيع المبرمج أن يحدد أي من أجزاء الفصائل و الدوال و المتغيرات يستطيع المستخدم التوصل لها و التعامل معها و أيها لا يستطيع.

نستطيع تقسيم المعدلات modifiers حسب الفئات التي تعمل معها و هي كالأتي:

- معدلات للسيطره علي الوصول للفصائل والدوال و المتغيرات و هي public, private protected
- معدلات لإنشاء المتغيرات و الدوال و هي static
- معدلات لإنشاء فصائل و دوال مجردة و هي abstract

و لكي نستخدم المعدلات modifiers فإننا نقوم بكتابة إسم المعدل ثم إسم المتغير أو الداله أو الفصيلة كما في الأمثلة التالية:

1.

```
public class Emp
{
Body of the class Emp
}
```

2.

```
public static void main (string args [])
```

3.

```
private  employeeNumber;
```

4.

static final double Days=7;

بالرغم من أن استخدام المعدلات إختياري و ليس إجباري إلا أنه من المفيد جدا إستخدامها لأننا نستطيع عن طريقها تحديد طرق وصول مستخدمي الفصائل إلي الفصائل و الدوال الأعضاء و المتغيرات الأعضاء في هذه الفصائل.

1. السيطرة علي التوصل للدوال و المتغيرات Access control for methods and variables

المعدلات التي تسيطر علي الوصول للدوال و المتغيرات هي private, public, protected و هذه المعدلات تحدد أي من المتغيرات و الدوال متاحة و مرئية للفصائل الأخري.

بإستخدام طرق التوصل نستطيع السيطرة علي كيفية إستخدام فصيله معينة من خلال الفصائل الأخري.

أحيانا نحتاج إلي بعض المتغيرات و الدوال التي يجب أن تستخدم فقط من داخل الفصيلة نفسها و يجب إخفائها عن باقي الفصائل التي تتفاعل مع هذه الفصيلة و يطلق علي هذه العملية الكبسله Encapsulation .

الكبسله Encapsulation

الكبسله هي عملية تعبئة البيانات و الدوال في كيان واحد مع وضع قواعد لمنع أو السماح بقراءة أو تعديلات المتغيرات الموجودة في الفصيلة من فصيلة أخري. تقوم لغة جافا بتوفير 4 مستويات من السيطرة علي التوصل و هي:

1. المعدل العام public
2. المعدل الخاص private
3. معدل الوراثة protected
4. المعدل الإفتراضي default

المستوي الرابع معناه أننا لا نستخدم أي كلمة محجوزة modifier مع المتغير أو الدالة. فالمتغير أو الدالة التي يعلن عنها بدون معدل يصبح متاح لأي فصيلة في نفس الحزمة. أي أن كل الفصائل الموجوده في هذه الحزمة تستطيع أن تري و تعدل المتغير المعلن عنه بدون معدل و كذالك الدوال المعرفه بدون معدل تستطيع أي فصيلة موجوده في نفس الحزمة أن تستدعيها, فلغة جافا عبارة عن مكتبة من الفصائل

المرتبة في حزم و كل من هذه الحزم تخص موضوعا محددا مثل الشبكات, الواجهات الرسومية و غيرها.

فلا تستطيع أي فصيلة خارج الحزمة التوصل إلي المتغيرات أو الدوال المعلن عنها بدون معدل و هذا المستوي من السيطره علي التوصل لا يمكننا من السيطرة علي المتغيرات و الدوال بشكل كبير, و لكن يوجد اثنان من المعدلات يوفران طريقة أكثر تحديدا لمستويات التوصل و هما private, public

Private Access

عند إستخدام private مع أي متغير عضو أو دالة عضو فإننا نخفيه بالكامل من أن يُستخدم من أي فصيلة. فالفصيلة الوحيدة التي تستطيع رؤيته أو التعامل معه هي الفصيلة المعلن فيها هذا المتغير أو الدالة, و هذا ينطبق علي الفصائل الموروثة من هذه الفصيلة فهذه الفصائل لا تستطيع التوصل لهذا المتغير أو الدالة المعلن عنها بـ private.

شكل الإعلان عن المتغيرات و الدوال المحمية كالتالي:

```
private int a;
private void modify ()
{
// method body
}
```

تفيدنا المتغيرات المعلن عنها بـ private في حالتين:

1. عندما تكون الفصائل الأخري لا تحتاج إلي إستخدام هذا المتغير.
2. عند الخوف من أن تقوم فصيلة أخري بتغيير قيم هذا المتغير بقيم غير موغوب فيها.

مثال:

```
class SuccStudents
{
    int x;
    private int successors;
    private float salary;
    void setSuccStud(int s)
    {
```

```
        successors = s; // هذه الداله تستطيع التعامل مع المتغير المحمي و المعرف في نفس
        الفصيله التي تنتمي إليها هذه الدالة
    }
    private  void print ()
    {
        System.out.println("the Number of successed students =  "+ successors);
    }
}

class mainClass
{
    public static void main (String args[])
    {
        SuccStudents stud=new SuccStudents();// إنشاء هدف من الفصيلة السابقة
        stud. successors = 90; // هذه الجمله خاطئة لأن الهدف يحاول الوصول إلي متغير محمي
        موجود في فصيلة أخري لذالك يصدر المترجم خطأً عند هذه الجملة
        stud.print(); // هذه الجملة كذالك خاطئة لأن الهدف يريد التعامل مع دالة محمية موجودة
        في فصيلة أخري
        stud. setSuccStud(10); // هذه الجملة صحيحه لأن الهدف يستطيع الوصول إلي الطريقه
        الغير محمية و الموجوده في فصيلة أخري
        stud. x=20;    // هذه الجملة كذالك صحيحه لأن المتغير العضو في الفصيلة السابقة غير
        محمي و بامكان الهدف التعامل معه
      }
}
```

Public Access

في بعض الأحيان نحتاج أن نجعل متغير أو دالة متاحة لجميع الفصائل التي تريد إستخدام هذا المتغير أو الدالة , قد يتبادر إلي الذهن بأن الحل هو بعدم كتابة معدل, لاكننا إذا عرفنا متغير أو دالة بدون كتابة معدل Modifier فسيكون المتغير أو الدالة متاحة لأي فصيلة موجودة في الحزمة

الموجوده فيها فصيلة هذا المتغير أو هذه الدالة. في هذه الحالة الفصائل الموجودة في الحزم الأخري لا تستطيع الوصول إلي هذه المتغيرات أو الدوال.

إذا أردنا جعل متغير أو دالة ما متاحة لجميع الفصائل التي تريد إستخدام هذا المتغير أو هذه الدالة علينا أن نستخدم المعدل public فعند الإعلان عن متغير أو دالة public يصبح لجميع الفصائل في أي حزمة الحق في التوصل لهذا المتغير أو لهذه الدالة.

الإعلان عن متغير عضو أو دالة عضو في فصيلة ما بإستخدام المعدل public يأخذ الشكل التالي:

```
public  int a;
public  void modifyA ()
{
// method body
}
```

3. معدل مستوي الوراثة Protected Access

يقوم المعدل من نوع protected بالعمل علي الحد من التوصل للمتغيرات و الدوال إلا عن طريق المجموعتين التاليتين:

- الفصيلة الفرعية subclass من هذه الفصيلة
- كل الفصائل الموجودة في نفس الحزمة

أي أن المعدل protected مثل المعدل default بإستثناء أن الفصائل الموروثة سواء الموجودة في نفس الحزمة أو في حزم أخري تستطيع ان تتوصل إلي المتغيرات أو الدوال المسبوقة بـ protected و نعلن عن المتغيرات و الدوال النوع protected بالشكل التالي:

```
protected  int a;
protected  void modifyA ()
{
// method body
}
```

و يعتبر هذا النوع من مستويات التوصل مفيدا جدا في الوراثة.

مقارنة المستويات Comparing levels of access control

منعا للإرتباك سنقوم بجمع مستويات التوصل المختلفة في جدول واحد, الجدول التالي يوضح مجال رؤية عناصر البيانات و عناصر الدوال الموجودة في فصيلة ما لكل من محددات (معدلات) الوصول.

مكان التوصل Visibility	مستويات معدلات التوصل			
	public	protected	default	private
From the same class	Yes	Yes	Yes	Yes
From any class in the same package	Yes	Yes	Yes	No
From any class outside the package	Yes	No	No	No
From subclass in the same package	Yes	Yes	Yes	No
From subclass outside the package	Yes	Yes	No	No

4. الوراثة و مستويات التوصل Access control and Inheritance

عندما ننشيء فصيلة فرعية subclass و نقوم بعملية override لدالة موجودة في الفصيلة العليا superclass لابد أن نأخذ في الاعتبار موضوع مستويات التوصل access contrl في الدالة الأصلية و كقاعدة عامة:

- الدوال المُعلن عنها public في الفصيلة العليا لابد أن تكون public في كل الفصائل الفرعية.
- الدوال المُعلن عنها protected في الفصيلة العليا لابد أن تكون أيضا إما public أو protected في كل الفصائل الفرعية و لا يمكن أن تكون في أي حال من الأحوال private.
- الدوال المعلن عنها بدون معدل التوصل Access modifier في الفصيلة العليا يمكن أن نعلن عنها private في الفصائل الفرعية.

دوال التوصل Accessor Methods

لكي نفهم ماذا نعني بدوال التوصل لنفرض أننا أعلننا عن متغير عضو في الفصيلة 1 (class1) و لكن هذا المتغير يمثل بيانات نخاف أن تتوصل إليه الفصائل الأخري و تغيرمن قيمته, و ماذا لو أردنا أن نجعل بعض الفصائل تتوصل إلي هذا المتغير و بعضها لا تستطيع التوصل. اذا ما الحل ؟

اذا قلنا بأننا سنستخدم private فإن كل الفصائل الاخري لن تستطيع التوصل لهذا المتغير في هذه الحاله لن تحل المشكلة. و الحل هو في إستخدام دوال التوصل للمتغيرات المُعلن عنها private هذا يعني بأننا سنقوم بإنشاء دوال للتوصل إلي هذا المتغير و هذه الدوال ستكون public كما في المثال التالي:

```
public  class  Employee
 {
   private int   empNum;
   private double empSalary;
   private String managerName;
   Employee () //constructor without arguments
   {
    empNum=999;
    empSalary=250,5;
    mangerName="Ahmed";
   }
  Employee (int  Number)//constructor with arguments
   {
    empNum = Number;
    empSalary =250,5;
    managerName = "Ahmed";
   }
  public int getempNum() // Access Method (get access method)
  {
    return  empNum ;
   }
   public getempSalary () // Access Method (get access method)
   {
```

```
    return  empSalary;
  }
   public  String getmanageName() // Access Method (get access method)

   {
    return  managerName ;
    }
    public void  setempNum (int num) // Access Method (set access method)
    {
      empNum=num;
     }
    public  void set ampsalry (double  salary ) // (set access method)
    {
      empsalary  =salary ;
     }
    public void setmanagerName (String name) //  (set access method)
    {
      managerName =name;
     }
}// end class Employee
public class demoConstructor extends Employee
{
  public  static  void  main (string [  ] args )
   {
    demoConstructor Emp1 = new demoConstructor();
    demoConstructor Emp2 = new demoConstructor();
    System.out.println ("Employee number for Emp1 is:"+Emp1.getempNum ());
```

```
  Emp2.setempSalary (300,50)
  System .out .println("Employee salary for Emp2 is:"+ Emp2.getempSalary())
 }
}// end class Employee
```

تنقسم دوال التوصل إلي نوعان واحدة للقراءة و يبدأ إسمها بالكلمة get مثل public int getempNum() وواحدة للكتابة و يبدأ إسمها بالكلمة set مثل public void setempNum (int num).

Final classes, Methods and variables

ان final modifier يستخدم مع الفصائل و الدوال و المتغيرات ليوضح أنهم لن يتغيرو و لكن معناه يتغير مع كل منهم كالأتي :

- Final class تعني أننا لن نستطيع أن ننشيء من هذه الفصيلة فصائل فرعية أي لا نستطيع القيام بالتوريث منها
- Final method تعني أننا لن نستطيع أن نقوم بعملية override لهذه الدالة في أي فصيلة فرعية
- Final variable تعني أننا لن نستطيع تغيير قيمة هذا المتغير و هذا النوع من المتغيرات يسمي بالثوابت constants

و نستفيد من استخدام final للدوال في جعل عملية التنفيذ أسرع لأننا نجعل المترجم لا يبحث عن عمليات تعديل قد تكون تمت لهذه الدالة, كذالك الإعلان عن فصيله بهذا الشكل يجعل تنفيذ الفصيلة أكثر سرعة.

محاضره (9)

المشيداتConstructors

1. تعريف المشيدات

المشيدات: هي عبارة عن دوال لها نفس اسم الصنف (class) وتستخدم عند انشاء هدف (object) من هذا الصنف. لإنشاء هدف نستطيع عمل هذا بإ حدى طريقتين:

- الطريقه الاولي

```
className  objectName;
objectNam = new className ();
```

- الطريقه الثانيه

يمكن دمج الجملتين السابقتين بجملة واحدة

```
className objectName =new className ();
```

توضيح

وظيفة الجملة className objectName :هو إنشاء غرض باسم objectName ليكون عضواً من أعضاء الصنف className وهذا يعني بأنة سيرث كل ماهو داخل قوسي الصنف

وظيفة الكلمة New : هو حجز حيز في الذاكرة للغرض او الهدف objectName

الداله class Name() : هي مشيدة constructor او دالة بانية .

2. إرسال الوسائط (arguments) إلى المشيدات (constructors).

يمكن إرسال وسائط arguments الى المشيدات الخاصة بأصناف محدد ة.

لنفرض أن لدينا البرنامج التالي .

```
public  class Employee
{
 private  int   empNum;
 private  int   empSalry;
 Employee  ()
```

```
 {
   empNum =999;
  }
 // other   methods   go    here
}
```

نلاحظ في هذا البرنامج بأن مشيد الصنف ستلحق القيمة 999 للحقل empNum من اجل كل غرض يتم انشاؤه من نفس نوع هذا الصنف, مثلا اذا أنشأنا هدفا جديدا و اسميناه partTimeWorker فإن قيمة الحقل empNum ستكون 999 لهذا الهدف:

```
Employee partTimeWorker = newEmployee( );
```

في كثير من الاحيان قد يتطلب العمل إدخال قيمة من قبل المستخدم عند انشاء الهدف في هذه الحالة نحتاج لوجود مشيد موجود ضمن الكلاس يقوم بمهمة تمرير هذه القيمة الى المتغير empNum مثلاً:

```
Employee (int num)
  {
    empNum= num ;
}
```

ولادخال قيمة ما للمتغير empNum عند انشاء الهدف السابق مثلاً نقوم بكتابة العبارة التالية :

```
Employee  partTimeWorker = new  Employee (891);
```

يمكن للصنف أن يحتوي علي أكثر من داله بانيه (مشيد) و هذا يسمي التحميل الزائد للبانيات. Overloaded constructors و إذا لم نقم بتعريف داله بانيه داخل الصنف فإنه يتم إنشاء الداله البانيه تلقائيا Default constructor

3. برنامج يوضح العمل مع المشيدات

```
public  class  Employee
 {
   private int   empNum;
   private double empSalary;
   private String managerName;
   Employee ()
```

```
{
  empNum=999;
  empSalary=250,5;
  mangerName="Ahmed";
}
Employee (int  Number)
{
 empNum = Number;
 empSalary =250,5;
 managerName = "Ahmed";
}
public int getempNum()
{
 return  empNum ;
}
public getempSalary ()
{
  return  empSalary;
}
public  String getmanageName()
{
  return  managerName ;
 }
 public void  setempNum (int num)
 {
    empNum=num;
   }
```

```
  public  void set ampsalry (double  salary )
  {
    empsalary  =salary ;
   }
  public void setmanagerName (String name)
  {
    managerName =name;
   }
}// end class Employee
public class demoConstructor
{
 public  static  void  main (string [  ] args )
  {
   Employee Emp1 = new Employee ();
   Employee Emp2 = new Employee (789);
   System.out.println ("Employee number for Emp1 is:"+Emp1.getempNum ());
   Emp2.setempSalary (300,50)
   System .out .println("Employee salary for Emp2 is:"+ Emp2.getempSalary())
  }
 }// end class Employee
```

و ليتمكن المبرمج من أعادة إستخدام الأصناف التي كتبها سابقا دون الحاجه إلي إعادة كتابتها من جديد, لابد من وضع هذه الاصناف في حزمه (Package) و **الحزمه Package :** هي عبارة عن حاوية (container) تحتوي علي مجموعة من الاصناف المترابطة مع بعضها البعض ترابطا منطقيا و من فوائد استخدام الحزم أيضا امكانية إستخدام نفس الإسم لأكثر من صنف حيث أنه يمكن أن يكون لدينا الكثير من الأصناف التي تحمل نفس الإسم فيمكن أن نضع هذه الأصناف في حزم مختلفة و بالتالي يمكن إستخدام أكثر من صنف يحمل نفس الإسم في مكان واحد. و تتم عملية إنشاء الأصناف داخل حزمه بوضع الكلمه المحجوزة Package في بداية الملف الذي يحتوي علي تعريف الصنف أو الأصناف

متبوعة بإسم الحزمة, و بعد عملية الترجمة الناجحة للبرنامج يتم تخزين الأصناف و تحديدا الملفات ذات الإمتداد class في هذه الحزمة. و إذا لم تكن هذه الحزمه موجودة يتم انشاءها بعد انتهاء عمليه الترجمة.

مثلا يمكن أنشاء الحزمة javaCh.constructors.Ex التي ستحتوي علي جميع الأصناف التي تم تعريفها في المثال السابق بكتابة الجملة التالية: package javaCh.constructors.Ex في بداية الملف demoConstructor . بحيث سيتم تخزين الأصناف: Employee و demoConstructor في المجلد Ex الموجود داخل المجلد constructors و الموجود داخل المجلد javaCh. في المثال السابق نلاحظ وجود أكثر من صنف في نفس الملف و هي : Employee و demoConstructor بعد الترجمة سيتحول كل صنف من الأصناف الموجوده في الملف demoConstructor إلي ملف منفصل ذو إمتداد class . و يجب أن لا يحتوي الملف الواحد علي أكثر من صنف معرف كصنف عام public.

اذا سيتم انشاء الملفات التالية: Employee.class و demoConstructor.class في الحزمة: javaCh.constructors.Ex

محاضره (10)

الأهداف Objects

بعد أن تعرفنا على مفاهيم البرمجة بالأهداف وتعرفنا على أنواع البيانات والدوال وغيرها من المواضيع. حان الوقت للتعرف على الأهداف (Object) بصوره موسعه.

1-إنشاء الأهداف Creating new Object

البرنامج في لغة الجافا يتكون من مجموعه من الفصائل(calsses) وهذه الفصائل هي القوالب (Templates) التي تنشئ منها الاهداف (Object) وهذه الأهداف هي التي تقوم بالتعامل مع الدوال الأعضاء (Member Methods) والمتغيرات الأعضاء (Member Variables) الموجودة في الفصيلة.

لكي ننشئ هدف من فصيلة ما نتبع القاعدة العامه التالية:

```
className ObjectName=new className();
```

ومعناها كتابة اسم الفصيلة المطلوب تعريف هدف منها ثم اسم الهدف الجديد ثم كتابة المؤشر = ثم ألكلمه المحجوزة(new) ثم كتابة اسم الفصيلة يتبعها القوسين()

مثال:

```
Random x=new Random();
Integer I=new Integer (7);
```

عملية أنشاء هدف تتكون من جزئيين :الجزء الأيمن مسئول عن توليد الهدف في الذاكرة والجزء الأيسر يسمى إسم الهدف أو الموشر (Reference) لهذا الهدف.

Integer(7) & Random() :استدعاء لدوال البناء(constructor) الخاصة بالفصائل Integer&Random

برنامج توضيحي لإنشاء أهداف:

```
import java.util.Random
class RandomNumbers
```

```
{
    Public static void main(string[]args)
    {
        Random r1,r2;//
        r1= new Random();//
        system.out.println("Random value 1:"+r1.next Double());
        r2= new Random(8756);
        system.out.println("Random value 2:"+r2.next Double());
    }
}
```

الداله Random()	هي دالة بناء بدون معاملات وناتج هذه الداله هو توليد أعداد عشوائية محصورة بين 0.0و1.0 وهذه القيم العشوائية ستجدها متغيره في كل مره يتم فيها تنفيذ البرنامج
الداله Random(8756)	هي دالة بناء ولكنها تأخذ معاملاً (قيمه) هذه القيمة تسمى بذره (seed) وهذه خاصية من خواص الفصيلة (Random) أي أنك إذا أعطيتها بذره فإن القيمة العشوائية الناتجة عن هذه الدالة لا تتغير
الداله nextDouble()	هذه الداله تقوم بإرجاع الرقم التالي من سلسة الأرقام العشوائية المحصورة بين 0.0 .....1.0

2- دور الكلمة المحجوزة new

قلنا أن القاعدة العامة لإنشاء هدف هي القاعدة التالية:

```
className ObjectName=new ckassName();
Random r1=new Random ();
```

فما دور الكلمة المحجوزة (new)؟

عند إستخدام الكلمه المحجوزة (new) فإن هناك عدة خطوات يقوم بها المترجم (Compiler) وهي:

1- يتم إنشاء هدف (instance) من الفصيلة المعلن عنها.

2- يتم حجز جزء من الذاكرة (Member Allocation) لهذا الهدف.

3- يتم استدعاء دالة البناء (Constructor) الخاصة بهذه الفصيلة

3- المؤشر (this)

للمفتاح (this) دور مهم مع دوال البناء (Constructor) وعموماً فإن (this) تشير إلى الهدف الحالي وتستخدم هذه الكلمة عند استدعاء دوال البناء وتستخدم أيضاً لحل مشكلة أسماء المتغيرات والأعضاء المتشابهة.

مثال:

```
class Employee
{
 String name;
 int age ;
 double salary;
 Employee(String name)
 {
   this.name=name; // هنا نقوم بتمرير القيمه المرسله عند إستدعاء المشيد إلي المشيد ليعطيها
لمتغير الفصيلة و لأن الأسماء متشابهة نستخدم المؤشر ليشير إلي هدف الفصيلة الذي تنشأه الداله البانية
    System.out.println("Employee name is:"+name);
   }// end of the first constructor Employee( )
  Employee(String name , double sal)
  {
    this (name); // هنا إستدعاء لدالة البناء السابقة الموجودة في هذه الفصيلة و التي تأخذ معاملا
     واحدا
    salary =sal;
    system.out.println("Employee Salary is :"+salary);
   }//end of the second constructor  Employee ( , )
  Public static void main(string[]args){
    Employee emp1 =new Employee("mohammed");
    Employee emp1 =new Employee("ahmed",356.6);
   }//end of the main
}//end of the class
```

4- طرق التوصل للمتغيرات إعطائها قيم (Variable accessing)

لكي نقوم بإنشاء متغير ألصيغه ألعامه هي التالية:

Type variableName;

مثلاَ int x;

هنا نقول أن (x) متغير من نوع (int) وهذا صحيح ولكننا إذا أعلنا عن المتغير داخل فصيلة فإن x يسمى متغير عضو في الفصيلة (instance variable)

```
class myclass
{
int x ; // x is an instance variable
}
```

المتغير العضو (instance variable) :هو متغير عادي معلن عنه في داخل الفصيلة

متغير الفصيلة (class varibable) :هو متغير عادي أيضا معلن عنه في داخل الفصيلة ولكنه تسبقه ألكلمه المحجوزة (static) عند الإعلان عنه.

```
class myclass
{
int x; // instance variable
static int t ;  // class variable
}
```

ما الفرق بين المتغير العضو في الفصيلة ومتغير الفصيلة؟

الفرق في كيفية تعامل المترجم (Compiler) مع الأهداف في حجز الذاكرة.

نفرض أننا نستخدم الفصيلة السابقة (myclass) لإنشاء أهداف :

```
mycalss c1,c2;
c1=new myclass();
c2=new myclass();
```

عند إنشاء هذان الهدفان يقوم المترجم بعمل نسخه من كافة المتغيرات الأعضاء (instance variable)

لكل هدف بحيث أن كل هدف يكون له النسخه الخاصة به من كل المتغيرات الأعضاء(instance variable)

ويقوم بحجز موقع واحد في الذاكرة لمتغير الفصيلة (class variable) لتتعامل كل الأهداف المنشأه من هذه الفصيلة مع هذا المتغير وليس مع نسخا منه لكل هدف كما في المتغيرات الأعضاء instance) variable).

الشكل التالي يوضح حجز أماكن في الذاكرة للمتغيرات المعلن عنها في الفصيلة myclass

10	20	50
C1.x	C2.x	y

فكلا الهدفان c1,c2 يمتلك قيمة مختلفة x

C1.x=10

C2.x=20

لا كنهما يمتلكان قيمة واحدة y

C1.y=50

C2.y=50

بمعنى ان الهدفان c1,c2 يشيران الى موقع واحد في الذاكرة بالنسبة للمتغير y

القاعدة العامة للتوصل إلى المتغيرات :

Objectname .variablename=value;

C1.y=60;

C1.x=70;

5- المراجع للأهداف References to objects

المرجع هو نوع من المؤشرات يستخدم لمعرفة قيم هدف , والمؤشرات (Pointers) هي نوع من انواع المتغيرات التي تستخدم بكثرة في لغة C\C++ وتستخدم هذه المتغيرات للتعامل مع الذاكرة (memory) بشكل مباشر.

عن طريق المتغيرات من نوع (Pointers) نستطيع ان نعرف عنوان أي متغير في الذاكرة كما لو أننا نشير إليه من هنا جاء أسم المؤشرات (Pointers), ولكن إستخدام المؤشرات (Pointers) مثلاً في لغة C++ بالرغم من انها نقطة قوة لهذة اللغة إلا أنة في نفس الوقت ينتج عند استخدامها الكثير من الأخطاء في حالة عدم التعامل معها بحذر. في لغة جافا لاتوجد متغيرات من نوع (Pointers) ولكن بدلاً من ذلك تقوم لغة جافا بإعطاءنا متغيرات تقوم بالتعامل مع الذاكرة ولكن بشكل غير مباشر .

فمثلاً عندما نقوم بإنشاء هدف Object ونقوم بجعل هذا الهدف يتوصل لداله (method) مثل الجملة التالية :

```
Str.length()
```

فإننا في الحقيقة لانستخدم الهدف و لكن نستخدم مؤشر لهذا الهدف

مثال

```
import java.awt.Point;

class referenceUse {

public static void main (String[] args) {

   Point p1,p2;

   p1=new Point(150,250);

   p2=p1;

   p1.x=200;

   p1.y=300;

   System.out.println("\n point1: " +p1.x +" , " +p1.y);

   System.out.println(" \n point2: "+p2.x +" , " +p2.y);

   }//End main

}//End class
```

الفصيلة Point موجودة ضمن الحزمة الفرعية awt وهذة الحزمة موجودة ضمن الحزمة الأساسية java ولكي نستخدم اي فصيلة موجودة في حزمة لابد ان نخبر المترجم اين توجد هذه الفصيلة عن طريق الأمر import .

Point p1.p2; : الإعلان عن هدفان هما p1,p2 من الفصيلة Point .

P1=new Point(150,250); : إعطاء قيم أولية للهدف p1 عن طريق دالة البناء الموجودة في الفصيلة Point .

p2=p1; :قمنا بعمل مرجع Reference من p2 إلى p1 ومعنى ذلك أننا جعلنا الهدف p2 يشير إلى مكان الهدف p1 في الذاكرة لذلك سنجد ان ناتج السطرين :

```
System.out.println("\n point1: " +p1.x +" , " +p1.y);
System.out.println(" \n point2: "+p2.x +" , " +p2.y);
```

هو نفسة

```
point1:200,300
point2:200,300
```

وهذا ناتج عن عمل المرجع(Reference) p2=p1;

6- تحديد الفصلية لهدف ما Determing The class of an object

لتحديد الفصيلة التي ينتمي إليها هدف ما نستخدم الطريقة العامة التالية :

```
objName.getclass().getName();
```

مثال لنفرض ان لدينا هدف p من الفصيلة Point لكي نعرف ما الفصيلة التي ينتمي إليها هذا الهدف والحزمة التي توجد فيها هذه الفصيلة نقوم بالتالي :

1- الإعلان عن متغير من النوع String مثلا st
2- نقوم بتخصيص التعبير p.getClass().getName(); لهذا المتغير بالشكل التالي:

```
st=p.getClass().getName();
```

ثم نقوم بإخراج قيمة هذا المتغير:System.out.println(st); و يكون الناتج هو التالي:

```
java.awt.Point
```

محاضره (11)

الطرق الخاصة بالسلاسل "string"

السلسلة الرمزية (string) :هي عبارة عن مجموعه من الرموز(characters) المتجاورة , في معظم لغات البرمجة تكون السلسلة الرمزية (string) مصفوفة من الرموز (character). في Java السلسلة الرمزية هي كائن من الصنف المسمى (String) ويحتوي هذا الصنف على العديد من الطرق الخاصة بمعالجة هذا النوع من البيانات . هناك عملية واحده يمكن استعمالها على السلاسل الرمزية دون الحاجة إلى استدعاء أي طريقة من طرق الصنف String وهي عملية الدمج"+" حيث تقوم هذه العملية بدمج سلسلتين رمزيتين لتكوين سلسله رمزيه واحده

1- تعريف المتغيرات من النوع String

تتم عملية تعريف المتغيرات من نوع السلاسل الرمزية String باستخدام اسم الصنف String بدل نوع البيانات عند تعريف متغيرات من نوع بدائي (primitive data type)

الصيغة العامة : **type variableName ;**

مثال

```
String str ;
str = " hello ";
```

تم تعريف المتغير str من النوع String وهنا يعتبر str كائن من الصنف string , فالصنف string معرف ضمن المكتبة java.lang.String والتي يتم استدعائها تلقائيا ضمن أي برنامج يتم كتابته.

يتم إنشاء عنصر String باستخدام الكلمة المفتاحية new

الصيغة العامة :

```
String objectName = new String ("text");
```

مثال :

```
String  aGreeting = new String ("hello");
```

قمنا بتعريف عنصر بالاسم aGreeting من النوع String ووضعنا فيه قيمه ابتدائية هي hello

يمكننا إنشاء مصفوفة عناصرها من نوع String بالشكل التالي :

الصيغة العامة :

String [ ] ArrayName = {"element1"[, element2 , …]};

مثال :

String[ ] friends = {"Ahmed","Moamar","Saad"};

2- بعض الطرق الخاصة بالسلاسل الرمزية

لنفرض أن لدينا السلسلة s لنبين وظيفة الطرق المختلفة

اسم الطريقة	وظيفة الطريقة
s.length()	ترجع الطريقة length() طول السلسلة الرمزية s
عملية المقارنة بين سلسلتين رمزيتين	
s.comparTo(t)	تقوم الطريقة بمقارنة السلسلة الرمزية (s) مع السلسلة الرمزية (t) وتعيد 0 إذا كانت (s) تساوي (t) و تعيد رقم سالب إذا كانت (s) اقل من (t) وتعيد رقم موجب إذا كانت (s) أكبر من (t)
s.comparToIgnoreCase(t)	تعمل هذه الطريقة نفس عمل الطريقة s.comparTo(t) ولاكن مع إهمال حالة الحروف (كبيرة أم صغيرة)
s.equals(t)	تعيد true إذا كان (s) يساوي (t)
s.equalsIgnoreCase(t)	تعمل هذه الطريقة نفس عمل الطريقة s.equals(t)) ولاكن مع إهمال حالة الحروف (كبيرة أم صغيرة)
s.startsWith(t)	تعيد true إذا كانت (s) تبدأ بالسلسلة الرمزية (t)
s.endsWith(t)	تعيد true إذا كانت (s) تنتهي بالسلسلة الرمزية (t)
عمليات البحث	
s.indexOf(t)	ترجع موقع أول مكان توجد فيه (t) داخل السلسلة (s)

s.indexOf(t , i)	ترجع موقع أول مكان توجد فيه (t) داخل (s) بعد الموقع (i)
s.lastIndexOf(c)	ترجع موقع أخر مكان يوجد فيه الحرف المخزن في المتغير c داخل السلسلة الرمزية s
عمليات أخذ جزء من السلسلة s	
s.charAt(i)	ترجع الحرف الموجود في الموقع i داخل السلسلة s
s.substring(i)	ترجع جزء من السلسلة s بدءا من الموقع i وحتى النهايه
عمليات التعديل على السلسلة	
s.toLowerCase()	إنشاء سلسلة رمزية جديدة تحتوي كل ما في السلسلة s بعد تحويل كل الحروف إلى حروف صغيرة
s.toUpperCass()	إنشاء سلسلة رمزية جديدة تحتوي كل ما في السلسلة s بعد تحويل كل الحروف إلى حروف كبيرة
s.replace(c1 , c2)	إنشاء سلسلة رمزية جديدة من السلسلة s بعد تبديل كل من (c2 , c1) وهما من النوع char

3- مقارنة السلاسل Comparing String

تعتبر السلسلة غرضا من الأغراض التي يمكن إنشاؤها في لغة Java لذلك علينا إتباع طرق محدده لتنفيذ عمليات المقارنة علي مثل هذه العناصر. ويجب ان نفهم كذلك كيفية حجز حيز في الذاكرة من اجل هذا النوع من المتحولات

فإذا قمنا بالإعلان عن متغير وليكن y من النوع int و ألحقنا به القيمة (13) عندها سنخصص حجره في الذاكرة عنوانها عبارة عن المتحول y وفي هذه ألحجره تخزن القيمة (13) كما في الشكل التالي :

حيز في الذاكرة عنوانه y

int y = 13 ; => y ⟶ 13

في حال قمنا بإلحاق قيمه جديدة في موقع الذاكرة الذي يشير إليه المتحول y عندها ستحذف القيمة 13 وتستبدل بقيمه جديدة

int y = 6 ; => y ⟶ 6

إن هذه العملية تختلف عندما نقوم بالتعريف عن عناصر من نوع string ولنفرض أننا نريد التصريح عن متحول من نوع string بالشكل التالي:

String aGreeting = "hello" ; => aGreeting ⟶ Hello

الان سنقوم بالحاق قيمه جديد في نفس المتحول

String aGreeting = "Welcome" ; => aGreeting ⟶ Welcome

وبالتالي يمكن استنتاج ما يلي :

عند الحاق قيمة جديدة لمتحول من نوع string فان حجز الذاكرة القديم يبقى على حاله في الذاكرة ويضع القيمة الجديدة في موقع جديد في الذاكرة وينقل المتغير الذي يشير إلى موقع الذاكرة القديم إلى موقع الذاكرة الجديد. لذلك فان عمليات المقارنة التي من الممكن إجرائها بين عناصر ذات نوع أساسي تختلف من حيث تقنية العمل عن العناصر ذات النوع string, فعلى اعتبار أن كل قيمه جديدة من نوع string تقوم بحجز حيز ذاكره جديد مع المحافظة على القيمة الملحفة السابقة فان عملية مساواة مثلا(==) سيكون مفهومها كالتالي:

هل يشير المتغيران إلى نفس الحيز من الذاكرة ؟

مثال :

```
String a = "Welcome";
String b = "Welcome";
```

لنطبق عملية المقارنة التالية

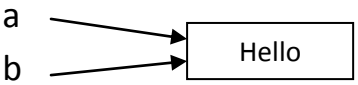
```
if ( a == b )
  System.out.println ( " true " );
else
  System.out.println ( " true " );
```

والنتيجة false وذلك لان المتحولين يشيران إلى حيزي ذاكره يحتويان قيمة متشابهة.

في حالة كانت التصريحات كالتالي:

```
String  a = "Hello" , b;
b=a;
```

في هذه الحالة ستكون النتيجة true ويمكن تشبيه ذلك المخطط



دالة المقارنة ()equals

زودتنا لغة الجافا بطريقه تساعدنا على مقارنة محتويات حيزي الذاكرة وليس مقارنة متحولين يشيران إلى حيزي ذاكره هذه الطريقة هي equals

مثال :

```
String Name = "Rashid";
String anotherName = "Rashid";
if (Name.equals (anotherName))
    System.out.println ("Name's the same");
if (anotherName.equals (Name))
    System.out.println ("Name's the same");
if (Name.equals ("Rashid"))
    System.out.println ("Name's the same");
```

إن نتائج المقارنة في البرنامج كلها صائبة (true) لذللك سيتم طباعة جمل الخرج التالية :

```
Name's the same
Name's the same
Name's the same
```

توجد طريقة أخرى تقوم بنفس الوظيفة للطريقة ()equals ولكنها لا تميز بين الأحرف الكبيرة والصغيرة وهي ()equalsIgnoreCase

دالة المقارنة compareTo()

توجد في لغة Java طريقة مقارنه تعطي معلومات أكثر عن عملية المقارنة وهي compareTo() . عندما تستخدم هذه الطريقة لمقارنه غرضين (objects) من النوع String فإنها ستعيد القيمة (0) فقط إذا كان للغرضين نفس القيمة فإذا كان هناك اختلاف بين السلسلتين ستعيد الطريقة قيمه سالبه إذا الغرض المستدعى للطريقة اصغر من الوسيط الخاص بالطريقة وتعطي قيمه موجبه إذا كان العكس .

إن تحديد حرف ما أهو اقل أو أكبر من حرف أخر يعود إلى ترتيب ورود الأحرف ضمن الأبجدية إي أن الحرف (a) هو اقل من الحرف (b) والذي بدوره اقل من (c) وهكذا...

مثال: ليكن لدينا المتحول Name يحتوي على قيمة "Murad" وبالتالي إذا طبقنا الطريقة بالشكل("Name.compeneTo("Muaath

هذه الطريقة ستعيد قيمه موجبه وهذا يشير إلى أن الكلمة Murad اكبر من الكلمة Muaath وهذا لا يعني أن الكلمة Murad تحتوي على أحرف أكثر من الكلمة Muaath ولكن تسلسل الأحرف ضمن التسلسل الأبجدي هو الذي يحدد كبر احد المتحولين كالتالي:

- الحرف M في الكلمة Murad والحرف M في الكلمة Muaath متساويان
- الحرف u في الكلمة Murad والحرف u في الكلمة Muaath متساويان
- الحرف r في الكلمة Murad والحرف a في الكلمة Muaath غير متساويان

فالحرف r يقع بعد الحرف a في الترتيب الأبجدي, هذا يعني أنة أكبر.

محاضره (12)

تقنيات المصفوفة المتقدمة Advantage Array Techniques

1. ترتيب عناصر المصفوفه ذات الأنواع الأساسية Sorting primitive array elements

يقصد بالترتيب عملية تنظيم سلسلة من العناصر وفق ترتيب منطقي محدد فعندما نريد ترتيب مجموعة قيم بدءا من القيمة الأقل وانتهاء بالقيمة الأكبر يدعى هذا الترتيب بالترتيب التصاعدي Ascending order أما إذا قمنا بترتيب العناصر بالعكس أي من العنصر الكبير وحتى الصغير فهذا يدعى بالترتيب التنازلي Descending order.

نفترض إننا نريد تبديل القيم بين متحولين var1, var2 وقد ألحقت بهذين المتغيرين القيم التالية (10,12)

```
int var1=10;
int var2=12;
```

فإذا أردنا أن نبدل قيم المتغيرين السابقين var1=12& var2=10 , فسيكون من الخطأ كتابة الجملتين السابقتين بالشكل التالي:

```
var1=var2;
var2= var1;
```

وهذا يعني إننا أضعنا قيمة var 1 و هي القيمة 10 لان المتغير الأول سيأخذ قيمة المتغير الثاني 12 ثم يأخذ المتغير الثاني قيمة المتغير الأول و هي الأن 12. حل هذه المشكلة يكمن في استخدام متحول ثالث يلعب دور الوسيط وتكون طريقة التبديل بين المتغيرين كالتالي:

```
temp= var1;
var1= var2;
var2 =temp;
```

مثال: ترتيب متحولين ترتيباً تصاعدياً بحيث تصبح القيمة الأقل للمتغير var 1 سنكتب العبارات التالية لترتيب العناصر ترتيبا تصاعديا:

```
if (var 1>var2)
{
  temp= var1;
  var1= var2;
  var2 =temp;
```

```
}
```

نلاحظ أن ترتيب قيمتين عمليه سهله, ولكن ماذا لو كان عدد العناصر كبيرا, في هذه الحالة أفضل ما يمكن القيام به هو أن نخزن القيم في مصفوفة ثم نقوم بعمليه الترتيب التي نريدها بسهوله

مثال :

```
int [ ] numbers  = { 80 , 30 , 90 , 20 , 50 };
```

الان يجب مقارنه أول عنصرين في المصفوفة فإذا لم يكونا مرتبين تصاعديا نجري عمليه التبديل اللازمة بين العنصرين ضمن المصفوفة ليصبحا مرتبين تصاعديا ثم بعد ذلك نفحص العنصرين الثاني والثالث ونجري عمليه التبديل إذا لزم الأمر و هكذا حتى إنتهاء المصفوفة. هذه العمليات البرمجية التي تقوم بهذه المهمة موضحه بالشكل التالي:

```
for ( b = 0 ; b < 4 ; ++b )
    if (numbers [ b ] > numbers [ b + 1 ] )
    {
            temp = numbers [ b ];
            numbers [ b ] = numbers [ b + 1 ];
            numbers [ b + 1 ] = temp;
     }
```

إلى هذا الحد نكون قد وضعنا أكبر عنصر في نهاية اللائحة ولكن حتى نستطيع ترتيب باقي عناصر المصفوفة بنفس الطريقة المتبعة لابد من استخدام حلقتين متداخلتين العلوية من اجل تكرار عمليه المقارنة و الثانية للتمكن من مقارنه عنصر ما مع جميع عناصر المصفوفة. هذه العمليات البرمجية التي تقوم بهذه المهمة موضحه بالشكل التالي:

```
for ( a = 0 ; a < ( numbers.length – 1) ; ++a )
        for ( b = 0 ; b < ( numbers.length – 1) ; ++b )
            if ( numbers [ b] > numbers [b + 1] )
            {
                temp = numbers [ b ];
                numbers [ b ] = numbers [ b + 1 ];
                numbers [ b + 1 ] = temp;
             }
```

ملاحظه: عند القيام بترتيب سلسله من الأرقام فان هذا الترتيب يعتمد على الترتيب الحسابي لها , أما عند القيام بترتيب سلسله من الأحرف فان هذا الترتيب يعتمد شفرات هذه الرموز ضمن الشفرة Unicode

2- ترتيب مصفوفة من الأغراض Sorting array of objects

يمكننا ترتيب مصفوفة أغراض بنفس الطريقة التي استخدمناها في ترتيب مصفوفة عناصرها من أنواع أساسيه. الاختلاف الرئيسي بين النوعين (النوع الاساسي و النوع الكلاس) يحدث عندما نقوم بتنفيذ عمليه المقارنة والتي تحدد متى يجب تنفيذ عمليه المبادلة بين عنصري مصفوفة. فعمليه المقارنة يجب أن تجري على الحقول المكونة لكل غرض.

مثال: ليكن لدينا برنامج يقوم بإدخال رقم الموظف وراتبه الشهري وبالتالي عند كل إدخال سينشئ غرض له حقلان الأول لرقم الموظف و الثاني لراتب الموظف . لإنشاء مثل هذه الأغراض لابد من إنشاء مصفوفة عناصرها أغراض objects .

يمكن إنشاء مثل هذه المصفوفة بالعبارة التالية داخل الدالة main مثلا:

```
Employee [  ] someEmp = new Employee [ 5 ]
```

الصنف Employee يمتلك الشكل التالي :

```
public class Employee
{
    private int empNum;
    private double empSal;
    public Employee(int e , double s)
    {
        empNum = e;
        empSal = s;
     }
    public int getEmpNum()
    {
        return empNum;
    }
    public double getEmpSal()
    {
        return empSal;
     }
```

```
    public void setEmpNum( int n )
    {
          empNum = n;
     }
     public void setEmpSal(double s)
    {
          empSal = s;
     }
}
```

بعد إنشاء هذه المصفوفة من الأغراض يتم تمريرها إلى طريقه تتطلب وسيطين الأول هو اسم المصفوفة التي تحوي الأغراض , و الثاني هو عدد عناصر المصفوفة تسمى هذه الطريقة arraySort()

الطريقة array Sort() تمتلك الشكل التالي:

```
public static void arraySort(Employee[] array , int len)
{
    int a , b;
    Employee temp;
    int arraylength = len – 1;
    for ( a = 0 ; a < arraylength ; ++a )
        for ( b = 0 ; b < arraylength ; ++b )
          if ( array [ b ] . getEmpSal() > array [ b+1 ] . getEmpSal ())
            {
              temp = array[ b ];
              array[ b ] = array[ b + 1 ];
              array [ b + 1 ] = temp;
            }
}
```

أسلوب هذه الطريقة مخصص لترتيب مصفوفة من الأغراض وكل غرض يحتوي على حقول , وهذا الأسلوب يعتمد على اختيار احد حقول الأغراض ليكون المرجعية في ترتيب أغراض المصفوفة و في مثالنا هذا تم اعتماد الحقل empSal الذي يخزن راتب الموظف.

3- ترتيب السلاسل Sorting strings

عند التفكير في كتابة برنامج يقوم بترتيب مصفوفة عناصرها من النوع string, يجب الانتباه إلي أن السلاسل تسمى بالعناوين addresses,وبالتالي لن نتمكن من تحديد السلاسل الواجب تنفيذ عملية التبديل عليها وذلك لعدم التمكن من تحديد الأكبر والأصغر بسهوله

مثال: ترتيب مجموعه من الأسماء وطباعتها حسب الترتيب .

```
public class sortString
{
 public static void main(string[]args)
  {
    String[] students={"karim","kamal","Tamer"};
    int x;
   system.out.println("Befor sort");
   for(x=0;x<students.length;x++)
        system.out.println(students[x]);
    sortString(student, student.length);
    system.out.println("\n after sort \n");
    for(x=0;x<students.length;x++)
           system.out.println(student[x]);
   }//end of main
  Public static void sortString(String[]array, int len)
    {
      int a,b;
      String temp;
      int highsize=len-1;
      for(a=0;a<highsize;++a)
          for(b=0;b<highsize;++b)i
             if (array[b].compareTo(array[b+1])>0)
               {
                 temp=array[b];
                 array[b]= array[b+1];
                 array[b+1]=temp;
                }
    }//end sortString
}// end of the class
```

محاضره (13)

الوراثة Inheritance

1. مفهوم الوراثة

تتم عملية الوراثة بين الأصناف من خلال إشتقاق صنف من صنف أخر , ففي هذه الحالة يرث الصنف المشتق Subclass جميع محتويات الصنف المشتق منه Super Class بإستثناء المحتويات المعرفة علي أنها خاصة private . و تتم عملية اشتقاق الأصناف من بعضها بإستخدام الكلمة المحجوزة extends. لغة جافا لا تدعم الوراثة المتعددة أي أن الصنف لا يمكن أن يرث أكثر من صنف.

مثال:

```
//B.java
 class A
 {
   …
   …
   .. .
  } // end of class A
 public class B extends A
{
  …
  …
  …
 }
```

في هذا المثال تم تعريف الصنف A و من ثم تم تعريف الصنف B المشتق من الصنف A و الذي سيرث كل محتوياته و نستطيع أن نقول أن الصنف A هو الأب و الصنف B هو الإبن, حيث سيرث الابن بعض أو كل صفات الأب.

2. الوصول للطرق و البيانات الموروثة

يتم الوصول للبيانات و الطرق لصنف ما بشكل مباشر داخل الصنف المشتق و كأنها معرفة داخله باستثناء البيانات و الطرق المعرفة علي أنها خاصة بالصنف الذي عرفت فيه حيث أنها لا تورث للأصناف المشتقة من هذا الصنف.

مثال:

```
// Cars.java
class Transportation
{
  protected static int x=12;
  private int y=19;
  public static void meth1()
  {
   System.out.println("Calling meth1() from class Cars.");
   }
  private void meth2()
   {
    System.out.println("will not be called from Cars");
   }
} // end of class Transportation
public class Cars extends Transportation
{
 public static void main(String args[ ])
 {
   meth1();
   System.out.println(x);
   // meth2(); // meth2() has private access in Transportation
   // System.out.println(y); // y has private access on Transportation
  }
} // end of class Cars
```

في المثال السابق تم تعريف الصنف Transportation ليحتوي علي متغيرين صنف هما :x و هو محمي و y و هو خاص و يحتوي علي طريقتين الطريقة العامة و هي meth1() و الطريقة الخاصة meth2() و من ثم تم تعريف الصنف Cars بحيث تم إشتقاق هذا الصنف من الصنف Transportation و في هذه الحالة يعتبر الصنف Transportation هو الصنف الأب Super Class و الصنف Cars هو الصنف الإبن SubClass. في هذا المثال يتم توريث يتم توريث جميع صفات الصنف الأب Transportation إلي الصنف الإبن Cars باشتثناء الصفات المعرفة علي أنها خاصة. في الصنف الإبن تم استخدام المتغير x و الطريقة meth1() و المعرفين في صنف الأب Transportation علي أنهما قابلين للتوريث, بينما لم نستطع إستخدام المتغير y و الطريقة meth2() لأنهما معرفان علي أنهما خاصين بالصنف الأب Transportation و عند محاولة استخدامهما تظهر رسائل خطأ كما هو موضح في المثال.

عندما تقوم الطريقة المعرفة داخل الصنف باستخدام أحد مكونات هذا الصنف ففي بعض الأحيان لا يوجد هناك ما يثبت أن هذه العملية تجري علي المكون الصحيح لذالك فإن لغة الجافا توفر المؤشر this الذي يشير إلي الهدف أو العنصر المطلوب و بالتالي فإن هذا الموشر يضمن أن تتم العملية علي المكون الصحيح.

مثال: كيف يوضح يتصرف البرنامج بدون استخدام this

```
//C.java
class A
{
  protected int a=9;
 } // end of class A
 class B extends A{
  void test()
   {
      int a=22;
      System.out.println("a = "+a);
    }
 } // end of class B
 public class C{
  public static void main(String args[])
  {
    B acc=new B();
    acc.test();
    }
} // end of class C
```

مخرجات هذا البرنامج هي طباعة ما يلي: a=22 لأن جملة الطباعة ستخرج المتغير الخاص بالطريقة ( )test و ليس المتغير الخاص بالصنف.

مثال: كيف يوضح يتصرف البرنامج باستخدام this

```
//C.java
class A
{
  protected int a=9;
 } // end of class A
 class B extends A{
  void test()
   {
      int a=22;
      System.out.println("a = "+this.a);
    }
 } // end of class B
 public class C{
  public static void main(String args[])
  {
    B acc=new B();
    acc.test();
    }
} // end of class C
```

مخرجات هذا البرنامج هي طباعة ما يلي: a=9 لأن جملة الطباعة ستخرج المتغير الخاص بالصنف و ليس المتغير الخاص بالطريقة ( )test, و ذالك بإستخدام المؤشر this و الذي حدد أن المتغير المقصود هو المتغير a الموجود في الصنف الحالي B و بما أن الصنف B لا يحتوي علي متغير باسم a سيتم إستخدام المتغير a التابع للصنف A و الذي تم اشتقاق الصنف الحالي منه.

بينما إذا أحتوي الصنف المشتق SubClass و الصنف المشتق منه SuperClass علي متغير أو طريقة بنفس الإسم, ففي هذه الحالة سواء تم إستخدام المؤشر this او لم يتم أستخدامه داخل الصنف المشتق SubClass فانه يتم الرجوع للمتغير أو الطريقة التابعة لهذا الصنف المشتق SubClass و لن يتم الرجوع بأي حال من الأحوال إلي المتغير أو الطريقة المعرفة داخل الصنف المشتق

منه SuperClass. عند الحاجه للرجوع للطريقة أو المتغير التابع للصنف المشتق منه Super Class من داخل الصنف المشتق SubClass مع وجود تعريف لطريقة أو لمتغير بنفس الإسم داخل الصنف الحالي SubClass لابد من إستخدام المؤشر super و الذي يمكننا من إستخدام محتويات الصنف المشتق منه.

مثال: يوضح يوضح كيف يتصرف البرنامج باستخدام super و بدون إستخدام هذا المؤشر

```
//C.java
class A
{
  void test1()
  {
    System.out.println("The test1() method in class A");
   }
 } // end of class A
 class B extends A
 {
   void test()
   {
     super.test1();
     test1();
    }
    void test1()
    {
      System.out.println("The test1() method in class B");
    }
  } // end of class B
  public class C
   {
    public static void main(String args[])
    {
      B acc=new B();
      acc.test();
     }
  } // end of class C
```

مخرجات البرنامج:

```
The test1() method in class A

The test1() method in class B
```

3. إستبدال الطرق الموروثة Methods overriding

قد نحتاج في بعض البرامج إلي استبدال و تغيير طبيعة عمل الطرق الموروثة من صنف الأب إلي صنف الإبن, ففي هذه الحالة لابد من استبدال الطريقة الموروثة Methods overriding و ذالك بإعادة تعريف هذه الطريقة بالشكل الذي نريد, و اذا أحتجنا إلي الوصول للطريقة الموروثة لابد من إستخدام المؤشر super

```
// Override.java
class A {
  int i, j;
  A(int a, int b)
  {
    i = a;
    j = b;
   }
   // output  i & j
  void show()
   {
     System.out.println("i  and  j: " + i + "  " + j);
   }
}
class B extends A
{
  int k;
  B(int a, int b, int c)
  {
    // call the superClass constructor
    super(a, b);
```

```
    k = c;
  }
  // overriding the superClass method  show() in A)
  void show()
  {
     System.out.println("k: " + k);
   }
}
class Override {
   public static void main(String args[])
  {
     A Ob = new A(1, 2);
     B subOb = new B(1, 2, 3);
     Ob.show(); // call  show() of superclass A
     subOb.show(); // call  show() of superclass B
     }
  }
```

محاضره (14)

تكملة الوراثة Inheritance

1. مفهوم التغليف Encapsulation

كل كائن يخزن حالته في واحد أو أكثر من المتغيرات فالشكل العام لتعريف متغير كائن هو :

access_Specifier Type variable_Name;

مثال:

private double balance;

access_Specifier: محدد الوصول للمتغير و بصفة عامة نستخدم المحدد خاص لمتغيرات الكائنات,

و هذا يعني أنه يمكن الوصول إلي هذه المتغيرات فقط بدوال نفس الفصيلة المعرف فيها هذا المتغير و لا يمكن تغييره من أي دالة أخري ولذالك نستطيع القول بأن متغير الكائن غير ظاهر للمبرمج الذي يستخدم الفصيلة المعرف فيها هذا المتغير, هذه العملية من إخفاء البيانات data hiding تسمي تغليف Encapsulation و هذا مبدأ هام من مبادئ البرمجه بالكائنات OOP.

لكل كائن من الكائنات المنشأه من صنف ما نسخته من المتغيرات

مثال:

Account myAcc=new Account();

myAcc. balance;

myAcc.deposit(500);

2. الواثة Inheritance

الوراثة هي المبدأ الثاني من مبادءئ البرمجه بالكائنات OOP و التي يمكن الإستفادة منها في لغة جافا لتطوير البرامج حيث يمكن إستخدم الفصائل التي تم تصميمها و تنفيذهاو تاكدنا من أنها تعمل بصورة جيدة ثم نكتب فصيلة جديدة يضاف إليها الطرق و البيانات الجديدة فقط و ترث الطرق و البيانات الموجودة في الفصيلة القديمة التي يمكن إعتبار الفصيلة الجديدة إمتدادا لها.

خصائص الوراثة:

- إعادة إستخدام شفرة البرنامج code reuse حيث يمكن إستخدام فصائل موجودة و بذالك نوفر الجهد المبذول لكتابة الفصائل التي نحتاجها لحل مشكلة ما.
- الفصيلة التي تورث تسمي الفصيلة العامة (العليا) super class لأنها تحتوي علي الطرق و البيانات المشتركة و تسمي فصيلة الأب parent class.
- الفصيلة التي ترث تسمي الفصيلة الفرعية subclass لأنها تحتوي علي الطرق و البيانات الخاصة المضافة و تسمي الفصيلة الإبن child class.
- لتحقيق عملية الوراثة (توريث فصيلة قديمة إلي فصيلة جديدة) عند إنشاء الفصيلة الجديدة نقوم بكتابة إسم الفصيلة الجديده ثم الكلمة المحجوزة الدالة علي الوراثة extends و التي تعني أن هذه الفصيلة إمتداد للفصيلة القديمة التي يكتب إسمها بعد كلمة extends, الصيغة العامة للوراثة هي:

class subclassName extends superclassName

Example:

class B extends A

- عندما نقوم بإنشاء فصيلة و لا نحدد اسم الفصيلة الأب تفترض لغة جافا أن هذه الفصيلة ترث من الفصيلة الأم Object, الفصيلة Object تحتوي علي عدد صغير من الطرق العامة لجميع الكائنات في جافا
- عند إنشاء كائن جديد من الفصيلة الفرعية ينادي أولا المنشيء الموجود في الفصيلة العامة ليعطي قيمة إبتدائية للمتغيرات الموجودة فيها ثم ينفذ المنشيء الموجود في الفصيلة الفرعية لإعطء قيم أولية للمتغيرات الجديدة.
- الكائن المنشئ من الفصيلة الفرعية يعتبر حالة خاصة من الفصيلة العامة و يمكنه استدعاء الطرق الموجودة فيها كما ينادي الطرق الموجوده في الفصيلة الفرعية.

مثال يبين الوراثة:

في المثال التالي نبنين وراثة الفصيلة B للفصيلة A نلاحظ من المثال التالي أن الدوال تعمل مع عناصر الفصيلة الاصلية و مع عناصر الفصيلة الفرعية المشتقة من الفصيلة الأصلية.

```
// Inheritance.java
// create super class A.
class A
{
  int i, j;
  void showij()
  {
    System.out.println("i and j: " + i + " " + j);
   }
}
// create subclass B
class B extends A
{
  int k;
  void showk( )
  {
    System.out.println("k: " + k);
   }
   void sum( )
   {
     System.out.println("i+j+k: " + (i+j+k));
    }
}
// class, that use two classes (A and B)
class Inheritance {
public static void main(String args[ ])
```

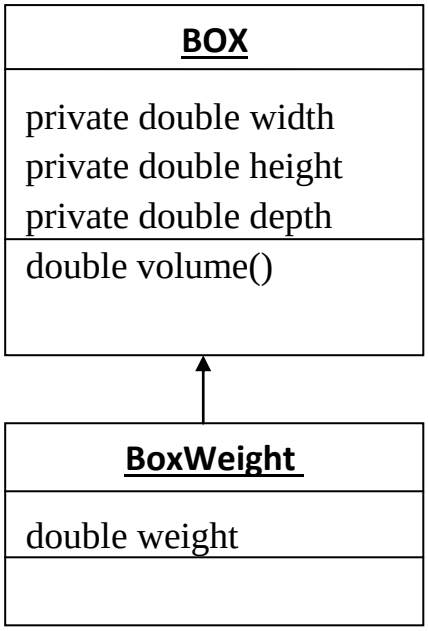
```
{
   A superOb = new A();
   B subOb = new B();
   superOb.i = 10;
   superOb.j = 20;
   System.out.println("Contents of superOb: ");
   superOb.showij();
   System.out.println();
   /* sub class can access to all public members of super class. */
   subOb.i = 7;
   subOb.j = 8;
   subOb.k = 9;
   System.out.println("Contents of subOb: ");
   subOb.showij();
   subOb.showk();
   System.out.println();
   System.out.println("Sum of i, j and k in subOb:");
   subOb.sum();
  }
}
```

3. المخطط الهرمي للوراثة inheritance hierarchies

المخطط الهرمي للوراثة عادة يمثل بشجرة حيث معظم الفصائل التي تمثل المفاهيم العامة قريبة من الجذر root و الفصائل الأكثر تخصصية قريبة من الفروع brunches سنوضح بمثال التسلسل الوراثي للفصائل.

مثال: في البرنامج التالي نقوم بتعريف فصيلة أساسية نسميها Box ثم نورثها للفصيلة BoxWeight لكي نستطيع إنشاء عناصر Box ذات أوزان, في داخل منشئ constructor الفصيله الفرعية نقوم بإستدعاء منشئ الفصيلة الأصلية عن طريق الكلمة المفتاحية super

يمكن تمثيل المخطط الهرمي للوراثة للفصيلتين بالشكل التالي:



البرنامج:

```
// DemoSuper.java

class Box {
  private double width;
  private double height;
  private double depth;
  // constructor of the super class
  Box(double w, double h, double d) {
    width = w;
    height = h;
    depth = d;
  }
  // calculate the volume
  double volume() {
    return width * height * depth;
  }
}
// the subclass “BoxWeight"
class BoxWeight extends Box {
  double weight; // object weight
  // constructor of the subclass
  BoxWeight(double w, double h, double d, double m) {
    super(w, h, d); // call the superClass constructor
```

```
    weight = m;
  }
}
class DemoSuper {
  public static void main(String args[ ]) {
    BoxWeight mybox1 = new BoxWeight(10, 20, 15, 34.3);
    double vol;
    vol = mybox1.volume();
    System.out.println("Volume of mybox1 is " + vol);
    System.out.println("Weight of mybox1 is " + mybox1.weight);
    System.out.println();
  }
}
```

4. الفصائل المجرده Abstract classes

ما معني الفصيلة المجردة abstract class؟

هي أعلي فصيلة في هرم التوريث superclass و هذه الفصيلة مفيدة لننشيء منها فصائل فرعية أما هي بحد ذاتها لا تستخدم بمعني أن الطرق الموجوده فيها لا تعالج أي بيانات و الصيغة العامة للإعلان عن هذا النوع من الفصائل هي كالتالي:

```
public abstract class className
{
    // class body
}
```

مثال:

```
// File  "AbstractAreas.java"
// abstract class
public abstract class Figure
{
    double dim1;
    double dim2;
    Figure(double a, double b)
    {
      dim1 = a;
      dim2 = b;
    }
    // abstract method area()
    abstract double area();
```

```
}// end of the class Figure
class Rectangle extends Figure
{
   Rectangle(double a, double b)
   {
      super(a, b);
    }
    // overriding area() method for Rectangle
    double area()
    {
       System.out.println("Inside Area for Rectangle.");
       return dim1 * dim2;
     }
}// end of the class Rectangle
class Triangle extends Figure
{
   Triangle(double a, double b)
   {
      super(a, b);
    }
    // overriding area() method for Triangle
    double area()
    {
       System.out.println("Inside Area for Triangle.");
       return dim1 * dim2 / 2;
     }
}// end of the class Triangle

class AbstractAreas
{
  public static void main(String args[ ])
  {
     // Figure f = new Figure(10, 10); // we can't do that
     Rectangle r = new Rectangle(9, 5);
     Triangle t = new Triangle(10, 8);
     Figure figref; // OK, the object will not be created
     figref = r;
     System.out.println("Area is " + figref.area());
     figref = t;
     System.out.println("Area is " + figref.area());
   }
}
```