Visual Basic .NET

في بداية الأمر خرج أناس عرفوا بإسم المهندسين والمصممين والعلماء اللذين قاموا بتصميم وهيكلة جهاز الحاسوب وقد نتج عن ذلك جهاز ضخم إحتوى على 5 طوابق تقريباً من النوع الكبير وكانت فكرة إنشاء جهاز حاسب قد نتجت أثر ضخامة المعلومات وإزالة طريقة الكتابة على الورق وإزالة الوقت الكبير للعمليات الحسابية الأربعة .

- كانت العمليات الأساسية للحاسوب تبرمج عن طريق النظام الثنائي 1 , 0 لأن جهاز الحاسوب لم يكن يفهم سوى هذا النظام .
- ثم ظهرت لغة التجميع المعروفة بالإسمبلي Assemble .
- IBM كانت من الشركات التي في هذا المجال بحيث كانت تصمم جهاز حاسب آلي وتضع فيه نظام تشغيل صغير Operating System (O.S) والتي أدت فيما بعد إلى تطور مصطلح نظام التشغيل والحاجة لتطويره .
- في ذلك الوقت ظهر طالبين من إحدى الجامعات الأمريكية تمكنا من تطوير وتصميم نظام تشغيل خاص بهما عرف باسم Dos (Disks Operating System) بمعنى نظام تشغيل الأقراص .
- عندما نظرت شركة IBM إلى النظام الذي صمم بواسطة هذين الطالبين تبنت نظامهما بلكامل وعملت على تشجيعهما .
- بواسطة لغة الأسمبلي كان العمل صعباً وكانت الجمل طويلة وعدد سطورها طويلة , فلو أراد المبرمج مثلاً طباعة كلمة فقط عليه أن يكتب 15 أو 20 سطر لتنفيذ ذلك . لذلك كان لابد من إيجاد طريقة جديدة لتوفير الوقت والتقليل من حجم البرنامج فقاموا بابتكار طريقة جديدة وهي طريقة الإجراءات حيث قاموا بوضع أكثر من Instruction بمكان واحد فقط , ومن هنا نشأ مفهوم المكتبات التي تضم أكثر من دالة وتطورت البرمجة فعرفت باسم Structured Programming أي البرمجة التركيبية . نتيجة لذلك طورت عدة لغات مثل COBOL و Basic و C و Pascal و Fortron والتي عرفت بالجيل الثالث للغات البرمجة .
- كانت شركة Apple Macintosh قد عملت على تطوير نظام فريد من نوعة فقد كان يستخدم الصور للدلالة على الأعمال , فقامت ببرمجة شكل الملف والمجلد و ... الخ , وذلك بدل الشاشة السوداء وبهذا النظام تم معرفة الـ Interface حيث أصبح هنا التفاعل بشكل مباشر من خلال الرسومات .
- شركة SUN كانت تمتلك النظام المفتوح Open Source المسمى Unix نسبة إلى صانعة وقامت بتشكيل نظام جديد عرف باسم Lnix والذي يستخدم الواجهات في عملة .
- ظهرت في هذه اللحظة التسابق لإنتاج نظام تشغيل كامل من واسطة كبريات الشركات (MS , SUN , Apple Mac) .
- وايضاً في هذه الأثناء وبينما كان هناك تقدم وتطور من قبل SUN , Apple Macintosh كانت Microsoft تعيش ركود نسبي لأنها كانت تعتمد نظام الـ Command Prompt في تلك الأثناء .
- شعرت Microsoft بعجزها وبتنحيها عن السوق فقامت بطرح تطبيق Application وليس نظام تشغيل يعتمد الواجهة الرسومية والذي عرف بنظام Windows 3.X .

- ظهرت الحاجة لتوحيد طريقة البرمجة لدى المبرمجين لكي يكون الكود ديناميكي وسلس وله قواعد وشروط , ولكي لا يخرج كل مصمم بأفكار بعيدة عن الهدف المقصود فظهر مفهوم الـ Object Oriented Programming (OOP) وأدت لظهور العديد من اللغات منها Small Talk , C++ بما عرفت بالجيل الرابع للغات البرمجة .
- بواسطة البرمجة الموجهة بالأهداف أنتجت Microsoft شبه نظام سمي Windows 95 وسبب أنه كان شبه نظام أنه كان يعتمد على نظام الـ Dos في أداءه و عمله .
- وظهرت الحاجة للإخراج جيل برمجيات متكاملة من خلال شركة Microsoft والتي سميت بـ Microsoft Office .
- وبعد سنتين طرحت Microsoft (Windows 97) ولكنه فشل سريعاً فبعد شهرين فقط تم سحبه من الأسواق لرداءته.
- ولكن سرعان ما قدمت Microsoft أول نظام تشغيل مستقل بواجهة رسومية والذي عرف باسم Windows 98 , وهنا ظهرت الحاجة لإنشاء برمجيات متكاملة على طريقة الـ Enterprise والتي تعني وجود جميع الحلول في مجموعة واحدة .
- قامت Microsoft بالتفكير بفكرة لإجبار جميع العملاء على إستخدام نظامها التشغيلي فقامت بطرح معالج لبعض اللغات مثل Basic , C , فسهلت على المبرمجين الحصول بالمجان على الـ API (Application Program Interface) والتي تعني واجهة البرامج التطبيقية المستخدمة من النظام , فقامت بالتسهيل على المبرمج فمثلاً إذا أراد المبرمج كتابة كود لإظهار نافذة فقط فإنه سيبقى وقت طويل جداً وسيكتب كود طويل جداً , فقامت Microsoft بطرح المكتبات لهذه الغاية بحيث ينتج نافذة كما في نظام التشغيل بوقت وكلفة قليلين جداً . وبذلك نجحت Microsoft بجذب العديد من المبرمجين والعملاء إليها بواسطة هذه الطريقة . ولكن البرنامج من هذه العملية لن يشتغل إلا على Windows .
- قام المبرمجين بكتابة البرامج المختلفة بواسطة نظام API لأنه كان سلس وسهل الإستعمال ونتيجة لذلك فقد زادت مبيعات شركة Microsoft أضعافاً مضاعفة والتي وصلت سنوياً إلى 3.5 مليار دولار .
- في هذه الأثناء قامت شركة SUN بطرح لغة جديدة من إنتاجها عرفت باسم JAVA لملاحقة Microsoft وبذلك تكون هذه الشركة قد قطعت شوطاً كبيراً بهذا المجال .
- ظهرت مكتبتان هنا من إنتاج الشركتان وهما :
  - SDK (Software Developer Kit) : من شركة Microsoft .
  - JDK (JAVA Developer Kit) : من شركة SUN .
- كان يوجد جانب لم تعيه كلتا الشركتين وهو قواعد البيانات Database والتي أصبحت حالياً الجانب المهم والأكبر في إستخدام الحاسب الألي فقد كانت شركة Oracle الرائدة في هذا المجال والسباقة إليه . وظهرت شركة Fox Pro التي أيضاً كانت من الشركات الكبير في هذا المجال .
- تنبهت شركة Microsoft باكراً إلى خطورة هذا الجانب فقامت بشراء ملكية شركة Fox Pro وضمتها لممتلكاتها وقامت لاحقاً بإنشاء المكتبة الخاصة للـ Database التى عرفت باسم SQL .
- وبإستخدام قواعد البيانات أنتجت Microsoft لغات جديدة (vB و VC++ وVFoxpro) وضمتها لقائمتها البرمجية .

- وباستخدام تكنولوجيا الإنترنت فقد قامت Microsoft بطرح لغة خاصة لمعالجة صفحات الإنترنت والتي سميت فيما بعد باسم InterDev وهي عبارة عن صفحات إنترنت نشطة ذات الإمتداد المعروف ASP (Active Server Pages) .
- كانت شركة SUN تعمل على تحديث وتطوير لغتها JAVA أول بأول من حيث قواعد البيانات والتعامل مع الإنترنت .
- بسبب الطمع والجشع الذي كان عند Microsoft فقد قامت بالخطوة القاتلة وهي إنها أنشأت لغة جديدة وسمتها باسم Visual J++ , ولكنها لم تسلم بتلك الفعلة فقد قامت شركة SUN برفع دعوة قضائية عليها ولأنها صاحبة اللغة فقد خسرت Microsoft تلك القضية وتم تغريمها مبالغ طائلة ومنعت من استعمالها والتطوير عليها لذلك السبب .
- ظهرت مشاكل عديدة في لغات Microsoft من أهمها :
  - إن مكونات COM التي كانت تعتمد عليها اعتمادا كلياً لغات Microsoft كانت تعتمد كلياً على مسجل النظام Windows Registry , وأي مشكلة تحدث في المسجل تؤثر على باقي المكونات المثبتة في الجهاز، ولن تستطيع استخدامها إلا بإعادة التركيب Reinstall وهذه العملية معقدة، إذ تتطلب منك نسخ ملفات المكونات ومن ثم تسجيلها في المسجل وإعدادها والتحقق من الإصدارات, وأي خطأ في عملية تثبيت البرامج، يؤدي إلى حدوث كارثة في جهاز المستخدم والتأثير على باقي البرامج المثبتة في الجهاز، ليكون الحل الوحيد إعادة تهيئة Format القرص الصلب. كما لا يمكنك استخدام إصدارين مختلفين لنفس المكون بسبب مشكلة تسمى Versioning ليس هذا مجال تفصيلها.
  - إن تطوير البرامج مسألة معقدة جدا وتتطلب دراية كافية في التعامل مع التقنيات المختلفة، فلكي تطور مواقع ويب ديناميكية عليك تعلم VBScrip (إن كانت من جهة العميل) وتعلم ASP (إن كانت من جهة الخادم) , وان أردت بناء نظم قواعد بيانات عملاقة عليك إتقان لغات الاستعلام المتقدمة كـ T-SQL للحصول على اكبر قدر من تحسين للكفاءة , وان أردت تطوير المكونات بفاعلية ودون حدود عليك تعلم احد لغات البرمجة المتقدمة كـ Visual C++ , وان أردت مخاطبة تطبيقات Microsoft Office الشهيرة فلا مخرج لك إلا باستخدام VBA, وإن أردت تطوير برامج تعمل تحت نظم Windows بسهولة وكسر حاجز الوقت فستجد ضالتك في Visual Basic. ليس هذا فقط، بل حتى الوظائف المتشابهة تنجز بتقنيات مختلفة، فلديك مثلا التقنيات RDO ،DAO،ADO,Data لتطوير التطبيقات المعتمدة على قواعد البيانات Databases , وهناك أيضا مجموعة من التقنيات كـ GDI،DirectX,OpenGL لتطوير النظم التي تعتمد على الصور والرسوم بكثرة.
  - وفي هذه الأثناء كانت شركة SUN تسير على الطريق الصحيح وتعمل على تطوير لغتها أولاً بأول بفضل إمتلاكها للغة واحدة وإمتلاكها لتقنية واحدة في التعامل مع قواعد البيانات (JDBC) .
- نظرت Microsoft بنظرة عميقة للموضوع وأسفر عن ذلك خطة جديدة مع العام 2000 وهي إخراج لغات موحدة لها نفس مكتبات التشغيل ولها إطار عمل واحد Framework وأدت إلى ظهور لغات جديدة عرفت بمجموعة الدوت نت (NET .) فقدمت لغات جديدة مثل Visual C# و Visual J# (لغة من تصميم Microsoft من الصفر وقاموا بتطويرها لتحاكي بشكل كبير جداً لغة JAVA) وضمتها في مجموعة Visual Studio 7 . Net .

وحقيقة حينما أصدرت (ميكروسوفت) أوّل نسخة من لغة **Visual Basic** عامَ **1991**، لم يكن في حسبانِها أنّها ستكتسبُ كلّ هذه الشهرةِ وستحقق كلّ هذه الشعبيّة! إنّ لغة **BASIC** القديمة تُعَدّ من أسهل لغاتِ البرمجة، ولكّنها لم تستطع الصمود في المنافسة مع لغاتِ البرمجة الأخرى بسبب قدراتِها المحدودة.

كانَ ذلك كذلك، حتّى أصدرت (ميكروسوفت) إصدارات **VB** المتتابعة، لتنتقل لغة **BASIC** من قِفار **Dos** المجدبة إلى مراعي **Windows** الخصبة، مانحةً للمبرمج القدرة على إنشاءِ برامج ذات واجهة مرئيّة، بأسهلِ طريقةٍ وفي أسرع وقت.

ومنذئذٍ ولغة **VB** تتصدّر قائمة مبيعات لغات البرمجة، لتدخلَ في بناءِ التطبيقاتِ التجاريّة وتطبيقاتِ قواعد البياناتِ البسيطة، وبرامج الوسائط المتعدّدة **Multimedia** والكثير من الألعاب.

ولكن للأسف.. دائما وأبدا كانت **VB** أدنى من باقي لغاتِ البرمجة، فتطبيقاتُها أبطأ نسبيّا وأكبر حجما، وتعاني من بعض أوجهِ القصور في الأداء.

ولقد استمرّت (ميكروسوفت) في تطوير **VB** عبرَ ستّ إصداراتٍ مختلفة، وفي كلّ إصدارٍ جديدٍ كانت تعالجُ بعضَ المشاكل القديمةِ وتضيفُ المزيدَ من القدرات، لتضيقَ الفجوةُ شيئًا فشيئا بين **VB** وباقي لغات البرمجة.

ثمّ أخيرا أقدمت (ميكروسوفت) على الخطوة التي طال انتظارها.. أصدرت نسخة جديدةً بكلّ المقاييس من **VB**، بنتها من جذورها **From scratch** لتجعلها نِدّا حقيقيّا لـ **++VC**، بحيثُ يمكنك أن تقولَ بثقة: إنّ العصرَ الذهبيَّ لـ **++VC** آخذٌ في الأفول بلا رجعة، حيثُ سينحصرُ استخدامها في تصميم المحرّكات **Engines** التي تدخل في بناء تطبيقات أخرى، أو في كتابة الكود الذي يتيح للكمبيوتر التحكمّ في آلات أخرى، ولكنّ استخدامها سيتراجع بلا شكّ في تطبيقات الإنترنت والتطبيقات التجاريّة وتطبيقات قواعد البيانات والوسائط المتعدّدة ومعظم الألعاب وما شابه، نظرًا لصعوبتِها وتعقيدِها وطول الوقت اللازم للبرمجة بها!

وأعتقدُ أنّ هذا هو السبب الذي دفع (ميكروسوفت) لإصدار اللغة الجديدة **#C**، التي تُعتبر توأما لـ **VB** إلا إنّها تستخدم قواعد **++C** في كتابة الأوامر، ممّا يشكّلُ لمبرمجي **++VC** إغراءً تصعبُ مقاومته للانتقال إليها.

ولكن مهما كانت سهولة **#C**، فإنّ **VB** يصرعُها في هذا المضمار، فهو أقربُ ما يكونُ للّغةِ الإنجليزيّة العاديّة، ولا يحتوي على الرموز الكثيرةِ المملّة التي تملأ **++C**، مثل ; ، ++ ، == ، ||.... إلى آخر هذه الرموز التي تجعلُ احتمالاتِ الخطإ عند كتابةِ الكودِ أعلى، وتجعلُ البرنامجَ أصعبَ فهمًا وأقلَّ ألفة عندَ قراءته.

الجديد في VB.Net

إنّ التطويرات التي لحقت بـ VB.Net كثيرة جداً بحيثُ لن نستطيع أن نحيط بها كلّها هنا.. ولكن يكفي أن نذكر منها ما يلي:

- لم تعد لـ VB واجهة استخدام مستقلّة، فكلّ لغات VS.Net تستخدم واجهة واحدة لكلّ المبرمجين ، مليئة بالأدواتِ التي تُسهّل بطريقة مدهشة عمليّة تصميم البرنامج.. إنّ هذه الميزة تسمح لك بإنشاء تطبيقات تدخل فيها أكثر من لغة برمجة .
- إمكانيّات جديدةٌ في نافذة محرّر الكود، منها قيام اللغة بكتابة جملة نهاية المقطع تلقائيّا، بمجرّد ضغط زرّ Enter.. فمثلا: لو كتبت جملة:

```
If X = 0 Then
```

فإنّ محرّر الكود سيضيف الجملة التالية تلقائيّا:

```
End If
```

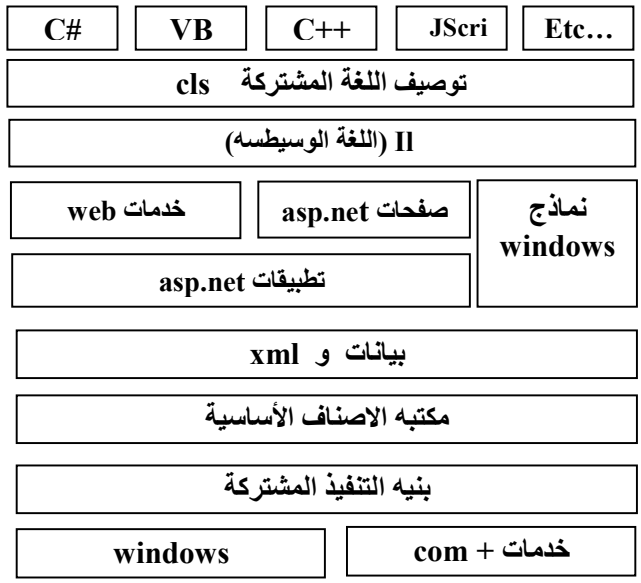
بل إنّك لو لم تكتب كلمة Then فسيكتبها لك محرّر الكود تلقائيّا! هذا بالإضافة إلى أنّه سيضع مؤشّر الكتابة داخل مقطع If، وسيقوم بتنسيق المسافات البادئة تلقائيّا، بحيث يبدو الكود منظّما وواضحا عند قراءته.

كمّا أنّ هناك تحسيناتٍ كثيرةً في تلميحات الشاشةِ التي تعرضُ قيمَ المتغيّراتِ وأنواعَها ومعاملاتِ الإجراءاتِ والدوالِّ وقيمَها المعادة، مع نبذة عن وظيفة كلّ دالة وكلّ معامل , Wordwrap , Line Numbers , ...

- لم يعد هناك أيّ قصور في مترجم الكود Compiler، فكلّ لغات VS.Net تعمل على مترجم واحد، ممّا يعني أنّ VB قد صار بقوّة وسرعة وكفاءة VC++ .
- أصبح بإمكانك معالجة الأخطاء، عن طريق استخدام معالجات الاستثناءات Exception في جملة Try.. Catch.. End Try.
- VB.Net مبنيّة بالكامل على مفهوم البرمجةِ بالكائنات Object Oriented Programming، لدرجة أنّ الأعدادَ الصحيحة Integers والنصوص Strings والمصفوفات Arrays قد صارت خلايا Classes، وصارت لهذه العناصر خصائصُ ووسائلُ جاهزة.. فمثلا، أصبحت لديك وسائل جاهزة تنتمي للمتغيّر النصّيّ، تسمح لك بالبحث فيه أو تقطيعه أو استبدال أجزاء منه... إلخ... كما صارت لديك وسائلُ جاهزةٌ لعكس المصفوفةِ وترتيبها والبحثِ فيها!
- صار بإمكانك استخدام مفاهيم الوراثة Inheritance,وإن كان مسموح بوراثة خليّة واحدة فقط Single Inheritance للتسهيل.
- يمنحك إطار العمل Net Framework. ثروة هائلة من الخلايا Classes، تقدّر بأكثر من 5000 خليّة رئيسية ، تفعل كلّ ما تحلم به وأكثر، بحيث تريحك بدرجة كبيرةٍ من الاحتياج لاستخدام دوال API الخاصّة بالويندوز، بما فيها من تعقيد ومشاكل..
- أصبح بإمكانك تعريف المتغيّرات داخل مقاطع الجمل الشرطيّة If Statements والجمل التكراريّة Loops، بحيث تكون معزولة عن المتغيّرات الموجودة خارج هذه المقاطع .
- هناك إمكانيّات هائلة في مجال الرسم والتلوين تمنحها لك مكتبة +GDI.. يكفي أن تعرف أنّ بإمكانك الآن رسم منحنيات معقّدة، وتكوين أشكال مركّبة من مجموعة خطوط ومضلّعات ومنحنيات، وتلوين السطوح بألوان متدرّجة، وتحديد شكل مساحة الرسم , فلتره الصور و دمجها بجودة عالية جداً لم تكن بحسبان أحد!! , وتحديد درجة الشفافية، وتدوير الرسوم وتغيير مقاييسها تكبير أو تصغير ..
- يمكنك استخدام المؤشّرات Pointers في بعض الأحيان، للقراءة والكتابة في الذاكرة، كما يمكنك أداء عمليّات معقّدة على النظام System لم تكن لتحلم بها!
- أصبح بإمكانك تقسيم برنامجك لمجموعة من العمليّات المستقلّة Threads، ممّا يعني أنّ برنامجك يستطيع القيام بأكثر من عمليّة في نفس اللحظة Multithreading.
- هناك إمكانيّات جديدة رائعة للتعامل مع قواعد البيانات، تمنحها لك تقنية ADO.Net، بالإضافة للعديد من الأدوات المرئيّة التي تساعدك في إنشاء تطبيقات قواعد البيانات بأقلّ قدر من الكود وأكثر دقة .
- أمّا الجديد تماما، فهو قدرتك على تصميم صفحات الإنترنت بنفس الطريقة التي تصمّم بها النماذج العاديّة، مع كتابة كود VB بمعظم إمكانيّاته، لإنشاء تطبيقات ASP تعمل على الخوادم Servers بدون كتابة حرف واحد من لغة ASP!!

محتويات إطار العمل .NET Framework

والآن سيتمحور حديثي حول معمارية ومحتويات إطار عمل .NET Framework. , يمكنني ان اقسم لك محتوياته إلى أكثر من 10 طبقات، ولكني فضلت تقليص العدد– للتسهيل عليك -كما ترى في أحد الشكلين التاليين:-

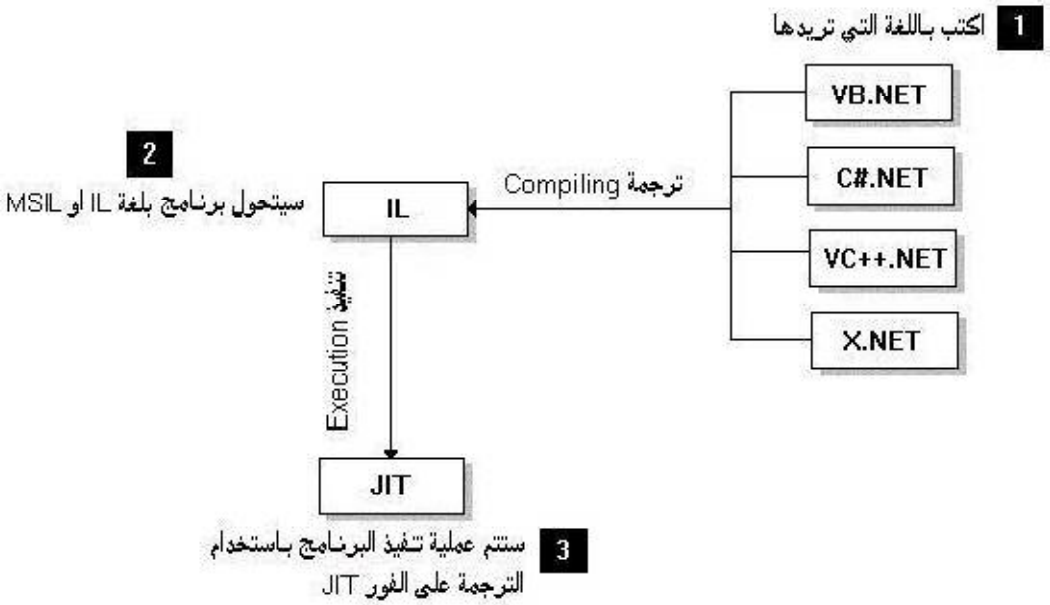


مكتبة فئات Base Class Library (تسمى أيضا مكتبة فئات .NET Framework Class Library) هي عبارة عن مئات الفئات الموجودة في عشرات ملفات الـ DLL تعتبر كنزا غاليا يسيل له لعاب المبرمجين الجادين، حيث تحتوي كل ما تحتاجه لإنجاز برامجك ومشاريعك بدءاً بتقديم فئات لوصف البيانات الأساسية (كـ Integer ،String) إلى إدارة خرج ودخل الملفات I/O File Processing , مسارات التنفيذ Threading , الصور والرسوم، نماذج Windows Forms , نماذج Web Forms , الاتصال بقواعد البيانات ADO .NET. وغيرها الكثير.

بالنسبة للغة التنفيذ المشتركة Common Language Runtime – (CLR) فهي موحدة لمعايير جميع لغات .NET. الأخرى، كما أنها المسئولة عن عمليات إدارة الذاكرة Memory Management , تفريغ مصادر النظام باستخدام Garbage Collection أخطاء وقت التنفيذ Exception Handling .

أخيرا، نظام التشغيل الذي يمكنك من تثبيت إطار عمل .NET Framework.عليه هو Windows فقط ، ولكن قد ترى في القريب العاجل نظم تشغيل أخرى داعمة له.

أما أسلوب الترجمة على الفور Just In Time Compiling - (JIT) , وهي تقنية تقوم بترجمة البرنامج عند تنفيذه حيث ينتج أفضل شيفرة تتناسب مع الجهاز الذي سيعمل عليه البرنامج مما ينتج عنه نتائج إيجابية جيدة (هذا عند الحديث عن تحسين الكفاءة Optimization) وحتى تعلم كيف يحدث ذلك تابع الشكل التالى :-



مراحل ترجمة وتنفيذ البرنامج

اختر أي لغة تناسب مزاج أناملك واكتب الشيفرة بها، وعند قيامك بعملية الترجمة سيتم تحويل ملف البرنامج إلى ملف شبيه بالملفات التنفيذية Executable File مكتوب بلغة جديدة اسمها Intermediate Language (تختصر IL أو MSIL). حيث تحتوي على شيفرات البرنامج ولكنها غير قابلة للتنفيذ مباشرة، بل يشترط وجود مترجم على الفور JIT Compiler حيث يقوم بترجمة هذا الملف الثنائي إلى لغة الآلة معطيا أفضل التعليمات بحيث تناسب الجهاز الحالي، فعندما تصمم برنامجك مرة واحدة فقط باستخدام Visual Basic .NET فاعلم ان البرنامج سيستفيد من كل مصادر العتاد ونظام التشغيل الذي يتم تنفيذ البرنامج فيه, عملية الترجمة JIT تقوم بترجمة كل جزء من البرنامج عند الحاجة أي عند استدعاء وظيفة معينه فيه، مع معرفة أن الترجمة تتم مرة واحدة فقط، ولن يشعر المستخدم بأي بطء في عملية تنفيذ البرنامج عند تشغيله مرة أخرى، فقد تمت ترجمته بالشكل المناسب لجهازه .

بيئة التطوير المتكاملة *Integrated Development Environment*

إنَّ VS.NET هي بيئة تطوير متكاملة Integrated Development Environment (IDE)، لبناءِ واختبار وتصحيح البرامج المتنوّعة: تطبيقات ويندوز Windows Applications، تطبيقات الإنترنت Web Applications، الخلايا Classes، وأدوات التحكم الخاصّة Custom Controls، وهي تمنحُك العديدَ من الأدواتِ المرئيّةِ Visual Tools لإنشاء واجهة التطبيق Application Interface بشكلٍ شبه آليّ، ولتسريع وتسهيل أداءِ العديدِ من المهامِّ في التصميم والبرمجة.

لهذا فقد خُصّص هذا الفصلُ ليجعلكَ تألفُ هذه البيئة، وتعرفُ كيفَ تُساعدُ أدواتُها على سرعةِ وسهولةِ تصميم واجهةِ البرنامج، وكتابةِ الكود.

ونحنُ هنا بصددِ أن نتعرّفَ على المكوناتِ الرئيسيّةِ لبيئةِ التطوير المتكاملة IDE، خاصّة تلك التي سنحتاجُ إليها لبناءِ تطبيق ويندوز بسيط.

صُمّمت VS.NET لاستضافةِ أيِّ لغةٍ من لغاتِ البرمجة، وهناكَ الآنَ بالفعلِ عديدٌ من الشركاتِ التي تعملُ على تطوير لغاتٍ مختلفةٍ لتتوافقَ مع VS.NET، وقريبًا ستجدُ أنّ بعضَنا يبرمجُ تطبيقات ويندوز في VS.NET بلغة كوبول أو لغة فورتران!

إنَّ الواجهة المرئيّة للبرنامج ليست مرتبطة بلغةِ برمجةٍ بعينِها، ونفسَ الأدواتِ التي ستستخدمُها لتطوير واجهةِ تطبيقك، ستجدُ أنَّ كلَّ المبرمجينَ يستخدمونَها مهما اختلفتِ لغةُ البرمجةِ التي يستخدمونَها لكتابةِ الأوامر!

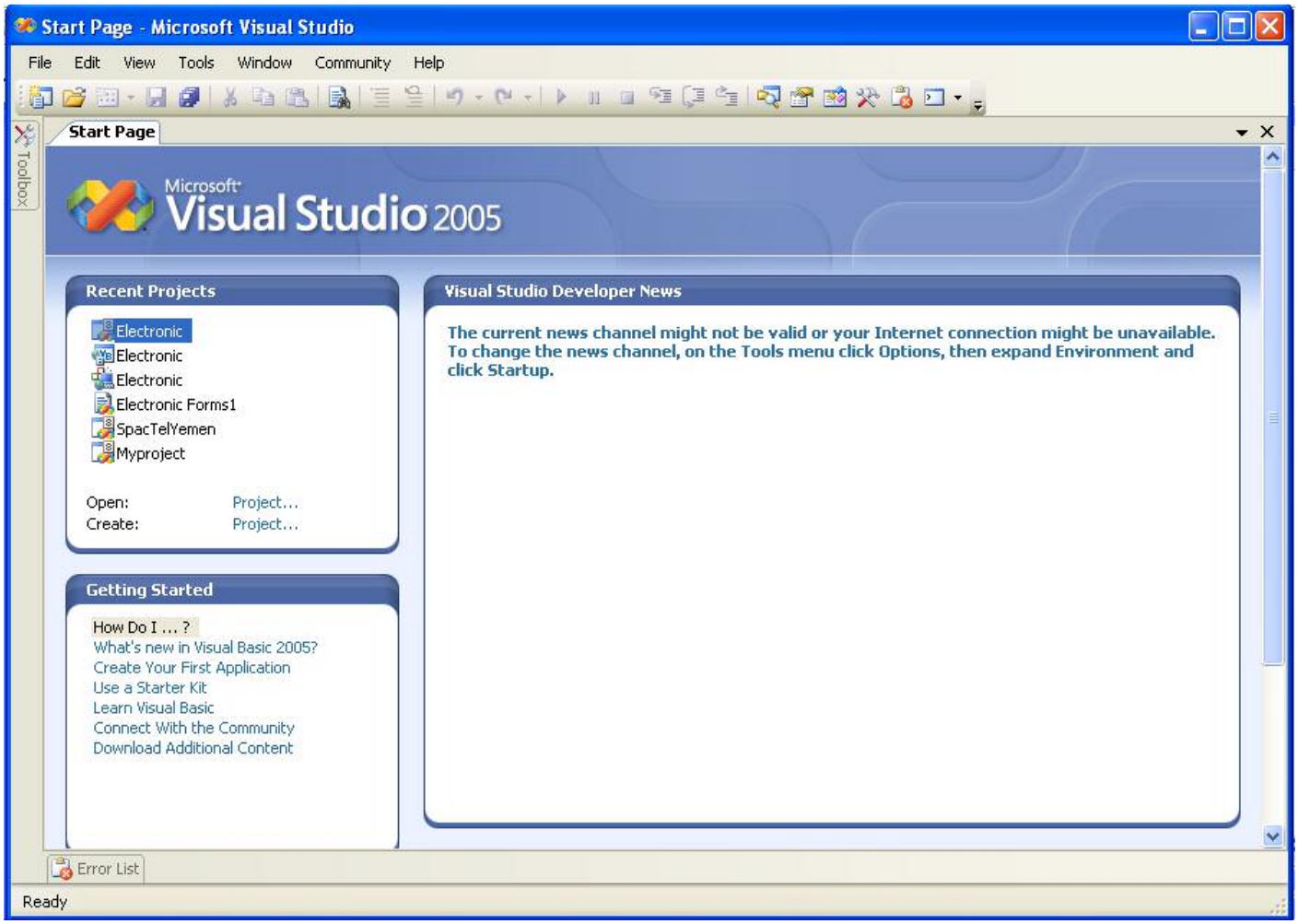
ينطبقُ نفسُ الأمر كذلك على الأدواتِ التي ستستخدمُها للاتصال بقواعدِ البيانات والتعامل معها، فهي غير معتمدةٍ على اللغةِ التي تستخدمُها.

إنّ جعلَ بيئةِ التطوير مستقلّة عن لغاتِ البرمجة، يمنحُ المبرمجينَ سهولة في تعلّم أيّ لغة، حيث لن يضطرّوا لتعلّم البيئة الخاصة بكل لغة جديدة يقدمون على تعلّمها، كما أنّ ذلك يجعلُ استخدام كائناتٍ Objects كتبت بلغاتٍ مختلفةٍ أمرًا أيسر وأكثرَ توافقًا.. ليس هذا فحسب، بل يتيح كذلك إنشاء تطبيقات تستخدم أجزاءً مكتوبة بلغات برمجة مختلفة، مع إمكانيّة فتح كلّ هذه الأجزاء معا لتصميمها في وقت واحد!

هذا، وتحتوي بيئة التطوير المتكاملة IDE على العديدِ من الأدواتِ المرئيّةِ، كمصمّم القوائم الرئيسية Menu Designer، الذي يساعدكُ على إنشاءِ القوائم الرئيسية Menus، وتحديدِ أسمائها وخواصّها، وما يندرج تحتها من أوامر Commands و قوائم فرعية Submenus، كلّ ذلك بطريقةٍ مرئيّة (أي باستخدام الفأرةِ Mouse ولوحةِ المفاتيح Keyboard ونافذة الخصائص Properties Window، بدلا من كتابةِ كود لذلك).

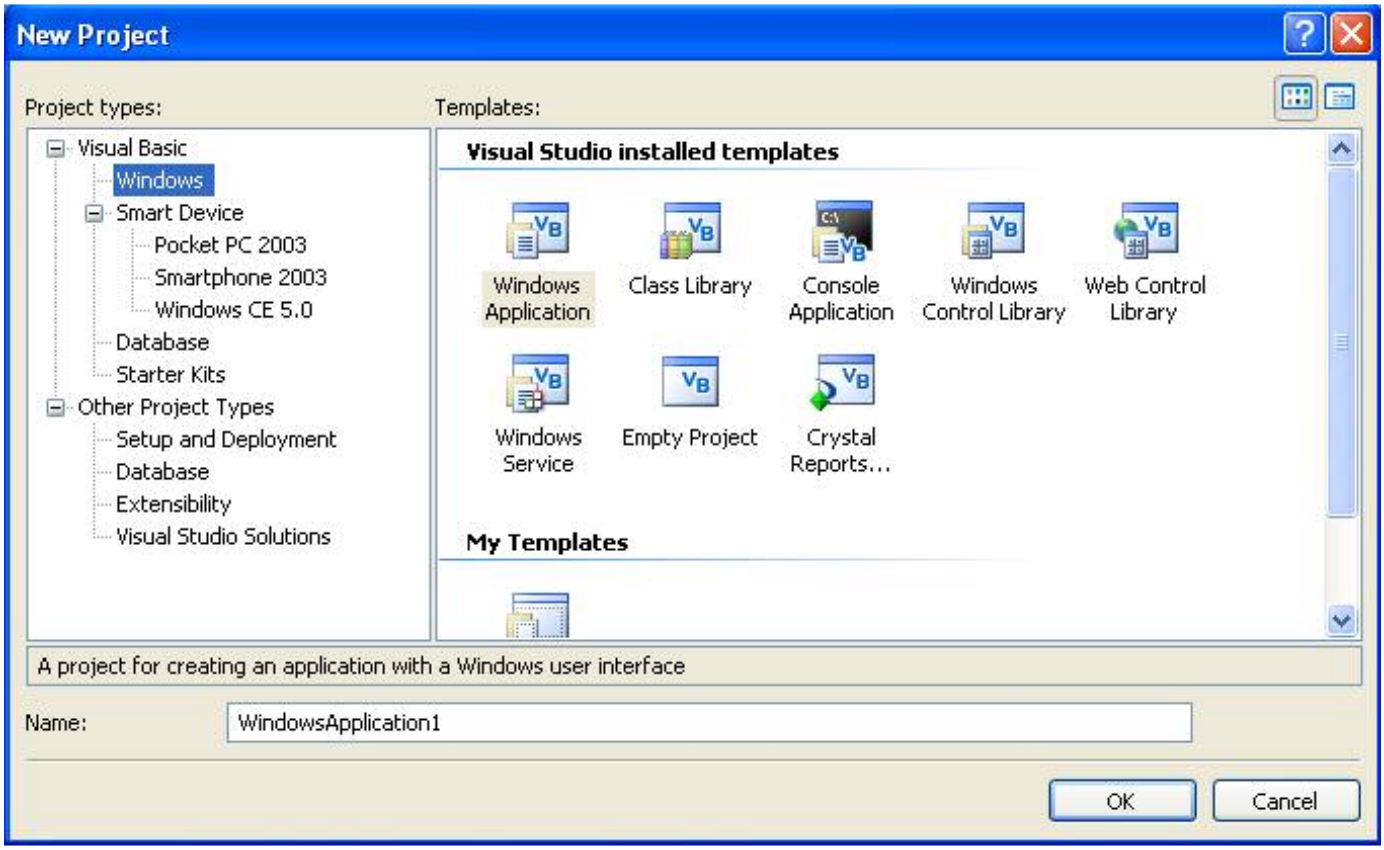
كما تحتوي بيئة التطوير كذلك على الأدواتِ التي تُمكّنك من تصميم Design وتنفيذ Execute وتصحيح أخطاء Debug برامجك.

صفحةُ البداية Start Page:



والشئ الأهم في هذه الشاشة هو احتوائها على الجزء Recent Projects والتي يتعلق بإظهار أخر مشاريع تم إنشاءها أو تم التعامل معها , بالإضافة إلى إمكانية فتح أي مشروع والذي تم تخزينه في إحدى وحدات التخزين من خلال الخيار Open, بالإضافة إلى إمكانية إنشاء مشروع جديد من خلال الخيار Create .

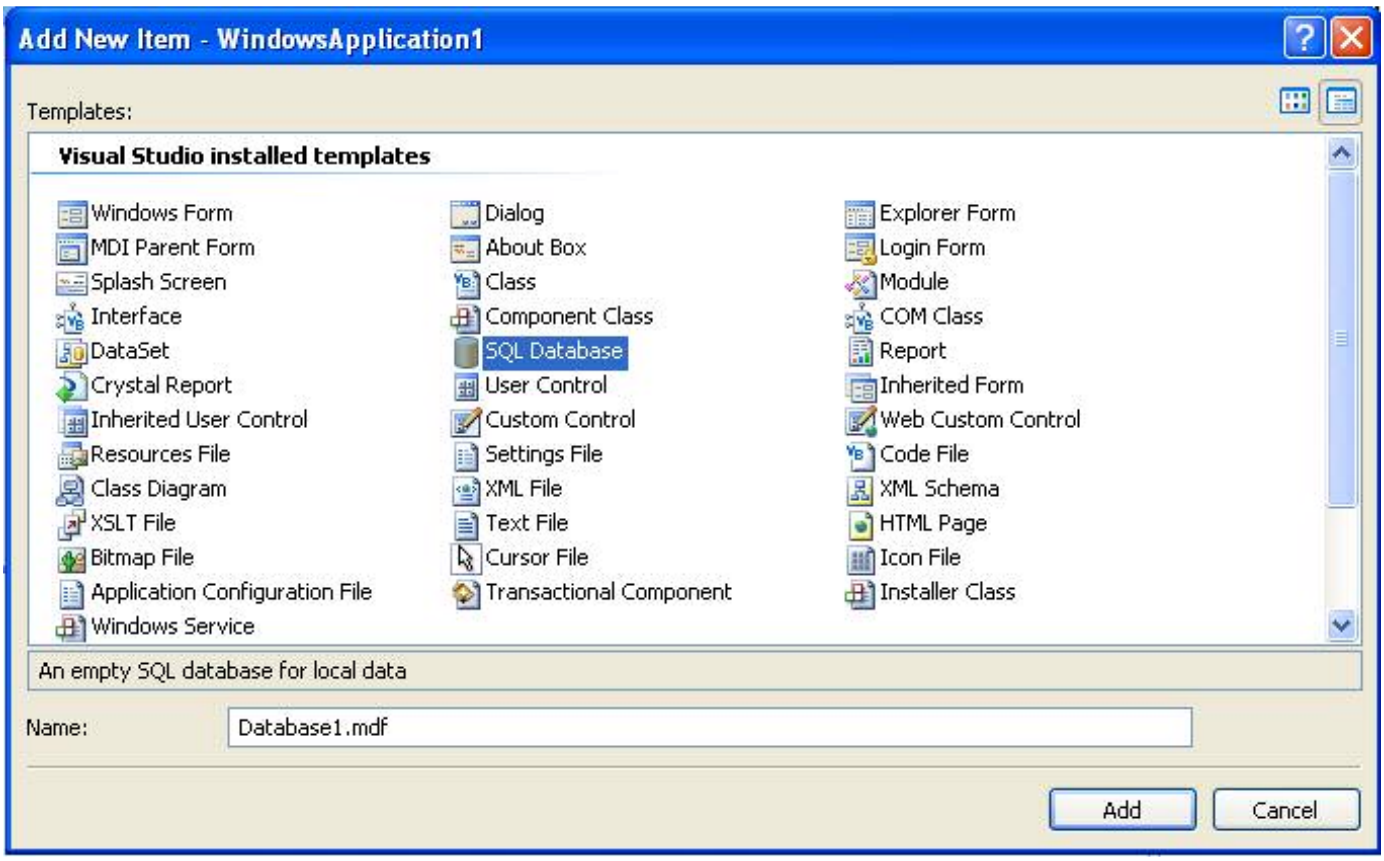
مشروع جديد New Project



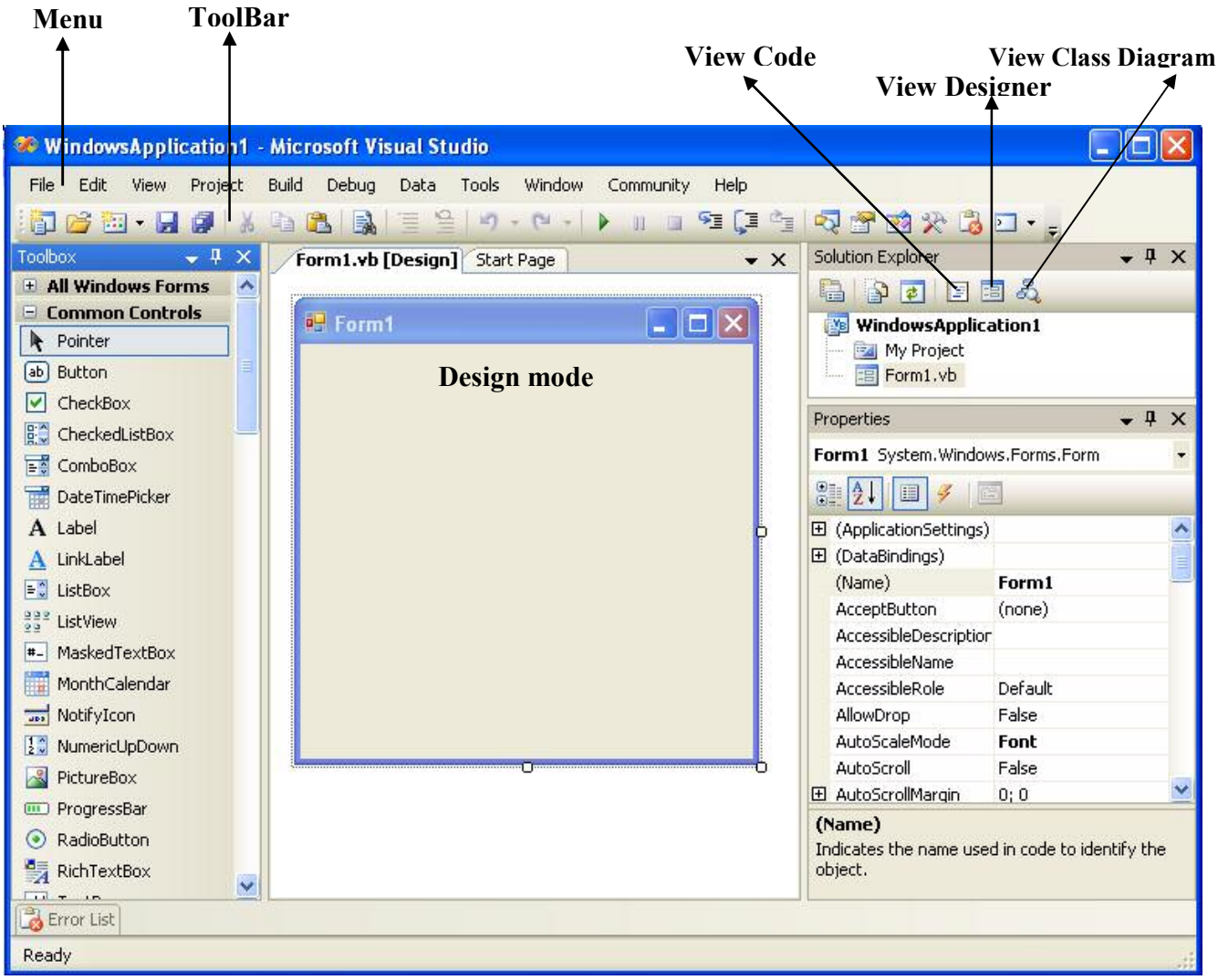
يتيح صندوق الحوار السابق تحديد أنواع المشاريع التي يمكن إنشاءها في بيئة VS.NET ومن أهم هذه المشاريع :-

- تطبيقات Windows.
  - Windows Application.
  - Class Library .
  - Console Application .
  - وغيرها الكثير .
- تطبيقات Web Application (ASP.NET).
- تطبيقات الموبايل و الكمبيوتر المحمول الذكي Smart Device.
- تطبيقات التثبيت Setup .
- تطبيقات قواعد البيانات DataBase .
- وغيرها .

عنصر جديد Add New Item



بيئة التطوير IDE



النافذة الرئيسية هي مصمم النماذج، وسطحه الرماديُّ هو نافذة تطبيقك الجديد في طور التصميم **Design mode**.
ويمكنك بواسطة مصمم النماذج، أن تصمم الواجهة المرئية للتطبيق، حيثُ يمكنكُ أن تضعَ العديد من مكونات واجهة ويندوز **Windows Interface Components** على النموذج، كما يمكنك أن تبرمج التطبيق بكتابة أوامره.

المتغيرات والثوابت Variables And Constants

❖ أولاً المتغيرات Variables:

وإذا كان أساس إتقان اللغات الطبيعية هو تعلم حروف ومفردات تلك اللغة، فان أساس إتقان لغات البرمجة هو تعلم المتغيرات والثوابت التي تبني بها إجراءات برامجك , كما في أيّ لغة برمجة تقوم المتغيّرات بتخزين القيم أثناء تنفيذ البرنامج.. وطبعًا سُمّيت متغيّراتٍ، لأنّك تستطيع تغيير قيمها في أيّ لحظةٍ أثناء تنفيذ البرنامج.

وللمتغيّر اسمٌ وقيمة.. فمثلا: المتغيّر "اسم المستخدم" UserName يمكن أن تُوضع به القيمة "محمد".. والمتغيّر "الخصم" Discount يمكن أن تُوضع به القيمة 0.35.

تلاحُظ هنا أنّ القيمتين "محمد" و 0.35 مختلفتان، فالأولى نص String لهذا تمّ وضعها بينَ علامتَيْ تنصيص، بينما الثانية قيمة رقميّة Numeric Value.

وكما ذكرنا سابقا، فإنّ المتغيّرات في VB.NET ليست مجرّد أسماءٍ أو مخازن للقيم.. إنّها كذلك كِيانات ذكيّة لتخزين وإجراء العمليات على القيم.. باختصار: إنّها كائنات Objects، لها وسائلها وخصائصها الخاصة بها.

فمثلا: هذه الجملة تعرّفُ متغيّرا (كائنا) من النوع "تاريخ" Date:

```
Dim MyDate As Date
```

ويمكنك وضع تاريخ في هذا المتغير بجملةٍ كالتالية:

```
MyDate = #1/1/2003#
```

بل ويمكنك إجراء بعض العمليات على هذا المتغيّر مثل:

```
MyDate.AddYears(3)
```

حيث سيكونُ ناتج هذه العمليّة تاريخٌ جديد، يزيد بثلاث سنواتٍ عن التاريخ الأوّل.. هذا التاريخ الجديد يمكن تخزينه في أيّ متغيّر آخر كالتالي:

```
Dim NewDate As Date
NewDate = MyDate.AddYears(3)
```

لاحظ أنّ المتغير الذي يقع على يسار علامة "="، هو فقط الذي تتغيّر قيمته، بعد إجراء العملية التي تقع على يمينها..

كما أنّ بإمكانك قراءة المتغيّر وتغيير قيمته في جملةٍ واحدة.. مثال:

```
MyDate = MyDate.AddYears(3)
```

حيث يقرأ المترجم التاريخ الموجود في المتغير MyDate، ويضيف عليه ثلاث سنوات، ثم يخزّن التاريخ الجديد في نفس المتغيّر.. بعد تنفيذ هذه العملية، ستصير قيمة المتغير MyDate هي: #1/1/2006#.

الجميلُ في الأمر، هو أنّ معظم الوظائف التي تحتاجها للتعامل مع التواريخ و الأرقام والنصوص، تمنحها لك اللغة جاهزةً، لتريحك من عناء كتابتها من البداية. فإذا احتجت لمعالجة هذه القيم، فاكتب اسم المتغير متبوعا بنقطة "."، وستظهر لك قائمة تعرض لك كل خصائصه، وكل الدوال الجاهزة التي يمكن تطبيقها عليه، وفي معظم الحالات ستجد أسماءها معبّرةً عن وظائفها، بالإضافة لأنّ مجرد تحديد أيّ منها في القائمة (بضغطه مرّة واحدة بالفأرة أو الانزلاق إليه بالأسهم من لوحة المفاتيح) يؤدّي إلى ظهور تلميح على الشاشة، يشرح لك وظيفة هذا العضو.

في معظم لغاتِ البرمجة، يجبُ تعريفُ المتغيّراتِ أولا قبلَ استخدامها.. إنّ هذا يجعل الأمرَ أيسرَ بالنسبة لمترجم الكود Compiler، ففي كلّ مرّة يصادف المترجم متغيّرا، عليه أن يُنشئه في الذاكرة، ونتيجة لاعتباراتٍ في تنظيم الذاكرة، فإن مثل هذه العملية تستهلك

بعض الوقت، ممّا يُبطئ البرنامج.. ولكن لو كان المترجم يعرف كل متغيرات البرنامج وأنواعها سلفا قبل أن يبدأ ترجمة البرنامج، ففي هذه الحالة سيتحسّن الأداء لأقصى درجة.

لقد كانت من أشهر سمات VB، عدم إرغامه للمبرمج على تعريف كلّ المتغيرات.. لقد صارت هذه السمة منتقدةً الآن بشدّةٍ، ليس فقط للأسباب المتعلّقة بسرعة الترجمة وكفاءة الأداء، ولكن أيضا لأن تعريف المتغيّر يُمكّن المترجم من اصطياد أخطاء كثيرةٍ، سواء في وقت التصميم Design Time أو وقت الترجمة Compile Time، بدلا من أن تُفاجئكَ في وقت التشغيل Runtime.

فمثلا: عندما تعرّف متغيرا من النوع "تاريخ" Date، لا يمكن أن يسمح لك المترجم بوضع عدد صحيح Integer فيه.. أيضا لن يسمح لك المترجم باستخدام خاصية "شهر" Month الخاصة بالمتغيرات من النوع "تاريخ" Date، مع متغيّر من النوع "عدد صحيح" Integer.

إن تعريف المتغيّرات يمكّن المترجم من اصطياد مثل هذين النوعين من الأخطاء، أثناء كتابتك للكود، فلا تكاد تترك السطر الذي كتبتها به، حتّى يضع تحته خطًا متعرّجا، لو حلّقت فوقه بالفأرة، فسيظهر لك تلميح على الشاشة يصف الخطأ الذي ارتكبته.

لكل تلك الأسباب، فإنّ المترجم في VB.NET في الوضع التلقائيّ، لن يسمح لك باستخدام المتغيرات بدون تعريف، إلا إذا طلبت أنت ذلك منه صراحة ـ كما سنرى فيما بعد ـ وذلك على عكس الوضع الذي كان في VB6.

مطلوب منك أيضا، تحديد نوع المتغيّر.. وللتسهيل، يمكنك استخدام رموز الأنواع التراثية في لغة البيزيك، مثل $ التي ترمز للنصوص.. فمثلا يمكنك كتابة: Dim Note$

لتعريف المتغيّر Note كمتغيّر نصّيّ.. ولو حلّقت بالفأرة فوق هذا السطر في بيئة التطوير، لظهر لك تلميح على الشاشة مكتوب فيه التالي: Dim Note As String

مّما يعني أنّ الجملتين متكافئتين تمامًا.

الشروط الواجب توافرها عند تعريف المتغيرات:

- ألا يكون كلمة من كلمات اللغة الأساسيّة (تلك التي تراها باللون الأزرق).. مثل Sub وFor و If وغيرها.. إنّ الجملة التالية غير مقبولة:

```
Dim Sub As Integer
```

ولكن لو كنت مصرّا على مثل هذا الأمر، فيمكنك أن تضع الاسم بين قوسين مضلعين [ ].. هذه الجملة مقبولة:

```
Dim [Sub] As Integer
```

لكن عليك في كلّ موضع تستخدم فيه المتغيّر أن تحيطه بالقوسين المضلعين :

```
[Sub] =5
```

- ألا يزيد عن 255 حرفًا، وهو رقم كبير بالفعل بما يكفي.
- أن يتكون من كلمة واحدة لا تتخللها المسافات.. ويمكن استخدام الشرطة المنخفضة "_" للفصل بين مقاطع الكلمة بدلا من المسافات.
- لا يبدأ بأرقام، وإن كان من الممكن أن تتوسطه أرقام، أو ينتهيَ بها.
- لا يحتوي على أيٍ من: علامات التنصيص أو الأقواس أو النقطة "."، ولا علامات العمليات الحسابية أو علامات المقارنة الحسابية أو المنطقية، فكل هذه العلامات محجوزة لوظائف أخرى.
- غير مسموح بتكرار اسم المتغيّر داخل نفس النطاق، فلا يمكن تعريف متغيّرين متماثلين في الاسم داخل نفس الإجراء، وإن كان من الممكن تكرار نفس اسم المتغيّر لكن في إجراءات مختلفة.

والمتغيرات في لغة البيزيك تتجاهل حالة الأحرف Case-insensitive، فالأسماء myAge و myage و MYAGE، كلّها متكافئة، وتشير لنفس المتغيّر.. معنى هذا أنّك لا تستطيع استخدام هذه الكلمات لتعريف ثلاثة متغيرات مختلفة، فكلّها اسما واحدا.

أنواع المتغيرات Types of Variables

الأصل	نوع المتغير	الحجم بالذاكرة بالبايت	القيمة التي يمكن تخزينها
متغيرات صحيحة	Byte	1	0 - 255
	SByte	1	-128 - 127
	Short OR Int16	2	-32768 - 32767
	UShort OR UInt16	2	0 - 65535
	Integer OR Int32	4	-2147483648 - 2147483647
	UInteger OR UInt32	4	0 - 4294967295
	Long OR Int64	8	-9223372036854775808 - 9223372036854775807
	ULong OR UInt64	8	0 - 18446744073709551615
متغيرات كسرية	Single	4	-3.40282e+038f - 3.40282e+038f
	Double	4	-1.79769e+308 - 1.79769e+308
	Decimal	8	-79228162514264337593543950335m - 79228162514264337593543950335m
متغير نصي	String	2 X عدد الحروف	أي سلسلة نصيه من حروف وأرقام ورموز و ...
متغير حرفي	Char	2	أي حرف أو رقم أو رمز أو ...
متغير منطقي	Boolean	1	القيمة true أو false
متغير التاريخ والوقت	DateTime	8	تأريخ أو وقت بكل التنسيقات المختلفة .
كائن	Object	8	أي قيمة من القيم السابقة .

1. المتغيرات الرقميّة Numeric Variables

إنّ للمتغيرات الرقميّة أنواعًا عديدة، تبعًا لحجم العدد ودقته العشريّة.. وعليك أنت أن تحّدد النوع الذي يناسبك، واضعًا في الاعتبار أنّ الأعداد ذات الدقة العشريّة الأكبر، تكون العمليات عليها أبطأ من الأعداد الصحيحة والأعداد ذات الدقة العشريّة الأقلّ.

ملحوظة:

الرمز E في عدد مثل 45– 1.401298 E هو طريقة الكمبيوتر لتمثيل الأس العشري، أي أنّ هذا الرقم يساوي:1.401298×10^{-45}.

العمليّات على الأرقام:

لن تكون هناك فائدة إذا كنت ستضع الأرقام في متغيّرات، دون أن تستطيع أن تُجريَ عليها بعض العمليّات الحسابيّة.. وفي الجدول التالي علامات العمليّات الحسابيّة الأساسيّة:

+	علامة الجمع.
–	علامة الطرح.
*	علامة الضرب.
/	علامة القسمة.. ويمكن أن يكون الناتج عددا صحيحا أو به أرقام عشريّة.. فمثلا: X = 7 / 2 ستعطي الناتج 3.5.
\	علامة القسمة أيضا، ولكنّ الناتج هو العدد الصحيح فقط.. فمثلا : X = 7 \ 2 ستعطي الناتج 3. ويمكن أداء نفس العمليّة باستخدام الدالة Int، لو شئت ألا ترتبك بين علامتي القسمة المتشابهتين، وذلك كالتالي: X = Int (7/2)
Mod	إحدى علامات القسمة أيضا، ولكنّها تعطي الباقي من القسمة فحسب.. فمثلا: X = 7 Mod 2 سيعطي الناتج 1، الذي هو عبارة عن باقي القسمة.
^	الأسّ.. فمثلا $2 \times 2 \times 2$ تُكتب رياضيّا بالصيغة 2^3، وتكتب في البرمجة كالتالي: 2 ^ 3 ولو أردت أن تعبّر عن الجذر التكعيبيّ مثلا، فارفع العدد للأسّ (1 ÷ 3) كالتالي: 2 ^ (1/3)

ويجب أن ألفت انتباهك إلى أهمّيّة وضع الأقواس في العمليّات المتداخلة، وذلك حتّى تضمن صحّة إجراء العمليّة بالترتيب الذي تريدها به.. إنّ الترتيب الطبيعيّ الذي يجرى به VB العمليّات الحسابيّة يسير تبعا للقواعد التالية:

- يتمّ تنفيذ ما بين الأقواس أولا.
- إذا لم تكن هناك أقواس يتمّ تنفيذ الأسس أولا.
- ثمّ يتمّ تنفيذ الضرب والقسمة.
- ثمّ بعد ذلك يتمّ تنفيذ الجمع والطرح.

لهذا فإنّ الصيغة: 8^(1/3)

تعطي الناتج 2، وذلك لأنّ القوس ينفّذ أولا، فتصبح العمليّة هي الجذر التكعيبيّ للعدد 8.. ولكن لو أزلت الأقواس كالتالي:

8^1/3

فإنّ الناتج سيكون 2.666666، وذلك لأنّ الأس ينفّذ أولا، فتصبح العمليّة كالتالي: 8/3

بقي شيءٌ هامّ.. ماذا لو أردت أن نزيد قيمة متغيّر بمقدار 1 مثلا ؟

في هذه الحالة سنقوم بالتالي:

```
X = 5
Y = X + 1
X = Y     ' 6 صارت قيمة المتغيّر
```

حيث اعتمدنا على متغيّر وسيط، جعلنا قيمته هي ناتج جمع المتغيّر الأصليّ مع الواحد، ثمّ نقلنا قيمته إلى المتغيّر الأصلي.

ولكنّ مثل هذه العمليّة تتكرّر مرارا في البرمجة، حيث تحتاج مرارا لزيادة قيم المتغيّرات أو إنقاصها، أو ضربها في رقم... إلخ.

فلو كان على المبرمج أن يكتب هذه الخطوات في كلّ مرّة، لصارت البرمجة جحيما لا يُطاق!

وبالتالي يمكنك أداء هذه العمليّة في سطر واحد مباشرةً كالتالي: X = X + 1

لأوّل وهلةٍ ستبدو لك الصيغة غريبة، ولكن حاول أن تقرأها كالتالي: قيمة X الجديدة تساوي قيمته القديمة + 1.

ولا يشترط أن أجمع على المتغير الرقم 1 فحسب، فهذه العمليّات أيضا مباحة: X = X + 13

```
X = X + X
Y = 5
X = X + Y
```

ولا يقتصر الأمر على الجمع فحسب، بل يمتدّ إلى باقي العمليّات الحسابيّة:

```
X = X – 4     ' 4 إنقاص المتغيّر بمقدار
X = X * 2     ' 2 ضرب المتغيّر في
X = X / 9     ' 9 قسمة المتغيّر على
X = X ^ 3     ' 3 رفع المتغيّر للأس
```

كان هذا هو ما اعتاده مبرمجو VB6.. ولكنّ هناك تسهيلا إضافيّا تقدّمه لك VB.Net، عن طريق استخدام الرموز =+ و =- و =* و =/ و =^ و... والجدول التالي يريك كيفيّة استخدامها:

الطريقة المختصرة المكافئة	الطريقة التقليديّة
X + = 1	X = X + 1
X += Y	X = X + Y
X -= 4	X = X – 4
X *= 2	X = X * 2
X /= 9	X = X / 9
X ^= 3	X = X ^ 3

وأنت حرّ في اختيار الصيغة التي تريحك.

نطاق الأعداد :

والآن، استخدم الكود التالي لاختبار أنواع المتغيرات.. ستجد أنّنا نستخدم خاصيّة "أصغر قيمة" MinValue، و"أكبر قيمة" MaxValue، لعرض نطاق كل نوع منها.

```
MessageBox.Show ("Byte " & Byte.MinValue & "    To     " & Byte.MaxValue)
MessageBox.Show ("SByte " & SByte.MinValue & "  To     " & SByte.MaxValue)
MessageBox.Show ("Short " & Short.MinValue & "  To     " & Short.MaxValue)
MessageBox.Show ("Byte " & Byte.MinValue & "    To     " & Byte.MaxValue)
MessageBox.Show ("int16 " & Int16.MinValue & "  To     " & Int16.MaxValue)
MessageBox.Show ("Integer " & Integer.MinValue & "  To     " & Integer.MaxValue)
MessageBox.Show ("int32 " & Int32.MinValue & "  To     " & Int32.MaxValue)
MessageBox.Show ("Long " & Long.MinValue & "    To     " & Long.MaxValue)
MessageBox.Show ("int64 " & Int64.MinValue & "  To     " & Int64.MaxValue)
MessageBox.Show ("Single " & Single.MinValue & "To     " & Single.MaxValue)
MessageBox.Show ("Double " & Double.MinValue & "To     " & Double.MaxValue)
MessageBox.Show ("Decimal " & Decimal.MinValue & "  To     " & Decimal.MaxValue)
```

ما لا نهاية والقيم الشاذّة الأخرى Infinity And Other Oddities:

يستخدم VB في الحسابات الرقميّة القيمتين "ليس رقمًا" NaN (التي هي اختصار لتعبير Not a Number)، و "ما لا نهاية" Infinity، وذلك لإعلامك بأنّ شيئا ليس على ما يرام قد تمّ في حساباتك، لتتّخذ التصرّف المناسب، بدلا من أن يعرض لك VB رسالة خطإ كما كان يحدث في الماضي.

1. ما لا نهاية Infinity:

إنّ بعض العمليات الحسابيّة ـ كالقسمة على صفر ـ تعطي ما لا نهاية.. ولو استدعيت الدالة "حوّل إلى نصّ" ToString على هذه القيمة، لكان الناتج هو النص "Infinity".. فلْنَرَ هذا المثال:

```
Dim X As Double = 999
MsgBox( X / 0 )
```

سيظهر لك النصّ "Infinity" في الرسالة.

سببٌ آخر للما لا نهاية، هو قسمة عدد كبير جدًّا على عدد صغير جدا جدا، لدرجة تجعل الناتج يتجاوز حدود المتغيّر المزدوج.. جرّب هذا المثال:

```
Dim LargeVar = 10 ^ 999, SmallVar = 10 ^ -999
MsgBox(LargeVar / SmallVar)
```

أيضا سيظهر لك النصّ "Infinity" في الرسالة.

وإذا ما عكست هذه العمليّة، بقسمة العدد الصغير جدا على العدد الكبير جدا، فسيكون الناتج صفرا.. الواقع أنّ الناتج سيكون عددا صغيرا صغيرا جدا جدا، ولكنّ المتغير المزدوج لا يستطيع التعامل مع مثل هذه الأعداد القريبة جدا من الصفر.

2. ليس رقما NaN:

هذه القيمة توضّح أن ناتج عمليّة ما ليس معرّفا: ليس رقما، ولا صفرا، ولا حتّى ما لا نهاية.

فمثلا، ستنتج لك هذه القيمة، لو حاولت أن تحسب اللوغاريتم لعدد سالب، أو لو حاولت أن تقسم صفرا على صفر (0/0)، فهما عمليّتان غير معرّفتان رياضيّا، وليس لهما معني.. جرّب هذا المثال:

```
Dim Var1 As Double = 0, Var2 As Double = 0
MsgBox(Var1 / Var2)
```

ستظهر لك رسالة تعرض النص: "NaN".. لاحظ أنّك لو جعلت قيمة Var2 صغيرة جدّا مثل 1E-299، فإنّ الناتج سيكون صفرا.. ولو جعلت قيمة Var1 صغيرة جدّا، فسيكون الناتج ما لا نهاية.

3. اختبار وجود "ما لا نهاية" و" ليس رقما" Testing For Infinity And NaN:

الأمر بسيط: للتحقق من هاتين القيمتين، استخدم الدالتين: "إنّه ليس رقما" IsNaN و"إنّها ما لا نهاية" IsInfinity، اللتين تجدهما من أعضاء المتغيرات المفردة والمزدوجة , ولمعرفة إشارة الما لا نهاية، استخدم الدالتين: "إنها ما لا نهاية سالبة" IsNegativeInfinity و" إنها ما لا نهاية موجبة" IsPositiveInfinity.
وإليك مثال توضيحيّ:

```
Dim Var1 = 0, Var2 = 0, Result As Double = Var1 / Var2
If Double.IsInfinity(Result) Then
    If Double.IsPositiveInfinity(Result) Then
          MsgBox("هذا رقم كبير جدا.. لا يمكن الاستمرار")
    Else ' ما لا نهاية سالبة
          MsgBox("هذا رقم صغير جدا.. لا يمكن الاستمرار")
    End If
ElseIf Double.IsNaN(Result) Then
     MsgBox("خطأ ليس برقم")
Else 'لم يحدث أي خطأ
     MsgBox("النتيجة هي: " & Result)
End If
```

هذا الكود سيعرض الرسالة "ليس برقم" NaN.. ولكن لو غيّرت قيمة Var1 إلى 1، فستنتج ما لا نهاية موجبة، ولو غيرتها إلى -1، فستنتج ما لا نهاية سالبة.

2. المتغيرات المنطقية Boolean Variables:

المتغيرات المنطقيّة تخزّن واحدةً فقط من القيمتين: "صواب" True و"خطأ" False، وهي في الأساس أعداد صحيحة، فالقيمة "صواب" تعادل -1، والقيمة "خطأ" تعادل صفرا.. وفي الواقع، أي قيمة غير صفرية، تعتبر True.
ويمكنك تعريف المتغيرات من هذا النوع، بجملة كالتالية:

```
Dim B As Boolean
```

ولو لم تحدّد قيمة ابتدائيّة، فستكون قيمة المتغيّر المبدئيّة False .
ولكن كيف نضع القيم في المتغيّر B؟.. إنّ كلّ التعبيرات التالية متاحة:

```
Dim B As Boolean = True
B = False
B = CheckFailure() ' دالة تعيد قيمة منطقيّة
```

ولا يتوقف الأمر عند هذا الحدّ، بل يمكن أن تضع في المتغيّر B نتيجة أيّ عمليّة مقارنة، كالتالي:

```
B = TextBox1.Text = ""
```

أعرف أنّ هذا التعبير قد يكون غريبا عليك.. تعال نعيد كتابته مرّة أخرى مع وضع قوسين، حتّى يسهل علينا فهمه:

```
B = (TextBox1.Text = "")
```

الآن سيفعل VB ما يأتي: سيتحقق أولا من صحّة العلاقة الموجودة بين القوسين، فإذا كانت صحيحة، يضع القيمة True في المتغيّر B ، وإن كانت خاطئة، يضع القيمة False في المتغيّر B .
ولكي تدرك مدى أهمّيّة هذه الصيغة، تعال نكتبها بالطريقة التقليديّة المكافئة لها:

```
If TextBox1.Text = "" Then
    Failure = True
Else
    Failure = False
End If
```

واضح جدّا أنّ الصيغة الأولى تختصر خمسة أسطر في سطر واحد فقط!

3. المتغيّرات النّصّيّة String Variables

يتم تعريفها كالتالي:

```
Dim SomeText As String
```

الآن يصبح بإمكانك أن تضع أيّ نصّ، مهما كانت مكوناته (حروف، رموز، أرقام، علامات تنسيق... إلخ)، ومهما كان طوله (يمكن أن يصل إلى 2 جيجا بايت، أي 2 مليار حرف!).

وهذه الجمل تُريك كيف تستخدم المتغيّر النصّيّ:

```
Dim AString As Strin
AString = "اكتب ما تريد في هذا المتغيّر"
AString = "" ' متغير نصّي فارغ
AString = "ولكن بإمكانك أن تكتب بهذا المتغيّر نصا جديدا أطول من القديم"
AString = "25000"
```

طبعا لاحظت أنّ النصّ يوضع بين علامتي تنصيص.. وهذا بالتأكيد سيدفعك للتساؤل: ماذا لو أردت أن أضع علامة التنصيص نفسها في متغيّر؟

في هذه الحالة يجب أن تكتب أربع علامات تنصيص كالتالي:

```
Dim AString As String = """"
```

وهناك حلّ آخر، هو استخدام الدالة Chr للتعبير عن علامة التنصيص كالتالي:

```
Dim AString As String = Chr(34)
```

سؤال آخر سيراودك: ما الفارق بين التعبيرين التاليين:

```
Dim ANumber As Int16 = 25000
Dim AString As String = "25,000"
```

إن كلا المتغيرين يحمل قيمة مختلفة عن الآخر:

1- **فالمتغير النصّيّ AString يحتوي على ستة حروف، وهي "2" و "5" و "," و "0" و "0" و "0"، بينما العدد الصحيح ANumber يحتوي على رقم واحد هو 25000.**
2- **بما أن كل حرف يتم تخزينه في وحدتي ذاكرة 2 Bytes، فإن المتغيّر النصّيّ يخزن حروفه السّتّة في 12 وحدة ذاكرة، بينما يخزّن العدد الصحيح العدد 25000 في 2 (وحدتان فقط) .**

ولكن برغم هذه الخلافات، فإنّ بإمكانك أن تستخدم المتغيّر النصّيّ AString في العمليات الحسابيّة، وتستخدم المتغيّر الرقميّ ANumber في العمليّات النّصّيّة، حيث يقوم VB بالعمليات اللازمة للتحويل بين النوعين (ما لم تمنعه من القيام بذلك، باستخدام جملة Option Strict On).

ملحوظة : لم يعد بإمكانك تعريف متغيّر نصّيّ ذي طول ثابت Fixed-length String.

دمج النصوص Concatenation:

ماذا لو أردت أن تلحم نصيّن، بحيث ينتج نصّ جديد يجمع الاثنين معا؟

بسيطة.. استخدم علامة الجمع "+" كما اعتدت في استخدامها بين المتغيّرات الرقميّة:

```
Dim X As String = "Saba "
X = X + "University  "
Or
X +="University  "
```

وحتّى لا تشعر النصوص بالغيرة من الأرقام، فإنّ بإمكانك استخدام الصيغة المختصرة التالية:

```
X += " Presentation "    أو    X &= " Presentation  "
```

ولو كنت تخشى من الارتباك في قراءة الكود بين الجمع الحسابيّ وتشبيك النصوص، فاستخدم العلامة & بدلا من العلامة +:

```
Dim X As String = "Saba "
X = X & "University  "
Or
X &="University  "
```

4. المتغيّرات الحرفيّة Character Variables

هي متغيّرات تحجز وحدتا ذاكرة 2 Bytes، يمكن أن تخزّن فيها حرفا واحدا.. ونظرا لأنّ الكمبيوتر لا يعرف شيئا غير الأرقام، وأي شيء يُحفظ به يجب أن يتمّ تمثيله بأرقام تدلّ عليه، فإن الكمبيوتر يمثّل الحروف بأرقام من النوع "عدد قصير بدون إشارة" Unsigned Short Integers (UInt16).. فمثلا الحرف "a" يمثّله الرقم 65.

لهذا فإنّ بإمكانك تحويل الأعداد الصحيحة إلى حروف، باستخدام الدالة ()Chr، وتحويل الحروف إلى أرقام، باستخدام الدالة ()Asc.. جرّب ما يلي:

```
MsgBox(Asc("A"))
```

ستجد أن نافذة المخرجات تعرض الرقم 65.

ويمكنك استخدام جملة كالتالية لتعريف متغيرات حرفيّة:

```
Dim Char1, Char2 As Char
```

ملاحظة:

لا تستطيع مقارنة حرف بعدد مباشرة.. فجملة كالتالية هي جملة مرفوضة:

```
If Char1 = 65 Then Char1 = 97
```

والصحيح أن تستخدم الجملة التالية:

```
If Char1 = Chr(65) Then Char1 = Chr(97)
```

وبإمكانك أن تضع قيمة ابتدائيّة في المتغيّر الحرفيّ عند تعريفه، سواء أكانت حرفا أم نصّا (في الحالة الأخيرة سيتم وضع أول حرف فقط في المتغيّر).. جرّب ما يلي:

```
Dim Char1 As Char = "a", Char2 As Char = "ABC"
MessageBox.Show(Char1)
MessageBox.Show(Char2)
```

ستجد أن شاشة المخرجات ستعرض ما يلي:

a
A

ولكن الطريقة السابقة لن تعمل إذا كان اختيار "التحويل الدقيق" فعالا Option Strict On، لهذا يجب استخدام الطريقة الآتية للتعبير عن الحروف:

```
Dim Char1 As Char = "a"c
```

حيث يخبر الحرف c، VB أنّ ما بين علامتي التنصيص حرف وليس نصا.. وفي هذه الحالة يجب أن يكون ما بينَ علامتي التنصيص حرف واحد بالضبط، حيث لن يقبل VB جملة كالتالية:

```
Dim Char1 As Char = "abc"c
```

إن وضع حرف c بجوار علامتي التنصيص هو طريقة VB لكي يرمز للحروف، ويمكن استخدامها في أيّ موضع كالتالي:

```
Dim Char1 As Char = "a"c
If Char1 = "b"c Then exit sub
```

والمتغيّر الحرفيّ يمتلك بعض الوسائل الشيّقة، مثل "إنّه حرف" IsLetter و"إنّه رقم" IsDigit و"إنّه علامة ترقيم" IsPunctuation.”“ ... إلخ , فلو جرّبت الجملة التالية:

```
MessageBox.Show (Char.IsDigit("2"))
MessageBox.Show (Char.IsPunctuation("'"))
```

لظهر لك في نافذة المخرجات كلمة "True" , جرّب أيضا ما يلي:

```
Dim A As Char = "2"
MessageBox.Show (Char.IsDigit(a))
```

ستحصل على نفس النتيجة.. طبعا يمكنك أن تلاحظ أن الدالة IsDigit هي دالة مشتركة Shared، يمكن استخدامها من المتغير الحرفي، أو من خليّة النوع الأساسيّة Char Class.. بل حتّى يمكن استخدامها من أيّة دالة ناتجها حرف.. جرّب المثال التالي:

```
Dim X As String = "M2f"
MessageBox.Show (Char.IsDigit(X, 1))
```

ستحصل على نفس النتيجة، وهي True.. ولكننا نحتاج لبعض الإيضاح فالدالة "حروف" Char تأخذ رقم الحرف الذي تريد معرفته في النص، وتُرجع لك هذا الحرف.. هذا مع ملاحظة أن الحرف الأول في النص يوجد في الموضع رقم 0.. إلى هنا يمكن التوقف، حيث يمكن استخدام جملة كالتالية:

```
MessageBox.Show (X.Chars(1))
```

حيث سيظهر لك في نافذة المخرجات ما يلي:

2

حيث إنّ "2" هو الحرف الموجود في الموضع 1 (الحرف الثاني) في النصّ.

ولكن بما أنّ الدالة ()Chars تُرجع حرفا، إذن فيمكن تطبيق وسائل الحروف عليها، كالتالي:

```
Console.WriteLine(Char.IsDigit(X.Chars(1)))
```

وتلاحظ أنّنا لم نرسل حرفا لدالة "إنّه رقم" IsDigit كما فعلنا من قبل، ولكن أرسلنا لها المتغيّر النصّيّ X، وموضع الحرف فيه، الذي نريد اختبار كونه رقما أم لا.. ولا بأس في هذا، فالدالة IsDigit لها تعريفان، يختلفان فقط في نوعية المعاملات التي تقبلها الدالة: أحدهما يمكن أن ترسل فيه معاملا واحدا للدالة، هو الحرف الذي تريد اختباره، والآخر ترسل فيه معاملين للدالة: أحد النصوص وموضع الحرف المراد اختباره فيه.

5. المتغيرات الزمنيّة Date Variables:

يتم تخزين متغيّرات التاريخ والوقت بتنسيق خاص كأعداد مزدوجة، بحيث يمثّل العدد الصحيح التاريخ، ويمثّل الجزء العشريّ الزمن.

ويتم تعريف هذا النوع من المتغيّرات والكتابة فيه بطرق كالتالية:

```
Dim MyDate As Date
MyDate = #11/01/2006#
MyDate = #11/01/2006 6:29:11 PM#
MyDate = "Nov 1, 2006"
MyDate = Now() ' دالة "الآن" تعطيك التاريخ والوقت الحاليّين.
```

تلاحظ أنّنا نكتب التاريخ بطريقتين: إما كتاريخ رقميّ مكتوب بين علامتي ##، أو كتاريخ نصّيّ مكتوب بين علامتي تنصيص ""، حيث سيتم تحويله ضمنيّا في الحالة الأخيرة، إلى تنسيق التاريخ والوقت المناسب.

كما أنّ بإمكانك استخدام الدوال الجاهزة التي يمنحها لك VB للتعامل مع التاريخ والوقت.. فمثلا: يمكنك استخدام الدالة "فرق التاريخ" ()DateDIff، لحساب الفرق بين تاريخين مختلفين، وبالوحدة الزمنيّة التي تختارها (سنوات , أشهر , أسبوع, أيام ، ساعات، دقائق , ثواني ...).. إليك هذا المثال، لحساب عدد الأيّام التي عشناها في هذه الألفيّة:

```
Dim Days As Long
 Days = DateDIff(DateInterval.Day, #12/31/2000#, Now())
```

6. المتغيّرات الكائنات Object:

هذا النوع يستطيع تخزين أيّ نوع من البيانات، سواء من الأنواع السابقة، أو من أيّ كائن آخر تعرّفه اللغة، أو تعرّفه أنت.إذن في هذا النوع العامّ يمكن أن نضع فيه أيّ نوع من البيانات دونما قلق! للأسف: لا تبدو الأمور بهذه البساطة، ففعلُ ذلك سيكون له آثار جانبيّة سيّئة، فقبل أن يستخدم VB هذا المتغيّر الكائن، لا بد له أن يعرف نوع البيانات التي به أولا، ليقوم بعمليات التحويل المناسبة، للتعامل مع هذا النوع من البيانات.. فمثلا، لو كان المتغيّر الكائن يحتوي على عدد صحيح، فعلى VB أن يحوّله إلى نصّ قبل لصقه بنصّ آخر.. مثل هذه التحويلات تمثّل عبئا على سرعة البرنامج.. لذا فعليك استخدام المتغيّر الكائن Object في حالات الضرورة فحسب.

```
Dim Var As Object
Var = 5 : MsgBox(Var.GetType.ToString)
Var = 5.1 : MsgBox(Var.GetType.ToString)
Var = "A"c : MsgBox(Var.GetType.ToString)
Var = Now() : MsgBox(Var.GetType.ToString)
Var = "Saba" : MsgBox(Var.GetType.ToString)
```

الخيار "التعريف الصريح"Option Explicit

يمكنك أن تكتب الجملة التالية في بداية أي ملف، قبل تعريف النموذج أو الخليّة أو قالب الكود Module:

```
Option Explicit Off
```

وذلك لإخبار مترجم الكود أنّك ستستخدم المتغيّرات في هذا الملفّ، بدون أن تحتاج لتعريفها أولا.. في هذه الحالة سيفترض المترجم أنّ هذه المتغيّرات يمكنها أن تستوعب أي نوع من أنواع البيانات، لهذا سيعتبرها من النوع "كائن" Object، حيث سيقوم بتعديل نوع كل متغيّر فيما بعد، تبعا لنوع البيانات التي تخزّنها فيه.

ملحوظة: لم يعد بالإمكان استخدام المتغيرات من النوع Variant، وفي المقابل يمكنك استخدام المتغير من النوع Object لتخزين أي نوع من أنواع البيانات والكائنات.

الخيار "التحويل الدقيق" Option Strict:

هذا الخيار مغلق في الوضع التلقائيّ، ليسمح للغة بنقل البيانات بين أنواع مختلفة دون استخدام دوال التحويل، ففي هذه الحالة، تقوم اللغة بإجراء التحويلات المناسبة تلقائيّا.انظر هذا المثال:

```
Dim I As Integer = 10 , S As String = "11"
Msgbox(I + S) ' ستظهر القيمة 21 في شاشة المخرجات (جمع حسابي)
Msgbox (I & S) ' ستظهر القيمة 1011 في شاشة المخرجات (جمع نصّي)
```

ويمكنك أن توقف هذه الخاصيّة، بكتابة الجملة التالية في بداية الملفّ:

```
Option Strict On
```

وبهذا لن يمكنك تنفيذ المثالين السابقين، حيث ستجد أنّ خطوطا متعرجة قد ظهرت ، لتدل على وجود خطإ كالتالي: "خيار التحويل الدقيق يمنع التحويل التلقائي من نص إلى مزدوج" Option strict disallows implicit conversions from String To Double. وهنا يصبح من الضروري إستخدام الجدول أدناه .

ولكن هذا لا يعني أنّ كل أنواع التحويلات الضمنيّة ستصير مستحيلة، فما زال التحويل بين أنواع المتغيرات الرقميّة يتم آليّا، ولكن في اتجاه واحد فقط : حينما تضع المتغير الأصغر في المتغيّر الأكبر، كأن تضع "عددا صحيحا" Integer في "عدد مزدوج" Double.. ولكن العكس ليس مسموحا به، كأن تحاول وضع "عدد مزدوج" Double في "عدد صحيح" Integer، لأنّ هناك احتمالا كبيرا ألا يستوعب العدد الصحيح قيمة العدد المزدوج.

الدالة	القيمة التي تعود بها
CBool	Boolean
CByte	Byte
CChar	Char
CDate	Date
CDbl	Double
CDec	Decimal
CInt	Integer
CLng	Long
CObj	Object
CShort	Short
CSng	Single
CStr	String

الاختصارات المحدّدة للأنواع Data Type Identifiers:

كما ذكرنا من قبل، يمكنك ألا تذكر نوع المتغيّر، إذا لصقت الرمز الخاص به في نهاية اسمه، مثل: Dim MyText$
فعلامة الدولار $ تدل على متغيّر نصّيّ String.. ويمكنك بعد ذلك أن تستخدم المتغيّر بعلامة الدولار $MyText أو بدونها MyText، عند كتابة الكود.والجدول التالي يوضّح الاختصارات المستخدمة مع كل نوع من المتغيرات:

نص String	$
عدد صحيح Integer (Int32)	%
عدد طويل Long (Int64)	&
عدد مفرد Single	!
عدد مزدوج Double	#
عدد عشري Decimal	@

كما في الأمثلة التالية (كلاً على حدى) :

```
Dim Var$="Saba" : MsgBox(Var.GetType.ToString)

Dim Var% : MsgBox(Var.GetType.ToString)

Dim Var! : MsgBox(Var.GetType.ToString)

Dim Var# : MsgBox(Var.GetType.ToString)

Dim Var@ : MsgBox(Var.GetType.ToString)
```

❖ ثانياً الثوابت Constants :

بشكل افتراضي، الثوابت العددية الصحيحة يتعامل معها المترجم على أنها من النوع Integer , والأعداد العشرية من النوع Double

```
MsgBox (10) 'Integer قيمة من النوع
MsgBox (5.5) 'Double قيمة من النوع
```

مع ذلك، يمكنك تحديد نوع الثابت لزيادة سرعة إسناد القيم، فتستطيع استخدام الذيل "L" للنوع Long , الذيل "S" للنوع Short , الذيل "D" للنوع Decimal , والذيل "F" للنوع Single

```
Dim X,Y As Long
X = 100
Y = 100L 'الإسناد التالي أسرع وذلك لعدم إجراء التحويل الواسع
```

فكرة الثوابت المسماة شبيه بفكرة المتغيرات، إلا أن قيم الثوابت المسماة لا يمكن تعديلها وقت التنفيذ، وذلك لأنها تستبدل بقيمها أثناء عملية الترجمة للبرنامج، ويتم حفظها في ملف البرنامج النهائي (كـ EXE مثلاً) استخدم الكلمة المحجوزة Const لتعريف ثابت جديد:

```
Const NAME = "Saba University"
MsgBox (NAME)
```

تحديد نوع الثابت أمر مفضل لزيادة السرعة، بينما يكون إلزامي إن فعلت العبارة Option Strict On

```
Const NAME As String = " Saba University"
```

التركيبات

عبر الزمن ومع الأيام، ستبدأ بتعريف أنواع خاصة بك في برامجك الجدية تعرف بالتركيبات ، والتي دعمتها لغتنا المتميزة VB.Net بقوة .وهنا سأتحدث عن أهم التركيبات وهي النوع Enums و التركيبات من نوع Structures كما سأخصص فقرة كاملة حول المصفوفات Arrays وكذلك سنتحدث عن كل من قائمة المصفوفة ArrayList و المجموعات النوعية Generic .

i. التركيبات من نوع Enums

يمكنك تعريف نوع معين من أنواع المتغيرات التي لم تكن موجودة بالأصل في اللغة وتخدم احتياجاتك الخاصة , ولكن بشرط حصر مجال لهذه القيم التي ستسندها من خلال ما يعرف بالمجاميع المرقمة (Enumeration) , وهنا يمكننا القيام بذلك من خلال استخدم الكلمة المحجوزة Enum لتعريف تركيب جديد إما على مستوى الوحدة البرمجية Module , أو داخل تركيب آخر ولكن من النوع Structure . كما يجب مراعاة بأنه يمكننا كتابة هذه القيم باللغة التي نريدها .

هذا المثال عرف فيه تركيب يمثل أيام الأسبوع :

```
Enum Day
        Saturday
        Sunday
        Monday
        Tuesday
        Wednesday
        Thursday
        Friday
End Enum
```

والآن يمكنك استخدام التركيب السابق وتعريف متغيرات جديدة منه:

```
Dim x As Day = Day.Friday
```

تقنيا، تصنف التركيبات من النوع Enums ضمن الثوابت، فهي كالثوابت المسماة - التي تطرقت لها سابقا - حيث أن قيمها تستبدل أثناء عملية الترجمة .

ثم ترسل إليها المتغيرات من نفس نوع التركيب أو قيم التركيب مباشرة:

```
Dim X As Day = Day.Friday
MessageBox.Show(X)
MessageBox.Show(X.ToString)
```

ii. التركيبات من نوع Structures

يعرف هذا النوع من التركيبات بالأنواع المعرفة من قبل المستخدم User Defined Types– (UDT) , بحيث تمكنك من دمج أنواع مختلفة من المتغيرات وضمها في تركيب أو كتلة واحدة. استخدم الكلمة المحجوزة Structure لتعريف تركيب جديد:

```
Structure Person
    Dim Name As String
    Dim Age As Byte
End Structure
```

ثم تعرف متغيرات جديدة من هذا التركيب وتتعامل معها كالمتغيرات العادية :

```
Dim MS As Person
MS.Name = "Saba"
MS.Age = 99
MsgBox( MS.Name )        ' Saba
MsgBox( MS.Age )         ' 99
```

المزيد أيضا، يمكنك نسخ قيم التركيبات بانسيابية كاملة كما تفعل مع المتغيرات العادية، شريطة أن تكون التركيبات متطابقة :

```
Dim MS2 As Person
MS2 = MS
MsgBox(MS2.Name)         ' Saba
MsgBox(MS2.Age)          ' 99
```

لا تنسى أن التركيبات من نوع Structure يمكن أن تكون متداخلة Nested أي يحتوي بعضها بعضا :

```
Structure Person
    Structure PersonAddress
        Dim City As String
        Dim Country As String
    End Structure
    Dim Name As String
    Dim Age As Byte
    Dim Address As PersonAddress
End Structure
```

الوصول إلى عناصر التركيب المحضون يتم من خلال التركيب الحاضن لها بكل منطقية:

```
Dim ms3 As Person
ms3.Name = "Saba"
ms3.Age = 99
ms3.Address.City = "Taiz"
ms3.Address.Country = "Republic of Yemen"
```

بالإضافة إلى المتغيرات، عليك معرفة أن التركيبات من نوع Structure في VB.Net هي تركيبات مطورة ومرنة جداً جداً (مثل التركيبات الموجودة في C++) , فهي تمكنك من تعريف عناصر إضافية في داخل التركيب كالطرق Methods والخصائص Properties :

```
Structure Person
    Dim Name As String
    Dim Age As Byte
    تعريف طريقة أو إجراء '
    Sub ShowData( )
        MsgBox( Name & "    " & Age )
    End Sub
End Structure
```

مرة أخرى، يمكنك الوصول إلى عناصر التركيب واستدعاء طرقه بنفس الطريقة الانسيابية :

```
Dim MS4 As Person
MS4.Name = "Saba"
MS4.Age = 99
MS4.ShowData( )
```

أن المشيدات Constructers مدعومة بشكل مخفي في التركيبات من النوع Structures حيث أن الإجراء Sub New( ) معرف بشكل تلقائي في التركيب دون أن تراه . ولماذا تم إخفائه؟ وما الفائدة منه؟ الفائدة ببساطة إسناد قيم ابتدائية لمتغيرات التركيب , فلو حاولت إسناد القيم وقت التصريح كما فعلنا سابقا عند التصريح عن المتغيرات :

```
Structure Person
    Dim Name As String = "Saba"
    Dim Age As Byte = 99
End Structure
```

سيظهر لك المترجم رسالة خطأ تفيد بأنك لا تستطيع فعل ذلك وهنا يأتي دور المشيد المخفي ( )Sub New الذي يقوم بإسناد قيم ابتدائية للمتغيرات (0 للمتغيرات العددية، لا شيء للمتغيرات الحرفية، والقيمة Nothing للكائنات) , مع ذلك، يمكنك تعريف مشيد ( )Sub New بنفسك عن طريق تطبيق مبدأ يعرف بإعادة التعريف لعمل ذلك ، أضف وسيطات إضافية مع الإجراء ( )Sub New :

```
Structure Person
    Dim Name As String
    Dim Age As Integer
    Structure PersonAddress
        Dim City As String
        Dim Country As String
    End Structure
    Dim Address As PersonAddress
    Sub ShowData()
        MsgBox(Name & "  " & Age & "  " & Address.City & "  " & Address.Country)
    End Sub
    Sub New ( ByVal PersonName As String, ByVal PersonAge As Byte,
              ByVal PersonCity As String, ByVal PersonCountry As String )
        Name = PersonName
        Age = PersonAge
        Address.City = PersonCity
        Address.Country = PersonCountry
    End Sub
End Structure
```

رغم أن الوظيفة الأساسية للإجراء ( )Sub Newهي العمل كمشيد، إلا انه لن يتم استدعائه بمجرد إنشاء كائن من التركيب فيما لو صرحت عن متغير جديد بالطرق التقليدية، والدليل جرب هذه الأسطر:

```
Dim MS5 As Person   ' هنا لن يتنفذ المشيد
MS5.Name = "Saba"
MS5.Age = 99
MS5.Address.City = "Taiz"
MS5.Address.Countrey = "Republic of Yemen"
Dim MS6 As New Person("Saba", 99, "Aden", "Yemen")   ' هنا يتنفذ المشيد
```

iii. المصفوفات Arrays

يمكنك VB.Net من تعريف المصفوفات سواء كانت أحادية البعد أو متعددة الأبعاد والتي قد تصل إلى 32 بعد (Dimension) :

```
Dim OneDim(9) As Integer    'Items=10
Dim TwoDims(1, 1) As String 'Items= 2 * 2 =4
```

يمكنك فورا البدء بعملية إسناد القيم لها– كما تفعل مع المتغيرات العادية - مع ا لعلم أن بدء الترقيم لفهرس المصفوفات يبدأ بالرقم 0 :

```
OneDim(0) = 100
'........
OneDim(9) = 1000
TwoDims(0, 0) = " I "
TwoDims(0, 1) = " am "
TwoDims(1, 0) = " From "
TwoDims(1, 1) = " Yemen "
```

وان كنت مستعجلاً في عملية إسناد القيم، فإن هذا متاح لك في سطر التصريح مباشرةً، شريطة عدم تحديد عدد عناصر المصفوفة:

```
Dim OneDim() As Integer = {100, 200, 300, 400, 500, 600, 700, 800, 900, 1000}
Dim TwoDims(,) As String = {{" I ", " am "}, {" From ", " Yemen "}}
```

المصفوفات السابقة تسمى مصفوفات ديناميكية Dynamics Arrays لأننا لم نحدد عدد عناصرها، الميزة في هذا النوع من المصفوفات هو إمكانية تغيير حجمها من وقت لأخر باستخدام الكلمة المحجوزة ReDim , مع الإشارة إلى أن عناصر المصفوفة المراد تغيير حجمها باستخدام ReDim سوف تلغى :

```
ReDim OneDim(99)    : ReDim TwoDims(10, 10)
MsgBox(OneDim(0))   ' 0
```

مع ذلك، يمكنك تغيير حجم المصفوفة دون المخاطرة بفقد بياناتها باستخدام الكلمة المحجوزة Preserve

```
ReDim Preserve OneDim(500)
ReDim Preserve TwoDims(1, 500)
```

ولكن لا يمكن تغيير إلا عدد عناصر البعد الأخير فقط وكذلك لا يمكنك تغيير عدد أبعاد المصفوفة الديناميكية سواء استخدمت Preserve أو لم تستخدمها:

```
ReDim Preserve OneDim(500, 500)
ReDim Preserve TwoDims(2, 500)
ReDim TwoDims(100)
```

في المقابل، تستطيع تدمير المصفوفة الديناميكية لتحرير المساحة في الذاكرة في أي وقت تريده باستخدام لأمر Erase :

```
Erase OneDim  : Erase TwoDims
```

والأن كيف ننسخ قيمة مصفوفة إلى أخرى باستخدام الطريقة Clone( ) :

```
Dim Y() As Integer = OneDim.Clone 'Y إلى X نسخ المصفوفة
```

انهي حديثي عن المصفوفات بذكر الدالة UBound( ) التي تعود برقم فهرس العنصر الأخير للمصفوفة، والدالة LBound( ) التي تعود برقم الفهرس للعنصر الأول , ولقراءة قيم المصفوفتين (هنا يجب الإنتباه لأبعاد المصفوفة) :

```
Dim OneDim() As Integer = {100, 200, 300, 400, 500, 600, 700, 800, 900, 1000}
For i As Int16 = LBound(OneDim) To UBound(OneDim)
    MsgBox(OneDim (i))
Next
' يمكن طباعة عناصر أي مصفوفة ومهما كان بعدها بثلاث أسطر فقط
For Each c As String In TwoDims
    Msgbox ( C )
Next
MsgBox("Completed Read TwoDims")
```

يمكن ترتيب عناصر المصفوفة بسهولة كما يلي : Array.Sort(OneDim)

يمكن عكس المصفوفة بسهولة أكثر كما يلي : Array.Reverse(OneDim)

---

iv. ArrayList

لشرح المفهوم بطريقة سريعة تابع الكود التالي على سبيل المثال :

```
Dim myList as New ArrayList ( )
myList.Add(18989)
myList.Add(3455)
```

في الكود السابق ستجد ان الـ myList استخدمت و كأنها Array of Integers و لكن المشكلة الاساسية ان الـ Compiler لن يرفض جملة مثل هذه :

```
myList.Add("Saba University")
```

و هذا لان الـ ArrayList تقبل اي اي نوع و تقوم بعمل Boxing له في نوع Object. هكذا فقدنا ميزة الـ Safe Type . ولهذا فعند استعادة القيم من الـ ArrayList فإن المترجم يقوم بعمل Conversion Type آلياً .

```
For Each c As Object In myList
        Msgbox ( C )
Next
```

v. Generics

إحدى التقنيات الجديدة التي ظهرت في نسخة VB.Net 2005 و انفردت بها هو أن المكتبة BCL (Base Class Library) تحتوي منذ الآن على فضاء اسماء جديد و هو System.Collections.Generic و يحتي هذا الفضاء على عدة كلاسات تمكنك من انشاء مجموعات نوعيه (Generic Collections) . و يقال نوعية Generic لأنه عند التعريف أنت تقوم بتحديد مساحة محددة للكائنات التي تريد و لا تقوم بتحديد نوع . ان الـ Generics تمكننا من كسب الوقت في عملنا فالآن و مع VB.Net 2005 يمكنك عن طريق الـ Generics انشاء Collections خاصة بك في سطر واحد لا أكثر . لم يعد ضروريا الآن ان تنشأ Collection لكل نوع فما عليك الآن سوى استعمال هذه الامكانية الجديدة و تحدد النوع الذي تريد انشاء Collection له.
الآن دعونا من الكلام النظري الذي أظن انه غير مفهوم للكثيرين و لننتقل الى سرد أمثلة تطبيقية حتى نتمكن من فهم هذه التقنية الجديدة. تابع هذا الكود :

```
Dim MyNewList As New List(Of Integer)
MyNewList.Add(1020)
MyNewList.Add(3040)
```

ستجد في الكود السابق اننا استخدمنا الـ Generics في انشاء List من نوع معين اي انها لن تقبل اي متغيرات او قيم من اي نوع اخر اي اننا اذا كتبنا هذا السطر :

```
MyNewList.Add("Saba University")
```

سيعترض الـ Compiler بشدة على هذه الجملة بسبب ان الـ MyNewList لا تقبل Strings أيضا يمكنك استخدام القيم من الـ List مباشرة على انها Integer بدون الحاجة لعمل التحويل (Cast) .
كما ان استخدام الـ Generics غير مقصور على الـ Abstract Types مثل الـ integer و الـ string و الـ float و ... الخ بل انه يمتد الى اي نوع اخر من الـ types اي اننا يمكننا ان نبني List من نوع PersonObject . إنظر التالي :
لنفرض أن لدينا مثلا Class اسمه Person يحتوي على ما يلي :

```
Public Class Person
    Public Name As String
    Public Age As Byte

    Public Sub New(ByVal PersonName As String, ByVal PersonAge As Byte)
        Name = PersonName
        Age = PersonAge
    End Sub
End Class
```

و الآن عندما أنشأنا كلاس مثل هذا يمكننا انشاء لائحة أشخاص بكل سهولة وبسطر واحد فقط و كما يلي :

```
Dim Persons As New System.Collections.Generic.List(Of Person)
```

استطعنا الآن و بكل سهولة انشاء لائحة من الأشخاص و التي تحتوي فقط على نوع Person . إنظر الشفرة التالية :-

```
Dim Persons As New List(Of Person)
Dim Person_1 As New Person("Person_1", 25)
Persons.Add(Person_1)

Dim Person_2 As New Person("Person_2", 30)
Persons.Add(Person_2)

For Each c As Person In Persons
    MessageBox.Show( c.Name & "      " & c.Age )
Next
```

كما تلاحظ ففي السطر الأول قمنا بانشاء شخص جديد و أعطيناه اسم و عمر عن طريق الـ Constructor ثم بعد ذلك أضفناه الى لائحة الأشخاص التي أنشأناها مسبقا و ذلك عن طريق الـ method المسماة Add و بعد ذلك أنشانا شخص جديد و أضفناه الى اللائحة ثم انتقلنا الى عمل حلقة تكرارية حيث أننا نمر على كل عنصر من عناصر اللائحة Persons واذا كان العنصر من نوع Person يعني شخص نقوم بإظهار اسمه و عمرة .

جمل التحكّم في المسار Flow-Control Statements

ما هي جمل التحكّم في المسار ؟:

إنّ البرمجة أعمق من أن تكون مجرّد تعريف متغيّرات.. إنّها تفكير منطقيّ يعتمد على حساب كل الاحتمالات، لاتخاذ الأفعال المناسبة لكل احتمال.. لهذا فلا بد أن توجد طرق نتحكّم بها فيما ينفذ ومتى ينفذ من البرنامج.فالكمبيوتر بدون مثل هذه الطرق سيبدو كما لو كان مجرّد آلة حاسبة ضخمة!

vi. جملة الشرط If...Then

تستطيع أن تختبر حدوث شرط معيّن، فإذا كان صحيحا يتمّ تنفيذ مقطع الشرط، وإن كان خاطئا يقفز التنفيذ إلى جملة نهاية الشرط End If.ولكن ما هو الشرط ؟ الشرط هو أيّ متغيّر منطقي أو أيّ تعبير يعطي نتيجة منطقيّة (True أو False)، مثل:

```
X > 3
Y < 2 And X = 5
Not((X + Y = 5) Or (X - Y < 3))
```

وكما قلنا آنفا، إذا كان مقطع الشرط جملة واحدة، فلا حاجة بنا إلى جملة نهاية الشرط End if كالتالي :

```
If Condition Then Statement
```

ويمكن تنفيذ أكثر من جملة في سطر واحد، بفصلها بنقطتين قائمتين ":" كالتالي:

```
If Condition Then Statement1 : Statement2: Statement3
```

مثال:

```
Dim X As Integer = 3, Y As Integer = 5
If X > Y Then X += 1:  Y -= 2 : X /= Y : Y*= X
```

ويمكن كتابة جملة الشرط السابقة بالطريقة التقليديّة كالتالي:

```
If X > Y Then
   X += 1
   Y -= 2
   X /= Y
   Y*= X
End If
```

وفي حالة التعامل مع متغيّر منطقيّ، يمكن استخدام جملة الشرط كالتالي:

```
Dim Check As Boolean = True
If Check = True Then Check = False
If Check = False Then Exit Sub
```

ولاختصار الكتابة، يعتبر VB أن قيمة المتغيّر المنطقيّ = True إذا لم يذكر ذلك صراحة.. بمعنى أنّ الجملة التالية:

```
If Check Then Check = False
```

مكافئة تماما للجملة:

```
If Check = True Then Check = False
```

جملة الشرط If...Then...Else:

هذه طريقة أخرى لكتابة جمل الشرط، نحتاج إليها إذا أردنا تنفيذ مقطع من الكود لو كان الشرط صحيحا، ومقطع آخر لو كان خاطئا:

```
If Condition Then
      Statement 1
Else
      Statement 2
End If
```

ويمكن التحقق من أكثر من شرط، فإذا لم يتحقق أيّ منها يقفز البرنامج لتنفيذ مقطع "ما عدا ذلك" Else، وذلك تبعا للصيغة التالية:

```
If Condition 1 Then
      Statement 1
ElseIf Condition 2 Then
      Statement 2
Else
      Statement3
End If
```

ولا قيود عليك في إضافة العدد الذي تريده من جمل ElseIf.

مثال:

```
Dim Mark As Integer = 83
If Mark < 50 Then
      MsgBox("راسب")
ElseIf Mark < 65 Then
      MsgBox("مقبول")
ElseIf Mark < 80 Then
      MsgBox("جيد")
ElseIf Mark < 90 Then
      MsgBox("جدا جيد") ' هذا هو المقطع الذي سينفذ
ElseIf Mark <= 100 Then
      MsgBox("ممتاز")
Else
      MsgBox("تأكد من القيمة المدخلة")
End If
```

وتمتاز جملة ElseIf بأنّ VB يختبر الشروط من أعلى لأسفل، وعند نجاح واحد منها لا يختبر باقي الشروط، بل ينفّذ المقطع الخاص بالشرط المتحقق ثمّ يقفز مباشرة لجملة نهاية الشرط.. لهذا فالمثال السابق أفضل من المثال التالي، مع أنّ كليهما يؤدّي نفس الوظيفة:

```
Dim Mark As Integer = 83
If Mark < 50 Then
      MsgBox("راسب")
End If
If Mark < 65 And Mark >= 50 Then
      MsgBox("جيد")
End If
If Mark < 90 And Mark >= 75 Then
      MsgBox("جدا جيد")
End If
If Mark >= 90 Then
      MsgBox(" ممتاز")
End If
```

ولكنّ هذا لا يمنع أنك قد تحتاج إلى مجموعة جمل شرط مستقلّة متتالية، وذلك في الحالات التي لا ترتبط فيها الشروط ببعضها، بحيث يمكن حدوث أكثر من شرط منها معا في نفس الوقت.

وكذلك AndAlso.. أو غير ذلك OrElse:

تعرف طبعا أنّ الشرط يمكن أن يتكوّن من مجموعة شروط جزئيّة تربطها معا المعاملات And أو Or أو Not أو Xor.. في هذه الحالة يتمّ التحقّق من كلّ شرط على حدة أولا، ليتم استبدل كلّ شرط بناتجه (True أو False).. وبهذا تتحوّل الجملة إلى مجموعة قيم True و False تربطها المعاملات المنطقيّة.. وطبعا ناتج ذلك في النهاية هو قيمة واحدة فقط: True أو False.. ولو جرّبت الكود التالي:

```
Dim B As Boolean = True
If B And MsgBox("رسالة اختبار1") = MsgBoxResult.OK Then
      MsgBox("رسالة اختبار2")
End If
```

فستجد أنّ الرسالة الأولى ستظهر، وعندما تضغط موافق، فإنّ الشرط :

MsgBox("رسالة اختبار1") = MsgBoxResult.OK

ستصبح نتيجته True، وبما أنّ قيمة B هي True، فإنّ الشرط يتحوّل بالنسبة لـ VB إلى:

```
(True And True)
```

وطبعا ناتج هذا الشرط هو True، وبالتالي تظهر الرسالة الثانية.

ولكن لو جرّبت الجملة التالية:

```
If B = False And MsgBox("رسالة 1") = MsgBoxResult.OK Then
      MsgBox("رسالة 2")
End If
```

فستجد أنّ الرسالة الأولى ستظهر ولن تظهر الرسالة الثانية.

وستتساءل: لماذا ظهرت الرسالة الأولى، مع أنّ الشرط B = False هو شرط غير متحقّق؟.. إنّ كون الشرط الأوّل في عمليّة And خاطئا، يعنى بالضرورة أنّ ناتج العمليّة سيكون False مهما كان ناتج الشرط الثاني.. فلماذا اختبر VB الشرط الثاني مع أنّه بلا فائدة؟.. أليس هذا تضييعا للوقت؟معك حقّ، ولكن أنت الذي بيده تحديد ذلك، فلربّما كنت تريد عرض الرسالة في كلّ الأحوال!

لذلك لقد ابتكر لك VB.Net معاملا جديدا اسمه "وكذلك" AndAlso، وذلك لاختصار التحقّق من الشروط عديمة الفائدة.. فلو كان الشرط الأوّل خاطئا، فلن يتمّ التحقّق من باقي الشروط.. فمثلا لو جرّبت:

```
If B = False AndAlso MsgBox("رسالة 1") = MsgBoxResult.Cancel Then
      MsgBox("رسالة 2")
End If
```

فلن تظهر لك الرسالة الأولى.

أمّا لو جرّبت:

```
If B AndAlso MsgBox("رسالة 1") = MsgBoxResult.Cancel Then
      MsgBox("رسالة 2")
End If
```

فإنّ كون الشرط الأوّل صحيحا لا يحسم عمليّة And، لهذا فلا بدّ أن يتمّ التحقّق من الشرط الثاني.. لهذا فستظهر لك الرسالة الأولى، ولكن لأنّ الشرط الثاني لن يكون صحيحا (لأنّك ستضغط زر Ok بينما الشرط يتحقّق من أنّك تضغط زر Cancel)، فإنّ الرسالة الثانية لن تظهر.

ولتدرك أهمّيّة هذا المعامل الجديد، انظر مثلا كيف يكتب مبرمج VB6 كودا يتأكّد من أنّ الجذر التربيعيّ للعدد X أكبر من 2.. في هذه الحالة يجب أن تتحقّق أولا من أنّ X ليس عددا سالبا:

```
If X >= 0 Then
      If Sqrt(X) > 2 Then MsgBox("أكبر من 2")
End If
```

ولو حاول مبرمج VB6 أن يكتب ذلك كالتالي :

```
If X >= 0 And Sqrt(X) > 2 Then MsgBox("أكبر من 2")
```

فسيعرّض برنامجه للانهيار، لأنّ خطأ سيحدث لو كان العدد X سالبا، لأنّ جملة الجذر التربيعيّ ستنفذ في كلّ الأحوال، وبهذا يبدو الشرط الأوّل عديم الفائدة!

ولكن الآن تستطيع كمبرمج VB.Net أن تكتب:

```
If X >= 0 AndAlso Math.Sqrt(X) > 2 Then MsgBox("أكبر من 2")
```

واثقا تمام الثقة أنّ أيّ خطإ لن يحدث، وذلك لأنّ كون X عددا سالبا سيعيق التحقّق من الشرط الثاني.. هذه هي عبقرّية المعامل AndAlso.

ولديك موقف مشابه في حالة Or، فكون الشرط الأوّل في عمليّة Or صحيحا، يعنى بالضرورة أنّ ناتج العمليّة سيكون True مهما كان ناتج الشرط الثاني، لهذا لو أردت اختصار الوقت وإهمال الشرط الثاني في هذه الحالة، فاستخدم المعامل "أو غير ذلك" OrElse.. بهذه الطريقة ستضمن عدم اختبار أيّ شرط إلا إذا كان الشرط السابق له خاطئا.. جرّب استبدال Or و OrElse بـ And و AndAlso في الأمثلة السابقة.

الخلاصة:

استخدم And و Or إذا أردت إجبار VB على اختبار كلّ الشروط، وذلك إذا كان بعض هذه الشروط يحتوي على دوال تريد استدعاءها دائما مهما كانت نتيجتها.
أمّا إذا أردت تسريع التنفيذ وتسهيل كتابة الكود، فاستخدم AndAlso و OrElse لتجنّب اختبار الشروط التي لا داعيَ لاختبارها.

vii. جملة اختيار الحالة Select Case:

هذه حالة خاصة من جملة الشرط، لتسهيل كتابة الكود عندما تكون كل المقارنات بين متغير واحد، ومجموعة من القيم من نفس نوع المتغيّر.. ويوضع اسم المتغيّر بعد جملة Select Case، بينما توضع القيم المختلفة واحدة بواحدة بعد كلمة Case، متبوعة بالمقطع الذي سيتم تنفيذه في حالة تحقق التساوي:

```
Select Case Var
      Case Value 1
            Statement 1 (يتم تنفيذه حينما يساوي التعبير القيمة1)
      Case Value 2
            Statement 2 (يتم تنفيذه حينما يساوي التعبير القيمة2)
      ……….
      Case Else
            Statement (يتم تنفيذه حينما لا يساوي التعبير أي قيمة مما سبق)
End Select
```

لماذا استخدمنا كلمة تعبير وليس كلمة متغيّر؟.. ببساطة لأنّك تستطيع اختبار قيمة المتغيّر X، أو اختبار قيمة التعبير X * 2 كذلك.. وهكذا.وإليك هذا المثال:

```
Select Case Now.DayOfWeek
      Case DayOfWeek.Monday
            MsgBox("أسبوعاً ممتعاً")
      Case DayOfWeek.Friday
            MsgBox("عطلة ممتعة")
      Case Else
            MsgBox("مرحباً بعودتك")
End Select
```

وفي هذا المثال نختبر في أيّ يوم من الأسبوع نحن، باستخدام الخاصّيّة "يوم الأسبوع" DayOfWeek الخاصّة بالمتغيّرات من النوع تاريخ.. ولكن أين هو ذلك المتغيّر من النوع تاريخ؟.. ليس موجودا مباشرة، ولكن يمكن افتراض وجوده، لأنّ الدالة Now ترجع قيمة من نوع هذا المتغيّر، ولقد ذكرنا سابقا أنّ القيم المعادة يمكن استخدام أعضاء نوعها.

بعد ذلك، نختبر القيم التي تعنينا في كل "حالة" Case.. وتلاحظ أنّنا نستخدم تعبيرات رقميّة لأيّام الأسبوع بدلا من الأرقام، مع أنّ الأرقام يمكن استخدامها، ولكنّ هذه الطريقة ستكون أسهل في الفهم عند قراءتها.

وكلّ ما سيحدث، هو أن VB سيختبر الحالات حالة حالة من أعلى إلى أسفل، وعند تحقّق إحداها سينفّذ المقطع الخاصّ بها، ثم سيقفز لجملة نهاية الاختيار End Select، دون أن يكمل اختبار باقي الحالات.

أمّا لو لم تتحقّق أيّ حالة، فسينفّذ VB مقطع "حالة أخرى" Case Else (إن كان موجودا، فبإمكانك ألا تكتبه).

ولكن هل أنا مضطرّ لاختبار قيمة واحدة في كلّ حالة؟

لا بالطبع، فبإمكاننا وضع أكثر من قيمة في الحالة الواحدة، بحيث يكفي حدوث إحداها ليتمّ تنفيذ مقطع حالته.. انظر للمثال التالي:

```
Select Case myDay.Saturday
      Case 0
            MsgBox("أسبوعاً ممتعاً")
      Case 1, 2, 3, 4, 5
            MsgBox("مرحباً بعودتك")
      Case 6
            MsgBox("عطلة ممتعة")
End Select
```

والآن قارن جملة اختيار الحالة مع جملة الشرط التالية:

```
Dim X As myDay = myDay.Fri
If X = 0 Then
      MsgBox("أسبوعاً ممتعاً")
Else
      If X >= 1 And X <= 5 Then
            MsgBox("مرحباً بعودتك")
      Else
            MsgBox("عطلة ممتعة")
      End If
End If
```

أعتقد أنّك تشاركني الرأي، في أنّ جملة اختيار الحالة أسهل بكثير.

تراكيب التكرار Loop Structures

إنّ قدرة الكمبيوتر على تكرار أي جزء من الكود ـ خاصّة مع سرعته الفائقة ـ هي ما تجعله مريحا جدا للبشر، ليحمل عنهم عناء البطء والملل. ويقدّم لك VB هذه التركيبات التكراريّة:

1. جملة التكرار "من إلى" For...Next:

هذا هو التركيب التكراريّ الوحيد الذي يتطلّب أن تعرف عدد المرّات التي سيتمّ تنفيذه فيها، حيث يعتمد على وجود عداد للتكرار Counter، تحدّد له قيمة البداية، وقيمة النهاية، والخطوة Step التي سيقفزها (القيمة التي سيزيد أو ينقص بها) كلّما وصل التنفيذ لكلمة "التالي" Next:

```
For Var = initial Value To Stop Value [Step Increase Or Decrease]
      Statement
Next Var
```

ملاحظة : في كتب البرمجة، وجود أي جزء من الصيغة بين قوسين مضلعين [ ]، معناه أنّ هذا الجزء اختياريّ Optional، لستَ مجبرا على كتابته في الكود.

وهذه هي الطريقة التي يُنفّذ بها VB هذا التركيب:

1- توضع قيمة البداية في المتغيّر الذي يعمل كعدّاد للتكرار.
2- يتم اختبار إذا ما كان العدّاد أكبر من قيمة النهاية أم لا، فإذا كان أكبر، يخرج التنفيذ من جملة التكرار إلى السطر التالي لجملة Next.. ومن الممكن أن تكون قيمة الزيادة سالبة، في هذا الحالة يجب أن تكون قيمة النهاية أصغر من قيمة البداية.. وسيتأكّد VB ممّا إذا كان العداد أصغر من قيمة النهاية أم لا.. تذكّر أنّ -3 أصغر من -2.
3- يتم تنفيذ الجمل الموجودة في مقطع التكرار.
4- عند وصول التنفيذ إلى جملة Next، يتم جمع قيمة الزيادة على العدّاد.. وطبعا لو كانت قيمة الزيادة سالبة، فستقلّ قيمة العداد.. ولو لم تكن قيمة الزيادة مذكورة، فسيتم اعتبارها 1.
5- يتم تكرار الخطوات 2، 3، 4، حيث يتم تنفيذ مقطع التكرار في كلّ دورة، حتّى يتجاوز العدّاد قيمة النهاية، فيخرج التنفيذ من مقطع التكرار.

مثال: هذا البرنامج سيعرض لك 10 رسائل متتالية، كل منها تحمل رقمها (من 1 إلى 10) :

```
Dim I As Integer
For I = 1 To 10
    MsgBox(I)
Next I
```

أو يتم تعريف المتغير ضمن الحلقة نفسها كما يلي :

```
For I As Byte = 1 To 10
     MsgBox (I)
Next I
```

ملاحظة : رغم أنّ ذكر اسم المتغيّر بعد كلمة Next ليس إجباريّا، إلا إنّني أنصحك به، لأنّه سيسهّل عليك قراءة الكود، خاصّة في الحالات التي تتداخل فيها أكثر من جملة تكرار.

ويمكن الخروج في الحال من جملة التكرار "من إلى" باستخدام جملة Exit For، ولو جرّبت المثال التالي، فستجد أنّ الرسالة ستظهر لك خمس مرات فقط بدلا من 10 :

```
For I As Byte = 1 To 10
    MsgBox(I)
    If I = 5 Then Exit For
Next I
```

أهمّ شيء يجب أن تعرفه، هو أنّ معاملات التكرار (البداية والنهاية والخطوة) يتم قراءتها مرة واحدة في بداية تنفيذ تركيب التكرار، لهذا فإنّ تغيير قيم هذه المعاملات في مقطع التكرار نفسه، لن يكون له أي تأثير.
ولو أعدت كتابة المثال الأوّل بالطريقة التالية، فلن يتغيّر شيء في تنفيذ البرنامج:

```
Dim I As Integer, X As Integer = 10
For I = 0 To X
    MsgBox(I)
    X = 5 ' لن تؤثّر هذه الجملة في قيمة نهاية جملة التكرار
Next I
```

إنّ هذا يسمح لك بجعل قيمة النهاية تعبيرا مبنيا على العدّاد نفسه، دون أن تخشى من أن يؤثّر تغيّر العدّاد عليها.. فمثلا، الجملة التكراريّة التالية ستعرض 10 رسائل :

```
Dim I As Integer = 20
For I = 1 To I / 2
     MsgBox(I)
Next I
```

هنا يجب أن أحذرك من هذا الخطأ الشائع الذي قد يؤدي لانطلاق استثناء: ذلك هو اعتماد شرط التوقف على طول متغير نصي، ثمّ تغيير النص في ثنايا جملة التكرار ليصبح طوله أقصر!.. في هذه الحالة سيستمر التكرار اعتمادا على الطول القديم للنص.. انظر لهذه الجملة:

```
Dim I As Integer, S As String= "12345"
For I = 0 To S.Length-1
     MsgBox(S.Chars(I))
     S= "1234"
Next I
```

في البداية يكون طول النص 5 أحرف.. لهذا تكون الجملة التكرارية من 0 إلى 4.. ولكن بعد تغيير طول النص إلى 4 حروف فقط لا يتأثر شرط التوقف بذلك.. لهذا يزداد العداد إلى أن يصل إلى 4.. هنا ينطلق استثناء، نتيجة أنه لا يوجد حرف في النص في الموضع رقم 4!!
لهذا فانتبه جيدا لمثل هذه الحالات.. جملة For في VB لا تصلح إلا مع الأوامر التي تُنفذ لعدد مرات محدد مقدما وغير قابل للتغيير أثناء التنفيذ

بينما المقطع التالي لن ينتهي أبدا، لأنّ العدّاد ينقص 1 قبل أن تتم زيادة قيمته في جملة Next، ممّا يعمل على ثبات قيمته إلى الأبد:

```
For  I As Integer = 1 To 10
     MsgBox(I)
     I -= 1
Next I
```

ولإيقاف مثل هذا البرنامج العالق، انتقل إلى نافذة لغة البرمجة، واضغط Ctrl+Pause أو Ctrl+Break (على حسب ما هو مكتوب على هذا الزر في لوحة المفاتيح).أعتقد أنّ هذا المثال سيردعك عن استخدام مثل هذه الطريقة الخطيرة للتلاعب في عدد مرات تكرار المقطع.ولكنّك أحيانا لا تعلم بالضبط عدد المرات التي تريد تكرار المقطع بها.. فماذا تفعل؟

2. تركيب التكرار "نفّذ مرارا" Do...Loop:

هذا التركيب أيضا ينفّذ مقطعا من الكود، ولكنّه أعمّ من متكرّرة For... Next، لأنّه يعتمد على صحّة شرط ما لاستمرار التكرار، بدلا من الاعتماد على قيمة العدّاد.. إنّ هذا يُمكّنك من تعقيد هذا الشرط لأيّ درجة. وهناك صيغ عديدة لهذا التركيب:

1- الصيغة اللانهائيّة:

```
Do While Condition
      Statement
Loop
```

مثال :

```
Dim I As Integer
Do While I <= 10
      MsgBox(I)
Loop
```

وهي لا نهائيّة، لأنّ تركيب التكرار لا يحتوي على أي شرط، وما لم يتم الخروج من هذا المقطع بجملة Exit Do، فهو لن ينتهي أبدا.

2- صيغة "نفّذ بينما":

```
Do While Condition
      Statement
      مقدار الزيادة أو النقصان'
Loop
```

مثال :

```
Dim I As Integer
Do While I <= 10
      MsgBox(I)
      I = I + 1
Loop
```

وفي هذه الصيغة، يتم تنفيذ المقطع بينما يكون الشرط صحيحا، فإذا صار الشرط خطأ توقف التنفيذ.. ويتم اختبار الشرط في بداية تنفيذ المقطع، ثمّ في كل مرّة يتم فيها تنفيذ كلمة التكرار Loop.وهناك صيغة فرعيّة من هذه الصيغة:

```
Do
       Statement
Loop While Condition
```

وتمتاز هذه الصيغة بأنّها تُنفّذ المقطع مرّة أولا، قبل أن تتحقّق من صحّة الشرط، وبهذا تضمن تنفيذ المقطع مرّة واحدةً على الأقلّ. كما في مثالنا التالي :

```
Dim I As Integer = 11
Do
      MsgBox(I)
      I = I + 1
Loop While I <= 10
```

3- صيغة "نفّذ حتّى":

```
Do Until Condition
      Statement
Loop
```

مثال :

```
Dim I As Integer
Do Until I > 10
      MsgBox(I)
      I = I + 1
Loop
```

وهي عكس الصيغة السابقة، فالتنفيذ سيستمرّ طالما أنّ الشرط خطأ، فإذا تحقّق توقف التنفيذ.. وهناك صيغة فرعيّة هذه:

```
Do
      Statement
Loop Until Condition
```

3. تركيب التكرار "بينما" While :

وهو مماثل تماما لتركيب" نفّذ بينما" Do While، مع اختلاف الكلمات فقط، لدرجة أنّك تتساءل: ما هي الحكمة في إضافة هذا التركيب لـ VB؟ ويمكن اصطناع متكرّرة لا نهائيّة عن طريق صيغة كالتالية:

```
Do While Condition
      Statement
Loop
```

مثال :

```
Dim I As Integer
While I <= 10
      MsgBox(I)
End While
```

4. الانتقال الحرّ عن طريق "اذهب إلى" GoTo:

من أدوات التحكّم في مسار تنفيذ البرنامج أيضا، جملة "اذهب إلى" GoTo، حيث تمنحها اسم لافتة وضعتها في مكان ما من الإجراء الحاليّ، لتنتقل إليها:

مثال (1) :

```
      MsgBox("1")
      GoTo Line1
      MsgBox("2")
Line1:
      MsgBox("3")
```

وفي هذا المثال، سيتم عرض الرسالة "1" ثم الرسالة "3"، لأنّ جملة GoTo تجعل التنفيذ يقفز للسطر التالي للافتة المسمّاة Line1، وبهذا لن يتم عرض الرسالة "2" أبدا.

واللافتة هي أي اسم مقبول (يسير على قواعد تعريف المتغيرات)، تكتبه في بداية سطرٍ وتتبعه بنقطتين متعامدتين ":".

ويمكن استخدام GoTo شرطيّا، كالتالي:

مثال (2) :

```
      Dim X As Integer = 3
      MsgBox("1")
      If X > 2 Then GoTo Line1
      MsgBox("2")
Line1:
      MsgBox("3")
```

مثال (3) [لفهم حقيقة هذه الجملة إمعن النظر إلى المثال التالي]:

```
Dim i As Byte = 0
MS:
        i += 1
        MsgBox(i)
   If i <> 5 Then GoTo MS
```

ولكن لا يمكن استخدام GoTo للانتقال من إجراء إلى إجراء آخر.

5. جملة For Each

نختم موضوع تراكيب التكرار بهذه الجملة الأكثر من رائعة لمدى سهولة التعامل معها وفي نفس الوقت تقليل الجهد والتعب إذا استفدنا من الوظيفة التي تقدمها لنا هذه الجملة , تصور أن في أحد نماذج برامجك وجدت بأن لديك عدد كبيرة من أدوات مربع النص (إفترض 100 أداة) وفجأة وجدت نفسك بحاجة لكتابة الكود الذي يقوم بمسح كافة محتويات هذه المربعات (يالايايالهول ماذا سنفعل !!!) لكمية الكود المكرر الذي سنقوم بتكرارة 100 مرة TextBox1.Text="" وهكذا مع بقية الأدوات . إذاً ما هو الحل لمثل هذه الأساة ؟ الحل من خلال جملتنا For Each لا حظ الشفرة المقابلة لأي عدد من أدوات مربع النص كما يلي :

```
For Each C As Control In Me.Controls
      If TypeOf myControl Is TextBox Then myControl.Text = ""
Next
```

تفادي الاستثناءات Catching Exceptions

من التراكيب الجديدة والمهمة في VB.NET ظهور التركيبة Try...Catch...[Finally]...End Try والمتخصصة بتصيد الأخطاء الاستثنائية والتي يتوقع حدوثها أثناء تنفيذ البرنامج ولكي نتعرف أكثر على هذه التركيبة فدعونا نتعرف على الصيغة العامة لها وهي على النحو التالي :

```
Try
      'الشفرة المتوقع حدوث إستثناء بها
Catch
      'الشفرة التي ستنفذ إن وقع الإستثناء
Finally
      'سيتم تنفيذ هذه الشفرة دائماً
End Try
```

ولكي نفهم هذه التركيبة جيداً إنظر إلى المثال التالي والمزود بالشرح الكافي :

```
Try
      Dim X As Short = 1000 \ InputBox("يرجى إدخال قيمة المقام")
      ' لما كان ناتج العملية يعتمد إعتماد كلي على القيمة التي سيدخلها
      ' المستخدم وبالتالي هنالك نسبة كبيرة لحدوث خطأ , فماذا لو قام المستخدم
      ' بإدخال حرف أو قام بإدخال القيمة صفر , لذا بالتأكيد سيظهر خطأ
      ' لذا ومع مثل هذه الحالة يتم وضع الكود الذي نشك بأنه سيتسبب بحدوث خطأ
      ' بعد الكلمة Try
Catch ex As DivideByZeroException  ' إذا حدث خطأ القسمة على الصفر
      MsgBox("Sorry,You Cann't Divided By Zero")
Catch ex As OverflowException  ' إذا حدث خطأ الإفاضة
      MsgBox("The Result is too Larg")
Catch ex As Exception  ' من حسن الحظ فهذا كلاس عام يحوي كل الأخطاء الإستثنائية
      MsgBox(ex.Message)
Finally  ' سيتم تنفيذ هذا المقطع سواءاً حدث خطأ أو لم يحدث وهذا المقطع إختياري
      MsgBox("Processing Completed")
End Try
```

ومن المثال السابق المفصل لابد أن تكون صيغة الـ Try...Catch...End Try قد اتضحت وقد تبين وضيفة كل جزء فيها , لذا نحاول من الأن فصاعداً استخدام هذه التركيبة لضمان سلامة أي برنامج في وقت التنفيذ .

النماذج Forms

أجزاء النموذج:

يعتبر شريط العنوان من أهم أجزاء النموذج، حيث يظهر عنوان النموذج، وحيث تظهر أزرار التحكم (إغلاق **Close**، تكبير **Maximize**، تصغير **Minimize**)، وحيث تظهر كذلك أيقونة النموذج، والتي عند ضغطها بالفأرة، أو ضغط زر **Alt** ثم مسطرة المسافات من لوحة المفاتيح، تظهر قائمة التحكّم **Control Menu**، والتي تحتوي على هذه الأوامر:

أوامر قائمة التحكّم	وظيفتها
استعادة Restore	يستعيد هذا الأمر حجم النموذج الأصليّ، حينما يكون في أقصى حجم له Maximized.
تحريك Move	يسمح للمستخدم بتحريك النموذج على الشاشة باستخدام لوحة المفاتيح، حيث سيظهر رسم يمثّل أربعة أسهم متعامدة في منتصف شريط العنوان، ليخبرك أنّ ضغط الأسهم من لوحة المفاتيح في هذه اللحظة سيحرّك النموذج.. لإنهاء التحريك اضغط زر الإدخال Enter ليستقر النموذج في الموضع الجديد، أو زر الإلغاء Esc ليبقى النموذج في موضعه القديم.
تغيير الحجم Size	يشبه الأمر السابق، إلا إنّ ضغط الأسهم من لوحة المفاتيح يؤدّي لتغيير حجم النموذج.
تكبيرMaximize	يجعل النموذج يصل لأقصى حجم له ليملأ الشاشة.
تصغير Minimize	يجعل النموذج يختفي من على الشاشة، ليصير مجرّد أيقونة على شريط مهام الويندوز Task Bar، (ذلك الموجود في الجزء السفليّ من الشاشة).. ولاستعادة النموذج مرة أخرى، اضغط على هذه الأيقونة في شريط المهام بالفأرة، أو اضغط Alt+Tab من لوحة المفاتيح ولا ترفع إصبعك عن زر Alt.. سيظهر لك شريط عليه أيقونات كل النماذج المفتوحة حاليا في الويندوز، اضغط زر Tab أكثر من مرّة دون أن ترفع إصبعك عن زر Alt أبدا، حتّى تجد أن المستطيل الأسود قد انتقل ليحيط بأيقونة نموذجك.. عند هذا اترك الزّرين، وسيظهر لك نموذجك مرة أخرى.
إغلاق Close	يغلق النموذج نهائيّا.

وسنتعرّف الآنَ على الخصائص التي تمنحها لنا لغة البرمجة للتحكّم في هذه العناصر، لتصميم النموذج بالمظهر المناسب.

أهمّ خصائص النموذج في وضع التصميم:

لقد رأيت بالفعل كل أنواع النماذج أثناء تعاملك مع الويندوز.. فمثلا، أشرطة الأدوات الحرّة Floating Toolbars ما هي إلا نماذج ذات أشرطة عناوين ضيّقة.. كذلك فإنّ مربعات الحوار التي تعرض رسائل التحذير أو تطلب منك اختيار ملف ما، هي أيضا نماذج.. ولكن لماذا تبدو هذه النماذج مختلفة في الشكل؟.. إنّ هذا يرجع للعديد من خصائص النموذج، هي التي سنتعرف لها الآن:

1. زرا الموافقة والإلغاء AcceptButton, CancelButton:

زر الموافقة AcceptButton هو ذلك الزر الذي يتم ضغطه تلقائيًّا عند ضغط زر الإدخال Enter، بغض النظر عن الأداة النشيطة في تلك اللحظة.. بالمثل، فإنّ زر الإلغاء CancelButton، هو ذلك الزر الذي يتم ضغطه تلقائيًّا عند ضغط زر الإلغاء Esc.. .

2. تحجيم تلقائي AutoSize:

لو جعلت قيمة هذه الخاصّيّة "صواب" True، فسيتغير حجم النموذج تلقائيّا ليتناسب مع الأدوات الموجودة بداخل هذا النموذج.. والقيمة الافتراضيّة لهذه الخاصّيّة "False".. .

3. انزلاق تلقائي AutoScroll:

لو اخترت لهذه الخاصّية القيمة صواب، فستظهر أحد أو كلا شريطي الانزلاق Scroll Bars تلقائيّا، في حالة تغيير حجم النموذج للدرجة التي تختفي فيها بعض أدواته. وبفضل هذه الخاصّيّة يمكنك أن تصمّم نماذج ضخمة مزدحمة بالأدوات، دون أن تقلق بخصوص حجم الشاشة ودقتها.

وتنزلق هذه المنزلقات تلقائيا لتعرض الأداة النشيطة حاليا، وكلما انتقل المستخدم لأداة أخرى بضغط زر Tab، يتم عرضها آليا.

4. هامش الانزلاق التلقائي AutoScrollMargin:

هذا هو الهامش الذي يُترك حول الأدوات الموجودة على النموذج، بحيث لو صار النموذج أصغر من المستطيل الذي تشكّله الهوامش، تظهر المنزلقات.

وهذه الخاصّيّة هي كائن، يحتوي على خاصيتين فرعيتين: العرض والارتفاع، حيث يتوجّب عليك وضع قيمتيهما.

ولفعل ذلك من الكود، استخدم جملتين كالتاليتين:

```
Me.AutoScrollMargin.Width = 40
Me.AutoScrollMargin.Height = 40
```

5. طراز الإطار FormBorderStyle:

تتحكّم هذه الخاصية في مظهر النموذج، وهي تأخذ واحدة من القيم الموضحة في الجدول التالي:

القيمة	معناها
بدون None	سيظهر النموذج بدون إطار وبدون شريط عنوان، بحيث لا يمكن تغيير حجمه أو تحريكه.. حاول أن تتجنب هذه القيمة.
متغير الحجم Sizable	وهي القيمة الافتراضية، حيث يتم عرض النموذج التقليدي، القابل للتحريك وتغيير الحجم.
مثبت مجسم Fixed3D	نموذج بإطار مرئي، مرتفع عن مساحة النموذج الرئيسية، ولكن لا يمكن تغيير حجمه.
مثبت حواري FixedDialog	نموذج يستخدم في مربعات الحوار، لا يمكن تغيير حجمه.
مثبت مفرد FixedSingle	نموذج لا يمكن تغيير حجمه، بإطار عبارة عن خط مفرد.
نافذة أدوات مثبتة FixedToolWindow	نموذج لا يمكن تغيير حجمه، بزر إغلاق فقط، وبدون زري تصغير وتكبير، وبدون الأيقونة ومربع التحكم.
نافذة أدوات متغيرة الحجم SizableToolWindow	مماثل للطراز السابق، لكنه قابل لتغيير حجمه.

6. مربع التحكّم ControlBox:

يمكنك جعل هذه الخاصية "خطأ" False، لإخفاء أيقونة النموذج، وبالتالي لا يتمكن المستخدم من إظهار مربع التحكم، كما ستختفي أزار التكبير والتصغير.. ولو جعلت خاصية النص Text نصا فارغا ""، فسيختفي شريط العنوان كلّيّة!

7. مراجعة الأزرار KeyPreview:

قم بتجربة الشفرة التالية في حالة جعل قيمة هذه الخاصية True و False ولاحظ الفرق :

```
Private Sub Form1_KeyDown(ByVal sender As Object, ByVal e As _
    System.Windows.Forms.KeyEventArgs) Handles Me.KeyDown
            If e.KeyCode = Keys.F12 Then
                MsgBox("See You Later")
            End If
End Sub
```

8. زر التصغير، وزر التكبير MinimizeBox, MaximizeBox:

اجعل أيا من هاتين الخاصيتين خطأ، لإيقاف عرض زر تصغير النموذج أو زر تكبيره.

9. عرض في شريط المهامّ ShowInTaskbar:

إذا جعلت هذه الخاصيّة True، فستظهر اسم النموذج في شريط مهامّ الويندوز (في الغالب يكون في الجزء السفليّ من الشاشة).

10. عرض أيقونة البرنامج ShowIcon:

إذا جعلت هذه الخاصيّة True، فستظهر أيقونة النموذج في شريط مهامّ الويندوز

11. أصغر حجم وأكبر حجم MinimumSize, MaximumSize:

استخدم هاتين الخاصيتين لمنع المستخدم من تصغير النموذج أو تكبيره عن حد معيّن. وهاتان الخاصيتان كائنان، يمكن تغيير قيمهما من الكود كما يلي:

```
Me.MinimumSize = New Size(400, 300)
```

12. حالة النافذة WindowState:

يمكنك بهذه الخاصيّة قراءة أو تغيير حالة النافذة.. ولهذه الخاصيّة 3 قيم: مكبّرة Maximized ومصغّرة Minimized وعاديّة Normal.. والحالة الأخيرة تعني عرض النموذج بالأبعاد التي تحدّدها خاصيتا العرض Width والارتفاع Height والقمّة Top واليسار Left.

13. طراز علامة تغيير الحجم SizeGripStyle:

تمكنك هذه الخاصية من عرض علامة في ركن النموذج السفليّ الأيمن، لتدل على قابلية النموذج لتغيير حجمه.. وهي تأخذ واحدة من هذه القيم:

القيمة	التأثير
تلقائي Auto	علامة تغيير الحجم ستظهر عند الاحتياج إليها.. وهي القيمة الافتراضيّة.
عرض Show	سيتم عرض علامة تغيير الحجم دائما.
إخفاء Hide	علامة تغيير الحجم لا تعرض على الإطلاق.. ولكن هذا لا يمنع تغيير حجم النموذج بالفأرة كالمعتاد.

14. موضع البداية StartPosition:

تحدد هذه الخاصيّة موضع النموذج عندما يتم عرضه للمرّة الأولى.. وهي تأخذ القيم التالية:

القيمة	تأثيرها
مركز محتويه CenterParent	سيتم توسيط النموذج حول مركز النافذة التي تحتويه.
مركز الشاشة CenterScreen	سيتم توسيط النموذج حول مركز الشاشة.
يدوي Manual	سيعرض النموذج تبعا لموضعه وحجمه كما تحددهما أنت.
حدود الويندوز الافتراضيّة WindowsDefaultBounds	سيعرض النموذج تبعا للموضع والحجم الافتراضيين كما يحددهما الويندوز.
موضع الويندوز الافتراضي WindowsDefaultLocation	سيتم عرض النموذج تبعا للموضع الافتراضيّ كما يحدده الويندوز، ولكن بالحجم الذي حدّدته أنت.

ما يجب أن تلاحظه هنا، هو أنّ استخدامك لأيّ قيمة غير القيمة Manual سيؤديّ لتجاهل الإحداثيّات التي وضعتها للنموذج ـ حتّى لو كانت من الكود ـ عند عرض النموذج لأوّل مرّة (سواء باستخدام الوسيلة Show أو الوسيلة ShowDialog أو جعل خاصيّة True = Visible)، ولكن بعد ذلك يمكنك تغيير موضع النموذج وحجمه من الكود بطريقة عاديّة، دون أن يكون لخاصيّة StartPosition أيّ تأثير.

15. القمّة واليسار والعرض والارتفاع Top, Left, Width, Height:

استخدم هذه الخصائص لتحديد موضع الحافة العليا والحافة اليسرى للنموذج وعرضه وارتفاعه.. وبهذا تكون قد وصفت حجم النموذج وموضعه بطريقة كاملة.

16. يعلو الجميع TopMost:

لو جعلت هذه الخاصية "صواب"، فستجعل هذا النموذج يعلو باقي نماذج تطبيقك، بحيث يظل مرئيا بكامله، حتّى لو نشّطت النماذج الأخرى بضغطها بالفأرة، مثلما يحدث مع نموذج البحث والاستبدال.

17. تثبيت الهامش وانطباق الحافّة Anchoring and Docking:

عندما تصمّم نموذجا قابلا لأن يغيّر المستخدم حجمه، تواجهك مشكلة خطيرة: كيف ستبدو الأدوات عندما يتغير حجم النموذج؟.. هل سيختفي بعضها؟.. هل ستبدو منقرةً وغير متناسقة مع الأبعاد الجديدة للنموذج؟

لهذا كان عليك في الماضي أن تكتب بعض الكود لمعالجة هذه المشكلة، بتعديل موضع الأدوات كلما تغيّر حجم النموذج، وذلك باستخدام الحدث "تغيّر الحجم" Resize Event. ولكن إنّ VB.NET يقدّم لك خاصيتين جديدتين موجودتين في معظم الأدوات، لتريحاك من هذا العناء.. هاتان الخاصيتان هما: "تثبيت الهامش" Anchor و"انطباق الحافة" Dock.

وتتيح لك خاصيّة Anchor، تثبيت واحد أو أكثر من هوامش الأداة بالنسبة لحافة النموذج المقابلة لها، بحيث يظل هذا الهامش ثابتا، مهما غيّر المستخدم من مساحة النموذج، حيث تتغيّر أبعاد الأداة تلقائيّا، للمحافظة على هذا الهامش. جرّب وضع مربع نص على نموذج جديد، وابحث عن خاصّية "تثبيت الهامش" في نافذة الخصائص.. اضغط زر الإسدال في خانة القيمة.. ستظهر لك الأداة كمربّع رماديّ، يتصل بحوافّ النموذج بمستطيلات رفيعة والآن كل ما عليك فعله، هو أن تضغط الوصلة التي تريد تثبيت الهامش من جهتها، حيث ستكتسب لونا رماديا دليلا على تثبيتها، ولو ضغطتها مرة أخرى، فسيزول التثبيت . والآن جرّب أوضاعا مختلفة لتثبيت الهوامش، ثم غيّر أبعاد النموذج بالفأرة لترى الفرق بشكل واضح .

لدينا أيضا خاصّيّة "انطباق الحافة" Dock، والتي تمكّنك من جعل أيّ من حوافّ الأداة منطبقة على حافة النموذج المقابلة.. إنّ القيمة الافتراضيّة لهذه الخاصّيّة هي "ولا واحدة" None.

تعالَ نجرّب هذه الخاصّيّة على مربع نص على نموذج جديد.. وقبل كل شيء أنصحك بجعل الخاصية MultiLine للأداة "True" ثم اضغط زر الإسدال لهذه الخاصية.. ستظهر لك مجموعة من المربعات، يمكنك أن تضغط منها ما يناسبك.. فمثلا لو ضغطت المربع الموجود في المركز، فستنطبق كل حواف الأداة على حواف النموذج، بمعنى أنّ الأداة ستحتل كل مساحة النموذج.

18. مجموعة ادوات مظهر النموذج Appearance

حيث نجد أن إسماء هذه الخصائص يدل على وظيفة كل منها ومن أهمها :

```
BackColor, BackgroundImage, BackgroundImageLayout, Font, ForeColor, Cursor,
RightToLeft
```

19. درجة الإعتام Opacity:

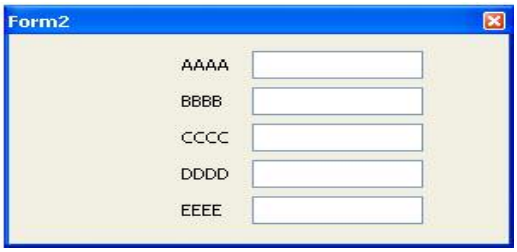
هذا الخاصيّة تأخذ نسبة مئويّة، كلّما قلّت زادت شفافية النموذج، حتّى إنّك إذا جعلت هذه الخاصيّة صفرا، فلن ترى سطح النموذج على الإطلاق!.. وبزيادة هذه الخاصيّة، تقلّ شفافية النموذج.. وفي الوضع التلقائي، تكون قيمة هذه الخاصيّة 1 (100%)، بحيث يكون النموذج معتما تماما (غير شفّاف). لاحظ أنّ هذه الخاصيّة تؤثّر على النموذج ككلّ، بما في ذلك شريط عنوانه والأدوات التي عليه.

جرّب أن تجعل هذه الخاصيّة في نافذة الخصائص صفرا وشغّل التطبيق.. لن ترى أيّ شيء، رغم أنّ النموذج معروض أمامك.. ولكي تتأكّد من ذلك، اضغط زر Alt ثمّ مسطرة المسافات.. ستجد أنّ قائمة التحكّم لهذا النموذج قد ظهرت.. اضغط Close لإغلاق هذا النموذج الخفيّ.حاول تجريب نسبٍ مختلفةٍ لهذه الخاصيّة.

أزرار الوصول Access Keys:

الآنَ نريد تخصيص بعض الأزرار من لوحة المفاتيح لضغط زر ما، أو الوصول مباشرةً إلى مربع نصٍّ ما، بمجرد ضغط زر Alt مع واحد من هذه الأزرار.

إنظر النموذج التالي :-



نريد الآن إضافة حرف وصول لخانة "AAAA".. كل ما عليك فعله، هو وضع العلامة "&" قبل الحرف الذي تريده، في خاصية "نص" Text الخاصة بأداة عنوان AAAA.. فليكن هذا الحرف هو "A".. إذن يجب أن تكتب نص اللافتة كالتالي: "A&AAA".. ستجد أن حرف "A" في اللافتة قد كتب وتحته خط ، معنى هذا أنّه حرف وصول، وأنّ ضغط Alt+A من لوحة المفاتيح، سيؤدّي للانتقال إلى هذا الحقل.

ولكن كيف؟.. ألم نَقُلْ إنّ مربع العنوان لا يتم الانتقال إليها؟!

نعم، ولهذا سيتم الانتقال إلى الأداة التالية في ترتيب الانتقال TabIndex، وهي هنا مربع النص الخاص بإدخال AAAA.. ولا يوجد ما يمنع جعل حرف الوصول الخاص بخانة الشركة أي حرف آخر، مثل "o": AA&AA، أو AAA&A... إلخ.

واجهة الوثائق المتعدّدة:

في كلّ ما سبق، كنّا نتعامل مع نموذج واحد ولكن ماذا لو كان لدينا نموذجان يظهران على الشاشة دون أن تربطهما واجهة واحدة.. إنّ هذا الوضع ليس مفضلا، فقد يزعج المستخدم أن يفتح 5 نماذج مثلا في تطبيقك، فتبدو مبعثرة على الشاشة بلا نظام.. فإذا أراد أن يصغرها أو يكبّرها، اضطرّ إلى أن يصغرّ أو يكبّر كل واحد منها على حدة!

وأفضل حل لهذه المشكلة، هو أن تكون هناك نافذة رئيسيّة للتطبيق، تحتوي كلّ النماذج الأخرى التي يفتحها المستخدم في هذا التطبيق، وتحتوي أيضا على القوائم الرئيسيّة Menus للتطبيق.. هذه النافذة الرئيسيّة هي ما يسمّى بواجهة الوثائق المتعدّدة Multiple Document Interface (MDI).. فإذا أراد المستخدم تصغير كل نماذج التطبيق، فما عليه سوى تصغير النافذة الرئيسيّة، لتختفي من على الشاشة بكلّ ما تحويه من نماذج.. وإذا حرّكها، تحرّكت بكلّ ما تحويه من نماذج، وإذا أغلقها، أغلقت بكلّ ما تحويه من نماذج (يُغلق التطبيق نهائيّا).

ويجب هنا أن نقرّر هذه الحقائق:

- تسمّى واجهة الوثائق المتعدّدة النافذة الرئيسيّة أو النافذة الأمّ.
- تسمّى أي نافذة أخرى النافذة الداخليّة (الابنة).
- من البَدَهِيّ أن يكون للتطبيق نافذة رئيسيّة واحدة، ولكن لا قيود على عدد النوافذ الداخليّة التي يمكن أن يحتويها.

- في وقت التصميم تظهر لك كلّ نافذة مستقلّة، سواء الرئيسيّة أو الداخليّة، ولكن عند تشغيل البرنامج، تظهر النوافذ الداخليّة محتواة بالكامل داخل النافذة الرئيسيّة.
- توجد قائمة رئيسيّة Menu للنافذة الرئيسيّة، ويمكن أن توجد قوائم للنماذج الداخليّة، ولكن عند عرض النموذج الداخليّ، يتمّ دمج عناصر قائمته مع عناصر القائمة الرئيسيّة، ليظهرا معا على النافذة الرئيسيّة.

ولكي يتضح لك الأمر فلنتعرف على كيفيّة بناء التطبيقات متعددة الوثائق.

ابدأ مشروعا جديدا، وأسمِه MDIProject.. وفي متصفّح المشاريع حدّد النموذج، وغيّر اسمه من Form1.vb إلى MDIForm.vb.. ولكي تجعل هذا النموذج هو النافذة الرئيسيّة MDI، حدّد خاصيّة "إنّه يحتوي على وثائق متعدّدة" IsMdiContainer في نافذة الخصائص ، واجعل قيمتها True.. لاحظ تغيّر شكل النموذج بعد قيامك بهذا. غيّر كذلك قيمة خاصيّة Text إلى "النافذة الرئيسيّة".

الآن نحنُ بصدد إنشاء النماذج الداخليّة.. إنّ بإمكانك أن تنشئ أيّ عدد من هذه النماذج، وإن كان الغالب أن تنشئ نموذجا واحدا تستخدمه كقالب، حيث يمكنك أن تعرض منه أيّ عدد تريده من النسخ.

الآن أضف نموذجا جديدا للمشروع وأسمه ChildForm.

ولعرض هذا النموذج كنموذج داخليّ ضمن النافذة الرئيسيّة، سننشئ القائمة File، ومن خلالها يمكن للمستخدم عرض نسخة جديدة من هذا النموذج باستخدام الأمر New.. ابدأ أولا بإنشاء القوائم التالية:

عنوان القائمة	اسمها البرمجيّ
ملفّ File	FileMenu
جديد New	FileNew
خروج Exit	FileExit

والآن اكتب الكود التالي في كود الأمر New:

```
ChildForm.MdiParent = Me
ChildForm.Show()
```

هذا هو الأمر الذي يجعل أيّ نافذة تنتمي للنافذة الرئيسيّة، حيث سيتمّ عرضها داخلها، وستظهر قوائمهما متداخلة.

أدوات الويندوز الأساسيّة Basic Windows Controls

وفرت لنا بيئة VS.Net مجموعة من كبيرة من الأدوات التي لا تتغيير مهما اختلفت اللغة المستخدمة في هذه البيئة مما يسهل فهم بقية لغات VS.Net وهنا سوف نتطرق لأهم هذه الأدوات وأكثرها استخداماً وهي كما يلي :-

1. مربّع النصّ TextBox

وهو غنيّ عن التعريف، فلقد رأينا مرارا، كيف يمكن استخدامه لكتابة وتحرير النصوص ولذلك سيتم شرح أهم خصائصها بشيء من التفصيل لنتجنبها فيما بعد مع الأدوات الأخرى .

- أهم الخصائص **Basic Properties**:

i. **مجموعة أدوات المظهر Appearance**

ومن أسماءها يمكن التعرف على وظائفها ومن أهمها :

```
BackColor,BorderStyle,Cursor,Font,ForeColor,RightToLeft,Text,UseWaitCursor
```

ii. **الأسطر Lines**

وهذه الخاصية عبارة عن مصفوفة من نوع String غير محددة الطول حيث توضع بها مجموعة من النصوص التي سوف يتم إضافتها إلى الأداة , يمكنك بهذه الخاصيّة قراءة أي سطر في مربع النصّ، بمجرّد تحديد رقمه.. فمثلا: يمكن قراءة السطر الأوّل باستخدام تعبير Lines(0)، والثاني باستخدام تعبير Lines(1).. وهكذا.

iii. **متعدّد الأسطر MultiLine:**

بجعل هذه الخاصيّة صوابا، يمكن للمستخدم كتابة أسطر عديدة في مربع النصّ، وذلك بضغط زر الإدخال Enter للانتقال لسطر جديد.

وفي الوضع الافتراضيّ، تكون قيمة هذه الخاصية خطأ، ممّا يعني أنّ مربع النص قابل لكتابة سطر واحد فقط.

iv. **محاذاة النصّ TextAlign:**

يمكن أن تكون واحدة من ثلاث قيم: اليسار Left، واليمين Right، والوسط Center.

v. **المنزلقان ScrollBars:**

تسمح لك هذه الخاصيّة باختيار المنزلق الذي تودّ ظهوره (الأفقي أو الرأسي أو كليهما أو ولا واحد منهما)، في حالة تجاوز النص لحدود مربع النص.

ملاحظتان:

- لا يظهر أي من المنزلقين إلا إذا كان مربع النص متعدد الأسطر.
- في حالة وجود المنزلق الأفقي، لا يلتفّ السطر الطويل إلى السطر التالي.

vi. التفاف الأسطر WordWrap:

لو كانت هذه الخاصية "صوابا" (وهي القيمة الافتراضيّة بالفعل)، فإنّ السطر الذي يتجاوز طوله عرض مربع النصّ، سيلتفّ إلى السطر التالي، بدلا من الاحتياج إلى منزلق أفقيّ لقراءته.

ملاحظة:

ليس لهذه الخاصيّة أي تأثير على النص الأصليّ، فالسطر الذي تراه ملتفا ليتم إكماله في سطر تالٍ ما زال سطرا واحدا.. إنّها مجرّد طريقة للعرض.

vii. أقصى طول MaxLength:

القيمة الافتراضيّة لهذه الخاصيّة هي 32,767، وهي أقصى طول كان مربع النص يحتويه في VB6.. ولكن لو جعلت قيمة هذه الخاصيّة صفرا، يمكنك أن تكتب حوالي 2 مليار حرف (2 جيجا) في مربع النصّ!
ويمكن وضع هذه القيمة بأيّ رقم تريده (5 مثلا)، لمنع المستخدم من كتابة أي نص أطول من هذا.

viii. نص Text:

تحمل هذه الخاصيّة النصوص التي تكتب في مربع النص. وهذه الخاصيّة من نفس نوع المتغيرات النصّيّة String، لهذا يمكن معاملتها كما تعامل المتغيرات من هذا النوع: ترسلها كمعامل للدوال التي تستقبل نصوصا، تضيف محتوياتها إلى محتويات نص آخر، أو تستخدم معها وسائل خليّة النصوص String Class.
مثلا: هذا التعبير يحسب طول النص الموجود في مربع النص:

```
Dim StrLen As Integer = TextBox1.Text.Length
```

والذي ما زال يمكن حسابه بالطريقة القديمة:

```
Dim StrLen As Integer = Len(TextBox1.Text)
```

أو حسابه مباشرة بهذه الطريقة الثالثة:

```
Dim StrLen As Integer = TextBox1.TextLength
```

ويمكنك إفراغ مربع النص ومسح كل محتوياته بوضع نص فارغ فيه:

```
TextBox1.Text = ""
```

أو باستخدام الوسيلة "مسح" Clear:

```
TextBox1.Clear
```

ix. للقراءة فقط ReadOnly:

بجعل هذه الخاصيّة "صوابا"، لن يتمكّن المستخدم من تغيير النص المعروض في مربع النص.
وكما ذكرنا سابقا، فإنّ هذه الخاصيّة كانت تحمل في الماضي اسم "مغلق" Locked، ولكنّ هذا الاسم الآن صار لخاصيّة أخرى، تقوم بتثبيت الأداة على النموذج، حتّى لا يتم تحريكها في وقت التصميم.

x. حالة الأحرف CharacterCasing

حيث يتم تقييد حالة الحروف المدخلة إلى الأداة بإحدى القيم التالية :
Lower (الأحرف صغيرة) , Upper(الأحرف كبيرة) , Normal(بحسب ما يدخلة المستخدم) .

xi. حرف كلمة المرور PasswordChar:

عندما تستخدم مربع نصّ لاستقبال كلمة المرور من المستخدم، يمكنك أن تضع أي حرف في هذه الخاصيّة، ليتم تحويل كل ما يكتبه المستخدم إلى هذا الحرف، بحيث لا يرى المحيطون به الحروف الحقيقيّة لكلمة المرور.. وفي هذه الحالة، لن يتمكّن أي مستخدم من نسخ أو قصّ أي جزء من كلمة المرور.

عامّة، لا تؤثّر هذه الخاصيّة على قيمة النص الموجود في مربع النصّ، فالحرف الذي تراه مكتوبا هو مجرد طريقة للعرض، ولكن لو استخدمت خاصية "نص" Text، فيمكنك التعامل مع النص الأصليّ الذي أدخله المستخدم.

xii. إضافة نص AppendText:

تمكّنك هذه الوسيلة من إضافة أي نص على نهاية ذلك الموجود في مربع النص، وهي أسرع بمراحل من استخدام الطريقة التقليديّة:

```
TextBox1.Text = TextBox1.Text & "نص جديد"
```

الآن يمكنك استخدام الطريقة التالية:

```
TextBox1.AppendText("نص جديد")
```

حيث ستتم إضافة جملة "نص جديد" على النص الحاليّ، فإذا ما أردت أن تتمّ كتابتها في سطر مستقلّ، فاستخدم التعبير:

```
TextBox1.AppendText(vbCrLf & "نص جديد ")
```

حيث vbCrLf هو ثابت من ثوابت VB، حيث Cr هي اختصار كلمة "منعطف السطر" Carriage Return، وهو رمز يدلّ على نهاية السطر، يناظر الرقم 13 في الأرقام الدالة على الحروف ASCII، و Lf هي اختصار تعبير "مغذي السطر" Line Feed، وهو رمز يدل على بداية السطر، يناظر الرقم 10.. الخلاصة أنّ هذا الثابت يؤدّى لإضافة سطر جديد لمربع النص، بطريقة بديلة لاستخدام الجملة المتاحة التالية:

```
TextBox1.AppendText(Chr(13) & Chr(10) & "نص جديد")
```

xiii. تفعيل Enabled

تمنحك هذه الأداه إمكانية التعامل مع مربع النص وقت التنفيذ في حالة إذا كانت قيمة هذه الخاصية (True) أما إذا كانت قيمة هذه الخاصية (False) فلن تتمكن من التفاعل مع هذه الأداة .

xiv. مرئي Visible

تمنحك هذه الأداه إمكانية رؤية مربع النص وقت التنفيذ في حالة إذا كانت قيمة هذه الخاصية (True) أما إذا كانت قيمة هذه الخاصية (False) فلن تتمكن من التفاعل مع هذه الأداة .

xv. تثبيت الهامش وانطباق الحافة Anchoring and Docking (تم شرحه سابقاً)

- **التربّص بضربات الأزرار Capturing stroke Keys:**

رغم أنّ الأحداث الناشئة عن ضغط أزرار لوحة المفاتيح ليست موقوفة على مربّع النصّ، إلا إنّ هذه الأداة هي أهمّ موضع تحتاج فيه لمعرفة الأزرار المضغوطة والاستجابة لها لأن أغلب ما يضيفه المستخدم للنماذج وقت التنفيذ من خلال هذه الأداة , من أهم و أكثر الأحداث التي يمكن استخدامها في هذا الصدد مجموعة الأحداث KeyPress, KeyDown, KeyUp الذي يحدث كلما ضغط المستخدم زرا من لوحة المفاتيح، حيث يمنحك المعامل e القدرة على معرفة الزر الذي تمّ ضغطه. تعالَ نرى كيف نستخدم الحدث KeyPress :

مثال : ماذا لو أردنا منع المستخدم من إضافة شيء (حرف أو رقم أو رمز) داخل مربع النص فكيف نقوم بذلك :

```
Private Sub TextBox1_KeyPress(ByVal sender As Object, ByVal e As _
        System.Windows.Forms.KeyPressEventArgs) Handles TextBox1.KeyPress
    If Not IsNumeric(e.KeyChar) Then
        MsgBox(e.KeyChar)
        e.Handled = True    ' لوقف إستجابة الأداة لضغط الزر ( غير الأرقام )
    End If
End Sub
```

لاحظ أنك لو وضعت في أحد الحدثين KeyDown, KeyUp الشفرة السابقة فإن الأداة سوف تستجيب لضغط الزر لآن الحدث يطلق بعد الضغط .

- **التربّص بحروف الوظائف Capturing Function Keys:**

في برنامج Notepad الخاصّ بالويندوز، يمكنك استخدام F5 لكتابة التاريخ في الموضع الحالي من النصّ.. ماذا لو أردت أن تفعل المثل في تطبيق TextPad؟

في هذه الحالة لن يمكنك استخدام الحدث "ضغط الزر" KeyPress، لأنّ هذا الحدث لا ينطلق عند ضغط أزرار الوظائف! لكن لحسن الحظ، لدينا الحدثان "انخفاض الزر" KeyDown و"ارتفاع الزر"KeyUp"، اللذان يستجيبان لحروف الوظائف. وبخلاف حدث "ضغط الزرّ" فإنّ هذين الحدثين يستقبلان الرقم الدال على الحرف المضغوط وليس الحرفَ نفسه، وذلك من خلال التعبير e.KeyCode.

ويجب أن تلاحظ أنّ رقم الزرّ KeyCode، مختلف عن ذلك المسمّى ASCII، ففي الشفرة الأخيرة، يكون لكل حرف رقم مختلف، فمثلا الحرف الكبير A يختلف رقمه عن الحرف الصغير a.. ولكن في رقم الزر KeyCode، فإنّ لكل زر على لوحة المفاتيح رقما خاصا به، وبهذا تجد أنّ حرفي A و a لهما نفس الرقم لأنّهما على نفس الزرّ.. إذن كيف نفرّق بينهما؟

إنّ هذين الحدثين يمنحانك القدرة على معرفة حالة الأزرار Shift و Ctrl و Alt، عن طريق اختبار إذا ما كانت الخصائص التالية صوابا أم خطأ: e.Shift و e.Control و e.Alt.. وبهذا تستطيع أن تعرف بالضبط أن حرف A الكبير هو المقصود، ما دام زر Shift مضغوطا، والعكس بالعكس. وإذا أردت أن تعرف إذا ما كان حرفا تحكّم مضغوطين معا، استخدم تعبيرا كالتالي:

```
Private Sub TextBox1_KeyDown(ByVal sender As Object, ByVal e As
            System.Windows.Forms.KeyEventArgs) Handles TextBox1.KeyDown
    If e.Control And e.Alt Then MsgBox("Yes")
End Sub
```

ولا تشغل ذهنك بأرقام الأزرار، فبإمكانك استخدام مرقم الأزرار Keys، لتتعامل بطريقة أوضح وأسهل مع الأزرار.. انظر لهذا المثال الذي يشرح نفسه ولا يحتاج لأيّ إيضاح:

```
Public Sub Editor_KeyUp(sender As Object, e As KeyEventArgs)
                                                Handles Editor.KeyUp
    Select Case e.KeyCode
        Case Keys.F5 : TextBox1.SelectedText = Now().ToLongDateString
        Case Keys.F6 : TextBox1.SelectedText = Now().ToLongTimeString
    End Select
End Sub
```

2. القوائم Lists

- أنواع مختلفة من القوائم:

لدينا ثلاث أدوات، تُستخدم كل منها لعرض مجموعة من العناصر كقائمة، بحيث يمكن للمستخدم أن يختار منها عنصرا أو أكثر: "القائمة" ListBox، و"القائمة الاختياريّة" CheckedListBox، و"القائمة المركّبة" ComboBox.

والقائمة الاختياريّة، تعرض كلّ عنصرٍ من عناصرها وبجواره مربع اختيار، بحيث يمكن للمستخدم أن يضع علامة الاختيار بجوار أيّ عدد من عناصرها.

أمّا القائمة المركّبة فهي مماثلة للقائمة التقليديّة، إلا إنّها تظهر في حجم أقلّ، حيث يمكن ضغط زر إسدالها لاختيار أيّ من عناصرها، قبل أن تعود للانكماش، بالإضافة إلى أنّها تعمل كذلك كمربّع نصّ، حيث يمكن للمستخدم أن يكتب فيها ما يريد، دونَ أن يكون مقيّدا بالاختيار من عناصرها فقط.

ويمكن ملء هذه القوائم باستخدام نافذة الخصائص، باستخدام خاصيّة "العناصر" Items، والتي يمكن كتابتها بضغط زر الانتقال، حيث ستظهر لك نافذة تحرير النصوص.. اكتب فيها ما تريد من العناصر، كل عنصر في سطر مستقلّ.. وبهذا ستظهر لك هذه العناصر في القائمة عند تشغيل البرنامج.

ولكي تتعامل مع عناصر القائمة من الكود، يمكنك استخدام مجموعة العناصر Items Collection، وطبعا صار مألوفا لديك شكل المجموعات Collections، فهي تشابه المصفوفات، فمثلا أول عنصر في القائمة هو Items(0)، ويليه Items(1)، وهكذا.

وككلّ المجموعات، تحتوي مجموعة عناصر القائمة على الخواص التقليديّة: "العدد" Count، و"إضافة" Add، و"إزالة" Remove، و"إفراغ" Clear. ويمكنك التعامل مع العناصر المحدّدة في القائمة بهذه الوسائل:

- تحديد العناصر SelectionMode (One, None, MultiExtended, MultiSimple).
- رقم العنصر المحدّد SelectedIndex:
- العنصر المحدّد SelectedItem:
- مجموعة العناصر المحدّدة SelectedItems Collection:
- مجموعة أرقام العناصر المحدّدة SelectedIndices Collection:

فمثلا، يمكنك استخدام الكود التالي لعرض كلّ العناصر المحدّدة في القائمة قبل كتابة الشفرة السابقة إجعل الخاصية (SelectionMode = MultiExtended أو MultiSimple)

مثال (SelectedIndex):

```
MsgBox(ListBox1. SelectedIndex)
```

مثال (SelectedItem):

```
MsgBox(ListBox1.SelectedItem)
```

مثال (SelectedItems) :

```
Dim Itm As Object
For Each Itm In ListBox1.SelectedItems
      MsgBox(Itm)
Next
```

مثال (SelectedIndices):

```
MsgBox(ListBox1.SelectedIndices.Count)
```

كما يمكنك البحث عن أي عنصر في القائمة بإحدى هاتين الطريقتين، اللتين تعملان سواء كانت القائمة مرتّبة أم لا، وكلتاهما لا تراعي حالة الأحرف:

- "ابحث عن النص" **FindString**: للبحث التقريبي عن أوّل عنصر في القائمة يحتوي على نصّ البحث.. فمثلا لو بحثت بهذه الوسيلة عن النصّ"VB.Net" فمن الممكن أن تعثر على النصّ " VB.Net2005". وتعيد لك هذه الدالة رقم يشير إلى موقع النص في القائمة في حالة وجودة أو تعيد (-1) إذا لم يعثر عليه .
- "ابحث عن النصّ المطابق" **FindStringExact**: للبحث عن أول عنصر في القائمة يطابق نصّ البحث.

وتقبل هاتان الوسيلتان معاملين: عنصر البحث، ورقم الخانة التي يبدأ منها البحث في القائمة، وهو معامل اختياريّ يمكن ألا ترسله، وفي هذه الحالة سيتم البحث من أوّل عنصر في القائمة.

كما يمكن البحث عن أحد العناصر مع مراعاة حالة الأحرف، باستخدام الوسيلة **IndexOf** الخاصّة بمجموعة العناصر **Items Collection**، كالتالي:

```
MsgBox(ListBox1.Items.IndexOf("VB.NET"))
```

وكما هو واضح، لا تقبل هذه الطريقة تحديد الخانة التي يبدأ منها البحث.

- **إضافة العناصر لمجموعة العناصر:**

يمكن استخدام أي من الصيغتين التاليتين:

```
ListBox1.Items.Add(عنصر)

ListBox1.Items.Insert(العنصر , موقع أو رقم العنصر)
```

انظر لهذا المثال، الذي يضيف عناصر مصفوفة لقائمة:

```
Dim words(1000) As String
For j As Int16 = 0 To 999
    words(j) = j
Next
ListBox1.BeginUpdate() 'لإيقاف إنعاش القائمة حتى تكون إضافة العناصر أسرع
For I As Int16 = 0 To 999
    ListBox1.Items.Add(words(I))
Next
ListBox1.EndUpdate() ' لإعادة إنعاش القائمة
```

- **حذف عنصر من القائمة:**

يمكن حذف أي عنصر من القائمة إذا كنت تعرف رقمه، باستخدام الوسيلة "حذف من الموضع" كالتالي:

```
ListBox1.Items.RemoveAt(رقم العنصر)
```

كما يمكن حذف العنصر نفسه باستخدام الوسيلة "حذف" كالتالي:

```
ListBox1.Items.Remove(عنصر)
```

سواء كان هذا العنصر نصًّا أم كائنا.. فإذا كان المراد حذف نصّ، وتصادف تكرار هذا النصّ أكثر من مرّة، يتمّ حذف النصّ الأوّل فقط.

- **التأكّد من وجود عنصر معين في القائمة:**

يمكنك استخدام الوسيلة "تحتوي" Contains كالتالي:

```
Dim Itm As String = "VB.NET"
If ListBox1.Items.Contains(Itm) Then
      ListBox1.Items.Add(Itm)'   إذا لم يجد الكلمة يقوم بإضافتها
End If
```

هذا بالإضافة إلى الوسيلة "رقم العنصر" IndexOf التي تكلمنا عنها من قبل، حيث تُرجع -1 إذا لم يكن العنصر موجودا في القائمة.

- **بعض الخصائص الخاصة بالقائمة المركّبة ComboBox:**

1. **عرض القائمة المنسدلة DropDownWidth**

لديك ثلاثة خيارات لهذه الخاصية الخاصة بالقوائم المركّبة، التي تأخذ القيم التالية:

منسدلة DropDown	وهي القيمة الافتراضيّة.. ستظهر القائمة المركّبة كمربّع نصّ يمكن أن يكتب المستخدم به، وقائمة منسدلة يمكن الاختيار منها.
منسدلة كالقائمة DropDownList	تماثل القيمة السابقة، إلا إنّ المستخدم لن يكون بمقدوره الكتابة في مربّع النصّ.
بسيطة Simple	ستظهر القائمة المركّبة كمربّع نصّ يمكن أن يكتب المستخدم به، وقائمة غير منسدلة يمكن الاختيار منها.

2. **نصّ Text:**

تمكّنك هذه الخاصيّة من قراءة النصّ المكتوب حاليا في مربع النص الخاص بالقائمة المركّبة.. ولكن كيف يمكن السماح للمستخدم بإدخال عناصر جديدة في القائمة المركّبة، وذلك لأنّ مجرّد الكتابة في مربّع نصّ القائمة، لا يضيف النصّ المكتوب لعناصر القائمة.. لهذا يجب أن تضيف أنت ما كتبه المستخدم لعناصر القائمة، إمّا بوضع زر لهذا الغرض، أو بتحديث عناصر القائمة عندما يتركها المستخدم منتقلا إلى أيّ أداة أخرى (حدث "فقد مؤشّر الكتابة" LostFocus)، أو عندما يضغط المستخدم زر الإدخال Enter بعد كتابة العنصر الجديد في مربع نصّ القائمة المركّبة.

```
Private Sub ComboBox1_KeyPress(ByVal sender As Object, ByVal e As
        system.Windows.Forms.KeyPressEventArgs) Handles ComboBox1.KeyPress
    If e.KeyChar = Chr(13) Then ComboBox1.Items.Add(ComboBox1.Text)
End Sub
```

3. **القيمة المعروضة SelectedIndex**

```
ComboBox1.SelectedIndex = 0
ComboBox1.SelectedIndex = -1
```

3. أدوات صناديق الحوار الشائعة

ستجد هاتين الأداتين في صندق الأدوات Tool Box باسمي OpenFileDialog و SaveFileDialog، وهما متماثلتان في معظم خصائصهما، لهذا سنتناولهما بالشرح معا..

إنّ كلتا الأداتين تمكّن المستخدم من اختيار اسم الملفّ وامتداده والمسار الذي سيُقرأ منه أو يحفظ فيه، كما تمنحان المبرمج القدرة على تحديد الامتدادات التي يُسمح للمستخدم بالتعامل معها.

ولكن يجب أن تعرف بأن هاتين الأداتين لا تقومان بفتح الملفّ أو حفظه.. كلّ ما تفعلانه هو منح المستخدم واجهة مريحة، وعند إغلاقهما يمكنك قراءة اسم الملفّ الذي اختاره المستخدم، وعليك أنت أن تقوم بفتحه أو حفظه كما يحلو لك، وللتسهيل تمنحك الأداة OpenFileDialog الوسيلة OpenFile لفتح الملفّ للقراءة فقط، وتمنحك الأداة SaveFileDialog الوسيلة SaveFile لحفظ الملفّ, ولعرض أيّ من مربّعي الحوار، استخدم الوسيلة ShowDialog.

- خصائصهما:

1. الامتداد الافتراضيّ DefaultExtension:

هذا هو الامتداد الذي سيظهر للمستخدم عند فتح مربّع الحوار، والذي سيتمّ حفظ الملفّ به إذا لم يتمّ اختيار غيره.. وهذه الخاصيّة تأخذ قيمة نصّيّة على غرار "txt." أو ".bin".. إلخ.

2. إضافة الامتداد AddExtension:

إذا جعلت هذه الخاصيّة True، فإنّ الامتداد الافتراضيّ الذي حدّدته بالخاصيّة السابقة ستتمّ إضافته تلقائيّا لاسم الملفّ الذي اختاره أو كتبه المستخدم، ما لم يختر امتدادا آخر.

3. تحقّ من وجود الملفّ CheckFileExists:

إذا جعلت هذه الخاصيّة True، فإنّ مربّع الحوار يتحقّق قبل إغلاقه من أنّ الاسم الذي كتبه المستخدم هو اسم ملفّ موجود بالفعل على الجهاز، وإلا أطلق رسالة تحذير.

4. تحقّق من وجود المسار CheckPathExists:

إذا جعلت هذه الخاصيّة True، فإنّ مربّع الحوار يتحقّق قبل إغلاقه من أنّ المسار الذي كتبه المستخدم في اسم الملفّ، هو مسار موجود بالفعل على الجهاز، وإلا أطلق رسالة تحذير.

5. اسم الملفّ FileName:

يمكنك أن تضع في هذه الخاصيّة اسم الملفّ الذي تريد عرضه للمستخدم عند فتح مربع الحوار، كما يمكنك أن تحصل منها على اسم الملفّ الذي اختاره المستخدم بعد إغلاق مربّع الحوار.

6. تحقّق من الأسماء ValidateNames:

إذا كانت هذه الخاصيّة True (وهي القيمة الافتراضيّة ولا يجب تغييرها)، فسيتحقّق مربّع الحوار من صلاحية الاسم الذي يدخله المستخدم، ومن خلوّه من الحروف غير المسموح بها، قبل إغلاق مربّع الحوار.

7. المرشّح Filter:

يمكنك بهذه الخاصيّة أن تحدّد نوعيّة الملفّات التي تظهر في قائمة نوع الملفّات في مربّع الحوار.. فمثلا: لعرض ملفّات النصوص فقط، استخدم الجملة التالية:

```
OFD.Filter = "Text files|*.txt"
```

لاحظ أنّ العلامة "|" تفصل بين امتداد الملفّ ووصفه (ذلك الذي يظهر للمستخدم في قائمة أنواع الملفّات).

ومن الممكن أن تعرض أكثر من امتداد لنفس النوع، فمثلا: يمكن أن يكون ملفّ النصوص من النوع وثيقة "doc." بالإضافة إلى النوع "txt".. في هذه الحالة يمكنك أن تستخدم جملة كالتالية:

```
OFD.Filter = "Text files|*.txt;*.doc"
```

لاحظ أنّ باستطاعتك إضافة أيّ عدد من الامتدادات مع الفصل بين كلّ منها بفاصلة منقوطة ";".

وفي الغالب تحتاج لعرض أنواع مختلفة من الملفّات، مثل ملفّات النصوص والصور.. في هذه الحالة استخدم الجملة التالية:

```
OFD.Filter = "Text files|*.txt;*.doc|Images|*.BMP;*.GIF;*.JPG"
```

حيث استخدمنا العلامة "|" أيضا للفصل بين النوعين.. لاحظ أنّنا لم نضع أيّ مسافات في هذا النصّ، وإلا فإنّها ستظهر للمستخدم أو سيعتبرها مربّع الحوار جزءا من الامتداد.

وأحيانا تضع للمستخدم اختيارا لعرض كلّ الملفّات.. افترض أنّنا نريد أن نعرض للمستخدم في قائمة أنواع الملفّات العناصر التالية:

Text files , Images , Texts & Images , All files

في هذه الحالة استخدم الجملة التالية:

```
OFD.Filter = "Text files|*.txt;*.doc" & _
             "|Images|*.BMP;*.GIF;*.JPG" & _
             "|Texts & Images|*.txt;*.doc;*.BMP;*.GIF;*.JPG" & _
             "|All files|*.*"
```

ولكن في معظم الأحيان، تلاحظ أنّ قائمة أنواع الملفّات تعرض نوع الملفّ وبجواره امتداداته، مثل:

Text files(*.txt, *.doc)

Images(*.BMP, *.GIF, *.JPG)

Texts & Images(*.txt, *.doc, *.BMP, *.GIF, *.JPG)

All files(*.*)

في هذه الحالة كلّ ما عليك هو تطوير الوصف لإضافة الامتداد بجواره كالتالي:

```
OFD.Filter = "Text files(*.txt, *.doc)|*.txt;*.doc" & _
             "|Images(*.BMP, *.GIF, *.JPG)|*.BMP;*.GIF;*.JPG" & _
             "|Texts & Images(*.txt, *.doc, *.BMP, *.GIF, *.JPG)" & _
             "|*.txt;*.doc;*.BMP;*.GIF;*.JPG|All files(*.*)|*.*"
```

8. رقم المرشّح FilterIndex:

إذا أردت أن تختار نوعا من أنواع الملفّات بحيث يكون معروضا للمستخدم عند فتح مربّع الحوار، فاستخدم هذه الخاصيّة.. فمثلا لو افترضنا أنّ قائمة الأنواع تحتوي على الأنواع الأربعة السابقة، فيمكنك أن تعرض للمستخدم ملفّات الصور Images عن طريق جعل هذه الخاصيّة = 2.

9. المجلّد المبدئيّ InitialDirectory:

استخدم هذه الخاصيّة لتحديد المجلّد الذي تريد عرض محتوياته للمستخدم عندما يفتح مربّع الحوار.

فمثلا لجعل هذا المجلّد هو مجلّد البرنامج، استخدم الجملة التالية:

```
OpenFileDialog1.InitialDirectory = Application.ExecutablePath
```

10. احتفظ بالمجلّد RestoreDirectory:

لو جعلت هذه الخاصيّة True، فسيعرض مربّع الحوار آخر مجلّد كان به المستخدم في المرّة السابقة، ما لم يتمّ تحديد المجلّد المبدئيّ بالخاصيّة السابقة.

- **بعض الخصائص الخاصّة فقط بمربّع حوار "فتح ملفّ":**

1. متعدّد الاختيار MultiSelect:

لو جعلت هذه الخاصيّة True، فسيتمكّن المستخدم من اختيار أكثر من ملفّ من مربعّ فتح ملفّ، عن طريق ضغط هذه الملفّات بالفأرة مع ضغط زر Ctrl من لوحة المفاتيح.. وطبعا يجب أن تكون هذه الملفّات المختارة كلّها في نفس المجلّد.

أسماء الملفّات FileNames:

ترجع هذه الخاصيّة مصفوفة نصّيّة تحتوي على كلّ أسماء الملفّات التي اختارها المستخدم في مربّع فتح ملفّ.

عرض للقراءة فقط ShowReadOnly:

إذا جعلت هذه الخاصيّة True، فسيتمّ عرض مربّع اختيار للقراءة فقط على مربّع الحوار.

اختيار للقراءة فقط ReadOnlyChecked:

إذا جعلت هذه الخاصيّة True، فسيتّم وضع علامة (ü) على مربّع اختيار للقراءة فقط عند فتح مربّع الحوار.. كما يمكنك بعد إغلاق المستخدم لمربّع الحوار، أن تستخدم هذه الخاصيّة لمعرفة إذا ما كان المستخدم قد اختار فتح الملفّ للقراءة فقط أم لا، فإذا كان الأمر كذلك، فيجب عليك أن تفتح الملفّ للقراءة فقط، ولا تسمح بتغيير محتوياته.

- **الأداة ColorDialog**

تعرض هذه الأداة صندوق حوار الألوان لتمكن المستخدم من اختيار اللون بطريقة افضل، ويعتبر استخدام هذه الأداة سهل جدا حيث سيكون جل تركيزك على الخاصية Color :

```
With ColorDialog1
      If .ShowDialog = Windows.Forms.DialogResult.OK Then
            Me.BackColor = .Color
      End If
End With
```

- الأداة **FontDialog**

اما الأداة **FontDialog** فتعرض لك صندوق حوار الخطوط **Fonts** , تحوي على مجموعة من الخصائص الاضافية كالخاصية **ShowColor** لتظهر قائمة الالوان (يمكنك الاستعلام عن اللون المختار عن طريق الخاصية **Color**) , يمكنك فتح صندوق الحوار هذا بنفس الطرق السابقة :

```
With FontDialog1
      .ShowColor = True
      If .ShowDialog = Windows.Forms.DialogResult.OK Then
          TextBox1.Font = .Font
          TextBox1.ForeColor = .Color
      End If
End With
```

4. الأداة Button

استخدام هذه الأداة معروف وسهل جدا حتى لمستخدمين **Windows** العاديين، وهو زر يتم ضغطه لتنفيذ أوامر معينة لا يوجد الكثير لأخبرك به حول هذا الزر سواء وجود خاصيتين تابعة لنافذة النموذج تؤثران تاثير بسيط على هذا الزر هما **CancelButton** و**AcceptButton** تحدد كما أوضحنا سلفاً .

5. الأداة CheckBox

تملأ هذه الأداة اغلب تطبيقات **Windows** يمكنك تحديد ما اذا كانت الأداة مختارة باسناد القيمة **True** إلى الخاصية **Checked** و **False** لالغاء الاختيار، عندما يقوم المستخدم بالنقر على الأداة سيتم عكس قيمة خاصيتها **Checked** بشكل تلقائي، مع ذلك تستطيع منع هذا التغيير باسناد القيمة **False** إلى الخاصية **AutoCheck** , لتحصر المسؤولية عليك في كتابة الشيفرات البرمجية واللازمة لتغيير قيمة الخاصية **Checked** .

الخاصية **CheckAlign** مثل الخاصية **TextAlign** تماما، ويكمن الفرق في ان الاولى خاصة بموقع رمز المربع فقط، بينما الثانية فخاصة بالنص المرافق لرمز المربع

6. الأداة RadioButton

وجه الشبه بين هذه الأداة والأداتين التي قبلها هو ان كلاهم مشتق وراثيا من الفئة **ButtonBase** والتي تحتوي على خصائص اضافية كـ **FlatStyle** لتحديد شكل ثلاثي الأبعاد **3D** والخاصية **Appearance** التي تمكنك من استخدام شكل الزر **Button** مع الأداتين **CheckBox** و **RadioButton** .

يمكنك إسناد القيمة **True** للخاصية **Checked** التابعة لهذه الأداة لاختيارها، مع العلم ان باقي الأدوات في نفس المجموعة (نفس الأدوات المحضونة في الأداة الحاضنة) سيتم إسناد القيمة **False** لخصائصها **Checked** .

7. الأداة LinkLabel

وتشبه هذه الأداة لحد كبير مربع العنوان **Lable** بإستثناء إمكانية هذه الأداة التعامل مع الإرتباطات **Links** ولعل أهم خصائص هذه الأداة الخاصية **ActiveLinkColor** و **LinkColor** و **LinkVisited** وهي الأفضل من حيث المظهر والمضمون لفتح الإرتباطات كما يلي :

```
Process.Start("http://www.SabaUni.net/")
```

8. الأداة ToolTip

تمكنك الأداة ToolTip من عرض مستطيل التلميح على الأداة واستخدامها سهل جدا، فكل ما هو مطلوب منك إضافة نسخة من الأداة ToolTip على النموذج، وبذلك تضيف لخاصية ToolTip on ToolTip1 لكل أداة موجودة على نافذة النموذج، أسند قيمة حرفية لهذه الخاصية الجديدة في كل أداة ليتم عرضها كتلميح ان ظل مؤشر الفأرة فترة من الوقت دون تحريك على سطح الأداة .

9. أدوات أخرى

- الأداة PictureBox أداة بسيطة تمكنك من وضع صور عليها في خاصيتها Image , كما تستطيع تحجيم الصورة لتغطي كامل الأداة أو تحجيم الأداة لتغطي كامل الصورة عن طريق الخاصية SizeMode .
- الاداتان Panel و GroupBox كلاهما من النوع الحاضن Container مثل نافذة النموذج، ابرز الفروق بينهما في دعم الخاصية AutoScroll حيث الثانية تفتقر هذه الخاصية كما انك لا تستطيع اخفاء حدودها، مع ذلك توجد ميزة ليست موجودة في الأداة الاولى وهي النص الظاهر في احد زوايا الأداة تحدده عن طريق الخاصية Text .
- إن أردت أداة مثل الأداة TextBox ولكنها تدمج تنسيقات مختلفة من الخطوط، الألوان، الاحجام، وحتى الصور، فقد تحتاج إلى الأداة RichTextBox والتي تحتوي على خصائص وطرق كثيرة جدا .
- يمكنك تسهيل اختيار القيم العددية بدلا من كتابة الارقام باستخدام الأداة NumericUpDown .
- كما تستطيع التسهيل عليه اكثر باختيار قيم التاريخ عن طريق الاداتين DateTimePicker و MonthCalendar .
- وان كانت المهام المنجزة طويلة، فيفضل عرض شريط نسبة مئوية للمستخدم باستخدام الأداة ProgressBar

- اخيرا، أداة المؤقت Timer تمكنك من تنفيذ شيفرات في حدثها الوحيد Tick والذي يتم تنفيذه كل فترة معينة تحددها في خاصية الأداة Interval (وحدتها 0.001 ثانية) , يمكنك بدء تنفيذ الحدث باسناد القيمة True إلى الحدث Enabled أو القيمة False لإيقافة .

مثال :أضف Lable ثم أضف Timer Control على النموذج واذهب إلى خصائص الـ Timer وضع 1000 في خاصية الـ Interval والمقصود بها الفترة التي يتم فعل شيئ بعدها وهنا وضعنا 1000 ملي سكند وتعادل ثانية واحدة. إضغط مرتين على Timer Control ليظهر حدث Tick واكتب الكود التالي:

```
Label1.Text = Now.Date + Now.TimeOfDay
```

حيث أن الدالة Now تعيد لنا الوقت والتاريخ وقد أسندنا القيمة إلى Label1

الآن في حدث التحميل للفورم ضع هذا الكود الذي يأمر المؤقت بالبدئ :

```
Timer1.Start()
```

الأن يمكنك إيقاف المؤقت عن العمل عن طريق إستخدام الأمر Stop:

```
Timer1.Stop()
```

الوراثة Inheritance

تعتبر الوراثة من أهم مواضيع أي لغة برمجة غرضيه التوجه (OOP) ومنها لغتنا VB.NET ويكمن أهمية هذا الموضوع لأسباب عديدة من أهمها القدرة على إعادة استخدام الكود الذي كتبته من قبل، دون الحاجة لإعادة كتابة الكود من جديد كلما تغيرت جزئيّة صغيرة في الكود الذي تعاملت معه مسبقاً , لفهم هذه التقنية البالغة الأهمية ينبغي التعرف والمعرفة الجيدة لمفهوم الوراثة كما يلي :

- الوراثة Inheritance /

هي مقدرة خلية (يطلق عليها الخلية الفرعية Sub Class أو الخلية المشتقة Derived Class) على أن ترث كل خصائص ووسائل خلية أخرى (يطلق عليها الخليّة الأم Parent Class أو الخليّة الأساسيّة Base Class) ، مع إمكانيّة احتوائها على الجديد وكذلك إمكانية تطوير ما ورثته ليتم استبدالها بأخرى معدّلة من خلال تقنية إعادة التعريف OverLoads .

- أعتقد بأن مفهوم الوراثة قد اتضح نوعاً ما كخطوة أولى ولتوضيح ذلك أكثر دعونا نتعرف باختصار على بعض أهم المزايا التي نجنيها من استخدام تقنية الوراثة :

- إمكانية تطوير الخلية الأساسية (Base Class) من خلال إضافة دوال وإجراءات ومتغيرات وأحداث جديدة لم تكن متواجد في الخلية الأساسية وكذلك إمكانية إعادة تعريف ما كان متواجد في الخلية الأساسية وذلك بعد اشتقاقها وراثياً من الفئة الأساسية.
- إذا أكتشف بأن هنالك خطأ موجود في الخلية الأساسية وهذا الخطأ ظهر في جميع الخلايا المشتقة فيمكن معالجة هذا الخطأ بسهوله كبيرة من خلال معالجة الخلية الأساسية وبالتالي معالجة الأخطاء في جميع الخلايا المشتقة (Derived Classes) .
- الوراثة تجنب الخلية المشتقة من إعادة تعريف لكل ما هو موجود في الخلية الأساسية (سواء كانت متغيرات , دوال , إجراءات , أحداث , ... الخ) , أما إذا كان لديك تعريفات جديدة فقم بتعريفة في الخلية المشتقة لتكون خاصة بهذه الخلية.
- إمكانية تطوير الخلايا الموجودة في أي ملف DLL, وذلك بعد إضافة الملف (كـ References) إلى برنامجك .
- كما أنه لا يخفى عليك بأن جميع الفئات المعرفة في مكتبة فئات إطار عمل NET Framework. تستخدم تقنية الوراثة.
- خليتك الجديدة نفسها قد يرثها شخص آخر ويستخدمها لإنشاء خليّة ثالثة، ... وهكذا!

ملاحظة /

- لإنشاء (Constructor) فئة جديدة إستخدم الكلمة المحجوزة New .
- لتدمير (Destructor) الفئة نهائياً إستخدم الدالة Dispose .

➢ وراثة أحدى الفئات الموجودة ضمن مكتبة فئات إطار عمل .NET FRAMEWORK

هنا سنقوم بإنشاء أداة خاصة بنا (باسم MyTextBox) تمثل هذه الأداة أداة TextBox وذلك بعد إشتقاقها وراثياً من الفئة TextBox , ومن ثم تطوير هذه الأداة لتشمل أشياء إضافية لم تكن موجودة في الخلية الأساسية TextBox , إذا فلنبدأ بكتابة مثال بسيط يقوم بما سبق :-

❖ المرحلة الأولى (مرحلة الإنشاء):

- قم بإضافة Class باسم MyTextBox.
- اكتب السطر التالي الذي يقوم بتوريث الخلية الجديدة (MyTextBox) لكل ماهو موجود في الخلية الأساسية (TextBox) وذلك باستخدام الكلمة المحجوزة Inherits ليصبح لديك الكود التالي :-

```
Public Class MyTextBox
    Inherits TextBox  ' TextBox عملية الوراثة من الخلية الأساسية
End Class
```

وهنا فسنجد أن الخلية الجديدة (MyTextBox)تعمل بالضبط كالخلية الأساسية(TextBox) .

- قم بإضافة الأداة الجديدة (MyTextBox) إلى النموذج وستجد بالفعل بأن هذه الأداة تشبه 100% الأداة TextBox الاعتيادية .

❖ المرحلة الثانية (مرحلة التطوير (إنشاء دالة)) :

- لكي نجعل أداتنا الجديدة أكثر تطوراً بحيث تمتلك بعض الخصوصيات الخاصة بها على سبيل المثال دالة جديدة (باسم Reverse) تقوم بعكس النص Text الموجود في داخل الأداة (MyTextBox) , لذا دعونا نعود إلى خليتنا السابقة ونقوم بتعديل الشفرة الموجودة في داخلها لتصبح بالشكل التالي :

```
Public Class MyTextBox
    Inherits TextBox  ' TextBox عملية الوراثة من الخلية الأساسية

    Public Function Reverse(ByVal b As Boolean) As Boolean
        If b = True Then
            Dim S As String = MyBase.Text
            MyBase.Text = ""
            Dim i As Short = S.Length - 1
            Do While i >= 0
                MyBase.Text &= S(i)
                i -= 1
            Loop
        End If
    End Function
End Class
```

```
طريقة أخرى '
If b Then
    Dim X() As Char = Me.Text
    Array.Reverse(X)
    Me.Text = X
End If
```

- عود إلى النموذج السابق وقم بإنشاء زر(Button ولتكن خاصية الـ Text لهذا الزر هي " عكس النص").
- إكتب في داخل هذا الزر الكود التالي :-

```
MyTextBox1.Reverse(True)
```

- قم بعملية التنفيذ وستتضح لك وظيفة هذا الدالة الجديدة التي أضافت لك شيء جديد لأداتك الجديدة .

❖ **المرحلة الثالثة (مرحلة التطوير (إنشاء خاصية جديدة)) :**

- **مرة أخرى نحتاج لتطوير أداتنا الجديدة MyTextBox بحيث تحوي خاصية جديدة ولتكن CreatedDate حيث يتم من خلال هذه الخاصية إعطاء تأريخ للإنشاء وهنا نقوم بإضافة هذه الخاصية للأداتنا الجديدة ومن خلال تعديل الشفرة السابقة بحيث يضاف إليها ما يلي :-**

```
Private MyDate As Date ' متغير يحمل قيمة الخاصية

Property CreatedDate () As Date
    Get      ' دالة الحصول على القيمة
        Return (MyDate)
    End Get
    Set(ByVal value As Date)  ' دالة إعطاء قيمة إلى الخاصية
        MyDate = value
    End Set
End Property
```

- **والآن يمكنك العودة إلى النموذج وإنشاء زر (Button ولتكن خاصية Text هي " تغيير التأريخ") وإكتب بهذا الزر الكود التالي :**

```
MyTextBox1.CreatedDate = Now    ' لتغيير قيمة الخاصية
MsgBox(MyTextBox1.CreatedDate) ' للحصول على قيمة الخاصية
```

- **قم بعملية البناء ومن ثم التنفيذ وستتضح لك وظيفة هذا الخاصية الجديدة التي أضافت لك شيء جديد لأداتك الجديدة .**

❖ **المرحلة الرابعة (مرحلة التطوير (إنشاء حدث جديدة)) :**

- **مرة أخرى نحتاج لتطوير أداتنا الجديدة MyTextBox بحيث تحوي حدث جديدة وليكن msEvent حيث يتم من خلال هذا الحدث القيام بأي شئ وهنا نقوم بإضافة هذا الحدث للأداتنا الجديدة ومن خلال تعديل الشفرة السابقة بحيث يضاف إليها ما يلي :-**

```
Event msEvent(ByVal ms As String)
```

- **الأن قم بعملية البناء Build وإنتقل إلى محرر الشفرة وسترى الحدث الجديد msEvent موجود في أداتنا الجديد MyTextBox ليضيف شيء جديد إلى هذه الأداة .**

❖ المرحلة الخامسة (مرحلة إعادة التعريف Overloads) :

- الأن ماذا لو لم تكن مقتنعاً بعمل دالة أو خاصية معينة موجودة في أداتنا MyTextBox على سبيل المثال : وكما عرفنا في المحاضرات السابقة (صفحة 57) بأن الخاصية WordWarp الخاصة بالتفاف الأسطر لها قيمتين هما (True,False) فماذا لو أردنا تغيير قيمة هذه الخاصية أو بمعنى أصح إعادة تعريف هذه الخاصية لتصبح في أداتنا الجديدة MyTextBox لها قيمة وحيدة وهي (يعمل أو Working) وبالتالي التخلص من القيمة المنطقية التي كانت موجودة بالسابق , لذا ومع مثل هذه الحالة فلا مفر من إستخدام الكلمة المحجوزة Overloads (إعادة تعريف) ومن أجل ذلك لنقم بعملية إعادة التعريف للخاصية WordWarpلتحمل القيمة Working لذا ركز جيداً في الشفرة المضافة التالية في أداتنا MyTextBox:-

```
Dim myVarWordWarp As String ="Working"      ' متغير يحمل قيمة الخاصية
Overloads Property WordWarp() As String
       Get
           Return myVarWordWarp
       End Get
      Set(ByVal value As String)
           myVarWordWarp = value
      End Set
End Property
```

- قم بعملية البناء ومن ثم إنتقل إلى الخاصية WordWarp لتجدها تحمل القيمة الوحيدة Working .

❖ المرحلة السادسة (تحويل الفئة الجديدة إلى شكل ملف DLL) :

الأن وفي هذه المرحلة الأخيرة التي نختم بها هذا الجزء من هذا الموضوع الأكثر من مهم فماذا لو أردت توزيع فئاتك وأدواتك الجديدة بحيث تحول هذه الفئات إلى ملفات ومن النوع DLL بحيث يمكن إستخدامها في التطبيقات والبرامج المختلفة , وللقيام بذلك فكل ما يتطلب منك هو إنشاء نفس الفئة أو الخلية الجديدة التي انشائناها وبنفس الاسم ولكن الاختلاف الوحيد في نوع المشروع حيث أنه من الضروري أن يكون نوع المشروع هو Library Class وبعد عملية البناء فإنك ستجد بأن ملف من النوع DLL وبنفس التسمية (MyTextBox) قد أنشئ بداخل المجلد Bin التابع للمشروع . لذا وبعد حصولك على هذا الملف يمكن توزيع هذا الكلاس أو الخلية وإمكانية تطويره واستخدامه في التطبيقات المختلفة .

- **وأخيراً: قم بإنشاء بعض من الخصائص والدوال والأحداث لأدواتك الجديدة بما يلبي احتياجاتك .**

➢ الوراثة بين الفئات المختلفة

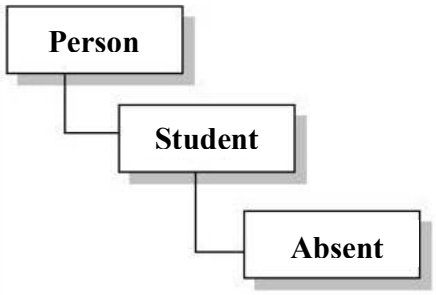
إذا رجعنا إلى الخلف قليلاً وبالضبط إلى الفئة Person في مثالنا (صفحة 37) وقمنا بإنشاء هذه الفئة الأساسية , ومن ثم قمنا باشتقاق هذه الفئة من فئة جديدة باسم Student تمتلك إضافة لما هو موجود في الفئة الأساسية الأعضاء (Grade , Std_Uni_No) لتصبح لدينا الفئتين وبالشكل التالي :

```
'الفئة الأساسية
Public Class Person
    Public Name As String
    Public Age As Byte
End Class

'الفئة المشتقة من الفئة Person
Public Class Student
    Inherits Person
    Public Std_Uni_No As String
    Public Grade As Integer
End Class

'الفئة المشتقة من الفئة Student
Public Class Absent
    Inherits Student
    Public Abs_Date As Date
End Class
```

وبالتالي وصلنا إلى العلاقة التالية :



شكل يوضح العلاقة الوراثية بين الفئات.

وهنا سنجد بأن الكائنات المنشئة من الفئة Absent يمكنها الوصول إلى كل الأعضاء الموجودة في الفئتين Person , Student وهذا منطقي جداً .لذا سنجد بأننا لو كتبنا الشفرة التالية فسنجدها صحيحة :

```
Dim A1 As New Absent
'أعضاء الفئة الأساسية Person
A1.Name = "Ali" : A1.Age = 20
'أعضاء الفئة المشتقة Student
A1.Std_Uni_No = 200404130108 : A1.Grade = 1
'أعضاء الفئة المشتقة Absent
A1.Abs_Date = #1/12/1987#

MsgBox(A1.Name & " " & A1.Age & " " & A1.Std_Uni_No & " " & A1.Grade & " " & A1.Abs_Date)
```

❖ Me , MyClass , MyBase

- **أولاً : Me و MyBase**

كما عرفنا سابقاً بأن المؤشر Me يشير إلى الفئة الحالية , وبالتالي فإن المؤشر MyBase سوف يشير إلى أعضاء الفئة القاعدية . دعونا ننتقل إلى المثال التالي ليتضح الفرق بشكل أكبر :-

```
' الفئة القاعدية
Public Class BaseClass
    Sub BaseMethod()
        MsgBox("Base Class")
    End Sub
End Class
```

```
' الفئة المشتقة
Public Class DerivedClass
    Inherits BaseClass
    Sub DerivedMethod()
        ' سيتم إستدعاء الإجراء في الفئة القاعدية
        Me.BaseMethod()
        ' سيتم إستدعاء الإجراء في الفئة القاعدية
        MyClass.BaseMethod()
        ' سيتم إستدعاء الإجراء في الفئة القاعدية ايضا
        MyBase.BaseMethod()
    End Sub
End Class
```

```
' في أي حدث لأي أداة أو نموذج
  Dim P As New DerivedClass
  P.DerivedMethod()
```

قد يبدو من الوهلة الأولى بالتشابة الكبير بين كل من Me و MyBase و MyClass كما في المثال السابق , لكن ماذا لو كان لدينا حالة إعادة تعريف Overloads هنا ستختلف النتائج لذا لنقوم ببعض التعديلات للشفرة السابقة :

```
' الفئة القاعدية
Public Class BaseClass
    Overridable Sub BaseMethod()
        MsgBox("Base Class")
    End Sub
End Class
```

```
' الفئة المشتقة
Public Class DerivedClass
    Inherits BaseClass
    Overrides Sub BaseMethod()
        MsgBox("Derived Class")
    End Sub
    Sub DerivedMethod()
        ' سيتم إستدعاء الإجراء في الفئة الحالية
        Me.BaseMethod()
        ' سيتم إستدعاء الإجراء في الفئة الحالية
        MyClass.BaseMethod()
        ' سيتم إستدعاء الإجراء في الفئة القاعدية
        MyBase.BaseMethod()
    End Sub
End Class
```

```
' في أي حدث لأي أداة أو نموذج
  Dim P As New DerivedClass
  P.DerivedMethod()
```

المثال السابق يوضح الفرق بين المؤشرين Me و MyBase وبشكل واضح جداً .

- **ثانياً : Me و MyClass**

أما عن المؤشرين Me و MyClass وكما عرفنا سابقاً بأن المؤشر Me يشير إلى الفئة الحالية وعند إعادة التعريف Overrides لأي عضو (دالة و خاصية و ...) فإننا سنجد بأن العضو الذي تم إعادة تعريفة هو الذي سيتم استدعائه وليس العضو في الفئة الحالية ولفهم ذلك جيداً انظر المثال التالي :

```
' الفئة القاعدية
Public Class BaseClass
    Overridable Sub BaseMethod()
        MsgBox("Base Class")
    End Sub
    Sub TestMethod()
        ' سيتم إستدعاء الإجراء في الفئة المشتقة
        Me.BaseMethod()
        ' سيتم إستدعاء الإجراء في الفئة القاعدية
        MyClass.BaseMethod()
    End Sub
End Class
```

```
' الفئة المشتقة
Public Class DerivedClass
    Inherits BaseClass
    Overrides Sub BaseMethod()
        MsgBox("Derived Class")
    End Sub
End Class
```

```
' في أي حدث لأي أداة أو نموذج
Dim P As New DerivedClass
P.TestMethod()
```

المثال السابق يوضح الفرق بين المؤشرين Me و MyClass وبشكل واضح جداً .

❖ NotInheritable

قد تحتاج أحياناً لأي سبب كان أن تمنع أي شخص من أن يرث أو يشتق فئة عملت فيها وتعبت بها وسهرت في بناءها وهنا يمكنك القيام بذلك من خلال الكلمة المحجوزة NotInheritable :

```
Public NotInheritable Class BaseClass
      . . . . . .
      . . . . . .
End Class
```

❖ MustInherit

إستخدام الكلمة المحجوزة MustInherit في سطر تعريف الفئة تمنع تعريف كائن من نوع هذه الخلية مباشرة , ولكن يمكن إشتقاق هذه الفئة من خلال إنشاء فئة أخرى ترث الفئة السابقة , انظر المثال التالي :

```
' الفئة القاعدية
   Public MustInherit Class BaseClass
         . . . . . .
         . . . . . .
   End Class
' الفئة المشتقة
   Public Class DerivedClass
       Inherits BaseClass
   End Class
' في أي حدث لأي أداة أو نموذج
   Dim P1 As New DerivedClass  ' جملة صحيحة
   Dim P2 As New BaseClass     ' رسالة خطأ
```

❖ MustOverride

بعض الطرق والخصائص في الفئات لابد من أن يتم إعادة قيادتها لكي تتمكن من التعامل وبمرونة كبيرة مع الفئة المشتقة ، ولعل السبب الرئيسي لاستخدامنا الكلمة المحجوزة MustOverride هو اختلاف نوع البيانات واختلاف طريقة عرضها في أكثر من فئة لذلك أضفنا الكلمة المحجوزة MustOverride حتى تجبر مستخدم الفئة من إعادة تعريف الطريقة لحظة اشتقاقها:

```
الفئة القاعدية
Public MustInherit Class BaseClass
    MustOverride Sub ShowDate()
End Class
الفئة المشتقة
Public Class DerivedClass
    Inherits BaseClass
    Public Overrides Sub ShowDate()
      . . . . . .
    End Sub
End Class
```

❖ محددات الوصول Access Specifiers

من المواضيع بالغة الحساسية هو قابلية الوصول والتعامل مع الأعضاء والفئات وهذا ما نطلق علية محددات الوصول , وبالتالي هنا نمتلك خمس كلمات محجوزة تستخدم كمحددات وصول ولكل كلمة منها وظيفة بالتأكيد تختلف عن الأخرى , إذاً دعونا نلقي الضوء على هذا المحددات وكما يلي :

- **Private (خاص)**

إذا حدد محدد لأي عضو من النوع Private أو Dim فهذا يعني أنة قد حدد مستوى حماية خاص لهذا العضو بحيث لا يمكن رؤية أو التعامل مع هذا العضو إلا من داخل هذه الفئة (Class) أو الجزء (Block) التي عرف فيها فقط .

- **Public (عام)**

إذا حدد محدد لأي عضو من النوع Public فهذا يعني أنة قد حدد مستوى حماية عام لهذا العضو بحيث يمكن رؤية أو التعامل مع هذا العضو من داخل المشروع الذي عرف فيه أو من داخل أي مشروع أخر , كما أن جميع أعضاء الفئة تكون Public كمستوى حماية افتراضي .

- **Friend (صديق المشروع)**

إذا حدد محدد لأي عضو من النوع Friend فهذا يعني أنة قد حدد مستوى حماية صديق المشروع لهذا العضو بحيث يمكن رؤية أو التعامل مع هذا العضو من داخل المشروع الذي عرف فيه فقط أما من داخل أي مشروع أخر فهذا مرفوض حتى وإن تم إضافة مرجعية لهذا المشروع . كما أن جميع الفئات تكون Friend كمستوى حماية افتراضي .

- **Protected (محمي)**

إذا حدد محدد لأي عضو من النوع Protected فهذا يعني أنة قد حدد مستوى حماية محمي لهذا العضو بحيث يعامل معاملة العضو من المستوى الخاص (Private) ولكن الفرق الوحيد هو في أن العضو ذو مستوى الحماية Protected يمكن الوصول إليه من داخل الفئة المشتقة Derived Class أما التعامل فهو يشبه التعامل مع الـ Private .

- **Protected Friend (محمي وصديق)**

إذا حدد محدد لأي عضو من النوع Protected Friend فهذا يعني أنة قد حدد مستوى حماية مختلط (محمي + صديق) لهذا العضو بحيث يعامل معاملة العضو من المستوى صديق (Friend) ومضاف إليه إمكانية الوصول من خارج المشروع في حالة الوراثة .

Example :

```
Public Class Class1
    Dim X1 As Byte
    Private X2 As Byte
    Protected X3 As Byte
    Public X4 As Byte
    Friend X5 As Byte
    Protected Friend X6 As Byte
    Sub Test1()
        Dim X7 As Byte
    End Sub
    Sub Test2()
        X1 = 1
        X2 = 1
        X3 = 1
        X4 = 1
        X5 = 1
        X6 = 1
        X7 = 1
    End Sub
End Class
```

- ممكن لأنه من النوع خاص ولكن على مستوى الفئة الأولى (Dim)
- ممكن لأنه من النوع خاص ولكن على مستوى الفئة الأولى (Private)
- ممكن لأنه من النوع محمي وهو يعامل معاملة الخاص ويمكن إستخدامة في الفئة المشتقة (Protected)
- ممكن لأنه من النوع عام ويمكن إستخدامه في أي مكان وأي مشروع(Public)
- ممكن لأنه من النوع صديق ويمكن إستخدامه في أي مكان على مستوى المشروع الواحد (Friend)
- ممكن لأنه من النوع صديق و محمي ويستخدامه في أي مكان على مستوى المشروع الواحد أو في الفئة المشتقة حتى وإن كانت في مشروع أخر (Protected Friend)
- غير ممكن لأنه من النوع خاص وعلى مستوى الطريقة Test 1 (Dim)

```
Public Class Class2
    Inherits Class1
    Sub Test3()
        X1 = 1
        X2 = 1
        X3 = 1
        X4 = 1
        X5 = 1
        X6 = 1
        X7 = 1
    End Sub
End Class
```

- غير ممكن لأنه من النوع خاص ولا يمكن إستخدامة إلى داخل الفئة الأولى .
- غير ممكن لأنه من النوع خاص ولا يمكن إستخدامة إلى داخل الفئة الأولى .
- ممكن لأنة من النوع محمي ويمكن إستخدامة في داخل الخلية المشتقة فقط .
- ممكن لأنة من النوع عام ويمكن إستخدامة في أي مكان وفي أي مشروع .
- ممكن لأنة من النوع صديق ولأن الفئة الثانية موجودة في نفس المشروع .
- ممكن لأنة من النوع صديق و محمي ولأن الفئة الثانية موجودة في نفس المشروع.
- غير ممكن لأنه من النوع خاص وعلى مستوى الطريقة Test 1 في الفئة الأولى فقط.

في أي مكان في أي إجراء أو حدث ١

```
Dim C1 As New Class1
C1.X1 = 1
C1.X2 = 1
C1.X3 = 1
C1.X4 = 1
C1.X5 = 1
C1.X6 = 1
C1.X7 = 1
```

- غير ممكن لأنه من النوع خاص ولا يمكن إستخدامة إلى داخل الفئة الأولى .
- غير ممكن لأنه من النوع خاص ولا يمكن إستخدامة إلى داخل الفئة الأولى .
- غير ممكن لأنة من النوع محمي ولايمكن إستخدامة إلا في داخل الخلية المشتقة فقط .
- ممكن لأنة من النوع عام ويمكن إستخدامة في أي مكان وفي أي مشروع .
- ممكن لأنة من النوع صديق ويمكن إستخدامة في أي مكان في نفس المشروع .
- ممكن لأنة من النوع صديق و محمي ويمكن إستخدامة في أي مكان في نفس المشروع
- غير ممكن لأنه من النوع خاص وعلى مستوى الطريقة Test 1 في الفئة الأولى فقط.

❖ **الفرق بين ByVal و ByRef**

من الأمور بالغة الأهمية في بناء الدوال والإجراءات فهم الفرق بين الكلمتين المحجوزتين (Keywords) والتي تستخدم في تمرير الوسيطات Parameters وتعني الكلمتين على التوالي بالقيمة (ByValue) وبالمراجع (ByReference), كما أن كتابة الوسيط بدون أي ممرر يعني اعتباره من النوع ByVal وهذا ما سنجده يكتب بشكل تلقائي , إذا ما أهمية الكلمتين السابقتين وما هو الفرق بينهما , لعل المثال التالي يناقش الفرق بينهما :

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
                       System.EventArgs) Handles MyBase.Load
        Dim X As Byte = 3
        MsgBox(" X Value = " & X)       ' 3 تطبع القيمة الأصلية
        MyValProcedure(X)
        MsgBox(" X Value ByVal= " & X) ' 3 لاتؤثر على القيمة الأصلية يطبع
        MyRefProcedure(X)
        MsgBox(" X Value ByRef= " & X) ' 5 تؤثر على القيمة الأصلية يطبع
    End Sub

' BYVal إجراء يتعامل مع
    Sub MyValProcedure(ByVal X As Byte)
        X += 2
      'القيمة تتغير في داخل الإجراء فقط ولا تؤثر على القيمة الأصلية
    End Sub
' BYVal إجراء يتعامل مع
    Sub MyRefProcedure(ByRef X As Byte)
        X += 2
      'القيمة تتغير في داخل الإجراء وتؤثر على القيمة الأصلية
    End Sub
```

خلاصة :

- **ByVal : تعني ByValue وبالتالي ومع هذه الـ Keyword يتم إنشاء متغير جديد يحمل القيمة المرسلة وفي موقع جديد بالطبع في الذاكرة وبالتالي فإن التغيير إن تم يكون في الإجراء فقط , أما القيمة الأصلية فتبقى كما هي .**
- **ByRef : تعني ByReference وبالتالي ومع هذه الـ Keyword يتم إرسال مرجع إلى الإجراء بحيث يشير هذا المرجع إلى موقع المتغير المرسل في الذاكرة لذلك إذا تغيرت قيمة هذا الموقع يعني تغيير القيمة لهذا المتغير بشكل عام .**

Microsoft SQL SERVER

هي عبارة عن قاعدة بيانات مركزية تقوم بإدارة قواعد البيانات وتوزيعها عبر شبكات الحاسوب .

ولقد بدء ظهور قواعد البيانات المركزية بشكل مكثف في نهاية الثمانينات ،فلقد كان الجميع قبل ذلك يستخدم البرامج التي صنعت بلغات البرمجة العادية التي كانت تخزن بياناتها في ملفات خاصة بها ، والمشكلة الأساسية في تلك البرامج كانت محدودية استخدام البيانات ، اعني انك لا تستطيع الاستعلام عن البيانات بطريقة أخرى غير الطريقة التي صمم بها البرنامج ومن هنا بدأت فكرة قواعد البيانات عامة ، فبإمكان المستخدم فتح ملف البيانات عن طريق برنامج الاستعلام الرئيسي الذي يأتي مع قاعدة البيانات وإجراء كل الاستعلامات التي يحلم بها ومن هنا ظهرت قواعد البيانات وظهرت أيضا لغة SQL المخصصة للاستعلام في قواعد البيانات ، وبدأت تتطور وانتقلت العديد من الشركات لاستخدامها ، نظرا لسهولة التعامل معها وسرعة برمجتها ولكن مع زيادة حجم المؤسسات وبداية ظهور شبكات الكومبيوتر أصبحت قواعد البيانات بحاجة إلى أن تعمل على أكثر من جهاز في نفس الوقت ، فتطورت برامج إدارة قواعد البيانات وأصبحت قادرة على فتح نفس الملفات المخزنة في الجهاز المركزي من عدة أجهزة كومبيوتر في نفس الوقت وهنا بدأت بعض المشاكل بالظهور ولقد حلت بعض هذه المشاكل وليس كلها لأنه مع زيادة حجم البيانات وزيادة عدد الأجهزة المتصلة بالشبكة أصبح صعب إدارة ملفات قاعدة البيانات المخزنة على الجهاز المركزي ، كما أن أمنها كان معرض للخطر دائما ، فبإمكان الجميع الوصول إلى الملف المركزي الذي يحتوي على البيانات ويعبث به ، أو حتى أن يصل إلى بيانات لا صلاحية له باستخدامها كما أن الاستعلامات المتزايدة على قواعد البيانات زادت من الضغط على الشبكة فكما تعلم يتطلب الاستعلام عن شخص ما البحث في كل قاعدة البيانات حتى إيجاده بها , وبالتالي الضغط على الشبكة بالحصول على المعلومات المطلوبة ، طبعا الشبكة قد تتحمل طلب أو طلبين معا ، ولكن ماذا بالنسبة للبنوك مثلا ، هناك ألاف السجلات وعشرات العمليات في نفس الوقت ولذلك بدأت الحاجة إلى تطوير قواعد البيانات العادية , والانتقال إلى قواعد البيانات المركزية .

ومن هنا ظهرت قواعد البيانات المركزية التي يمكن فهمها بشكل جيد من خلال أهم النقاط التالية :

- فهي عبارة عن كيان متواجد بداخل الجهاز المركزي وتخزن البيانات فيه , ولكن الاختلاف بينها وبين الأنواع الأخرى من قواعد البيانات هو في أن التعامل مع البيانات لا يتم إلا من خلال البرنامج الذي يعمل في الجهاز المركزي والذي يسمى محرك قواعد البيانات المركزية . ومن النقطة السابقة يكون قد تم فصل المستخدم النهائي عن الملف الرئيسي لقواعد البيانات فلو كنت بحاجة إلى استعلام معين ، فسيقوم برنامجك بطلب ذلك الاستعلام من محرك قواعد البيانات المركزية الموجود في الجهاز المركزي ، حيث بدوره سيقوم هو بالاستعلام ومن ثم يعطي النتيجة فقط للجهاز العادي الذي طلب الاستعلام وبذلك يكون قد أنهى كابوس إغراق الشبكة بالبيانات .
- كما أن محرك قواعد البيانات المركزية مسئول عن حماية البيانات ، عكس ما كان من قبل حيث كان الجميع يستطيع الوصول إلى كل البيانات المخزنة ، ولكن باستخدام النظام الجديد أصبح فقط من لديهم صلاحية الوصول قادرين على ذلك ، كما أصبح بإمكان مدير الشبكة إعطاء صلاحيات مختلفة للمستخدمين ، فقد يمنع مستخدم من إضافة بضاعة جديدة على جدول البضاعة ويسمح له بالنظر عليها فقط، وقد يمنعه نهائيا من الوصول إلى جدول معين، أو فقط على جزء من جدول .
- كما أن محرك قواعد البيانات المركزية أصبح يقوم بعمليات النسخ الاحتياطي والحفاظ على البيانات من التلف أوتوماتيكياً، وذلك بفحصها باستمرار ونسخها على أشرطة النسخ الاحتياطي وإعلام مدير النظام بأية مشاكل صغيرة بداخلها وهكذا استمر تطور قواعد البيانات المركزية إلى يومنا هذا .
- من أشهر قواعد البيانات المركزية وأكثرها استخداما التي ظهرت حتى الآن في الأسواق :

1. Oracle 8i , 9i
2. SQL Server 2000 , 2005
3. IBM DB2
4. Sybase
5. Informix
6. Borland IntraBase

Microsoft SQL Server And Microsoft Access

الميزة	Microsoft SQL Server	Microsoft Access
حجم قاعدة البيانات	اكبر من 1 تيرابايت (1024 جيجا)	2 جيجابايت
عدد الجداول	2,147,483,647 (اكثر من ٢ بليون)	32,768
عدد المستخدمين	غير محدود	255 مستخدم حكد أقصى
الأمان	مدمج مع أمان Windows 2000	تعتمد على مجموعة العمل او المبرمج
اداوت تحليل البيانات	مدعومة	غير مدعومة
دعم انظمة SMP	مدعومة	غير مدعومة
اصلاح مشاكل قاعدة البيانات	اعادة قاعدة البيانات الى اي وقت تختاره	تعتمد على النسخ الاحتياطية التى قمت بعملها
النسخ الاحتياطية	مدعومة	غير مدعومة
صلاحيات المستخدمين	دعم كامل من صميم SQL Server	تعتمد على المبرمج

ما هي انواع البيانات التى يمكن التعامل معها في الـ SQL SERVER 2005

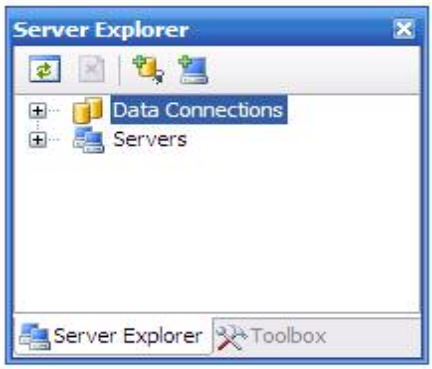
النوع	القيمة المخزنة	المدى	معلومات عن النوع
char	حرفي	1 إلى 8000 حرف	وهو عبارة عن نوع يجعل حقل البيانات يتقبل عدد معين من الحروف , ويحجز نفس الخانات في الذاكرة , حتى وإن أدخل إليها أقل .
nchar	حرفي	1 إلى 4000 حرف	وهو أيضا يأخذ حروف ولكن حروف من النوع Unicode أي يمكنة تخزين حروف كل اللغات وليس الإنجليزية فقط , ولكن يحمل نفس عيب الـ Char .
varchar	حرفي	1 إلى 8000 حرف	نطلق عليه نوع مطاطي بيغير الحجم المخزن على حسب عدد الحروف المخزنة فيه ، ولكن أقصى عدد حروف بياخده هو ما حددته أنت .
varchar(max)	حرفي	1 إلى 8000 حرف	يعني كأنك كتبت بالشكل .. varchar(8000) .. بس لو دخلت مثلا كلمة مكونة من 10 حروف فقط فسيخزنها في مكان يتسع لـ 10 حروف فقط وليس 8000 .
nvarchar	حرفي	1 إلى 4000 حرف	نفس قصة varchar بس بياخد حروف Unicode (أي لغة).
nvarchar(max)	حرفي	1 إلى 4000 حرف	نفس قصة varchar(max) بس بياخد حروف Unicode .
text	حرفي	2 مليار حرف	ولكن تخزين هذا الكم الهائل من البيانات داخل خلية بياناتك ومع إستخدامه بكثرة يزيد من بطء قاعدة بياناتك .. لذلك لو ستدخل بيانات بحجم كبير جدا فنصيحتى أن تخزنه في ملف خارجي أفضل..
ntext	حرفي	1 مليار حرف	مثل الـ النوع text بس بيخزن حروف بالنظام Unicode .
image	نظام ثنائي	2 مليار بت	مثل النوع text بالضبط بس بيخزن بيانات بالبينري .. مثل الصور والصوت والأفلام ..
binary	نظام ثنائي	8000 بايت	هذا النوع بيخزن بيانات بالبينري.. زي الصور والصوت والأفلام .

varbinary	نظام ثنائي	8000 بايت	نفس النوع binary لكن من النوع المطاطي الذي يحجز حجم ما يحتويه فقط .. " يوفر مساحة ولكن سرعة العمل معه تكون أقل من الـ binary "
varbinary(max)	نظام ثنائي	8000 بايت	يأخذ كل الـ 8000 بايت ويأخذ منها ما يتم تخزينه به فقط ..
bit	نظام ثنائي	0 أو 1	يستخدم لتخزين ناتج العملية البرمجية .. 1 أي صحيحة ، صفر أي خاطئة.
int	أرقام صحيحة	2,147,483,648 - إلى +2,147,483,647 يعني 2 بليون وشوية	أرقام صحيحة ولا تقبل الكسور . والنوع int بياخد مساحة 4 بايت .
smallint	أرقام صحيحة	32,768 – حتى الرقم 32,767	مثل النوع int بالضبط بس بيقبل من أرقام أصغر , وبياخد مساحة 2 بايت.
tinyint	أرقام صحيحة	من الرقم . حتى الرقم 255	وهو أصغر نوع في أنواع الأعداد الصحيحة وبياخد مساحة 1 بايت .
bigint	أرقام صحيحة	من الرقم – 9,223,372,036,854,775,808 حتى الرقم 9,223,372,036,854,775,807	وهو أكبر نوع في أنواع الأعداد الصحيحة , وبياخد مساحة 8 بايت .
decimal أو numeric	أرقام عشرية	من الرقم – 0.00000000000000000000000000000000000001 حتى الرقم 10,000,000,000,000,000,000,000,000,000	بيقبلوا يعني 38 صفر .. يمكن أن يحمل أرقام صحيحة أو كسرية .. بس هنا أنت بتاخد إلي أنت محتاجة من الرقم.. يعني لو تحتاج الرقم يكون مكون من 6 خانات للأرقام الصحيحة وخانتين للأرقام العشرية .. حتكتب نوع بياناتك decimal(8,2) . ويمكنك تحويل الرقم لرقم صحيح بإختيار . مثلا : decimal(8,0) يعني محتاج رقم مكون من 8 خانات وبدون كسور . أو ممكن كله كسور .. مثلا decimal(8,8) ..
float	أعداد حقيقية	من الرقم – 1.79 أس 308 إلى الرقم 1.79 أس 308 .. بقوة 15 رقم عشري	الأعداد إلي فيها كسور دقيقة.. ويستخدم في علوم الفلك والحسابات الدقيقة جدا ، . ولا تستطيع تحديد عدد الأرقام العشرية .. النوع هو إلي بيطلعها على حسب نسبة الرقم المتبقي .. وبياخد حجم 8 بايت
real	أعداد حقيقية	من الرقم –3.40 أس 38 إلى الرقم 3.40 أس 38 بقوة 7 أرقام عشرية	مثل النوع float بالضبط بس، وبياخد حجم 4 بايت .
money	عملة	من –922,337,203,685,477.5808 حتى 922,337,203,685,477.5807	وهو النوع المخصص للتعامل مع الأموال ,ويحتل مساحة 8 بايت , والنوع money يقبل حتى 4 خانات عشرية فقط .
smallmoney	عملة	من –214,748.3648 حتى 214,748.3647	مماثل للنوع money ولكن يحتل مساحة 4 بايت .
datetime	التاريخ والوقت	من تاريخ الأول من يناير 1753 حتى 31 ديسمبر 9999	وهو المسؤول عن تخزين التاريخ والوقت وهو يقبل ويخزن الساعة والدقيقة والثانية والمليثانية هكذا 00:00:00:000 ويحتل مساحة 8 بايت
smalldatetime	التاريخ والوقت	من تاريخ الأول من يناير 1900 حتى 6 يونية 2079	مماثل للنوع datetime ويخزن الساعة والدقيقة فقط ويحتل مساحة 4 بايت
xml	بيانات xml		هو نوع جديد في الـ sql server 2005 يقبل بيانات xml بكامل تنسيقها ويمكنك البحث فيه عن طريق لغة الإستعلام داخل الـ xml المسماه xquery

خطوات انشاء قاعدة بيانات SQL SERVER 2005 Express

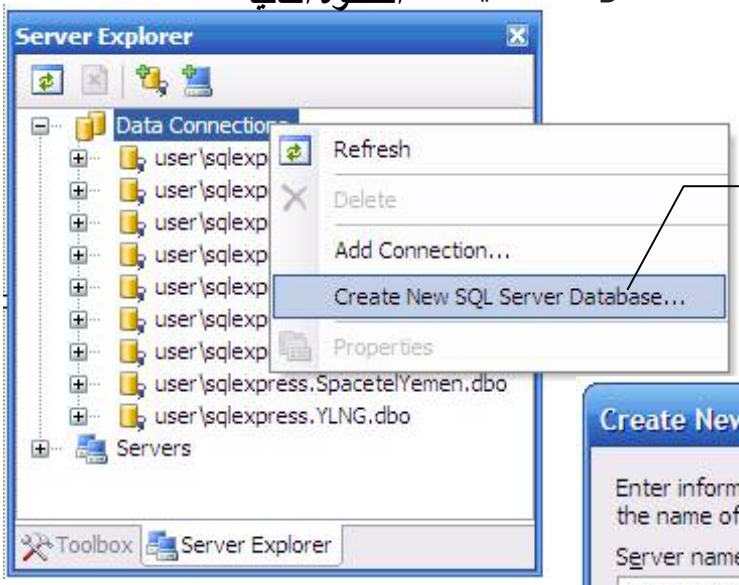
1- من صندوق Server Explorer الذي يمكن الوصول إلية من يسار بيئة الدوت نت حيث سنجد مربع حوار بنفس الاسم , وإذا لم يكن موجود فإننا سنجده في قائمة View وبنفس الاسم ايضاً .. إنظر الشكل التالى :

الخطوة الأولى



2- مقوم بإنشاء قاعدة البيانات من خلال الخطوات التالية :

الخطوة الثانية



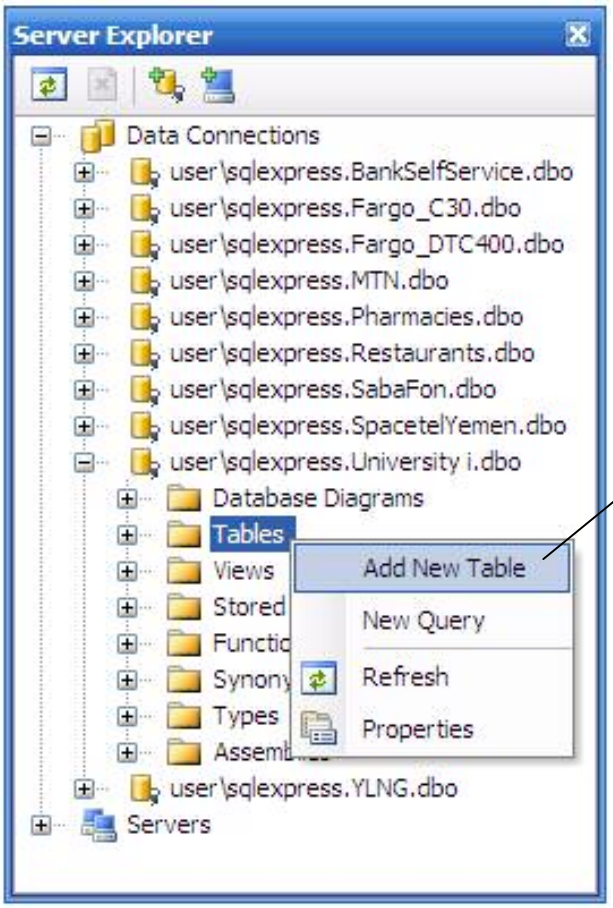
من هنا نقوم بإنشاء قاعدة بيانات جديدة .

الخطوة الثالثة

اسم الخادم ..
وهذا هو إسم الخادم للنسخة التي نعمل بها .

اسم قاعدة البيانات ..
وهذا هو إسم قاعدة البيانات التي سوف نتعامل معها

الخطوة الرابعة



نقوم بإنشاء جداول قاعدة البيانات ..
جدول الطلاب Students
جدول الأقسام Sections

dbo.Students: T...ss.University i)

Column Name	Data Type	Allow Nulls
Std_No	int	☐
Std_Name	varchar(50)	☐
Std_Age	tinyint	☑
Std_Sec_No	tinyint	☑
		☐

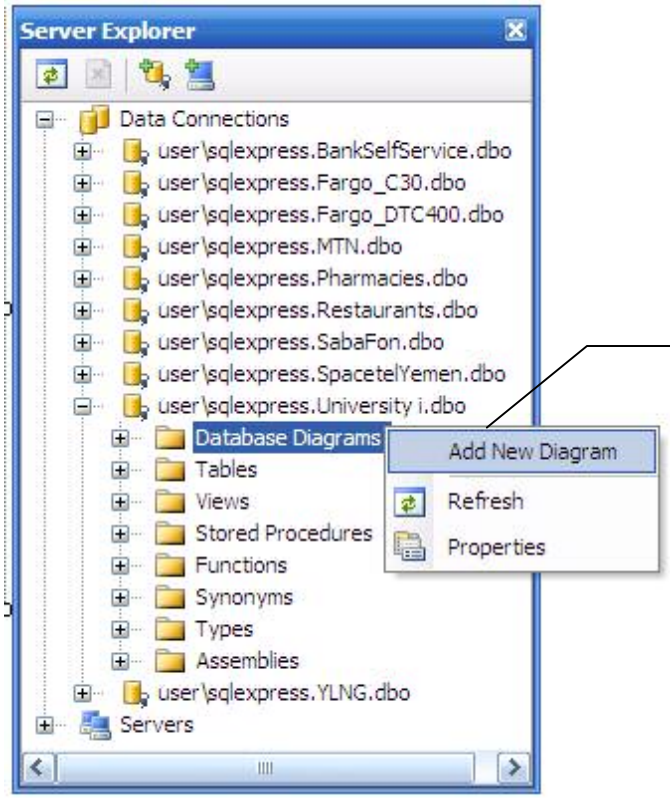
جدول الطلاب Students

dbo.Sections: T...ss.University i)

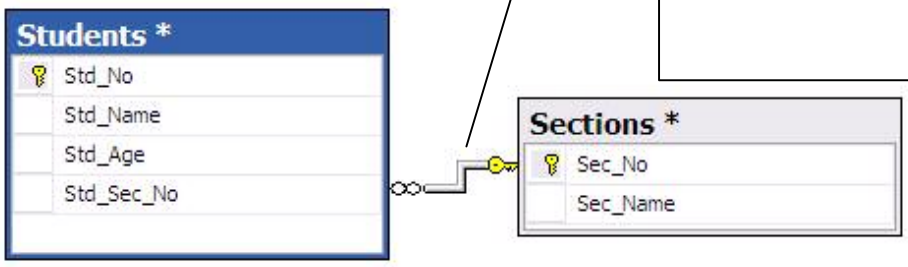
Column Name	Data Type	Allow Nulls
Sec_No	tinyint	☐
Sec_Name	varchar(50)	☐
		☐

جدول الأقسام Sections

الخطوة الخامسة



نقوم بإنشاء علاقات الجداول من هنا , وسوف نقوم بعمل علاقة 1 ------ < ∞ بين الجدولين .



نضيف الجدولين ومن ثم نستخدم طريقة السحب لبناء العلاقة ..
جدول الأقسام ---------- < Master (Sec_No)
جدول الطلاب ---------- < Detail (Std_Sec_No)

ADO.NET

نبذة عن ADO.NET (ActiveX Data Objects)

هي مجموعة من الفئات مشمولة في فضاء الأسماء System.Data غرضها الوصول إلى مصادر البيانات Data Sources تحت أنظمة قواعد بيانات متعددة الأنواع (Microsoft Access أو SQL Server أو Oracle) مما يعني قدرتك على الوصول إلى أي قاعدة بيانات مهما كانت الشركة المنتجة لها .
يطلق على هذه الحزمة من الفئات في هذا الإصدار من اللغة بـ ADO.NET 2 Or ADO.NET 2005 وما يهمنا هنا هو إن ADO.NET تقوم باستخدام مزودات البيانات للاتصال بمصادر البيانات ومن ثم جلب هذه البيانات وتعديلها وإعادتها لمصادرها وحفظها هناك . تحتوي ADO.NET على مجموعة كبيرة من الفئات، ولكن اغلبها مشتقة من خمس فئات رئيسية هي : DataSet ، DataAdapter ، DataReader ، Command ، Connection

وقبل الإنتقال إلى الخطوة التالية وتسهيلاً للتعامل مع ADO.NET يجب قبل كل شيء أن نستدعي فضاء الأسماء المناسب لمزود البيانات (Provider) الذي نتعامل معه ، والذي تندرج تحته كائناته الخاصة به للاتصال بقاعدة البيانات و ومعالجتها ، ومن مميزات استدعاء فضاء الأسماء أنه يغنينا عن كتابة مسار الكائنات التي سوف نستخدمها ، فقط نكتب اسم الكائن ..

- فعند التعامل مع قواعد بيانات Server SQL نستدعي فضاء الأسماء System.Data.SqlClient وذلك بكتابة الكود التالي في أعلى صفحة الكود :

```
Imports System.Data. SqlClient
```

وتندرج تحته مجموعة من الكائنات ولكن ما يهمنا منها كمبتدئين ما يلي :

SqlCommand , SqlConnection , SqlDataAdapter , SqlDataReader ,
, SqlException , SqlTransaction.

- وعند التعامل مع قواعد بيانات Microsoft Access نستدعي فضاء الأسماء System.Data.OleDb وذلك

```
Imports System.Data.OleDb
```

بكتابة الكود التالي في أعلى صفحة الكود :
وتندرج تحته مجموعة من الكائنات ولكن كل ما يهمنا منها كمبتدئين ما يلي :

OleDbCommand , OleDbConnection , OleDbDataAdapter , OleDbDataReader ,
, OleDbException , OleDbTransaction .

الاتصال بقواعد البيانات

للوصول إلى البيانات المخزنة في قاعدة البيانات والقراءة منها أو الكتابة فيها يجب أن نكوّن اتصال ناجح معها ، ويتم ذلك من خلال كائن الاتصال المسمى Connection ، ولتجهيز هذا الكائن نحتاج إلى أعطائه معلومات عن ملف قاعدة البيانات الذي نريد أن نتصل به (من أهمها مزود البيانات وخادم البيانات واسم ومكان قاعدة البيانات و كلمة المرور مع كلمة السر و...) ، وهذه المعلومات تكوّن لنا بما يسمى سلسلة أو كائن الاتصال (Connection String).

- **سلسلة الاتصال (Connection String)**

وهي عبارة عن مجموعة من العوامل (Parameters) الضرورية للاتصال بقواعد البيانات ، وتختلف هذه العوامل بناءً على نوع مزود البيانات الذي سوف نتحدث عنه ، ومن أهم ما يهمنا من هذه العوامل ما يلي :

- **Provider** : وهو عبارة عن اسم مزود البيانات وسوف نذكر هنا نوعين هم الأكثر انتشاراً ، النوع الأول Microsoft.Jet.OLEDB.4.0 وهو للتعامل مع قواعد بيانات برنامج Microsoft Access ، والثاني SQLOLEDB وهو مخصص للتعامل مع قواعد بيانات برنامج SQL Server و MSDAORA.1 وهو مخصص للتعامل مع Oracle .
- **Data Source** : وهو اسم خادم البيانات (أللذي يحتوي على جداول البيانات التي نريد أن نتصل بها) ، ونجد أن خادم البيانات في برنامج Microsoft Access هو اسم ملف قاعدة البيانات الذي يحمل امتداد MDB ، بينما برنامج SQL Server فهو عبارة عن اسم الـ Server المثبت على الجهاز .
- **Intial Catalog**: وهو عبارة عن اسم قاعدة البيانات الموجودة في الخادم بالنسبة لبرنامج SQL Server.
- **UserID/Password** : وهو عبارة عن اسم المستخدم وكلمة المرور لقاعدة البيانات (إن وجدت) .

```
Provider=SQLOLEDB;Data Source= ServerName ;Initial Catalog=You'reDB;
User ID=You'reUserName;Password=You'rePassword
```

سلسة اتصال بملف قاعدة بيانات برنامج SQL Server :

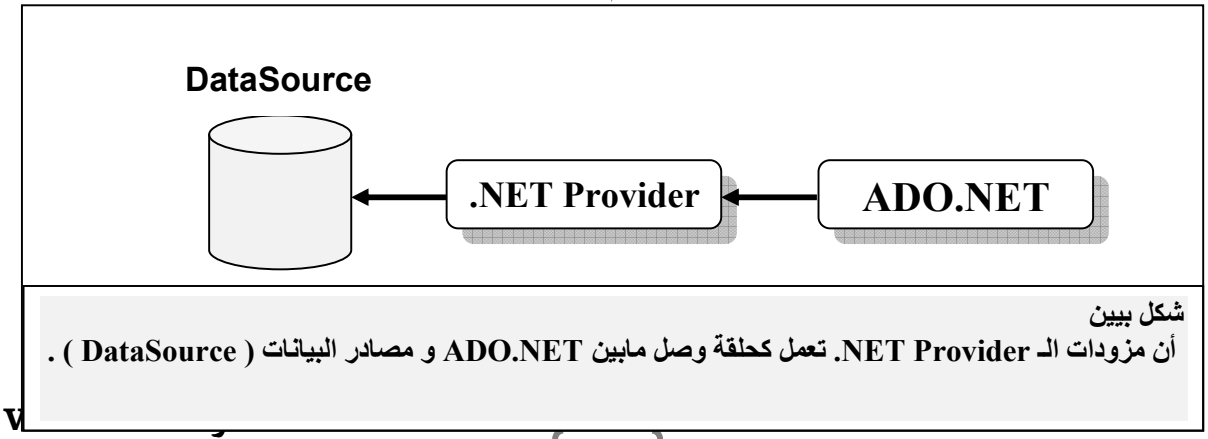
```
Provider=Microsoft.Jet.OLEDB.4.0;DataSource=c:\You'reMDB.mdb;User ID=
You'reUserName;Jet OLEDB:Database Password=You'rePassword
```

سلسة اتصال بملف قاعدة بيانات برنامج Microsoft Access :

```
Provider=MSDAORA.1;User ID=system;password =Manager
```

سلسة اتصال بقاعدة بيانات Oracle :

ملاحظة : لا يختلف التعامل مع SQL Server عن بقية أنواع قواعد البيانات الأخرى ، وكل ما عليك هو أن تستبدل عبارة Sql بعبارة OleDb.



شكل بيين
أن مزودات الـ .NET Provider تعمل كحلقة وصل مابين ADO.NET و مصادر البيانات (DataSource) .

طرق الاتصال بقواعد البيانات :

لو أردنا أن نتصل بقواعد البيانات بواسطة ADO.NET يجب أن نختار طريقة الاتصال المناسبة ، فلدينا طرقتين للاتصال وهما الاتصال المتصل (Connected Mode) و الاتصال المنفصل (Disconnected Mode) . ولكل منهما امتيازاته وعيوبه و إستراتجياته الخاصة به في العمل ، غير أن الأخير يعتبر الأمثل في التعامل مع قواعد البيانات ، وهنا بالتحديد يضيع أغلب من يرغبون تعلم برمجة قواعد البيانات من خلال ADO.NET .

الاتصال المتصل (Connected Mode)

في الوضع المتصل يتم الاتصال مع مصدر البيانات وتجري كافة العمليات فالاتصال مستمر بين شيفراتك المصدرية وبين مصدر البيانات الأساسي، وأي خلل في مصدر البيانات أو ايقاف مؤقت، سيؤدي إلى خلل في شيفراتك المصدرية.
وتتلخص فكرته في أن نقوم بإجراء اتصال مع قاعدة البيانات من خلال الكائن Connection ، وبعد قبوله نقوم بفتح الاتصال ثم نقوم بتنفيذ مجموعة من الأوامر التي نحتاجها من خلال الكائنين Command و DataReader (مثل قراءة البيانات من أجل عرضها للمستخدم (DataReader) ، أو تحديث البيانات من إضافة و تعديل وحذف وحفظ ، أو إلغاء عملية التحديث ، أو البحث عن بيانات معينة أو الإبحار والتجول في البيانات (Command)) وبعد الانتهاء نقوم بإغلاق الاتصال ، ويعتبر إغلاق الاتصال ضرورياً لان معظم مصادر البيانات تدعم عدداً محدوداً من الاتصالات المفتوحة . وهذا يعني أننا يجب أن نحافظ على الاتصال حتى ننتهي من الهدف الذي تم من أجله الاتصال .
وهو مناسب في حاله بناء التطبيقات المفردة ، أي عندما يكون للنظام مستخدم واحد وتوجد واجه المستخدم ومخزن قاعدة البيانات على نفس الجهاز , ولا يمكن استخدامه في حالة التطبيقات التي يحتاجها الكثير من المستخدمين ، ويرجع السبب في ذلك إلى أن مثل هذا النوع من الاتصال يستهلك جزءاً مهم من موارد النظام ، ويترتب علية تدني أداء هذه التطبيقات . ولأن معظم مصادر البيانات تدعم عدداً محدوداً من الاتصالات المفتوحة كما أسلفنا سابقاً . ويعد ذلك من أكبر عيوب هذا النوع بالإضافة إلى أنه يتطلب كتابة الكثير من الأكواد البرمجية كما سوف نشاهد في الجزء العملي من المحاضرة ، وذلك يعد عائق كبير أمام صيانة الكود وتحديثه من وقت لأخر .
ولعمل اتصال من هذا النوع نحتاج إلى ثلاث كائنات أساسية نستخدمها على النحو التالي :
Connection , Command , DataReader

الاتصال المنفصل (Disconnected Mode)

يعد الوضع المنفصل هو الأمثل للتعامل مع قواعد البيانات ، وذلك نظراً للإمكانيات والكائنات المتنوعة التي تمكن المستخدم من أخذ نسخة من الجداول المخزنة في قاعدة البيانات ، وقراءة البيانات المخزنة فيها ، وتعديلها و عرضها وتصفيتها ومعالجتها أو إنشاء علاقات فيما بينها ، وذلك لفترات طويلة ، دون أن تستهلك الكثير من موارد النظام ، ويترتب على ذلك عمل التطبيقات بكفاءة عاليه . ولأن معظم مصادر البيانات تدعم عدداً محدوداً من الاتصالات المفتوحة ، فإن الوضع المنفصل استطاع أن يتجاوز هذه المشكلة .

وتتلخص فكرته في أن نقوم بإجراء اتصال بقاعدة البيانات ، وذلك بواسطة كائن الاتصال Connection كما ذكرنا في السابق ، وبعد قبول الاتصال نقوم بأخذ نسخة من البيانات المخزنة في مصدر البيانات "Data Sources" (السيرفر بالنسبة لـ SQL Server و Oracle , و ملف قاعدة البيانات بالنسبة لـ Microsoft Access) ، ثم نقوم بتخزينها في الذاكرة ، وذلك من خلال كائن يكون بمثابة مستودع للبيانات ويسمى DataSet (مجموعة البيانات) ، ولكي تتم هذه العملية يحتاج هذا الكائن إلى كائن أخر يسمى DataAdapter (موفق البيانات) يكون بمثابة وسيط بين مجموعة البيانات "DataSet " ومصادر البيانات "Data Sources" ، حيث يقوم DataAdapter بعملية فتح الاتصال بقاعدة البيانات و استيراد وتصدير البيانات من وإلى مصدرها الأساسي ، وبعد تخزين البيانات في DataSet ، فإنها تصبح منفصلة عن مصدر البيانات ، لان الكائن DataAdapter يقوم بشكل تلقائي أيضاً بقطع الاتصال بقاعدة البيانات ، وهذا الانفصال والاستقلال هو السبب في تسمية هذه الطريقة بالاتصال المنفصل أو الوضع المنفصل ، وهو السر في تفوقه على الوضع المتصل ، لأنه يتغلب على جمع عيوبه ومشاكله .
من ذلك نستطيع أن نقول أن الكائنات الأساسية في الوضع المنفصل هي: Connection , DataSet , DataAdapter.

أولاً : الاتصال المتصل Connected Mode

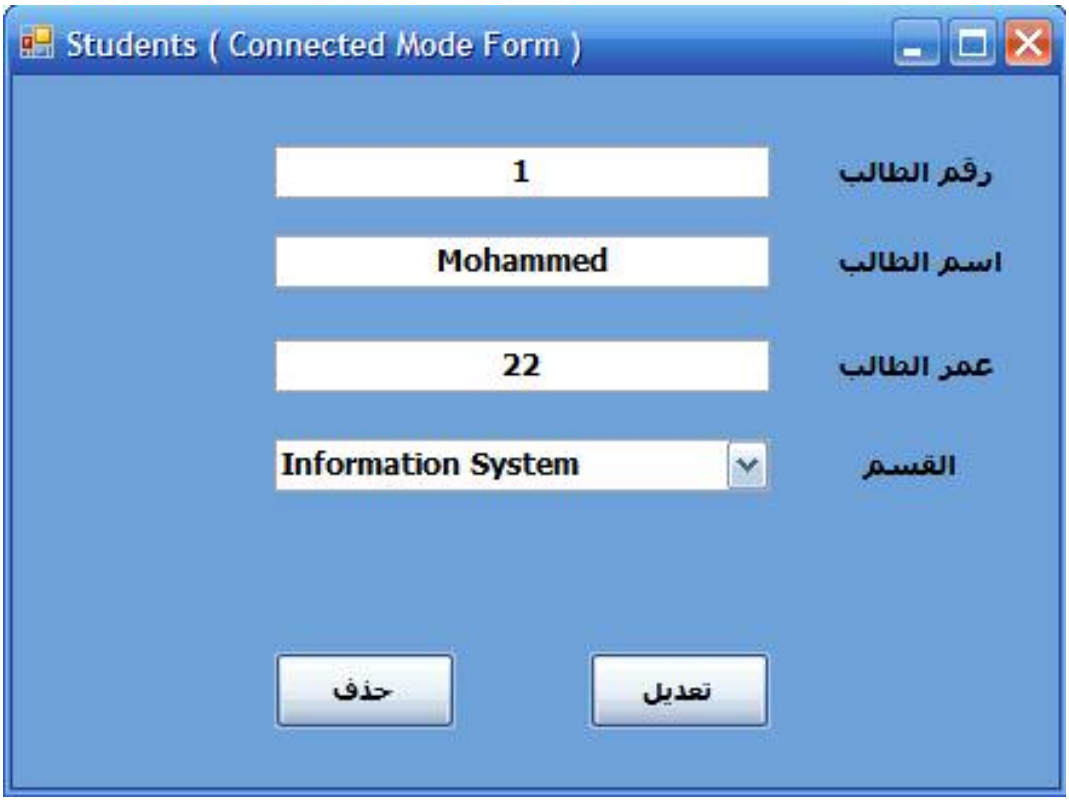
الأن سوف ننتقل إلى الجزء الأهم وهو إنشاء شاشة بسيطة تحتوي على مايلي :

1- ثلاثة مربع نصوص (Three TextBoxs) :-
- رقم الطالب StdNo (TextBox1) .
- اسم الطالب StdName (TextBox2) .
- عمر الطالب StdAge (TextBox3) .

2- قائمة منسدلة واحدة (One ComboBox) – قسم الطالب StdSec (ComboBox1) .

3- زرين أمر (Two Buttons) :-
- الزر الأول لعمليتي الحفظ + التعديل For Insert And Update (Button1) .
- الزر الثاني لعملية الحذف For Delete (Button2) .

الان لنقم ببناء شاشة بهذا الشكل : –



FrmStudents | AddNew

```
Imports System.Data.SqlClient
Public Class FrmStudents
    Dim Con As New SqlConnection("Data Source=.\SQLEXPRESS;AttachDbFilename
                    =|DataDirectory|\University.mdf;Integrated Security=True;
                    Connect Timeout=120;User Instance=True")

    Dim Cm As New SqlCommand
```

تعريف كائن الاتصال الذي من خلاله يمكننا التعامل مع قاعدة البيانات .. بعد ما حددنا لهذا الكائن سلسلة الاتصال

تعريف كائن أوامر يمكن من خلاله تنفيذ مجموعة أوامر SQL المعروفة (Insert , Update , Delete , Select , ...)

```
    Private Sub AddNew()
        For Each c As Control In Controls
         If TypeOf c Is TextBox Then c.Text=""
         If TypeOf c Is ComboBox Then c.Text=""
        Next
        Button2.Text = "حفظ"
        Button4.Enabled = False
        Try
            If Con.State = ConnectionState.Open Then Con.Close()
            Con.Open()
            Cm.CommandText = "Select max(Std_No) From Students"
            Cm.Connection = Con
            If IsDBNull(Cm.ExecuteScalar) Then
                TextBox1.Text = 1
            Else
                TextBox1.Text = Cm.ExecuteScalar + 1
            End If
            TextBox2.Focus()
        Catch ex As Exception
            MessageBox.Show(ex.Message)
        Finally
            Con.Close()
        End Try
    End Sub
```

إجراء يتم من خلاله التالي وبالترتيب :
- تنظيف الشاشة.
- تفعيل عملية الحفظ فقط .
- إعطاء ترقيم تلقائي لرقم الطالب , لتجنب أخطاء التكرار

دالة يمكن من خلالها تنفيذ جملة Select عندما يكون ناتج هذه الجملة سجل أو قيمة واحدة فقط .وهنا يكون قيمة جملة الاستعلام هي قيمة واحدة تمثل أكبر رقم طالب مخزن في جدول الطلاب .

(وباستخدام هذه الدالة يمكن التخلي عن الصيغة المعروفة لـ Select من استخدام DataReader و كذلك جملة Do...While..)

```
    Private Sub FrmStudents_Load(.....) Handles MyBase.Load
        AddNew()
    End Sub
```

سيتم من خلالها إستدعاء إجراء Addnew في كل مرة يتم فيها تحميل

```
    Private Sub TextBox1_KeyPress(.....) Handles TextBox1.KeyPress
        If Asc(e.KeyChar) <> 13 AndAlso Asc(e.KeyChar) <> 8
                                AndAlso Not IsNumeric(e.KeyChar) Then
            MessageBox.Show("يرجى إدخال أرقام فقط")
            e.Handled = True
        End If
    End Sub
```

يتم من خلال هذه الشفرة منع المستخدم من إدخال غير الأرقام في خانه رقم الطالب ..بالإضافة إلى إستثناء زري الـ Enter و Backspace .

FrmStudents

```
Private Sub Button2_Click(عمليتي الحفظ و التعديل) Handles Button2.Click
    If TextBox1.Text = "" Then
            Er.SetError(TextBox1, "يرجى التأكد من إدخال رقم الطالب")
            Exit Sub
    End If
    Er.Dispose()
    If TextBox2.Text = "" Then
       Er.SetError(TextBox2, "يرجى التأكد من إدخال اسم الطالب")
       Exit Sub
    End If
    Er.Dispose()
    If ComboBox1.Text = "" Then
       Er.SetError(ComboBox1, "يرجى التأكد من إدخال قسم الطالب")
       Exit Sub
    End If
    Er.Dispose()
   Try
      Con.Open()

    If Button2.Text = "حفظ" Then
        Cm.CommandText="insert into Students Values(@No,@Name,@Age,@Sec_No)"
    Else
        Cm.CommandText = "Update Students Set std_Name=@Name,Std_Age=@Age,
                                             Std_Sec_No=@Sec_No Where Std_No=@No"
    End If
    With Cm.Parameters
          .Clear()
          .AddWithValue("@No", TextBox1.Text)
          .AddWithValue("@Name", TextBox2.Text)
          .AddWithValue("@Age", TextBox3.Text)
          .AddWithValue("@Sec_No", ComboBox1.Text)
    End With
        Cm.Connection = Con
        Cm.ExecuteNonQuery()
        Msgbox "تمت عملية تحديث البيانات بنجاح"
    AddNew()
    Catch ex As Exception
       MessageBox.Show(ex.Message)
    Finally
       Con.Close()
    End Try
End Sub
```

هنا استخدمنا الفئة ErrorProvider للتحسس إذا حدث خطاء ما غير مرغوب فيه قبل القيام بعملية معينه مثل الحفظ أو التعديل هنا , ليتم تشغيلها من خلال إعطاء خطأ للمستخدم باستخدام هذه الدالة ليتم إشعار المستخدم بخطئه , وإنهاء العملية .

إذا كانت العملية الحالية هي حفظ يتم كتابة نص أمر الحفظ في داله CommandText ,
ويتم تحديد القيمة المقابلة لكل بارامتر في CommandText من خلال Parameters ومن ثم يتم تحديد الإتصال الذي سوف ينفذ الأمر عليه من خلال الداله Connection .. ومن ثم يتم تنفيذ أمر الـ Sql على الاتصال المحدد من خلال الدالة ExecuteNonQuery .
والأربع الدوال السابقة متفرعة من الفئة SqlCommand ,

بعد القيام بعملية الحفظ أو التعديل يتم إستدعاء الإجراء AddNew()
ليتم تنظيف الشاشة , وجلب السجل التالي للطالب وتفعيل عملية الحفظ

كما نعلم بأنه في الاتصال المتصل لا بد أولا من فتح الاتصال مع قاعدة البيانات من خلال Con.Open ومن ثم القيام بالعملية التي نحتاجها سواء (حفظ , تعديل , حذف , ...) ومن إغلاق الإتصال بالقاعدة من خلال Con.Close.لأنه لا يسمح بفتح أكثر من إتصال مع قاعدة البيانات .
وهذا عيب أساسي بهذا النوع من الإتصال .

FrmStudents

```
Private Sub Button4_Click( عملية الحذف) Handles Button4.Click
    Try
      Con.Open()
      Cm.CommandText = "Delete From Students Where Std_No=" &
                                                    TextBox1.Text
      Cm.Connection = Con
      Cm.ExecuteNonQuery()
      MessageBox.Show("تمت عملية الحذف بنجاح")
      AddNew()
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    Finally
        Con.Close()
    End Try
End Sub
```

يتم من خلال هذا الزر القيام بعملية حذف بيانات الطالب المحدد .. ولا تختلف عن السابق

```
Private Sub TextBox1_KeyDown( البحث) Handles TextBox1.KeyDown
  If e.KeyCode = Keys.Enter Then
    Try
      Con.Open()
      Cm.CommandText ="Select*From Students Where Std_No="& TextBox1.Text
      Cm.Connection = Con
      Dim Dr As SqlDataReader = Cm.ExecuteReader
       If Dr.HasRows Then
          Dr.Read
          TextBox2.Text = Dr.Item(1)
          TextBox3.Text = Dr.Item(2)
          ComboBox1.Text = Dr.Item(3)
          Button1.Text = "تعديل"
          Button2.Enabled = True
       Else
          MsgBox("لايوجد طالب بهذا الرقم")
           AddNew()
       End If
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    Finally
        Con.Close()
    End Try
  End If
End Sub
```

في حاله الضغط على المفتاح Enter في لوحة المفاتيح يتم البحث في جدول الطلاب عن الطالب الذي تم كتابة رقمة ليتم عرضة في الحقول المقابلة على الشاشة .

تنفيذ قارئ الأوامر لتحميل البيانات لداخل قارئ البيانات

تعريف متغير من نوع قارئ بيانات لقراءة بيانات الاستعلام

للتأكد من وجود بيانات بالقارئ أم لا

إن وجدت بيانات للطالب يتم البدأ بقراءة البيانات من قارئ البيانات وعرضها

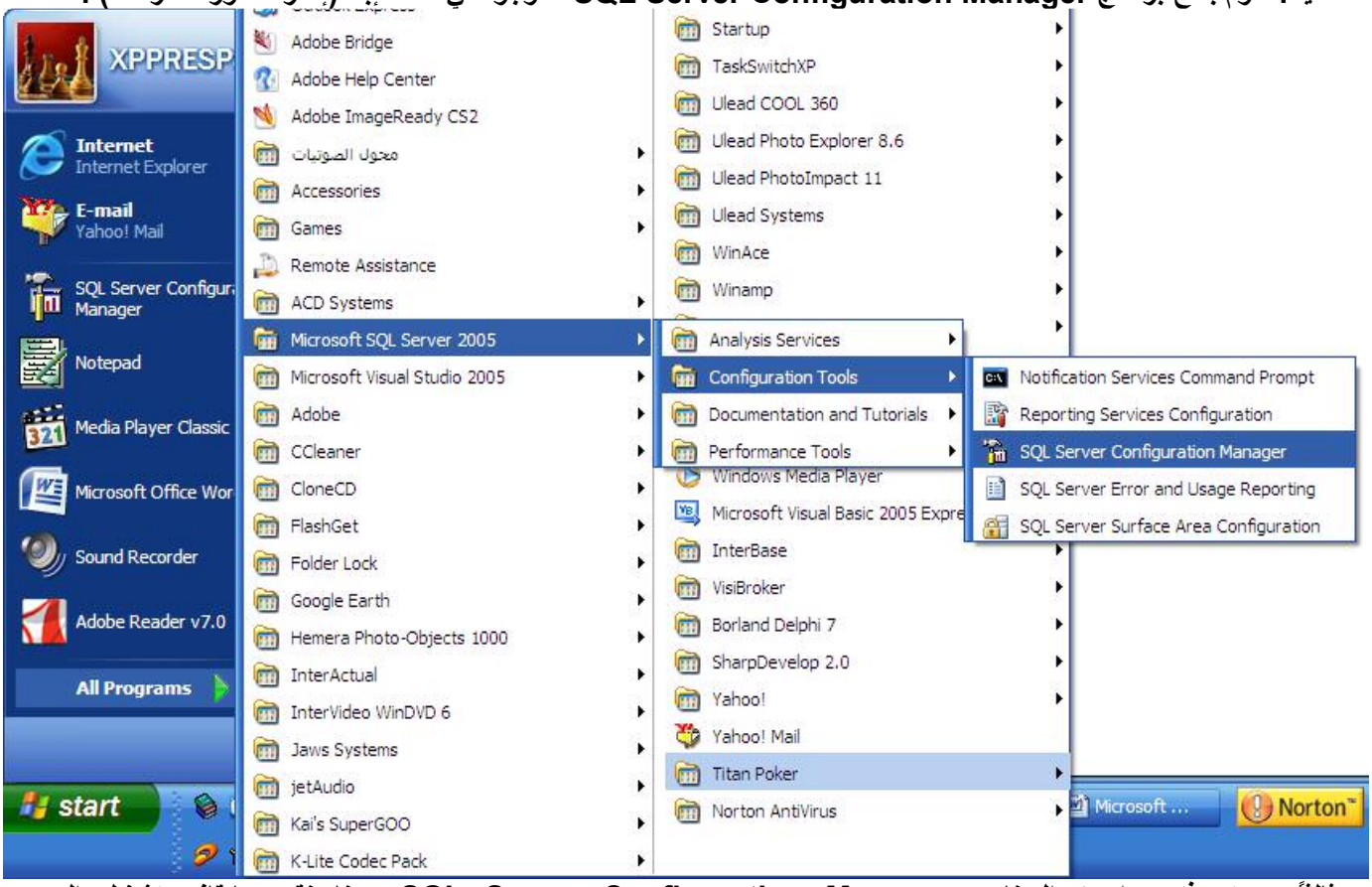
إذا وجدت بيانات الطالب يتم تفعيل عملية التعديل وكذلك الحذف

إذا لم توجد بيانات للطالب المحدد يتم إنهاء العمليه والخروج من الإجراء

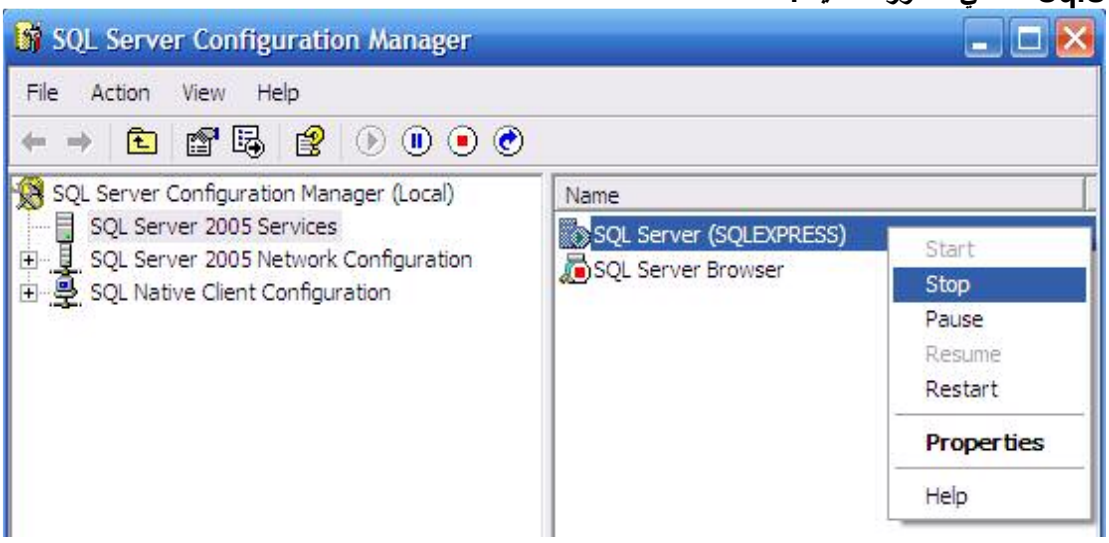
بعد إنتهاء عملية البحث سواء نجحت عمليه البحث أو لم تنجح فإنه يتم مباشرة إغلاق الاتصال بقاعدة البيانات .. للاستعداد لعمليه جديدة .

خطوات إرفاق قاعدة بيانات SQL SERVER Express لمشروعك ليعمل على أي جهاز

أولاً : قبل كل شيء نقوم بإنشاء قاعدة SqlServer بالطريقة التي تحدثنا عنها سابقاً (في الصفحات 76,75, 77) .
ثانياً : نقوم بفتح برنامج SQL Server Configuration Manager الموجود في قائمة إبدأ (إنظر الصورة المرفقة) :



ثالثاً : يتم فتح واجهه البرنامج SQL Server Configuration Manager وهنا نقوم بإيقاف تشغيل الـ SqlServerExpress كما في الصورة التالية :

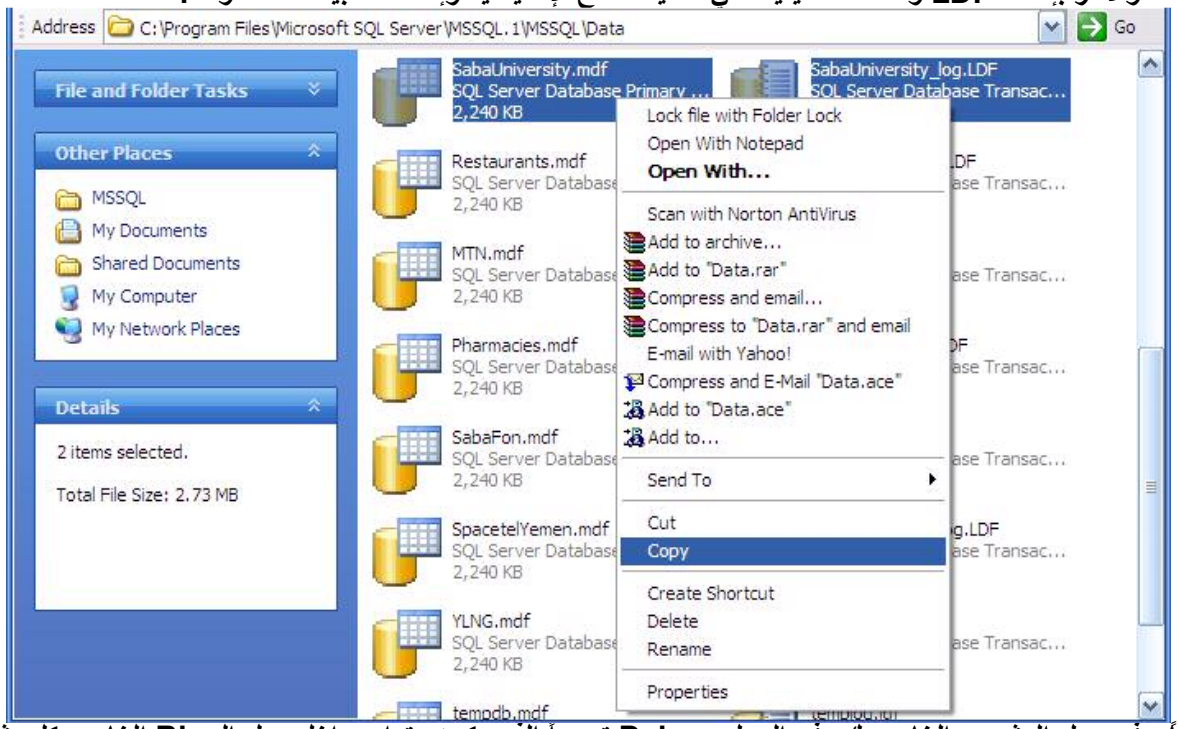


رابعاً : ننتقل إلى المسار الافتراضي الذي تنزل فيه Sql Server 2005 Express وهو كما يلي :

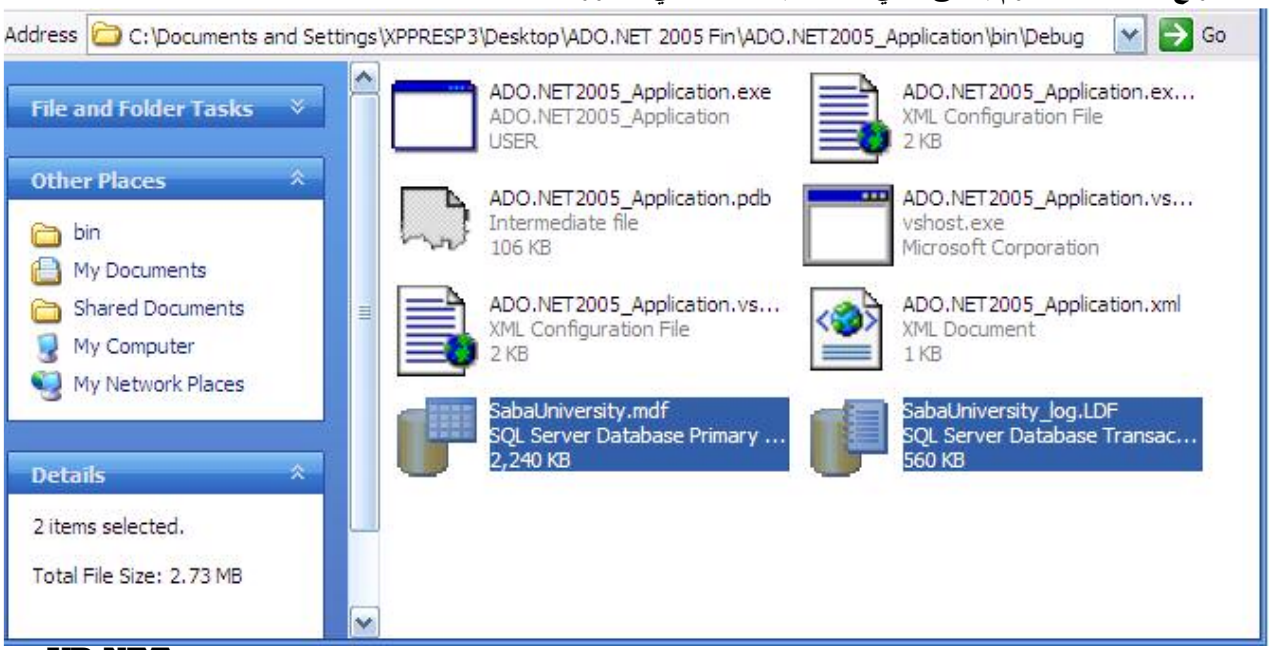
```
C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data
```

خامساً : بعد الإنتقال إلى المسار السابق ستظهر لنا جميع قواعد بيانات Sql Server 2005 Express التي قمنا بإنشاءها في بيئة الـ Net 2005. وبعد ذلك نقوم بإختيار ملفي قاعدة البيانات الخاصة بمشروعنا ونسخهما حيث أن هذين الملفين نوعين:

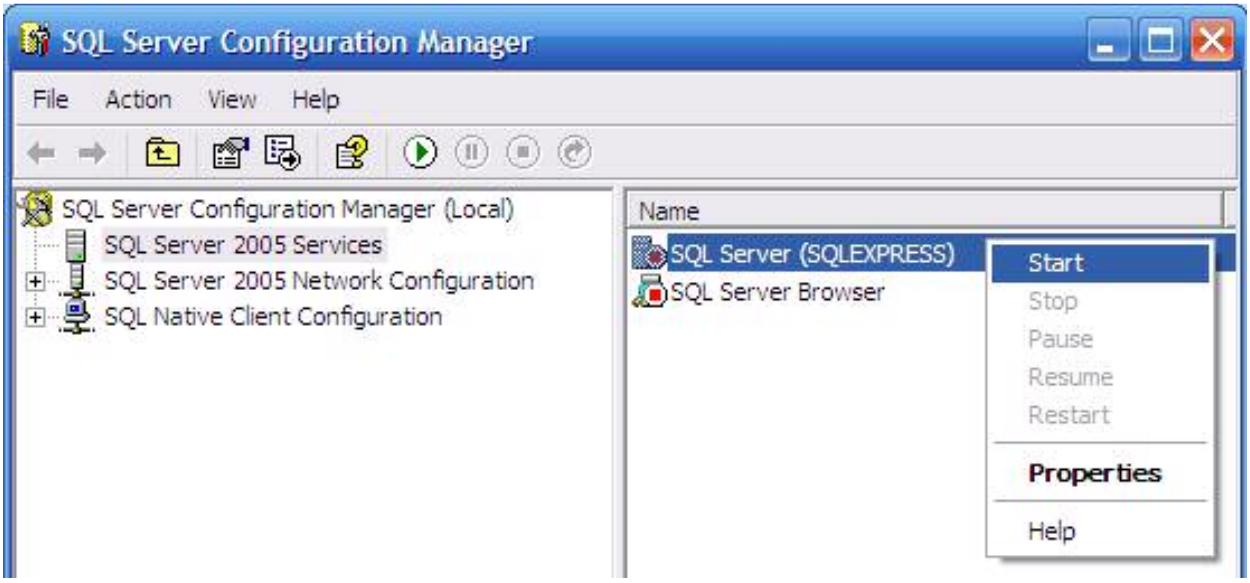
- أحدهما بإمتداد MDF وهو يمثل قاعدة البيانات بمحتوياتها .
- والأخر بإمتداد LDF وهذا الملف يفيدنا في عمليه النسخ الإحتياطية وإستعادة البيانات المفقودة .



سادساً : في مجلد المشروع الخاص بك وفي المجلد Debug تحديداً الذي يكون متواجد داخل مجلد الـ Bin الخاص بكل مشروع من مشاريع الـ Net. نقوم بلصق ملفي قاعدة البيانات كما في الصورة التالية :



سابعاً : نقوم بإعادة نفس الخطوة الثالثة ولكن الأن نقوم بتشغيل SqlServerExpress لنتمكن من التعامل معها كما في الصورة التالية :



ثامناً : عندما نقوم بإنشاء كائن الإتصال Connection String نجعل سلسلة الإتصال (وبسطر واحد فقط) بالشكل التالي :

```
Dim Con As New SqlConnection("Data Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\
إسم قاعدة البيانات .mdf;Integrated Security=True;Connect Timeout=120;User Instance=True")
```

بالنسبة لاسم قاعدة البيانات ففي مثالنا هذا هي University