الفصل الرابع
استخدام التوريث
Inheritance

نشرح في هذا الفصل مفهوم التوريث وكيفية استخدام التصنيفات كتصنيفات أساسية لتصنيفات أخرى.

بانتهاء هذا الفصل سنتعرف علي:

- مفهوم التوريث Inheritance.
- أساسيات التوريث.
- تعريف التصنيفات المشتقة.
- الوصول إلى الوظائف الموروثة Accessing Inherited Methods
- تغطية الخصائص والوظائف داخل التصنيفات المشتقة Overriding
- استخدام الكائن MyBase
- استخدام الكائن MyClass
- متى نستخدم التوريث
- استنساخ الوظائف Method Overloading

مفهوم التوريث Inheritance

يدعم **Visual Basic** التوريث **Inheritance** وهى القدرة على تعريف تصنيفات **Base Classes** لاستخدامها كأساس لتصنيفات أخرى مشتقة **Inherited Classes**. وتقوم التصنيفات المشتقة بوراثة الوظائف والخصائص والأحداث الموجودة بالتصنيف الأساسى مع القدرة على الإضافة إليها. كما يمكن تعريف وظائف جديدة داخل التصنيفات المشتقة بنفس اسم الوظائف الموجودة بالتصنيف الأساسى وتحتوى على تمثيل مختلف وهو ما يسمى **Method Overriding** "التحميل الزائد للطريقة".

وجميع التصنيفات التى تقوم بإنشائها داخل **Visual Basic** قابلة للوراثة والاشتقاق من خلال التصنيفات الأخرى. ومن خلال التوريث يمكنك إنشاء التصنيف واختبار عمله بشكل صحيح مرة واحدة ثم إعادة استخدامه كأساس لتصنيفات أخرى وبالتالى فلست فى حاجة إلى إنشاء كود التصنيف مرةً أخرى أو إعادة اختباره.

أساسيات التوريث

تستخدم كلمة **Inherits** فى تعريف تصنيف جديد يسمى تصنيف مشتق **Derived Class** مبنى على تصنيف موجود مسبقاً يسمى تصنيف أساسى **Base Class**. وترث التصنيفات المشتقة جميع الخصائص والوظائف والأحداث والمتغيرات والثوابت الموجودة بتصنيفاتها الأساسية مع القدرة على إضافة المزيد من العناصر إليها. ويمكن إجمال القواعد العامة لعملية التوريث فيما يلى:

- جميع التصنيفات قابلة للوراثة مع عدا التصنيفات التى تحتوى على الكلمة الأساسية **NotInheritable**، كما يمكن اشتقاق التصنيفات الموجودة بالمشروع الحالى أو داخل مشروع آخر بشرط تضمينه فى مراجع المشروع الحالى.
- على عكس لغات البرمجة التى تتيح التوريث المتعددة، يتيح **Visual Basic** مستوى واحداً من التوريث فقط، فلكل تصنيف مشتق تصنيفاً أساسياً واحداً. ولكن من الممكن أن يرث التصنيف المشتق أكثر من واجهة **Interface**.

- لمنع الوصول إلى العناصر المقيدة داخل التصنيف الأساسى، يجب أن يكون مستوى الوصول الخاص بالتصنيف المشتق مساوى أو أكثر تقيداً من التصنيف الأساسى. فعلى سبيل المثال، التصنيف العام **Public** لا يمكنه توريث تصنيف صديق **Friend** أو تصنيف خاص **Private**، كما لا يستطيع التصنيف الصديق توريث التصنيف الخاص.

ويتم تمثيل التوريث داخل **Visual Basic** باستخدام الكلمات الأساسية الآتية:

- كلمة **Inherits** ويتم من خلالها تحديد التصنيف الأساسى.
- كلمة **NotInheritable** وتستخدم لمنع المبرمجين من استخدام التصنيف كتصنيف أساسى لأى تصنيف آخر.
- كلمة **MustInherit** وتستخدم مع التصنيفات التى يجب استخدامها كتصنيفات أساسية لتصنيفات مشتقة فقط، ولا يمكن استخدامها بمفردها. ولا يمكن إنشاء حالات من هذا النوع من التصنيفات وإنما يتم إنشاء حالات من التصنيفات المشتقة فقط.

تعريف التصنيفات المشتقة Inherited Classes

يتم تعريف التصنيفات المشتقة من تصنيفات أساسية باستخدام كلمة **Inherits**، حيث يتم كتابة كلمة **Inherits** متبوعةً باسم التصنيف الأساسى كأول سطر داخل تعريف التصنيف المشتق. لتوضيح ذلك، نفترض أن لدينا تصنيفاً أساسياً باسم **Class1** يحتوى على الكود التالى:

```
Class Class1
  Sub Method1()
    MessageBox.Show("This is a method in the base class.")
  End Sub
  Overridable Sub Method2()
    MessageBox.Show("This is another method in the base
class.")
  End Sub
End Class
```

يحتوى التصنيف **Class1** كما ترى على وظيفة باسم **()Method1** وأخرى باسم **()Method2**، الأولى غير قابلة لإعادة التمثيل داخل التصنيفات المشتقة من هذا التصنيف، بينما تقبل الثانية وذلك لاحتوائها على كلمة **Overridable**.

على فرض أننا نرغب فى إنشاء تصنيف جديد باسم **Class2** يحتوى على جميع وظائف وخصائص التصنيف الأول **Class1**، يتم استخدام كلمة **Inherits** هكذا:

```
Class Class2
  Inherits Class1
  Public Field2 as Integer
  Overrides Sub Method2()
    Messagebox.Show("This is a method in a derived class.")
  End Sub
End Class
```

قمنا بدايةً بتعريف التصنيف ثم استخدمنا العبارة **Inherits Class1** للدلالة على اشتقاقه من التصنيف **Class1** وبذلك يكون التصنيف **Class1** تصنيفاً أساسياً والتصنيف **Class2** تصنيفاً مشتقاً. كما قمنا فى التصنيف المشتق بإعادة تمثيل الوظيفة **()Method2** لأنها قابلة للإخفاء (أو لإعادة التمثيل) كما ذكرنا منذ قليل.

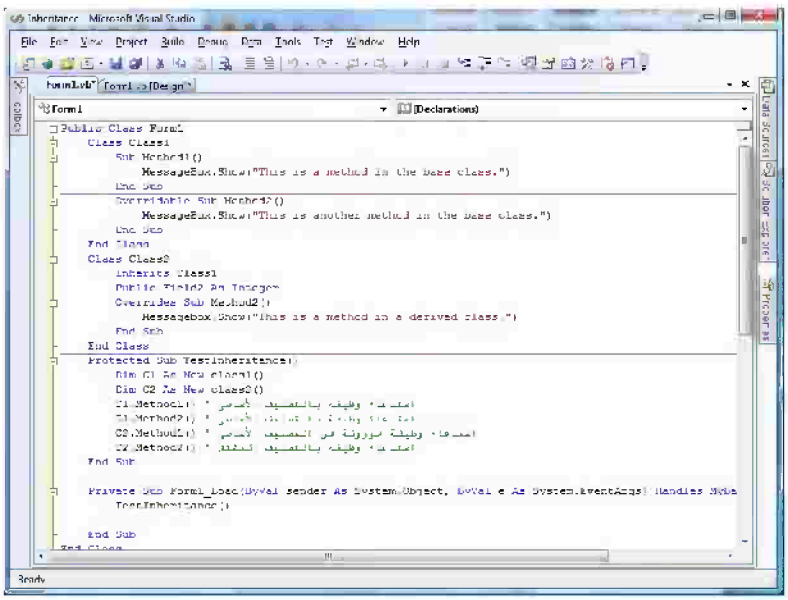
الوصول إلى الوظائف الموروثة Accessing Inherited Methods

فى حالة احتواء التصنيف الأساسى على وظيفة أو أكثر، فإن التصنيفات المشتقة من هذا التصنيف ترث هذه الوظائف وتستطيع استخدامها كما لو أن هذه الوظائف جزءاً لا يتجزأ منها. لتوضيح ذلك، دعنا نختبر عمل التصنيفين السابقين. قم بإنشاء إجراء جديد باسم مناسب وليكن **TestInheritance** ثم قم بإدخال الكود التالى:

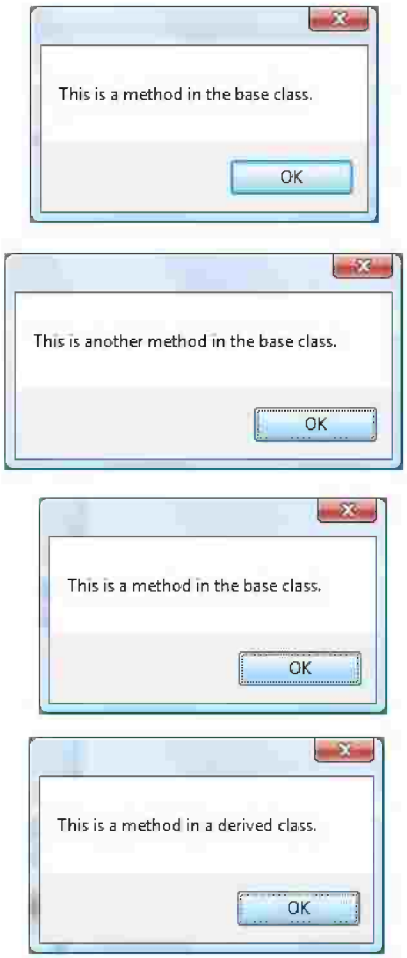
```
Protected Sub TestInheritance()
  Dim C1 As New class1()
  Dim C2 As New class2()
  C1.Method1() ' استدعاء وظيفة بالتصنيف الأساسى
  C1.Method2() ' استدعاء وظيفة بالتصنيف الأساسى
  C2.Method1() ' استدعاء وظيفة موروثة من التصنيف الأساسى
```

```
استدعاء وظيفة بالتصنيف المشتق ' C2.Method2()
End Sub
```

قمنا هنا بتعريف حالة (كائن) من التصنيف الأساسى **Class1** باسم **C1** وحالة من التصنيف المشتق باسم **C2**. لذا فعند استدعاء الوظيفة **()Method1** مع الكائن **C1** يتم استدعاء الوظيفة **()Method1** الموجودة بالتصنيف الأساسى، وكذلك الحال عند استدعاء الوظيفة **()Method2** مع نفس الكائن. أما عند استدعاء الوظيفتين مع الكائن **C2**، فيتم فى الحالة الأولى استدعاء الوظيفة **()Method1** الموجودة بالتصنيف الأساسى كما لو أنها جزءً من التصنيف المشتق، بينما يتم فى الثانية استدعاء الوظيفة **()Method2** الموجودة بالتصنيف المشتق وذلك لأننا أعدنا تمثيل الوظيفة داخل التصنيف المشتق. يوضح شكل ٤-١ التالى الكود السابق داخل نافذة كود نموذج التطبيق **Inheritance** الموجود على القرص المدمج المرفق بينما يوضح شكل ٤-٢ الرسائل الأربعة التى تظهر عند تشغيل التطبيق.



شكل ٤-١التصنيفات داخل نافذة الكود



شكل ٤-٢ نتيجة تشغيل التطبيق

تغطية الخصائص والوظائف داخل التصنيفات المشتقة Overriding

يرث التصنيف المشتق افتراضياً جميع الخصائص والوظائف الموجودة بالتصنيف الأساسى، وتسلك داخل التصنيف المشتق نفس سلوكها داخل التصنيف الأساسى. فإذا أردت أن يسلك أي من هذه الخصائص أو الوظائف سلوكاً مختلفاً، يمكنك تغطيتها بخصائص ووظائف أخرى وهو ما يسمى **Overriding**، وفيه يتم تعريف تمثيل آخر للوظيفة التى ترغب فى تغطيتها داخل التصنيف المشتق، وبذلك يكون لها سلوكان مختلفان، الأول داخل التصنيف الأساسى والآخر داخل التصنيف المشتق.

ويمكنك التحكم فى تغطية الخصائص والوظائف باستخدام كلمات التعديل التالية:

- كلمة **Overridable** وتستخدم مع الخاصية أو الوظيفة داخل التصنيف الأساسى لتعنى أنها قابلة للتمثيل مرةً أخرى (أى قابلة للتغطية) داخل التصنيفات المشتقة.
- كلمة **Overrides** وتستخدم داخل التصنيفات المشتقة مع الخاصية أو الوظيفة التى ترغب فى تغطيتها.
- كلمة **NotOverridable** وتستخدم داخل التصنيف الأساسى مع الخاصية أو الوظيفة التى لا ترغب فى إعادة تمثيلها (تغطيتها) داخل التصنيفات المشتقة. ولا داعى لاستخدام هذه الكلمة مع الخصائص والوظائف التى تحتوى على السلوك **Public** لأنها بطبيعتها غير قابلة للتغطية بالتصنيفات المشتقة.
- كلمة **MustOverride** وتستخدم داخل التصنيف الأساسى مع الخاصية أو الوظيفة التى يجب إعادة تمثيلها داخل التصنيفات المشتقة، وفى هذه الحالة يتكون تعريف الوظيفة من العبارة **Sub** أو **Function** أو **Property** فقط ولا يمكن استخدام العبارة **End Sub** أو **End Function** وهذا طبيعى لأن الوظيفة سيتم تعريفها داخل جميع التصنيفات المشتقة. كما يجب تعريف الوظائف التى تحتوى على كلمة **MustOverride** داخل التصنيفات التى تحتوى على كلمة **MustInherit**.

استخدام الكائن MyBase

تستخدم كلمة **MyBase** عند استدعاء الوظائف الموجودة بالتصنيف الأساسى والتى تم تغطيتها داخل التصنيف المشتق. لتوضيح ذلك دعنا نرى الكود التالى:

```
Class DerivedClass Inherits BaseClass
  Public Overrides Function CalculateShipping(ByVal Dist As
Double, ByVal Rate As Double) As Double
    استدعاء الوظيفة الموجودة بالتصنيف الأساسى وتعديل قيمتها الناتجة'
    Return MyBase.CalculateShipping(Dist, Rate) * 2
  End Function
End Class
```

فى هذا الكود تم اشتقاق التصنيف **DerivedClass** من التصنيف الأساسى **BaseClass** الذى يحتوى بدوره على دالة باسم **()CalculateShipping** ثم قمنا بتغطية هذه الدالة داخل التصنيف المشتق بدالة أخرى مماثلة إلا أنها تقوم بإرجاع ضعف القيمة الناتجة من الدالة الموجودة بالتصنيف الأساسى. ولكى نشير إلى الدالة الموجودة بالتصنيف الأساسى، استخدمنا كلمة **MyBase** والتى تعنى أن ما بعدها موجود بالتصنيف الأساسى وليس التصنيف الحالى.

وعند استخدام كلمة **MyBase** يجب ملاحظة ما يلى:

- تشير **MyBase** إلى التصنيف الأساسى المباشر وعناصره المشتقة، ولا يمكن استخدامها فى الوصول إلى العناصر الخاصة **Private** الموجودة بالتصنيف.
- كلمة **MyBase** عبارة عن كلمة أساسية وليست كائن، لذا لا يمكن تخصيصها لمتغير أو تمريرها إلى إجراء أو استخدامها فى مقارنة داخل عبارة **IS**.
- ليس بالضرورة أن تعرف الوظيفة المشار إليها من خلال كلمة **MyBase** داخل التصنيف الأساسى المباشر وإنما يمكن تعريفها داخل تصنيف أساسى آخر موروث بطريقة غير مباشرة.
- لا يمكنك استخدام **MyBase** فى استدعاء وظائف التصنيف الأساسى التى تحتوى على كلمة **MustOverride**.

- لا يمكنك استخدام **MyBase** فى الإشارة إلى نفسها، فالعبارة التالية على سبيل المثال غير صحيحة:
- **MyBase.MyBase.BtnOK_Click()**
- لا يمكنك استخدام **MyBase** داخل الوحدات النمطية **Modules**.
- لا يمكنك استخدام **MyBase** فى الإشارة إلى العناصر الموجودة بالتصنيف الأساسى وتحتوى على مستوى الوصول **Friend** إذا كان التصنيف الأساسى موجوداً داخل تجمع آخر.

استخدام الكائن MyClass

يمكنك استخدام كلمة **MyClass** فى استدعاء وظيفة داخل التصنيف الأساسى قابلة للتغطية **Overridable** مع التأكد من استدعاء تمثيل الوظيفة داخل التصنيف الأساسى لا داخل أيٍ من التصنيفات المشتقة. ويجب عند استخدام كلمة **MyClass** مراعاة النقاط الآتية:

- كلمة **MyClass** عبارة عن كلمة أساسية وليست كائن، لذا لا يمكن تخصيصها لمتغير أو تمريرها إلى إجراء أو استخدامها فى مقارنة داخل عبارة **IS**.
- لا يمكنك استخدام **MyClass** داخل الوحدات النمطية **Modules**.
- تشير **MyClass** إلى التصنيف المحيط وعناصره الموروثة.
- يمكنك استخدام **MyClass** فى الإشارة إلى العناصر المشتركة **Shared**.
- يمكنك استخدام **MyClass** فى الإشارة إلى الوظائف المعرفة داخل التصنيف الأساسى ولا تحتوى على تمثيل بداخله، وفى هذه الحالة تكون كلمة **MyClass** مماثلة تماماً لكلمة **MyBase**.

متى نستخدم التوريث

يتم استخدام التوريث فى الحالات التالية:

- حينما يكون التصنيف المشتق أحد أنواع التصنيف الأساسى، فمثلاً نقول الأسد أحد أنواع الحيوانات. فى هذه الحالة يمكنك تعريف تصنيف للحيوانات وتصنيف آخر للأسود، على أن يقوم تصنيف الأسود بوراثة تصنيف الحيوانات، أو بمعنى آخر يكون تصنيف الأسود مشتقاً من تصنيف الحيوانات. أما إذا كان التصنيف جزءاً من تصنيف آخر، فلا ينصح باستخدام التوريث فى هذه الحالة.
- عند الرغبة فى استخدام الكود الموجود بالتصنيفات الأساسية أكثر من مرة داخل التصنيفات المشتقة.
- عند الرغبة فى تطبيق نفس خصائص ووظائف التصنيف ولكن بأنواع بيانات أخرى.
- عند الرغبة فى بناء شكل هرمى للتصنيفات.
- عند الرغبة فى إجراء تعديلات كبيرة على التصنيفات المشتقة عن طريق تعديل التصنيف الأساسى فقط.

ولتوضيح الفرق بين استخدام الكود العادى واستخدام التوريث، دعنا نرى الكود التالى والذى يتم فيه إنشاء الإجراء **Draw()** المستخدم فى إنشاء الكائنات الرسومية المختلفة كالدائرة والخط على سبيل المثال:

```
Sub Draw(ByVal Shape As DrawingShape, ByVal X As Integer, _
    ByVal Y As Integer, ByVal Size As Integer)
  Select Case Shape.Type
    Case shpCircle
    ' يتم هنا وضع كود رسم الدائرة
    Case shpLine
    ' يتم هنا وضع كود خط الرسم
  End Select
End Sub
```

يتم فى هذا الإجراء كما ترى استخدام عبارة **Select** فى التعرف على الشكل المراد رسمه ثم تنفيذ الكود المناسب لهذا الشكل. وهذا الكود على الرغم من بساطته إلا أن به بعض العيوب. فعلى فرض أننا أردنا فيما بعد تضمين إمكانية رسم القطع الناقص بالإضافة إلى

الدائرة والخط، يجب فى هذه الحالة تعديل الكود الأصلى، كما سيحتاج رسم القطع الناقص إلى معامل إضافى نظراً لرسمه بقطرين وليس قطراً واحداً، وهو مالا يتماشى مع خط الرسم. فإذا ما أردنا إضافة إمكانية رسم المضلع، فسنحتاج إلى معامل آخر. وهنا تظهر التوريث لتحل كل هذه المشاكل، حيث يتم تعريف الوظائف بالتصنيف الأساسى مع ترك تمثيلها داخل التصنيفات المشتقة وهو ما يعطى مرونة شديدة فى التعامل مع الكود دون الحاجة إلى تغييره من البداية كما هو الحال فى الكود السابق، كود الإجراء **()Draw**.

لاستخدام التوريث فى تمثيل عملية رسم الكائنات، سنقوم بدايةً بتعريف التصنيف الأساسى وليكن باسم **Shape** ويحتوى على الكود التالى:

```
MustInherit Class Shape
  Public X As Integer
  Public Y As Integer
  MustOverride Sub Draw()
End Class
```

وقد استخدمنا هنا كلمة **MustInherit** للدلالة على أن هذا التصنيف تصنيفاً أساسياً يجب اشتقاقه ولا يجوز استخدامه فى تعريف حالات (كائنات) جديدة. كما استخدمنا كلمة **MustOverride** مع الإجراء **()Draw** للدلالة على وجوب تمثيل هذا الإجراء داخل التصنيفات المشتقة كلٌ حسب حالته.

وبذلك يمكنك تمثيل ما تشاء من عناصر رسومية من خلال تصنيفات مشتقة من التصنيف **Shape**. فعلى سبيل المثال، يمكنك تعريف التصنيف **Line** المستخدم فى رسم الخطوط المستقيمة كما يلى:

```
Class Line
  Inherits Shape
  Public Length As Integer
  Overrides Sub Draw()
    يتم هنا وضع الكود المستخدم فى رسم الخطوط المستقيمة'
  End Sub
End Class
```

كما يمكنك تعريف التصنيف **Rectangle** المشتق من التصنيف **Line** (المشتق بدوره من التصنيف **Shape**) والمستخدم فى رسم المستطيلات هكذا:

```
Class Rectangle
  Inherits Line
  Public Width As Integer
  Overrides Sub Draw()
    يتم هنا وضع الكود المستخدم فى رسم المستطيلات '
  End Sub
End Class
```

وهكذا يمكنك تعريف ما تشاء من تصنيفات مشتقة لرسم الأشكال المختلفة دون أن يؤثر أحد هذه التصنيفات على الآخر.

استنساخ الوظائف Method Overloading

فى حالة الرغبة فى إنشاء مجموعة من الوظائف (الدوال) التى تقوم كلها بوظيفة واحدة مع اختلاف معاملات كلٍ منها، والحل فى هذه الحالة يكمن فى استخدام معاملات اختيارية أو بتخصيص اسم مميز لكل وظيفة على حده، ولكن مع البرمجة الموجهة بالكائنات يمكنك استخدام تقنية استنساخ الوظائف **Method Overloading** والتى يتم فيها إنشاء أكثر من وظيفة بنفس الاسم إلا أن كلا منها يحتوى على معاملات مختلفة. ويتم تعريف الوظيفة المستنسخة باستخدام كلمة **Overloads** وهى من الكلمات الأساسية داخل **Visual Basic**.

لتوضيح ذلك، نفترض أن لدينا وظيفة باسم **GetWord()** تقوم باسترجاع كلمة داخل عبارة نصية. ونرغب فى استخدام هذه الوظيفة فى أداء ثلاث حالات مختلفة، الأولى استرجاع أول كلمة فى العبارة النصية والثانية استرجاع كلمة من مكان معين داخل العبارة النصية والثانية استرجاع كلمة معينة داخل العبارة النصية، لذا نقوم بإنشاء ثلاث وظائف بنفس الاسم وتحتوى على التوقيع التالى:

```
Function GetWord() As String
Function GetWord(ByVal Position As Integer) As String
Function GetWord(ByVal Search As String) As String
```

فالأولى لا تحتوى على أية معاملات لاسترجاع أول كلمة بالعبارة، بينما تحتوى الثانية على معامل من النوع **Integer** لاسترجاع كلمة تبدأ من مكان معين داخل العبارة، كما تحتوى الثالثة على معامل واحد أيضاً ولكن من النوع **String** لاسترجاع كلمة معينة موجودة داخل العبارة. وعند الرغبة فى تنفيذ أيٍ من الوظائف الثلاث، يتم استدعاء وظيفة واحدة باسم **GetWord()** ويتولى **Visual Basic** اختيار الوظيفة المناسبة من الوظائف الثلاث تبعاً لعدد المعاملات الممررة للوظيفة ونوع هذه المعاملات. فعلى سبيل المثال، فى الكود التالى:

```
MyString.GetWord()
MyString.GetWord(4)
MyString.GetWord("waleed")
```

لا تحتوى العبارة الأولى على أية معاملات، لذا يتم استدعاء الحالة الأولى من الوظيفة، بينما تحتوى العبارة الثانية على معامل واحد من النوع **Integer**، لذا يتم استدعاء الحالة الثانية من الوظيفة، أما العبارة الثالثة فتحتوى على معامل واحد أيضاً ولكن من النوع **String**، لذا يتم استدعاء الحالة الثالثة من الوظيفة.