الفصل الخامس

التوريث والطرق Inheritance & Methods

نشرح في هذا الفصل مفهوم التوريث وكيفية استخدام الفصائل كفصائل أساسية من فصائل أخرى.
بالانتهاء من هذا الفصل ستكتسب المعارف وتتدرب على المهارات التى تجعلك قادرا على:

- *مفهوم التوريث* **Inheritance**.
- *تعريف الفصائل المشتقة.*
- *الوصول إلى الطرق الموروثة* **Accessing Inherited Methods**
- *تغطية الخصائص والطرق داخل الفصائل المشتقة* **Overriding**
- *استخدام الكائن* **MyBase**
- *استخدام الكائن* **MyClass**
- *متى نستخدم التوريث*
- *استنساخ الطرق* **Method Overloading**

مفهوم التوريث Inheritance

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

وجميع الفصائل التى تقوم بإنشائها داخل **Visual Basic** قابلة للوراثة والاشتقاق من خلال الفصائل الأخرى. ومن خلال التوريث يمكنك إنشاء الفصيل واختبار عمله بشكل

صحيح مرة واحدة ثم إعادة استخدامه كأساس لفصائل أخرى وبالتالى فلست فى حاجة إلى إنشاء كود الفصيل مرةً أخرى أو إعادة اختباره.

أساسيات التوريث

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

- جميع الفصائل قابلة للوراثة مع عدا الفصائل التى تحتوى على الكلمة الأساسية **NotInheritable**، كما يمكن اشتقاق الفصائل 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 فصيل مشتق فصيلاً أساسياً واحداً. ولكن من الممكن أن يرث الفصيل المشتق أكثر من واجهة **Interface**.
- لمنع الوصول إلى العناصر المقيدة داخل الفصيل الأساسى، يجب أن يكون مستوى الوصول الخاص بالفصيل المشتق مساوى أو أكثر تقيداً من الفصيل الأساسى. فعلى سبيل المثال، الفصيل العام **Public** لا يمكنه توريث فصيل صديق **Friend** أو فصيل خاص **Private**، كما لا يستطيع الفصيل الصديق توريث الفصيل الخاص.

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

- كلمة **Inherits** ويتم من خلالها تحديد الفصيل الأساسى.
- كلمة **NotInheritable** وتستخدم لمنع المبرمجين من استخدام الفصيل كفصيل أساسى لأى فصيل آخر.
- كلمة **MustInherit** وتستخدم مع الفصائل التى يجب استخدامها كفصائل أساسية لفصائل مشتقة فقط، ولا يمكن استخدامها بمفردها. ولا يمكن إنشاء حالات من هذا النوع من الفصائل وإنما يتم إنشاء حالات من الفصائل المشتقة فقط.

تعريف الفصائل المشتقة Inherited Classes

يتم تعريف الفصائل المشتقة من فصائل أساسية باستخدام كلمة **Inherits**، حيث يتم كتابة كلمة **Inherits** متبوعةً باسم الفصيل الأساسي كأول سطر داخل تعريف الفصيل المشتق. لتوضيح ذلك، نفترض أن لدينا فصيلاً أساسياً باسم **Class1** يحتوى على الكود التالي:

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

على فرض أننا نرغب فى إنشاء فصيل جديد باسم **Class2** يحتوى على جميع وظائف وخصائص الفصيل 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 الفصيل ثم استخدمنا العبارة **Inherits Class1** للدلالة على اشتقاقه من الفصيل **Class1** وبذلك يكون الفصيل **Class1** فصيلاً أساسياً والفصيل **Class2** فصيلاً مشتقاً. كما قمنا فى الفصيل المشتق بإعادة تمثيل الوظيفة **()Method2** لأنها قابلة للإخفاء (أو لإعادة التمثيل) كما ذكرنا منذ قليل.

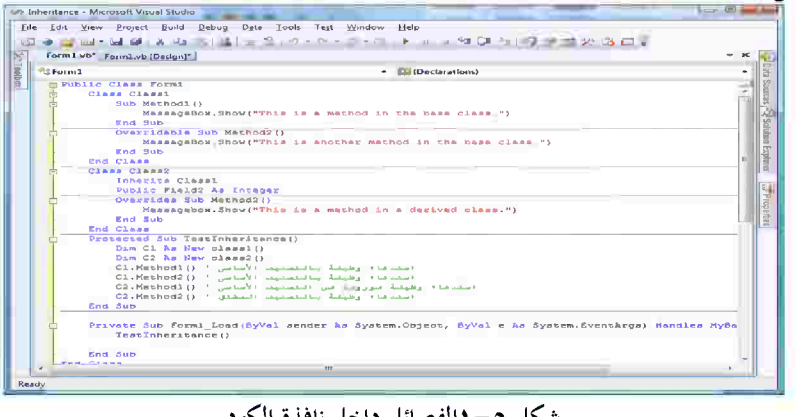
استدعاء الطرق الموروثة Calling Inherited Methods

فى حالة احتواء الفصيل الأساسى على وظيفة أو أكثر، فإن الفصائل المشتقة من هذا الفصيل ترث هذه الطرق وتستطيع استخدامها كما لو أن هذه الطرق جزءاً لا يتجزأ منها.

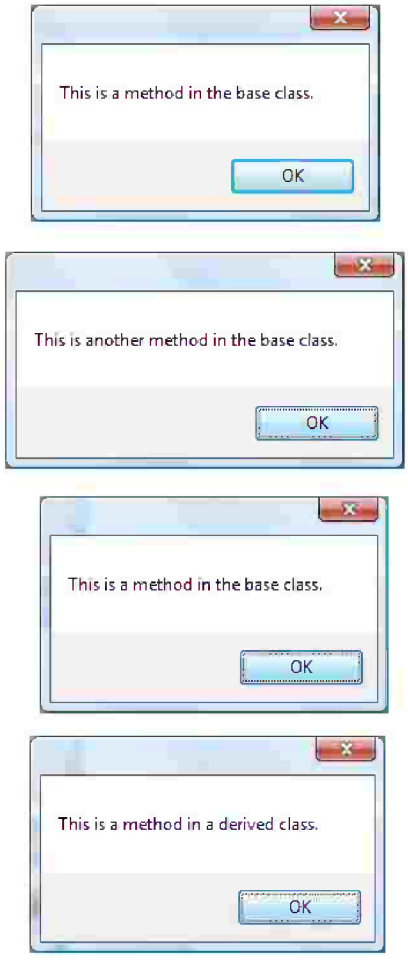
لتوضيح ذلك، دعنا نختبر عمل الفصيلين السابقين. قم بإنشاء إجراء جديد باسم مناسب وليكن **TestInheritance** ثم قم بإدخال الكود التالى:

```
Protected Sub TestInheritance()
  Dim C1 As New class1()
  Dim C2 As New class2()
  C1.Method1() ' استدعاء وظيفة بالفصيل الأساسى
  C1.Method2() ' استدعاء وظيفة بالفصيل الأساسى
  C2.Method1() ' استدعاء وظيفة موروثة من الفصيل الأساسى
  C2.Method2() ' استدعاء وظيفة بالفصيل المشتق
End Sub
```

قمنا هنا بتعريف حالة (كائن) من الفصيل الأساسى **Class1** باسم **C1** وحالة من الفصيل المشتق باسم **C2**. لذا فعند استدعاء الوظيفة **()Method1** مع الكائن **C1** يتم استدعاء الوظيفة **()Method1** الموجودة بالفصيل 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 بالفصيل الأساسى كما لو أنها جزءً من الفصيل المشتق، بينما يتم فى الثانية استدعاء الوظيفة **()Method2** الموجودة بالفصيل المشتق وذلك لأننا أعدنا تمثيل الوظيفة داخل الفصيل المشتق. يوضح شكل ٥-١ التالى الكود السابق داخل نافذة كود نموذج التطبيق **Inheritance** الموجود على القرص المدمج المرفق بينما يوضح شكل ٥-٢ الرسائل الأربعة التى تظهر عند تشغيل التطبيق.



شكل ٥-١الفصائل داخل نافذة الكود



شكل ٥-٢ نتيجة تشغيل التطبيق

تغطية الخصائص والطرق داخل الفصائل المشتقة Overriding

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

ويمكنك التحكم فى تغطية الخصائص والطرق باستخدام كلمات التعديل التالية:

- كلمة **Overridable** وتستخدم مع الخاصية أو الوظيفة داخل الفصيل الأساسى لتعنى أنها قابلة للتمثيل مرةً أخرى (أى قابلة للتغطية) داخل الفصائل المشتقة.
- كلمة **Overrides** وتستخدم داخل الفصائل المشتقة مع الخاصية أو الوظيفة التى ترغب فى تغطيتها.
- كلمة **NotOverridable** وتستخدم داخل الفصيل الأساسى مع الخاصية أو الوظيفة التى لا ترغب فى إعادة تمثيلها (تغطيتها) داخل الفصائل المشتقة. ولا داعى لاستخدام هذه الكلمة مع الخصائص والطرق التى تحتوى على السلوك **Public** لأنها بطبيعتها غير قابلة للتغطية بالفصائل المشتقة.
- كلمة **MustOverride** وتستخدم داخل الفصيل الأساسى مع الخاصية أو الوظيفة التى يجب إعادة تمثيلها داخل الفصائل 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 الفصائل المشتقة. كما يجب تعريف الطرق التى تحتوى على كلمة **MustOverride** داخل الفصائل التى تحتوى على كلمة **MustInherit**.

استخدام الكائن MyBase

تستخدم كلمة **MyBase** عند استدعاء الطرق الموجودة بالفصيل الأساسى والتى تم تغطيتها داخل الفصيل 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 بالفصيل الأساسى وتعديل قيمتها الناتجة"
    Return MyBase.CalculateShipping(Dist, Rate) * 2
  End Function
End Class
```

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

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

- تشير **MyBase** إلى الفصيل الأساسى المباشر وعناصره المشتقة، ولا يمكن استخدامها فى الوصول إلى العناصر الخاصة **Private** الموجودة بالفصيل.
- 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 الفصيل الأساسى المباشر وإنما يمكن تعريفها داخل فصيل أساسى آخر موروث بطريقة غير مباشرة.
- لا يمكنك استخدام **MyBase** فى استدعاء وظائف الفصيل 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 بالفصيل الأساسى وتحتوى على مستوى الوصول **Friend** إذا كان الفصيل الأساسى موجوداً داخل تجمع آخر.

استخدام الكائن MyClass

يمكنك استخدام كلمة **MyClass** فى استدعاء وظيفة داخل الفصيل الأساسى قابلة للتغطية **Overridable** مع التأكد من استدعاء تمثيل الوظيفة داخل الفصيل الأساسى لا داخل أيٍ من الفصائل 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 الفصيل المحيط وعناصره الموروثة.
- يمكنك استخدام **MyClass** فى الإشارة إلى العناصر المشتركة **Shared**.
- يمكنك استخدام **MyClass** فى الإشارة إلى الطرق المعرفة داخل الفصيل 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 الفصيل المشتق أحد أنواع الفصيل الأساسى، فمثلاً نقول الأسد أحد أنواع الحيوانات. فى هذه الحالة يمكنك تعريف فصيل للحيوانات وفصيل آخر للأسود، على أن يقوم فصيل الأسود بوراثة فصيل الحيوانات، أو بمعنى آخر يكون فصيل الأسود مشتقاً من فصيل الحيوانات. أما إذا كان الفصيل جزءاً من فصيل آخر، فلا ينصح باستخدام التوريث فى هذه الحالة.

- عند الرغبة فى استخدام الكود الموجود بالفصائل الأساسية أكثر من مرة داخل الفصائل المشتقة.
- عند الرغبة فى تطبيق نفس خصائص ووظائف الفصيل ولكن بأنواع بيانات أخرى.
- عند الرغبة فى بناء شكل هرمى للفصائل.
- عند الرغبة فى إجراء تعديلات كبيرة على الفصائل المشتقة عن طريق تعديل الفصيل 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 الطرق بالفصيل الأساسى مع ترك تمثيلها داخل الفصائل 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 الفصيل 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 الفصيل فصيلاً 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 الفصائل المشتقة كلٌ حسب حالته.

وبذلك يمكنك تمثيل ما تشاء من عناصر رسومية من خلال فصائل مشتقة من الفصيل **Shape**. فعلى سبيل المثال، يمكنك تعريف الفصيل **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 الفصيل **Rectangle** المشتق من الفصيل **Line** (المشتق بدوره من الفصيل **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 فصائل مشتقة لرسم الأشكال المختلفة دون أن يؤثر أحد هذه الفصائل على الآخر.

استنساخ الطرق Method Overloading

فى حالة الرغبة فى إنشاء مجموعة من الطرق (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 الطرق **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 الطرق الثلاث، يتم استدعاء وظيفة واحدة باسم **GetWord()** ويتولى **Visual Basic** اختيار الوظيفة المناسبة من الطرق 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.