

الفصل الرابع استخدام التوريث Inheritance

نشرح في هذا الفصل مفهوم التوريث وكيفية استخدام

التصنيفات كتصنيفات أساسية لتصنيفات أخرى.

بانتها هذا الفصل سنتعرف على:

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

وجميع التصنيفات التي 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 وبالتالي فلست في حاجة إلى إنشاء كود التصنيف مرة أخرى أو إعادة اختباره.

أساسيات التوريث

تستخدم كلمة **Inherits** في تعريف تصنيف جديد يسمى تصنيف مشتق **Derived Class** مبنى على تصنيف موجود مسبقاً يسمى تصنيف أساسي **Base Class**. وتورث 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 يلي:

- جميع التصنيفات قابلة للوراثة مع عدا التصنيفات التي تحتوى على الكلمة الأساسية **NotInheritable**، كما يمكن اشتقاق التصنيفات الموجودة بالمشروع الحالى أو داخل مشروع آخر بشرط تضمينه في مراجع المشروع الحالى.
- على عكس لغات البرمجة التي 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 الأساسي، يجب أن يكون مستوى الوصول الخاص بالتصنيف المشتق مساوياً أو أكثر تقييداً من التصنيف الأساسي. 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 الأساسي.
- كلمة **NotInheritable** وتستخدم لمنع المبرمجين من استخدام التصنيف كتصنيف أساسي لأي تصنيف آخر.
- كلمة **MustInherit** وتستخدم مع التصنيفات التي 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 الأساسي كأول سطر داخل تعريف التصنيف المشتق. لتوضيح ذلك، نفترض أن لدينا تصنيفاً أساسياً باسم **Class1** يحتوي 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
```

يحتوي 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 في إنشاء تصنيف جديد باسم **Class2** يحتوي 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 في التصنيف المشتق بإعادة تمثيل الوظيفة **Method2()** لأنها قابلة للإخفاء (أو لإعادة التمثيل) كما ذكرنا منذ قليل.

الوصول إلى الوظائف الموروثة Accessing Inherited Methods

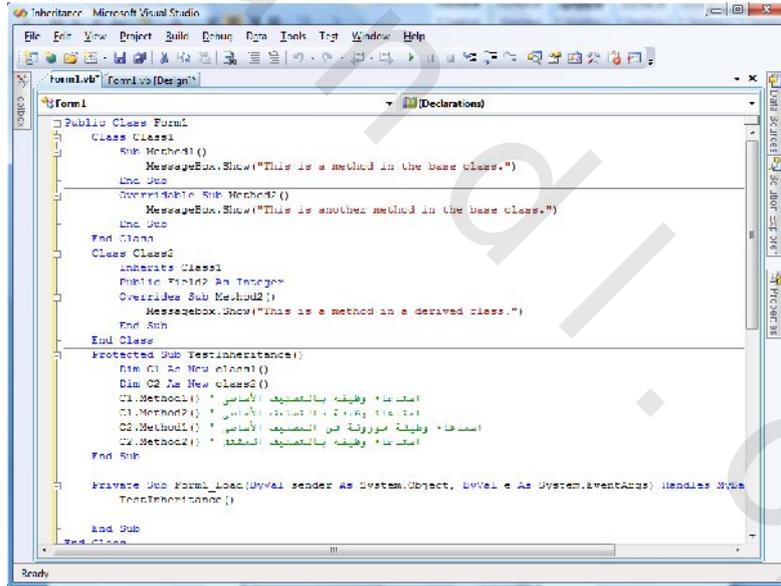
في حالة احتواء التصنيف الأساسي على وظيفة أو أكثر، فإن التصنيفات المشتقة من هذا التصنيف ترث هذه الوظائف وتستطيع استخدامها كما لو أن هذه الوظائف جزءاً لا يتجزأ منها. لتوضيح ذلك، دعنا نختبر عمل التصنيفين السابقين. قم بإنشاء إجراء جديد باسم مناسب وليكن **TestInheritance** ثم قم بإدخال الكود التالي:

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

استدعاء وظيفة بالتصنيف المشتق ' C2.Method2()

End Sub

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



```
Public Class Form1
    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

    Class Class2
        Inherits Class1
        Public Field? As Integer
        Overrides Sub Method2 ()
            MessageBox.Show("This is a method in a derived class.")
        End Sub
    End Class

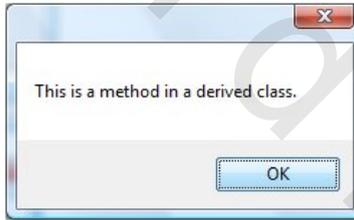
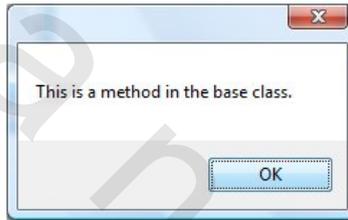
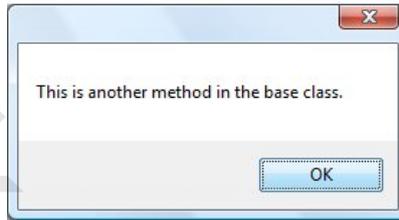
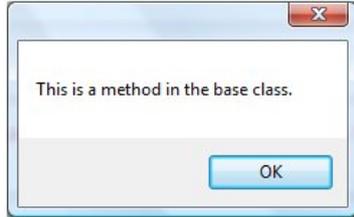
    Protected Sub TestInheritance()
        Dim C1 As New Class1 ()
        Dim C2 As New Class2 ()

        C1.Method1() : * استدعاء وظيفة بالتصنيف الأساسي *
        C1.Method2() : * استدعاء وظيفة بالتصنيف الأساسي *
        C2.Method1() : * استدعاء وظيفة مورثة من التصنيف الأساسي *
        C2.Method2() : * استدعاء وظيفة بالتصنيف المشتق *

    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        TestInheritance()
    End Sub
End Class
```

شكل ٤-١ التصنيفات 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 الأساسي، وتسلك داخل التصنيف المشتق نفس سلوكها داخل التصنيف الأساسي. 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 التي ترغب في تغطيتها داخل التصنيف المشتق، وبذلك يكون لها سلوكان مختلفان، الأول داخل التصنيف الأساسي والآخر داخل التصنيف المشتق.

ويمكنك التحكم في تغطية الخصائص والوظائف باستخدام كلمات التعديل التالية:

- كلمة **Overridable** وتستخدم مع الخاصية أو الوظيفة داخل التصنيف الأساسي لتعني أنها قابلة للتمثيل مرة أخرى (أي قابلة للتغطية) داخل التصنيفات المشتقة.
- كلمة **Overrides** وتستخدم داخل التصنيفات المشتقة مع الخاصية أو الوظيفة التي ترغب في تغطيتها.
- كلمة **NotOverridable** وتستخدم داخل التصنيف الأساسي مع الخاصية أو الوظيفة التي لا ترغب في إعادة تمثيلها (تغطيتها) داخل التصنيفات المشتقة. ولا داعي لاستخدام هذه الكلمة مع الخصائص والوظائف التي تحتوي على السلوك **Public** لأنها بطبيعتها غير قابلة للتغطية بالتصنيفات المشتقة.
- كلمة **MustOverride** وتستخدم داخل التصنيف الأساسي مع الخاصية أو الوظيفة التي يجب إعادة تمثيلها داخل التصنيفات المشتقة، وفي هذه الحالة يتكون تعريف الوظيفة من العبارة **Sub** أو **Function** أو **Property** فقط ولا يمكن استخدام العبارة **End Sub** أو **End Function** وهذا طبيعي لأن الوظيفة سيتم تعريفها داخل جميع التصنيفات المشتقة. كما يجب تعريف الوظائف التي تحتوي على كلمة **MustOverride** داخل التصنيفات التي تحتوي على كلمة **MustInherit**.

استخدام الكائن MyBase

تستخدم كلمة **MyBase** عند استدعاء الوظائف الموجودة بالتصنيف الأساسي والتي تم تغطيتها داخل التصنيف المشتق. لتوضيح ذلك دعنا نرى الكود التالي:

```
Class DerivedClass Inherits BaseClass
Public Overrides Function CalculateShipping(ByVal Dist As Double, ByVal Rate As Double) As Double
```

استدعاء الوظيفة الموجودة بالتصنيف الأساسي وتعديل قيمتها الناتجة'

```
Return MyBase.CalculateShipping(Dist, Rate) * 2
End Function
End Class
```

في هذا الكود تم اشتقاق التصنيف **DerivedClass** من التصنيف الأساسي **BaseClass** الذي يحتوي 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 الأساسي. ولكي نشير إلى الدالة الموجودة بالتصنيف الأساسي، استخدمنا كلمة **MyBase** والتي تعني أن ما بعدها موجود بالتصنيف الأساسي وليس التصنيف الحالي.

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

- تشير **MyBase** إلى التصنيف الأساسي المباشر وعناصره المشتقة، ولا يمكن استخدامها في 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 في مقارنة داخل عبارة **IS**.
- ليس بالضرورة أن تعرف الوظيفة المشار إليها من خلال كلمة **MyBase** داخل التصنيف الأساسي المباشر وإنما يمكن تعريفها داخل تصنيف أساسي آخر موروث بطريقة غير مباشرة.
- لا يمكنك استخدام **MyBase** في استدعاء وظائف التصنيف الأساسي التي تحتوي على كلمة **MustOverride**.

- لا يمكنك استخدام MyBase في 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 في الإشارة إلى العناصر الموجودة بالتصنيف الأساسي وتحتوي على مستوى الوصول Friend إذا كان التصنيف الأساسي موجوداً داخل تجمع آخر.

استخدام الكائن MyClass

يمكنك استخدام كلمة MyClass في استدعاء وظيفة داخل التصنيف الأساسي قابلة للتغطية Overridable مع التأكد من استدعاء تمثيل الوظيفة داخل التصنيف الأساسي لا داخل أي من التصنيفات المشتقة. ويجب عند استخدام كلمة MyClass مراعاة النقاط الآتية:

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

متى نستخدم التوريث

يتم استخدام التوريث في الحالات التالية:

- حينما يكون التصنيف المشتق أحد أنواع التصنيف الأساسي، فمثلاً نقول الأسد أحد أنواع الحيوانات. في هذه الحالة يمكنك تعريف تصنيف للحيوانات وتصنيف آخر للأسود، على أن يقوم تصنيف الأسود بورثة تصنيف الحيوانات، أو بمعنى آخر يكون تصنيف الأسود مشتقاً من تصنيف الحيوانات. أما إذا كان التصنيف جزءاً من تصنيف آخر، فلا ينصح باستخدام التوريث في هذه الحالة.
 - عند الرغبة في استخدام الكود الموجود بالتصنيفات الأساسية أكثر من مرة داخل التصنيفات المشتقة.
 - عند الرغبة في تطبيق نفس خصائص ووظائف التصنيف ولكن بأنواع بيانات أخرى.
 - عند الرغبة في بناء شكل هرمي للتصنيفات.
 - عند الرغبة في إجراء تعديلات كبيرة على التصنيفات المشتقة عن طريق تعديل التصنيف الأساسي فقط.
- ولتوضيح الفرق بين استخدام الكود العادي واستخدام التوريث، دعنا نرى الكود التالي والذي يتم فيه إنشاء الإجراء (**Draw()**) المستخدم في 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 في هذا الإجراء كما ترى استخدام عبارة **Select** في 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 في هذه الحالة تعديل الكود الأصلي، كما سيحتاج رسم القطع الناقص إلى معامل إضافي نظراً لرسمه بقطرين وليس قطراً واحداً، وهو مالا يتماشى مع خط الرسم. فإذا ما أردنا إضافة إمكانية رسم المضلع، فسنتحتاج إلى معامل آخر. وهنا تظهر التوريث لتحل كل هذه المشاكل، حيث يتم تعريف الوظائف بالتصنيف الأساسي مع ترك تمثيلها داخل التصنيفات المشتقة وهو ما يعطى مرونة شديدة في التعامل مع الكود دون الحاجة إلى تغييره من البداية كما هو الحال في الكود السابق، كود الإجراء (Draw).
لاستخدام التوريث في تمثيل عملية رسم الكائنات، سنقوم بدايةً بتعريف التصنيف الأساسي وليكن باسم Shape ويحتوى على الكود التالي:

```
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 في تعريف حالات (كائنات) جديدة. كما استخدمنا كلمة **MustOverride** مع الإجراء (Draw) للدلالة على وجوب تمثيل هذا الإجراء داخل التصنيفات المشتقة كل 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 في رسم الخطوط المستقيمة كما يلي:

```
Class Line
  Inherits Shape
  Public Length As Integer
  Overrides Sub Draw()
```

يتم هنا وضع الكود المستخدم في رسم الخطوط المستقيمة

```
End Sub
End Class
```

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

```
Class Rectangle
Inherits Line
Public Width As Integer
Overrides Sub Draw()
```

يتم هنا وضع الكود المستخدم في 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

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

لتوضيح ذلك، نفترض أن لدينا وظيفة باسم GetWord() تقوم باسترجاع كلمة داخل عبارة نصية. ونرغب في استخدام هذه الوظيفة في أداء ثلاث حالات مختلفة، الأولى استرجاع أول كلمة في 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 وتحتوي على التوقيع التالي:

```
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 بالعبرة، بينما تحتوى الثانية على معامل من النوع **Integer** لاسترجاع كلمة تبدأ من مكان معين داخل العبرة، كما تحتوى الثالثة على معامل واحد أيضاً ولكن من النوع **String** لاسترجاع كلمة معينة موجودة داخل العبرة. وعند الرغبة في تنفيذ أي 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، في الكود التالي:

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

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

