إخفاء الطرائق وتعدّد الأشكال
Method Hiding and Polymorphism

الكلمات المفتاحية

تعدّد الأشكال، إخفاء الطرائق، تجاوز الطرائق.

ملخص الفصل

نبيّن في هذا الفصل كيفية استخدام الكلمة المفتاحية new لإخفاء أعضاء الصفّ من حقول وصفوف مركبة وطرائق، ونوضّح من خلال عدّة أمثلة الفارق بين تجاوز الطرائق وإخفاء الطرائق.

الأهداف التعليمية

يتعرّف الطالب في هذا الفصل على:

- الفارق بين تعدّد الأشكال الساكن وتعدّد الأشكال الديناميكي
- إخفاء الحقول والصفوف المتداخلة
- إخفاء الطرائق
- الفارق بين إخفاء الطرائق وتجاوز الطرائق

مقدمة

في لغة C#، يَسمح تعدّد الأشكال Polymorphism للصفوف بتنجيز Implementation عدّة طرائق واستدعائها باستخدام نفس الاسم. ويوجد نوعان لتعدّد الأشكال:

- تعدّد الأشكال الساكن Static Polymorphism: حيث يتمّ الربط بين اسم الطريقة والتنجيز الخاص بها خلال مرحلة الترجمة Compile-Time. وكمثال على هذا النوع، التحميل الزائد لطرائق الصف الواحد.
- تعدّد الأشكال الديناميكي Dynamic Polymorphism: وفي هذا النوع، يتمّ الربط بين اسم الطريقة والتنجيز الذي سيتمّ تنفيذه خلال فترة التشغيل Run-Time. وكمثال على هذا النوع، تجاوز طرائق Method Overriding الصف الأساس في الصفوف الوارثة منه.

1. إخفاء الطرائق

عند التصريح عن طريقة ما بأنها افتراضية Virtual، يتحتّم على كل صفٍ مشتقٍ يريد تجاوزها أن يستخدم الكلمة override في توقيع الطريقة. ولكن في بعض الأحيان، قد يتكرّر تعريف طريقة من الصف الأساس ضمن الصف المشتق مع المحافظة على نفس التوقيع، ومع اختلاف جسم الطريقة في الصف المشتق عن جسمها في الصف الأساس. وعند استدعاء الطريقة مع غرض من الصف المشتق، يتمّ تنفيذ جسم الطريقة الموجود في الصف المشتق وكأنّ الطريقة غير موجودة في الصف الأساس. وهذا ما يُسمى بإخفاء الطريقة Method Hiding.

مثال:

نوضّح كيفية استخدام مفهوم إخفاء الطريقة في الرمّاز الآتي:

```
using System;
namespace MethodHiding
{
    class Rectangle
    {
        protected double length, width; // fields
        public Rectangle(double l, double w)
        {   // constructor
            length = l; width = w;
        }
        public void Display()
        {
            Console.WriteLine("  Length: {0}, Width: {1}", length, width);
        }
    }//end class Rectangle
    class TableTop : Rectangle
    {
        private double cost;// price per meter square
        public TableTop(double l, double w, double c) : base(l, w)
        {
            cost = c;
        }
        public double GetCost()
        {
            return length * width * cost;
```

```
        }
        public void Display()
        {
            //use base display and then show more data
            base.Display();
            Console.WriteLine("  Cost: {0}", GetCost());
        }
    }// end class TableTop
    class Test
    {
        static void Main()
        {
            TableTop t = new TableTop(4.5, 7.5, 100.0);
            t.Display();
            Console.ReadKey();
        }// end Main
    }// end class Test
}//end namespace MethodHiding
```

نلاحظ أنّه في الصف Rectangle، توجد الطريقة Display التي تقوم بطباعة طوله length وعرضه width. وفي الصف المشتق TableTop، توجد الطريقة Display التي تطبع الطول والعرض والتكلفة GetCost. وللطريقة Display نفس التوقيع في الصفين، ولكن يختلف سلوكها من صف لآخر. وعند تنفيذ الرمّاز السابق.

يكون الخرج:

```
Length: 4.5, Width: 7.5
Cost: 3375
```

نلاحظ أنّ عملية ترجمة الرمّاز تمّت بنجاح ولكن ظهرت رسالة التحذير:

```
'TableTop.Display()' hides inherited member 'Rectangle.Display()'.
Use the new keyword if hiding was intended.
```

والتي تفيد بأنّ الطريقة TableTop.Display قد أخفت الطريقة Rectangle.Display، وأنّه يجب استخدام الكلمة المفتاحية new مع الطريقة TableTop.Display للدلالة على عملية الإخفاء.
نضيف الكلمة new كما هو مطلوب:

```
new public void Display()
{
    //use base display and then show more data
    base.Display();
    Console.WriteLine("  Cost: {0}", GetCost());
}
```

ثمّ نعيد التنفيذ، فنحصل على نفس الخرج السابق وتختفي رسالة التحذير.

1.1. إخفاء الحقول

لا يقتصر استخدام الكلمة المفتاحية new على إخفاء الطرائق، فيمكن استخدامها لإخفاء الحقول وأعضاء أخرى. وفي حال تمّ استخدام new مع أحد الأعضاء ولا يوجد أي عضو موروث يحمل نفس اسم العضو، سيؤدّي ذلك لإصدار رسالة تحذير عند تنفيذ الرمّاز المصدري.

مثال:

في الرمّاز الآتي، يتمّ إخفاء الحقل الساكن a:

```
using System;
namespace FieldHiding
{
    public class BaseClass
    {
        public static int a = 15;
        public static int b = 30;
    }
    public class DerivedClass : BaseClass
    {
        // Hide field 'a'.
        new public static int a = 800;
        static void Main()
        {
            // Display the new value of a:
            Console.WriteLine(a);

            // Display the hidden value of a:
            Console.WriteLine(BaseClass.a);
            // Display the unhidden member b:
            Console.WriteLine(b);
        }//end Main
    }
}
```

Output:

```
800
15
30
```

2.1. إخفاء الصفوف المتداخلة

يمكن استخدام الكلمة المفتاحية new أيضاً في إخفاء الصفوف المتداخلة.

مثال:

في الرمّاز الآتي، يتمّ إخفاء الصف المتداخل NestedClass:

```
using System;
namespace NestedClassHiding
{
    public class BaseClass
    {
        public class NestedClass
        {
            public int a = 400;
            public int b;
        }
    }
    public class DerivedC : BaseClass
    {
        // Nested type hiding the base type members.
        new public class NestedClass
        {
            public int a = 800;
            public int b;
            public int c;
        }
        static void Main()
        {
            // Creating an object from the overlapping class:
            NestedClass nc1 = new NestedClass();
            // Creating an object from the hidden class:
            BaseClass.NestedClass nc2 = new BaseClass.NestedClass();
            Console.WriteLine(nc1.a);
            Console.WriteLine(nc2.a);
        }//end Main
    }
}
```

Output:

```
800
400
```

إنّ إخفاء الطرائق هو تعدّد أشكال ديناميكي، ولا يقتصر على الطرائق غير الافتراضية، فيمكن تطبيقه على الطرائق الافتراضية أيضاً، أي يمكن استخدام الكلمة المفتاحية new مع المحدِّد virtual للدلالة على تنجيز جديد للطريقة الافتراضية.

2. أمثلة عن تعدّد الأشكال

توجد حالات كثيرة لتعدّد الأشكال بنوعيه الساكن والديناميكي، ولكن ما يهمنا في سياق الفصل الحالي هو توضيح الفارق بين إخفاء الطرائق وتجاوز الطرائق. ولذلك، ستقتصر الأمثلة الآتية على حالة إخفاء طريقة افتراضية، وحالة التجاوز المتسلسل، وأخيراً الإخفاء المتسلسل.

ملاحظة: عند التصريح عن طريقة ما، يجب عدم استخدام المحدّدين new وoverride معاً لأنّ دور كلّ منهما يُعارض دور الآخر، فبينما يقوم new بإنشاء طريقة جديدة تحمل نفس الاسم، يقوم override بتعديل سلوك طريقة موروثة.

لا تقتصر عملية تجاوز الطرائق على الطرائق الافتراضية والمتجاوِزة، وإنّما يتعدى ذلك إلى الطرائق المجرّدة والتي يجب تجاوزها لأنّها لا تمتلك تنجيزاً خاصاً بها، وسيتمّ تناول هذا الموضوع في الفصل القادم.

المثال الأول: إخفاء طريقة افتراضية

في الرمآز الآتي، يتمّ استخدام التجاوز والإخفاء. فالطريقة العامة F موجودة في الصفوف A وB وC وD، ولها نفس التوقيع في الصفوف الأربعة، وD يرث مباشرة C الذي يرث مباشرة B، والذي بدوره يرث مباشرة A. والطريقة F افتراضية في A وC، ويتمّ تجاوزها في B وD. ويتمّ إخفاء F في C باستخدام الكلمة new، أي أنّه أصبح لـ F تنجيزٌ جديدٌ في C. وفي الطريقة Main، تمّ إنشاء الغرض d من D، ثمّ تمّ إنشاء المرجع a من A للدلالة على d، والمرجع b من B للدلالة على d، والمرجع c من C للدلالة على d. وبعد ذلك، تمّ استدعاء الطريقة F من قبل a ثمّ b ثمّ c ثمّ d.

```
using System;
class A
{
    public virtual void F()
    {
        Console.WriteLine("A.F");
    }
}
class B : A
{
    public override void F()
    {
        Console.WriteLine("B.F");
    }
}
class C : B
```

```
{
    new public virtual void F()
    {
        Console.WriteLine("C.F");
    }
}
class D : C
{
    public override void F()
    {
        Console.WriteLine("D.F");
    }
}
class Test
{
    static void Main()
    {
        D d = new D();
        A a = d;
        B b = d;
        C c = d;
        a.F();
        b.F();
        c.F();
        d.F();
    }
}
```

وبعد تنفيذ الرمّاز السابق، يتوقّع البعض أن يتمّ استخدام تنجيز F الخاص بـ a، ثمّ تنجيز F الخاص بـ b، فتنجيز F الخاص بـ c، وأخيراً تنجيز F الخاص بـ d. أي أنّه يجب طباعة الأسطر A.F ثمّ B.F ثمّ C.F ثمّ D.F. ولكن ذلك لم يحدث، وتمّ الحصول على الخرج:

```
B.F
B.F
D.F
D.F
Press any key to continue ...
```

ما الذي حصل أثناء التشغيل لكي يظهر الخرج السابق؟

في حال عدم إهمال علاقات التجاوز والإخفاء التي تربط بين الصفوف، يمكن ملاحظة ما يأتي:

- يحتوي C على طريقة افتراضية مصدرها A (المحدّد virtual في A والمحدّد override في C)، ويحتوي D على طريقة افتراضية مصدرها C (المحدّد virtual في C والمحدّد override في D). وتخفي الطريقة التي مصدرها C الطريقة التي مصدرها A باستخدام الكلمة new، ولهذا السبب، التجاوز المستخدم في D سيُطبّق على الطريقة التي مصدرها C، ولا يمكن تطبيقه على الطريقة التي مصدرها A.
- يحتوي الغرض a من الصف A مرجعاً للغرض d من الصف D، فعند استدعاء الطريقة a.F، سيتّضح أنّها افتراضية في A، وسيتمّ البحث عن توفّر طريقة متجاوزة لها خلال هرمية الوراثة بين A وD ليتمّ تنفيذها. وفي الصف B، توجد طريقة متجاوزة للطريقة a.F. وبما أنّ الصف C قد أخفى الطريقة، يتمّ اعتبار الطريقة C.F جديدة في هذه الحالة، وسيتمّ تنفيذ الطريقة F الموجودة في B، ولذلك تمّت طباعة B.F.
- يحتوي الغرض b من الصف B مرجعاً لنفس الغرض d من الصف D المُشار إليه من قبل a، وعند استدعاء الطريقة b.F، سيتّضح بأنّه تمّ تجاوزها في B (أي تمّ تغيير سلوكها إلى سلوك خاص بـ B)، وسيتمّ تنفيذ الطريقة F الموجودة في B، ولذلك تمّت طباعة B.F.
- يحتوي الغرض c من الصف C مرجعاً لنفس الغرض d من الصف D المُشار إليه من قبل a وc، وعند استدعاء الطريقة c.F، سيتّضح أنّها افتراضية في C وسيتمّ البحث عن توفّر طريقة متجاوزة لها خلال هرمية الوراثة بين C وD ليتمّ تنفيذها. سيتّضح بأنّه تمّ تجاوزها في D، وسيتمّ تنفيذ الطريقة F الموجودة في D، ولذلك تمّت طباعة D.F.
- الغرض d منتسخ من الصف D، وعند استدعاء الطريقة d.F، سيتّضح بأنّه تمّ تجاوزها في D (أي تمّ تغيير سلوكها إلى سلوك خاص بـ D)، وسيتمّ تنفيذ الطريقة F الموجودة في D، ولذلك تمّت طباعة D.F.

المثال الثاني: التجاوز المتسلسل

نعدِّل الصف C في الرمآز الوارد في المثال الأول كما يأتي:

```
class C : B {
    public override void F( ) {
        Console.WriteLine("C.F");
    }
}
```

أي ألغينا الإخفاء عن الطريقة C.F ولم تعد طريقة افتراضية وإنّما أصبحت طريقة متجاوزة مثل الطرائق B.F و D.F، وبقيت الطريقة A.F فقط افتراضية، وهي مصدر الطرائق الثلاث المتجاوزة.

وبعد التنفيذ، يتمّ الحصول على الخرج الآتي:

ولتفسير الخرج السابق، نورد ما يأتي:

```
D.F
D.F
D.F
D.F
Press any key to continue ...
```

عند استدعاء الطريقة a.F، سيتّضح أنّها افتراضية في A، وسيتمّ البحث عن توفّر طريقة متجاوزة لها خلال هرمية الوراثة بين A وD ليتمّ تنفيذها، وسيتمّ المرور بـ B.F ولكنّها متجاوزة (أي يوجد احتمال لتغيير سلوكها)، ثمّ C.F ولكنّها متجاوزة، ثم D.F وهي متجاوزة وهي في آخر هرمية الوراثة، ولذلك تمّت طباعة D.F.

- عند استدعاء الطريقة b.F، سيتّضح بأنّها متجاوزة، وسيتمّ البحث عن توفّر طريقة متجاوزة لها خلال هرمية الوراثة بين B وD ليتمّ تنفيذها، سيتمّ المرور بـ C.F ولكنّها متجاوزة، ثمّ D.F وهي متجاوزة وهي في آخر هرمية الوراثة، ولذلك تمّت طباعة D.F.
- عند استدعاء الطريقة c.F، سيتّضح أنّها متجاوزة، وسيتمّ البحث عن توفّر طريقة متجاوزة لها خلال هرمية الوراثة بين C وD ليتمّ تنفيذها. سيتّضح بأنّه تمّ تجاوزها في D، وسيتمّ تنفيذ الطريقة D.F وهي في آخر هرمية الوراثة، ولذلك تمّت طباعة D.F.
- الغرض d منتسخ من الصف D، وعند استدعاء الطريقة d.F، سيتّضح بأنّه تمّ تجاوزها في D (أي تمّ تغيير سلوكها إلى سلوك خاص بـ D)، وسيتمّ تنفيذ الطريقة F الموجودة في D، ولذلك تمّت طباعة D.F.

المثال الثالث: الإخفاء المتسلسل

في الرمّاز الآتي، يتمّ استخدام التجاوز والإخفاء. فالطريقة العامة M موجودة في الصفوف A وB وC وD، ولها نفس التوقيع في الصفوف الأربعة. وD يرث مباشرة C الذي يرث مباشرة B والذي بدوره يرث مباشرة A. والطريقة M افتراضية في A، ويتمّ إخفاؤها في B وC وD باستخدام الكلمة new. أي أنّه أصبح لـ F أربع تنجيزات، وفي كلّ صف لها تنجيز مختلف عن البقية. وفي الطريقة Main، تمّ إنشاء الغرض d من D، ثمّ تمّ إنشاء المرجع c من C للدلالة على d، والمرجع b من B للدلالة على c، والمرجع a من A للدلالة على b. وبعد ذلك، تمّ استدعاء الطريقة M من قبل a ثمّ b ثمّ c ثمّ d.

```
using System;
namespace HidingSequence
{

    class A
    {
        public virtual void M()
        {
            Console.Write("A");
        }
    }
    class B : A
    {
        public new void M()
        {
            Console.Write("B");
        }
    }
    class C : B
    {
        public new void M()
        {
            Console.Write("C");
        }
    }
    class D : C
    {
        public new void M() { Console.Write("D"); }
        static void Main()
        {
            D d = new D();
            C c = d; B b = c; A a = b;
            a.M(); b.M(); c.M(); d.M();
            Console.ReadKey();
        } //end main }
    }
}
```

وبعد التنفيذ، نحصل على الخرج:

```
ABCD
```

ويُفسَّرُ ذلك بأنّه من أجل كلّ غرض تمّ استدعاء التنجيز المتاح في صفّه، وذلك لأنّ كلّ صف يوفّر تنجيزاً جديداً للطريقة M مستقلاً عن التنجيزات الأخرى، فلا حاجة للبحث عن تنجيزات أخرى.

الأنشطة المرافقة

التمرين الأول

بعد تنفيذ الرمّاز الآتي، حصلنا على الخرج الموضّح بجانبه، فسّر كيفية الحصول عليه:

```
using System;
namespace MethodHidingExample
{
    class A { public virtual void M() { Console.Write("B"); }
}
    class B : A { public override void M() { base.M(); } }
    class C : B { new public void M() { base.M(); } }
    class D : C
    {
        new public void M() { base.M(); }
        static void Main()
        {
            D d = new D();
            C c = d; B b = c; A a = b;
            a.M(); b.M(); c.M(); d.M();
            Console.ReadKey();
        } //end main
    }
}
```

Output:

```
BBBB
```

التمرين الثاني

اختر الإجابة الصحيحة الدالّة على الخرج الذي سيظهر بعد تنفيذ الرمّاز الآتي، وعلِّل إجابتك:

```
using System;
namespace MethodHidingExample {
class SBad  {
public virtual string M1() { return "Bad.M1"; }
public virtual string M2() { return M1(); }
public virtual string M3() { return "Bad.M3"; }
}
class SSad : SBad {
public override string M2() { return "Sad.M2"; }
public override string M3() { return "Sad.M3"; }
}
class Test  {
static void Main() {
SBad var2 = new SSad();
Console.WriteLine(var2.M1());
Console.ReadKey();
}
}
}
```

Output:

1- Sad.M2
2- Bad.M3
3- Bad.M1
4- Sad.M3
5- Compiler error

المراجع

1. https://docs.microsoft.com/en-us/dotnet/csharp/
2. Dan Clark: Beginning C# Object-Oriented Programming, Berkeley, CA, Apress, 2013
3. “The C# Programming Language”, 4th Edition, Anders Hejlsberg, et al. 2010