

الباب الثامن
الوصلات البينية
(Interfaces)

محتويات الباب

- تعريف الوصلات البيئية (Interfaces)
- إعلان الوصلة البيئية
- تطبيق الوصلة البيئية
- التطبيق الصريح للوصلة البيئية
- استخدام المؤثر is لاختبار النمط
- استخدام المؤثر as لاختبار النمط
- إخفاء الأعضاء الموروثة (new)

تعريف الوصلات البينية (Interfaces)

نتعامل فى الحياة اليومية مع الكثير من الوصلات البينية للأجهزة مثل التليفزيون وجهاز التسجيل والفيديو. ولو أنك تجولت فى أحد المحلات بحثاً عن جهاز تسجيل فى المعارضات ، فإنك لن تحتاج إلى فتح جهاز التسجيل لى تعرف إمكاناته. إن الواجهة أو الوصلة البينية تخبرك عما يحتويه الجهاز. فتستطيع مثلاً أن تحدد إذا كان الجهاز يستخدم ميكروفون خارجى للتسجيل أو يمكنه نسخ الشرائط وهكذا. ولو قررت الشركة المصنعة إضافة أو حذف بعض إمكانات الجهاز، فإن ذلك سوف يظهر بوضوح فى الوصلة البينية.

وفى مجال الفصائل والمنشآت ، فإنك تستطيع بمجرد إلقاء نظرة على الوصلة البينية أن تعرف ما تحتويه الفصيلة التى تطبقها. أى أن الوصلة البينية عبارة عن العقد الذى تستطيع من خلاله أن تعرف سلوك الفصيلة أو المنشأ. وعندما نصف الوصلة البينية بأنها عقد ، فهذا يوضح جانب الإلزام. بمعنى أن الشركة لا يجوز لها أن تتضمن جهاز التسجيل شكل الميكروفون دون الدائرة الإلكترونية التى تعمل فى الخلفية. كذلك فلو كانت الوصلة البينية محتوية على اسم أسلوب ما ، فمن اللازم لأى فصيلة أو منشأ تطبق هذه الوصلة أن تتضمن تطبيق هذا الأسلوب.

إعلان الوصلة البينية

يتم إعلان الوصلة البينية طبقاً للصورة الآتية:

```
[attributes] [modifiers] interface identifier [:base-list] {interface-body};
```

حيث:

- attributes: صفات إضافية اختيارية
 - modifiers: معدلات اختيارية ، وتشمل الكلمة new و توليفة مسموح بها من معدلات التوصل.
 - identifier: اسم الوصلة البينية
 - base-list: أسماء الوصلات البينية التي ترثها الوصلة مفصولة بفاصلة ، وهي اختيارية.
 - interface-body: جسم الوصلة و يحتوى على إعلانات أعضائها.
- وتحتوى الوصلة البينية على إعلانات الأعضاء فقط دون التطبيق ، وعلى سبيل المثال:

```
interface ICounter  
{  
    void Count(int i);  
    int SetCounter();  
}
```

كما نرى أن الإعلان يحتوى على أسماء وأنماط الأساليب وأنماط البارامترات المستخدمة بها. وهذا يشبه عينات الدوال (Prototypes) فى

لغة سي++ ، أو الميّا داتا. أما تطبيق الأساليب فيقع بداخل الفصائل أو المنشآت التي تطبق الوصلات البيئية.

ومن الجائز أن ترث الوصلة البيئية أكثر من وصلة بيئية كالمثال الآتي (وفي مجال الوصلات البيئية فإن الوراثة والتطبيق لهما نفس المعنى):

```
interface IComboBox: ITextBox, IListBox
{
    أعضاء الوصلة البيئية //
}
```

معنى ذلك أن الوصلة IComboBox ترث أعضاء الوصلات البيئية: ITextBox و IListBox.

ويجوز أن تحتوى الوصلة البيئية على أى من الدوال الأعضاء الآتية:

- أساليب (Methods)
 - خواص (Properties)
 - فهارس (Indexers)
 - أحداث (Events) – سيلي شرحها
- ولا يجوز أن تحتوى الوصلة البيئية على حقول.

تطبيق الوصلة البيئية (Interface Implementation)

يجوز تطبيق الوصلة البيئية بواسطة فصيلة أو منشأ. ويتم تطبيق الوصلة كالمثال الآتي:

```
class MyDerivedClass: IMyInterface1
{
```

```
// التطبيق  
}
```

وبموجب هذا الإعلان فإنه يصبح إلزاماً على الفصيلة MyDerivedClass أن تطبق أعضاء الوصلة البينية IMyInterface1. ومن الجائز تطبيق أكثر من وصلة بينية في نفس الفصيلة أو المنشأ كالمثال الآتي:

```
class MyDerivedClass: IMyInterface1, IMyInterface2  
{  
    // التطبيق  
}
```

كما يجوز تطبيق وصلة بينية أو أكثر ، ووراثة فصيلة في نفس الوقت كالمثال الآتي:

```
class MyDerivedClass: MyBaseClass, IMyInterface1  
{  
    // التطبيق  
}
```

ملاحظة:

لاحظ أن صيغة الوراثة للفصيلة هي نفسها صيغة تطبيق الوصلة البينية ولكن المعنى يختلف.

مثال (8-1)

يوضح المثال التالي وصلة بينية بالاسم IPoint تتضمن الخواص المستخدمة في فصيلة النقطة Point. وكما نرى في المثال أن الوصلة البينية لا تحتوى على تطبيق الخواص ، كما أنها لا تحتوى على أية حقول حيث جاءت إعلانات الحقول بداخل الفصيلة.

```
// Example 8-1.cs
// interface example

using System;

interface IPoint
{
    // Property signatures:
    int Myx
    {
        get; set;
    }
    int Myy
    {
        get; set;
    }
}

class Point : IPoint
{
    private int x;
    private int y;

    public Point(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    // Property implementation: تطبيق الخاصية بالفصيلة
    public int Myx
    {
        get {return x;}
        set {x = value;}
    }
    public int Myy
    {
        get {return y;}
        set {y = value;}
    }
}
}
```

```
class MyClass
{
    static void Main()
    {
        Point myPoint = new Point(12,300);
        Console.WriteLine("My point is created at: ");
        DisplayMyPoint(myPoint);
    }
    static void DisplayMyPoint(IPoint myPoint)
    {
        Console.WriteLine("{0},{1}", myPoint.Myx, myPoint.Myy);
    }
}
```

تنفيذ البرنامج:

```
My point is created at: (12,300)
```

ملاحظات على البرنامج السابق:

لاحظ أن الفصيلة Point لا بد أن تطبق محتويات الوصلة البينية IPoint ، وإلا فإن المترجم يعترض. يتضح لنا من ذلك أن وظيفة الوصلة البينية الأساسية هي إلزام الفصائل الوارثة بعقد معين في تطبيقها. لاحظ أيضاً إمكان إمرار بارامتر من نمط الوصلة البينية IPoint إلى الأسلوب DisplayMyPoint. وبالطبع يمكنك أيضاً أن تمرر بارامتر من النمط Point.

التطبيق الصريح للوصلة البينية (Explicit Interface Implementation)

فلنعتبر الفصيلة `MyClass` ؛ لو كانت هذه الفصيلة تطبق وصلة بينية `IMyInterface` فإنه يمكننا تطبيق أعضاء الوصلة بداخل الفصيلة كالمثال الآتي:

```
string IMyInterface.MyMethod()  
{  
    // التطبيق  
}
```

وكما نرى في هذا المثال أنه قد تم تأهيل اسم العضو `MyMethod()` باسم الوصلة البينية `IMyInterface`. ويسمى هذا بالتطبيق الصريح للوصلة البينية.

ولنفرض أننا خلقنا هدفاً من الفصيلة `MyClass` كالاتي:

```
MyClass mc = new MyClass();
```

يمكننا أيضاً أن نخلق هدفاً من الوصلة البينية ، ويتم إعلان الهدف باستخدام الإسقاط كالمثال الآتي:

```
IMyInterface mi = (IMyInterface) mc;
```

وفي هذه الحالة فإن التوصل إلى أحد الأعضاء (مثل `MyMethod()`) يتم من خلال هدف الوصلة `mi` ، وعلى سبيل المثال:

```
Console.WriteLine(mi.MyMethod());
```

بل إنه من الخطأ – فى هذه الحالة – محاولة التوصل إلى العضو عن طريق هدف الفصيلة مثل:

```
Console.WriteLine(mc.MyMethod()); // error
```

إن هذا يؤدي إلى خطأ فى الترجمة.

متى نحتاج إلى مثل هذا الكود؟

هناك بعض الحالات التى يصبح استخدام التطبيق الصريح للوصلة ضرورياً. وعلى سبيل المثال قد تطبق الفصيلة وصلتين مختلفتين فى نفس الوقت. وقد يتصادف أن تحتوى كل وصلة على أساليب تحمل نفس الأسماء (مثل MyMethod()). فى مثل هذه الأحوال يصبح التطبيق الصريح للوصلة ضرورياً. فإذا كانت هدف الوصلة الأولى هو mi1 وهدف الوصلة الثانية هو mi2 ، فإن التوصل للأسلوب عن طريق هدف الوصلة البينية لا يحتمل أى لبس ، أى:

```
Console.WriteLine(mi1.MyMethod());
```

```
Console.WriteLine(mi2.MyMethod());
```

ولكن استخدام هدف الفصيلة سوف يؤدي إلى لبس بالتأكد.

مثال (2-8)

فى هذا المثال نقدم برنامجاً لتحويل درجات الحرارة من مئوى إلى فهرنهايت وبالعكس. والبرنامج يحتوى على الوصلتين ITemp1 و ITemp2. وتحتوى كل منهما على أسلوب بالاسم Convert. وتقوم الفصيلة TempConverter بتطبيق الوصلتين تطبيقاً صريحاً. وكما نرى

في الأسلوب الرئيسي (Main) أننا قد أنشأنا هدفاً من كل وصلة بينية استخدمناه في التوصل إلى عضو الوصلة المعينة.

```
// Example 8-2.cs
// Explicit interface implementation example.

using System;

public interface ITemp1 // الوصلة الأولى
{
    double Convert(double d);
}

public interface ITemp2 // للوصلة الثانية
{
    double Convert(double d);
}

public class TempConverter: ITemp1, ITemp2
{
    // التطبيق الصريح للوصلة الأولى
    double ITemp1.Convert(double d)
    {
        // Convert to Fahrenheit:
        return (d * 1.8) + 32;
    }

    // لتطبيق الصريح للوصلة الثانية
    double ITemp2.Convert(double d)
    {
        // Convert to Celsius:
        return (d - 32) / 1.8;
    }
}

class MyClass
{
    public static void Main()
```

```
{
// Create a class instance: هدف الفصيلة
TempConverter cObj = new TempConverter();

// Create instances of interfaces
// Create a From -Celsius-to-Fahrenheit object: هدف الوصلة الأولى
ITemp1 iCF = (ITemp1) cObj;
// Create From -Fahrenheit-to-Celsius object: هدف الوصلة الثانية
ITemp2 iFC = (ITemp2) cObj;

// Initialize variables:
double F = 32;
double C = 20;

// Print results:
// تحويل من مئوية إلى فهرنهايت
Console.WriteLine("Temperature {0} C in Fahrenheit:
{1:F2}",C, iCF.Convert(C));
// تحويل من فهرنهايت إلى مئوية
Console.WriteLine("Temperature {0} F in Celsius: {1:F2}", F,
iFC.Convert(F));
}
}
```

تنفيذ البرنامج:

```
Temperature 20 C in Fahrenheit: 68
Temperature 32 F in Celsius: 0
```

تدريب (8-1)

عدّل المثال السابق لكي يستقبل درجات الحرارة من لوحة الأزرار بأحد النظامين ويطبّع الدرجة بالنظام الآخر.

استخدام المؤثر is لاختبار النمط

يستخدم المؤثر `is` في اختبار النمط للأهداف المختلفة أثناء تشغيل البرنامج. وهو يستخدم في صورة تعبير بولياني بالصورة:

```
expression is type
```

حيث `type`: نمط مرجع.

ونتيجة هذا التعبير هي `true` أو `false`. وعلى سبيل المثال فإن التعبير الآتي:

```
myObj is MyClass
```

يستخدم لمعرفة إذا ما كان الهدف `myObj` يتبع الفصيلة `MyClass`. ونتيجة التعبير إما `true` في حالة النجاح (الإيجاب) أو `false` في حالة الفشل (النفى).

أما التعبير:

```
myObj is IMyInterface
```

فيستخدم لاختبار إذا ما كان الهدف `myObj` يتبع فصيلة تطبق الوصلة البينية `IMyinterface`. وهو أيضاً يرجع أحد القيمتين `true` أو `false`.

مثال (8-3)

في هذا المثال نستخدم المؤثر `is` لاختبار نمط هدف ينتمي إلى الفصيلة `MyClass` والوصلتين `I1` و `I2` وذلك باستخدام تعبير شرطي واحد.

```
// Example 8-3.cs
// is operator example

using System;

interface I1
{
```

```
}  
interface I2  
{  
}  
class Class1: I1, I2  
{  
}  
class MyClass  
{  
    static bool TestType(object obj)  
    {  
        if (obj is I1 | obj is I2 | obj is Class1)  
            return true;  
        else  
            return false;  
    }  
    public static void Main()  
    {  
        Class1 c = new Class1();  
        Console.WriteLine(TestType(c));  
    }  
}
```

تنفيذ البرنامج:

True

استخدام المؤثر as لاختبار النمط

يمثل المؤثر as في استخدامه المؤثر is ، وهو يأخذ الصورة:

expression as type

حيث type: نمط مرجع.

والنتيجة ترجع قيمة التعبير (expression) في حالة النجاح أو ترجع القيمة null في حالة الفشل. وهو يماثل عملية الإسقاط لكنه لا يتسبب في حدوث خطأ استثنائي عند الفشل.

وهذا التعبير يكافئ التعبير الشرطي:

expression is type ? (type)expression : (type)null

مثال (8-4)

في هذا المثال نستخدم الأسلوب TestType() لاختبار الأهداف. وهو يقرر إذا كان الهدف محل الاختبار من النمط string أم لا.

```
// Example 8-4.cs
// The as operator

using System;

public class MyClass
{
    static void TestType(object o)
    {
        if (o as string != null)
            Console.WriteLine ( "The object  \"{0}\" is a string.", o);
        else
            Console.WriteLine ( "The object  \"{0}\" is not a string.", o);
    }

    static void Main()
    {
        object o1 = "Hello World!";
        object o2 = 123;
        TestType(o1);
        TestType(o2);
    }
}
```

تنفيذ البرنامج:

```
The object "Hello World!" is a string.  
The object "123" is not a string.
```

تدريب (8-2)

عدّل البرنامج السابق لكي يطبع النتيجة كالمثال الآتي:

The object "Hello World!" is a string.

The object "123" is not a string. It is System.Int32.

The object "12.34" is not a string. It is System.Double.

إخفاء الأعضاء الموروثة (new)

استخدمنا من قبل المؤثر **new** في إعلان الأهداف. أما الاستخدام الثاني للكلمة **new** فهو استخدامها كمعدّل (Modifier) في إعلان الأعضاء التي تحمل نفس الأسماء كما الأعضاء الموروثة من فصيلة الأساس ، وذلك بغرض إخفاء العضو الموروث. وعلى سبيل المثال ، لو كان لدينا الفصيلة الآتية:

```
public class MyBaseClass // فصيلة الأساس  
{  
    public int myInt;  
    public void MyMethod() { }  
}
```

عندما نورث هذه الفصيلة فإن الفصيلة المشتقة سوف ترث جميع الأعضاء. ولنفرض أننا نرغب في إعلان عضو جديد بالفصيلة المشتقة يحمل نفس الاسم **MyMethod()** لأداء وظيفة مختلفة تماماً ، فإننا عندئذ نستخدم المعدّل **new** في إعلان العضو بالصورة الآتية:

```
public class MyClass: MyBaseClass // الفصيلة المشتقة
```

```
{  
  العضو الموروث – يحمل نفس الاسم // new public void MyMethod() { }  
}
```

أى أن المعدل new يؤدي إلى إخفاء أى العضو الذى يحمل نفس الاسم بالفصيلة الموروثة. كما أن الأسلوب الذى يستخدم المعدل new يخفى الخصائص والحقول والأنماط التى تحمل نفس الاسم كما يخفى الأساليب التى تحمل نفس التوقيع.

ولو أنك ترجمت هذه الفصائل (المذكورة بأعلاه) بدون استخدام الكلمة new فإن المترجم سوف يرسل رسالة تحذير يطلب منك تأكيد نيتك فى إخفاء عضو الفصيلة الأساسية ، وذلك باستخدام الكلمة new:

```
warning CS0108: The keyword new is required on  
'MyClass.MyMethod()' because it hides inherited member  
'MyBaseClass.MyMethod()'
```

ولا يجوز استخدام المعدل new مع المعدل override فى نفس الإعلان فهذا يؤدي إلى خطأ فى الترجمة. ومع ذلك فإن استخدام المعدل new مع المعدل virtual مسموح به ، وهو يعنى تأكيد نية المبرمج على إخفاء العضو موروث وبدء نقطة تخصص جديدة فى شجرة الوراثة جديدة.

تطوير طرازات البرامج (Versioning)

إن إمكانية إخفاء الأعضاء الموروثة تشبه إمكانية ركوب الأساليب الافتراضية (بالمعدل override) ، وكلاهما تفيد فى تصميم طرازات جديدة من البرامج مع حفظ التوافق مع الطرازات السابقة من البرنامج. وعلى سبيل المثال ، فلنفرض أنك تستخدم فصيلة أنتجتها شركة الكمبيوتر المصرية (ECC) بالاسم EccClass كالاتى:

```
public class EccClass { ... }  
public class MyClass: EccClass { ... }
```

وقد استدعتك الحاجة أثناء تطبيق الفصيلة الموروثة أن تضيف الأسلوب
MyMethod كالاتى:

```
public class MyClass: EccClass  
{  
    public virtual MyMethod() { ... } // أسلوب جديد فى برنامجك  
}
```

وقد كان برنامجك يعمل على ما يرام حتى أصدرت شركة الكمبيوتر
المصرية طرازاً جديداً من برنامجها متضمناً الأسلوب MyMethod
(الذى يؤدي نفس العمل) بالصورة الآتية:

```
public class EccClass  
{  
    public virtual MyMethod() { ... } // أسلوب جديد بالطراز الثانى  
    ...  
}
```

فى هذه الحال سوف تحدث مشكلة ، ولكن الحل فى استخدام الكلمة new
. إن استخدامك للكلمة new لإعلان الأسلوب MyMethod فى برنامجك
يؤكد نيتك على إخفاء الأسلوب الذى يحمل نفس الاسم بالفصيلة
EccClass ، كما يمنع رسالة التحذير من المترجم ، أى:

```
public class MyClass: EccClass  
{  
    public new virtual MyMethod() { ... }  
}
```

كما يمكنك كبتيل استخدام المعدل override إذا كان فى نيتك ركوب
الأسلوب الموجود بفصيلة الأساس.

مثال (8-5)

في هذا المثال ترث الفصيلة MyDerivedClass الفصيلة MyBaseClass. وتحتوي كل من الفصيلتين على فصيلة عضوة بالاسم MyClass. ولكي نجعل الفصيلة الموجودة في الفصيلة المشتقة تخفي الأخرى بالفصيلة الأساسية، فقد استخدمنا المؤثر new كمعدل للإعلان. وقد أمكن التوصل إلى الأعضاء بكل من الفصيلتين MyClass والتعامل مع أعضائهما.

```
// 8-5.cs
// Example on hiding members using "new"

using System;

public class MyBaseClass
{
    public class MyClass
    {
        public int myInt = 123;

        public virtual string MyMethod()
        {
            return "Hello from the Base class!";
        }
    }
}

public class MyDerivedClass : MyBaseClass
{
    // The following nested class hides the base class member:
    new public class MyClass
    {
        public int myInt = 321;

        public virtual string MyMethod()
```

```

    {
        return "Hello from the Derived class!";
    }
}

static void Main()
{
    // Create an object from the "new" MyClass:
    MyClass myObj1 = new MyClass();

    // Create an object from the hidden MyClass :
    MyBaseClass.MyClass myObj2 = new MyBaseClass.MyClass();

    Console.WriteLine("Value from the 'new' MyClass: {0}",
myObj1.myInt);
    Console.WriteLine("Value from the 'hidden' MyClass:
{0}",myObj2.myInt);

    Console.WriteLine("Message from the 'new' MyClass: {0}",
myObj1.MyMethod());
    Console.WriteLine("Message from the 'hidden' MyClass: {0}",
myObj2.MyMethod());
}
}

```

تنفيذ البرنامج:

```

Value from the 'new' MyClass: 321
Value from the 'hidden' MyClass: 123
Message from the 'new' MyClass: Hello from the Derived class!
Message from the 'hidden' MyClass: Hello from the Base class!

```

ملاحظات على البرنامج السابق:

لاحظ أننا خلقنا هدفاً من كل فصيلة من الفصيلتين التي تحمل الاسم MyClass. وبالرغم من أن كل فصيلة تحتوي على حقل بنفس الاسم وأسلوب بنفس الاسم ، ولكن لم يلزمنا استخدام المعدل new إلا مرة واحدة عند إعلان الفصيلة التي تحتويهما.

إخفاء أعضاء الوصلات البينية

يمكن تطبيق هذا المبدأ على الوصلات البينية أيضاً. فمثلاً الإعلان التالي يستخدم خاصية بالاسم M1:

```
interface IBase
{
    int M1 { get; set; }
}
```

ويمكنك في نفس البرنامج أن تستخدم الاسم M1 في إعلان أسلوب بالوصلة البينية المشتقة (IDerived) كالتالي:

```
interface IDerived: IBase
{
    new int M1();
}
```

في هذه الحالة فإن العضو M1 بالوصلة المشتقة يخفي العضو M1 بالوصلة الموروثة. وهنا يصبح استخدام التطبيق الصريح لأعضاء الوصلة البينية ضرورياً:

```
class MyClass: IDerived
{
    private int m1;
    // تطبيق صريح للخاصية
    int IBase.M1
    {
        get { return m1; }
        set { m1 = value; }
    }
    // تطبيق صريح للأسلوب
    void IDerived.M1() { }
}
```

كما يمكنك تطبيق الخاصية M1 تطبيقاً صريحاً وتطبيق الأسلوب M1 تطبيقاً معتاداً:

```
class MyClass: IDerived
{
    private int m1;
    // تطبيق صريح للخاصية
    int IBase.M1
    {
        get { return m1; }
        set { m1 = value; }
    }
    // تطبيق عادي للأسلوب
    public void M1() { }
}
```

ومن الجائز أيضاً استخدام التطبيق الصريح للأسلوب والتطبيق العادي للخاصية كالاتي:

```
class MyClass: IDerived
{
    private int m1;
    // تطبيق عادي للخاصية
    public int M1
    {
        get { return m1; }
        set { m1 = value; }
    }
    // تطبيق صريح للأسلوب
    void IDerived.M1() { }
}
```