الباب الثامن

الوصـــلات البينية

(Interfaces)

محتويات الباب

- تعريف الوصلات البينية (Interfaces)
- إعلان الوصلة البينية
- تطبيق الوصلة البينية
- التطبيق الصريح للوصلة البينية
- 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 المعروضات ، فإنك لن تحتاج إلى فتح جهاز التسجيل لكى 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 تضمّن جهاز التسجيل شكل الميكروفون دون الدائرة الإليكترونية 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 التى ترثها الوصلة مفصولة بفاصلة ، وهى 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 سى++ ، أو الميتا داتا. أما تطبيق الأساليب فيقع بداخل الفصائل أو المنشآت التى تطبق الوصلات البينية.

ومن الجائز أن ترث الوصلة البينية أكثر من وصلة بينية كالمثال الآتى (وفى مجال الوصلات البينية فإن الوراثة والتطبيق لهما نفس المعنى):

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

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

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

- أساليب (Methods)
- خواص (Properties)
- فهارس (Indexers)
- أحداث (Events) ـ سيلى شرحها

ولا يجوز أن تحتوى الوصلة البينية على حقول.

تطبيق الوصلة البينية (Interface Implementation)

يجوز تطبيق الوصلة البينية بواسطة فصيلة أو منشأ. ويتم تطبيق الوصلة كالمثال الآتى:

```
class MyDerivedClass: IMyInterface1
{
```

```
  // التطبيق
}
```

وبموجب هذا الإعلان فإنه يصبح إلزاما على الفصيلة MyDerivedClass أن تطبق أعضاء الوصلة البينية IMyInterface1.

ومـــن الجائـــز تطبيق أكثر من وصلة بينية فى نفس الفصيلة أو المنشأ كالمثال الآتى:

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

كما يجوز تطبيق وصلة بينية أو أكثر ، ووراثة فصيلة فى نفس الوقت كالمثال الآتى:

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

ملاحظة:

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

مثال (1-8)

يوضـــح المـــثال الـــتالى وصـــلة بينية بالاسم IPoint تتضمن الخواص المســـتخدمة فى فصيلة النقطة Point. وكما نرى فى المثال أن الوصلة البينـــية لا تحـــتوى 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.Write("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 لابد أن تطبق محتويات الوصلة البينية IPoint ، وإلاّ فإن المترجم يعترض. يتضح لنا من ذلك أن وظيفة الوصلة البينية الأساسية هى إلزام الفصائل الوارثة بعقد معين فى 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 الآتى:

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

وكما نرى فى 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 كالآتى:

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

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

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

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

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

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

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

إن هذا يؤدى 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.Write(mi1. MyMethod());
Console.Write(mi2. MyMethod());
```

ولكن استخدام هدف الفصيلة سوف يؤدى إلى لبس بالتأكيد.

مثال (2-8)

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

فـــى الأســـلوب الرئيسى (Main) أننا قد أنشأنا هدفاً من كل وصلة بينية استخدمناه فى 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  // الوصلة 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;
  }
  // التطبيق 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 مئوى إلى فهرنهيت
    Console.WriteLine("Temperature {0} C in Fahrenheit:
{1:F2}",C, iCF.Convert(C));
    //  تحويل من فهرنهيت إلى مئوى
    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 لكى يستقبل درجات الحرارة من لوحة الأزرار بأحد النظامين ويطبع الدرجة بالنظام الآخر.

استخدام المؤثر is لاختبار النمط

يســـتخدم المؤثـــر **is** فـــى اختبار النمط للأهداف المختلفة أثناء تشغيل البرنامج. وهو يستخدم فى صورة تعبير بوليانى بالصورة:

```
expression is type
```

حيث type: نمط مرجع.

ونتـــيجة هذا التعبير هى true أو false. وعلى سبيل المثال فإن التعبير الآتى:

```
myObj is MyClass
```

يســـتخدم لمعـــرفة إذا مـــا كان الهدف myObj يتبع الفصيلة MyClass.

ونتـــيجة التعبير إما true فى حالة النجاح (الإيجاب) أو false فى حالة الفشل (النفى).

أما التعبير:

```
myObj is IMyInterface
```

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

مثال (3-8)

فـــى هذا المثال نستخدم المؤثر is لاختبار نمط هدف ينتمى إلى الفصيلة MyClass والوصلتين I1 و I2 وذلك باستخدام تعبير شرطى 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)
      ret  urn true;
    else
      return false;
  }
  public static void Main()
  {
    Class1 c = new Class1();
    Console.WriteLine(TestType(c));
  }
}
```

تنفيذ البرنامج:

```
True
```

استخدام المؤثر as لاختبار النمط

يماثل المؤثر **as** فى استخدامه المؤثر is ، وهو يأخذ الصورة:

```
expression as type
```

حيث type: نمط مرجع.

والنتـــيجة تـــرجع قيمة التعبير (expression) فى حالة النجاح أو ترجع القيمة null فى حالة الفشل. وهو يماثل عملية الإسقاط لكنه لا يتسبب فى حدوث خطأ استثنائى عند الفشل.

وهذا التعبير يكافىء التعبير الشرطى:

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

مثال (4-8)

فى 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()
  {
    obje ct 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 (2-8)

عدّل البرنامج السابق لكى يطبع النتيجة كالمثال الآتى:

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** فى إعلان الأهداف. أما الاستخدام الثانى للكلمة new فهو استخدامها كمعدّل (Modifier) فى إعلان الأعضاء التى 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 فى إعلان عضو جديد بالفصيلة المشتقة يحمل نفس الاسم MyMethod() لأداء وظيفة مختلفة تماماً ، فإننا عندئذ نستخدم المعدّل new فى إعلان العضو بالصورة الآتية:

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

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

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

ولـــو 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 inheri ted member 'MyBaseClass.MyMethod()'
```

ولا يجـــوز استخدام المعدّل new مع المعدّل override فى نفس الإعلان فهـــذا يؤدى إلى خطأ فى الترجمة. ومع ذلك فإن استخدام المعدّل new مع المعدّل 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 إمكانـــية إخفـــاء الأعضـــاء الموروثة تشبه إمكانية ركوب الأساليب الافتراضـــية (بـــالمعدّل override) ، وكلاهما تفيد فى تصميم طرازات جديدة من البرامج مع حفظ التوافق مع الطرازات السابقة من البرنامج.

وعلـــى ســـبيل المـــثال ، فلـــنفرض أنك تستخدم فصيلة أنتجتها شركة الكومبيوتر المصرية (ECC) بالاسم EccClass كالآتى:

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

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

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

وقـد كان برنامجك يعمل على ما يرام حتى أصدرت شركة الكومبيوتر المصـرية طـرازاً جديـداً من برنامجها متضمناً الأسلوب MyMethod (الذى يؤدى 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 اسـتخدامك للكلمة new لإعلان الأسلوب MyMethod فى برنامجك يؤكـد نيـتك علـى إخفـاء الأسلوب الذى يحمل نفس الاسم بالفصيلة EccClass ، كما يمنع رسالة التحذير من المترجم ، أى:

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

كمـا يمكـنك كبديل استخدام المعدّل override إذا كان فى نيتك ركوب الأسلوب الموجود بفصيلة الأساس.

مثال (8-5)

فـــى هذا المثال ترث الفصيلة MyDerivedClass الفصيلة MyBaseClass. وتحتوى كل من الفصيلتين على فصيلة عضوة بالاسم MyClass. ولكى نجعـــل الفصيلة الموجودة فى الفصيلة المشتقة تخفى الأخرى بالفصيلة الأساســية ، فقــد اســتخدمنا المؤثر new كمعدّل للإعلان. وقد أمكن التوصـــل إلـــى الأعضـــاء بكـــل 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 s  tring 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("Me  ssage 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 التى تحمل الاسم MyClass. وبالرغم من أن كل فصيلة تحتوى على حقل بنفس الاسم وأسلوب بنفس الاسم ، ولكن لم يلزمنا استخدام المعدّل new إلاّ مرة واحدة عند إعلان الفصيلة التى تحتويهما.

إخفاء أعضاء الوصلات البينية

يمكـــن تطبـــيق هـــذا المبدأ على الوصلات البينية أيضاً. فمثلاً الإعلان التالى يستخدم خاصية بالاسم M1:

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

ويمكـــنك فـــى نفـــس البرنامج أن تستخدم الاسم M1 فى إعلان أسلوب بالوصلة البينية المشتقة (IDerived) كالآتى:

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

فـــى هـــذه الحالـــة فإن العضو M1 بالوصلة المشتقة يخفى العضو M1 بالوصـــلة الموروثـــة. 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() { }
}
```

كمـــا يمكـــنك 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 عادى للأسلوب
    public void M1() { }
}
```

ومـــن الجائز أيضاً استخدام التطبيق الصريح للأسلوب والتطبيق العادى للخاصية كالآتى:

```
class MyClass: IDerived
{
private int m 1;
  //   تطبيق عادى للخاصية
  public int M1
  {
    get { return m1; }
    set { m1 = value; }
  }
  //   تطبيق صريح للأسلوب
  void IDerived.M1() { }
}
```