نوشته‌ی مسعود درویشیان

لینک مستقیم این مطلب در وب‌تارگت

استفاده از reference variable های interface

شما در سی‌شارپ می‌توانید یک reference variable از interface تعریف کنید. به عبارت دیگر، در سی‌شارپ می‌توانید interface reference variable بسازید. این‌چنین متغیری می‌تواند به هر شیءای که interface اش را اجرا می‌کند، رجوع کند. هنگامی‌که متد یک شیء را از طریق interface reference صدا می‌زنید، آن نسخه از متد که شیء مربوط به آن، interface را اجرا کرده است، اجرا می‌شود. این پروسه شبیه به استفاده از base class reference برای دسترسی به شیء derived class است (که در قسمت‌های قبلی با آن آشنا شدید).

مثال زیر استفاده از interface reference را نشان می‌دهد:

```
using System;
public interface ISeries
{
    int GetNext();
    void Reset();
    void SetStart(int x);
}
class ByTwos : ISeries
{
    int start;
    int val;

    public ByTwos()
    {
        start = 0;
        val = 0;
    }

    public int GetNext()
    {
        val += 2;
        return val;
    }

    public void Reset()
    {
        val = start;
    }

    public void SetStart(int x)
    {
```

```
        start = x;
        val = start;
    }
}
class MainClass
{
    static void Main()
    {
        ByTwos twoOb = new ByTwos();
        ISeries ob;
        ob = twoOb;

        for (int i = 0; i < 5; i++)
        {
            Console.WriteLine("Next value is: " + ob.GetNext());
        }
    }
}
```

در Main() متغیر ob، یک reference برای ISeries interface است. این بدان معناست که ob می‌تواند reference های هر شیءای که ISeries را اجرا می‌کنند، در خود ذخیره کند. در مثال بالا، از آن برای رجوع به twoOb استفاده می‌شود که شیء ByTwos است و کلاس ByTwos نیز ISeries interface را اجرا می‌کند.

نکته‌ی دیگر این است که interface reference variable فقط به متدهایی دسترسی دارد که در خود interface تعریف شده‌اند. بنابراین interface reference نمی‌تواند به متد و دیگر عناصری که در شیء مورد نظر تعریف شده‌اند دسترسی داشته باشد.

Interface Properties

همانند متدها، properties در interface بدونه بدنه تعریف می‌شوند. در زیر فرم کلی تعریف property در interface را می‌بینید:

```
// interface property
type name {
    get;
    set;
}
```

اگر تنها از get یا set استفاده کنید، property شما read-only یا write-only خواهد بود. اگرچه تعریف property در interface مشابه با تعریف auto-implemented property در کلاس است، اما این دو یکی نیستند. این روش تعریف property در interface باعث نمی‌شود که auto-implement باشد بلکه این تنها مشخص کننده‌ی نام و نوع property

است. همچنین اجازه تغییر access modifier را در قسمت get و set ندارید. به‌عنوان مثال نمی‌توانید set accessor را در interface به‌صورت private در نظر بگیرید.

به مثال زیر توجه کنید:

```
using System;
public interface ISeries
{
    int Next
    {
        get;
        set;
    }
}
class ByTwos : ISeries
{
    int val;
    public ByTwos()
    {
        val = 0;
    }

    public int Next
    {
        get
        {
            val += 2;
            return val;
        }
        set
        {
            val = value;
        }
    }
}
class SeriesDemo3
{
    static void Main()
    {
        ByTwos ob = new ByTwos();

        for (int i = 0; i < 5; i++)
            Console.WriteLine("Next value is " + ob.Next);
        Console.WriteLine("\nStarting at 21");
        ob.Next = 21; for (int i = 0; i < 5; i++)
            Console.WriteLine("Next value is " + ob.Next);
    }
}

/* Output

Next value is 2
Next value is 4
Next value is 6
Next value is 8
Next value is 10
```

```
Starting at 21
Next value is 23
Next value is 25
Next value is 27
Next value is 29
Next value is 31

 */
```

Interface indexers

یک interface می‌تواند indexer را نیز در خود داشته باشد. فرم کلی یک indexer ساده‌ی یک بعدی در interface به شکل زیر است:

```
// interface indexer
element-type this[int index] {
    get;
    set;
}
```

همانند قبل، اگر تنها از get یا set استفاده کنید، indexer شما read-only یا write-only خواهد بود. همچنین مجاز به استفاده از access modifier در accessor های indexer تعریف شده در interface نیستید.

به مثال زیر توجه کنید:

```
using System;
public interface ISeries
{
    // An interface property.
    int Next
    {
        get;
        set;
    }
    // An interface indexer.
    int this[int index]
    {
        get;
    }
}
class ByTwos : ISeries
{
    int val;
    public ByTwos()
    {
        val = 0;
    }
    public int Next
    {
        get
```

```
        {
            val += 2;
            return val;
        }
        set
        {
            val = value;
        }
    }
    public int this[int index]
    {
        get
        {
            val = 0;
            for (int i = 0; i < index; i++)
                val += 2;
            return val;
        }
    }
}

class SeriesDemo4
{
    static void Main()
    {
        ByTwos ob = new ByTwos();

        for (int i = 0; i < 5; i++)
            Console.WriteLine("Next value is " + ob.Next);
        Console.WriteLine("\nStarting at 21");
        ob.Next = 21;
        for (int i = 0; i < 5; i++)
            Console.WriteLine("Next value is " +
            ob.Next);
        Console.WriteLine("\nResetting to 0");
        ob.Next = 0;

        for (int i = 0; i < 5; i++)
            Console.WriteLine("Next value is " + ob[i]);
    }
}

/* Output

Next value is 2
Next value is 4
Next value is 6
Next value is 8
Next value is 10

Starting at 21
Next value is 23
Next value is 25
Next value is 27
Next value is 29
Next value is 31

Resetting to 0
Next value is 0
Next value is 2
Next value is 4
```

```
Next value is 6
Next value is 8

 */
```

در برنامه‌ی بالا، ISeries interface یک read-only indexer دارد که عنصر i ام را بازمی‌گرداند.

Interface و ارث‌بری

یک interface می‌تواند از یک interface دیگر ارث‌بری کند. برای انجام این امر، از syntax مشابه ارث‌بری در کلاس‌ها استفاده می‌شود. هنگامی‌که که یک کلاس قصد اجرای interface ای را دارد که آن interface از interface دیگری ارث بری کرده است، کلاس باید تمام اعضای تعریف شده در زنجیره‌ی ارث‌بری را اجرا کند.

به مثال زیر توجه کنید:

```
using System;
public interface IA
{
    void Meth1();
    void Meth2();
}

// IB now includes Meth1() and Meth2() -- it adds Meth3().
public interface IB : IA
{
    void Meth3();
}

// This class must implement all of IA and IB.
class MyClass : IB
{
    public void Meth1()
    {
        Console.WriteLine("Implement Meth1().");
    }
    public void Meth2()
    {
        Console.WriteLine("Implement Meth2().");
    }
    public void Meth3()
    {
        Console.WriteLine("Implement Meth3().");
    }
}
class IFExtend
{
    static void Main()
    {
        MyClass ob = new MyClass();
```

```
        ob.Meth1();
        ob.Meth2();
        ob.Meth3();
    }
}

/* Output

Implement Meth1().
Implement Meth2().
Implement Meth3().

 */
```

در برنامه‌ی بالا، اگر ()Meth1 را پاک کنید می‌بینید که با خطای compile-time مواجه می‌شوید. همان‌طور که ذکر شد، هر کلاسی که interface را اجرا می‌کند باید زنجیره‌ی ارث‌بری آن را برای اجرا در نظر بگیرد.

هنگامی که یک interface از interface دیگری ارث‌بری می‌کند این امکان وجود دارد که در derived interface یک عضو تعریف شود و این عضو با یکی از اعضای base interface هم‌نام باشد. در این مواقع عضو موجود در base interface دیگر دیده نمی‌شود و شما یک پیغام هشدار را خواهید دید. برای رفع پیغام هشدار می‌توانید قبل از تعریف آن عضو در derived interface، از کلمه‌ی کلیدی new استفاده کنید.

Explicit Implementations

هنگامی‌که یکی از اعضای interface را اجرا می‌کنید، می‌توانید نام آن عضو را به همراه نام interface اش بنویسید. انجام این کار باعث ساختن explicit interface member implementation یا به‌طور خلاصه explicit implementation می‌شود.

به نمونه‌ی زیر دقت کنید:

```
interface IMyInterface
{
    int Calculate(int x);
}
class MyClass : IMyInterface
{

    public int IMyInterface.Calculate(int x)
    {
        return x / 2;
    }
}
```

همان‌طور که می‌بینید، هنگام اجرای متد ()Calculate نام interface آن را نیز پیش از آن قرار داده‌ایم.

ساختن explicit implementation از interface method می‌تواند دو دلیل داشته باشد: ۱. هنگامی‌که یک interface method را از طریق explicit implementation می‌سازید، متد ساخته شده از طریق اشیای کلاس قابل دسترسی نخواهد بود بلکه از طریق interface reference به آن دسترسی خواهید داشت. از این‌رو، explicit implementation روش دیگری برای اجرای interface method است اما این متد، دیگر یک عضو public از کلاس‌تان نیست. ۲. برای یک کلاس امکان‌پذیر است که دو interface را اجرا (implement) کند و این امکان وجود دارد که هردوی آن‌ها متدهایی با یک نام و یک signature داشته باشند. در این موارد استفاده از explicit implementation باعث رفع ابهام می‌شود چراکه شما قبل از نام متد، نام interface آن را نیز مشخص می‌کنید.

به مثال زیر دقت کنید:

```
using System;
interface IDimensions
{
    float Length();
    float Width();
}

class Box : IDimensions
{
    float lengthInches;
    float widthInches;

    public Box(float length, float width)
    {
        lengthInches = length;
        widthInches = width;
    }

    // Explicit interface member implementation:
    float IDimensions.Length()
    {
        return lengthInches;
    }

    // Explicit interface member implementation:
    float IDimensions.Width()
    {
        return widthInches;
    }

    public static void Main()
    {
        // Declare a class instance "myBox":
        Box myBox = new Box(30.0f, 20.0f);

        // Declare an interface instance "myDimensions":
```

```
        IDimensions myDimensions = (IDimensions)myBox;

        /* Print out the dimensions of the box by calling the methods
           from an instance of the interface:                       */
        Console.WriteLine("Length: {0}", myDimensions.Length());
        Console.WriteLine("Width: {0}", myDimensions.Width());

        /* The following commented lines would produce compilation
           errors because they try to access an explicitly implemented
           interface member from a class instance:

           Console.WriteLine("Length: {0}", myBox.Length());
           Console.WriteLine("Width: {0}", myBox.Width());
        */
    }
}

/* Output

Length: 30
Width: 20

 */
```

دقت کنید که خط کدهای زیر به این دلیل comment شده‌اند که باعث به‌وجود خطا می‌شوند. یک interface member که explicitly implemented است نمی‌تواند از طریق اشیای کلاس (class instance) قابل دسترسی باشد:

```
// Console.WriteLine("Length: {0}", myBox.Length());
// Console.WriteLine("Width: {0}", myBox.Width());
```

اما خط کدهای زیر به دلیل این‌که از interface reference استفاده کرده است، بدون مشکل اجرا می‌شود:

```
Console.WriteLine(“Length: {0}”, myDimensions.Length());
Console.WriteLine(“Width: {0}”, myDimensions.Width());
```

به مثال زیر دقت کنید:

```
using System;
// Declare the English units interface:
interface IEnglishDimensions
{
    float Length();
    float Width();
}
// Declare the metric units interface:
interface IMetricDimensions
{
    float Length();
    float Width();
}
// Declare the "Box" class that implements the two interfaces:
// IEnglishDimensions and IMetricDimensions:
class Box : IEnglishDimensions, IMetricDimensions
{
    float lengthInches;
```

```
    float widthInches;
    public Box(float length, float width)
    {
        lengthInches = length;
        widthInches = width;
    }
    // Explicitly implement the members of IEnglishDimensions:
    float IEnglishDimensions.Length()
    {
        return lengthInches;
    }
    float IEnglishDimensions.Width()
    {
        return widthInches;
    }
    // Explicitly implement the members of IMetricDimensions:
    float IMetricDimensions.Length()
    {
        return lengthInches * 2.54f;
    }
    float IMetricDimensions.Width()
    {
        return widthInches * 2.54f;
    }
    public static void Main()
    {
        // Declare a class instance "myBox":
        Box myBox = new Box(30.0f, 20.0f);

        // Declare an instance of the English units interface:
        IEnglishDimensions eDimensions = (IEnglishDimensions)myBox;
        // Declare an instance of the metric units interface:
        IMetricDimensions mDimensions = (IMetricDimensions)myBox;

        // Print dimensions in English units:
        Console.WriteLine("Length(in): {0}", eDimensions.Length());
        Console.WriteLine("Width (in): {0}", eDimensions.Width());
        // Print dimensions in metric units:
        Console.WriteLine("Length(cm): {0}", mDimensions.Length());
        Console.WriteLine("Width (cm): {0}", mDimensions.Width());
    }
}

/* Output

Length(in): 30
Width (in): 20
Length(cm): 76.2
Width (cm): 50.8

 */
```

همان‌طور که در برنامه‌ی بالا می‌بینید، interface های IEnglishDimensions و IMetricDimensions دارای متدهایی با یک نام و یک signature هستند و این امر باعث می‌شود که هنگام اجرا کردن آن‌ها ابهام به‌وجود آید اما با اعمال explicit implementation این ابهام برطرف می‌شود.