زنگ سی‌شارپ – قسمت چهل و سوم

نوشته‌ی مسعود درویشیان

لینک مستقیم این مطلب در وب‌تارگت

در قسمت‌های قبل با ارث‌بری آشنا شدید، در این قسمت با Interface که یکی از مهم‌ترین ویژگی‌های سی‌شارپ است، آشنا می‌شوید. یک interface مجموعه‌ای از متدها را تعریف می‌کند که توسط یک کلاس اجرا خواهند شد. یک interface هیچ متدی را اجرا نمی‌کند، از این‌رو، interface یک سازه‌ی کاملاً منطقی است که فقط نشان‌دهنده‌ی قابلیت و عملکرد است و هیچ قسمت اجرایی ندارد.

Interfaces

بعضی مواقع در برنامه‌نویسی شی‌گرا تعریف اینکه یک کلاس چه کاری را باید انجام دهد، می‌تواند مفید باشد اما اینکه این‌کار را به چه روشی انجام می‌دهد مهم نیست. شما پیش از این با این چنین نمونه‌ای که abstract method نام داشت آشنا شدید. یک abstract method متدی را با یک return type و یک نام، تعریف می‌کند اما چیزی را اجرا نمی‌کند بلکه derived class باید abstract method هایی که در base class تعریف شده‌اند را اجرا کند. از این‌رو، abstract method مشخص کننده‌ی interface یک متد است، نه قسمت اجرایی. اگرچه abstract classes و abstract methods مفید و کاربردی هستند اما می‌توان این مفهوم را به شکل کامل‌تری نیز بیان کرد. در سی‌شارپ، شما می‌توانید interface یک کلاس را به‌طور کامل از بخش اجرایی آن جدا کنید که این کار توسط کلمه‌ی کلیدی interface انجام می‌شود.

Interface از نظر syntax متشابه با abstract class است. در interface نیز متدها بدنه ندارند و این بدین معنی است که در interface متدها اجرا نمی‌شوند. Interface مشخص می‌کند که چه کاری باید انجام شود اما به چگونه‌گی انجام شدن آن اهمیت نمی‌دهد و شما هرطور که مایل هستید متد مورد نظر را اجرا می‌کنید. هنگامی‌که یک interface تعریف می‌شود، هر تعداد کلاس که شما مد نظر دارید می‌توانند این interface را اجرا کنند. همچنین یک class می‌تواند به تعداد دل‌خواه interface اجرا کند.

برای اجرای یک interface، کلاس باید بدنه‌ی متدهای تعریف شده در interface را فراهم آورد. هر کلاس، آن‌طور که بخواهد برای اجرای این متدها (بدنه‌هایی که در کلاس خودش برای متدهای interface آماده کرده است) اقدام می‌کند. بنابراین دو کلاس می‌توانند یک interface را به روش‌های مختلفی اجرا کنند اما هردو کلاس شامل تمام متدهایی که در interface مشخص شده است، می‌باشند. با استفاده از interface، سی‌شارپ به شما اجازه می‌دهد جنبه‌ی One Interface, Multiple Method از polymorphism را به‌کار گیرید.

Interface توسط کلمه‌ی کلیدی interface تعریف می‌شود. در زیر فرم ساده شده‌ی یک interface را می‌بینید:

```
interface name {
    ret-type method-name1(param-list);
    ret-type method-name2(param-list);
    // ...
    ret-type method-nameN(param-list);
}
```

اسم interface توسط name مشخص می‌شود. متدها نیز توسط return type، نام و پارامترها (signature) تعریف می‌شوند. این متدها در واقع abstract method هستند. همان‌طور که پیش‌تر ذکر شد، در interface، متدها بدنه‌ی اجرایی ندارند و از این‌رو، کلاسی که interface دارد باید تمام متدهای تعریف شده در interface را اجرا کند. در یک interface، متدها implicitly public هستند. یعنی به‌صورت پیش‌فرض و خودکار public هستند و شما اجازه‌ی تغییر این حالت را ندارید.

در زیر یک نمونه از interface را می‌بینید که مشخص کننده‌ی interface یک class است که یک سری عدد را تولید می‌کند:

```
public interface ISeries
{
    int GetNext(); // return next number in series
    void Reset(); // restart
    void SetStart(int x); // set starting value
}
```

نام این interface را ISereis انتخاب کردیم. اگرچه پیش‌وند I ضروری نیست اما اکثر برنامه‌نویسان از این پیش‌وند استفاده می‌کنند تا تفاوت interface را با class مشخص سازند. ISeries به‌صورت public تعریف شده است، بنابراین می‌تواند توسط هر کلاسی، در هر برنامه‌ای اجرا شود.

علاوه بر متدها، interface می‌تواند دارای property، indexer و event باشد. در حال حاضر تمرکز بحث روی methods, properties و indexers خواهد بود. با event ها در مقالات آینده آشنا خواهید شد. Interface ها نمی‌توانند data member داشته باشند. آن‌ها همچنین دارای constructor، destructor و operator methods نیستند و هیچ عضوی نمی‌تواند در آن‌ها به‌صورت static تعریف شود.

اجرای interface ها

زمانی‌که یک interface تعریف می‌شود، یک یا چند کلاس می‌توانند این interface را اجرا کنند. برای اجرای یک interface، نام آن را بعد از نام کلاس (به همان طریقی که نام base class را می‌نوشتید) می‌نویسید.

فرم کلی کلاسی که یک interface را اجرا می‌کند به شکل زیر است:

```
class class-name : interface-name {
    // class-body
}
```

در این‌جا، نام آن interface که قرار است توسط کلاس اجرا شود با interface-name مشخص شده است. هنگامی‌که کلاسی می‌خواهد یک interface را اجرا کند، بایستی به‌طور کامل آن interface را اجرا کند و نمی‌تواند فقط بخشی از آن را برای اجرا انتخاب کند.

یک کلاس می‌تواند بیشتر از یک interface را اجرا کند. برای این منظور، پس از نام کلاس، لیست اسامی interface های مورد نظر را توسط کاما از هم جدا می‌کند. یک کلاس هم می‌تواند از یک کلاس دیگر ارث‌بری کند و هم چندین interface را اجرا کند. در این مورد باید نام base class را در ابتدای لیستی که توسط کاما از هم جدا کرده‌اید، قرار دهید.

متدهایی که interface را اجرا می‌کنند باید public باشند زیرا متدها درون interface به‌صورت implicitly public هستند و از این‌رو اجرای آن‌ها نیز باید public باشد. همچنین return type و signature متدهای اجرایی باید دقیقاً با متدهای تعریف شده در interface مطابقت داشته باشد.

در زیر مثالی می‌بینید که در آن ISeries interface اجرا شده است:

```
public interface ISeries
{
    int GetNext(); // return next number in series
    void Reset(); // restart
    void SetStart(int x); // set starting value
}

// Implement ISeries.
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;
    }
}
```

همان‌طور که می‌بینید، کلاس ByTwos تمام متدهای تعریف شده در ISeries را اجرا می‌کند. توجه کنید که کلاس نمی‌تواند فقط بخشی از interface را اجرا کند و مجبور به اجرای تمام آن است.

به برنامه‌ی زیر توجه کنید:

```
using System;
public interface ISeries
{
    int GetNext(); // return next number in series
    void Reset(); // restart
    void SetStart(int x); // set starting value
}

// Implement ISeries.
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;
    }
}

// Demonstrate the ISeries interface.
class SeriesDemo
{
    static void Main()
    {
        ByTwos ob = new ByTwos();

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

        Console.WriteLine("\nResetting");
        ob.Reset();

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

        Console.WriteLine("\nStarting at 100");
        ob.SetStart(100);

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

/* Output

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

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

Starting at 100
Next value is 102
Next value is 104
Next value is 106
```

```
Next value is 108
Next value is 110

*/
```

کلاس‌ها می‌توانند به‌منظور افزایش قابلیت‌شان نیز interface را اجرا کنند که این امر بسیار رایج است. برای مثال، نسخه‌ی دیگری از ByTwos را در زیر می‌بینید که علاوه بر اعضای interface، شامل متد جدید GetPrevious() نیز است:

```
// Implement ISeries and add GetPrevious().
class ByTwos : ISeries
{
    int start;
    int val;
    int prev;
    public ByTwos()
    {
        start = 0;
        val = 0;
        prev = -2;
    }
    public int GetNext()
    {
        prev = val;
        val += 2;
        return val;
    }
    public void Reset()
    {
        val = start;
        prev = start - 2;
    }
    public void SetStart(int x)
    {
        start = x;
        val = start;
        prev = val - 2;
    }
    // A method not specified by ISeries.
    public int GetPrevious()
    {
        return prev;
    }
}
```

همان‌طور که می‌بینید، افزودن GetPrevious() نیازمند انجام تغییراتی در اجرای متدهای تعریف شده در ISeries است و از آنجا که interface برای این متدها ثابت می‌ماند، تغییرات یکپارچه به نظر رسیده و باعث خراب شدن کدهای قبلی نمی‌شود. این یکی از مزایای interface است.

پیش‌تر ذکر شد که هر کلاسی می‌تواند یک interface را اجرا کند. برای مثال، در زیر کلاسی به اسم Primes داریم که دنباله‌ای از اعداد اول را تولید کرده و ISeries interface را اجرا می‌کند. توجه کنید که اجرای ISeries در این‌جا کاملاً با اجرای این interface توسط ByTwos متفاوت است:

```
public interface ISeries
{
    int GetNext(); // return next number in series
    void Reset(); // restart
    void SetStart(int x); // set starting value
}

// Use ISeries to implement a series of prime numbers.
class Primes : ISeries
{
    int start;
    int val;

    public Primes()
    {
        start = 2;
        val = 2;
    }

    public int GetNext()
    {
        int i, j;
        bool isprime;

        val++;
        for (i = val; i < 1000000; i++)
        {
            isprime = true;
            for (j = 2; j <= i / j; j++)
            {
                if ((i % j) == 0)
                {
                    isprime = false;
                    break;
                }
            }
            if (isprime)
            {
                val = i;
                break;
            }
        }
        return val;
    }

    public void Reset()
    {
        val = start;
    }

    public void SetStart(int x)
    {
        start = x;
```

```
        val = start;
    }
}
```

نکته‌ی کلیدی این‌جاست که ByTwos و Primes دنباله‌ی کاملاً نامربوطی از اعداد را تولید کرده اما هنوز هردو ISeries را اجرا می‌کنند. بنابراین هر کلاس آن‌طور بخواهد می‌تواند یک interface را اجرا کند.

به دو مثال زیر برای درک بهتر interface دقت کنید:

مثال:

```
using System;
interface IValue
{
    int Count { get; set; } // Property interface
    string Name { get; set; } // Property interface
}

class Image : IValue // Implements interface
{
    public int Count // Property implementation
    {
        get;
        set;
    }

    string name;

    public string Name // Property implementation
    {
        get { return this.name; }
        set { this.name = value; }
    }
}

class Article : IValue // Implements interface
{
    public int Count // Property implementation
    {
        get;
        set;
    }

    string name;

    public string Name // Property implementation
    {
        get { return this.name; }
        set { this.name = value.ToUpper(); }
    }
}

class Program
{
    static void Main()
```

```
    {
        IValue value1 = new Image();
        IValue value2 = new Article();

        value1.Count++; // Access int property on interface
        value2.Count++; // Increment

        value1.Name = "Mona Lisa"; // Use setter on interface
        value2.Name = "Resignation"; // Set

        Console.WriteLine(value1.Name); // Use getter on interface
        Console.WriteLine(value2.Name); // Get
    }
}

/*
  Out put

  Mona Lisa
  Resignation

 */
```

مثال:

```
using System;

public interface ITransactions
{
    // interface members
    void showTransaction();
    double getAmount();
}

public class Transaction : ITransactions
{
    private string tCode;
    private string date;
    private double amount;
    public Transaction()
    {
        tCode = " ";
        date = " ";
        amount = 0.0;
    }
    public Transaction(string c, string d, double a)
    {
        tCode = c;
        date = d;
        amount = a;
    }
    public double getAmount()
    {
        return amount;
    }
    public void showTransaction()
```

```
    {
        Console.WriteLine("Transaction: {0}", tCode);
        Console.WriteLine("Date: {0}", date);
        Console.WriteLine("Amount: {0}", getAmount());

    }
}
class Tester
{
    static void Main(string[] args)
    {
        Transaction t1 = new Transaction("001", "8/10/2012", 78900.00);
        Transaction t2 = new Transaction("002", "9/10/2012", 451900.00);
        t1.showTransaction();
        t2.showTransaction();
        Console.ReadKey();
    }
}

/*
Output

Transaction: 001
Date: 8/10/2012
Amount: 78900
Transaction: 002
Date: 9/10/2012
Amount: 451900

*/
```

ادامه‌ی مبحث interface را در قسمت‌های بعد دنبال کنید.