Программирование C#

11. Интерфейсы и структуры

Карбаев Д.С., 2015

Интерфейсы

- В C# предусмотрено разделение интерфейса класса и его реализации с помощью ключевого слова **interface**.
- С точки зрения синтаксиса интерфейсы подобны абстрактным классам. Но в интерфейсе ни у одного из методов не должно быть тела. Это означает, что в интерфейсе вообще не предоставляется никакой реализации.
- Как только интерфейс будет определен, он может быть реализован в любом количестве классов. Кроме того, в одном классе может быть реализовано любое количество интерфейсов.
- Один и тот же интерфейс может быть реализован в двух классах по-разному. Тем не менее в каждом из них должен поддерживаться один и тот же набор методов данного интерфейса.

Интерфейсы

- упрощенная форма объявления интерфейса.

```
interface имя{
возвращаемый_тип имя_метода1 (список_параметров) ;
возвращаемый_ тип имя_метода2 (список_параметров) ;
// ...
возвращаемый_тип имя_методаN (список_параметров) ;
}
```

где имя — это конкретное имя интерфейса. В объявлении методов интерфейса используются только их возвращаемый_тип и сигнатура.

- Пример объявления интерфейса для класса, генерирующего последовательный ряд чисел.

```
public interface ISeries {
int GetNext(); // возвратить следующее по порядку число
void Reset(); // перезапустить
void SetStart(int x); // задать начальное значение
}
```

Интерфейсы

- общая форма реализации интерфейса в классе.

```
class имя_класса : имя_интерфейса {
//тело класса
}
```

- В классе допускается реализовывать несколько интерфейсов. В этом случае все реализуемые в классе интерфейсы указываются списком через запятую.
- В классе можно наследовать базовый класс и в тоже время реализовать один или более интерфейс. В таком случае имя базового класса должно быть указано перед списком интерфейсов, разделяемых запятой.
- Методы, реализующие интерфейс, должны быть объявлены как **public**.

Интерфейсы: генерация четных чисел

```
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 SeriesDemo{
  static void Main(){
    ByTwos ob = new ByTwos();
    for (int i = 0; i < 5; i++)
    Console.WriteLine("Следующее число равно "
        + ob.GetNext());
    Console.WriteLine("\nСбросить");
    ob.Reset();
    for (int i = 0; i < 5; i++)
     Console.WriteLine("Следующее число равно "
        + ob.GetNext());
    Console.WriteLine("\nНачать с числа 100");
    ob.SetStart(100);
    for (int i = 0; i < 5; i++)
     Console.WriteLine("Следующее число равно "
        + ob.GetNext());
    }
  }
```

Следующее число равно 2
Следующее число равно 4
Следующее число равно 6
Следующее число равно 8
Следующее число равно 10

Сбросить.
Следующее число равно 2
Следующее число равно 4
Следующее число равно 6
Следующее число равно 8
Следующее число равно 10

Начать с числа 100.
Следующее число равно 102
Следующее число равно 104
Следующее число равно 106
Следующее число равно 108
Следующее число равно 110

Интерфейсы: генерация четных чисел

```
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;
        }
        // Метод, не указанный в интерфейсе ISeries
        public int GetPrevious(){
            return prev;
        }
    }
```

Интерфейсы: генерация простых чисел

```
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;
    }}
```

```
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 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;
}}
```

```
class SeriesDemo2{
    static void Main(){
        ByTwos twoOb = new ByTwos();
        Primes primeOb = new Primes();
        ISeries ob;
        for (int i = 0; i < 5; i++){
            ob = twoOb;
            Console.WriteLine("Следующее четное число равно " + ob.GetNext());
            ob = primeOb;
            Console.WriteLine("Следующее простое число " + "равно " + ob.GetNext());
        }}}
```

Интерфейсные свойства

- Аналогично методам, свойства указываются в интерфейсе без тела. Ниже приведена общая форма объявления интерфейсного свойства.

```
// Интерфейсное свойство
тип имя{
get;
set;
}
```

- Очевидно, что в определении интерфейсных свойств, доступных только для чтения или только для записи, должен присутствовать единственный аксессор: get или set соответственно.

```
public interface ISeries{
        int Next{// Интерфейсное свойство,
            get; // возвратить следующее по порядку число
            set; // установить следующее число
}       }
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("Следующее число равно " + ob.Next);
        Console.WriteLine("\nНачать с числа 21");
        ob.Next = 21;
        for (int i = 0; i < *5; i++)
            Console.WriteLine("Следующее число равно " + ob.Next);
    }
}
```

```
Следующее число равно 2
Следующее число равно 4
Следующее число равно 6
Следующее число равно 8
Следующее число равно 10
Начать с числа 21
Следующее число равно 23
Следующее число равно 25
Следующее число равно 27
Следующее число равно 2 9
Следующее число равно 31
```

Интерфейсные индексаторы

- В интерфейсе можно также указывать индексаторы. Общая форма:

```
// Интерфейсный индексатор
тип_элемента this[int индекс]{
get;
set;
}
```

- Как и прежде, в объявлении интерфейсных индексаторов, доступных только для чтения или только для записи, должен присутствовать единственный аксессор: get или set соответственно.

```
public interface ISeries{
        int Next{
             get; // возвратить следующее по порядку число
             set; // установить следующее число
         }
         // Интерфейсный индексатор
         int this[int index]{
             get; // возвратить указанное в ряду число
         }}
class ByTwos : ISeries{
         int val;
         public ByTwos(){
             val = 0;
         }
         public int Next{
             get{
                 val += 2;
                 return val;
             }
             set{
                 val = value;
             }
         }
         // Получить значение по индексу
         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();
            // Получить доступ к последовательному ряду чисел с помощью свойства
            for (int i = 0; i < 5; i++)
                Console.WriteLine("Следующее число равно " + ob.Next);
            Console.WriteLine("\nНачать с числа 21");
            ob.Next = 21;
            for (int i = 0; i < 5; i++)
                Console.WriteLine("Следующее число равно " + ob.Next);
            Console.WriteLine("\nСбросить в 0");
            ob.Next = 0;
            // Получить доступ к последовательному ряду чисел с помощью индексатора
            for (int i = 0; i < 5; i++)
                Console.WriteLine("Следующее число равно " + ob[i]);
        }}
```

```
Следующее число равно 2
Следующее число равно 4
Следующее число равно 6
Следующее число равно 8
Следующее число равно 10
Начать с числа 21
Следующее число равно 23
Следующее число равно 25
Следующее число равно 27
Следующее число равно 29
Следующее число равно 31
Сбросить в 0
Следующее число равно 0
Следующее число равно 2
Следующее число равно 4
Следующее число равно 6
Следующее число равно 8
```

```
// Пример наследования интерфейсов
    public interface IA{
        void Met1();
        void Met2();
    }
// В базовый интерфейс включены методы Met1() и Met2(),
// а в производный интерфейс добавлен еще один метод – Met3().
public interface IB : IA{
    void Meth3();
}
// В этом классе должны быть реализованы все методы интерфейсов IA и IB.
class MyClass : IB{
    public void Met1(){
        Console.WriteLine("Реализовать метод Met1().");
    }
    public void Met2(){
        Console.WriteLine("Реализовать метод Met2().");
    }
    public void Met3(){
        Console.WriteLine("Реализовать метод Met3().");
    }}
class IFExtend{
    static void Main(){
        MyClass ob = new MyClass();
        ob.Met1(); ob.Met2(); ob.Met3();
    }}
```

Явные реализации

- Все методы и поля интерфейса по умолчанию являются **публичными** и не включают модификаторов доступа (private, protected), также поля не могут быть статичными (static).
- При реализации члена интерфейса имеется возможность указать его имя полностью вместе с именем самого интерфейса. В этом случае получается явная реализация члена интерфейса, или просто явная реализация.
- Так, если объявлен интерфейс IMyIF

```
interface IMyIF {
  int MyMeth(int x);
}
```

то следующая его реализация считается вполне допустимой:

```
class MyClass : IMyIF {
  int IMyIF.MyMeth(int x) {
    return x / 3;
  }
}
```

Явные реализации

```
// Реализовать член интерфейса явно
   interface IEven{
       bool IsOdd(int x); bool IsEven(int x);
   }
   class MyClass : IEven {
       // Явная реализация (здесь метод является закрытым по умолчанию)
       bool IEven.IsOdd(int x){
           if ((x % 2) != 0) return true;
           else return false;
       }
       // Обычная реализация
       public bool IsEven(int x){
           IEven o = this; // Интерфейсная ссылка на вызывающий объект
           return !o.IsOdd(x);
       }}
   class Demo{
       static void Main(){
           MyClass ob = new MyClass();
           bool result = ob.IsEven(4);
           if (result) Console.WriteLine("4 - четное число.");
           // result = ob.IsOdd(); // Ошибка, член IsOdd интерфейса IEven недоступен
           // Но следующий код написан верно, поскольку в нем сначала создается
           // интерфейсная ссылка типа IEven на объект класса MyClass, а затем по
           // этой ссылке вызывается метод IsOdd().
           IEven iRef = (IEven)ob;
           result = iRef.IsOdd(3);
           if (result) Console.WriteLine("3 - нечетное число.");
  }    }
```

```
// Воспользоваться явной реализацией для устранения неоднозначности.
    interface IMyIF_A{
        int Meth(int x);
    }
    interface IMyIF_B{
        int Meth(int x);
    }
    // Оба интерфейса реализуются в классе MyClass.
    class MyClass : IMyIF_A, IMyIF_B{
        // Реализовать оба метода Meth() явно.
        int IMyIF_A.Meth(int x){ return x + x; }
        int IMyIF_B.Meth(int x){ return x * x; }
        // Вызывать метод Meth() по интерфейсной ссылке,
        public int MethA(int x){
            IMyIF_A a_ob = this;
            return a_ob.Meth(x); // вызов интерфейсного метода IMyIF_A
        }
        public int MethB(int x){
            IMyIF_B b_ob;
            b_ob = this;
            return b_ob.Meth(x); // вызов интерфейсного метода IMyIF_B
    }  }
    class FQIFNames{
        static void Main() {
               MyClass ob = new MyClass();
               Console.Write("Вызов метода IMyIF_A.Meth(): ");
               Console.WriteLine(ob.MethA(3);
               Console.Write("Вызов метода IMyIF_B.Meth(): ");
               Console.WriteLine(ob.MethB(3));
    }    }
```

Вызов метода IMyIF_A.Meth(): 6
Вызов метода IMyIF_B.Meth(): 9

Стандартные интерфейсы для среды .NET Framework

- Для среды .NET Framework определено немало стандартных интерфейсов, которыми можно пользоваться в программах на C#.
- Так, в интерфейсе **System. IComparable** определен метод **CompareTo ()**, применяемый для сравнения объектов, когда требуется соблюдать отношение порядка.
- Стандартные интерфейсы являются также важной частью классов коллекций, предоставляющих различные средства, в том числе стеки и очереди, для хранения целых групп объектов.
- Так, в интерфейсе **System.Collections.ICollection** определяются функции для всей коллекции, а в интерфейсе **System.Collections .IEnumerator** — способ последовательного обращения к элементам коллекции.

Структуры

- Объекты конкретного класса доступны по ссылке, в отличие от значений простых типов, доступных непосредственно.
- Структура подобна классу, но ***относится к типу значения***, а не к ссылочному типу данных.
- Структуры объявляются с помощью ключевого слова **struct** и с точки зрения синтаксиса подобны классам. Общая форма объявления структуры:

```
struct имя : интерфейсы {
// объявления членов
}
```

- Поскольку структуры не поддерживают наследование, то их члены нельзя указывать как abstract, virtual или protected.

```
struct Book{
        public string Author;
        public string Title;
        public int Copyright;
        public Book(string a, string t, int c){
            Author = a; Title = t;  Copyright = c;
}       }
// Продемонстрировать применение структуры Book,
class StructDemo{
    static void Main(){
        Book book1 = new Book("Герберт Шилдт", "Полный справочник по C# 4.5", 2014);
        Book book2 = new Book();
        Book book3;
        Console.WriteLine(book1.Author+", "+book1.Title+", (c) "+book1.Copyright);
        Console.WriteLine();
        if (book2.Title == null)
            Console.WriteLine("Член book2.Title пуст.");
        //А теперь ввести информацию в структуру book2.
        book2.Title = "О дивный новый мир";
        book2.Author = "Олдос Хаксли";
        book2.Copyright = 1932;
        Console.Write("Структура book2 теперь содержит:\n");
        Console.WriteLine(book2.Author + ", " +
        book2.Title + ", (c) " + book2.Copyright);
        Console.WriteLine();
        // Console.WriteLine(book3.Title); // неверно, этот член структуры
        // нужно сначала инициализировать
        book3.Title = "Красный шторм";
        Console.WriteLine(book3.Title); // теперь верно
}   }
```

Герберт Шилдт, Полный справочник по C# 4.5, (c) 2014
Член book2.Title пуст.
Структура book2 теперь содержит:
Олдос Хаксли, О дивный новый мир, (c) 1932
Красный шторм

Структуры: присваивание

```
struct MyStruct{
  public int x;
}

class StructAssignment{
 static void Main(){
  MyStruct a;
  MyStruct b;
  a.x = 10; b.x = 20;
  Console.WriteLine("a.x {0}, b.x {1}",
                    a.x, b.x);
  a = b; b.x = 30;
  Console.WriteLine("a.x {0}, b.x {1}",
                    a.x, b.x);
 }
}
```

a.x 10, b.x 20
a.x 20, b.x 30

```
class MyClass{
  public int x;
}

class ClassAssignment{
 static void Main(){
  MyClass a = new MyClass();
  MyClass b = new MyClass();
  a.x = 10; b.x = 20;
  Console.WriteLine("a.x {0}, b.x {1}",
                    a.x, b.x);
  a = b; b.x = 30;
  Console.WriteLine("a.x {0}, b.x {1}",
                    a.x, b.x);
 }
}
```

a.x 10, b.x 20
a.x 30, b.x 30

```
 // Структуры удобны для группирования небольших объемов данных,
struct PacketHeader{
        public uint PackNum; // номер пакета
        public ushort PackLen; // длина пакета
    }
// Использовать структуру PacketHeader для создания записи финансовой транзакции
class Transaction{
    static uint transacNum = 0;
    PacketHeader ph;
    string accountNum;
    double amount;
    public Transaction(string ace, double val){
        // создать заголовок пакета
        ph.PackNum = transacNum++;
        ph.PackLen = 512; // произвольная длина
        accountNum = ace; amount = val;
    }
    // Сымитировать транзакцию.
    public void sendTransaction(){
        Console.WriteLine("Пакет #: " + ph.PackNum +
        ", Длина: " + ph.PackLen + ",\n Счет #: " + accountNum +
        ", Сумма: {0:C}\n", amount);
    }
    // Продемонстрировать применение структуры в виде пакета транзакции,
    class PacketDemo{
        static void Main(){
            Transaction t = new Transaction("31243", -100.12);
            Transaction t2 = new Transaction("AB4655", 345.25);
            Transaction t3 = new Transaction("8475-09", 9800.00);
            t.sendTransaction(); t2.sendTransaction(); t3.sendTransaction();
}   }   }
```

```
Пакет #: 0, Длина: 512,
Счет #: 31243, Сумма: ($100.12)
Пакет #: 1, Длина: 512,
Счет #: AB4655, Сумма: $345.25
Пакет #: 2, Длина: 512,
Счет #: 8475-09, Сумма:
$9,800.00
```