Лекция 34

Абстрактные классы и Интерфейсы

Абстрактные классы

Абстрактный метод создается с помощью указываемого модификатора типа **abstract**. У абстрактного метода отсутствует тело, и поэтому он не реализуется в базовом классе. Это означает, что он **должен быть переопределен** в производном классе, поскольку его вариант из базового класса просто непригоден для использования.

Определение абстрактного метода:

abstract тип имя(список_параметров);

У абстрактного метода отсутствует тело. Модификатор **abstract** может применяться только в методах экземпляра, но не в статических методах (**static**). Абстрактными могут быть также индексаторы и свойства.

Абстрактные классы

Класс, содержащий один или больше абстрактных методов, должен быть также объявлен как **абстрактный**, и для этого перед его объявлением **class** указывается модификатор **abstract**.

У абстрактного класса него не может быть объектов.

Когда производный класс наследует абстрактный класс, в нем должны быть реализованы все абстрактные методы базового класса. В противном случае производный класс должен быть также определен как **abstract**. Таким образом, атрибут **abstract** наследуется до тех пор, пока не будет достигнута полная реализация класса.

Пример 1

```
abstract class TwoDShape {
    double pri_width;
    double pri_height;
    public TwoDShape() { // Конструктор по умолчанию
        Width = Height =0.0;
         name = "null";
    }
    public TwoDShape(double w, double h, string n) {
     // Параметризированный конструктор
        Width = w;
        Height = h;
        name = n;
    }
```

Пример 1

```
  public TwoDShape(double x, string n) {
// Сконструировать объект равной ширины и высоты
      Width = Height = x;          name = n;
  }
  public TwoDShape(TwoDShape ob) {
 // Сконструировать копию объекта TwoDShape
      Width = ob.Width;          Height = ob.Height;
      name = ob.name;
  }
   public double Width {
  // Свойства ширины и высоты объекта
        get { return pri_width; }
        set { pri_width = value < 0 ? -value : value; }
   }
```

Пример 1

```
    public double Height {
        get { return pri_height; }
        set { pri_height = value < 0 ? -value : value; }
    }
    public string name { get; set; }
    public void ShowDim() {
        Console.WriteLine("Ширина и высота равны " +
Width + " и " + Height);
    }
    public abstract double Area();
     // Теперь метод Area () является абстрактным
}
```

Пример 1

```
class Triangle : TwoDShape {
    string Style;
    public Triangle() { Style = "null";  }
    public Triangle(string s, double w, double h) : base (w,
  h, "треугольник") { Style = s; }
    public Triangle(double x) : base(x, "треугольник")
      { Style = "равнобедренный"; }
    public Triangle(Triangle ob) : base(ob)
       { Style = ob.Style; }
   public override double Area()
      { return Width * Height / 2; }
   public void ShowStyle()
      { Console.WriteLine("Треугольник " + Style); }
}
```

Пример 1

```
class Rectangle : TwoDShape {
     public Rectangle(double w, double h) : base(w, h,
   "прямоугольник"){ }
     public Rectangle(double x) : base(x,
   "прямоугольник") { }
    public Rectangle(Rectangle ob) : base(ob) { }
    public bool IsSquare() {
       if (Width == Height) return true;
       return false;
    }
    public override double Area() {
       return Width * Height;
   }
}
```

Пример 1

```
class AbsShape {
   static void Main() {
      TwoDShape[] shapes = new TwoDShape[4];
      shapes[0] = new Triangle("прямоугольный", 8.0, 12.0);
      shapes[1] = new Rectangle(10);
      shapes[2] = new Rectangle(10, 4);
      shapes[3] = new Triangle(7.0);
 //   shapes[3] = new TwoDShape(10, 20, "общая форма"); // ошибка
      for (int i=0; i < shapes.Length; i++) {
         Console.WriteLine("Объект — " + shapes[i].name);
         Console.WriteLine("Площадь равна " + shapes[i].Area());
         Console.WriteLine() ;
      }
   }
}
```

Интерфейсы

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. В нем указывается только, **что** именно **следует делать**, **но не как это делать**. Как только интерфейс будет определен, он может быть реализован в любом количестве классов. Кроме того, в одном классе может быть реализовано любое количество интерфейсов.

Интерфейсы

```
interface имя{
    // интерфейсные методы
    возвращаемый_тип имя_метода1(параметры);…
    возвращаемый_тип имя_методаN(параметры);
    тип имя{ // интерфейсное свойство
        get;
        set;
    }
    тип_элемента this[int индекс]{ // индексатор
        get;
        set;
    }
}
```

Интерфейсы

```
public interface ISeries {
   int GetNext();
   void Reset();
   void SetStart(int x);
}
```

Помимо **методов**, в интерфейсах можно также указывать **свойства**, **индексаторы** и **события**. Интерфейсы не могут содержать члены данных. В них **нельзя** также определить **конструкторы**, **деструкторы** или **операторные методы**. Ни один из членов интерфейса не может быть объявлен как **static**.

Интерфейсы

Общая форма реализации интерфейса в классе:

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

Реализовать интерфейс **выборочно и только по частям нельзя**.

В классе допускается реализовывать **несколько интерфейсов**, которые указываются после двоеточия **списком через запятую**.

Если класс наследует базовый класс и в тоже время реализует один или более интерфейсов, то **имя базового класса должно быть указано перед списком интерфейсов**, разделяемых запятой.

Интерфейсы

Методы, реализующие интерфейс, должны быть объявлены как **public**. Кроме того, возвращаемый тип и сигнатура реализуемого метода должны точно соответствовать возвращаемому типу и сигнатуре, указанным в определении интерфейса.

В классах, реализующих интерфейсы, разрешается и часто практикуется определять их собственные дополнительные члены.

Для добавления интерфейса в проект можно нажать правой кнопкой мыши на проект и в контекстном меню выбрать **Add-> New Item** и в диалоговом окне добавления нового компонента выбрать **Interface.**

Пример 2

```
public interface ISeries {
   int GetNext(); // возвратить следующее по порядку число
   void Reset(); // перезапустить
   void SetStart(int x); // задать начальное значение
}
class ByTwos : ISeries { // реализовать интерфейс ISeries
   int start;
   int val;
   int prev;
   public ByTwos() {
      start = 0;
      val = 0;
      prev = -2;
   }
```

Пример 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;
}
```

Пример 2

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

using System;
class SeriesDemo {
  static void Main() {
    ByTwos ob = new ByTwos();
    for (int i=0; i < 5; i++)
      Console.WriteLine ("Следующее число равно " +
    ob.GetNext() + ", а предыдущее - " + ob.GetPrevious());
```

Пример 2

```
    Console.WriteLine("\nСбросить");
    ob.Reset ();
    Console.WriteLine ("Следующее число равно " +
  ob.GetNext() + ", а предыдущее - " + ob.GetPrevious());
    Console.WriteLine("\nНачать с числа 100");
    ob.SetStart(100);
    for (int i=0; i < 5; i++)
    Console.WriteLine ("Следующее число равно " +
  ob.GetNext() + ", а предыдущее - " + ob.GetPrevious());
    }
}
```

Интерфейсы

Интерфейс может быть реализован в **любом количестве классов**.

В **C#** допускается объявлять **переменные ссылочного интерфейсного типа**, т.е. переменные ссылки на интерфейс. Такая переменная может ссылаться на любой объект, реализующий ее интерфейс. При вызове метода для объекта посредством интерфейсной ссылки выполняется его вариант, реализованный в классе данного объекта.

Пример 3

```
using System;
public interface ISeries { // определить интерфейс
   int GetNext(); // возвратить следующее по порядку число
   void Reset(); // перезапустить
   void SetStart(int x); // задать начальное значение
}
class ByTwos : ISeries {
   int start;
   int val;
   public ByTwos() {
      start = 0;
      val = 0;
   }
```

Пример 3

```
  public int GetNext() {
    val += 2;
    return val;
  }
  public void Reset() {
    val = start;
  }
  public void SetStart(int x) {
    start = x;
    val = start;
  }
}
```

Пример 3

```
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;
  }
```

Пример 3

```
  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; // ссылка на интерфейс
```

Пример 3

```
    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());
    }
  }
}
```

Интерфейсы

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

В определении интерфейсных свойств, доступных только для чтения или только для записи, должен присутствовать единственный аксессор: **get** или **set** соответственно.

Интерфейсы

Несмотря на то что объявление свойства в интерфейсе очень похоже на объявление автоматически реализуемого свойства в классе, между ними все же имеется отличие.

При объявлении в интерфейсе **свойство не становится автоматически реализуемым**.

В этом случае указывается только имя и тип свойства, а его реализация предоставляется каждому реализующему классу.

Кроме того, при объявлении свойства в интерфейсе **не разрешается указывать модификаторы доступа для аксессоров**.

Например, аксессор **set** не может быть указан в интерфейсе как **private**.

Пример 4

```
using System;
public interface ISeries {
    int Next { // интерфейсное свойство
       get; // возвратить следующее по порядку число
       set; // установить следующее число
   }
}
class ByTwos : ISeries { // реализовать интерфейс ISeries
   int val;
   public ByTwos() {  val = 0;  }
   public int Next { // получить или установить значение
      get {  val += 2;   return val;  }
      set { val = value; }
   }
}
```

Пример 4

```
class SeriesDemo3 {
  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);                          // 2, 4, 6, 8, 10
    Console.WriteLine("\nНачать с числа 21");
    ob.Next = 21;
    for (int i=0; i < 5; i++)
      Console.WriteLine("Следующее число равно " + ob.Next);
                                                  // 21, 23, 25, 27, 29, 31
    }
  }
}
```

Интерфейсы

В интерфейсе можно также указывать **индексаторы**.

тип_элемента **this**[int индекс]{
get;
set;
}

В объявлении интерфейсных индексаторов, доступных только для чтения или только для записи, должен присутствовать единственный аксессор: **get** или **set** соответственно.

Пример 5 (ООП)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace WindowsFormsAppРазное{
  public partial class Form2 : Form{ // класс формы
   public Form2() { InitializeComponent(); } //
    конструктор
```

Пример 5 (ООП)

```
private void button1_Click(object sender, EventArgs e){
// использование интерфейса
  ByTwos ob = new ByTwos();
// получить доступ к последовательному ряду чисел
// с помощью свойства
  for (int i = 0; i < 5; i++)
    richTextBox2.AppendText("Следующее число равно
" + ob.Next.ToString() + "\n"); // 2,4,6,8,10
  richTextBox2.AppendText("\nНачать с числа 21\n");
  ob.Next = 21;
  for (int i = 0; i < 5; i++)
    richTextBox2.AppendText("Следующее число равно
" + ob.Next.ToString() + "\n");   // 21, 23, 25, 27, 29, 31
```

Пример 5 (ООП)

```
  richTextBox2.AppendText("\nСбросить в 0\n");
  ob.Reset();
  // получить доступ к последовательному ряду чисел
  // с помощью индексатора
  for (int i = 0; i < 5; i++)
     richTextBox2.AppendText("Следующее число равно
" + ob[i].ToString() + "\n");        // 0, 2, 4, 6, 8
  ob.SetStart(100);
  richTextBox2.AppendText("\nНачать с числа 100\n");
  for (int i = 0; i < 5; i++)
     richTextBox2.AppendText("Следующее число равно
" + ob.Next.ToString() + "\n");
}
```

Пример 5 (ООП)

```
public interface ISeries{
    void Reset(); // перезапустить
    void SetStart(int x); // задать начальное значение
    int Next { // интерфейсное свойство
        get; // возвратить следующее по порядку число
        set; // установить следующее число
    }
    int this[int index] { // интерфейсный индексатор
        get; // возвратить указанное в ряду число
    }
}
```

Пример 5 (ООП)

```
class ByTwos : ISeries{ // реализовать интерфейс ISeries
    int start;        int val;
    public ByTwos(){ start = 0;   val = 0;}
    public int Next{ // свойство
        get { val += 2;        return val;}
        set { val = value; }
    }
    public int this[int index]{ // индексатор
        get{
            val = 0;
            for (int i = 0; i < index; i++) val += 2;
            return val;
        }
    }
```

Пример 5 (ООП)

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

Пример 6 (ООП)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace WindowsFormsAppРазное{

    public partial class Form2 : Form  {
        public Form2()  {   InitializeComponent();  }
```

Пример 6 (ООП)

```
private void button6_Click(object sender, EventArgs e) {
    Random rnd = new Random();
    int n = 5;
    By1 ob = new By1(n);
   By2 ob2 = new By2(n);
   for (int i = 0; i < n; i++) // использование
индексатора
         ob2[i] = ob[i] = rnd.Next(-150, 100);
   // int[] m = new int[n];
   // for (int i = 0; i < n; i++)  m[i] =  rnd.Next(-15, 10);
   // ob.Next = m; // если определен set в свойстве Next
   richTextBox2.AppendText("\n Массив ob исходный \n");
   for (int i = 0; i < n; i++)
richTextBox2.AppendText(ob[i].ToString() + "\t");
```

Пример 6 (ООП)

```
richTextBox2.AppendText("\n Среднее арифметическое элементов массива ob - " + ob.Res(ob.Next) + "\n");
ob.Sett(ob.Next);
richTextBox2.AppendText("\n Массив ob после изменения \n");
for (int i = 0; i < n; i++)
    richTextBox2.AppendText(ob[i].ToString() + "\t");
richTextBox2.AppendText("\n");
Array.Sort(ob.Next);
richTextBox2.AppendText("\n Массив ob после сортировки \n");
for (int i = 0; i < n; i++)
    richTextBox2.AppendText(ob[i].ToString() + "\t");
richTextBox2.AppendText("\n");
richTextBox2.AppendText("\n Массив ob2 исходный \n");
for (int i = 0; i < n; i++)
    richTextBox2.AppendText(ob2[i].ToString() + "\t");
```

Пример 6 (ООП)

```
   richTextBox2.AppendText("\n");
   richTextBox2.AppendText("\n Частное 1-го и 2-го элементов
    массива ob2 - " + ob2.Res(ob2.Next) + "\n");
    ob2.Sett(ob2.Next);
    richTextBox2.AppendText("\n Массив ob2 после изменения \n");
    for (int i = 0; i < n; i++)
        richTextBox2.AppendText(ob2[i].ToString() + "\t");
    richTextBox2.AppendText("\n");
    Array.Sort(ob2.Next);
  richTextBox2.AppendText("\n Массив ob2 после сортировки \n");
    for (int i = 0; i < n; i++)
        richTextBox2.AppendText(ob2[i].ToString() + "\t");
    richTextBox2.AppendText("\n");
  }
}
```

Пример 6 (ООП)

```
public interface ISers {
    double Res(int[] z);
    void Sett(int[] z);
    int[] Next  { // интерфейсное свойство
        get;
      //  set; // не установлено
    }
    int this[int index]  { // интерфейсный индексатор
        get;
        set;
    }
}
```

Пример 6 (ООП)

```
class By1 : ISers { // реализовать интерфейс ISeries
    int[] a; // ссылка на базовый массив
    int len; // длина массива

    public By1(int size) { // конструктор
        a = new int[size];
        len = size;
    }
    public int[] Next  { // свойство
        get { return a; }
     //   set { a = value; }
    }
```

Пример 6 (ООП)

```
public int this[int index] {
    get { return a[index]; }
    set  {
        if (value > 0) a[index] = value;
        else a[index] = 0;
    }
}
public double Res(int[] z) {
// среднее арифметическое элементов массива
    double rez = 0;
    for (int i = 0; i < len; i++) rez += z[i];
    return rez / len;
}
```

Пример 6 (ООП)

```
        public void Sett( int[] z) {
           // значения массива увеличить на 1
            for (int i = 0; i < len; i++) z[i] += 1;
        }
      }
class By2 : ISers{
    int[] a; // ссылка на базовый массив
    int len; // длина массива
    public By2(int size) { a = new int[size];    len = size;  }
    public int[] Next    { // свойство
         get { return a; }
       //  set { a = value; }
    }
```

Пример 6 (ООП)

```
    public int this[int index] {
        get { return a[index]; }
        set {  if (value < 0) a[index] = value;
                 else a[index] = 0;
        }
    }
    public double Res(int[] z) { // частное 1 и 2 элементов массива
        return (double)(z[0] + z[1]) / len;
    }
    public void Sett(int[] z) {
        // значения массива изменены на -1
       for (int i = 0; i < len; i++) z[i] -= 1;
    }
}
```

Пример 7

```
using System;
public interface IA {
   void Meth1();
   void Meth2();
}
public interface IB : IA {
   void Meth3();
}
class MyClass : IB {
   public void Meth1() {
      Console.WriteLine("Реализовать метод Methl().");
   }
   public void Meth2() {
      Console.WriteLine("Реализовать метод Meth2().");
   }
```

Пример 7

```
  public void Meth3() {
    Console.WriteLine("Реализовать метод Meth3().");
  }
}
class IFExtend {
  static void Main() {
    MyClass ob = new MyClass();
    ob.Meth1();
    ob.Meth2();
    ob.Meth3();
  }
}
```

Интерфейсы

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;
  }
}
```

Интерфейсы

Для явной реализации интерфейсного метода могут быть **две причины**.

Во-первых, когда интерфейсный метод реализуется с указанием его полного имени, то такой метод оказывается доступным не посредством объектов класса, реализующего данный интерфейс, а по интерфейсной ссылке. Следовательно, явная реализация позволяет реализовать интерфейсный метод таким образом, чтобы он не стал открытым членом класса, предоставляющего его реализацию.

И **во-вторых**, в одном классе могут быть реализованы два интерфейса с методами, объявленными с одинаковыми именами и сигнатурами. Но неоднозначность в данном случае устраняется благодаря указанию в именах этих методов их соответствующих интерфейсов.

Пример 8

```
using System;  // 1 случай
interface IEven {
   bool IsOdd(int x);        bool IsEven(int x);
}
class MyClass : IEven {
   bool IEven.IsOdd(int x) { // явная реализация
      if ((x%2) != 0) return true;
      else return false;
   }
public bool IsEven(int x) { // обычная реализация
      IEven o = this; // интерфейсная ссылка на вызывающий объект
      return !o.IsOdd(x);
   }
}
```

Пример 8

```
class Demo {
   static void Main() {
      MyClass ob = new MyClass ();
      bool result;
      result = ob.IsEven(4);
      if (result) Console.WriteLine("4 — четное число.");
      // result = ob.IsOdd(4); // ошибка
      IEven iRef = (IEven) ob;
      result = iRef.IsOdd(3);
      if (result) Console.WriteLine("3 — нечетное число.");
   }
}
```

Пример 9

```
using System;  //  2 случай
interface IMyIF_A {
   int Meth(int x);
}
interface IMyIF_B {
   int Meth(int x);
}
class MyClass : IMyIF_A, IMyIF_B {
   int IMyIF_A.Meth(int x) {
      return x + x;
   }
   int IMyIF_B.Meth(int x) {
      return x * x;
   }
```

Пример 9

```
  public int MethA(int x){
    IMyIF_A a_ob;  // интерфейсная ссылка
    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
  }
}
```

Пример 9

```
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));
   }
}
```

Операторы is и as

В **C#** существует **два способа** узнать, поддерживает ли объект данный интерфейс. Первый заключается в применении оператора **is**. Его синтаксис:

выражение **is** тип

Оператор **is** возвращает значение **true**, если выражение (которое должно иметь ссылочный тип) может быть безопасно (то есть без вызова исключения) приведено к типу, указанному справа от ключевого слова **is**, обозначающего операцию.

Операторы is и as

Второй способ заключается в применении оператора **as**. Его синтаксис:

выражение **as** тип

Оператор **as** сочетает в себе оператор **is** и приведение типа. Он сперва проверяет допустимость требуемого преобразования (то есть возвратит ли оператор **is** значение **true**), а затем выполняет приведение типа, если оно безопасно. В противном случае (когда операция **is** возвратила бы **false**) оператор **as** возвращает значение **null**.

Применение оператора **as** позволяет обойтись без обработки исключений. В то же время удается избежать двойной проверки допустимости приведения типа.

Пример 10

```
using System;
interface IStorable {
   void Read();
   void Write(object obj);
   int Status { get; set; }
}

interface ICompressible { // новый интерфейс
   void Compress();
   void Decompress();
}
```

Пример 10

```
public class Document : IStorable {
   public Document(string s) {
      Console.WriteLine("Создание документа: {0}", s);
   }
   public void Read() { // IStorable.Read
      Console.WriteLine("Реализация метода Read для
    IStorable");
   }
   public void Write(object o) { // IStorable.Write
      Console.WriteLine("Реализация метода Write для
    IStorable");
   }
```

Пример 10

```
public int Status { // IStorable.Status
   get { return status;  }
   set { status = value;  }
}
/* // реализация интерфейса ICompressible
public void Compress(){
   Console.WriteLine("Реализация Compress");
}
public void Decompress() {
   Console.WriteLine("Реализация Decompress");
}
 */
   private int status = 0;
}
```

Пример 10

```
public class Tester {
   static void Main() {
      Document doc = new Document("Test Document");
      if (doc is IStorable) {
         IStorable isDoc = (IStorable) doc; // операция
       // приведения типа объекта к типу интерфейса
         isDoc.Read();
      }
      if (doc is ICompressible) { // отрицательный результат
            ICompressible icDoc = (ICompressible) doc;
            icDoc.Compress();
      }
   }
}
```

Пример 11

```
static void Main() {
   Document doc = new Document("Test Document");
   IStorable isDoc = doc as IStorable;
   if (isDoc != null) isDoc.Read();
   else Console.WriteLine("IStorable не поддерживается");
   ICompressible icDoc = doc as ICompressible;
   if (icDoc != null) icDoc.Compress();
   else Console.WriteLine("Compressible не
    поддерживается");
}
```

Интерфейсы

IEnumerable Составляет список объектов классов коллекций с помощью оператора foreach

ICollection Реализуется всеми классами коллекций для обеспечения доступа к методу CopyTo() и свойствам Count, IsSynchronizednSyncRoot

IComparer, IComparable Сравнивает два объекта классов коллекций при их сортировке

ICloneable Позволяет клонировать объекты

IList Используется объектами классов коллекций, индексируемыми как массив

IDictionary Используется классами коллекций, осуществляющими доступ по ключу или значению, таких как Hashtable nSortedList

IdictionaryEnumerator Позволяет просмотреть (с помощью оператора foreach) объекты классов коллекций, поддерживающих интерфейс IDictionary

Пример 12

```
class Program{  // нет копирования
    static void Main(string[] args)    {
        Person p1 = new Person { Name="Tom", Age = 23 };
        Person p2 = p1;
        p2.Name = "Alice";
        Console.WriteLine(p1.Name); // Alice
        Console.Read();
    }
}

class Person{
    public string Name { get; set; }
    public int Age { get; set; }
}
```

Пример 13

```
public interface ICloneable{
    object Clone();
}
```

Реализация интерфейса в классе **Person** могла бы выглядеть следующим образом:

```
class Person : ICloneable{
    public string Name { get; set; }
    public int Age { get; set; }
    public object Clone()    {
        return new Person { Name = this.Name, Age =
    this.Age };
      // return this.MemberwiseClone(); // или
    }
}
```

Пример 13

Использование:

```
class Program{
    static void Main(string[] args)    {
        Person p1 = new Person { Name="Tom", Age = 23 };
        Person p2 = (Person)p1.Clone();
        p2.Name = "Alice";
        Console.WriteLine(p1.Name); // Tom
        Console.Read();
    }
}
```

Пример 14

```
public interface ICloneable{ object Clone(); }
class Person : ICloneable{
    public string Name { get; set; }
    public int Age { get; set; }
    public Company Work { get; set; }
    public object Clone()    {
        Company company = new Company { Name =
    this.Work.Name };
        return new Person   {
            Name = this.Name,    Age = this.Age,
            Work = company
        };
    }
}
```

Пример 14

```
class Company{
    public string Name { get; set; }
}
```

В этом случае при копировании новая копия будет указывать на другой объект **Company**:

```
Person p1 = new Person { Name = "Tom", Age = 23,
            Work = new Company { Name = "Microsoft" } };
Person p2 = (Person)p1.Clone();
p2.Work.Name = "Google";
p2.Name = "Alice";
Console.WriteLine(p1.Name); // Tom
Console.WriteLine(p1.Work.Name); // Microsoft
```

Сортировка объектов. Интерфейс IComparable

Для сортировки наборов сложных объектов применяется интерфейс **IComparable**.

```
public interface IComparable{
    int CompareTo(object o);
}
```

Метод **CompareTo** предназначен для сравнения текущего объекта с объектом, который передается в качестве параметра **object o**. На выходе он возвращает целое число, которое может иметь одно из трех значений:

Меньше нуля. Значит, текущий объект должен находиться перед объектом, который передается в качестве параметра

Равен нулю. Значит, оба объекта равны

Больше нуля. Значит, текущий объект должен находиться после объекта, передаваемого в качестве параметра

Пример 15

```
class Person : IComparable{
    public string Name { get; set; }
    public int Age { get; set; }
    public int CompareTo(object o)    {
        Person p = o as Person;
        if (p != null)
            return this.Name.CompareTo(p.Name);
        else
            throw new Exception("Невозможно сравнить два
    объекта");
    }
}
```

Здесь в качестве критерия сравнения выбрано свойство **Name** объекта **Person**.

Пример 15

Применение:

```
Person p1 = new Person { Name = "Bill", Age = 34 };
Person p2 = new Person { Name = "Tom", Age = 23 };
Person p3 = new Person { Name = "Alice", Age = 21 };

Person[] people = new Person[] { p1, p2, p3 };
Array.Sort(people);

foreach(Person p in people){
    Console.WriteLine("{0} - {1}", p.Name, p.Age);
}
```

Пример 15

Интерфейс **IComparable** имеет обобщенную версию, поэтому ее можно сократить и упростить его применение в классе Person:

```
class Person : IComparable<Person>{
    public string Name { get; set; }
    public int Age { get; set; }
    public int CompareTo(Person p) {
        return this.Name.CompareTo(p.Name);
    }
}
```

Применение компаратора

Кроме интерфейса **IComparable** платформа .**NET** также предоставляет интерфейс **IComparer**.

```
interface IComparer{
    int Compare(object o1, object o2);
}
```

Метод **Compare** предназначен для сравнения двух объектов **o1** и **o2**. Он также возвращает три значения, в зависимости от результата сравнения: если первый объект больше второго, то возвращается число больше 0, если меньше - то число меньше нуля; если оба объекта равны, возвращается ноль.

Пример 16

```
class PeopleComparer : IComparer<Person>{
    public int Compare(Person p1, Person p2)    {
      if (p1.Name.Length > p2.Name.Length)  return 1;
      else if (p1.Name.Length < p2.Name.Length) return -1;
            else return 0;
    }
}
Person p1 = new Person { Name = "Bill", Age = 34 };
Person p2 = new Person { Name = "Tom", Age = 23 };
Person p3 = new Person { Name = "Alice", Age = 21 };
Person[] people = new Person[] { p1, p2, p3 };
Array.Sort(people, new PeopleComparer());
foreach(Person p in people){
   Console.WriteLine("{0} - {1}", p.Name, p.Age);
}
```

Контрольные вопросы

1. Чем интерфейс отличается от абстрактного класса?
2. Должен ли класс реализовывать все методы всех своих интерфейсов-предков?
3. Какие операции используются для проверки, реализует ли класс заданный интерфейс?

Контрольные вопросы

Вопрос 1. Класс Tester реализует интерфейсы **IFoo** и **IBar**:

```
interface IFoo{
    void Execute();
}
interface IBar{
    void Execute();
}
class Tester : IFoo, IBar{
    public void Execute()    {
        Console.WriteLine("Tester Executes");
    }
}
```

Метод **Execute**() какого именно интерфейса реализует класс **Tester**?

Контрольные вопросы

Вопрос 2. Класс **Tester** реализует интерфейсы IFoo и IBar:

```
interface IFoo{
    void Execute();
}

interface IBar{
    void Execute();
}
```

Контрольные вопросы

Вопрос 2 (продолжение).

```
class Tester : IFoo, IBar{
    void IFoo.Execute()    {
        Console.WriteLine("IFoo Executes");
    }
    void IBar.Execute()    {
        Console.WriteLine("IBar Executes");
    }
}
```

Реализация метода **Execute**() из какого интерфейса будет вызвана в следующем коде:

```
Tester t = new Tester();
t.Execute();
```