ИНТЕРФЕЙСЫ

ОПРЕДЕЛЕНИЯ

В ООП

Интерфейс — множество операций, которое определяет набор услуг (службу), предоставляемых классом или компонентом

В ЯЗЫКЕ C#

Интерфейсом **называется чисто абстрактный класс, содержащий только описания без реализации**

ЦЕЛИ ПРИМЕНЕНИЯ

1. Определение действий, выполняемых классом, без указания способа их выполнения, т.е. необходимость отделить описание класса от реализации.

2. Поддержка множественного наследования в языке C#

СИНТАКСИС ИНТЕРФЕЙСА

```
[атрибуты][спецификаторы] interface
имя_интерфейса [:список_родителей]
{
тело_класса
}
```

Спецификаторы:

new, public, protected, internal и private.

Спецификатор new применяется для вложенных интерфейсов. Остальные спецификаторы управляют видимостью интерфейса. По умолчанию интерфейс доступен только из сборки, в которой он описан(internal).

ПРИМЕР ОБЪЯВЛЕНИЯ

```
// Этот интерфейс определяет возможность
 // нарисовать геометрическую фигуру

interface IDraw
{
// автоматически (неявный образом) этот член
// интерфейса становится абстрактным

void Draw();
}
```

***Примечание*. Имена интерфейсов принято начинать с буквы I.**

ОТЛИЧИЯ ИНТЕРФЕЙСА ОТ АБСТРАКТНОГО КЛАССА

1. При объявлении интерфейса все его методы неявно имеют модификатор **public**, и явное указание модификатора доступа приведет к ошибке (даже если он будет **public**);
2. методы интерфейса не содержат тела с реализацией. После объявления метода следует точка с запятой;
3. интерфейс не может содержать данных, в нем могут быть объявлены только методы, события, свойства и индексаторы;
4. класс, наследующий интерфейс, обязан полностью реализовать все методы интерфейса, в то время как потомок абстрактного класса может реализовать лишь некоторые методы родительского абстрактного класса, оставаясь абстрактным классом;
5. класс может иметь в списке предков несколько интерфейсов, при этом он должен определять все их методы.

Реализация интерфейса

После объявления интерфейса следует создать класс (или классы), реализующий интерфейс. Класс, исполняющий интерфейс, называется *интерфейсным*. При объявлении интерфейсного класса сначала указывается базовый класс, если он есть, а затем имя интерфейса:

```
class  имя класса : имя интерфейса
     {
 тело класса
      }
```

Тело интерфейсного класса может содержать не только реализацию методов интерфейса, но и не описанные в интерфейсе методы, а также поля, свойства и события. Однако он *должен* содержать реализацию всех интерфейсных методов.

Пример реализации интерфейса IDraw в классах Rect и Ellipse

```
interface IDraw   // объявление интерфейса
   {
      void Draw();
   }
class Rect : IDraw  {   // интерфейсный класс Rect
// переопределение интерфейсного метода Draw() в классе Ellipse
      public void Draw()
      {
         Console.WriteLine("Рисуем прямоугольник");
      }
   }
class Ellipse : IDraw  {  // интерфейсный класс Ellipse
// переопределение интерфейсного метода Draw() в классе Ellipse
      public void Draw()
      {
         Console.WriteLine("Рисуем эллипс");
      }
   }
```

Пример реализации интерфейса IDraw в классах Rect и Ellipse (продолжение)

```
 static void Main(string[ ] args) {
// объявление переменной val как ссылки на интерфейс IDraw. Эта ссылка
// может ссылаться на любой объект, который реализует интерфейс IDraw
        IDraw[ ] val = new IDraw[2];
// ссылка на интерфейс получает адрес объекта интерфейсного класса Rect
        val[0] = new Rect();
// ссылка на интерфейс получает адрес объекта интерфейсного класса Ellipse
        val[1] = new Ellipse();
        for (int i = 0; i < val.Length; i++)
         {
            val[i].Draw();
         }
    }
```

Результаты работы этой программы имеют вид:
Рисуем прямоугольник
Рисуем эллипс

НАСЛЕДОВАНИЕ

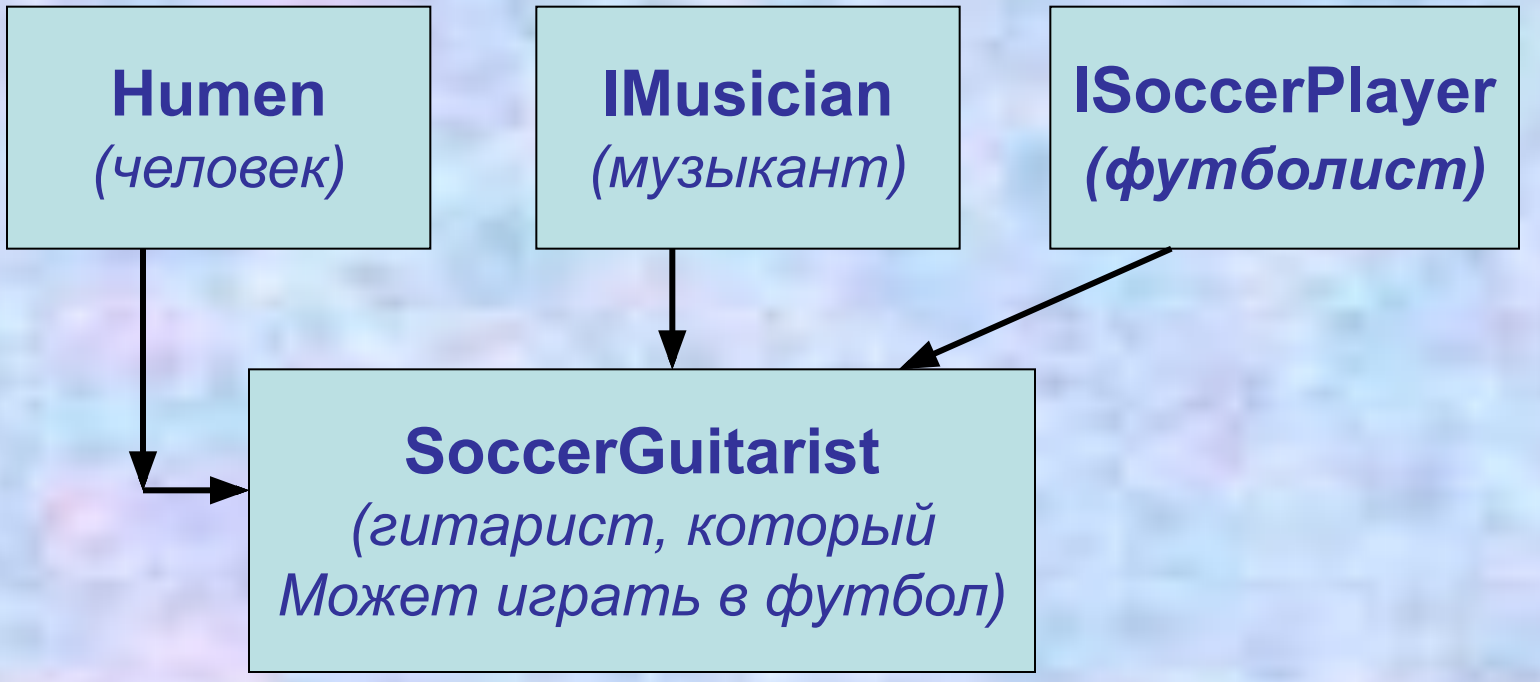
ИНТЕРФЕЙСОВ

МНОЖЕСТВЕННОЕ НАСЛЕДОВАНИЕ

Один класс может реализовывать (или наследовать) несколько интерфейсов одновременно. Это значит, что производный класс в языке C# может наследоваться только от одного базового класса и любого количества интерфейсов.

Каждый класс в языке C# имеет один базовый класс System.Object и дополнительно может иметь любое количество базовых интерфейсов.

Наследование от нескольких базовых интерфейсов



Наследование от нескольких базовых интерфейсов (пример)

```
interface IMusician    {    // интерфейс «музыкант»
        void Tune ();
        void Play();
      }
interface ISoccerPlayer    {  // интерфейс «футболист»
        void Play();
    }
class Human    {          //  базовый клас «человек»
        public Human()
        {
            Console.WriteLine("Я человек");
                  }
    }
```

Наследование от нескольких базовых интерфейсов (продолжение примера)

```
// класс SoccerGuitarist производный от  Human и реализует  интерфейсы классов
// IMusician и ISoccerPlayer

  class SoccerGuitarist : Human, IMusician, ISoccerPlayer {
      public void Tune()
      {
         Console.WriteLine("Я гитарист");
      }

// IMusician.Play()  – это явная реализация метода  Play()  в классе  SoccerGuitarist, требует
//  указания полного  имени метода, которое  включает имя его интерфейса – IMusician
// модификатор доступа при явной реализации указывать нельзя!
     void IMusician.Play()  {   Console.WriteLine("Я играю на гитаре");        }

// ISoccerPlayer.Play()  – это явная реализация метода  Play() в классе SoccerGuitarist,требует
// указания полного имени метода, которое включает имя его интерфейса – ISoccerPlayer
   void ISoccerPlayer.Play()
      {
         Console.WriteLine("Я играю в футбол");
      }
   }
```

Наследование от нескольких базовых интерфейсов (продолжение примера)

```
static void Main(string[] args)
        {
// при создании объекта производного класса неявно вызывается конструктор
// по умолчанию базового класса Human
            SoccerGuitarist val = new SoccerGuitarist();
             val.Tune();
// при доступе к методу Play() выполняется явное приведение ссылки val
// к типу интерфейса, метод которого мы вызываем
            ((IMusician)val).Play();
            ((ISoccerPlayer) val).Play();
        }
```

Результаты работы этой программы имеют вид:

Я человек
Я гитарист
Я играю на гитаре
Я играю в футбол

Особенности реализации методов явным образом при наследовании от нескольких интерфейсов:

1. При доступе к методу необходимо явно приводить ссылку к типу интерфейса, метод которого мы вызываем, или вызвать метод через ссылку на этот интерфейс.

2. Нельзя указывать для реализуемого метода модификатор доступа.

3. Нельзя объявить метод как виртуальный.

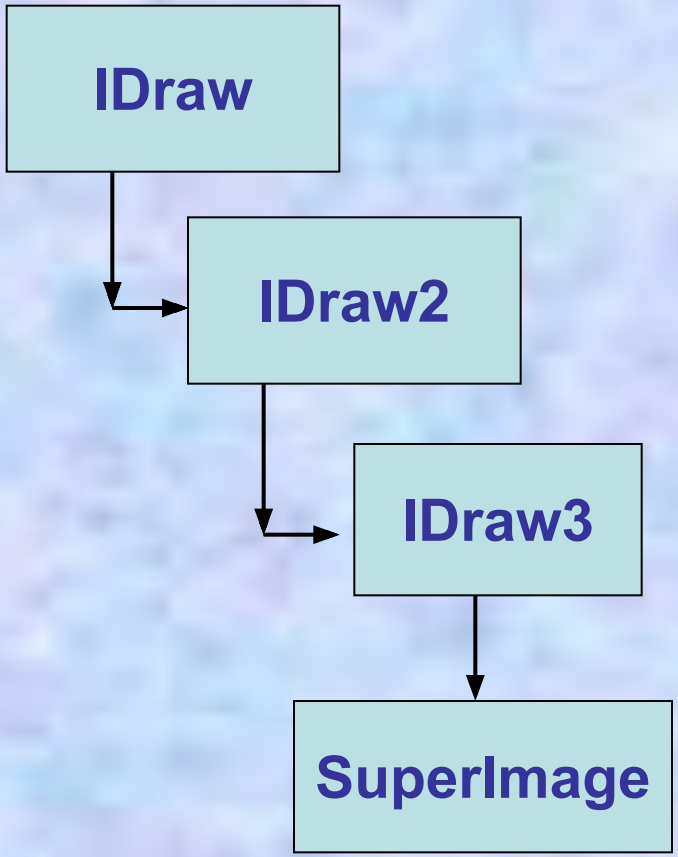
Создание иерархий интерфейсов

```
// Базовый интерфейс IDraw
   interface IDraw    {
      void Draw();
   }

// интерфейс, производный от IDraw
interface IDraw2 : IDraw {
void DrawToPrinter();  }

// интерфейс, производный от IDraw2
interface IDraw3 : IDraw2  {
void DrawToMetaFile();  }

 public class SuperImage : IDraw3
{ }
```



Создание иерархий интерфейсов (пример)

```
public class SuperImage : IDraw3
  {
     // явная реализация  метода интерфейса
     void IDraw.Draw()
     {
        Console.WriteLine("Вывод на консоль...");      // Вывод на консоль
     }
     void IDraw2.DrawToPrinter()
     {
        Console.WriteLine("Вывод на принтер...");      // Вывод на принтер
     }
     void IDraw3.DrawToMetaFile()
     {
        Console.WriteLine("Вывод в файл...");     // Вывод в файл
     }
  }
```

Создание иерархий интерфейсов (продолжение примера)

```
static void Main(string[] args)
      {
         Console.WriteLine("***** Работа с классом SuperImage *****");
         SuperImage si = new SuperImage(); //создание объекта класса
         IDraw itfDraw = (IDraw) si; // Получаем ссылку на интерфейс IDraw
         itfDraw.Draw();
 // получаем ссылку на интерфейс IDraw3, для чего используем оператор is
// оператор is выполняет проверку принадлежности объекта определенному типу
// тип объекта сравнивается с тем, который записан справа от оператора is
 // если объект не поддерживает интерфейс условие станет равно false
         if (itfDraw is IDraw3)          {
            IDraw3 itfDraw3 = (IDraw3)itfDraw;
            itfDraw3.DrawToMetaFile();
            itfDraw3.DrawToPrinter();
         }
```

Результаты работы этой программы имеют вид:

***** Работа с классом SuperImage *****

Вывод на консоль…

Вывод в файл…

Вывод на принтер…

Стандартные интерфейсы .NET. Интерфейс IComparable

Интерфейс IComparable позволяет производить сортировку объектов, основываясь на специально определенном внутреннем ключе.

Определение интерфейса:

```
interface IComparable
  {
  int CompareTo(object o):
   }
```

Интерфейс IComparable (пример)

Пусть пользователь создал массив объектов класса **Car** следующим образом:

```
Car[] myAutos = new Car[5];
myAutos[0] = new Car(123, "Волга");
myAutos[1] = new Car(6, "Ауди");
myAutos[2] = new Car(16, "Мерседес");
myAutos[3] = new Car(13, "Запорожец");
myAutos[4] = new Car(26, "Нива");
```

Задача: Выполнить сортировку созданного массива по различным критериям: номеру автомобиля и его названию.

Интерфейс IComparable (пример)

Для выполнения сортировки по номеру автомобиля в классе Car необходимо выполнить явную реализцию метода **CompareTo интерфейса IComparable:**

```
int IComparable.CompareTo(object o)
    {
        Car temp = (Car)o;
        if(this.CarID > temp.CarID)
            return 1;
        if(this.CarID < temp.CarID)
            return -1;
        else    return 0;
    }
```

Интерфейс IComparer

Для реализации сортировки еще по одному критерию – имени автомобиля воспользуемся возможностями еще одного стандартного интерфейса − **IComparer**, определенного в пространстве имен **System.Collections.**

// Стандартный способ сравнения двух объектов

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

IComparer обычно не реализуется напрямую внутри класса, объекты которого необходимо сортировать (в нашем случае — Car). Этот интерфейс реализуется во вспомогательных классах, которых может быть любое количество — по одному вспомогательному классу на каждую переменную, по которой производится сортировка.

Стандартные интерфейсы .NET. Интерфейсы IComparable и IComparer (пример применения)

```
public class Car : IComparable
    {// этот вспомогательный класс нужен для сортировки объектов
    // полю по PetName   он реализует интерфейс IComparer
    private class SortByPetNameHelper : IComparer
     {
     public SortByPetNameHelper()  {} ;
     // сравниваем имена объектов (PetName)
     int IComparer.Compare(object o1, object o2)       {
           Car t1 = (Car)o1;
           Car t2 = (Car)o2;
           return String.Compare(t1.PetName, t2.PetName);
     }
     }
```

Стандартные интерфейсы .NET. Интерфейсы IComparable и IComparer (продолжение примера применения)

```
// поля класа .
 private int CarID;  //номер автомобиля
 private string petName;   // название автомобиля
 // методы-свойства класса
 public int ID {
      get { return CarID; }
      set { CarID = value; }
 }
 public string PetName     {
      get { return petName; }
      set { petName = value; }
 }
```

Стандартные интерфейсы .NET. Интерфейсы IComparable и IComparer (продолжение примера применения)

```
// конструкторы
  public Car() {} ;
  public Car(int id, string name)
  { this.CarID = id; this.petName = name;}
  //реализация интерфейса  IComparable
  int IComparable.CompareTo(object o)
        {
        Car temp = (Car)o;
        if(this.CarID > temp.CarID)
              return 1;
        if(this.CarID < temp.CarID)
              return -1;
        else return 0;
  }
// свойство возвращает результат сравнения имен автомобилей
  public static IComparer SortByPetName
  { get { return (IComparer) new SortByPetNameHelper(); } }
}
```

Стандартные интерфейсы .NET. Интерфейсы IComparable и IComparer (продолжение примера применения)

```
static void Main(string[] args)      {
            // Создание массива объектов типа  Car
              Car[] myAutos = new Car[5];
              myAutos[0] = new Car(123, "Волга");
              myAutos[1] = new Car(6, "Ауди");
              myAutos[2] = new Car(16, "Мерседес");
              myAutos[3] = new Car(13, "Запорожец");
              myAutos[4] = new Car(26, "Нива");
 // Выводим созданный массив объектов.
  Console.WriteLine("***** Вывод  не отсортированных данных*****");
        foreach(Car c in myAutos)
```

Стандартные интерфейсы .NET. Интерфейсы IComparable и IComparer (продолжение примера применения)

```
    Console.WriteLine("{0} {1}", c.ID, c.PetName);
    // Используем возможности реализованного интерфейса IComparable.
                Array.Sort(myAutos);
    // Выводим информацию из упорядоченного массива
    Console.WriteLine("\n***** Сортировка по номерам автомобилей *****");
          foreach(Car c in myAutos)
                Console.WriteLine("{0} {1}", c.ID, c.PetName);
    // сортировка по названию автомобилей  используем вариант перегруженного
  // в классе System.Array  метода Sort
    // Array.Sort(myAutos, new SortByPetName());
          Array.Sort(myAutos, Car.SortByPetName);
          // Выводим информацию из упорядоченного массива
    Console.WriteLine("\n***** Сортировка по названиям автомобилей *****");
          foreach(Car c in myAutos)
                Console.WriteLine("{0} {1}", c.ID, c.PetName);
      }
```

Стандартные интерфейсы .NET. Интерфейсы IComparable и IComparer (продолжение примера применения)

Результаты работы этой программы имеют вид:

```
***** Вывод  не отсортированных данных*****
123 Волга
6 Ауди
16 Мерседес
13 Запорожец
26 Нива
***** Сортировка по номерам автомобилей *****
6 Ауди
13 Запорожец
16 Мерседес
26 Нива
123 Волга
***** Сортировка по названиям автомобилей *****
6 Ауди
123 Волга
13 Запорожец
16 Мерседес
26 Нива
```