Интерфейсы

- Интерфейсом называют особый класс, который инкапсулирует абстрактный набор функциональных возможностей.
- Объявляется интерфейс с помощью специального ключевого слова interface:

interface IИмя { содержимое }

С помощью интерфейсов можно описать, какими функциями должен обладать класс и что он должен уметь делать. При этом интерфейс не имеет самого кода и не реализует функции.

Особенности интерфейсов

Как специальный (можно сказать, упрощенный) класс, интерфейс обладает следующими особенностями:

- может содержать только объявления методов или свойств (без определений). Такие элементы называют абстрактными;
- элементы интерфейса всегда открыты (public по определению) и модификаторы защиты при их объявлении также не указываются;
- имя интерфейса рекомендуют начинать с I.

Особенности интерфейсов

Что может определять интерфейс?

- методы;
- свойства;
- индексаторы;
- события;
- статические поля и константы.

Особенности интерфейсов

При объявлении класса или структуры, поддерживающих интерфейс, имя интерфейса указывается в списке базовых классов. У структур такой список может состоять только из интерфейсов.

Если интерфейс поддерживает производный класс, имя его базового класса в списке должно быть первым.

В любом случае все элементы поддерживаемого интерфейса (то есть абстрактные методы) должны быть определены (реализованы) в классе или структуре.

В условиях отсутствия в C# множественного наследования от классов интерфейсы позволяют поддержать неограниченное количество вариантов поведения (то есть *полиморфизм*).

Операции для работы с интерфейсами

- Операция is бинарная логическая. Возвращает значение true, если тип объекта совместим с указанным интерфейсом, и false – в противном случае. Проще говоря, операция проверяет, есть ли в составе данного объекта элементы указанного интерфейса. Синтаксис: **Name is IName**.
  Name – имя объекта; **IName** – имя интерфейса.
- Операция as также бинарная и возвращает ссылку на элемент-интерфейс в составе объекта
  IName i = Name as IName;
  Здесь i – ссылка на интерфейс. Если тип данного объекта не поддерживает данный интерфейс, операция as возвращает null. Говорят, что при использовании данной операции реализуется доступ через интерфейсную ссылку. После этого допустим следующий вызов:
  i.ИмяЭлементаИнтерфейса;
  ИмяЭлементаИнтерфейса – имя метода или свойства.

Операции для работы с интерфейсами

Ссылку на элемент-интерфейс (который входит в состав объекта) можно получить и с помощью операции явного приведения типов (это называется *объектной ссылкой*):

…(IName)Name.ИмяЭлементаИнтерфейса …

Один и тот же интерфейс может поддерживаться (и реализовываться) типами из различных иерархий наследования. Это даёт возможность использовать полиморфизм для семантически несовместимых объектов, если все они поддерживают один интерфейс. К примеру, можно объявить массив интерфейсных объектов

IName[] Имя = { new Type1(), new Type2() …};

и тогда в цикле с помощью инструкции

… Имя[i].ИмяЭлементаИнтерфейса …

можно осуществить доступ к элементу интерфейса конкретного объекта, если Type1, Type2 – типы, реализующие интерфейс IName.

Стандартные интерфейсы

- ICloneable
- IComparable и IComparer
- IEnumerator и IEnumirable
- IDisposible

ICloneable

- Данный интерфейс предназначен для создания *глубоких* копий объектов.

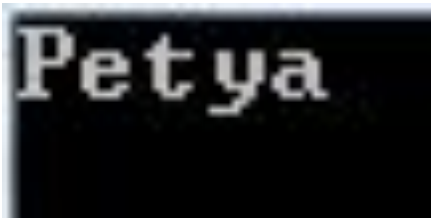
- Интерфейс *клонируемый* содержит объявление метода Clone():

public interface ICloneable
{ object Clone();}

Зачем он нужен?

- Пример:

```
public class ClassUser
{
    1 reference
    public string Login { set; get; }

    1 reference
    public string Password { set; get; }
}
```



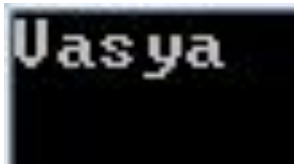

```
static void Main(string[] args)
{
    ClassUser cl1 = new ClassUser() { Login = "Vasya", Password = "Pupkin" };
    ClassUser cl2 = cl1;
    cl2.Login = "Petya";
    Console.WriteLine(cl1.Login);
    Console.ReadKey();
}
```

Зачем он нужен?

- В данном случае объекты cl1 и cl2 будут указывать на **один и тот же объект** в памяти, поэтому изменения свойств в переменной cl2 затронут также и переменную cl1.
- Чтобы переменная cl2 указывала на новый объект, но со значениями из cl1, его необходимо **клонировать**.

Реализация

```
public class ClassUser : ICloneable
{
    5 references
    public string Login { set; get; }

    3 references
    public string Password { set; get; }

    0 references
    public object Clone()
    {
        return new ClassUser() { Login = this.Login, Password = this.Password };
    }
}
```




```
static void Main(string[] args)
{
    ClassUser cl1 = new ClassUser() { Login = "Vasya", Password = "Pupkin" };
    ClassUser cl2 = (ClassUser)cl1.Clone();
    cl2.Login = "Petya";
    Console.WriteLine(cl1.Login);
    Console.ReadKey();
}
```

Можно проще

- Для сокращения кода копирования мы можем использовать специальный метод **MemberwiseClone()**, который возвращает копию объекта

```
public class ClassUser : ICloneable
{
    3 references
    public string Login { set; get; }

    1 reference
    public string Password { set; get; }

    1 reference
    public object Clone()
    {
        return MemberwiseClone();
    }
}
```

Недостаток

- Этот метод реализует **поверхностное (неглубокое) копирование**.

- Если в классе есть поля-объекты от других классов, то в объекте-клоне создастся не новый объект, а копируется ссылка от текущего

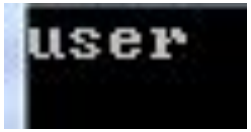
Добавим класс

```
public class Role
{
    0 references
    public string RoleName { set; get; }
}
```

```
public class ClassUser : ICloneable
{
    3 references
    public string Login { set; get; }

    1 reference
    public string Password { set; get; }

    0 references
    public Role UserRole { set; get; }

    1 reference
    public object Clone()
    {
        return MemberwiseClone();
    }
}
```

Что получаем

```
static void Main(string[] args)
{
    ClassUser cl1 = new ClassUser()
    {
        Login = "Vasya",
        Password = "Pupkin",
        UserRole = new Role() { RoleName = "admin" }
    };
    ClassUser cl2 = (ClassUser)cl1.Clone();
    cl2.UserRole.RoleName = "user";
    Console.WriteLine(cl1.UserRole.RoleName);
    Console.ReadKey();
}
```



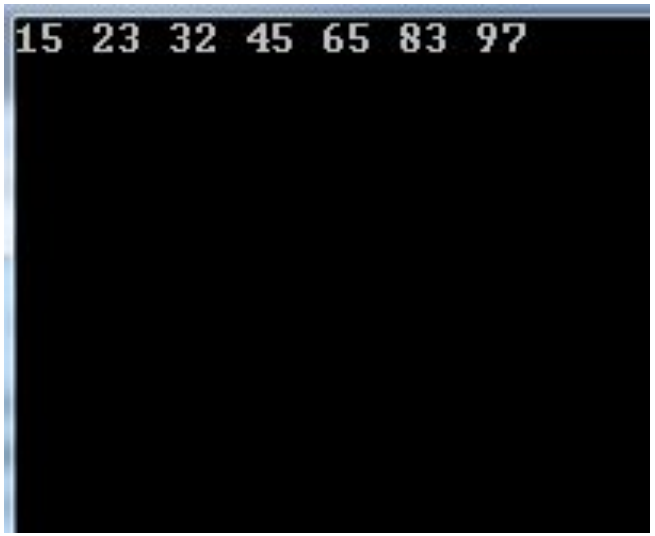
Глубокое копирование

```
public object Clone()
{
    return new ClassUser()
    {
        Login = this.Login,
        Password = this.Password,
        UserRole = new Role() { RoleName = this.UserRole.RoleName }
    };
}
```

```
public object Clone()
{
    return new ClassUser()
    {
        Login = this.Login,
        Password = this.Password,
        UserRole = (Role)this.UserRole.Clone()
    };
}
```

IComparable

```
static void Main(string[] args)
{
    int[] numbers = new int[] { 97, 45, 32, 65, 83, 23, 15 };
    Array.Sort(numbers);
    foreach (int n in numbers)
    {
        Console.Write(n + " ");
    }
    Console.ReadKey();
}
```



IComparable

- Однако метод Sort по умолчанию работает только для наборов примитивных типов, как int или string. Для сортировки наборов сложных объектов применяется интерфейс **IComparable**.

IComparable

Интерфейс IComparable (компарабельный) – простое сравнение – объявлен в пространстве имён System, содержит всего один метод, возвращающий результат сравнения двух объектов (текущего и obj):

interface IComparable { int CompareTo(object obj)}

Возвращаемое значение:

0 – объекты равны;

> 0 – текущий больше obj;

< 0 – текущий меньше obj.

Реализация

```
public class ClassUser : ICloneable, IComparable
{
    4 references
    public string Login { set; get; }

    2 references
    public string Password { set; get; }

    2 references
    public Role UserRole { set; get; }

    2 references
    public object Clone()...

    0 references
    public int CompareTo(object obj)
    {
        if (obj is ClassUser)
        {
            var user = obj as ClassUser;
            return this.Login.CompareTo(user.Login);
        }
        else
            throw new Exception("Невозможно сравнить два объекта");
    }
}
```

Еще вариант

```
public int CompareTo(object obj)
{
    if (obj is ClassUser)
    {
        var user = obj as ClassUser;
        var res = this.UserRole.RoleName.CompareTo(user.UserRole.RoleName);
        if (res != 0)
        {
            return res;
        }
        return this.Login.CompareTo(user.Login);
    }
    else
        throw new Exception("Невозможно сравнить два объекта");
}
```