# Определение интерфейсов

**Интерфейсы могут определять:**

**- Методы**

**- Свойства**

**- Индексаторы**

**- События**

**- Статические поля и константы (с C# 8.0)**

**Не могут определять нестатические переменные. Члены интерфейсов public по умолчанию. Сами интерфейсы, как и классы, по умолчанию internal. Но и к членам и к самим интерфейсам можно применять другие модификаторы доступа.**

**interface** IMyInterface

**{**

// константа

const int myConst **=** 0**;**

// статическая переменная

static int myStatic **=** 0**;**

// свойство

string MyProperty **{** **get;** **set;** **}** // это не автосвойство, а свойство без реализации

// реализация свойства по умолчанию

int MyIntProperty **{** **get** **{** **return** 0**;** **}** **}**

// метод

void MyMethod**();**

// метод с реализацией по умолчанию

void MyMethod2**()** **{** **...** **}** // с C# 8.0

// статический метод должен иметь реализацию по умолчанию

static void MyStaticMethod**()** **{** **...** **}**

**delegate** void MyDelegate**(**string text**);** // определение делегата для события

// событие

**event** MyDelegate MyEvent**;**

**}**

# Применение интерфейсов

**interface** IMyInterface **{** **...** **}**

class MyClass **:** IMyInterface **{** **...** **}**

struct MyStruct **:** IMyInterface **{** **...** **}**

**Если методы и свойства интерфейса не имеют модификатора доступа, то по умолчанию они public. При их реализации можно применять только public.**

**interface** IMovable

**{**

void Move**()** **=>** **...** // реализация по умолчанию

**}**

class Person **:** IMovable**;**

**...**

Person p **=** **new** Person**();**

// p.Move(); - ошибка, т.к. класс Person не имеет реализации метода Move(), но:

IMovable p **=** **new** Person**();**

p**.**Move**();** // используется реализация по умолчанию интерфейса

Множественная реализация интерфейсов

class MyClass **:** IMyInterface1**,** IMyInterface2 **...** **{** **...** **}**

# Явная реализация интерфейсов

**interface** ISchool

**{**

void Study**();**

**}**

**interface** IUniversity

**{**

void Study**();**

**}**

class Person **:** ISchool**,** IUniversity

**{**

// public void Study() => ... // реализация метода Stydy() сразу обоих интерфейсов

//или :

**public** ISchool**.**Study**()** **=>** **...** // метод существует только для интерфейса ISchool

**public** IUniversity**.**Study**()** **=>** **...** // метод существует только для интерфейса //IUniversity

**}**

Person person **=** **new** Person**();**

// person.Study(); ошибка, т.к. в Person нет реализации этого метода, она только для //интерфейсов

**((**ISchool**)** person**).**Study**();**

// или :

// безопасное приведение типа

**if** **(**person **is** IUniversity univirsity1**)** univirsity1**.**Study**();**

// или :

IUniversity univirsity2 **=** **new** Person**();**

univirsity2**.**Study**();**

Модификаторы доступа

**Если члены интерфейса не public, то можно использовать явную реализацию интерфейса без модификаторов:**

**interface** IMyInterface

**{**

**protected** **internal** string Name **{** **get;** **}**

**}**

class MyClass **:** IMyInterface

**{**

string name**;**

// явная реализация свойства в виде автосвойства

string IMyInterface**.**Name **{** **get** **=>** name**;** **}** // обратиться к такому свойству можно //только через объект типа интерфейса

**}**

**Либо использовать неявную реализацию с модификатором public:**

**interface** IMyInterface

**{**

**protected** **internal** string Name **{** **get;** **}**

**}**

class MyClass **:** IMyInterface

**{**

string name**;**

// неявная реализация свойства в виде автосвойства

**public** string Name **{** **get** **=>** name**;** **}** // обратиться к такому свойству можно через //объект типа класса

**}**

# Реализация интерфейсов в базовых и производных классах

**В абстрактном классе, расширяющем интерфейс, можно не реализовывать метод интерфейса, а сделать его абстрактным.**

**interface** MyInterface **{** string MyMethod**();** **}**

**abstract** class MyAbstractClass **{** **abstract** string MyMethod**();** **}**

class MyClass **:** MyAbstractClass **{** **public** **override** string MyMethod**()** **=>** "MyClass"**;** **}**

**Название класса-родителя идет перед названием интерфейсов:**

class MyClass **:** MyBaseClass**,** MyInterface1**,** MyInterface2**...**

**1. Если в классе-родителе и в расширяемом интерфейсе одинаковые методы, то его реализация в родителе переопределяет метод из интерфейса. В дочернем классе тогда его можно не переопределять.**

**interface** MyInterface **{** string MyMethod**();** **}**

class MyBaseClass **{** **public** string MyMethod**();** **}**

class MyClass **:** MyBaseClass**,** MyInterface **{** **}**

**2. Либо переопределить (override), если в родительском классе этот метод virtual (или abstract).**

**interface** MyInterface **{** string MyMethod**();** **}**

class MyBaseClass **:** MyInterface

**{**

**public** **virtual** string MyMethod**()** **=>** "MyBaseClass"**;**

**}**

class MyClass **:** MyBaseClass

**{**

**public** **override** string MyMethod**()** **=>** "MyClass"**;**

**}**

MyBaseClass myBaseClass **=** **new** MyClass**();**

string result **=** myBaseClass**.**MyMethod**();** // MyClass - т.к. есть переопределенный метод

MyInterface myInterface **=** **new** MyClass**();**

result **=** myInterface**.**MyMethod**();** // MyClass

**3. Либо в дочернем классе скрыть метод**

**interface** MyInterface **{** string MyMethod**();** **}**

class MyBaseClass **:** MyInterface

**{**

**public** string MyMethod**()** **=>** "MyBaseClass"**;**

**}**

class MyClass **:** MyBaseClass // класс наследуется от класса-родителя

**{**

**public** **new** string MyMethod**()** **=>** "MyClass"**;**

**}**

MyBaseClass myBaseClass **=** **new** MyClass**();**

string result **=** myBaseClass**.**MyMethod**();** // MyBaseClass - т.к. нет переопределенного метода

MyInterface myInterface **=** **new** MyClass**();**

result **=** myInterface**.**MyMethod**();** // MyBaseClass - т.к. интерфейс реализован только в базовом классе

MyClass myClass **=** **new** MyClass**();**

result **=** myClass**.**MyMethod**();** // MyClass

**4. Либо повторно реализовать интерфейс в классе-наследнике.**

**interface** MyInterface **{** string MyMethod**();** **}**

class MyBaseClass **:** MyInterface

**{**

**public** string MyMethod**()** **=>** "MyBaseClass"**;**

**}**

class MyClass **:** MyBaseClass**,** MyInterface // класс также напрямую наследуется от интерфейса

**{**

**public** **new** string MyMethod**()** **=>** "MyClass"**;**

**}**

MyBaseClass myBaseClass **=** **new** MyClass**();**

string result **=** myBaseClass**.**MyMethod**();** // MyBaseClass - т.к. для BaseClass нет переопределенного метода

MyInterface myInterface **=** **new** MyClass**();**

result **=** myInterface**.**MyMethod**();** // MyClass - т.к. получается, что интерфейс реализован в классе-наследнике

MyClass myClass **=** **new** MyClass**();**

result **=** myClass**.**MyMethod**();** // MyClass

**5. Либо явно реализовать интерфейс**

**interface** MyInterface **{** string MyMethod**();** **}**

class MyBaseClass **:** MyInterface

**{**

**public** string MyMethod**()** **=>** "MyBaseClass"**;**

**}**

class MyClass **:** MyBaseClass**,** MyInterface // класс также напрямую наследуется от интерфейса

**{**

**public** **new** string MyMethod**()** **=>** "MyClass"**;**

// явная реализация интерфейса

string MyInterface**.**MyMethod**()** **=>** "MyInterface"**;**

**}**

MyBaseClass myBaseClass **=** **new** MyClass**();**

string result **=** myBaseClass**.**MyMethod**();** // MyBaseClass - т.к. для BaseClass нет переопределенного метода

MyInterface myInterface **=** **new** MyClass**();**

result **=** myInterface**.**MyMethod**();** // MyInterface - т.к. интерфейс реализован в классе-наследнике

MyClass myClass **=** **new** MyClass**();**

result **=** myClass**.**MyMethod**();** // MyClass

# Наследование интерфейсов

**В отличие от классов к интерфейсам мы не можем применять модификатор sealed, чтобы запретить наследование интерфейсов. Но интерфейсы могут ипользовать new для сокрытия методов.**

**interface** MyInterface1 **{** string MyMethod**()** **=>** "MyInterface1"**;** **}**

**interface** MyInterface2 **{** **new** string MyMethod**)()** **=>** "MyInterface2"**;** **}**

class MyClass **:** MyInterface2 **{}**

MyInterface1 myInterface1 **=** **new** MyClass**();**

string result **=** myInterface1**.**MyMethod**();** // MyInterface1

MyInterface2 myInterface2 **=** **new** MyClass**();**

result **=** myInterface2**.**MyMethod**();** // MyInterface2

**При наследовании интерфейсов, как и классов, дочерний интерфейс должен иметь тот же уровень доступа или более строгий. Т.к., если он будет более открытый, то он откроет и методы и поля своего родительского класса / интерфейса.**

**public** **interface** MyInterface1 **{}**

**internal** **interface** MyInterface2 **:** MyInterface1 **{}**

# Интерфейсы как ограничения обобщений

**interface** MyInterface1 **{}**

**interface** MyInterface2 **{}**

class MyClass1 **:** MyInterface1**,** MyInterface2 **{}**

// ниже параметр типа Т должен реализовывать сразу оба интерфейса, как MyClass1

class MyClass2**<**T**>** **where** T**:** MyInterface1**,** MyInterface2

**{**

void MyMethod**(**T myInterface**)** **{** **...** **}**

**}**

MyClass1 myClass1 **=** **new** MyClass1**();**

MyClass2**<**MyClass1**>** myClass2 **=** **new** MyClass2**<**MyClass1**>();**

myClass2**.**MyMethod**(**myClass1**);**

## Обобщенные интерфейсы

**interface** MyInterface**<**T**>** **{** void MyMethod**(**T attribute**);** **}**

class MyClass1**<**T**>** **:** MyInterface**<**T**>** **{** **public** void MyMethod**(**T attribute**)** **{** **...** **}** **}**

// можно явно указать, какой будет тип параметра Т

class MyClass2 **:** MyInterface**<**int**>** **{** **public** void MyMethod**(**int attribute**)** **{** **...** **}** **}**

# Копирование объектов. Интерфейс ICloneable

**Реализация интерфейса ICloneable позволяет клонировать объекты.**

## Поверхностное копирование

class MyClass **:** ICloneable

**{**

**public** int Id **{** **get;** **set** **}**

**public** MyClass**(**int id**)** **=>** Id **=** id**;**

**public** **object** Clone**()** // реализация метода интерфейса ICloneable

**{**

**return** **new** MyClass**(**Id**);**

// MemberwiseClone(); - либо так - это поверхностное копирование

// если в классе есть ссылочные поля, то скопируются ссылки этих полей

**}**

**}**

MyClass myClass **=** **new** MyClass**(**1**);** // id = 1

MyClass myClass2 **=** myClass**.**Clone**();** // создание клона с id = 1

## Глубокое копирование

class Person **:** ICloneable

**{**

**public** Company MyCompany **{** **get;** **set;** **}**

**public** Person**(**Company company**)** **=>** MyCompany **=** company**;**

**public** **object** Clone**()** **=>** **new** Person**(new** Company**());**

**}**

class Company **{}**

# Сортировка объектов. Интерфейс IComparable

**Для сортировки наборов сложных объектов применяется интерфейс IComparable. Он имеет всего один метод:**

**public** **interface** IComparable

**{**

int CompareTo**(object?** o**);**

**}**

**Результат метода:**

**- 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;**

class Person **:** IComparable

**{**

**public** int Age **{** **get;** **set** **}**

**public** int CompareTo**(object?** o**)**

**{**

**if** **(**o **is** Person person**)**

**return** Age**.**CompareTo**(**person**.**Age**);**

// или return Age - person.Age;

**else** **throw** **new** ArgumentException**();**

**}**

**}**

**Интерфейс IComparable имеет обобщенную версию:**

class Person **:** IComparable**<**Person**>**

**{**

**...**

**public** int CompareTo**(**Person**?** person**)** **{** **...** **}**

**}**

## Применение компаратора

**public** **interface** IComparer**<in** T**>**

**{**

int Compare**(**T**?** x**,** T**?** y**);**

**}**

**Компаратор в качестве второго параметра принимает метод Array.Sort(Array array, IComparer comparer).**

class MyComparer **:** IComparer**<**Person**>**

**{**

**public** int Compare**(**Person**?** p1**,** Person**?** p2**)** **{** **...** **}**

**}**

Person**[]** people **=** **new** Person**[...];**

Array**.**Sort**(**people**,** **new** MyComparer**);**