

# ИНТЕРФЕЙСЫ

**Понятие интерфейса.**

**Синтаксис объявления интерфейсов.**

**Примеры создания интерфейсов.**

**Интерфейсные ссылки.**

**Интерфейсные индексы, свойства.**

**Наследование интерфейсов.**

**Проблемы сокрытия имен при наследовании интерфейсов.**

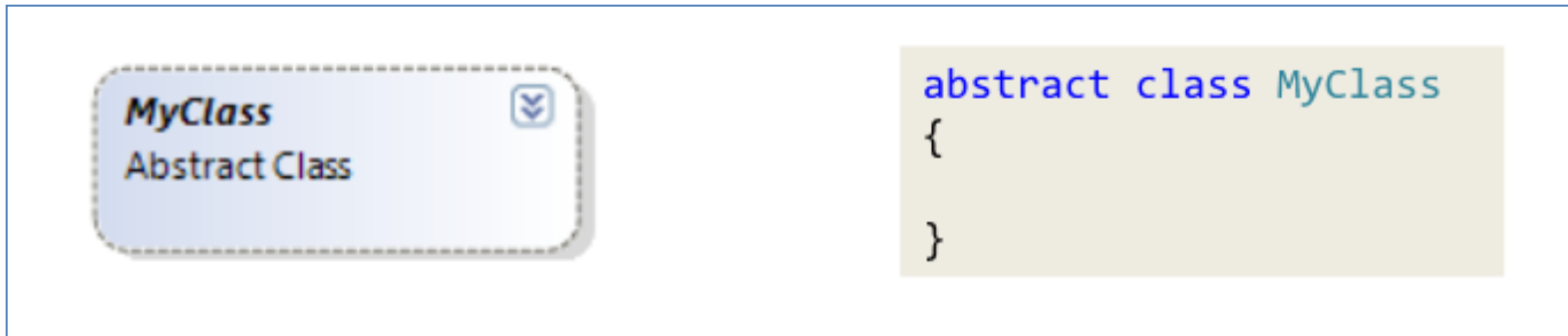
**Анализ стандартных интерфейсов.**

## Понятие абстракции

**Абстракция** в ООП – это придание объекту характеристик, которые отличают его от всех других объектов, четко определяя его концептуальные границы.

**Абстрагирование** в ООП – это способ выделить набор значимых характеристик объекта, исключая из рассмотрения незначимые.

**Абстрактный класс** в ООП - это **базовый** класс, который не предполагает создания экземпляров через вызов конструктора напрямую, но экземпляр абстрактного класса создается неявно при построении экземпляра производного конкретного класса.



Ключевое слово **abstract** может использоваться с **классами, методами, свойствами, индексами и событиями**

## Возможности и ограничения абстрактных классов:

- Экземпляр абстрактного класса **создать нельзя через вызов конструктора** напрямую, но экземпляр абстрактного класса создается неявно **при построении экземпляра производного конкретного класса.**
- Абстрактные классы **могут содержать как абстрактные, так и неабстрактные члены.**
- Неабстрактный (конкретный) класс, являющийся производным от абстрактного, **должен содержать фактические реализации всех наследуемых абстрактных членов.**

# Возможности абстрактных методов:

- Абстрактный метод является **неявным виртуальным методом**.
- Создание **абстрактных методов** допускается только в абстрактных классах
- Тело абстрактного метода отсутствует; создание метода просто заканчивается **двоеточием**, а после сигнатуры ставить фигурные скобки ({ }) не нужно
- Реализация предоставляется **методом** **переопределения `override`**, который является членом неабстрактного класса.

**В мире объектно-ориентированного программирования уже достаточно давно подвергается критике концепция наследования.**

### **Аргументов достаточно много:**

- дочерний класс наследует все данные и поведение родительского, что нужно не всегда (а при доработке родительского в дочерний класс вообще попадают данные и поведение, которые не предполагались на момент разработки дочернего);
- виртуальные методы менее производительные, а в случае, если язык позволяет объявить неvirtуальный метод, то как быть, если в наследнике нужно его перекрыть (можно пометить метод словом new, но при этом не будет работать полиморфизм, и использование такого объекта может привести к неожиданному поведению, в зависимости от того, к какому типу приведен объект в момент его использования);
- если возникает необходимость множественного наследования, то в большинстве языков оно отсутствует, а там, где есть (C++), считается трудоемким;
- есть задачи, где наследование как таковое не может помочь — если нужен контейнер элементов (множество, массив, список) с единым поведением для элементов разных типов, и при этом нужно обеспечить статическую типизацию, то здесь помогут обобщения (generics).

## Понятие интерфейса.

***Интерфейс (interface)*** представляет собой не более чем просто именованный набор абстрактных членов.



## Интерфейс определяет ряд методов для реализации в классе.

- **НЕВОЗМОЖНО** создать экземпляр интерфейса
- интерфейс может содержать **ТОЛЬКО абстрактные члены** (методы, свойства, события или индексаторы).
- в интерфейсе ни у одного из методов не должно быть тела (реализации).
- интерфейс может быть реализован **в любом количестве классов**.
- в одном классе может быть реализовано **любое количество интерфейсов**.

# Синтаксис объявления интерфейсов.

```
interface Имя
{
    возвращаемый_тип имя_метода_1 (список_параметров);
    возвращаемый_тип имя_метода_2 (список_параметров);
    // ...
    возвращаемый_тип имя_метода_N (список_параметров);
}
```

**В интерфейсах все члены «по умолчанию» с модификатором доступа **public**.**

- В интерфейсах можно указывать **методы, свойства, индексаторы, события**.
- Интерфейсы не могут содержать **поля**.
- В интерфейсах нельзя определять **конструкторы, деструкторы или операторные методы**.
- В интерфейсе не должно быть **static-членов**.
- Реализовать интерфейс выборочно или по частям нельзя.

```
interface IWork // создание интерфейса
{
    void Working(int delta); // доступ по умолчанию public
                               // отсутствует реализация методов
}
```

```
class Truck : IWork // наследование интерфейса IWork
{
    // обязательная реализация методов из интерфейса IWork
    public void Working(int delta)
    {
        // тело метода
    }
}
```

```
class Bus : IWork // наследование интерфейса IWork
{
    public void Working(int delta)
    {
        // тело метода
    }
}
```

# Интерфейсные ссылки.

```
IWork Worker; // объявление интерфейсной ссылки!  
Truck truck = new Truck();  
Worker = truck; // присваивание объекта интерфейсной ссылке !  
Worker.Working(2);  
  
Bus bus = new Bus();  
Worker = bus;  
Worker.Working(120);  
  
// создание массива интерфейсных ссылок!  
IWork[] vehicles = new IWork[2];  
    vehicles[0] = new Truck();  
    vehicles[1] = new Bus();  
  
    foreach (IWork worker in vehicles)  
    {  
        worker.Working(24);  
    }
```

# Интерфейсные свойства.

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

- в определении **интерфейсных свойств**, доступных только для чтения или только для записи, должен присутствовать единственный аксессор: get или set соответственно.
- при объявлении свойства в интерфейсе **не разрешается указывать модификаторы доступа** для аксессоров.

# Интерфейсные индексаторы

```
// Интерфейсный индексатор  
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.

```
interface IWork      // создание интерфейса
{
    void Working(int delta);
    // интерфейсное свойство (только для чтения)
    string Status { get; }
    // интерфейсный индексатор (только для чтения)
    string this[int index] { get; }
}
```

```
class Bus : IWork // наследование интерфейса Iwork
{
    public void Working(int delta)
    {
        // тело метода
    }

    public string Status
    {
        get
        {
            // тело асессора get
        }
    }

    public string this[int index]
    {
        get
        {
            // тело асессора get
        }
    }
}
```



# Наследование интерфейсов.

- один интерфейс может наследовать другой.
- интерфейс может наследоваться от нескольких интерфейсов.
- **синтаксис наследования** интерфейсов такой же, как и у классов.
- при наследовании методы базового интерфейса в производном интерфейсе **не реализуются**.
- когда в классе реализуется один интерфейс, наследующий другой интерфейс, в классе **должны быть реализованы все члены**, определенные **в цепочке наследования интерфейсов**

```
interface IStudy: ILive, IWork, IComparable
{
    void Study();
}
```

```
interface IWork : IBasic           // наследование интерфейса IBasic
{
    void Working();
}
```

```
interface IBasic                   // базовый интерфейс
{
    void Drive(double V);
    void Update();
}
```

```
// наследование класса Car и интерфейсов IWork ILoading
class Truck : Car, IWork, ILoading
{
}
}
```

## Проблемы сокрытия имен при наследовании интерфейсов.

- Когда один интерфейс наследует другой, то в производном интерфейсе **может быть объявлен член**, скрывающий член **с аналогичным именем в базовом интерфейсе**.
- сокрытие имен происходит в том случае, если член в производном интерфейсе **объявляется таким же образом, как и в базовом интерфейсе**.
- если не указать в объявлении члена производного интерфейса ключевое **слово new**, то компилятор выдаст соответствующее **предупреждающее сообщение**

# Соккрытие имен при наследовании интерфейсов.

```
interface IWork : IBasic           // наследование интерфейса IBasic
{
    // МЕТОД С ТАКИМ ИМЕНЕМ ИМЕЕТСЯ В ИНТЕФЕЙСЕ Ibasic
    // т.е. скрывает метод Working из IBasic
    new void Working();
}

interface IBasic                   // базовый интерфейс
{
    void Drive(double V);
    void Working();
}
```

# Явная реализация интерфейсов.

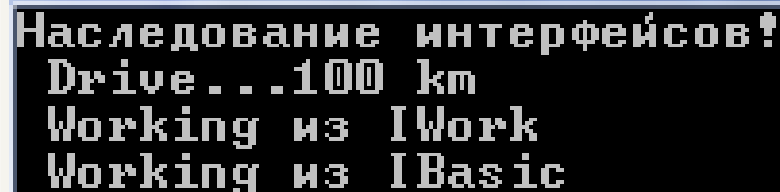
```
class Truck : IWork // наследование интерфейса IWork
{
//явная реализация интерфейса IWork (модификатор доступа private)
    void IWork.Working()
    { Console.WriteLine(" Working из IWork"); }

//явная реализация интерфейса IBasic (модификатор доступа private)
    void IBasic.Working()
    { Console.WriteLine(" Working из IBasic"); }

    public void Drive(double V)
    { Console.WriteLine(" Drive...{0} km",V); }
}
```

```
Truck truck = new Truck();
truck.Drive(100);
// truck.Working(); // ошибка
IWork work = truck;
work.Working();
```

```
IBasic basic = truck;
basic.Working();
```



```
Наследование интерфейсов!
Drive...100 km
Working из IWork
Working из IBasic
```

## Для явной реализации интерфейсного метода могут быть две причины:

- интерфейсный метод реализуется **с указанием его полного имени**. Данный метод оказывается доступным не посредством **объектов класса**, реализующего данный интерфейс, а по **интерфейсной ссылке**.
- в одном классе **могут быть реализованы два интерфейса с методами**, объявленными с **одинаковыми именами и сигнатурами**.

# Анализ стандартных интерфейсов.

```
class Car : IComparable // наследование от интерфейса IComparable
{
    public int ID { get; set; }
    public string Name { get; set; }
    public double Speed { get; set; }
    public double Weight { get; set; }

    public override string ToString()
    {
        return String.Format("№{0}. {1} V={2}, M={3}", ID, Name, Speed, Weight);
    }

    // реализация метода CompareTo из интерфейса IComparable
    public int CompareTo(object obj)
    {
        Car NextCar = (Car)obj;
        if (NextCar.Speed > this.Speed) return -1;
        else return 1;
    }
}
```

```
Car[] masCars = new Car[];
Array.Sort(masCars); // СОРТИРОВКА МАССИВА, ЭЛЕМЕНТАМИ КОТОРОГО
                     // ЯВЛЯЮТСЯ ОБЪЕКТЫ КЛАССА Car
```

# Анализ стандартных интерфейсов.

```
class Car
{
    public string Name { get; set; }
    public int Speed { get; set; }
    public override string ToString()
    {
        return Name;
    }
}
```

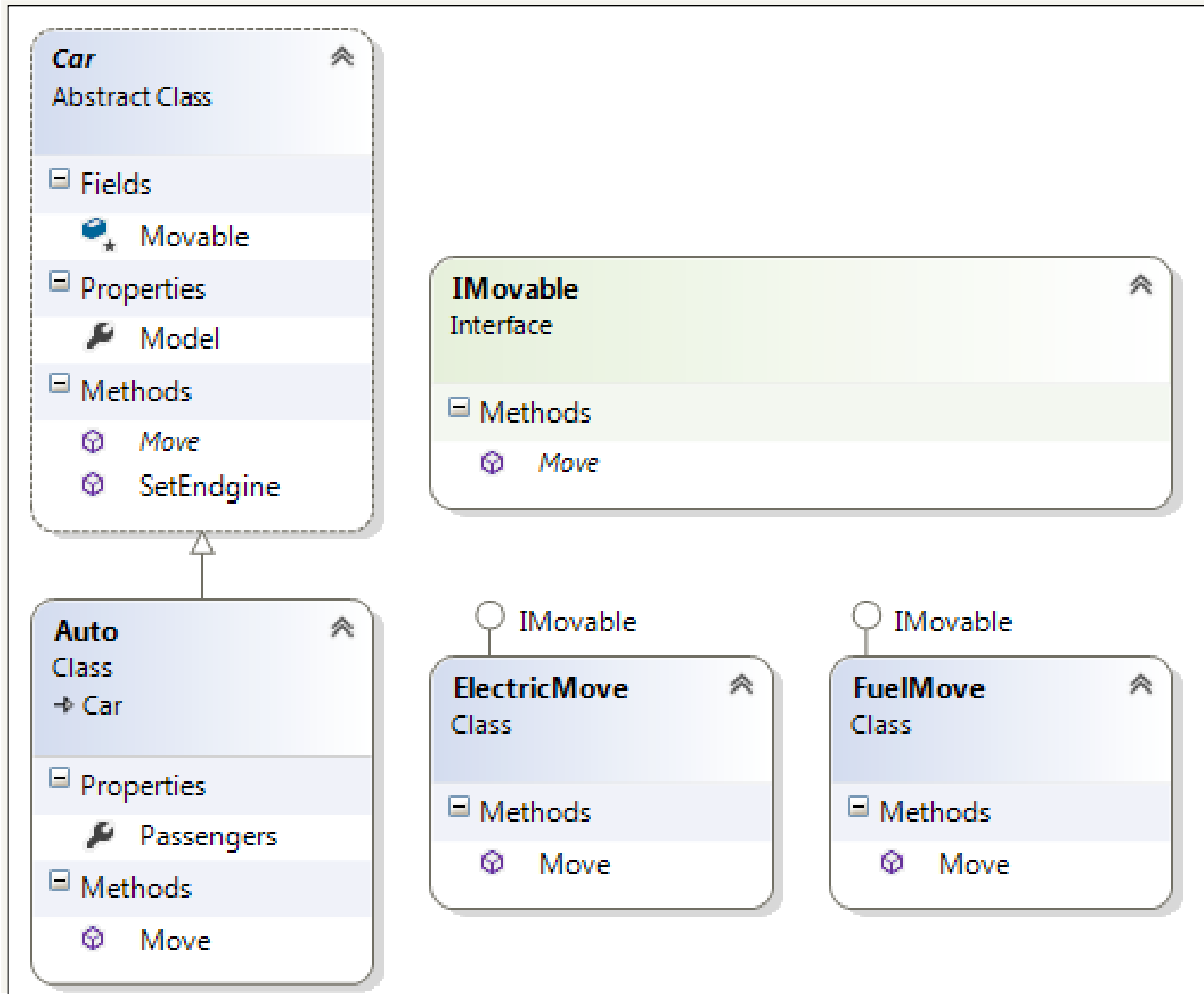
```
public class CarsComparer : IComparer
{
    public int Compare(object x, object y)
    {
        return ((Car)x).Speed.CompareTo(((Car)y).Speed);
    }
}
```

```
Array.Sort(cars, new CarsComparer());
```

```
public static void Sort(Array array, IComparer comparer);
```



# Использование интерфейсов.



# Использование интерфейсов.

```
interface IMovable
{
    void Move();
}

// Алгоритмы
class FuelMove : IMovable
{
    public void Move()
    {
        Console.WriteLine("Перемещение на бензине");
    }
}

class ElectricMove : IMovable
{
    public void Move()
    {
        Console.WriteLine("Перемещение на электричестве");
    }
}
```

# Использование интерфейсов.

```
abstract class Car
{
    public string Model { get; set; } // модель автомобиля
    protected IMovable Movable;
    public void SetEngine(IMovable movable)
    {
        Movable = movable;
    }
    public abstract void Move();
}

class Auto : Car
{
    public int Passengers { get; set; } // кол-во пассажиров
    public override void Move()
    {
        Movable.Move();
    }
}
```

# Использование интерфейсов.

```
class Program
{
    static void Main(string[] args)
    {
        Car car = new Auto() { Model = "BMW" };
        car.SetEngine(new FuelMove());
        car.Move();

        car.SetEngine(new ElectricMove());
        car.Move();

        Console.ReadKey();
    }
}
```

# Выбор между интерфейсом и абстрактным классом

- если какое-то понятие можно описать с точки зрения **функционального** назначения, не уточняя конкретные детали реализации, то следует использовать интерфейс.
- если требуются некоторые детали **реализации**, то данное понятие следует представить абстрактным классом.

## Преимущество использования интерфейсов:

- Класс или структура может реализовать **несколько интерфейсов** (допустимо множественное наследование от интерфейсов).
- Если класс или структура реализует интерфейс, она получает **только имена и сигнатуры метода**.
- Интерфейсы **определяют поведение экземпляров производных классов**.
- Базовый класс может обладать **ненужным функционалом**, полученным от других его базовых классов, чего **можно избежать, применяя интерфейсы**.