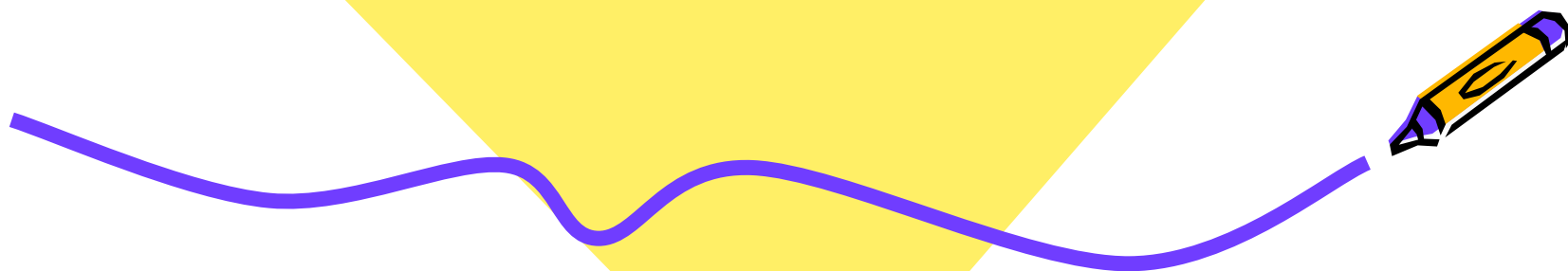




# Лекция 5. Интерфейси и абстрактни класове.



# Интерфейси

- Интерфейсите описват функционалност, която се поддържа от множество обекти. Подобно на класовете и структурите, интерфейсите също се състоят от членове, но се различават от тях по това, че дефинират само прототипите на членовете си, без конкретната им реализация.
- Членовете могат да бъдат: **методи, свойства, индексатори и събития**. В интерфейс **не могат да се дефинират конструктори, деструктори, полета, константи и вложени типове и не могат да се предефинират оператори**.
- Към членовете на интерфейс не може да се прилагат модификатори на достъпа - по подразбиране всички членове са с абстрактни, с глобална видимост (**public abstract по подразбиране**).
- От интерфейсите **не могат да се създават обекти чрез инстанциране**.



# Интерфейси

- Интерфейс може да наследи един или повече други интерфейса, като е възможно да предефинира или скрива техните членове.
- Интерфейсите се наследяват и **реализират** от класове или структури, които трябва да имплементират всички дефинирани в тях членове. Един клас или структура може да имплементира повече от един интерфейс.
- Конкретните имплементации на даден интерфейс вече могат да се инстанцират и да се присвояват на променливи от тип интерфейс.
- Препоръчително е имената на интерфейсите да започват с 'I', като например IDisposable, ISerializable, ICloneable, IEnumerator и др.



# Вариант 1. Множествено наследяване.

using System;

interface IFoo

{ void DoSomething1();}

interface IBar

{ void DoSomething2();}

class MyObject : IFoo, IBar

{ public void DoSomething1() //!!!public

{ Console.WriteLine("This is IFoo Interface"); }

public void DoSomething2()

{ Console.WriteLine("This is IBar Interface"); }

}

class Program

{ static void Main()

{ MyObject m = new MyObject();

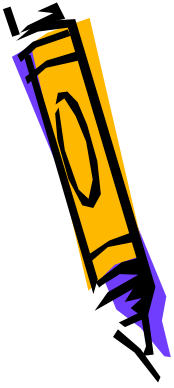
m.DoSomething1();

m.DoSomething2();

}

} //Резултат: *This is IFoo Interface*

*This is IBar Interface*

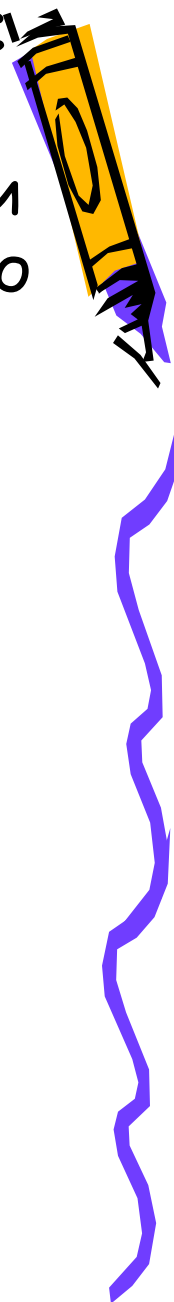


# Реализиране и използване на интерфейси

- Коментар:

Виждаме, че class MyObject наследява интерфейси IFoo, IBar (**Multiple Inheritance**) във вариант 1, като имплементира конкретните им методи DoSomething1(), DoSomething2().

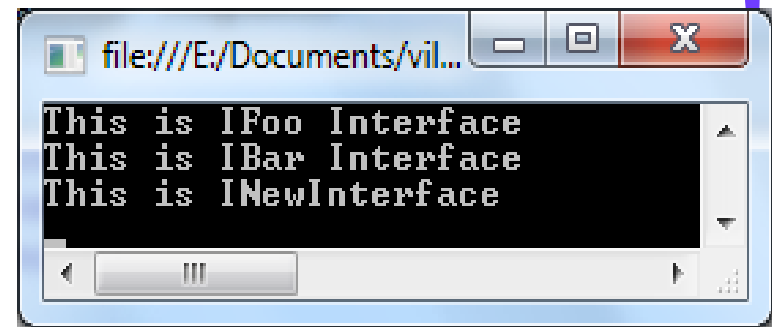
Виждаме, че class MyObject наследява един интерфейс INewInterface, който от своя страна наследява интерфейси IFoo, IBar (**Multiple Inheritance**) във вариант 2, като имплементира конкретните им методи DoSomething1(), DoSomething2().



## Вариант 2. Даден интерфейс, наследява други интерфейси

```
using System;
interface IFoo { void DoSomething1();}
interface IBar { void DoSomething2();}
interface INewInterface : IFoo, IBar { void DoSomething3();}
class MyObject : INewInterface
{
    public void DoSomething1() //!!!public
    { Console.WriteLine("This is IFoo Interface"); }
    public void DoSomething2()
    { Console.WriteLine("This is IBar Interface"); }
    public void DoSomething3()
    { Console.WriteLine("This is INewInterface"); }
}

class Program
{
    static void Main()
    {
        MyObject m = new MyObject();
        m.DoSomething1();
        m.DoSomething2();
        m.DoSomething3();
    }
}
```



## Вариант 3: еднакви имена на методи

using System;

**interface** IFoo { void DoSomething1();}

**interface** IBar { void DoSomething1();}

**class** MyObject : IFoo, IBar

```
{    void IFoo.DoSomething1()    //!!! без public
    { Console.WriteLine("This is IFoo Interface"); }
    void IBar.DoSomething1() //без public
    { Console.WriteLine("This is IBar Interface"); }
}
```

**class** Program

```
{    static void Main()
```

```
{    MyObject m = new MyObject();
```

IFoo foo = (IFoo)m; //Използването е чрез ИНТЕРФЕЙСИТЕ, но няма new

IBar bar = (IBar)m;

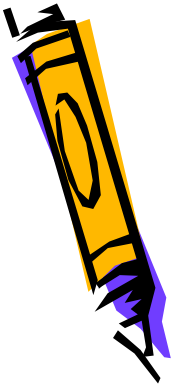
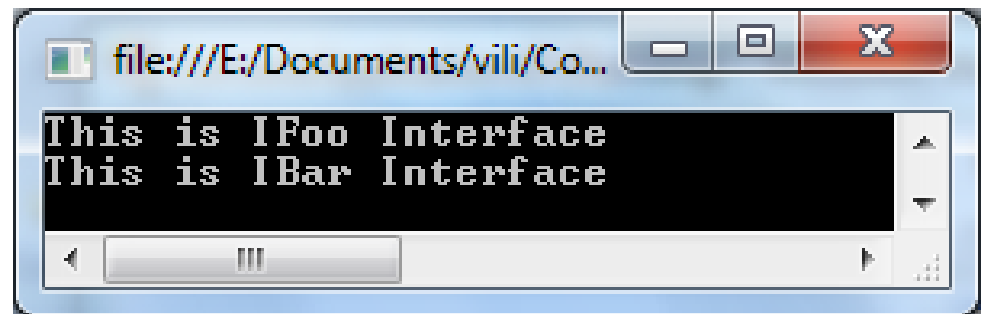
**foo.DoSomething1();**

**bar.DoSomething1();**

Console.ReadLine();

}

}



Трябва също да обърнем внимание на начина по който се използва интерфейса в 3-тия случай. Ние не можем да използваме еднаквите методи на интерфейсите по друг начин, освен показания, а именно:

1. Създаваме инстанция на класа.
2. Дефинираме обект от всеки тип интерфейс = на инстанцията на класа.
3. Извикваме метода, чрез обекта от тип интерфейс.





# Абстрактни класове

- Абстрактните класове приличат на интерфейсите по това, че те не могат да се инстанцират, защото могат да съдържат дефиниции на неимплементирани методи, но за разлика от интерфейсите могат да съдържат и описани действия. Те са комбинация между клас и интерфейс – представляват частично имплементирани класове. Имат имплементация за някои от методите си, а други методи са обявени като абстрактни, без имплементация.
- Абстрактният клас предоставя само сигнатура или декларация на абстрактните си методи, оставяйки имплементацията им на породените или вложени класове.
- Абстрактните методи и класове се обявяват с ключовата дума `abstract`.
- **Тъй като абстрактният клас е непълен, той не може да се инстанцира.** Той трябва да бъде наследен за да може да се използва дефинираната в него функционалност. **Ето защо, един абстрактен клас не може да бъде обявен като „sealed“.**
- Класът, наследяващ абстрактен клас трябва да имплементира всички абстрактни методи на базовия клас, в противен случай – също трябва да се обяви като абстрактен.
- Клас, който наследява даден абстрактен клас, имплементирайки всички негови абстрактни методи се нарича конкретен клас за дадения абстрактния клас.

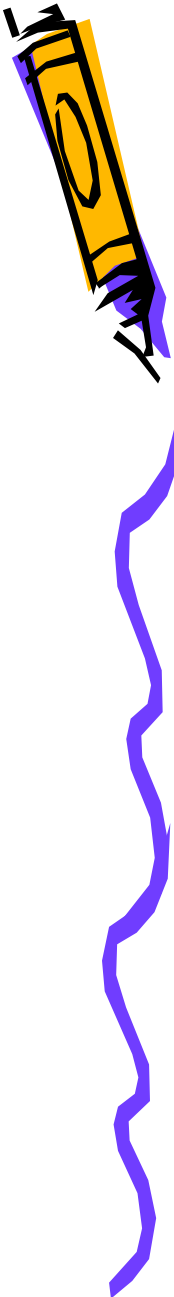


# Абстрактни класове - пример

```
public abstract class Car
{
    public void Move()
    {
        // Move the car
    }

    abstract public int TopSpeed
    {
        // Retrieve the top speed in Kmph
        get;
    }

    public abstract string BrandName //марка
    {
        get;
    }
}
```



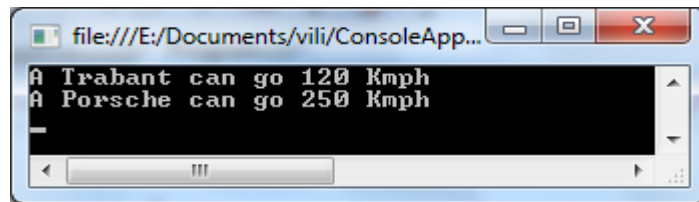
# Абстрактни членове

- Ключовата дума **abstract** в декларацията на класа го определя като абстрактен. Виждаме, че тя може да се приложи и към член. Абстрактни могат да бъдат методите, свойствата, индексаторите и събитията.
- !!! Абстрактните членове не могат да имат имплементация, както и член, който не е абстрактен, не може да бъде оставен без такава.
- Ако в един клас е дефиниран абстрактен член, класът задължително трябва да бъде обявен за абстрактен. В противен случай получаваме грешка при компилация. Обратното не е задължително, тоест допустимо е да имаме абстрактен клас, на който всички членове са дефинирани.

## Наследяване на абстрактни класове

- Тъй като абстрактните класове са класове, те имат същата структура - същият набор от членове (полета, константи, вложени типове и т. н.), същите модификатори на видимостта и дори същите механизми за наследяване, но с някои особености. Нека разширим предходния пример:





```
file:///E:/Documents/vili/ConsoleApp...
A Trabant can go 120 Kmph
A Porsche can go 250 Kmph
```

```
using System;
public abstract class Car
{
    public void Move()
    { // Move the car }
    abstract public int TopSpeed
    {
        // Retrieve the top speed in Kmph
        get;
    }
    public abstract string BrandName
    { get; }
}
public class Trabant : Car
{
    public override int TopSpeed
    {
        get
        { return 120; } }
}
```

```
        public override string
        BrandName
        {
            get
            { return "Trabant"; }
        }
    }
    public class Porsche : Car
    {
        public override int TopSpeed
        {
            get { return 250; }
        }

        public override string
        BrandName
        {
            get
            { return "Porsche"; }
        }
    }
}
```

```
public class AbstractTest
{
    static void Main()
    {
        Car[] cars = new Car[] {new Trabant(), new Porsche() };
        foreach (Car car in cars)
        {
            Console.WriteLine("A {0} can go {1} Kmph",
                               car.BrandName, car.TopSpeed); } } }
```

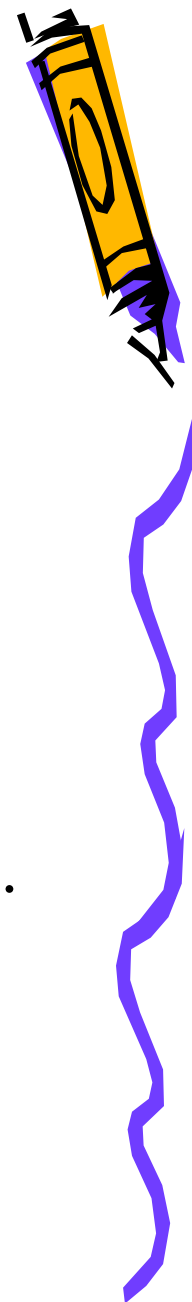


- Виждаме, че въпреки че абстрактният клас не може да се инстанциира директно, обектите от наследяващите го класове могат да се разглеждат като обекти от неговия тип. По показания начин можем да използваме абстрактни базови класове, за да задействаме полиморфизъм, или, казано по-общо, да създадем абстрактен корен, на дърво от класове.
- В примера ползвахме ключовата дума **override**, с която указваме, че даден метод в класа наследник припокрива (замества) оригиналния наследен метод от базовия си клас. В случая базовия клас не предоставя имплементация за припокриваните методи, така че припокриването е задължително. Ще разгледаме ключовата дума **override** и нейното действие след малко. Нека сега продължим с абстрактните класове.



## Частично реализиране на абстрактните членове

- Възможно е абстрактен клас, съдържащ абстрактни членове, да бъде наследен, без всичките му абстрактни членове да бъдат реализирани. Възможно е също клас, който имплементира абстрактните членове на абстрактния си родител, да дефинира допълнително и свои членове, също абстрактни. В този случай класът-наследник също трябва да бъде деклариран като абстрактен, защото съдържа абстрактни членове.
- Тези възможности правят още по-гъвкав инструментариума за създаване на йерархии от класове и моделиране на реалния свят. Ще илюстрираме тази възможност със следното разширение на предходния пример:

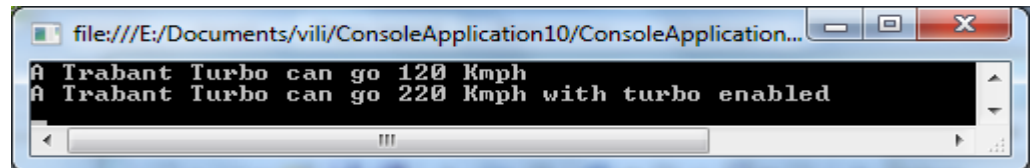


```
using System;
public abstract class Car
{
    public void Move()
        { // Move the car }
    abstract public int TopSpeed
        {
    // Retrieve the top speed in Km/h
        get;    }
    public abstract string BrandName
        {
        get;    }
    }
    abstract public class TurboCar: Car
    {
        protected Boolean mTurboEnabled =
        false;
```

```
        public void EnableTurbo()
            { mTurboEnabled = true; }
        public void DisableTurbo()
            { mTurboEnabled = false; }
    }
    public class TrabantTurbo: TurboCar
    {
        override public int TopSpeed
        {
            get
            { return mTurboEnabled ? 220 : 120; }
        }
        override public string BrandName
        {
            get
            { return "Trabant Turbo"; }
        }
    }
}
```

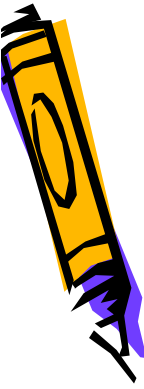
```
public class AbstractTest
{
    static void Main()
    {
        TurboCar turboCar = new TrabantTurbo();
        Console.WriteLine("A {0} can go {1} Km/h", turboCar.BrandName,
            turboCar.TopSpeed);
        turboCar.EnableTurbo(); //след включване на Turbo-то

        Console.WriteLine("A {0} can go {1} Km/h with turbo enabled",
            turboCar.BrandName, turboCar.TopSpeed);
    }
}
```



The screenshot shows a console window titled "file:///E:/Documents/vili/ConsoleApplication10/ConsoleApplication...". The output text is as follows:

```
A Trabant Turbo can go 120 Km/h
A Trabant Turbo can go 220 Km/h with turbo enabled
```



Създадохме класа **TrabantTurbo**, който реализира абстрактните свойства, индиректно наследени от класа **TurboCar**.

Класът **TurboCar** е разширение на класа **Car**, който също като него е абстрактен, но предоставя допълнителна функционалност за включване на режим "турбо".

**!!! Ако един клас наследи абстрактен клас, но не предостави дефиниции за всички негови абстрактни членове, той трябва задължително също да бъде обявен за абстрактен.**





- В дефинициите на членовете в горните примери забелязваме употребата на запазената дума **override**. Без нея те не биха могли да бъдат компилирани. Това е така, защото въпросните **абстрактни** членове са всъщност „**виртуални**“.
- В някои обектно-ориентирани езици за програмиране (например в C++) абстрактните членове се обявяват за виртуални.
- Ние знаем, че при полиморфизъм виртуални членове се дефинират, като в дефиницията им се укаже ключовата дума **virtual**. Трябва да се отбележи също, че всички абстрактни членове, включително и тези, дефинирани в интерфейсите (и те са абстрактни, тъй като нямат имплементация), са винаги виртуални.

