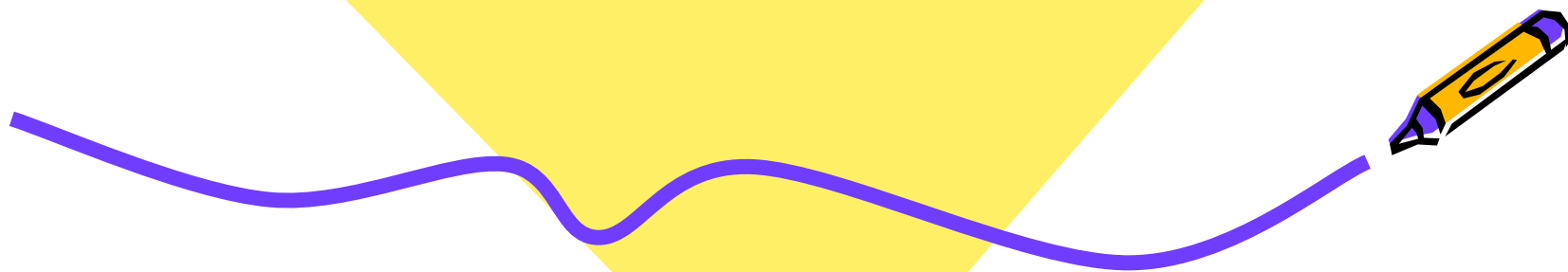


# Лекция 4. Методи.



# Методи

- Всички функции, описани в класа, се явяват методи (или още **член-функции**) на класа. Методите описват възможни операции, над обектите на типа (в частност на класа), в който са дефинирани, като по този начин определят и поведението на обектите на типа.
- В C# функции могат да бъдат дефинирани единствено като членове на тип (клас или структура), за разлика от други обектно-ориентирани езици, където е допустимо използване на **глобални функции** – такива, които не са обвързани с конкретен тип и са общодостъпни.
- В C# функции, които се достъпват без да е нужна инстанция на даден клас, се дефинират като **статични**. Статичните методи се викат през името на своя клас:



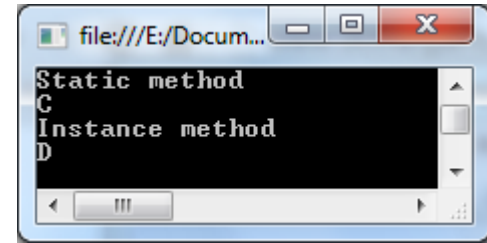
using System;

class Program

```
{ static char MethodC()
    { Console.WriteLine("Static method");
      return 'C';
    }
  char MethodD()
  { Console.WriteLine("Instance method");
    return 'D';
  }
```

static void Main()

```
{ //Call the static methods on the Program type.
  Console.WriteLine(Program.MethodC());
  // Create a new Program instance and call the instance methods.
  Program programInstance = new Program();
  Console.WriteLine(programInstance.MethodD());
  Console.ReadLine();
}
```



Общ синтаксис на описание на метод:

[спецификатори] <тип> <име> ([списък\_от\_параметри])

```
{  
    Тяло_на_метода;  
    [return стойност;]  
}
```

където:

- [спецификатори] - спецификатор `static` и модификаторите за достъп, които задават нивото на видимост на метода.
- <тип> - това е типа на връщаната от метода стойност - **задължителен** елемент от описанието на метода. Това може да е всеки валиден .NET тип, включително тип, създаден от програмиста. Ако метода не връща стойност, типа на връщания резултат е `void` (и в този случай в тялото на метода отсъства оператор `return` ).
- <име> - това е името на метода, уникален идентификатор в рамките на текущата област на видимост, зададен от програмиста с отчитане на изискванията за създаване на идентификаторите в C#.
- [списък\_от\_параметри] - Методите могат да приемат параметри, които се описват като последователност от разделени със запетая двойки: <тип> <идентификатор>. Типът на параметрите може да бъде всеки валиден .NET тип.



## Особености на предаване на параметри

- Както и в другите езици за програмиране, така и тук, параметрите се използват за обмен на информация между извикващия и извикания метод. Параметрите в C# могат да се разделят на четири типа: параметри-стойности, параметри-референции, изходни параметри и параметри-масиви.
- При предаване на параметри по стойност, а това е начина за предаване по подразбиране за всички стойностни типове, методът получава и работи с копие на действителния параметър. Следователно, действителният параметър не се променя.
- При предаване на параметър по референция, а това е начинът за предаване по подразбиране за референтните типове, методът получава копие на адреса на параметъра, което му позволява достъп и работа с действителния параметър.
- Ето защо, промяната на **my\_age** (стойностен тип) в **InternalMagic()** не се отразява в **main()**, докато промяната на **myName** и **myGender** - се отразява в **main()** - (обекта **p** е референтен тип ).



## Особености на предаване на параметри

using System;

public class Person

{ public string myName;

public string myGender;

public void InternalMagic(Person x, int age)

{ Person y=x;

y.myName="Ann";

y.myGender="F";

age = 22;

}

static void Main()

{ Person p=new Person();

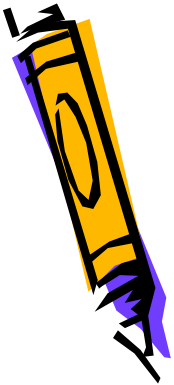
p.myName="Tom"; p.myGender="M"; int my\_age = 25;

Console.WriteLine(String.Format("{0} {1} {2}", p.myName,  
p.myGender, my\_age)); // Tom M 25

p.InternalMagic(p, my\_age);

Console.WriteLine(String.Format("{0} {1} {2}", p.myName,  
p.myGender, my\_age)); // Ann F 25

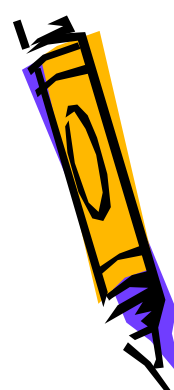
}



# Методи с еднакви имена

- в C# е допустимо един тип да има два и повече метода с едно и също име, но методите трябва да имат различна **сигнатура**.
- Комбинацията от името, броя и типа на параметрите на метод се нарича **сигнатура**.
- !!!Ако два метода имат едно и също име, то те задължително трябва да се различават по **сигнатура**, както това е и в следващия пример:



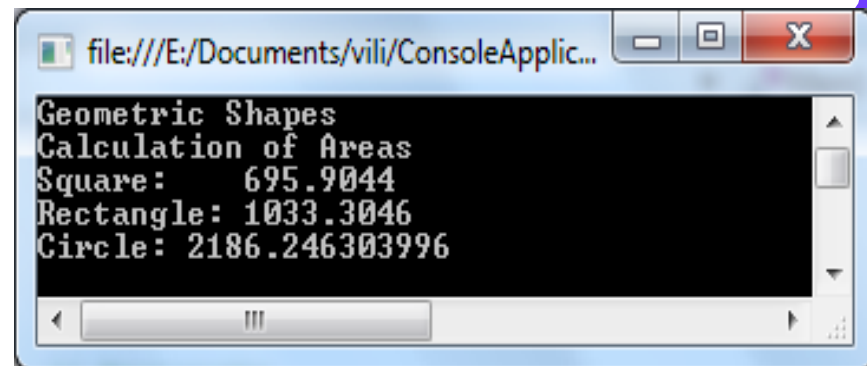


```
public class Geometry
{
    // Unknown Shape
    public double CalculateArea()
    {
        return 0;
    }

    // Square
    public double CalculateArea(double side)
    {
        return side * side;
    }


    // Rectangle
    public double CalculateArea(double l, double h)
    {
        return l * h;
    }

    /* Circle Този метод няма проблем да се
    компилира
    public float CalculateArea(float radius)
    {
        return radius * radius * 3.14159f;
    }
    */
    public double CalculateArea(double r, int unused)
    {
        //подаваме един не-използваем параметър
        return r * r * 3.14159;
    }
}
```



```
file:///E:/Documents/vili/ConsoleApplic...
Geometric Shapes
Calculation of Areas
Square: 695.9044
Rectangle: 1033.3046
Circle: 2186.246303996
```

```
public class Exercise
{
    static void Main()
    {
        var geo = new Geometry();
        System.Console.WriteLine("Geometric Shapes");
        System.Console.WriteLine("Calculation of Areas");
        System.Console.Write("Square: ");
        System.Console.WriteLine(geo.CalculateArea(26.38));
        System.Console.Write("Rectangle: ");
        System.Console.WriteLine(geo.CalculateArea(39.17,
        26.38));
        System.Console.Write("Circle: ");
        System.Console.WriteLine(geo.CalculateArea(26.38, 0));
    }
}
```





- Какво ще стане, ако добавим метод:

```
public double CalculateArea(int k, int h)
{ // some area
    return k * h;
}
```

...//Не е очевидно кой метод ще се извика при извикването:

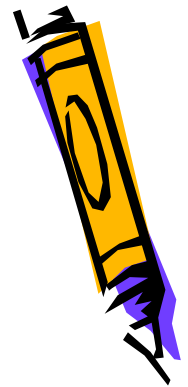
```
System.Console.Write("Rectangle: ");
System.Console.WriteLine(geo.CalculateArea(4, 3));
```

//Това трябва да се избягва!!!

//Резултат: 13



- Методите с еднакви имена са проява на полиморфизъм, едно от основните свойства на ООП. За един програмист е много по-лесно да запомни едно име на метод и да го използва за работа с различни типове данни, като решението коя точно версия на метода да се извика се възлага на компилатора.
- Този принцип се използва широко в библиотеката от класове на .Net. Например, в стандартен клас Console, методът WriteLine е преизползван многократно (19 пъти) за извеждане на променливи от различни типове.



# ключовите думи **ref** или **out**

- При предаване на стойностен тип променлива като параметър на метод, в стека се предава копие на стойността на променливата. За да се работи във метода с истинския обект може да се използват ключовите думи **ref** или **out**:
- Ключова дума **out** е подходяща за **изходни** параметри, тоест за параметри, чрез които ще връщаме стойност в извикващия метод.
- **Фактическите параметри могат да не бъдат инициализирани** преди предаването им, тъй като инициализацията се извършва на практика от извиквания метод.



# ключовите думи **ref** или **out**

- Ключова дума **ref** се използва за предаване по референция на входно-изходни параметри. Промените, които методът прави по подадените му по референция параметри, изменят истинските стойности на параметрите, а не техни копия от стека, и затова са видими от кода, извикал метода (както се вижда и от примера).
- **Фактическите параметри**, които се предават по този начин трябва задължително да бъдат инициализирани преди предаването им.



# КЛЮЧОВИТЕ думи ref или out

using System;

class Program

```
{ static void swap(ref int x, ref int y, out int z)
    {int t; t = x; x = y; y = t; z = x + y;}
```

static void Main()

```
{ int x = 5; int y = 67; int suma;
```

```
swap(ref x, ref y, out suma);
```

```
Console.WriteLine("x={0}, y={1}, suma={2}", x, y,
    suma);
```

```
} } //Резултат: X=67, y=5, suma=72
```



# Параметър **this** в екземплярни методи

- Като членове на класове, екземплярните методи се намират в паметта в един единствен екземпляр и се използват от всичките обекти на един клас. Всеки обект, автоматично предава на извиквания нестатичен метод един скрит параметър **this**, в който се съхранява референцията на обекта.
- В явен вид, параметър **this** се използва за това да върне от метода референцията на извикващият обект, а също така и за идентификацията на поле в случай, че неговото име съвпада с името на параметър на метода, например:



# Параметър this в екземплярни методи

using System;

class Circle

```
{ int x = 0; int y = 0; int radius = 3;
```

```
    public Circle T() // връща референцията на екземпляр на класа
```

```
    { return this; } //ако махнем този код - грешка!!!
```

```
    public void Set(int x, int y, int r)
```

```
    {this.x = x; this.y = y; radius = r;}
```

```
    public void Get(){Console.WriteLine("x ={0},y={1}, r={2}",x,y,radius);}
}
```

class Program

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

```
    { Circle cr = new Circle();
```

```
        //Създаване на екземпляр на клас cr
```

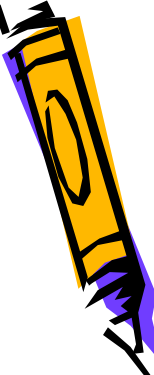
```
        cr.Set(1, 1, 10); //Ако напишем:
```

```
        Circle b = cr.T(); //Вместо Circle b = cr;
```

```
        //Получаваме референцията на cr, аналог Circle b=cr;
```

```
        cr.Get();
```

```
    } } //x=1, y=1, r=10
```



# Методи с променлив брой параметри

- В C# можем да дефинираме методи с променлив брой параметри.
- Пример за такъв метод, който неведнъж сме ползвали в нашите примери, е **Console.WriteLine(...)**.
- В общия случай указваме, че методът приема произволен брой параметри със служебната дума **params** в списъка от параметри. Тя може да се използва най-много веднъж в дефиницията на даден метод и задължително се прилага към последния изреден параметър, който трябва да бъде масив, приемащ множеството от параметрите. Не може да се комбинира с **ref** (не е и необходимо - параметъра е масив, тоест - референтен тип).
- В следващия пример е дефиниран метод **Multiply(...)**, който изчислява произведението на произволен брой цели числа от тип **int**. На метода от примера могат да бъдат подадени както произволен брой променливи от тип **int**, така и масив от тип **int**, т. е. допустими извиквания са както **Multiply(1, 2, 3, 4, 5)**, така и **Multiply(new int[] { 1, 2, 3, 4, 5 })**.





# Методи с променлив брой параметри

using System;

class Test

{ **static int Multiply(params int[] a)**

{ int pr = 1;

foreach (int arg in a)

{

pr\*= arg;

}

return pr;

}

static void Main()

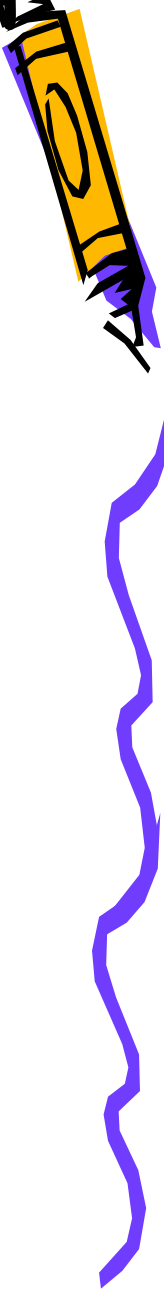
{ int Pr = Multiply(1, 2, 3, 4, 5);

//int Pr = Multiply(new int[] { 1, 2, 3, 4, 5 });

Console.WriteLine(Pr); **// 120**

Console.ReadKey();

}



# Методи с променлив брой параметри

В частност Main() method също може да приема произволен брой параметри чрез своя параметър args - масив от тип string:

Пример: (example.cs)

```
using System;
```

```
class Test
```

```
{ static void Main(string[] args)
```

```
{     int sum=0;
```

```
    Console.WriteLine("You entered the following {0} command  
line arguments:", args.Length);
```

```
    for (int i = 0; i < args.Length; i++)
```

```
    {        Console.WriteLine("{0}", args[i]);
```

```
        sum += Convert.ToInt32(args[i]);
```

```
    }
```

```
    Console.WriteLine("The sum is " + sum);
```

```
}
```

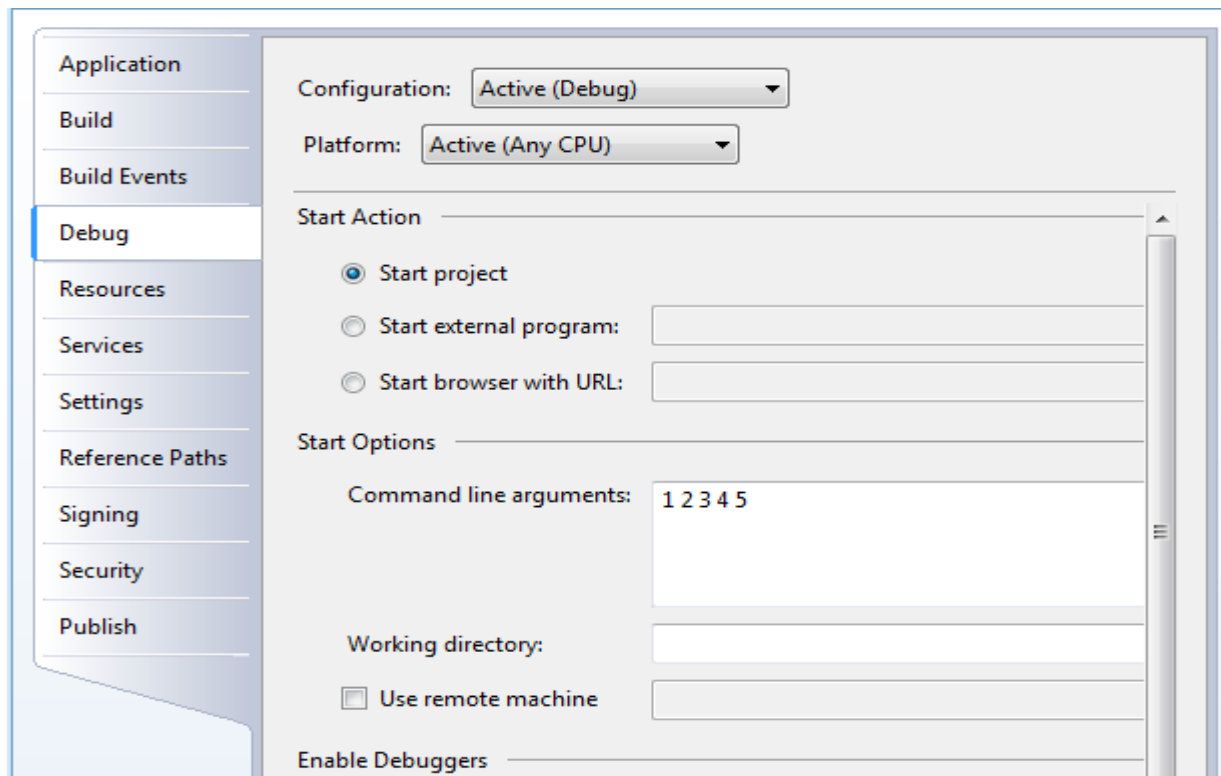
```
}
```



# Методи с променлив брой параметри

Във Visual Studio аргументите се задават по следния начин:

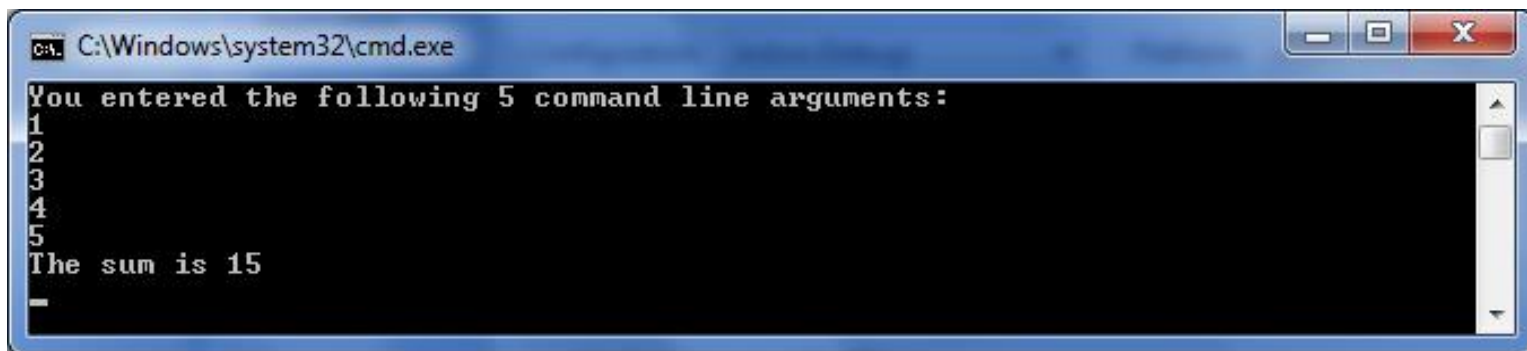
- Щракаме с десен бутон на мишката върху името на ансамбъла. Избираме Properties.
- В прозореца Properties избираме раздел Debug
- И в поле Command line Arguments задаваме например аргументите: **1 2 3 4 5**
- Изпълнение: Debug - Start Without Debugging



# Методи с променлив брой параметри

Когато се изпълнява програмата от команден ред, то стойностите на аргументите се задават при извикването на .exe файла, например за example.exe:

**example 1 2 3 4 5 <Enter>**



```
cmd C:\Windows\system32\cmd.exe
You entered the following 5 command line arguments:
1
2
3
4
5
The sum is 15
-
```

The screenshot shows a Windows command prompt window titled "cmd C:\Windows\system32\cmd.exe". The window contains the following text: "You entered the following 5 command line arguments:", followed by a list of numbers 1 through 5, each on a new line. Below the list, it says "The sum is 15". The window has a standard Windows title bar with minimize, maximize, and close buttons.

# Опционални параметри на методи (параметри с подразбиращи се стойности)

C# позволява да се използват опционални (незадължителни) параметри. За такива параметри трябва да се зададе стойност по подразбиране. Освен това, тези параметри са последни в списъка:

```
static int OptionalParam(int x, int y, int z=5, int s=4)
{
    return x + y + z + s;
}
```

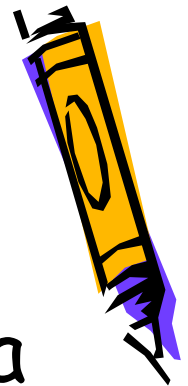
Тъй като последните два параметъра са обявени като опционални (не задължителни), то може да изпуснем един или двата от тях:

```
static void Main(string[] args)
{
    Console.WriteLine(OptionalParam(2, 3));           //14
    Console.WriteLine(OptionalParam(2, 3, 10));       //19
    Console.ReadLine();
}
```



# Именовани параметри на методи

- В предния пример, при извикването на методите, стойностите на параметрите се предават в реда на обявяване на тези параметри в метода.
- Можем да нарушим този порядък, използвайки така наречените "именовани параметри":



# Именовани параметри

```
static int OptionalParam(int x, int y, int z=5, int s=4)
{
    return x + y + z + s;
}
```

```
static void Main(string[] args)
{    Console.WriteLine(OptionalParam(x:2, y:3));
```

//14

//опционалният параметър z използва стойност по подразбиране

```
Console.WriteLine(OptionalParam(y:2, x:3, s:10));
```

//20

```
Console.ReadLine();
```

```
}
```



# Полиморфизъм. Предефиниране на методи

- Полиморфизъм - буквално означава приемането на различни форми на един обект.
- Нека **един базов клас A**, представящ обща категория от обекти, има един метод **Action()**, който се наследява от множество класове, например **B, C и D** описващи по-тесни категории обекти.
- Въпреки, че **B, C и D** споделят **метод Action()**, те го реализират по различен начин. В този случай, в базовия клас, метод **Action()** се указва с изричен модификатор **virtual**, а в породените класове - с изричен модификатор **override**. Чрез модификатор **virtual** се реализира предефиниране на съответния виртуален член (в случая - метод Action()) в породените класове.





# Полиморфизъм. Предефиниране на методи

using System;

class Test

```
{ static void Main()
{ Child theChild=new Child();      Parent theParent = theChild;
  Console.WriteLine("Age = {0}", theParent.age);
  Console.Write("Profession ="); theParent.Profession();
}
}
```

class Parent

```
{ public int age = 50;
  public virtual void Profession()
  { Console.WriteLine(" Teacher"); }
}
```

class Child : Parent

```
{ public new int age = 20;
  public override void Profession()
  { Console.WriteLine(" Student"); }
}
```

Какво ще изведе  
кода?



# Полиморфизъм. Предефиниране на методи

using System;

class Test

{ static void Main()

{ Child theChild=new Child(); Parent theParent = theChild;

Console.WriteLine("Age = {0}", theParent.age);

Console.Write("Profession ="); theParent.Profession();

}

}

class Parent

{ public int age = 50;

public **virtual** void Profession()

{ Console.WriteLine(" Teacher"); }

}

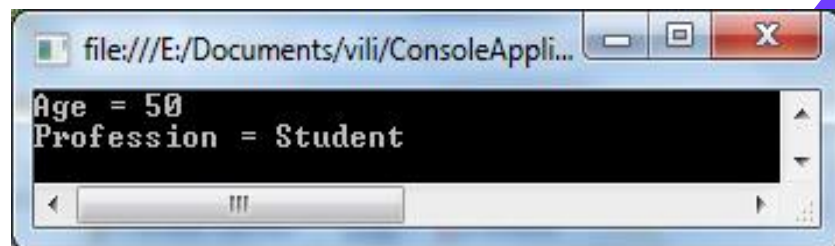
class Child : Parent

{ public **new** int age = 20;

public **override** void Profession()

{ Console.WriteLine(" Student"); }

}



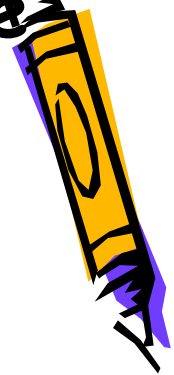
# Предефиниране и скриване на членове

- Тълкуване на резултата:

**Age=50**

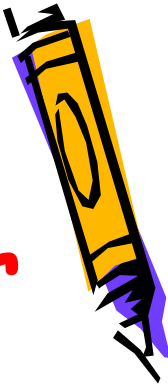
**Profession = Student**

- Вижда се, че чрез модификатор override се реализира предефиниране на виртуалния метод Profession() в породения клас Child.
- Чрез модификатор new в породения клас Child се реализира скриване на член от базовия клас (в случая - на полето age).
- Ако няма модификатор new компилаторът ще изведе предупреждение за да заостри вниманието върху това.



# Предефиниране на оператори (Overloading operators)

- **public static връщан\_тип operator  
име\_оператор(параметри)**
- Когато препокриваме един оператор, трябва задължително да го обявим като **public static**, тъй като той ще се използва за всички обекти на дадения клас
- Следва връщания тип, след което ключовата дума **operator**.
- Следва името на оператора и параметрите му.



# Предефиниране на оператори

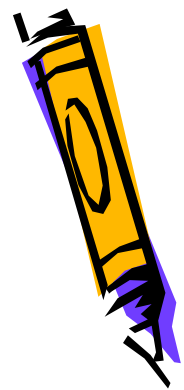
## Пример:

- Нека имаме клас State (държава).

**class State**

```
{  public string Name { get; set; } // название  
    public int Population { get; set; } // население  
    public double Area { get; set; } // площ  
...}
```

- Ако искаме да използваме класа за да обединим няколко държави, тогава тази задача може да се реши с предефиниране на оператора "+".
- Освен това, ще предефинираме и операторите за сравнение - ">" и "<", с помощта на които ще сравняваме две държави:



## Предефиниране на оператори +, < и >:

//+ - връща нова, обединена държава

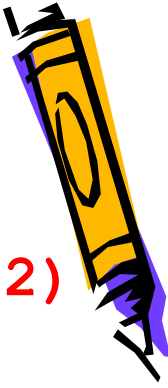
```
public static State operator+ (State s1, State s2)
{...}
```

//< - връща true, ако s1 е по-малка по площ  
държава от s2

```
public static bool operator < (State s1, State s2)
{...}
```

//> - връща true, ако s1 е по-голяма по площ  
държава от s2

```
public static bool operator > (State s1, State s2)
{...}
```



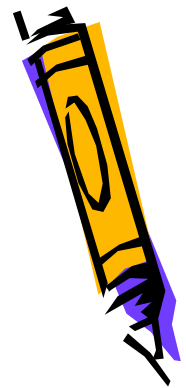
```
class State
{
    public string Name{ get; set; }
    public int Population{get;set;}
    public double Area{ get; set; }

    public static State operator+
    (State s1, State s2)
    {
        string name = s1.Name;
        int people =
        s1.Population+s2.Population;
        double area = s1.Area+s2.Area;
        // връща новата обединена
        държава
        return new State { Name = name,
        Area = area, Population =
        people};
    }
}
```

```
public static bool operator<
(State s1, State s2)
{
    if (s1.Area < s2.Area)
        { return true; }
    else
        { return false;}
}

public static bool operator>
(State s1, State s2)
{
    if (s1.Area > s2.Area)
        { return true; }
    else
        { return false;}
}
}
```

- Тъй като всички предефинирани оператори са бинарни (прилагат се върху 2 обекта), то за всеки предефиниран оператор се предвиждат по два параметъра.
- Следва примерен код за използване на предефинираните оператор  $+$ ,  $<$  и  $>$ :





```
static void Main()
```

```
{ State s1 = new State{ Name = "State1", Area = 300, Population = 100 };  
  State s2 = new State{ Name = "State2", Area = 200, Population = 70 };
```

```
  if (s1 > s2)
```

```
  { Console.WriteLine("State s1 bigger than state s2");  
  }
```

```
  else if (s1 < s2)
```

```
  { Console.WriteLine("State s1 smaller than state s2");  
  }
```

```
  else
```

```
  { Console.WriteLine("State s1 equals state s2");  
  }
```

```
  State s3 = s1 + s2;
```

```
  Console.WriteLine("State name : {0}", s3.Name);
```

```
  Console.WriteLine("State area : {0}", s3.Area);
```

```
  Console.WriteLine("State population : {0}", s3.Population);
```

```
}  ИЗХОД:
```

```
  State s1 bigger than state s2
```

```
  State name : State1
```

```
  State area :500
```

```
  State population :170
```



# Изброим тип - enum

- Представяват набор от логически свързани константи.

```
enum days
```

```
{ monday, tuesday, wednesday, thursday, friday, saturday, sunday }
```

- Обявяват се с помощта на оператора enum.
- Следва името на изброимия тип, след което целочислен тип (byte, int, short, long).
- Ако целочислен тип не е указан, то се подразбира int.
- Следва списъка на елементите, разделени със запетая.

```
enum days:int
```

```
{ monday=0, tuesday=1, wednesday=2, thursday=3, friday=4, saturday=5, sunday=6 }
```



enum operation

{

add = 1,

/\* всеки следващ елемент по  
подразбиране се увеличава с 1 \*/

subtract, // 2

multiplay, // 3

divide // 4

}



enum operation

{

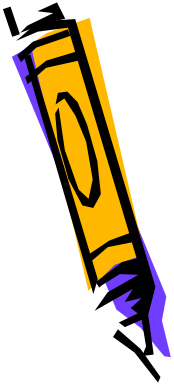
add = 2,

subtract = 4,

multiplay = 8,

divide = 16

}



```
using System;
class Program
{ enum Operation
    { add = 1,
      subtract,
      multiplay,
      divide
    }
}
```

```
static void MathOp(double x, double y,
Operation op)
{ double result = 0.0;
  switch (op)
  { case Operation.add:
    result = x + y;
    break;
    case Operation.subtract:
    result = x - y;
    break;
    case Operation.multiplay:
    result = x * y;
    break;
    case Operation.divide:
    result = x / y;
    break;
  }
}
```

```
Console.WriteLine("Резултатът от
операцията е равен {0}", result);
```

```
static void Main(string[] args)
{ // Типът на операцията се задава
  // чрез константата Operation.add, която
  // е равна на 1
  MathOp(10, 5, Operation.add);
  // Типът на операцията се задава
  // с константата Operation.multiplay,
  // която е равна на 3
  MathOp(11, 5,
Operation.multiplay);
  Console.ReadLine();
}
```

