

СЪДЪРЖАНИЕ

I. Принципи на Обектно-ориентираното програмиране в .NET.

1. Предимства и особености на ООП
2. Основни принципи

II. Основни понятия

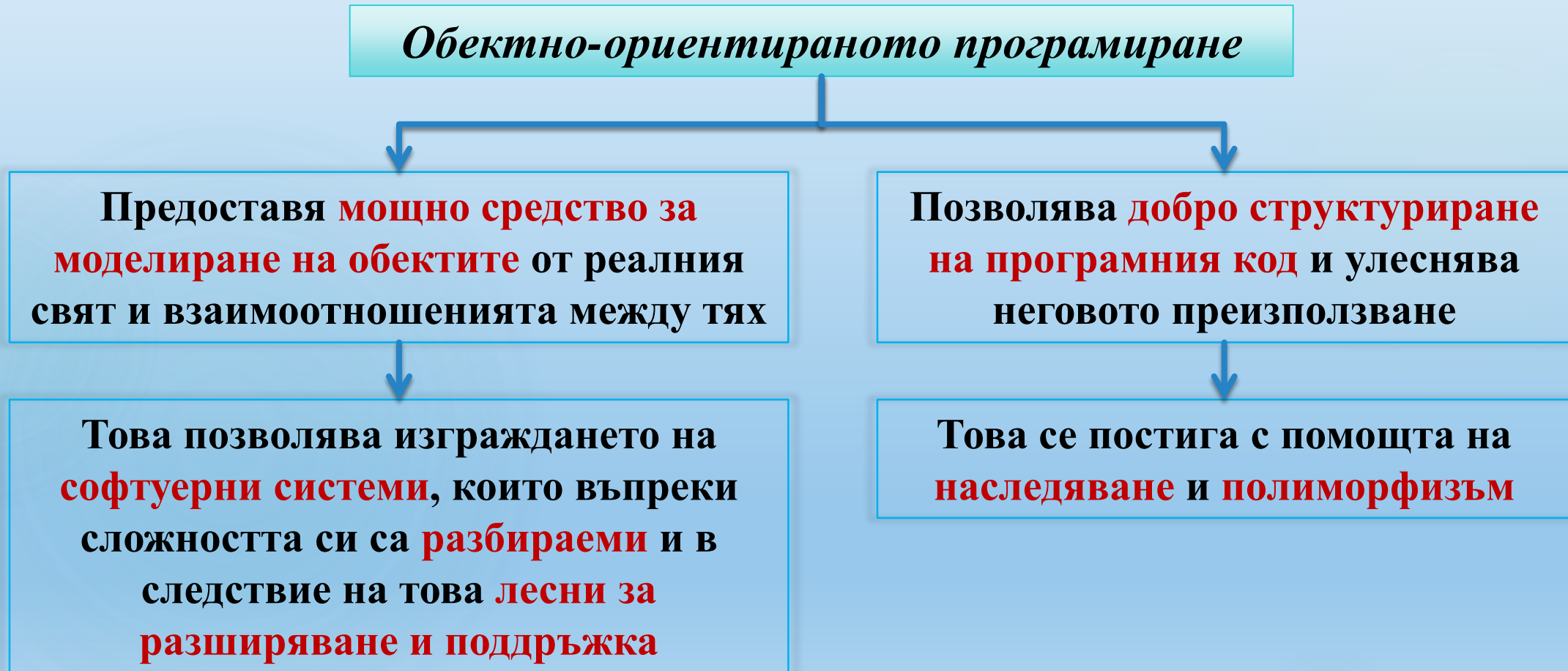
1. Общи понятия
2. Променливи в C#
3. Константи в C#
4. Идентификатори в C#
5. Примитивни типове данни
6. Конвертирането на типове данни
7. Въвеждане на входни данни
8. Операторите в C#

9. Класът Math
10. Работа със низове
11. Условни конструкции в C#
12. Цикли
13. Масиви в C#

I. ПРИНЦИПИ НА ОБЕКТНО-ОРИЕНТИРАНОТО ПРОГРАМИРАНЕ В .NET.

2

1. Предимства и особености на ООП



I. ПРИНЦИПИ НА ОБЕКТНО-ОРИЕНТИРАНОТО ПРОГРАМИРАНЕ В .NET.

2

2. Основни принципи на ООП

Капсулация на данните



Скриване на ненужните
детайли за обектите и
откриване към външния
свят само на важните
техни характеристики и
свойства

Наследяване



Класът наследник
наследява всички
обекти, свойства и
действия (методи)
на друг клас,
наричан базов

Полиморфизъм



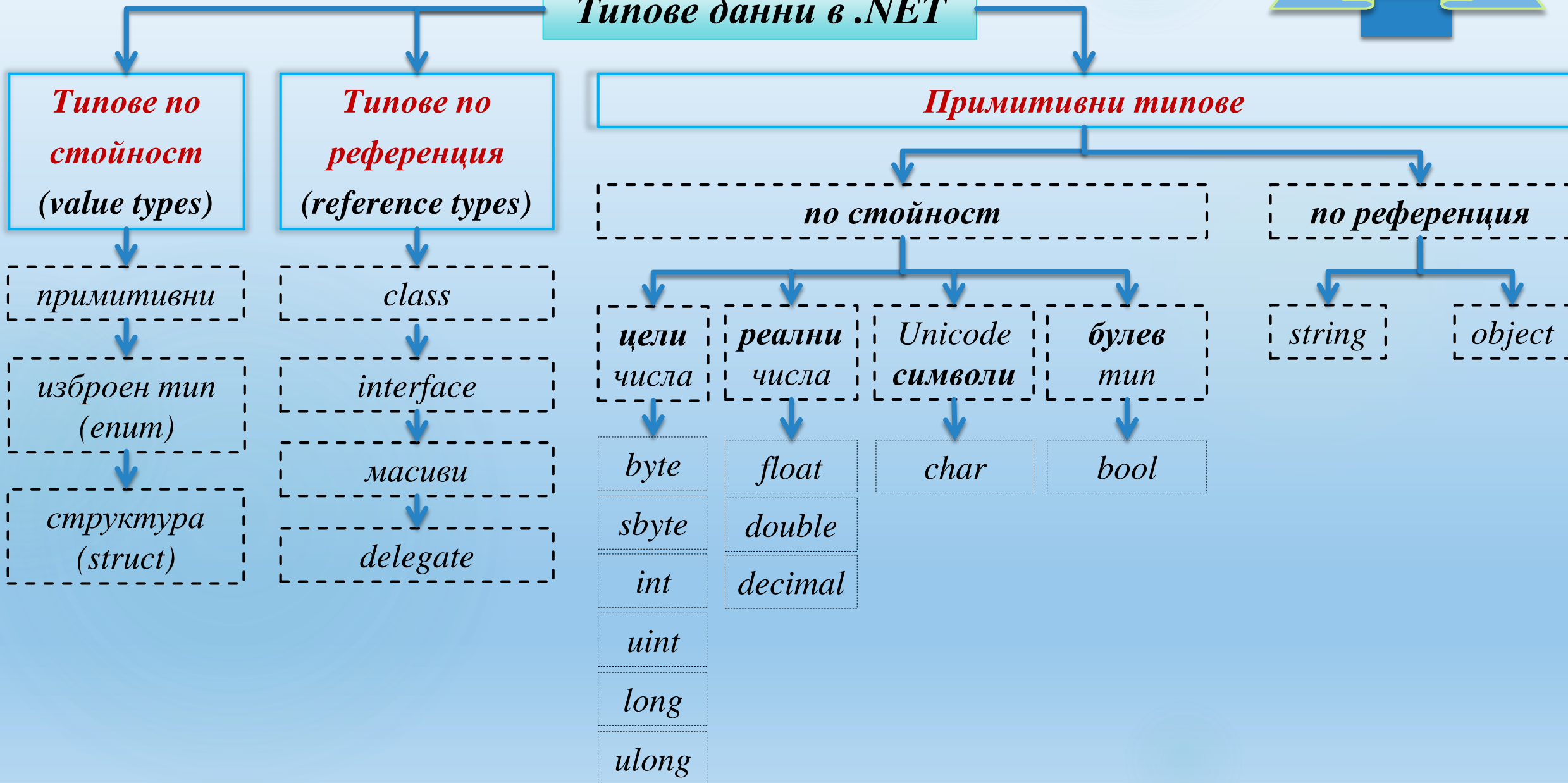
Приемането на
различни форми от
един обект

II. ОСНОВНИ ПОНЯТИЯ

1. Общи понятия

2

Типове данни в .NET



II. ОСНОВНИ ПОНЯТИЯ

1. Общи понятия



Запазена дума за
дефиниране на
изброен тип

Име на
енумерацията

```
public enum Days
{
    Saturday,
    Sunday,
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday
};
```

Изброени типове (**enum** /enumerations/) – Изброените типове в C# се състоят от множество именувани константи. Дефинират се със запазената дума enum. Изброените типове се използват за задаване на една измежду няколко възможности.

```
Days today = Days.Friday;
if (today == Days.Friday)
{
    Console.WriteLine("Днес е петък.");
}
```

Инстанция на
енумерацията

Инстанциите на изброените типове могат да приемат една от дефинираните в тях стойности



Структури (struct) – Структурите в .NET Framework представляват **съвкупност** от полета с данни.

Структурите, както и класовете, **могат да дефинират**:

- Конструктори
- Полета
- Свойства
- индексатори
- други членове

```
struct Point
{
    public int mX, mY;
}
```

```
struct Square
{
    public Point mLocation;
    public int mSize;
    public Color mBorderColor;
    public Color mSurfaceColor;
}
```

```
struct Color
{
    public byte mRedValue;
    public byte mGreenValue;
    public byte mBlueValue;
}
```

Клас

Тяло на класа

Членове на класа:

- *полета*, или член-променливи (*fields*)
- *константи* (*constants*)
- *методи*, или член-функции (*methods*)
- *свойства* (*properties*)
- *индексатори* (*indexers*)
- *събития* (*events*)
- *оператори* (*operators*)
- *конструктори* (*constructors*)
- *деструктори* (*destructors*)
- *вложени типове* (класове, структури, изброени типове и др.)

Видимост на членовете:

- *public*
- *protected internal*
- *internal*
- *protected*
- *private*

Дефиниция на клас

```
class Students
{
    //тяло на класа
}
```

Име на класа

II. ОСНОВНИ ПОНЯТИЯ 1. Общи понятия



Цялостна структура на един клас

```
class Student
```

```
{
```

```
    private string mFirstName;
```

```
    private string mLastName;
```

```
    private const double PI = 3.14;
```

```
    public Student(string aStudentName, int age)
```

```
    {
```

```
        //.....
```

```
    }
```

```
    public string StoreExamResult(string aSubject, double aGrade)
```

```
    {
```

```
        // ...
```

```
    }
```

```
}
```

Полета

(член-променливи)

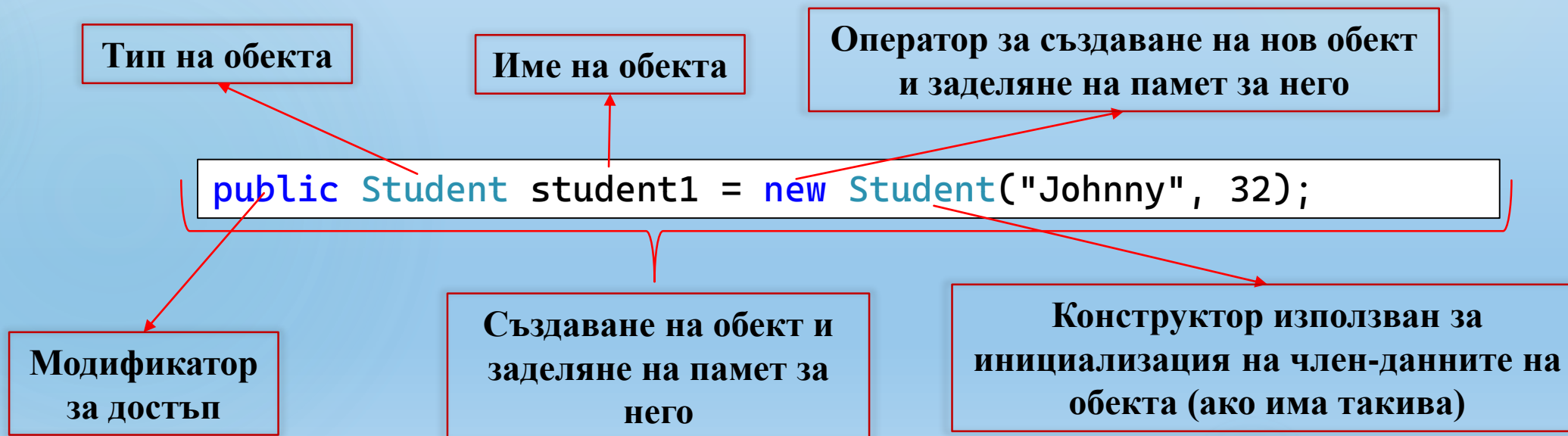
Константа

Конструктор

Метод

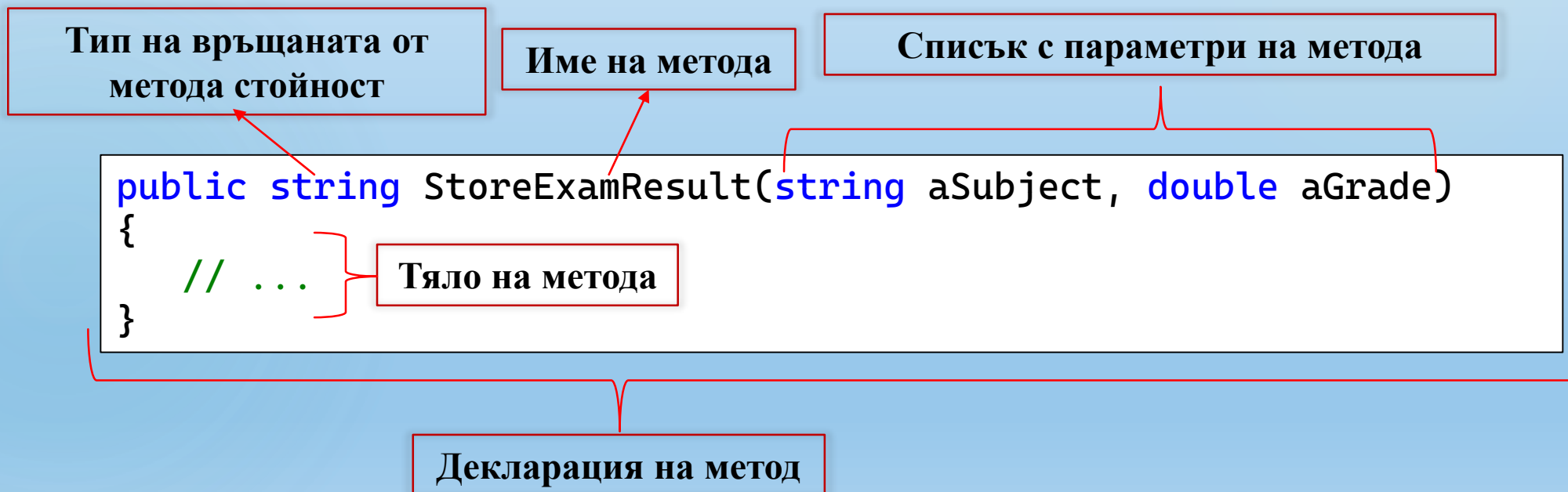
Клас - класовете са категории от обекти, споделящи общи свойства и операции, които могат да се извършват върху тях. Например класът "студент" представя множеството от всички студенти.

Обект - обект наричаме конкретен елемент от даден клас (инстанция), например студентът Тодор Георгиев, трети курс, ядрена физика в СУ.

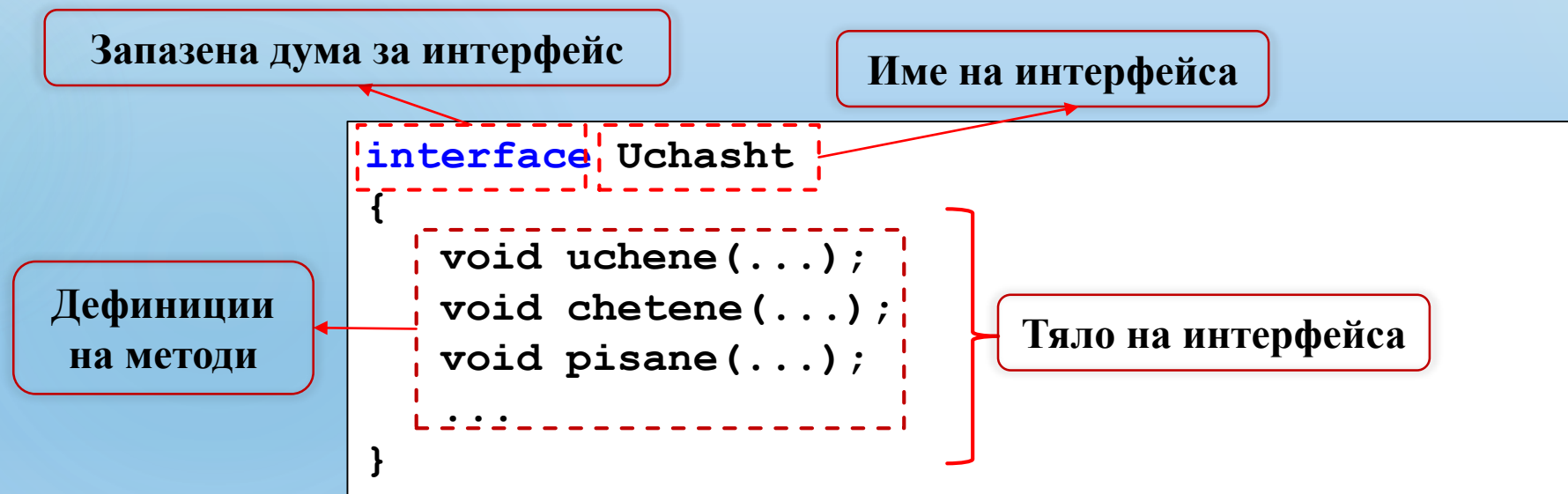


Инстанциране - процесът на създаване на обект от даден клас е инстанциране. Обектите, създадени при инстанциране на даден клас, се наричат негови инстанции.

Метод - Метод е действие, което всички обекти от даден клас могат да извършват. Например всички обекти от класа "студент" могат да извършват действието "явяване на изпит".



Интерфейс - Интерфейсът е описание на съвкупност от действия, които даден обект може да извършва. Ако един обект може да извършва всички действия от даден интерфейс, казваме че обектът реализира, или имплементира интерфейса. Класът "студент", например, би могъл да реализира интерфейса "учащ" съдържащ действието "учене", четене“, „писане“, „Явяване на изпити“.



Наследяване (Inheritance) на класове - Наследяване в ООП наричаме възможността **един клас, наричан наследник, да придобие свойства и действия на друг клас – родител (базов клас).**

Пример: Лъвът е от семейство котки. Всички котки имат четири лапи, хищници са, преследват жертвите си. Тази функционалност може да се напише веднъж в клас Котки и всички хищници да я преизползват – тигър, пума, рис и т.н.

Класът, **който наследяваме**, се нарича **клас-родител** или още **базов клас** (base class, super class).

В **.NET** и други модерни езици за програмиране **един клас може да наследи само един друг клас (single inheritance)**, за разлика от C++, където се поддържа множествено наследяване (multiple inheritance).

```
public class Kotki
```

```
{  
    private bool male;  
    public Kotki() : this(true) { }  
  
    public Kotki(bool male)  
    {  
        this.male = male;  
    }  
    public bool Male  
    {  
        get  
        {  
            return male;  
        }  
        set  
        {  
            this.male = value;  
        }  
    }  
}
```

Базов клас

```
public class Lion : Kotki
```

```
{  
    private int weight;  
    public Lion(bool male, int weight):base(male)  
    {  
        this.weight = weight;  
    }  
    public int Weight  
    {  
        get  
        {  
            return weight;  
        }  
        set  
        {  
            this.weight = value;  
        }  
    }  
}
```

Клас-наследник

В горния пример в конструктора на класа **Lion** използваме ключовата дума **base**. Тя указва да бъде използван базовият клас и позволява достъп до негови методи, конструктори и член-променливи. С **base()** можем да извикаме конструктор на базовия клас. С **base.method(...)** можем да извикаме метод на базовия клас, да му подаваме параметри и да използваме резултата от него. С **base.field** можем да вземем стойността на член-променлива на базовия клас или да ѝ присвоим друга стойност.



base може да се използва изрично, за яснота. **base.method(...)** извиква метод, който задължително е от базовия клас. Такъв код се чете по-лесно, защото знаем къде да търсим въпросния метод.

Имайте предвид, че ситуацията с **this** не е такава. **this** може да означава както метод от конкретния клас, така и метод от който и да е базов клас.

II. ОСНОВНИ ПОНЯТИЯ



2. Променливи в C#

Променливите са **контейнери за съхраняване на стойности на данни**.

В C# има различни типове променливи (дефинирани с различни ключови думи спрямо типа данни, за който се отнасят), например:

int - съхранява цели числа (цели числа), без десетични знаци, като **123** или **-123**;

double - съхранява числа с плаваща запетая с десетични знаци, като **19.99** или **-19.99**;

char - съхранява единични знаци, като **'a'** или **'B'**; Стойностите на Char са заобиколени от **единични кавички**;

string - съхранява текст, като например **"Hello World"**. Стойностите на низовете са оградени с **двойни кавички**;

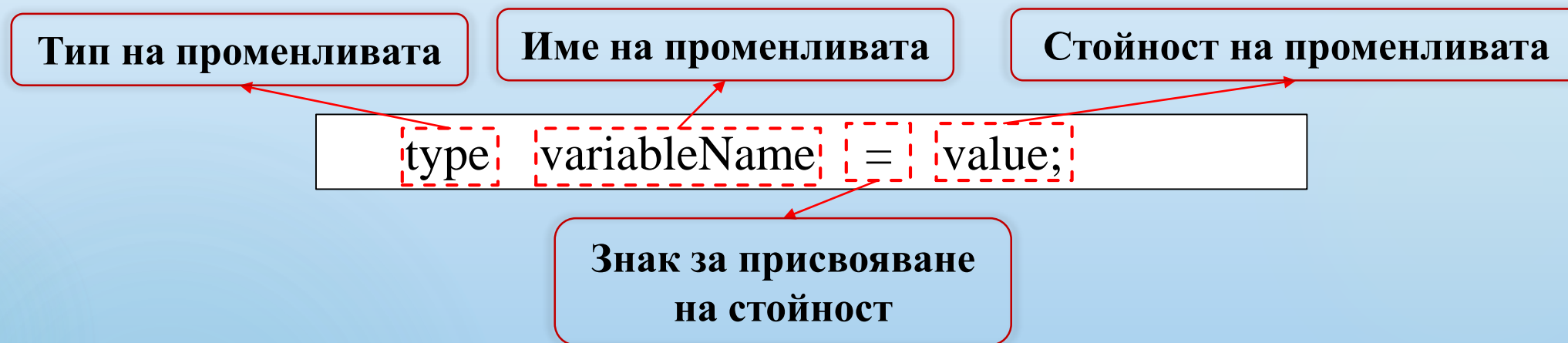
bool - съхранява стойности с две състояния: **true** или **false**.

II. ОСНОВНИ ПОНЯТИЯ



Деклариране (създаване) на променливи

За да създадете променлива, трябва да посочите **типа** и да й **присвоите стойност**:



Примери за създаване на променливи:

```
string name = "John";  
Console.WriteLine(name);
```

```
int myNum;  
myNum = 15;  
Console.WriteLine(myNum);
```

```
int myNum = 5;  
double myDoubleNum = 5.99D;  
char myLetter = 'D';  
bool myBool = true;  
string myText = "Hello";
```

II. ОСНОВНИ ПОНЯТИЯ



Показване (отпечатване) на променливи

Методът **WriteLine()** често се използва за отпечатване на стойността на дадена променлива/променливи в прозореца на конзолата. За да комбинирате текст и променлива, използвайте знака +:

```
string firstName = "John ";  
string lastName = "Doe";  
string fullName = firstName +  
lastName;  
Console.WriteLine(fullName);
```

За числови стойности знакът + работи като математически оператор (забележете, че тук използваме променливи от целочислен тип - **int**):

```
int x = 5;  
int y = 6;  
Console.WriteLine(x + y); // Print the value of x + y
```

II. ОСНОВНИ ПОНЯТИЯ



Деклариране на много променливи от един тип

```
int x = 5, y = 6, z = 50;  
Console.WriteLine(x + y + z);
```

Можете също да присвоите една и съща стойност на множество променливи в един ред:

```
int x, y, z;  
x = y = z = 50;  
Console.WriteLine(x + y + z);
```

II. ОСНОВНИ ПОНЯТИЯ



3. Константи в C#

Ако не искате другите (или вие самите) да презаписват съществуващите стойности на вече създадени променливи, можете да добавите ключовата дума **const**. Това ще декларира променливата като "константа", което означава **непроменлива и само за четене**:

```
const int myNum = 15;  
myNum = 20; // error
```

Ключовата дума **const** е полезна, когато искате променливата **винаги** да **съхранява една и съща стойност**, така че другите (или вие) да не объркат кода ви. Пример, който често се нарича константа, е **числото PI (3.14159...)**.

Забележка: **Не можете да декларирате константна променлива, без да ѝ присвоите стойност. Ако го направите, ще възникне грешка!!!**

II. ОСНОВНИ ПОНЯТИЯ



4. Идентификатори в C#

Всички C# променливи трябва да бъдат идентифицирани с уникални имена. Тези уникални имена се наричат идентификатори. Идентификаторите могат да бъдат кратки имена (като **x** и **y**) или по-описателни имена (**age**, **sum**, **totalVolume**).

Забележка: Препоръчително е да използвате описателни имена, за да създадете разбираем и поддържаем код !

```
int minutesPerHour = 60; // OK
// OK, но не е толкова лесно да се разбере какво е m
int m = 60;
```

Общите правила за именуване на променливи са:

- Имената могат да съдържат букви, цифри и долна черта (_)
- Имената трябва да започват с буква
- Имената трябва да започват с малка буква и не могат да съдържат интервал
- Имената са чувствителни към главни и малки букви ("myVar" и "myvar" са различни променливи)
- Запазени думи (като `int` или `double`) не могат да се използват като имена

II. ОСНОВНИ ПОНЯТИЯ



5. Примитивни типове данни в C#

Тип данни	Стойност по подразбиране	Минимална стойност	Максимална стойност
sbyte	0	-128	127
byte	0	0	255
short	0	-32768	32767
ushort	0	0	65535
int	0	-2147483648	2147483647
uint	0u	0	4294967295
long	0L	-9223372036854775808	9223372036854775807
ulong	0u	0	18446744073709551615
float	0.0f	$\pm 1.5 \times 10^{-45}$	$\pm 3.4 \times 10^{38}$
double	0.0d	$\pm 5.0 \times 10^{-324}$	$\pm 1.7 \times 10^{308}$
decimal	0.0m	$\pm 1.0 \times 10^{-28}$	$\pm 7.9 \times 10^{28}$
boolean	false	Възможните стойности са две – true или false	
char	'\u0000'	'\u0000'	'\uffff'
object	null	-	-
string	null	-	-

Типът данни определя размера и типа на стойностите на променливата. **Важно е да използвате правилния тип данни за съответната променлива** за да се избегнат грешки, за спестяване на време и памет. Най-често срещаните типове данни са:

II. ОСНОВНИ ПОНЯТИЯ



5. Примитивни типове данни в C#

Примери:

```
int myNum = 100000;  
long myNum = 1500000000000L;  
float myNum = 5.75F;  
double myNum = 19.99D;
```

Специални символи добавени
след числовите стойности

Научни числа - число с плаваща запетая може също да бъде научно число със знака "e", за да посочи степента на 10:

```
float f1 = 35e3F;  
double d1 = 12E4D;  
Console.WriteLine(f1);  
Console.WriteLine(d1);
```

35000
120000

II. ОСНОВНИ ПОНЯТИЯ



6. Конвертирането (**casting**) на типове данни в C#

Конвертиране на тип данни е, когато се присвоява стойност от един тип данни на друг тип. В C# има два вида конвертиране:

- **Неявно конвертиране** (автоматично) - конвертиране на тип с по-малък размер в тип с по-голям размер, например: `char -> int -> long -> float -> double`

```
int myInt = 9;
double myDouble = myInt;    // Automatic casting: int to double
Console.WriteLine(myInt);   // Outputs 9
Console.WriteLine(myDouble); // Outputs 9
```

- **Изрично конвертиране** (ръчно) - конвертиране на тип с по-голям размер в тип с по-малък размер, например: `double -> float -> long -> int -> char`

```
double myDouble = 9.78;
int myInt = (int)myDouble;    // Manual casting: double to int
Console.WriteLine(myDouble);  // Outputs 9.78
Console.WriteLine(myInt);     // Outputs 9
```

II. ОСНОВНИ ПОНЯТИЯ



Методи за преобразуване на типове

Възможно е също така изрично да конвертирате типове данни **чрез използване на вградени методи**, като **Convert.ToBoolean**, **Convert.ToDouble**, **Convert.ToString**, **Convert.ToInt32 (int)** и **Convert.ToInt64 (long)**:

```
int myInt = 10;  
double myDouble = 5.25;  
bool myBool = true;
```

```
Console.WriteLine(Convert.ToString(myInt));    // convert int to string  
Console.WriteLine(Convert.ToDouble(myInt));    // convert int to double  
Console.WriteLine(Convert.ToInt32(myDouble));  // convert double to int  
Console.WriteLine(Convert.ToString(myBool));   // convert bool to string
```

II. ОСНОВНИ ПОНЯТИЯ



7. Въвеждане на входни данни

Вече научихте, че `Console.WriteLine()` се използва за извеждане (отпечатване) на стойности. Сега ще разгледаме **`Console.ReadLine()`**, за да получим въвеждане от потребителя. В следващия пример потребителят може да въведе своето потребителско име, което се съхранява в променливата **`userName`** от тип **`string`**. След това ще отпечатаме стойността на `userName`:

```
// Type your username and press enter
Console.WriteLine("Enter username:");

// Create a string variable and get user input from the
// keyboard and store it in the variable
string userName = Console.ReadLine();

// Print the value of the variable (userName), which will
// display the input value
Console.WriteLine("Username is: " + userName);
```

II. ОСНОВНИ ПОНЯТИЯ

2

Потребителско въвеждане и числа

Методът `Console.ReadLine()` връща низ. Следователно не можете да получите информация от друг тип данни, като `int`. Следната програма ще предизвика грешка:

```
Console.WriteLine("Enter your age:");  
int age = Console.ReadLine();  
Console.WriteLine("Your age is: " + age);
```

Съобщението за грешка ще бъде нещо подобно:

```
Cannot implicitly convert type 'string' to 'int'
```

Както се казва в съобщението за грешка, **не можете неявно (автоматично) да преобразувате тип „string“ в „int“**. Можете да конвертирате всеки тип чрез използване на изрично (ръчно) конвертиране, като използвате един от методите **Convert.To**:

```
Console.WriteLine("Enter your age:");  
int age = Convert.ToInt32(Console.ReadLine());  
Console.WriteLine("Your age is: " + age);
```

Ръчно конвертиране

II. ОСНОВНИ ПОНЯТИЯ



8. Операторите в C#

Операторите се използват за извършване на операции с променливи и стойности.

Въпреки че операторът + често се използва за събиране на две стойности, той може да се използва и за събиране на променлива и стойност или променлива и друга променлива:

```
int sum1 = 100 + 50;           // 150 (100 + 50)
int sum2 = sum1 + 250;         // 400 (150 + 250)
int sum3 = sum2 + sum2;         // 800 (400 + 400)
```

Аритметични оператори

Operator	Name	Description	Example
+	Събиране	Събира две стойности	x + y
-	Изваждане	Изважда една стойност от друга	x - y
*	Умножение	Умножава две стойности	x * y
/	Делене	Разделя една стойност на друга	x / y
%	Modulus	Връща остатък от делението	x % y
++	Инкрементиране	Увеличава стойността на променливата с 1	x++
--	Декрементиране	Намалява стойността на променливата с 1	x--

II. ОСНОВНИ ПОНЯТИЯ



Оператори за присвояване

Операторите за присвояване се използват за присвояване на стойности на променливи. В примера по-долу използваме оператора за присвояване (=), за да присвоим стойност 10 на променлива, наречена x. Операторът за добавяне на присвояване (+=) добавя стойност към променлива:

```
int x = 10;  
x += 5;
```

Оператор	Пример	Същото като
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
=	x = 3	x = x 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

II. ОСНОВНИ ПОНЯТИЯ



Оператори за сравнение

Операторите за сравнение се използват за сравняване на две стойности:

Забележка: Върнатата стойност на сравнение е True или False. В следващия пример използваме оператора по-голямо от (>), за да разберем дали 5 е по-голямо от 3:

```
int x = 5;  
int y = 3;  
Console.WriteLine(x > y); // returns True because 5 is greater than 3
```

Оператор	Име	Пример
==	Равно на	x == y
!=	Различно от	x != y
>	По-голямо от	x > y
<	По-малко от	x < y
>=	По-голямо или равно на	x >= y
<=	По-малко или равно на	x <= y

II. ОСНОВНИ ПОНЯТИЯ



Логически оператори

Логическите оператори се използват за определяне на логиката между променливи или стойности:

Оператор	Име	Описание	Пример
&&	логическо И	Връща True, ако и двете твърдения са верни	$x < 5 \ \&\& \ x < 10$
	логическо ИЛИ	Връща True, ако едно от твърденията е вярно	$x < 5 \ \ x < 4$
!	Логическо НЕ	Обръща резултата, връща False, ако резултатът е верен	$!(x < 5 \ \&\& \ x < 10)$

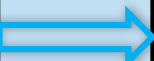
II. ОСНОВНИ ПОНЯТИЯ




9. Класът Math в C#

Класът Math в C# има много методи, които позволяват да се изпълняват математически задачи върху числа. Някои от по-важните и често използвани методи са:


- **Math.Max(x,y)** - методът може да се използва за намиране на най-високата стойност между две стойности например **x** и **y**

`Math.Max(5, 10);`  10


- **Math.Min(x,y)** - методът може да се използва за намиране на най-малката стойност между две стойности например **x** и **y**

`Math.Min(5, 10);`  5

- **Math.Sqrt(x)** - методът може да се използва за намиране на корен квадратен на **x**

`Math.Sqrt(9);`  3

- **Math.Abs(x)** - метод връща абсолютната (положителна) стойност на **x**

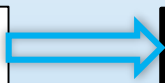
`Math.Abs(- 9);`  9

II. ОСНОВНИ ПОНЯТИЯ

2

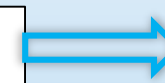
- **Math.Round(x,y)** - методът закръглява число до най-близкото цяло число

```
Math.Round(5.37);
```



5

```
Math.Round(6.687);
```



7

10. Работа със низове (string)

- **дължина на низ (string length)** - **Низът в C# всъщност е обект**, който съдържа свойства и методи, които могат да извършват определени операции върху низове. Например дължината на низ може да бъде намерена със **свойството Length**:

```
string txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
```

```
Console.WriteLine("The length of the txt string is: " + txt.Length);
```

The length of the txt string is: 26

- **методите ToUpper() и ToLower()** - връщат копие на низа, преобразуван в главни или малки букви:

```
string txt = "Hello World";
```

```
Console.WriteLine(txt.ToUpper());
```



HELLO WORLD

```
Console.WriteLine(txt.ToLower());
```



hello world

II. ОСНОВНИ ПОНЯТИЯ

2

- слепване на низове – методът *Concat()* :

```
// create string
string str1 = "C# ";

// create string
string str2 = "Programming";

// create string
string str3 = "Introduction";

// join two strings
string joinedString = string.Concat(str1, str2, str3);
Console.WriteLine("Joined string: " + joinedString);
```

Joined string: C# Programming Introduction

II. ОСНОВНИ ПОНЯТИЯ

2

- **сравняване на низове** - В С# можем да правим сравнения между два низа с помощта на метода **Equals()**. Методът **Equals()** проверява дали два низа са равни или не. Например:

```
// create string
string str1 = "C# Programming";
string str2 = "C# Programming";
string str3 = „C# Course“;
```

```
// compare str1 and str2
```

```
Boolean result1 = str1.Equals(str2);
```

```
Console.WriteLine("string str1 and str2 are equal: " + result1);
```



True

```
//compare str1 and str3
```

```
Boolean result2 = str1.Equals(str3);
```

```
Console.WriteLine("string str1 and str3 are equal: " + result2);
```



False

II. ОСНОВНИ ПОНЯТИЯ



- низови escape последователности -

Синтаксис	Описание
\'	Вмъква единична кавичка
\"	Вмъква двойна кавичка
\\	Вмъква наклонена черта

Синтаксис	Описание
\0	Вмъква 0 в текста
\n	Вмъква нов ред в текста
\t	Вмъква табулация

II. ОСНОВНИ ПОНЯТИЯ

2

- **интерполация на низове** - в C# можем да използваме интерполация на низове, за да вмъкнем променливи в низ. За интерполация на низ низовият литерал трябва да започва със знака \$. Например:

```
public static void Main(string[] args)
{
    // create string
    string name = " C# Course";

    // string interpolation
    string message = $"Welcome to {name}";
    Console.WriteLine(message);
}
```



Welcome to C# Course

В горния пример използваме променливата **name** вътре в низа на съобщението за да укажем точно къде в оригиналния низ искаме да вмъкнем новия текст.

Забележете това!!! - низовият литерал започва с \$ променливата име се поставя във фигурните скоби { }

II. ОСНОВНИ ПОНЯТИЯ



- други често използвани методи за работа с низове:

Методи	Описание
Format()	връща форматиран низ
Split()	разделя низа на подниз
Substring()	връща подниз от низ
Compare()	сравнява низови обекти
Replace()	замества посочения стар знак с посочения нов знак
Contains()	проверява дали низът съдържа подниз
Join()	обединява дадените низове с помощта на посочения разделител
Trim()	премахва всякакви начални и завършващи празни интервали
EndsWith()	проверява дали низът завършва с дадения низ
IndexOf()	връща позицията на посочения знак в низа
Remove()	връща знаци от низ

II. ОСНОВНИ ПОНЯТИЯ



Методи	Описание
ToUpper()	преобразува низа в главни букви
ToLower()	преобразува низа в малки букви
PadLeft()	връща низ, подплатен с интервали или с определен Unicode знак отляво
PadRight()	връща низ, подплатен с интервали или с определен Unicode символ отясно
StartsWith()	проверява дали низът започва с дадения низ
ToArray()	преобразува низа в масив от символи
LastIndexOf()	връща индекс на последното срещане на определен низ

II. ОСНОВНИ ПОНЯТИЯ



- **достъп до знаците в низ** - Можете да получите достъп до знаците в низ, като се обърнете към неговия индексен номер в квадратни скоби [].

Този пример отпечатва първия знак в низа **myString**:

```
string myString = "Hello";  
Console.WriteLine(myString[0]);
```

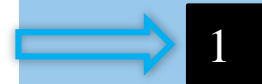


H

!!! Забележка: Индексите на низове започват с 0: [0] е първият знак. [1] е вторият знак и т.н.

Можете също така да намерите позицията на индекса на конкретен знак в низ, като използвате метода **IndexOf()**:

```
string myString = "Hello";  
Console.WriteLine(myString.IndexOf("e"));
```



1

II. ОСНОВНИ ПОНЯТИЯ




- методът *Substring()* - извлича знаците от низ, започвайки от указаната позиция/индекс на символа, и връща нов низ. Този метод често се използва заедно с *IndexOf()*, за да получите конкретната позиция на символа:

```
// Full name
string name = "John Doe";

// Location of the letter D
int charPos = name.IndexOf("D");

// Get last name
string lastName = name.Substring(charPos);

// Print the result
Console.WriteLine(lastName);
```



Doe

11. Условни конструкции в C#

C# има следните условни оператори:

- Използвайте **if**, за да посочите блок от код, който да бъде изпълнен, ако дадено условие е вярно;
- Използвайте **else**, за да посочите блок от код, който да бъде изпълнен, ако същото условие е невярно;
- Използвайте **else if**, за да посочите ново условие за тестване, ако първото условие е невярно;
- Използвайте **switch**, за да посочите много алтернативни блокове код, които да бъдат изпълнени.

II. ОСНОВНИ ПОНЯТИЯ



- Използвайте **if**, за да посочите блок от код, който да бъде изпълнен, ако дадено условие е вярно;

```
if (condition)
{
    // block of code to be executed if the condition is True
}
```

```
if (20 > 18)
{
    Console.WriteLine("20 is greater than 18");
}
```

20 is greater than 18

II. ОСНОВНИ ПОНЯТИЯ

2

- Използвайте **else**, за да посочите блок от код, който да бъде изпълнен, ако същото условие е невярно;

```
int time = 20;  
if (time < 18)  
{  
    Console.WriteLine("Good day.");  
}  
else  
{  
    Console.WriteLine("Good evening.");  
}
```

Good evening.

В примера отляво, времето (20) е по-голямо от 18, така че условието е False. Поради това преминаваме към условието else и отпечатваме на екрана „Добър вечер“. Ако часът е по-малък от 18, програмата ще отпечата "Добър ден".

II. ОСНОВНИ ПОНЯТИЯ



- Използвайте **else if**, за да посочите ново условие за тестване, ако първото условие е невярно;

```
if (condition1)
{
    // block of code to be executed if condition1 is True
}
else if (condition2)
{
    // block of code to be executed if the condition1 is false and condition2 is True
}
else
{
    // block of code to be executed if the condition1 is false and condition2 is False
}
```

II. ОСНОВНИ ПОНЯТИЯ

2

- Използвайте **else if**, за да посочите ново условие за тестване, ако първото условие е невярно;

```
int time = 22;  
if (time < 10)  
{  
    Console.WriteLine("Good morning.");  
}  
else if (time < 20)  
{  
    Console.WriteLine("Good day.");  
}  
else  
{  
    Console.WriteLine("Good evening.");  
}
```



Good evening.

В примера отляво времето (22) е по-голямо от 10, така че първото условие е False. Следващото условие в оператора else if също е False, така че преминаваме към условието else, тъй като условие1 и условие2 са False - и отпечатваме на екрана „Добър вечер“.

Ако обаче часът беше 14, нашата програма щеше да изпише „Добър ден“.

II. ОСНОВНИ ПОНЯТИЯ

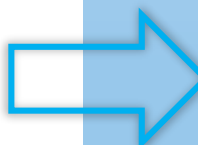
2

- **Съкратена форма на условен оператор If...Else (троичен оператор)** - Съществува и съкратено if else, което е известно като троичен оператор, защото се състои от три операнда. Може да се използва за замяна на няколко реда код с един ред. Често се използва за замяна на прости изрази if else:

```
variable = (condition) ? expressionTrue : expressionFalse;
```

Вместо да пишем дълъг код

```
int time = 20;  
if (time < 18)  
{  
    Console.WriteLine("Good day.");  
}  
else  
{  
    Console.WriteLine("Good evening.");  
}
```



Можем да използваме кратка версия

```
int time = 20;  
string result = (time < 18) ? "Good day." : "Good evening.";  
Console.WriteLine(result);
```

II. ОСНОВНИ ПОНЯТИЯ

2

- **Switch** - Използвайте командата `switch`, за да изберете един от много кодови блокове, които да бъдат изпълнени.

```
switch(expression)
{
    case x:
        // code block
        break;
    case y:
        // code block
        break;
    default:
        // code block
        break;
}
```

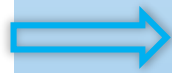
Ето как работи конструкторът със `switch`:

- Изразът за превключване (`expression`) се оценява веднъж;
- Стойността на израза се сравнява със стойностите на всеки случай;
- Ако има съвпадение, асоциираният блок код се изпълнява;
- Ако няма открито съвпадение се изпълнява конструкторът на **default**;
- След изпълнение на коя да е от конструкциите следва прекъсване на конструктора `switch` – това става с ключовата дума **break**

II. ОСНОВНИ ПОНЯТИЯ



```
int day = 4;
switch (day)
{
    case 1:
        Console.WriteLine("Monday");
        break;
    case 2:
        Console.WriteLine("Tuesday");
        break;
    case 3:
        Console.WriteLine("Wednesday");
        break;
    case 4:
        Console.WriteLine("Thursday");
        break;
}
```



Thursday

II. ОСНОВНИ ПОНЯТИЯ

2

12. Цикли в C#

Циклите могат да изпълнят блок от код, докато се достигне определено условие. Циклите са удобни, защото спестяват време, намаляват грешките и правят кода по-четлив.

- **While** – преминава през блок от код, докато дадено условие е True:

```
while (condition)
{
    // code block to be executed
}
```

```
int i = 0;
while (i < 5)
{
    Console.WriteLine(i);
    i++;
}
```



0
1
2
3
4

В примера от дясно кодът в цикъла ще се изпълнява отново и отново, докато променлива (i) е по-малка от 5:

II. ОСНОВНИ ПОНЯТИЯ

2

- **Do/While** – Цикълът do/while е вариант на цикъла while. Този цикъл ще изпълни кодовия блок веднъж, преди да провери дали условието е вярно, след което ще повтаря цикъла, докато условието е вярно.

```
do
{
    // code block to be executed
}
while (condition);
```

Примерът от дясно използва do/while цикъл. Цикълът винаги ще се изпълнява поне веднъж, дори ако условието е невярно, тъй като кодовият блок се изпълнява преди условието да бъде тествано:

```
int i = 0;
do
{
    Console.WriteLine(i);
    i++;
}
while (i < 5);
```



0
1
2
3
4

II. ОСНОВНИ ПОНЯТИЯ

2

- **for** – Когато знаете точно колко пъти искате да преминете през блок от код, използвайте цикъла for вместо цикъл while:

```
for (statement 1; statement 2; statement 3)
{
    // code block to be executed
}
```

```
for (int i = 0; i <= 10; i = i + 2)
{
    Console.WriteLine(i);
}
```



0
2
4
6
8
10

Обяснение на примера:

statement1 - задава променлива преди началото на цикъла (int i = 0).

statement2 - дефинира условието за изпълнение на цикъла (i трябва да е по-малко от 5) - Ако условието е вярно, цикълът ще започне отначало, ако е невярно, цикълът ще приключи.

statement3 - увеличава стойност (i++) всеки път, когато кодovият блок в цикъла е изпълнен.

II. ОСНОВНИ ПОНЯТИЯ

2

- **Foreach** – използва се много често за преминаване през елементи в масив:

```
foreach (type variableName in arrayName)
{
    // code block to be executed
}
```

```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
foreach (string i in cars)
{
    Console.WriteLine(i);
}
```



Volvo
BMW
Ford
Mazda

II. ОСНОВНИ ПОНЯТИЯ

2

- **Прекъсване и възстановяване работата на цикли (Break и Continue)** – **Break** се използва за „излизане“ от цикъла, а командата **Continue** се използва когато искаме да се прекъсне една итерация (в цикъла), ако възникне определено условие, и цикъла да продължава със следващата итерация.

```
for (int i = 0; i < 10; i++)  
{  
    if (i == 4)  
    {  
        break;  
    }  
    Console.WriteLine(i);  
}
```

0
1
2
3

Този пример излиза от цикъла,
когато i е равно на 4

```
for (int i = 0; i < 10; i++)  
{  
    if (i == 4)  
    {  
        continue;  
    }  
    Console.WriteLine(i);  
}
```

0
1
2
3
5
6
7
8
9

Този пример пропуска стойността
на 4

13. Масиви в C#

Масивите се използват за съхраняване на множество стойности в една променлива, вместо да се декларират отделни променливи за всяка стойност.

```
string[] cars;
```

Декларация на масив

С горния пример декларираме променлива, която съдържа масив от низове.

За да вмъкнем стойности в него, можем да използваме литерал на масив - поставете стойностите в списък, разделен със запетая, във фигурни скоби:

```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

За да създадете масив от цели числа, можете да напишете:

```
int[] myNum = {10, 20, 30, 40};
```

II. ОСНОВНИ ПОНЯТИЯ



- Достъп до елементите на масив:

```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};  
Console.WriteLine(cars[0]);
```

⇒ Volvo

- Промяна на елемент от масив:

```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};  
cars[0] = "Opel";  
Console.WriteLine(cars[0]);
```

⇒ Opel

- Определяне броя на елементите в масив - За да разберете колко елемента има един масив, използвайте свойството **Length**:

```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};  
Console.WriteLine(cars.Length);
```

⇒ 4

II. ОСНОВНИ ПОНЯТИЯ

2

- Други начини за създаване на масив:

```
// Create an array of four elements, and add values later
```

```
string[] cars = new string[4];
```

```
// Create an array of four elements and add values right away
```

```
string[] cars = new string[4] {"Volvo", "BMW", "Ford", "Mazda"};
```

```
// Create an array of four elements without specifying the size
```

```
string[] cars = new string[] {"Volvo", "BMW", "Ford", "Mazda"};
```

```
// Create an array of four elements, omitting the new keyword, and without specifying the size
```

```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

II. ОСНОВНИ ПОНЯТИЯ



- **Преминаване през масиви** - Можете да преминете през елементите на масива с цикъла **for** и да използвате свойството **Length**, за да укажете колко пъти трябва да се изпълнява цикълът.

Следният пример извежда всички елементи в масива **cars**:

```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (int i = 0; i < cars.Length; i++)
{
    Console.WriteLine(cars[i]);
}
```



Volvo
BMW
Ford
Mazda

II. ОСНОВНИ ПОНЯТИЯ



- **Сортиране на масиви** - Има различни налични методи за сортиране на масиви, например **Sort()**, който сортира масив по азбучен ред или във възходящ ред:

```
// Sort a string
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
Array.Sort(cars);
foreach (string i in cars)
{
    Console.WriteLine(i);
}
```



BMW
Ford
Mazda
Volvo

```
// Sort an int
int[] myNumbers = {5, 1, 8, 9};
Array.Sort(myNumbers);
foreach (int i in myNumbers)
{
    Console.WriteLine(i);
}
```



1
5
8
9

II. ОСНОВНИ ПОНЯТИЯ

2

- Други полезни методи за масиви - като Min, Max и Sum, могат да бъдат намерени в пространството на имената **System.Linq**:

```
static void Main(string[] args)
{
    int[] myNumbers = {5, 1, 8, 9};
    Console.WriteLine(myNumbers.Max()); // returns the largest value
    Console.WriteLine(myNumbers.Min()); // returns the smallest value
    Console.WriteLine(myNumbers.Sum()); // returns the sum of elements
}
```