



## 2.Основи на езика C#



# СЪДЪРЖАНИЕ

- ЧАСТ 1

- Принципи при дизайна на езика C#
- Програма на C#
- Вход и изход от конзолата
- Типове данни в C#
  - Типове по референция
  - Типове по стойност
  - Примитивни типове
- Предаване на параметри



# СЪДЪРЖАНИЕ

- Константи
- Var
- Dynamic
- Boxing
- Unboxing
- ЧАСТ 2
  - 9. Оператори
  - 10. Програмни конструкции
  - 11. Особености за C#
    - C# 2.0
    - C# 3.0
    - C# 4.0



# Принципи при дизайна на езика C#

- Компонентно Ориентиран - софтуерът се изгражда чрез съединяване на отделни компоненти и описание на взаимодействието между тях
- Обектно Ориентиран език - гради се на основните принципи на ООП . В .NET Framework всички типове наследяват системният тип `System.Object` , в следствие , на което всички данни се третират като обекти



# Принципи при дизайна на езика C#

- Силна типизираност и типова безопасност
  - Използват се силно типизирани указатели – референции.
  - Безопасно преобразуване на типове
- Статична типизираност



# Принципи при дизайна на езика C#

- Автоматично управление на паметта и ресурсите
- Заделянето и използването на паметта се управлява автоматично от Common Language Runtime
- Механизъм за управление на изключения



# Програма на C#

- Дефиниция - Съвкупност от дефиниции на класове структури и други типове
- Namespaces – класовете логически се разполагат в namespaces
- Методът Main() – входната точка на една програма. Във всяка програма някой от класовете съдържа този метод



# Програма на C#

- Пример
  - Конзолно приложение
  - Демонстрация на типова безопасност
  - Демонстрация на изключение



# Вход и изход от конзолата

- Осъществява се чрез класа `System.Console`
- Предоставя основна функционалност за четене и писане на экрана
- Ако конзолата не съществува няма ефект
- Методи за вход и изход



# Типове данни в C#

- Всички типове наследяват от `System.Object`, и за това имат гарантиран минимален набор от функции:
  - `GetHashCode()`;
  - `GetType()`;
  - `ToString()`;
  - `Equals()`;
- В общия случай обекти се създават с оператора `new`



# Типове данни в C#

- Преобразуване на типове
  - CLR позволява преобразуване на един тип до себе си или до който и да е негов базов тип (*implicit type cast*).
  - Преобразуване на тип към негов наследник става изрично и не винаги успешно (*explicit type cast*)



# Типове данни в C#

- **is** оператор - проверява дали обектът е съвместим с дадения тип и връща `true` или `false`. Никога не предизвиква изключение
- **as** оператор - проверява дали обектът е съвместим с дадения тип и връща референция към обекта, ако да или `null` в противен случай
- Примери за преобразуване



# ТИПОВЕ ПО РЕФЕРЕНЦИЯ

- Представляват типово-обезопасени указатели към обект
- Съхраняват се в динамичната памет (*managed heap*)
- При извикване на метод се предават по референция (по адрес)
- Унищожават се автоматично от garbage collector на CLR, когато не се използват или са извън обхват



# ТИПОВЕ ПО РЕФЕРЕНЦИЯ

- Могат да приемат стойност `null`
- Възможно е няколко променливи да сочат към един и същ обект от референтен тип
- При присвояване на референтни типове се копира само референцията



# ТИПОВЕ ПО СТОЙНОСТ

- Наследяват `System.ValueType`
- Съхраняват стойността си в стека за изпълнение на програмата
- Не приемат стойност `null`, защото не са референции



# ТИПОВЕ ПО СТОЙНОСТ

- Унищожават се при излизане на съответната променлива от обхват
- При извикване на метод се предават по стойност, записана в стека
- При присвояване на стойностни типове се копира тяхната стойност



# ПРИМИТИВНИ ТИПОВЕ

- Типове , които се поддържат от компилатора
- Отговарят на типове от FCL
- По стойност
  - byte, sbyte, int, uint, long, ulong – цели числа
  - float(Single), double - реални числа с плаваща запетая
  - decimal – реални числа с фиксирана точност



# ПРИМИТИВНИ ТИПОВЕ

- `char` – символ
- `bool` – булев тип
- Структури (`DateTime`) и изброени типове
- По референция
  - `object`
  - `string`
  - класове



# ПРИМЕР

```
public static void ValueTypeDemo()
{
    SomeRef r1 = new SomeRef(); // заделя памет в динамичната памет
    SomeVal v1 = new SomeVal(); // заделя памет в стека

    r1.x = 5; // пренасочване на референцията
    v1.x = 5; // промяна на стойността

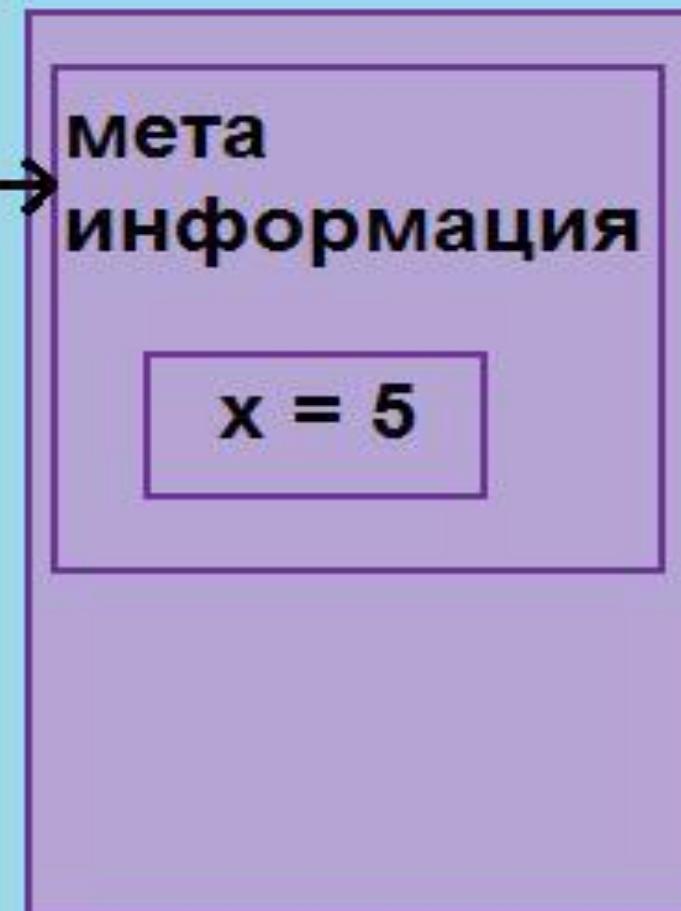
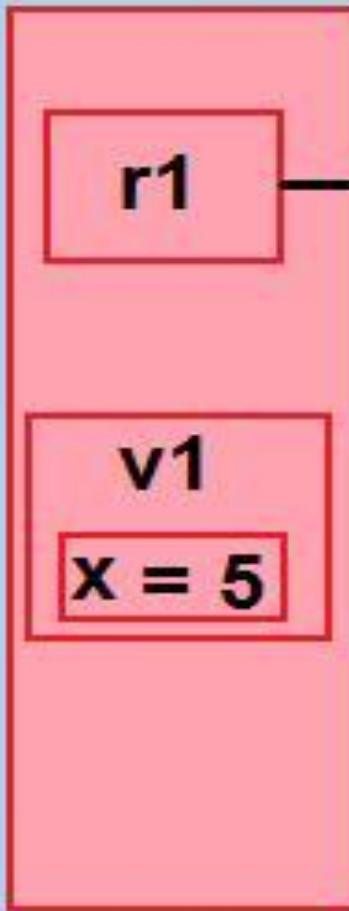
    Console.WriteLine(r1.x); // 5
    Console.WriteLine(v1.x); // 5
}
```



# ПРИМЕР

стек

динамична памет





# ПРИМЕР

```
public static void ValueTypeDemo()
{
    SomeRef r2 = r1; // Копира само референцията
    SomeVal v2 = v1; // Заделя памет в стека и копира и член-данныте

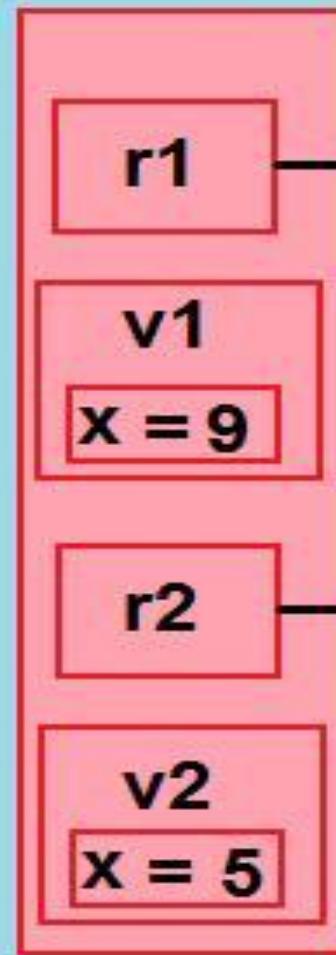
    r1.x = 8; // променя r1.x и r2.x
    v1.x = 9; // променя само v1.x

    Console.WriteLine(r1.x); // 8
    Console.WriteLine(v1.x); // 9
    Console.WriteLine(r2.x); // 8
    Console.WriteLine(v2.x); // 5
}
```

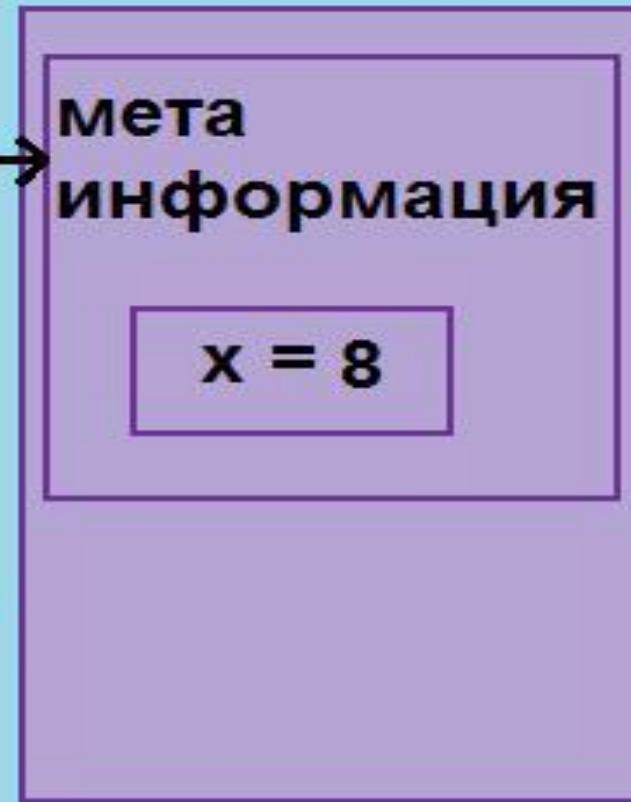


# ПРИМЕР

стек



динамична памет





# ПРЕДАВАНЕ НА ПАРАМЕТРИ

- Параметрите на методите могат да се предават по няколко начина
  - `in` (по подразбиране)
    - предаване по стойност за `value types`
    - предаване по референция за `reference types`



# ПРЕДАВАНЕ НА ПАРАМЕТРИ

## – ref

- предаване по адрес за value types и по адрес на референцията за reference types
- инициализацията може да се извършва от извикващия метод – достъпът е за четене и писане

## – out

- предаване по адрес за value types и по адрес на референцията за reference types
- инициализацията се извършва от извиквания метод, а преди нея достъпът е само за писане



# КОНСТАНТИ

- Дефиниция – символ означаващ стойност, която не се променя
- Compile-time – декларират се със запазената дума const. Задължително се инициализират в момента на декларирането им и не могат да се променят след това. По време на компилация се заместват със стойността си.



# КОНСТАНТИ

- Runtime – декларират се като полета с модifikатора `readonly`. Полета на типа, които са само за четене. Инициализират се по време на изпълнение и повече не се променят



# VAR (Type Inference)

- Тип по подразбиране
- Променливи от тип по подразбиране са силно типизирани
- Компилаторът определя типа

```
var i = 10; // implicitly typed
int i = 10; // explicitly typed
```
- Определя променлива, чиито тип се определя в зависимост от резултата на израза , който е използван за инициализацията й по време на създаването ѝ.



# BOXING AND UNBOXING

- Boxing – на референтен тип се присвоява стойностен
  - `object o = new object ();`
  - `int a = 0;`
  - `o = a;`
- Unboxing – процесът на извлечане на опакована стойност от динамичната памет
  - `int b = (int)o;`



# cool C#



# Оператори 1/5 (основни)

**x = y = z**

->

**x.y**

**(T)x**

**x[i]**

**x as T**

**new T**

**sizeof(T)**



# Оператори – 2/5 (аритметични)

$2 * 2$

$9 + 9$

checked

$x *= 3$

$x += 10$

{

$4 / 3$

$11 - 4$

int.MaxValue + 1

$5.0 / 3.0$

$x -= 12$

unchecked

$x /= 6$

$x++$

{

$7 \% 2$

$++x$

int.MaxValue + 1

$x \%= 8$

$x--$

}

$--x$



# Оператори – 3/4 (битови)

2 & 2

x &= 3

4 | 3

x |= 6

7 ^ 2

x ^= 8

9 << n

x <<= 10

11 >> 4

x >>= 12

~x



# Оператори – 4/5 (логически)

$2 > 2$

$x \&\& y$

$x \geq 3$

$x || y$

$4 < 3$

$!a$

$x \leq 6$

$x ? \text{ifTrue} : \text{ifFalse}$

$7 == 7$

$x ?? y [?? z]$

$x != 8$

$\text{true}$   
 $\text{false}$



# Оператори 5/5 (дефиниране)

- Операторите са **статични методи** и можем да си ги предефинираме.
- Предефинираме оператори само ако са в единия от участващите **класове**.
- Могат да се предефинират само някои оператори

**+ , - , \* , / , % , & , | , << , >> , + , - , ! , ~ , ++ , -- , true ,  
false , == , != , < , > , <= , >= , cast**



# ПРОГРАМНИ КОНСТРУКЦИИ

- Елементарни програмни конструкции – най-простите елементи на програмата
  - Присвояване -  
 $<\text{променлива}> = <\text{израз}>$
  - Извикване на метод
  - Създаване на обект
- Съставни – състоят се от няколко други конструкции в блок



# Конструкции за управление

## If - else

- Условни конструкции

if ( x ) { A }

if ( x ) { A } else { B }

if ( x ) { A } else if ( y ) { B } else { C }



# Конструкции за управление switch

```
switch (action)
{
    case "Add":
    case "AnotherAdd":
        // <ADD>
        break;
    case "Remove":
        // <REMOVE>
        break;
    default:
        // ...
        break;
}
```



# Програмни конструкции за управление 1

```
for (int i = 0; i < 5; i++)  
{  
    //...  
}
```



# Програмни конструкции за управление 1

```
for (  
    int i = 0, j=0;  
    i < 5 && j < 5;  
    total += i++ + (j += 2)  
){ }
```



# Програмни конструкции за управление 1

**foreach(var item in list)**

{

// използваме item.

// не променяме list!

}



# Програмни конструкции за управление 1

**while( boolExp ) { A }**

**do { A } while ( boolExp )**



# Програмни конструкции за управление

- Конструкции за переход в цикли:
  - break
  - continue



# Програмни конструкции за управление

- Конструкции за переход
  - goto
  - return

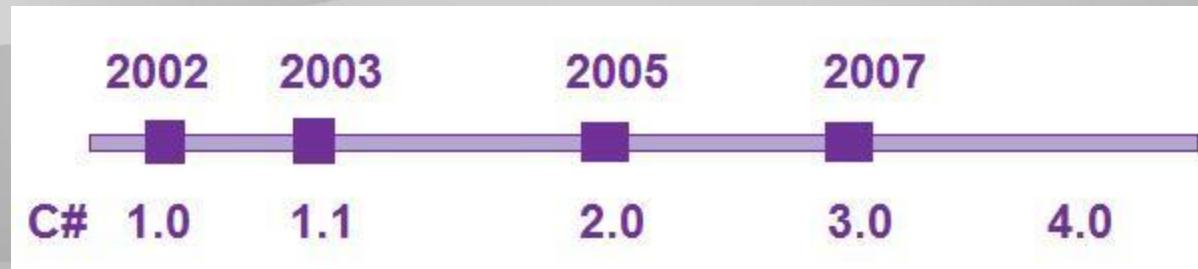


# Програмни конструкции за управление

- Конструкции за управление на изключенията
  - try – catch- finally
  - try-catch
  - try-finally
  - throw
- Специални конструкции
  - lock
  - fixed
  - unsafe



# Особенности за C#





# Характеристики на C# 2.0

- Generics
  - `List<DateTime> dateTimeList = new List<DateTime>();`
  - `dateTimeList.Add(DateTime.Now);`
  - `dateTimeList.Add(DateTime.MinValue);`
- Преизползваемост на алгоритми, които работят върху различни видове обекти
- Безопасност на типовете



# Характеристики на C# 2.0

- Анонимни методи
- Итератори
  - За реализацията се използва `yield` – ключова дума, която създава итератори



# Характеристики на C# 2.0

- **Nullable типове**
  - `System.Nullable<T>` - представя правилно стойностите на типа `T`, но предоставя и една допълнителна стойност - `null` за типа
  - Поддръжка на всички оператори



# Характеристики на C# 3.0

- Променливи с тип по подразбиране
  - Компилаторът предполага типа на променливата по време на изпълнение
  - Променливата трябва да е локална и инициализирана
- Extension методи
  - Позволява се добавяне на нови методи към вече дефинирани класове



# Характеристики на C# 3.0

- Ламбда изрази
  - Още по-опростени анонимни методи
  - Компилаторът предполага типа и модifikаторите на параметрите
- Анонимни типове
  - Автоматични декларации на наредени двойки типове, проекция на типове
  - `var person = new { name = "Ann", age = 15 };`



# Характеристики на C# 3.0

- **Масиви с тип по подразбиране**
  - Масиви, чиито типове се определят от елементите, с които са инициализирани (елементите са от подобни типове)



# Характеристики на C# 3.5

- **Language Integrated Query (LINQ)**
  - Разширява .NET езиците със синтаксис за правене на заявки към данни.
  - Методите за работа с данни могат да се извикват чрез C# ключови думи
  - Връщат готови обекти
  - Работи с различни видове данни
    - LINQ to XML
    - LINQ to SQL
    - LINQ to Objects



# Характеристики на C# 4.0

- Нов псевдо-тип `dynamic`.
- Незадължителни параметри и параметри със стойност по подразбиране
- Именувани параметри
- Еквивалентност на типове



# Въпроси ?