

1

Osnove C#-a

2016/17.04

Microsoft .NET Framework

2

- nastao s idejom iste osnovice za izradu lokalnih i Internet aplikacija
 - začetak krajem 90-tih, prva verzija 2002. g
 - trenutna verzija 4.6.2.
- neovisnost o jeziku
 - jezici Visual Basic .NET, Visual C++ .NET, C# i drugi
 - zajednička knjižnica osnovnih razreda Base Class Library (BCL)
 - Framework Class Library (FCL) kao nadgradnja
 - zajednički, opći tipovi podataka Common Type System (CTS)
 - CLS (Common Language Specification) – podskup CTS-a zajednički za sve .NET jezike
 - zajednički pogon programa - Common Language Runtime (CLR)
- neovisnost o platformi ... na kojoj postoji CLR
 - iako tako zamišljen, .NET je namijenjen uglavnom Windows platformi
 - Mono – (djelomična) implementacija .NET Frameworka za Linux

.NET Core

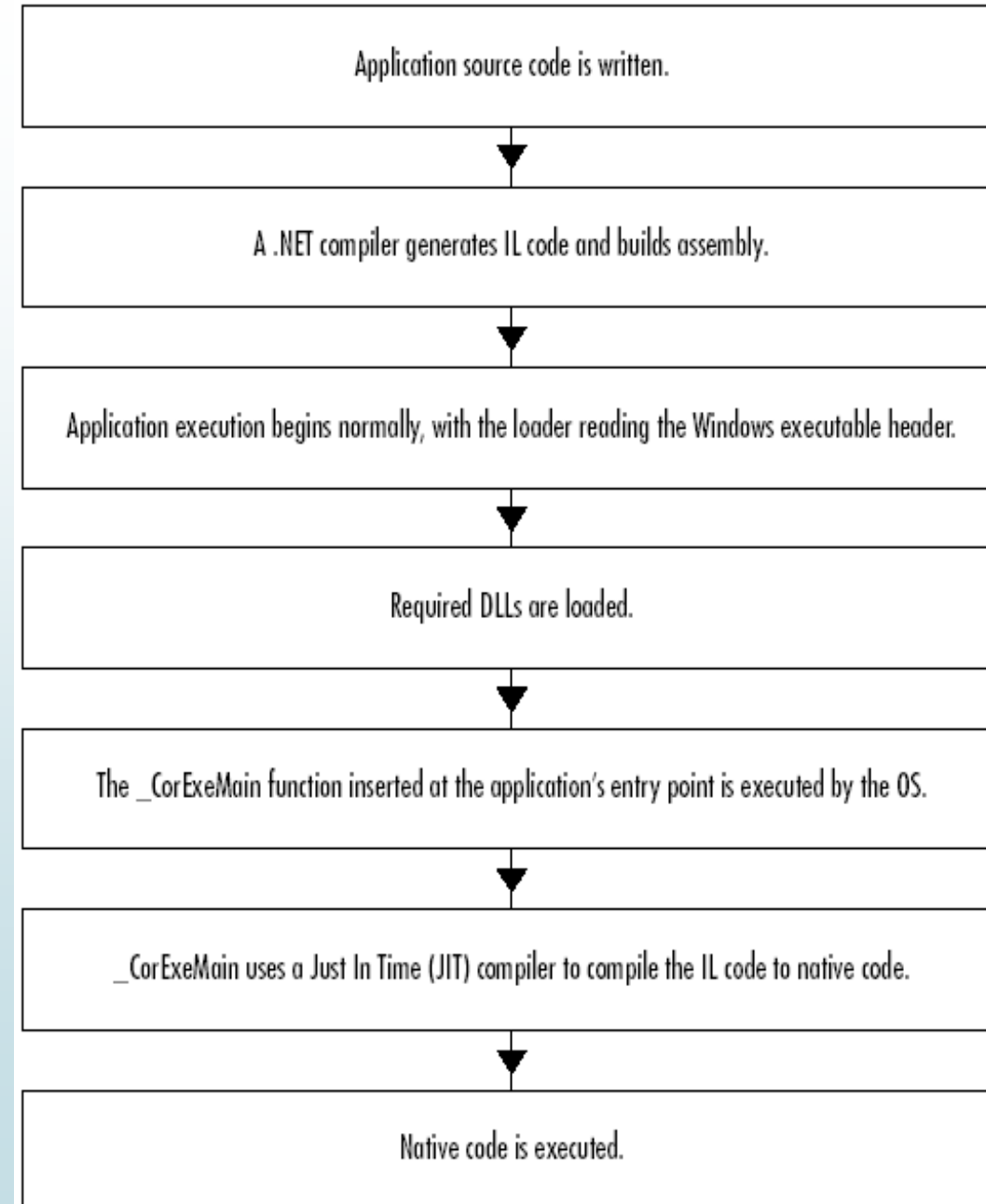
3

- Nije podskup .NET Frameworka, ali dijeli dio funkcionalnosti
 - Standard propisan kao .NET Standard (trenutno 1.6)
 - Podjela na modularne pakete dohvatljive korištenjem alata NuGet
- Dostupan za Windowse, OSX i razne (ne nužno sve) distribucije Linuxa
 - .NET Core 1.1, ožujak 2017.
 - CoreCLR
- Što trenutno može .NET Core?
 - konzolne aplikacije
 - web aplikacije i web servisi pisani u ASP.NET Core-u
 - Objektno-relacijsko preslikavanje prema nekoliko tipova sustava za upravljanjem bazama podataka korištenjem alata Entity Framework Core
- Što ne može .NET Core?
 - ne postoji podrška za samostalne (desktop) aplikacije
 - velik broj paketa trećih strana još nije prilagođen za .NET core

Microsoft .NET Core/Framework

4

- C# prevoditelj
 - prevodi izvornik kod u C#-u (ili nekom drugom jeziku) u poseban međujezik MSIL
- MS Intermediate Language (MSIL)
 - MSIL se izvodi u virtualnom stroju, a ne izravno na procesoru računala
- Common Language Runtime (CLR)
 - zajednički pogon programa
 - interpretira MSIL naredbe
 - koristi JIT prevodilac
- Just-In-Time compiler
 - pri prvom pokretanju programa prevodi MSIL u strojni kod
- Asemblij (Assembly)
 - skup prevedenih razreda



➤ Jezik C# - Anders Hejlsberg et al.

- vođen događajima (event-driven),
- objektno usmjeren (object-oriented),
- otvoren mreži (network-aware),
- vizualan (visual) – interaktivno sučelje razvojne okoline i aplikacija

➤ Trenutna verzija C# 7.0

Stvaranje prvog programa (.NET Core)

6

➤ Komanda linija

- u nekoj mapi pokrenuti `dotnet new console NazivPrograma`
- programski kod urediti u proizvoljnom uređivaču teksta
- `dotnet restore` (dovlači pakete uključene unutar projekta)
- `dotnet run`

➤ Visual Studio Code

- u nekoj mapi pokrenuti `dotnet new console NazivPrograma`
- Visual Studio Code → Open Folder
- Potvrdi dohvat paketa, a zatim F5

➤ Visual Studio 2015

- File → New project → .NET Core → Console Application
- Unijeti naziv i lokaciju projekta
- F5 (Debug) ili CTRL+F5 (Execute)

➤ Identifikatori

- nazivi razreda, varijabli, ... koje određuje programer
- sastoje se od slova, znamenki i podcrte, prvi znak mora biti slovo

➤ Ključne riječi (keywords)

- rezervirane riječi jezika, posebni identifikatori koji u jeziku imaju predefinirano značenje
- definicije objekata, naredbe i direktive za prevođenje

➤ Operatori slični kao u C-u i Javi: <https://msdn.microsoft.com/en-us/library/6a71f45d.aspx>

➤ Programske strukture

- odvajanje naredbi s ;
- blokovi programa - { }

➤ C# razlikuje velika i mala slova!

➤ Komentari

- Retkovni komentar – tekst od znakova `///
do kraja retka`
- Blok komentar – tekst oblika `/* ... */`, može se rasprostirati u više redaka

Struktura C# programa

8

➤ Program

- sadrži jedan ili više razreda

➤ Razred (klasa, class)

- sadrži članove – svojstva (varijable) i postupke

➤ Nema globalnih varijabli i funkcija

- sve se zbiva unutar tijela razreda

➤ Ulazna točka (entry point)

- postupak static void Main

➤ Korištenje knjižnice (using)

- svaki prevedeni razred ima manifest - definiciju sučelja koju mogu koristiti drugi programi, uključujući one napisane u nekom drugom jeziku

➤ Primjer Osnove\Pozdrav

```
// korištenje knjižnice
using System;

class Pozdrav// zaglavlje razreda
// tijelo razreda
{
    // metoda Main
    static void Main(string[] args)

    // blok naredbi
    {
        // naredba WriteLine
        Console.WriteLine("Pozdrav!");
    }
}
```


Prostor imena (namespace)

9

- Izvorni .NET kod je organiziran u prostore imena (namespaces)
 - C# program se može sastojati od više datoteka
 - Svaka datoteka sadrži jedan ili više prostora imena
 - Isti prostor imena može se protezati (deklarirati) u više datoteka.
- Prostor imena
 - Prostor imena sadrži definicije razreda, struktura, sučelja, pobrojanih tipova, delegata i deklaraciju drugih prostora imena
 - Prostor imena u kojem su deklarirani svi ostali prostori imena i tipovi podataka u .NET Frameworku je System
 - Ako se u programu eksplicitno ne definira prostor imena, C# kod je sadržan u globalnom imeniku (*global namespace*)
- Koncept prostora imena omogućuje postojanje istih imena u različitim prostorima imena
 - Jedinstvenost imena tipova u prostoru imena i samog prostora imena osigurana je preko tzv. potpunih imena (*fully qualified names*)

Definiranje prostora imena

10

➤ Prostor imena N1 je član globalnog prostora imena

➤ Njegovo puno ime je N1

➤ Prostor imena N2 je član prostora N1

➤ Njegovo puno ime je N1.N2

➤ Razred C1 je član od N1

➤ Njegovo puno ime je N1.C1

➤ Ime razreda C2 se pojavljuje dva puta, ali je njegovo puno ime jedinstveno

➤ N1.C1.C2 ili

➤ N1.N2.C2.

```
namespace N1{           // N1
    class C1{           // N1.C1
        class C2{       // N1.C1.C2
        }
    }
    namespace N2{       // N1.N2
        class C2{       // N1.N2.C2
        }
    }
}
```

Korištenje prostora imena

11

➤ Referenciranje koda u prostoru imena obavlja se

- kvalificiranim imenom (prostora imena i identifikator razreda), npr. `System.Console`
- direktivom `using` ili `using static`: eksplicitno se uvode sva imena sadržana u prostoru imena ili svi potupci nekog razred
 - `using System`; referencira imenik u kojem je definiran razred `Console`, pa se `Console` može direktno koristiti
- zamjenskim imenom – koristi se za definiranje skraćenih naziva i uklanjanje neodređenosti kad u različitim imenicima postoje razredi istog imena.

```
using System; // navod imenika
using stdout = System.Console; // alias razreda
using static System.Console;
class Pozdrav {
    static void Main(string[] args) {
        System.Console.WriteLine("Kvalificirano!");
        Console.WriteLine("Izravno!"); // radi using System
        stdout.WriteLine("Zamjenski!"); // radi using stdout =
        WriteLine("Zbog static using System.Console");
    }
}
```

Ključne riječi

12

- Sve ključne riječi definirane su malim slovima

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	get
goto	if	implicit	in	int
interface	internal	is	lock	long
namespace	new	null	object	operator
out	override	params	private	protected
public	readonly	ref	return	sbyte
sealed	set	short	sizeof	stackalloc
static	string	struct	switch	this
throw	true	try	typeof	uint
ulong	unchecked	unsafe	ushort	using
virtual	void	volatile	while	

- [https://msdn.microsoft.com/en-us/library/x53a06bb\(v=vs.140\).aspx](https://msdn.microsoft.com/en-us/library/x53a06bb(v=vs.140).aspx)

Ključne riječi ovisno o kontekstu

13

➡ <https://msdn.microsoft.com/en-us/library/x53a06bb.aspx>

add	alias	ascending	async	await
descending	dynamic	from	get	global
group	into	join	let	orderby
partial	remove	select	set	value
var	where	yield		

➤ Čitanje standardnog ulaza

- `Console.ReadLine()` – čita niz znakova
- `Console.Read()` – čita znak

➤ Pisanje na standardni izlaz

- `Console.Write("Pozdrav! ");` piše argument(e) i ostaje u istom retku
- `Console.WriteLine("Pozdrav! ");` piše argument(e) i znak za skok u novi redak
- `Console.WriteLine("x = {0}, y = {1}", x, y);` ispis više argumenata
- `Console.WriteLine($"x = {x}, y = {y}");` ispis vrijednosti varijabli

➤ Primjeri primjene slijede u narednim programima

➤ Za više informacija pogledati dokumentaciju

Formatirani ispis, numerički formati

15

Character	Description	Examples	Output
C or c	Currency	<code>Console.Write("{0:C}", 2.5);</code> <code>Console.Write("{0:C}", -2.5);</code>	\$2.50 (\$2.50)
D or d	Decimal	<code>Console.Write("{0:D5}", 25);</code>	00025
E or e	Scientific	<code>Console.Write("{0:E}", 250000);</code>	2.500000E+005
F or f	Fixed-point	<code>Console.Write("{0:F2}", 25);</code> <code>Console.Write("{0:F0}", 25);</code>	25.00 25
G or g	General	<code>Console.Write("{0:G}", 2.5);</code>	2.5
N or n	Number	<code>Console.Write("{0:N}", 2500000);</code>	2,500,000.00
X or x	Hexadecimal	<code>Console.Write("{0:X}", 250);</code> <code>Console.Write("{0:X}", 0xffff);</code>	FA FFFF

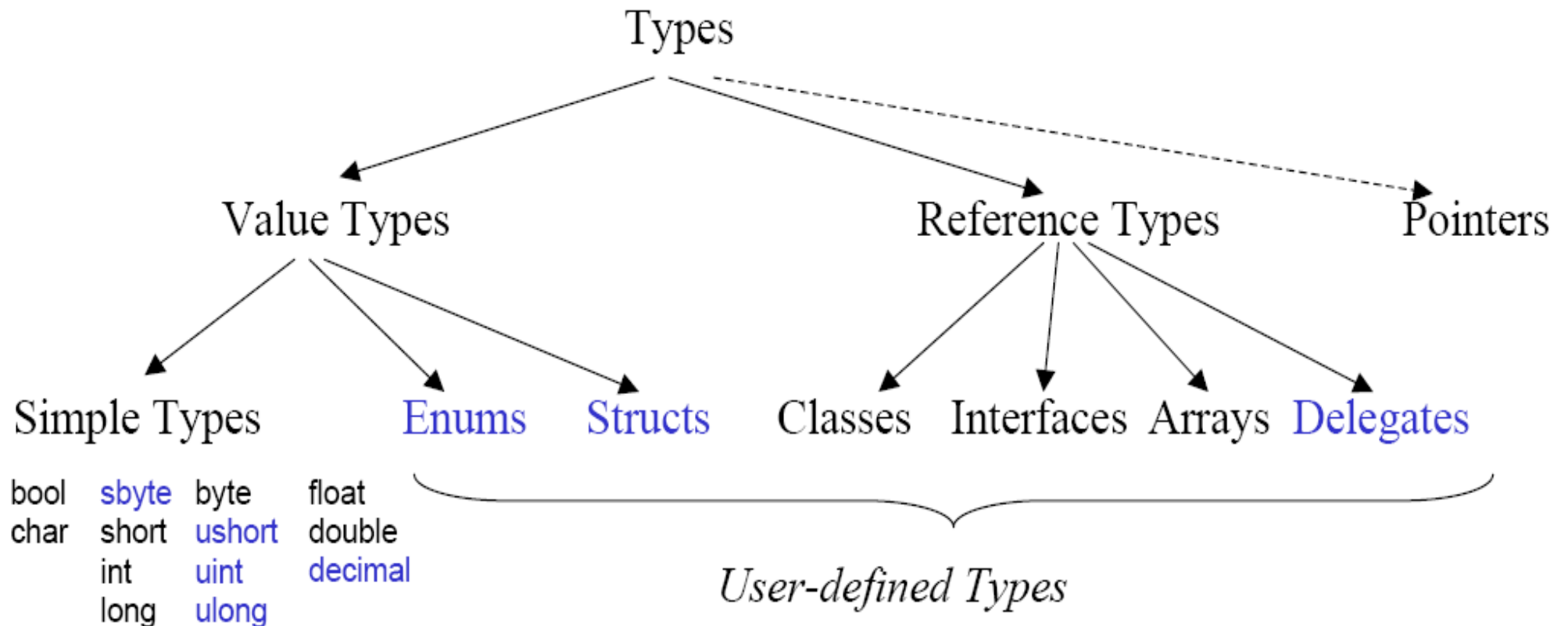
Tipovi podataka

16

➤ Svi tipovi izvode se iz osnovnog tipa `System.Object`

➤ "sve je objekt"

➤ primjeri: `3.ToString()` ; `"abc".ToUpper()` ;



➤ Osnovne kategorije tipova podataka:

- Vrijednosti (value types) – sadrže podatke
 - osnovni tipovi (int, float, char...), enum, struct
- Reference (reference types) – pokazivači (pointers)
 - razredi (class types),
 - sučelja (interface types),
 - delegati (delegate types),
 - polja (array types) ,
 - znakovni nizovi (strings)

➤ Varijable i pridruživanje vrijednosti

- varijabla - imenovana memorijska lokacija određenog tipa i poznate veličine te vrijednosti koja se u njoj nalazi a koja se može promijeniti
- prije uporabe (zadavanja ili korištenja vrijednosti) varijablu treba deklarirati

```
int broj1;      // <type> <variable-name>;  
broj1 = 45;     // <variable-name> = <exp>
```

Ugrađeni tipovi podataka

18

C# Type	.Net Framework (System) type	Broj bajtova	Interval brojeva
sbyte	System.Sbyte	1	-128 do 127
short	System.Int16	2	-32768 do 32767
int	System.Int32	4	-2147483648 do 2147483647
long	System.Int64	8	-9223372036854775808 do 9223372036854775807
byte	System.Byte	1	0 do 255
ushort	System.UInt16	2	0 do 65535
uint	System.UInt32	4	0 do 4294967295
ulong	System.UInt64	8	0 do 18446744073709551615
float	System.Single	4	$\pm 1.5 \times 10^{-45}$ do $\pm 3.4 \times 10^{38}$ 7 točnih znamenki
double	System.Double	8	$\pm 5.0 \times 10^{-324}$ do $\pm 1.7 \times 10^{308}$ 15 točnih znamenki
decimal	System.Decimal	12	$\pm 1.0 \times 10^{-28}$ do $\pm 7.9 \times 10^{28}$ 28 točnih znamenki
char	System.Char	2	Unicode znak (16 bit)
bool	System.Boolean	1	true ili false

Konverzija tipova

19

➤ Implicitna konverzija – bez navođenja operatora konverzije

➤ Standardno int->double, int->long, ...

➤ Eksplicitna konverzija – Primjer:  Osnove\Konverzija

➤ operator konverzije tipa (cast)

```
int c = (int) 4.5;
```

➤ ToString()

```
int a = 154;  
string s = a.ToString();
```

➤ Parse()

```
int c = Int32.Parse(s);
```

➤ razred Convert

```
decimal d = Convert.ToDecimal(c);
```

Boxing/Unboxing

20

➤ Boxing

- pretvorba nekog vrijednosnog tipa podataka u referencijski tip object
- alocira instancu i kopira vrijednost u novostvoreni objekt

➤ Unboxing

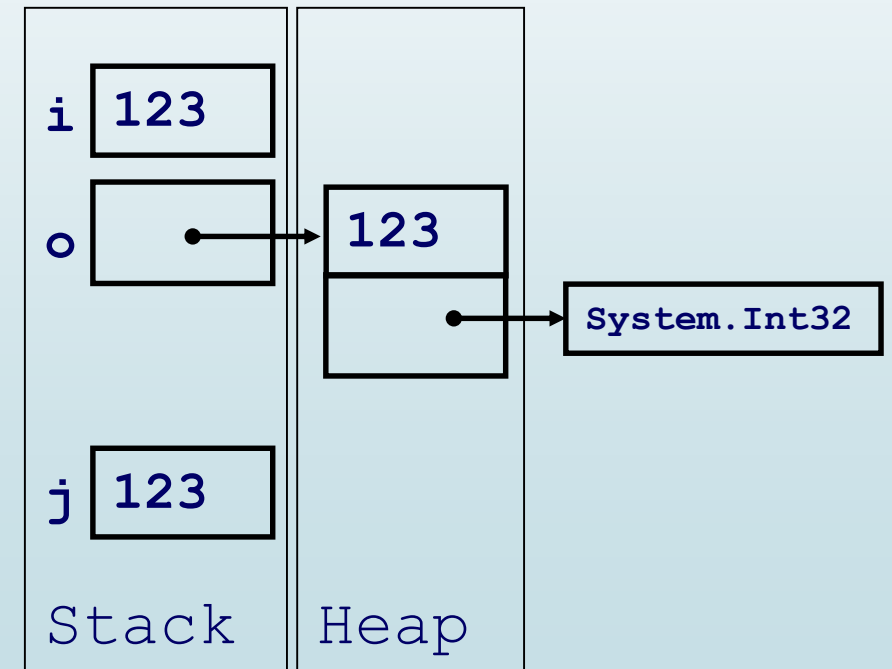
- izdvajanje vrijednosnog tipa podataka iz tipa object
- kopiranje vrijednosti (objekta) s reference

```
int i = 123; // A value type
```

```
object o = i; // Boxing
```

```
int j = (int) o; // Unboxing
```

➤ Primjer Osnove\Boxing



Naredbe za upravljanje programskim tokom

21

➤ Selekcija

- if (expression) statement1 [else statement2]
- switch (expression) { case *constant-expression*: statement jump-statement [default: statement jump-statement] } (nema propadanja kao u C-u)

➤ Petlje

- while (expression) statement
- for ([initializers]; [expression]; [iterators]) statement
- **foreach** (*type identifier in expression*) statement
 - *expression* se ne smije mijenjati za vrijeme *foreach* petlje i nije moguće napisati identifier = ...
- do statement while (expression);

➤ Skokovi

- break , continue, goto
 - goto *identifier*;
 - goto case *constant-expression*;
 - goto default;
- return [expression];

```
int[] numbers =  
    {4, 5, 6, 1, 2, 3, -2, -1, 0};  
for(int i=0; i<numbers.length; i++){  
    Console.WriteLine(numbers[i]);  
}  
foreach (int i in numbers){  
    Console.WriteLine(i);  
}
```

Razredi

22

```
class Tocka
```

```
{
```

```
    public int x;
```

```
    public int y;
```

```
    public Tocka(int x, int y){
```

```
        this.x = x;
```

```
        this.y = y;
```

```
    }
```

```
    public int Kvadrant(){
```

```
        if (x > 0){
```

```
            if (y > 0) return 1;
```

```
            else return 4;
```

```
        } else {
```

```
            if (y > 0) return 2;
```

```
            else return 3;
```

```
        }
```

```
    }
```

```
}
```

► Primjer  Osnove \ RazredTocka

Javne varijable

Konstruktor

Instanciranje
objekta

Javni postupak

```
Tocka t = new Tocka(1,-2);  
Console.WriteLine("Točka ({0},{1})  
je u {2}.kvadrantu.", t.x, t.y,  
t.Kvadrant());
```

Poziv postupka

Pristup javnoj
varijabli

Razredi i članovi razreda

23

➤ U jeziku C# razredi se sastoje od članova (members):

- Konstante (constants)
- Atributi (fields)
- Konstruktori (constructors)
- Finalizatori (finalizers)
- Postupci (methods)
- Svojstva (properties)
- Indekseri (indexers)
- Operatori (operators)
- Događaji (events)
- Statički konstruktori (static constructors)
- Ugniježđeni tipovi (nested types)

Razred i objekt

24

- Objekt je jedna instanca razreda
- Novi objekt se stvara operatorom `new` nakon čega slijedi poziv konstruktora
 - razred s početnim postupkom (`Main`) ne mora se eksplicitno instancirati
- `this` – referenca na trenutnu instancu razreda

➤ Modifikatori pristupa razredima i članovima

- `public` – pristup nije ograničen
- `private` - pristup ograničen na razred u kojem je član definiran
- `protected` – pristup ograničen na razred i naslijeđene razrede
- `internal` – pristup ograničen na program u kojem je razred definiran
- `protected internal` - pristup dozvoljen naslijeđenim razredima (bez obzira gdje su definirani) i svima iz programa u kojem je razred definiran

➤ Pretpostavljeni modifikatori

- `internal` za razrede, sučelja, delegate i događaje
- `private` za članske varijable, svojstva i postupke i ugniježdene razrede
- `public` za postupke sučelja i članove enumeracija
 - nije ni dozvoljen drugačiji modifikator

➤ Izvedeni razred ne može imati veću dostupnost od baznog razreda

Ostali značajniji modifikatori

26


► Neki od značajnijih modifikatora


- `abstract` – razred može biti samo osnovni razred koji će drugi nasljeđivati
- `const` – atribut (polja) ili lokalna varijabla je konstanta
- `new` – modifikator koji skriva naslijeđenog člana od člana osnovnog razreda
- `readonly` – polje poprima vrijednost samo u deklaraciji ili pri instanciranju
- `sealed` – razred ne može biti naslijeđen
- `static` – jedini, zajednički član svih instanci razreda (ne kopija nastala s instancom)
- `virtual` – postupak ili dostupni član koji može biti nadjačan u naslijeđenom razredu – (prilikom nadjačavanja dodaje se modifikator `override`)

► Ostali modifikatori: <http://msdn.microsoft.com/en-us/library/dd469484.aspx>

Statički članovi, konstante i atributi (fields)

27

- `Static` se može primijeniti na attribute, postupke, svojstva, operatore i konstruktore
 - Statički član pripada tipu, a ne instanci.
- Modifikator `const`
 - Vrijednost konstante određena je pri prevođenju i ne može se mijenjati
 - Konstante su po prirodi statičke, pa ih se ne može eksplicitno proglasiti statičkim ni pristupati im preko reference na instancu objekta
- Atributi
 - Varijable kojima se može promijeniti vrijednost pristupom članu
 - Nepromjenjivi atributi – modifikator `readonly`
 - ne dopuštamo promjene, a vrijednosti nisu poznate u vrijeme prevođenja
- Primjer  Osnove \ Razred


- Konstruktor je postupak koji se obavlja pri stvaranju instance
 - Standardni (default) konstruktor i preopterećenje konstruktora s argumentima
 - Ne vraća vrijednost, služi za inicijalizaciju instance
 - Ime mu je jednako imenu razreda
- Primjer  Osnove\Razred

```
Razred r1 =  
    new Razred();  
Razred r2 =  
    new Razred(13, 666);  
Razred r2 =  
    new Razred() {var=4};
```

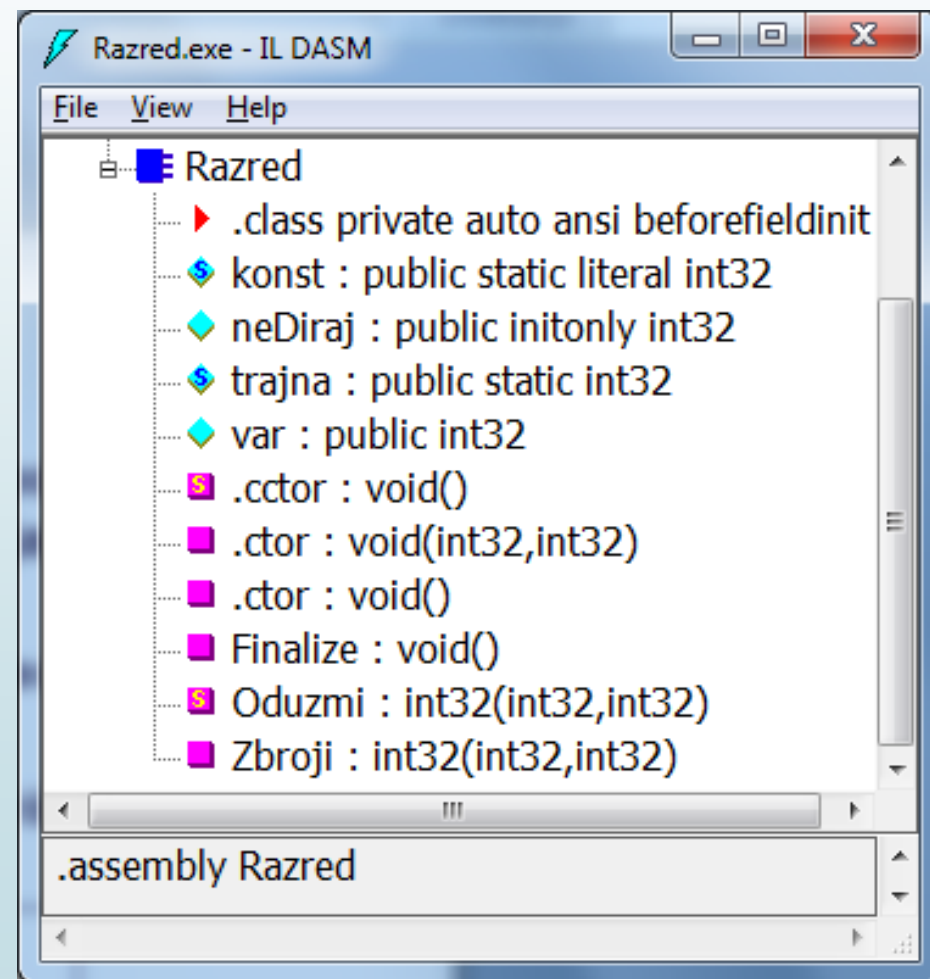
```
class Razred  
{  
    //atributi  
    public readonly int neDiraj;  
    public int var = 0;  
    ...  
    //konstruktori  
    public Razred(){  
        neDiraj = 0;  
    }  
    public Razred(int neDiraj0, int var0)  
    { neDiraj = neDiraj0;  
      var = var0;  
    }  
}
```

Finalizatori (Destruktori)

29

- Finalizator je postupak (Finalize) koji se automatski poziva neposredno prije uništenja objekta od strane sakupljača smeća (Garbage Collector).
 - Piše se u obliku destruktora - jednak je nazivu razreda s predznakom '~'
 - Ne vraća vrijednost
 - Koristi se za brisanje tzv. "unmanaged" resursa (ako su korišteni)
- Primjer  Osnove\Razred

```
//finalizator se piše u obliku destruktora
~Razred() {
    Console.WriteLine("Finalizer");
}
```



Sakupljač smeća

30

- Garbage Collector
- Oslobađa memoriju koja je bila zauzeta referencama koje više ne postoje.
- Sakupljanje smeća obavlja tijekom izvođenja programa.
- Ne može se unaprijed točno odrediti kada će se memorija osloboditi, to jest kada će se finalizator pozvati, ali ga se može (pokušati) potaknuti (metodom `GC.Collect`).
- Sakupljač smeća u .NET-u koristi generacijski model (Generacije 0, 1 i 2). Preživljavanjem čišćenja objekti se sele iz generacije 0 u 1, odnosno iz 1 u 2.


Postupci (metode)

31

➤ Postupak je programska funkcija pridružena razredu.

➤ Skrivanje člana (učahurivanje - enkapsulacija)

➤ Pristup skrivenom članu javnim postupcima

➤ Primjer:  Osnove\Postupci

```
class Postupak{  
    private int brOp = 0; // brojač pozvanih operacija  
    public int Zbroji(int x, int y) {  
        ++brOp; return x + y;  
    }  
    public int Razlika (int x, int y) { ++brOp; return x-y; }  
    public int GetBrOp() { return brOp; }  
    ...  
}
```

```
Postupak p = new Postupak();  
int zbroj = p.Zbroji(13, 15);  
int razlika = p.Razlika(13, 15);  
int brojOperacija = p.GetBrOp();
```

Argumenti postupaka

32

- standardno (bez modifikatora) prenose se po vrijednosti – "call by value"

```
public static void SwapByVal (int a, int b);  
SwapByVal (a, b); //neuspješno...
```

- ref modifikator – argumenti su reference ("call by reference")

- Prije poziva postupka argumenti MORAJU biti inicijalizirani
- ref parametru argument mora eksplicitno biti predan navođenjem ref.

```
public static void SwapByRef (ref int a, ref int b);  
SwapByRef (ref a, ref b);
```

- out modifikator – izlazni argumenti


- U trenutku poziva out postupka argumenti ne moraju biti inicijalizirani.
- Pri izlasku iz postupka out argumenti MORAJU biti postavljeni.

```
public static void TestOut(out int val, int defval);  
TestOut(out i, 13);
```

- Primjer  Osnove\Postupci

Varijabilni broj argumenata

33

- Prijenos varijabilnog broja argumenata korištenjem ključne riječi `params`
 - koristi se kad broj argumenata nije unaprijed poznat
 - polje argumenata može biti bilo kojeg tipa
- Primjer:  Osnove\Postupci

```
public static void TestParams(params object[]  
    args) {  
    Console.WriteLine("Params: ");  
    for(int i= 0; i< args.Length; i++)  
        Console.WriteLine("{0}:{1}", i, args[i])  
}
```

```
TestParams("jedan", "dva", 3);  
TestParams();
```

Opcionalne vrijednosti postupaka

34

- Opcionalni argumenti kojima se zadaje pretpostavljena vrijednost

- Navode se zadnji po redu

- Primjer:  Osnove\Postupci

```
public static void TestDefault(int a, int b = 99999,
    string s = "default string") {
    Console.WriteLine("a = {0}, b = {1}, s = {2}", a, b, s);
}
```

```
TestDefault(1, 2, "RPPP");
TestDefault(2, 15);
TestDefault(3);
```


- Imenovani argumenti:

- Prilikom poziva navodi se naziv argumenta i vrijednost odvojeni dvotočkom
 - Imenovani argumenti se navode zadnji

```
TestDefault(2, s:"moj string");
TestDefault(s: "moj string", b:55, a:4);
```

Svojstva (properties) (1)

35

- Svojstvo je postupak pristupa zaštićenim varijablama instance
- Primjer  Osnove\Postupci\Postupak_v2.cs
 - umjesto postupka za pristup skrivenom članu koristi se tzv. Svojstvo

```
class Postupak_v2{  
    private int brOp;  
    public int BrOp {  
        get { //dohvat vrijednosti člana  
            return brOp;  
        }  
        //set { //omogućava promjenu vrijednosti  
        //    brOp = value;  
        //}  
    }  
}
```

```
Postupak_v2 p2 = new Postupak_v2();  
zbroj = p2.Zbroji(a, b);  
brojOperacija = p2.BrOp;  
p2.BrOp = 5 ; //mora postojati set
```

Svojstva (2)

36

➡ Nije nužno da svojstvo samo vraća ili postavlja vrijednost varijabli

➡ Može sadržavati i složeniji kod


➡ Primjer  Osnove\SvojstvaIndekseri

```
Temperatura X = new Temperatura();  
X.Fahrenheit = 70;  
Console.WriteLine("{0} = {1}", X.Fahrenheit, X.Celsius);
```

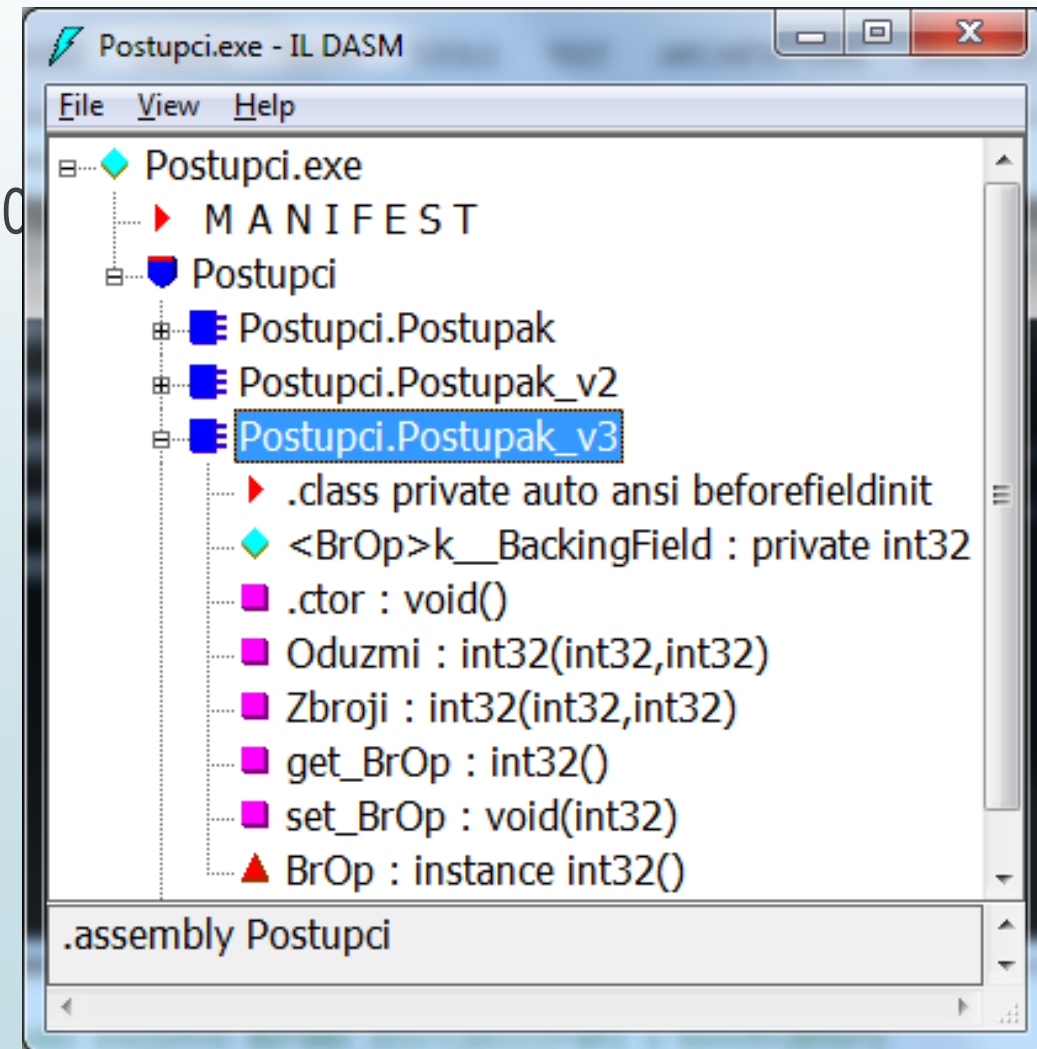
```
class Temperatura{  
    private float T;  
    public float Celsius{  
        get { return T - 273.16f; }  
        set { T = value + 273.16f; }  
    }  
    public float Fahrenheit{  
        get { return 9f / 5 * Celsius + 32; }  
        set { Celsius = (5f / 9) * (value - 32); }  
    }  
}
```


Automatska svojstva

37

- Koristi se u slučaju kad svojstvo služi samo kao omotač oko privatne varijable
- Može se postaviti modifikator pristupa (npr. private) za get i/ili set
- Interno se stvara varijabla za pohranu i kôd za dohvat i pridruživanje
- Početna vrijednost se može postaviti od C# 6.0
- Primjer  Osnove\Postupci\Postupak_v3.cs

```
class Postupak_v3 {  
    public int BrOp {  
        get;  
        private set;  
    } = 0;  
    ...  
}
```




- omogućavaju korištenje objekta kao niza (pristup elementima operatorom []), ali argument ne mora nužno biti broj
 - sintaksa uobičajena za svojstva (get i set)
- Primjer:  Razredi\SvojstvaIndekseri

```
class TemperaturaCollection {  
    public Temperatura this[int index]    // Indeksir  
    {  
        set{  
            if (index >= 0 && index < nizTemperatura.Length)  
                nizTemperatura[index] = value;  
            else throw new Exception("Pogreška!");  
        }  
        get{  
            if (index >= 0 && index < nizTemperatura.Length)  
                return nizTemperatura[index];  
            else throw new Exception("Pogreška!");  
        }  
    }  
}
```

Preopterećenje postupaka (method overloading)

39

- Više postupaka jednakog imena u istom razredu
 - moraju imati različitu signaturu (prototip), tj. različiti tip ili redoslijed argumenata
- Primjer:  Osnove\PreopterecenjePostupaka

```
static double Maximum(double x, double y, double z) {  
    Console.WriteLine("double Maximum( double x, double y,  
        double z )");  
    return Math.Max(x, Math.Max(y, z));  
}  
  
static int Maximum(int x, int y, int z) {  
    Console.WriteLine("int Maximum( int x, int y, int z )");  
    return Math.Max(x, Math.Max(y, z));  
}
```

```
// koji Maximum se poziva (double ili int)?  
Console.WriteLine("maximum1: " + Maximum(1, 3, 2));  
Console.WriteLine("maximum2: " + Maximum(1.0, 3, 2));
```

Preopterećenje operatora (operator overloading)

40

➤ Operatori koji se mogu preopteretiti

➤ unarni: + - ! ~ ++ -- true false

➤ binarni: + - * / % & | ^ << >> == != > < >= <=

➤ Primjer: Razredi\PreopterećenjeOperatora

```
public static ComplexNumber operator +(ComplexNumber x, ComplexNumber y){  
    return new ComplexNumber(x.Re + y.Re, x.Im + y.Im);  
}  
  
public static ComplexNumber operator *(ComplexNumber x, ComplexNumber y){  
    return new ComplexNumber(x.Re * y.Re - x.Im * y.Im,  
                             x.Re * y.Im + y.Re * x.Im);  
}
```

```
Console.WriteLine(x + " + " + y + " = " + (x + y));  
Console.WriteLine(x + " - " + y + " = " + (x - y));  
Console.WriteLine(x + " * " + y + " = " + (x * y));
```


Vrijednosti i reference

41

- Koju vrijednost imaju varijable `val1` i `val2` nakon sljedećeg programskog odsječka?

```
int val1 = 0;
int val2 = val1;
val2 = 5;
```

- Koliko je `ref1.Value` nakon sljedećeg programskog odsječka?

```
class Razred
{
    public int Value = 0;
}
Razred ref1 = new Razred();
Razred ref2 = ref1;
ref2.Value = 123;
```

`new` operator stvara novu instancu razreda `Razred` i vraća pokazivač na nju

`ref2` pokazuje gdje i `ref1`

- Primjer  Osnove\VrijednostiReference

Nabrajanje (Enum)

42

➡ Pobrojani tip (enum) koji se sastoji od imenovanih konstanti

➡ implicitno nasljeđuje System.Enum, a interno su vrijednosti tipa int

➡ Primjer  Osnove\Enum

```
enum Dani { Ponedjeljak, Utorak, Srijeda, Cetvrtak, Petak, Subota, Nedjelja }
```

```
DaniPoRedu dan1 = DaniPoRedu.Ponedjeljak;  
DaniPoRedu dan2 = DaniPoRedu.Utorak;  
DaniPoRedu dan3 = DaniPoRedu.Ponedjeljak;  
if (dan1 == dan3 && dan2 == DaniPoRedu.Utorak)  
    Console.WriteLine("Uvjet zadovoljen");
```

➡ Ako želimo da dani idu od 1 nadalje:

```
enum DaniPoRedu {  
    Ponedjeljak=1, Utorak, Srijeda, Cetvrtak, Petak, Subota, Nedjelja  
}
```

➡ Postoji i varijanta s oznakom FlagsAttribute ispred enumeracija pri čemu se konstantama pridjeljuju vrijednosti koje su potencije broja 2

➤ Polja određene duljine:

```
int[] a = new int[10]; // inicijalno vrijednosti 0  
int[] b = new int[] { 1, 2, 3};  
int[] c = { 1, 2, 3};
```

➤ Za duljinu polja moguće je navesti i varijablu

```
int n = 3;  
int[] d = new int[n];
```

➤ Dohvat elementa na mjestu i: b[i]

➤ Broj elemenata u polju: b.Length

➤ Primjer Osnove\Polja

```
for (int i = 0; i < b.Length; i++)  
    Console.Write(b[i]);
```

Neki od statičkih postupaka u razredu System.Array

44

- `Sort` – sortira polje ili dio polja, opcionalno koristeći vlastiti postupak za usporedbu
- `BinarySearch` – binarno pretražuje polje i vraća indeks traženog elementa ili
 - bitovni komplement indeksa prvog elementa većeg od traženog, kad traženi nije u polju a vrijednost traženog je manja od preostalih elemenata polja, odnosno bitovni komplement broja elemenata polja, kad traženi nije u polju a vrijednost traženog je veća od vrijednosti najvećeg u polju
- `Copy` – kopira čitavo polje ili dio polja polja u novo polje
- `Reverse` – obrće redoslijed članova polja

```
int[] z = new int[] { 3, 1, 2 };  
System.Array.Sort(z);  
Console.WriteLine("Tražim 3 na ind.{0}",  
    Array.BinarySearch(z, 3));  
...  
Array o = new int[3];  
Array.Copy(z, o, 2);  
Array.Reverse(o);
```

Pravokutna višedimenzionalna polja

45

➡ Pravokutna (eng. rectangular)

```
int[,] a = new int[3,5]; //inicijalno vrijednosti 0

int[,] b = new int[2,3] { {1, 2, 3}, {4, 5, 6}};
//int[,] b = new int[,] { { 1, 2, 3 }, { 4, 5, 6 } };

int[,] c = { {1, 2, 3}, {4, 5, 6}};

int[,,,] d = new int[5,6,7,8];

for (int i = 0; i < 2; i++)
    for (int j = 0; j < 3; j++)
        Console.Write (b[i, j]);
```

Nazubljena višedimenzionalna polja

46

➡ Nazubljena (eng. jagged) ili *polja polja*

```
int[][] a = new int[][] {  
    new int[] {2,3,4}, new int[] {5,6,7,8,9}  
};  
  
int[][] b = { new int[] {2,3,4}, new int[] {5,6,7,8}};  
  
for (int i = 0; i < b.Length; i++){  
    for (int j = 0; j < b[i].Length; j++){  
        Console.Write(b[i][j]);  
        Console.WriteLine();  
    }  
  
int[][] c;  
c = new int[5][];  
c[0] = new int[3]; c[1] = new int[7]; ...
```

► Tip vrijednosti koji služi za pohranu jednog znaka

► `char znak = '٣'; // arapski broj 3`

► `char znak = '\n'; // novi red`

► `char znak = '\u0663'; // Unicode kod u za arapski br. 3`

► Neki atributi i postupci

► `ToUpper` pretvara znak u odgovarajuće *Unicode* veliko slovo

► npr. `System.Char.ToUpper(znak);`

► `ToLower` pretvara znak u odgovarajuće *Unicode* malo slovo

► `IsDigit` pripada li znak skupini *Unicode* decimalnih znamenaka


► `IsLetter` pripada li znak skupini *Unicode* slova

► `MaxValue` najveća moguća vrijednost za `Char`

► `MinValue` najmanja moguća vrijednost za `Char`

Stringovi

48


- `System.String` (isto što i `string`) – niz znakova
 - String se ne može mijenjati → postupci na *stringu* stvaraju novi (promijenjeni) *string*.
 - objekti koji nisu `string` imaju postupak `ToString()`, npr. `123.ToString()`
- Primjer  Osnove\Stringovi
 - `string a = "Primjer za String";`
 - Duljina: `a.Length // 17`
 - Uklanjanje znakova: `a.Remove(7,4) // PrimjerString`
 - Velika slova: `a.ToUpper() // PRIMJER ZA STRING`
 - Mala slova: `a.ToLower() // primjer za string`
 - Podniz: `a.Substring(2, 5) // imjer`
 - Uklanjanje vodećih i pratećih praznina `a.Trim()`
 - Rastavljanje stringa na dijelove: `string[] stringovi = a.Split('r', ' ');`
`// za niz s početka daje niz {"P", "imje", "za", "St", "ing"}`
 - Spajanje u string: `string.Join("-", "spoji", "u", 1, "string"); // vraća spoji-u-1-string`

Razred StringBuilder

49

- `System.Text.StringBuilder` – promjenjiv niz znakova

- Sve promjene primjenjuju se na izvornom nizu.

- Primjer  Osnove\Stringovi

- `StringBuilder a= new StringBuilder("Primjer za String");`

- Umetanje: `a.Insert(7,'i')` → Primjeri za String

- Dodavanje: `a.Append("Builder")` → Primjeri za StringBuilder

- Dodavanje: `a.AppendFormat(" {0}/{1}", 3, "A")`
→ Primjeri za StringBuilder 3/A

Nulabilni tipovi

50

- Varijable, odnosno vrijednosni tipovi mogu imati nedefiniranu vrijednost
- Struktura Nullable za neki tip
- Definira se kao `Nullable<vrijednosni tip>` ili sa sufiksom upitnik ?

```
int? num = null;  
if (num.HasValue) // isto što i if (!(num == null))  
    Console.WriteLine("num = " + num.Value);  
else  
    Console.WriteLine("num = Null");
```

- Napomena: String se ne deklarira kao nulabilan
 - String može biti prazan (tzv. null-string) ali to ne znači da je null
 - `string s = string.Empty` // isto što i `s = ""`
 - postavljanje na null znači da nije ni prazan
 - `string s = null;` // vrijedi `s != ""`, `s == null`
- Postupak `string.IsNullOrEmpty`

Tipovi podataka *var* i *dynamic*

51

➤ Varijable definirane kao tip *var*

- Deklaracija je moguća samo unutar određenog postupka
- Stvarni tip podatka se određuje prilikom kompilacije

```
var sb = new StringBuilder();  
StringBuilder sb = new StringBuilder();
```

➤ Varijable definirane kao *Dynamic* varijable

- Stvarni tip podatka se određuje prilikom izvršavanja

```
dynamic s;  
s = "Neki tekst";  
Console.WriteLine(s.GetType());  
s = 12;  
Console.WriteLine(s.GetType());
```

- Zaobilazi se provjera prilikom kompilacije (većinom nepoželjno osim u iznimnim slučajevima)
- Može biti argument funkcije

Zadaci za vježbu (1/2)

52

- Napisati program koji iz polja stringova ispisuje one koji sadržavaju zadani podniz.
- Napisati program koji
 - učitava tekst sa standardnog ulaza
 - mijenja ga u velika slova
 - ispisuje tekst u obrnutom poretku.
- Demonstrirati generator pseudoslučajnih brojeva
 - Upotrijebiti razred `System.Random`

Zadaci za vježbu (2/2)

53

- Implementirati razred `Planet`
 - Konstruktor neka prima radijus, gravitaciju i ime planeta.
 - Razred sadrži i statičku varijablu za brojanje instanciranih planeta, svojstva `Ime`, `Radijus`, `Gravitacija` i postupak `GetCount()`.
 - Demonstrirati rad implementiranog razreda.
- Implementirati razred `PlanetCollection`
 - Razred predstavlja listu planeta.
 - Iskoristiti razred iz prethodnog zadatka s planetima.
 - Napisati postupak `Add` za dodavanje planeta.
 - Planetima se može pristupati preko indeksa.
- Implementirati razred `Nebo` koji
 - sadrži `Planet` i njegovu udaljenost od njegovog sunca.
 - Dodati postupak sortiranja planeta po udaljenosti od sunca te postupak za računanje “udaljenosti” dvaju planeta.