Uvod u programski jezik C#

Prof. dr Suzana Stojković

Dr Martin Jovanović

Dipl. inž. Ivica Marković

Dipl. inž. Teodora Đorđević

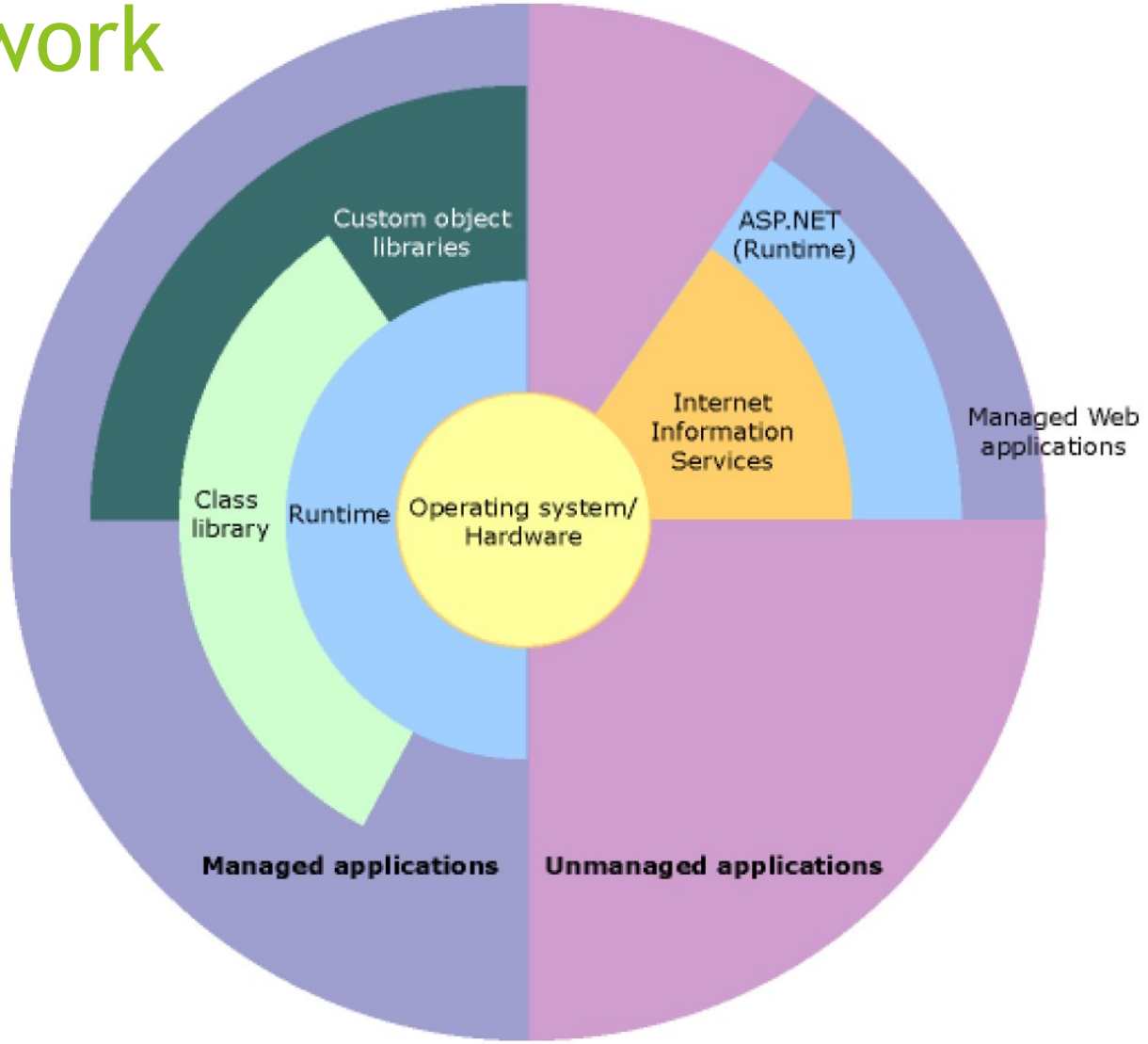
C# - osnovni podaci

- Nastao 2000. godine
- Razvijen je od strane Microsoft-ovog tima koji je predvodio Anders Hejlsberg
- Ciljevi koje je C# trebalo da ostvari:
  - Jednostavan, savremen, objektno-orijentisan programski jezik opšte namene
  - Jaki tipovi podataka, provera granica nizova, provera korišćenja neinicijalizovanih promenljivih, upravljanje memorijom i automatsko oslobađanje memorije (*garbage collection*, slično kao u Javi)
  - Važna je robusnost softvera i produktivnost programera
  - Prenosivost koda na različite platforme
  - Podrška za internacionalizaciju/lokalizaciju
  - Nije planirano da bude jednako efikasan u performansama kao C/C++

.NET Framework

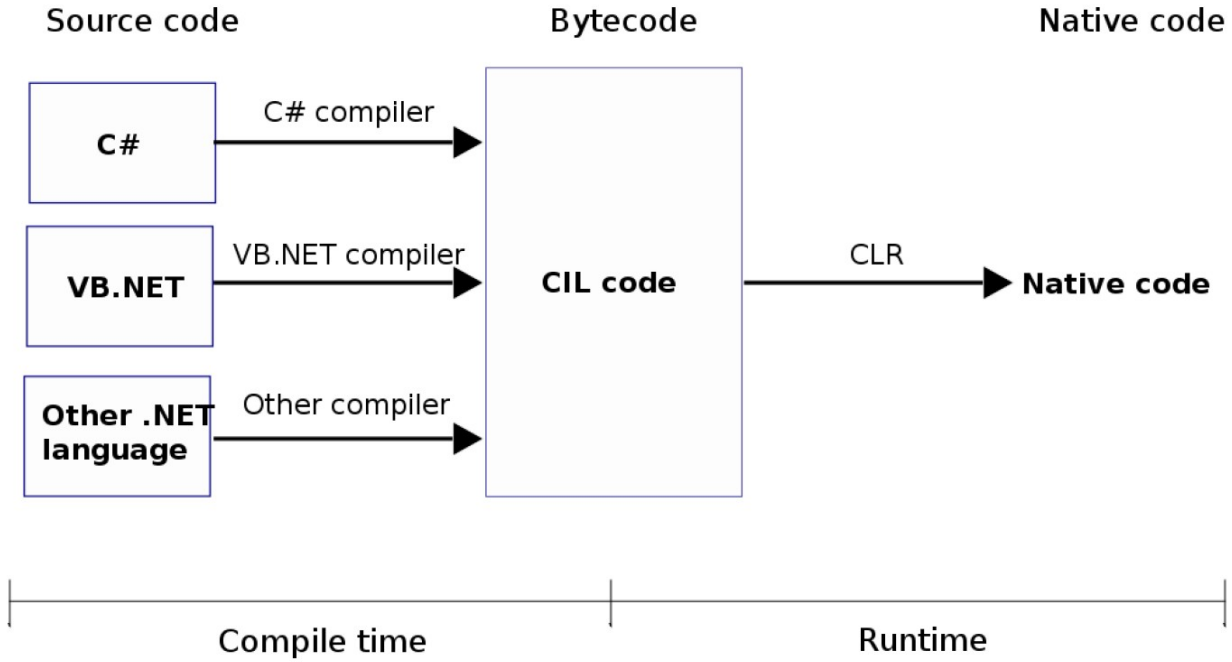
- Softversko okruženje razvijeno od strane Microsoft-a
- Uključuje veliku biblioteku klasa - *Framework Class Library* (FCL)
- Uključuje virtualnu mašinu - *Common Language Runtime* (CLR)
- Specifikacija i standard koji mora da ispuni .NET Framework je nazvana *Common Language Infrastructure* (CLI)
- Postoje i druga softverska okruženja koja ispunjavaju CLI standard, a najpoznatije je Mono

.NET Framework

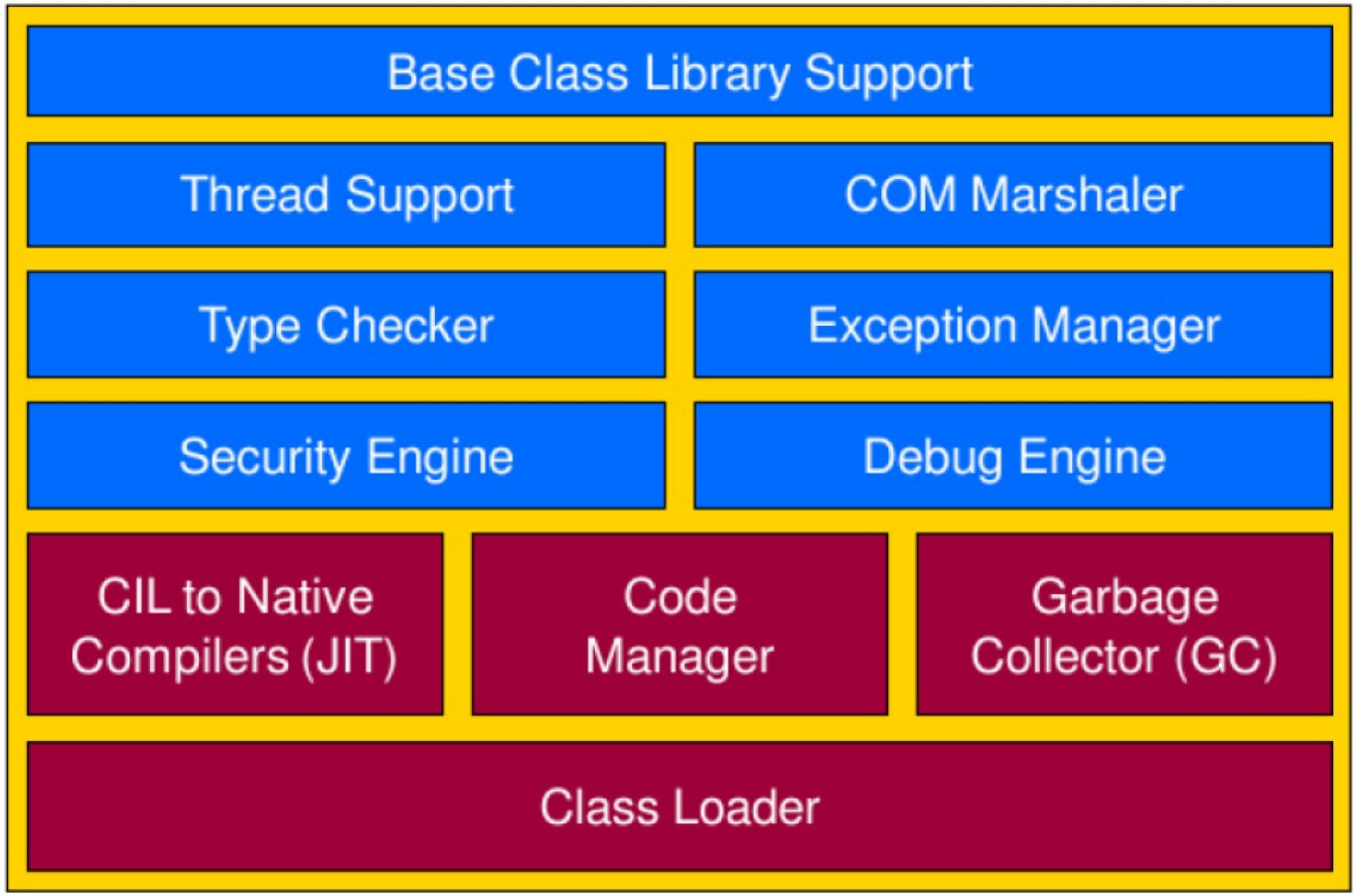


Common Language Runtime (CLR)

- *Common Intermediate Language* (CIL) – objektno orijentisan asemblerski jezik
- .NET framework podržava više programskih jezika, a izvorni kod iz svakog od .NET jezika se prevodi u CIL
- Stari naziv za CIL je *Microsoft Intermediate Language* (MSIL)
- CLR u fazi izvršenja prevodi CIL kod u asemblerski kod za konkretnu platformu (*native code*)



Common Language Runtime (CLR)



.NET Framework i C# verzije

Verzija	Datum	.NET Framework	Visual Studio
C# 1.0	januar 2002.	.NET Framework 1.0	Visual Studio .NET 2002
C# 1.2	april 2003.	.NET Framework 1.1	Visual Studio .NET 2003
C# 2.0	novembar 2005.	.NET Framework 2.0	Visual Studio 2005
C# 3.0	novembar 2007.	.NET Framework 2.0 (bez LINQ/Query) .NET Framework 3.0 (bez LINQ/Query) .NET Framework 3.5	Visual Studio 2008 Visual Studio 2010
C# 4.0	april 2010.	.NET Framework 4	Visual Studio 2010
C# 5.0	avgust 2012.	.NET Framework 4.5	Visual Studio 2012 Visual Studio 2013
C# 6.0	jul 2015.	.NET Framework 4.6, .NET Core 1.1	Visual Studio 2015
C# 7.0	mart 2017.	.NET Framework 4.7	Visual Studio 2017
C# 7.1	avgust 2017.	.NET Core 2.0	Visual Studio 2017
C# 7.2	novembar 2017.		Visual Studio 2017
C# 7.3	novembar 2017.	.NET Framework 4.8, .NET Core 2.2	Visual Studio 2017
C# 8	septembar 2019.	.NET Core 3.0	Visual Studio 2019

Razvojno okruženje Visual Studio

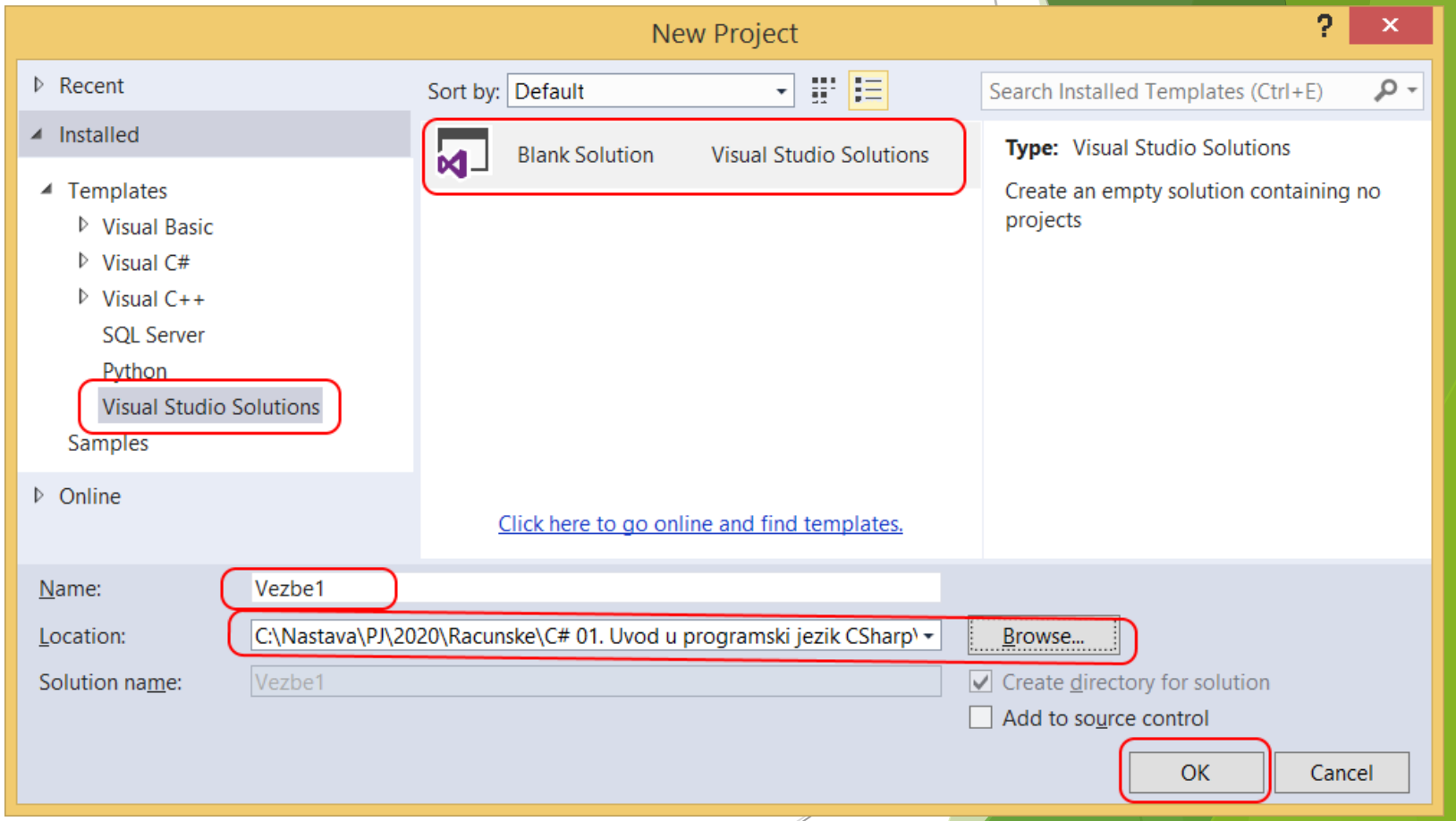
- U okviru ovog kursa obradićemo detalje zaključno sa verzijom C# 5.0
- Naravno, sve što se radi u okviru je neophodna osnova za novije verzije C#-a i važi i u novijim verzijama C#-a, ali nema dovoljno prostora i vremena da se obrade kasnije dodate funkcionalnosti
- Za ovaj kurs je dovoljna verzija Visual Studio 2013
- Može da se koristi i neka novija verzija, a u računarskim učionicama za polaganje kolokvijuma je instaliran Visual Studio 2013
- Dovoljna je bilo koja besplatna verzija programa Visual Studio (u novijim verzijama besplatna je Community Edition, u starijim Express Edition)

Način organizacije projekata u VS

- U razvojnom okruženju Eclipse smo više projekata grupisali u okviru istog radnog prostora (*workspace*)
- U VS se za grupisanje više projekata koristi poseban “nadprojekat” - *solution*
- Nadalje ćemo za svaki čas da imamo poseban *solution* u kome će biti grupisani pojedinačni projekti

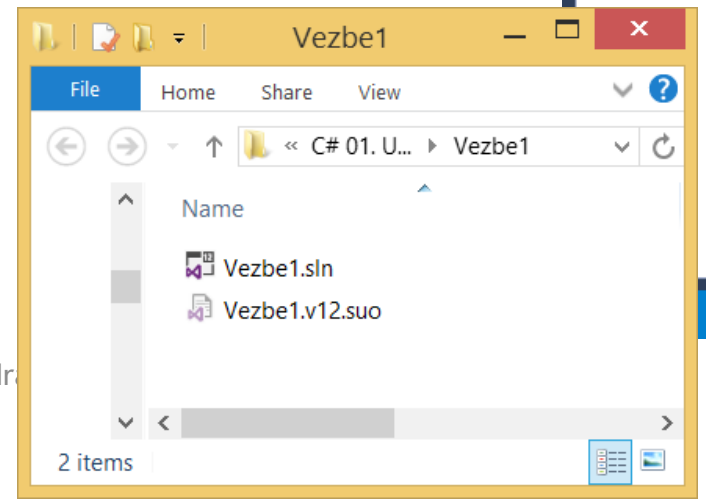
Kreiranje *solution*-a u VS

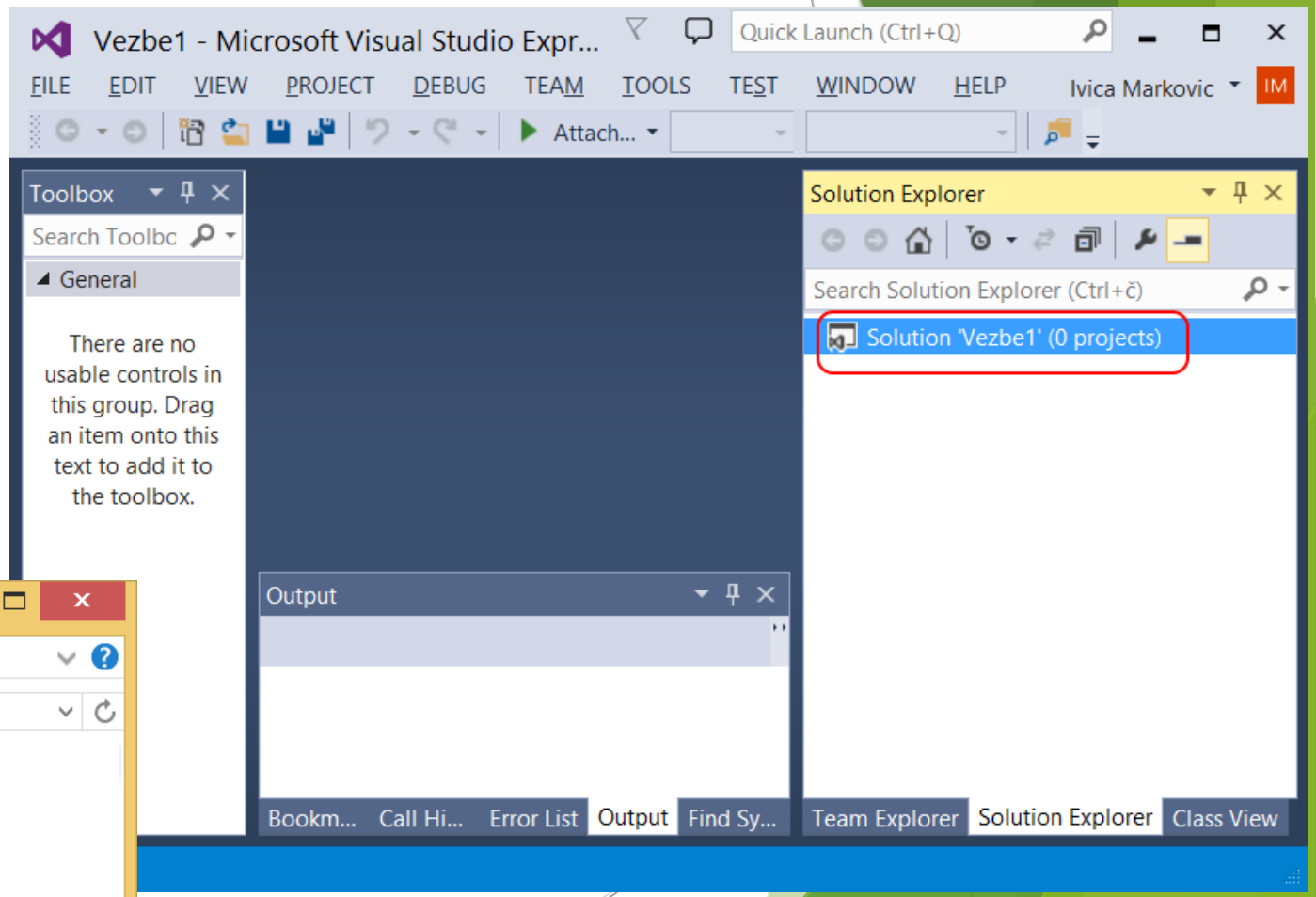
- Biramo opciju u meniju File → New Project… (ili prečicu na tastaturi Ctrl + Shift + N)
- Dobijamo dijalog kao slici
- U stablu sa leve strane biramo opciju “Visual Studio Solutions”
- Na centralnoj površini biramo opciju “Blank Solution”
- U polju “Name” upisujemo ime *solution*-a po želji
- Dugmetom “Browse...” biramo lokaciju za novi *solution*
- Klikom na dugme “OK” potvrđujemo kreiranje



Kreiranje *solution*-a u VS

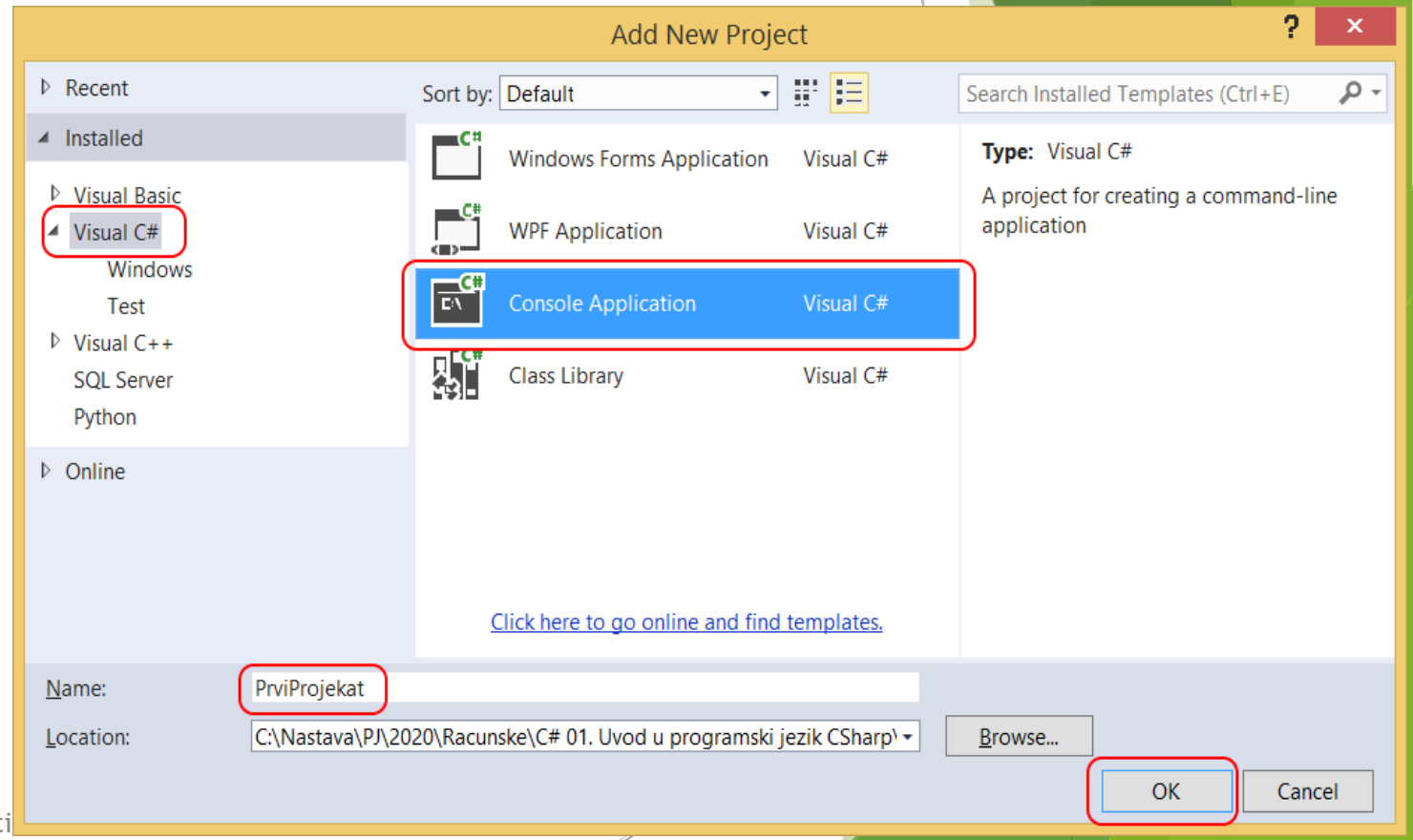
- Kreirani solution možemo da vidimo u “Solution Explorer” prozoru sa desne strane glavnog prozora
- Desnim klikom na *solution* “Vezbe1” pa izborom opcije “Open Folder in File Explorer” iz kontekstnog menija možemo da vidimo sadržaj foldera u kome je kreiran *solution*
- Podaci o solution-u se nalaze u fajlu sa ekstezijom .SLN





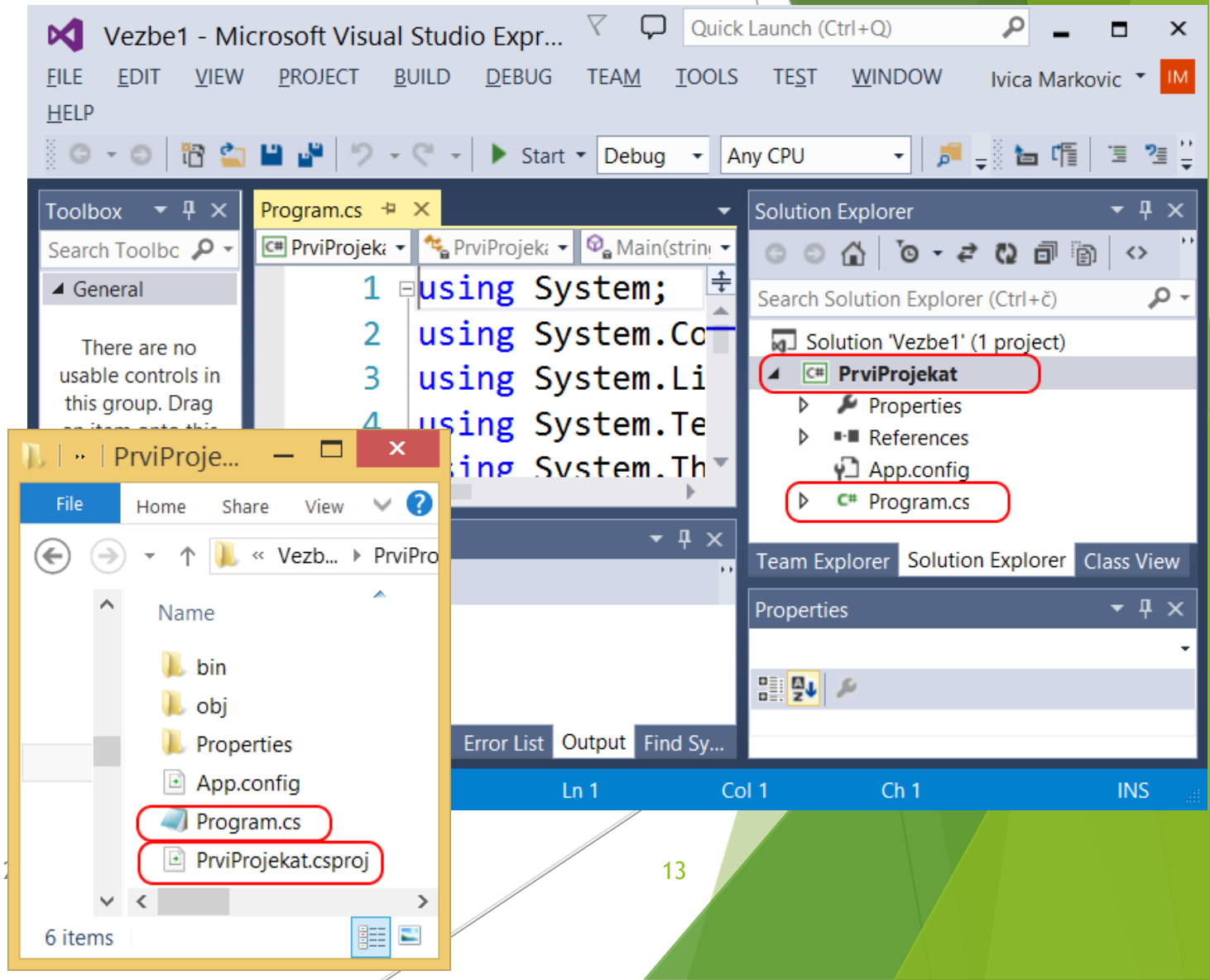
Kreiranje projekta u VS

- Isto kao kod kreiranja *solution*-a biramo opciju u meniju File→New Project… (ili prečicu na tastaturi Ctrl + Shift + N), a možemo i na treći način desnim klikom na postojeći *solution* pa izborom opcije Add → New Project…
- Dobijamo poznati dijalog za kreiranje projekta
- U stablu sa leve strane biramo opciju “Visual C#”
- Na centralnoj površini biramo opciju “Console Application”
- U polju “Name” upisujemo ime projekta "PrviProjekat"
- Klikom na dugme “OK” potvrđujemo kreiranje projekta



Kreiranje projekta u VS

- Kreirani projekat možemo da vidimo u “Solution Explorer” prozoru sa desne strane glavnog prozora, ispod ranije kreiranog *solution*-a
- Uz projekat je odmah kreirana klasa “Program” koja sadrži main metodu
- Ako pogledamo kreirani projekat na fajl sistemu, naći ćemo više novih fajlova i podfoldera
- Podaci o projektu se čuvaju u fajlu sa ekstenzijom .CSPROJ
- Programski kod se čuva u fajlovima sa ekstenzijom .CS
- Posle kompajliranja projekta u podfolderu BIN će se generisati izvršni fajl sa ekstenzijom .EXE



Pokretanje projekta u VS

- Opcije za pokretanje projekta i debagiranje su identične (iz menija i prečice na tastaturi) kao i za C/C++ projekte
- Za prikaz na konzoli koristimo statičku metodu Write ili WriteLine iz klase Console:

```
Console.WriteLine("Zdravo!");
```

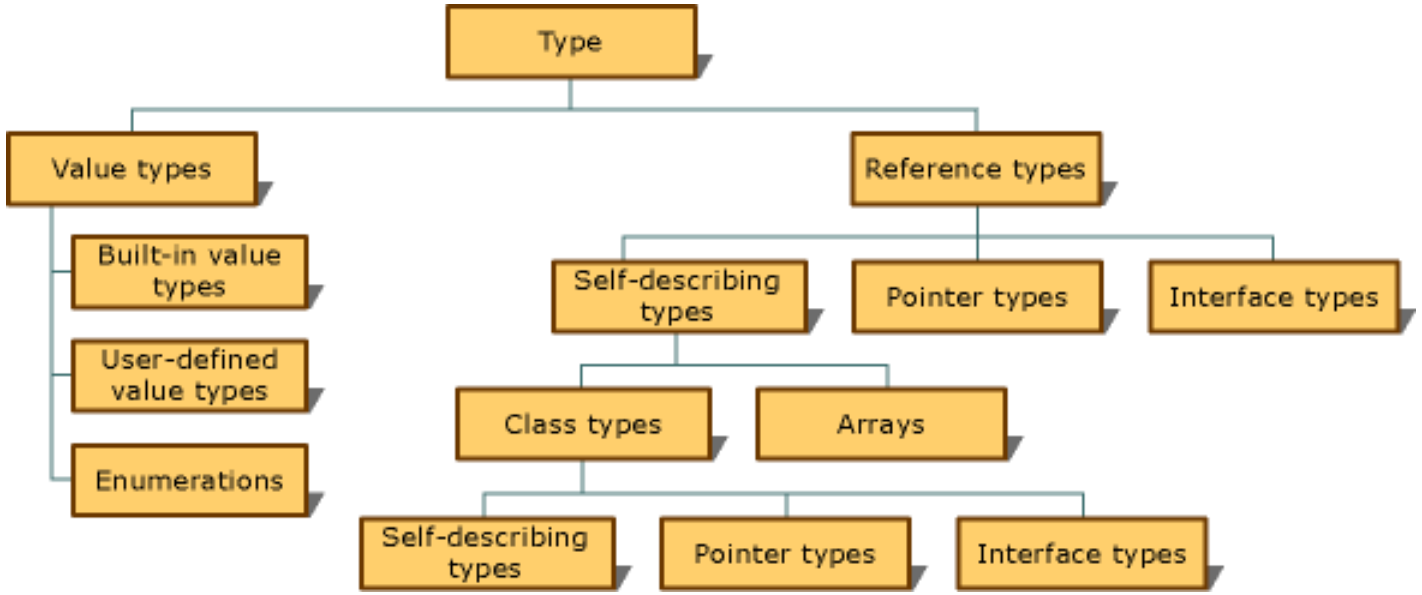
- Ulaz/izlaz će detaljnije biti obrađen kasnije

Tipovi podataka u C#-u

- Slično kao i u Javi postoji osnovna klasa iz koje su izvedeni svi ostali tipovi podataka - System.Object
- Umesto punog imena koje uključuje i naziv prostora imena System (*namespace*, slično kao paket u Javi), ako se prethodno u naš fajl uključi taj *namespace* može da se koristi samo ime klase Object
- Umesto System.Object uvek može da se koristi ključna reč jezika C# object – to je alijas za System.Object
- Svi tipovi podataka u C#-u (i ugrađeni i korisnički definisani) se dele u 2 grupe:
  - Vrednosni tipovi podataka (*value types*)
  - Referentni tipovi podataka (*reference types*)

Common Type System (CTS)

- Definiše tipove podataka koje podržava CLR i koje svaki od .NET jezika mora da implementira
- Npr. CTS definiše System.Int32 kao tip za predstavljanje celobrojnog podatka veličine 4 bajta, a odgovarajući tip u C# jeziku se zove *int*(tačnije C# dodaje *int* kao alijas, a moguće je koristiti i naziv System.Int32 u C#-u)



Vrednosni tipovi podataka

- Izvedeni iz klase System.ValueType, koja je iz osnovne klase System.Object
- Dakle, vrednosni tipovi ne nasleđuju direktno System.Object
- Ključne reči kojima se deklarišu vrednosni tipovi podataka: *struct* i *enum*
- Možemo da definišemo nove, korisničke vrednosne i referentne tipove podataka (u Javi možemo da dodajemo samo referentne tipove podataka)
- Podela:
  - Ugrađeni vrednosni tipovi podataka (*built-in value types*)
  - Korisnički definisani vrednosni tipovi podataka (*user-defined value types*) - struct
  - Enumeracije (*enumerations*) - enum

Ugrađeni vrednosni tipovi podataka
Celobrojni tipovi

Oznaka tipa	Broj bitova	Opseg	Sufiks za literal
sbyte	8	-2^7 do $2^7 - 1$	
byte	8	0 do $2^8 - 1$	
short	16	-2^{15} do $2^{15} - 1$	
ushort	16	0 do $2^{16} - 1$	
char	16	0 do $2^{16} - 1$	
int	32	-2^{31} do $2^{31} - 1$	
uint	32	0 do $2^{32} - 1$	U, u
long	64	-2^{63} do $2^{63} - 1$	L, l
ulong	64	0 do $2^{64} - 1$	UL, ul

Ugrađeni vrednosni tipovi podataka Realni tipovi

- Realni tipovi podataka u pokretnom zarezu

Oznaka tipa	Broj bitova	Opseg	Preciznost	Sufiks za literal
float	32	$\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$	7 cifara	F, f
double	64	$\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$	15 cifara	D, d

Ugrađeni vrednosni tipovi podataka Realni tipovi

- Realni tip podataka u fiksnom zarezu
- Ne postoji u Javi
- Uveden zbog finansijskih podataka (zbog zahteva za većom preciznošću od preciznosti tipa double)

Oznaka tipa	Broj bitova	Opseg	Preciznost	Sufiks za literal
decimal	128	1.0×10^{-28} to 7.9×10^{28}	28 cifara	M, m

Ugrađeni vrednosni tipovi podataka

- Logički tip podataka
- *bool* umesto *boolean* u Javi

Oznaka tipa	Broj bitova	Opseg	Sufiks za literal
bool	8	true, false	

Korisnički definisani vrednosni tipovi podataka - Strukture

- Ključna reč *struct*
- Kao i klase predstavljaju kolekcije atributa, metoda i svojstava (*property* će biti objašnjen u poglavlju o klasama)
- Namenjene su za predstavljanje „lakih“ objekata (*lightweight*) npr. tačka u prostoru
- Alokacija u memoriji objekta tipa strukture je efikasnija i brža u odnosu na alokaciju instance bilo koje klase
- To je postignuto tako što je svaki objekat tipa strukture ograničen da ima fiksnu veličinu u bajtovima pa se **alocira u statičkoj zoni memorije** (*stack*)
- Zbog prethodnog ograničenja **kod struktura ne postoji nasleđivanje** i ne može se definisati izvedena struktura ili klasa iz neke strukture (za razliku od klasa)

Korisnički definisani vrednosni tipovi podataka - Strukture

- Definicija

```
struct Point
{
    public int x, y;
    public Point(int x, int y) { this.x = x; this.y = y; }
}
```

- Instanciranje objekta
  - Point a = new Point(10, 10);

    a.x = 20;

Korisnički definisani vrednosni tipovi podataka - Strukture

- Efikasnost – tačku u 2D prostoru možemo na sličan način da predstavimo i strukturom i klasom, ali je struktura efikasnija u radu sa memorijom
- Videti primere u projektima StrukturaPoint i KlasaPoint
  - class Point

```
{
  public int x, y;
  public Point(int x, int y) { this.x = x; this.y = y; }
}
```

  - struct Point

```
{
  public int x, y;
  public Point(int x, int y) { this.x = x; this.y = y; }
}
```

Korisnički definisani vrednosni tipovi podataka - Strukture

- Efikasnost
  - class Program

```
{
  static void Main(string[] args) {
  Point[] points = new Point[100];
        // ako radimo sa klasama imamo 100 dodatnih poziva operatora new
  for (int i = 0; i < 100; i++) points[i] = new Point(i, i);
      // ako radimo sa strukturama kreiranje objekata u petlji nije potrebno
      // i možemo samo da odradimo dodelu vrednosti postojećim objektima
      // dovoljno je for (int i = 0; i < 100; i++) { points[i].x = i; points[i].y = i; }
  }
}
```

Korisnički definisani vrednosni tipovi podataka – Enumeracije (Nabrajanja)

- Enumeracija je posebni vrednosni tip koji je određen skupom vrednosnih konstanti
- Deklaracija novog enum tipa:
  - enum DanUNedelji

```
{
    Ponedeljak,
    Utorak,
    Sreda,
    Cetvrtak,
    Petak,
    Subota,
    Nedelja
}
```

Korisnički definisani vrednosni tipovi podataka – Enumeracije (Nabrajanja)

- Videti primer projekta Enumeracije
- Deklaracija promenljive tipa enumeracije i dodela vrednosti toj promenljivoj:
  - DanUNedelji danas = DanUNedelji.Ponedeljak;
- Korišćenje promenljive tipa enumeracije:
  - static void Main(string[] args)

```
{
    DanUNedelji danas = DanUNedelji.Ponedeljak;
   if (danas != DanUNedelji.Subota && danas != DanUNedelji.Nedelja)
       Console.WriteLine("Danas je radni dan." );
}
```

Korisnički definisani vrednosni tipovi podataka – Enumeracije (Nabrajanja)

- Enumeracija se interno predstavlja celobrojnim podatkom
- To nam potvrđuje i konverzija tipa iz narednog primera
- Ako se ne zada drugačije taj tip je *int* „po deafult-u“

```
static void Main(string[] args)
{
    for (int i = 0; i < 7; i++)
      Console.WriteLine((DanUNedelji) i );
}
```

Korisnički definisani vrednosni tipovi podataka – Enumeracije (Nabrajanja)

- Može da se zada da je enumeracija i nekog manjeg tipa radi uštede prostora
  - enum DanUNedelji **: byte**

```
{
    Ponedeljak,
    Utorak,
    Sreda,
    Cetvrtak,
    Petak,
    Subota,
    Nedelja
}
```

Korisnički definisani vrednosni tipovi podataka – Enumeracije (Nabrajanja)

- Vrednosti u enumeraciji podrazumevano kreću od nule, a može da se zada i proizvoljna početna vrednost
- U narednom primeru Ponedeljak uzima vrednost 1, Utorak vrednost 2, Sreda vrednost 3...
  - enum DanUNedelji

```
{
    Ponedeljak = 1,
    Utorak,
    Sreda,
    Cetvrtak,
    Petak,
    Subota,
    Nedelja
}
```

Referentni tipovi podataka

- Izvedeni iz klase System.Object
- Ključne reči kojima se deklarišu referentni tipovi podataka: *class*, *interface* i *delegate*
- Referentni tipovi podataka se dinamički alociraju
- Promenljive referentnih tipova (obično ih zovemo objekti) sadrže samo reference na podatke
- Primer: klasa System.String, ravnopravno se koristi i ključna reč – alijas *string*

Vrednosni i referentni tipovi - poređenje

- Vrednosti
  - Promenljive vrednosnog tipa sadrže same vrednosti
  - Promenljive referentnog tipa sadrže pokazivače na vrednosti (podatke) u memoriji
- Osnovna klasa
  - Vrednosni tipovi su izvedeni iz klase System.ValueType (zabranjeno je nasleđivanje vrednosnog tipa pa ValueType zabranjuje dalje nasleđivanje)
  - Referentni tipovi su izvedeni iz klase System.Object
- Dodela (operator =)
  - Kod vrednosnog tipa naredba a = b; kopira vrednost promenljive b u a
  - Kod referentnog tipa naredba a = b; kopira **referencu** na iste podatke u memoriji

Zauzimanje memorije

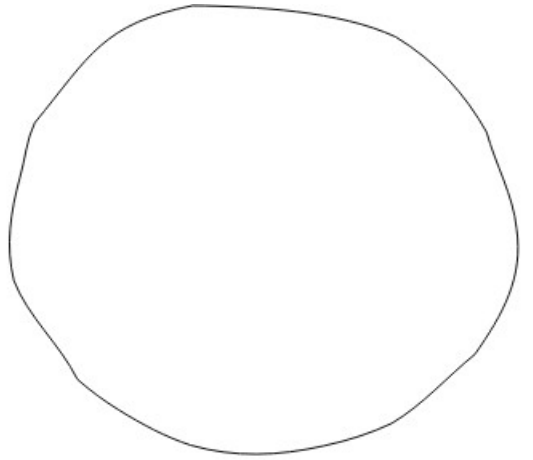
- .NET Framework tj. CLR razlikuje statičku i dinamičku zonu memorije
- Statička zona memorije se naziva još i *stack* (promenljive se smeštaju u stek u onom redosledu kojim smo ih deklarisali u programu i brišu se sa steka onako kako izlazimo iz opsega važenja promenljive)
- Dinamička zona memorije se naziva još i *heap* (objekti se tamo smeštaju pozivom operatora *new* i redosled smeštanja određuje sam CLR prema raspoloživom slobodnom prostoru)

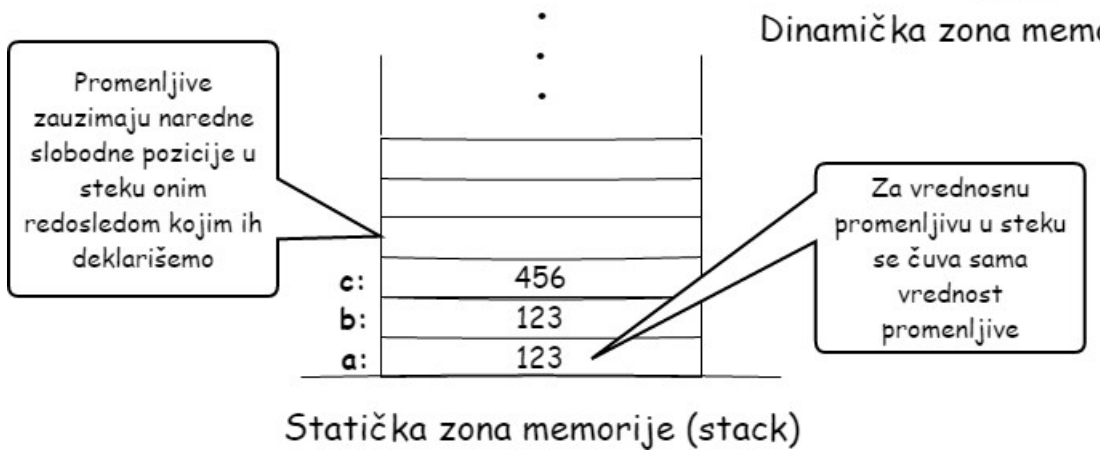
Zauzimanje memorije – Vrednosni tipovi

- Promenljiva sadrži samu vrednost
- Operator dodele zaista radi kopiranje
- Primer zauzimanja memorije za deo programa:
  - int a = 123;

    int b = a;

    int c = 456;



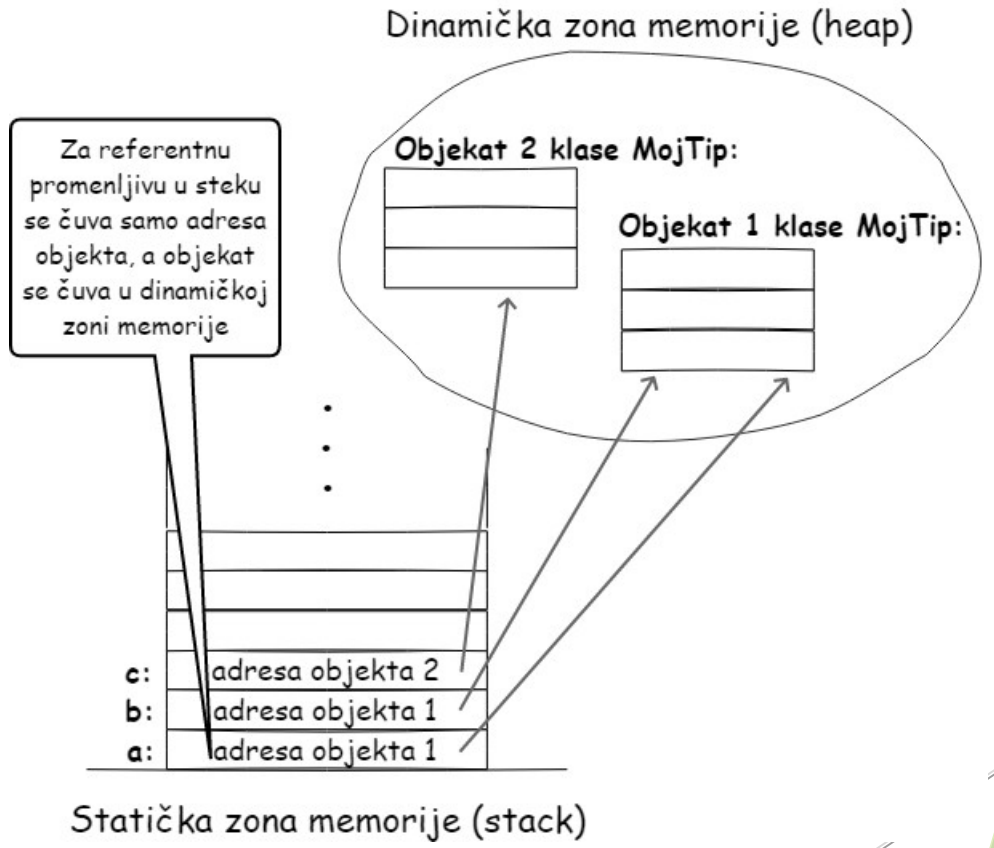


Zauzimanje memorije – Referentni tipovi

- Promenljiva referentnog tipa samo pokazuje na lokaciju (slčno pokazivačima u C/C++u)
- Promenljiva se nalazi u *stack*-u, podatak (objekat) se nalazi u *heap*-u
- Primer (MojTip je neka klasa):
  - MojTip a = new MojTip();

    MojTip b = a;

    MojTip c = new MojTip();

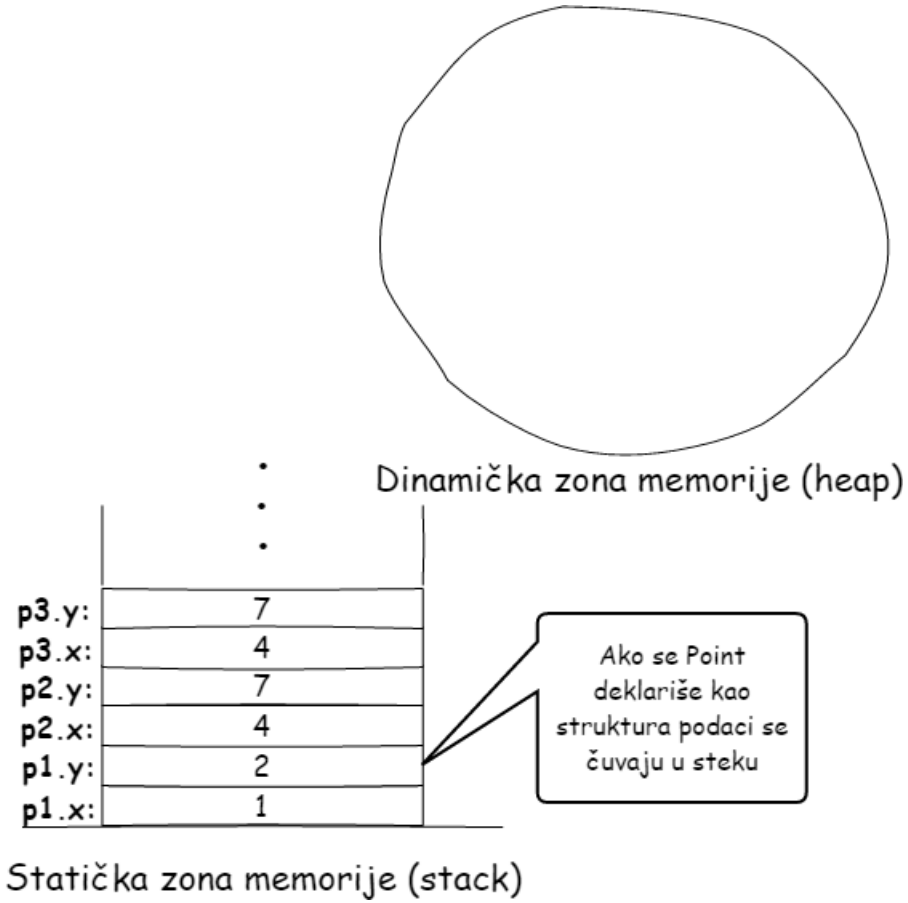


Zauzimanje memorije – Strukture

- Strukture su vrednosni tipovi pa se čuvaju u statičkoj zoni memorije
- Atributi strukture se tako raspoređuju da zauzimaju sukcesivne memorijske lokacije
- Ako je Point definisana kao na prethodnim slajdovima struct Point…
- Primer:
  - Point p1 = new Point(1, 2);

    Point p2 = new Point(4, 7);

    Point p3 = p2;

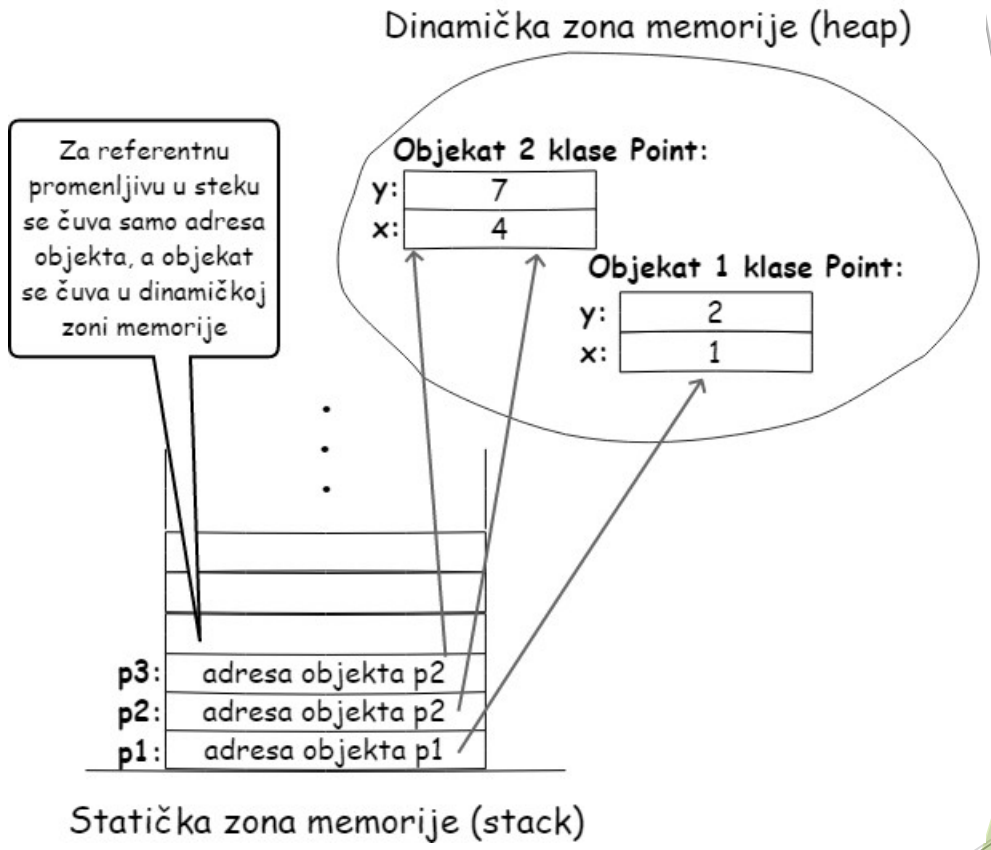


Zauzimanje memorije – Klase

- Klase su referentni tipovi pa se čuvaju u dinamičkoj zoni memorije
- Ako je Point definisana kao na prethodnim slajdovima class Point…
- Primer:
  - Point p1 = new Point(1, 2);

    Point p2 = new Point(4, 7);

    Point p3 = p2;



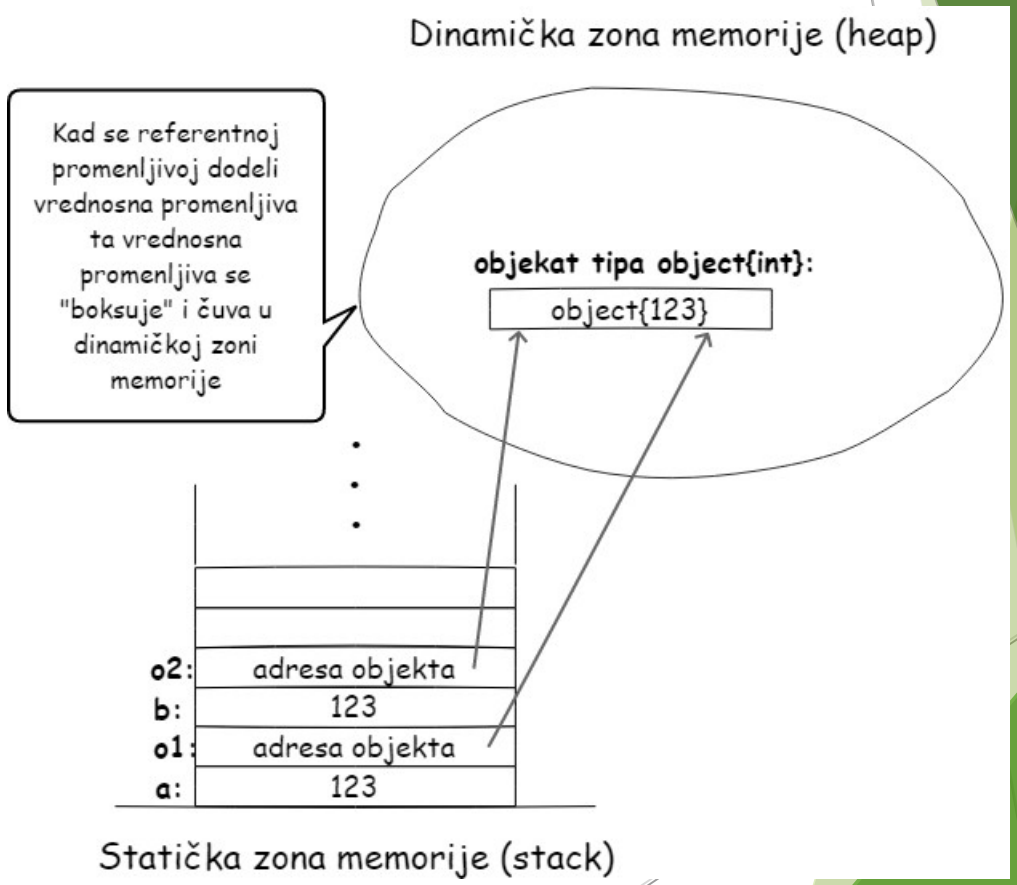
Boxing i *Unboxing* (Pakovanje i raspakivanje)

- Bitna razlika u odnosu na Javu – ne postoje *wrapper* klase (Integer, Long, Float, Double)
- Pakovanje vrednosne promenljive u objekat referentnog tipa se naziva *boxing*
- Inverzna operacija se naziva *unboxing*
- Primer:
  - int a = 123;

    object o1 = a; //boxing

    int b = (int) o1; //unboxing

    object o2 = o1;



Boxing i *Unboxing* vs kastovanje

- „Boksovana“ promenljiva se čuva kao poseban tip (npr. „boksovani“ int se čuva kao poseban tip object{int} koji je automatski generisan od strane CLR u toku izvršenja programa)
- Primer koji ne radi:
  - int a = 123;

    object o1 = a;

    float b = (float)o1; // nemoguća konverzija iz object{int} u float
- Primer koji radi:
  - int a = 123;

    object o1 = a;

    float b = (float)(int)o1; // prvo se izvrši *unboxing* pa konverzija iz int u float

Promenljive

- Provera dodele vrednosti pre korišćenja
  - int i; //i je lokalna promenljiva

    Console.WriteLine(i); //error CS0165: Use of unassigned local variable 'i'
- Podrazumevane (*default*) vrednosti promenljivih:
  - Dodeljuju se samo promenljvama koje su instance klasa, statičkim promenljivama, elementima nizova.
  - Za referentne tipove podrazumevana vrednost je null

    string s; // s == null
  - Za vrednosne tipove podrazumevana vrednost je određena podrazumevanim konstruktorom

    double d; // d == 0.0

Nizovi

- Jednodimenzioni niz
  - Primer: int[] array = new int[30];
  - Za razliku od Jave ovo nije sintaksno ispravno: int niz [] = new int[30];
- Nizovi nizova (jagged array)
  - int[][] niz = new int[2][];

    niz[0] = new int[100];

    niz[1] = new int[5];
  - Elementi svakog od 3 rezervisana niza zauzimaju sukcesivne memorijske lokacije
  - Sami nizovi neće zauzeti sukcesivne memorijske lokacije
  - Za razliku od Jave ne radi: int[][] niz = new int[2][2];

Nizovi

- Višedimenzioni niz (matrica)
  - Elementi se čuvaju u sukcesivnim memorijskim lokacijama
  - Primer dvodimenzione matrice:

    ```
    int[,] niz = new int[5, 9];
    niz[3, 8] = 100; // pristup elementu sa indeksom vrste 3 i indeksom kolone 8
    ```
- Moguć je veći broj dimenzija
  - Primer:

    ```
    float[, , ,] m = new float[2, 3, 4, 5];
    ```
- Za razliku od Jave i C++a kod pravih matrica (koje nisu nizovi nizova) **ne postoji poseban par zagrada []** za svaku dimenziju, dimenzije su razdvojene znakom **, (zarez)** u okviru istih zagrada

Leksički elementi C# jezika

- Beli simboli
- Komentari
- Tokeni
  - Identifikatori
  - Literali
  - Ključne reči
  - Operatori
  - Separatori

Leksički elementi C# jezika
Komentari

- Kao i u C/C++ i Javi postoje sledeće 2 vrste komentara:
  - `// komentar koji važi do kraja reda, ili`
  - `/* komentar koji može da se proteže kroz više redova,`
    `a važi sve do znaka */`
- Postoje i dokumentacioni komentari:
  - `/// <summary>`
  - `/// Main metoda je početna tačka izvršenja programa.`
  - `/// </summary>`
  - `/// <param name="args"></param>`
  - `static void Main(string[] args)`

Leksički elementi C# jezika
Identifikatori

- Identifikator počinje slovom ili znakom '_'
- Nadalje sledi niz simbola iz skupa { <slovo>, <cifra>, _ }
- Identifikator ne može da sadrži znak '$' za razliku od Jave
- Validni identifikatori: x, a1, _b2, prom3
- Postoji prefiks @ koji se ne smatra delom identifikatora, ali nam omogućava da ključne reči koristimo kao identifikatore:
  - prom3 i @prom3 je isti identifikator
  - @if, @for su validni identifikatori
  - **ove identifikatore koristiti samo ako je neophodno**

Leksički elementi C# jezika
Literali

- Logički
- Celobrojni
- Realni
- Znakovni (character)
- String literali
- null

Leksički elementi C# jezika
Logički literali

- Postoje 2 logičke vrednosti, kao i u Javi
  - true
  - false
- Tip podataka je *bool* (za razliku od *boolean* u Javi)

Leksički elementi C# jezika
Celobrojni literali

- Mogu biti zapisani u dekadnom i heksadekadnom brojnom sistemu:
  - dekadni zapis sadrži samo dekadne cifre (npr. 25)
  - heksadekadni zapis počinje sa 0x ili 0X i sadrži heksadekadne cifre (0x1F, 0X1F, 0x1f)
- Celobrojni literali mogu opciono da sadrže jedan od sledećih sufiksa:
  - U, u, L, l, UL, Ul, uL, ul, LU, Lu, lU, lu (sve kombinacije malih i velikih slova U i L, gde U označava neoznačeni broj – *unsigned*, a L podatak tipa *long*)
- Sufiksi određuju tip literala na sledeći način:
  - nema sufiksa – literal je prvog od tipova u kome se može predstaviti *int*, *uint*, *long*, *ulong*
  - U ili u - literal je prvog od tipova u kome se može predstaviti *uint*, *ulong*
  - L ili l – literal je prvog od tipova u kome se može predstaviti *long* ili *ulong*
  - UL, Ul, uL, ul, LU, Lu, lU, ili lu – literal je tipa *ulong*

Leksički elementi C# jezika
Realni literali

- Celobrojni literali mogu opciono da sadrže jedan od sledećih sufiksa:
  - F, f, D, d, M, m
- Sufiksi određuju tip literala na sledeći način:
  - nema sufiksa – literal je tipa *double*
  - F ili f - literal je tipa *float*
  - D ili d – literal je tipa *double*
  - M ili m – literal je tipa *decimal*

Leksički elementi C# jezika
Znakovni literali

- Predstavljaju se jednim znakom između jednostrukih apostrofa
  - 'x', '1', '^', '.'
- *Escape character* \ oduzima specijalno značenje znakovima koji ga imaju ili daje specijalno značenje „običnim“ znakovima
  - '\\' predstavlja \
  - '\'' predstavlja apostrof

Escape sequence	Character name	Unicode encoding
\'	Single quote	0x0027
\"	Double quote	0x0022
\\	Backslash	0x005C
\0	Null	0x0000
\a	Alert	0x0007
\b	Backspace	0x0008
\f	Form feed	0x000C
\n	New line	0x000A
\r	Carriage return	0x000D
\t	Horizontal tab	0x0009
\v	Vertical tab	0x000B

Leksički elementi C# jezika
String literali

- Predstavljaju se nijednim ili više znakova između dvostrukih navodnika
  - “Programski jezik C#”, “Hello world!”, “Ovo je escape character: \\”
- Postoje i doslovni string literali (*verbatim string literals*) koji počinju znakom @
- Skraćuju zapis stringa koji ima dosta specijalnih karaktera
- Parovi običnih i doslovnih stringova koji imaju istu vrednost
  - “Hello world!” i @”Hello world!”
  - “c:\\Program Files\\Microsoft Visual Studio 11.0\\” i

    @“c:\Program Files\Microsoft Visual Studio 11.0\”
  - “jedan\r\ndva\r\ntri" i

    @“jedan

    dva

    tri"

Leksički elementi C# jezika
Specijalni literal: null

- Isto kao u Javi služi da označi promenljivu nekog referentnog tipa koja trenutno nema dodeljenu vrednost tj. ne pokazuje ni na jedan objekat u memoriji:
  - String s; //posle izvršenja ove naredbe s ima vrednost null

    s = “ABC”; //posle izvršenja ove naredbe s ima vrednost “ABC”

Leksički elementi C# jezika
Ključne reči

Zelenom bojom su označene reči koje u Javi imaju identičan oblik i značenje, a reči zadate crvenom bojom ćemo obraditi u okviru ovog kursa

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

Leksički elementi C# jezika
Operatori

- aritmetički
- operatori dodele
- operatori za inkrementiranje i dekrementiranje
- logički operatori
- operatori bitskog pomeranja
- operatori poređenja
- operatori za pristup članovima
- operator poziva funkcije
- operator za kreiranje novog objekta
- operatori za konverziju tipa (cast)
- operatori za ispitivanje tipa objekta
- operator zarez (comma operator)
- ternarni uslovni operator
- *null coalescing* operator

Leksički elementi C# jezika
Operatori

- Za razliku od Jave, moguće je predefinisanje operatora od strane korisnika
- Slično, ali ne potpuno isto kao u C++u
- Biće obrađeno u temi o klasama

Leksički elementi C# jezika
Aritmetički operatori

- Aritmetički operatori su identični kao u Javi i C/C++u
  - **+** aritmetičko sabiranje
  - **-** aritmetičko oduzimanje
  - ***** aritmetičko množenje
  - **/** aritmetičko deljenje
    - kada su operandi celobrojni, ovo deljenje je celobrojno, i razlomljeni deo se odseca
    - kada je makar jedan operand realan, deljenje je realno.
  - **%** ostatak pri deljenju (operandi su celobrojni)
- Moguće je primeniti aritmetički operator nad dva podatka različitih tipova pri čemu je rezultat uvek višeg tipa od ta dva
- Poseban slučaj operatora + ukoliko je bar jedan od operanada objekat klase String, onda znači nadovezivanje (konkatenaciju) stringova

Leksički elementi C# jezika
Operatori dodele

- Operatori dodele su identični kao u Javi i C/C++u
  - `=` x=2 (klasična dodela)
  - `+=` x+=2 je isto što i x=x+2
  - `*=` x=x*2
  - `/=` x=x/2
  - `&=` x=x&2
  - `|=` x=x|2
  - `^=` x=x^2
  - `<<=` x=x<<2
  - `>>=` x=x>>2

Leksički elementi C# jezika - Operatori za inkrementiranje i dekrementiranje

- Operatori za inkrementiranje i dekrementiranje su identični kao u Javi i C/C++u
  - ++
  - --
- Mogu biti prefiksni i postfiksni tj. operacija inkrementiranja odnosno dekrementiranja izvršiće se pre ili nakon neke druge operacije, zavisno da li operator stoji sa leve ili sa desne strane operanda

Leksički elementi C# jezika
Logički operatori

- Logički operatori su identični kao u Javi
  - operator and: **&&**
  - operator or: **||**
  - operator not: **!**

- Obe strane operatora treba da budu logički izrazi (tip bool). Rezultat operatora je takođe tipa bool.
- Primer za logičko i: bool log = a>b && c==d;

Leksički elementi C# jezika
Operatori po bitovima

- Operatori po bitovima su identični kao u Javi
- Logički operatori po bitovima:
  - operator and: **&**
  - operator or: **|**
  - operator xor: **^**
  - operator not: **~**
- Operatori pomeranja po bitovima (shift):
  - pomera operand a za b bitova u desno: a **>>** **b**
  - pomera operand a za b bitova u levo: a **<<** **b**

Leksički elementi C# jezika
Operatori poređenja

- Operatori poređenja su identični kao u Javi
  - `==` jednako
  - `!=` nije jednako
  - `<` manje
  - `>` veće
  - `<=` manje ili jednako
  - `>=` veće ili jednako.

Leksički elementi C# jezika - Operatori za poziv funkcija i pristup članovima

- I ovi operatori su identični kao u Javi
- Operator poziva funkcije **()**
  - objekat.funkcija()
- Operatori za pristup članovima...
  - ...niza: **[]**
  - …klase, interfejsa ili prostora imena (*namespace*): **.**

Leksički elementi C# jezika
Operator new i operator zarez

- Identično kao u Javi – operator **new** služi za kreiranje novog objekta
- Slično kao u Javi – operator zarez (comma operator) **,** povezuje više izraza u jedan.

Leksički elementi C# jezika
Operatori za konverziju tipova

- Standardni operator za konverziju (poznat iz C/C++a i Jave) **()**
  - Primenjiv i nad vrednosnim i nad referentnim tipovima
  - int x;

    x = (int)12.34; // 12.34 bi inače bio double
  - Ukoliko konverzija nije moguća generiše izuzetak
- Novi operator za konverziju **as**
  - Primenjiv **samo** nad referentnim tipovima
  - Ukoliko konverzija nije moguća ne generiše izuzetak, samo vraća null
  - Object obj = "abc";

    String s = obj as String;

Leksički elementi C# jezika
Operatori za ispitivanje tipa objekta

- Umesto **instanceof** u Javi u C#u je ključna reč **is**
  - Object obj = new Point(1, 2);

    Point p;

    if (obj is Point) // vraća true ako je objekat obj instanca klase Point ili // neke od klasa izvedenih iz klase Point

      p = (Point)obj;
- Operator **typeof** u kombinaciji sa metodom GetType() iz klase Object
  - Object obj = new Point(1, 2);

    Point p;

    if (obj.GetType() == typeof(Point))) // typeof se izvršava još u vreme kompajliranja, // GetType u vreme izvršenja, a oba vraćaju objekat tipa Type

      p = (Point)obj;

Leksički elementi C# jezika
Ternarni uslovni operator

- Isto kao i u Javi ternarni operator se zadaje tokenima **?** i **:**
- Primer
  - int max(a,b) {
    if (a>b)
    return a;
    else
    return b;
    }
  - int max(a,b) {
    // isti efekat se postiže ovim operatorom:
    return (a>b) ? a : b
    }

Leksički elementi C# jezika
Null coalescing operator

- Specifičan za C#
- Piše se **??**
- Značenje: **x = a ?? b** se može zapisati i korišćenjem ternarnog operatora **x = (a != null) ? a : b**
- Primer:
  - Object obj = 1;
  - String s = obj as String ?? ""; // obj as String u ovom slučaju vraća null

Leksički elementi C# jezika
Separatori

- Separatori u C# su isti kao i u Javi
- ()
  - omeđuju logički uslov u if, while
  - omeđuju postavljanje brojača for petlje itd
- {}
  - omeđuju jedan blok koda
- ;
  - označavaju kraj jedne elementarne "naredbe" u kôdu

Upravljačke strukture

- Linearna
- Grananje
  - if (if-else)
  - switch-case-default
  - try-catch (za obradu izuzetaka)
- Petlja
  - for
  - while
  - do-while
  - foreach
- Skok
  - break
  - continue
  - return
  - throw (za obradu izuzetaka)

Upravljačke strukture
Linearna upravljačka struktura

- {
  naredba;
  naredba;
  ...
  naredba;
  }

Upravljačke strukture
If (if-else) struktura

- if (logički izraz) <naredba>
- if (logički izraz) <naredba> else <naredba>
- Else blok se uparuje sa najbližim if uslovom
- Primer:
- if (a >= 0)

```
    if (a == 0)
        Console.WriteLine("Broj je nula");
    else
        Console.WriteLine("Broj je strogo veći od nule");
```

Upravljačke strukture
Switch-case-default struktura

- switch (izraz)

  {

  case <vrednost_1>: <naredba> break;

  case <vrednost_2>: <naredba> break;

  …

  default: <naredba> break;

  }
- Obavezna je naredba break na kraju svake opcije pa čak i posle default opcije
- U suprotnom dobija se greška pri kompajliranju

Upravljačke strukture
Switch-case-default struktura

- Primer:
- switch (dan)

```
{
    case DanUNedelji.Ponedeljak:
            Console.WriteLine("Monday");
            break;
    case DanUNedelji.Utorak:
            Console.WriteLine("Tuesday");
            break;
    . . .
    case DanUNedelji.Subota:
            Console.WriteLine("Saturday");
            break;
    case DanUNedelji.Nedelja:
            Console.WriteLine("Sunday");
            break;
    default:
        Console.WriteLine("Nepostojeći dan");
            break;
}
```

Upravljačke strukture
Switch-case-default struktura

- Primer:
- DanUNedelji dan = DanUNedelji.Ponedeljak;

```
switch (dan)
{
    case DanUNedelji.Ponedeljak:
    case DanUNedelji.Utorak:
    case DanUNedelji.Sreda:
    case DanUNedelji.Cetvrtak:
    case DanUNedelji.Petak:
        Console.WriteLine("Radni dan");
        break;
    case DanUNedelji.Subota:
    case DanUNedelji.Nedelja:
        Console.WriteLine("Vikend");
        break;
}
```

- Moguće je objediniti više case opcija u jednu
- Opcija default nije obavezna

Upravljačke strukture
Strukture petlje

- For petlja
  - for(int i=1; i<10; i++) naredba;
  - for(int i=1; i<10; i++) {niz naredbi};
- While petlja
  - while (logički izraz) naredba;
  - while (logički izraz) {niz naredbi};
- Do-while petlja
  - do naredba while (logički izraz);
  - do {niz naredbi} while (logički izraz);

Upravljačke strukture
Foreach petlja

- foreach (tip promenljiva in kolekcija) naredba;
- Za sada smatramo da je kolekcija običan niz, a na narednim časovima će biti reči o još nekim tipovima podataka koji mogu predstavljati kolekcije
- Isto radi kao foreach u Javi samo što se u C# koriste ključne reči foreach i in, a u Javi for i znak :
- Primer:
- int[] niz = { 5, 10, 15, 20, 25 };

```
foreach (int element in niz)
    Console.WriteLine(element);
```

Upravljačke strukture
Naredbe skoka

- Naredba break
  - Potpuno prekida izvršenje najbliže switch, while, do, for ili foreach naredbe unutar koje se nalazi
  - Ako ima više ugnježdenih switch, while, do, for ili foreach naredbi break prekida izvršenje samo naredbe na najnižem nivou ugnježdenja (njoj najbliža naredba)
- Naredba continue
  - Prekida tekuću i počinje novu iteraciju najbliže while, do, for ili foreach naredbe u okviru koje se nalazi
  - Ako ima više ugnježdenih while, do, for ili foreach naredbi continue počinje novu iteraciju naredbe na najnižem nivou ugnježdenja (njoj najbliža naredba)

Upravljačke strukture
Naredbe skoka

- Naredba goto
  - goto labela ;
  - goto case konstanta ; // skok na opciju u okviru switch-case strukture
  - goto default ; // skok na default opciju u okviru switch-case strukture
  - Naredba bezuslovnog skoka na zadatu programsku liniju
- Naredba return
  - return; // kraj metoda tipa void
  - return izraz; // kraj metoda koji nešto vraća
- Naredba throw
  - throw (izraz); // baca izuzetak