Akademija tehničko-vaspitačkih strukovnih studija

.NET tehnologije

Prof. dr Zoran Veličković, dipl. inž. el.

2022/23.

Prof. dr Zoran Veličković, dipl. inž. el.

.NET tehnologije

Tipovi podataka u .NET-u i C#-u

Sistem i konverzija tipova

(5)

Sadržaj

- **Tipovi podataka i memorija**
  - Memorija za podatke u C#
  - Uklanjanje nekorišćenih objekata iz memeorije u C#
  - Arhitekture steka i hipa
- **Tipovi podataka u C#**
  - Karakteristike vrednosnih tipova u C#
  - Karakteristike referencnih tipova u C#
  - Hijerarhija tipova u C#
- **Sistem zajedničkih tipova u .NET**
  - Sistem zajedničkih tipova u C#
  - CTS/CLS specifikacija
  - Ugrađeni tipovi .NET-a
  - Mapiranje osnovnih C# u .NET tipove
    - Hijerarhija numeričkih .NET tipova
- **Konverzija tipova**
  - Implicitna konverzija
  - Eksplicitna konverzija
  - Konverzija brojeva u/iz stringove
    - String kao referencna promenljiva
- **Predstava tipova u memoriji**
  - Vrednostni tipovi
  - Referencni tipovi
  - Razlika između referencnih i vrednosnih tipova

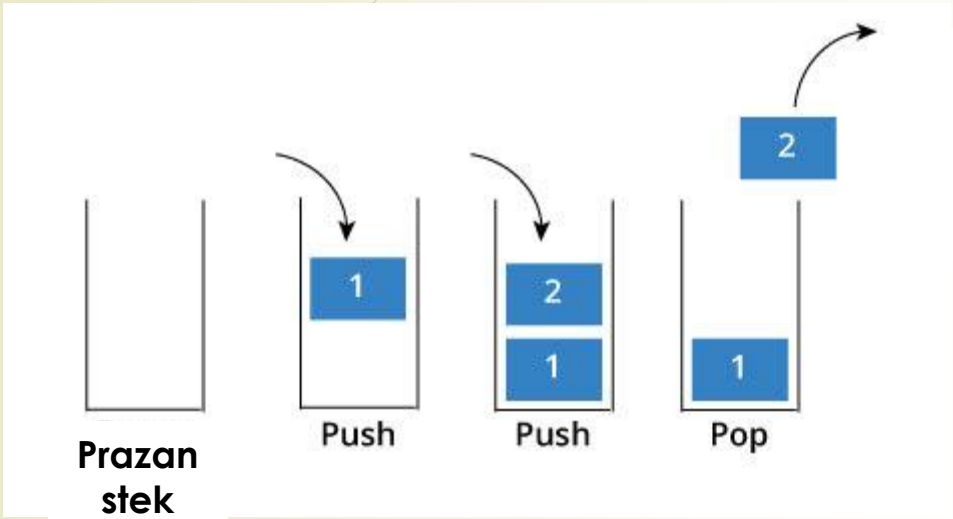
Tipovi podataka i memorija

- Već znamo da kod **OO** jezika (primer Java), pa tako i u .NET-u postoje **DVA** osnovna tipa podataka:
  - **Vrednosni** i
  - **Referencni**.
- **VREDNOSNI TIPOVI** su već **UGRAĐENI** u .NET i C# i njih korisnik ne može izgraditi već ih kao takve samo može izabrati i koristiti.
- Osnovna **RAZLIKA** između **VREDNOSNIH** i **REFERENCNIH** tipova je u **NAČINU PRISTUPA** podacima koji su njima predstavljeni.
- Za razumevanje ove razlike neophodno je poznavanje **DINAMIKE DODELE** memorije.
- **MEMORIJA ZA PODATKE** aplikacije je podeljena na **DVA** osnovna segmenta:
  - **STEK** (engl. *Stack*) segment (komponentu), i
  - **HIP** (engl. *Heap*) segment, odnosno, dinamičku memoriju.

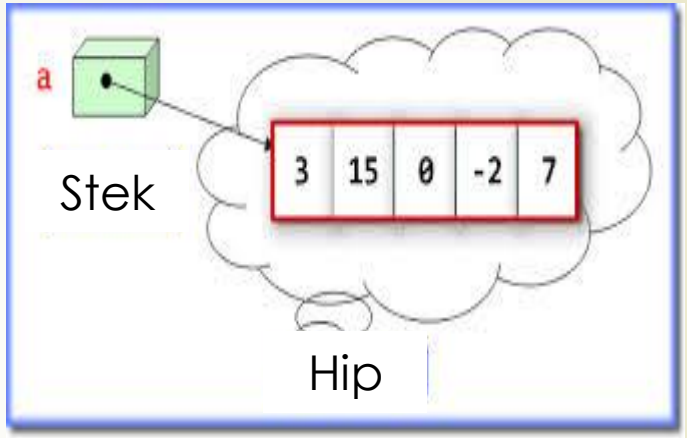
Mememorija za podatke u C#

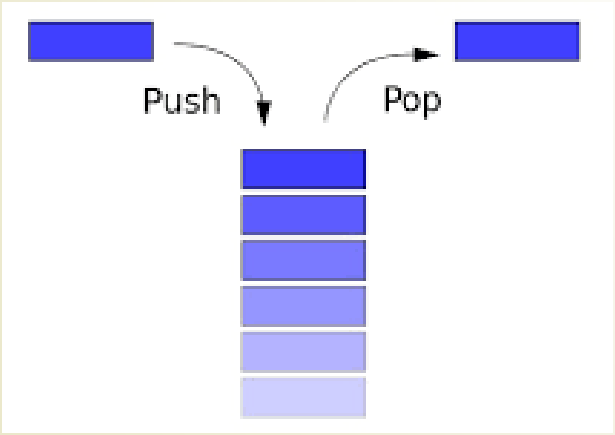
- **STEK** je specijalni deo memorijskog prostora koji je rezervisan za **IZVRŠAVANJE APLIKACIJE**.
- Kada se neka metoda pozove, **SVE PROMENLJIVE** koje ona koristi se **POSTAVLJAJU** (kaže se i „guraju“) na **STEK**.
- Ako pozvana funkcija poziva neku drugu funkciju, takođe, i **ONA** postavlja svoje promenljive na **STEK**.
- Kada se završi funkcija koja je zadnja pozvana, sve njene promenljive **IZLAZE IZ PODRUČJA VAŽENJA**, i bivaju “SKINUTE” sa steka (tako da se taj deo steka može **ponovo koristi** od strane drugih metoda).
- Sa druge strane, **HIP** je **ODVOJENI** memorijski prostor za pravljenje **OBJEKATA** koji se mogu ponovo (kaže se i iznova) koristiti.
- **KO** (ili šta?) upravlja alokacijom (rezervacijom) memorije u .NET-u, odnosno, C#-u?
  - Odgovor: **CLR** upravlja alokacijom **HIP MEMORIJE** za objekte i kontroliše **uklanjanje objekata** koji se više ne koriste.

Arhitekture steka i hipa

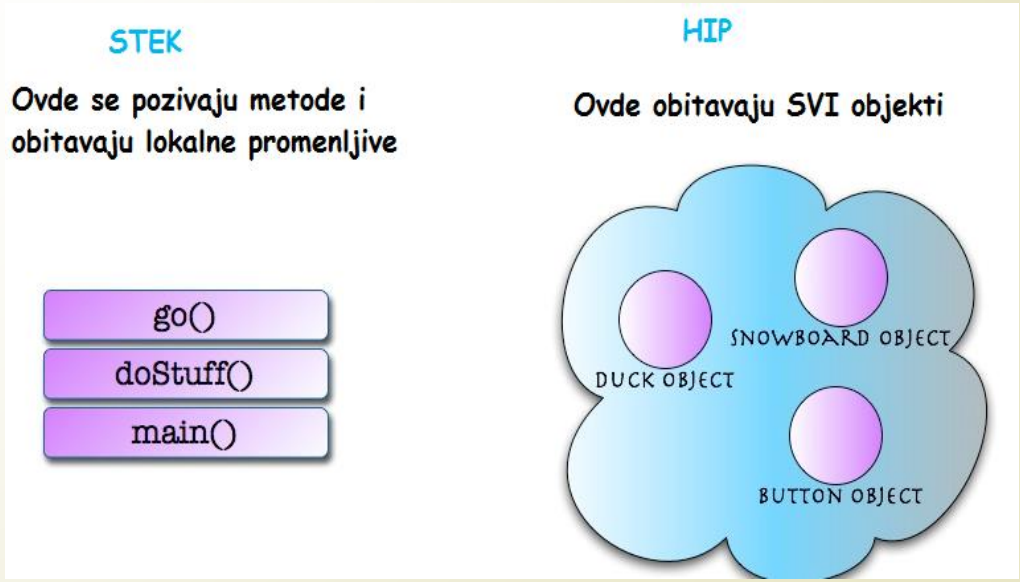


Ovo **je prvi objekt** koji se može koristiti iz steka





Metode za rad sa stekom



Karakteristike statičke/dinamičke memorije

PARAMETAR	STEK	HIP
Tip strukture podataka	**Linearna** struktura podataka.	**Hijerarhijska** struktura podataka
Brzina pristupa	**Velika**	**Sporija** u komparaciji sa stekom
Upravljanje memorijskim prostorom	**OS** upravljanje memorijskim prostorom tako da memorija nikada **neće** postati fragmentirana.	Hip prostor se ne koristi efikasno. Može doći do **fragmentiranja** memorije na blokove koji se prvo dodeljuju, a potom oslobađaju.
Pristup	Samo **lokalne** promenljive	Omogućava se **globalan** pristup promenljivama.
Ograničenje veličine prostora	Ograničenje veličine steka zavisi od **OS**-a.	**Nema** ograničenja veličine memorijskog prostora.
Promena veličine	Promenljive **ne mogu** promeniti veličinu.	Promenljive **mogu** promeniti veličinu.
Dodela memorije	Memororija se rezerviše kao **kontinualni** blok.	Memororija se rezerviše **slučajnim** rasporedom.
Dodela i oslobađanje	**Automatski** kompajlerskim naredbama.	**Ručno** programerskom intervencijom.
Oslobađanje	**Ne zahteva** se oslobađanje memorije.	**Zahteva** se eksplicitno oslobađanje memorije
Cena	**Manja**	**Veća**
Implementacija	Stek se može implementirati na **3** načina: 1) nizom, 2) dinamičkom memorijom i 3) povezanom listom.	Hip se može implementirati pomoću: 1) **niza** ili 2) **stabala**.
Glavni problem	**Nedostatak** memorije	**Fragmentacija** memorije
Fleksibilnost	Fiksna veličina	Moguća promena veličine
Vreme pristupa	Brže	Sporije

Uklanjanje nekorišćenih objekata u C#

- **UKLANJANJE** nekorišćenih objekata u **OO** jezicima se obavlja **AUTOMATSKI**, a naziva se **SKUPLJANJE ĐUBRETA** (engl. *Garbage collection*).
- Svi podaci koji su predstavljeni **VREDNOSNIM TIPOM** smeštaju se na **STEK**.
- Kada promenljiva **VREDNOSNOG TIPA** **IZAĐE** iz oblasti važenja, ona se **UNIŠTAVA** i njemu pripadajuća memorija se **USTUPA** (operativnom sistemu ili programskom jeziku) na korišćenje (tako da se može se dodeliti drugim promenljivama).
- Sa druge strane, promenljive **REFERENCNOG TIPA** postoje na **DVE ODVOJENE** memorijske lokacije.
- "**PRAVI PODACI**" se nalaze u **HIP**-u, dok se promenljiva koja sadrži **POKAZIVAČ** (u OO terminologiji **referenca**) na objekat nalazi na **STEKU**.
- Kada se pozove REFERENCNA PROMENLJIVA, pozivaocu se zapravo vraća **MEMORIJSKA ADRESA** objekta na koji se ukazuje.

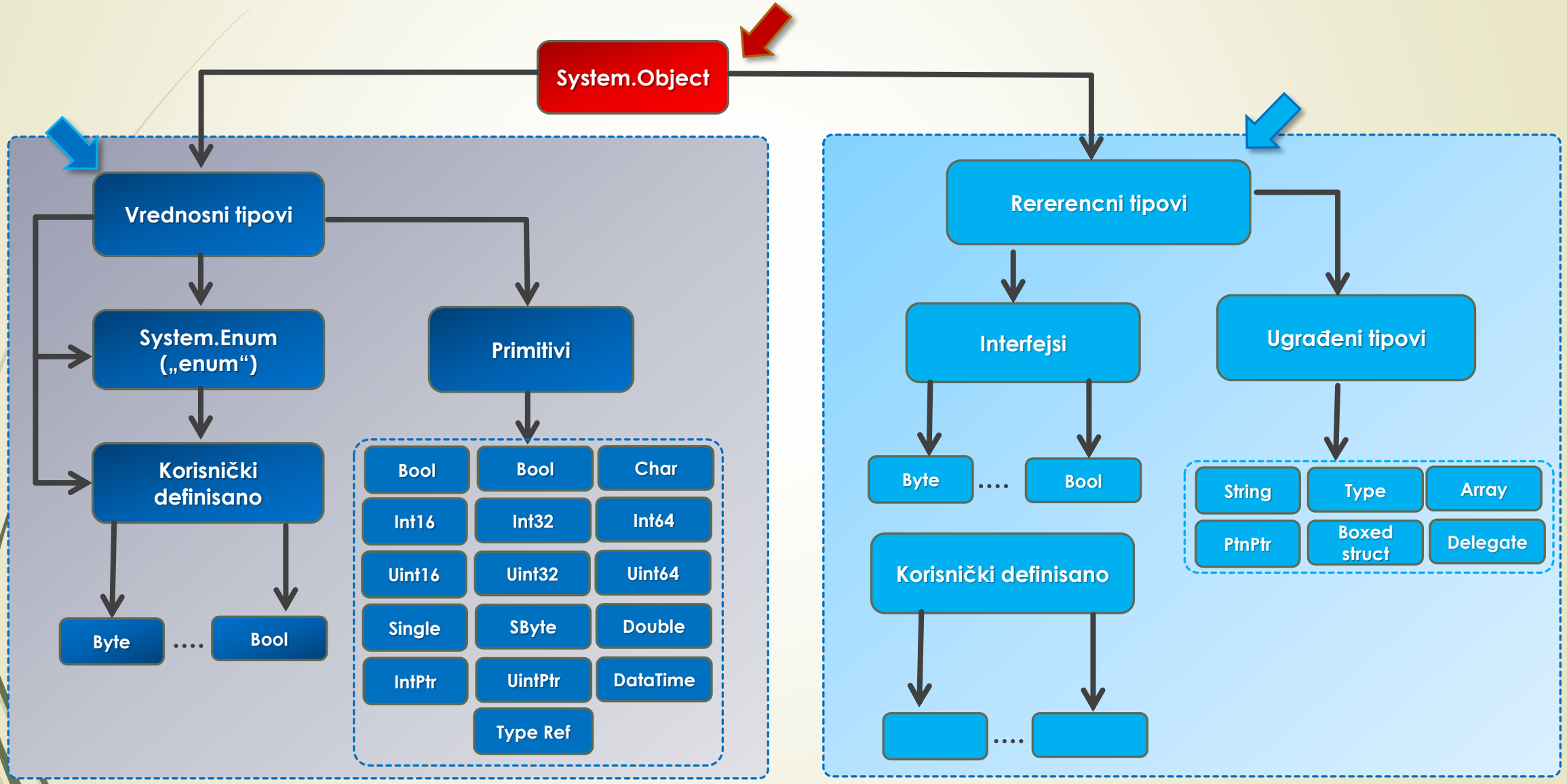
Karakteristike vrednosnih tipova u C#

- Kada promenljiva izađe iz oblasti važenja, **REFERENCA OBJEKTA** se **UNIŠTAVA**, ali **NE** i **SAM OBJEKT**!
- Ako postoji neka **DRUGA REFERENCA** na taj objekt (koja nije uništena), njemu se može pristupiti pomoću nje bez problema (pogledajte primer na poslednjem slajdu današnjeg predavanja).
- Tek ako objekat **NE POSEDUJE** DRUGE REFERENCE on je **PODLOŽAN** skupljanju đubreta.
- Primer generičkih - **VREDNOSNIH TIPOVA** uključuju primitivne tipove:
  - **Integer (int C#)**
  - **Boolen (boolC#)**
  - **Char (char C#)**
  - **Structure (struct C#)**
  - **Enumeration (enum C#) …**
- U zagradama se nalaze **rezervisane** – ključne reči **C#** jezika za dekleraciju **VREDNOSNIH TIPOVA**.

Karakteristike referencnih tipova u C#

- Predstavnici **REFERENCNIH TIPOVA** su:
  - **Klase**,
  - **Interfejsi**,
  - **Delegati**,
  - **Nizovni tipovi**.
- **SVI TIPOVI** u .NET-u pa i u C# (bilo vrednosni bilo referencni) potiču od **GENERIČKOG OBJEKTA** pod nazivom **`Object`**.
- Otuda potiče **JEDNAKO PONAŠANJE** svih objekata i u C#.
- Ovaj **GENERIČKI OBJEKT** - **`Object`** se nalazi u imenskom prostoru **`System`**.
- O **IMENSKIM PROSTORIMA** C#-a i .NET-a nešto više na sledećim predavanjima.
- Pogledajte na sledećem slajdu kako izgleda **HIJERARHIJSKA STRUKTURA** svih tipova u .NET-u.

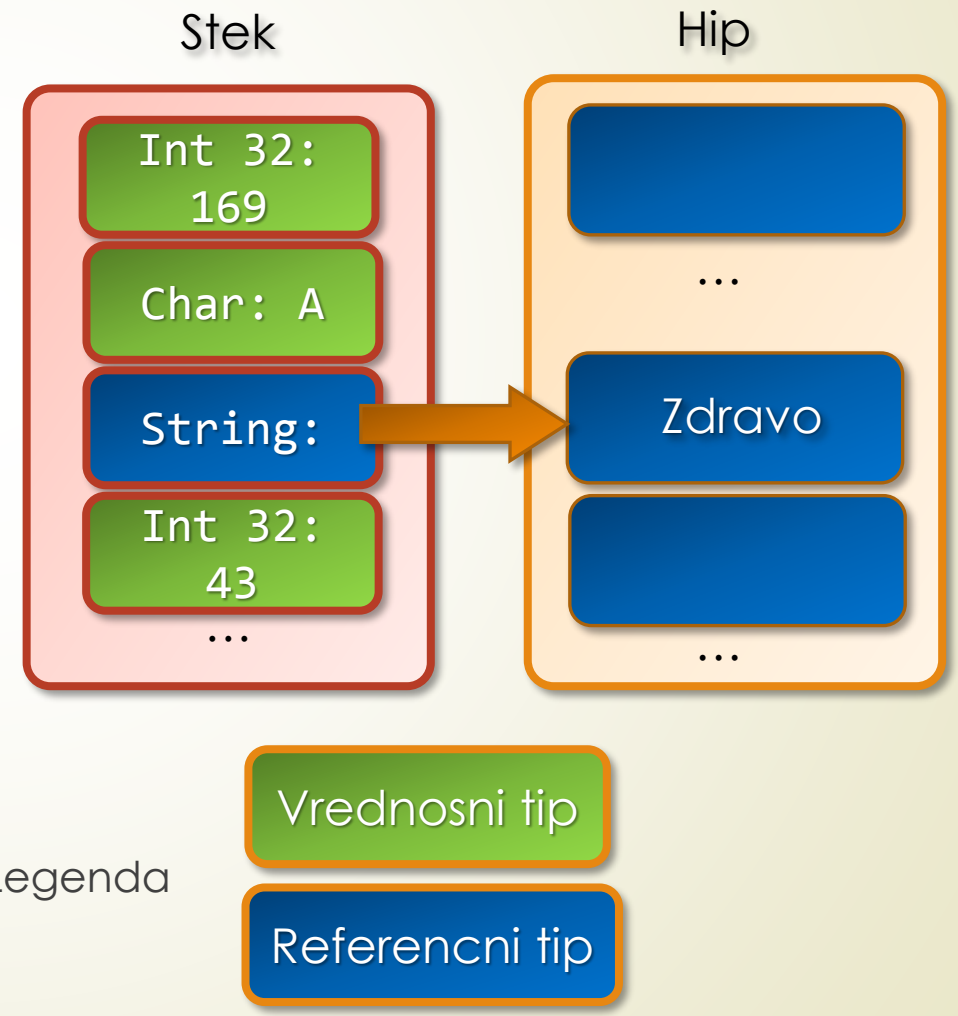
Hijerarhija tipova u C# (1)



Hijerarhija tipova u C# (2)

Predstavljanje **REFERENCNIH** i **VREDNOSNIH** promenljivih u memoriji

Memorija je podeljena na dva dela: **HIP** i **STEK**



Sistem zajedničkih tipova u .NET-u

- Već znamo, SISTEM ZAJEDNIČKIH TIPOVA **CTS** (engl. *Common Type System*) obezbeđuje da kod napisan na **JEDNOM** .NET programskom jeziku može da komunicira sa kodom napisanim na **DRUGOM**!
- Ovo je realizovano **FORMALNOM SPECIFIKACIJOM** sistema zajedničkih tipova CTS koja opisuje kako su tipovi **DEFINISANI** i kako se **PONAŠAJU**.
- ZAJEDNIČKA SPECIFIKACIJA TIPA - **CTS** treba da opiše sledeće članove:
  - **POLJA** (promenljiva podatka) predstavlja **deo stanja** objekata.
  - **METODE**, funkcija koja preduzima neku operaciju nad objektom.
    - Poseduje **IME**, **POTPIS** i **MODIFIKATORE**.
    - POTPIS određuje konvenciju pozivanja: **BROJ** parametra, **TIPOVE** parametara i **TIP** rezultata koji vraća metoda.
  - **SVOJSTVA**, u zavisnosti sa koje se strane posmatra, može izgledati i kao **polje** ili kao **metoda**.
    - Određuje **stanje** objekta PRE nego što mu se pristupi.

Sistem zajedničkih tipova u C# (1)

- **CTS** obezbeđuje **PRAVILA ZA VIDLJIVOST** i **PRISTUPANJE** članovima nekog C# tipa.
- Tako, označavanjem nekog tipa **kao javnog** (engl. *public*), on postaje **VIDLJIV** i **PRISTUPAČAN SVIM METODAMA** iz **BILO KOG SKLOPA**.
- Tip **`internal`** je vidljiv samo unutar **ISTOG SKLOPA**.
- Dakle, **TIPOM** se proverava da li "pozivaoc" **IMA PRAVO PRISTUPA** njegovim članovima.
- Opcije za pristup u **.NET**-u su:
  - **`private`**, **`family`** (**protected** u C#),
  - **`family i assembly`** (C# ne poseduje ovaj tip),
  - **`assembly`** (**internal** u C#),
  - **`family`** ili **`assembly`** (u C# **protected internal**),
  - **`public`**.

Sistem zajedničkih tipova u C# (2)

- **SINTAKSA** upotrebljenog programskog jezika u .NET-u može biti **RAZLIČITA**, ali je **PONAŠANJE TIPOVA IDENTIČNO**!
- Već je napomenuto, u C#-u tip **`Object`** je osnova svih drugih tipova i poseduje **MINIMALNI SKUP PONAŠANJA** koje **karakterišu tipove** u .NET okruženju.
- ZAJEDNIČKA SPECIFIKACIJA JEZIKA **CLS** (engl. *Common Language Specification*) je **MINIMALNI SKUP** ODLIKA koje **KOMPAJLERI** koji podržavaju .NET **MORAJU POSEDOVATI**.
- **NADSKUP/PODSKUP** karakteristika programskih jezika u .NET okruženju je prikazan na sledećoj slici.

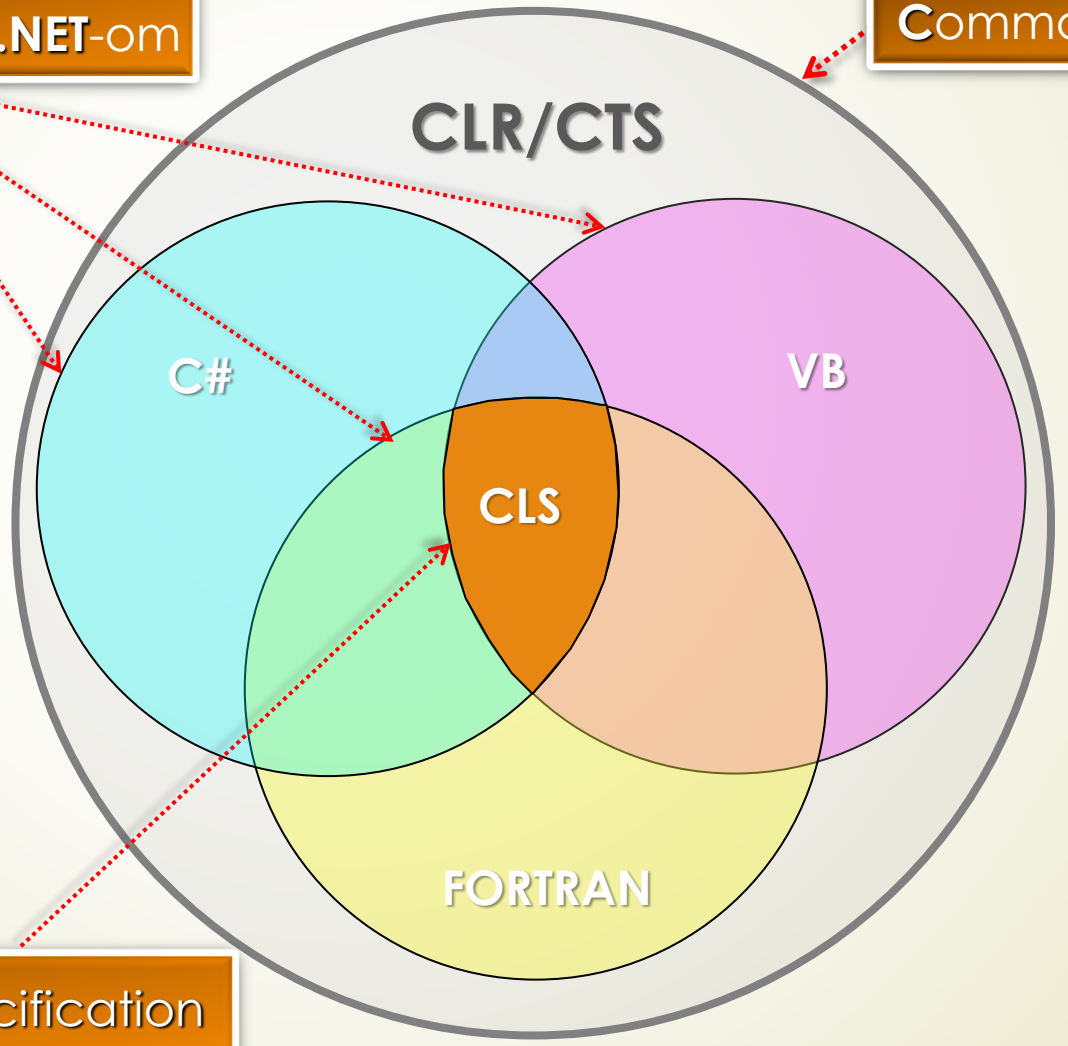
CTS/CLS

Jezici podržani **.NET**-om

Svaki podržani programski jezik poseduje **SVOJE** tipove

Svaki podržani programski jezik mora da poseduje **MINIMALNI SKUP** zajedničkih tipova

Common **L**anguage **S**pecification



Common **T**ype **S**ystem

CTS tipovi
Byte
Char
Boolean
SByte
Int16
UInt16
Int32
UInt32
Single
Double
Decimal
Int64

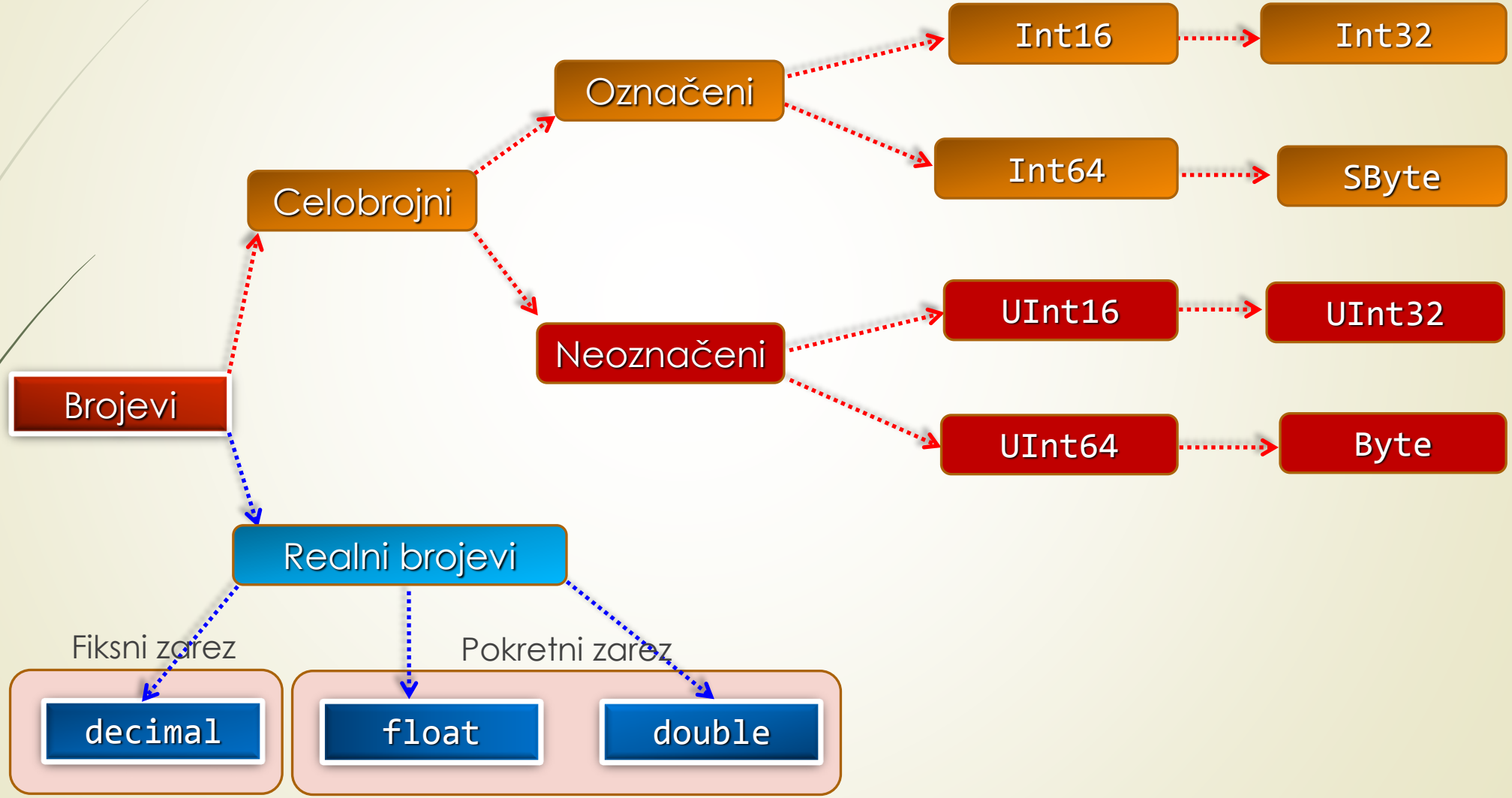
Ugrađeni tipovi .NET-a (1)

- C# sadrži **UGRAĐENU PALETU TIPOVA** koja odgovara **CLS** zahtevima.
- **PRESLIKAVANJE** (engl. *mapping*) **osnovnih tipova** iz C# u **.NET TIPOVE** omogućava **PREPOZNAVANJE** - RAZMENJIVANJE OBJEKATA formiranih drugim .NET jezicima koji podržavaju CLS.
- Zapamtite, SVAKI TIP ima **POZNATU** i **NEPROMENLJIVU** veličinu!
- Pored OSNOVNIH (ugrađenih) tipova C# nudi još **DVA VREDNOSNA TIPA**:
  - **`Enum`** (nabrojivi tipovi – već obrađeni) i
  - **`Struct`** (biće obrađeni zajedno sa klasama na na lab. vežbama).
- Setite se, prilikom izbora celobrojnog tipa treba voditi računa o **NAJVEĆOJ VREDNOSTI** koja se želi da sačuva.
- Za tipove **`float`**, **`double`** i **`decimal`** u C# se može se birati:
  - **veličina** ali i
  - **preciznost**!

Mapiranje osnovnih **C#** u **.NET** tipove

C# TIP	VELIČINA (B)	.NET TIP	OPIS
`byte`	1	`Byte`	Neoznačene celobrojne vrednosti od 0 **do** 255.
`char`	2	`Char`	Unicode karakteri.
`bool`	1	`Boolean`	Istina ili laž.
`sbyte`	1	`SByte`	Označene celobrojne vrednosti od -128 **do** 127.
`short`	2	`Int16`	Označene celobrojne vrednosti (short) od -32,768 **do** 32,767).
`ushort`	2	`UInt16`	Neoznačene celobrojne vrednosti (short) od 0 **do** 65,535.
`int`	4	`Int32`	Označene celobrojne vrednosti između -2,147,483,648 i 2,147,483,647.
`uint`	4	`UInt32`	Označene celobrojne vrednosti između 0 i 4,294,967,295.
`float`	4	`Single`	Floating point brojevi. Vrednosti od aproksimativno +/-1.5 * 10^{-45} **do** aproksimativno +/-3.4 * 10^{38} sa 7 značajnih cifara.
`double`	8	`Double`	Floating point vrednosti sa **dvostrukom preciznošću**; vrednosti od aproksimativno +/-5.0 * 10^{-324} **do** aproksimativno +/-1.8 * 10^{308} sa 15-16 značajnih cifara.
`decimal`	16	`Decimal`	Fiksna pozicija decimalne tačke, preciznost do 28 cifara. Tipično se koristi u finansijskim izračunavanjima. Zahtevaju sufix "`m`" ili "`M`": `decimal myMoney = 300.5m;`
`long`	8	`Int64`	Označene celobrojne vrednosti u opsegu od -9 223 372 036 854 775 808 **do** +9 223 372 036 854 775 807

Hijerarhija numeričkih **.NET** tipova



Konverzija tipova (1)

- Objekti jednog tipa se mogu **KONVERTOVATI** u drugi tip i to:
  - **EKSPLICITNO** ili
  - **IMPLICITNO**.
- **IMPLICITNA KONVERZIJA** se odvija **AUTOMATSKI** i o njoj vodi računa sam **KOMPAJLER**.
- Pri implicitnoj konverziji **NE DOLAZI DO GUBITAKA** podataka.
  - Vrednost tipa `short int` (2 bajta) se **MOŽE** implicitno konvertovati u `int` tip (4 bajta).
  - Kod **obrnute** konverzije rizikuje se da se **IZGUBI** deo podataka.
- Kod **EKSPLICITNE KONVERZIJE**, korisnik sam "**zahteva**" od prevodioca da obavi konverziju, iako, ona može potencijalno dovesti do problema.
- Za posledice ekplicitne konverzije odgovara **PROGRAMER**.
- Operator eksplicitne konverzije naziva se **KAST** (engl. *cast*) **OPERATOR**.
  - Primer: `MyInt = (int) myLong`

Konverzija tipova (2)

- Primer implicitne konverzije, **OK**!
  - `short x = 5;`
  - `int y = x;` // implicitna konverzija
- Primer implicitne konverzije, **LOŠA**!
  - `short x;`
  - `int y = 500;`
  - `x = y;` // kompajler ovo ne ume i neće da uradi! Zašto?
- Tip `char` predstavlja znak iz **Unicoda** (16 bitova)!
- **LITERALI** tipa `char` mogu biti:
  - **Prosti**, (A,B, ...)
  - **Unicode znaci** (`\u0041`, `\u0042`, ...)
  - **Izlazne sekvence** (escape karakteri)
- Postavljanjem simbola **@ ISPRED STRINGA** daje slično rešenje: `path=@"c:\MyApp\MyFiles"`

Char	Značenje
\'	Single quote
\"	Double quote
\\	Backslash
\0	Null
\a	Alert
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab

Konverzija stringova u brojeve

```
string countString = "10";

// Konvertuj string "10" u numeričku vrednost 10.
int count = Convert.ToInt32(countString);


// Konvertuj numeričku vrednost 10 u string "10".
countString = Convert.ToString(count);
```

Korišćenje statičkih metoda klase **Convert**

ISTI REZULTAT

```
int count;
string countString = "10";
// Konvertuje string "10" u numeričku vrednost 10.
count = Int32.Parse(countString);
```

Korišćenje statičke metode **Int32.Parse()**

String kao referencna promenljiva

Deklaracija NIZA STRINGOVA u C#

```
static void Main(string[] args)
 {
     string[] stringArray = {"jedan", "dva", "tri"};

     for (int i = 0; i < stringArray.Length; i++)
      {
         System.Diagnostics.Debug.Write(stringArray[i] + " ");
      }

 }
```

```
Izlaz:
jedan dva tri
```

Dodeljivanje vrednosti (1)

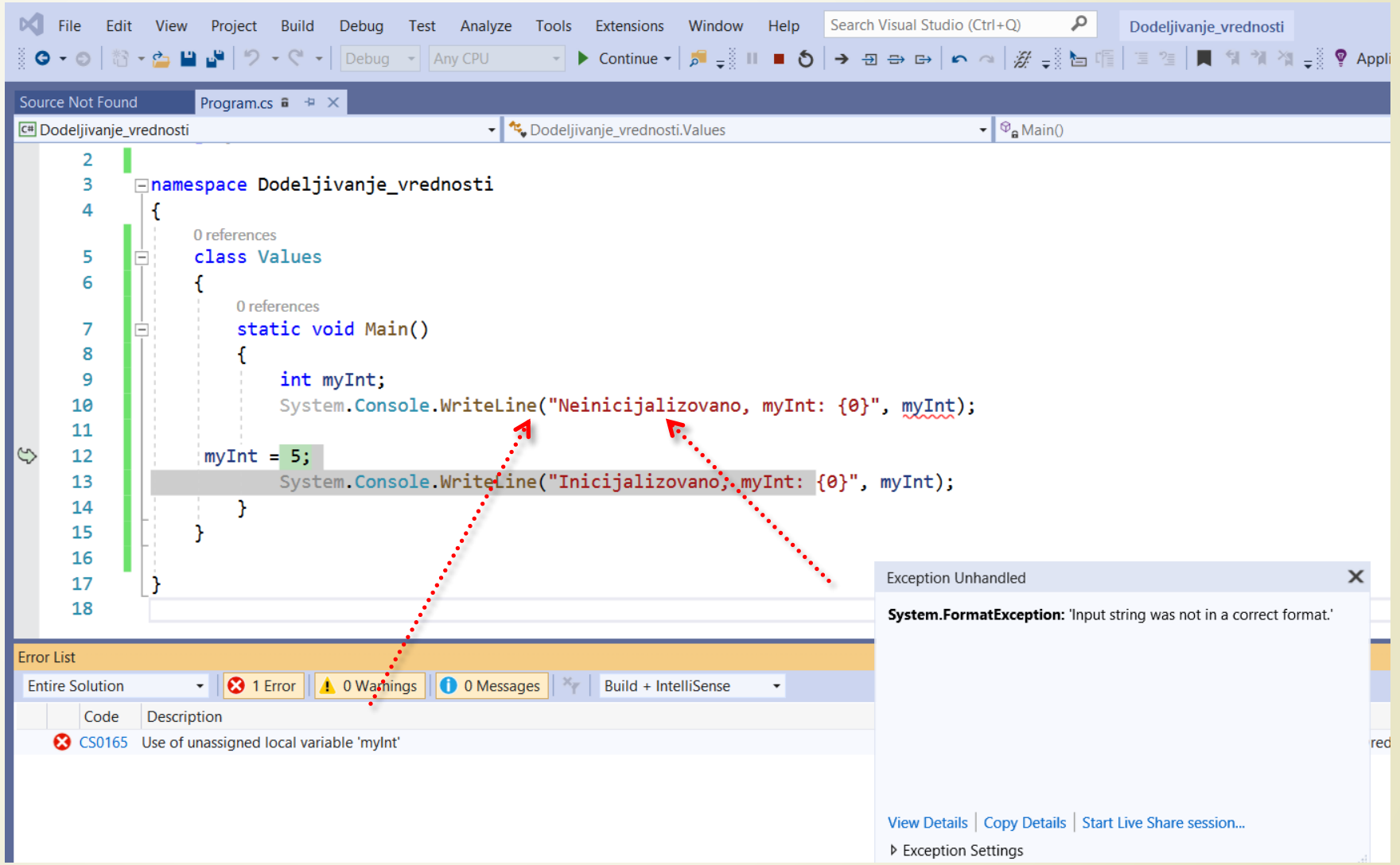
```
class Value
    {
        static void Main(string[] args)
        {
            int myInt = 7;
            System.Console.WriteLine("Inicijalizovano, myInt: {0}", myInt);
            myInt = 5;
            System.Console.WriteLine("Posle dodeljivanja  myInt: { 0}", myInt);
        }
    }
```

Dodeljivanje vrednosti promenljivoj **myInt**: prvo 7, a potom 5.

```
Inicijalizovano, myInt: 7
Posle dodeljivanja  myInt: 5
```

Dodeljivanje vrednosti (2)

- Svaki pokušaj pristupu neinicijalizovanoj promenljivoj izaziva **grešku** ili u terminologiji OO programiranja izuzetak - **exeptions**!
- Obrada izuzetaka u C# će biti posebno obrađena.

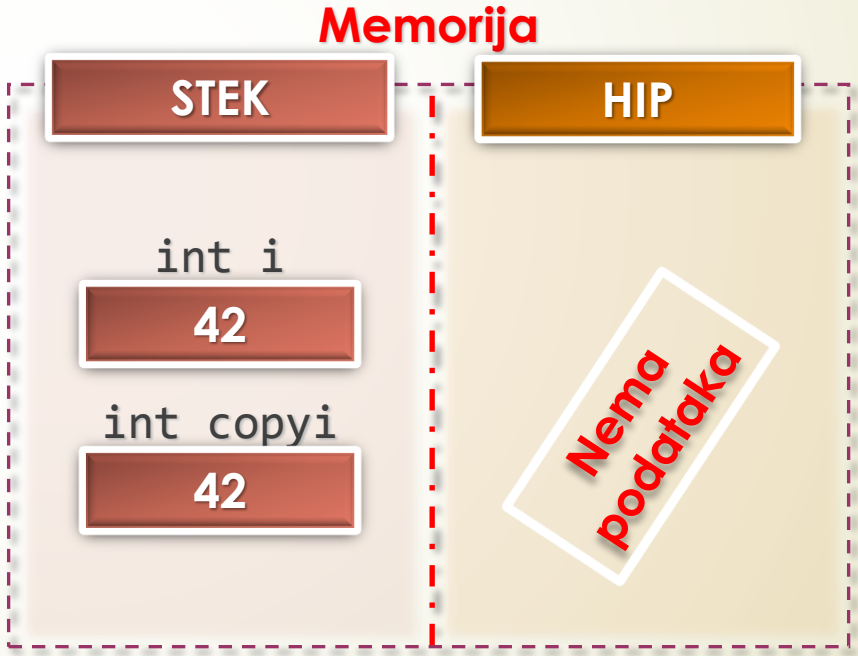


Predstava vrednostnih tipova u memoriji (1)

```
int i = 42;          // dekleracija i inicijalizacija promenljive i
int copyi = i;       // promenljiva copyi sadrži kopiju podataka iz lokacije i
i++;                 // inkrementiranje promenljive i nema efekta na promeljivu copyi
```

```
int i = 42;
i=42;


int copyi = i;
copyi = i;
```



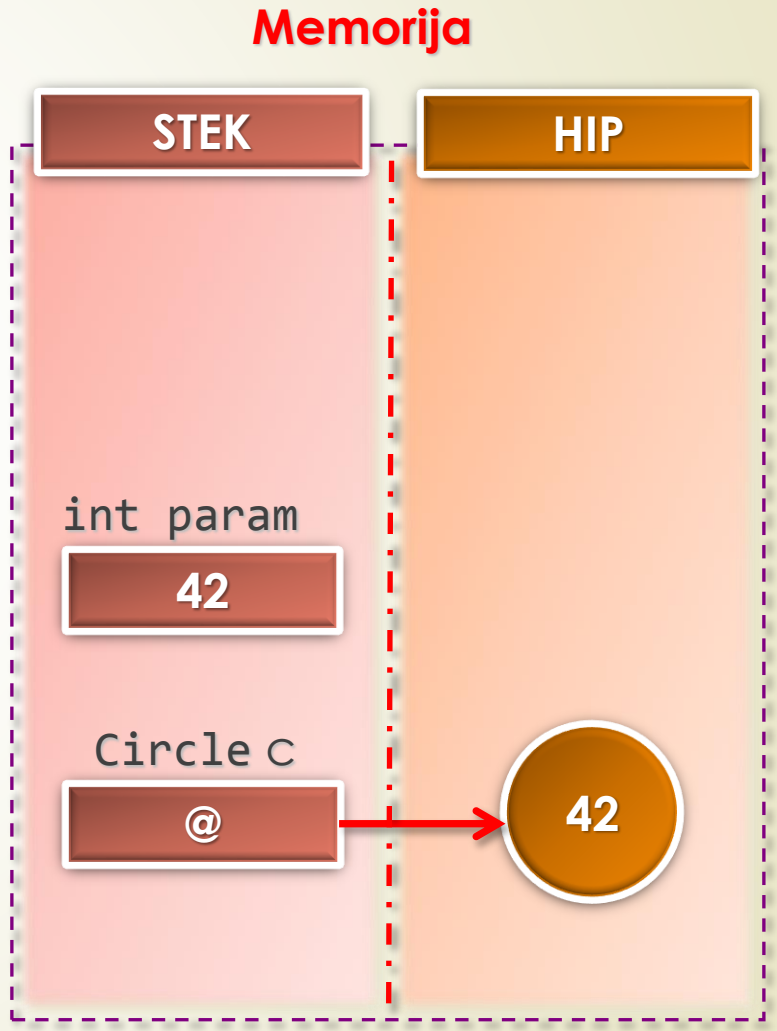
- Ovo je **UOBIČAJEN** način rada sa **VREDNOSNIM TIPOVIMA** podataka, tu nema nikakvih izmena u .NET-u.

Predstava referencnih tipova u memoriji (1)

```
void Method(int param)
   {
        Circle c;
        c = new Circle(param);
        ...
   }
```

```
int param = 42
void Metod (int param)
  {
       Circle c;
       c = new
       Circle(param);
            ...

   }
```



Predstava referencnih tipova u memoriji (2)

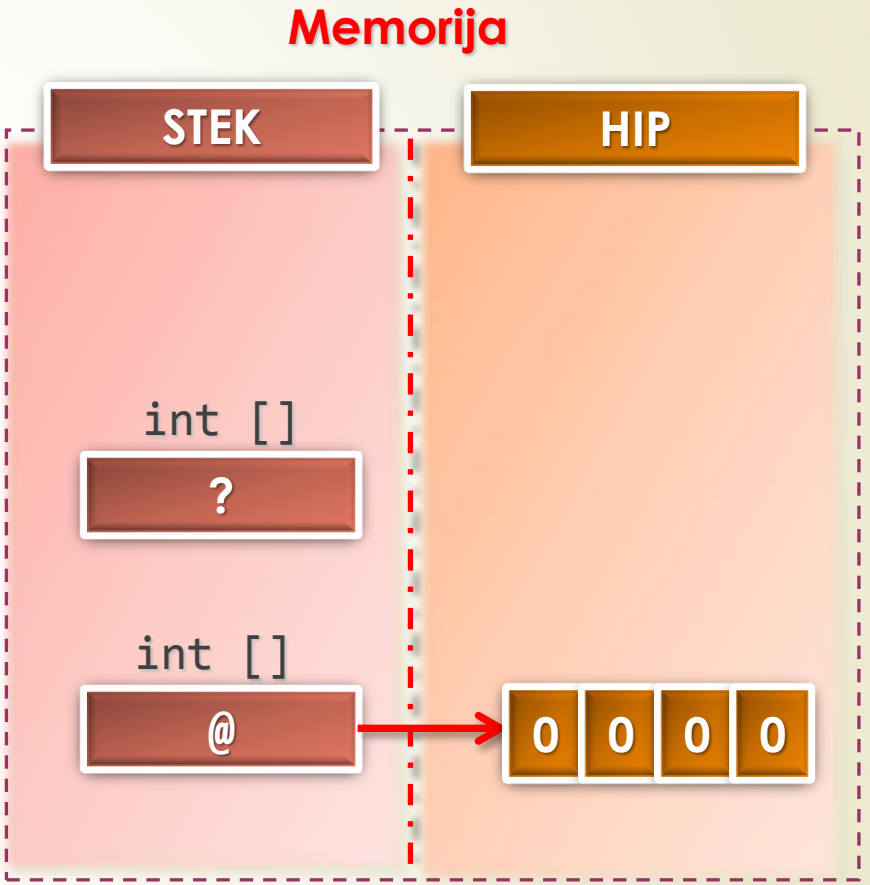
// dekleracija
integer niza

```
pins = new int[4];
```

```
. . .

int [] pins;


pins = new
int[4];
```



- NEINCIJALIZOVANE, ali deklarisani integer nizovi od strane korisnika se **PODRAZUMEVANO** postavljaju na vrednost **0**.

Razlika između referencnih i vrednosnih tipova

- Ovo je **UOBIČAJEN** način rada sa vrednosnim tipovima podataka, tu nema nikakvih izmena u .NET-u.

```
int x, y;
x = 15;
y = x;
x = 30;
// koja je vrednost promenljive y?
```

Vrednosni tip, **x**

y=15

```
System.Windows.Form x, y;
x = new System.Windows.Form();
x.Text ="Ovo je Forma 1";
y = x;
x.Text ="Ovo je Forma 2";
// Koja je vrednost promenljive y.Text?
```

Referencni tip promenljive, **x**

Formira se referenca na **ISTI** objekat.

y.Text="Ovo je forma 2"