STRUKTURA C# PROGRAMA

Struktura programa

C# program se sastoji od:

- Namespace deklaracije
- Klase
- Metoda Klase
- Atributa Klase
- Main metode
- Naredbi i izraza
- Komentara

```
using System;
using BIBLIOTEKA;

namespace IME_NAMESPACEA
{
    class IME_KLASE
    {
     OPCIONE_DEKLARACIJE_PROMENLJIVIH_I_METODA

        static void Main(string[] args)
        {
            NAREDBE
        }

     OPCIONE_DEKLARACIJE_PROMENLJIVIH_I_METODA
    }
}
```

Namespace & Using

- Prva linija C# koda najčešće je **using System**;
- Using ključna reč koristi se da uključi Sistemski **Namespace** u vaš Program

- Namespace **(imenski prostor)** je u suštini kolekcija klasa

- Program uglavnom uključuje veći broj imenskih prostora
- Omogućava stvaranje imenskih prostora i izbegavanja konflikta imena

- Vrlo slično Java paketima (uz sitne razlike):
  - Struktura foldera ne mora da odgovara strukturi Namespacea
  - 1 fajl može da sadrži više namespaceova

Prekvalifikacije za IT

Namespace & Using

- Postoji mogućnost da se više programskih konstrukta (npr. klasa) nazove istim imenom, dok su u različitim Namespaceovima

```
namespace first_space {
   class namespace_cl {
      public void func() {
         Console.WriteLine("Inside first_space");
      }
   }
}
namespace second_space {
   class namespace_cl {
      public void func() {
         Console.WriteLine("Inside second_space");
      }
   }
}
class TestClass {
   static void Main(string[] args) {
      first_space.namespace_cl fc = new first_space.namespace_cl();
      second_space.namespace_cl sc = new second_space.namespace_cl();
      fc.func();
      sc.func();
      Console.ReadKey();
   }
}
```

Namespace & Using

- Postoji mogućnost i ugnježdenog definisanja Namespacea:

```
namespace namespace_name1 {

   // code declarations
   namespace namespace_name2 {
      // code declarations
   }
}
```

- Ključna reč **using** govori da program koristi imena iz Namespacea, npr:
  - Umesto System.Console.WriteLine("Hello there");
  - Pišemo samo Console.WriteLine("Hello there");
  - Jer smo program započeli sa Using System
- Možemo koristiti i tzv. Alias
  - using Alias = System.Console;
  - Alias.WriteLine("Hi");

Klasa

- Fundament objekto orjentisanog programiranja
- Veći značaj imaće na sledećem kursu
- Klase sadrže metode i atribute koji određuju klasu
- Trenutno imamo jednu klasu, i samo jednu (main) methodu

Main metoda

```
using System;
using System.Collections.Generic;

namespace PrvaAplikacija
{
   class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Zdravo Svete!");
            Console.ReadLine();
        }
    }
}
```

- Početna tačka svakog programa
- Postoji samo jedna **main** metoda
- Glavna metoda navodi ono što klasa radi kada se izvršava.

Komentari

- Komentari se koriste za objašnjavanje koda
- Kompajleri ignorišu sve komentare, oni su tu radi programera
- Postoje jednolinijski i višelinijski komenatari

- Višelinijski C# komentar počinje se /* završava se sa */ i između toga može da piše sve što programeru olakšava rad

```
/* U komentarima u C# može da stoji mnogo toga,
 uključujući i sintaksno neispravan kod (a?=b-a/a_=
 ili kukučava slova, или чак и ћирилица*/
```

- Jednolinijeki komentar počinje sa //

```
// Ovo je moj prvi komentar
```

SINTAKSA C# PROGRAMSKOG JEZIKA

Alfabet

```
~ ! # % & * ( ) - _ + = [ ]
{ } : ; ‘ “ ? < > / , \ |
```

- ali i kombinacije

```
==   --   ++   >>   <<
```

- plus **SLUŽBENE REČI**
- plus `null`, `true`, `false`

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	goto	if	implicit	in	in (generic modifier)	int
interface	internal	is	lock	long	namespace	new
null	object	operator	out	out (generic modifier)	override	params
private	protected	public	readonly	ref	return	sbyte
sealed	short	sizeof	stackalloc	static	string	struct
switch	this	throw	true	try	typeof	uint
ulong	unchecked	unsafe	ushort	using	virtual	void
volatile	while					

Identifikatori ...

- Fundamentalna stvar u programiranju
- Služe za imenovanje raznih stvari u programu (imenskih prostora, klasa, metoda, promenljivih ...)
- Jako je važno razumeti kako se daju imena
- Dobar programer koristi i deklariše identifikatore na konzistentan način (uvek dosledno)
- Identifikator je sekvenca karaktera, brojeva i donje crte ('_')
- Identifikator započinje slovom ili donjom crtom

```
N   n   _rate   x15   quite_a_long_name   HelloWorld
```

... Identifikatori

- U identifikatoru nisu dozvoljena prazna mesta (space)

```
ovoJeValidanIdentifikator
ovi nije Validan identifikator
```

- U C# se pravi razlika između malih i velikih slova (case-sensitive)

`HelloWorld`, `helloworld`, `HELLOWORLD` i `hElloWorLD` su sve validni identifikatori, ali različiti

- Rezervisane reči ne mogu biti identifikatori (`class`, `public`, `static`, `if`, `else`, `while` i ostale)
- Identifikatori koji se sastoje od više reči mogu koristiti crticu za njihovo razdvajanje ili se svako početno slovo reči osim kod prve piše veliko: **`helloWorld`**

Notacije

- Savetuje se odabiranje jedne i držanje te iste tokom kodiranja
- Najpopularnija je tzv. Kamilja notacija
- Popularna je i Mađarska notacija
- C++ ima specifičnu notaciju itd. (počinje donjom crtom)

- Sve u svemu, treba odabrati jednu i se držati konvencija (prema programskom jeziku)

Code Snippets

- Skraćenice koje olakšavaju kodiranje
- Ima ih jako mnogo, izaberite svoje i ubrzajte rad *(cw, try, for, if...)*

- Primer:
  - kucajte cw
  - Dva puta pritisnite tab na tastaturi
  - Dobićete Console.WriteLine()

- Lista (većine opšte prihvaćenih snipeta): https://msdn.microsoft.com/en-us/library/z41h7fat.aspx

Prekvalifikacije za IT

PROMENLJIVE

Promenljive ...

- Program transformiše ulazne podatke u izlazne (rezultat)
- Mogu se koristiti i međurezultati
- Podaci se čuvaju u promenljivama, slično opštem broju u matematici
- Predstavljaju i mesto u memoriji
- Dodela vrednosti (nije isto što i izjednačavanje u matematici)

```
a = 5;      b = a + 13;      a = a + 2;
```


... Promenljive ...

- Dodela vrednosti (nije isto što i izjednačavanje u matematici), jer je promenljiva i mesto u memoriji
- Prvo se pročita stara vrednost, te se ona uvede u izraz, nakon čega se rezultat izraza upiše umesto stare vrednosti

```
a = 3;
a = a + 2;
        a + 2
        3 + 2
        5
a = 5;
```

... Promenljive ...

```
static void Main(string[] args)
{
int a = 3;
int b;
Console.WriteLine("Vrednost promenljive a je:");
Console.WriteLine(a);
b = a + 12;
Console.WriteLine("Vrednost promenljive b=a+12 je:");
Console.WriteLine(b);
a = a + 2;
Console.WriteLine("Vrednost promenljive a=a+2 je:");
Console.WriteLine(a);
}
```

Posle svega saberite a i b, rezultat sačuvajte u b i prikažite b

... Promenljive

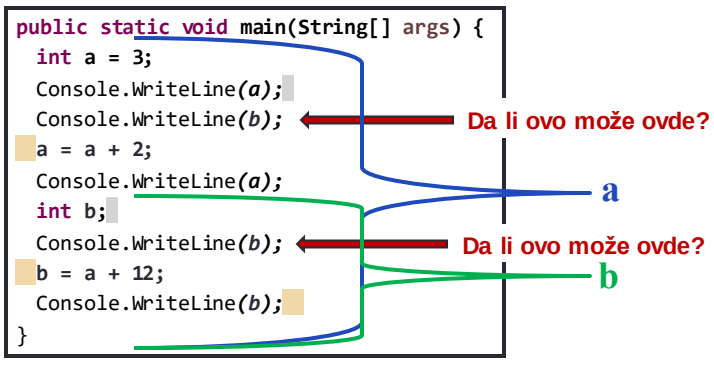
- Za svaku promenljivu definiše se:
  - **Naziv**, identifikator, preko koje se identifikuje i koristi, mora imati deskriptivnu komponentu da bi se lakše identifikovala i koristila
    - **C# razlikuje mala i velika slova!**
  - **Tip podatka** koji se čuva u promenljivoj (ceo broj, razlomljen broj, karakter, tekst, niz, logička promenljiva, niz od 40 brojeva) a koji određuje i koliko mesta u memoriji zauzima promenljiva
  - **Vrednost promenljive** (dobra praksa je uvek dodeljivati vrednost promenljive)

    ```
    int a = 2;
    int b;   kolika je početna vrednost b?
    ```
  - Ograničenje na vrednost promenljive (5 < ocena < 10)

Doseg promenljive ...

- Mesto deklarisanja promenljive nije ograničeno (može bilo gde u kodu)
- Preporučuje se da se promenljive deklarišu na jednom mestu (najbolje na početak programskog bloka), jer je lakše za održavanje programskog koda
- Ako nije drugačije označeno, promenljiva važi (vidljiva je) od mesta deklarisanja do kraja programskog bloka
- Programski blok (programska celina) određena **{ }**

... Doseg promenljive



Tipovi podataka ...

- Definiše dostupne tipove podataka kojima se može manipulisati i koji se mogu skladištiti u nekom programu
- Tip podatka određuje:
  - **količinu memorije koju će zauzeti promenljiva**
  - **opseg mogućih vrednosti**
  - **dozvoljene operacije**
- C# podržava više primitivnih tipova podataka koji opisuju :
  - cele brojeve (byte, short, int, long) – 10, 15, 1024, -45
  - realne brojeve (float, double) – 15.5, -75.02, -10e12
  - karaktere (char) – 'a', 'A', '1', '\n', '\t', '\", '\\'
  - logički tip - bool (boolean) – true, false

... Tipovi podataka

Tip	Veličina	Početna vrednost	Opseg
bool	n/a	false	true ili false
byte	8 bita	0	od -128 do 127
char	16 bita	(unsigned)	od \u0000' do \uffff' ili 0 to 65535
short	16 bita	0	od -32768 do 32767
int	32 bita	0	od -2147483648 do 2147483647
long	64 bita	0	od -9223372036854775808 do 9223372036854775807
float	32 bita	0.0	od 1.17549435e-38 do 3.4028235e+38
double	64 bita	0.0	4.9e-324 to 1.7976931348623157e+308

Neposredni operandi ...

- Neposredni operandi su konstantne vrednosti koje se unose direktno u kod
- Neposredni operandi za char tip se uvek navode kroz navodnike i predstavljaju (može biti) samo jedan karakter:

  'a', 'A', '1'
- Pored običnih znakova, mogu se koristiti i specijalni karakteri: '\n' (novi red), '\t' (tab), '\'' ('), '\\' (\)
  - Iako su navedena dva karaktera, tumače se kao jedan
- Celi brojevi se navode kao: 12, 444, -579 i zavisnosti od svoje veličine biće tumačeni kao byte, short ili int, ako se želi da ceo broj bude long, onda se iza broja navodi sufiks „L“: 12L, -579L

... Neposredni operandi

- Pored decimalnih brojeva, mogu se koristiti i binarni (0b1001, 0b1111), oktalni (0124, 055, 07) i heksadecimalni brojevi (0xA19B, 0x458DF)
- Kada se zapisuje jako dugačak broj moguće je cifre razdvojiti donjom crtom (nema efekta na kompajler, samo je lakše pročitati) – 6_000_000 je isto što i 6000000
- Kod razlomljenih brojeva, koristi se decimalna tačka, a ne zarez: 12.34, -12.367
- Svi decimalni brojevi se zapisuju kao double, ako se želi da broj bude float, onda se iza broja navodi sufiks „F“: 12.34F, -12.367F
- Moguće je decimalne brojeve zapisati i u eksponencijalnom obliku: 1.3e12 (isto što i $1.3 \cdot 10^{12}$)

String - osnovno

- Pored osnovnih tipova, C# sadrži i druge kompleksnije tipove, ali su oni u suštini objekti (više o njima kasnije)
- Ipak, jedan od ovih tipova je značajan, jer se puno koristi, a to je String
- String predstavlja niz karaktera
- Neposredni operand tipa String navodi sa kao niz karaktera između dva znaka navoda:
  - "Hello World!", "\tDragan Torbica", "Programiranje je \"cool\". "
  - Prazan string (empty string) - ""
  - Praviti razliku između " " (dva dvostruka navodnika za string) i ' ' (dva jednostruka navodnika za karakter)

Deklaracija promenljive

- Promenljiva se deklariše ali joj se ne dodeljuje inicijalna vrednost

tip identifikator;

```
int godina;
float pi;
short sat;
char znak;
```

- Ako vrednost nije eksplicitno navedena, **C# kompajler dodaje predefinisanu vrednost (**početna vrednost)

Definicija promenljive

- Promenljiva se deklariše, ali joj se dodeljuje inicijalna vrednost

tip identifikator = vrednost;

```
int godina = 2011;
float pi = 3.14;
short sat = 23;
char znak = 'a';
```

- Dobra praksa je uvek inicijalizovati promenljivu na nama poznatu i željenu vrednost

Inicijalizacija promenljive

- Dodela vrednosti naknadno već deklarisanoj ili inicijalizovanoj promenljivoj

identifikator = vrednost;

```
godina = 2014;
sat = sat – 1;
```

- **Da bi se promenljiva koristila mora prethodno biti deklarisana !**
- Da ponovimo, konkretna vrednost zove se **neposredni operand** jer joj se vrednost ne može menjati

Primeri rada sa promenljivama

```
/*****************************************************************************************************
*  Primer jednostavnog programa koji racuna zaradu na godisnjem nivou za investiciju od 1000 dinara po godisnjoj
*  kamatnoj stopi od 0.027. Ukupna zarada i vrednost investije posle godinu dana se ispisuju u konzolu.
*****************************************************************************************************/
namespace PrvaAplikacija {
    class Program  {
        static void Main(String[] args) {
        /* Deklaracija promenljivih. */
        double investicija; // Vrednost investicije.
        double stopa;       // Kamatana stopa.
        double zarada;      // Ukupna kamata po godini.
        /* Izracunavanje. */
        investicija = 1000;
        stopa = 0.027;
        zarada = investicija * stopa;   // Sracunavanje zarade.
        investicija = investicija + zarada;
        // Izracunavanje investicije posle godinu dana sa kamatom.
        // (Obratiti paznju da nova vrednost investicije prepisuje staru.)
        /* Ispisivanje rezultata. */
        Console.Write("Za godinu dana zaradjeno je: ");
        Console.Write(zarada);
        Console.Write(" dinara.");
        Console.Write("\nUkupna vrednost investije je sada: ");
        Console.Write(investicija);
        Console.Write(" dinara.");
    } // kraj main metode
} } // kraj klase, kraj namespacea
```


Zadaci za vežbanje

1. Program koji obračunava transakciju dinara u evro po kursu 118.5, pretpostaviti da se menja 155000 dinara.

   / je operator deljenja

2. Program koji računa prosečnu potrošnju goriva na pređenih 100 kilometara ako je pređeno 350 km a potrošeno 17 l goriva.

   ***** je operator množenja

3. Program koji računa površinu i obim kruga poluprečnika

   $r = 20$

   $P = r^2\pi \qquad O = 2r\pi$

Konstante

- Identifikator čija vrednost se ne menja u programu
- Čuva se u memoriji, ali joj se vrednost ne može menjati
- Mora se eksplicitno deklarisati tip i inicijalizovati vrednost
- Da bi se promenljiva proglasila za konstantu ispred njenog tipa mora da se navede rezervisana reč const

```
const tip identifikator = vrednost;
```

```
const int PUNOLETAN = 18;
const short RADNIDANSATI = 8;
const double PI = 3.14159265358979323846433;
```

- U C# programskom jeziku je uobičajeno da identifikatori konstanti budu napisani svim velikim slovima (prema konvenciji)

Primeri rada sa konstantama

- Šta su kandidati za konstante u prethodnim primerima?
- **kurs**
- **kmprosek**
- **Pi**

- Da li su sve one kandidati i u realnim problemima?

Prekvalifikacije za IT

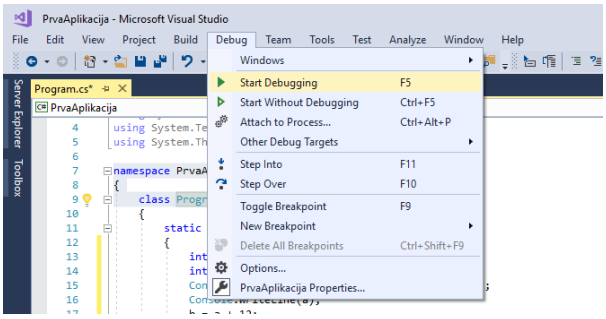
OSNOVNO O DEBUGGING-U

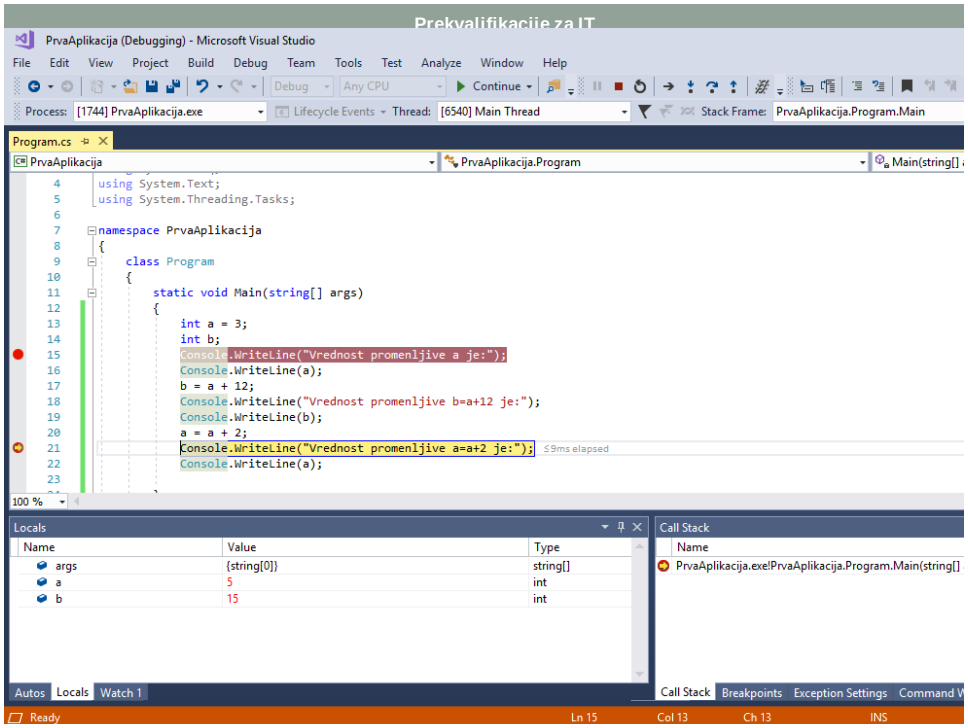
Debugging ...

- Proces pronalaženja i ispravljanja grešaka u kodu
- Fundamentalna veština u programiranju
- **Najbolje je ne uvoditi greške u kod uopšte!**
  - Pronaći testirano parče koda koji radi ono što želite od pre i uključiti ga u ono što trenutno radite (**reuse**)
  - Razmisliti o problemu, osmisliti rešenje, pa tek onda kodirati (**design**)
  - Koristiti preporučene metode/tehnike kako bi se izbegle uobičajene greške (**best practices**)
- **Pronaći i ispraviti greške u ranim fazama razvoja**
  - Testirati dizajn
  - Testirati svako novo parče koda
  - Testirati kod kao celinu

... Debugging

- Vratićemo se na testiranje i kasnije
- Kako debagirati kod u Visual Studio okruženju?
  - Koristiti tačke prekida (**Breakpoints**)
  - Pokrenuti program u debug režimu (**Start Debugging, F5**)
  - Pratiti stanja promenljivih preko **Locals/Watch** panela
  - Koristiti **Step Into** (**F11**) i **Step Over** (**F10**) za izvršavanje linije po linije koda





Immediate Window