OOP

P02 – Uvod u OOP

U ovom poglavlju naučit ćete...

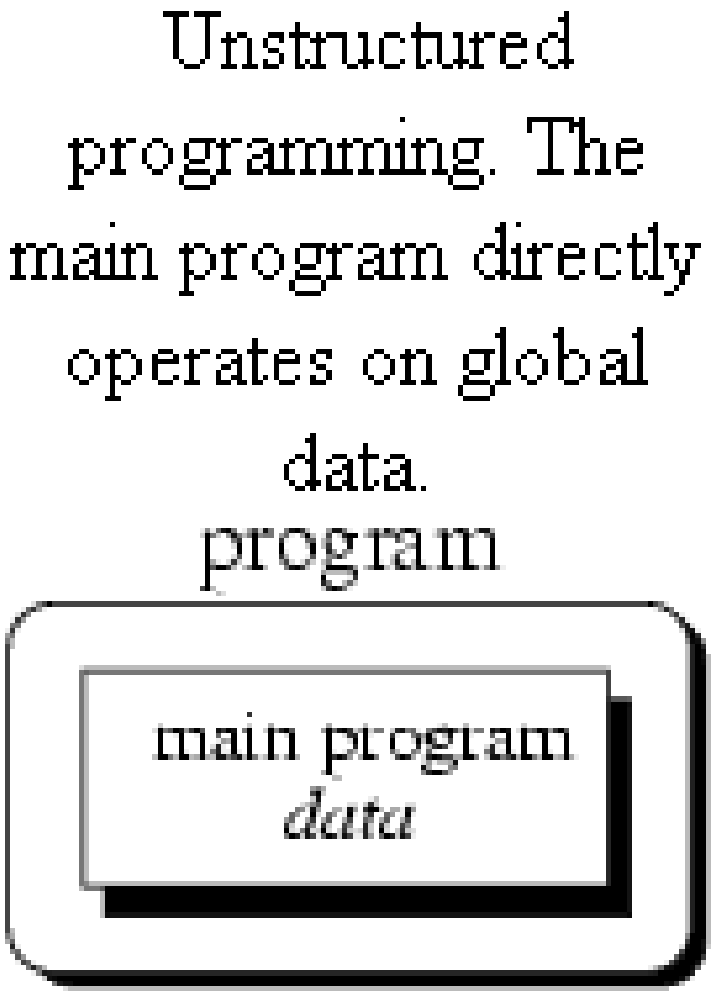
- Tehnike programiranja
- Svojstva objektno orijentiranog programiranja
- Što je programski jezik C#
- Primjer programa - “Hello world”

Pregled tehnika programiranja

- Nestrukturirano programiranje
- Proceduralno programiranje
- Modularno programiranje
- Objektno orijentirano programiranje
- Generičko programiranje

Nestrukturirano programiranje

- Samo jedan program, npr. Main
- Svi nizovi naredbi su u jednom programu - glavnom programu
- Primjer: Fortran, asembler, stari Basic



Strukturirano programiranje

- Naziva se i proceduralno programiranje
- Za svaku zadatak stvara se procedura (funkcija)
- Procedure se pozivaju iz glavnog programa

Figure 2.2: Execution of procedures. After processing flow of controls proceed where the call was made.

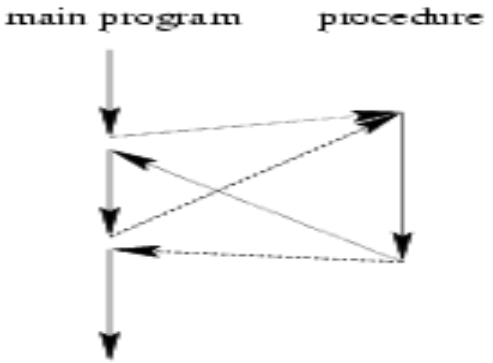
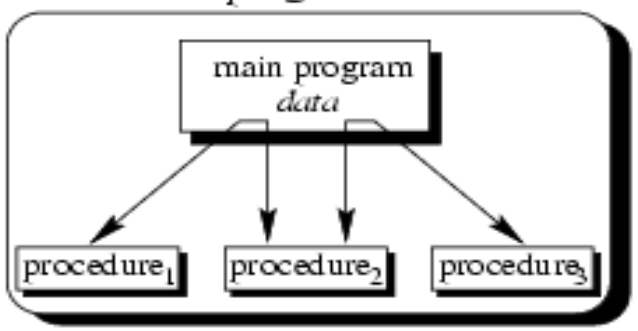


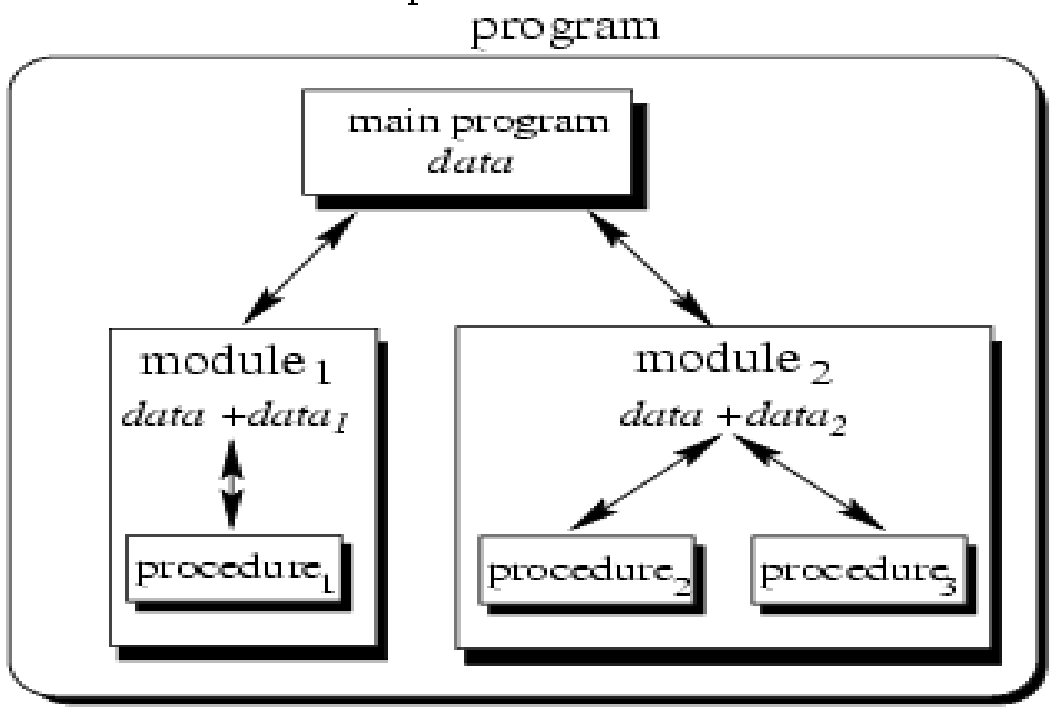
Figure 2.3: Procedural programming. The main program coordinates calls to procedures and hands over appropriate data as parameters.



Modularno programiranje

- Procedure s nekim zajedničkim funkcionalnostima grupiraju se zajedno u zasebne module
- Program se sastoji od nekoliko manjih modula
- Svaki modul može imati svoje vlastite podatke

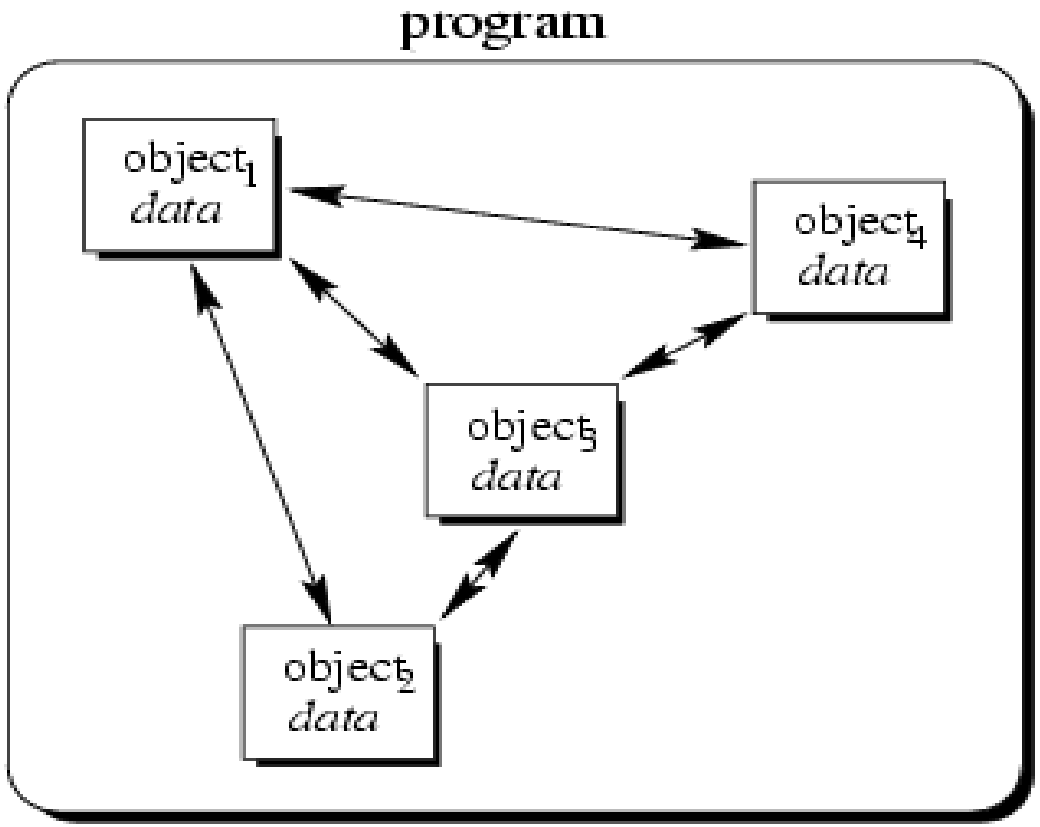
Figure 2.4: Modular programming. The main program coordinates calls to procedures in separate modules and hands over appropriate data as parameters.



Objektno orijentirano programiranje

- Radi sa (na) objektima koji se smatraju osnovnom građevnom jedinicom objektno orijentiranih jezika
- Više se usredotočuju na podatke umjesto na procedure

Figure 2.6: Object-oriented programming. Objects of the program interact by sending messages to each other.



Primjer 1:

```
#include …

void main()
{
   int a,b,c;
   clrscr();
   cout << "Enter the first number";
   cin >> a;
   cout << "Enter the second number";
   cin >> b;
   c=a+b;
   cout << "The sum is:" << c;
   getch();
}
```

Primjer 2:

```
#include …
int add(int,int);

void main()
{

   int a,b,c;
   clrscr();
   cout << "Enter the first number";
   cin >> a;
   cout << "Enter the second number";
   cin >> b;
   c=add(a,b);
   cout<<"The sum is:" << c;
   getch();
}

int add(int x,int y)
{
   int z=x+y;
   return z;
}
```

Primjer 3:

```
#include …

class Addition
{
   int a,b,c;
   public:
   void read()
   {
      cin >> a;
      cin >> b;
   }
   void add()
   {
      c=a+b;
   }
   void display()
   {
      cout << "The sum is:"
<< c;
   }
};
```

```
void main()
{
   Addition obj;   //object creation
   cout << "Enter the numbers";
   obj.read();
   obj.add();
   obj.display();
   getch();
}
```

Nedostaci proceduralnog programiranja

- Loše modeliranje „stvarnog svijeta”
- Nema privatnosti
- Nema istinske ponovne uporabe
- Teže održavanje i proširivanje većih projekata

Karakteristike OOP paradigme

Prednosti:

- Bolja organizacija i modularnost koda.
- Lakše održavanje i nadogradnja (nasljeđivanje, polimorfizam).
- Povećana ponovna iskoristivost (klase i objekti).
- Enkapsulacija poboljšava sigurnost i stabilnost podataka.

Nedostaci:

- Složenije za naučiti i implementirati.
- Više koda i resursa (memorija, CPU).
- Ponekad nepotrebno komplicirano za male projekte.

PP vs OOP

Feature	Procedure Programming (PP)	Object oriented Programming (OOP)
Divided Into	In PP Program is divided into small parts called **functions**	In OOP, program is divided into parts called **objects**.
Importance	In PP, Importance is not given to **data** but to functions as well as **sequence** of actions to be done.	In OOP, Importance is given to the data rather than procedures or functions because it works as a **real world**.
Approach	PP follows **Top Down approach**.	OOP follows **Bottom Up approach**
Access Specifiers	PP does not have any access specifier	OOP has access specifiers named Public, Private, Protected, etc.
Data Moving	In PP, Data can move freely from function to function in the system.	In OOP, objects can move and communicate with each other through member functions.
Expansion	To add new data and function in PP is not so easy.	OOP provides an easy way to add new data and function
Data Access	In PP, Most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOP, data can not move easily from function to function, it can be kept public or private so we can control the access of data.
Data Hiding	PP does not have any proper way for hiding data so it is **less secure**.	OOP provides Data Hiding so provides **more security**.
Overloading	In PP, Overloading is not possible.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.
Examples	Example of PP are : C, VB, FORTRAN, Pascal.	Example of OOP are : C++, JAVA, VB.NET, C#.NET.

„OOP 4 pillars”

Apstrakcija

Enkapsulacija

Nasljeđivanje

Polimorfizam

Osnovi principi OOP-a (1)

- Apstrakcija – svojstvo OOP-a
  - odvajanje bitnih od nebitnih detalja u zadanom kontekstu
  - kreiranje sustava „otpornog” na promjene
  - primjer: Automobil (mehaničarska radnja) i Automobil (dizajnerski studio)

Osnovi principi OOP-a (2)

- **Enkapsulacija** = korisnici nekog objekta ne mogu mijenjati unutrašnja stanja i metode objekata.
- **Nasljeđivanje** je također važna osobina objektno orijentiranog programiranja. Ono je posljedica generalizacije. (Student – Osoba)
- **Polimorfizam** = pojava da ista metoda djeluje na različite načine, ovisno o objektu na koji se primjenjuje

Osnovi principi OOP-a (3)

- Objekti oko nas posjeduju razne statičke i dinamičke osobine
  - objekti iste vrste dijele iste osobine
  - svaki objekt pripada nekoj vrsti
  - objekt = svojstva + metode + događaji

Osnovi principi OOP-a (4)

- vrsta objekta = klasa
- klasa = apstrakcija određenih karakteristika konkretnih objekata iste vrste
  - određuje njegovo sučelje, njegova svojstva, metode i događaje preko kojih možemo komunicirati s njim u programu
- objekt = instanca neke klase.

Osnovi principi OOP-a (5)

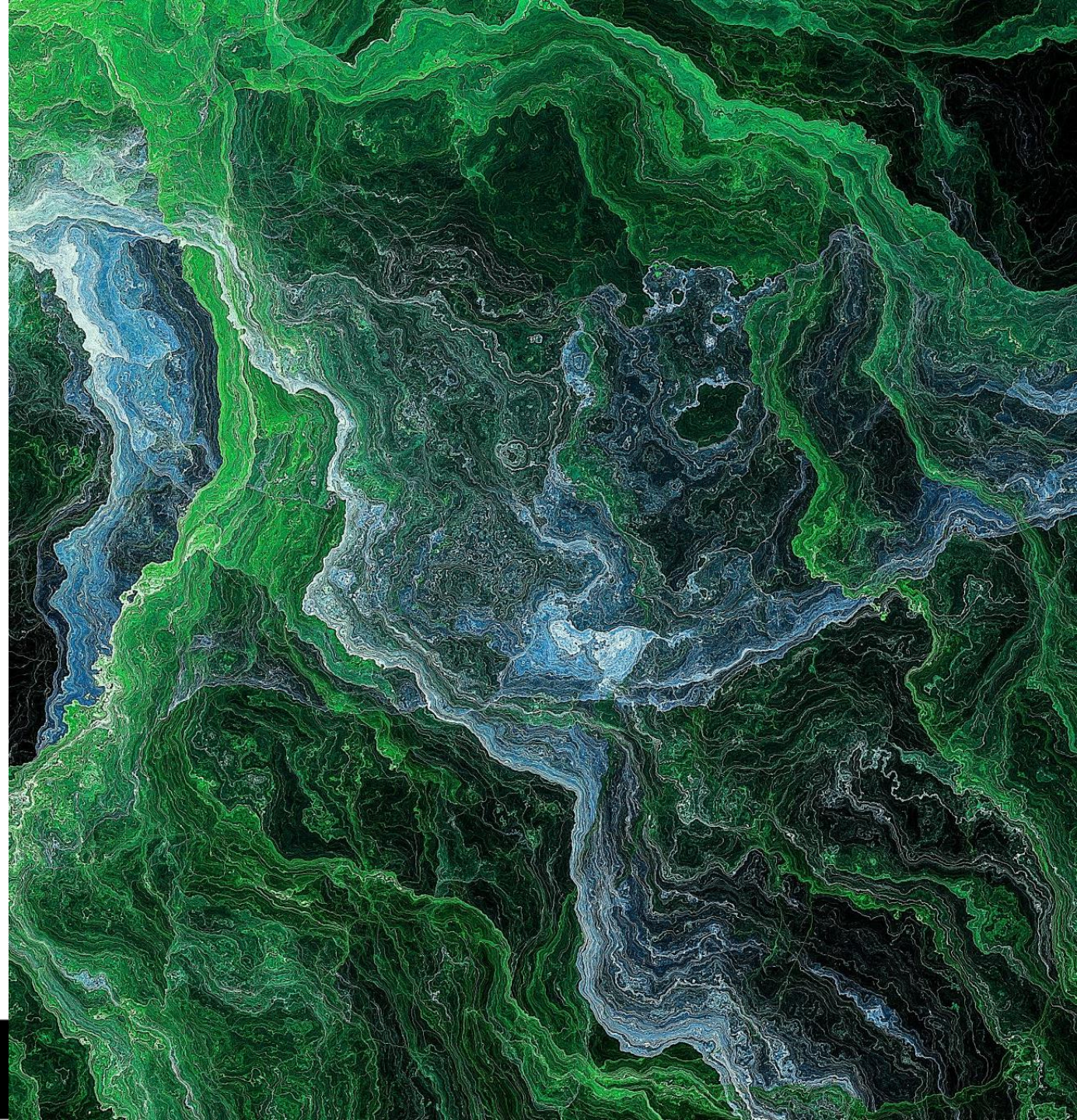
```
public class Auto
{
    private string Marka;
    private int Godiste;

    public void Pokreni()
    {
        Console.WriteLine("Auto se pokreće!");
    }
}
...
...
...
Auto mojAuto = new Auto();
mojAuto.Marka = "Toyota";
mojAuto.Godiste = 2020;
mojAuto.Pokreni();
```

https://en.wikipedia.org/wiki/List_of_object-oriented_programming_languages

OOP jezici

Programski jezik C# (1)

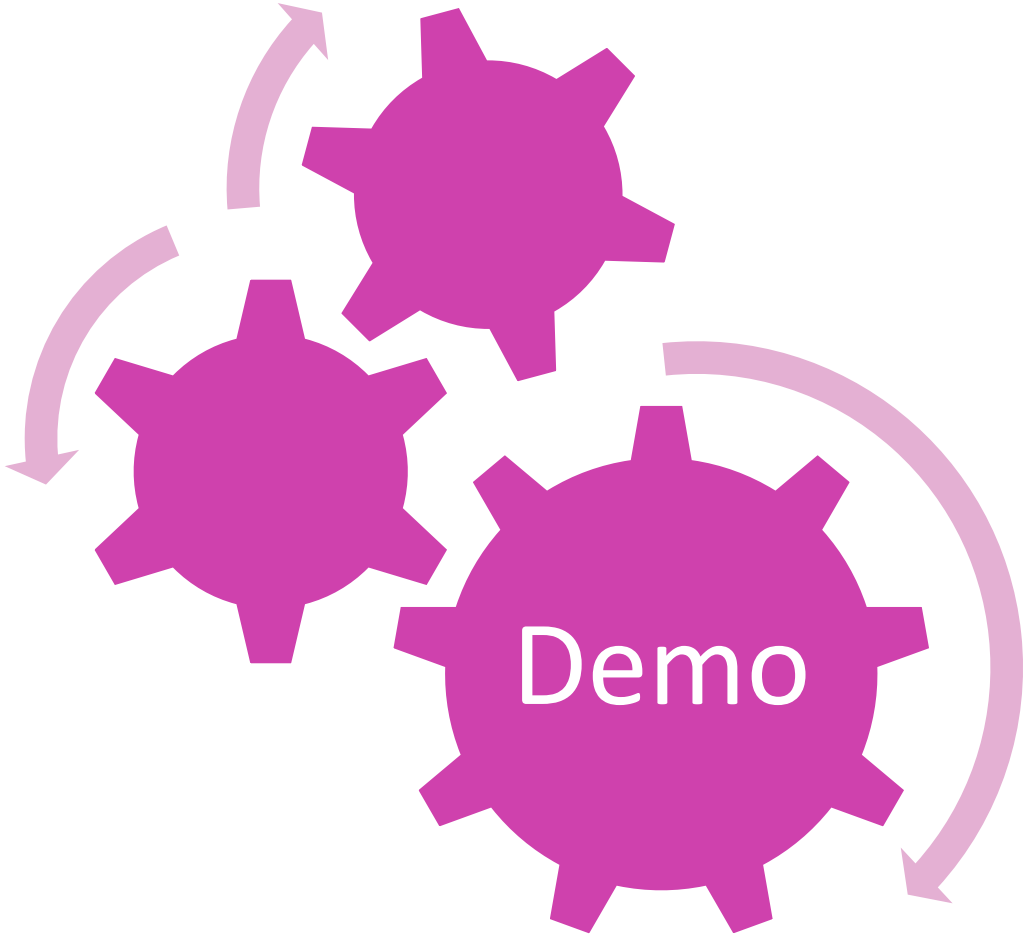
- Jednostavan; sadrži oko 80 ključnih riječi i desetak ugrađenih tipova
- Programi u programskom jeziku C# organizirani su kao projekti.
- Svaki projekt sadrži neke obavezne datoteke, ali možemo dodati proizvoljan broj datoteka i mapa koje možemo organizirati po želji u cilju lakšeg snalaženja u svojoj poslovnoj logici



Programski jezik C# (2)

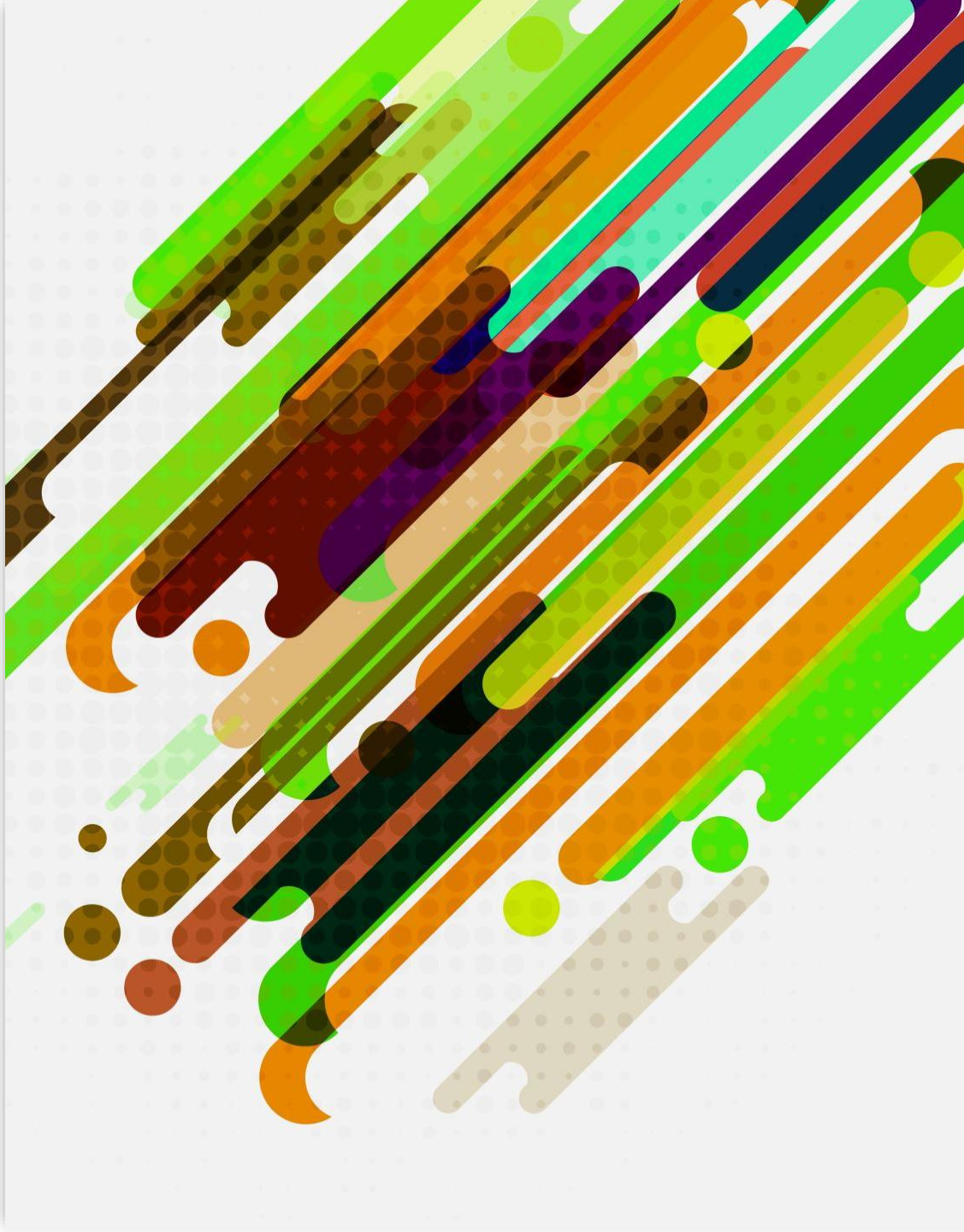
- Prevođenjem C# programskog kôda iz svih datoteka u projektu nastaje jedan **sklop** (engl. assembly), .dll ili .exe
- Na .NET platformi to je osnovna jedinica za ponovnu upotrebu
- Programi napisani u C# izvršavaju se u zajedničkom okruženju za sve .NET jezike – (CLR), što omogućuje korištenje klasa implementiranih u bilo kojem .NET programskom jeziku.

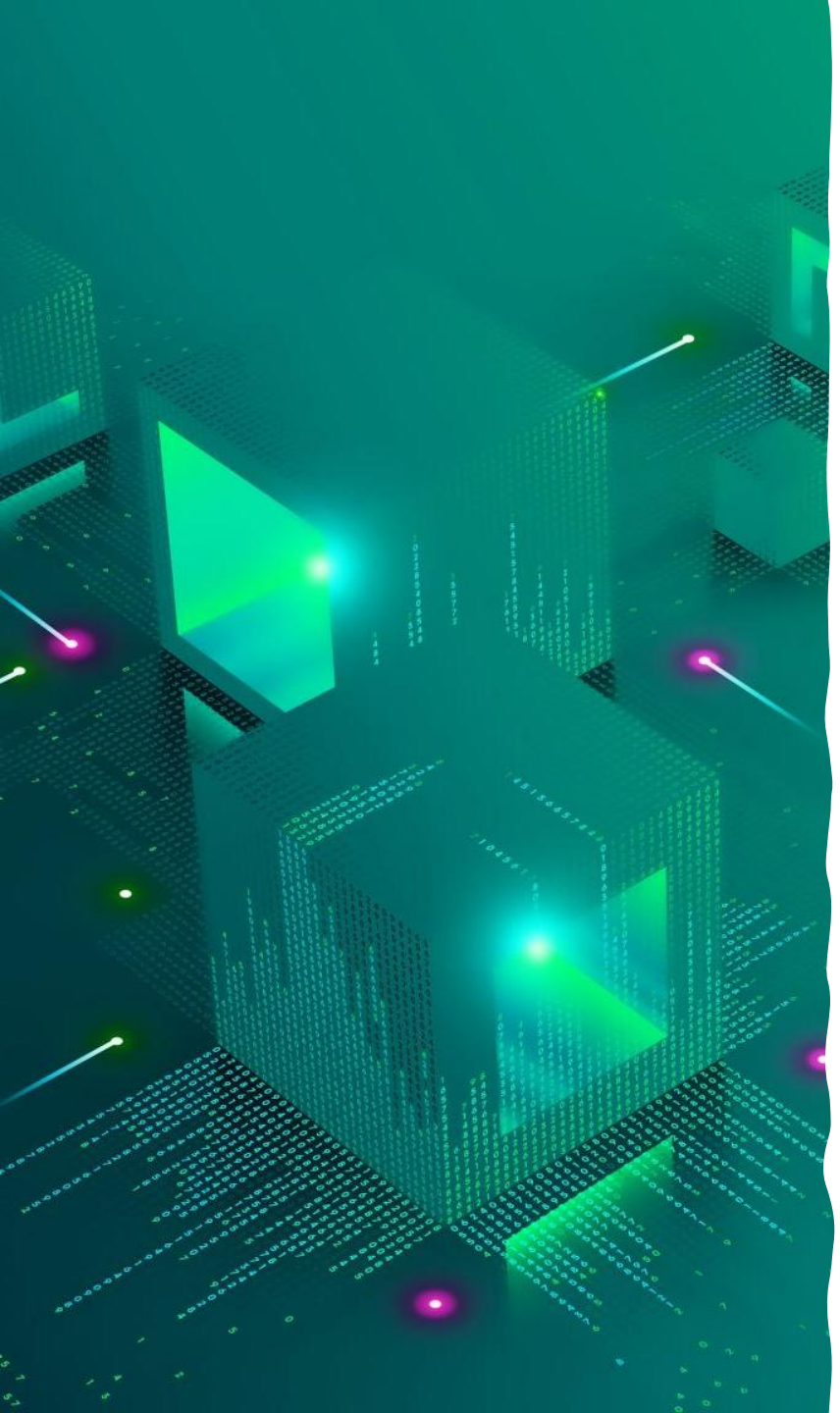
Demo – Hello world u C# programskom jeziku



Hello world…

- Programski jezik C# razlikuje velika i mala slova
- Ostavljanje praznina je uglavnom stvar osobnog odabira programera - budite konzistentni
- Klasa = podatkovni tip
- Metode - Main() – entry point
- Ključna riječ static
- Komentari





OSNOVE PROGRAMSKOG JEZIKA C#

Osnovni građevni elementi C#-a

- Ugrađeni i korisnički tipovi
- Varijable i konstante
- Identifikatori, izrazi, iskazi
- Kontrola tijeka
- Operatori

Tipovi

- C# je strogo tipiziran jezik.
- Svakom objektu kojeg stvorimo moramo deklarirati njegov **tip** (engl. *type*).
- Tip objekta govori prevoditelju koliko je objekt velik (tj. koliko troši memorije) i koje su njegove mogućnosti.

Tipovi...

- Tipove u C#-u možemo podijeliti na **ugrađene** (engl. *intrinsic*) i **korisnički definirane** tipove. Ugrađeni tipovi dio su samog jezika, dok korisničke definira programer.
- Prema načinu spremanja njihovih vrijednosti u memoriju tipovi u C#-u se dijele na **vrijednosne** i **referentne**. Vrijednosni tip čuva svoju vrijednost u memoriji dodijeljenoj na *stogu*, a referentni na *hrpi*.

Stog

- **Stog** (engl. *stack*) je podatkovna struktura koja se koristi za spremanje elemenata po načelu "posljednji unutra prvi van" (kao stupac tanjura naslaganih jedan na drugi).
- Stog se odnosi na područje memorije unaprijed alocirano za pokretanje programa i u njega se između ostalog spremaju lokalne varijable.

Hrpa

- **Hrpa** (engl. *heap*) je područje memorije koje se koristi za alociranje prostora referentnim tipovima, npr. objektima. Kad se objekt alocira na hrpu, za taj proces potrebno je pamtiti gdje se on u memoriji nalazi.
- To automatski odrađuje CLR, a mi objektu pristupamo na isti način kao da se radi o običnoj varijabli. Ipak, za varijablu koja pristupa objektu često se koristi naziv **referenca**.

Ugrađeni tipovi

- Programski jezik C# ima ugrađeni skup tipova od kojih se svaki od njih preslikava u temeljni tip kojeg podržava .NET zajednička jezična platforma.
- Svaki tip ima određenu i nepromjenjivu veličinu. Na primjer, cjelobrojni tip int uvijek ima veličinu 4 bajta jer se on preslikava u .NET-ov temeljni tip Int32 (4 bajta po 8 bitova je 32).

Ugrađeni tipovi...

Tip	Veličina (u bajtovima)	.NET tip	Opis
`bool`	1	`Boolean`	*true* ili *false*
`byte`	1	`Byte`	od 0 do 255 (od 0 do 2^8-1)
`char`	2	`Char`	pojedinačni *Unicode* znak
`sbyte`	1	`SByte`	od -128 do 127 (od -2^7 do 2^7-1)
`short`	2	`Int16`	od -32 768 do 32 767 (od -2^{15} do 2^{15}-1)
`ushort`	2	`UInt16`	od 0 do 65 535 (od 0 do 2^{16}-1)
`int`	4	`Int32`	od -2 147 483 648 do 2 147 483 647 (od -2^{31} do 2^{31}-1)
`uint`	4	`UInt32`	od 0 do 4 294 967 295 (od 0 do 2^{32}-1)
`float`	4	`Single`	7 decimala, od $\pm1{,}5 \times 10^{-45}$ do $\pm3{,}4 \times 10^{38}$
`double`	8	`Double`	15-16 decimala, od $\pm5{,}0 \times 10^{-324}$ do $\pm3{,}4 \times 10^{308}$
`decimal`	16	`Decimal`	28-29 decimala, od $\pm1{,}0 \times 10^{-28}$ do $\pm3{,}4 \times 10^{28}$
`long`	8	`Int64`	od -9 223 372 036 854 775 808 do 9 223 372 036 854 775 807 (od -2^{63} do 2^{63}-1)
`ulong`	8	`UInt64`	od 0 do 18 446 744 073 709 551 615 (od 0 do 2^{64}-1)

Tip char

- Tip **char** predstavlja *unicode* znak.
- Konstante tipa char pišu se unutar jednostrukih navodnika ('A', 'a', '1'...)
- Char može biti jednostavan znak (npr. 'A'), *Unicode* znak (npr. '\u0041') ili *kontrolni* znak (npr. '\t')
- **Kontrolni znakovi** (engl. *escape characters*) su posebni *literali* sastavljeni od 2 znaka

Tip char...

Znak	Značenje
`\'`	Jednostruki navodnik (apostrof)
`\"`	Dvostruki navodnik
`\\`	Obrnuta kosa crta (engl. *backslash*)
`\0`	Null
`\a`	Upozorenje
`\b`	Brisanje unatrag
`\f`	Nova stranica
`\n`	Novi red
`\r`	Prijelaz u sljedeći red
`\t`	Vodoravni tabulator
`\v`	Okomiti tabulator

Pretvaranje ugrađenih tipova

- Objekt jednog tipa može se *eksplicitno* ili *implicitno* pretvoriti u objekt drugog tipa.
- **Implicitne** pretvorbe automatski vrši sam prevoditelj i u njima se informacije ne mogu izgubiti.

```
int x = 5;
double y = x;  // Implicitna pretvorba
```

Pretvaranje ugrađenih tipova

- U obrnutoj pretvorbi može doći do gubitka informacija i stoga prevoditelj neće prevesti sljedeći programski kôd:

```
double y = 5;
int x = y;  // Neće se prevesti!
```

Pretvaranje ugrađenih tipova

- Ukoliko baš striktno želimo provesti svoj naum i u varijablu tipa int spremiti vrijednost za koju smo svjesni da je cjelobrojna, onda moramo napraviti **eksplicitnu** pretvorbu:

```
double y = 5;
int x = (int)y;  // Eksplicitna pretvorba
```

Varijable

- **Varijabla** je lokacija u memoriji s točno određenim tipom koja je u programu dostupna preko svog imena.
- Imena varijabli i općenito imena svih ostalih elemenata koje definiramo u programu (tipovi, metode, svojstva, konstante, objekti...) moraju počinjati slovom ili donjom crtom.

Obavezna deklaracija

- Neki programski jezici nakon deklariranja automatski inicijaliziraju varijable na unaprijed određene inicijalne vrijednosti, ali ne i programski jezik C#.
- U C#-u prije korištenja lokalne varijable i deklaracija i inicijalizacija moraju biti eksplicitno navedeni. To znači da svaku lokalnu varijablu prije nego koristimo moramo inicijalizirati na neku početnu vrijednost.

Enumeracije

- **Enumeracija** je vrijednosni tip koji se sastoji od skupa imenovanih konstanti.

```
enum imeEnumeracije [:osnovniTip]        {
                const1 = vrijednost1,
                const2 = vrijednost2,
                …
}

public enum StatusZapisa
{
   Neaktivan = 0,
   Aktivan = 1,
   Arhiviran = 2,
   Obrisan = 3
}
```

Stringovi

- Svaki tekstualni podatak (riječ, rečenica, natpis...) zapravo je niz vrijednosti tipa char. Za takve nizove znakova u programskom jeziku C# definiran je poseban tip **string**.

```
string imeNiza;
```

Iskazi

- Potpuna programska uputa u C#-u naziva se **iskazom** (engl. *statement*). Programi se sastoje od nizova iskaza. Svaki iskaz mora završavati znakom **točka-zarez** (*;*)

```
string ime;
ime = "SAB";
int x = 5;
```

Iskazi uvjetnog grananja

- Uvjetno grananje stvara se *uvjetnim iskazom* na koji upućuju ključne riječi **if**, **else** i **switch** i do njega se može doći samo kad je uvjetni iskaz istinit.

Naredba if

- Iskazi **if...else** označavaju grananje temeljeno na **uvjetu**. Uvjet je izraz koji se testira na početku iskaza if.

```
if (izraz)
{
        iskaz;
}
```

if else

```
if (izraz)
{
        iskaz1;
}
[else
{
        Iskaz2;
}]
```

switch

- Ugniježđeni if iskazi teško se čitaju, teško ih je ispravno napisati i samim tim je kasnije teško u njima uočiti i ispraviti pogreške
- Srećom, kad nam je zadan širok raspon mogućnosti, u kojima se radi usporedba (==), iskaz **switch** predstavlja čitljiviju alternativu

```
switch (izraz)
{
    case vrijednost_izraza_1:
        // Blok_iskaza_1
        break;
...
    case vrijednost_izraza_N:
        // Blok_iskaza_N
        break;
    default:
```

switch

Iteracije

- Pod iteracijskim iskazima podrazumijevamo iskaze koji omogućuju višestruko ponavljanje izvođenja (iteriranje) istog bloka iskaza. Iteracijski iskazi nazivaju se još i petlje.
- U radu s petljama koriste se prije bezuvjetni programski skokovi: break, continue i return.

while

- Semantika **while** petlje je: "Dok je izraz istinit, obavljaj taj posao", dok je sintaksa sljedeća:

while (izraz) iskaz

do while

- Iskaz while se ne mora izvesti niti jednom ako je njegov izraz na početku bio nesitinit. Ako želimo osigurati da se iskaz izvede barem jednom koristit ćemo petlju **do..while**.
- Njezina sintaksa glasi:

```
do {
        iskaz;
} while (logički izraz)
```

for

- inicijalizacija varijable
- testiranje vrijednosti varijable
- izvođenje niza iskaza
- povećanje varijable (i++) - iteriranje
- Petlja **for** omogućuje kombiniranje tih koraka u jedan iskaz petlje:

```
for ([inicijalizatori]; [izraz]; [iteratori]) {
        iskaz
}
```

Operatori

- Operator je simbol koji uzrokuje da C# pokrene određenu akciju. Primitivni ugrađeni tipovi (npr. int) podržavaju razne operatore i u ovom poglavlju ćemo se upoznati s najvažnijim među njima.

Operator pridruživanja

- S **operatorom pridruživanja** (=) već smo se zapravo upoznali u mnogim prethodnim primjerima u ovom priručniku. Ovaj operator uzrokuje promjenu vrijednosti operanda sa svoje lijeve strane u vrijednost koja se nalazi s njegove desne strane.

Matematički operatori

- U programskom jeziku C# koristi se pet matematičkih operatora: četiri za standardne aritmetičke operacije i peti za vraćanje ostatka pri dijeljenju cjelobrojnih vrijednosti.
    - Zbrajanje (+)
    - Oduzimanje (-)
    - Množenje (*)
    - Cjelobrojno dijeljenje (/)
    - Dijeljenje s ostatkom (%)

Operatori za uvećavanje i umanjivanje

- **Operator oduzimanja i pridruživanja** (-=)
- **Operator množenja i pridruživanja** (*=)
- **Operator dijeljenja i pridruživanja** (/=)
- **Operator modulo i pridruživanja** (%=)
- **Operator uvećavanja za** 1 (++)
- **Operator umanjivanja za** 1 (--)

Relacijski operatori

- **Relacijski operatori** koriste se za usporedbu dvije vrijednosti, a kao rezultat vraćaju true ili false (*istinito* ili *neistinito*).
    - Jednako ==
    - Nije jednako !=
    - Veće od >
    - Veće ili jednako >=
    - Manje od <
    - Manje ili jednako <=

Logički operatori

- U primjerima sa if iskazima testirali smo samo pojedinačni uvjet. Međutim, česte su situacije u kojima ćemo morati testirati jesu li dva (ili više) uvjeta istiniti, što ako je jedan istinit, a drugi ne, i razne druge kombinacije.
    - I &&
    - ILI ||
    - NE !

Slijednost operatora

Kategorija	Operatori
Primarni	`(x) x.y x->x f(x) a[x] x++ x-- new typeof sizeof checked unchecked`
Unarni	`+ - ! ~ ++x --x (T)x x*x x&x`
Množenje	`* / %`
Zbrajanje	`+ -`
Pomak	`<< >>`
Relacijski	`< > <= >= is as`
Jednakost	`== !=`
Logičko I	`&`
Logičko ekskluzivno ILI	`^`
Logičko ILI	`|`
Uvjetno I	`&&`
Uvjetno ILI	`||`
Uvjetni (ternarni)	`?:`
Pridruživanje	`= *= /= += -= <<= &= ^= |=`

Ternarni operator

- Jedini operator u programskom jeziku C# koji zahtijeva tri izraza je **ternarni operator (?:)**. Njegova sintaksa glasi:

uvjetni izraz ? izraz1 : izraz2

Hvala na pažnji!