

Teorija

- ◆ PRVI PROGRAM
 - ◆ Komentarji
 - ◆ Izpisovanje na zaslon
 - ◆ Konstante, znaki, števila, nizi
 - ◆ Matematični operatorji
 - ◆ Nizi
 - ◆ Konstante
 - ◆ Spremenljivke
 - ◆ Cela števila: tip int
 - ◆ Prireditveni stavek
 - ◆ Realna števila: tip double
 - ◆ Pretvorba realnega v celo število
 - ◆ Standardne funkcije
 - ◆ Zamenjava dveh spremenljivk
- ◆ BRANJE PODATKOV
 - ◆ Metoda ReadLine
 - ◆ Metoda Read
 - ◆ Pretvarjanje celega števila v niz
 - ◆ Naključna števila
- ◆ POGOJ
 - ◆ IF Stavek
 - ◆ Popoln pogojni stavek
 - ◆ Nepopoln pogojni stavek
 - ◆ Relacijski operatorji
 - ◆ Logične vrednosti
 - ◆ Logične operacije
 - ◆ Gnezdeni Pogojni Stavek
 - ◆ Switch
- ◆ RANDOM CLASS
- ◆ ZANKE
 - ◆ While
 - ◆ Do while
 - ◆ For zanka
 - ◆ Break and Continue

- ◆ Foreach
- ◆ NIZI
 - ◆ Znaki - tip char
 - ◆ Metode in lastnosti za delo z nizi
 - ◆ PadLeft
- ◆ TABELE
 - ◆ Enodimenzionalne Tabele
 - ◆ Razširitev tabele
 - ◆ Vstavljanje na zadnjem mestu
 - ◆ Večdimenzionalne tabele
- ◆ METODE
 - ◆ Zakaj v metodah ne izpisujemo/beremo
 - ◆ Vrste metod
 - ◆ Razlaga posameznih stakov in besed
 - ◆ Klic metode
- ◆ OBJEKTI
- ◆ RAZREDI
 - Dostopnost podatkov
 - Ustvarjanje objektov
 - Naslov objekta
 - Kako se lotiti načrtovanja rešitve z OOP?
 - Zgradba
 - ◆ Konstruktor
 - this
 - ◆ Preobtežene metode
 - ◆ Objektne Metode
 - ◆ Objektne vs Statične
 - ◆ Tabele objektov
 - ◆ Dostop do stanj objektov
- ◆ Serializacija
 - ◆ File
 - ◆ Datoteke - osnove
 - ◆ Directory
 - ◆ Path
 - ◆ File
 - ◆ Branje in Pisanje

PRVI PROGRAM

Console Application - aplikacije brez grafičnega vmesnika

Forms Application - namizne aplikacije s podporo grafičnega vmesnika

Class Library - naenjene gradnji knjižnic razredov

Komentarji

```
//ENOVRSIČNI KOMENTAR
/*
VEČ
VRSTIČNI
KOMENTAR
*/
```

Izpisovanje na zaslon

</> Console.WriteLine

Console.Write(niz); -> kakršnokoli zaporedje znakov zapisanih med dva dvojna narekovaja

Console.WriteLine(niz); -> izpis in prehod v novo vrsto

Console.WriteLine("Izpis " narekovaja"); -> izpis narekovaja znotraj niza

Backslash

\n -> prehod v novo vrsto

\t -> tabulator

Console.WriteLine("Izpis znanka\""); -> izpis znaka \" znotraj niza

Console.WriteLine("Izpis dvojnega narekovaja"C#"""); -> izpis znaka " znotraj niza: "

Konstante, znaki, števila, nizi

Znaki: A, /, e,...

Zaporedja znakov (niz): " Tole je niz"

Cela števila: 12, 124123, 312

Decimalna števila: 4213.31231, 123.44,... (ločilo je decimalna pika)

</> operator %

Ostanek pri celoštevilskem deljenju

Console.WriteLine(7 / 4); -> izpis 3

◁▷ Celoštevilsko deljenje

Obe število sta CELI

Console.WriteLine(1 / 2); -> izpis 0

◁▷ Pravo deljenje

Console.WriteLine(1.0 / 2); -> izpis 0.5 (eden izmed števil je decimalno število)

Matematični operatorji

Matematični operatorji

➊ Matematični operatorji

+ -; seštevanje in odštevanje

/: celoštevilsko deljenje

*: množenje

%: ostanek pri CELOŠTEVILSKEM deljenju

Prioriteta operacij je enaka kot v matematiki. Če je več enakovrednih operatorjev, se vrednost izraza računa od leve proti desni.

Nizi

Nize lahko seštevamo "Mojca" + "Novak".

Kaj pa če mešamo nize in števila?

Console.WriteLine ("2 * 3 = " + 2 * 3); -> izpis bo 2 * 3 = " 2 * 3

Operator + število pretvori v niz in ga doda prejšnjemu nizu!

Pravilno:

Console.WriteLine("2 * 3 = " + (2*3));

Konstante

Ne spremenjajo svoje vrednosti

```
const int POVECAJ_ZA = 3;
```

Spremenljivke

- ◆ V njih hranimo vrednost
- ◆ So shranjene v pomnilniku
- ◆ Ime spremenljivke v C# pišemo z malo in vsako novo besedo z veliko
- ◆ Tip spremenljivke določa zalogo vrednosti
- ◆ Vnaprej jih povemo pred uporabo,
- ◆ Deklaracija, Inicializacija, Definicija

Cela števila: tip int

- ◆ Omejen obseg (+- 2mrd)
- ◆ Deklaracija + pritejanje začetne vrednosti = definicija

```
int x; //deklaracija (napoved);  
x= 10; // inicializacija (določanje začetne vrednosti)  
int x = 10; //definicija (napoved + inicializacija)
```

Prireditveni stavek

- ◆ V spremenljivko shranimo vrednost: ime_spremenljivke = izaz
- ◆ Izračuna se vrednost izraza dobljena vrednost se shrani v spremenljivko

Realna števila: tip double

- ◆ Realna števila (decimalna števila)
- ◆ Decimalna pika
- ◆ NIMA Celoštevilskega deljenja (%)

Pretvorba realnega v celo število

</> Eksplicitna konverzija

```
double realno = 21.956789;  
int celo = (int) realno; -> 21 (decimalni del se odreže)
```

</> Pretvorba z zaokroževanjem

```
double stevilo = 21.956789;  
int celo = Convert.ToInt32(stevilo); -> število se zaokrožoo -> 22
```

Standardne funkcije

- ◆ Elementarne funkcije (logaritem, sinus,...) so zajete v razredu **Math**
- ◆ Npr: Sqrt (koren), Sqr (kvadriranje), Pow (Potenciranje), Abs (absolute), Sin (sinus - argumenti v radianih),...
- ◆ **Math.PI**, **Math.E**

Zamenjava dveh spremenljivk

💡 Zamenjava dveh števil

```
int stevilo = 38;  
int enice = stevilo % 10;  
int desetic = stevilo / 10;  
int novo_stevilo = enice * 10 + desetice;
```

BRANJE PODATKOV

Metoda ReadLine

Metoda **ReadLine** razreda **Console**

- ◆ rezultat metode je niz (string)
- ◆ Tisto, kar vnesemo shranimo v spremenljivko

leftrightarrow Pretvorba iz niza v število

```
String niz = Console.ReadLine();  
int stevilo = Int.Parse(niz); ali  
int stevilo = Int.ParseInt(Console.ReadLine());
```

Metoda Read

Metoda **Read** vrne **CELO ŠTEVILO**.

Pretvarjanje celega števila v niz

Iz int v String je več načinov:

- ◆ `int` + `""` dobimo `string`
- ◆ Z metodo `Convert.ToString()`;
- ◆ z metodo `ToString()`

Naključna števila

Razred `Random` potrebujemo ga za generator naključnih števil

Ustvarjanje naključnega celega števila: metoda `Next()`

ZGORNJA MEJA NI VKLJUČENA

Ustvarjanje naključnega realnega števila: metoda `NextDouble()` - metoda je **BREZ** parametra!

🔗 Ustvarjanje naključnega celega števila

```
int maks = int.MaxValue;  
Random naklj = new Random(); // inicializacija generatorja  
int stevilo = naklj.Next(); // naključno celo št.. Med 0 in maks  
  
int poljubno = naklj.Next(500); // naključno celo število med 0 in 499 -> zgornja  
meja ni vključena  
int medDvema = naklj.Next(-30, 90); // vrne celo število med -30 in 89
```

POGOJ

IF Stavek

Naredili primer z If-elseif-else. Gnezdenja if stavka

Popoln pogojni stavek

Popolni pogojni stavek je vejitev na dve veji.

- ◆ Preverimo pogoj p
- ◆ Če je pogoj resničen (true), izvedemo stavek1 (ali več stavkov)
- ◆ Če pogoj ni resničen (false), izvedemo stavek2 (ali več stavkov)
- ◆ Veji se združita in program se nadaljuje

Nepopoln pogojni stavek

To obliko uporabimo, če takrat, ko pogoj ni izpolnjen ne naredimo ničesar

- ◆ Preverimo pogoj p
- ◆ Če je pogoj resničen (true) izvedemo stavek1 (ali več stakov, ki jih zapišemo v bloke)
- ◆ Če pogoj ni resničen (false) se ne zgodi nič
- ◆ Veji se združita, program se nadaljuje

</> If stavek

```
if(pogoj) { ... stavek1, ...}  
else { stavek1,... }
```

Relacijski operatorji

Primerjanje (relacijski operatorji)

- ◆ > večje
- ◆ < manjše
- ◆ = večje ali enako
- ◆ <= manjše ali enako
- ◆ == enako
- ◆ != različno

Logične vrednosti

- ◆ Tip `bool`
- ◆ Vrednosti le `true` in `false`

Logične operacije

Lahko združujemo z operatorji

- ◆ `&&` in
- ◆ `||` ali
- ◆ `!` ne

Gnezdeni Pogojni Stavek

Znotraj pogojnega stavka je lahko poljuben pogojni stavek

```
//Gnezdeni pogojni stavek
if(pogoj) {
    if(pogoj) {
        ...
    }
}
```

Switch

Uporabimo, ko želimo program razveziti na več vej.

```
switch (spremenljivkaAliPogojAlilzraz) //POZOR: spremenljivka ne more biti poljubnega
tipa

{
    case vrednost1: stavek1; break;

    case vrednost2: stavek2; break;

    ...

    case vrednostN: stavekN; break;

    default:

        stavki;
}
```

Vrednost, ki jo testiramo v switch stavku mora biti tipa bool, char, string, int (števni tipi): dovoljeni so še nekateri drugi tipi,...

Vsaka veja switch stavka se mora zaključiti z enim od stavkov break, continue ali pa return.

Switch stavke lahko tudi gnezdimo. S switch stavkom lahko tudi testiramo niz:

```
case "tenis": ...
```

```
short stevilo = 0;

switch (stevilo)

{

    case 0:

        Console.WriteLine("NIČ");

        return;

    case 1:

        Console.WriteLine("ENA");

        return;

}
```

```
Console.WriteLine("Stavek: ");

int pike = 0, vejice = 0, dvopicje = 0;

string stavek = Console.ReadLine();

for (int i = 0; i < stavek.Length; i++)

{

    switch (stavek[i])

    {

        case ' ':

            pike++; break;

        case ',':

            vejice++; break;

    }

}
```

```

        case 'V':
            dvopicje++; break;

    }

}

Console.WriteLine("Število pik: " + pike + "\nvejice: " + vejice + "\ndvopičja: " + dvopicje);

Console.ReadKey();

```

RANDOM CLASS

Spoznali Random razred. Naredili kratek primer

C#	Razlaga
Next(n)	Naključno celo število
Next(int n)	Naključno celo število tipa int iz intervala [0, n)
Next(int n, int m)	Naključno število tipa int iz intervala [n, m)
NextDouble()	Naključno število tipa double iz intervala [0, 1)

DEKLARACIJA IN UPORABA

```

Random rnd = new Random();
int stevilo = rnd.Next(); //naključno število
int stevilo = rnd.Next(0, 100); // število med 0 in 99

```

ZANKE

While

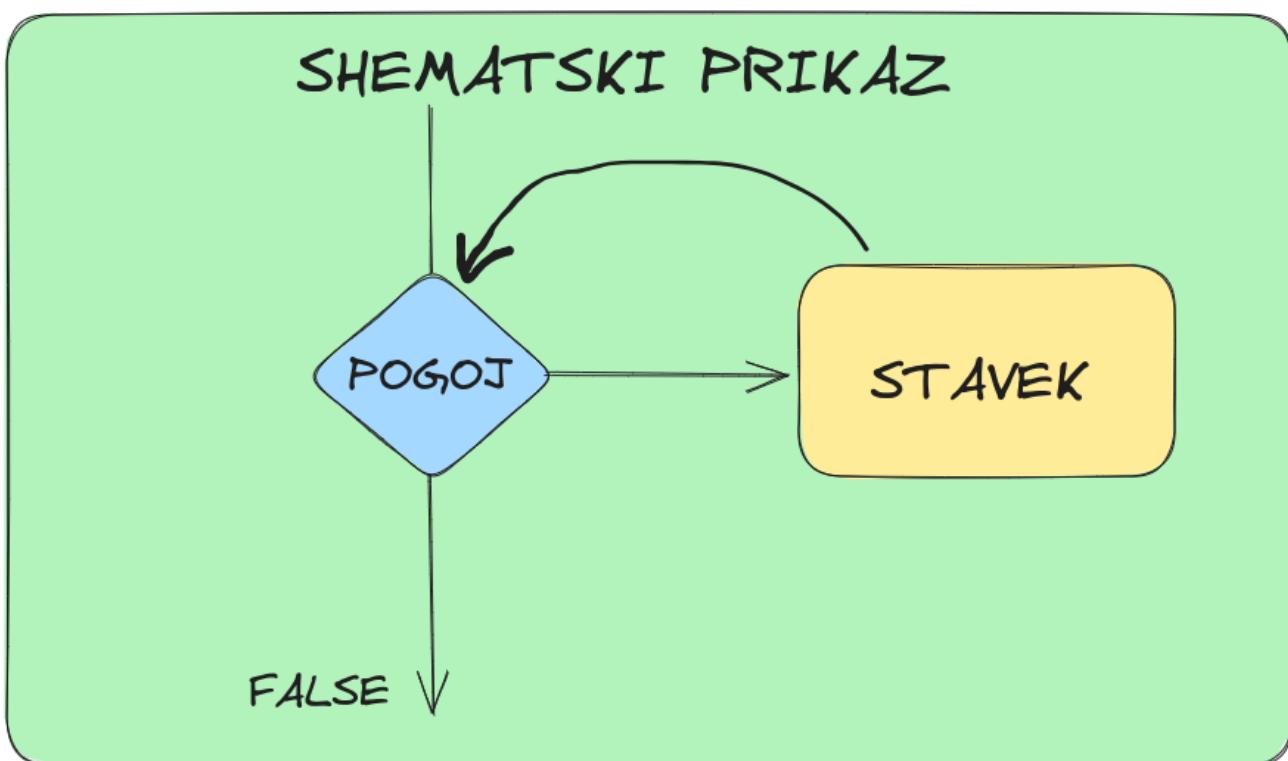
Se izvaja dokler je pogoj resničen.

WHILE SINTAKSA:

```
// WHILE SINTAKSA
while(pogoj)
{
    stavek1;
    ...
    ...
    staven n;
}
```

IZVAJANJE:

- ◆ Preveri pogoj. Če je resničen, izvedi savke v zavitih oklepajih in tako naprej
- ◆ Dokler je logični pogoj izpolnjen, se izvajajo svaki med zavitim oklepajema



NAJPOMEMBNEJŠE ZNAČILNOSTI:

Pogoj preverjamo na začetku zanke,
 Zanka se izvaja dokler je pogoj izpolnjen,
 Če pogoj ni izpolnjen že na začetku, se zanka ne izvede niti enkrat,
 Pogoj, ki ga preverjamo je lahko sestavljen

NAJPOGOSTEJŠE NAPAKE:

Napačen pogoj (zanka se nikoli ne konča)

Pozabimo na oklepaje { .. }

Zapišemo podpičje takoj za pogojem

KAKO SESTAVLJAMO PROGRAM Z ZANKO

Premislimo kaj se dogaja v splošnem (Tekoča ponovitev zanke, pišemo i-ti krog...)

Dogajanje na začetku (pred vstopom v zanko)

Dogajanje na koncu

Do while

Izvede stavke v oklepajih in nato preveri pogoj, če je resničen ponovno izvede stavek.

ZNAČILNOSTI:

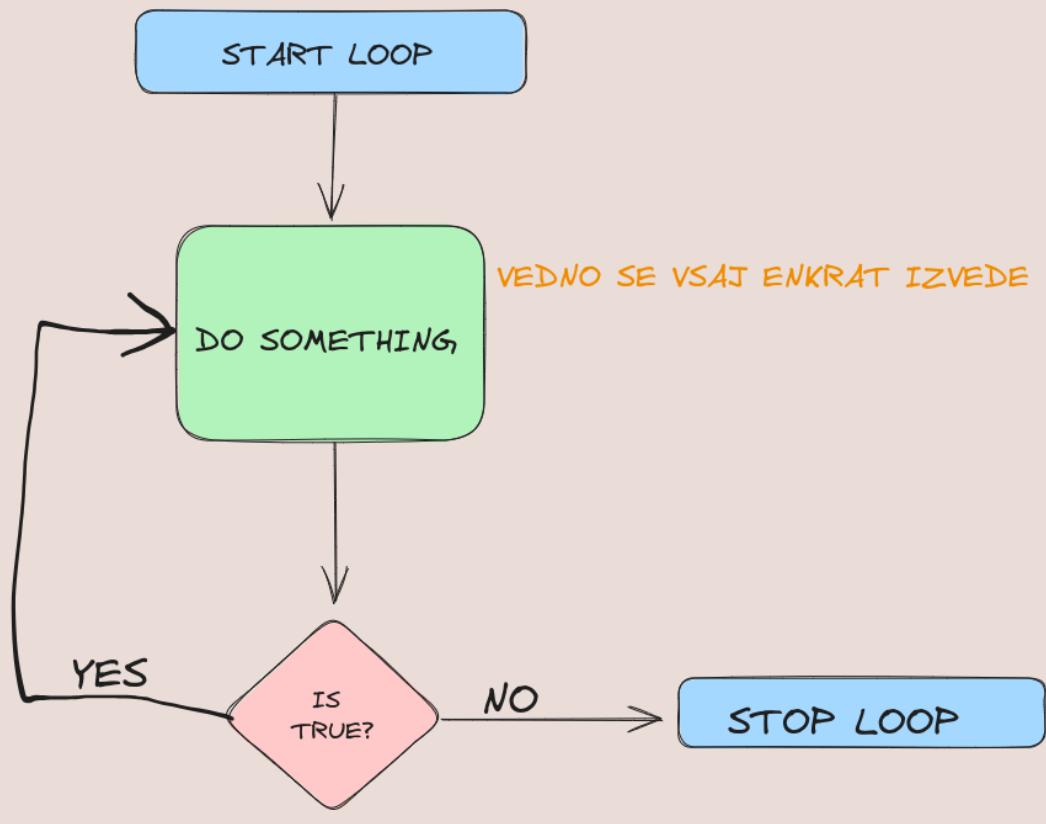
Pogoj je testiran na koncu zanke

Zanka se izvaja dokler je pogoj izpolnjen

Zanka se v vsakem primeru izvede vsaj enkrat, četudi pogoj ni izpolnjen že na začetku

Pogoj, ki ga testiramo je lahko sestavljen

```
do {  
    stavki;...  
} while(pogoj)
```

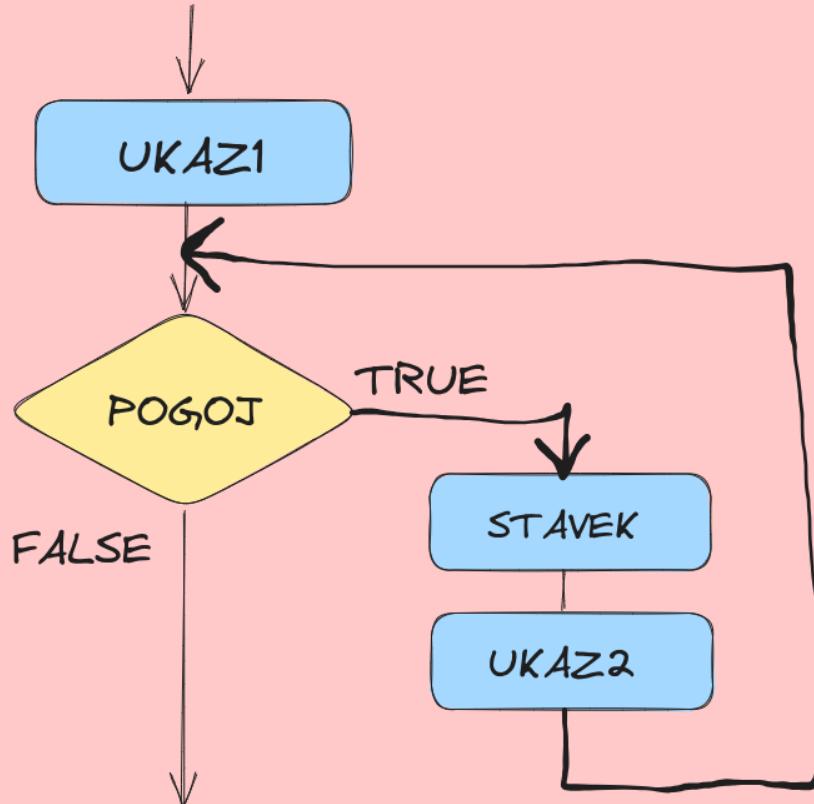


For zanka

Sestavljena je iz treh delov: *ukaz, pogoj in korak*

Števec na začetku nastavimo
Uporabljamo ko moramo šteti ponovitve.

SINTAKSA:



Paziti moramo da se zanka konča drugače lahko imamo neskončno zanko.

Lahko združimo več stavkov:

```
for(int i = 0, j=10; i < 10 && j>0; i++; j--)  
{  
    ....  
}
```

Break and Continue

break povzroči izstop iz (najbolj notranje) zanke tipa for, while, ali do while
continue pri zankah povzroči skok na pogoj zanke ter sproži ponovno preverjanje pogoja. Če je ta izpolnjen, se izvajanje zanke nadaljuje sicer pa se zanka zaključi

Foreach

Je vrsta zanke, v kateri se ponavlja množica stavkov za vsak element neke tabele ali množice objektov.

💡 Lastnosti foreach zanke

- ◆ V zanki NE uporabljam indeksov
- ◆ Elementi si sledijo **PORIČNOSTI**
- ◆ Ne moremo uporabiti za spremembo elementov tabele po kateri se sprehajamo oz. objektov
- ◆ Na začetku zanke je ustvarjena spremenljivka poljubnega tipa, ki avtomatično pridobi vrednost elementa neke tabele
- ◆ Uporabimo jo lahko le za **obdelavo celotne tabele** in ne dela tabele
- ◆ Iteracija poteka od indeksa **0** do indeksa **Length-1**, iteracija v obratni smeri NI možna
- ◆ Uporabljamo za obdelavo tabel in nizov, ter za obdelavo zbirk podatkov

💡 Splošna oblika foreach zanke

```
foreach(tip spremenljivka in tabela) {  
    ... stavki ...  
}
```

Ustvarimo enodimensionalno tabelo 10 decimalnih števil in jo napolnimo z naključnimi decimalnimi števili med 0 in 1, zaokroženimi na 3 decimalke. Vsebino tabele nato izpišimo s pomočjo zanke foreach!

```
double[] tabela = new double[10];  
  
for (int i = 0; i < tabela.Length; i++)  
{  
    tabela[i] = Math.Round(rnd.NextDouble(), 3);  
}  
  
Console.WriteLine("Vrednosti: ");  
foreach (double vrednost in tabela)  
{  
    Console.Write(vrednost + " ");  
}
```

NIZI

Niz je zaporedje znakov med znakoma " "

Če en izraz NI niz, se ta pretvori v niz!

Primer:

```
string Naslov = "Cankarjeva ulica " + 23 ;
```

Nize beremo z `Console.ReadLine()`

Do posameznega znaka pridemo tako: `niz[indeks]` pri čemer je niz ime spremenljivke z nizom, indeks pa zaporedna številka znaka. **ZNAKE ŠTEJEMO OD 0 DALJE!**

`niz[0]` - prvi znak

`niz[niz.Length - 1]` je zadnji znak

Znaki - tip char

Ustvarimo: `char znak;` ali `char znak = 'm'`

Znake lahko računamo, saj gre dejansko za kode znakov.

Primer: `(char)('a' + 2) => 'c'`

Metode in lastnosti za delo z nizi

DOLŽINA NIZA: `Length`

DOSTOP ZNAKA NA I-TEM MESTU V NIZU: `niz[indeks]`

CONTAINS: Ali niz vsebuje podani podniz

IndexOf: pozicija zaporedja znakov znotraj niza

```
//PRIMER UPORABE INDEXOF
string niz = "Programiranje 1";
int st = niz.IndexOf("gram");

Console.WriteLine(st); //dobimo izpis 3
```

EQUALS/COMPARE: Za primerjavo nizov `stavek1.Equals(stavek2)`

`String.Compare(stavek1, "bla")` -> vrne 0, če sta enaka, -1, če je stavek1 manjši od "bla", in 1, če je stavek1 večji od "bla".

INSERT: Vstavljanje podniza v niz (povemo kje in kaj želimo vstaviti)

REMOVE: Odstranjevanje podniza iz niza

REPLACE: Zamenjava podniza v nizu

SUBSTRING: pridobivanje podniza iz niza, če želimo vzeti del niza. Če podamo en parameter želimo vzeti vse znake

```
string niz = "Programiranje 1";
string noviNiz = niz.Substring(6,3); //iz niza niz želimo vzeti 3 znake od 6 znak naprej!
//Nov niz bo mir
```

ToLower/ToUpper: Pretvorba črk v nizu v male oz. velike črke

TRIM: Odstranjevanje vodilnih in končnih presledkov iz niza

PadLeft

Lahko uporabimo za lepši formatiran izpis

```
// PRIMER PADLEFT-A
Console.WriteLine(niz.PadLeft(20, '*')) // naredi tole *****PROGRAMIRANJE1
```

TABELE

Enodimenzionalne Tabele

Napoved (deklaracija) tabele: `podatkovniTip[] imeTabele;`

Incializacija tabele: `imeTabele = new podatkovniTip[velikost];`

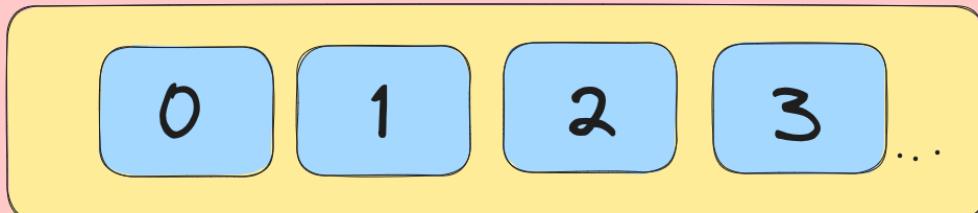
Deklaracija in incializacija: `podatkovniTip[] imeTabele = new podatkovniTip[velikost];`

 Začetna vrednost

Začetna vrednost, ko ustvarimo tabelo (ne napolnimo jo) je vedno **0** za številske vrednosti, ne glede na velikost tabele. Pri stringu je prazen niz, pri **char** pa prazen znak

INDEKSI V TABELI

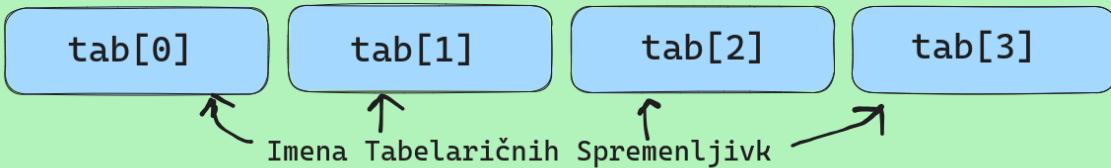
INDEKSI V TABELI



TABELARIČNA SPREMENLJIVKA

TABELARIČNA SPREMENLJIVKA

`tab[4] = 100;` ← vrednost tab.
spremenljivke
↑ ↑
Ime tabele indeks



DOLOČANJE VREDNOSTI TABELARIČNIH SPREMENLJIVK

- ◆ Tabelo najprej ustvarimo `int[] tab = new int[5];`
 - ◆ Tabelaričnim elementob priredimo vrednosti: `tab[0] = 20` , ...

- ◆ Ali pa v for zanki napolnimo vrednosti

Za izpis elementov v tabeli moramo vsakega posebej obravnavati najlažje to dosežemo s for zanko.

Dimenzijo tabele lahko določi sam uporabnik z uporabo `Console.ReadLine()`

NAJPOGOSTEJŠE NAPAKE

Napačna deklaracija in inicializacija tabele

Primerjanje dveh tabel (*ENakost dveh tabel lahko ugotavljamo le tako, da najprej primerjamo njuni dimenziji, nato pa še vsak element posebej*)

Razširitev tabele

```
Array.Resize(ref tabela, vrednost)
```

Najprej povemo referenco na tabelo torej tisto tabelo, ki jo želimo razširiti nato pa podamo vrednost, največkrat je to `tabela.Length + 1` in jo za 1 mesto povečamo.

Vstavljanje na zadnjem mestu

```
tabela[tabela.Length - 1] = 23; // Tukaj vstavimo na 3 indeksu 23
```

⚠ Shift elementov (pride prav pri seminarski)

```
for(int i = 0; i < tabela.length; i++) {  
    Console.WriteLine(tabela[i + 1]) // Zadnji element  
}
```

Tabele bodo na kolokviju/seminarski/izpitu

Zadnji elementi bodo vedno `tab.length - 1`

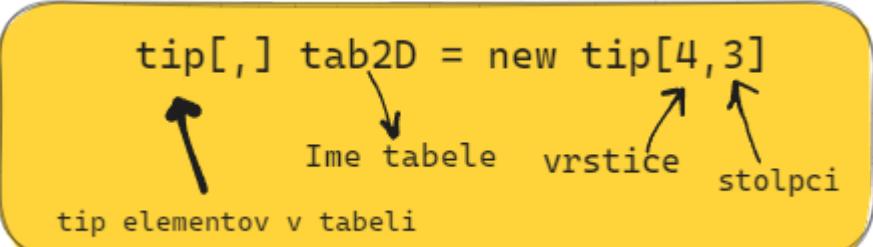
Večdimenzionalne tabele

Tabele so lahko dvo ali večdimenzionalne. Dvodimenzionalne tabele vsebujejo vrstice in stolpce pravimo jim tudi *matrike*,

SPLOŠNA DEKLARACIJA DVODIMENZIONALNE TABELE:

```
PodatkovniTip[,] imeTabele = new PodatkovniTip[vrstic,stolpcv];
```

tip[,] tab2D = new tip[4,3]



tip elementov v tabeli
Ime tabele vrstice stolpci

Primer izgleda tabele:

tab[0,0]	tab[0,1]	tab[0,2]
tab[1,0]	tab[1,1]	tab[1,2]
tab[2,0]	tab[2,1]	tab[2,2]
tab[3,0]	tab[3,1]	tab[3,2]

Lahko jo deklariramo na tri načine.

1. NAPREJ DEKLARIRAMO TABELO, KASNEJE PA JI DOLOČIMO ŠE VELIKOST

```
int[,] tabela; //deklaracija tabelarične spremenljivke  
  
tabela = new int[10,20]; /*2-dim tabela 10 vrstic in 20 stolpcv*/
```

2. OB DEKLARACIJI TABELARIČNE SPREMENLJIVKE Z OPERATROJEM NEW TAKOJ ZASEŽEMO POMNILNIK

```
int[,] tabela = new int[10, 20];
```

3. TABELO NAPREJ NAPOVEMO, NATO Z OPERATORJEM NEW ZASEŽEMO POMNILNIK IN JO ŠE INICIALIZIRAMO

```
int[,] tabela = new int[2, 3] { { 1, 1, 1 }, { 2, 2, 2 } };
```

⚠ Elementi ob inicializaciji

Elementi, so vedno 0, razen če jih mi ne določimo

ZGLED - TABELA NAKLJUČNIH ŠTEVIL

```
int[,] tabelaNakljucnihStevil = new int[5, 10];  
  
Random rnd = new Random();  
  
for (int i = 0; i < tabelaNakljucnihStevil.GetLength(0); i++) //indeks za vrstico  
{  
    for (int j = 0; j < tabelaNakljucnihStevil.GetLength(1); j++) //indeks za stolpec  
    {  
        tabelaNakljucnihStevil[i, j] = rnd.Next(-10, 11);  
    }  
}
```

💡 Dolžina tabele

Dolžino tabele dobimo z **GetLength(0)** vrstice in **GetLength(1)** stolpci

ZGLED - TABELA ŠTEVIL

```
int[,] tabela2 = new int[10, 10];  
  
for (int vrstice = 0; vrstice < tabela2.GetLength(0); vrstice++)  
{  
    for (int stolpci = 0; stolpci < tabela2.GetLength(1); stolpci++)  
    {  
        if (vrstice == stolpci) // diagonalni elementi dobijo vrednost 1  
    }  
}
```

```

    {
        tabela2[vrstice, stolpci] = 1;
    }
    else if (vrstice < stolpci) //elementi nad diagonalo dobijo vrednost 2
    {
        tabela[vrstice, stolpci] = 2;
    }
}
Console.WriteLine(String.Join(" ", tabela2.Cast<int>()));

```

ZGLED VSOTA IN POVPREČNA VREDNOST

```

Random nak = new Random();

int[,] tab = new int[10, 10];
int vsota = 0;
double povprecje = 0;

for (int vrstica = 0; vrstica < tab.GetLength(0); vrstica++)
{
    for (int stolpec = 0; stolpec < tab.GetLength(1); stolpec++)
    {
        tab[vrstica, stolpec] = nak.Next(0, 100);
        vsota += tab[vrstica, stolpec];
    }
}

povprecje = vsota / 100.0;
Console.WriteLine("Vsota števil = {0} \nPovprečje števil = {1}", vsota, povprecje);

```

ABECEDA ZNAKOV

Ustvari dvodimenzionalno tabelo znakov dimenzijs 5 x 5. Tabelo napolni z znaki abecede, od znaka 'A' naprej.

```

char[,] tab2 = new char[5, 5];
Random rnd2 = new Random();
char trenutniZnak = 'A';

for (int vrstica = 0; vrstica < tab2.GetLength(0); vrstica++)

```

```
{  
    for (int stolpec = 0; stolpec < tab2.GetLength(1); stolpec++)  
    {  
        //tab2[vrstica, stolpec] = Convert.ToChar(rnd2.Next(65, 91));  
        tab2[vrstica, stolpec] = trenutniZnak;  
        trenutniZnak++;  
    }  
}  
Console.WriteLine(String.Join(" ", tab2.Cast<char>()));
```

💡 Sortiranje tabele

Array.sort(EnoDImenzionalna tabela)

METODE

Zakaj v metodah ne izpisujemo/beremo

Če podatek preberemo metodo ne bomo mogli uporabljati, če bomo podatek npr. izračunali.

Če rezultat izpišemo, metode ne bomo mogli uporabiti, če bomo hoteli z rezultatom še kaj početi.

Če metoda nekaj vrača --> return

Če metoda nekaj izpiše --> void

Vrste metod

Prva delitev:

- ◆ **STATIČNE**
- ◆ **OBJEKTNE**

Druga delitev:

- ◆ **METODE, KI NE VRAČAJO NIČESAR**
- ◆ **METODE, KI VRAČAJO REZULTAT**

Tip metode = tip rezultata, ki ga metoda vrača

Metode, lahko sprejemajo argumente (parametri).

```
// POIŠČIMO NAJDALJŠO BESEDO V NIZU
public static string NajdaljsaBeseda(String[] tabela) {
    string najdaljsaBeseda = tabela[0];

    for(int i = 0; i < tabela.Length; i++) {
        if(tabela[i].Length > najdaljsaBeseda.Length) {
            najdaljsaBeseda = tabela[i];
        }
    }

    return najdaljsaBeseda;
}
```

Razlaga posameznih stavkov in besed

public - Vsi deli programa lahko uporabijo to metodo

static - Statična metoda

int - metoda tipa int, kar pomeni, da bo rezultat tipa **int**

Max - Ime metode, začnemo z veliko začetnico!

(int a, int b) - parametra/argumenta metode

return - vračanje rezultata (tip rezultata mora biti enak tipu metode)

Formalni argument za opis delovanja metode **(int a, int b)**

Dejanski argument je začetna vrednost argumenta metode **Max(12, 16)**

```
tip metode      ime metode      argumenti metode
↑              ↑                  ↗
public static int Sestevek(int a, int b){ formalni argumenti

    return a + b; vračanje rezultata
}
```

Klic metode

Metoda tipa `void` ne vrača ničesar kličemo tako, da napišemo ime metode in njene dejanske parametre

```
`ime(dejanski parametri)`
```

Metoda, ki ni tipa void kličemo **vedno v stavku**:

```
Console.WriteLine(ime_metode(dejanski_parametri));  
Podatkovni_tip spremenljivka = ime_metode(dejanski_parametri)
```

⚠ Pri klicu metode moramo paziti

- ◆ Tip metode
- ◆ Ujemanje parametrov
- ◆ Prenaša se vrednost parametra

Napiši metodo, ki dobi dva parametra, poljuben STAVEK in poljuben ZNAK. Metoda naj ugotovi in vrne kolikokrat se v stavku pojavi izbrani znak

```
int rez = Kolikokrat("Scuderia Ferrari", 'r');  
Console.WriteLine("Izbrani znak se ponovi {0}", rez);  
  
public static int Kolikokrat(String stavek, char znak)  
{  
    int stevec = 0;  
    for (int i = 0; i < stavek.Length; i++)  
    {  
        if (stavek[i].Equals(znak))  
        {  
            stevec++;  
        }  
    }  
  
    return stevec;  
}
```

Napiši metodo, ki ne sprejme nobenih parametrov, izračuna in vrne pa vsoto vseh dvomestnih števil, ki so deljiva s 5.

```
int rezultat = VsotaDvomestnihDeljivih();
Console.WriteLine("Vsota dvomestnih števil, ki so deljiva s 5 je {0}", rezultat);

public static int VsotaDvomestnihDeljivih()
{
    int vsota = 0;
    for (int i = 10; i <= 100; i++)
    {
        if (i % 5 == 0) vsota += i;
    }

    return vsota;
}
```

OBJEKTI

RAZREDI

Z razredom opišemo kako je neka vrsta objektov videti (*razred je abstraktna definicija objekta torej opis, kako je naša škatla videti znotraj*).

Če želimo nek razred uporabiti moramo obstajati vsaj en **primerek** razreda. Primerek razreda imenujemo **objekt**. Ustvarimo ga s ključno besedo **new**:

```
imeRazreda primerek = new imeRazreda();
```

Lastnosti objektov so zapisane v razredu (*class*). Ta opisuje katere **podatke** hranimo o objektih in katere so **metode** objektov na sporočila. Stanje objektov opišemo s spremenljivkami (**polja/komponente**), njihovo obnašanje pa z metodami.

```

public class MojR
{
    private string mojNiz;

    public MojR(string nekNiz)
    {
        mojNiz = nekNiz;
    }

    public void Izpisi()
    {
        Console.WriteLine(mojNiz);
    }
}

public static void Main(string[] arg)
{
    MojR prvi;           ← oznaka (ime) objekta
    prvi= new MojR("Pozdravljen, moj prvi objekt v C#!"); ← kreiranje objekta
    prvi.Izpisi();       ← ukaz objektu
}

```

Definicija razreda

V metodah se navezujemo na lastnosti objekta (`this`), nad katerim kličemo metodo

this je referenca na objekt, ki smo ga naredili iz razreda

PREDNOSTI DELA Z OBJEKTI:

- ◆ Uporabniku ni nikoli potrebno vedeti, kaj se dogaja znotraj objekta, objekt mora znati le ustvariti, ta pa mu sam sporoča katere lastnosti in metode pozna
- ◆ Enkapsulacija (dostopnost in skrivanje podatkov)

Če želimo nek razred uporabiti moramo iz njega ustvariti vsaj eno instanco:

`imeRazreda instanca = new imeRazreda();`

Dostopnost podatkov

- Metoda ali polje je privatno (**private**), kadar je dostopno le znotraj razreda.
- Metoda ali polje je javno (**public**), kadar je dostopno tako znotraj, kot tudi izven razreda.
- Metoda ali polje je zaščiteno (**protected**), kadar je vidno le znotraj razreda, ali pa v podedovanih (izpeljanih) razredih.

```

class Oseba {
    public string Ime;
    public string Priimek;
    public int LetnicaRojstva;

    public void izpis() {
        Console.WriteLine("Ime: " + ime + " Priimek " + priimek + " rojen leta " +
LetnicaRojstva);
    }
}

class Program {
    public static void Main(String[] args) {
        // KREIRANJE OSEB BREZ KONSTRUKTORJA
        Oseba o1 = new Oseba(); //ustvarimo objekt in mu dodelimo vrednosti
        o1.Ime = "Jure";
        o1.Priimek = "Zajc";
        o1.LetnicaRojstva = 1999; */

        Console.WriteLine("{0} {1} rojen leta {2}", o1.Ime, o1.Priimek, o1.LetnicaRojstva);
        //izpis podatkov za osebo1

        o1.izpis(); //klic metode za izpis podatkov, iste podatke izpiše kot zgornji izpis

        Oseba o2 = new Oseba(); //ustvarimo objekt in mu dodelimo vrednosti
        o2.Ime = "Janko";
        o2.Priimek = "Kresnik";
        o2.LetnicaRojstva = 1846;

        o2.izpis(); //klic metode za izpis podatkov za osebo2
    }
}

```

Ustvarjanje objektov

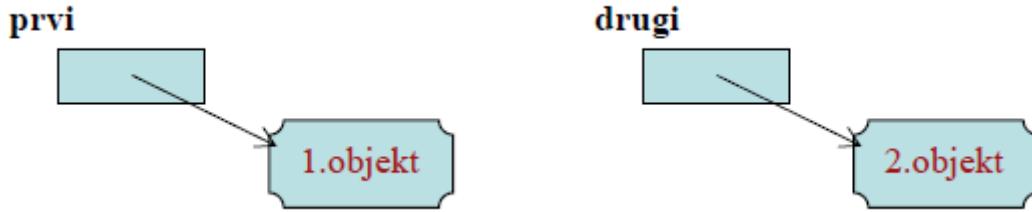
Z naslednjim ukazom v pomn. naredimo objekt tipa MojR, Novo ustvarjeni objekt se nahaja na naslovu *prvi*.

```
prvi = new MojR("Pozdravljen moj prvi objekt v C#!");
```

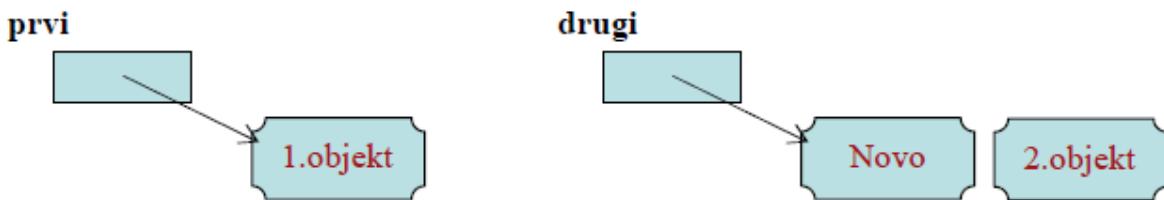
Naslov objekta

V spremenljivki *prvi* hranimo objekt no v resnici je v spremenljivki shranjen **naslov** objekta. Zato po **MojR prvi, drugi** nimamo še nobenega objekta, le dve spremenljivki v katerih lahko shranimo naslov, kjer bo operator *new* ustvaril nov objekt.

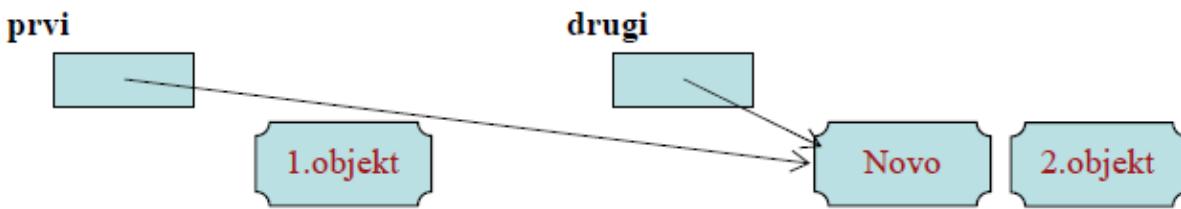
```
prvi = new MojR("1. Objekt");
drugi = new MojR("2. Objekt");
```



```
drugi = new MojR("Novo");
```



```
prvi = drugi
```



Kako se lotiti načrtovanja rešitve z OOP?

1. Z analizo ugotovimo kakšne objekte potrebujemo za reševanje
2. Pregledamo ali že imamo na voljo ustrezen razred
3. Sestavimo ustrezen razred (določimo polja in metode našega razreda)
4. Sestavimo glavni program, kjer s pomočjo objektov rešimo problem

Zgradba

Razrede pogosto uporabimo, da v celoto združimo podatke o neki stvari. Takrat v razred *zapremo* le polja, ki imajo vsa javni dostop. V razredu pa metod ni.

Primer:

```
class Zgradba {  
    public int kvadratura;  
    public int stanovalci;  
  
}
```

Če sedaj ustvarimo objekt tipa *Zgradba*

```
Zgradba hiša = new Zgradba(); // nov objekt tipa Zgradba
```

do polj *kvadratura* in *stanovalci* dostopamo z

```
hiša.stanovalci = 4 ali hiša.kvadratura = 2500
```

```
mojObjekt.nekoPolje dostop do te spremenljivke.
```

Konstruktor

Konstruktor je metoda, ki se pokliče ob kreiranju (new) novega objekta.

V razredu je lahko več konstruktorjev z različnimi parametri.

- ◆ Konstruktor je metoda za nastavljanje začetnih vrednosti polj objekta
- ◆ Nima tipa
- ◆ Ima enako ime kot razred
- ◆ Lahko jih je več a razlikovati se morajo po številu oz. tipu parametrov

this

this označuje objekt, ki ga *obdelujemo*. V konstruktorju je to objekt, ki ga ustvarjamo (nanaša se na lastnost/komponento objekta, ki se ustvari).

```
public Oseba(string ime, string priimek, int letnicaRojstva)  
{  
    this.Ime = ime;  
    this.Priimek = priimek;  
    this.LetnicaRojstva = letnicaRojstva;  
}
```

```
// KLIC OSEBE S POMOČJO KONSTRUKTORJA
Oseba o3 = new Oseba("France", "Prešern", 1800);
Oseba o4 = new Oseba("Ivan", "Cankar", 1876);

o3.izpis();
o4.izpis();
```

Preobtežene metode

Method overloading.

Znotraj istega programa imamo več istih metod razlikovati se moramo le po:

- ◆ število argumentov
- ◆ imenih argumentov

Objektne Metode

Svoj pravi pomen dobi sestavljanje razredov takrat, ko objektu pridružimo še metode, torej znanje nekega objekta. Kot že vemo iz uporabe vgrajenih razredov, metode nad nekim objektom kličemo tako, da navedemo ime objekta, piko in ime metode. Tako, če želimo izvesti metodo WriteLine nad objektom Console, napišemo

```
Console.WriteLine("Hello World");
```

Objektne vs Statične

OBJEKTNE METODE

- ◆ Objektne metode so vezane na določen objekt (instanciran primer razreda).
- ◆ Te metode lahko dostopajo do lastnosti (atributov) in metod tega objekta prek ključne besede `this`.
- ◆ Objektne metode imajo lahko dostop do stanja objekta (vrednosti njegovih atributov) ter ga lahko spreminjajo.

```
public class Oseba
{
    public string Ime { get; set; }
    public int Starost { get; set; }
```

```

public void PredstaviSe()
{
    Console.WriteLine($"Pozdravljen, moje ime je {this.Ime} in sem star {this.Starost}
let.");
}

class Program
{
    static void Main(string[] args)
    {
        Oseba oseba1 = new Oseba();
        oseba1.Ime = "Janez";
        oseba1.Starost = 30;
        oseba1.PredstaviSe(); // Klic objektne metode
    }
}

```

STATIČNE METODE

- ◆ Statične metode niso vezane na določen objekt in so lahko klicane neposredno prek razreda.
- ◆ Te metode nimajo dostopa do stanja določenega objekta (ker niso vezane nanj).
- ◆ Statične metode so običajno uporabljene za skupno funkcionalnost, ki ni odvisna od stanja objekta.

Primer:

```

public class Matematika
{
    public static int Sestej(int a, int b)
    {
        return a + b;
    }
}

class Program
{
    static void Main(string[] args)
    {
        int rezultat = Matematika.Sestej(5, 3); // Klic statične metode
        Console.WriteLine(rezultat); // Izpis: 8
    }
}

```

```
}
```

Tabele objektov

Ko sestavimo nov razred sestavimo dejansko nov tip podatkov. Ta je enakovreden v C# in njegovim knjižnicam vgrajenim tipom. Torej lahko ustvarimo **tabelo objektov**.

Sestavimo razred *Zajec*:

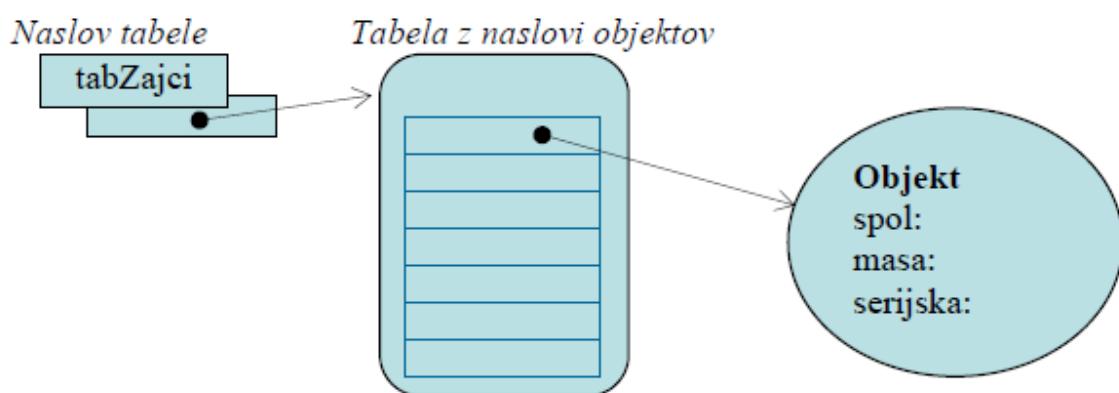
```
public class Zajec
{
    public string serijska;
    public bool spol;
    public double masa;
}
```

Recimo, da potrebujemo tabelo 100 zajcev, to lahko naredimo na slednji način:

Zajec[] tabZajci; tabela, kjer bomo hranili zajce

tabZajci = new Zajec[250]; tabela velikosti 250

V tem trenutku še nimamo nobenega objekta tipa *Zajec*, le prostor za njihove naslove. Šele ko napišemo *tabZajci[0] = new Zajec();* smo s tem ustvarili novega zajca (nov objekt tipa *Zajec*) in podatke o tem, kje tan ovi objekt je, shranili v spremenljivko.



Dostop do stanj objektov

Če ne želimo, da uporabnik dostopa do vseh podatkov oz. da jih ne spreminja nesmiselno lahko jih skrijemo.

PRIMER:

Napišimo razred Avto s tremi polji (znamka, letnik in registerska). Začetne vrednosti objektov nastavimo s pomočjo konstruktorja. Polje letnik naj bo zasebno, zato napišimo metodo, ki bo letnik spremenila le v primeru, da bo le-ta v smiselnih mejah. Napišimo še metodo za izpis podatkov o določenem objektu.

```
public class Avto
{
    public string znamka;
    // polje letnik je zasebno, zato ga lahko določimo le s konstruktorjem, sprememimo pa le z
    // metodo SpemeniLetnik
    private int letnik;
    public string registerska;
    // konstruktor
    public Avto(string zn, int leto, string stevilka)
    {
        znamka = zn;
        letnik = leto;
        registerska = stevilka;
    }
    public bool SpremeniLetnik(int letnik)
    {
        if ((2000 <= letnik) && (letnik <= 2020))
        {
            this.letnik = letnik;
            return true; // leto je smiselno, popravimo stanje objekta in vrnemo true
        }
        return false; // leto ni smiselno, zato ne sprememimo nič in vrnemo false
    }
    // metoda za izpis podatkov o določenem objektu
    public override string ToString()
    {
        return ("Znamka: " + znamka + ", Letnik: " + letnik + ", Registrska številka: "
        + registerska);
    }
    static void Main(string[] args)
    {
        // novemu objektu nastavimo začetne vrednosti preko konstruktorja
        Avto novAvto = new Avto("Citroen", 1999, "KR V1-02E");
        Console.WriteLine(novAvto.ToString()); // v izpisu bo letnik 1999
        novAvto.SpremeniLetnik(208); // letnik 208 I dovoljen, objektu se letnik E spremeni
```

```

        Console.WriteLine(novAvto.ToString()); // v izpisu bo letnik ostal 1999
        novAvto.SpremeniLetnik(2008); // letnik 2008 JE dovoljen, objektu se letnik
        spremeni
        Console.WriteLine(novAvto.ToString()); // v izpisu bo letnik 2008
        Console.ReadKey();
    }
}

```

Uporabnikom lahko torej s pomočjo zasebnih polj (*private*) preprečimo, da *kukajo* v zgradbo objektov ali pa da jim določajo nesmislne vrednosti. Če pa želijo kaj spremeniti morajo uporabiti metode. Potrebujemo torej

Metode za dostop do stanj (za dostop do podatkov v objektu) -> get

Metode za nastavljanje stanj (spreminjanje podatkov o objektu) -> set

Serializacija

Fil

Metode za delo z datotekami so v statičnem razredu File.

Najbolj uporabne metode:

- ◆ Preverimo, če datoteka obstaja `File.Exists("datoteka.txt")`
- ◆ Preberemo celo besedilo iz datoteke `string vsebina = `File.ReadAllText("datoteka.txt")`
- ◆ Zapišemo besedilo v datoteko `File.WriteAllText("datoteka.txt", "nova vsebina")`
 - ◆ Ta ukaz naredi novo datoteko, če le-ta še ne obstaja
 - ◆ Če datoteka, že obstaja, jo prepiše z novo vsebino!
- ◆ Preberemo celo besedilo iz datoteke kot seznam vrstic `string[] vrstice = File.ReadAllLines("datoteka.txt");`
- ◆ Zapišemo seznam vrstic v datoteko `File.WriteAllLines("datoteka.txt", vrstice);`
 - ◆ Ukaz naredimo novo datoteko, če ne obstaja
 - ◆ Če datoteka že obstaja, jo prepiše z novo vsebino

Datoteke - osnove

Glede na tip zapisa:

- ◆ **tekstovne** datoteke
- ◆ **binarne** datoteke

Za delo z datotekami uporabljam namenski prostor *using System.IO*;

Razredi za delo z imeniki, datotekami in potmi do datotek:

- ◆ *Directory* - za ustvarjanje, urejanje, brisanje ali pridobivanje informacij o imenikih
- ◆ *File* - za ustvarjanje, urejanje, brisanje ali za pridobivanje informacij o datotekah
- ◆ *Path* - za pridobivanje informacij o poteh do datotek

Directory

Najpomembnejše metode razreda *Directory*:

- ◆ **Exists(path)** - vrne logično vrednost, ki ponazarja ali nek imenik obstaja ali ne.
- ◆ **CreateDirectory(path)** - ustvari imenik v navedeni poti
- ◆ **Delete(path)** - brisanje imenika in njegove vsebine
- ◆ **GetFiles(path)** - pridobivanje imen datotek navedene poti
- ◆ **GetDirectories(path)** - pridobivanje imen map navedene poti
- ◆ **GetCreationTime(path)** - pridobivanje datuma kreiranja mape
- ◆ **GetDirectoryRoot(path)** - pridobivanje imena logične enote, na kateri se nahaja določena mapa
- ◆ **GetCurrentDirectory()** - pridobivanje poti do tekoče mape

Path

Najpomembnejše metode razreda *Path*:

- ◆ **GetFullPath(path)** - pridobivanje celotne poti do datoteke

File

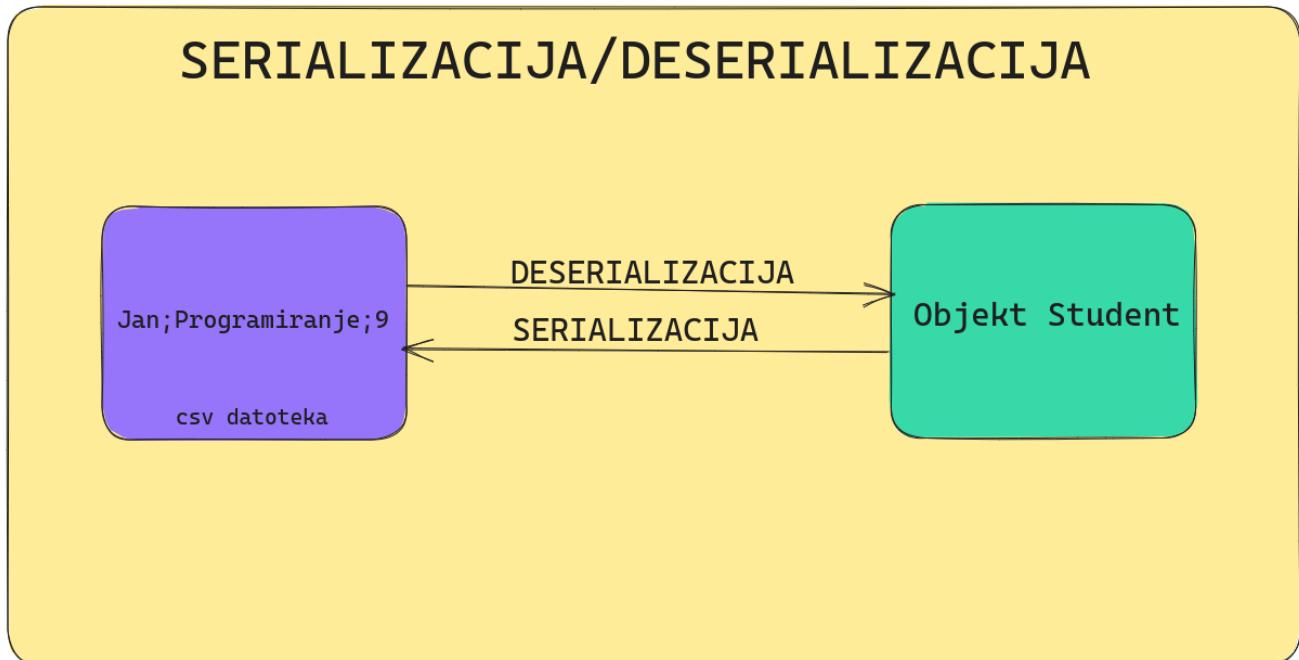
Najpomembnejše metode razreda *File*:

- ◆ **Exists(path)** - vrne logično vrednost, ki ponazarja ali neka datoteka obstaja ali ne
- ◆ **Delete(path)** - brisanje datoteke
- ◆ **Copy(source, dest)** - kopiranje datoteke iz izvorne poti do končne poti
- ◆ **Move(source, dest)** - premik datoteke iz izvorne poti do končne poti
- ◆ **CreateText(path)** - ustvarjanje ali odpiranje tekstovne datoteke za pisanje; parameter path vsebuje pot do datoteke in njeni ime
- ◆ **OpenText(path)** - odpiranje obstoječe tekstovne datoteke za branje; pot + ime datoteke
- ◆ **OpenRead(path)** - odpiranje obstoječe datoteke za branje. pot + ime

- ◆ **OpenWrite(path)** - odpiranje obstoječe datoteke za pisanje; pot + ime
- ◆ **AppendText(path)** - odpiranje obstoječe tekstovne datoteke za pisanje in dodajanje teksta v to datoteko. pot + ime. Ustvari datoteko če ne obstaja
- ◆ **ReadAllText(path)** - odpiranje obstoječe tekstovne datoteke, branje vseh vrstic iz te datoteke, nato pa še zapiranje datoteke. pot + ime
- ◆ **WriteAllText(path, string stavek)** - ustvarjanje nove datoteke, zapis stavka v datoteko in zapiranje datoteke. Če datoteka s tem imenom že obstaja bo prepisana
- ◆ **WriteAllLines(path, string[] tabela)** - ustvarjanje nove datoteke, zapis stavkov (*iz tabele stavkov*) v datoteko in zapiranje datoteke. Če datoteka s tem imenom že obstaja bo prepisana.
- ◆ **AppendAllText(path, stavek)** - zaps stavka v datoteko. Če datoteka še ne obstaja bo ustvarjena nova.

Serializacija je zapis podatkov v datoteko

Deserializacija je obratno od serializacije. Iz serializiranih podatkov v nizu kreiramo objekt(e) v programu.



Branje in Pisanje

StreamWriter - pisanje

StreamReader - branje

`File.CreateText("Datoteka.txt")` - kreiranje datoteke

`File.CreateText(@"C:\PRO1\Datoteka.txt")` - v drugi imenik (@ - znaki se jemljejo dobesedno)

```
Console.WriteLine("Ime datoteke: ");
string ime = Console.ReadLine();

if(File.Exists(ime)) Console.WriteLine("Datoteka s tem imenom že obstaja!");
else {
    File.CreateText(ime);
    Console.WriteLine("Datoteka ustvarjena");
}
```

Primer zapisovanja v datoteko

```
StreamWriter sw = File.CreateText("Datoteka.txt");
sw.WriteLine("Novi stavek");
sw.Close(); //OBVEZNO ZAPREMO DA SE ZAPIŠEJO PODATKI V DATOTEKO
```

```
public static void BranjePodatkov()
{
    string vsebinaDatoteke = File.ReadAllText("Student.txt");

    string[] deli = vsebinaDatoteke.Split(":");

    double vsota = 0;
    double povp = 0;

    string ime = "";

    for (int i = 1; i < deli.Length; i++)
    {
        vsota += Int32.Parse(deli[i]);
        ime = deli[0];
    }

    povp = vsota / (deli.Length - 1);
    Console.WriteLine("Student " + ime + " ima povprečno oceno " + povp );
}
```

```
public static void BranjeZnakov()
{
    StreamReader sr = File.OpenText("Student.txt");
    int beri = sr.Read();

    while (beri != -1)
    {
        Console.WriteLine((char)beri);
        beri = sr.Read();
    }

    sr.Close();
}
```

```
public static void Kraji()
{
    StreamReader sr = File.OpenText("Kraji.txt");
    StreamWriter sw = File.CreateText("Kraji2.txt");
    string vrstica = sr.ReadLine();

    int vsotaPreb = 0;
    string najvecjiKraj = "";
    int najvecPreb = 0;

    while (vrstica != null)
    {
        string[] deli = vrstica.Split(";");
        int trenutnoPreb = Int32.Parse(deli[2]);
        vsotaPreb += Int32.Parse(deli[2]);
        if (najvecPreb < trenutnoPreb)
        {
            najvecPreb = trenutnoPreb;
            najvecjiKraj = deli[0];
        }

        sw.WriteLine(deli[0] + " - " + deli[1]);

        vrstica = sr.ReadLine();
    }
}
```

```

        sw.WriteLine(vsotaPreb);

        Console.WriteLine("Največji kraj je " + najvecjiKraj + " s " + najvecPreb + " prebivalci");

        sr.Close();
        sw.Close();
    }

```

V tekstovni datoteki Zneski.txt je zapisano neznano število števil tipa double. Vsebino datoteke prenesite v tabelo, zneske tam uredite po velikosti in jih nato zapišite nazaj v datoteko z istim imenom.

```

public static void Zneski()
{
    Random rnd = new Random();
    int stZapisov = rnd.Next(100, 201);

    string vsebinaDatoteke = "";

    for (int i = 0; i < stZapisov; i++)
    {
        vsebinaDatoteke += rnd.NextDouble() * 100 + "\n";
    }

    File.WriteAllText("Zneski.txt", vsebinaDatoteke);

    string[] tabZneskov = new string[stZapisov];

    for (int i = 0; i < tabZneskov.Length; i++)
    {
        tabZneskov[i] = (rnd.NextDouble() * 100).ToString();
    }

    File.WriteAllLines("Zneski1.txt", tabZneskov);
    string[] tabStevil = File.ReadAllLines("Zneski.txt");
    Array.Sort(tabStevil);
    File.WriteAllLines("Zneski.txt", tabStevil);
}

```

Ustvari tekstovno datoteko Stevila.txt. Vanjo zapiši mnogokratnike števila 5 od 0 do +200, v vsako vrstico natanko 10 števil, med števili naj bodo

presledki. Nalogo reši na dva načina (zapis celotne vsebine datoteke naenkrat, zapis po vrsticah). Na koncu obstoječo datoteko prepišite z novo, v kateri bodo še rezultati vsote treh števil v vsaki vrstici!

```
public static void Stevila()
{
    StreamWriter pisi = File.CreateText("Stevila.txt");
    int stVrstic = 0;

    for (int i = 0; i < 200; i += 5)
    {
        pisi.Write(i + " ");
        stVrstic++;

        if(stVrstic % 10 == 0) pisi.WriteLine();
    }

    pisi.Close();

    string[] tabela = File.ReadAllLines("Stevila.txt"); //vsebino datoteke prenesemo v tabelo nizov
    //v vsaki vrstici te tabele so cela števila ločena s presledki razen v zadnji
    //Tabelo bomo obdelali in vsako vrstico splitali glede na presledke  for(int i = 0; i <
    tabela.Length; i++)
    {
        string[] tabStevil = tabela[i].Split(' ');
        int vsota = 0;

        //Obdelamo tabelo števil a izpustimo zadnjo vrstico ker je prazna
        for (int j = 0; j < tabStevil.Length - 1; j++)
        {
            vsota += Int32.Parse(tabStevil[j]);
        }

        tabela[i] += " Vsota " + vsota;
    }

    //Obdelamo tabelo zapišemo nazaj v datoteko
    File.WriteAllLines("Stevila.txt", tabela);
}
```

**V datoteki Manekenke.txt so shranjeni podatki o velikosti manekenk. V vsaki vrsti je zapisana velikost manekenke v centimetrih (celo število), nato pa sledita ime in priimek.
Polja so ločena z dvopičji.

Primer datoteke:

179:Cindy:Crawford

182:Naomi:Campbel

185:Nina:Gazibara

180:Elle:Mac Perhson

180:Eva:Herzigova

Napiši program, ki prebere to datoteko in na zaslon izpiše višino, ime in priimek največje in najmanjše manekenke. V zgornjem primeru bi program deloval takole:
Najmanjša je Cindy Crawford, ki meri 179 cm.
Največja je Nina Gazibara, ki meri 185 cm.**

```
public static void Manekenke()
{
    StreamReader beri = File.OpenText("Manekenke.txt");
    string vrstica = beri.ReadLine();

    int maxVisina = 0;
    string najVecjaManekenka = "";

    int minVisina = 9999;
    string najmanjsaManekenka = "";

    while (vrstica != null)
    {
        string[] podatki = vrstica.Split(":");
        int trenutnaVisina = Int32.Parse(podatki[0]);

        //shranujemo največjo manekenko
        if (trenutnaVisina > maxVisina)
        {
            maxVisina = trenutnaVisina;
            najVecjaManekenka = podatki[1] + " " + podatki[2];
        }
        else if (trenutnaVisina < minVisina)
        {
            minVisina = trenutnaVisina;
            najmanjsaManekenka = podatki[1] + " " + podatki[2];
        }
    }
}
```

```
        }

        if (minVisina > trenutnaVisina)
        {
            minVisina = trenutnaVisina;
            najmanjsaManekenka = podatki[1] + " " + podatki[2];
        }

        vrstica = beri.ReadLine();
    }

    Console.WriteLine("Največja manekenka je " + najVecjaManekenka + ", ki meri " +
maxVisina + "cm");
    Console.WriteLine("Najmanjša manekenka je " + najmanjsaManekenka + ", ki meri " +
minVisina + "cm");

    beri.Close();
}
```