PROGRAMSKI JEZICI:

ε - prazan string

GRAMATIKE:

abab

*	a*	ε, a, aaa, aa, aaaa ... ✓ b, 5, %, ab ↯
+	at aa*	a, aa, aaa, aaaa ... ✓ b, 5, %, ab, ε ↯
?	a?	ε, a ✓ b, 5, ab, b, aab ... ↯
{}	a{4} a{1,3} a{8,} a{,4}	aaaa a, aa, aaa aa, aaa, aaaa, ... ε, a, aa, aaa, aaaa
|	a|e|i|o|u	a ili e ili i ili o ili u
[ ]	[aeiou] ⇔ a|e|i|o|u može da bude i raspon [a-zA-z] [a-z]+ ⇒ [a-z][a-z][a-z] ...	

a+b+ aaa...bbb...
(ab)+ ababab...

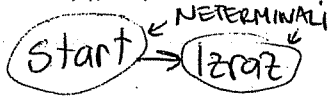
Napraviti reg. izraz za:

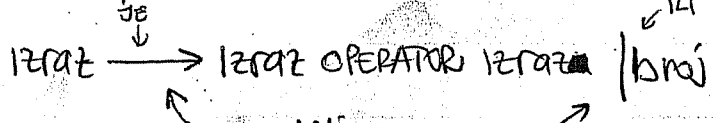
1. cele brojeve sa opcionim predznakom
   -?[1-9][0-9]* | 0
2. binarni string koji sadrži barem tri jedinice, jednu za drugom
   [0 1]* 111 [0 1]*
   (0|1)* 111 (0|1)*
   (0|1)* 1{3} (0|1)*
3. tekst (samo slova) koji počinje na slovo između b i f, ima bar 3 slova, treće slovo je m:
   [b-f][a-z]m[a-z]* ili [b-fB-F][a-zA-Z](m|M)[a-zA-Z]*
4. Imena promenljivih u programskom jeziku Java (mogu da počnu slovom, a sadrže slova, cifre i "_"):
   [a-zA-Z_][a-zA-Z_0-9]*
5. Pretime profesora ispred koga ide njegova titula: (Mr|dr)[A-Z](a-z)+
6. Datum rođenja: ne može da se uradi preko ovoga!

BNF: BAKUS-NAUROVA FORMA

1. terminali (tokeni)
2. neterminali
3. produkciona pravila
4. startni simbol

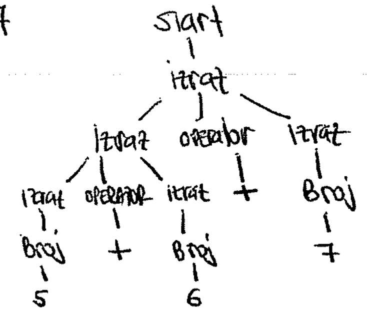
} sve to treba da definišemo da bi dobili gramatiku





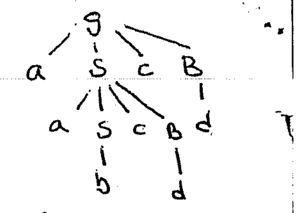
TERMINALI → Broj:: -?[1-9][0-9]*|0
→ Operator:: [+-*/...]

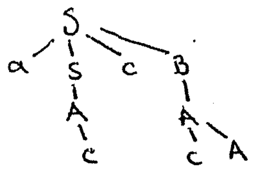
1. 5+6+7



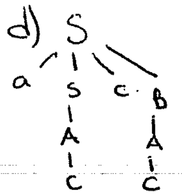
2. S → a S c B | A | b
A → c A | c
B → d | A

Koji od sledećih valja:
a) aabcdcd ✓
b) acccbd ✗
c) acd ✗
d) acccv ✓

a) 

b) 

ovo ne može...

d) 

C#:

```
using System;
namespace Primeri
{
  class Klasa
  {
    static void Main(string[] args)
    {
      System.Console.WriteLine("Hello World");
    }
  }
}
```

- ne moramo da pišemo System kod System.Console.WriteLine(); (*)
- namespace – logički grupiše klase, nešto kao paket, samo što logički pravi celinu, a paket fizički;
- Main – veliko slovo iako nije javna metoda jer treba neko da je pokrene; int može; malo slovo (string); niz; naziv promenljive može bilo koji da bude
- System – namespace; Console – klasa koja definiše kako se ponaša konzola; WriteLine – metoda klase Console za ispis

EXCEPTION – dešava se prilikom izvršavanja;
ERROR – dešava se kad prog. ne uspe da se pokrene;

Private – pristup samo u okviru te klase;
Public metode počinju velikim slovom;
FREE FORM (JAVA i C#) – nema ograničenja gde pišemo kod (može sve u jednoj liniji);

• Hello ko piše, nije potrebno u C#

```
namespace Primeri
{ class A {...}
  class B {...}
}
```

može biti više istih namespace-ova; moraju da se zovu drugačije od klasa iz prvog namespace-a

MOŽE:
namespace Primeri.Klase.Prva ← pod-namespace-ovi

Common Type System (CTS) → skup tipova zajedničkih za sve programske jezike; postoje:

1) UGRAĐENI:

a) vrednosni:

celobrojni		decimalni	
byte 8bit	sbyte	float 32bit	char 16bit
short 16	ushort	double 64	boolean 2bit
int 32	uint	decimal 128	
long 64	ulong		

b) referentni: System.Object object
System.String string

* byte – samo pozitivni brojevi od 0 do 255
sbyte – može i znak

2) KORISNIČKI-DEFINISANI:
a) vrednosni: struktura, enumeracija;
b) referentni: klase, nizovi, delegati;

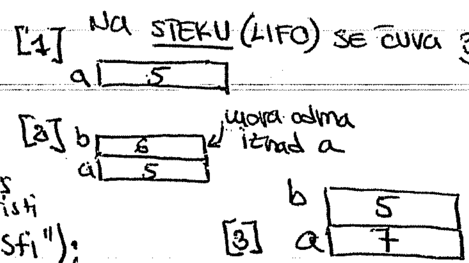
U C# NEMA POKAZIVAČA; KOD VREDNOSNIH...

PRIMER:
u main metodi pišemo

```
{ int a;           int a=5;
  a=5; [1]
  int b;
  b=6; [2]
  if(a==b)
    Console.WriteLine("isti");
  b=a;
  a=7; [3]
  index i;   [4]
  i=new index(5,2015); [5]
  index j;
```

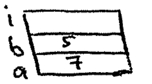
- if(a==b) – može i equals ali se ne koristi

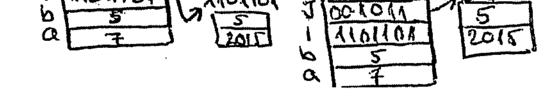
[1] na STEKU (LIFO) se čuva 32bita; (int)



[4] Pretpostavili smo da ima neki konstruktor u klasi index koji pozivamo;

```
class index
{
  public int broj;
  public int godina;
  public index(int broj, int godina)
  {
    this.broj = broj;
    this.godina = godina;
  }
```



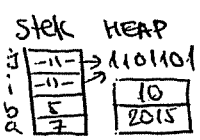


```
if (i==j)        [7]
    Console.WriteLine("isti")
```

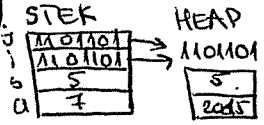
[7] NISU ISTI JER IMAJU različite adrese, iako vrše isto imaju iste vrednosti!

```
j=i;  [8]
i.broj = 10;  [9]
```

[9]



[6]

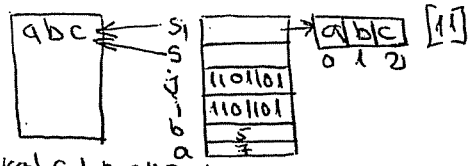


SADA SU i i j isti!

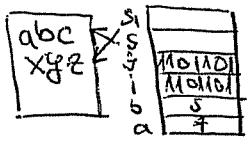
GARBAGE COLLECTOR POSLE NEKOG VREMENA „SKUPLJA" ONAJ HEAP SA ADRESOM 001011 ali može i da se upravlja? NE MOŽE NEMARIDOM - NEĆEMO RADITI. U JAVI

```
string s;
s="abc";
string s1;
s1="abc";
if (s==s1)
    Console.WriteLine("isti");
s1="xyz";  [11]
```

[10] ↑ isti su!



[11]



String je niz charova char[];

Kad god poredis stringove, poredis tekst a ne adrese! String ne možemo da promenimo već se kreira novi ispod.

String jeste referentni, ali se ponaša kao vrednosni tip;

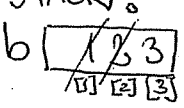
ENUMERACIJA → upis direktno u namespace-u
→ koristimo ih kad imamo potrebu da koristimo fiksne vrednosti (fiksni skup vrednosti)
→ u osnovi svake se nalazi 32-bitni integer (numerički tj. vrednosni su tip)
→ vrednosni su tip (radi se sa stekom)

```
enum Boja
{
    Plava,  //0 =5
    Zelena, //1 =2
    Crvena  //2 =10
}
```

možemo da im dodelimo vrednosti

```
Boja b;
b = Boja.Zelena;  [1]
Console.WriteLine(++b);  [2]
Console.WriteLine(++b);  [3]
if (b == Boja.Zelena)
```

STACK:



← ovako poredimo

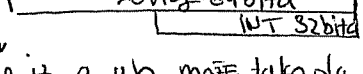
dovoljno

```
enum Boja
{
    Plava = Crvena, //3
    Zelena = 2,
    Crvena //3
}
```

← uvek je za 1 veća od prethodne

KOMPATIBILNOST TIPOVA:

```
int a = 5;
long b = 10;
b = a;  ✓   ← kopira se
a = b;  ↯   ← prepisuje se iz većeg(b) u manji(a)
```



Prepisuje iz a u b, može tako da se napiše zato što je iz manjeg u veće (32 u 64 bita). To je implicitan tapis. Eksplicitan (gde može i ne mora) bi bio: b=(long)a;

Pa dolazi do odsecanja 32 bita, ne može implicitno da se napiše već samo eksplicitno: a=(int)b;

```
double c=6.7;   ← 64bita
c=a;  ✓
a=c;  ↯
a=(int)c   ← odseca decimale i dodatne bitove, eksplicitno dobijamo broj 6 (**)
```

(*) Ovo se zove kastovanje i važi isključivo za numeričke tipove (int, float, double, ..., char, enum) kao i za objekte;

```
String s = Console.ReadLine();
```

(*) Convert važi za bilo koji tip;

```
a = Convert.ToInt32(b/c/s);   ← eksplicitno ne odseca nego zaokružuje npr. kad pusg c ćemo dobiti 7 (a gore smo dobili 6) (**)
```

Klasa koja se nalazi u System Namespace-u

Metoda klase Convert koja prebacuje u int

ToSingle ← u float

ToChar

5.0
5,0 `a = int.Parse(s);`
$5.00 `a = int.Parse(s, FormatProvider);`

⊛ PARSIRANJE JE SAMO ZA PREBACIVANJE STRINGOVA U INTEGER;

---

null - kad nešto nema vrednost, vrednosni tipovi ne mogu biti null (prazan "string ≠ NULL)

nullable tipovi - proširenje vrednosnih tipova tako da mogu da koriste null;

`nullable<int> X;` ← može da se napiše i kao: `int? X;`

`X = 5;`
`X = null;`
`int a;`
`a = null;` ↯ ← ne može
`X = a;` ✓ jer je vrednosni tip;
`a = X;` ↯ ← ne može veći u manji;
`a = (int)X` ↯ ← zašto?

int i null → X → a (samo int)

Proveriti:
$(0.1)^a + (0.2)^b = C$
da li je = 0,3

```
int a=5;
int b=5;
Console.WriteLine(a,b); ↯
```
← moramo da spojimo, može sa „+", ali pastoji; FORMAT METODA klase string

```
s = String.Format("Prvi broj: {0}, drugi: {1}", a, b);
```
isti placeholder može više puta da se ponovi, a i b popunjavaju odgovarajuće placeholdere, respektivno {0} i {1};

- Ako želimo da ispišemo {5}, stavićemo sledeći placeholder: {{{0}}}

`s = String.Format("Prvi broj je {0,6}, a drugi {1}", a, b);`
⇒ Prvi broj je ------5, a drugi 6

`\t` ← deli string; /tab
`\n` ← Prelazak u drugi red;

`Console.WriteLine("{0}{1}", a, b);`
↑ može i ovako, ne mora preko FORMAT

{1:C} ← tretira b kao valutu;
{1:P} ← -11- kao procenat;
{1:d} ← -11- mali datum;
{1:D} ← -11- veliki datum;

poravnanje je desno, ako želimo levo pišemo {0,-6}. Ako stavimo prevelik npr. string, a prevelo pratnih mesta odredimo, neće da iseče nego ispiše celo;

`String s = "C:\MyDocument\Proba";` ← buniće se zbog „\", zato pišemo „\\"
→ alternativa za oba problema je da napišemo `s = @"-11-";` ← ako želimo da vidimo navodnike, dupliramo ih "" "" ← @ verbatim stringovi

---

Domaći:

1. Tip DateTime ← 64 bita

13 je 1Pm

`DateTime dt = new DateTime(1992, 8, 19, 3, 32, 52);`
godina, mesec, dan, sati, minuti, sekunde

`Console.WriteLine(dt.ToString());` ← Pretvara u String

`= DateTime.Today;` ← vraća današnji datum bez vremena;
`.Now` ← -11- sa vremenom;
`.IsLeapYear` ← da li je prestupna godina

DateTime se tretira kao vrednosni tip ⇒ ide na stack

DateTime je (struct) a ne klasa pa zato dobijamo DateTime objekat a ne referencu kada deklarišemo varijablu da je to tipa. ON NE MOŽE da bude null (kao ni vrednosni tipovi), zato što nije referenca, ali može da bude nullable.

STRUCT - vrednosni tip koji se najčešće koristi da enkapsulira malu grupu povezanih varijabli npr.:

```
public struct Knjiga
{
    public decimal cena;
    public string naslov;
```
može da sadrži konstruktore, metode, konstante itd. ne može da bude protected zato što ne može da

2. OPERACIJE NAD TIPOM STRING

* STRINGOVI SU TIPA System.String

U C# == i != VAŽE i ZA Stringove (za razliku od Jave gde mora .equals);

1] **Split:** vraća niz stringova. Delovi originalnog stringa se zovu tokeni

```
string test = "ovo.je.test";
string[] tokeni = test.Split('.', '!');
```

* nije moguće menjati string već se prave kopije;
s[0] ← vraća slovo na prvoj poziciji reečnika;

2] **aggregate:** od niza stringova pravi jedan string

```
string test = "ovo je test";
string[] tokeni = test.Split(' ');
//Sad vraćamo na POČETNO stanje:
string test1 = tokeni.Aggregate((aggregate, current) => aggregate+" "+current);
```

3] **SUBSTRING:** izvlači deo stringa

* metode uvek imaju veliko početno slovo;

```
string test = "ovo je test";
string sub1 = test.Substring(6); // "test"
string sub2 = test.Substring(4,3); // "je"
```

4] **FORMAT:** radili smo ga ranije (sa placeholderima {0})

5] **IndexOf:** Vraća poziciju na kojoj se javlja određeni karakter ili više njih

← koristimo kad nam treba SUBSTRING ALI NE ZNAMO POZICIJU NA KOJOJ DA GA TRAŽIMO;

```
string test = "test";
int indeks1 = test.IndexOf("e"); //1
int indeks2 = test.IndexOf("st"); //2
int indeks3 = test.IndexOf(neštodrugo); //-1 ← nije ga pronašao
```

6] **REPLACE:**

```
string bad1 = "s#mething";
string bad2 = "is, happening.";
string fixed1 = bad1.Replace('#', 'o');
string fixed2 = bad2.Replace(".", string.Empty);
```

isto što i "" ← pratan string;

7] **TRIM:** za brisanje pratnog prostora ili nekih delova stringa...

```
string bad1 = "   ovo je string   ";
string bad2 = "aaovo je stringb";
string bad3 = "   ovo je string";
string bad4 = "ovo je string   ";
string fixed1 = bad1.Trim();
string fixed2 = bad2.Trim('a','b');
string fixed3 = bad3.TrimStart();
string fixed4 = bad4.TrimEnd();
// all fixed strings su isti :  "ovo je string"
```

8] Cases: pretvaranje slova u mala/velika

```
string up = "abcd".ToUpper();
string down = "ABCD".ToLower();
```

9] Insert: ubacivanje stringa na odredenu poziciju

```
string str = "NIKOLA SE PREZIVA";
string pun = str.Insert(18, "ugljicic");
```

10] Remove: brisanje stringova od odredene pozicije do kraja ili do druge pozicije

```
string str1 = "0123456";
string rezultat = str1.Remove(3); //012
string rezultat2 = str1.Remove(3,5); //0126
```

11] Length: vraća duzinu stringa

```
string ime = "Nikola";
Console.WriteLine(ime.Length); //6
```

OPERATORI: %, &&, ||, ==, !=, +=, -=, *=, ...

a++; } nema razlike osim ako
++a; } su u sklopu dvaju naredbi, npr.

```
int a=5;
Console.WriteLine(a++); //5
Console.WriteLine(++a); //7
```

prvo ispiše pa poveća za 1;
prvo poveća za 1, pa ispiše;

( )?□:□

```
if(Console.ReadLine() == "Da")
    a=5;
else
    a=0;
```

```
a = (Console.ReadLine() == "Da")?5:0;
```

mora biti istog tipa ili kompatibilno sa a;

```
int a = x?? b?? c?? -1;
```

Probaj x; Probaj b (ako ne prode x); Probaj c (ako pretnadni ne produ); uradi ako ništa ne prode;

```
switch( )
{
    default: ...
    case Boja.Plava
    ....
    ....
    break; //return...;
    case Boja.Zelena
    ....
    ....
    break;
}
```

radi default na kraju, nije bitan redosled;

```
switch( )
{
    case "da":
    case "dA":
    case "Da":
    :
}
```

ne mora break, ako nema nijedna linija koda ispod case, u suprotnom mora;

```
while( ) { }
```

```
do { } while( )
```

```
for( ) { }
```

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

ispisuje:

i	j
0	9
1	8
2	⋮

KOMBINOVANE PETLJE:

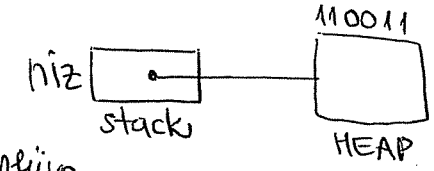
```
int a=0;
while (a<10)
{
    if(a==5)
        break;
    Console.WriteLine(a);
    a++;
}
```

ispisuje:
0
1
2
3
4

NIZOVI:

imamo implementirane red, stek itd. (iz struktura podataka);

```
using System.Collections;
ArrayList niz;
niz = new ArrayList();
niz.Add(1);
niz.Add(2);
niz.Count; //2
niz.Add("aac");
niz.Add(new Osoba("Pera"));
niz.Remove(1);
niz.Remove("aac");
niz.RemoveAt(0);
Console.WriteLine(niz[3]); //Primeri.Osoba
Console.WriteLine(niz[1].Ime);
Console.WriteLine(((Osoba)niz[3]).Ime); //ispisuje Pera
```



niz promenjive dužine

Brise članove niza koji se tako zovu → imaju tu vrednost

briše član na prvoj poziciji

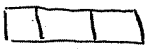
namespace

unutar ArrayList
*sve JE TIPA OBJECT!
NETIPIZIRAN ⇒ Nije Generic!

```
using System.Collections.Generic; ← isto kao gore sem što je tipizirano
List<int> niz;
List<Osoba> nizOsoba;
niz = new List<int>();
nizOsoba = new List<Osoba>();
nizOsoba.Add(new Osoba("Nikola"));
Console.WriteLine(((Osoba)nizOsoba[0]).ime); //Nikola
```

bez toga bi prijavio grešku!

*Lista je klasičan niz



LinkedList je ulančana Lista

spregnute liste

Performanse:

1) kada radimo sa vrednosnim tipovima, bolje je sa List (tipiziranim);
2) kada radimo sa referentnim tipovima onda je svejedno;

PROLAZ KROZ NIZ:

```
for (int i=0; i<niz.Count; i++)
{
    Console.WriteLine(niz[i]);
}
```

Length

```
foreach (int broj in niz)
{
    Console.WriteLine(broj);
}
```

* ne MOŽE npr. svaki drugi element ili npr. ako ŽELIMO ispis od poslednjeg do prvog;

Metoda – skup naredbi kojima smo dali neko ime jer su celina ili ih treba koristiti VIŠE puta;

Struktura = KLASA, osim što ne MOŽE DA SE nasleđuje;

```
{
Public void Inicijalizuj()
{ int broj=0;
}
Public void Dodaj()
{ broj++; ← ⇵ JER JE BROJ LOKALNA PROMENLJIVA;
}
}
```

```
{ int broj=8; ← globalni
Public void Inicijalizuj()
{ int broj = 0; broj = 0;
}   ↑ lokalni   ↑ globalni
Public void Dodaj()
{ broj++;
}
}
```

Samo za **for** petlju važi:

```
for (int i=0; i<5; i++)
{
...
} ← i se briše, ali samo vrednost... ostaje u memoriji sačuvano i
int i= ...; ← ne može! nema gde da upiše JER je ostalo zauzeto od for petlje...
```

stek

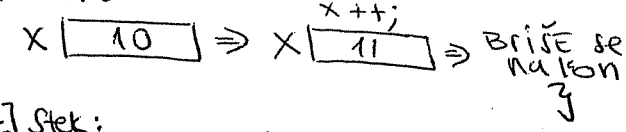
- **PRENOS PREKO VREDNOSTI:**

```
Public void Dodaj(int x)
{ x++;
}
Public void PromeniIme(Osoba y)
{ y.Ime = "Pera";
[*] y = new Osoba("Pera");
}
```

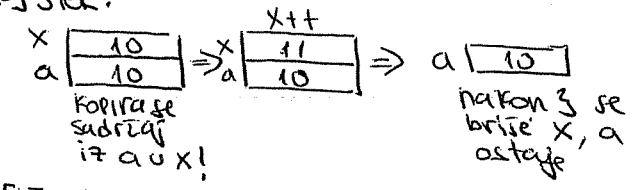
MAIN METODA:

```
Dodaj(10); [1]
int a = 10;
Dodaj(a); [2]
Osoba b = new Osoba("Lazar"); [3]
PromeniIme(b); [4]
```

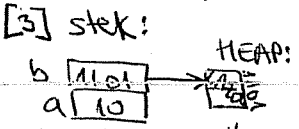
[1] STEK:



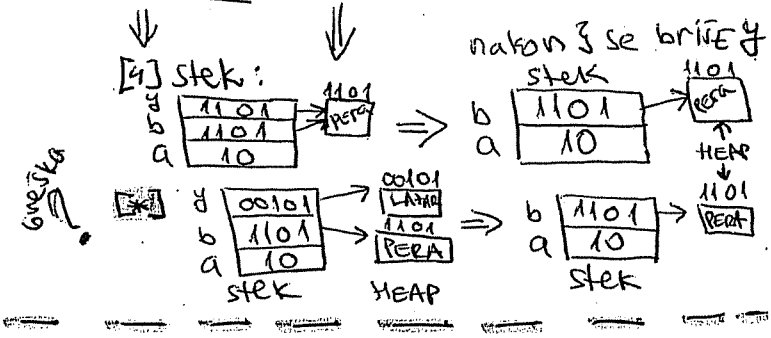
[2] Stek:



[3] stek:



[4] stek:



- **PRENOS PREKO REFERENCE:**

```
Public void Dodaj(ref int x)
                  out
{ x++; ← NE KOMPAJLIRA SE! IZBACUJE ERROR
}
Public void PromeniIme(ref Osoba y)
                       out
{ y.Ime = "Pera";
  y = new Osoba("Pera");
}
```

MAIN METODA:

```
Dodaj(10);
int a = 10;
```
KOMPAJLIRA SE →
```
Dodaj(ref a);
Osoba b = new Osoba("Lazar");
                out
PromeniIme(ref b);
           out
```

```
class Program
{
    public static void Metoda(int x, ref int y, out int z, int p)
    {
        x = 1;
        y = 2;
        z = 3;
        p = 4;
    }
    static void Main(string[] args)
    {
        int a = 0;
        int b = 0;
        int c = 0;
        int d = 0;
        Program.Metoda(a, ref b, out c, d);
        Console.WriteLine("{0}, {1}, {2}, {3}", a, b, c, d); // 0,2,3,0
    }
}
```

[1] stek: d, c, b, a = 0

[2] p, x, d, (z) c, (y) b, a = 0 — U p i x se kopiraju vrednosti iz a i d

[3] p = 4, x = 1, d = 0, (z) c = 3, (y) b = 2, a = 0 — Privremene su

[4] d = 0, c = 3, b = 2, a = 0

---

ref:
- vrednost je već postavljena
- metoda može da je pročita/izmeni
- mora biti inicijalizovan pre upotrebe

out:
- vrednost nije postavljena i ne može se pročitati metodom pre nego što se postavi
- ne mora biti inicijalizovan, prethodne vrednosti se ignorišu
- metoda mora da podesi vrednost

1,2 EKVIVALENTNI – koji pokrivaju samo potpuni skup

1) cc(de)*dd

cc dd ✓
ccddd ✗
ccdedd ✓
ccdededd ✓
ccdeddd ✗

2) R* = RR* ✗
R+ = RR* ✓
a|b|c = [a-c] ✓
a b|c = a(b|c) ✗
a|b* = (a|b)*

* potpis metoda uvek će biti

* ZADACI GRAMATIKE

- biće sve na zaokruživanje...

Number → Sign List

Sign → +
| −

List → List Bit
| Bit

Bit → 0
| 1

Imenovani / opcioni parametri:

DODELJIVANJE može biti:
1° pozicionо
2° preko imena

```
public void Metoda (int x, int y, string z)
 {...}
```

1° `Metoda (5, 10, "aaa");`
2° `Metoda (y:10, X:5, z:"aaa");`

```
public void Metoda (int x, inty , string z="nesto")
  {...}
```

opcioni parametar

3° `Metoda (5,10);`
4° `Metoda (y:10, X:5);`
`//može i 1° i 2°`

```
public void Metoda (int x, int y=0, string z)
  {...}
```

`Metoda (5, , "aaa");` Greška!
`Metoda (5, z:"aaa");` ✓
mora preko imena

Promenljivi broj parametara:

```
Public void Metoda (params int[] brojevi)
  {...}
```

params pakuje u niz

```
Metoda (1,5,10);
Metoda (5,10,20,25,30);
Metoda ();*
```

```
int[niz]=new int[10];
;
Metoda(niz);
```

*MOŽE i npr.:
(int prvi, params int[] brojevi)
tada se pri pozivu metode mora dodeliti broj (bar jedan). Ali ne bi radilo*

** na ovaj način funkcioniše npr.
`s=string.Format("{0},...",a,b,c,d,e);`
string tekst — Params string[]

```
Public int/void Metoda (int broj)
{
    broj++;   VRAĆA INT
    return broj; ← prekida celu metodu
    Console.WriteLine("..."); ← NE IZVRŠAVA SE
}
```

SKRAĆENICE:
ct + tab tab → KONSTRUKTOR
ctrl + re → GET/SET METODE
cw + tab tab → Console.WriteLine("");
try + tab tab → try/catch/finaly blok

```
Public void Metoda (int broj)
{
    broj++;
    if (broj==6)
        return; ← može void zato što ne vraća neku očekivanu vrednost
    else
        Console.WriteLine(broj);
}
```

```
Public int Metoda (int broj)
{
    broj++;
    if (broj==6)
        return 0;
    else
        Console.WriteLine(broj); ↯ Gvetka, mora da vraća int
}
* return broj; ← MOŽE OVAKO
```

main metoda:

```
int a=6;
Metoda(a);
//ako želimo da sačuvamo
izmene, pišemo:
a=Metoda(a);
```
↑ Preskočeno kreiranje objekta...

Overloading – više metoda sa istim imenom, a različitim potpisom.

Kada pozivamo metodu i Main metodi, treba da se pravi razlika.

```
Public void Metoda(int a, string b){...}   Metoda(5,"aa");
Public void Metoda(int b, string a){...}   ↯ Metoda(5,"aa");
Public void Metoda(string a, int b){...}   ↙ Metoda("aa",5);
Public void Metoda(int a, int b){...}      ↙ Metoda(5,5);
Public string Metoda(int a, string b){...} ↯ Metoda(5,"aa");
Private void Metoda(int a, string b){...}  ↯ Metoda(5,"aa");
* Public void Metoda(ref int a, string b){...} ↯ Metoda(5,"aa");
* Public void Metoda(out int a, string b){...} ↙ Metoda(ref a,"aa");
                                               ↙ Metoda(out a,"aa");
** ne mogu da stoje zajedno
   zato što su i ref i out prenos
   preko reference (u IL se svodi
   na isto);   Intermediate Language
```

GLEDA SE NATIV SAMO METODE KAO I TIP I BROJ PARAMETARA. OSTALO JE LEBITNO ZA OVERLOAD...

```
Public void Metoda(int x){...}
Public void Metoda(Long y){...}
int a=5;
Metoda(a); ← zove prvu

Long b=10;
Metoda(b); ← zove drugu

short c=3;
Metoda(c);
↓ po default-u se prevodi u int-32 i zove se prva

Public void Metoda(int x, string z="nesto"){}
Public void Metoda(int y){...}
Metoda(10); ← zove onu koja ima manji predradnji ⇒ drugu

Metoda((Long)c); ← zove se druga, kastujemo
Metoda(10); ← zove prvu

Metoda(y:c); ← zove drugu
```

DEFINISATI KLASU RAZLOMAK KOJA IMA I METODU SABERI KOJA PRIMA DVA RAZLOMKA I VRAĆA NJIHOV ZBIR, TAKOĐE U VIDU razlomka.

*ne obraćamo pažnju na deljenje sa nulom...

*Referenca(Eguals prenosi adresu)

```
class Razlomak
{
  Public int brojilac;
  Public int imenilac;
  Public Razlomak (int brojilac, int imenilac)
   { this.brojilac = brojilac;
     this.imenilac = imenilac;
   }
→ Public static Razlomak Saberi (Razlomak a, Razlomak b)
   { return new Razlomak (a.brojilac* b.imenilac+b.brojilac* a.imenilac, a.imenilac*b.im);
   }
   Public static void Main (String[] args)
   { Razlomak R1 = new Razlomak (3,5);
     Razlomak R2 = new Razlomak (2,7);
→    Razlomak R3 = Saberi (R1, R2);
     Console.WriteLine ("{0}/{1}", R3.brojilac, R3.imenilac); // 31/35 ← ispisuje se
   }
}
```

b.imenilac

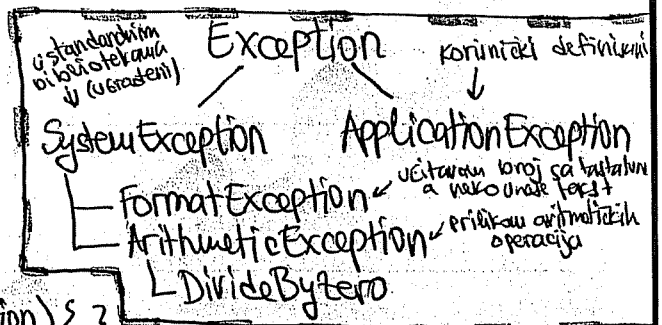
* $\frac{a}{b} + \frac{c}{d} = \frac{a \cdot d + c \cdot b}{b \cdot d}$

```
→ Public static Razlomak operator + (-11-) {-11-}
   ⋮
→ Razlomak R3 = R1 + R2;
```

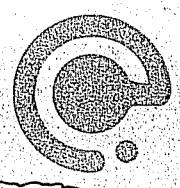
OBRADA GREŠAKA PRILIKOM IZVRŠAVANJA PROGRAMA (uokviriujemo potencijalno „opasan" deo koda):

```
try
{
  int c = Convert.ToInt32 (Console.ReadLine());
}
catch ← opšti exception, najširije (Exception)
{
}
finally ← kod se izvršava u svakom slučaju ako se uđe u try{}
{
}
```

ako ubacis slovo npr.



catch (FormatException) {...}
catch (DivideByZero) {...}
← Pišemo exceptione redom, od najkonkretnijih do najopštijih;



CROZ

Learn

* TAMO GDE nastane greška, ta LINIJA se VIŠE ne izvršava;

2. { Console.ForegroundColor = ConsoleColor.Red;

internal - default u okviru namespace-a;
Private - default u okviru klase;

toString je virtuelna metoda iz System.Object
\ ← želimo da se ispiše nešto (u CWL)

• DELEGATI:

- Tip podataka, referentni je ali ne referencira podatke već metode!
- DEFINIŠEMO ga unutar namespace-a (može i unutar klase) pa mu je zato default modifikator pristupa internal zbog čega postavljamo public najčešće;

```
Public delegate string MojDelegat (int a, int b);
```
← metoda koju referencira mora da ima odgovarajuće parametre (int, int) kao i odg. povratnu vrednost (string);

```
Public String A (int x, int y)
{ ...
}
```
(Public) — ne mora public ali paziti da l' se vidi tamo gde se poziva
može da stoji static, tada radimo preko naziva klase, a ne preko instance: d=Student.A;

* postoje i B, C, D metode...
* internal - ako su u istom assembly-ju onda radi...

```
MojDelegat d;
d=A;
d+=B;
d+=C;
```
} A B C } izvršava se i A i B i C. Ali se vraća samo ono što se desi u C

```
// d=D;  → ~~A B~~ C → D
d(5,10);
```
↑↑ obaveštava sve metode (A,B,C)

```
d=A;
d+=B;
d+=C;
d+=B;
d(5,10);   // A B C B
d-=A;      // ~~A~~ B C B
d-=B;      // B C ~~B~~
d(5,10);   // B C
d-=B;      // ~~B~~ C
d-=C;      // ~~C~~
d(5,10);   // Nije greška ništa ne priše
d-=D;      // Nije greška iako nismo dodeljivali D, ali mora ta metoda da bude definisana (kao i A,B,C)
```
← oba se izvrše (d+=B)
← briše se POSLEDNJE B

} Greška bi bila u slučaju da je null } primer: MojDelegat d; d(5,10); → kompajler javlja grešku!
← ipak jeste greška, BACA EXCEPTION!

• Događaji:

```
Public delegate int Delegat();
class Racun
{ string vlasnik;
  double stanje;
  public void Isplata(double iznos)
  { if(stanje >= iznos)
    { stanje -= iznos;
      Obavesti();
    }
  }
  Public Delegat IzvrsenaIsplata;
}
```
može event, ako to stavimo mora da bude void i mora "+=" ili "-=", ne može "="!

```
Public void Obavesti ()
{ if (IzvrsenaIsplata != null)
    IzvrsenaIsplata();
}
```
(*) ako bi bilo null, kompajler bi javio grešku pri pozivu u Main metodi;

```
class Program
{
  ... Main ...
  {
    Racun r1 = new Racun();
    r1.IzvrsenaIsplata = A;
  }
}
```

Potencijalna pitanja:

- Koje metode mogu u delegat?
  → isti tip parametara, ista povratna vrednost;
- Praćenje koda...