Visoka tehnička škola Niš

Studijski program:

Savremene računarske tehnologije

Internet programiranje

(5)

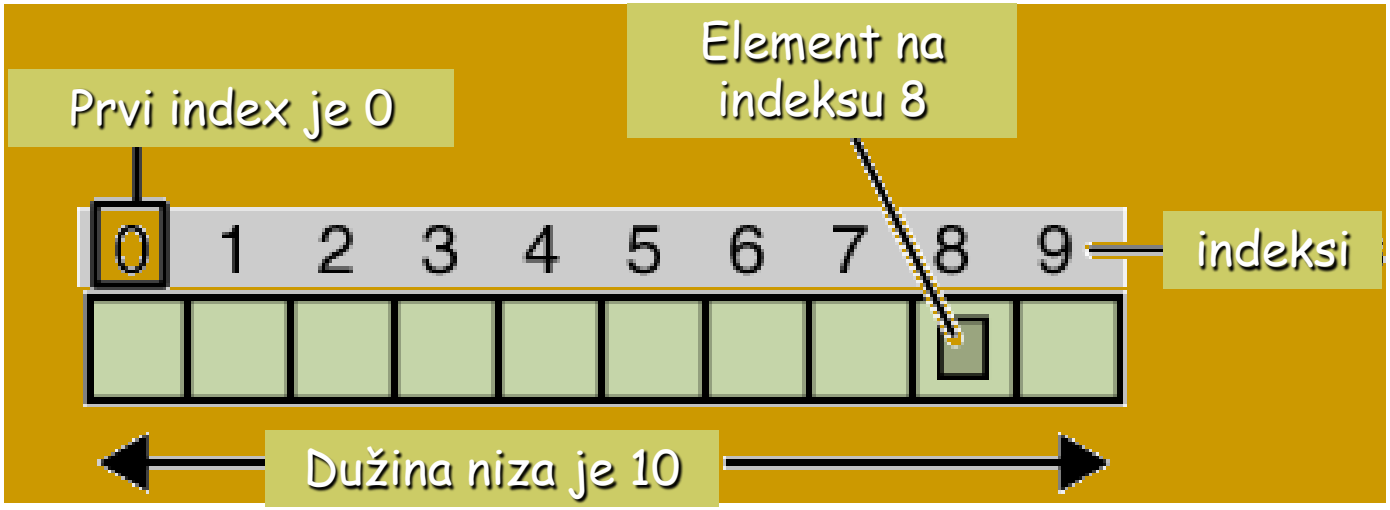
Nizovi, stringovi, operatori i upravljačke naredbe Jave

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

Novembar, 2018.

Nizovi u Javi (1)

- Kada je potrebno raditi sa **SKUPOVIMA ISTIH TIPOVA**, u Javi (kao i u C#, C++, …) se koriste **NIZOVI** (engl. arrays).
- Prema definiciji, **NIZ** je **IMENOVANI OBJEKAT** koji predstavlja skup promenljivih **ISTOG TIPA**.
- Za pristup konkretnim elementima skupa koristi se **INTEGER PROMENLJIVA** koja se naziva **INDEKS** (engl. index).



Nizovi u Javi (2)

- Promenljive **TIPA NIZA** poseduje **DVA** entiteta:
  1. **REFERENCU na objekat tipa niza;**
  2. **Sam OBJEKT – NIZ.**
- Promenljive tipa **NIZ**-a se mogu deklarisati na sledeća **DVA** ekvivalentna načina:
  1. **int[] primes;**
  2. **int primes[];**
- Promenljiva **primes** čuva prostor za **CELOBROJNI NIZ** koji **tek treba kreirati**!
- Za formiranje niza od **10 int vrednosti** sa referencom pod nazivom **primes** koristi se sledeći programski iskaz:

```
primes = new int[10]
```

Nizovi u Javi (3)

- Ključna reč operator **new** govori o **DODELI MEMORIJE**, a **int[10]** da je alocirana (rezervisana) memorija za **10 int** promenljivih – članova niza.
- Obzirom na činjenicu da promenljiva tipa **int** zauzima **4 bajta**, to znači da je za sam niz rezervisano **40 bajtova**, uz dodatak još **4 bajta** za promenljivu **primes** u kojoj se nalazi **referenca na niz**.
- Svi **ELEMENTI NIZA** se iniciraju na **PODRAZUMEVANU VREDNOST** i to za:
  - **Numeričke** vrednosti **na 0;**
  - **Bulove** promenljive **na false;**
  - **Char** promenljive **na '\u0000'.**
- **DUŽINA NIZA** može se dobiti pozivom na **SVOJSTVO length** koje je **pridruženo** ovom objektu.

Nizovi u Javi (4)

- **DEKLERACIJA** i **INICIJALIZACIJA** niza se može obaviti na sledeći način:

```
int[] primes = {2, 3, 5, 7, 11, 13, 17};
```

- Svojstvo **length** je određeno **BROJEM** inicijalizovanih članova.
- Navedene vrednosti se elementima niza dodeljuju redom:

```
primes[0]=2, primes[1]=3, ... , primes[6]=17.
```

- Primer dekleracije niza **samples** i njegovog **popunjavanja** slučajnim verdnostima.

...

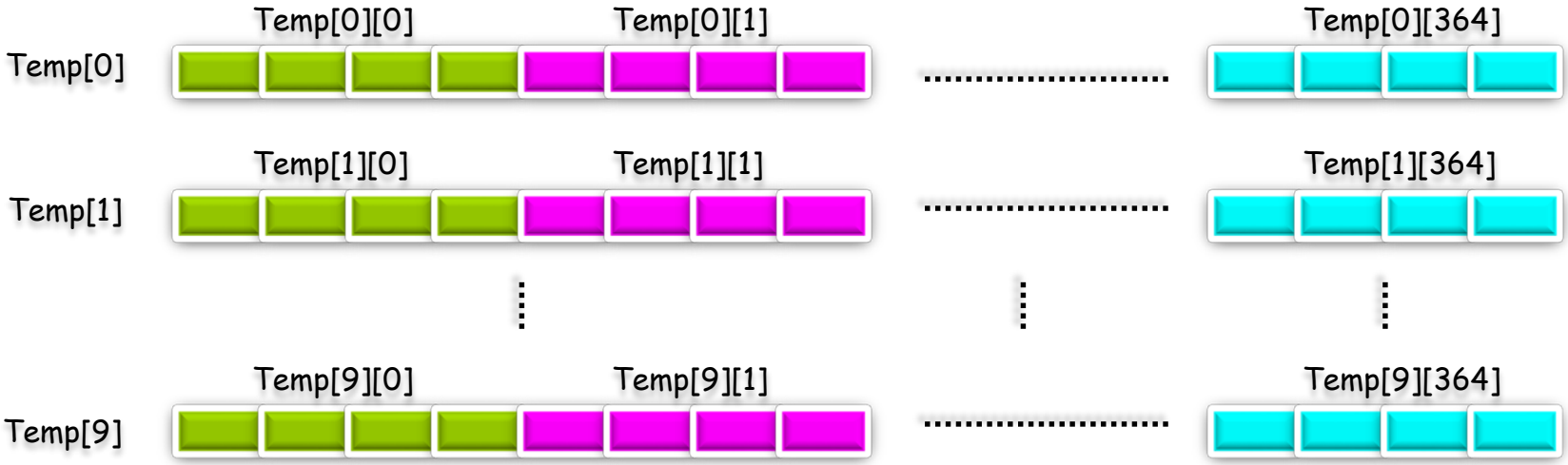
```
double[] samples = new double[50];          // niz od 50 double vr.
for(int i = 0; i < samples.length; i++) {
  samples[i] = 100.0*Math.random();         // Generisanje slučajnih vr.
}
```

Klasa Math, random - metoda!

Nizovi nizova u Javi

- Pored **JEDNODIMENZIONIH NIZOVA** (postoji samo **jedan indeks** za pristup elementima), mogu se formirati i **VIŠEDIMENZIONI NIZOVI** (pa tako i dvodimenzioni nizovi).
- Kod **2D nizova** formira se **MATRICA ELEMENATA** čija je pozicija određena **indeksima** koji se odnose na **HORIZONTALNU** i **VERTIKALNU** lokaciju.

```
float[][] Temp = new float[10][365];
```



Nizovi znakova u Javi

- Java može rezervisati prostor za **nizove karaktera** (znakova):

```
char[] message = new char[50];
```

- Deklaracija i inicijalizacija **znakovnog niza**:

```
char[] samoglasnici = {'a','e','i','o','u'};
```

- Inicijalizacija **znakovnog litetala**:

```
System.out.println("This is \n a string constant!");
```

- Zapravo radi se o **objektu** klase **String**, što znači da su mu pridružene **SPECIFIČNE METODE** za rad sa stringovima.
- Objekti tipa **String** se formiraju na sledeći način:

```
String myString = "My inaugural string";
```

- **NIZOVI ZNAKOVNIH NIZOVA** se mogu formirati na sledeći način:

```
String[] names = new String[5];
```

- Niz stringova se primenjuje kod metode **main**:

```
public static void main(String[] args)
```

Stringovi u Javi

- Java podržava **ZNAKOVNE NIZOVE** svojom klasom **String**.
- **STRING JE OBJEKT**, što podrazumeva da su mu pridružene **METODE** za rad sa ovim tipom promenljive.
- Mogu se definisati i **NIZOVI ZNAKOVNIH NIZOVA** (setite se dekleracije metode **main**, koja ima kao parametar **niz znakovnih nizova**:

```
main(String args[]).
```

- Objekt tipa string se može koristiti kao **ARGUMENT** metode **println()**.

```
String str = "Inicijalizacija string promenljive";

System.out.println(str);
```

- Objekti tipa string poseduju **SPECIFIČNE OSOBINE** i **ATRIBUTE** koje olakšavaju njihovo korišćenje.
- Veći deo standardnih klasa i metoda za obradu stringova je obrađen u C#-u.

Metode za rad sa Stringovima

- Java dozvoljava definisanje **NIZA OBJEKATA** tipa **String**, npr.:

```
String[ ] colors = {"red", "orange", "yellow", "green", "blue", "indigo", "violet"};
```

- Veliki broj operacija se mogu izvesti nad stringovima, a neke od nih su:
  - **SPAJANJE** znakovnih nizova;
  - **POREĐENJE** jednakosti znakovnih nizova;
  - **PROVERA** početka i kraja znakovnog niza;
  - **UREĐIVANJE** znakovnih nizova;
  - **PRISTUP** znacima znakovnih nizova;
  - **IZVLAČENJE** znakova iz znakovnog niza;
  - **TRAŽENJE** znakova u znakovnim nizovima;
  - **TRAŽENJE** znakovnih podnizova;
  - **IZVLAČENJE** znakovnih podnizova;
  - **TOKENIZOVANJE** znakovnog niza.

Spajanje i poređenje nizova u Javi

- Spajanje dva niza generisanjem **novog objekta** tipa **String** (pangrama):

```
myString = "The quick brown fox" + " jumps over the lazy dog";
```

- Za **POREĐENJE** znakovnih nizova koristi se metoda **equals()** pridružena **String** promenljivama:


```
System.out.println("string1.equals(string3) is true. " +
" so strings are equal.");
```

- Poziva se metod **equals()** za obekat tipa **String** koji referencira promenljivu **string1** i kao argument predaje refrencu na **string3**.
- Metod **equals()** poredi stringove (razlikuje velika i mala slova) i vraća **TRUE** ako su stringovi jednaki, odnosno **FALSE** u suprotnom slučaju.

Poređenje nizova u Javi (1)

- Metodom **startWith()** objekata tipa **String** se proverava da li znakovni niz **POČINJE** određenim nizom znakova.

```
String string1 = "Too many cooks";

    if(string1.startsWith ("Too") ) {

        System.out.println("The string does start with \ "Too\" too! ");

}
```

- Metoda **compareTo()** poredi **DUŽINU NIZOVA** i vraća:
  - **NEGATIVNU CELU VREDNOST**, ako je objekat **MANJI** od prosleđenog;
  - **0**, ako su **JEDNAKI**
  - **POZITIVNU CELU VREDNOST**, ako je objekat **VEĆI** od argumenta.
- Znak se određuje na osnovi numeričke vrednostu **unicode** znakova.

Pretraživanje nizova u Javi

- Pristupanje **POJEDINIM ZNAKOVIMA** u nizu vrši se prema **INDEKSU tipa int** koja predstavlja **pomeraj pozicije** znaka u odnosu na početak znakovnog niza.
- Metodom **charAt()** dobijaju se **POJEDINI ZNACI** iz objekta **String**.
- **ZA PRETRAŽIVANJE** znakovnih nizova koriste se metode **indexOf()** i **lastIndexOf()**.
- Ove metode imaju nekoliko pojavnih oblika, tako da se za **TRAŽENJE ZNAKA 'a'** u znakovnom nizu **text** primenjuje:

```
index = text.indexOf('a');

index = text.lastIndexOf('a');

index = text.indexOf('a', startIndex);
```

- Ove metode takođe imaju mogućnost **TRAŽENJE ZNAKOVNOG NIZA**:

```
indexOf(String str);
```

Izvlačenje znakovnih nizova

- Metod **substring()** **IZVLAČI PODNIZ** iz nekog znakovnog niza.
- Postoje **dve verzije**, jedna izvlači podniz **OD ZADATE VREDNOSTI**, a druga izvlači podniz **IZ ZADATOG OPSEGA**:

```
String place = "Palm Springs";

String lastWord = place.substring(5);

String segment = place.substring(7, 11);
```

- Metoda **split()** klase **String** koristi se za razbijanje **ZNAKOVA NA PODNIZOVE – TOKENE**, odvojene **separatorima**.
- Ova metoda vraća sve **TOKENE** nekog niza u vidu **niza objekata** tipa **String**.

```
String text = "to be or not to be, that is the question.";

String[] words = text.split("[, .]", 0);          // Separatori su:',', ' ' ili '.'
```

Promenljivi znakovni nizovi

- Objekti tipa **String** se **NE MOGU MENJATI**, ali se mogu izvršiti **modifikacije** i **kombinacije** postojećih objekata tipa **String**.
- Za rad sa znakovnim nizovima koji se **MOGU MENJATI** koriste se **DVE** klase:
  - **StringBuffer** (bezbedna za rad sa nitima) i
  - **StringBuilder** (nebezbedna za rad sa nitima).

```
StringBuffer aString = new StringBuffer("A stitch in time");
```

- **OBJEKTIMA** tipa **StringBuffer** je pridružen **MEMORIJSKI BLOK** koji se u žargonu programiranja naziva **BAFER**.
- **Dužina niza** u ovom objektu može da bude **različita** od dužine samog bafera.
- **KAPACITRT BAFERA** se **automatski povećava** prilikom dodavanja znakova.
- Kapacitet bafera se može dobiti na sledeći način:

```
int theCapacity = aString.capacity();
```

Promenljivi znakovni nizovi

- Metodom **append()** se na **KRAJ** postojećeg znakovnog niza tipa **StringBuffer** dodaje nov znakovni niz.

```
StringBuffer aString = new StringBuffer("A stitch in time");
aString.append("saves nine");
```

(engl. poslovica)

- Metod **append()** vraća referencu na **PROŠIRENI OBJEKT**.
- Postoje **više** korisnih verzija ove metode.
- Metodom **lastIndexOf()** se pretražuje objekt **StringBuffer** za određeni znakovni podniz.

```
StringBuffer phrase = new StringBuffer("one two three four";
int position = phrase.lastIndexOf("three");
```

- **ZAMENA PODNIZA** se obavlja metodom **replace()**.
- **UMETANJE** znakovnog niza koristi se metoda **insert()**.
- Izvlačenje znakova se obavlja metodama: **chatAt()** i **getChars()** slično kao kod **String** promenljivih.

Aritmetički operatori (1)

- **ARITHMETIČKI OPERATORI** su **SIMBOLI** koji obavljaju neku **aritmetičku operaciju**.
- U aritmetičkom izrazu mogu se **ISTOVREMENO** koristiti **VIŠE OPERATORA**.
- Ako se koriste više operatora u aritmetičkom izrazu, mora se voditi računa o **UNAPRED DEFINISANOM** specifičnom **PRIORITETU** (**redosledu**) njihove primene.
- **OPERATORI SA VEĆIM PRIORITETOM** se **PRVO** primenjuju!
- **UGRAĐENI PRIORITETI** aritmetičkih operatora je dat na sledećem slajdu.

Prioritet aritmetičkih operatora

Operator	Značenje	Prioritet
-	Unarni minus	Najviši
+	Unarni plus	Najviši
*	Množenje	Srednji
/	Deljenje	Srednji
%	Ostatak	Srednji
+	Sabiranje	Najniži
-	Oduzimanja	Najniži

- **PREDEFINISAN PRIORITET** operatora se može promeniti **ZAGRADAMA**.

Kombinovanje operatora

Operator	Operacija	Primer	Efekt
=	Dodeljivanje	Sum=5;	Sum=5;
+=	Sabiranje i dodeljivanje	Sum += 5;	Sum=Sum+5;
-=	Oduzimanje i dodeljivanje	Sum -= 5;	Sum=Sum-5;
*=	Množenje i dodeljivanje	Sum *= 5;	Sum=Sum*5;
/=	Deljenje i dodeljivanje	Sum /= 5;	Sum=Sum/5;

- Više aritmetičkih operatora se može **KOMBINOVATI** i zapisati na specifičan – skraćeni način.

Aritmetički operatori (2a)

```
// Prikaz osnovnih aritmetičkih operatora.
class BasicMath {
  public static void main(String args[]) {
    // arithmetika sa celim brojevima
    System.out.println("Integer Arithmetic");
    int a = 1 + 1;
    int b = a * 3;
    int c = b / 4;
    int d = c - a;
    int e = -d;
    System.out.println("a = " + a);
    System.out.println("b = " + b);
    System.out.println("c = " + c);
    System.out.println("d = " + d);
    System.out.println("e = " + e);
```

```
IZLAZ:
a=2
b=6
c=1
d=-1
e=1
```

Aritmetički operatori (2b)

```
// arithmetika koja koristi promenljive double
   System.out.println("\nFloating Point Arithmetic");
   double da = 1 + 1;
   double db = da * 3;
   double dc = db / 4;
   double dd = dc - a;
   double de = -dd;
   System.out.println("da = " + da);
   System.out.println("db = " + db);
   System.out.println("dc = " + dc);
   System.out.println("dd = " + dd);
   System.out.println("de = " + de);
  }
}
```

```
IZLAZ:
da=2.0
db=6.0
dc=1.5
de=-0.5
e=0.5
```

Aritmetički operatori (2c)

```
// Prikaz nekoliko operatora označavanja.

class OpEquals {
  public static void main(String args[]) {
    int a = 1;
    int b = 2;
    int c = 3;
    a += 5;
    b *= 4;
    c += a * b;
    c %= 6;
    System.out.println("a = " + a);
    System.out.println("b = " + b);
    System.out.println("c = " + c);
  }
}
```

```
IZLAZ:
a=6
b=8
c=?
```

Aritmetički operatori (2d)

```
// Demonstriranje ++ i --.
class IncDec {
  public static void main(String args[]) {
    int a = 1;
    int b = 2;
    int c;
    int d;
    c = ++b;
    d = a++;
    c++;
    System.out.println("a = " + a);
    System.out.println("b = " + b);
    System.out.println("c = " + c);
    System.out.println("d = " + d);
  }
}
```

```
IZLAZ:
a=2
b=3
c=4
d=1
```

Logički operatori (1)

- **LOGIČKI OPERATORI** kombinuju logičke vrednosti **true** (istina) i **false** (laž) u **JEDNU VREDNOST** true i false.
- Postoje **ŠEST** osnovnih logičkih operatora.
- Moguće su **KOMBINACIJE** logičkih operatora i operatora dodeljivanja.

Logički operatori (2)

Operator	Rezultat
&	Logička konjunkcija
|	Logička disjunkcija
^	Isključiva logička disjunkcija
||	Kratkospojena disjunkcija
&&	Kratskospojena konjunkcija
!	Logička negacija
&=	Dodeljivanje uz logičku disjunkciju
|=	Dodeljivanje uz logičku konjunkciju
^=	Dodeljivanje ul logičku isključivu disjunkciju
==	Jednako
!=	Različito
?:	Ternarni (trojni) opertaor

Logički operatori (2a)

```
// Prikaz bulovih operatora.
class BoolLogic {
  public static void main(String args[]) {
    boolean a = true;
    boolean b = false;
    boolean c = a | b;
    boolean d = a & b;
    boolean e = a ^ b;
    boolean f = (!a & b) | (a & !b);
    boolean g = !a;
    System.out.println("       a = " + a);
    System.out.println("       b = " + b);
    System.out.println("     a|b = " + c);
    System.out.println("     a&b = " + d);
    System.out.println("     a^b = " + e);
    System.out.println("!a&b|a&!b = " + f);
    System.out.println("      !a = " + g);
  } }
```

```
IZLAZ:
a=true
b=false
a|b=true
a&b=false
a^b=true
a&b|a&b=true
!a=false
```

Ternarni (trojni) operator ?:

- Trojni (ternarni) operator može da zameni određene vrste uslovnih iskaza.
- Opšti oblik se može prikazati na sledeći način:

Vrednost = Izraz_1 ? Izraz_2 : Izraz_3

- **Izraz_1** može biti **BILO KOJI IZRAZ** čiji je **rezultat** tipa **boolean**.
- Ako **Izraz_1** ima vrednost **true**, onda se izračunava **Izraz_2**, u suprotnom **Izraz_3**.
- Rezultat operacije **?** je rezultat onog izraza koji se izračunava.

```
i = 10;
  k = i < 0 ? -i : i;          // apsolutna vrednost promenljive i
  System.out.print("Absolute value of ");
  System.out.println(i + " is " + k);
```

Operatori nad bitovima

Operator	Rezultat
~	Negacija nad bitovima
&	Konjunkcija nad bitovima
|	Disjunkcija nad bitovima
^	Isključiva disjunkcija
>>	Pomeranje u desno
>>>	Pomeranje u desno sa unošenjem nula
<<	Pomeranje u levo
&=	Dodeljivanje uz konjunkciju nad bitovima
|=	Dodeljivanje uz disjunkciju nad bitovima
^=	Dodeljivanje uz isključivu disjunkciju nad bitovima
>>=	Dodeljivanje uz pomeranje u desno
>>>=	Dodeljivanje uz pomeranje u desno sa unošenjem nula
<<=	Dodeljivanje uz pomeranje u levo

Operatori nad bitovima (1a)

```
// Prikaz bitskih logčkih operatora.
class BitLogic {
  public static void main(String args[]) {
    String binary[] = {
      "0000", "0001", "0010", "0011", "0100", "0101", "0110", "0111",
      "1000", "1001", "1010", "1011", "1100", "1101", "1110", "1111"
    };
    int a = 3;                  // 0 + 2 + 1 or 0011 binarno
    int b = 6;                  // 4 + 2 + 0 or 0110 binarno
    int c = a | b;
    int d = a & b;
    int e = a ^ b;
    int f = (~a & b) | (a & ~b);
    int g = ~a & 0x0f;
```

Operatori nad bitovima (1b)

```
 System.out.println ("       a = " + binary[a]);
  System.out.println ("       b = " + binary[b]);
  System.out.println ("     a|b = " + binary[c]);
  System.out.println ("     a&b = " + binary[d]);
  System.out.println ("     a^b = " + binary[e]);
  System.out.println ("~a&b|a&~b = " + binary[f]);
  System.out.println ("      ~a = " + binary[g]);
 }
}
```

```
IZLAZ:
a=0011
b=0110
a|b=0111
a&b=0010
a^b=0101
~a&b|a&~b=0101
~a=1100
```

Upravljačke naredbe

- Upravljačke naredbe se koriste za **KONTROLISANJE TOKA PROGRAMA** i njegovo **GRANANJE** na osnovu stanja promenljivih.
- Postoje tri kategorije ovih naredbi:
  - **naredbe uslovljavanja,**
  - **naredbe ciklusa** i
  - **naredbe skoka**.
- Obradićemo sledeće naredbe:
  - **if**
  - **switch**
  - **for**
  - **while**
  - **do while**
  - **break**
  - **continue** i
  - **return**.

If naredba

```
int a, b;
if(a < b)
    a = 0;
else b = 0;

boolean dataAvailable;
if (dataAvailable)
    ProcessData();
else
    waitForMoreData();

int bytesAvailable;
if (bytesAvailable > 0) {
    ProcessData();
    bytesAvailable -= n;
} else
    waitForMoreData();
```

Logički operator **MANJE**

Logički operator **VEĆE**

Kombinovanje operatora

If-else-if naredba

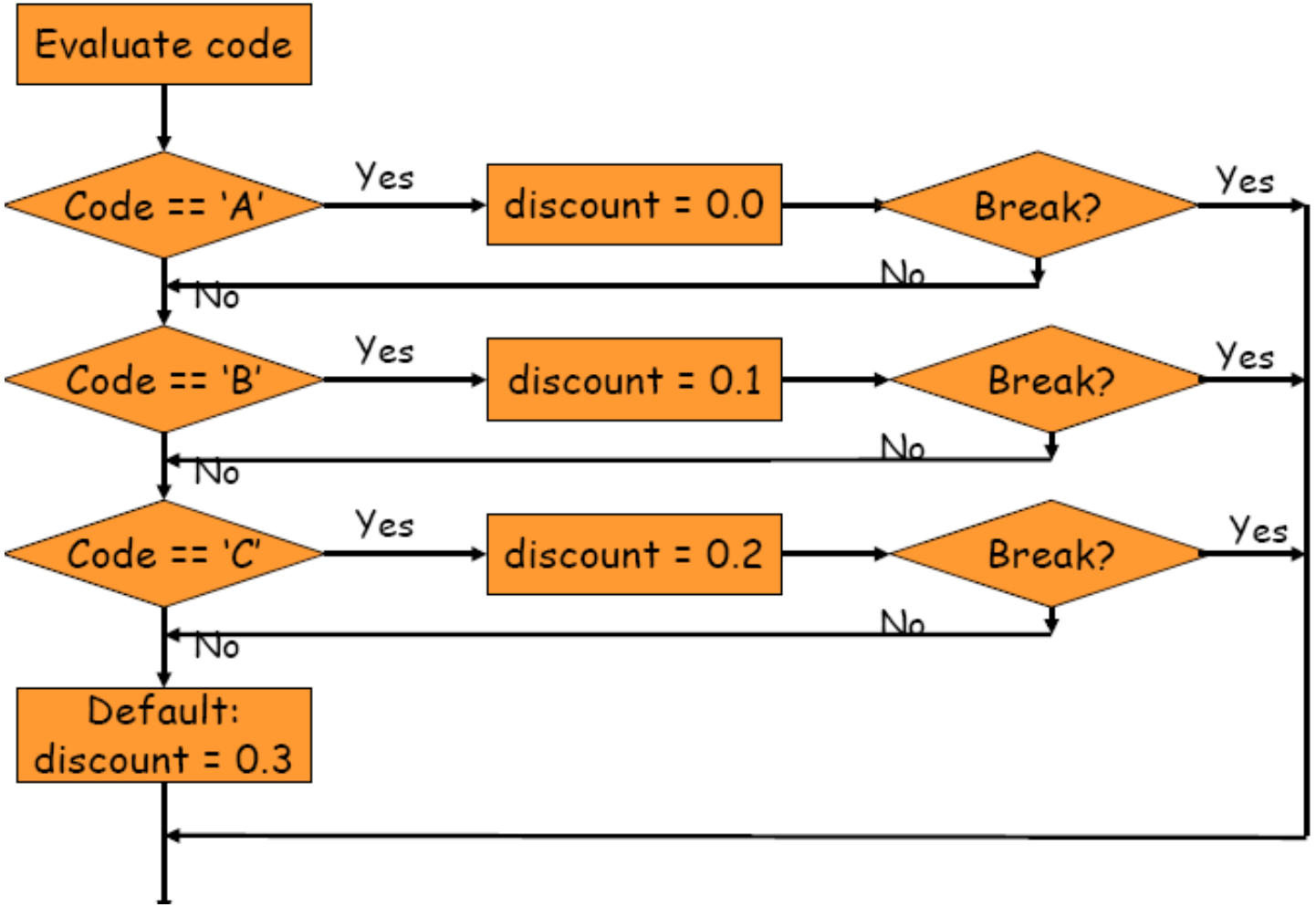
```
// Demonstracija if-else-if naredbe.
class IfElse {
  public static void main(String args[]) {
    int mesec = 4;     // April
    String sezona;
    if(mesec == 12 || mesec == 1 || mesec == 2)
      sezona = "Zima";
    else if(mesec == 3 || mesec == 4 || mesec == 5)
      sezona = "Proleće";
    else if(mesec == 6 || mesec == 7 || mesec == 8)
      sezona = "Leto";
    else if(mesec == 9 || mesec == 10 || mesec == 11)
      season = "Jesen";
    else
      sezona = "Lažni mesec !";

    System.out.println("April je u " + sezona + ".");
  }
}
```

Logički **ILI** operatori

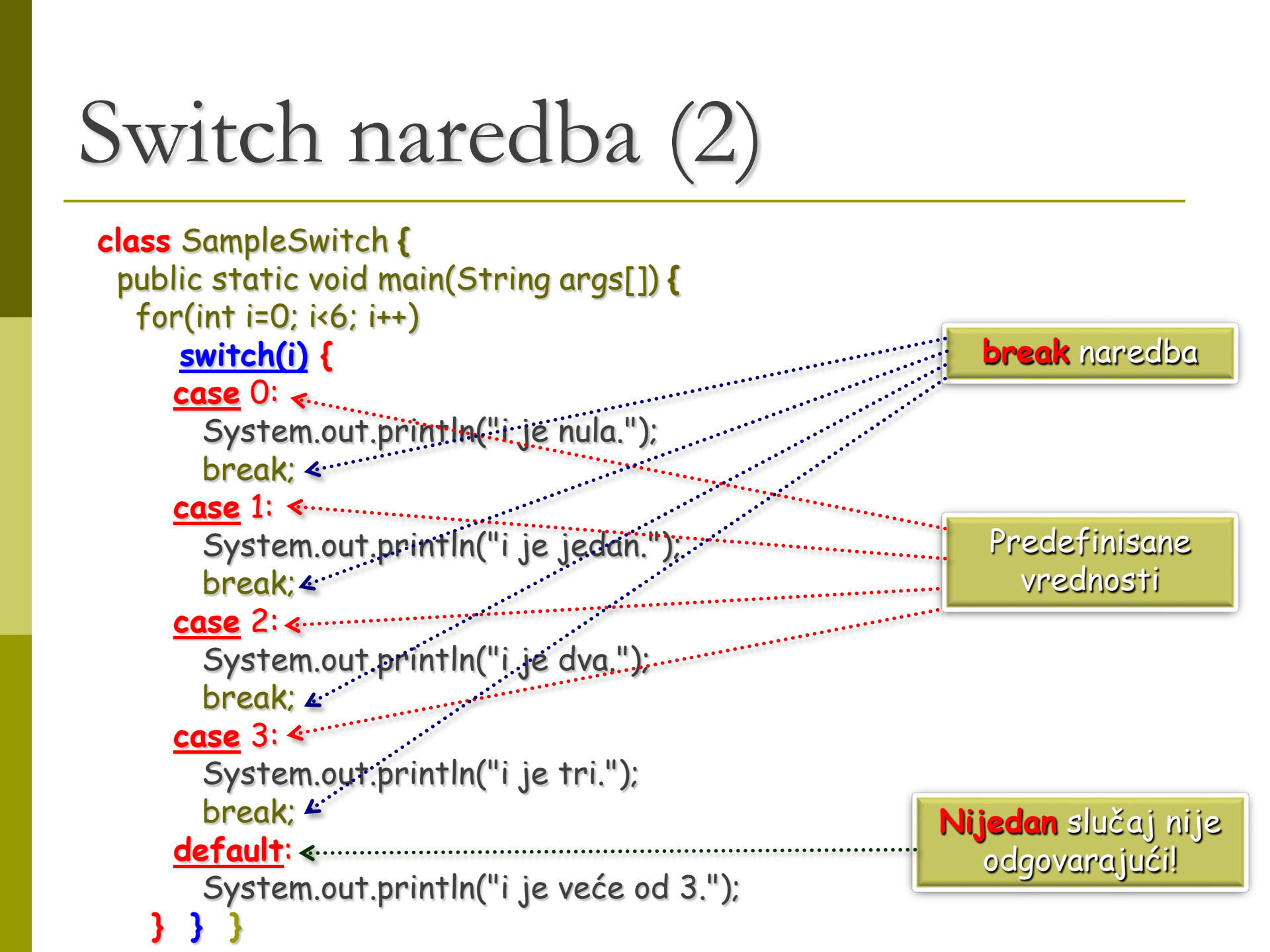
Logički operatori **JEDNAKO**

Switch naredba (1)



Switch naredba (2)

```
class SampleSwitch {
  public static void main(String args[]) {
    for(int i=0; i<6; i++)
      switch(i) {
        case 0:
          System.out.println("i je nula.");
          break;
        case 1:
          System.out.println("i je jedan.");
          break;
        case 2:
          System.out.println("i je dva.");
          break;
        case 3:
          System.out.println("i je tri.");
          break;
        default:
          System.out.println("i je veće od 3.");
  } } }
```



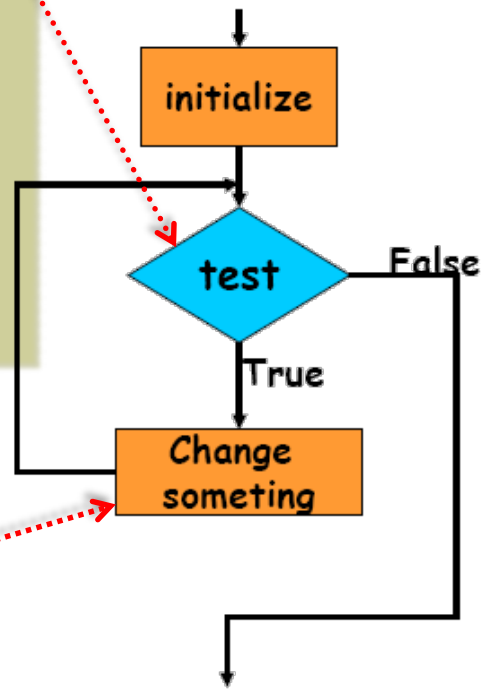
While naredba

```
class While {
  public static void main(String args[]) {
    int n = 10;

    while(n > 0) {
      System.out.println("tick " + n);
      n--;
    }
  }
}
```

Prvo se vrši ispitivanje

Pa se tek onda izvršava ovaj blok koda



Do-While naredba

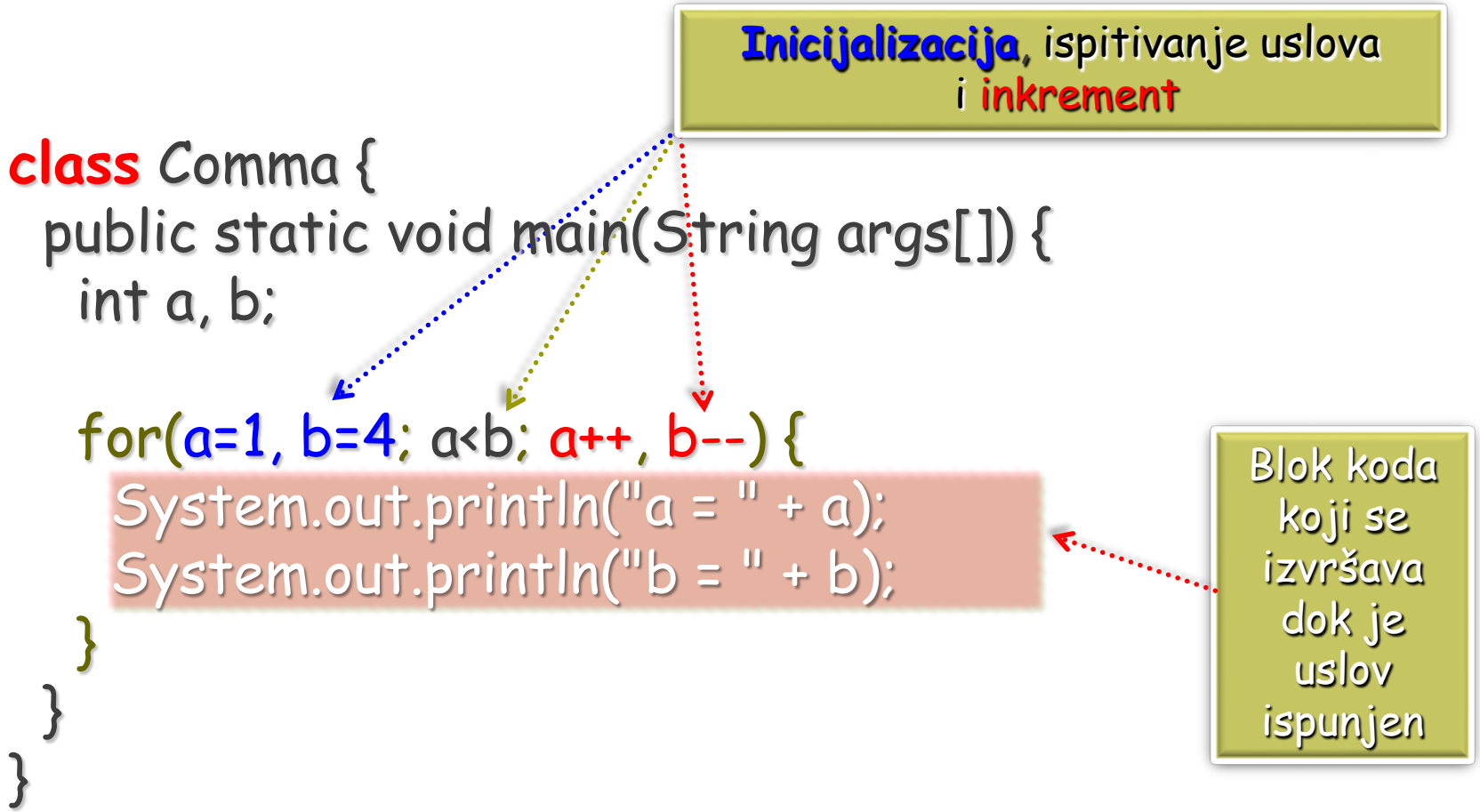
```
class DoWhile {
  public static void main(String args[]) {
    int n = 10;

    do {
      System.out.println("tick " + n);
      n--;
    } while(n > 0);
  }
}
```

Prvo se izvrši blok koda

Pa se tek onda vrši ispitivanje uslova

For naredba


Inicijalizacija, ispitivanje uslova i inkrement
class Comma {
public static void main(String args[]) {
int a, b;
for(a=1, b=4; a<b; a++, b--) {
System.out.println("a = " + a);
System.out.println("b = " + b);
}
}
}
Blok koda koji se izvršava dok je uslov ispunjen

For-each naredba

```
class ForEach {
  public static void main(String args[]) {
    int brojevi []={1,2,3,4,5,6,7,8,9,10};
    int zbir=0;

    for(int x: brojevi) {
      System.out.println(" Broj je: " + x);
          zbir += x;
    }
    System.out.println(" Zbir iznosi: " + zbir);
  }
}
```

niz: brojevi

Foreach verzija for naredbe

x uzima vrednosti iz niza: brojevi

For-each za višedim. nizove

```
class ForEach {
  public static void main(String args[]) {
    int brojevi [][]={ {1,2,3}, {4,5,6}, {7,8,9} };
    int zbir=0;

    for(int x []: brojevi) {
        for(int y: x) {
      System.out.println(" Broj je: " + x);
        zbir += x;
    }
}
    System.out.println(" Zbir iznosi: " + zbir);
  }
}
```

2D niz: brojevi

Referenca na 1D niz tipa int

y uzima vrednosti iz 1D niza x

Naredbe skoka, break

```
class BreakLoop {
  public static void main(String args[]) {
     for(int i=0; i<100; i++) {
        if(i == 10) break;
          System.out.println("i: " + i);
    }
    System.out.println("Petlja kompletirana.");
  }
}
```

Terminira petlju for ako je i jednako 10

Gde se nastavlja petlja posle prekida?

Naredbe skoka: continue (1)

```
class Continue {
  public static void main(String args[]) {
    for(int i=0; i<10; i++) {
      System.out.print(i + " ");
        if (i%2 == 0) continue;
      System.out.println("");
    }
  }
}
```

```
IZLAZ:
01
23
45
67
89
```

Preskače se ostatak petlje for ako je uslov ispunjen

Naredba skoka: continue(2)

```
class ContinueLabel {
  public static void main(String args[]) {
    outer: for (int i=0; i<10; i++) {
             for(int j=0; j<10; j++) {
               if(j > i) {
                 System.out.println();
                 continue outer;
               }
             System.out.print(" " + (i * j));
           }
         }
         System.out.println();
  }
}
```

Naredba continue definiše mesto nastavka

Naredbe skoka: return

```
class Return {
  public static void main(String args[]) {
    boolean t = true;

    System.out.println("Before the return.");

      if(t) return;     // return to caller

    System.out.println("Ovo se neće izvršiti.");
  }
}
```

t je **uvek** true!, zato je grananje if naredbe def.

Vraćanje kontrole pozivaocu

Ovo se nikad neće izvršiti!!? Zašto

Primer

- Vrednost funkcije sin (x) se može odrediti prema sledećem izrazu:

$$\sin(x) = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)!} x^{2k+1}$$

1. Odrediti vrednost funkcije sin (x) za x=π/6 u 20 koraka, a posle svakog koraka štampati dobijenu vrednost sume.
2. Koliko koraka sumiranja treba obaviti da greška bude manja od 10^{-3}.

Primer, faktorijel

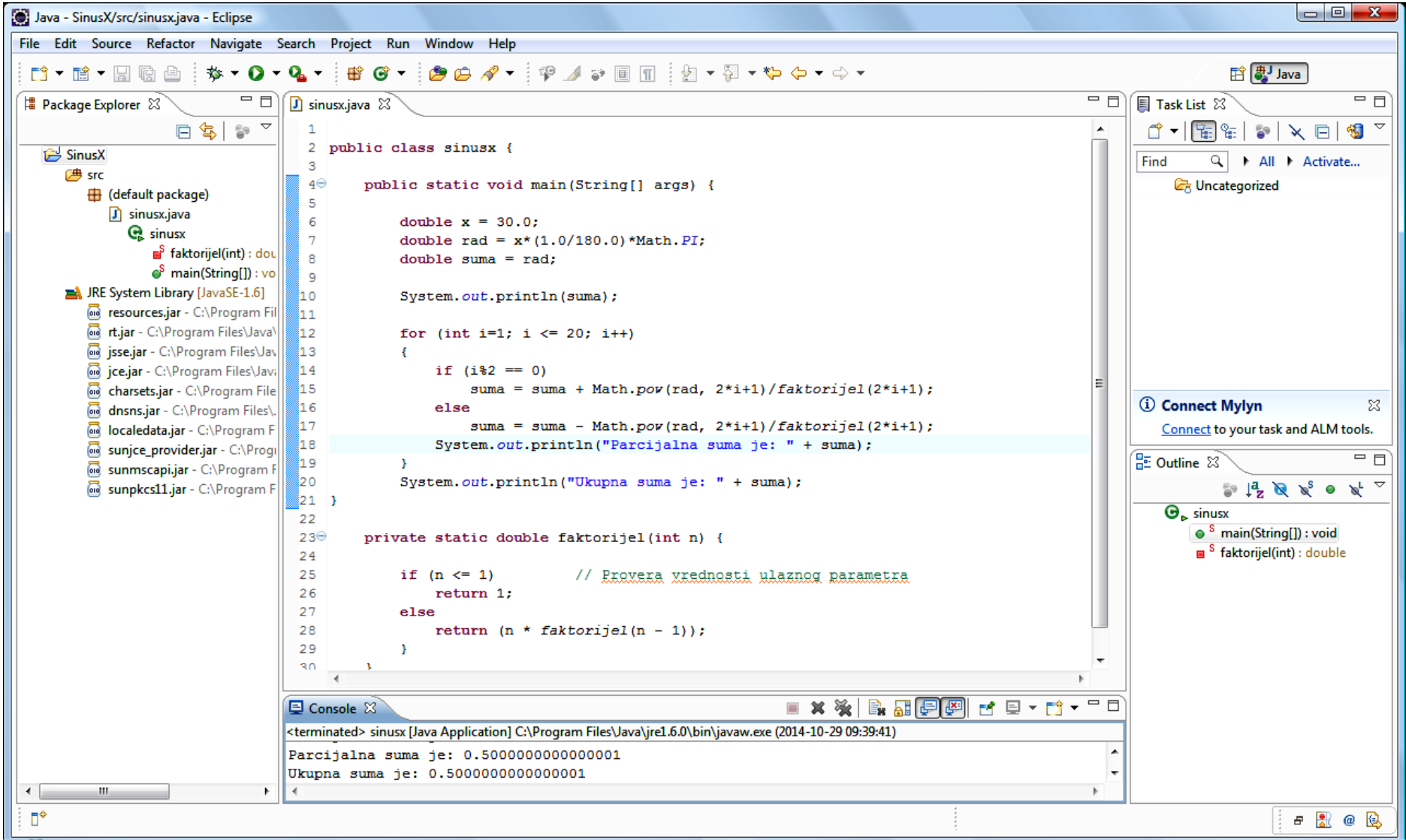
- Metoda za izračunavanje faktorijela, **faktorijel()**:

```
public static double faktorijel(int n)
   {
        if (n <= 1)              // Provera vrednosti ulaznog parametra
                return 1;
        else
                return n * faktorijel(n - 1);
   }
```

Primer, sin(x)

```
public class sinusx {
    public static void main(String[] args) {
    double x = 30.0;
    double rad = x*(1.0/180.0)*Math.PI;          // [0] -> rad
    double suma = rad;                           // prvi element sume, x1
          for (int i=1; i <= 20; i++)
           {
              if (i%2 == 0)
                   suma = suma + Math.pow(rad, 2*i+1)/faktorijel(2*i+1);
              else
                    suma = suma - Math.pow(rad, 2*i+1)/faktorijel(2*i+1);
          System.out.println("Parcijlna suma je" + suma);
           }
System.out.println(suma);
}
```

Primer u Eklips-u, sinus x



Primeri za domaći

Eksponencijalna funkcija:

$$\mathrm{e}^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots \text{ za sve } x$$

Prirodni logaritam:

$$\log(1-x) = -\sum_{n=1}^{\infty} \frac{x^n}{n} \text{ za } |x| \leq 1,\ x \neq 1$$

$$\log(1+x) = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{x^n}{n} \text{ za } |x| \leq 1,\ x \neq -1$$

Konačan geometrijski red:

$$\frac{1-x^{m+1}}{1-x} = \sum_{n=0}^{m} x^n \quad \text{za } x \neq 1 \text{i } \ m \in \mathbb{N}_0$$

Beskonačan geometrijski red:

$$\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n \text{ za } |x| < 1$$

Varijante beskonačnih geometrijskih redova:

$$\frac{x^m}{1-x} = \sum_{n=m}^{\infty} x^n \quad \text{za } |x| < 1 \text{i } \ m \in \mathbb{N}_0$$

$$\frac{x}{(1-x)^2} = \sum_{n=1}^{\infty} n x^n \quad \text{za } |x| < 1$$

Kvadratni korijen:

$$\sqrt{1+x} = \sum_{n=0}^{\infty} \frac{(-1)^n (2n)!}{(1-2n)n!^2 4^n} x^n \text{ za } |x| < 1$$

Osnova prirodnog logaritma

```
public class Prirodni_logaritam {
public static void main(String[] args) {
        double e_na_X = 0;
        double X = 1;
        int n = 0;
        double GRESKA = 0.00001;
        while ( (Math.E - e_na_X) >= GRESKA )
  {
     e_na_X = e_na_X + (Math.pow(X, n)/faktorijel(n));
     System.out.printf("%e", e_na_X);
      n++;
     System.out.printf("%d\n", n);
    }

System.out.printf("Konacno n= %d", n);
}
```

Primer u Eklips-u, e