UNIVERZITET U BEOGRADU

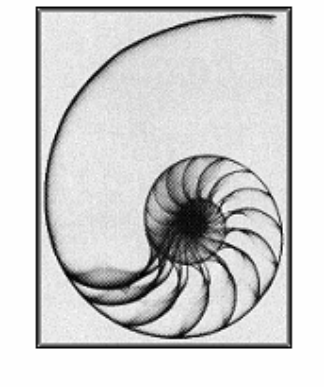
FAKULTET ORGANIZACIONIH NAUKA

(Laboratorija za softversko inženjerstvo)

OSNOVNI KONCEPTI JAVE

(Delovi predavanja i vežbi iz predmeta PROJEKTOVANJE SOFTVERA)

Autori: dr Siniša Vlajić, prof. dr Vidojko Ćirić, dipl. ing. Dušan Savić



Beograd - 2003.

Sadržaj

1. Radno okruženje Jave

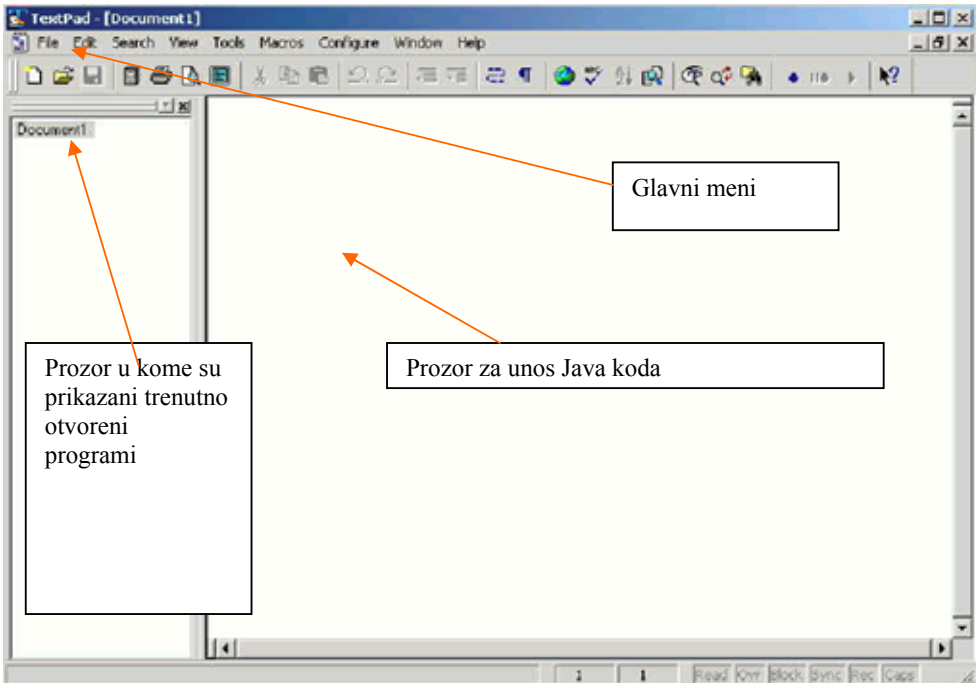
Osnovno radno okruženje koje je potrebno da bi se program napisao, preveo i izvršio u Javi sadrži:

- ✓ Java Developer Kit (JDK), koji se može pronaći na Sun-ovom sajta: www.java.sun.com;
- ✓ Editor u koji će se unositi izvorni Java programski kod (Npr. može se koristiti Microsoft-ov Notepad ili TextPad);

Pored navedenog osnovnog okruženja, Java program se može pisati i u nekom od složenijih razvojnih okruženja, koja su distribuirale velike firme, kao što su: Sun (Forte for Java) i Borland (JBuilder)
Poželjno je da na računaru bude instalirana i Java dokumentacija radi lakšeg razvoja i održavanja Java programa.

1.1 Pisanje programa

Kao radno okruženje za pisanje i testiranje primera iz ovog praktikuma poslužio nam je TextPad editor. Navedeni editor treba instalirati nakon instalacije Sun-ovog JDK-a[1].
Radna površina TextPada se sastoji iz tri dela.



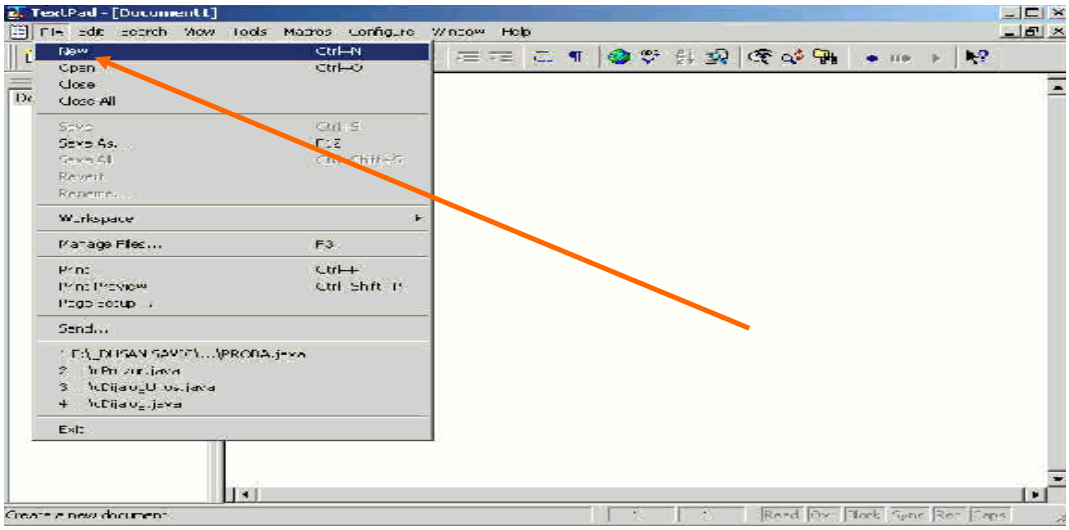
Na vrhu se nalazi meni (File, Edit, Search, View, Tools, Macros, Configure, Window, Help) koji omogućava da se kreira nova Java datoteka, otvori postojeća, ..., itd.

U centralnom delu se nalazi prozor u koji se unosi Java programski kod, dok se na levoj strani nalazi prozor u kome su prikazani trenutno otvoreni Java programi.

Objasnićemo postupak pisanja jednostavnog programa u Javi:

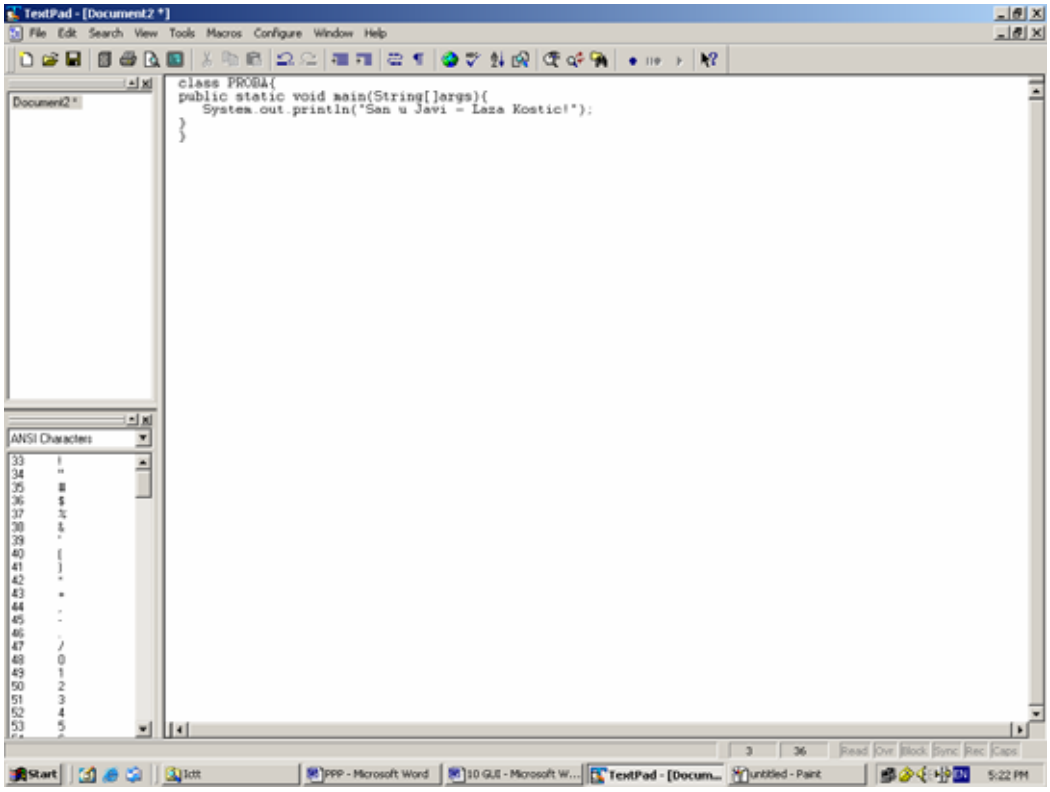
1. Iz *File* menija se bira opcija *New*.

[1] Instaliranje Sunovog JDK je veoma jednostavno, tako da to nismo razmatrali u ovom praktikumu. Polaznicima kursa, koji pokriva ovaj praktikum, biće u praktičnim okolnostima rada na računaru objašnjen postupak instaliranja Sunovog JDK-a.

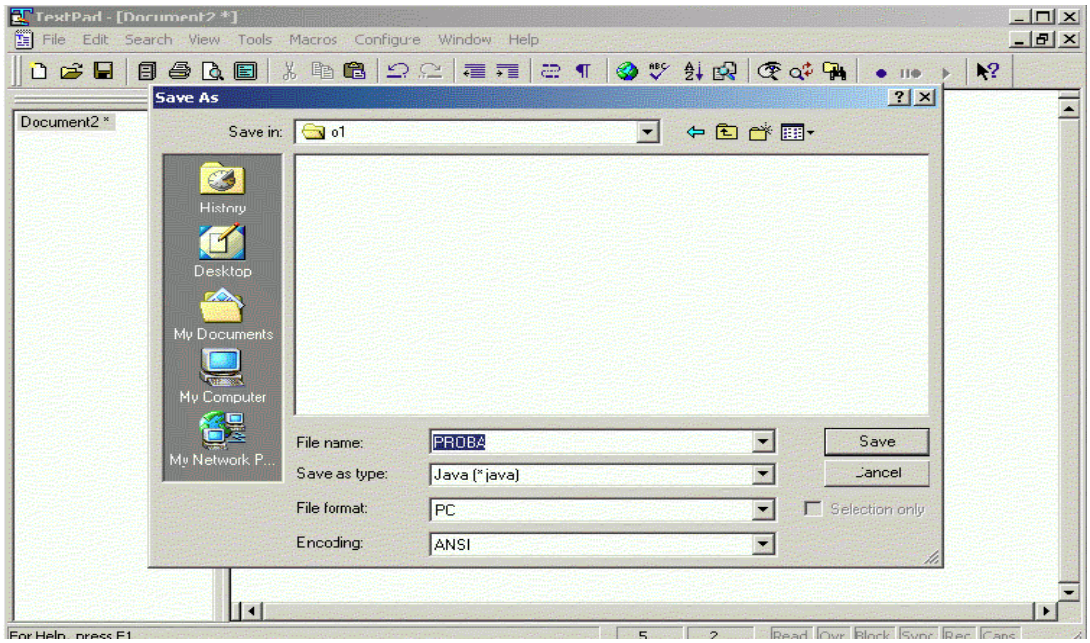


2. Nakon toga se može uneti Java program:

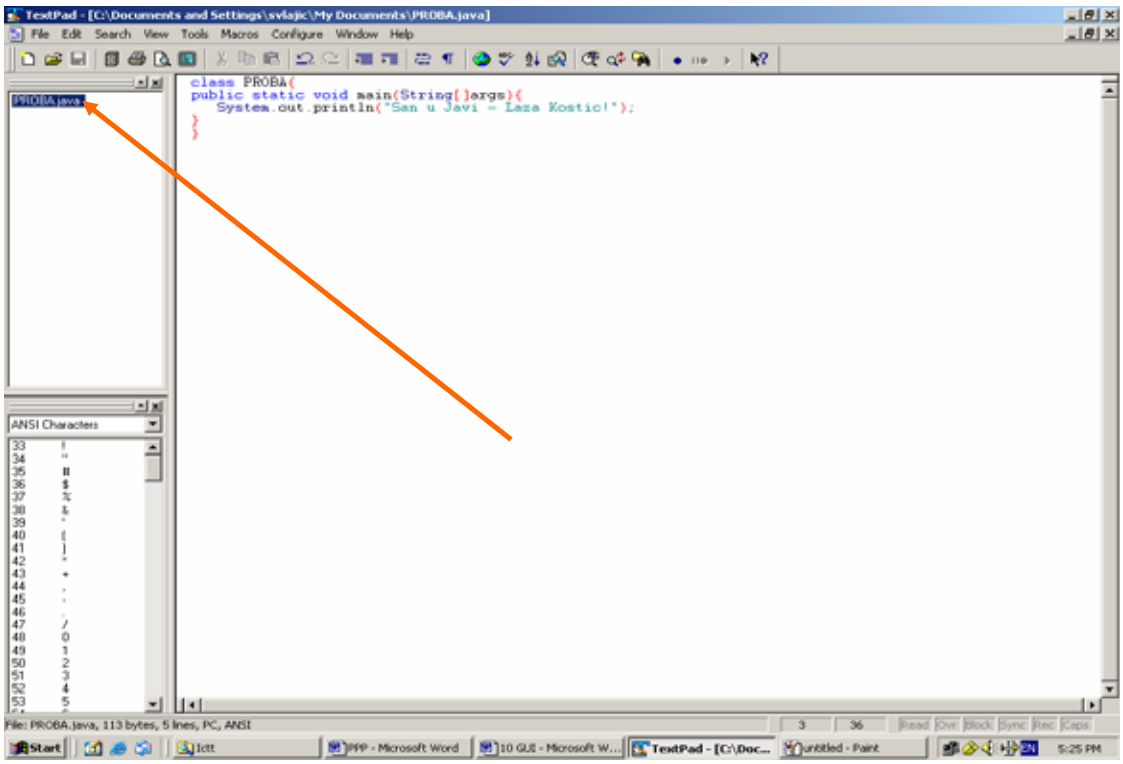
```
class PROBA{
public static void main(String[]args){
   System.out.println("San u Javi – Laza Kostić!");
}
}
```



Pre nego što započne prevođenje Javinog programa, isti mora biti sačuvan u datoteci koja ima isti naziv kao klasa Javinog programa koja sadrži static metodu main(). Iz *File* menija se bira opcija *Save*. U polje *File* name se unosi ime datoteke .(U našem primeru datoteka će biti nazvana PROBA).

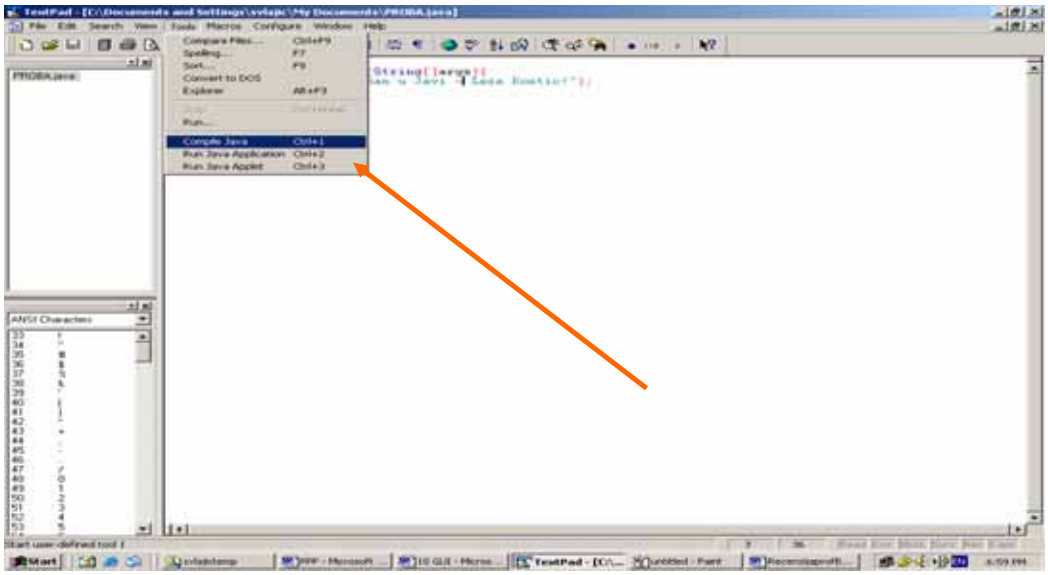


Nakon pamćenja u levom prozoru se prikazuje ime sačuvane datoteke.

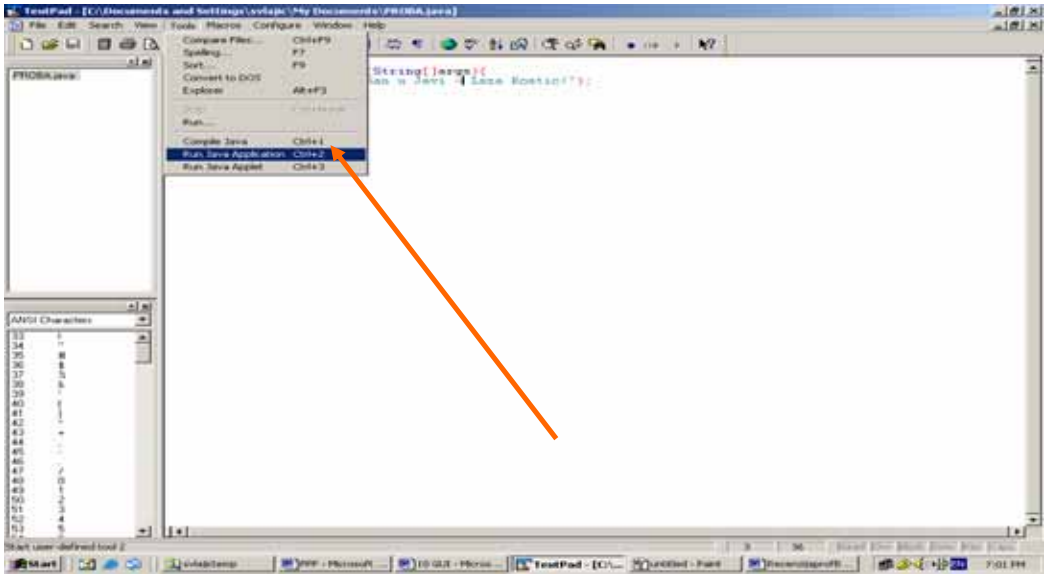


1.2 Prevođenje i izvršavanje programa

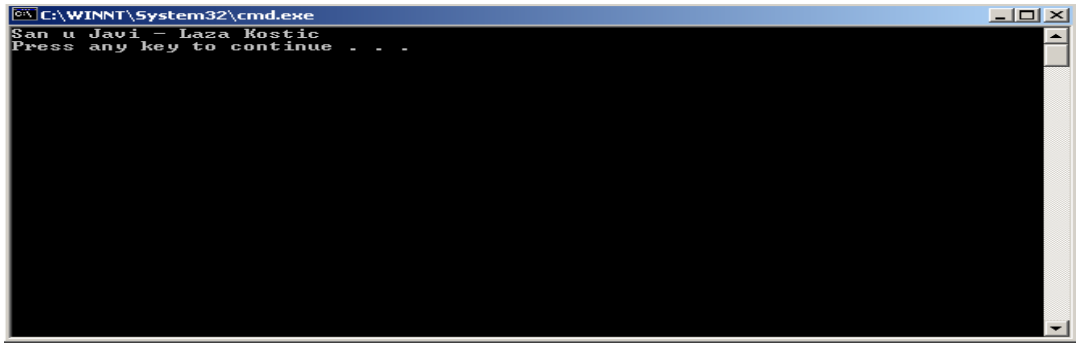
Javin prevodilac se naziva *javac*. Da bi se izvršili prevođenje Java programa mora se pozvati prevodilac koji je integrisan u TextPad-u (inače on se automatski integriše ako TextPad instaliramo nakon JDK-a, mada ga možemo i kasnije pridodati).
Iz menije *Tools* bira se opcija *Compile Java*.



Kao rezultat prevođenja se dobija datoteka *PROBA.class*, ukoliko program nije imao sintaksnih grešaka. Nakon toga se može startovati program izborom opcije *Run Java Application* iz *Tools* menija.

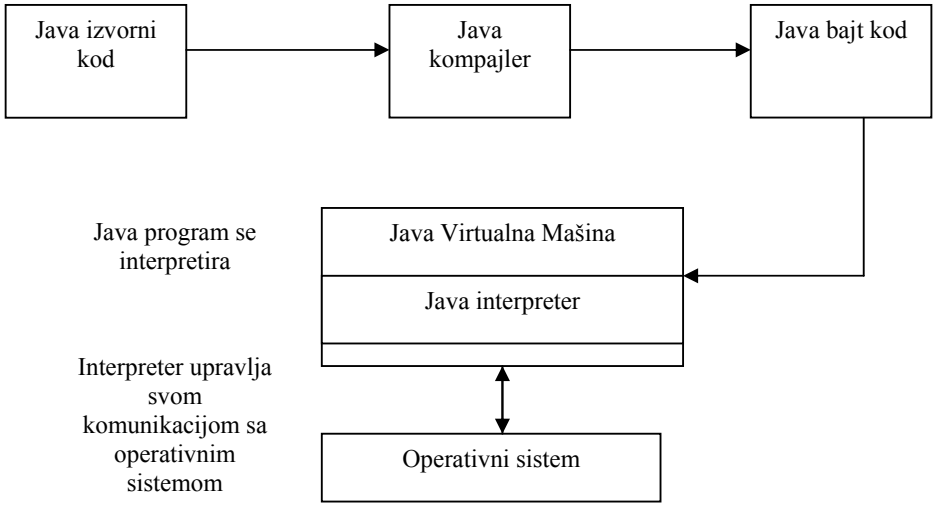


Kao rezultat izvršenja programa na ekranu se dobija poruka: “San u Javi – Laza Kostic ”.



1.3 Pozadina izvršavanja Java programa

Java se može izvršiti na različitim operativnim sistemima, pod pretpostavkom da je za operativni sistem napravljena JVM, koja preslikava naredbe Jave u naredbe konkretnog operativnog sistema. Postupak kompajliranja i izvršavanja Java programa se može predstaviti preko sledeće slike:



Java kompajler konvertuje Java izvorni kod u bajt programski kod.. Bajt programski kod se interpretira pomoću JVM, jer on sadrži naredbe koje prepoznaje JVM i koje preslikava u naredbe konkrektnog operativnog sistema. U toku interpretiranja bajt programskog koda JVM proverava validnost programa (da nije došlo do kvara u programu) i bezbednost njegovog aktiviranja. JVM može da radi samostalno ili da bude deo Web čitača.
Zbog činjenice da se Java program sastoji od bajt kodova, a ne od izvornih mašinskih naredbi, on je u potpunosti izolovan od hardvera na kome radi.

1.4 Zadaci

Zadatak OZ1: Napisati program u Javi koji će prikazati neku poruku na ekranu.

Zadatak OZ2: Napisati program koji će na ekranu da prikaže stringove koji su prihvaćeni kao parametri programa.

Tematska jedinica	**Radno okruženje Jave**	
Datum:		
Zadatak		
Ime i prez. studenta – br. ind.		
Laborant:		

2. Definicija klase i deklaracija objekta

Klasa predstavlja apstraktnu predstavu skupa objekata koji imaju iste osobine. Klasa se sastoji od atributa[2] i metoda. Atributi i metode klase se nazivaju članice klase. Objekat predstavlja jedno konkretno pojavljivanje (primerak,instancu) svoje klase.

2.1 Opšti oblik definicije klase

```
class imeklase
 { tip atribut1;
            ...
   tip atributn;

   tip ime_metode1(lista parametara)
         { telo metode1}
  ...
    tip ime_metodem(lista parametara)
       { telo metodem}
}
```

- Definicija metoda klase i njihova realizacija (implementacija) se vrše istovremeno za razliku od C++, gde se definicija metoda (*****.**h** datoteke) i njhova realizacija (*****.**cpp** datoteke) odvajaju.
- Ekstenzija za datoteku u kojoj se čuva Java klasa je *****.**java.**

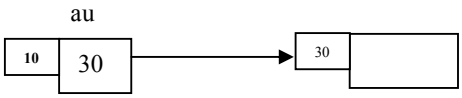
2.2 Konkretna definicija klase i njenih članica

```
class AutomatNovca
{  double Stanje;                    // atribut – promenljiva instanci klase
   void Ulaganje(double Stanje1) { Stanje = Stanje + Stanje1;} // metode klase
   double Podizanje(double) { Stanje = Stanje - Stanje1; return Stanje;}
}
```

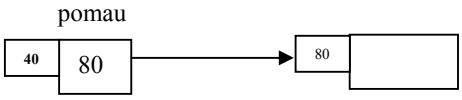
2.3 Deklaracija objekta

- Objekat predstavlja jedno pojavljivanje klase. Objekat se deklariše i kreira na sledeći način: *AutomatNovca au = new AutomatNovca();*
- Za razliku od C++, sa *au* se radi kao sa statičkim objektom, u smislu poziva njegovih članica.
- *au* je u suštini referenca na objekat a ne sam objekat.
- Referenca na objekat *au* može da dobije referencu na bilo koji objekat koji je klase AutomatNovca(i klasa koje su izvedene iz klase AutomatNovca, što ćemo kasnije detaljno objasniti). Npr:

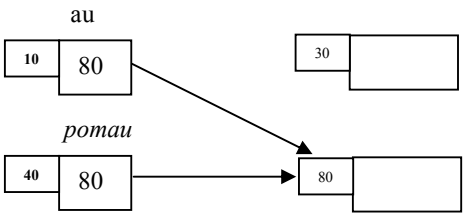
AutomatNovca au = new AutomatNovca();



AutomatNovca pomau = new AutomatNovca();



au = pomau;



[2] U Javi se koristi termin instance variables kada se želi ukazati na atribute klasa.

- Treba naglasiti da *au* dobija referencu na objekat, na koji pokazuje *pomau*. Ne pravi se posebna kopija za *au* koja ima vrednost kao atributi na koje ukazuje *pomau.*
- Za objekte *au* i *pomau* se kažu da su referentni objekti jer oni sadrže referencu, odnosno adresu objekta(U C++ novo kreiranom objektu se pristupa preko: **pomau* ili u Pascalu: *pomau^*).

2.4 Poziv članica objekta

```
tr.Stanje = 14; // Poziv atributa objekta
tr.Podizanje(10); //Poziv metode objekta
```

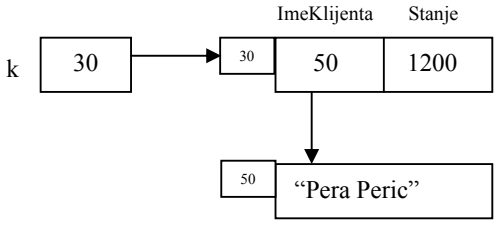
2.5 Primeri - Klase

Primer K1:

/ Programski zahtev: Definisati klasu AutomatNovca koja ima atribute ImeKlijenta tipa String i Stanje tipa double. Deklarisati objekat klase AutomatNovca. Napuniti objekat sa konkretnim vrednostima. Prikazati objekat na standardnom izlazu. */*

```
import java.util.*;

class AutomatNovca{
String ImeKlijenta;
double Stanje;
public static void main(String args[])
  { AutomatNovca k = new AutomatNovca ();
    k.ImeKlijenta = "Pera Peric";
    k.Stanje = 1200;
    System.out.println("Klijent: " + k.ImeKlijenta + ". Stanje: " + k.Stanje);
  }
}

//Rezultat:
// Klijent: Pera Peric. Stanje: 1200.0
```

Izgled operativne memorije:



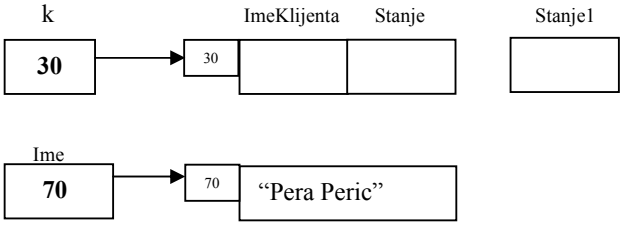
Primer K2:

/ Programski zahtev: Definisati klasu AutomatNovca koja ima atribute ImeKlijenta tipa String) i Stanje tipa double i metode UnesiImeKlijenta, Ulaganje i PrikaziKlijenta. Deklarisati objekat klase AutomatNovca. Koristeci navedene metode napuniti objekat.Prikazati objekat na standardnom izlazu.*/*

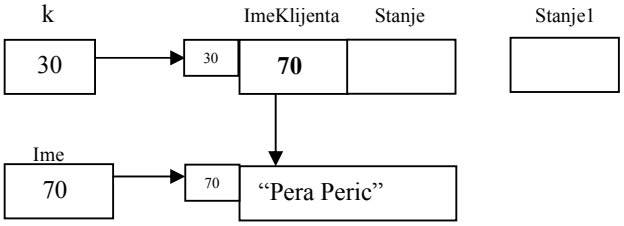
```
import java.util.*;

class AutomatNovca{

String ImeKlijenta;
double Stanje;

void UnesiImeKlijenta(String Ime) // Linija A
{ ImeKlijenta = Ime; } // Linija B
void Ulaganje (double Stanje1) // Linija C
{Stanje = Stanje1;} // Linija D
void PrikaziKlijenta ()
 {  System.out.println("Klijent: " + ImeKlijenta + ". Stanje: " + Stanje);}
```

```
public static void main(String args[])
  { AutomatNovca k = new AutomatNovca();
    k.UnesiImeKlijenta("Pera Peric");
    k.Ulaganje(1200);
    k.PrikaziKlijenta ();
  }
}

//Rezultat:
// Klijent: Pera Peric. Stanje: 1200.0
```

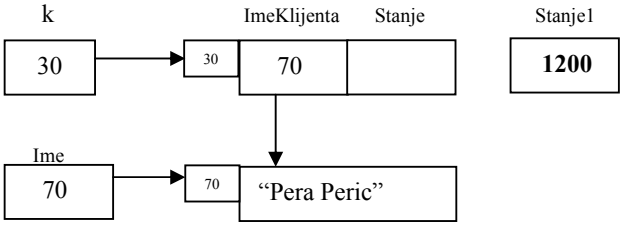
Izgled operativne memorije – Linija A:



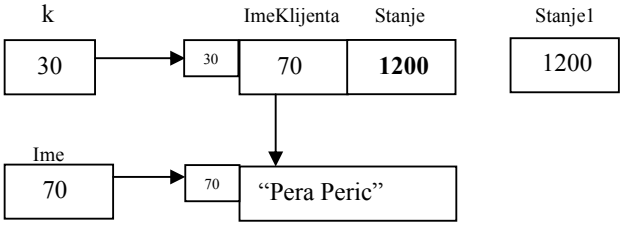
Izgled operativne memorije – Linija B:



Izgled operativne memorije – Linija C:



Izgled operativne memorije - Linija D:



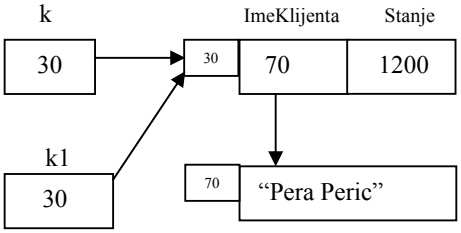
Primer K3:

*/*Programski zahtev: Definisati klasu AutomatNovca koja ima atribute ImeKlijenta tipa String I Stanje tipa double . Deklarisati objekat (k) klase AutomatNovca. Napuniti objekat sa konkretnim vrednostima. Prikazati objekat na standardnom izlazu. Deklarisati novi objekat (k1) klase AutomatNovca. Omoguciti da objekat k1 radi sa objektom k. */*

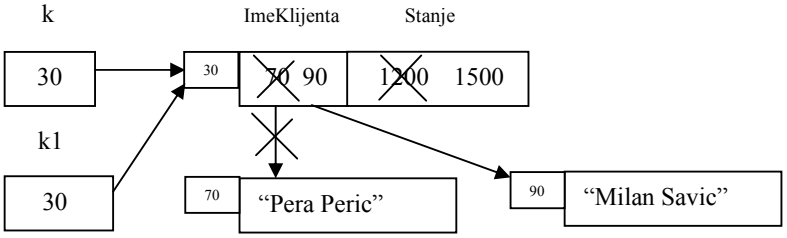
```
import java.util.*;

class AutomatNovca{
String ImeKlijenta;double Stanje;
```

```
public static void main(String args[])
  {
    AutomatNovca k = new AutomatNovca();
    k.ImeKlijenta = "Pera Peric";
    k.Stanje = 1200;
    System.out.println("Klijent: " + k.ImeKlijenta + " Stanje: " + k.Stanje);
    AutomatNovca k1  = k;   // Linija A
    k1.ImeKlijenta = "Milan Savic";
    k1.Stanje = 1500; // LinijaB
    System.out.println("Klijent: " + k.ImeKlijenta + " Stanje: " + k.Stanje);

  }}
```

Izgled operativne memorije - Linija A:



Izgled operativne memorije – Linija B:




```
//Rezultat:
//
// Klijent: Pera Peric Stanje: 1200.0
// Klijent: Milan Savic Stanje: 1500.0
```

2.6 Zadaci - Klase

Zadatak KZ1: Napraviti klasu Student koja ima atribute BrojIndeksa tipa String, ImeStudenta tipa String i TekucaGodina tipa int. Deklarisati objekat klase Student. Napuniti objekat sa konkretnim vrednostima. Prikazati objekat na standardnom izlazu.
Zadatak KZ2: Napraviti klasu Predmet koja ima atribute SifraPredmeta(int), ImePredmeta (String) i NazivProfesora(String). Deklarisati objekat klase Predmet i metode UnesiSifruPredmeta, UnesiNazivPredmeta, UnesiImeProfesora i PrikaziPredmet Koristeci navedene metode napuniti objekat.Prikazati objekat na standardnom izlazu.

Pitanja:

1. Koja je razlika između klase i objekta?
2. Šta se dešava kod deklaracije: AutomatNovca k = new AutomatNovca();?
3. Kako se pristupa članicama objekta (dati primer)?

Pitanja sa odgovorima:

1.Šta će se desiti ako se ispred main() izostavi static i/ili public?
Odgovor: Iskompajliraće se program, ali kod njegovog pokretanja pojaviće se poruka:
Exception in thread 'main' java.lang.NoSuchMethodError:main
2.Šta će se desiti ako se izostavi parametar args[] u main()?
Odgovor: Iskompajliraće se program ali kod njegovog pokretanja pojaviće se poruka:
Exception in thread 'main' java.lang.NoSuchMethodError:main

3.Da li se može da se kreira više objekata (au,au1) kod naredbe:
AutomatNovca au,au1=new AutomatNovca()?
…
au.Ulaganje(25);
Odgovor:Ne može. Kod kompajliranja se javlja poruka: variable au might not have been initialized au.Ulaganje(25);
U navedenom primeru kreiraće se samo objekat au1 i sa njim može da se radi za razliku od objekta au koji nije kreiran i samim tim nad njim se ne mogu izvršiti metode njegove klase. Svaki od objekata u Javi se mora nezavisno kreirati sa new naredbom.
4. Šta će se desiti ukoliko se kod definicije klase stavi Class kao ključna reč?
*Odgovor: Kod definicije klase mora se obavezno staviti **c**lass a ne **C**lass jer Java kompajler reaguje na mala i velika slova.*
5. Da li je na kraju definicije klase obavezno iza zagrade '}' staviti znak ';'.
Odgovor: Za razliku od C++-a, gde se na kraju definicije klase obavezno stavlja ';', kod Jave to može da se stavi ali nije obavezno.

Tematska jedinica	**Definicija klase i deklaracija objekta**	
Datum:		
Zadatak		
Ime i prez. studenta – br. ind.		
Laborant:		

3. Metode klase

- Metode predstavljaju funkcije klase u kojoj su definisane.
  class imeklase
  { tip atribut1;
  …
  tip atributa;
  tip ime_metode1(lista parametara)
  { telo metode1}
  …
  tip ime_metodem(lista parametara)
  { telo metodem}
  }

3.1 Sastav metode

Svaka metoda sadrži:
a) tip koji vraća (ili void ukoliko ne vraća nikakav tip),
b) naziv,
c) listu parametara i
d) telo metode.

Ukoliko metoda vraća neku vrednost tada se koristi ***return vrednost.***
double *Podizanje(double Stanje1) { Stanje = Stanje - Stanje1;* ***return Stanje****;}*

3.2 Parametrizirane metode

Ukoliko metoda sadrži parametre, za nju se kaže da je ona parametrizirana metoda.
*double Podizanje(****double Stanje1****) { Stanje = Stanje - Stanje1; return Stanje;}*

3.3 Konstruktori

Konstruktori predstavljaju specijalne metode pomoću kojih se vrši inicijalizacija objekta (dodeljivanje početnih vrednosti atributima objekta), prilikom njihovog kreiranja.

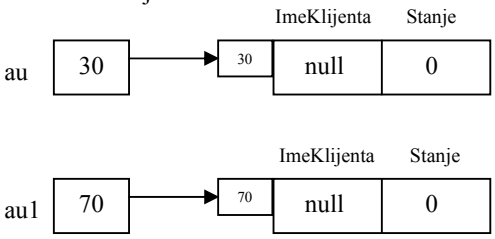
- Konstruktor ima isti naziv kao i ime klase.
- Konstruktor se pokreće kod deklaracije objekta.
- Konstruktori ne vraćaju nikakav tip podataka (čak ni void).

Default konstruktori

Default konstruktori postavljaju atribute svakog objekta na istu početnu vrednost.

```
class AutomatNovca
  {  String ImeKlijenta;      double Stanje;
      // Default konstruktor
    AutomatNovca(){ ImeKlijenta = null; Stanje = 0;}
     public static void main(String args[])
      { // Kod deklaracije objekta poziva se default konstruktor .
          AutomatNovca au = new AutomatNovca();
          AutomatNovca au1 = new AutomatNovca();
        ...
      } ... }
```

Izgled operativne memorije:

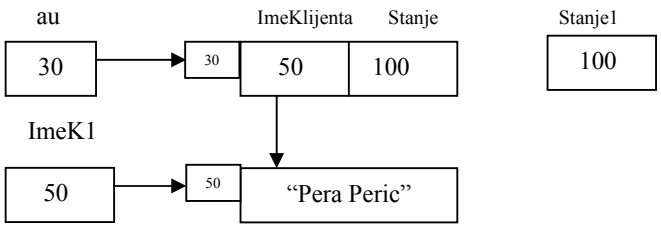


Parametarski konstruktori

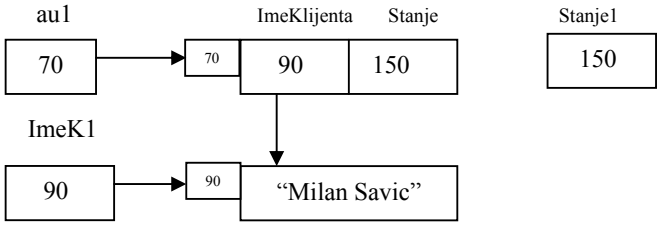
Parametarski konstruktori postavljaju atribute objekta na vrednosti koje su definisane parametrima kod deklaracije objekta.

```
class AutomatNovca
  { String ImeKlijenta;
    double Stanje;
      // Parametarski ili copy konstruktor
    AutomatNovca(String ImeK1, double Stanje 1){ ImeKlijenta = ImeK1; Stanje = Stanje1;}
     public static void main(String args[])
       { // Kod deklaracije  objekta poziva se copy konstruktor.
           AutomatNovca au = new AutomatNovca("Pera Peric",100); // Linija A
          AutomatNovca au1 = new AutomatNovca("Milan Savic",150); // Linija B
         ...
       }
... }
```

Izgled operativne memorije Linija A, uključujući i poziv konstruktora:



Izgled operativne memorije Linija B, uključujući i poziv konstruktora:

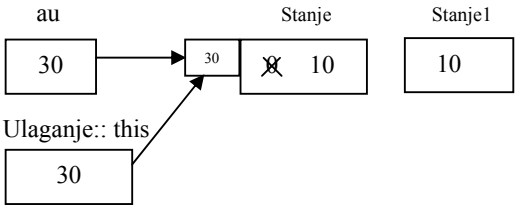


3.4 Rezervisana reč this

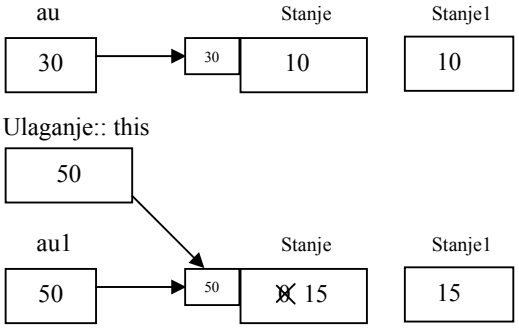
Rezervisana reč this predstavlja referencu na objekat koji je pozvao sopstvenu metodu klase.

```
class AutomatNovca
 {   double Stanje;
    AutomatNovca(){ Stanje = 0;}
     void Ulaganje(double Stanje1)
        {Stanje = Stanje + Stanje1;}
      // Moglo je da se napise this.Stanje = this.Stanje + Stanje1;
      // sto pokazuje da this ukazuje na tekuci objekat koji je
     // pozvao metodu Ulaganje.
    ...

    public static void main(String args[])
     { AutomatNovca au = new AutomatNovca();
       AutomatNovca au1 = new AutomatNovca();
      au.ulaganje(10); // Linija A
      au1.ulaganje(15); // Linija B
     }
}
```

Izgled operativne memorije - Linija A:



Izgled operativne memorije - Linija B:

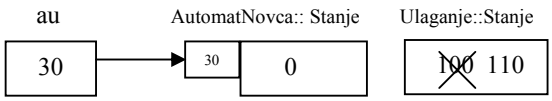


3.5 Sakrivanje atributa klasa preko parametara metoda

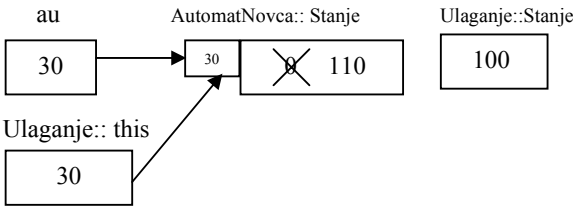
U Javi je za razliku od C++ moguće da parametri metoda i lokalne promenljive imaju isti naziv kao i atributi klase. Na taj način dolazi do preklapanja naziva, pri čemu parametri ili lokalne promenljive metode ***sakrivaju atribute klase***. Atributu klase u tom slučaju, ne može se pristupiti jer poziv atributu klase preuzima parametar ili lokalna promenljiva metode.

```
class AutomatNovca
 {  double Stanje;
    AutomatNovca(){ Stanje = 0;}
    void Ulaganje(double Stanje)
        {Stanje = Stanje + 10;} //Linija A
        // U navedenom slucaju se poziva parametar Stanje a ne
        // atribut Stanje. Navedeni problem se resava tako sto se
        // ispred atributa klase stavlja this.
        // {this.Stanje = Stanje + 10;} // Linija B
        // this.Stanje – pristupa se atributu klase
        // Stanje – pristupa se parametru klase.


public static void main(String args[])
      { AutomatNovca au = new AutomatNovca();
        au.ulaganje(100);
      }
 ...
}
```

Izgled operativne memorije – Linija A:



Izgled operativne memorije – Linija B:



3.6 Sakupljanje smeća

Za razliku od C++ koji ima specijalnu metodu destruktor, Java nema takvu metodu. Pošto je uloga destruktora vezana za uništavanje objekta, postavlja se sledeće pitanje: Koji mehanizam kod Jave obezbeđuje uništavanje objekata koji se više ne koriste?
Navedeni mehanizam se naziva ***garbage collection (sakupljanje smeća)***, koji radi na sledećem principu: analiziraju se objekti programa i traže se oni objekti na koje više ne ukazuju referentni objekti. Takvi objekti se sakupljaju i uništavaju. Vreme pokretanja navedenog mehanizma nije vezano za izlazak objekta iz oblasti njegovog važenja već se to radi s vremena na vreme i nije pod neposrednom kontrolom onoga ko pravi program, kao što je to kod C++.

3.7 Metoda finalize()

Metoda finalize() se pokreće svaki put kada mehanizam ***garbage collection*** pokuša da uništi objekat. Navedene metoda se pokreće neposredno pre uništenja objekta.
Opšti oblik navedene metode je:

```
protected void finalize()
{ // kod finalizacije
}
```

Pošto se ne zna vreme kada će se pokrenuti mehanizam ***garbage collection,*** ne može se sa sigurnošću tvrditi kada će se i da li će se uopšte pokrenuti metoda finalize().

3.8 Preklapanje metoda

U Javi kao i u C++ moguće je definisati istoimene metode u okviru jedne klase. U tom slučaju se kaže da su metode preklopljene (overloaded). Preklapanje metoda predstavlja jedan oblik polimorfizma (polimorfno - nešto što ima više oblika). Međutim u okviru jedne klase preklopljene metode moraju da se razlikuju po potpisu (po broju i/ili tipu parametara).

Primer neregularnog preklapanja metoda:

```
class AutomatNovca
  {  double Stanje;
     void Ulaganje(double Stanje1)
          { Stanje = Stanje + Stanje1;}

 // Nize navedena metoda preklapa vise navedenu metodu i one se razlikuju  po tipu
// onoga sto vracaju. Medjutim broj i tip argumenata je isti tako da ce se kod
// kompajliranja javiti greska: Ulaganje(double ) is already defined in AutomatNovca

     double Ulaganje(double Stanje1)
          { Stanje = Stanje + Stanje1; return Stanje;}
   ...}
```

Sledeći primer pokazuje regularno preklapanje istoimenih metoda:

```
class AutomatNovca
  {  double Stanje;
     void Ulaganje(double Stanje1)        { Stanje = Stanje + Stanje1;}
 // Nize navedena metoda preklapa vise navedenu metodu i one se razlikuju po tipu
// tipu parametraStanje1. To je dovoljan uslov da se regularno kompajlira program.
    void Ulaganje(int Stanje1)          { Stanje = Stanje + Stanje1;}
   ...
}
```

Iz metode main() koja je deklarisana u okviru klase AutomatNovca preklopljena metoda Ulaganje() se poziva na sledeći način:

```
AutomatNovca au = new AutomatNovca();
au.Ulaganje(5); // Poziva se metoda Ulaganje() koja kao parametar ima int.
au.Ulaganje(5.5); // Poziva se metoda Ulaganje() koja kao parametar ima double.
```

3.9 Preklapanje konstruktora

Kao i bilo koje druge metode tako se i konstruktor metode mogu preklopiti.

```
class AutomatNovca
  {  double Stanje;
     // Preklopljeni konstruktori
     AutomatNovca(){ Stanje = 0;}
     AutomatNovca(double Stanje1){ Stanje = Stanje1;}
     public static void main(String args[])
      { // Kod deklaracije objekta poziva se default konstruktor .
          AutomatNovca au = new AutomatNovca();
          AutomatNovca au1 = new AutomatNovca(15.5);
        ...
      }
...}
```

3.10 Prenos argumenata metoda

Postoje dva načina prenosa argumenata:

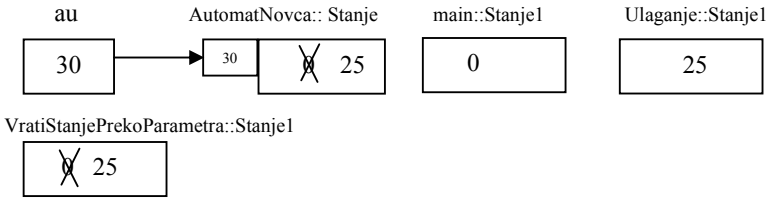
a) preko vrednost (call-by-value)

Kod navedenog pristupa se vrednost argumenta kopira u formalni parametar metode[3]. Formalni parametar predstavlja posebnu kopiju koja ima vrednost istu kao što je vrednost argumenta. Sve što se dešava sa kopijom neće se odraziti na argument.

[3] Java pravi razliku između pojma argument i pojma parametar. Argument (ono što smo zvali stvarni parametar) je vezan za poziv metode, dok je parametar (ono što zovemo formalni parametar) vezan za definiciju metode koja je pozvana.

```
class AutomatNovca
 {  double Stanje;
     AutomatNovca(){ Stanje = 0;}
    // Navedeni parametar Stanje1, ima smisla da se prenese preko
    // vrednosti, jer je on u navedenom slucaju samo ulazni parametar.
   void Ulaganje(double Stanje1)         { Stanje = Stanje + Stanje1;}
    ...
    // Navedeni parametar Stanje1 nece izvrsiti postavljeni zahtev jer
   // parametar Stanje1 nece, kada se vrati u main()   imati  vrednost
   // zeljenog atributa Stanje. U navedenom slucaju od parametra se
   // zahteva da bude ulazno/izlazni parametar sto on ne moze biti kod
   // prenosa parametara preko vrednosti.
  void VratiStanjePrekoParametra(double Stanje1)     { Stanje1 = Stanje;  }

     public static void main(String args[])
       { AutomatNovca au = new AutomatNovca();
         double Stanje1 = 0;
         au.Ulaganje(25);
         ...
         au.VratiStanjePrekoParametra(Stanje1); }
    // Stanje1 će na kraju da prikaže 0 umesto željene vrednosti 25.
 }
```

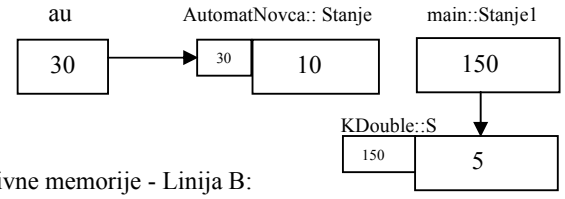
Izgled operativne memorije:



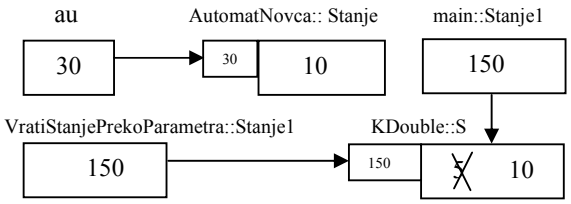
b) preko reference (call-by-reference)
Kod navedenog pristupa se formalnom parametru metode prosleđuje referenca na argument. To znači da sve što se bude radilo sa formalnim parametrom imaće uticaj na argument koji je prosleđen do formalnog parametra.
Ukoliko bi želeli da postignemo efekat vraćanja vrednosti preko parametra, što ne može da se ostvari prenosom parametara preko vrednosti, potrebno je proslediti objekat čiji je atribut realnog tipa.

```
class AutomatNovca
 {  double Stanje;
     AutomatNovca(){ Stanje = 10;}
    void VratiStanjePrekoParametra(KDouble Stanje1)    { Stanje1.S = Stanje;  }

     public static void main(String args[])
       { AutomatNovca au = new AutomatNovca();
         KDouble Stanje1  = new KDouble(); // Linija A
         ...
         au.VratiStanjePrekoParametra(Stanje1); // Linija B
        System.out.println("Vrednost stanja je:" + Stanje1.S);
       // Stanje1.S će na kraju da prikaže željenu vrednost 25.
       }}
class KDouble
{ double S; KDouble() { S = 5;}}
```

Izgled operativne memorije - Linija A:



Izgled operativne memorije - Linija B:



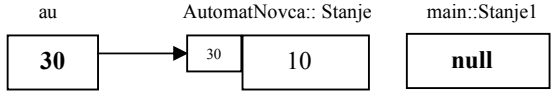
- ***Kod Jave prosti tipovi se uvek prenose preko vrednosti dok se objekti uvek prenose preko reference***

3.11 Vraćanje objekta pomoću metode

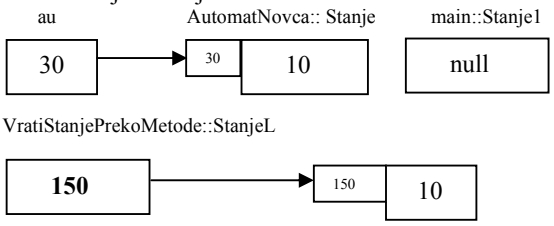
Metode mogu da vrate bilo koji tip podataka. Samim tim one mogu da vrate klasu (koja je po definiciji tip podatka) ili neki prost tip (int,char,…).

```
class AutomatNovca
 {  double Stanje;
    AutomatNovca(){ Stanje = 10;}
     public static void main(String args[])
       { AutomatNovca au = new AutomatNovca();
        KDouble Stanje1 = null; // Linija A
        ...
        // Referentni objekat Stanje1 dobija referencu na objekat koji
        // je kreiran u metodi VratiStanjePrekoMetode().
        Stanje1 = au.VratiStanjePrekoMetode ();
       System.out.println("Vrednost stanja je:" + Stanje1.S); // Linija C
       }
     // Ovde se kao rezultat metode vraca objekat StanjeL.
    KDouble VratiStanjePrekoMetode ()
     { KDouble StanjeL = new  KDouble();
       StanjeL.S = Stanje;
       return StanjeL; // Linija B
      }
}
class KDouble
{ double S;
 Kdouble() { S = 5;}}
```

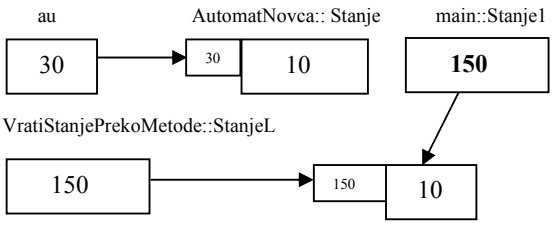
Izgled operativne memorije – Linija A:



Izgled operativne memorije – Linija B:



Izgled operativne memorije – Linija C:



3.12 Ključna reč static

Kada se javi potreba da neka od članica klase bude deljiva, kako za objekte klase kojoj pripada tako i za objekte drugih klasa, ispred nje se stavlja ključna reč static. Na taj način static članica, ukoliko je:

a) atribut, ponaša se kao globalna promenljiva,
b) metoda, ponaša se kao sopstvena funkcija[4].

Pristup do static članica je sledeći:

Ime_Klase.ime_static_članice

pri čemu ***Ime_Klase*** označava klasu kojoj pripada static članica.

Kod poziva nekog programa[5] pre kreiranja bilo kog objekta, poziva se static metoda main(). Ukoliko main() metoda ne bi bila static metoda, poziv programa ne bi mogao da se izvrši jer se ne static metode mogu pozvati tek nakon kreiranja objekta, koji će ih pozvati.

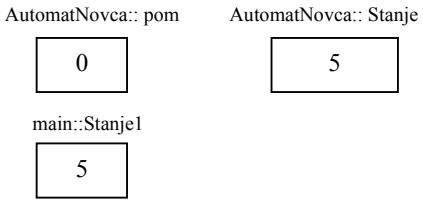
Static metode mogu da:

- pozivaju direktno druge static metode.
- direktno pristupaju static atributima.

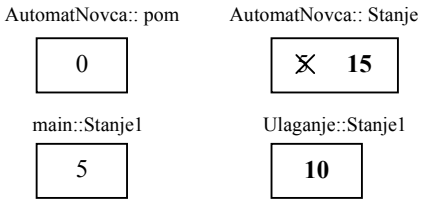
```
class AutomatNovca
 { static double Stanje;
   static double pom = 0; // 1 korak
   double Stanje2;
    public static void main(String args[]) //3 korak
      {  double Stanje1;
       Stanje1 = Stanje; // direktan poziv static_atributa   // Linija A
       ...
          System.out.println("Vrednost stanja je:" + Stanje1);
          // Isto tako moglo je da se napise
          System.out.println("Vrednost stanja je:" + Stanje);

        Ulaganje(10); // direktan poziv druge_static metode Linija B
        Podizanje (5); // ne moze se direktno pozvati ne static metoda
       Stanje2 = 15;// ne moze se direktno pristupiti_ne static atributu
      }
    static // 2 korak
       { Stanje = 5;}
   static void Ulaganje(double Stanje1)  { Stanje = Stanje + Stanje1;}
    void Podizanje(double Stanje1)    { Stanje = Stanje – Stanje1;}
}
```

[4] U C++ bilo je moguće da se definišu sopstvene funkcije (funkcije koje se defini{u kao nezavisne od bilo koje klase). U Javi ne postoji koncept sopstvenih funkcija. Sve funkcije, makar se one ponašale i kao sopstvene funkcije, definišu se u okviru klase.
[5] Ime programa je isto kao i ime klase u kojoj se nalazi main() metoda

Izgled operativne memorije – Linija A:



Izgled operativne memorije – Linija B:



Static metode ne mogu da:

- pozivaju direktno ne static metode.
- pristupaju direktno ne static atributima.
- koriste ključne reči *this* i *super*.

Kod izvršenja klase koja ima static članice redosled izvršenja je sledeći:

1. Inicijalizuju se static atributi ukoliko im je dodeljena neka vrednost.
   static double pom = 0; // 1 korak

2. Izvršava se static blok[6] koji vrši inicijalizaciju static atributa.
   static // 2 korak
   { Stanje = 5;}

3. Poziva se main() metoda.
   public static void main(String args[]) //3 korak
   {...}

Pomoću static atributa i metoda može da se simulira klasičan proceduralni pristup u programiranju. Jedino razlika, između klasičnog proceduralnog programa i "objektnog proceduralnog (static) programa", se odnosi na okvir u kome se pišu ti programi. Okvir kod klasičnog proceduralnog programa je datoteka u okviru koje se piše program, dok je kod " objektno proceduralnog programa" okvir i datoteka i klasa u kojoj se piše program.

Primer za **Klasičan proceduralni program**

U datoteci test.cpp nalazi se sledeći program:

```
f1() // definisanje sopstvene funkcije
{a = 5; // pristup globalnoj promenljivoj
}

int a; // definisanje globalne promenljive
main()
{ f1(); // poziv sopstvene funkcije
}
```

Kod klasičnog proceduralnog programa okvir je datoteka test.cpp

Primer za **Objektni proceduralni (static) program**

U datoteci test.java nalazi se sledeći program:

```
class test
 {  static void f1() // static metoda
    {  a = 5; // pristup static atributu
    }
```

[6] Static blok moze da se zamisli kao konstruktor za inicijalizaciju static atributa.

```
static int a; // definisanje static atributa
 public static void main(Strings args[])
    { f1(); // poziv static metode }
 }
```

Kod objektno proceduralnog (static) programa okvir je datoteka test.java i klasa test.

Ključna reč final za definisanje konstanti u programu

Ukoliko se u klasi javi potreba za definisanjem konstanti, koristi se ključna reč final ispred imena atributa klase.

```
class AutomatNovca
 {...
    final double porez = 0.2; // Atribut porez je definisan kao konstanta.[7]
  ...}
```

Ključna reč final ima i drugu ulogu, osim navedene, što ćemo kasnije objasniti.

3.13 Zadaci – Metode klase

Zadatak MZ1: Za svaki od navedenih primera dati izgled operativne memorije.
Zadatak MZ2: Napisati parametriziranu metodu UnesiNazivPredmeta i neparametriziranu metodu PrikaziPredmet klase Predmet.
Zadatak MZ3: Napisati default i parametarski konstruktor klase Predmet.
Zadatak MZ4: Pokazati na primeru metode PrikaziPredmet klase Predmet kako se vrši preklapanje metoda.
Zadatak MZ5: Pokazati na primeru metode UnesiImeProfesora klase Predmet kako se parametri prenose preko vrednosti.
Zadatak MZ6: Pokazati na primeru metode VratiImeProfesora klase Predmet kako se parametri prenose preko reference.
Zadatak MZ7: Pokazati kako se objekti vracaju preko metoda na primeru metode VratiPredmet klase predmet.
Zadatak MZ8: Dati primer static atributa za klasu Predmet. Pokazati i objasniti kako se njemu pristupa iz različitih objekata klase Predmet.

Pitanja:

1. Kada se pozivaju konstruktor metode?
2. Koja je glavna razlika između default i parametarskih konstruktor metoda?
3. Objasniti značenje ključne reči this.
4. Kako se sakrivaju atributi klasa preko parametara metoda?
5. Kada se pokreće mehanizam sakupljanja smeća (garbage collection)?
6. Koja je svrha metode finalize?
7. Šta znači preklapanje metoda?
8. Da li se konstruktor metode mogu preklopiti?
9. Kako se prenose parametri u Javi?
10. Kako se vraćaju objekti preko metoda?
11. Da li static metode mogu da pozivaju ne static atribute i metode?
12. Koja je glavna razlika između static i običnih metoda?

Tematska jedinica	**Metode klase**	
Datum:		
Zadatak		
Ime i prez. studenta – br. ind.		
Laborant:		

[7] Kao što je poznato konstante, za razliku od promenljvih, ne mogu menjati svoju vrednost u toku izvršenja programa. Njihova vrednost za svo vreme izvršenja programa ostaje ista.

4. Nasleđivanje

Nasleđivanje predstavlja jedan od osnovnih koncepata objektno-orijentisanog programiranja. Pomoću koncepta nasleđivanja, klasa koja je izvedena iz osnovne klase, nasleđuje sve atribute i metode osnovne klase.
Java osnovnu klasu naziva natklasa (superclass) dok izvedenu klasu naziva potklasa (subclass).

4.1 Nadklasa i podklasa

U Javi klase se nasleđuju korišćenjem ključne reči **extends.**
Opšti oblik definicije nasleđivanja klasa je:

```
class Natklasa
{...}

class Potklasa  extends Natklasa // Klasa Potklasa nasledjuje klasu Natklasa.
{...}
```

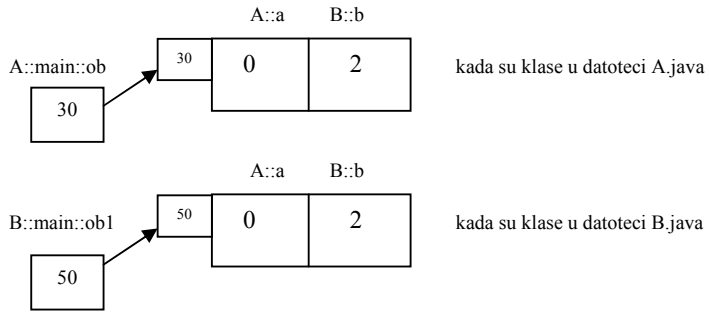
Potklasa može da nasledi samo jednu klasu (za razliku od C++ koji dozvoljava višestruko nasleđivanje klasa).
Kod nasleđivanja i natklasa i potklasa mogu da imaju static metodu main. U zavisnosti od imena datoteke, u kojoj se nalaze klase, izvršava se static metoda main klase, koja ima isti naziv kao i ime datoteke[8].

Npr. ukoliko postoji klasa A i iz nje izvedena klasa B,

```
class A
{ int a;
      // Ova metoda se poziva kada se klase A i B nalaze u datoteci A.java.
    public static void main(String args[])
    {B ob=new B();
     ob.stampaj("main A");}
     A(){a=0;}
  }

class B extends A // Klasa B nasledjuje klasu A
 {int b;
  B(){b=2;}
// Ova metoda se poziva kada se klase A i B nalaze u datoteci B.java.
 public static void main(String args[])
 {B ob1=new B();  ob1.stampaj("main B"); }
 void stampaj(String s) {System.out.println("A i B su: " + a +" "+ b + "Metoda je pozvana iz :" + s);}
}
```

tada se, u slučaju da se ove dve klase nalaze u datoteci A.java, izvršava metoda main iz klase A. Ukoliko se ove dve klase nalaze u datoteci B.java izvrašava se metoda main iz klase B. U obadva slučaja se iz metode main poziva metoda *Prikazi* iz klase B.

Izgled operativne memorije:



[8] Ovo je inače generalni principi kada imamo dve klase u jednoj datoteci nezavisno od taga da li između njih postoji veza nasleđivanja.

U Javi nije dozvoljeno da klasa nasledi samu sebe.

```
Class A extends A  // Klasa A ne moze da nasledi samu sebe.
{...}
```

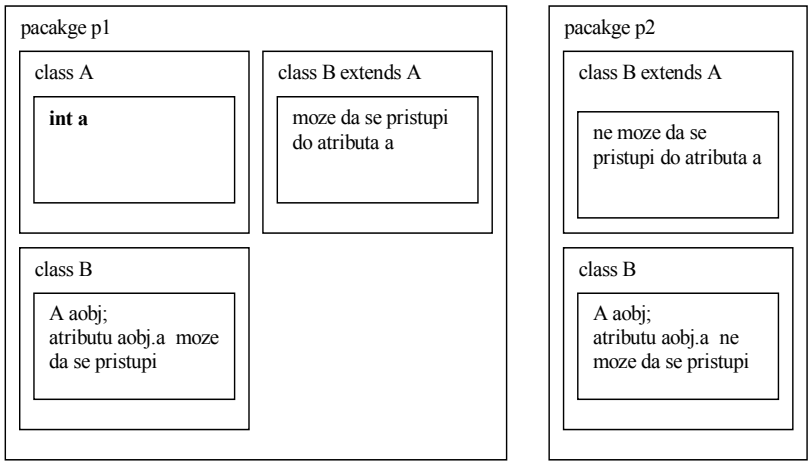
4.2 Način pristupa do članica nadređene klase

Iz potklase može se pristupiti samo onim članicama natklase koje su javne za potklasu. Članice natklase su javne za potklasu u sledećim slučajevima:

a) ispred članice natklase ne postoji nikakav način pristupa, što je podrazumevano javan pristup do članice natklase za sve klase koje se nalaze u paketu[9] gde se nalazi natklasa. To znači da potklase, ukoliko se nalaze u istom paketu gde je i nadklasa, mogu da pristupe do njenih članica. Ukoliko natklasa i potklase nisu vezane za pakete a nalaze se u istom folderu način pristupa iz podklase do članica natklase je javan.

```
class A
{ int a;  // Ispred navedene clanice ne stoji nikakav nacin pristupa sto je
// podrazumevano javan pristup do clanice natklase za sve klase koje se
// nalaze u paketu gde se nalazi natklasa
     public static void main(String args[])
    {B ob=new B();
      ob.stampaj("main A");}
     A(){a=0;}
     void stampaj(){System.out.println("A je: " +a);}
}
class B extends A
 {int b;
  B(){b=2;}
// Iz izvedene klase direktno se poziva clanica a natklase.
 void stampaj(String s) {System.out.println("A i B su: " + a +" "+ b);  }
}
```

Navedeni program će raditi, u slučaju kada klase A i B pripadaju istom paketu[10] i u slučaju kada klase A i B nisu vezane za pakete a nalaze se u istom folderu.

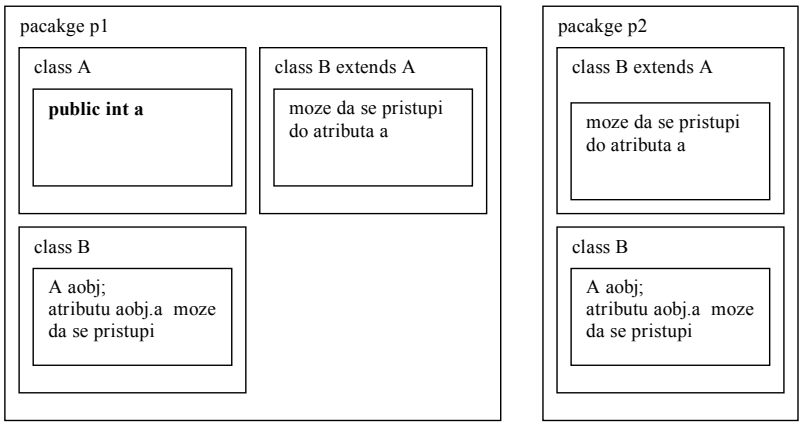


[9] Ono što je sada važno da naglasimo je to, da za jedan paket moze biti vezano više datoteka. Ime paketa treba da odgovara imenu foldera u kome se nalaze datoteke koje su vezane za taj paket. Ukoliko se eksplicitno ne navede paket kome klase u jednoj datoteci pripadaju, podrazumeva se da klase navedene datoteke pripadaju neimenovanom paketu. Klase koje se nalaze u istom folderu a ne pripadaju nijednom paketu se ponašaju, što se tiče načina pristupa, kao da pripadaju istom paketu.

b) ispred članice natklase stoji ključna reč **public,** kojom se omogućava javan pristup do članica natklase iz svih klasa nezavisno od toga kom paketu one pripadaju. To znači da potklase iz svih paketa mogu da pristupe do članica natklase. Ukoliko natklasa i potklase nisu vezane za pakete a nalaze se u istom folderu način pristupa iz podklase do članica natklase je javan.

```
class A
{ public int a;  // Ispred navedene clanice stoji nacin pristupa public
// kojim se omogucava javan pristup do clanice natklase za sve klase
// nezavisno od toga kom paketu one pripadaju.
     public static void main(String args[])
    {B ob=new B();
     ob.stampaj("main A");}
 ...}
class B extends A
 { ...
// Iz izvedene klase direktno se poziva clanica a natklase.
 void stampaj(String s) {System.out.println("A i B su: " + a +" "+ b);  }
}
```

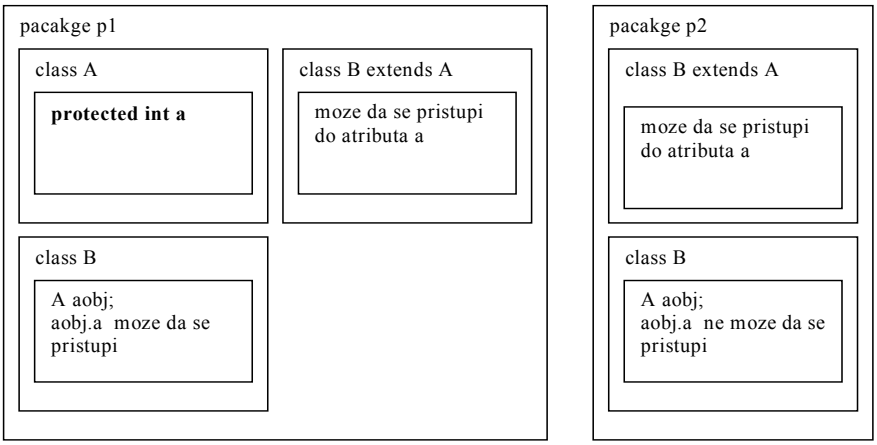
Navedeni program će raditi u svim slučajevimo nezavisno od toga kom paketu pripadaju klase A i B i u slučaju kada klase A i B nisu vezane za pakete a nalaze se u istom folderu.



c) ispred članice natklase stoji ključna reč **protected,** kojom se omogućava javan pristup do članica natklase, iz svih klasa koje se nalaze u istom paketu gde je i nadklasa i svih potklasa nezavisno od toga kom paketu one pripadaju.

```
class A
{ protected int a;  // Ispred navedene clanice stoji nacin pristupa protected
// kojim se omogucava javan pristup do clanice natklase za sve potklase
// nezavisno od toga kom paketu one pripadaju i svih klasa koje se nalaze u
// istom paketu gde se nalazi i natklasa.
     public static void main(String args[])
    {B ob=new B();
     ob.stampaj("main A");}   ...}


class B extends A
{ ...
// Iz izvedene klase direktno se poziva clanica a natklase.
 void stampaj(String s) {System.out.println("A i B su: " + a +" "+ b);  }
}
```

Navedeni program će raditi u svim slučajevimo nezavisno od toga kom paketu pripadaju klase A i B, pod uslovom da je klasa B podklasa klase A, u slučaju kada se klase A i B nalaze u istom paketu, i u slučaju kada klase A i B nisu vezane za pakete a nalaze se u istom folderu.



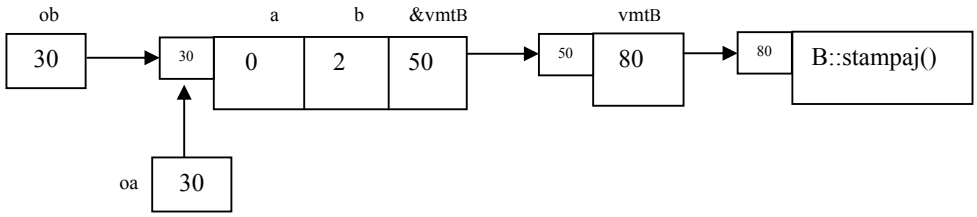
d) Ukoliko ispred članice natklase stoji ključna reč **private,** onemogućava se javan pristup do članica natklase iz svih klasa (izuzev same klase u kojoj se nalaze razmatrane članice; to znači da se članice jedne klase mogu međusobno pozivati nezavisno od toga koji je način pristupa vezan za njih) nezavisno od toga kom paketu one pripadaju.

4.3 Kompatibilnost objektnih tipova

Objekat natklase može da dobije referencu na bilo koji objekat koji pripada klasi koja je izvedena iz natklase.

```
class A
{int a;
 public static void main(String args[])
    {B ob=new B();
     A  oa;
     oa = ob; // Referentnom objektu a moze da se dodeli referenca na bilo koji
              // objekat koji pripada klasi koja je izvedena iz klase A. U ovom
              // slucaju objekat b pripada klasi B koja je izvedena iz klase A.
     oa.stampaj();} // Pozvace se metoda stampaj() klase B.
     A(){a=0;}
     void stampaj(){System.out.println("A je: " +a);} }

class B extends A
 {  int b;
    B(){b=2;}
    void stampaj(){System.out.println("A i B su: " +  a +" "+ b);  }}
```

Izgled operativne memorije:

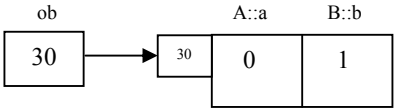


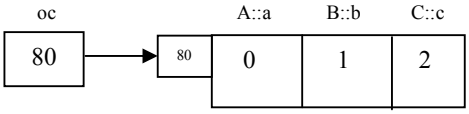
4.4 Rezervisana reč super

U Javi ključna reč **super** se koristi da ukaže na klasu koja je na prvom višem nivou hijerarhije od klase, koja u okviru neke od svojih metoda, koristi ključnu reč **super.**
Postoje dva slučaja korišćenja ključne reči super:
a) kada se želi pristupiti članicama nadređene klase
Opšti oblik pristupa članicama nadređene klase je:
super.članicanadklase

```
class A
{ int a;
    public static void main(String args[])
   {  B ob=new B(); ob.stampaj();
      C oc=new C(); oc.stampaj();
    }
    A(){ a=0;}
    void stampaj(){System.out.println(a);}
}

class B extends A
{ int b;
  ...
  B(){ b=1;}
  // Iz klase B kod poziva super.a poziva se A.a. Takodje smo mogli umesto super.a da
  // pozovemo super.stampaj().
void stampaj(){System.out.println("A i B su: " + super.a +" "+ b);  }}

class C extends B
 { int c;
  ...
  C(){ c=2;}
  // Iz klase C kod poziva super.b poziva se B.b.
  void stampaj(){System.out.println("A, B, C  su: " + a + " " + super.b +" "+ c);  }
}
```

Izgled operativne memorije:





b) kada se želi pristupiti konstruktoru nadređene klase
Opšti oblik pristupa konstruktoru nadređene klase je:
super(lista parametara) za parametrizirane konstruktore
ili
super() za default konstruktore

```
class A
{ int a;
    public static void main(String args[])
   { B ob=new B(); ob.stampaj();}
    A(){a=0;}
    A(int a1) { a = a1;}
    void stampaj() {System.out.println("A je: " +a);}}
```

```
class B extends A
{ int b;
   B(){super(); // za default konstruktor klase A  ili  super(5); za
                // parametrizirani konstruktor klase A.
       b=2;}
  void stampaj(){System.out.println("A i B su: " + super.stampaj() +"  "+ b);}
}
```

Naredba **super()** koja poziva konstruktor nadklase mora u konstruktoru tekuće klase da bude prva naredba.

```
B(){super(); // metoda super() je prva naredba konstruktora B().
     b=2;}
```

Ukoliko bi pokušali da pozovemo:

```
  B(){b=2;
      super(); // metoda super() nije prva naredba konstruktora.
    }
```

kod kompajliranja bi se javila greška: " *call to super must be first statement in constructor."*

Takođe, ukoliko bi pokušali iz neke ne konstruktor metode, da pozovemo preko naredbe super(), konstruktor nadređene klase, javila bi se kod kompajliranja ista poruka o grešci: "call to super …"

4.5 Pozivanje konstruktora u hijerarhiji klasa

Konstruktori, u hijerarhiji klasa, se pozivaju po redosledu izvođenja, od natklase ka potklasama. Naredba super() ukoliko postoji, kod eksplicitnog poziva konstruktora nadklase, ne utiče na redosled izvršenja konstruktora (jer se naredba super u kontruktoru tekuće klase uvek poziva kao prva naredba). Ukoliko naredba super() ne postoji izvršavaju se podrazumevani konstruktori klasa.

4.6 Redefinisanje metoda (override)

Koncept klase omogućava da u okviru jedne klase postoje istoimene metode (preklopljene – overloaded metode) koje se razlikuju po potpisu (broju i tipu argumenata). Koncept nasleđivanja klasa omogućava da izvedene klase mogu imati metode koje imaju isto ime i potpis kao i osnovne klase. Za takve metode se kaže da one prekrivaju (override) metode osnovne klase.

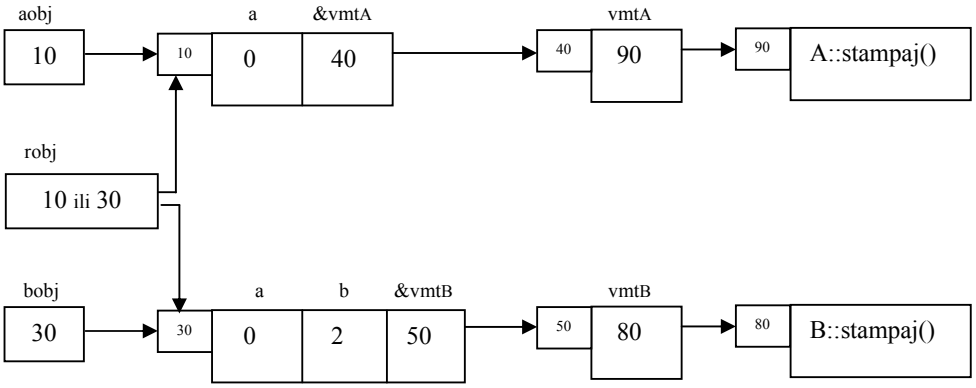
```
class A
{int a;
      public static void main(String args[])
    {B ob=new B();
     A  oa;
     oa = ob;
     oa.stampaj();} // Pozvace se metoda stampaj() klase B.
     A(){a=0;}
     void stampaj(){System.out.println("A je: " +a);}
}

class B extends A
{
…
// Ova metoda prekriva (override) metodu stampaj() klase A.
void stampaj(){ System.out.println("A i B su: " +  a +" "+ b);  }
}
```

4.7 Kasno povezivanje metoda

Kod programskog jezika C++ postoje obične i virtuelne metode. Obične metode se povezuju sa programom u vreme kompajliranja (early bindig), dok se virtuelene metode povezuju sa programom u vreme izvršenja programa (late binding), pomoću tabela virtuelnih metoda. Kod Jave sve su metode podrazumevano virtuelne, što znači da se one povezuju sa programom u vreme izvršenja programa. Kasno povezivanje metoda zajedno sa konceptom nasleđivanja i konceptom kompatibilnosti objektnih tipova omogućava jak polimorfizam.

```
import java.io.*;
class A
{ int a;
  public static void main(String args[]) throws IOException
 {  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    A aobj = new A();   B bobj=new B();
    A robj; // Referentni objekat
   char c;

   System.out.println("Unesi znak '1' ili '2': ");
   c= (char) br.read();
   // U vreme kompajliranja se ne zna sta ce biti dodeljeno do objekta raobj.
   if ( c == '1')   robj = aobj;
   else             robj = bobj;

    raobj.stampaj();} // Pozvace se metoda stampaj() od one klase ciji je objekat
                      // izabran u vreme izvrsenja programa.
    A(){a=0;}
    void stampaj(){System.out.println("A je: " +a);}
}
class B extends A
{ int b;
  B(){b=2;}
  void stampaj(){ System.out.println("A i B su: " +  a +" "+ b);  }
}
```

Izgled operativne memorije:



4.8 Rezervisana reč final kod definisanja klase

Ukoliko se ključna reč final koristi kod definicije klase, onemogućava se nasleđivanje takvih klasa.

```
final class A
{ ...}

class B extends A // Klasa B ne moze da nasledi klasu A jer je ista final.
 {...}
```

4.9 Klasa Object

Java sadrži klasu **Object** iz koje su izvedene sve druge klase. Object klasa je natklasa svih drugih klasa. *Na taj način je moguće definisati referentni objekat klase Object koji može da prihvati objekat bilo koje klase.*

4.10 Primeri - Nasleđivanje

Primer N1:

```
/*Programski zahtev: Pokazati odnos između nadklase i podklase na primeru klasa Osoba (natklasa) i
Student (potklasa ). Takođe pokazati kako se vrši prekrivanje metode Prikazi(). */

import java.util.*;

class Osoba
{ String MLB;
  String ImePrezime;
  String Adresa;
  Osoba(){MLB ="";ImePrezime="";Adresa="";}
  Osoba(String MLB1,String ImePrezime1,String Adresa1) {
  MLB=MLB1;ImePrezime=ImePrezime1;Adresa=Adresa1;}
  void Prikazi(){System.out.println("MLB: " + MLB + " Ime i prezime: " + ImePrezime + " Adresa:" +
                                        Adresa);}
 }

class Student extends Osoba
{ String BrojIndeksa; int Godina;
  Student(){BrojIndeksa="";Godina=0;}
  Student(String MLB1,String ImePrezime1,String Adresa1, String BrojIndeksa1, int Godina1)
      { super(MLB1,ImePrezime1,Adresa1); BrojIndeksa = BrojIndeksa1; Godina = Godina1;}
  void Prikazi(){ super.Prikazi();  System.out.println("Broj indeksa: " + BrojIndeksa + " Godina: " +
                                                        Godina);}}
class Glavna
 {  public static void main(String args[])
    {  Student k = new Student();
       k.Prikazi();
       Student k1 = new Student("B123434","Pera Peric","Beogradska 1","123-99",3);
       k1.Prikazi();
     }
}

// Rezultat:
//
// MLB: ImePrezime: Adresa:
// BrojIndeksa: Godina:
// MLB: B123434 ImePrezime: Pera Peric Adresa: Beogradska 1
// BrojIndeksa:123-99 Godina:3
```

Primer N2:

```
/*Programski zahtev: Pokazati nacin pristupa do clanica natklase Osoba  iz podklase Student)*/

import java.util.*;
class Osoba
{ public      String MLB;
  private     String ImePrezime;
  protected String Adresa;
              char Pol;

Osoba(String MLB1,String ImePrezime1,String Adresa1,char Pol1)
     { MLB=MLB1;ImePrezime=ImePrezime1; Adresa=Adresa1;Pol=Pol1;}
}
```

```
class Student extends Osoba
{ String BrojIndeksa;
  int Godina;

Student(String MLB1,String ImePrezime1,String Adresa1, String BrojIndeksa1, int Godina1, char
Pol1) { super(MLB1,ImePrezime1,Adresa1,Pol1); BrojIndeksa = BrojIndeksa1; Godina = Godina1;}
void Prikazi()
  {  System.out.println(MLB);
     // System.out.println(ImePrezime); Ne moze da pristupi jer je private
     System.out.println(Adresa);
     System.out.println(Pol);
     System.out.println(BrojIndeksa);
     System.out.println(Godina);
   }
}


class Glavna
 {
  public static void main(String args[])
    { Student k1 = new Student("B123434","Pera Peric","Beogradska 1","123-99",3,'M');
       k1.Prikazi();
       System.out.println(k1.MLB);
      // System.out.println(k1.ImePrezime); Ne moze da pristupi jer je private
       System.out.println(k1.Adresa);
       System.out.println(k1.Pol);
       System.out.println(k1.BrojIndeksa);
       System.out.println(k1.Godina);
    }}

// Rezultat:
//
// B123434
// Beogradska 1
// M
// 123-90
// 3
// B123434
// Beogradska 1
// M
// 123-90
// 3
```

Primer N3:

```
/*Programski zahtev: Pokazati kompatibilnost objektnih tipova na primeru klasa Osoba (natklasa)
Student (potklasa). */

import java.util.*;

class Osoba
{ String MLB;
  String ImePrezime;
  String Adresa;
  Osoba(String MLB1,String ImePrezime1,String Adresa1) {
 MLB=MLB1;ImePrezime=ImePrezime1;Adresa=Adresa1;}
  void Prikazi(){System.out.println("MLB: " + MLB + " Ime i prezime: " + ImePrezime + " Adresa:" +
  Adresa);}}

class Student extends Osoba
 { String BrojIndeksa;
    int Godina;
```

```
Student(String MLB1,String ImePrezime1,String Adresa1, String BrojIndeksa1, int Godina1)
      { super(MLB1,ImePrezime1,Adresa1); BrojIndeksa = BrojIndeksa1; Godina = Godina1;}
void Prikazi(){ super.Prikazi(); System.out.println("Broj indeksa: " + BrojIndeksa + " Godina: " +
Godina);}
}

class Glavna
 {  public static void main(String args[])
    {  Osoba os;
       Student k1 = new Student("B123434","Pera Peric","Beogradska 1","123-99",3);
       os = k1; //Kompatibilnost objektnih tipova
       os.Prikazi(); //Poziva se metoda Prikazi klase Student.    }
}

// Rezultat:
//
// MLB: B123434 ImePrezime: Pera Peric Adresa: Beogradska 1
// BrojIndeksa:123-99 Godina:3
```

Primer N4:

```
/*Programski zahtev: Pokazati pozivanje kostruktora u hijerarhiji klasa  na primeru klasa Osoba
(natklasa) - Student (potklasa). */

import java.util.*;
class Osoba
 { String MLB;
   String ImePrezime;
   String Adresa;
   Osoba(){System.out.println("Konstruktor - osoba - default");}
   Osoba(String MLB1,String ImePrezime1,String Adresa1)
     { System.out.println("Konstruktor - osoba - param.");
      MLB=MLB1;ImePrezime=ImePrezime1;Adresa=Adresa1;}
 }

class Student extends Osoba
 { String BrojIndeksa;
    int Godina;
   Student(){System.out.println("Konstruktor - student - default");}
   Student(String MLB1,String ImePrezime1,String Adresa1, String BrojIndeksa1, int Godina1)
       { super(MLB1,ImePrezime1,Adresa1);
         System.out.println("Konstruktor - Student - param.");
         BrojIndeksa = BrojIndeksa1; Godina = Godina1;
       }
}

class Glavna
 {  public static void main(String args[])
    {   Student k = new Student();
        Student k1 = new Student("B123434","Pera Peric","Beogradska 1","123-99",3);
    }
}

// Rezultat:
//
// Konstruktor - osoba – default
// Konstruktor - student - default
// Konstruktor - osoba - param.
// Konstruktor - Student - param.
```

Primer N5:

```
/*Programski zahtev: Pokazati kako se vrsi kasno pozivanje metoda na primeru metode Prikazi koja se
nalazi u klasama Osoba (natklasa) i Student (potklasa). */

import java.util.*;

class Osoba
{ String MLB;
  String ImePrezime;
  String Adresa;
  Osoba(String MLB1,String ImePrezime1,String Adresa1) {
  MLB=MLB1;ImePrezime=ImePrezime1;Adresa=Adresa1;}
  void Prikazi(){System.out.println("MLB: " + MLB + " Ime i prezime: " + ImePrezime + " Adresa:" +
  Adresa);}
}

class Student extends Osoba
 { String BrojIndeksa;
    int Godina;
    Student(String MLB1,String ImePrezime1,String Adresa1, String BrojIndeksa1, int Godina1)
       { super(MLB1,ImePrezime1,Adresa1); BrojIndeksa = BrojIndeksa1; Godina = Godina1;}
    void Prikazi(){ super.Prikazi();    System.out.println("Broj indeksa: " + BrojIndeksa + " Godina: " +
                                                          Godina);}
}

class Glavna
 {  public static void main(String args[])
    {  Osoba o;
       Osoba o1 = new Osoba("C345345","Milan Savic","Niska 5");
       Student k = new Student("B123434","Pera Peric","Beogradska 1","123-99",3);

      for (int i=1;i<3;i++)
        { if (i % 2 ==0) // % vraca ostatak od deljenja.
             o = o1;
          else
             o=k;
          o.Prikazi();
      }
   }
}

// Rezultat:
//
// MLB: B123434 ImePrezime: Pera Peric Adresa: Beogradska 1
// BrojIndeksa:123-99 Godina:3
// MLB: C345345 ImePrezime: Milan Savic Adresa: Niska 5
```

4.11 Zadaci - Nasleđivanje

Zadatak NZ1: Nacrtati izgled operativne memorije za svaki od navedenih primera.
Zadatak NZ2: Pokazati odnos između nadklase i podklase na primeru klasa Projekat (natklasa) i Zadatak(potklasa). Takođe pokazati na kako se vrši prekrivanje metode Prikazi().
Zadatak NZ3: Pokazati nacin pristupa do clanica natklase Projekat iz podklase Zadatak.
Zadatak NZ4: Pokazati kompatibilnost objektnih tipova na primeru klasa Projekat(natklasa) - Zadatak(potklasa).
Zadatak NZ5: Pokazati pozivanje konstruktora u hijerarhiji klasa na primeru klasa Projekat(natklasa) - Zadatak(potklasa).
Zadatak NZ6: Pokazati kako se vrsi kasno pozivanje metoda na primeru metode Prikazi koja se nalazi u klasama Projekat(natklasa) i Zadatak(potklasa).

Pitanja:

1. Šta je nasleđivanje?
2. Kakav je način pristupa članici klase ako ispred nje stoji protected?
3. Kakav je način pristupa članici klase ako ispred nje stoji private?
4. Šta je kompatibilnost objektnih tipova?
5. Kada se koristi rezervisana reč super?
6. Kako se pozivaju konstruktori u hijerarhiji klasa?
7. Šta znači prekrivanje ili redefinisanje metoda?
8. Koja je razlika između ranog i kasnog povezivanja metoda?

Tematska jedinica	**Nasleđivanje**	
Datum:		
Zadatak		
Ime i prez. studenta – br. ind.		
Laborant:		

5. Apstraktne klase

Apstraktne klase se definišu na isti način kao i obične klase, ali one za razliku od običnih klasa ne mogu da imaju pojavljivanja. Apstraktne klase se javljaju u slučaju postojanja potrebe da klasa ima bar jednu apstraktnu metodu[11]. Klasa koja ima jednu ili više apstraktnih metoda mora da se definiše kao apstraktna klasa. Apstraktna metoda nema telo i odgovornost realizacije te metode se prenosi do metode klase koja je nasledila apstraktnu klasu, u kojoj se nalazi ta apstraktna metoda. Ispred apstraktne klase i apstraktnih metoda se stavlja ključna reč **abstract.**

Apstraktne klase i metode se definišu na sledeći način:
a) abstract class ime_klase {…} - apstraktne klase
b) abstract tip ime_metode(lista parametara) - apstraktne metode

Abstraktna klasa ne može imati apstraktne konstruktore niti statičke metode. Potklase apstraktne klase moraju realizovati sve apstraktne metode natklase. U suprotnom one takođe postaju apstraktne klase.
Abstraktne klasa mogu da imaju main() metodu. Samim tim program može da se izvrši pozivom apstraktne klase.

```
abstract class A
{ int a;
      public static void main(String args[])
   { A oa = new A(); // ne moze da se kreira jer je klasa A abstraktna.
     B ob=new B();
    A oa1; // jedino moze da se definise referentni objekat apstraktne klase.
    oa1 = ob; //  referentni objekat apstraktne klase moze da dobije referencu
              // objekta konkretne klase.
   oa1.stampaj();} // referentni objekat apstraktne klase moze da pozove
                   // clanice objekta konkretne klase.
    A(){a=0;}
   abstract  void stampaj(); // apstratna metoda nema telo.
}
```

[11] Apstraktne klase ne moraju da imaju ni jednu apstraktnu metodu.

```
class B extends A
 {
int b;
B(){b=2;}
// navedena metoda predstavlja realizaciju apstraktne klase stampaj iz
// apstrakne klase A.
void stampaj(){ System.out.println("A i B su: " +  a +" "+ b);  }
}
```

5.1 Rezervisana reč final kod metoda klasa

Ukoliko se ključna reč final koristi kod definicije metode, onemogućava se prekrivanje takvih metoda u izvedenim klasama.

```
class A
{ ...
   // Definisanje final metode koja se ne moze prekriti.
  final   void stampaj(){System.out.println("A je: " +a);}
}
class B extends A
 {...
// Ova metoda se ne moze prekriti, jer je metoda stampaj() klase A definisana kao final.
void stampaj(){ System.out.println("A i B su: " +  a +" "+ b);  }
}
```

Kod poziva final metode iz objekta koji pripada klasi, u kojoj je definisana final metoda, pri kompaliranju, u program se neposredno ugrađuje kod koji se nalazi u telu final metode. Sličan postupak se dešava kod inline metoda u C++ i makroa kod C programskog jezika. Navedeni postupak se ne može izvesti, ukoliko se final metoda poziva iz referentnog objekta koji pripada klasi koja je iznad u hijerarhiji u odnosu na klasu kojoj pripada final metoda.

```
import java.io.*;
class  A
šint a;
public static void main(String args[])
throws IOException
{BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
          A oa=new A();
          B ob=new B();
          A a1;
          char d;

          System.out.println("Unesi znak '1' ili '2': ");
          d=(char)br.read();
          if (d=='1')
               a1=oa;
          else
              a1=ob;
          a1.stampaj(); // Referentni objekat a1 u vreme kompajliranja ne zna ko ce mu
// biti dodeljen, objekat oa ili objekat ob, odnosno ne zna da li ce biti povezan sa
// obicnom  metodom stampaj() – klase A ili final metodom stampaj() -  klase B.
        ob.stampaj();  // U vreme kompajliranja se zna da objekat b poziva final metodu
                       // stampaj(). Tada je moguce da telo metode:
                       // System.out.println("A i B su: "+ a +" "+ b);
                       // bude neposredno  ugradjeno u kod programa kod kompaliranja.
 }
A(){a=0;}
void stampaj(){System.out.println("A je:" +a);}}
```

```
class B extends A
{int b;
B(){b=2;}
final void stampaj(){System.out.println("A i B su: "+ a +" "+ b);}
}
```

5.2 Primeri - Apstraktne klase

Primer AK1:

```
/*Programski zahtev: Dati primer Osobe kao Apstraktne klase i Studenta kao konkretne klase.*/

import java.util.*;

abstract class Osoba
 { String MLB;   String ImePrezime;   String Adresa;
  Osoba(String MLB1,String ImePrezime1,String Adresa1) {
  MLB=MLB1;ImePrezime=ImePrezime1;Adresa=Adresa1;}
  abstract void Prikazi();
}
class Student extends Osoba
{ String BrojIndeksa;
  int Godina;
  Student(String MLB1,String ImePrezime1,String Adresa1, String BrojIndeksa1, int Godina1)
      { super(MLB1,ImePrezime1,Adresa1); BrojIndeksa = BrojIndeksa1; Godina = Godina1;}
  void Prikazi(){  System.out.println("MLB: " + MLB + " Ime i prezime: " + ImePrezime + " Adresa:"
                  + Adresa + "Broj indeksa: " + BrojIndeksa + " Godina: " + Godina);}
}

class Glavna
 {  public static void main(String args[])
    {
       // Osoba o = new Osoba("C345345","Milan Savic", "Niska 5"); Ne moze da ima pojavljivanja.
        Student k1 = new Student("B123434","Pera Peric","Beogradska 1","123-99",3);
       k1.Prikazi();

    }
}
// Rezultat:
//
// MLB: B123434 ImePrezime: Pera Peric Adresa: Beogradska 1
// BrojIndeksa:123-99 Godina:3
```

5.3 Zadaci - Apstraktne klase

Zadatak AKZ1:

Dati primer Projekta kao Apstraktne klase i Zadatka kao konkretne klase.

Pitanja:

1. Da li apstraktne klase mogu da imaju pojavljivanja?
2. Da li metode konkretnih klasa moraju da realizuju sve metode apstraktnih klasa ili jednu od njih? Šta se dešava ukoliko ne realizuju sve metode?
3. Da li apstraktne klase mogu da imaju main metodu? Da li main metoda kod apstraktnih klasa može da se izvršava?

Tematska jedinica	**Apstraktne klase**	
Datum:		
Zadatak		
Ime i prez. studenta – br. ind.		
Laborant:		

6. Interfejsi

Interfejs je koncept koji razdvaja specifikaciju metoda od njihove implementacije.

6.1 Definisanje interfejsa

Opšti oblik:
nacin_pristupa interface ImeInterfejsa
{
deklaracija konstanti;
potpisi metoda;
}
nacin_pristupa može biti public ili ga nema.

6.2 Realizacija interfejsa

Interfejs, odnosno njegove metode se implementiraju pomoću klasa.

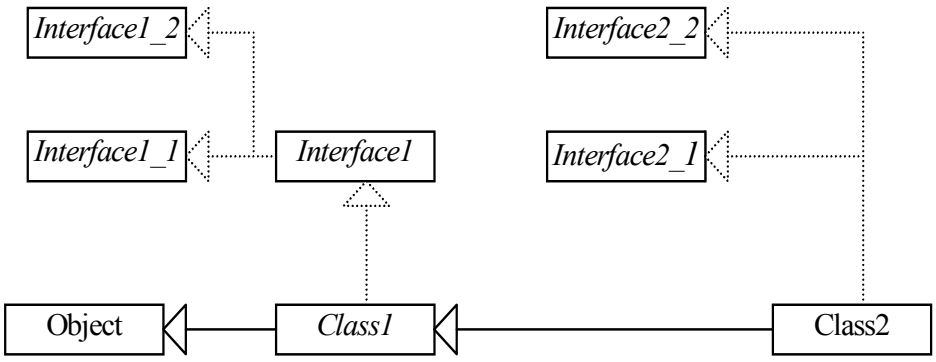
Opšti oblik:
nacin_pristupa class imeklase implements ime_interfejsa
{ deklaracija konstanti;
deklaracija atributa;
metode koje implementiraju metode iz interfejsa
}

nacin_pristupa može biti public ili ga nema.
Kada se implementiraju metode intefejsa one se deklarišu kao public.

6.3 Interfejsi i apstraktne klase

- Kod interfejsa se mogu definisati samo konstante (ne atributi). Apstraktna klasa može da ima atribute.
- Svaka metoda u interfejsu ima samo potpis bez implementacije. Apstraktna klasa može da ima konkretne metode.
- Pošto su sve metode definisane u interfejsu apstraktne metode, Java ne zahteva da se ispred imena metode stavi ključna reč abstract.
- Kod apstraktnih metoda apstraktnih klasa mora se staviti ključna reč abstract, jer apstraktne klase mogu da imaju i konkretne metode.
- Metoda koja realizuje metodu iz interfejsa mora biti javna.
- Interfejs i apstraktna klasa ne mogu imati svoja pojavljivanja.

6.4 Odnos između klasa i interfejsa



Postoje sledeća pravila kod nasleđivanja:
- ✓ Interfejs može da nasledi više interfejsa.
- ✓ Klasa može da nasledi više interfejsa.
- ✓ Klasa ne može da nasledi više klasa. Klasa može da nasledi samo jednu klasu.
- ✓ Interfejs ne može da nasledi klasu.

6.5 Primeri - Interfejsi

Primer I1:

```
/*Programski zahtev: Dati primer Osobe kao Interfejsa i Studenta kao konkretne klase.*/

import java.util.*;
interface Osoba{
void Napuni(String MLB1,String ImePrezime1,String Adresa1, String BrojIndeksa1, int Godina1);
void Prikazi();
}

class Student implements Osoba{
String MLB;
String ImePrezime;
String Adresa;
String BrojIndeksa;
int Godina;
public void Napuni(String MLB1,String ImePrezime1,String Adresa1, String BrojIndeksa1, int
Godina1)
      { MLB=MLB1; ImePrezime=ImePrezime1; Adresa = Adresa1; BrojIndeksa = BrojIndeksa1;
Godina = Godina1;}
public void Prikazi(){ System.out.println("MLB: " + MLB + " ImePrezime: " + ImePrezime + "
Adresa:" + Adresa);  System.out.println("Broj indeksa: " + BrojIndeksa + " Godina: " + Godina);}
}

class Glavna
 {  public static void main(String args[])
   {    // Osoba o = new Osoba("C345345","Milan Savic", "Niska 5"); Ne moze da ima pojavljivanja.
      Student k1 = new Student();
      k1.Napuni("B123434","Pera Peric","Beogradska 1","123-99",3);
      k1.Prikazi();
   }}

// Rezultat:
//
// MLB: B123434 ImePrezime: Pera Peric Adresa: Beogradska 1
// BrojIndeksa:123-99 Godina:3
```

6.6 Zadaci - Interfejsi

Zadatak IZ1: Dati primer realizacije interfejsa Projekat.

Pitanja:

1. Da li interfejs može da nasledi klasu?
2. Da li interfejs može da ima atribute?
3. Kakav je način pristupa do metode koja realizuje metodu iz interfejsa?

Tematska jedinica	**Interfejsi**	
Datum:		
Zadatak		
Ime i prez. studenta – br. ind.		
Laborant:		

7. Rad sa stringovima

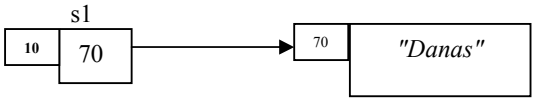
Stringovi predstavljaju niz ili sekvencu znakova (npr. reč "Danas"). String ne predstavlja predefinisani osnovni (primitivni) tip, kao što su int, char, boolean, float,... Umesto toga standardna Javina biblioteka sadrži klasu *String*.

7.1 Deklaracija i inicijalizacija promenljive tipa String

Deklaracija, kreiranje i inicijalizacija promenljive tipa string se radi na sledećih nekoliko načina:

- Deklaracija, kreiranje i inicijalizacija objekta na koji ukazuje promenljiva, sa zadom vrednošću.

String s1 = "Danas"; // Istovremeno se vrši i deklaracija promenljive s1 i kreiranje objekta (na adresi 70) na koji će pokazivati promenljiva s1 i inicijalizacija objekta na vrednost "Danas".

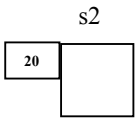


- Deklaracija, kreiranje i inicijalizacija objekta na koji ukazuje promenljiva, pomoću *copy* konstruktora.

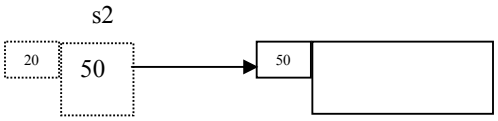
String s2 = new String("Zdravo");

Navedena naredba može se razložiti na sledeće tri naredbe (boldovane i podvučene):

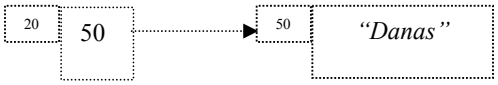
a) **String s2** = new String("Danas");; // Deklaracija promenljive. Alocira se prostor (na adresi 20) za promenljivu s2, koja će dobiti referencu (pokazivač) na objekat.



b) String **s2= new String**("Danas"); // konstruktor kreira objekat i vraća adresu (50) kreiranog objekta do s2.

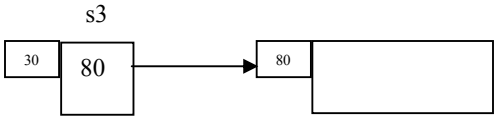


c) String s2 = new String **("Danas")**; // objekat na adresi 50 dobija vrednost "Danas".

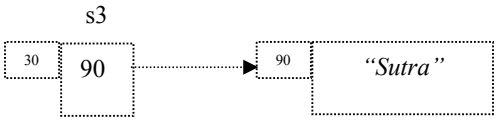


- Deklaracija, kreiranje i inicijalizacija objekta na koji ukazuje promenljiva, pomoću *default* konstruktora. i naknadno dodeljivanje vrednosti do objekta na koji ukazuje promenljiva

  String s3 = new String();

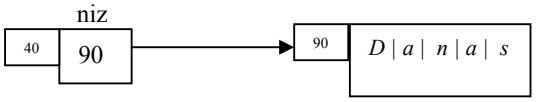


  s3 = "Sutra";

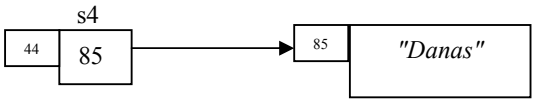


- Deklaracija, kreiranje i inicijalizacija objekta na koji ukazuje promenljiva, sa nizom znakova pomoću *copy* konstruktora.

  char niz[] = { 'D','a','n','a','s'};

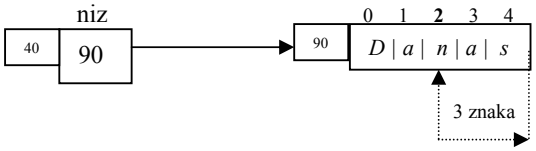


  String s4 = new String(niz);

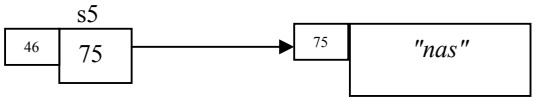


- Deklaracija, kreiranje i inicijalizacija objekta na koji ukazuje promenljiva, sa podnizom niza znakova pomoću *copy* konstruktora.

  char niz[] = { 'D','a','n','a','s'};

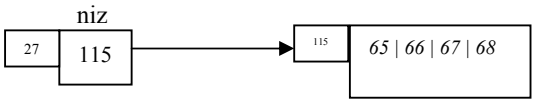


String s5 = new String(niz,2,3); // *s5* se inicijalizuje od treće pozicije promenljive *niz* sa 3 znaka ("nas")

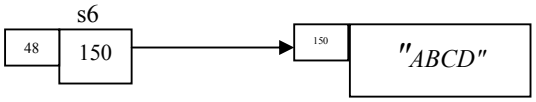


- Deklaracija, kreiranje i inicijalizacija objekta na koji ukazuje promenljiva, sa nizom bajtova pomoću *copy* konstruktora.

  byte asci[] = {65,66,67,68};

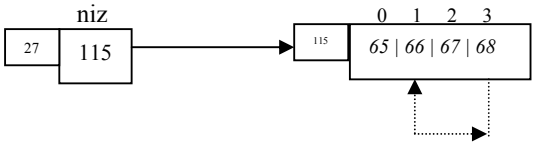


  String s6 = new String(asci);

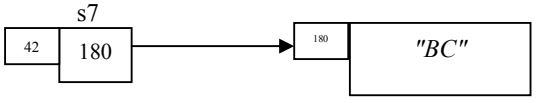


- Deklaracija, kreiranje i inicijalizacija objekta na koji ukazuje promenljiva, sa podnizom niza bajtova pomoću *copy* konstruktora.

  byte asci[] = {65,66,67,68};



  String s7 = new String(asci,1,2);



Primer (*String1.java*) koji prati navedena objašnjenja ima sledeći izgled:

```
class String1
{
  public static void main(String args[])
 {
    String s1 = "Zdravo";
    System.out.println(s1);

    String s2 = new String("Zdravo");
    System.out.println(s2);

    String s3 = new String();
    s3 = "Sutra";
    System.out.println(s3);

    char niz[] = { 'D','a','n','a','s'};
    String s4 = new String(niz);
    System.out.println(s4);

    String s5 = new String(niz,2,3);
    System.out.println(s5);

    byte asci[] = {65,66,67,68};
    String s6 = new String(asci);
    System.out.println(s6);

    String s7 = new String(asci,1,2);
    System.out.println(s7); }}

// Rezultat:
//
// Zdravo
// Zdravo
// Zdravo
// Danas
// nas
// ABCD
// BC
```

Zadatak STZ1: Nacrtati izgled operativne memorije za sledeći zadatak:

```
class String2
{  public static void main(String args[])
 {
    String s1 = "Danas";
    String s2 = new String("Sutra");
    char niz[] = { 'p',r',e',};
    String s3 = new String(niz,1,2);
    String s4=s3;
    String s5=s2;
    System.out.println(s1);
    System.out.println(s4);
    System.out.println(s5);
 }
}
```

Šta će biti prikazano na standardnom izlazu?

7.2 Dobijanje novog na osnovu postojećeg stringa

7.2.1 Nadovezivanje znakovnih nizova (concatenation)

Java koristi znak '+' kada vrši povezivanje 2 stringa. String može da se poveže sa vrednošću bilo kog predefinisanog tipa (te vrednosti se tada konvertuju u niz znakova). Vrednosti predefinisanih tipova ne mogu formirati string ukoliko u njihovom nadovezivanju ne postoji bar jedan String.

String s1 = "Danas";
String s2 = " je lep dan";
String s3 = s1 + s2;

Primer (String21.java):

```
class String21
{
  public static void main(String args[])
 {
            String s1 = "Danas";
            String s2 = " je lep dan";
            String s3 = s1 + s2;
            System.out.println("s3 je: " + s3);
            //String s4 = "Danas je" + petnesti + " dan meseca";  // ne moze da kompajlira
            // petnesti jer isti shvata kao promenljivu.
            String s4 = "Danas je " + 15 + " dan meseca";
            System.out.println(s4);
            int pp = 10;
            String s5 = "Danas je " + pp + " dan meseca";
            System.out.println(s5);
            String s6 = "Broj " + 15.23 + " je realan broj";
            System.out.println(s6);
            String s7 = "Logicka vrednost je " + true;
            System.out.println(s7);
            String s8 = "Ovo je znak " + 'A';
            System.out.println(s8);
            // String s8 = 4 + 1 + 2 + 3; // ne moze da se kompajlira jer su nekompatibilni tipovi.
            String s9 = 4 + "1" + 2 + 3;
            System.out.println(s9);
            String s10 = 4 + "1" + (2 + 3);
            System.out.println(s10);

 }
}

// Rezultat:
//
// s3 je: Danas je lep dan
// Danas je 15 dan meseca
// Danas je 10 dan meseca
// Broj 15.23 je realan broj
// Logicka vrednost je true
// Ovo je znak A
// 4123
// 415
```

7.2.2 Dobijanje podstringova

Iz nekog stringa moguće je izdvojiti podstring korišćenjem metode *substring()*. Postoje dva oblika metode *substring():*

1. *String substring (int pocetniIndeks); //* pocetniIndeks ukazuje na mesto u nizu od koga pocinje podniz koji ce biti izdvojen. Takav podniz ide do kraja niza iz koga se izdvaja.

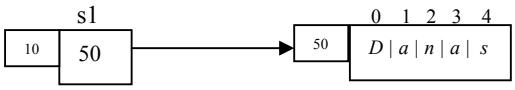
2. String substring (int pocetniIndeks, krajnjiIndeks); // pocetniIndeks ukazuje na mesto u nizu od koga pocinje podniz koji ce biti izdvojen. Dok krajnjiIndeks ukazuje na mesto u nizu na kome se zavrsava podniz koji ce biti izdvojen. Zavrsetak podniza ne ukljucuje mesto na koje ukazuje krajnjiIndeks.

Početak niza kreće od 0 indeksa.

Na primer (String22.java):

```
class String22
{
 public static void main(String args[])
 {
        String s1 = "Danas";
        String s2 = s1.substring(0,3);  // Dan
        System.out.println(s2);
        String s3 = s1.substring(2,3); // n
        System.out.println(s3);
        String s4 = s1.substring(2,4); // na
        System.out.println(s4);
        String s5 = s1.substring(2); // nas
        System.out.println(s5);
 }
}
// Rezultat:
//
// Dan
// n
// na
// nas
```

Izgled operativne memorije:



Zadatak STZ2: Napisati program koji će iz stringa koji ima vrednost “Danas je lep dan.” izdvojiti sledeće stringove:

a) “je lep”
b) “ dan”
c) “dan”
d) “nas”
e) “p”
f) "Danas je lep"
f) "."

Rešenje zadatka STZ2:

```
class StringZ1
{
 public static void main(String args[])
 {
        String s1 = "Danas je lep dan.";
        String s2 = s1.substring(6,12);
        System.out.println(s2);
        String s3 = s1.substring(12,16);
        System.out.println(s3);
        String s4 = s1.substring(13,16);
        System.out.println(s4);
        String s5 = s1.substring(2,5);
        System.out.println(s5);
```

```
        String s6 = s1.substring(11,12);
        System.out.println(s6);
        String s7 = s1.substring(0,12);
        System.out.println(s7);
        String s8 = s1.substring(16);
        System.out.println(s8);
        String s9 = s1.substring(16,17);
        System.out.println(s9);
}
```

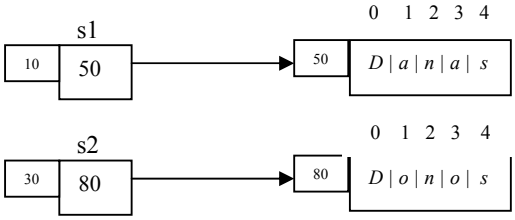
7.2.3 Promena nekog znaka sa drugim u stringu

Promena nekog znaka sa drugim u stringu se radi pomoću metode:
String replace(char kojiSeMenja, char saKojimSeMenja);

Na primer (String23.java):

```
class String23
{
  public static void main(String args[])
 {
        String s1 = "Danas";
        String s2=s1.replace('a','o');
        System.out.println(s1);
        System.out.println(s2);
 }
}

// Rezultat:
// Danas
// Donos
```

Izgled operativne memorije:



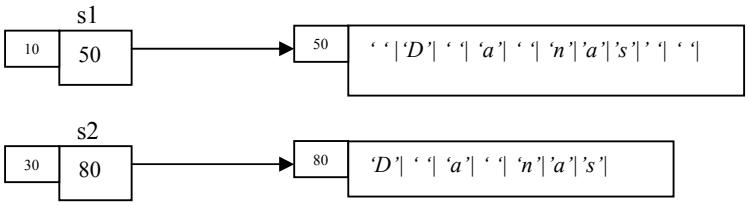
7.2.4 Brisanje praznih mesta sa početka i kraja stringa

Brisanje praznih mesta sa početka i kraja stringu se radi pomoću metode:
String trim();

Na primer (String24.java):

```
class String24
{  public static void main(String args[])
 {
        String s1 = " D a nas  ";
        String s2=s1.trim();
        System.out.println("String: " + s1 + " ima duzinu: " + s1.length());
        System.out.println("String: " + s2 + " ima duzinu: " + s2.length());
 }}

// Rezultat:
// String: D a nas   ima duzinu: 10
// String:D a nas ima duzinu: 7
```

Izgled operativne memorije:



7.2.5 Pretvaranje velikih u mala slova kod stringa i obrnuto

Pretvaranje svih velikih u mala slova se radi pomoću metode:
String toLowerCase();

Pretvaranje svih malih u velika slova se radi pomoću metode:
String toUpperCase();

Na primer (String25.java):

```
class String25
{   public static void main(String args[])
   { String s1 = "Danas";
     System.out.println(s1);

     String s2=s1.toLowerCase();
     System.out.println(s2);

     String s3=s1.toUpperCase();
     System.out.println(s3);

 }}
// Rezultat:
//
// Danas
// danas
// DANAS
```

7.3 Određivanje dužine stringa

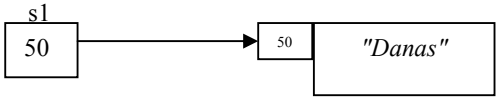
Dužina stringa se određuje pomoću metode:
int length();

Na primer (String3.java):

```
class String3
{
  public static void main(String args[])
  {
            String s1 = "Danas";
            System.out.println("String s1 ima duzinu: " + s1.length());
 }}
// Rezultat:
//
// String s1 ima duzinu: 5
```

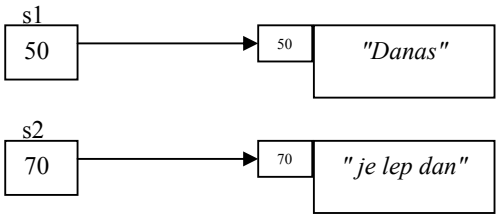
7.4 Promena vrednosti stringa

Kada se promenljivoj tipa String, npr. s1 doda neka vrednost:
String s1 = "Danas";
tada s1 dobija referencu (adresu) mesta gde se nalazi vrednost *"Danas"*.

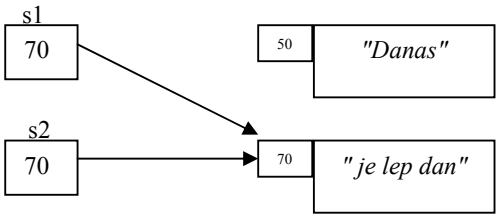


Sadržaj na koji pokazuje s1 ne može se menjati. Ono što se može menjati jeste sadržaj od s1, odnosno referenca koja ukazuje na neki drugi memorijski prostor.
Npr. za:

String s1 = "Danas";
String s2 = " je lep dan";

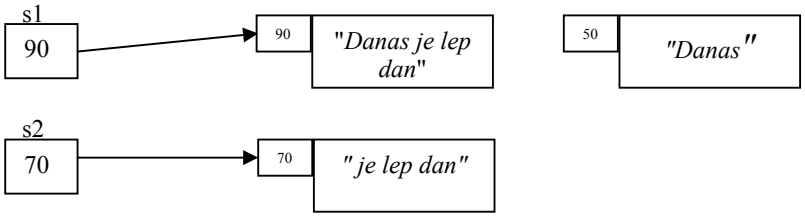


može se izvršiti naredba:
s1 = s2;



koja će s1 da dodeli sadržaj od s2. To znači da će i s1 i s2 da pokažu na istu memorijsku adresu.
Ukoliko se želi da s1 dobije vrednost *"Danas je lep dan"* potrebno je da se izvrši sledeće:

s1 = "Danas" + *s2*;
ili
s1 = "Danas" + *s1*;
Kao rezultat se dobija:

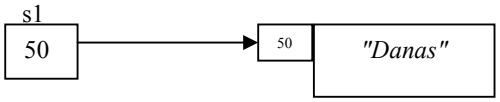


Možemo primetiti da je s1 dobilo adresu novog memorijskog prostora (adresa 90) na kome se nalazi vrednost *"Danas je lep dan"*.
Još jednom ponavljamo da ne postoji način u Javi, da se vrednost, na koji ukazuju s1 i s2. promene.

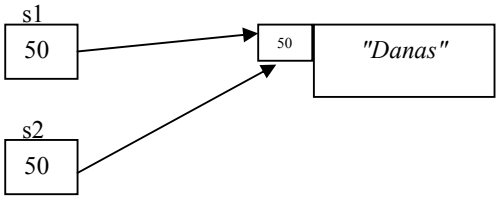
U sledećem primeru (String4.java):

```
class String4
{  public static void main(String args[])
   { String s1 = "Danas";
     String s2 = s1;
     s1 = s1 + " je lep dan";
    System.out.println(s1);
    System.out.println(s2);
 }}

// Rezultat:
//
// Danas je lep dan
// Danas
```

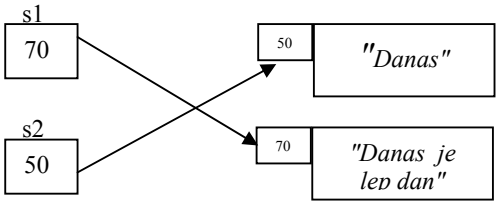
dešava se sledeće:
1. String s1 = "Danas";



2. String s2 = s1;



3. s1 = s1 + " je lep dan";



4. System.out.println(s1);
Na ekranu će se štampati: *Danas je lep dan*

5. System.out.println(s2);
Na ekranu će se štampati: *Danas*

Zadatak STZ3:

Na osnovu niže navedenog programskog koda (String41.java) nacrtajte izgled operativne memorije i dajte izlazni rezultat:

```
class String41
{  public static void main(String args[])
   {  String s1 = "Danas";
      String s2 = s1 + " je lep";
      String s3 = s2 ;
      String s4 = s1;
      String s5 = " dan";
      s1 = s2 + s5;
      s2 = s4 + " je petak";
      System.out.println(s1);
      System.out.println(s2);
      System.out.println(s3);
      System.out.println(s4);
      System.out.println(s5);
 }}

// Rezultat:
//
// Danas je lep dan
// Danas je petak
// Danas je lep
// Danas
//  dan
```

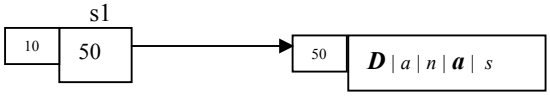
7.5. Pretraživanje znakova u stringu

7.5.1 Pozicioniranje na znakove u stringu na osnovu indeksa

Na određenu poziciju u stringu može se doći korišćenjem metode:
char charAt(int indeks);
Na primer:

```
char z1;
String s1 = “Danas”;
z1 = s1.charAt(0);
System.out.println(z1); // rezultat ce biti znak ‘D’.
z1 = s1.charAt(3);
System.out.println(z1); // rezultat ce biti znak ‘a’.
```

Izgled operativne memorije:



7.5.2 Pronalaženje znakova u stringu na osnovu vrednosti

Pronalaženje znakova u stringu se radi preko sledećih metoda:

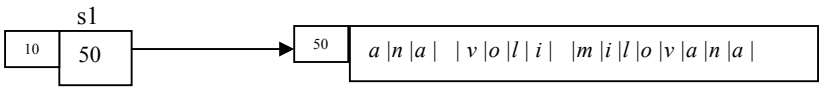
a) *int indexOf(int ZnakKojiseTrazi)*; // Vraća poziciju prvog znaka u stringu koji pronađe.
b) int lastIndexOf(int ZnakKojiseTrazi); // Vraća poziciju zadnjeg znaka u stringu koji pronađe.
c) *int indexOf(String NizkKojiseTrazi)*; // Vraća poziciju prvog podniza u stringu koji pronađe.
d) int lastIndexOf(String NizkKojiseTrazi); // Vraća poziciju zadnjeg podniza u stringu koji pronađe.
e) *int indexOf(int ZnakKojiseTrazi, int IndeksPocetka)*; // Vraća poziciju prvog znaka u stringu koji pronađe. Pretraživanje u stringu počinje od pozicije *IndeksPocetka* do kraja niza.
f) int lastIndexOf(int ZnakKojiseTrazi, int IndeksZavrsetka); // Vraća poziciju zadnjeg znaka u stringu koji pronađe. Pretraživanje u stringu počinje od početka niza do pozicije *IndeksZavrsetka.*
g) *int indexOf(String NizkKojiseTrazi, int IndeksPocetka)*; // Vraća poziciju prvog podniza u stringu koji pronađe.Pretraživanje u stringu počinje od pozicije *IndeksPocetka* do kraja niza.
h) int lastIndexOf(String NizkKojiseTrazi, int IndeksZavrsetka); // Vraća poziciju zadnjeg podniza u stringu koji pronađe. Pretraživanje u stringu počinje od početka niza do pozicije *IndeksZavrsetka.*

Na primer (String52.java):

```
class String52
{
  public static void main(String args[])
 {
    String s1 = "ana voli milovana"; // od 0 do 16 pozicije.
    System.out.println(s1.indexOf('a'));
            System.out.println(s1.lastIndexOf('a'));
            System.out.println(s1.indexOf("ana"));
            System.out.println(s1.lastIndexOf("ana"));

            System.out.println(s1.indexOf('i',8));
            System.out.println(s1.lastIndexOf('a',10));
            System.out.println(s1.indexOf("ana",5));
            System.out.println(s1.lastIndexOf("ana",5));

 }}

// Rezultat:
//
// 0
// 16
// 0
// 14
```

```
// 10
// 2
// 14
// 0
```

Izgled operativne memorije:



7.6 Kopiranje stringa u niz

Kopiranje niza u string se radi pomoću 2 metode:
1. Direktno kopiranje stringa u niz pomoću:
char [] toCharArray();

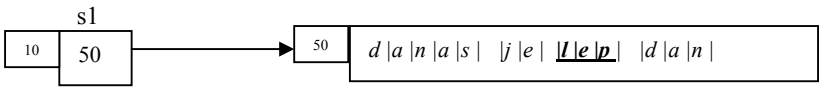
Na primer (String61.java):

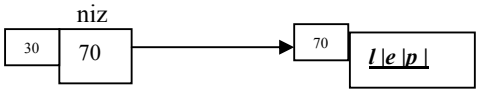
```
class String61
{  public static void main(String args[])
   { String s1 = "Danas";
     char niz[] = s1.toCharArray();
     // Moze i ovako:
     // char niz[];
     // niz = s1.toCharArray();
     System.out.println("Niz: " + String.valueOf(niz));
     // System.out.println("Niz: " + niz); Ovako ne moze
 }}

// Rezultat:
//
// Niz: Danas
```

2. Kopiranje stringa, od početne pozicije *pocIndeksStr* do pozicije *zavIndStr* (ne uključujući tu poziciju) u niz koji počinje od pozicije *pocIndeksNiza* sa metodom:
void getChars(int pocIndeksStr, int zavIndStr, char niz[], int pocIndeksNiza);

```
class String62
{   public static void main(String args[])
    { String s1 = "Danas je lep dan";
      char niz[] = new char[3]; // Mora da se alocira memorijski prostor za niz.
      s1.getChars(9,12,niz,0);
      System.out.println(niz);
    }
}

// Rezultat:
//
// lep
```

Izgled operativne memorije:





7.7. Poređenje stringova

7.7.1 Ispitivanje jednakosti stringova

Ukoliko se želi ispitati da li su dva stringa jednaka koristi se metoda:
boolean equals(Object str);

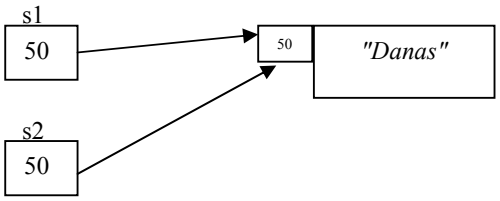
Ukoliko se želi ispitati da li su dva stringa jednaka, neuzimajući u obzir razliku između velikih i malih slova, koristi se metoda:
boolean equalsIgnoreCase(String str);

Na primer (String71.java):

```
class String71
{
 public static void main(String args[])
 {
        String s1 = "Danas";
        String s2 = "Danas";
        boolean signal = s1.equals(s2);
        System.out.println("Dva stringa su jednaka: " + signal);

        s2 = "DANAS";
        signal = s1.equals(s2);
        System.out.println("Dva stringa su jednaka: " + signal);

        signal = s1.equalsIgnoreCase(s2);
        System.out.println("Dva stringa su jednaka: " + signal);
 }
}
// Rezultat:
//
// Dva stringa su jednaka: true
// Dva stringa su jednaka: false
// Dva stringa su jednaka: true
```

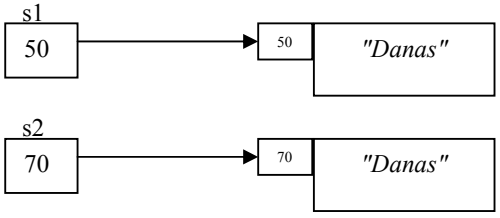
Dva stringa ne treba da se upoređuju pomoću operatora = = jer on ispituje njihov sadržaj (reference) a ne vrednosti na koje ukazuju reference. To znači da će operator = = vratiti true ukoliko dve stringa imaju iste reference.

Na primer naredba:
s1 = = s2
za dati slučaj:



vratiće true.

Međutim u sledećem slučaju:



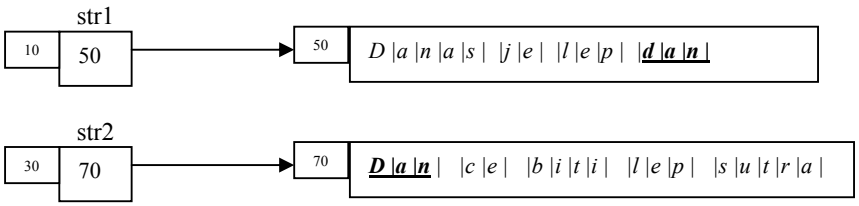
naredba:
s1= = s2
vratiće false, bez obzira što se vrednosti na koje ukazuju s1 i s2 iste.

7.7.2 Ispitivanje jednakosti dva podniza dva različita stringa
Ukoliko se želi ispitati da li su jednaka dva podniza dva različita stringa koristi se metoda:
Boolean regionMatches(int početniIndeksPrvogStringa, String Drugi, int početniIndeksDrugogStringa, int BrojZnakovaKojiSePorede);
Ukoliko se želi zanemariti razlika između velikih i malih slova koristi se istoimena metoda *regionMatches()* koja ima kao prvi parametar logičku promenljivu, koja ima vrednost true ukoliko se želi zanemariti razlika između velikih i malih slova, odnosno false ukoliko se želi praviti razlika između velikih i malih slova:
Boolean regionMatches(boolean zanemariRazliku, int početniIndeksPrvogStringa, String Drugi, int početniIndeksDrugogStringa, int BrojZnakovaKojiSePorede);

Na primer (String72.java):

```
class String72
{  public static void main(String args[])
 {
          String str1 = "Danas je lep dan";
          String str2 = "Dan ce biti lep sutra";

   boolean signal = str1.regionMatches(13,str2,0,3);
   System.out.println("Dva niza su jednaka: " + signal);
   signal = str1.regionMatches(true,13,str2,0,3);
   System.out.println("Dva niza su jednaka: " + signal);
 }
}
// Rezultat:
//
// Dva niza su jednaka: false
// Dva niza su jednaka: true
```

Izgled operativne memorije:



7.7.3 Ispitivanje da li string započinje ili se završava sa vrednošću drugog stringa

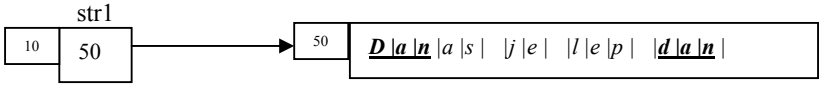
Kada se želi ispitati da li string započinje sa drugim stringom koristi se metoda:
boolean startsWith(String DrugiString);
Kada se želi ispitati da li se string završava sa drugim stringom koristi se metoda:
boolean endsWith(String DrugiString);

Na primer (String73.java):

```
class String73
{
  public static void main(String args[])
 {
          String str1 = "Danas je lep dan";

   boolean signal = str1.startsWith("dan");
   System.out.println("String str1 zapocinje sa stringom dan: " + signal);
   signal = str1.startsWith("Dan");
```

```
   System.out.println("String str1 zapocinje sa stringom dan: " + signal);
   signal = str1.endsWith("dan");
   System.out.println("String str1 se zavrsava sa stringom dan: " + signal);


 }}
// Rezultat:
//
// String str1 zapocinje sa stringom dan: false
// String str1 zapocinje sa stringom dan: true
// String str1 se zavrsava sa stringom dan: true
```

Izgled operativne memorije:



7.7.4 Ispitivanje koji je od dva stringa veći

Kada se želi ispitati koji od dva stringa su veći koristi se metoda:
int compareTo(String drugiString);
Ukoliko su dva stringa jednaka metoda vraća 0. Ukoliko je prvi string veći od drugog stringa metoda vraća vrednost veću od 0. Ukoliko je drugi string veći od prvog stringa metoda vraća vrednost manju od 0.
Kada se želi zanemariti razlika između velikih i malih slova kod upoređivanja stringova koristi se metoda:
int compareToIgnoreCase(String drugiString);

Na primer (String74.java)*;*

```
class String74
{  public static void main(String args[])
   {   String str1 = "Aca";
       String str2 = "Bilja";
       String str3 = "Daca";
       String str4 = "Aca";
       int signal = str1.compareTo(str2); // Aca < Bilja metoda vratila -1
       System.out.println(signal);
       signal = str3.compareTo(str2); // Daca > Bilja metoda vratila 2
       System.out.println(signal);
        signal = str1.compareTo(str4); // Aca = Aca metoda vratila 0
       System.out.println(signal);
   }
}
// Rezultat:
// -1
// 2
// 0
```

Zadatak STZ4: Kreirati i inicijalizovati dva stringa. Uporediti koji od ta dva stringa je veći. Ispitati da li je jedan od stringova podstring drugog stringa. Od ta dva stringa napraviti treći koji će prikazati parne pozicije prvog i neparne pozicije drugog stringa.

7.8 Konvertovanje stringova

U stringove se mogu konvertovati:

a) bilo koji od prostih tipova podataka:

static String valueOf(R broj); R ∈ (float,double)
static String valueOf(C broj); C ∈ (byte,short,int,long)
static String valueOf(char znak);
static String valueOf(boolean signal);

b) nizovi znakova:

static String valueOf(char znaci[]);

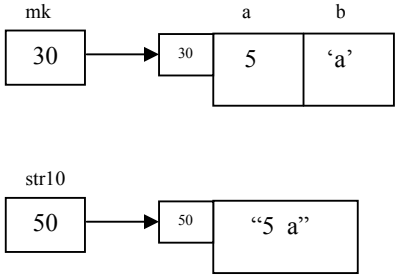
c) bilo koja klasa koja se napravi u Javi:

static String valueOf(Object klasa);

Na primer (String81.java):

```
class String81
{  public static void main(String args[])
   {
    float r1=678.89F;
    double r2=3456787.45;
    byte b=124;
    short s=1234;
    int c=63456;
    long lc=345645634;
    char znak='k';
    boolean signal=true;
    char niz[] = {'d','a','n','a','s'};
    MojaKlasa mk = new MojaKlasa(5,'a');;
    String str1 = String.valueOf(r1);
    String str2 = String.valueOf(r2);
    String str3 = String.valueOf(b);
    String str4 = String.valueOf(s);
    String str5 = String.valueOf(c);
    String str6 = String.valueOf(lc);
    String str7 = String.valueOf(znak);
    String str8 = String.valueOf(signal);
    String str9 = String.valueOf(niz);
    String str10 = String.valueOf(mk);

    System.out.println("Realni: " + str1 + " " + str2);
    System.out.println("Celi: " + str3 + " " + str4 + " " + str5 + " " + str6);
    System.out.println("Znak: " + str7);
    System.out.println("Logicki: " + str8);
    System.out.println("Niz: " + str9);
   System.out.println("MojaKlasa: " + str10);
 }}

class MojaKlasa
{  int a;   char b;
  MojaKlasa(int a1,char b1) { a=a1; b=b1;}
  public String toString() { return String.valueOf(a) + " " + String.valueOf(b); }
}

// Rezultat:
//
// Realni: 678.89 3456787.45
// Celi: 124 1234 63456 345645634
// Znak: k
// Logicki: true
// Niz: danas
// MojaKlasa: 5 a
```

Izgled operativne memorije za objekte *mk* i *str10*:



Stringovi se mogu konvertovati u:
b) bilo koji od prostih tipova podataka:
static float Float.parseFloat(String str) throws NumberFormatException;
static double Double.parseDouble(String str) throws NumberFormatException;
static byte Byte.parseByte(String str) throws NumberFormatException;
static short Short.parseShort(String str) throws NumberFormatException;
static int Integer.parseInt(String str) throws NumberFormatException;
static long Long.parseLong(String str) throws NumberFormatException;
char charAt(0);
static boolean valueOf(String str).booleanValue();

b) nizove znakova:
char [] toCharArray();

c) u bilo koju klasu koja se napravi u Javi. Medjutim u samoj klasi se mora napraviti metoda koja će učiniti konverziju.

```
class String82
{
  public static void main(String args[])
 {
  String str1="678.89";
  String str2="3456787.45";
 String str3="124";
 String str4="1234";
 String str5="63456";
 String str6="345645634";
 String str7="k";
 String str8="true";
 String str9 = "danas";
 String str10 = "5 a";
 float r1 = 0;double r2=0;byte b=0;short s=0;int c=0;long lc=0;
  char niz[];

    try { r1 = Float.parseFloat(str1);}
    catch(NumberFormatException n){System.out.println(n);}

    try { r2 = Double.parseDouble(str2);}
    catch(NumberFormatException n){System.out.println(n);}

    try { b = Byte.parseByte(str3);}
    catch(NumberFormatException n){System.out.println(n);}

    try { s = Short.parseShort(str4);}
    catch(NumberFormatException n){System.out.println(n);}

    try { c = Integer.parseInt(str5);}
    catch(NumberFormatException n){System.out.println(n);}
```

```
   try { lc = Long.parseLong(str6);}
   catch(NumberFormatException n){System.out.println(n);}

   char znak = str7.charAt(0);

   // boolean signal = Boolean.getBoolean(str8); nije htelo
   boolean signal = Boolean.valueOf(str8).booleanValue();
   niz = str9.toCharArray();

   MojaKlasa mk = new MojaKlasa();
   mk.KonverzijaStringaUMojuKlasu(str10);

   System.out.println("Realni: " + r1 + " " + r2);
   System.out.println("Celi: " + b + " " + s + " " + c + " " + lc);
   System.out.println("Znak: " + znak);
   System.out.println("Logicki: " + signal);
   System.out.print("Niz: ");
   for(int i=0;i<niz.length;i++) {System.out.print(niz[i]);}
   System.out.println("\nMojaKlasa: " + mk);

 }
}

class MojaKlasa
{
  int a;
  char b;
  MojaKlasa(){}
  MojaKlasa(int a1,char b1) { a=a1; b=b1;}
  public String toString() { return String.valueOf(a) + " " + String.valueOf(b); }
  void KonverzijaStringaUMojuKlasu(String str)
  { int poz = str.indexOf(' ');
    String pom1 = str.substring(0,poz);
    try { a = Integer.parseInt(pom1);}
    catch(NumberFormatException n){System.out.println(n);}

    String pom2 = str.substring(poz+1);
    b = pom2.charAt(0);
  }

}

// Rezultat:
//
// Realni: 678.89 3456787.45
// Celi: 124 1234 63456 345645634
// Znak: k
// Logicki: true
// Niz: danas
// MojaKlasa: 5 a
```

Zadatak STZ5: Nacrtati izgled operativne memorije za objekte *mk* i *str10*. Objasniti metodu *KonverzijaStringaUMojuKlasu()*.

7.9 Čitanje stringa preko standardnog ulaza

U Javi objekat iz koga se čita sekvenca bajtova naziva se ulazni tok (input stream), dok se objekat u koga se upisuje sekvenca bajtova naziva izlazni tok (output stram). Apstraktne klase iz kojih se izvode navedeni tokovi su InputStream i OutputStream. Pošto su byte orijentisani sistemi nepogodni za procesiranje informacija sačuvanih u Unicode formatu (Unicode format koristi 2 bajta po znaku), uvedene su apstraktne klase Reader i Writer.
Klasa System sadrže tri predefinisane promenjive koje se odnose na tokove: **in, out, err.** Navedene promenljive su public i static, što znači da se njima može pristupiti iz bilo kog dela programa preko klase System (npr. System.in).
System.out se odnosi na standardni izlazni tok, koji je podrazumevano konzola. System.in se odnosi na standardni ulazni tok, koji je podrazumevano tastatura. System.err se odnosi na standardni tok za greške, koji je takođe podrazumevano konzola.
System.in prima preko konzole podatke u binarnom obliku(formatu). Kada se System.in povezuje sa klasom koja definiše znakovni tok, vrši se konvertovanje podataka iz binarnog u znakovni oblik.
Povezivanje System.in promenljive sa znakovnim tokom se radi na sledeći način:
InputStreamReader br = new InputStreamReader(System.in);
Binarni podaci koje prihvata System.in se automatski konvertuju u znakovne podatke koji su u Unicode formatu, a koje podržava klasa InputStreamReader.
Klasa InputStreamReader ne podržava metodu readLine() koja je potrebna da bi se napunio string. Zbog toga se uvodi klasa BufferedReader koja podržava metodu readLine().
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
Objekat br predstavlja baferovan ulazni znakovni tok.

Na primer (Tokovi2.java)

```
import java.io.*;

class Tokovi2
{
  public static void main(String args[]) throws IOException
    { String s;
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      System.out.println("Unesite znakove. Za kraj unesite znak 'k'");
      while(true)
      { s =  br.readLine(); // Unos znakova preko konzole.
        System.out.println(s); // Prikaz znakova preko konzole.
        if (s.equals("k"))
          break;
              }
  }
}
```

```
// Izlazni rezultat:
// danas
// danas
// k
// k
```

Zadatak STZ6: Uneti broj kao string a zatim ga konvertovati u promenljivu tipa broj. Prikazati promenljivu tipa broja na ekranu.

7.10 Formatiranje stringa kod prikazivanja realnog broja

U slučaju kada se dele 2 broja npr:
x = 1000.0 / 3.0;
nakon izvršenja naredbe:
System.out.println(x);
dobija se:
3333.3333333333335

Ukoliko se ovaj broj želi prikazati na ekranu u nekom uređenom obliku koristi se klasa NumberFormat.

```
import java.text.*;

class String10
{
  public static void main(String args[])
    {
      double x = 10000.0/3.0;
      System.out.println(x);

      NumberFormat nf= NumberFormat.getNumberInstance();
      String str = nf.format(x); // Prikazuje u formatu kako je lokalni sistem podesen.
      System.out.println(str);

      nf.setMaximumFractionDigits(2); // 2 mesta je odvojeno za decimalni deo realnog broja.
      nf.setMinimumIntegerDigits(5); // 5 mesta je odvojeno za celobrojni deo realnog broja.
      str = nf.format(x);
      System.out.println(str);

    }
}

// Izlazni rezultat:
// 3333.3333333333335
// 3,333.333
// 03,333.33
```

7.11 Klasa StringBuffer

Klasa StringBuffer daje dopunske mogućnosti koje nema klasa String. Navešćemo osnovne metode klase StringBuffer:

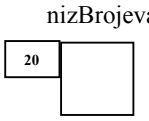
a) *StringBuffer()* - rezerviše se mesta za 16 znakova
b) *StringBuffer(int broj)* - rezerviše se mesta za onoliko znakova koliko je definisano parametrom broj.
c) *StringBuffer(String str)* - rezerviše se mesta za onoliko znakova koliko ima u parametru str. Takođe StringBuffer se inicijalizuje na vrednost parametra str.
d) *int length()* - vraća dužinu stringa.
e) *int capacity()* - vraća kapacitet bafera gde se može smestiti string.
f) *void ensureCapacity(int kapacitet)* – menja se kapacitet bafera.
g) *void setLength(int duzina)* – menja se dužina bafera. Tamo gde nema vrednosti dodaje se 0. Duzina ne sme biti manja od 0. Ako se zada dužina manja od trenutne dužine podaci izvan nove dužine biće izgubljeni.
h) *char charAt(int indeks)* – vraća znak iz stringa sa pozicije koja je definisana parametrom indeks.
i) *void setCharAt(int indeks, int znak)* – na poziciju u stringu koja je zadata parametrom indeks postavlja se znak koji je zadat parametrom znak.
j) *void getChars(int pocIndeksStr, int zavIndStr, char niz[], int pocIndeksNiza)* – kopiranje stringa, od početne pozicije pocIndeksStr do pozicije zavIndStr (ne uključujući tu poziciju) u niz koji počinje od pozicije pocIndeksNiza.
k) StringBuffer append(String str) – na kraj stringa se dodaje string zadat parametrom str.
l) *StringBuffer append(int broj)* – na kraj stringa se dodaje ceo broj (konvertovan u string) zadat parametrom broj. Na sličan način se mogu dodati stringu i drugi osnovni tipovi podataka (double,char,boolean,…).
m) *StringBuffer append(Object obj)* – na kraj stringa se dodaje objekat (konvertovan u string) zadat parametrom obj.
n) *StringBuffer insert (int indeks, String str)* – na mesto u stringu koje je zadato pozicijom indeks stavlja se string zadat parametrom str.
o) *StringBuffer insert (int indeks, char znak)* – na mesto u stringu koje je zadato pozicijom indeks stavlja se znak (ili neki od drugih osnovnih tipova) zadat parametrom str.
p) *StringBuffer insert (int indeks, Object obj)* – na mesto u stringu koje je zadato pozicijom indeks stavlja se objekat zadat parametrom obj.
q) *StringBuffer reverse* – izokreće se redosled znakova u stringu.

r) *StringBuffer delete(int pocetniIndeks, int zavrsniIndeks)* – briše se string od pozicije koja je zadata parametrom pocetniIndeks do pozicije koja je zadata parametrom zavrsniIndeks.
s) *StringBuffer deleteCharAt(int Indeks)* – briše se iz stringa znak koji se nalazi na poziciji koji je zadat parametrom Indeks.
t) *StringBuffer replace(int pocetniIndeks, int zavrsniIndeks, String str)* – brišu se iz stringa znakovi koji se nalaze između pozicija koji su zadati parametrom pocetniIndeks i zavrsniIndeks – 1. Zatim se pocev od pozicije koja je zadata parametrom pocetniIndeks unosi string koji je zadat parametrom str.
u) *String substring(int pocetniIndeks)* – vraća podstring od pozicije koja je zadata parametrom pocetniIndeks.
v) *String substring(int pocetniIndeks, int zavrsniIndeks)* – vraća podstring između pozicija koje su zadate parametrom pocetniIndeks i zavrsniIndeks – 1.

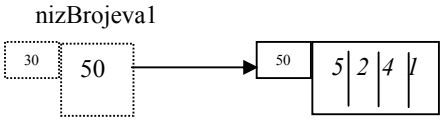
7.12 Rad sa nizovima

Niz predstavlja skup elemenata koji su istog tipa. Niz se deklariše na sledeći način:
Tip niz[];

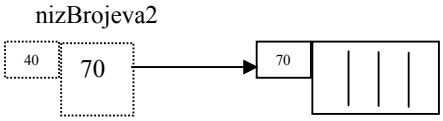
Na primer ukoliko imamo niz celih brojeva on se predstavlja na sledeći način:
int nizBrojeva[];



Niz se može kreirati i inicijalizuje kod deklaracije na sledeći način:
int nizBrojeva1[] = { 5,2,4,1};

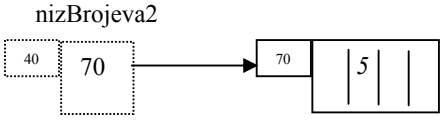


Ukoliko niz nije inicijalizovan na navedeni način, pre dodele vrednosti do elemenata niza, potrebno je da se niz kreira:
int nizBrojeva2[] = new int[4];



Tek nakon toga mogu se dodeliti vrednosti do elemenata niza.

NizBrojeva2[1] = 5;



Dvodimenzioni niz, npr celih brojeva, se deklariše i kreira na sledeći način:

int Matrica[][] = new int[2][3]; // Matrica koja ima 2 reda i 3 kolone.

Deklaracija niza ima 2 ekvivalentna oblika:

int Niz[] = new int[3];
int [] Niz = new int[3];

Ukoliko se pravi niz objekata neke klase, npr. klase MojaKlasa:
MojaKlasa mk[];

Prvo je potrebno da se kreira niz:
MojaKlasa mk[] = new MojaKlasa[5];

Nakon toga se kreira svaki pojedinačni objekat niza:
```
for(int i=0;i<5;i++)
  mk[i] = new MojaKlasa();
```

Na sledećem primeru objasnićemo niz (Niz1.java):

```
class Niz1
{   public static void main(String args[])
    {
            int nizBrojeva[] = new int[5];
            int i;
            nizBrojeva[0] = 4;
            nizBrojeva[1] = 2;
            nizBrojeva[2] = 7;
            nizBrojeva[3] = 6;
            nizBrojeva[4] = 3;

     System.out.print("Niz: ");
     for(i=0;i<5;i++) System.out.print(nizBrojeva[i]);

            MojaKlasa mk[] = new MojaKlasa[3];
            for(i=0;i<3;i++) mk[i] = new MojaKlasa(i+2);

            System.out.print("\nMoja klasa: ");
            for(i=0;i<3;i++)
            System.out.print(mk[i].a);
    }
}

class MojaKlasa
{int a;
MojaKlasa(int a1) { a=a1;}
}

// Izlazni rezultat:
//
// Niz: 42763
// Moja klasa: 234
```

Zadatak STZ7: Nacrtati izgled operativne memorije za objekte *nizBrojeva* i *mk.*

Tematska jedinica	**Rad sa stringovima**	
Datum:		
Zadatak		
Ime i prez. studenta – br. ind.		
Laborant:		

8. Paketi

U proceduralnim programskim jezicima, imena procedura ili funkcija nisu mogla da se dupliraju. Kod objektnih programskih jezika, uvođenjem koncepta klase, omogućeno je definisanje istoimenih metoda sa različitim potpisima u okviru jedne klase. Takođe je omogućeno da dve istoimene metode (sa istim potpisom), mogu da budu definisane u dve proizvoljne različite klase.

```
class A
{ ...
  m();
  m(int);
}

class B
{ ...
  m();
}

A a;
B b;
a.m (); // metoda m() pozvana preko objekta a
a.m (5); // metoda m(int) pozvana preko objekta a
b.m (); // metoda m() pozvana preko objekta b
```

U navedenom slučaju omogućeno je postojanje istoimenih metoda (polimorfizam metoda), ali nije omogućeno da se u okviru jednog programa definišu dve istoimene klase.
Koncept paketa je omogućio da u okviru jednog programa može biti pozvano dve ili više istoimenih klasa koje se nalaze u različitim paketima. Na taj način je omogućen polimorfizam klasa.

8.1 Preslikavanja između paketa, datoteka i klasa

- Jedan paket može biti vezan za više datoteka u kojima se nalaze klase. To praktično znači da više klasa može biti vezano za jedan paket.

Na primer paket *p1* se vezuje za datoteke test.java (sadrži klase A i B) i test1.java (sadrži klase C i D) na sledeći način:
U datoteci test.java imamo:

```
package p1;
class A{}
class B{}
```

U datoteci test1.java imamo:

```
package p1;
class C{}
class D{}
```

Iz navedenog primera se vidi da su obe datoteke, odnosno njihove klase vezane za paket p1.

- Jedna datoteka može biti vezana za samo jedan paket.

Na primer datoteka test.java se vezuje samo za jedan paket (p1):

```
package p1;
class A{}
class B{}
```

Ukoliko bi pokušali da vežemo datoteku za dva paketa, javila bi se greška kod kompajliranja.

```
package p1;
package p2; // Jedna datoteka ne moze da se veze za dva paketa.
Class A{}
class B{}
```

8.2 Veza paketa i sistema direktorijuma

Naziv paketa je direktno povezan sa sistemom direktorijuma tekućeg diska. To praktično znači da ime paketa treba da odgovara imenu direktorijuma u kome se nalaze datoteke vezane za taj paket.
Npr.ukoliko se želi da datoteka (test.java) bude vezana za paket (P1), u datoteku se upisuje ključna reč package, iza koje se navodi ime paketa (npr: package P1;). Ovakva datoteka se može kompajlirati bez problema sa:
javac test.java
ukoliko nije u nju uključen neki drugi paket (o čemu ćemo kasnije govoriti) i ukoliko se poziv izvodi iz tekućeg direktorijuma gde se nalazi datoteka test.java. Međutim kod pozivanja takvog programa javiće se greška ukoliko pokušate da pozovete:
java test

Ovo se dešava zbog toga što je datoteka test.java vezana za paket P1 (ukoliko test.java datoteka ne bi bila vezana za paket, ne bi se desila greška).
Postoje dva načina da se pozove izvršenje ovakve datoteke:

a) u prvom direktorijumu koji je iznad tekućeg direktorijuma treba da se pozove izvršenje navedene datoteke. Znači ukoliko je datoteka test.java bila u direktorijumu C:\test\P1, treba da se poziv izvršenja programa pokrene iz direktorijuma C:\test. Naglašavamo da se ne prebacuje datoteka test.java iz tekućeg u prvi viši direktorijum, nego se samo poziv izvršava iz direktorijuma koji je prvi viši u hijerarhiji (**C:\test\java P1.test**). Takođe može da se primeti da se datoteka ne poziva direktno već se ispred nje stavlja ime paketa (P1). Ovim se može shvatiti direktna veza između imena direktorijuma i imena paketa.
b) U sistemskoj promenljivoj CLASSPATH se navodi put do foldera koji je ujedno i paket za Javine klase. Naglašavamo da se taj folder ne uključuje u put. Npr. ukoliko imamo paket P1 koji se nalazi na: F:\java\kurs\P1, u promenljivoj CLASSPATH se navodi F:\java\kurs. Time se dobija mogućnost da se program **test** poziva iz bilo kog dela programa sa naredbom: **java P1.test**

*Važne napomene**:***

- Kod kompajliranja proizvoljne datoteke, koja je vezana za paket X, nije potrebno da ime datoteke (u kome se kompalira program) bude isto kao ime paketa X.

- Kod izvršavanja programa (datoteke) koji je vezan za paket X **potrebno je da ime datoteke** (u kome se izvršava program) bude isto kao ime paketa X.

8.3 Definisanje paketa

Navešćemo primer definisanja paketa p1 koji je vezan za klase datoteke test.java

```
package p1;
class A{}
class B{}
```

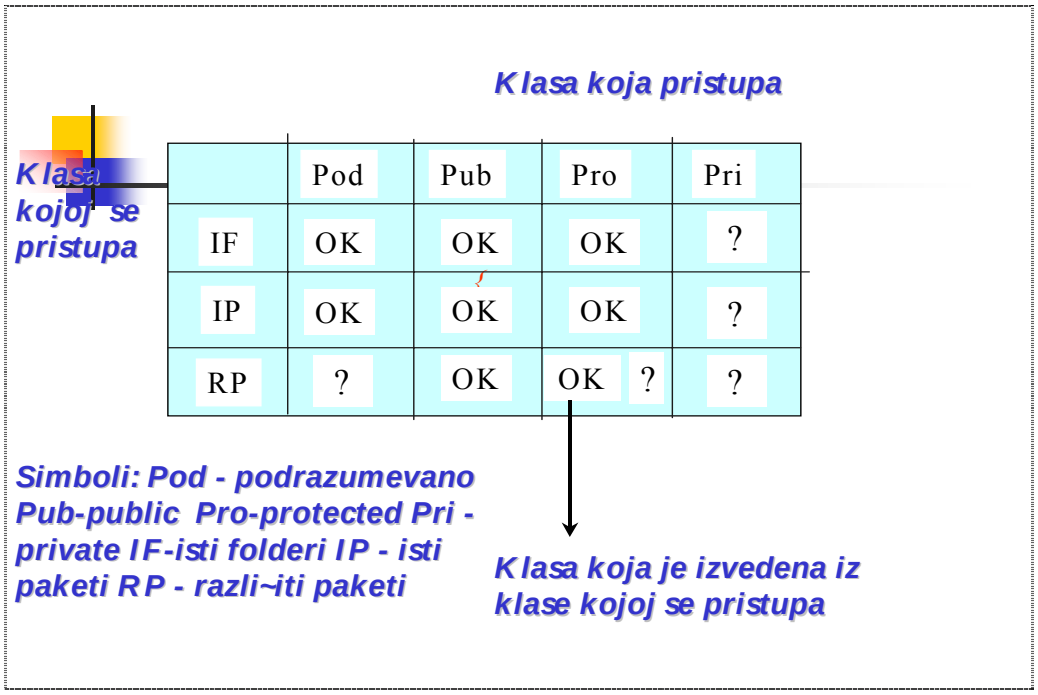
Opšti oblik za definisanje paketa je:
package paket1[.paket2[.paket3]];

Napomena: Navedeni redosled poziva paketa mora da bude u skladu sa sistemom direktorijuma.

8.4 Način pristupa članovima klase iz paketa

Pristup članovima neke klase (pristupna klasa) iz druge klase (pristupajuće klase) zavisi od:

a) načina pristupa koji stoji ispred članica pristupne klase (public, private, protected, podrazumevano - ništa se ne navodi).
b) mesta gde se nalaze pristupna i pristupajuća klasa.



- Ukoliko se pristupna i pristupajuća klasa nalaze u istom folderu ili u istom paketu tada će moći da se pristupi do članica pristupne klase iz pristupajuće klase, u svim slučajevima osim kada je private način pristupa stavljen ispred članica pristupne klase.

- Ukoliko se pristupna i pristupajuća klasa nalaze u različitim folderima tada će moći da se pristupi do članica pristupne klase iz pristupajuće klase, u slučaju kada je public način pristupa stavljen ispred članica pristupne klase ili protected u slučaju da je pristupajuća klasa izvedena iz pristupne klase.

8.5 Uvoženje paketa

Ukoliko se javi potreba da se klase iz nekog paketa uključe u tekući program tada se preko naredbe:
import paket1.(imeklase|*);

može uključiti neka od klasa paketa:
import paket1.imeklase;
ili sve klase paketa:
import paket1.*;

- Kod uvoženja datoteke Y (koja je vezana za paket X) **potrebno je da ime direktorijuma** (u kome se nalazi datoteka Y) bude isto kao ime paketa X.

npr: Datoteka Y.java sadrži sledeći kod:
package X
class Y {...}
class Z {...}
Datoteka Y.java nalazi se u folderu: C:\java\X

- Klasa Y iz datoteke Y.java se uvodi u datoteku test.java na sledeći način:

import X.Y;
class test{...}

Datoteka test.java treba da bude u direktorijumu: C:\java

- Klase Y i Z iz datoteke Y.java se uvode u datoteku test.java na sledeći način:

import X.;*
class test{...}

Datoteka test.java treba da bude u direktorijumu: C:\java

8.6 Primeri - Paketi

Primer P1:

```
/* Programski zahtev: Pokazati kako se povezuju, kompajliraju i izvršavaju klase ukoliko se nalaze u
istom folderu. Datoteka Glavna.java, Osoba.java i Student.java nalaze se istom folderu.*/
//Datoteka: Glavna.java
class Glavna
 {
  public static void main(String args[])
   {
      Student k1 = new Student("B123434","Pera Peric","Beogradska 1","123-99",3);
     k1.Prikazi();

   }
}

//Datoteka: Osoba.java
class Osoba{

String MLB;
String ImePrezime;
String Adresa;
Osoba(){MLB ="";ImePrezime="";Adresa="";}
Osoba(String MLB1,String ImePrezime1,String Adresa1) {
MLB=MLB1;ImePrezime=ImePrezime1;Adresa=Adresa1;}
void Prikazi(){System.out.println("MLB: " + MLB + " Ime i prezime: " + ImePrezime + " Adresa:" +
Adresa);}
}

//Datoteka: Student.java
class Student extends Osoba{

String BrojIndeksa;
int Godina;
Student(){BrojIndeksa="";Godina=0;}
Student(String MLB1,String ImePrezime1,String Adresa1, String BrojIndeksa1, int Godina1)
     { super(MLB1,ImePrezime1,Adresa1); BrojIndeksa = BrojIndeksa1; Godina = Godina1;}
void Prikazi(){ super.Prikazi();
                System.out.println("Broj indeksa: " + BrojIndeksa + " Godina: " + Godina);}
}
```

Primer P2:

```
/* Programski zahtev: Pokazati kako se povezuju, kompajliraju i izvršavaju klase ukoliko se nalaze u
istom paketu. Datoteka Glavna.java, Osoba.java i Student.java nalaze se istom istom paketu P2,  koji
se nalazi se nalazi u sledecem folderu: c:\PrimeriPaketa\P2.
Sistemska varijabla CLASSPATH dobila je sledecu vrednost: c:\PrimeriPaketa */

//Datoteka: Glavna.java
package P2;

class Glavna
 {  public static void main(String args[])
   {   Student k1 = new Student("B123434","Pera Peric","Beogradska 1","123-99",3);
     k1.Prikazi();
    }
}
```

```
//Datoteka: Osoba.java
package P2;

class Osoba{

String MLB;
String ImePrezime;
String Adresa;
Osoba(){MLB ="";ImePrezime="";Adresa="";}
Osoba(String MLB1,String ImePrezime1,String Adresa1) {
MLB=MLB1;ImePrezime=ImePrezime1;Adresa=Adresa1;}
void Prikazi(){System.out.println("MLB: " + MLB + " Ime i prezime: " + ImePrezime + " Adresa:" +
Adresa);}
}

//Datoteka: Student.java
package P2;

class Student extends Osoba{

String BrojIndeksa;
int Godina;
Student(){BrojIndeksa="";Godina=0;}
Student(String MLB1,String ImePrezime1,String Adresa1, String BrojIndeksa1, int Godina1)
      { super(MLB1,ImePrezime1,Adresa1); BrojIndeksa = BrojIndeksa1; Godina = Godina1;}
void Prikazi(){ super.Prikazi();
                System.out.println("Broj indeksa: " + BrojIndeksa + " Godina: " + Godina);}

}
```

Primer P3:

```
/* Programski zahtev: Pokazati kako se povezuju, kompajliraju i izvršavaju klase ukoliko se nalaze u
različitim paketima. Datoteka Glavna.java nalazi se u folderu c:\PrimeriPaketa\P3, Osoba.java u
folderu c:\PrimeriPaketa\P3\P31  i Student.java u folderu c:\PrimeriPaketa\P3\P31 Sistemska
varijabla CLASSPATH dobila je sledecu vrednost: c:\PrimeriPaketa */

//Datoteka: Glavna.java
package P3;

import P3.P31.*;
import P3.P32.*;
class Glavna
 {
  public static void main(String args[])
    {  Student k1 = new Student("B123434","Pera Peric","Beogradska 1","123-99",3);
       k1.Prikazi();
     }
}
//Datoteka: Osoba.java
package P3.P31;

public class Osoba{
String MLB; String ImePrezime; String Adresa;
protected Osoba(){MLB ="";ImePrezime="";Adresa="";}
protected Osoba(String MLB1,String ImePrezime1,String Adresa1) {
MLB=MLB1;ImePrezime=ImePrezime1;Adresa=Adresa1;}
protected  void Prikazi(){System.out.println("MLB: " + MLB + " Ime i prezime: " + ImePrezime + "
Adresa:" + Adresa);}
}
//Datoteka: Student.java
```

```
package P3.P32;
import P3.P31.*;

public class Student extends Osoba{
String BrojIndeksa;
int Godina;
public Student(){BrojIndeksa="";Godina=0;}
public Student(String MLB1,String ImePrezime1,String Adresa1, String BrojIndeksa1, int Godina1)
    { super(MLB1,ImePrezime1,Adresa1); BrojIndeksa = BrojIndeksa1; Godina = Godina1;}
public void Prikazi(){ super.Prikazi();
            System.out.println("Broj indeksa: " + BrojIndeksa + " Godina: " + Godina);}
}
```

8.7 Zadaci – Paketi

Zadatak PZ1: Pokazati kako se povezuju, kompajliraju i izvršavaju klase ukoliko se nalaze u istom paketu. Datoteka Glavna.java, Projekat.java i Zadatak.java nalaze se istom istom paketu PZ1, koji se nalazi se nalazi u sledecem folderu: c:\VezbaPaketa\PZ1.
Sistemska varijabla CLASSPATH treba da dobije sledecu vrednost: c:\ VezbaPaketa.

Zadatak PZ2: Pokazati kako se povezuju, kompajliraju i izvršavaju klase ukoliko se nalaze u različitim paketima. Datoteka Glavna.java nalazi se u folderu c:\VezbaPaketa\PZ2, Projekat.java u folderu c:\VezbaPaketa\PZ2\PZ21 i Zadatak.java u folderu c:\VezbaPaketa\PZ2\PZ22. Sistemska varijabla CLASSPATH treba da dobije sledecu vrednost: c:\VezbaPaketa.

Pitanja:

1. Koja je osnovna svrha paketa?
2. Da li se ime paketa u nekoj Java datoteci može razlikovati od imena foldera kod kompajliranja?
3. Šta bi se desilo ukoliko bi pri radu sa paketima ne bi bila korišćena sistemska promenljiva CLASSPATH?
4. Nacrtati tabelu koja povezuje načine pristupa do članica klasa i mesta gde se nalaze klase (u istom folderu, u istom paketu i različitim paketima).

Tematska jedinica	**Paketi**	
Datum:		
Zadatak		
Ime i prez. studenta – br. ind.		
Laborant:		

9. Obrada izuzetaka

9.1 Definicija izuzetaka (exception)

Izuzetak predstavlja slučaj koji se dešava u toku izvršenja programa koji može da poremeti normalan rad programa.
Izuzetke mogu generisati:

a) metode postojećih Javinih klasa ili
b) metode korisničkih klasa.

9.2 Vrste grešaka

Izuzeci se uglavnom javljaju onda kada se desi neka greška u programu. U tom smislu postoje 4 tipa mogućih grešaka:

- Greške pri unosu podataka (User input errors)
- Greške koje su vezane za rad hardvera računara (Device errors) – npr. flopi disk ne čita dobro podatke, štampač nije uključen i spreman za rad.
- Greške koje su vezane za razna fizička ograničenja – disk je pun i ne može da pamti nove podatke, raspoloživa memorija je nedovoljna za izvršenje programa (Physical limitations)
- Greške kod izvršenja metoda programa (Code errors).

Kada se desi neka greška u programu, metoda u kojoj se desila greška pravi (baca) izuzetak[12] koji se odnosi na tu grešku. Izuzetak se zatim hvata (catch) i obrađuje. Mesta u programu gde se hvata greška, nazivaju se tačke presretanja grešaka (error-trapping).

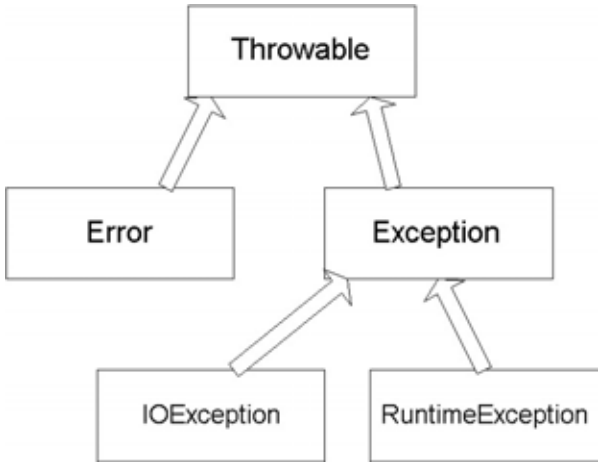
9.3 Klasifikacija izuzetaka

U Javi izuzeci su objekti klasa koje su izvedene iz klase Throwable. Iz klase Throwable, koja se nalazi u vrhu hijerarhije izvedene su dve klase:

- Exception i
- Error

Klasa Exception je takođe podeljena u dve podklase:

- RunTimeExpection
- IOException



9.4 Neuhvaćeni izuzeci

Ukoliko se u programu desi izuzetak koji ne može da se obradi korisničkim programom, pokreće se standardni Javin obrađivač koji prikazuje:

- tekst sa opisom izuzetka i
- stak pozvanih metoda u trenutku dešavanja izuzetka.

Na kraju standardni obrađivač prekida izvršenje programa.

[12] Izuzetak je objekat klase koja je izvedena iz klase Throwable.

Primer:

```
// Ovaj program ima gresku i nece se iskompajlirati.
class IZ1{
  static void proba() throws IllegalAccessException {
    System.out.println("Ulazak u proba metodu");
    throw new IllegalAccessException("IZUZETAK");
  }
  public static void main(String args[]) {
      proba();
  }
}
```

Izlaz:

```
…\KursJava\ThrowsDemo.java:5: unreported exception java.lang.IllegalAccessException; must be
caught or declared to be thrown
    throw new IllegalAccessException("IZUZETAK");
    ^
1 error
Tool completed with exit code 1
```

```
//Sada je ispravno
class IZ1{
  static void proba() throws IllegalAccessException {
    System.out.println("Ulazak u proba metodu");
    throw new IllegalAccessException("IZUZETAK");
  }
  public static void main(String args[]) {
    try {
      proba();
    } catch (IllegalAccessException e) {
      System.out.println("UHVACEN " + e);
    }
  }
}

// Rezultat:
//
// Ulazak u proba metodu
// UHVACEN java.lang.IllegalAccessExceptions: IZUZETAK
```

9.5 Korišćenje rezervisanih reči try i catch

Ukoliko se zna da će u pojedinim delovima programa doći do izuzetaka, potrebno je da se od tih delova programa naprave blokovi, koji će biti ograničeni zagradama ispred kojih će stajati ključna reč **try.**

```
try
{ // deo programa u kome se očekuje pojava izuzetaka
  …
}
```

Ukoliko se u navedenim delovima programa desi izuzetak (Exception e), on se obrađuje u bloku koji počinje sa ključnom rečju **catch.**

```
catch (Exception e)
{ // naredbe koje obradjuju izuzetak e
  …
}
```

Obradi izuzetaka uvek predhodi stvaranje (bacanje) izuzetaka što praktično znači da catch blok sledi try blok. Ova dva bloka zajedno čine jednu celinu i uvek se koriste u paru. Nijedan od navedenih blokova ne može se nezavisno koristiti bez drugog bloka.

Opšti oblik generisanja i obrade greške (try-catch par) je:

```
try
{ // deo programa u kome se očekuje pojava izuzetaka
  …
} catch (Exception e)
{ // naredbe koje obradjuju izuzetak
  …
}
```

Nakon obrade greške program nastavlja normalno da se izvršava (u suprotnom ako se ne obradi, kao što je malopre rečeno, prekida se izvršenje programa).

9.6 Prikazivanje opisa izuzetaka

Pomoću catch bloka se obrađuje izuzetak koji je generisan pri izvršenju naredbi try bloka. Tako generisan izuzetak je objekat koji ima svoju strukturu i ponašanje. Pošto je svaki izuzetak izveden iz klase Throwable, on nasleđuje njenu metodu *toString()* koja prikazuje tekst sa opisom izuzetka. Na taj način je moguće da se za svaki izuzetak vidi njen opis.

```
try
{ // deo programa u kome se očekuje pojava izuzetaka
  …
} catch (Exception e)
{ System.out.println(“Izuzetak: “ + e);}
```

Primer:

```
class IZ2{
  static void PodeliSaNulom() throws Exception {
    int a = 5/0;
  }
  public static void main(String args[]) {
    try {
      PodeliSaNulom();
    } catch (Exception e) {
      System.out.println("Uhvacena greska " + e);
    }
  }
}
```

9.7 Ugnježdene naredbe TRY

Jedan try blok može biti ugnježden unutar drugog try bloka. Ukoliko se u unutrašnjem try bloka desi izuzetak koji se ne može obraditi unutrašnjim catch blokom, tada se kontrola obrade izuzetka prenosi na spoljašnji catch blok. Ukoliko spoljašnji catch blok ne može obraditi izuzetak tada izuzetak obrađuje standardni Javin obrađivač.

```
try // Spoljasnji try blok
{ // Deo programa u kome se očekuje pojava izuzetka eX
  …
    try // Unutrasnji try blok
      { // Deo programa u kome se očekuje pojava izuzetaka eX i eY.
        // Ukoliko se ovde desi izuzetak eX on ce biti obradjen u spoljasnjem catch bloku.
        …
      } catch (ExceptionY eY){ Ovde se obradjuje izuzetak  eY} // Unutrasnji catch blok

} catch (ExceptionX eX) { Ovde se obradjuje izuzetak  eX } // Spoljasnji catch blok
```

Primer:

```
class NestTry {
  public static void main(String args[]) {
    try {
      int x = args.length;
      int y = 10 / x;

      System.out.println("x = " + x);

      try { // ugnjezdeni try block

        if(x = =1) x = x/(x-x); // deljenje nulom


        if(x = = 2) {
          int z[] = { 1 };
          z[10] = 150; // generise se out-of-bounds izuzetak
        }
      } catch(ArrayIndexOutOfBoundsException e) {
        System.out.println("out-of-bounds: " + e);
      }

    } catch(ArithmeticException e) {
      System.out.println("Deljenje 0: " + e);
    }
  }
}
```

9.8 Obrada (hvatanje) više izuzetaka preko CATCH

U jednom try bloku može da se desi više različitih tipova izuzetaka. U tom slučaju se za svaki tip izuzetka pravi poseban catch blok.

```
try
{ // Deo programa u kome se očekuje pojava izuzetka eX tipa ExceptionX i eY tipa ExceptionY
  …

} catch (ExceptionX eX) { Ovde se obradjuje izuzetak  eX }
  catch (ExceptionY eY) { Ovde se obradjuje izuzetak  eY }
```

Važna napomena: Podklase izuzetaka moraju da se definišu pre njihovih nadklasa. To znači da u navedenom primeru klasa ExceptionX ne sme da bude nadklasa klase ExceptionY.

Primer:

```
class MultiCatch {
  public static void main(String args[]) {
    try {
      int x = args.length;
      System.out.println("x = " + x);
      int y = 10 / x;
      int z[] = { 1 };
      z[11] = 150;
    } catch(ArithmeticException e) {
      System.out.println("Deljenje nulom: " + e);
    } catch(ArrayIndexOutOfBoundsException e) {
      System.out.println("Index niza oob: " + e);
    }
    System.out.println("Posle try/catch bloka.");
  }
}
```

9.9 Oglašavanje izuzetaka koje će da pravi (baca) metoda preko THROWS

Ukoliko se zna da će u okviru metode neke klase da bude napravljen izuzetak, koji neće biti obrađen u toj metodi, potpis metode se proširuje sa naredbom **throws.**
Iza naredbe **throws** se navode imena klasa očekivanih izuzetaka koji neće biti obrađeni u toj metodi.
Metoda koja poziva metodu koja ima neobrađene izuzetke, mora da sadrži try-catch blok koji će da prihvati i da obradi neobrađene izuzetke. U suprotnom se javlja greška kod kompajliranja programa.

```
class X
{  void m1 (int  a) throws ExceptionX
  { // Deo programa u kome se očekuje pojava izuzetka eX tipa ExceptionX koji nece biti obradjen u
    // metodi m1.
    …  }

 void m2()
  {   try  {   …
            m1(5); // Poziv metode u kojoj se desava neobradjeni izuzetak.
            …
       } catch (ExceptionX eX) { Ovde se obradjuje izuzetak  eX  koji se desio u metodi m1}
  }
}
```

Ukoliko u metodi m2() ne bi postojao par try-catch javila bi se greška kod kompajliranja, osim u slučaju da u metodi m2 navedemo *throws ExceptionX* čime odgovornost za obradu izuzetka prenosimo na neku drugu metodu[13]. Navedeni princip obrade izuzetaka važio bi i u slučaju kada bi metoda m2() bila u nekoj drugoj klasi a ne u klasi X.

9.10 Kako se prave (bacaju) izuzeci preko THROW

Ukoliko postoji potreba u programu se mogu praviti (bacati - throw) izuzeci pomoću naredbe **throw.**
Opšti oblik pravljenja izuzetaka je:
Throw pojavljivanjeTipaThrowable

Mogu se jedino praviti izuzeci, koji su tipa klase koja je izvedena iz klase Throwable.
Svaki izuzetak koji se napravi u programu mora se obraditi inače će program prekinuti da se izvršava.

Primer:

```
String readData (BufferedReader in) throws EOFException
{....
            while (...)
               {if  (ch = = -1) //nailazi na EOF
                        {if (n<len)
                                 throw new EOFException();
                        }...
            }
return s;
}
```

9.11 Ponovo bacanje izuzetka

Ukoliko se uhvati neki izuzetak, on se može posle obrade (prvi try-catch par) opet baciti. Nakon toga drugi try-catch par će ga uhvatiti i obraditi.

Primer:

```
class ThrowDemo {
  static void demoproc() {
    try {
      throw new NullPointerException("demo");
    } catch(NullPointerException e) {
      System.out.println("Uhvacen u  demoproc!");
      throw e; // ponovo bacanje izuzetka
    } }
```

[13] Metodu koja je pozvala metodu m2(), ili metodu koja je iznad u hijerarhiji poziva metode m2().

```
  public static void main(String args[]) {
    try {
      demoproc();
    } catch(NullPointerException e) {
      System.out.println("Ponovo uhvacen: " + e);
    }
  }
}
```

Izlaz:
Uhvacen u demoproc!
Ponovo uhvacen: java.lang.NullPointerException: demo

9.12 Pravljenje sopstvenih klasa izuzetaka

Moguće je praviti sopstvene klase izuzetaka. Ono što predstavlja osnovu svake takve klase je klasa Exception. To znači da se pri pravljenju korisničke klase izuzetka (klasa X u primeru) nasleđuje klasa Exception.

```
class X extends Exception
{…}
```

Primer:

```
class MojaKlasaIzuzetak extends Exception {
  private int broj;

  MojaKlasaIzuzetak(int broj1) {
    broj = broj1;  }

  public String toString() {
    return "Moj izuzetak: [" + broj + "]";
  }
}

class GeneratorIzuzetka {
  static void PraviIzuzetak(int broj) throws MojaKlasaIzuzetak {
    System.out.println("Prosledjen broj je: (" + broj + ")");
    if(broj > 10)
      throw new  MojaKlasaIzuzetak(broj);
    System.out.println("Nije napravio izuzetak!");
  }

  public static void main(String args[]) {
    try {
      PraviIzuzetak(1);
      PraviIzuzetak (20);
    } catch (MyException e) {
      System.out.println("Uhvatio izuzetak: " + e);
    }
  }
}
```

Izlaz:
Prosledjen broj je:(1)
Nije napravio izuzetak!
Prosledjen broj je:(20)
Uhvatio izuzetak: Moj izuzetak [20]

9.13 Rezervisana reč finally

Ukoliko je potrebno da se obavezno izvrše neke naredbe, nakon što se desio izuzetak, u metodi gde se desio izuzetak, koristi se **finally** blok koji sadrži željene naredbe.

```
try
{ // Deo programa u kome se očekuje pojava izuzetka eX
  …
} catch (ExceptionX eX) { Ovde se obradjuje izuzetak  eX }

finally () { Naredbe koje se izvrsavaju nakon sto se desio i obradio izuzetak eX }
```

Primer:

```
class PrimerFinally {
  static void metoda1() {
    try {  System.out.println("Izvrsava se metoda1");
            throw new RuntimeException("primer");
        } finally {System.out.println("finally metoda1");  }
  }

  static void metoda2() {
    try {
      System.out.println("Izvrsava se metoda2");
      return;
    } finally { System.out.println("finally metoda2"); }}

  static void metoda3() {
    try {
      System.out.println("Izvrsava se metoda3");
    } finally {
      System.out.println("finally metoda3"); }
  }

  public static void main(String args[]) {
    try {
      metoda1();
    } catch (Exception e) {  System.out.println("Uhvacen izuzetak");   }
    metoda2();
    metoda3();
  }
}
```

Izlaz:

```
Izvrsava se metoda1
finally metoda1
Uhvacen izuzetak
Izvrsava se metoda2
finally metoda2
Izvrsava se metoda3
finally metoda3
```

9.14 Javini ugrađeni izuzeci

U svom standardnom paketu *java.lang,* Java definiše više klasa izuzetaka. Većina ovih izuzetaka predstavljaju potklase standardnog tipa RuntimeException. Pošto se paket java.lang podrazumevano uvozi u sve Java programe, većina izuzetaka izvedenih iz klase RuntimeException je na raspolaganju. Njih ne treba unositi u listu naredbe *throws* nijedne od metoda. Ovi izuzeci se nazivaju *neproveravani* izuzeci jer prevodilac ne proverava da li metoda ove izuzetke obrađuje ili baca. *Proveravani* izuzeci iz paketa java.lang se moraju uključiti u u listu throws ukoliko dotična metoda generiše takve izuzetke, a sama ih ne obrađuje. Detalje o proveravanim i neproveravanim izuzecima možete naći u [5].

9.15 Zadaci - Izuzeci

Zadatak IZZ1: Napisati program koji ce da generise izuzetak u pozvanoj metodi, koji će biti obradjen u glavnom programu

Zadatak IZZ2: Napisati program koji ce da generise izuzetak pri deljenju broja sa 0, koji ce nakon toga biti obradjen.

Zadatak IZZ3: Dati primer obrade izuzetaka kod ugnjezdenih try naredbi.

Zadatak IZZ4: Napisati program koji ce da pokaže kako se izuzetak moze obraditi sa vise catch naredbi.

Zadatak IZZ5: Napisati program koji baca, hvata, obradjuje, pa opet baca taj izuzetak, hvata ga i obradjuje.

Zadatak IZZ6: Napraviti sopstvenu klasu za obradu izuzetka. Generisati objekat te klase i obraditi ga.

Pitanja:

1. Da li se generisani (bačeni) izuzetak mora obraditi?
2. Šta se događa ukoliko se ne obradi izuzetak?
3. O čemu se mora voditi računa kada više catch naredbi obrađuje izuzetak?
4. Kako se oglašava izuzetak u metodi?

Šta se postiže korišćenjem finaly bloka kod obrade izuzetaka?

Tematska jedinica	**Izuzeci**	
Datum:		
Zadatak		
Ime i prez. studenta – br. ind.		
Laborant:		