# Internet programiranje predavanje 14

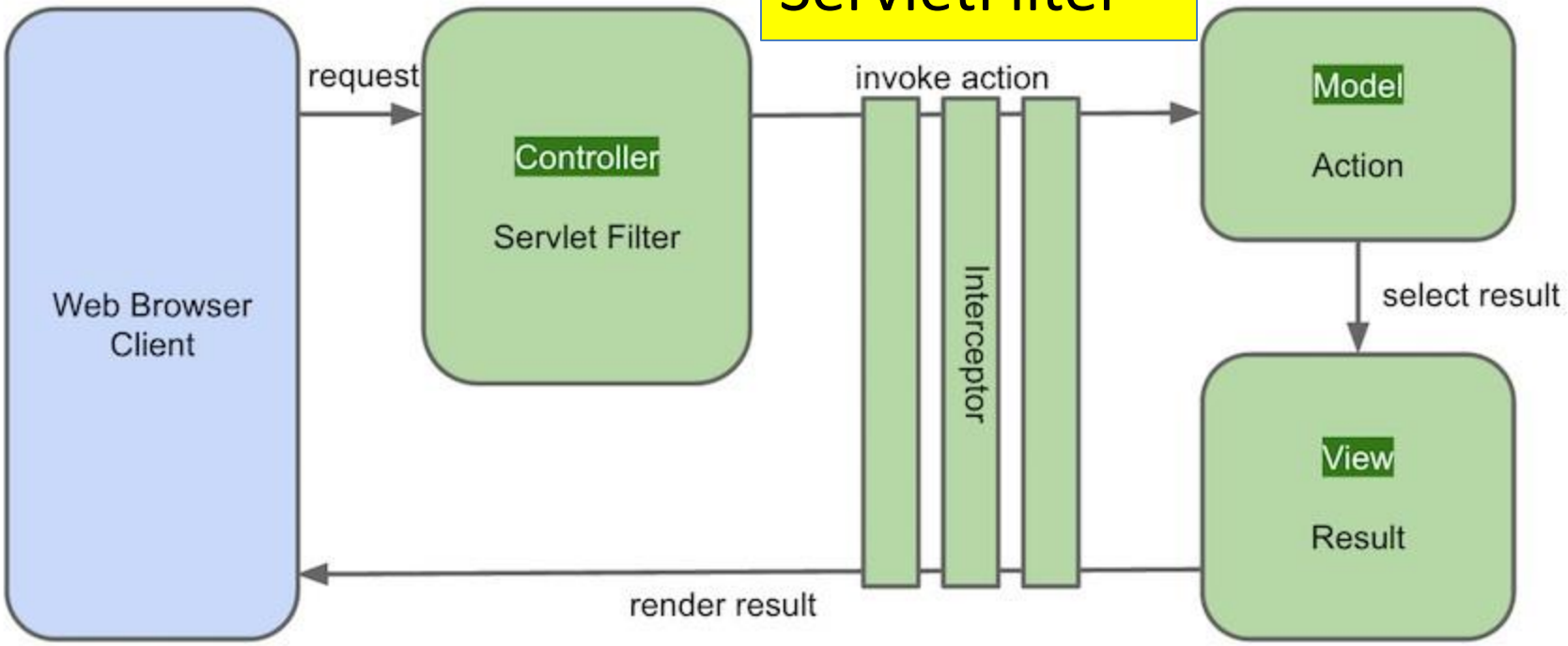
Prof. dr Miroslav Lutovac

mlutovac@viser.edu.rs

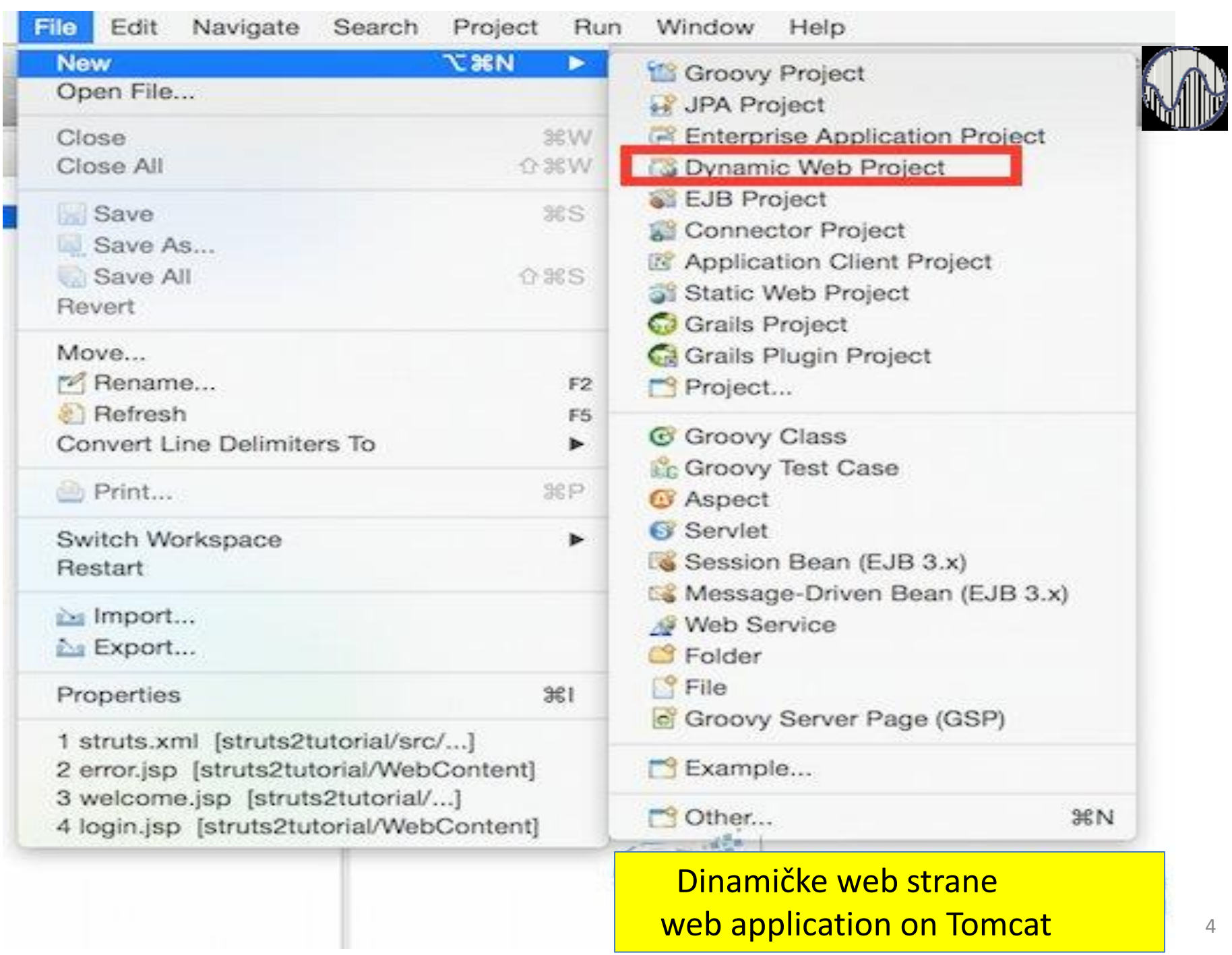
# Java Struts

- Struts je jedno od najpopularnijih okruženja za razvoj Java baziranih aplikacija
- Apache Software Foundation
- Zasnovan na **Model View Controller (MVC)** arhitekturi
- **Model**, interakcija sa standardnim tehnologijama za pristup podacima (JDBC, EJB , Hibernate , iBATIS , Object Relational Bridge
- **View** , radi sa JavaServer Pages , JSTL i JSF , Velocity Templates , XSLT  i drugim sistemima za prezentaciju

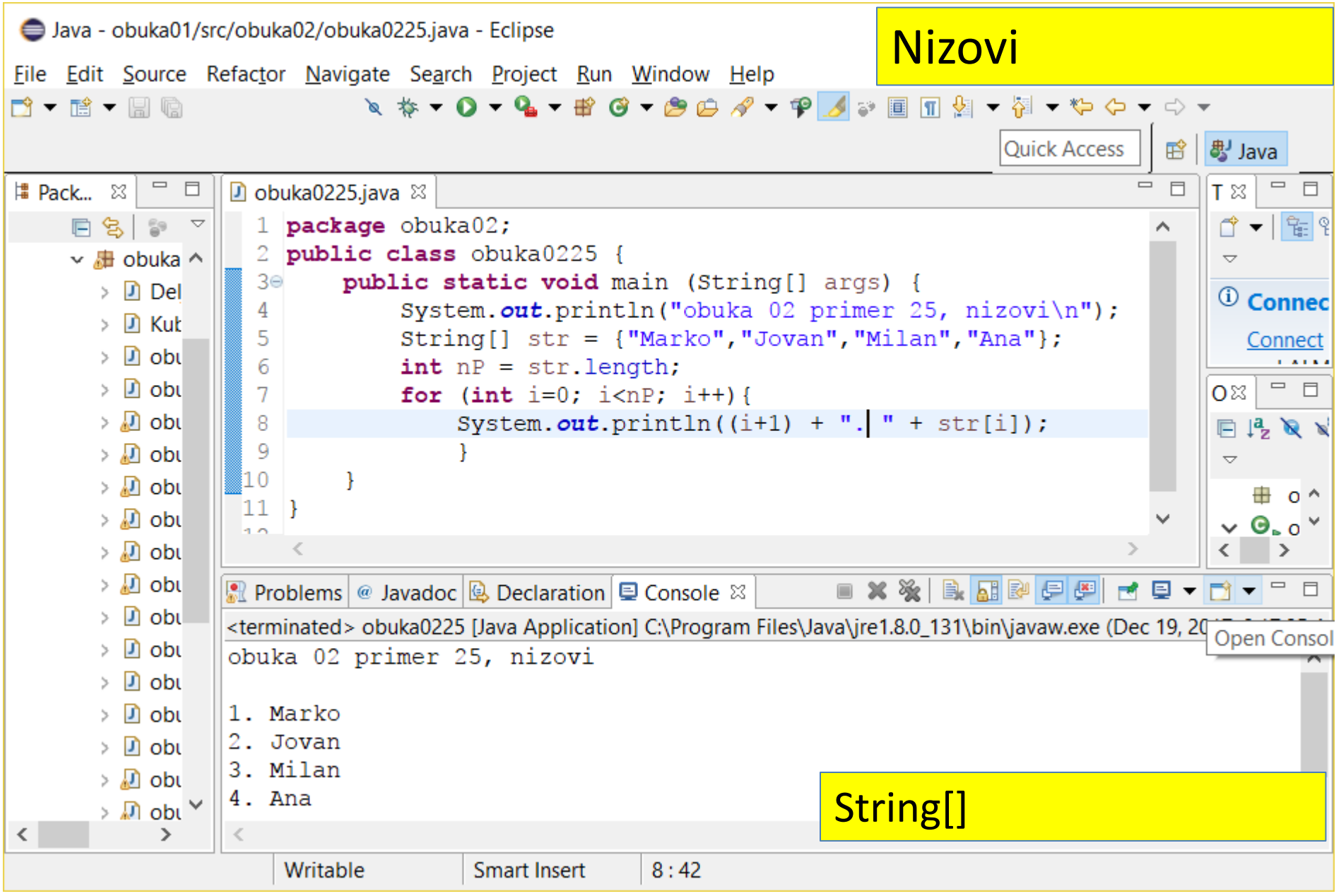
Actions
Results
ServletFilter

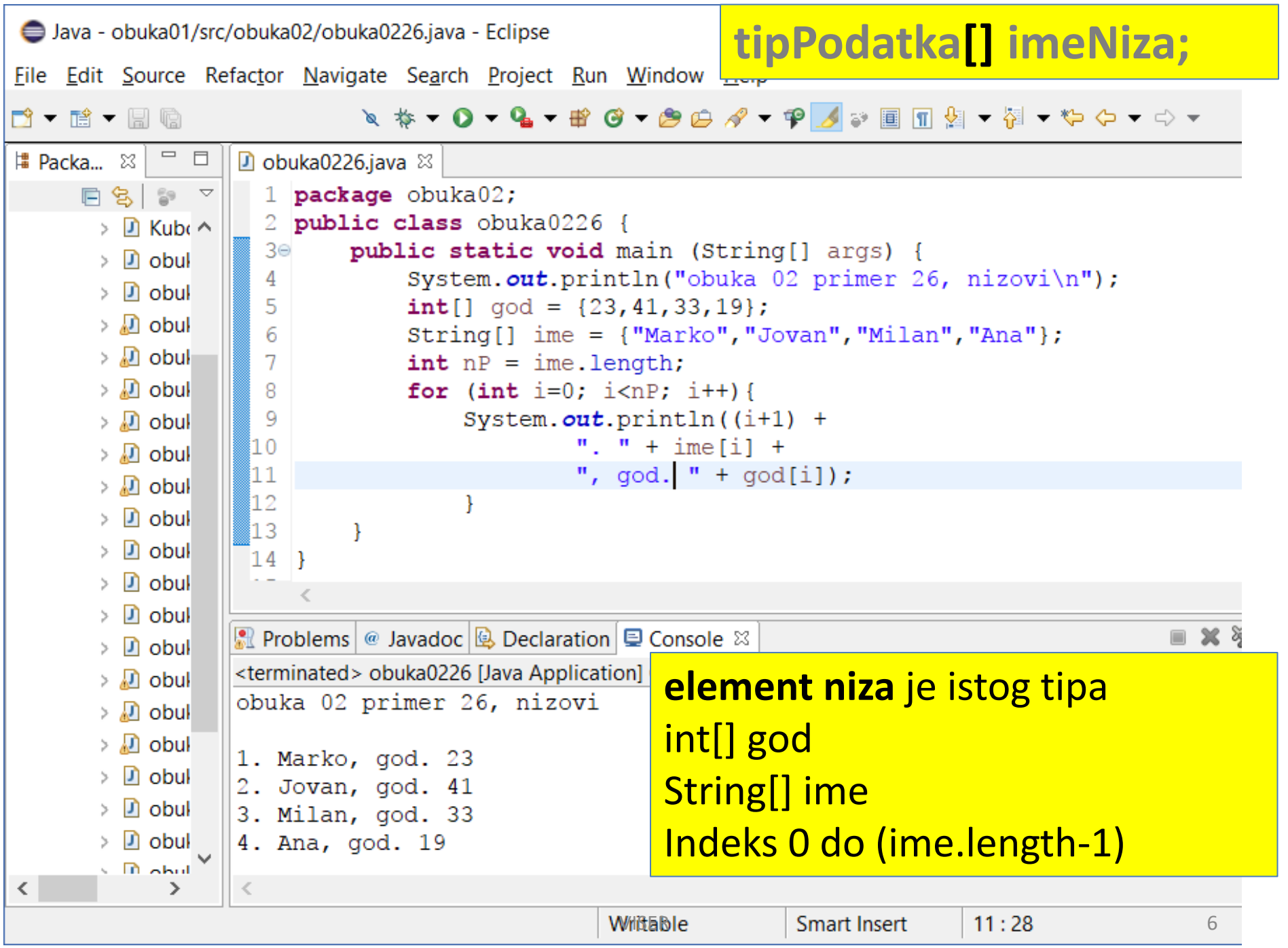



- ✓ The Struts Application Framework obezbeđuje set mehanizama koiji omogućava razvoj veb aplikacija
- ✓ Kombinacijom Java klasa, JSP i mapiranjima XML konfiguracionih fajlova, obezbeđuje se modularan pristup koji se lako održavaju kao veb aplikacije




Dinamičke web strane
web application on Tomcat


Java - obuka01/src/obuka02/obuka0225.java - Eclipse
Nizovi
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
obuka0225.java
1 package obuka02;
2 public class obuka0225 {
3 public static void main (String[] args) {
4 System.out.println("obuka 02 primer 25, nizovi\n");
5 String[] str = {"Marko","Jovan","Milan","Ana"};
6 int nP = str.length;
7 for (int i=0; i<nP; i++){
8 System.out.println((i+1) + ". " + str[i]);
9 }
10 }
11 }
Problems Javadoc Declaration Console
<terminated> obuka0225 [Java Application] C:\Program Files\Java\jre1.8.0_131\bin\javaw.exe
obuka 02 primer 25, nizovi
1. Marko
2. Jovan
3. Milan
4. Ana
String[]
Writable
Smart Insert
8 : 42

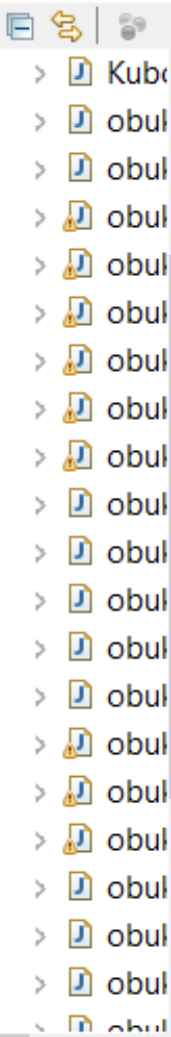

Java - obuka01/src/obuka02/obuka0226.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window
tipPodatka[] imeNiza;
obuka0226.java
1 package obuka02;
2 public class obuka0226 {
3 public static void main (String[] args) {
4 System.out.println("obuka 02 primer 26, nizovi\n");
5 int[] god = {23,41,33,19};
6 String[] ime = {"Marko","Jovan","Milan","Ana"};
7 int nP = ime.length;
8 for (int i=0; i<nP; i++){
9 System.out.println((i+1) +
10 ". " + ime[i] +
11 ", god. " + god[i]);
12 }
13 }
14 }
Problems Javadoc Declaration Console
<terminated> obuka0226 [Java Application]
obuka 02 primer 26, nizovi
1. Marko, god. 23
2. Jovan, god. 41
3. Milan, god. 33
4. Ana, god. 19
element niza je istog tipa
int[] god
String[] ime
Indeks 0 do (ime.length-1)
Writable
Smart Insert
11 : 28

obuka01/src/obuka02/obuka0227.java - Eclipse

**tipPodatka imeNiza[];**

File Source Refactor Navigate Search Project Run Window

Packa...



obuka0227.java

```
1  package obuka02;
2  public class obuka0227 {
3      public static void main (String[] args) {
4          System.out.println("obuka 02 primer 27, nizovi\n");
5          int god[] = {23,41,33,19};
6          String ime[] = {"Marko","Jovan","Milan","Ana"};
7          int nP = ime.length;
8          for (int i=(nP-1); i>=0; i--){
9              System.out.println((nP-i) +
10                     ". " + ime[i] +
11                     ", god. " + god[i]);
12             }
13     }
14 }
```

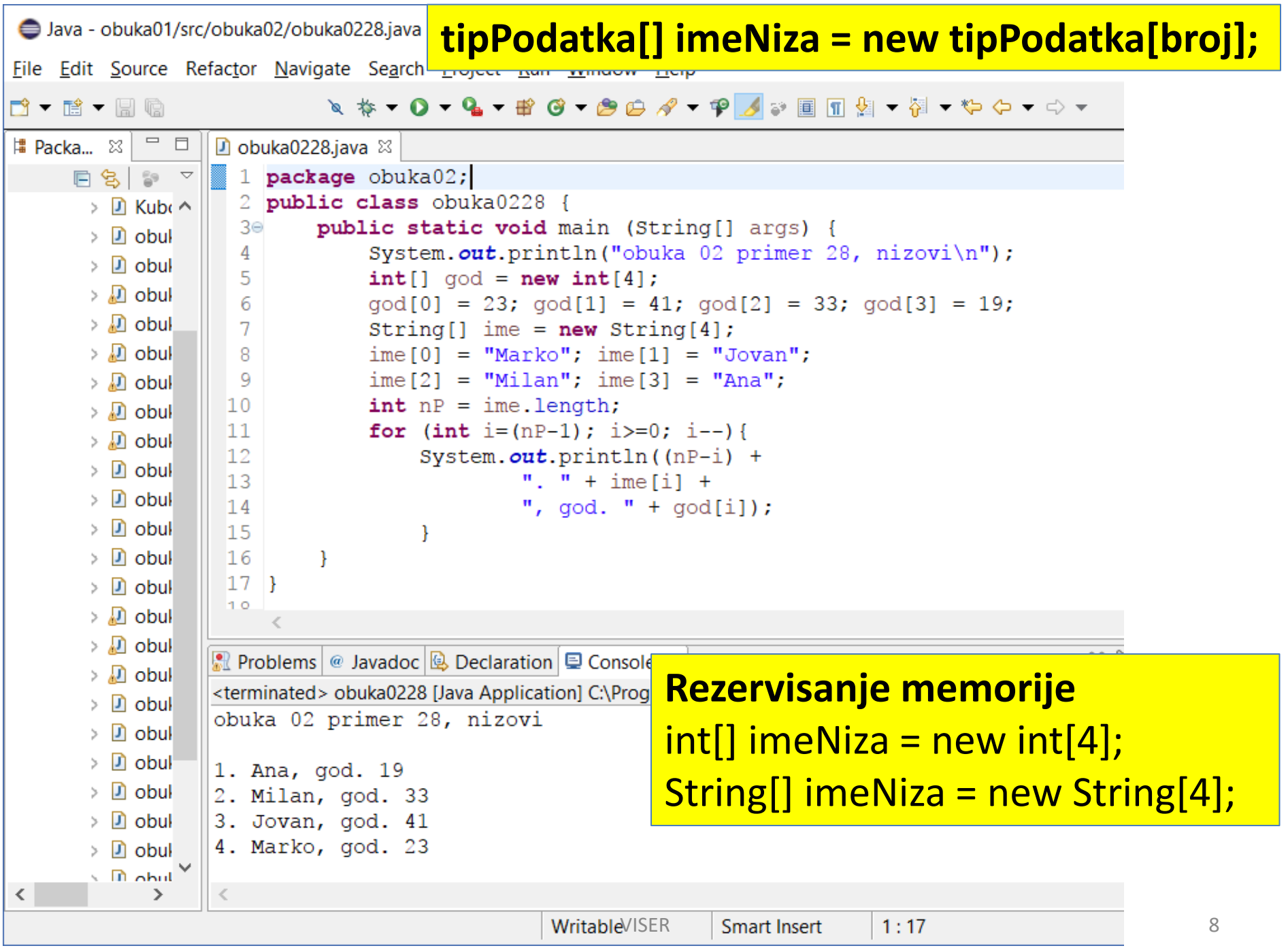
Problems | Javadoc | Declaration | Console

<terminated> obuka0227 [Java Application]

```
obuka 02 primer 27, nizovi

1. Ana, god. 19
2. Milan, god. 33
3. Jovan, god. 41
4. Marko, god. 23
```

**element niza** je istog tipa
int god[]
String ime[]
indeks (ime.length-1) do 0

Writable | Smart Insert | 9 : 37


tipPodatka[] imeNiza = new tipPodatka[broj];
Java - obuka01/src/obuka02/obuka0228.java
File Edit Source Refactor Navigate Search
obuka0228.java
1 package obuka02;
2 public class obuka0228 {
3 public static void main (String[] args) {
4 System.out.println("obuka 02 primer 28, nizovi\n");
5 int[] god = new int[4];
6 god[0] = 23; god[1] = 41; god[2] = 33; god[3] = 19;
7 String[] ime = new String[4];
8 ime[0] = "Marko"; ime[1] = "Jovan";
9 ime[2] = "Milan"; ime[3] = "Ana";
10 int nP = ime.length;
11 for (int i=(nP-1); i>=0; i--){
12 System.out.println((nP-i) +
13 ". " + ime[i] +
14 ", god. " + god[i]);
15 }
16 }
17 }
Problems Javadoc Declaration Console
<terminated> obuka0228 [Java Application] C:\Prog
obuka 02 primer 28, nizovi
1. Ana, god. 19
2. Milan, god. 33
3. Jovan, god. 41
4. Marko, god. 23
Writable SMART INSER
Smart Insert
1 : 17
Rezervisanje memorije
int[] imeNiza = new int[4];
String[] imeNiza = new String[4];

# BubbleSort

obuka0229.java

```java
package obuka02;
public class obuka0229 {
    public static void main (String[] args) {
        System.out.println("obuka 02 primer 29, nizovi\n");
        int[] god = new int[4];
        god[0] = 23; god[1] = 41; god[2] = 33; god[3] = 19;
        String[] ime = new String[4];
        ime[0] = "Marko"; ime[1] = "Jovan";
        ime[2] = "Milan"; ime[3] = "Ana";
        int nP = ime.length;
        int tmpGod = god[0];
        String tmpIme = ime[0];
        for (int j=1;j<nP;j++){
            for (int i=1;i<nP;i++){
                if (god[i-1]>god[i]){
                    tmpGod = god[i-1];
                    god[i-1] = god[i];
                    god[i] = tmpGod;
                    tmpIme = ime[i-1];
                    ime[i-1] = ime[i];
                    ime[i] = tmpIme;
                }
            }
        }
        for (int i=0; i<nP; i++){
            System.out.println((i+1) +
                    ". " + ime[i] +
                    ", god. " + god[i]);
            }
    }
}
```

Problems | Javadoc | Declaration | Console

<terminated> obuka0229 [Java Application] C:\Program Files\Java\jre1

```
obuka 02 primer 29, nizovi

1. Ana, god. 19
2. Marko, god. 23
3. Milan, god. 33
4. Jovan, god. 41
```

Zamenjuje mesta dva uzastopna člana po rastućem poretku

# 2D niz

obuka0230.java

```
1  package obuka02;
2  public class obuka0230 {
3      public static void main (String[] args) {
4          System.out.println("obuka 02 primer 30, 2D niz");
5          int[][] niz2D = new int[3][];
6          int[] b = {23, 41, 47, 19};
7          int[] c = {31, 32, 33, 34};
8          int[] d = {95, 96, 97, 98};
9          niz2D[0] = b;
10         niz2D[1] = c;
11         niz2D[2] = d;
12         int nP1 = niz2D[0].length;
13         int nP2 = niz2D.length;
14         System.out.println(nP1 + ", " + nP2);
15         for (int j=0;j<nP1;j++){
16             for (int i=0;i<nP2;i++){
17                 int tmp = niz2D[0][1];
18                 System.out.println((i) + " " + (j) +
19                         ", xij " + niz2D[i][j]);
20             }
21         }
22     }
23 }
```

```
obuka 02 primer 30, 2D niz
4, 3
0 0, xij 23
1 0, xij 31
2 0, xij 95
0 1, xij 41
1 1, xij 32
2 1, xij 96
0 2, xij 47
1 2, xij 33
2 2, xij 97
0 3, xij 19
1 3, xij 34
2 3, xij 98
```

# 2D niz

obuka0231.java

```
1  package obuka02;
2  public class obuka0231 {
3      public static void main (String[] args) {
4          System.out.println("obuka 02 primer 31, 2D niz");
5          int[][] niz2D = {
6                  {23, 41, 47, 19},
7                  {31, 32, 33, 34},
8                  {95, 96, 97, 98}};
9          int nP1 = niz2D[0].length;
10         int nP2 = niz2D.length;
11         System.out.println(nP1 + ", " + nP2);
12         for (int j=0;j<nP1;j++){
13             for (int i=0;i<nP2;i++){
14                 int tmp = niz2D[0][1];
15                 System.out.println((i) + " " + (j) +
16                         ", xij " + niz2D[i][j]);
17             }
18         }
19     }
20 }
21
```

Problems | Javadoc | Declaration | Console

<terminated> obuka0231 [Java Application] C:\Program Files\Java\jre1.8.0_131\bin\javaw.exe

```
obuka 02 primer 31, 2D niz
4, 3
0 0, xij 23
1 0, xij 31
2 0, xij 95
0 1, xij 41
1 1, xij 32
2 1, xij 96
0 2, xij 47
1 2, xij 33
2 2, xij 97
0 3, xij 19
1 3, xij 34
2 3, xij 98
```

## Citaj.java

**Funkcija, procedura, koja se koristi u drugoj klasi**

```
1  package obuka02;
2  //Citaj.java - Citanje podataka standardnih tipova iz ulaznog toka,
3  //             iz datoteke i s glavnog ulaza.
4
5  import  java.io.*;
6
7  public class Citaj {
8
9  private InputStream ut;     // Ulazni tok iz kojeg se cita.
10 private char c;              // Poslednji procitani znak.
11 private boolean eos;         // Indikator kraja toka.
12
13 public Citaj(InputStream uut) { ut = uut; }  // Inicijalizacija
14                                              //   ulaznim tokom.
15 public Citaj(String ime) throws FileNotFoundException  // Otvaranje
16  { ut = new FileInputStream(ime); }                    //   datoteke.
17
18 public boolean endS() { return eos; }         // Da li je kraj toka?
19
20 public char getChS() {      // Dohvatanje sledeceg znaka.
21  try { int i = ut.read(); return c = (eos = i == -1) ? ' ' : (char)i; }
22    catch (Exception g) { eos = true; return c = ' '; }
23 }

56 public int    IntS     ()  // Citanje jednog podatka tipa int.
57  { String s = StringS (); return !eos ? Integer.parseInt(s) : 0; }
58

83 public static int     Int     () { return gl.IntS     (); }
```

# Unosenje niza Korišćenjem ...

obuka0232.java | Citaj.java

```
1  package obuka02;
2  public class obuka0232 {
3      public static void main (String[] args) {
4          System.out.println("obuka 02 primer 32, metode");
5          System.out.println("   unesi broj clanova niza");
6          int u = Citaj.Int();
7          int i;
8          int a[] = new int[u];
9          int b= a.length;
10         for (i=0; i<b;i++){
11             System.out.println("   unesi za niz a njegov " + i + ". clan ");
12             a[i] = Citaj.Int();
13         }
14         for (i=0;i<b;i++){
15                 System.out.println("a[" + i + "] = " + a[i]);
16         }
17     }
18 }
```

Problems | Javadoc | Declaration | Console

<terminated> obuka0232 [Java Application] C:\Program Files\Java\jre1.8.0_131\bin\javaw.exe (Dec 24, 2017, 1:13:57 PM)

```
obuka 02 primer 32, metode
   unesi broj clanova niza
3
   unesi za niz a njegov 0. clan
4567
   unesi za niz a njegov 1. clan
89
   unesi za niz a njegov 2. clan
21
a[0] = 4567
a[1] = 89
a[2] = 21
```

# Unosenje niza

Java - obuka01/src/obuka02/obuka0232.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

obuka0232.java | Citaj.java

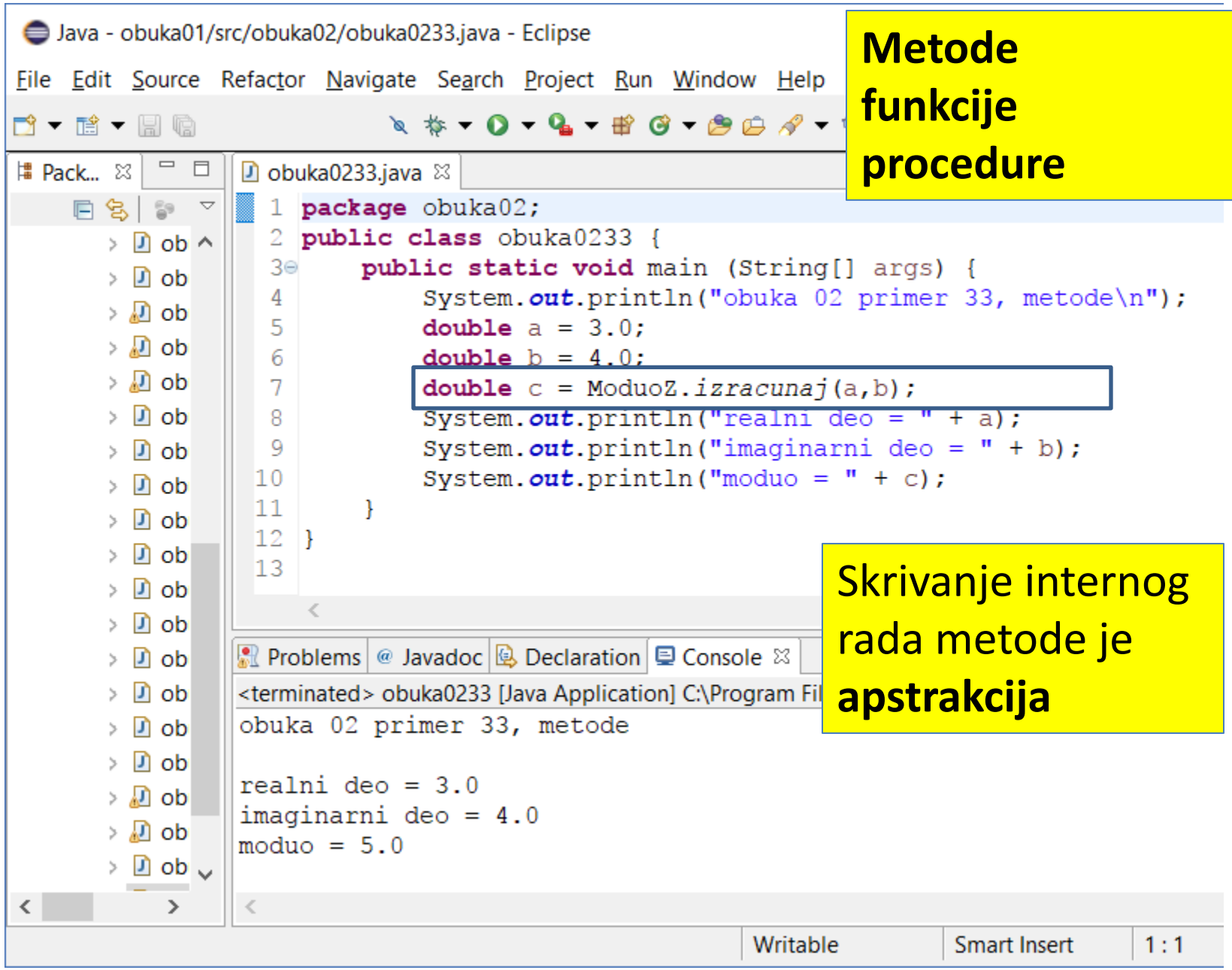
```
1  package obuka02;
2  public class obuka0232 {
3      public static void main (String[] args) {
4          System.out.println("obuka 02 primer 32, metode");
5          System.out.println("   unesi broj clanova niza");
6          int u = Citaj.Int();
7          int i;
8          int a[] = new int[u];
9          int b= a.length;
10         for (i=0; i<b;i++){
11             System.out.println("   unesi za niz a njegov " + i + ". clan ");
12             a[i] = Citaj.Int();
13         }
14         for (i=0;i<b;i++){
15                 System.out.println("a[" + i + "] = " + a[i]);
16         }
17     }
18 }
```

Stop rada

Problems | Javadoc | Declaration | Console

<terminated> obuka0232 [Java Application] C:\Program Files\Java\jre1.8.0_131\bin\javaw.exe (Dec 24, 2017, 1:16:11 PM)

```
obuka 02 primer 32, metode
   unesi broj clanova niza
456
   unesi za niz a njegov 0. clan
11
   unesi za niz a njegov 1. clan
23
   unesi za niz a njegov 2. clan
```




**Metode funkcije procedure**

Skrivanje internog rada metode je **apstrakcija**

# Metode

Java - obuka01/src/obuka02/obuka0234.java - Eclipse

Edit Source Refactor Navigate Search Project Run Window Help

obuka0234.java

```
1  package obuka02;
2  public class obuka0234 {
3      double absValue(double x){
4          if (x<0)
5          return -x;
6          else
7              return x;
8      }
9      public static void main (String[] args) {
10         obuka0234 objekat = new obuka0234();
11         System.out.println("obuka 02 primer 34, metode\n");
12         double broj = -3.0;
13         System.out.println("negativan broj = " + broj);
14         double novi = objekat.absValue(broj);
15         System.out.println("abs broja = " + novi);
16     }
17 }
```

Metoda koja će se koristiti za objekat u klasi

objekat u klasi

objekat u klasi
Koji koristi metodu

Problems | Javadoc | Declaration

<terminated> obuka0234 [Java Application

```
obuka 02 primer 34, metode

negativan broj = -3.0
abs broja = 3.0
```

**tip ime_metode argument_metode**
double absValue (double x)
tip je kao izlaz koji metoda vraća

Writable | Smart Insert | 18 : 1

# Metode

Java - obuka01/src/obuka02/obuka0235.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

obuka0235.java

```
1  package obuka02;
2  public class obuka0235 {
3      double sqrt(double x){
4          double epsilon = 1e-15;
5          double t = x;
6          while (Math.abs(t - x/t) > epsilon
7              t = (x/t + t) / 2.0;
8          }
9          return t;
10     }
11     public static void main (String[] args) {
12         obuka0235 objekat = new obuka0235();
13         System.out.println("obuka 02 primer 35, metode\n");
14         double broj = 2.0;
15         System.out.println("zadati broj = " + broj);
16         double novi = objekat.sqrt(broj);
17         System.out.println("kvadratni koren broja = " + novi);
18         double novi2 = Math.sqrt(broj);
19         System.out.println("koriscenjem Math.sqrt = " + novi2);
20     }
21 }
```

Metoda koja koristiti algoritam za izračunavanje sqrt()

U metodi se koristi **void** ako se ništa ne vraća, već. na primer. prikazuje rezultat na konzoli

Problems Javadoc Declaration

<terminated> obuka0235 [Java Application]

```
obuka 02 primer 35, metode

zadati broj = 2.0
kvadratni koren broja = 1.414213562373095
koriscenjem Math.sqrt = 1.4142135623730951
```

VISER

Writable Smart Insert 19 : 50

# Java objekti i klase 1

- **Tipovi podataka**, int, double, boolean, String
- **Promenljive**, pamte vrednosti određenog tipa
- **Nizovi**, pamte više vrednosti istog tipa
- **Kontrolne strukture**, grananje i petlje
- **Metode**, delovi koda kojima mogu da se proslede argumenti i da nešto izračunaju, odrade

# Java objekti i klase 2

- U nekim slučajevima se podaci različitog tipa pamte zajedno **(String[] imena, int[] godine)**
- Metode mogu da vrate samo rezultat jednog tipa
- Objekti su alati za primenu apstrakcije
- Koje detalje treba sakriti, a koje prikazati?

# Java objekti i klase 3

- Stanje, osobine objekta, promenljive koje pamtimo u okviru objekta (field, stste)
- Ponašanje, nešto što objekat može da uradi, metoda koja se definiše u objektu (methods)

# Java objekti i klase 4

- U realnom životu, automobil je objekt
- Automobil ima stanja, osobine (težina, boja), i ponašanje, metode (kreni i stani)
- Svi automobili imaju iste osobine, ali se osobine razlikuju od auta do auta
- Svi automobili imaju iste metode, ali se metode izvršavaju u različito vreme

# Java objekti i klase 5

| **Objekat, Object** | **Stanje, osobina, Properties** | **Ponašanje, Methods** |
|---|---|---|
| auto | ime, model, boja | kreni, vozi |

auto.ime = Fiat

auto.model = 500L

auto.tezina = 850kg

auto.boja = crvena

auto.kreni()

auto.vozi()

auto.koci()

auto.stani()

**Modularnost, sakrivanje informacija, Važan je interfejs**

# Java klase 1

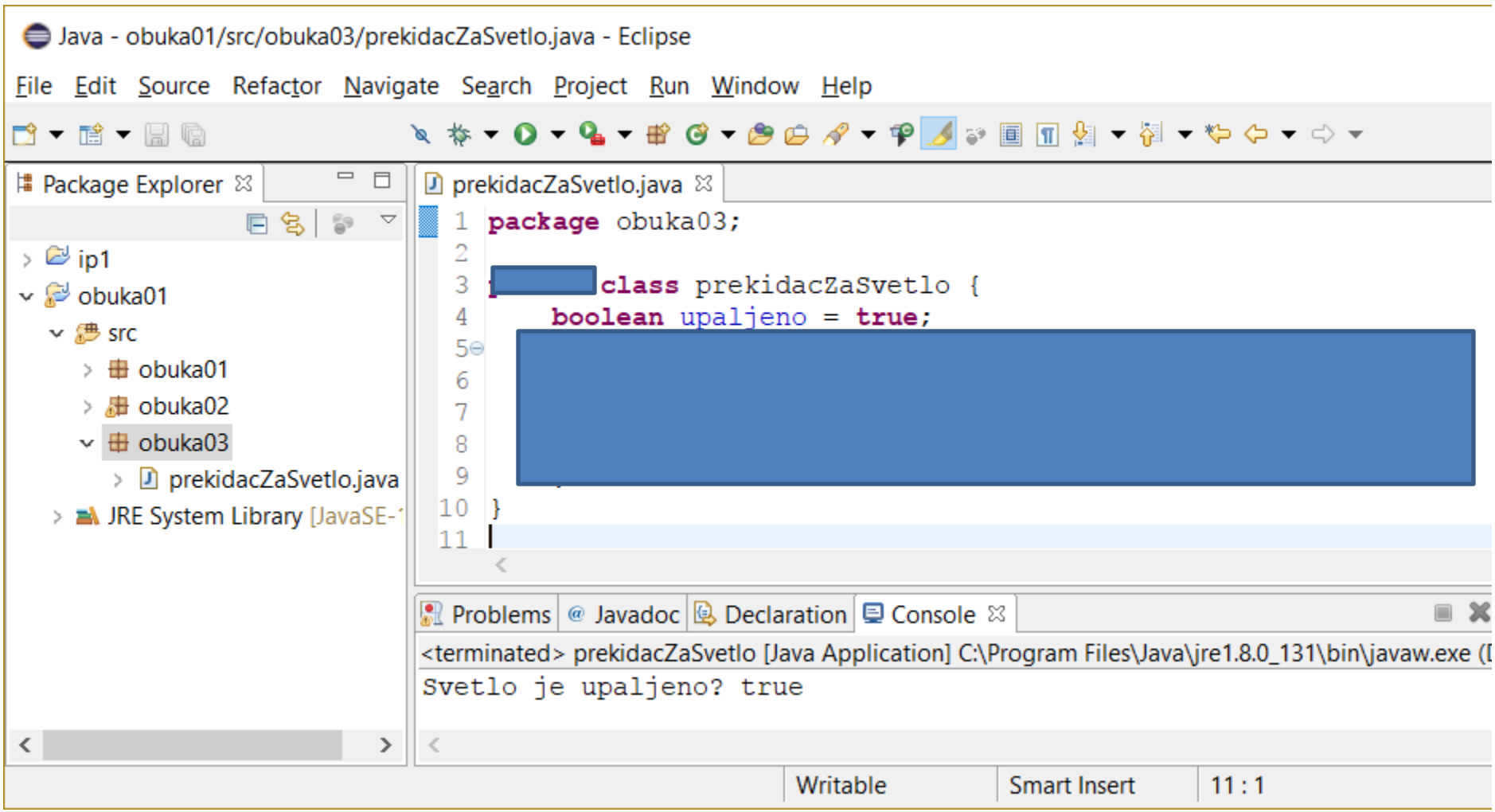
- Klasa je neophodna da bi u okviru nje definisali objekat
- Klasa je uzorak, kalup da bi se napravio objekat
- Klasa sadrži
  - Članove podataka (osobine, karakteristike objekta, klase)
  - Metode (ponašanje objekta klase)
  - Konstruktore (specijalne metode)

# Java klase 2

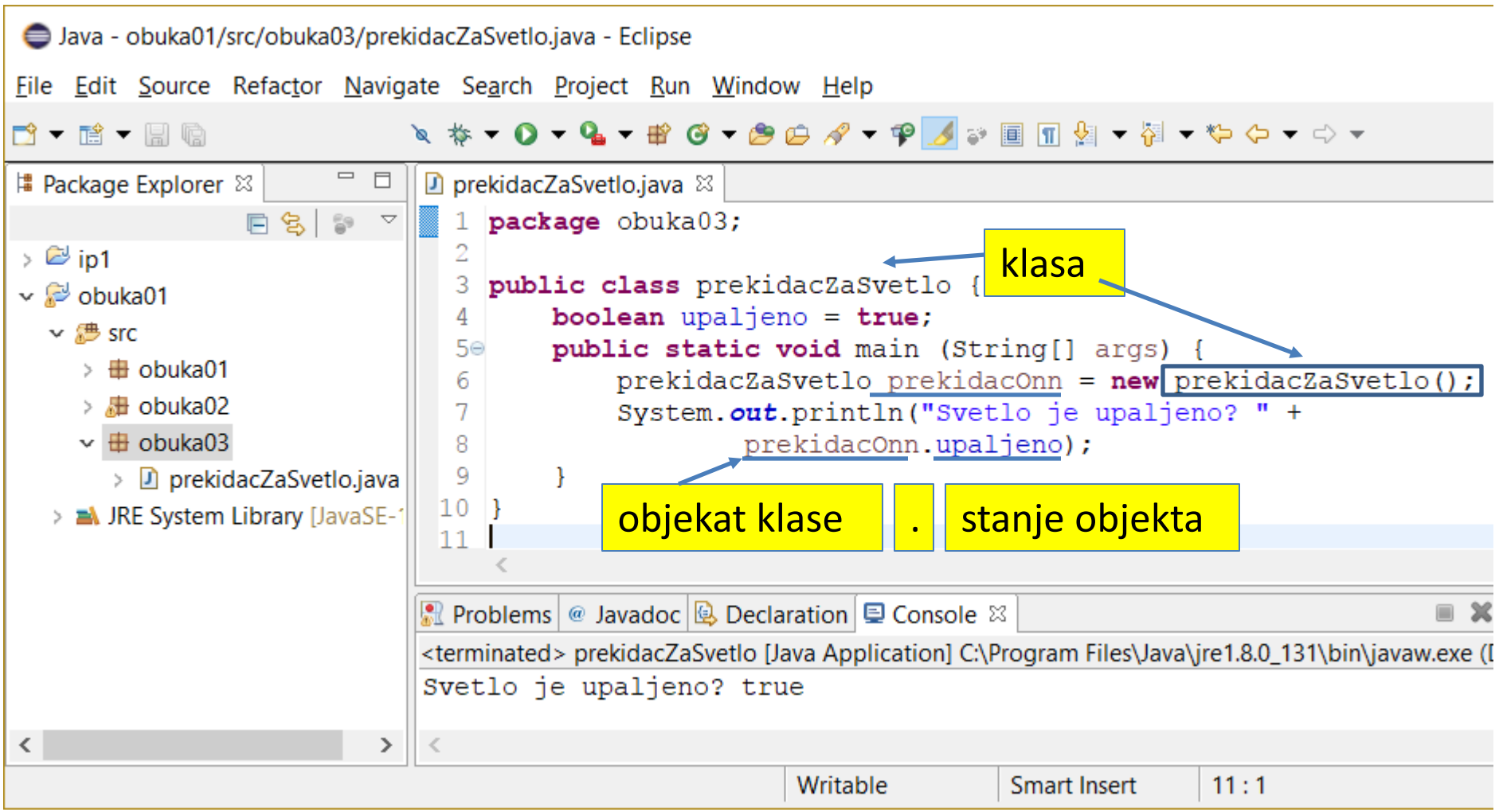
- Anatomija klase
  - Članove podataka (osobine, karakteristike objekta, klase)
  - Metode (ponašanje objekta klase)
  - Konstruktore (specijalne metode)

```
public class imeKlase {
Članovi podataka
Konstruktori
Metode
}
```

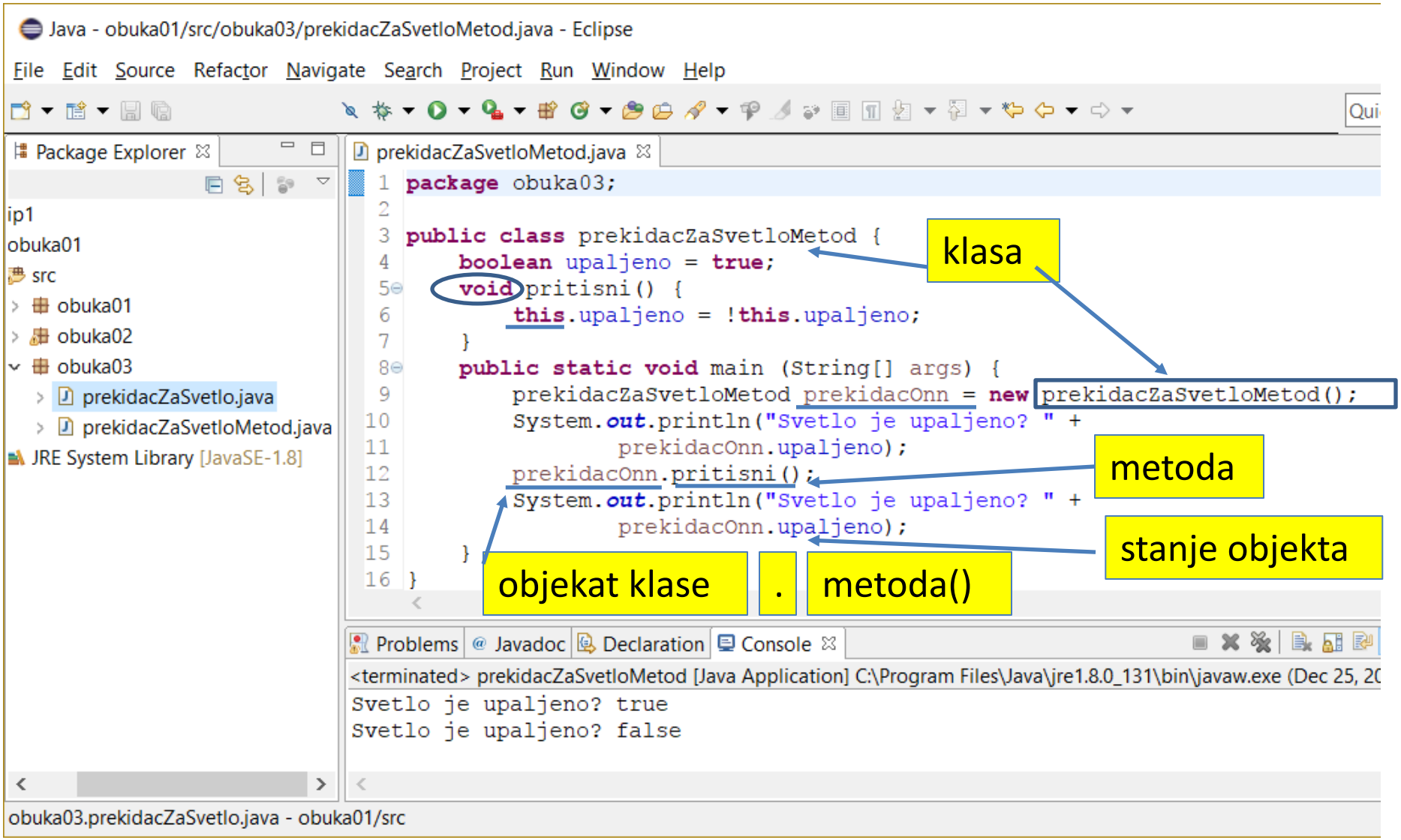
# Primer jednostavne klase



# Primer jednostavne klase, prikaz rezultata, stanje



# Primer jednostavne klase, stanje i metoda



# Primer jednostavne klase, stanje, metoda i konstruktor

Java - obuka01/src/obuka03/prekidacSvetlaKonst.java - Eclipse

File Edit Source Re

prekidacSvetlaKonst.java

```
1  package obuka03;
2
3  public class prekidacSvetlaKonst {
4      boolean upaljeno;
5      void pritisni() {
6          this.upaljeno = !this.upaljeno;
7      }
8      prekidacSvetlaKonst (boolean pocetnoStanje){
9          this.upaljeno = pocetnoStanje;
10     }
11     public static void main (String[] args) {
12         boolean pocetnoStanje=false;
13         prekidacSvetlaKonst prekidacOnn = new prekidacSvetlaKonst(pocetnoStanje);
14         System.out.println("Svetlo je upaljeno? " +
15                 prekidacOnn.upaljeno);
16         prekidacOnn.pritisni();
17         System.out.println("Svetlo je upaljeno? " +
18                 prekidacOnn.upaljeno);
19     }
20 }
21
```

Početno stanje

Objekat klase ima stanje, metodu i početno stanje koje može da postavi stanje na početku, a zatim se stanje menja metodom

Problems | Javadoc | Declaratic

<terminated> prekidacZaSvetloMetod

```
Svetlo je upaljeno? true
Svetlo je upaljeno? false
```

Ako se ne specificira da li je polje ili metod public ili private, podrazumeva se da je public

Writable | Smart Insert | 21 : 1

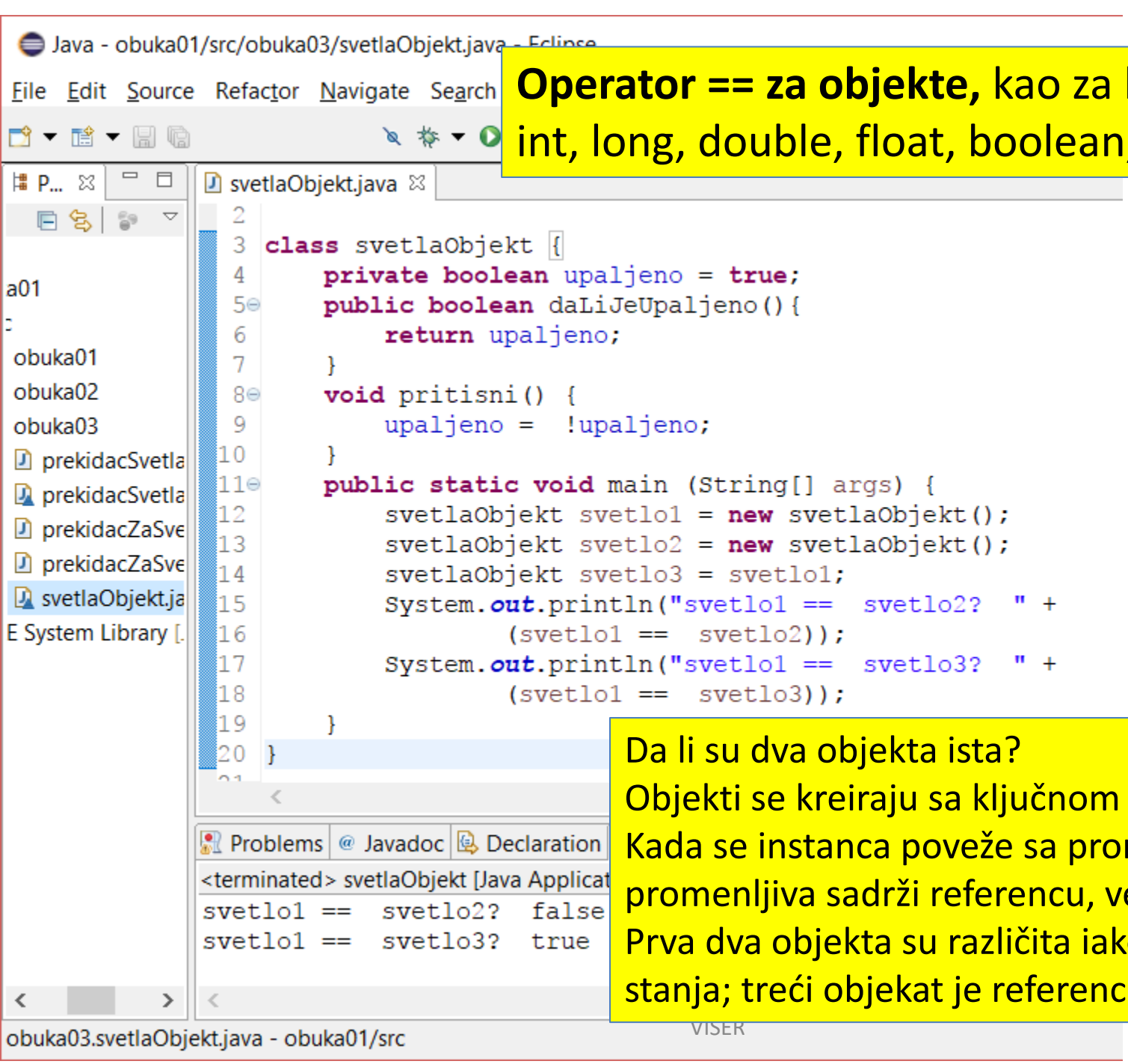
# Primer jednostavne klase, stanje, metoda, private

Polje upaljeno je definisano kao private
Ako se iz druge klase pristupa ovom polju, desiće se greška

```
1  package obuka03;
2
3  class prekidacSvetlaPriv {
4      private boolean upaljeno = true;
5      public boolean daLiJeUpaljeno(){
6          return upaljeno;
7      }
8      void pritisni() {
9          upaljeno =  !upaljeno;
10     }
11     public static void main (String[] args) {
12         prekidacSvetlaPriv prekidacOnn = new prekidacSvetlaPriv();
13         System.out.println("Svetlo je upaljeno? " +
14                 prekidacOnn.upaljeno);
15         prekidacOnn.pritisni();
16         System.out.println("Svetlo je upaljeno? " +
17                 prekidacOnn.upaljeno);
18     }
19 }
```

```
<terminated> prekidacSvetlaPriv [Java
Svetlo je upaljeno? true
Svetlo je upaljeno? false
```

Ne sme se pristupati private polju druge klase!




**Operator == za objekte,** kao za byte, short, int, long, double, float, boolean, char,

Da li su dva objekta ista?
Objekti se kreiraju sa ključnom reči **new**
Kada se instanca poveže sa promenljivom, promenljiva sadrži referencu, vezu, sa objektom
Prva dva objekta su različita iako imaju identična stanja; treći objekat je referenca na isti objekat

# Objektno orijentisani jezici

- Svakoj apstrakciji problema odgovara jedna klasa
- **Klasa** predstavlja prototip iz koga se kreira instanca
- **Objekat** predstavlja instancu klase, instanciranjem
- **Apstrakcija** – zapažanje osobina i ponašanje objekta
- **Enkapsulacija** - implementacija koja dovodi do ponašanja
- Enkapsulacija, da sakrije osobine klase koje nisu bitne, da sakrije strukturu klase

# Objektno orijentisani jezici

- Klasa ima
  1. **Interfejs**, spoljašnji izgled klase, sredstvo preko koga se ustanovljava da li klasa ima željeno ponašanje
  2. **Implementacija**, realizacija svih mehanizama koji dovode do željenog ponašanja
- Implementacija u programskim jezicima obuhvata kreiranje klase koje su identifikovane apstrakcijom
- Svaka klasa ima članove klase, podatke i funkcije

# Objektno orijentisani jezici

- Iz klase se pomoću šablona, uzorka, kalupa, proizvode instance – objekti – sa kojima se funkcionalnost složenog sistema preko interfejsa objekata (komunikacija)
- Uloga enkapsulacije da razdvoji javni i privatni deo klase
- **Javni deo** klase čini interfejs ka okolini preko koga se pristupa instancama klase
- **Privatni delovi** čine implementacione detalje i podatke koji za korisnika klase nisu važni pa ne mora ni da zna (vozač auta zna da auto ima pogonski deo, ali ne i kako radi)

# Objektno orijentisani jezici

- **Modularnost**, generisanje modula koji mogu da se kompajliraju odvojeno, ali imaju dobro definisane veze sa drugim modulima
- Za module je važno uočiti logičke celine
- Ugneždavanje, podpaketi, hijerarhija (klasa, objekata)
- **Hijerarhija** se ostvaruje preko nasleđivanja (inheritance)
- Jedna apstrakcija može biti vrsta druge apstrakcije sa dodatim osobinama
- Dodavanje osobenosti – autonomni entiteti

# Objektno orijentisani jezici

- **Tipiziranje**, svaka klasa definiše novi tip
- Instanciranjem klase dolazi se do objekta te klase
- **Vezivanje**
  - **Statičko**, objekti odgovaraju na poruke prema tipovima u vreme kompajliranja
  - **Dinamičko**, objekti odgovaraju na poruke prema svom tipu u vreme izvršavanja programa

# Priroda objekata

- Objekat je entitet koji ima stanja, ponašanja, identitet
- Strukture i ponašanja objekata se definišu preko klasa
- Objakat – opiplivo, vidljivo i **misaono (opaziti)**
- Stanje objekta su osobine i trenutne vrednosti osobina
- Stanje se izražava preko promenljivih definisanih u klasi
- Akcija nad objektom = šalje mu se poruka, menja stanje
- Stanje objekta je kumulativni rezultat ponašanja
- Mtoda objekta – poziv neke funkcije koja je definisana u klasi objekta

# Priroda objekata

- Vrste metoda
  - Modifikatori, menjaju stanje objekta
  - Selektori, pristupaju stanju bez promene stanja
  - Iteratori, pristupaju svim delovima objekta na definisan način
- Konstruktori – kreiranje objekta
- Destruktori – uništavanje objekta i oslobađanje prostora koji je objekat zauzimao

# Veze između objekata

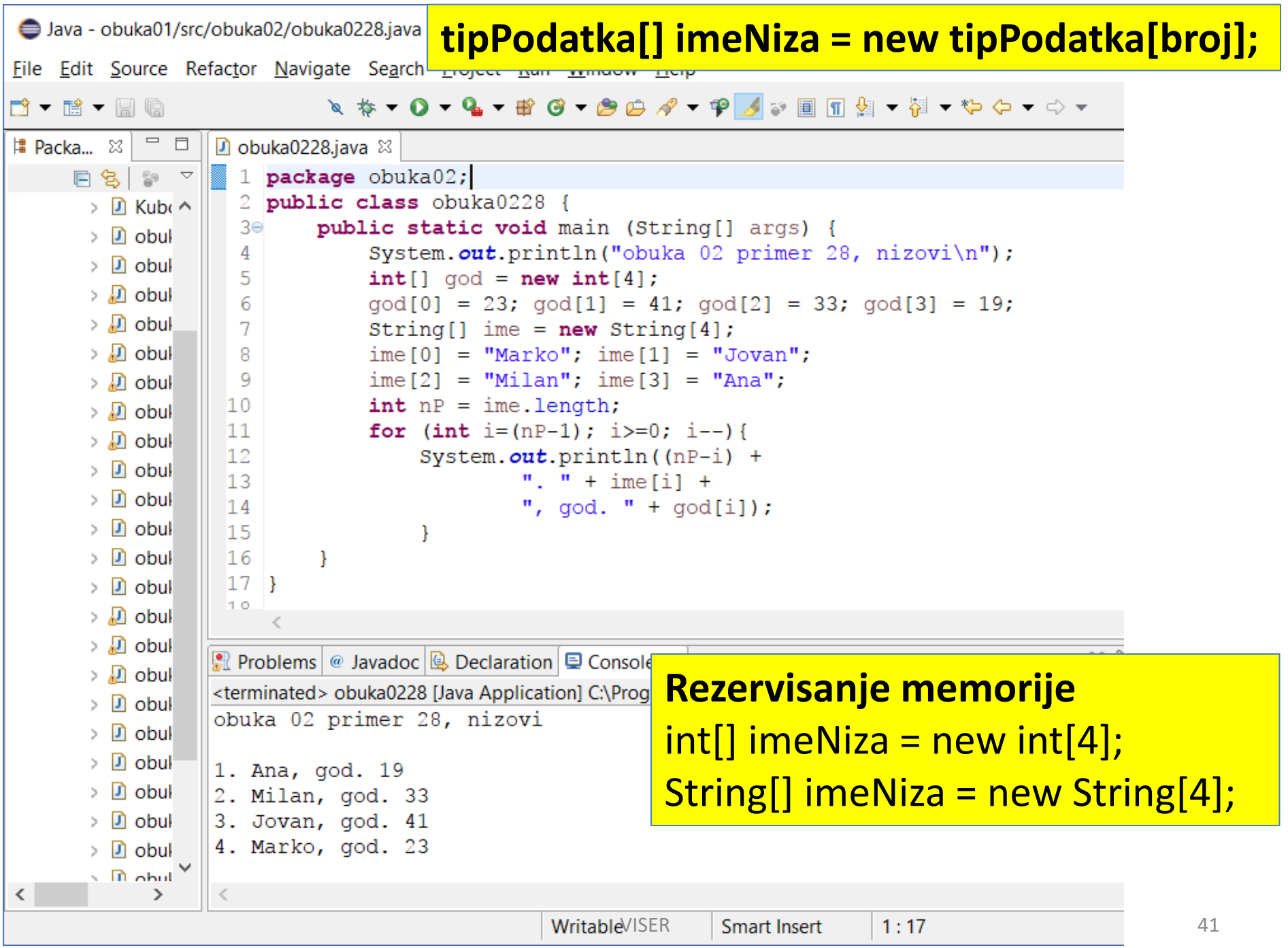
- Komunikacijom između objektima se postiže funkcionalnost
- Linkovi – fizičke i konceptualne veze između objekata
  - Aktivni objekti, operišu nad drugima, ne nad njima, objekat čiji se metod poziva
  - Serveri, ne operišu nad drugima, operiše se na njima, objekat koji se predaje kao argument metode
  - Agenti, operišu nad drugima, i drugi nad njima objekat čiji se metod poziva
- Da objekat bude vidljiv drugom objektu

# Veze između objekata

- Vidljivost objekata
  - Globalan za klijenta
  - Parametar klijentovih operacija
  - Deo klijentskog objekta
  - Lokalno deklarisan u operaciji klijenta
- Agregacij – jedan objekat je deo drugog, utiče na sveukupno stanje
- Agregat, kontejner je objekat koji sadrži druge objekte
- Objekat koji sadrži druge kao atribute

# Priroda klase

- objekata


tipPodatka[] imeNiza = new tipPodatka[broj];
Java - obuka01/src/obuka02/obuka0228.java
File Edit Source Refactor Navigate Search
obuka0228.java
1 package obuka02;
2 public class obuka0228 {
3 public static void main (String[] args) {
4 System.out.println("obuka 02 primer 28, nizovi\n");
5 int[] god = new int[4];
6 god[0] = 23; god[1] = 41; god[2] = 33; god[3] = 19;
7 String[] ime = new String[4];
8 ime[0] = "Marko"; ime[1] = "Jovan";
9 ime[2] = "Milan"; ime[3] = "Ana";
10 int nP = ime.length;
11 for (int i=(nP-1); i>=0; i--){
12 System.out.println((nP-i) +
13 ". " + ime[i] +
14 ", god. " + god[i]);
15 }
16 }
17 }
Problems Javadoc Declaration Console
<terminated> obuka0228 [Java Application] C:\Prog
obuka 02 primer 28, nizovi
1. Ana, god. 19
2. Milan, god. 33
3. Jovan, god. 41
4. Marko, god. 23
Writable Smart Insert 1 : 17
Rezervisanje memorije
int[] imeNiza = new int[4];
String[] imeNiza = new String[4];

# Nizovi

- Nedostatak, ne može se menjati broj članova niza kada se deklariše

**tipPodatka[] imeNiza = new tipPodatka[broj];**

Java - obuka01/src/obuka03/obuka0301.java -

File Edit Source Refactor Navigate Search

obuka0301.java

```
1  package obuka03;
2  public class obuka0301 {
3      public static void main (String[] args) {
4          System.out.println("obuka 03 primer 01, liste\n");
5          int[] god = new int[4];
6          god[0] = 23; god[1] = 41; god[2] = 33; god[3] = 19;
7          int nP = god.length;
8          for (int i=0; i<nP; i++){
9              System.out.println((i+1) +
10                     ". god. " + god[i]);
11             }
12         god[nP] = 23;
13         for (int i=0; i<nP+1; i++){
14             System.out.println((i+1) +
15                     ". god. " + god[i]);
16             }
17     }
18 }
```

Problems Javadoc Declaration Console

<terminated> obuka0301 [Java Application] C:\Program Fi

```
obuka 03 primer 01, liste

1. god. 23
2. god. 41
3. god. 33
4. god. 19
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
        at obuka03.obuka0301.main(obuka0301.java:12)
```

**Greška**

god[nP] = 23;

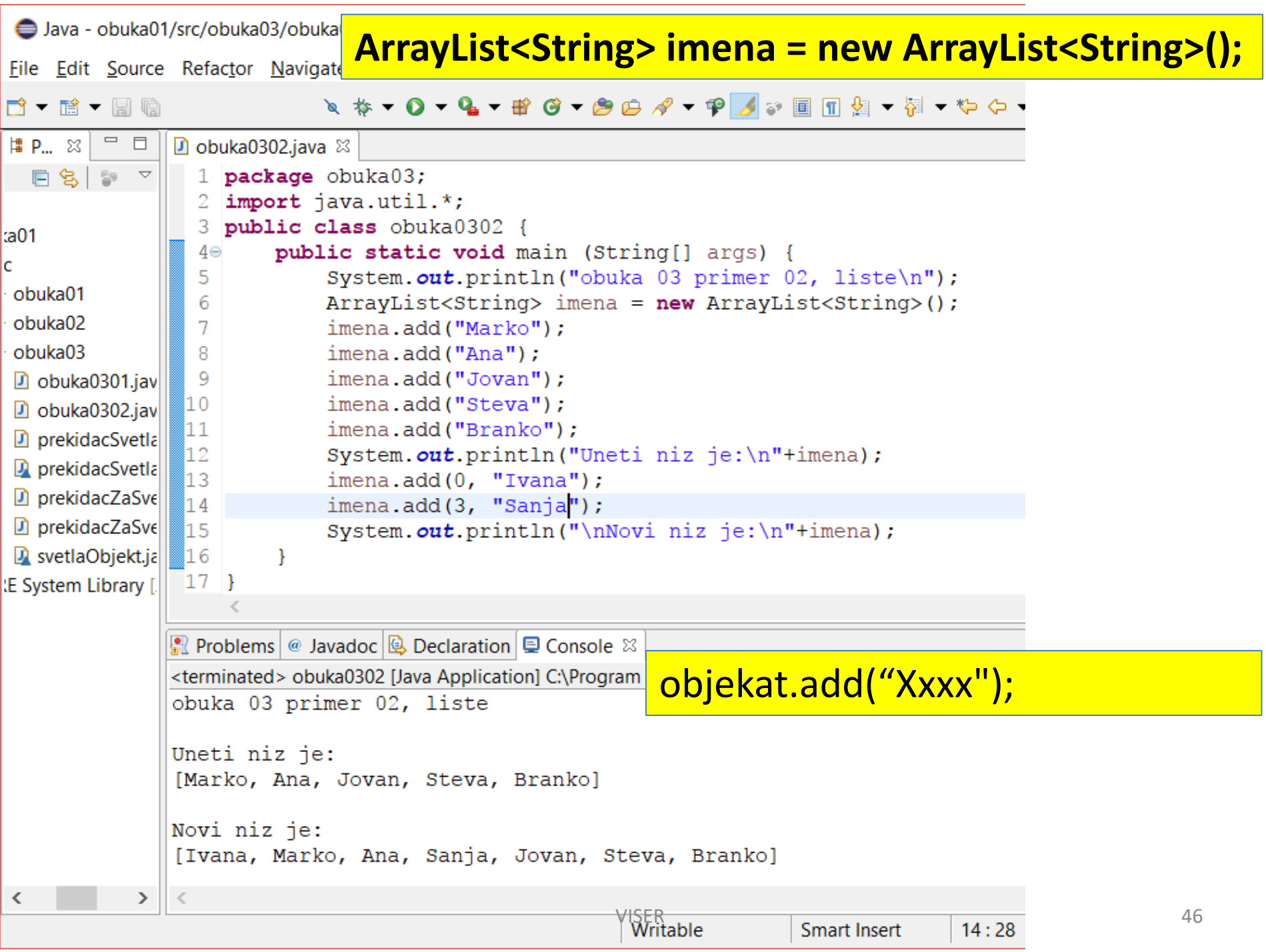
# Liste

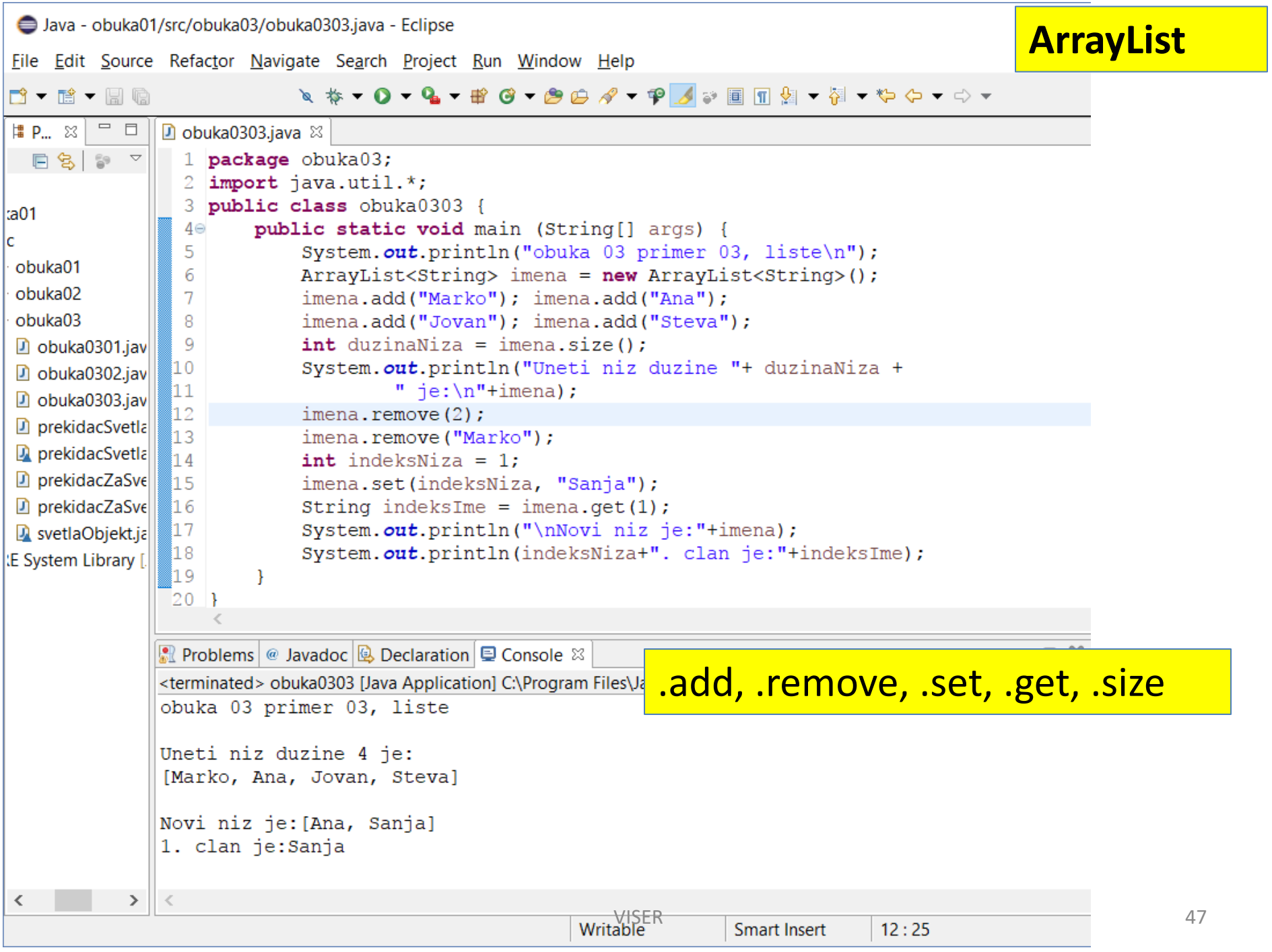
- **Array**
- Jedan tip podataka
- Fiksna dužina
- Imaju indekse
- Brže osnovne operacije
- Ne treba import

- **ArrayList**
- Objekti različitih klasa
- Promenljiva dužina
- Nemaju indekse
- Sporije osnovne operacije
- import java.util.*;

# Liste

- ArrayList<String> imeObjekta = new ArrayList<String>();
- imeObjekta.**add**("Xxxx");
  - imeObjekta.add(3, "Xxxx"); // na određenom mestu
- imeObjekta.**get**("Xxxx");
- imeObjekta.**set**(2,"Xxxx");
- imeObjekta.**size**("Xxxx");
- imeObjekta.**remove**("Xxxx");
  - imeObjekta.remove(0); // na određenom mestu
- imeObjekta. **contains**("Xxxx");
- imeObjekta. **clear**();


Java - obuka01/src/obuka03/obuka
File Edit Source Refactor Navigate
ArrayList<String> imena = new ArrayList<String>();
obuka0302.java
1 package obuka03;
2 import java.util.*;
3 public class obuka0302 {
4 public static void main (String[] args) {
5 System.out.println("obuka 03 primer 02, liste\n");
6 ArrayList<String> imena = new ArrayList<String>();
7 imena.add("Marko");
8 imena.add("Ana");
9 imena.add("Jovan");
10 imena.add("Steva");
11 imena.add("Branko");
12 System.out.println("Uneti niz je:\n"+imena);
13 imena.add(0, "Ivana");
14 imena.add(3, "Sanja");
15 System.out.println("\nNovi niz je:\n"+imena);
16 }
17 }
obuka01
obuka02
obuka03
obuka0301.jav
obuka0302.jav
prekidacSvetla
prekidacSvetla
prekidacZaSve
prekidacZaSve
svetlaObjekt.ja
E System Library
Problems Javadoc Declaration Console
<terminated> obuka0302 [Java Application] C:\Program
obuka 03 primer 02, liste
Uneti niz je:
[Marko, Ana, Jovan, Steva, Branko]
Novi niz je:
[Ivana, Marko, Ana, Sanja, Jovan, Steva, Branko]
objekat.add("Xxxx");
Writable
Smart Insert
14 : 28




# ArrayList

.add, .remove, .set, .get, .size

# ArrayList

Java - obuka01/src/obuka03/obuka0304.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

obuka0304.java

```
1  package obuka03;
2  import java.util.*;
3  public class obuka0304 {
4      public static void main (String[] args) {
5          System.out.println("obuka 03 primer 04, liste\n");
6          ArrayList<String> imena = new ArrayList<String>();
7          imena.add("Marko"); imena.add("Ana");
8          imena.add("Jovan"); imena.add("Steva");
9          int duzinaNiza = imena.size();
10         System.out.println("Uneti niz duzine "+ duzinaNiza +
11                 " je:\n"+imena);
12         imena.remove(2);
13         imena.remove("Marko");
14         int indeksNiza = imena.indexOf("Ana");
15         imena.set(indeksNiza, "Sanja");
16         String indeksIme = imena.get(indeksNiza);
17         System.out.println("\nNovi niz je:"+imena);
18         System.out.println(indeksNiza+". clan je:"+indeksIme);
19     }
20 }
```

Problems | Javadoc | Declaration | Console

<terminated> obuka0304 [Java Application] C:\Program Files\Ja

```
obuka 03 primer 04, liste

Uneti niz duzine 4 je:
[Marko, Ana, Jovan, Steva]

Novi niz je:[Sanja, Steva]
0. clan je:Sanja
```

.indexOf(), .contains()

# ArrayList

Java - obuka01/src/obuka03/obuka0305.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

obuka0305.java

```
1  package obuka03;
2  import java.util.*;
3  public class obuka0305 {
4      public static void main (String[] args) {
5          System.out.println("obuka 03 primer 05, liste\n");
6          ArrayList<String> imena = new ArrayList<String>();
7          imena.add("Marko"); imena.add("Ana");
8          imena.add("Jovan"); imena.add("Steva");
9          int duzinaNiza = imena.size();
10         System.out.println("Uneti niz duzine "+ duzinaNiza +
11                 " je:\n"+imena);
12         boolean imaIme = imena.contains("Marko");
13         System.out.println("ima ime Marko? "+imaIme);
14         int indeksNiza = imena.indexOf("Ana");
15         imena.set(indeksNiza, "Sanja");
16         System.out.println("\nNovi niz je:"+imena);
17         imena.clear();
18         System.out.println("niz je obrisan:"+imena);
19     }
20 }
```

Problems | Javadoc | Declaration | Console

<terminated> obuka0305 [Java Application] C:\Program Files\Ja

```
obuka 03 primer 05, liste

Uneti niz duzine 4 je:
[Marko, Ana, Jovan, Steva]
ima ime Marko? true

Novi niz je:[Marko, Sanja, Jovan, Steva]
niz je obrisan:[]
```

.clear(), .contains()

Writable | Smart Insert | 16 : 1

# LinkedList

- Ulančane liste,
  svaki element sadrži vrednost i vezu ka sledećem elementu
- Lista može da sadrži vezu ka prvom i poslednjem mestu, **kružne**
- **LinkedList**<String> imeObjekta = **new** LinkedList<String>();
- imeObjekta.**add**("Xxxx");
  - imeObjekta.**add**(2,"Xxxx");
- imeObjekta.**get**(0);
- imeObjekta.**set**(2,"Xxxx");
- imeObjekta.**removeFirst();** ;
  - imeObjekta. **removeLast**("Xxxx");

# LinkedList

Java - obuka01/src/obuka03/obuka0306.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

obuka0306.java

```
1  package obuka03;
2  import java.util.*;
3  public class obuka0306 {
4      public static void main (String[] args) {
5          System.out.println("obuka 03 primer 06, Linked List\n");
6          LinkedList<String> linkedlist = new LinkedList<String>();
7          /*add(String Element) is used for adding
8          * the elements to the linked list*/
9          linkedlist.add("stavka03");
10         linkedlist.add("stavka02");
11         linkedlist.add("stavka01");
12         System.out.println("u Linked List je: " +linkedlist);
13         /*Add First and Last Element*/
14         linkedlist.addFirst("stavkaPrva");
15         linkedlist.addLast("stavkaPoslednja");
16         System.out.println("u LinkedList je: " +linkedlist);
17         Object firstvar = linkedlist.get(0);
18         System.out.println("prvi element je: " +firstvar);
19         linkedlist.set(0, "stavkaNovaPrva");
20         Object firstvar2 = linkedlist.get(0);
21         System.out.println("novi prvi element je: " +firstvar2);
22         linkedlist.removeFirst();
23         linkedlist.removeLast();
24         System.out.println("LinkedList posle brisanja prvog i zadnjeg elementa: " +linkedlist);
25         linkedlist.add(0, "stavkaPrva2");
26         linkedlist.remove(2);
27         System.out.println("dobija se na kraju: " +linkedlist);
28         }
```

.get(), .set(), .add(), .removeXXX

Problems Javadoc Declaration Console

<terminated> obuka0306 [Java Application] C:\Program Files\Java\jre1.8.0_131\bin\javaw.exe (Jan 4, 2018, 10:38:28 AM)

```
obuka 03 primer 06, Linked List

u Linked List je: [stavka03, stavka02, stavka01]
u LinkedList je: [stavkaPrva, stavka03, stavka02, stavka01, stavkaPoslednja]
prvi element je: stavkaPrva
novi prvi element je: stavkaNovaPrva
LinkedList posle brisanja prvog i zadnjeg elementa: [stavka03, stavka02, stavka01]
dobija se na kraju: [stavkaPrva2, stavka03, stavka01]
```

Writable Smart Insert 27 : 47

# LinkedList

Java - obuka01/src/obuka03/obuka0307.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

obuka0307.java

```
1  package obuka03;
2  import java.util.*;
3  public class obuka0307 {
4      public static void main (String[] args) {
5          System.out.println("obuka 03 primer 07, Linked List\n");
6          LinkedList<String> linkedlist = new LinkedList<String>();
7          /*add(String Element) is used for adding
8          * the elements to the linked list*/
9          linkedlist.add("stavka03");
10         linkedlist.add("stavka02");
11         linkedlist.add("stavka01");
12         System.out.println("u Linked List je: " +linkedlist);
13         /*Add First and Last Element*/
14         linkedlist.addFirst("stavkaPrva");
15         linkedlist.addLast("stavkaPoslednja");
16         System.out.println("u LinkedList je: " +linkedlist);
17         Object firstvar = linkedlist.get(0);
18         Iterator it = linkedlist.iterator();
19         System.out.println("LinkedList elementi:");
20         while(it.hasNext()){
21            System.out.println(it.next());
22         }
23     }
24 }
```

Problems | Javadoc | Declaration | Console

<terminated> obuka0307 [Java Application] C:\Program Files\Java\jre1.8.0_131\bin

```
obuka 03 primer 07, Linked List

u Linked List je: [stavka03, stavka02, stavka01]
u LinkedList je: [stavkaPrva, stavka03, stavka02, stavka01, stavkaPoslednja]
LinkedList elementi:
stavkaPrva
stavka03
stavka02
stavka01
stavkaPoslednja
```

Writable | Smart Insert | 19 : 48

LinkedList Iterator,
hasNext() & next() metod

# Static i final

- Definisanje specifičnih promenljivih i metoda u okviru klase
- public double PI = 3.14159; // konstanta preko objekta
- public **static** double PI = 3.14159; // konstanta preko klase
  da ne mora da se pravi u svakom objektu klase,
  može da se proemni vrednost konstante pri kodovanju

# Static i final

Java - obuka01/src/obuka03/obuka0308.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

obuka0308.java

```
1  package obuka03;
2  public class obuka0308{
3      public double PI = 3.14159;
4      public double square(double x){
5          return x*x;
6      }
7      public static void main (String[] args) {
8          System.out.println("obuka 03 primer 08, Static, final\n");
9          obuka0308 objekat1 = new obuka0308();
10         System.out.println("PI vrednost objekta1 je: "+objekat1.PI);
11         double broj = 5.;
12         System.out.println("square metoda objekta1 za broj " +
13           broj+" je "+objekat1.square(broj));
14         obuka0308 objekat2 = new obuka0308();
15         System.out.println("2 PI vrednost objekta2 je: "+2*objekat2.PI);
16         System.out.println("square metoda objekta2 za broj " +
17                   (broj+1)+" je "+objekat1.square(broj+1));
18     }
19 }
```

uka01
uka02
uka03
obuka0301.java
obuka0302.java
obuka0303.java
obuka0304.java
obuka0305.java
obuka0306.java
obuka0307.java
obuka0308.java
prekidacSvetlaKons
prekidacSvetlaPriv.j
prekidacZaSvetlo.ja
prekidacZaSvetloM
svetlaObjekt.java
ystem Library [JavaS

U klasi obuka0308 sa instancama objekata objekat1 i objekat2 dobijamo vrednosti konstante PI, PI, i rezultate metode square

<terminate

```
obuka 03 primer 08, Static, final

PI vrednost objekta1 je: 3.14159
square metoda objekta1 za broj 5.0 je 25.0
2 PI vrednost objekta2 je: 6.28318
square metoda objekta2 za broj 6.0 je 36.0
```

Nepotrebno se čuva ista konstanta PI za svaki objekat klase, dovoljno bi bilo samo jednom

Writable | Smart Insert | 18 : 1

# Static i final

Konstanta je podatak koji pripada klasi a ne objektima koji nasleđuju klasu

Java - obuka01/src/obuka03/obuka0309.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

obuka0309.java

```
1  package obuka03;
2  public class obuka0309{
3      public static double PI = 3.14159;
4      public static double square(double x){
5          return x*x;
6      }
7      public static void main (String[] args) {
8          System.out.println("obuka 03 primer 09, Static, final\n");
9          System.out.println("PI vrednost klase je: "+obuka0309.PI);
10         double broj = 5.;
11         System.out.println("square metoda klase za broj " +
12           broj+" je "+obuka0309.square(broj));
13         System.out.println("2 PI vrednost klase je: "+2*obuka0309.PI);
14         System.out.println("square metoda klase za broj " +
15                 (broj+1)+" je "+obuka0309.square(broj+1));
16     }
17 }
```

U klasi obuka0309 nije potrebno praviti objekte da bi se dobila vrednost konstante PI i rezultate metode square

```
obuka 03 primer 09, Static, final

PI vrednost klase je: 3.14159
square metoda klase za broj 5.0 je 25.0
2 PI vrednost klase je: 6.28318
square metoda klase za broj 6.0 je 36.0
```

**static** je upotrebljen da bi se imala samo jedna vrednost u klasi za konstantu i jedna metoda

# Static i final

Java - obuka01/src/obuka03/obuka0310.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

obuka0310.java

```
1  package obuka03;
2  public class obuka0310{
3      public static double PI = 3.14159;
4      public static double square(double x){
5          return x*x;
6      }
7      public static void main (String[] args) {
8          System.out.println("obuka 03 primer 10, Static, final\n");
9          System.out.println("PI vrednost klase je: "+obuka0310.PI);
10         double broj = 5.;
11         System.out.println("square metoda klase za broj " +
12           broj+" je "+obuka0310.square(broj));
13         obuka0310.PI = 11.1;
14         System.out.println("PI vrednost klase je: "+ obuka0310.PI);
15     }
16 }
```

vrednost konstante PI može da se promeni

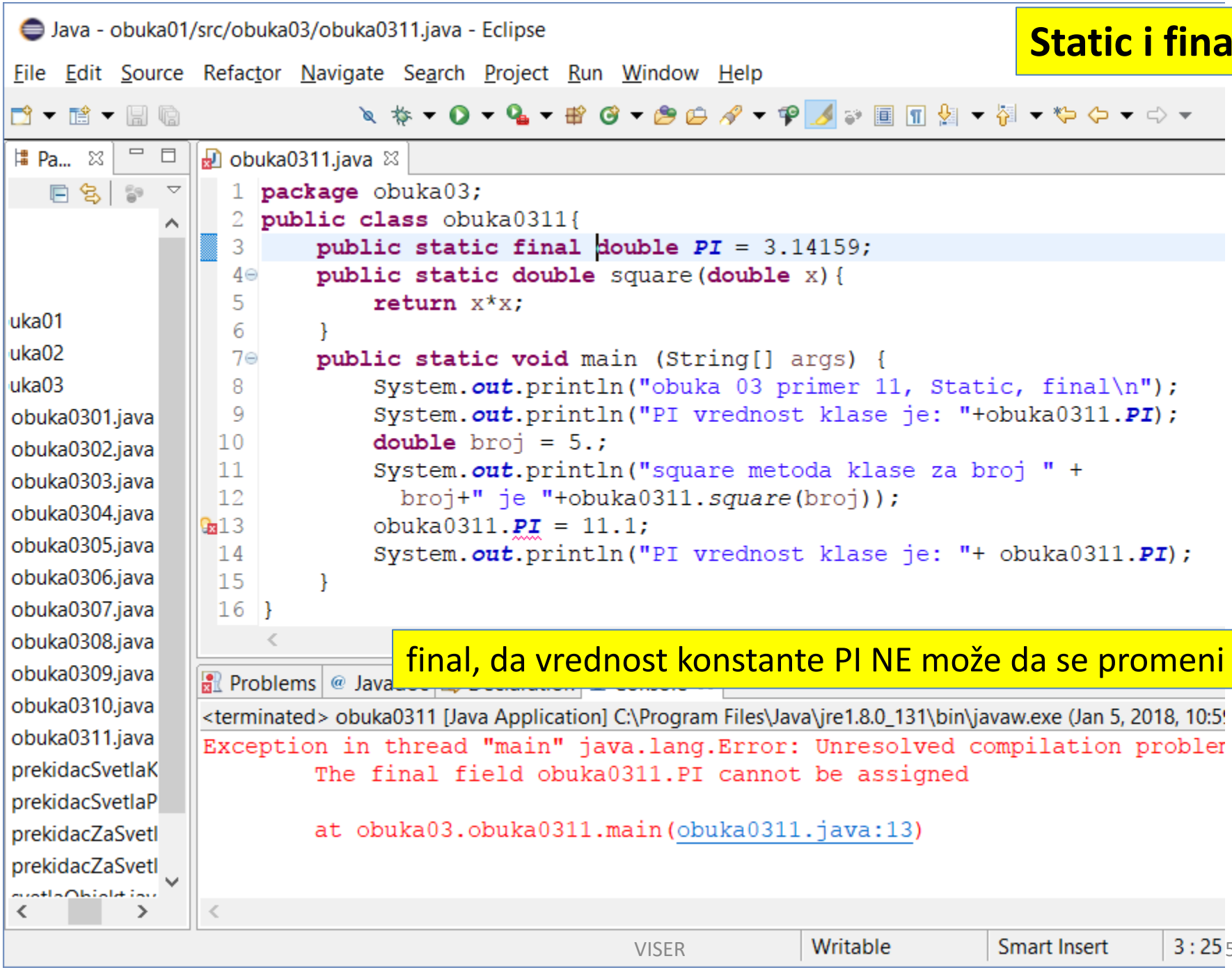
Problems Javadoc Declaration Console

<terminated> obuka0310 [Java Application] C:\Program Files\Java\jre1.8.0_131\bin\javaw.exe (Jan 5, 2018, 10:54:2

```
obuka 03 primer 10, Static, final

PI vrednost klase je: 3.14159
square metoda klase za broj 5.0 je 25.0
PI vrednost klase je: 11.1
```

VISER Writable Smart Insert 8 : 47

# Static i final



# Static i final

Java - obuka01/src/obuka03/obuka0312.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

obuka0312.java

```
1  package obuka03;
2  import java.awt.*;
3  public class obuka0312{
4      public static final double PI = 3.14159;
5      public static  Point ORIGIN = new Point(0, 0);
6      public static void main (String[] args) {
7          System.out.println("obuka 03 primer 12, Static, final\n");
8          System.out.println(obuka0312.ORIGIN);
9          obuka0312.ORIGIN = new Point(3, 4);
10         System.out.println(obuka0312.ORIGIN);
11         System.out.println("PI vrednost klase je: "+ obuka0312.PI);
12     }
13 }
```

ORIGIN može da se promeni

Problems

<terminated> obuka0312 [Java Application] C:\Program Files\Java\jre1.8.0_131\bin\javaw.exe (Jan 6, 2018, 11

```
obuka 03 primer 12, Static, final

java.awt.Point[x=0,y=0]
java.awt.Point[x=3,y=4]
PI vrednost klase je: 3.14159
```

VISER Writable Smart Insert 58 : 1

# Static i final

Java - obuka01/src/obuka03/obuka0313.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

obuka0313.java

```
1  package obuka03;
2  import java.awt.*;
3  public class obuka0313{
4      public static final double PI = 3.14159;
5      public static final Point ORIGIN = new Point(0, 0);
6      public static void main (String[] args) {
7          System.out.println("obuka 03 primer 13, Static, final\n");
8          System.out.println(obuka0313.ORIGIN);
9          //obuka0313.ORIGIN = new Point(3, 4);
10         System.out.println(obuka0313.ORIGIN);
11         System.out.println("PI vrednost klase je: "+ obuka0313.PI);
12     }
13 }
```

ORIGIN ne može da se promeni, javlja grešku za

**obuka0313.ORIGIN = new Point(3, 4);**

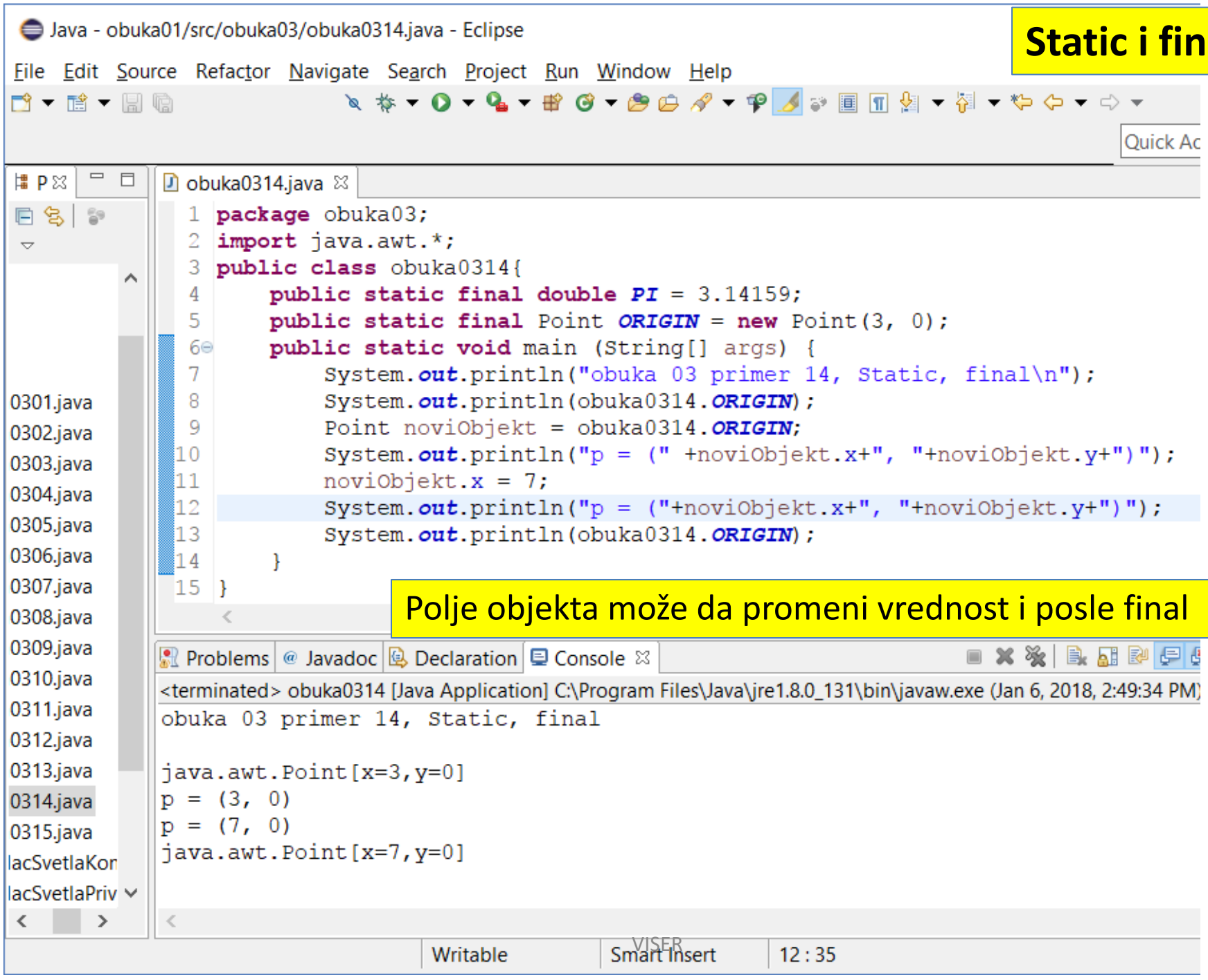
Problems

<terminated> obuka0313 [Java Application] C:\Program Files\Java\jre1.8.0_131\bin\javaw.exe (Jan 6, 2018, 11

```
obuka 03 primer 13, Static, final

java.awt.Point[x=0,y=0]
java.awt.Point[x=0,y=0]
PI vrednost klase je: 3.14159
```

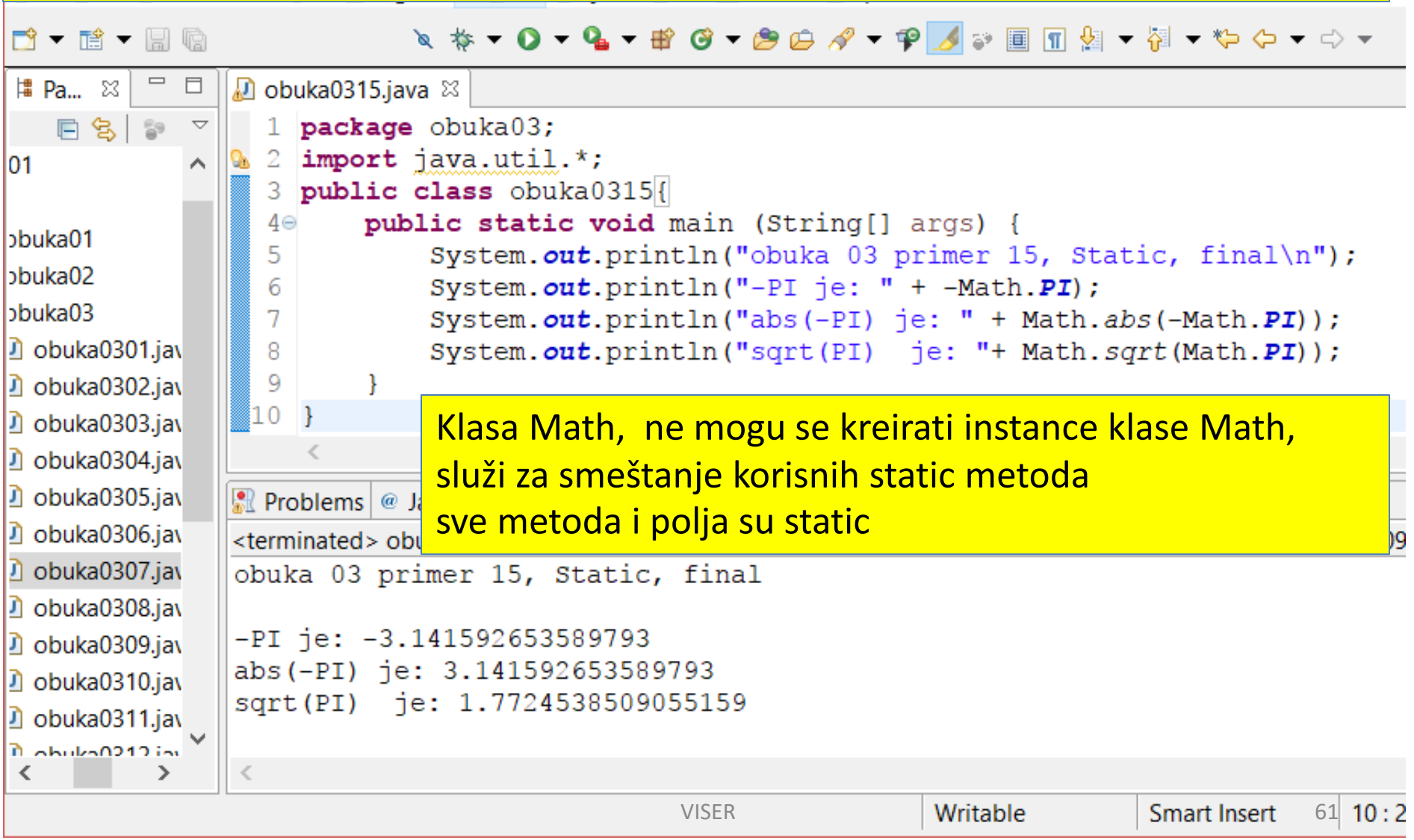
VISER Writable Smart Insert

# Static i final




Polje objekta može da promeni vrednost i posle final

**Static** treba koristiti za metode koje koriste samo argumente i ne koriste instance. Za metode kojima trebaju samo static podaci, kada se startuje program kada ne postoje objekti koje bi main metod mogao da koristi.




Klasa Math, ne mogu se kreirati instance klase Math,
služi za smeštanje korisnih static metoda
sve metoda i polja su static

# Pristup i organizacija

- Na računaru postoji hiljade fajlova
- Na Internetu postoji više miliona sajtova
- Projekat može imati hiljade klasa
- Kako pronaći određenu klasu, fajl ili sajt?
- Kako organizujemo rad u slučaju istih ili sličnih imena?
- Kako se može ograničiti pristup privatnim podacima?

# Oblast važenja

- Oblast važenja promenljive, polja, metode ili klase je deo programa iz koga se može pristupiti elementima
- Promenljivama se može pristupiti iz bloka u okviru koga su deklarisane
- Block se označava pomoću { i }
- Lokalne promenljive: Promenljive koje su definisane samo u trenutnom bloku
- Pristup polju, metama i klasama je definisana pomoću njihovih specifikatora

# Anonimne promenljive

- Anonimne promenljive se deifinišu i inicijalizuju, ali se nikada ne deklarišu
- U C, C++ anonimne promenljive se tretiraju kao greška, u Javi ne
- Ove promenljive nemaju ime i ne mogu se pozivati nemaju oblast važenja
- Primeri:

```
System.out.println("Hello");
(new String("Hello"))
(new int[] {1,2,3,4})
```

# Primer oblasti važenja

```
1. void foo(int x) {
2.     int y = 3;
3.     if (y > 0) {
4.         int z= 4;
5.         if (z> 0) {
6.             int w= 0;
7.           {
8.               int v = 1;
9.           } // end block
10.       } // end if (z>0)
11.    } // end if (y>0)
12.} // end
```

promenljive

- može se pristupiti promenljivama v, w, x, y, z?

# Primer oblasti važenja

```
class TestScope {
        int x = 0;
        void foo(int z) {
                int y = 20;
                x = 10;
                int z = 30; // Error
        }// end foo()
        void print() {
                System.out.println(x);
                foo(x);
                System.out.println(x);
                System.out.println(y); // error
        } // end print()
} // end TestScope
```

metoda

- x je definisano za celu klasu, y je definisano u okviru metoda foo(int).
- z je već definisan u foo(int) jer je argument metoda

# Primer oblasti važenja

```
class Scope{
        int x = 3;
        void foo(int y) {
                System.out.println(x);
                int x = 2;
                System.out.println(x);
                System.out.println(this.x);
                System.out.println(y);
        }
        public static void main(String[] args) {
                int x = 1;
                (new Scope()).foo(x); // AnonymousObject
        }
}
```

polja

# Primer oblasti važenja

petlje

```
int sigma(int n) {
      for (int i = 0; i<n; i++) {
      int sum += i;
      }
      return sum;
}
```

- Metod sigma bi trebalo da generiše rezultat koji je jednak sumi po i od i=0 do i=n, ali ima grešku

```
class TestScope {
        int x = 0;
        void foo() {
                int y = 20;
                x = 10;
        } // end foo
        void print() {
                int y = 0;
                System.out.println(x);
                foo();
                System.out.println(x);
                System.out.println(y);
        } // end print()
} // end Class TestScope
```

primer

- Rezultat: 0, 10, 0

```
class TestScope {
	int x = 0;
	int y = 0;
	void foo() {
		int y;
		y = 20;
		x = 10;
	}
	void print() {
		System.out.println(x);
		foo();
		System.out.println(x);
		System.out.println(y);
	}
}
```

primer

- Rezultat: 0, 10, 0

```
class TestScope {
      int x = 0;
      int y = 0;
      void foo() {
            y = 20;
            x = 10;
      }
      void print() {
            System.out.println(x);
            foo();
            System.out.println(x);
            System.out.println(y);
      }
}
```

primer

- Rezultat: 0, 10, 20

# Imenovanje promenljivih u velikim projektima

- Veliki projekti mogu imati na hiljade klasa
- Možete sarađivati i deliti klase sa ljudima koji se nalaze širom sveta
- Kako da budete sigurni da se ime vaše klase neće poklopiti sa nekim imenom neke druge?
- Možete koristiti dugačka jedinstvena imena, npr. “UtilityClassForPreparingTaxReturnsByAnthonyGfrom WestlandsNairobiKenya”
- Ovakav način je težak za pamćenje i korišćenje

# Hijerarhijska organizacija

- Imena su organizovana u hijerarhijski raspored
- “Bill Gates”: Porodica: Gates, Član: Bill
- rti.etf.bg.ac.rs
- 254-020-5555555: Kenya (254), Nairobi (020), Broj 5555555
- java.lang.String: String je klasa, u okviru lang paketa, koji se nalazi u okviru osnovnog java paketa

# Definicija paketa

- Potrebno je organizovati klase u skupove ili delove programa koji se nazivaju paketi
- Na ovaj način se smanjuje problem konflikta imena i prepoznavanja funkcionalnosti, na primer java.util.
- Može se ograničiti pristup na nivou paketa.
- Pripadnost klase paketu se definiše sa:

```
package imePaketa;
    class imeKlase{
    /* Telo klase */
    }
```

# Paketi

- Deklaracija paketa mora biti prva linija u fajlu koja nije komentar
- Prva klasa definisana u okviru fajla mora da ima isto ime kao i ime fajla
- Samo prva klasa može biti tipa public
- javac i java će tražiti navedene poddirektorijume ili JAR (Java Archive) fajlove za navedeni kod:
  - putanja (na Windows operativnom sistemu)
  - \mypkg\util za paket mypkg.util
  - putanja (na Unix operativnom sistemu) /mypkg/util/arrays za paket mypkg.util.arrays

# Korišćenje paketa

- Mogu se korisno definisati imena:
  - `java.util.Date d = new java.util.Date();`
- □ specifirane klase koje se koristite:
  - `import java.util.Date;`
  - `import java.util.ArrayList;`
- Sve klase iz paketa:
  - `import java.util.*;`
- Paketi mogu imati pod-pakete:
  - `import java.util.logging.*;`
- Default korišćeni paket:
  - `import java.lang.*;`

# Specifikatori pristupa

- Polja, metode, konstruktori i klase koje su definisane kao **public** omogućavaju pristup iz bilo koje klase bilo kog paketa
- Polja, metode, konstruktori i klase koje su definisane kao **protected** omogućavaju pristup samo klasama koje ih nasleđuju
- Polja, metode, konstruktori i klase koje su definisane sa **paketnim pristupom** omogućavaju pristup iz bilo koje klase istog paketa. Ovakav pristup se podrazumeva
- Polja, metode i konstruktori koji su definisani kao **private** omogućavaju pristup samo kodi iz klase u kojoj se pojavljuju

# Nivoi kontrole pristupa

## Access Modifiers

| Access Level | From classes in the other **packages** | From classes in the same **package** | From child classes | From the same **class** |
|---|---|---|---|---|
| **public** | yes | yes | yes | yes |
| **protected** | no | yes | yes | yes |
| **default** | no | yes | no | yes |
| **private** | no | no | no | yes |

| Accessible.. | private | package (default) | protected | public |
|---|---|---|---|---|
| From same class | Yes | Yes | Yes | Yes |
| From same package | No | Yes | Yes | Yes |
| From child classes | No | No | Yes | Yes |
| From anywhere | No | No | No | Yes |

Primer
**package examples**

```
package examples;

      class Person {

            String name;

            //Telo klase ce se dopuniti:

            // Dodavanjem polja za rodjendan

            // Dodavanjem metoda za definisanje rodjendana

            // Dodavanjem metoda za citanje rodjendana

      }
```

Primer
**package examples**

```
package examples;
      class Person {
            String name;
            java.util.Date birthDay;
            void setBirthday(java.util.Date d) {
                  this.birthDay = d;
            }
            java.util.Date getBirthday() {
                  return this.birthDay;
            }
      }
```

Primer
**package examples**

```
package examples;
      import java.util.Date;
      class Person {
            String name;
            Date birthDay;
            void setBirthday(Date d) {
                  this.birthDay = d;
            }
            Date getBirthday() {
                  return this.birthDay;
      }
}
```

```
package examples;
import java.util.Date;
import java.util.ArrayList;
      class Person {
            String name;
            ArrayList friends;
            Date birthDay;
            void setBirthday(Date d) {
                  this.birthDay = d;
            }
            Date getBirthday() {
                  return this.birthDay;
            }
      }
```

Primer
**package examples**

```
package examples;
import java.util.*;
      class Person {
            String name;
            ArrayList friends;
            Date birthDay;
            void setBirthday(Date d) {
                  this.birthDay = d;
            }
            Date getBirthday() {
                  return this.birthDay;
            }
      }
```

Primer
**package examples**

# Nasleđivanje

- U stvarnom svetu:
  nasleđujemo gene od roditelja
  nasleđujemo gene i od drugih predaka
  različite oči, težina, visina . . .
  veliki broj osobina nasleđujemo od naših roditelja
- U softveru:
  Nasleđivanje kod objekata je bolje definisano
  Objekti koji nasleđuju neke druge objekte, od njih nasleđuju i stanja (polja) i ponašanje (metode)

Primer
**Klasa Masai**

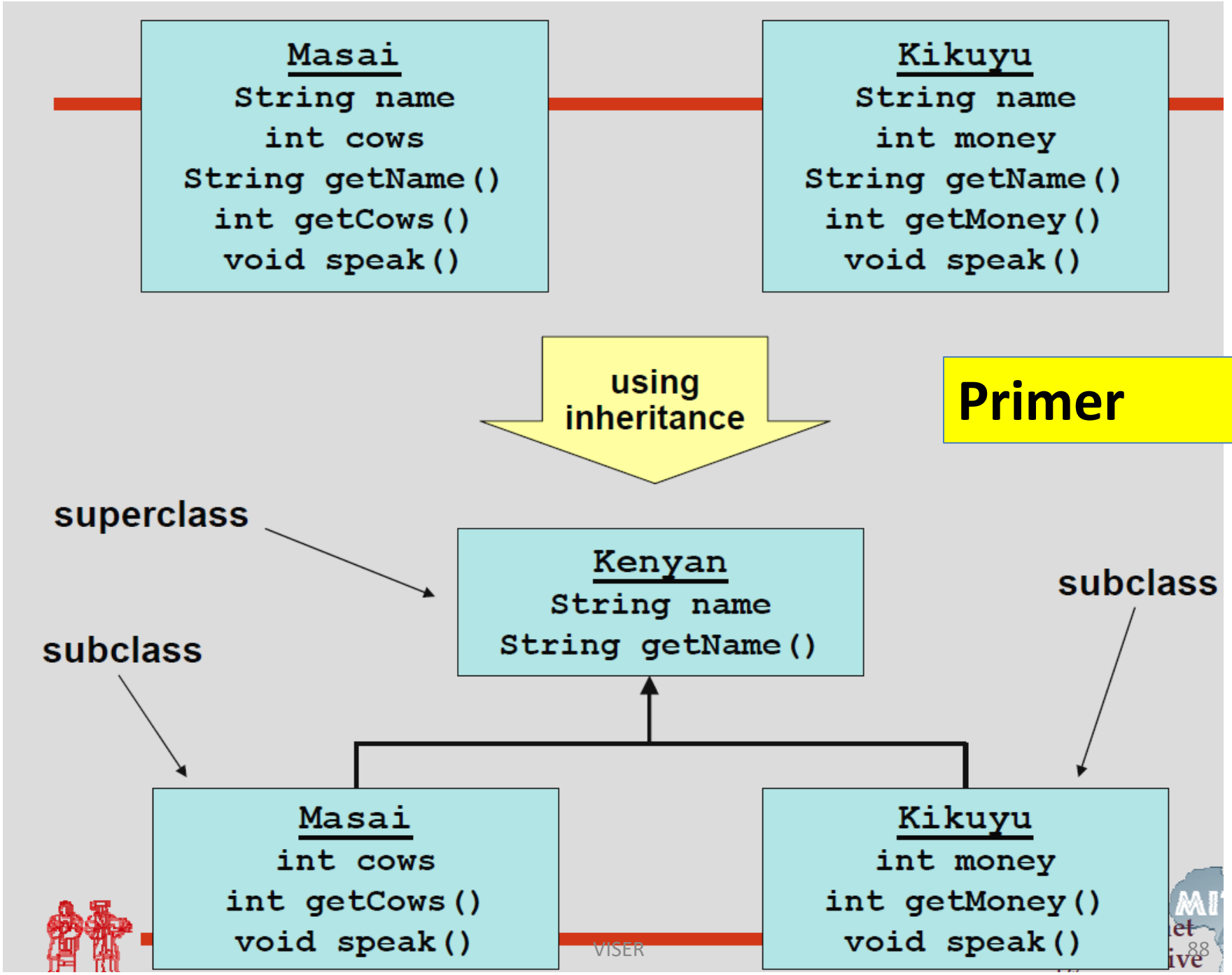
```
public class Masai{
	private String name;
	private int cows;
	public Masai(String n, int c) {
		name = n;
		cows = c;
	}
	public String getName() { return name; }
	public int getCows() { return cows; }
	public void speak() {
		System.out.println("Masai");
	}
}
```

Primer
**Klasa Kikuyu**

```
public class Kikuyu {
	private String name;
	private int money;
	public Kikuyu(String n, int m) {
		name = n;
		money = m;
	}
	public String getName() { return name; }
	public int getMoney() { return money; }
	public void speak() {
		System.out.println("Kikuyu");
	}
}
```

# Dupliranje koda

- Prethodne klase imaju isto polje name i isti metod getName
- Klase često imaju veliki broj zajedničkh polja i metoda
- Rezultat: dupliranje koda
- Koristeći nasleđivanje moguće je napisati novu klasu koja nasleđuje neku već postojeću
- Postojeća klasa čije osobine nasleđuju naziva se klasa “roditelj" ili superklasa
- Nova klasa koja nasleđuje superklasu naziva se klasa “potomak" ili subklasa
- Rezultat: ušteda koda


Masai
String name
int cows
String getName()
int getCows()
void speak()
Kikuyu
String name
int money
String getName()
int getMoney()
void speak()
using inheritance
Primer
superclass
Kenyan
String name
String getName()
subclass
subclass
Masai
int cows
int getCows()
void speak()
Kikuyu
int money
int getMoney()
void speak()

```
public class Kenyan {
      private String name;
      public Kenyan(Stringn) {
            name = n;
      }
      public String getName() {
            return name;
      }
}
```

Primer
**Kenyan superklasa**

```
public class Masai extends Kenyan{
      private int cows;
      public Masai(String n, int c) {
         super(n); // poziv Kenyan konstruktor
         cows = c;
      }
      public int getCows() {
            return cows;
      }
      public void speak() {
            System.out.println("Masai");
      }
}
```

Primer
**Masai podklasa**

```
public class Kikuyu extends Kenyan{
    private int money;
    public Kikuyu(String n, int m) {
        super(n); // poziv Kenyan konstruktor
        money = m;
    }
    public int getMoney() {
        return money;
    }
    public void speak() {
        System.out.println("Kikuyu");
    }
}
```

Primer
**Kikuyu podklasa**

Primer

```
Masai d = new Masai("Johnson" 23);
Kikuyu c = new Kikuyu("Sheila", 2200);
  System.out.println(d.getName() + " has " +
d.getCows() + " cows");
  System.out.println(c.getName() + " has " +
c.getMoney() + " shillings");
```

Johnson has 23 cows
Sheila has 2200 shillings

# Pravila nasleđivanja

- Koristi se službena reč **extends**
  da bi se naznačilo koja klasa nasleđuje koju klasu
- Podklasa nasleđuje
  sva polja i sve metode superklase
- Koristi se službena reč **super** u okviru konstruktora podklase da bi se pozvao kontruktor superklase

# Konstruktor podklase

- Prvo, konstruktor podklase mora da pozove konstruktor superklase
- Delovi koji pripadaju super klasi konstruišu se pre delova koji su deo same podklase
- Ako se ne pozove konstruktor superklase pomoću naredbe super i superklasa ima konstruktor bez argumenata, tada će se taj konstuktor pozvati implicitno

```
public class Food {
  private boolean raw;
  public Food() {
     raw = true;
   }
}
```

```
public class Beef extends Food {
  private double weight;
  public Beef(double w ) {
      weight = w
   }
}
```

```
public class Beef extends Food {
  private double weight;
  public Beef(double w) {
     super();
     weight = w
  }
}
```

Isto što i prva dva

```
public class A {
     public A() { System.out.println("I'm A");
     }
}
public class B extends A {
     public B() { System.out.println("I'm B");
     }
}
public class C extends B {
     public C() { System.out.println("I'm C");
     }
}
```

```
C x = new C();
?
```

Java - obuka01/src/obuka03/nasledjivanjeC.ja

File Edit Source Refactor Navigate Search

C:\Users\MIROSLAV\workspace2\obuka01\src\obuka03\nasledjivanjeA.java - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

new 1 | nasledjivanjeA.java

```
1  package obuka03;
2  public class nasledjivanjeA{
3      public nasledjivanjeA(){
4          System.out.println("Ja sam A");
5      }
6      public static void main (String[] args) {
7          System.out.println("obuka 03, nasledjivanje, I am A \n");
8      }
9  }
```

Java source file | length : 227 lines : 9 | Ln : 1 Col : 1 Sel : 0 | 0 | Windows (CR LF) | UTF-8 | INS

P

nasledjivanjeC.java

```
1  package obuka03;
2  public class nasledjivanjeC extends nasledjivanjeB{
3      public nasledjivanjeC(){
4          System.out.println("Ja sam C");
5      }
6      public static void main (String[] args) {
7          System.out.println("obuka 03, nasledjivanje, I am C \n");
8          nasledjivanjeC x = new nasledjivanjeC();
9          System.out.println(" --- C ---");
10     }
11 }
```

anjeA.java
anjeB.java
anjeC.java
01.java
02.java
03.java
04.java
05.java
06.java
07.java
08.java
09.java
0.java
1.java
2.java
3.java

Problems | Javadoc | Declaration | Console

<terminated> nasledjivanjeC [Java Application] C:\Program Files\Java\jre1.8.0_131\bin\javaw.exe (Jan 7, 20

```
obuka 03, nasledjivanje, I am C

Ja sam A
Ja sam B
Ja sam C
 --- C ---
```

Writal

```
package obuka03;
public class nasledjivanjeB extends nasledjivanjeA{
    public nasledjivanjeB(){
        System.out.println("Ja sam B");
    }
    public static void main (String[] args) {
        System.out.println("obuka 03, nasledjivanje, I am B \n");
        nasledjivanjeB x = new nasledjivanjeB();
        System.out.println(" --- B ---");
    }
}
```

# Preklapanje metoda

- Podklasa može preklopiti metod svoje superklase

```
class Therm {
  public double celsius;

  public Therm(double c) {
    celsius = c;
  }

  public double getTemp() {
    return celcius;
  }
}
```

```
class ThermUS extends Therm {

  public ThermUS(double c) {
    super(c);
  }

  // degrees in Fahrenheit
  public double getTemp() {
    return celsius * 1.8 + 32;
  }
}
```

```
ThermUS thermometer = new ThermUS(100);
System.out.println(thermometer.getTemp());
212
```

# Poziv metoda superklase

- Kada se preklopi određeni metod,
  moguće je pozvati kopiju tog metoda superklase
  koristeći naredbu super.metod()

```
class Therm {
  private double celsius;

  public Therm(double c) {
    celcius = c;
  }

  public double getTemp() {
    return celcius;
  }
}
```

```
class ThermUS extends Therm {

  public ThermUS(double c) {
    super(c);
  }

  public double getTemp() {
    return super.getTemp()
                * 1.8 + 32;
  }
}
```

# Geške pri kompajliranju

```
public static void main(String[] args) {
Kenyan a1 = new Kenyan();
a1.getName();
a1.getCows(); // Kenyan does not have getCows
a1.getMoney(); // Kenyan does not have getMoney
a1.speak(); // Kenyan does not have speak
Kenyan a2 = new Masai();
a2.getName();
a2.getCows(); // Kenyan does not have getCows
a2.getMoney(); // Kenyan does not have getMoney
a2.speak(); // Kenyan does not have speak
Masaid = new Masai();
d.getName();
d.getCows();
d.getMoney(); // Masaidoes not have getMoney
d.speak();
}
```

# Konverzija tipova

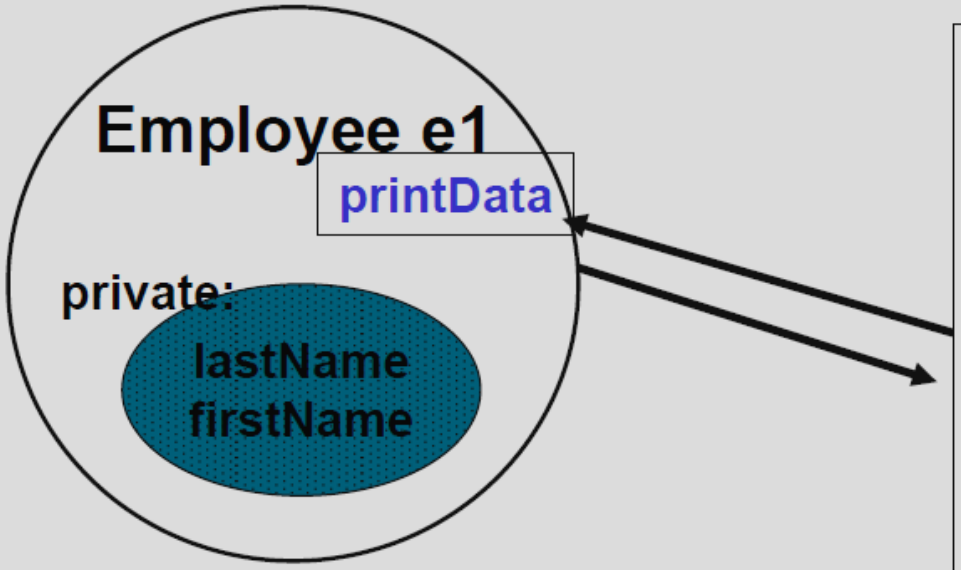
```
public static void main(String[] args) {
   Kenyan a1 = new Kenyan();
   ((Masai)a1).getCows(); //a1 is not a Masai
   ((Kikuyu)a1).getMoney(); //a1 is not a Kikuyu
   ((Masai)a1).speak(); //a1 is not a Masai

   Kenyan a2 = new Masai();
   ((Masai)a2).getCows();
   ((Kikuyu)a2).getMoney(); //a2 is not a Kikuyu
   ((Masai)a2).speak();

   Masaid = new Masai();
   ((Kikuyu)d).getMoney(); //d is not a Kikuyu

}
```

- Kompanija ima listu Zaposlenih. Potrebno je napraviti stranicu za plaćanje za svakog zaposlenog
- Na stranici se prikazuju opšti podaci (ime, odeljenje, iznos) za sve zaposlene

menadžeri
inženjeri

- Različiti tipovi zaposenih – menadžer, inženjer, softveraš
- Postoji stara verzija klase Employee ali je potrebno dodati nove podatke i metode za menadžere i inženjere
- Ako se uključi stara Employee klasa u sistem
  U okviru klase postoji metod printData() za svakog zaposlenog i ovaj metod samo prikazuje ime zaposlnog
  Potrebno je promeniti, da se prikazuju plaćanja

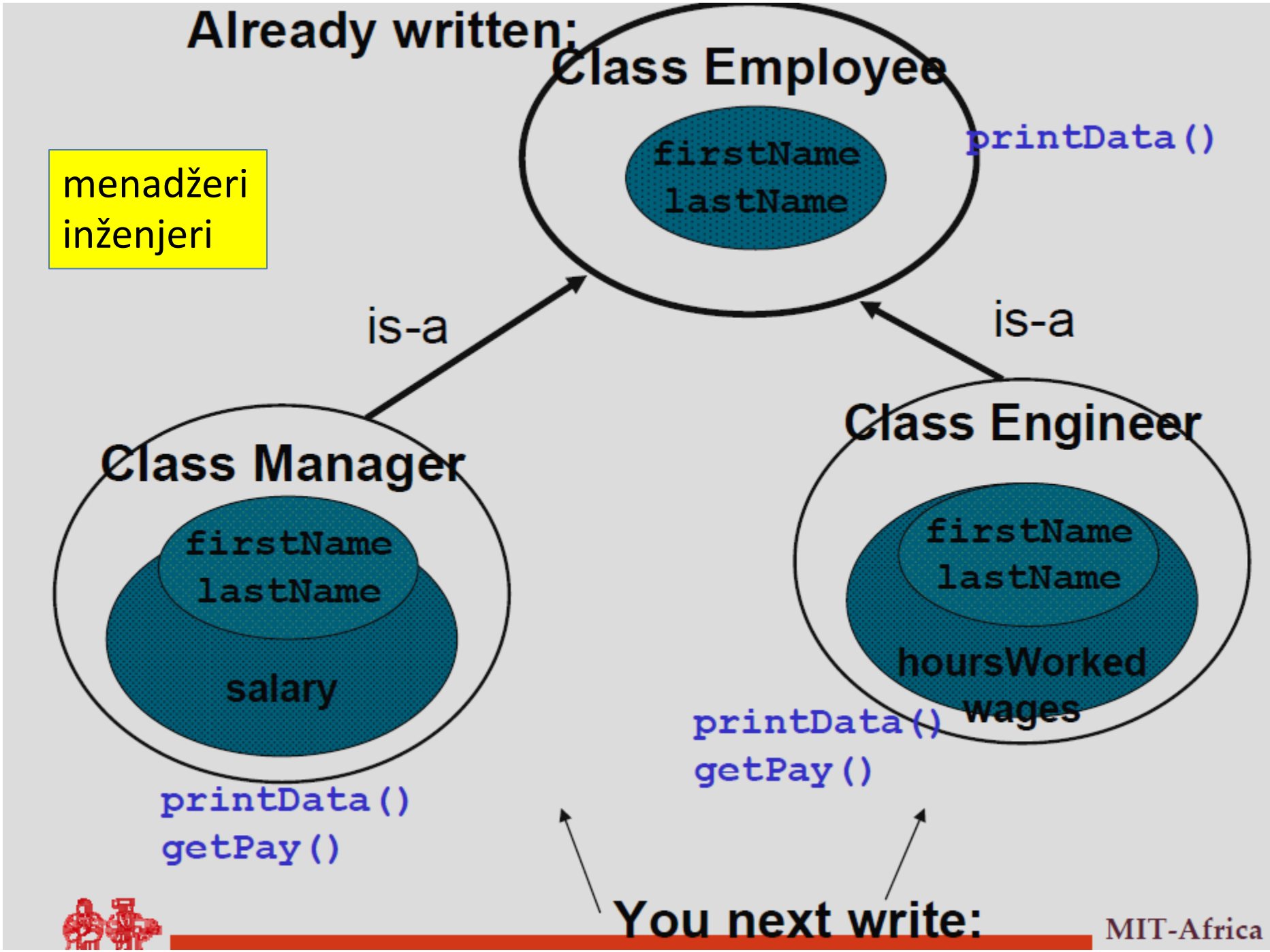
Encapsulation Message passing "Main event loop"




```
public … Main(…){
Employee e1…("Mary","Wang");
...
e1.printData();
// Prints Employee names.
...
}
```

# Jednostavna osnovna super klasa

```
class Employee {
      // Data
      private String firstName, lastName;
      // Constructor
      public Employee(String fName, String lName) {
         firstName= fName;
         lastName= lName;
      }
      // Method
      public void printData() {
         System.out.println(firstName+ " " + lastName);
      }
}
```


Already written:
Class Employee
firstName
lastName
printData()
menadžeri
inženjeri
is-a
is-a
Class Manager
firstName
lastName
salary
printData()
getPay()
Class Engineer
firstName
lastName
hoursWorked
wages
printData()
getPay()
You next write:

# Podklasa, izvedena klasa

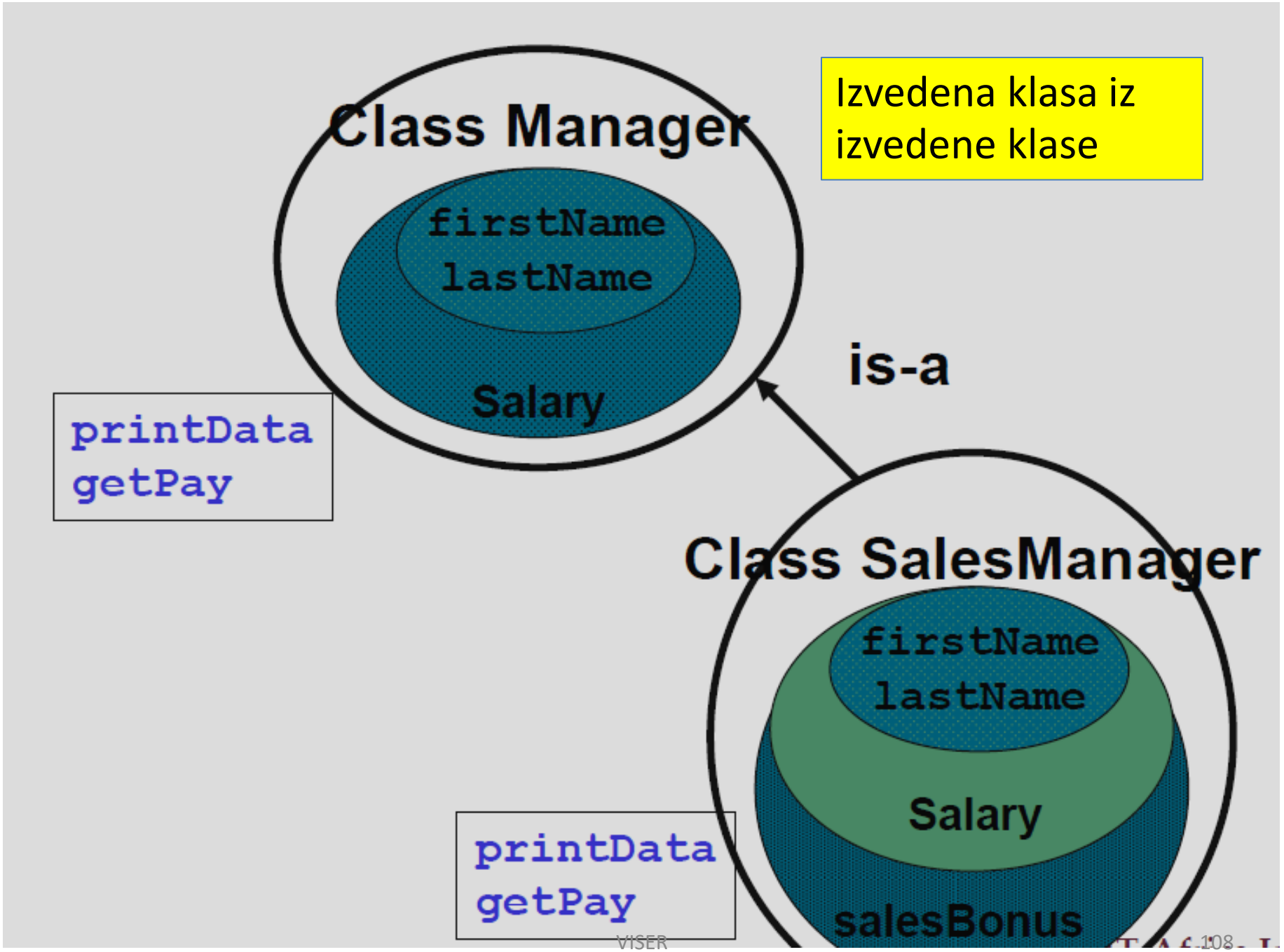
```
class Engineer extends Employee {
   private double wage;
   private double hoursWorked;
   public Engineer(StringfName, String lName,
         double rate, double hours) {
      super(fName, lName);
      wage = rate;
      hoursWorked= hours;
   }
   public double getPay() {
      return wage * hoursWorked;
   }
   public void printData() {
      super.printData();// PRINT NAME
      System.out.println("Weeklypay: $" + getPay();
}
```

inženjeri

# Podklasa, izvedena klasa

```
class Manager extends Employee {
private double salary;


public Manager(String fName, String lName, double sal){
super(fName, lName);
salary = sal; }


public double getPay() {
   return salary; }
public void printData() {
  super.printData();
  System.out.println("Monthlysalary: $" + salary);
  }
}
```

menadžeri


Izvedena klasa iz izvedene klase
Class Manager
firstName
lastName
Salary
printData
getPay
is-a
Class SalesManager
firstName
lastName
Salary
salesBonus
printData
getPay

# Izvedena klasa iz izvedene klase

```
class SalesManager extends Manager {
private double bonus; // Bonus Possible as commission
// A SalesManager gets constant salary of $1250
public SalesManager(StringfName, String lName, double b) {
   super(fName, lName, 1250.0);
   bonus = b; }

public double getPay() {
   return 1250.; }

public void printData() {
   super.printData();
   System.out.println("BonusPay: $" + bonus;
}
```

**SalesManager**

# Glavni metod

```
public class PayRoll{
  public static void main(String[] args) {
   // Could get Data from tables in a Database
   Engineer fred = new Engineer("Fred", "Smith", 12., 8.);
   Manager ann = new Manager("Ann", "Brown", 1500.);
   SalesManager mary = new SalesManager("Mary", "Kate", 2000.);
   // Polymorphism, or late binding


   Employee[] employees = new Employee[3];
   employees[0]= fred;
   employees[1]= ann;
   employees[2]= mary;
   for (inti=0; i < 3; i++)
     employees[i].printData();
   }
}
```

**Java zna tip objekta i bira odgovarajuću metodu u vreme izvršavanja**

# Rezultat koji daje glavna metoda

- Fred Smith
  **`Weekly pay: $96.0`**
- Ann Brown
  **`Monthly salary: $1500.0`**
- Mary Barrett
  **`Monthly salary: $1250.0`**
  **`Bonus: $2000.0`**
- Ne može da se piše :
  **`employees[i].getPay();`**
  zato što **`getPay()`** nije metoda superclase Employee
- Nasuprot, **`printData()`** je metoda Employee,
  pa Java može da nađe odgovarajuću vrednost

# Klasa Object

- Sve Javine klase implicitno nasleđuju klasu java.lang.Object
- Tako da svaka klasa koja se napiše automatski ima neke metode koji pripadaju klasi Object, kao šo su equals, hashCode, i toString

# Obrada grešaka pomoću izuzetaka

- Šta predstavlja izuzetak
- Uobičajena terminologija
- Zašto koristiti izuzetke
- Kako se dešava izuzetak
- Kako se obrađuje izuzetak
- O izuzecima koji se moraju obraditi i o onima koji se ne moraju
- Primeri Java izuzetaka
- Kako napisati svoj sopstveni izuzetak

# Izuzetak

- Izuzetak je događaj koji se dogodio tokom izvršavanja programa i može da prouzrokuje prekid normalnog toka izvršavanja instrukcija
- Mogući razlozi izuzetaka:
  - Pristup elementu izvan granica niza
  - Pokušaj upisa u read-only dokument
  - Pokušaj čitanja nakon kraja dokumenta
  - Slanje nelegalnih argumenata nekom metodu
  - Nelegalne aritmetičke operacije (deljenje sa 0, ...)
  - Otkaz hardverskih resursa

# Terminologija

- Kada se izuzetak (exception) dogodi koristi se izraz da je “bačen” izuzetak (throw)
- Kada se izvrše određene operacije sa izuzetkom kaže se da je izuzetak obrađen (handled)
- Deo koda pomoću koga se izvrše određene operacije sa izuzetkom naziva se obrada izuzetaka (exception handler)

# Kada se koriste izuzeci

- Pomoću kompajliranja ne mogu se odrediti sve greške
- Na ovaj način se odvaja kod za obradu izuzetaka od regularnog koda
  - Jasnoća i razumljivost koda (debagovanje, timski rad, ...)
  - Obrada izuzetaka se izvršava samo na jednom mestu
- Odvojeni su delovi koda za prepoznavanje greške, izveštavanje i obradu
- Mogu se grupisati i odvojiti različiti tipovi grešaka
- Mogu se obraditi greške koje prouzrokuju veoma specifične izuzetke

# Poruke kod izuzetaka

```
public class ArrayExceptionExample{
public static void main(String args[]) {
   String[] names = {“Bilha", “Robert"};
   System.out.println(names[2]);
   }
}
```

- Naredba println u kodu će dovesti do greške i slediće poruks pri obradi datog izuzetka:

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 2 at
ArrayExceptionExample.main(ArrayExceptionExample.java:4)
```

# Format poruke izuzetaka

```
[exception class]: [additional
description of exception] at
[class].[method]([file]:[linenumber])
```

## Primer

- Poruka kod izuzetka u primeru sa nizovima

```
java.lang.ArrayIndexOutOfBoundsException: 2 at
ArrayExceptionExample.main(ArrayExceptionExample.java:4)
```

- Koja je klasa izuzetka?

```
java.lang.ArrayIndexOutOfBoundsException
```

- Kom indeksu niza se pristupa, a on je izvan granica?

```
2
```

- Koji metod generiše izuzetak?

```
ArrayExceptionExample.main
```

- Koji fajl sadrži ovaj metod?

```
ArrayExceptionExample.java
```

- Koja linija fajla generiše izuzetak?

```
4
```

# Generisanje izuzetaka

- Svi metodi mogu koriste naredbu throw da bi prihvatili izuzetak

```
if (student.equals(null))
```

```
throw new NullPointerException();
```

- Naredba throw zahteva poseban argument: throwable objekat
- Ova vrsta objekata je instanca bilo koje podklase Throwable klase
- Ova klasa obuhvata sve tipove grešaka i izuzetaka
- U okviru opisa API mogu se pronaći i opisi svih throwable objekata

# Obrada izuzetaka

- Može se koristiti **try-catch** blok da bi se obradio izuzetak koji se desio

```
try{
// kod u okviru koga može da se desi izuzetak
}
catch([Tip izuzetka] e) {
// šta da se izvrši ako se desio izuzetak
}
```

# Obrada više izuzetaka istovremeno

- Više izuzetaka se može istovremeno obraditi pomoću više sukcesivnih catch blokova

```
try{
// kod u okviru koga može da se desi više izuzetaka
}
catch (IOException e) {
// obrada IOException
}
catch (ClassNotFoundException e2) {
// obrada ClassNotFoundException
}
```

# finally blok

- Može se koristiti opcioni finally blok na kraju try-catch bloka
- finally blok obezbeđuje mehanizam da reguliše sve što se desilo u okviru try bloka
- Može se iskoristiti za zatvranje fajla ili oslobađanja nekog drugog resursa

```
try{
 // kod u okviru koga može da se desi izuzetak
}
catch([Type of Exception] e) {
 // šta da se izvrši ako se desio izuzetak
}
finally{
 // naredbe u okviru bloka će se uvek izvršiti
 // bez obzira šta će se dogoditi u okviru bloka
}
```

# Izuzeci koji se ne mogu proveravati

- Izuzeci koji se ne mogu proveravati (unchecked exceptions) ili runtime izuzeci se pojavljuju u okviru Java runtime sistema
- Primeri
  - Aritmetičke operacije (deljenje sa 0)
  - pointer exceptions (pokušaj da se pristupi članu objekta pomoću null reference)
  - Izuzeci sa indeksima (pokušaj da se pristupi elementu niza sa indeksom koji je preveliki ili premali)
  - Metod koji ne sadrži catch ili specificira da može da generiše unchecked exceptions, takođe ih može generisati u nekim slučajevima

# Izuzeci koji se mogu proveriti

- Izuzeci koji se mogu proveriti (checked exceptions) ili non-runtime exceptions su izuzeci koji se mogu desiti izvan Java runtime sistema
- Na primer izuzeci koji se mogu desiti tokom ulazno/izlaznih operacija su non-runtime izuzeci
- Kompajler proverava da li su non-runtime izuzeci obrađeni (ili specificirani korišćenjem throws naredbe)

# Obrada izuzetaka koji se mogu proveriti

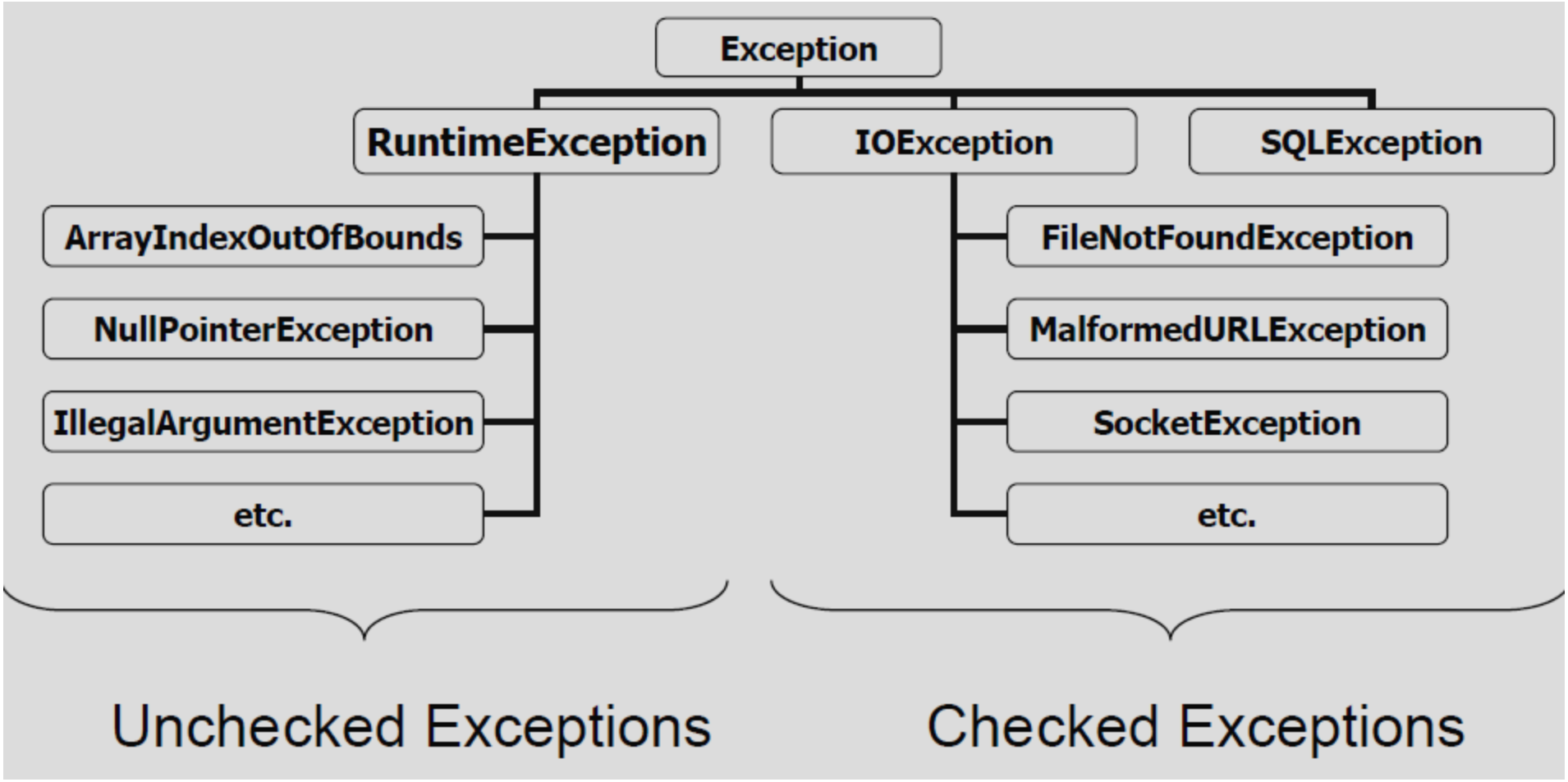
- Svaki metod mora obraditi izuzetak koji se može proveriti ili specificirati da takav izuzetak postoji (korišćenjem throws naredbe)

```
void readFile(Stringfilename) {
try {
FileReaderreader = new
FileReader("myfile.txt");
// read from file . . .
} catch (FileNotFoundExceptione) {
System.out.println("filewas not found");
}
}
```

- ili

```
void readFile(Stringfilename) throws
FileNotFoundException{
FileReaderreader = new FileReader("myfile.txt");
// read from file . . .
}
```

# Hijerarhija klasa izuzetaka



# Nasleđivanje i izuzeci

- Metod može specificirati manje izuzetaka, ali ne više od metoda koji nasleđuje

```
public class MyClass {
public void doSomething() throws IOException,
SQLException {
    // do something here
  }
}
public class MySubclass extends MyClass {
public void doSomething() throws IOException {
    // do something here
  }
}
```

# Pisanje sopstvenih izuzetaka

- Postoje najmanje 2 tipa konstruktora izuzetaka:

1. Default konstruktor: bez argumenata

```
NullPointerExceptione = new NullPointerException();
```

2. Konstruktor koji ima detaljnu poruku: postoji jedan String argument

```
IllegalArgumentExceptione e =
new IllegalArgumentException(“Broj mora biti pozitivan");
```

# Pisanje sopstvenih izuzetaka

- Sopstveni izuzeci moraju biti podklase klase Exception i moraju imati najmanje dva standardna konstruktora

```
public class MyCheckedException extends
IOException{
  public MyCheckedException() {}
  public MyCheckedException(String m){
  super(m);}
}
public class MyUncheckedException extends
RuntimeException{
  public MyUncheckedException() {}
  public MyUncheckedException(String m)
  {super(m);}
}
```

# Checked ili Unchecked?

- Ako korisnik očekuje da se izuzetak pojavi,
  trebalo bi da bude tipa checked
- Ako korisnik ne može da preduzme ništa da bi se oporavio od datog izuzetka,
  trebalo bi da bude unchecked tipa

# Zaključci

- Izuzeci narušavaju normalno izvršavanje instrukcija u okviru programa
- Izuzeci se obrađuju pomoću try-catch ili try-catch-finally bloka
- Metod specificira postojanje izuzetka pomoću throw naredbe
- Metod koji nema catch deo ili ne specificira da postoji unchecked izuzetak, može da ga generiše
- Svaki metod mora obraditi checked izuzetke ili specificirati da postoje
- Ako se piše spostveni izuzetak, on mora biti podklasa klase Exception i imati najmanje dva standardna konstruktora

# Ulaz-izlaz

1 –Ulaz/Izlaz

- Ulaz ili Izlaz, i Byte ili Character nizovi
- Najvažnije Stream klase i njihova upotreba
- Primeri čitanja iz i upisa u tekstualni fajl
- Primeri prihvatanja teksta sa tastature
- Upotreba bafera

2 –Uvod u postupak parsiranja

- Delimiteri
- StringTokenizer

# Osnove ulaz/izlaz

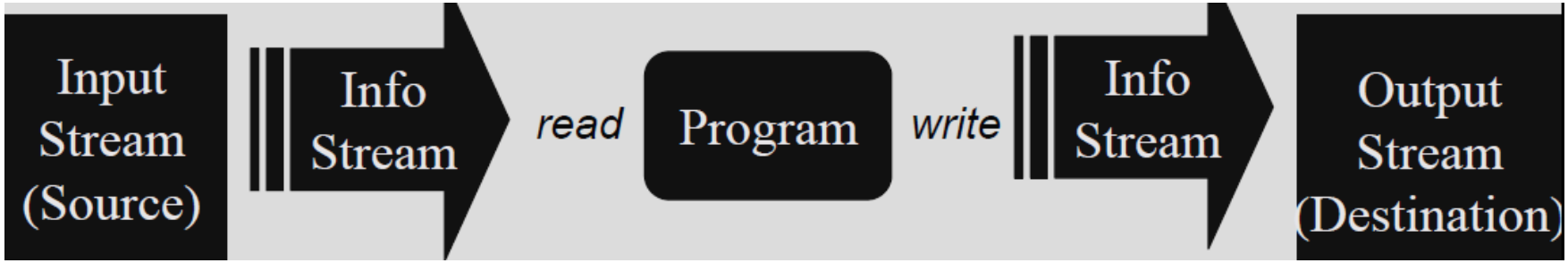
- I/O = Input/Output – ulaz/izlaz – komunikacija između računarskog programa i spoljašnjih izvora informacija
- Uključuje: - Čitanje ulaznih podataka iz izvora
- Upis rezultata u krajnje odredište
- Čitanje i upis su specifirani pomoću 4 abstraktne klase:
  - **`Reader`**
  - **`Writer`**
  - **`InputStream`**
  - **`OutputStream`**

# Java I/O Streams

- Java programi komuniciraju sa spoljašnjim svetom pomoću Streams
- Streams se koriste za čitanje i upis podataka
- I/O Streams su jednodirekcioni
- Ulazni (Input) stream se koristi za ulazne podatke u program
- Izlazni (output) stream se koristi za podatke koji predstavljaju rezultat izvršavanja programa
- Izvor i odredište podataka mogu biti:
  fajlovi, mrežne konekcije, drugi programi, ...

# Input i Output Stream

- Objekat pomoću koga možemo pročitati ulazne podatke je Input Stream
- Objekat pomoću koga možemo prikazati izlazne podatke je Output Stream



# Byte i Character

- Byte Streams se koristi da bi se čitali i upisivali podaci koji su u binarnom formatu (1 ili 0)
  - slike, zvuk, ...
- Character Streams se koristi da bi se čitali i upisivali podaci koji su u tekstualnom formatu (karakteri)
  - tekstualni fajlovi, web stranice, unos korisnika pomoću tastature, ...

# Najvažnije Stream klase

- FileInputStream
  - Čitanje podataka u binarnom formatu iz fajlova
- FileOutputStream
  - Upis podataka u binarnom formatu u fajlove
- FileReader
  - Čitanje tekstualnih podataka iz fajlova
- FileWriter
  - Upis tekstualnih podataka u fajlove

# Rad sa Stream klasama

1. Otvaranje streama instanciranjem novog stream objekta
2. Rad sa informacijama pomoću read/write, read/write u okviru metoda Stream klasa
3. Zatvaranje streama pozivom **`close()`** metoda datog objekta

# Java I/O klase

- Paket java.io sadrži klase koje s ekoriste za čitanje/upis podataka iz/u fajlove
- Da bi se čitali/upisivali podaci, moraju se intancirati podklase jedne od sledeće 4 abstraktne superklase:

| | input | output |
|---|---|---|
| byte | `InputStream` | `OutputStream` |
| character | `Reader` | `Writer` |

# Upotreba Reader klase

- Klasa Reader se koristi za čitanje karaktera ulaznog streama
- Ova klasa sarži metode za čitanje pojedinačnog karaktera i niza karaktera
  - **`int read()`**
- Klasa Reader je apstraktna, tako da se mora instancirati u podklasi koja mora preklopiti ove metode

# Čitanje iz tekstualnog fajla

```
public void readFile() {
   FileReader fileReader = null;
   try {
Step 1 →   fileReader = new FileReader("input.txt");
     int c = fileReader.read();
Step 2 {    while (c != -1) {
         char d = ((char)c);
         c = fileReader.read();
       }
   } catch (FileNotFoundException e) {
       System.out.println("File was not found");
   } catch (IOException e) {
       System.out.println("Error reading from file");
   }
   if (fileReader != null) {
Step 3 → try { fileReader.close(); }
       catch (IOException e) { /* ignore */ }
   }
}
```

# BufferedReader klasa

- BufferedReader je podklasa klase Reader
- Skuplja (buffers) karakter stream iz FileReader i sadrži metod **`readLine()`** za efikasno čitanje ulazne linije karaktera

```
FileReader fr = new FileReader("myFile.txt");
BufferedReader br = new BufferedReader(fr);
```

- Merod **`readLine()`** kao rezultat vraća vrednost **`null`** ako nema više linija za čitanje

# Upotreba BufferedReader klase

```
public void readFileWithBufferedReader() {
   BufferedReader bufferedReader = null;
   try {
      FileReader fr = new FileReader("input.txt");
      bufferedReader = new BufferedReader(fr);
     String line = bufferedReader.readLine();
      while (line != null) {
         // do something with line
         line = bufferedReader.readLine();
      }
   } catch (FileNotFoundException e) {
      System.out.println("File was not found");
   } catch (IOException e) {
      System.out.println("Error reading from file");
   }
   if (bufferedReader != null) {
      try { bufferedReader.close(); }
      catch (IOException e) { /* ignore */ }
   }
}
```

# Provera znanja

- Why can we not create instances of the Readerclass directly?
  - Readeris an Abstract class, and cannot be instantiated
- Which kind of stream would we use to read/write data in binary format?
  - Byte Streams
- Which kind of stream would we use to read/write data in text format?
  - Character Streams
- Why do we wrap a FileReaderwith a BufferedReaderbefore reading from a Text file?
  - BufferedReaderhas the readLine() method used to read entire lines

# Klasa Writer

- Klasa Writer je apstraktna klasa koja se koristi upis karakter stream-ova
- Ova klasa sadrži metode za upis pojedinačnog karaktera i stringova

```
void write(int c)
```

- BufferedWriter (podklasa klase Writer) sadrži metode za efikasni upis
- Metodom **`newLine()`** se upisuje prazna linija, a metodom **`write(String n)`** se upisuju konkretni podaci
- Kada se završi željena operacija potrebno je zatvoriti Write pomoću metoda **`close()`**

# Upis u tekstualni fajl

```
public void writeFileWithBufferedWriter() {
    BufferedWriter buffWriter = null;
    try {
        FileWriter fw = new FileWriter("output.txt");
        buffWriter = new BufferedWriter(fw);
        while (/*still stuff to write */) {
            String line = // get line to write
            buffWriter.write(line);
            buffWriter.newLine();
        }
    } catch (IOException e) {
        System.out.println("Error writing to file");
    }
    if (buffWriter != null) {
        try { buffWriter.close(); }
        catch(IOException e) { /* ignore */ }
    }
}
```

# Kopiranje fajlova

```
void copyFiles(String inFilename, String outFilename)
   throws FileNotFoundException {
 BufferedReader br = null;
 BufferedWriter bw = null;
 try {
   br = new BufferedReader(new FileReader(inFilename));
   bw = new BufferedWriter(new FileWriter(outFilename));
   String line = br.readLine();
   while(line != null) {
     bw.write(line);
     bw.newLine();
     line = br.readLine();
   }
 } catch (IOException e) {
   System.out.println("Error copying files");
 }

 if (br != null) {try {br.close();} catch(IOException e) {}}
 if (bw != null) {try {bw.close();} catch(IOException e) {}}
}
```

# Prihvatanje podataka sa tastature

- Unos podataka sa tastature u formi Streama se označava kao "standardni" unos, ali za čitanje unetih podataka potrebno je koristiti objekat klase Reader
- InputStream se koristi kao prelazna klasa, koja uzima podatke iz Streama i konvertuje ih u tip Reader
- Da bi se čitali karakteri pomoću InputStream, potrebno ih je podeliti u okviru InputStreamReader
- Da bi se čitala linija po linija, potrebno je podeliti InputStreamReader sa BufferedReader

# Prihvatanje podataka sa tastature

```
/**
 * Returns a line read from keyboard input.
 * Return null if there was an error reading the line.
 */
public void String readKeyboardLine() throws IOException {
  BufferedReader br = null;
  String line = null;
  try {
    br = new BufferedReader(new InputStreamReader(System.in));
    line = br.readLine();
  } catch (IOException e) {}

  if (br != null) {
    try { br.close(); }
    catch (IOException e) { /* ignore */ }
  }
  return line;
```

# Zaključak I/O

- Potrebno je proučiti hijerarhiju klasa InputStream i OutputStream, kao i Reader i Writer u okviru Java dokumentacije da bi se uočile sve postojeće podklase i metodi koji se mogu koristiti
- Treba koristiti Java API

# Parsiranje

- U okviru programa podaci se obično konvertuju u tekstualni format pre nego što se upišu u fajlove
- Kasnije je potrebno upisane podatke iz tekstualnih fajlova konvertovati u originalne podatke
- Proces dekodovanje teksta u podatke se još naziva i parsiranje

# Delimiteri

- Kada su podaci smešteni u tekstualnom formatu, delimiter karakteri se koriste da bi se odvojili delovi (tokens) podataka
- Na primer pojedinačna imena u listi su odvojena pomoću delimitera '#':

```
Lana#Pera#Maja#Mika
```

- Ista lista sa delimiterom novom linijom:

```
Lana
Pera
Maja
Mika
```

- Ostali delimiteri koji se koriste su ‘|’ ili ‘:’ ili tab

# StringTokenizer

- Kada se pokušava pročitati nova ulazna linija, kao rezultat se dobija jedan, u opštem slučaju dugački string
- Potrebno je pronaći delimitere u okviru tog stringa i odvojiti svaki pojedinačni deo informacije (tokens)
- Da bi se obavila ova operacija koristi se klasa StringTokenizer u okviru paketa java.util

# StringTokenizer

- Kada se formira tokenizer objekat, potrebno je specificirati koji se karakteri koriste kao delimiteri u konkretnom slučaju
- Default konstruktori pretpostavljaju da su “\t\n\r” delimiteri

```
StringTokenizerr = new StringTokenizer(line);
```

- Drugi konstruktori prihvataju kao argument String koji može definisati proizvoljan niz karaktera kao delimiter

```
String line = myFile.readline();
StringTokenizert = new StringTokenizer(line, “#”);
StringTokenizers = new StringTokenizer(line, “,\&\|”);
```

# StringTokenizer

- Korisni StringTokenizer metoda:
- Metod nextToken() u formi stringa kao rezultat vraća sledeći podatak između delimitera u okviru teksta
- Metod hasMoreTokens() u formi boolean vraća rezultat true ako se u tekstu nalazi još tokena

# StringTokenizer

- Printing out every name from a file where names are delimited by whitespace:

```
public void printNamesFromFile(String filename) {
  BufferedReader br = null;
  try {
    br = new BufferedReader(new FileReader(filename));
    String line = br.readLine();
    while(line != null) {
        StringTokenizer st = new StringTokenizer(line);
        while(st.hasMoreTokens()) {
          System.out.println(st.nextToken());
        }
        line = br.readLine();
    }
  } catch (IOException e) {
    System.out.println("Error reading from file.");
  }
  if (br != null) { try { br.close(); } catch(IOException e) {} }
}
```

# Parsiranje brojeva

- Često je potrebno parsirati brojeve smeštene kao tekst u okviru Java promenljivih
- Klase koje sadrže statičke metode za razne konverzije su
  **`int Integer.parseInt(Strings)`**
  **`double Double.parseDouble(Strings)`**
- Potrebno je obraditi izuzetak NumberFormatException ako se specificirani String ne može konvertovati u okviru promenljive

Citaj.java

```
package obuka02;
//Citaj.java –
// Citanje podataka standardnih tipova iz ulaznog toka,
//            iz datoteke i s glavnog ulaza.
import  java.io.*;
public class Citaj {
private InputStream ut;     // Ulazni tok iz kojeg se cita.
private char c;             // Poslednji procitani znak.
private boolean eos;        // Indikator kraja toka.
public Citaj(InputStream uut) { ut = uut; }
// Inicijalizacija ulaznim tokom.
public Citaj(String ime) throws FileNotFoundException
// Otvaranje
 { ut = new FileInputStream(ime); }
//   datoteke.
public boolean endS() { return eos; } // Da li je kraj toka?
```

```
public char getChS() {     // Dohvatanje sledeceg znaka.
 try { int i = ut.read(); return c = (eos = i == -1) ? ' ' : (char)i;
}
   catch (Exception g) { eos = true; return c = ' '; }
}
public char CharS() {      // Citanje jednog (nebelog) znaka.
 while (Character.isWhitespace(c = getChS()));
 return !eos ? c : ' ';
}
public String StringS() {  // Citanje jedne reci.
 String s = "";
 while ( Character.isWhitespace(c = getChS()) && !eos);
 if (eos) return "";
 s += c;
 while (!Character.isWhitespace(c = getChS()) && !eos) s += c;
 eos = false;
 return s;
}
```

Citaj.java

```
public String LineS() {     // Citanje jednog reda teksta.
 String s="";
 while ((c = getChS()) != '\n' && !eos) if (c != '\r') s += c;
 if (s.length() != 0) eos = false;
 return s;
}
public void getNLS()        // Preskakanje znakova do kraja reda.
 { while (c!='\n' && !eos) c = getChS(); c = '\0'; }
public byte   ByteS    ()  // Citanje jednog podatka tipa byte.
 { String s = StringS (); return !eos ? Byte.parseByte(s) : 0; }
public short  ShortS   ()  // Citanje jednog podatka tipa short.
 { String s = StringS (); return !eos ? Short.parseShort(s) : 0; }
public int    IntS     ()  // Citanje jednog podatka tipa int.
 { String s = StringS (); return !eos ? Integer.parseInt(s) : 0; }
public long   LongS    ()  // Citanje jednog podatka tipa long.
 { String s = StringS (); return !eos ? Long.parseLong(s) : 0; }
```

Citaj.java

```
public float  FloatS   ()  // Citanje jednog podatka tipa float.
 { String s = StringS (); return !eos ? Float.parseFloat(s) : 0; }

public double DoubleS  ()  // Citanje jednog podatka tipa double.
 { String s = StringS (); return !eos ? Double.parseDouble(s) : 0; }

public boolean BooleanS()  // Citanje jednog podatka tipa boolean.
 { String s = StringS (); return !eos ? Boolean.parseBoolean(s) :
false; }

// PODRSKA ZA CITANJE S GLAVNOG ULAZA.
```

Citaj.java

```
private static Citaj gl = new Citaj(System.in);   // Predstavnik
                                                  //   glavnog ulaza.
public static boolean end    () { return gl.endS    (); } // Varijante
public static char    getCh  () { return gl.getChS  (); } //   metoda
public static char    Char   () { return gl.CharS   (); } //   koje
public static String  String () { return gl.StringS (); } //   citaju sa
public static String  Line   () { return gl.LineS   (); } //   glavnog
public static void    getNL  () {        gl.getNLS  (); } //   ulaza.
public static byte    Byte   () { return gl.ByteS   (); }
public static short   Short  () { return gl.ShortS  (); }
public static int     Int    () { return gl.IntS    (); }
public static long    Long   () { return gl.LongS   (); }
public static float   Float  () { return gl.FloatS  (); }
public static double  Double () { return gl.DoubleS (); }
public static boolean Boolean() { return gl.BooleanS(); }
}
```

Citaj.java

**Profesor dr Miroslav Lutovac**
mlutovac@viser.edu.rs