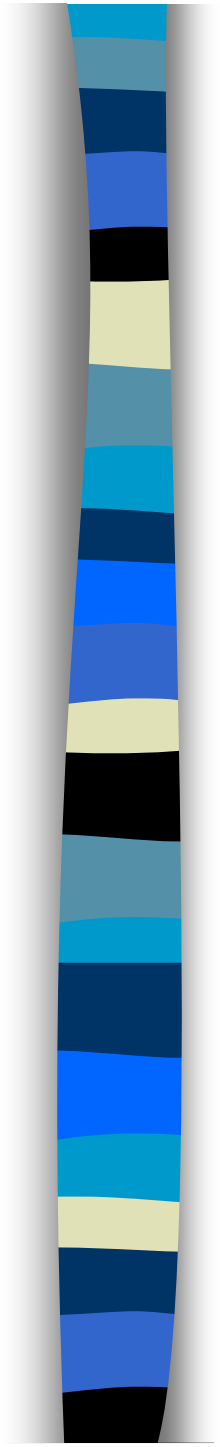


Objektno orijentisano programiranje

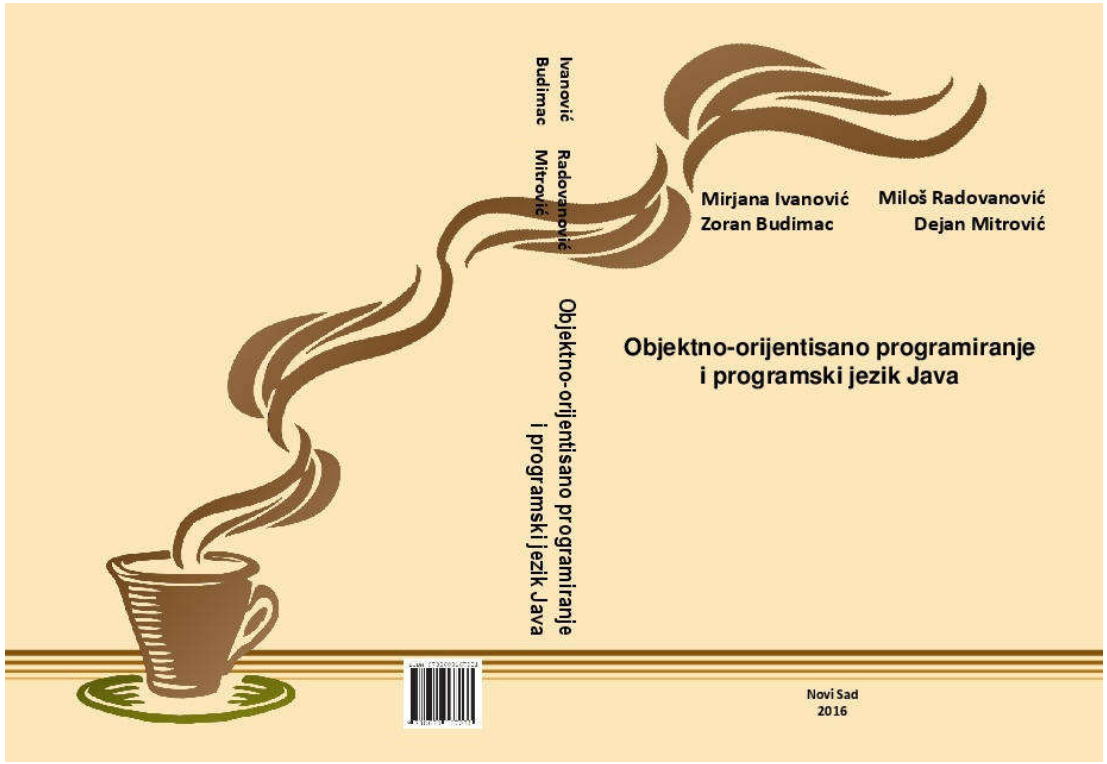
Aleksandra Klašnja-Milićević
Marko Marković

Teme koje će biti obrađene

- Uvod i koncepti OOP
- Klase
- Nasleđivanje
- Interfejsi
- Modifikatori, kreiranje instance klase, uništavanje instance klase
- Nabrojivi tip podataka
- Stringovi
- Operatori nad referencijalnim tipovima podataka
- Paketi
- Izuzeci
- Kolekcije
- Razvoj vizuelnih aplikacija

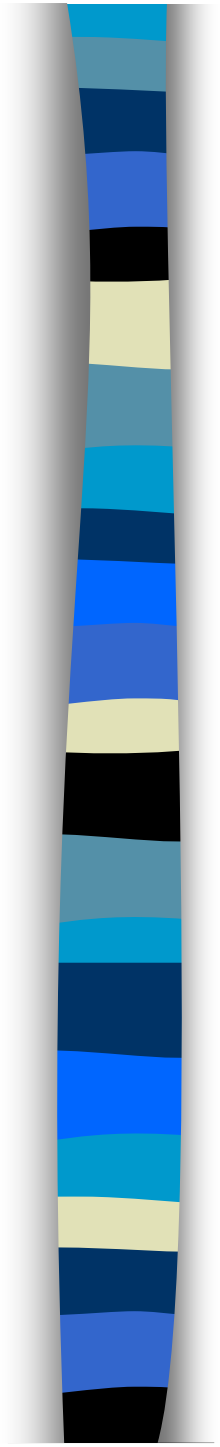
Knjiga

- Mirjana Ivanović
- Miloš Radovanović
- Zoran Budimac
- Dejan Mitrović



“Objektno orijentisano programiranje i programski jezik Java”
Novi Sad, 2016.

- **Prodajna mesta:**
- – Knjižara SOLARIS (SPENS), Novi Sad
- – Knjižara Studentski trg, Beograd



Osnovni principi objektno orijentisanog programiranja - OOP

OOP - osnovni koncepti

- **Objekat** - U softverskom smislu, objekat se može posmatrati kao neka vrsta paketa (svežanj) koji se sastoji od stanja i ponašanja.
  - Softverski objekti se koriste da modeliraju objekte iz realnog sveta, a za koji se pravi softversko rešenje.
- **Klasa** - Klasa je model/šema/plan/prototip na osnovu koga i po ugledu na koga se kreiraju objekti iste familije.
- **Nasleđivanje** – Nasleđivanje omogućuje moćan i prirodan mehanizam za organizaciju i struktuiranje softvera.
- **Paket** - paket je imenovani prostor/celina koja omogućava da se klase i interfejsi organizuju na smislen i logički povezan način.

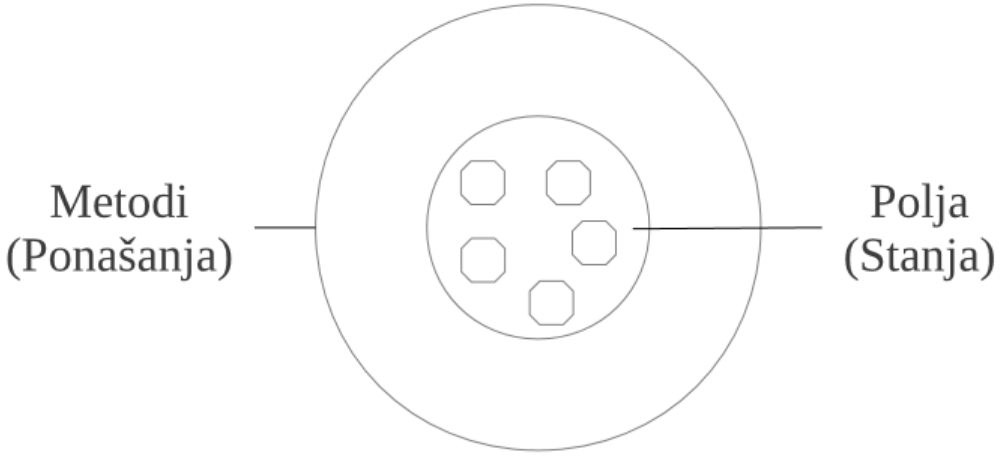
Objekti

- Objekti su ključni koncept objektno-orijentisanog programiranja, i njihova osnovna namena je da služe za predstavljanje objekata iz realnog sveta, koje prepoznajemo svuda oko sebe: automobili, kućni ljubimac, i tako dalje.
- Svi ovi realni objekti imaju dve zajedničke karakteristike: **stanje i ponašanje.**
- Tako, na primer, Automobil ima:
  - stanje koje obuhvata: model, boju, vrsta motora, konjske snage, broj vrata, i broj sedišta.
  - ponašanje Automobila obuhvata: otključaj vrata, promeni boju, promeni zupčanik, podigni ručnu kočnicu, da li je ručna kočnica podignuta, pritisni kočnicu, i povećaj brzinu.
- Kućni ljubimac ima:
  - stanje koje obuhvata: ime, boja, vrsta, i broj nogu, kao i
  - ponašanje koje obuhvata: kreće se, ispušta zvuk, pokazuje naklonost.

Objekti

- Identifikacija stanja i ponašanja objekata iz realnog sveta je sjajan način da se započne razmišljanje u duhu objektno-orijentisanog programiranja.
- Ako počnemo da posmatramo razne objekte iz okruženja oko nas, uočićemo svakako da oni variraju po svojoj složenosti i kompleksnosti i da imaju siromašnija ili bogatija i stanja i ponašanja.
- Takođe se može uočiti da neki objekti u sebi sadrže druge objekte.
- Sve ovo nam pomaže da objekte iz realnog okruženja „prevedemo“, odnosno na adekvatan način „prikažemo“ u svetu objektno-orijentisanog programiranja.

Objekti



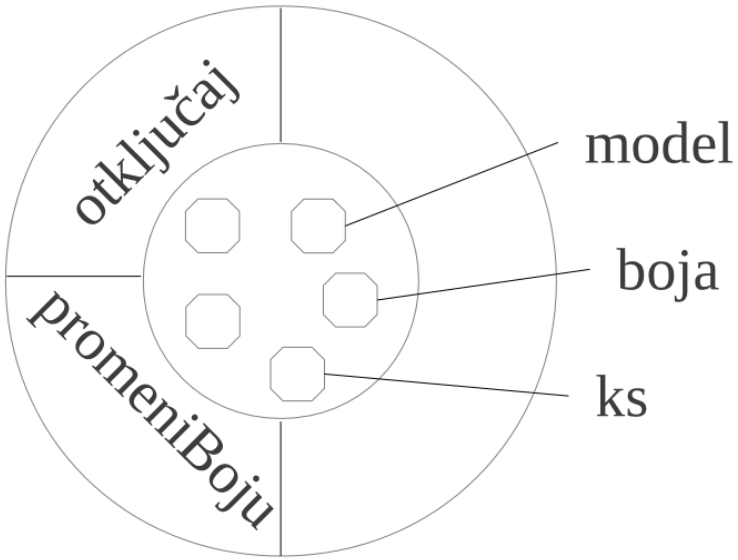
Slika 2.2: Softverski objekat

- Konceptualno, softverski objekti su slični objektima iz realnog sveta i oni se, naravno sastoje iz stanja i odgovarajućeg ponašanja.

Objekti

- Stanje objekta je organizovano u formi **polja**
  - (često se zovu i promenljive instance, odgovaraju pojmu promenljive u programskim jezicima) i
  - realizuje svoje ponašanje odgovarajućim metodima (koji odgovaraju pojmu procedura i funkcija u programskim jezicima).
- **Metodi** izvršavaju aktivnosti nad internim stanjem objekta i osnovni su mehanizam za komunikaciju između objekata.
- **Sakrivanje stanja objekta** i nedopuštanje direktnog pristupa i menjanja istog, kao i zahtev da se svaki vid interakcije objekta odvija isključivo preko metoda objekta je poznat kao „skrivanje podataka“ (*eng*. *data encapsulation*).
- Skrivanje podataka predstavlja jedan od fundamentalnih principa objektno-orijentisanog programiranja.

Objekti



Slika 2.3: Automobil modeliran kao softverski objekat

- Sa odgovarajućim vrednostima
  - stanja (polja koja predstavljaju model, boju, konjske snage) i
  - metodima za pristup i promenu tekućeg stanja (oključaj, promeni boju, ...)
- uspostavljena je kompletna kontrola i način pristupa i menjanja stanja objekta kao reakcija na poruke pristigle iz spoljašnjeg okruženja.

Objekti

- Pakovanje/uključivanje koda zajedno sa stanjem u jedinstvenu celinu tj. softverski objekat, pruža čitav niz prednosti:
- **Modularnost**: Izvorni kod jednog objekta može da se piše i menja nezavisno od izvornog koda drugih objekata.
- **Skrivanje informacija**: Kako se komunikacija sa objektom odvija preko metoda, njegova unutrašnja implementacija ostaje skrivena za spoljašnje okruženje.

Objekti

- **Ponovno korišćenje koda**: Ako neki objekat postoji (a napisao ga je i neki drugi programer), a pogodan je za rešenje vašeg problema, možete ga iskoristiti i uključiti u vaš novi program. Takva filozofija razvoja softvera omogućuje da specijalisti implementiraju i testiraju specifične objekte kojima možete verovati (da dobro funkcionišu), te ih bez dodatnih aktivnosti možete uključiti u vaš kod.

- **Jednostavna promena**: Ako neki objekat koji je uključen u vaš sistem iz nekih razloga ne funkcioniše dobro, možete ga jednostavno ukloniti iz sistema i uključiti umesto njega neki drugi objekat koji pouzdanije obavlja željenu aktivnost.

Objekti

- Objekat treba posmatrati kao celinu koja zna kako da reaguje i odgovori na konkretnu poruku.
- Međutim treba tako imati u vidu da objekti na istu poruku mogu da odgovore na različit/drugačiji način.
- Tako na primer poruka „štampaj“ može da produkuje različite rezultate u zavisnosti od toga kom je objektu poslata.
- Ta osobina objekata, da različiti objekti mogu na istu poruku da odgovaraju na različite načine, se zove polimorfizam (eng. polymorphism).

OOP - osnovni koncepti

- Objekti koji sadrže isti tip podataka i koji odgovaraju na iste poruke na isti način pripadaju istoj porodici objekata odnosno pripadaju istoj **klasi** (eng. class).
- U suštini, u OOP primarni koncept je klasa, i proizvoljan broj objekata (iz iste familije) se može kreirati koristeći klasu kao etalon (u literaturi je poznato puno sinonima za ovaj termin: model, uzorak, itd.).
- Međutim, može da postoji i čitav niz sličnih objekata koji nisu nastali iz iste klase.

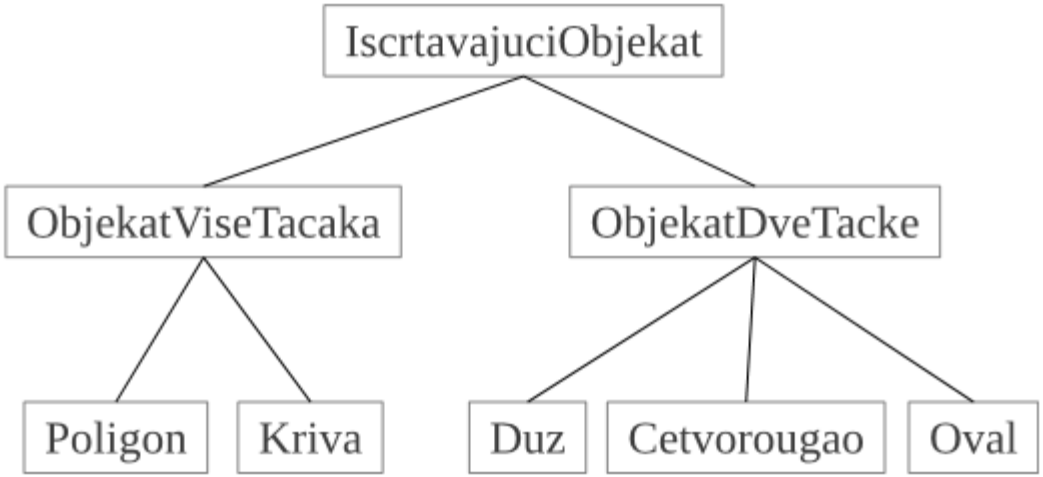
OOP - osnovni koncepti

- Tako, na primer, možemo posmatrati program za crtanje linija i geometrijskih oblika kao što su: duž, četvorougao, ovali, poligoni i krive.
- Dakle, svaki objekat koji se vidi na ekranu se može predstaviti softverskim objektom u programu.
- U tom slučaju uočavamo u programu pet klasa objekata koji se mogu iscrtavati na ekranu.
- Sve duži pripadaju jednoj klasi, svi četvorouglovi pripadaju drugoj klasi i tako redom.
- Takođe, jasno uočavamo da su ove klase u određenoj relaciji i da sve predstavljaju iscrtavajuće objekte, kao i da treba da imaju mogućnost da reaguju na poruku „iscrtaj se“.

OOP - osnovni koncepti

- Drugi nivo grupisanja, koji nije tako očigledan, može da se realizuje u odnosu na podatke koji su potrebni za crtanje svakog od ovih tipova objekata.
- Tako možemo grupisati poligone i krive kao „objekte više tačaka“, a duži, četvorougle i ovale kao „objekte dve tačke“.
- (Duž je određena sa dve krajnje tačke, četvorougao sa dve tačke i dva naspramna ugla, a oval sa dva naspramna ugla četvorougla u koji je upisan).
- Relacije između pomenutih koncepata, od kojih svaki predstavlja jednu klasu, su prikazani na slici:

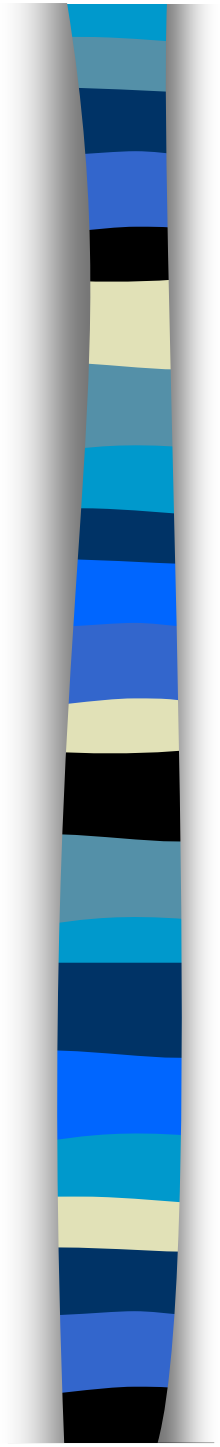
OOP - osnovni koncepti



- Klase *ObjekatViseTacaka* i *ObjektiDveTacke* su direktne podklase klase *IscrtavajućiObjekat*. Klasa *Duz* je direktna podklasa klase *ObjekatDveTacke* i indirektna klasa klase *IscrtavajuciObjekat*.
- Dakle, podklasa neke klase nasleđuje (eng. inherits) svojstva svoje nadklase. Podklasa takođe može da doda i neka nova svojstva (što je najčešći slučaj) na ona nasleđena iz nadklase, ali i da „promeni“ postojeća svojstva (eng. override).

OOP - osnovni koncepti

- Nasleđivanje je izuzetno moćan mehanizam u OOP stilu programiranja i rešavanja problema koji podržava dobru organizaciju i ponovnu upotrebu programskog koda.
- Klase su definitivno ponovo upotrebljive (eng. reusable) komponente.
- One se mogu dvostruko iskoristiti u novim programima:
  - ako u potpunosti odgovaraju takve kakve su bez izmena,
  - ako se uz manje ili veće izmene mogu prilagoditi novim programima, u tom slučaju se definiše nova klasa koja nasleđuje postojeću klasu.
- Na osnovu navedenog može se zaključiti da je OOP moćno oruđe za razvijanje softvera, ali i pogodno rešenje za ponovno korišćenje već razvijenog softvera (eng. software reuse).



Referencijalni tipovi podataka

Referencijalni tipovi podataka

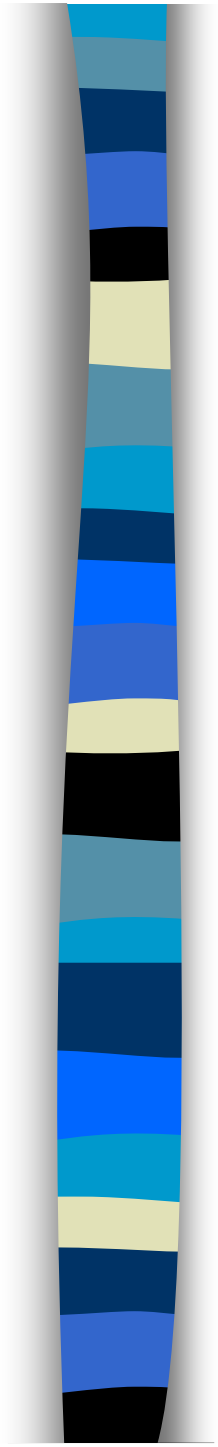
- Veći programi – pogodno je da se delovi programa koji čine **logičku celinu grupišu zajedno**, time se omogućava:
  - viši nivo apstrakcije
  - veća čitljivost progama
  - jednostavniji timski rad na programu
  - lakše održavanje, modifikacija i proširenje gotovog programa
- Kod jezika treće generacije, kao što su Pascal i Modula – 2, grupisanje podataka u slogove i nizove (složene tipove) se pokazalo kao **velika prednost** u odnosu na starije jezike koji koriste samo proste tipove podataka
- Iako pomenuti jezici podržavaju pravljenje složenih tipova podataka oni direktno ne obezbeđuju i grupisanje delova programskog koda sa podacima kojima taj deo koda pristupa

Referencijalni tipovi podataka

- Objektno orijentisano programiranje je novija paradigma programiranja – zasniva se na pravljenju i korišćenju tipova koji omogućavaju grupisanje podataka i delova programskog koda koji te podatke koristi
- Vrednosti (instance) takvih tipova se zovu **objekti**
- Java je objektno-orijentisan programski jezik. Bez korišćenja objekata ne može se napisati nijedan složeniji program.
- Tipovi pomoću kojih se u Javi prave objekti – **referencijalni tipovi**

Referencijalni tipovi podataka

- U Javi postoji pet vrsta referencijalnih tipova:
  - **Klase**
  - **Interfejsi**
  - **Nizovi**
  - **Nabrojivi tipovi**
  - **Lambda izrazi**
- Složeni tipovi podataka – referencijalni tip pravimo korišćenjem drugih referencijalnih i / ili prostih tipova
- Nazivaju se i **struktuirani tipovi**



Klase

Klase

- Klasa predstavlja model na osnovu kojeg će se praviti odgovarajući objekti u programu
- Klasa se sastoji od svojih članova - polja, metodi, konstruktori ...

Primer jedne klase

```
class Tacka {

  float x, y;

  Tacka(float X, float Y) {
    x = X;
    y = Y;
  }

  void transliraj(float promenaX, float promenaY) {
    x = x + promenaX;
    y = y + promenaY;
  }

  String opis() {
    return "Tacka sa Dekartovim koordinatama (" + x + ", " + y + ")";
  }
}
```

Polja klase – služe za čuvanje nekih podataka (vrednosti – predstavljaju stanja)

Konstruktor klase – za inicijalizaciju polja novo-kreiranog objekta

Metode – članovi klasa – izvršava se programski kod od kojeg se metod sastoji

Klase

- U realnom svetu postoji mnoštvo pojedinačnih objekata iste vrste.
- Npr. Postoji hiljade automobila istih karakteristika.
- Svaki od njih je napravljen od istog modela/šeme/plana/prototipa i sadrži iste komponenete.
- Neki konkretan automobil je jedna instanca (tj. objekat) klase automobil.

Primer 2.1: Mogući izgled klase `Automobil`

```
public class Automobil {
  // polja
  int konjskeSnage = 100;
  int brojVrata = 4;
  int godina = 2004;
  String jedinstvenBroj = "1M8GDM9A_KP042788";
  String boja = "zelena";
  String model = "Passat";
  String make = "Volkswagen";
  // ...

  // metodi
  void otvoriVrata() { /* ... */}
  void promeniBoju(String boja) { /* ... */}
  void promeniZupcanik(char oznaka) { /* ... */}
  boolean daLiJeRucnaKocnicaUkljucena() { /* ... */}
  void podigniRucnuKocnicu() { /* ... */}
  void spustiRucnuKocnicu() { /* ...*/}
  void pritisniBrzinu(float procenat) { /* ... */}
  void pritisniKocnicu(float procenat) { /* ... */}
  String ispisiStatus() { /* ... */}
}
```

Klase

Primer 2.2: Primer pravljenja objekata klase `Automobil` u programskoj klasi `AutoDemo`

```
class AutoDemo {
  public static void main(String[] args) {
    // glavni metod koji se startuje i izvrsava program tj. planirane
        aktivnosti

    // Kreiranje dva objekta klase Automobil
    Automobil auto1 = new Automobil();
    Automobil auto2 = new Automobil();

    // Poziv metoda ovih objekata
    auto1.promeniBoju("crvena");
    auto1.pritisniBrzinu(10f);
    auto1.promeniZupcanik('3');
    auto1.ispisiStatus();

    auto2.promeniBoju("crvena");
    auto2.podigniRucnuKocnicu();
    auto2.pritisniBrzinu(20f);
    auto2.promeniZupcanik('2');
    auto2.ispisiStatus();
  }
}
```

Promenljiva

- Pod promenljivom podrazumevamo:
  - lokalne promenljive
  - parametre metoda i konstruktora
  - polja klasa
- Vrednost promenljive, čiji je tip neka klasa, može biti samo referenca na objekat te klase ili konstanta **null.**
- Ako promenljiva ima vrednost **null**, tada ona ne pokazuje ni na jedan objekat. Vrednost **null** se može dodeliti promenljivoj bilo kog referencijalnog tipa.

Primer kreiranja instance klase

- Klase su tipovi podataka, a vrednosti ovakvih tipova podataka nazivamo instancama klasa ili objektima.
- Za kreiranje objekta neke klase koristimo **new** operator, nakon kojeg možemo navesti poziv konstruktora odgovarajuće klase.
- Rezultat primene **new** operatora - pokazivač na novokreirani objekat.

Primer programa – korišćenje klase Tacka

```
class program {
  public static void main(String[] args) {
    Tacka A;
    A = new Tacka(1, 3);
    System.out.println(A.opis());  // (1.0, 3.0)
    A.transliraj(1, 2);
    System.out.println(A.opis());  // (2.0, 5.0)
    A.x = 0;
    A.y = 0;
    System.out.println(A.opis());  // (0.0, 0.0)
  }
}
```

Dve glavne karakteristike objekata

- Objekti nastaju dinamički, u toku rada programa, primenom **new** operatora. **new** operatorom se alocira memorijski prostor koji objekat zauzima. U Javi se oslobađanje memorijskog prostora zauzetog od strane objekata vrši automatski.
- Promenljive referencijalnog tipa kao svoju vrednost ne sadrže objekte, već su njihove vrednosti pokazivači na objekte.

Posledica druge osobine

```
class program {
  public static void main(String[] args) {
    Tacka A = new Tacka(1, 1);
    System.out.println(A.opis());  // (1.0, 1.0)
    Tacka B = A;
    B.transliraj(2, 2);
    System.out.println(A.opis());  // (3.0, 3.0)
  }
}
```

Dve glavne karakteristike objekata

- Promenljiva kao svoju vrednost sadrži referencu na objekat a ne sam objekat.
- Upoređivanjem vrednosti dve promenljive operatorima za ispitivanje jednakosti == i != ispituje se da li te dve promenljive pokazuju ili ne pokazuju na isti objekat, a ne da li su dva objekta na koje one pokazuju jednaka ili nejednaka.
- Ako dve promenljive imaju vrednost null, tada one imaju jednaku vrednost.

Karakteristike objekata - primer

Primer – ispitivanje jednakosti

```
class program {
  public static void main(String[] args) {
    Tacka A = new Tacka(2, 2);
    Tacka B = new Tacka(2, 2);
    Tacka C = A;
    Tacka D = null;
    Tacka E = null;
    if (A == B)
      System.out.println("A == B");
    else
      System.out.println("A != B");
    if (A == C)
      System.out.println("A == C");
    else
      System.out.println("A != C");
    if (B == C)
    System.out.println("B == C");
    else
        System.out.println("B != C");
      if (D == E)
        System.out.println("D == E");
      else
        System.out.println("D != E");
    }
}
```

Ispis

```
A != B
A == C
B != C
D == E
```

Overloading

- **Overloading** – korišćenje istog imena ili simbola za označavanje više različitih konstrukcija u jeziku.
- U Javi je to mogućnost deklarisanja više istoimenih metoda i konstruktora u jednoj klasi.
- Ovi metodi se moraju međusobno razlikovati po broju ili tipu svojih formalnih parametara.
- Pored *overloading*-a imena metoda, u Javi se *overloading* koristi i kod dva operatorska simbola: **+** se koristi kao unarni plus (predznak), kao simbol sabiranja brojeva i kao simbol konkatenacije stringova a simbol **-** se koristi kao unarni minus (predznak) i kao simbol oduzimanja brojeva.

Overloading - primer

Primer overloading-a

```
class Recenica {
  String sadrzaj;
  Recenica() {
    sadrzaj = "";
  }
  Recenica(String pocetniSadrzaj) {
    sadrzaj = pocetniSadrzaj;
  }
  void dodaj(String rec) {
    sadrzaj += rec + " ";
  }
  void dodaj(String prvaRec, String drugaRec) {
    sadrzaj += prvaRec + " " + drugaRec + " ";
  }
  void ispisi() {
    System.out.println("Recenica je: " + sadrzaj);
  }
}
```

```
class program {
  public static void main(String[] args) {
    Recenica r = new Recenica();
    r.dodaj("Ovo");
    r.dodaj("je", "jedna");
    r.dodaj("recenica.");
    r.ispisi();
  }
}
```

Enkapsulacija

- Pod **dizajnom klase** podrazumevamo načine na koje će se objektima te klase pristupati, tj. kako ta klasa izgleda spolja: kojim poljima se može pristupati izvan tela klase, kojim metodima i šta ti metodi treba da urade.
- Pod **implementacijom klase** podrazumevamo konkretnu realizaciju svega onoga što smo prilikom dizajniranja klase zamislili.
- Svaki Java program se uglavnom sastoji od nekih klasa. Da bi se program lakše napravio, održavao i modifikovao, kao i da bi se omogućio timski rad na pravljenju programa, veoma je pogodno da implementacija svake klase ostane sakrivena koliko god je to moguće.
- Kako je neka klasa implementirana, to treba da zna samo programer koji ju je napravio. Kreatori drugih klasa ne moraju znati ništa o implementaciji posmatrane klase da bi je koristili.

Enkapsulacija

- Razlikujemo dva nivoa:
  - **Nivo dizajna klase** – spoljašnji izgled, način pristupa...
  - **Nivo implementacije klase** – konkretna realizacija
- **Enkapsulacija** – skrivanje detalja implementacije klase
- **Značaj**: nesmetano modifikovanje klasa, održavanje objektno-orijentisanih programa (posebno velikih)

Direktan pristup poljima klase Tacka (bez enkapsulacije)

```
class program {
  public static void main(String[] args) {
    Tacka A = new Tacka(2, 3);
    A.transliraj(12.889f, -3.45f);
    System.out.println("Udaljenost od koordinatnog pocetka je " +
      Math.sqrt(A.x * A.x + A.y * A.y) ); //kriticno mesto
  }
}
```

Enkapsulacija - Modifikacija prvobitne klase Tacka (1/3)

- Klasa **Tacka** je izgubila polja **x** i **y** a dobila je nova polja **ugao** i **duzina** koja služe za predstavljanje tačke u polarnim koordinatama.
- Dodat je metod **rotiraj** koji rotira tačku oko koordinatnog početka.
- Takođe su dodati pomoćni metodi kojima se vrši konverzija iz Dekartovih u polarne koordinate i obrnuto.
- Konstruktor i metodi **transliraj** i **opis** su pretrpeli izmene zbog prelaska na polarne koordinate

```
class Tacka {

  float ugao, duzina;  //ugao je u radijanima u intervalu (-Pi, Pi]
  Tacka( float X, float Y) {
    duzina = nadjiDuzinu(X, Y);
    ugao = nadjiUgao(X, Y);
  }

  // metodi za konverziju iz Dekartovih u polarne koordinate

  float nadjiDuzinu(float x, float y) {
    return (float) Math.sqrt(x*x + y*y);
  }
```

Enkapsulacija - Modifikacija prvobitne klase Tacka (2/3)

```
float nadjiUgao(float x, float y) {
   float duz = nadjiDuzinu(x, y);
   if (duz == 0)
     return 0; //mozemo vratiti bilo sta
   else {
     double ug = Math.asin(Math.abs(y) / duz);
     if ((y >= 0) && (x <= 0)) {  //drugi kvadrant
       ug = Math.PI - ug;
     } else if (y < 0) {
       if (x <= 0) {  //treci kvadrant
         ug = ug - Math.PI;
       } else {  //cetvrti kvadrant
         ug = - ug;
       }
     }
     return (float) ug;
   }
 }

 // metodi za konverziju iz polarnih u Dekartove koordinate

 float nadjiX(float ug, float duz) {
   if (duz == 0)
     return 0;
   else
     return (float) (Math.cos(ug) * duz);
 }
```

Enkapsulacija - Modifikacija prvobitne klase Tacka (3/3)

```
  float nadjiY(float ug, float duz) {
    if (duz == 0)
      return 0;
    else
      return (float) (Math.sin(ug) * duz);
  }
  void rotiraj(float zaUgao) {
    ugao += zaUgao;
    if (ugao > Math.PI)
      do
        ugao = (float) (ugao - 2 * Math.PI);
      while (ugao > Math.PI);
    else if (ugao <= -Math.PI)
      do
        ugao = (float) (ugao + 2 * Math.PI);
      while (ugao <= -Math.PI);
  }
  void transliraj(float promenaX, float promenaY) {
    float noviX = nadjiX(ugao, duzina) + promenaX;
    float noviY = nadjiY(ugao, duzina) + promenaY;
    duzina = nadjiDuzinu(noviX, noviY);
    ugao = nadjiUgao(noviX, noviY);
  }
  String opis() {
    return "Tacka sa Dekartovim koordinatama (" + nadjiX(ugao, duzina) +
           ", " + nadjiY(ugao, duzina) + ")";
  }
}
```

Enkapsulacija - Prvobitna klasa Tacka+enkapsulacija (1/2)

```
class Tacka {

private float x, y;

Tacka( float X, float Y) {
  x = X;
  y = Y;
}

void transliraj(float promenaX, float promenaY) {
  x = x + promenaX;
  y = y + promenaY;
}

String opis() {
  return "tacka sa Dekartovim koordinatama (" + x + ", " + y + ")";
}

float getX() {  //vraca x koordinatu
  return x;
}

float getY() {  //vraca y koordinatu
  return y;
}
```

Enkapsulacija - Prvobitna klasa Tacka+enkapsulacija (2/2)

```
  void setX(float x) {  // postavlja x koordinatu
    this.x = x;  //koristimo this jer je x i ime parametra i ime polja
  }

  void setY(float y) {  // postavlja y koordinatu
    this.y = y;
  }
}
```

- Sakrivanje polja **x** i **y** se postiže modifikatorom **private**
- Pošto su polja **x** i **y** sada privatni članovi klase **Tacka**, oni su van tela klase nevidljivi i njima se izvan klase **Tacka** ne može pristupati direktno
- Zbog toga su dodati metodi za čitanje Dekartovih koordinata **getX** i **getY**, kao i metodi za postavljanje koordinata tačke **setX** i **setY**
- Umesto direktnog pristupa poljima **x** i **y**, program koristi metode **getX** i **getY**

```
class program {
  public static void main(String[] args) {
    Tacka A = new Tacka(2, 3);
    A.transliraj(12.889f, -3.45f);
    System.out.println("Udaljenost od koordinatnog pocetka je " +
      Math.sqrt(A.getX()*A.getX() + A.getY()*A.getY()));
  }
}
```

Ključna reč this

- U metodima setX i setY smo koristili ključnu reč **this**.
- **this** je referenca posmatranog tekućeg objekta
- nju možemo koristiti u situacijama kada se pristupa nekom od članova klase čije ime je nevidljivo u telu metoda zbog toga što neki parametar metoda ima to isto ime.

```
  void setX(float x) {  // postavlja x koordinatu
    this.x = x;  //koristimo this jer je x i ime parametra i ime polja
  }

  void setY(float y) {  // postavlja y koordinatu
    this.y = y;
  }
}
```

Statička polja i metodi

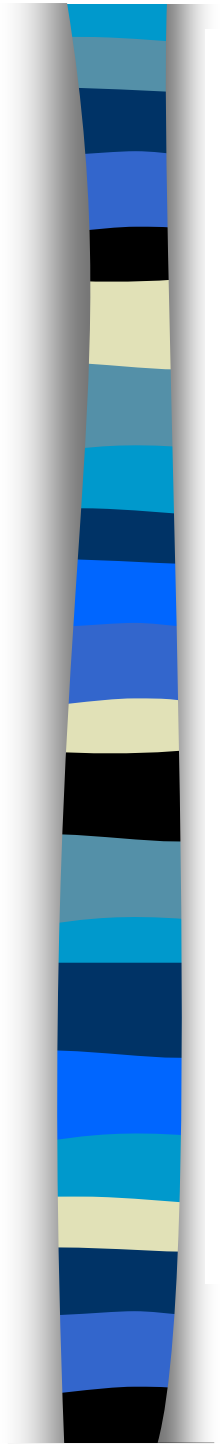
- Statički članovi klase - metodi i polja koja su deklarisana pomoću ključne reči **static.**
- Mogu se koristiti i bez prethodnog kreiranja instanci klase, oni se zapravo i ne odnose na instance klase, već na samu klasu.
- Svaki objekat klase sadrži svoju kopiju svih polja i metoda klase, osim onih polja i metoda koji su statički, jer statička polja i metodi pripadaju klasi a ne njenim instancama
- Statički metodi ne smeju pristupati poljima koja nisu statička, niti smeju pozivati metode koji nisu statički.
- Statičkim članovima klase se može pristupiti tako što se navede ime klase, tačka i ime statičkog člana.

Statička polja i metodi - primer

Primer upotrebe statičkih metoda

```
class Tacka {
  ...
  static float nadjiDuzinu(float x, float y) {
    ...
  }
  static float nadjiUgao(float x, float y) {
    ...
  }
  Tacka( float X, float Y) {
    ...
  }
  String opis() {
    ...
  }
  ...
```

```
class program {
  public static void main(String[] args) {
    for (int x = -10; x <= 10; x++)
      for (int y = -10; y <= 10; y++)
        System.out.println("(" + x + ", " + y +
          ") u Dekartovim je (" + Tacka.nadjiUgao(x, y) + ", " +
          Tacka.nadjiDuzinu(x, y) + ") u polarnim koordinatama.");
  }
}
```

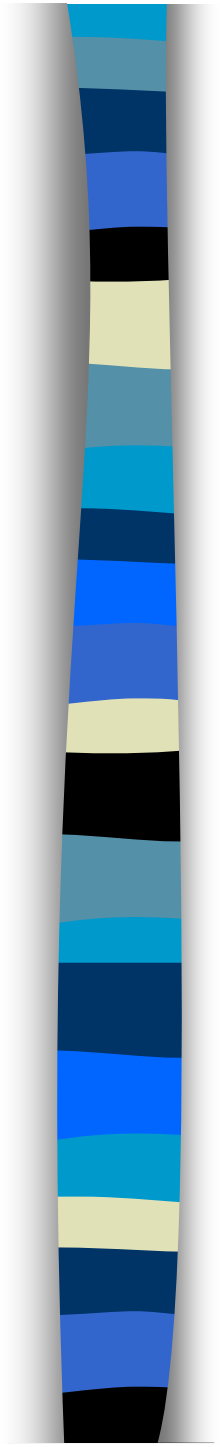


Teorijske vežbe 1

Objektno-orjentisano programiranje

Java je objektno-orjentisani PJ

- Program u izvršavanju (proces) – skup objekata koji razmenjuju poruke
- Objekat
  - podaci (atributi, polja)
  - metode koje manipulišu podacima objekta
- Poruka – poziv metoda nad nekim objektom
- Klasa
  - Definicija skupa objekata sa istim osobinama (atributima) i ponašanjem (metodama)
- Mehanizmi **enkapsulacije**, agregacije/kompozicije, nasleđivanja i polimorfizma



Uspešnost OO paradigme

- Visok stepen apstrakcije koji omogućava fokusiranje na problem
- Problemi iz realnog sveta su takvi da se prirodno modeluju objektima u interakciji
- Organizacija koda oko podataka
  - Podaci "diktiraju" kako se problem dekomponuje na manje podprobleme
- Dekompozicija problema u skup klasa omogućuje efikasan **timski rad, te i razvoj velikih programa**.

JRE i JDK

- JRE (Java Runtime Enviroment)
  - java virtuelna mašina
  - kompajlirana standardna java biblioteka
- JDK (Java Development Kit)
  - kompajler za Java programski jezik
  - Java virtuelna mašina
  - Standardna java biblioteka
  - ostali alati korisni prilikom razvoja Java aplikacija (Appletviewer, Javadoc, JConsole, Jar, itd.)
- **Da bi ste pokrenuli Java program treba vam JRE, da bi ste pravili (i pokretali) Java program treba vam JDK**
  - **Developeri – JDK, Useri – JRE**

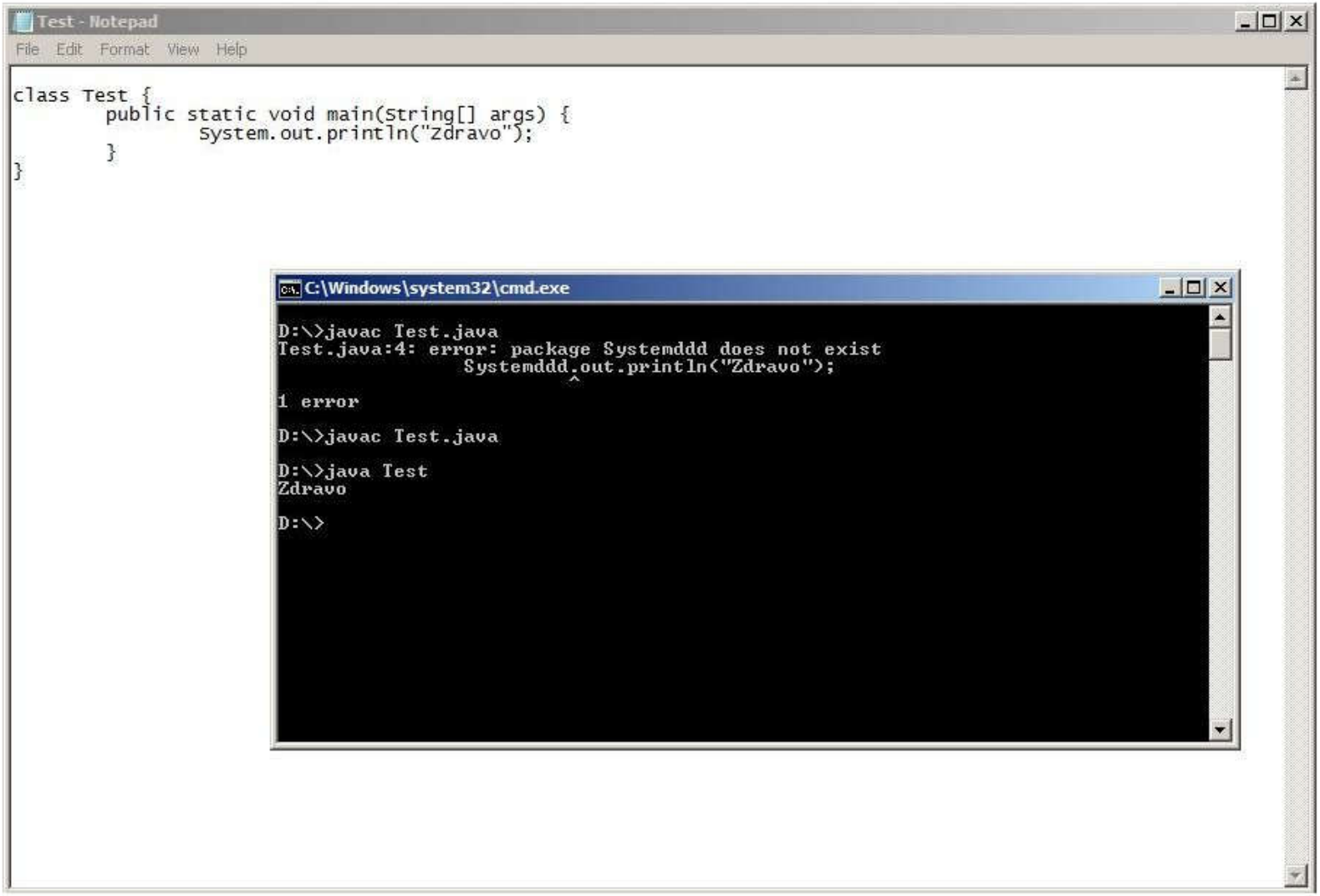
Razvojno okruženje

- Pisanje programa se sastoji se od:
  - Pisanja izvornog koda
  - Kompajliranja izvornog koda
  - Popravljanja sintaksnih grešaka
  - Pokretanja programa
  - Uočavanja bug-ova (semantičkih grešaka)
  - Popravljanja semantičkih grešaka

- IDE (Integrated Development Enviroment) omogućava da sve gore navedene stvari radimo u jednom programu i mnogo više

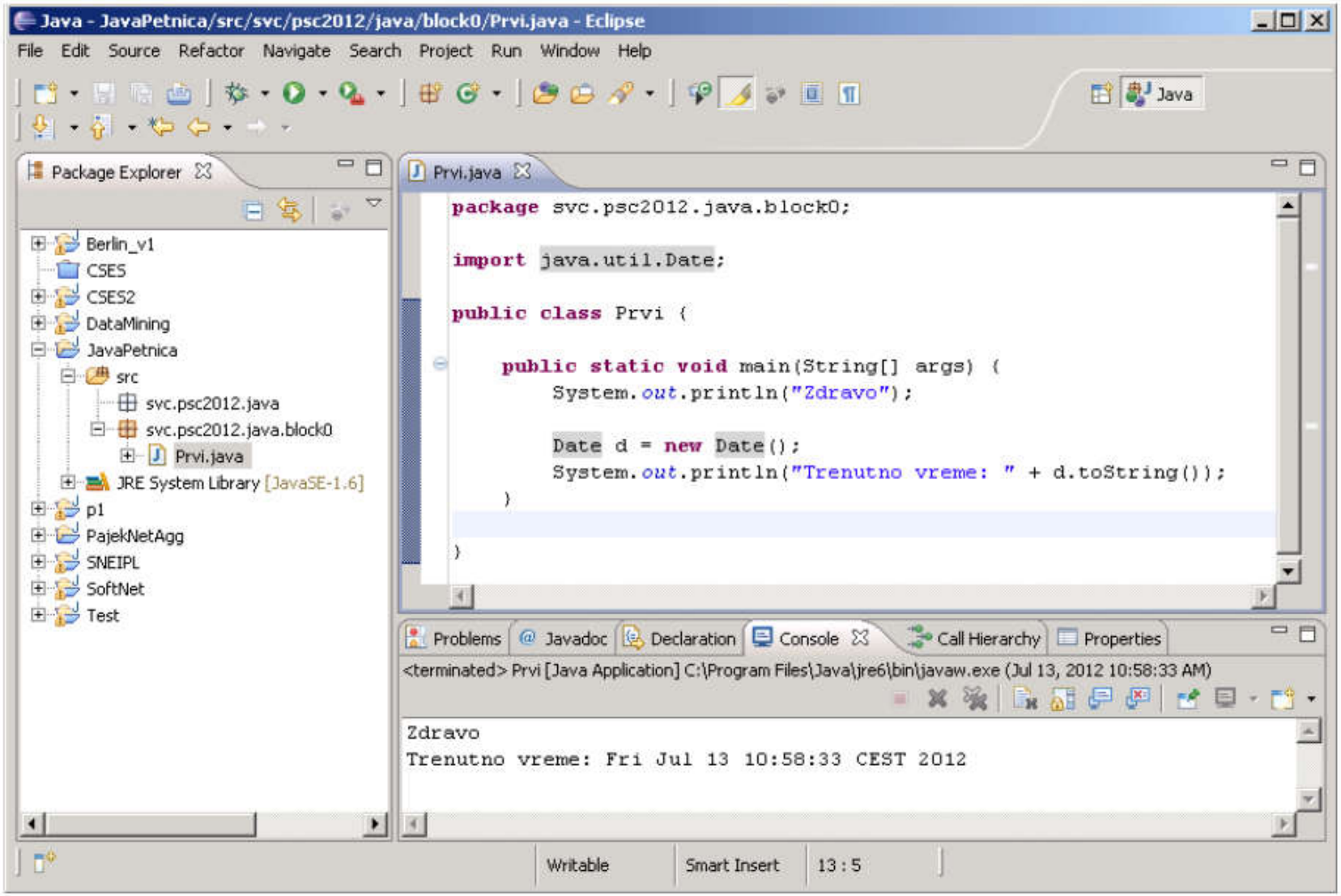
Pisanje programa u Javi bez razvojnog okruženja

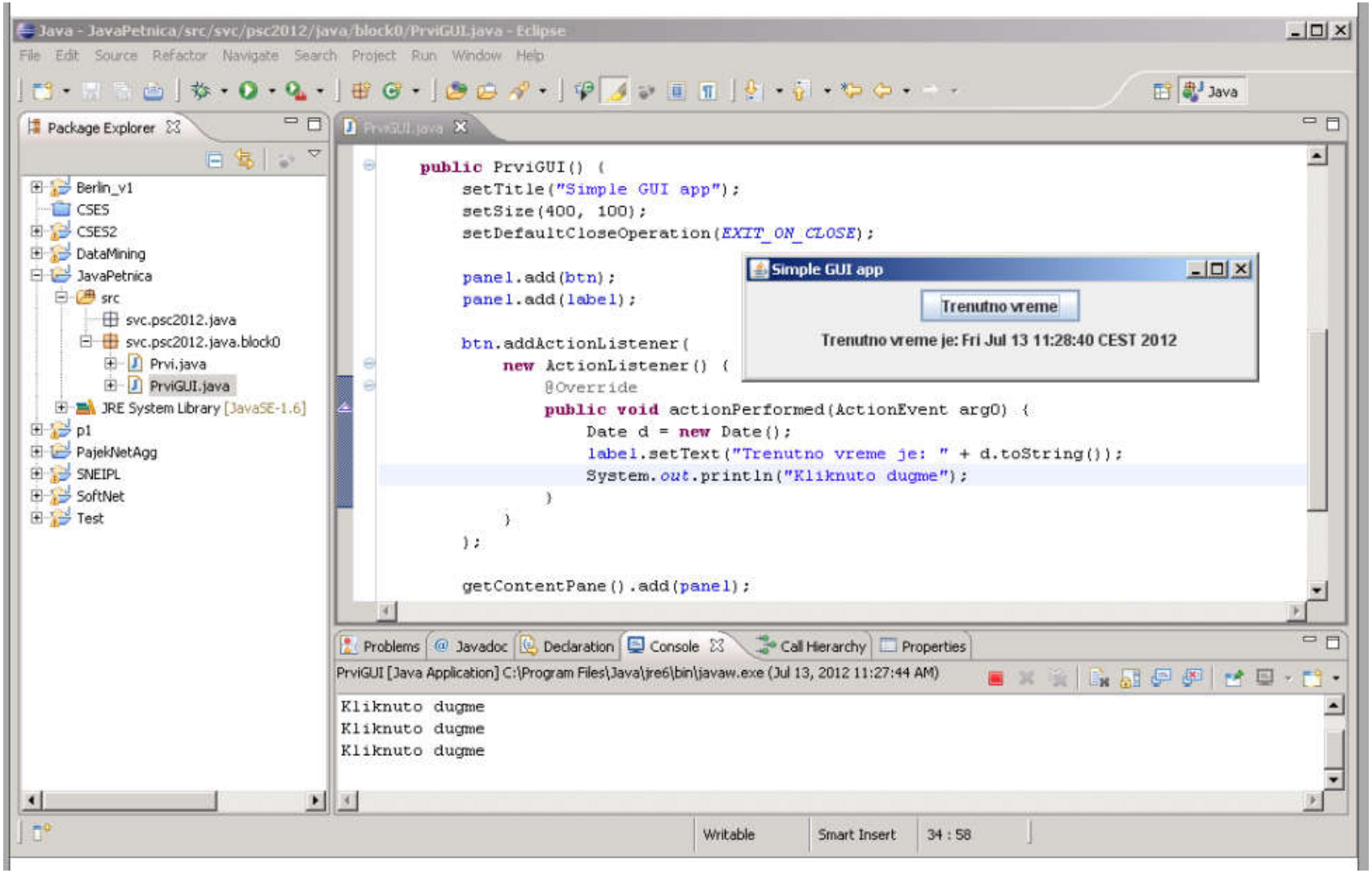
- Upotrebom bilo kog tekst editora otkucaj kod
- Kompajliraj kod korišćenjem “javac” programa
- Vrati se u editor i popravljaj sintaksne greške
- Pokreni kod korišćenjem “java” programa
- Vrati se u editor i popravljaj semantičke greške


Test - Notepad
File Edit Format View Help
class Test {
public static void main(String[] args) {
System.out.println("Zdravo");
}
}
C:\Windows\system32\cmd.exe
D:\>javac Test.java
Test.java:4: error: package Systemddd does not exist
Systemddd.out.println("Zdravo");
1 error
D:\>javac Test.java
D:\>java Test
Zdravo
D:\>

Razvojna okruženja za Javu

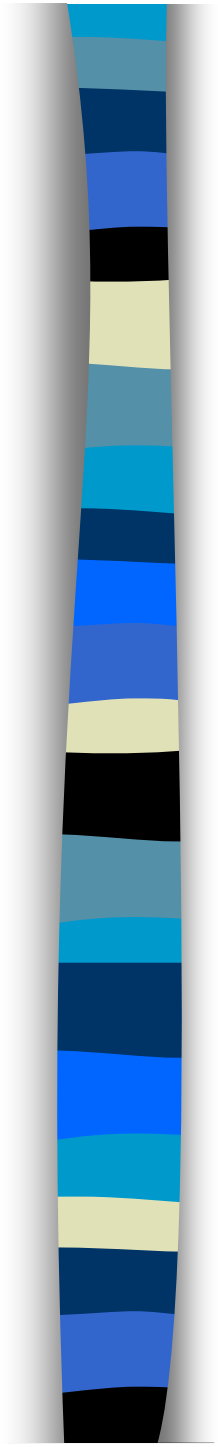
IDE	License	Written in Java	Windows	Linux	Mac OS X	Other platforms	GUI builder
BlueJ	GPL2+GNU linking exception	Yes	Yes	Yes	Yes	Solaris	No
DrJava	Permissive	Yes	Yes	Yes	Yes	Solaris	No
Eclipse JDT	EPL	Yes	Yes	Yes	Yes	Solaris	Yes
Geany	GPL	No	Yes	Yes	Yes	Solaris	No
Greenfoot	GPL	Yes	Yes	Yes	Yes	Solaris	No
IntelliJ IDEA	ALv2, proprietary	Yes	Yes	Yes	Yes		Yes
JBuilder	Proprietary	Yes	Yes	Yes	Yes	Solaris	Yes
JCreator	Proprietary	No	Yes	No	No		No
JDeveloper	Proprietary OTN JDeveloper License (freeware)	Yes	Yes	Yes	Yes	generic JVM	Yes
jGRASP	Proprietary (freeware)	Yes	Yes	Yes	Yes		No
KDevelop	GPL	No	No	Yes	No	Solaris	Unknown
MyEclipse	Proprietary	Yes	Yes	Yes	Yes		Yes
NetBeans	CDDL, GPL2	Yes	Yes	Yes	Yes	Solaris	Yes
Rational Application Developer	Proprietary	Yes	Yes	Yes	No	Solaris, AIX	Yes
Servoy	Proprietary	Unknown	Yes	Yes	Yes	Solaris	Yes
Xcode	Proprietary	No	No	No	Yes		Yes


Java - JavaPetnica/src/svc/psc2012/java/block0/Prvi.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Package Explorer
Berlin_v1
CSES
CSES2
DataMining
JavaPetnica
src
svc.psc2012.java
svc.psc2012.java.block0
Prvi.java
JRE System Library [JavaSE-1.6]
p1
PajekNetAgg
SNEIPL
SoftNet
Test
Prvi.java
package svc.psc2012.java.block0;
import java.util.Date;
public class Prvi {
public static void main(String[] args) {
System.out.println("Zdravo");
Date d = new Date();
System.out.println("Trenutno vreme: " + d.toString());
}
}
Problems Javadoc Declaration Console Call Hierarchy Properties
<terminated> Prvi [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Jul 13, 2012 10:58:33 AM)
Zdravo
Trenutno vreme: Fri Jul 13 10:58:33 CEST 2012
Writable
Smart Insert
13 : 5


Java - JavaPetnica/src/svc/psc2012/java/block0/PrviGUI.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Package Explorer
Berlin_v1
CSES
CSES2
DataMining
JavaPetnica
src
svc.psc2012.java
svc.psc2012.java.block0
Prvi.java
PrviGUI.java
JRE System Library [JavaSE-1.6]
p1
PajekNetAgg
SNEIPL
SoftNet
Test
PrviGUI.java
public PrviGUI() {
setTitle("Simple GUI app");
setSize(400, 100);
setDefaultCloseOperation(EXIT_ON_CLOSE);
panel.add(btn);
panel.add(label);
btn.addActionListener(
new ActionListener() {
@Override
public void actionPerformed(ActionEvent arg0) {
Date d = new Date();
label.setText("Trenutno vreme je: " + d.toString());
System.out.println("Kliknuto dugme");
}
}
);
getContentPane().add(panel);
Simple GUI app
Trenutno vreme
Trenutno vreme je: Fri Jul 13 11:28:40 CEST 2012
Problems Javadoc Declaration Console Call Hierarchy Properties
PrviGUI [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Jul 13, 2012 11:27:44 AM)
Kliknuto dugme
Kliknuto dugme
Kliknuto dugme
Writable
Smart Insert
34 : 58

Zadatak 1

- Ispitati koji od dva trougla ima veći obim.
- Dekompozicija problema
  - **Trougao je određen sa tri tačke**
    - Atributi klase trougao – tri tačke
    - Metode klase trougao
      - metod koji računa obim trougla
  - **Tačka je određena x i y koordinatama**
    - Atributi klase tačka – dve koordinate
    - Metode klase tačka
      - metod koji računa rastojanje do neke druge tačke



```
public class Point {

    private double x, y;

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public double distanceTo(Point p) {
        double dx = (x - p.x) * (x - p.x);
        double dy = (y - p.y) * (y - p.y);
        return Math.sqrt(dx + dy);
    }
}
```

this – implicitna referenca objekta na samog sebe

```
public class A {
  private int a, b;

  public A(int a, int b) {
    this.a = a;
    this.b = b;
  }

  public A(int x) {
    this(x, x);  // poziv konstruktora A(int, int)
  }
}
```

```java
public class Triangle {

	private Point a, b, c;

	// agregacija
	public Triangle(Point a, Point b, Point c) {
		this.a = a;
		this.b = b;
		this.c = c;
	}

	// kompozicija
	public Triangle(
		double ax, double ay,
		double bx, double by,
		double cx, double cy
	)
	{
		a = new Point(ax, ay);
		b = new Point(bx, by);
		c = new Point(cx, cy);
	}

	// to be continued
}
```

Agregacija VS kompozicija

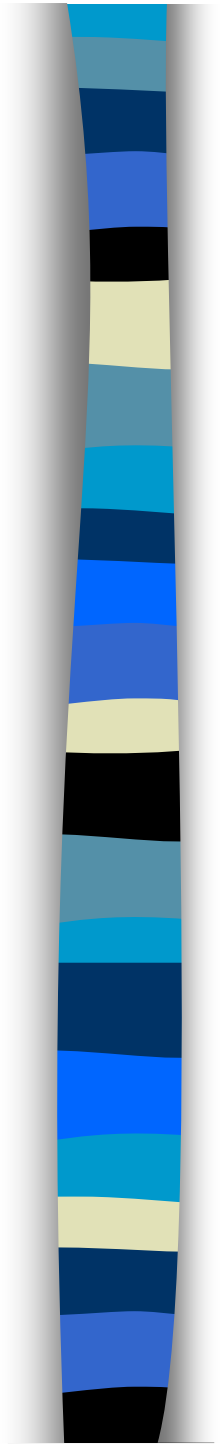
- Klase A i B su u relaciji agregacije ako
  - A ima atribut *x* tipa B
  - A ne instancira atribut x
- Klasa A i B su u relaciji kompozicije ako
  - A ima atribut x tipa B
  - A instancira atribut x

Agregacija VS kompozicija

- **Vrlo je važno prilikom dekompozicije problema pravilno uočiti da li je reč o agregaciji ili kompoziciji dve klase**
- **Klase A i B su u kompoziciji ako je B deo A koji ne može da postoji bez A**
  - Čovek ima srce, mozak, pluća, etc.
- **Klase A i B su u agregaciji ako je B deo A koji postoji nezavisno od A**
  - Čovek može biti član više socijalnih zajednica (biti u studentskoj organizaciji, fudbalskom timu, itd.)

U našem zadatku

- Jedna tačka može biti deo više trouglova
  - Trougao i tačka su u relaciji agregacije
  - Prvi konstruktor je bolji izbor
- U situaciji da imamo niz trouglova dužine n koji sadrži identične trouglove
  - Prvi izbor: imamo tri instance klase tačka
  - Drugi izbor: imamo n * 3 instance klase tačka



```
public class Triangle {

        private Point a, b, c;

        public Triangle(Point a, Point b, Point c) {
                this.a = a;
                this.b = b;
                this.c = c;
        }

        public double volume() {
                return a.distanceTo(b) +
                        a.distanceTo(c) +
                        b.distanceTo(c);
        }
}
```

```
public class Zadatak1 {
	public static void main(String[] args) {
		Triangle t1 = new Triangle(
			new Point(10, 3), new Point(7, 6), new Point(2, 5));

		Triangle t2 = new Triangle(
			new Point(12, 5), new Point(4, 7), new Point(9, 8));

		double t1v = t1.volume();
		double t2v = t2.volume();

		if (t1v > t2v) {
			System.out.println("t1 ima veci obim");
		} else if (t1v < t2v) {
			System.out.println("t2 ima veci obim");
		} else {
			System.out.println("Imaju podjednake obime");
		}
	}
}
```