EUCIP Core

Razvoj i primjena informacijskih sustava

Priprema 4.

Filip Debelić, Marijan Matić, Maria Žilić

Java je objektno – orijentirani jezik

U objektno orijentiranom programiranju osnovnu ulogu imaju objekti koji sadrže i podatke i funkcije (= metode). Program se konstruira kao skup objekata koji međusobno komuniciraju. Podaci koje objekt sadrži predstavljaju njegovo stanje, dok pomoću metoda on to stanje može mijenjati i komunicirati s drugim objektima.

Osnovne karakteristike objektno-orijentiranog programiranja:

- **Sve je objekt.** Objekte možemo gledati kao "varijable" koje sadrže podatke, ali mogu i izvršavati određene operacije kroz metode koje pripadaju tom objektu.
- **Program je skup objekata koji komuniciraju jedni s drugima**, na način da se poziva metoda koja pripada danom objektu.
- **Objekt ima svoju memoriju koja je opet sastavljena od objekata.**
- **Svaki objekt ima tip.** Svaki objekt je *instanca* neke klase. Sav izvorni kod Jave je podijeljen u klase!
- **Svi objekti istog tipa mogu primati iste poruke** (instance iste klase imaju iste metode)**.**

Objekt:

- ima podatke
- ima metode
- ima jedinstvenu adresu u memoriji
- instanciran je od neke konkretne klase (svaka klasa definira novi tip podatka).

Detalji Java koda:

- Reprezentacija klase u memoriji naziva se objekt. Općenito program može imati više objekata koji reprezentiraju istu klasu.

- Sav izvorni kod podijeljen u klase koje se nalaze u istoj ili različitim izvornim datotekama (s ekstenzijom .java).
- Svaki Java program počinje svoje izvršavanje pozivom metode main (to je analogno odgovarajućoj main funkciji u C-u ili C++-u), stoga jedna od klasa u programu (i samo jedna) mora i mati `main` metodu. Ta metoda mora biti deklarirana `public` i `static`. Sintaksa:

```
public class  ImePrograma{

    public static void main(String args[]) {

          …
    }

}
```

- Svaka linija koda završava znakom točka zarez (;) ili vitičastim zagradama koje označavaju početak i kraj bloka naredbi.
- Naziv klase pišemo početnim velikim slovom, naziv metode i varijabli pišemo početnim malim slovom, a svaka iduća riječ započinje velikim slovom.
- Deklaracija atributa u Javi: vidljivost tip imeAtributa
- Deklaracija metode u Javi:
  vidljivost tipPovratneVrijednosti imeMetode (ListaParametara){}
  Lista parametara može biti prazna ili oblika:
  tipParametra1 imeParametra1, tipParametra2 imeParametra2,…
  Ukoliko metoda ne vraća ništa tip povratne vrijednosti je `void`.
- Pozivanje metode neke klase odvija se pomoću dot-operatora.
  U primjeru ispod vidimo da bismo pozvali metodu neke klase, na primjer `ucini_nesto` u klasi `KorisnikKnjige`, ili metodu `napisi` klase `Knjiga`, moramo prvo instancirati objekt te klase i zatim metodu pozvati na objektno-orijentiranoj varijabli pomoću dot-operatora: `kk.ucini_nesto(book);`
  (u objektu `kk` pozvali smo metodu `ucini_nesto` i dali joj jedan parametar `book` tipa `Knjiga`)

```
class Knjiga {

   ...

   void  napisi(int boja) {
       // implementacija
   }

   ...

}
```

```
class  KorisnikKnjige{

     public static void main(String args[]) {


          Knjiga book = new Knjiga();

          KorisnikKnjige kk = new KorisnikKnjige();
          kk.ucini_nesto(book);
     }

     void ucini_nesto(Knjiga x) {
          x.napisi();
     }
}
```

- Naredba `if-than` je najosnovnija naredba za kontrolu toka. Ona kaže programu da izvrši određeni dio koda samo u slučaju da zadani logički izraz rezultira sa `true`. Uvjet `if` naredbe može biti bilo koji izraz koji rezultira sa `boolean` vrijednosti. Sintaksa glasi:

  ```
  if (uvjet) {blok naredbi ukoliko je uvjet true}
  ```

- Naredba `if-than-else` dodaje i blok naredbi koji se izvršava ukoliko je rezultat uvjeta `false`. Izvršava se samo jedan od blokova naredbi, ovisno o istinitosti uvjeta. Sintaksa glasi:

  ```
  if (uvjet)
       {blok naredbi ukoliko je uvjet true}
  else
       {blok naredbi ukoliko je uvjet false}
  ```

- Ukoliko želimo provjeriti više logičkih izraza koristimo tzv. if-elseif-else oblik, uz napomenu da else pripada najbližem prethodnom if-u. Sintaksa glasi:

```
if (uvjet1)
     {blok naredbi ukoliko je uvjet1 true}
else if (uvjet2)
     {blok naredbi ukoliko je uvjet2 true}
     else
          {blok naredbi ukoliko je uvjet2 false}
```

Primjer

```
public class Osoba {

    private String ime;
    private String prezime;

    public String getIme() {
        return this.ime;
    }

    public String getPrezime() {
        return this.prezime;
    }

    public Osoba(final String ime, final String prezime) {
        this.ime = ime;
        this.prezime = prezime;
    }

    void setIme(final String ime) {
        this.ime = ime;
    }

    void setPrezime(final String prezime) {
        this.prezime = prezime;
    }

    public String toString() {
        return getPrezime() + ", " + getIme();
    }

}
```

Komentari:

- Klasa `Osoba` ima dva atributa tipa `String` i šest metoda, od kojih niti jedna nije `main` metoda pa stoga klasa `Osoba` ne predstavlja potpuni program.
- Java pri identifikaciji metode koristi ne samo njeno ime, već i broj i tip parametara; stoga je moguće imati dvije metode istog imena, ali različitih povratnih vrijednosti.
- Obzirom da su atributi vidljivosti `private`, moramo postaviti `get` i `set` metode kako bi mogli dohvaćati i postavljati vrijednosti za

navedenu osobu. Set metoda (za postavljanje vrijednosti varijable) mora dobiti argument (od get metode) kako bi znali na koju vrijednost moraju postaviti varijablu.

- Ključnom riječi `this` referiramo se na atribute objekta unutar klase.
- Da bismo kreirali objekt tipa `Osoba` moramo konstruirati novu klasu koja će imati main metodu u kojoj ćemo takav objekt instancirati. Kreiranje objekata dane klase zovemo instanciranjem objekata, a same objekte zovemo instancama dane klase. Za instanciranje koristimo operator `new`.
- `Static` modifikator atribute i metode pretvara u zajedničke među svim objektima instanciranim iz te klase.
- Modifikator `final` pretvara varijablu ispred koje se nalazi u konstantu.

```
public class Grupa {

	private static Student osobaStudent;
	private static Osoba osobaVulgaris;

	private static Student getOsobaStudent(){
		return osobaStudent;
	}

	private static Osoba getOsobaVulgaris(){
		return osobaVulgaris;
	}

	public static void main(final String[] args){

		osobaStudent = new Student("Pero", "Peric");
		osobaVulgaris = new Osoba("Marko", "Maric");
		System.out.println(osobaStudent);
		System.out.println(osobaVulgaris);
	}

}
```

Komentari:

- Klasa Grupa ima deklarirane tri metode, od kojih je jedna `main`. Kada kompajliramo program koji sadrži klasu `Grupe` i pokrenemo ga, počet će se izvršavati njegova `main` metoda.
- U `main` metodi smo instancirali novi objekt tipa `Student`. Varijabla `osobaSudent` (tipa `Student`) je referenca na taj objekt.
- `Student` je svakako osoba, te je smisleno modelirati tu klasu tako da naslijedi metode iz `Osobe`, te da definira neke nove.

```
public class Student extends Osoba {

    public String toString() {
        return "Student: " + super.toString();
    }

    public Student(final String ime, final String prezime) {
        super(ime, prezime);
    }

}
```

Komentari:

- Ključna riječ `extends` signalizira da se klasa `Student` konstruira proširenjem klase `Osoba`. U osnovi to znači da klasa `Student` nasljeđuje sve podatke i metode iz klase `Osoba`, te deklarira samo nove varijable i metode. U toj situaciji govorimo da klasa `Student` proširuje klasu `Osoba` i da je `Osoba` nadklasa `Studenta` (`Student` je podklasa `Osobe`).
- Podklasa može ponovo deklarirati metodu deklariranu u nadklasi. U tom slučaju kažemo da je metoda prerađena, tj. podklasa daje novu implementaciju metode iz nadklase. To radimo pomoću reference `super`.

Testiranje izvornog koda, programiranje

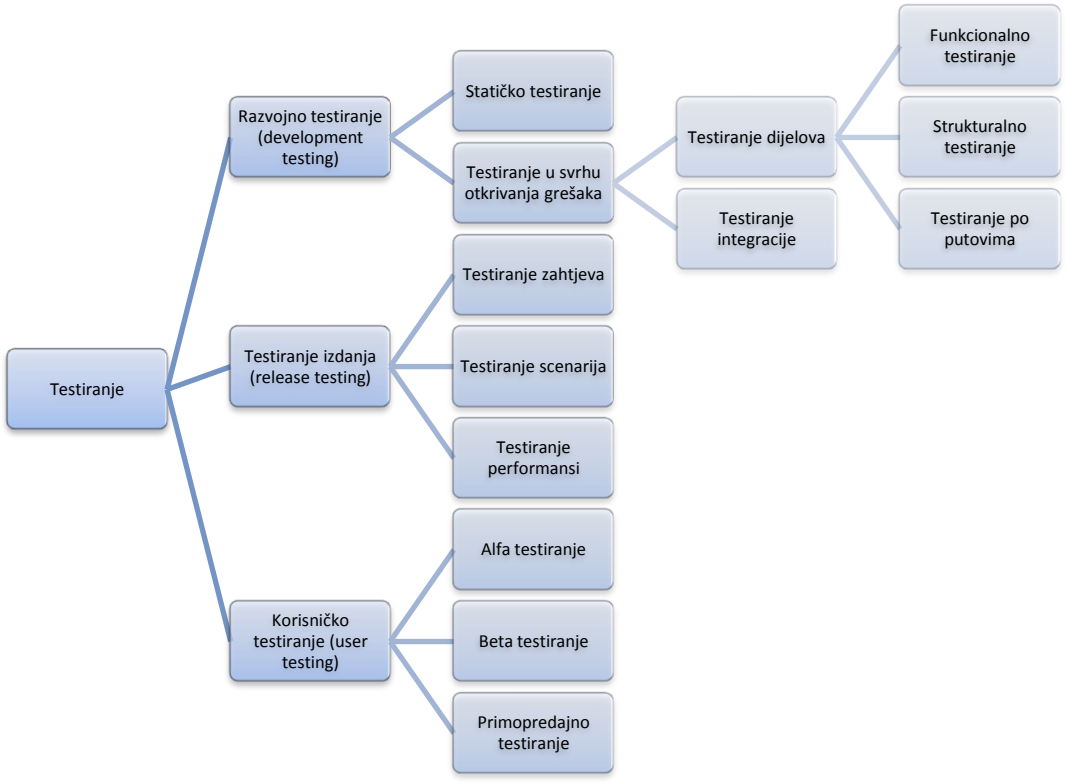
Testiranje je pokusno izvođenje softvera ili njegovih dijelova na umjetno pripravljenim podacima (analiza aplikacije) radi osiguravanja ispunjenja svih uvjeta postavljenih prilikom dizajniranja i specifikacije aplikacije. Testiranje aplikacije je dio razvojnog ciklusa aplikacije, a pisanje i održavanje testova može pomoći da se osigura ispravnost i kvaliteta aplikacije.

Svrha testiranja može biti

- **Verifikacija** (provjerava da li softver odgovara specifikaciji, dakle dokumentu o zahtjevima)
- **Validacija** (provjerava da li softver zadovoljava stvarnim potrebama korisnika)

Vrste i faze testiranja

Postoje razne vrste testiranja. Podjela napravljena je po kriteriju tko odnosno kada sudjeluje u testiranju prikazana je na donjoj slici



Razvojno testiranje radi se tijekom razvoja softvera od strane razvojnog tima i dijelimo ga na

- statičko testiranje koje se odvija bez stvarnog izvođenja softvera, a odnosi se na analizu i provjeru dokumenata, modela sustava, dizajna, odnosno programskog koda.
- testiranje u svrhu otkrivanja grešaka koje se svodi se na pokusno izvođenje softvera ili njegovih dijelova, na umjetno pripravljenim podacima, uz pažljivo analiziranje rezultata.

Nakon razvojnog testiranja poseban tim za testiranje unutar softverske kuće vrši **testiranje izdanja** te na kraju sami korisnici tijekom isporuke softvera bave se **korisničkim testiranjem** .

Testiranje dijelova softvera

Razlikujemo nekoliko metoda testiranja softvera koji se razlikuju po načinu odabira test podataka.

- **Funkcionalno testiranje** (black-box testing) – uspoređuje se ponašanje testiranog programa i zahtjeva specifikacije bez znanja o implementacijskim detaljima odnosno softver se promatra kao „crna kutija" o kojoj jedino znamo da bi ona, prema specifikaciji, za zadane ulaze trebala proizvesti zadane izlaze.
- **Strukturalno testiranje** (white-box testing) – uspoređuje ponašanje testiranog izvornog koda i predviđenog ponašanja izvornog koda uzimajući u obzir moguće greške u strukturi i logici izvornog koda. Na osnovu poznavanja interne strukture testiranog dijela softvera i korištenih algoritama odabiremo dodatne test primjere.
- **Testiranje po putovima** (path testing) – posebna vrsta strukturalnog testiranja, izvodi se na način da test primjere biramo sa idejom da se isproba svaki „nezavisni put" kroz dijagram toka kontrole testiranog dijela softvera.

Razine testiranja

Testovi su često grupirani prema nivou detaljnosti odnosno cilju samog testiranja pa prema tome razlikujemo:

- **Unit testiranje**
  Zadaća unit testiranja je izoliranje najmanjeg dijela izvornog koda kojeg možemo testirati (npr. metoda, objekt, modul, sučelje i sl.) i u tako izoliranom okružju provjera da li se testirani izvorni kod ponaša na predviđeni način. Provjerom funkcionalnosti izoliranih, malih cjelina smanjujemo poteškoće otkrivanja grešaka u izvornom kodu i samo vrijeme potrebno za otkrivanje istih, također unit testiranje dozvoljava automatizaciju testnog procesa.
  Unit testovi se kreiraju u procesu razvoja aplikacije, tj. u trenutku izrade izvornog koda (programiranja). Svaki unit test bi trebao biti neovisan od drugih testova.
  Za testiranje Java programa koristimo JUnit framework koji u sebi sadrži set metoda pomoću kojih testiramo izvorni kod napisan u Javi. JUnit je jednostavan alat za pisanje i izvršavanje automatiziranih testova, te pokriva Statement coverage testove.

- **Testiranje integracije**
  Cilj testiranja je otkriti greške koje nastaju zato što dijelovi na pogrešan način koriste međusobna sučelja. Provodi se nakon što se manji dijelovi softvera testiraju i zatim udruže u veću cjelinu.

- **Testiranje sustava kao cjeline**
  Testiramo cijeli sustav kako bi ustanovili da li zadovoljava zahtjeve te se ispituju nefunkcionalne karakteristike sustava (pouzdanost, uporabljivost…). Pogreške su moguće na mjestu sučeljavanja integriranih komponenti ili unutar funkcioniranja novonastalog

proizvoda. Testiranje se obavlja u okruženju u kojem je proizvod nastao.

- **Krajnji test**
  Cilj je potvrditi da softverski proizvod radi kvalitetno i da udovoljava sve zahtjeve korisnika. Testiranje se obavlja u realnim uvjetima u kojima će proizvod i funkcionirati, sa stvarnim podacima. Krajnji test treba potvrditi da softverski proizvod radi kvalitetno i da udovoljava sve zahtjeve korisnika
  Zadnja faza testiranja sustava, najčešće se odvija kao black-box test.

Pokrivenost koda (code coverage)

Pokrivenost koda (code coverage) je tehnika strukturnog testiranja izvornog koda (White-box testiranje) koja se koristi u testiranju softvera. Opisuje stupanj do kojeg je izvorni kod programa testiran. Testovi pokrivenosti koda nam prikazuju snagu i limite provedenih testova i pomažu u eliminaciji nedostataka.

Kriteriji kojima opisujemo koliko je izvorni kod testiran su razni, a osnovni kriteriji testiranja pokrivenosti koda su slijedeći:

- **Function coverage**: mjera koja određuje koliko funkcija izvornog koda je pozvano tijekom testiranja.
- **Statement coverage**: mjera koja određuje koliko izvršnih naredbi programa je izvršeno tijekom testiranja. Deklarativne naredbe koje generiraju izvršni kod se smatraju izvršnim naredbama, dok se naredbe grananja (npr. if, for, switch) pokrivaju ukoliko je pokrivena naredba koja kontrolira tok. Implicitne naredbe (npr. return) se ne pokrivaju ovim testom.
- **Decision coverage**: mjera koja određuje koliko je uvjetnog grananja izvršeno tijekom testiranja i da li je svaki logički izraz

korišten za grananje provjeren na točno (true) i netočno (false) stanje. Ova mjera pokriva sve if, while (for) i switch naredbe kao i naredbe za rukovanje iznimkama (npr. try).

- **Condition coverage**: mjera koja određuje da li je svaki uvjetni izraz provjeren na točno (true) i netočno (false) stanje. Uvjetni izraz je vrijednost logičkog izraza koji ne sadrži logičke operatore (npr. ?: je uvjetni izraz). Uvjeti se mjere neovisno jedan o drugome.
- **Multiple Condition Coverage**: mjera koja određuje da li su sve moguće kombinacije uvjeta testirane u logičkim izrazima koji sadrže više logičkih operatora.
- **Path coverage**: mjera koja određuje koliko različitih putanja u funkciji je izvršeno tijekom testiranja. Putanja je jedinstvena sekvenca grananja od ulaska u funkciju do izlaska iz funkcije.

Kombinacija Condition i Decision coverage testova ja najpogodnija za testiranje C, C++ i Java izvornog koda.

Zadatak

Koristeći 3. vježbe i ove pripreme, dolje prikazan UML dijagram klasa „prevedite" u Java kod na način da deklarirate sve atribute, metode i veze između klasa (implementacija metoda nije potrebna).