NAPOMENA: Vežbe podrazumevaju da je student ovladao teorijom iz dela Uvod u objektno orijentisano programiranje: "Objektno orijentisano programiranje" (1_uvod.ppt, 2_uvod.ppt i 3_uvod.ppt)

Nasleđivanje

Vrlo često novi programi nastaju proširivanjem prethodnih. Najbolji način za stvaranje novog softvera je imitacija, doterivanje i proširivanje postojećeg. Tradicionalni metodi razvoja softvera su zanemarivali ovakvo stanovište. U objektno orijentisanom programiranju to je osnovni način rada.

Ako se uzme slučaj gotove aplikacije, postavlja se pitanje kako iskoristiti već napisan izvorni kod. Jedan od načina je kopiranje i menjanje unutar kopije ono što je potrebno. Kod tradicionalnih jezika to je bila jedina mogućnost. Problem je kako da sve ostane dobro organizovano i kako da se do kraja razume originalni kod.

Relacija nasleđivanja omogućuje proširenje ponašanja postojeće klase. Npr. klasa B nasleđuje klasu A. Izvedena klasa B (podklasa) je jedna specijalna vrsta osnovne klase A (nadklasa) i klasa B nasleđuje sve atribute i sve metode od klase A.

Nova izvedena klasa B može da:

- proširi strukturu podataka osnovne klase A dodavanjem novih atributa
- proširi funkcionalnost osnovne klase A dodavanjem novih metoda
- izmeni funkcionalnost osnovne klase A redefinisanjem postojećih metoda

Java podržava samo jednostruko nasledjivanje – jedna klasa može naslediti samo jednu klasu dok više klasa može naslediti istu klasu. Nasleđivanje se u Javi naznačava ključnom rečju **extends**. Ključna reč ***super*** označava roditeljsku klasu. Ona se može koristiti i u metodama i u konstruktorima

Primer 1. Izmodelovati klasu Piksel koja nasleđuje klasu Tačka u 2D prostoru. Upotreba rezervisane reči ***super***.

```java
public class Tacka {
    public double x;
    public double y;

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

class Piksel extends Tacka {
    public String boja;

    public Piksel(double x, double y, String boja) {
        super(x, y);
        this.boja = boja;
    }

}
```

Izvedena klasa (podklasa) od svoje osnovne klase (nadklasa) nasleđuje sve atribute i metode koje nisu označene *private* modifikatorom vidljivosti. Ipak, funkcionalnost nasleđenih metoda se može izmeniti redefinisanjem (preklapanjem) datih metoda (engl. *method overriding*). To znači da u klasi naslednici postoji metoda istog imena i parametara kao i u baznoj klasi. Anotacija *@Override* se dodaje iznad metode kako bi se kompajleru naznačila namera preklapanja metode i time izbegle moguće greške.

Primer 2. Dopuniti klase iz prethodnog primera (primer 1) preklopljenim metodama *equals()* i *toString()*. Takođe, definisati funkciju *deepCopy()* koja će vršiti duboko kopiranje određene klase.

```
@Override
public String toString() {
    return "x:" + x + ", y:" + y;
}

@Override
public boolean equals(Object obj) {
    boolean isti = false;
    if (obj == null)
        return false;
    if (this == obj)
        return true;
    if (obj instanceof Tacka) {
        Tacka objTacka = (Tacka) obj;
        if (this.x == objTacka.x && this.y == objTacka.y) {
            isti = true;
        }
    }
    return isti;
}

public Tacka deepCopy(){
    return new Tacka(this.x, this.y);
}
```

Osnovna klasa koja nema nijedan konkretan (realan) objekat, već samo predstavlja generalizaciju izvedenih klasa, naziva se apstraktnom klasom. Apstraktna klasa može da sadrži apstraktne funkcije, koja su u ovoj klasi samo deklarisane, a nije i implementirane. Apstrakne klase ne mogu imati svoje objekte, već samo njene klase naslednice mogu da imaju objekte.

Primer 3. Izmodelovati apstraktnu klasu Figura koja sadrži apstraktne metode *izracunajPovrsinu()* i *izracunajObim()*. Takođe izmodelovati klasu Krug koja je izvedena od klase Figura i u kojoj se definišu apstraktne metode.

```
public abstract class Figura {

    double obim;
    double povrsina;

    public abstract double izracunajPovrsinu();
    public abstract double izracunajObim();
}
```

Interfejsi omogućuju povezivanje klase sa drugim programima. Ti drugi (spoljašni) programi ne moraju da znaju sadržaj klase već samo kako da pozovu traženu funkcionalnost. Interfejs je nalik apstraktnoj klasi ali nije klasa! On je samo obavezujući spisak metoda koje klasa koja implementira interfejs mora da podrži.

Polimorfizam opisuje situaciju kada se poziva metoda nekog objekta, a ne zna se unapred kakava je funkcionalnost te metode, niti koji je to konkretan nasleđeni objekat, zna se samo koja mu je bazna klasa.

Primer 4. Izmodelovati klasu Osoba i Direktor i RadnikUPogonu. Klase Direktor i RadnikUPogonu nasleđuju klasu Osoba. Za sve zaposlene osobe treba omogućiti računanje plate i poreza tj. izmodelovati interfejs ObracunPrihoda. Klase Direktor i RadnikUPogonu implementiraju interfejs ObracunPrihoda. Omogućiti odgovarajuću funkcionalnost računanja plata i poreza za radnike i direktore. Direktor poseduje dodatnu funkcionalnost da otpusti radnika.

```java
public interface ObracunPrihoda {

    public double obracunajPlatu();
    public double obracunajPorez();

}


class Direktor extends Osoba implements ObracunPrihoda {

    double osnovica;
    double koeficient;
    double bonus;

    // dodati ostatak
}

class RadnikUPogonu extends Osoba implements ObracunPrihoda {

    int godineStazaRadnika;
    double osnovica;
    double koeficient;

    // dodati ostatak
}
```

Plata za direktora se računa po formuli: P=koeficient * osnovica + bonus a za radnika P = (koeficient + 0.01*godineStazaRadnika) * osnovica. Porez na platu se računa tako što se iznos plate pomnoži sa visinom pdv-a (~20%). Za slučaj direktora, ukoliko je plata veća od 100 000 onda se porez povećava duplo.

Veze između klasa

Kada je potrebno rešiti neki problem koji sadrži više entiteta, dosta je lakše nacrtati odnos tih klasa pa tek onda pristupiti pisanju programskog koda. Jedan od načina je pravljenje dijagrama klasa koji jasno vizuelizuje odnos definisanih klasa. Veze između klasa su te koje definišu njihov međusobni odnos. Postoji više tipova veza a četiri najznačajnije su opisane u nastavku:

- **Asocijacija** – Klasa Student **koristi** klasu Bicikl (Student je povezan sa Biciklom)
- **Agregacija** – Klasa Student **ima** kolekciju klase Knjiga (Student poseduje Knjige koja će postojati i kad Studenta ne bude)
- **Kompozicija (jaka agregacija)** – Klasa Fakultet **sadrži** kolekciju klase Student (Student je učenik Fakulteta i ne može ostati student fakulteta koji npr. više ne postoji, prestane sa radom)
- **Nasleđivanje** – Klasa Student **je** klasa Osoba (Student je pored što studira i Osoba)

Zadaci

Zadatak 1. Napisati program koji omogućuje rad banke sa računima. Izmodelovati klasu *Racun*, *TekuciRacun*, *RacunStednje* i *Osoba*. Obezbediti da su svi atributi privatni (za klase naslednice), da postoje više konstuktora (bez parametara, parametri koji su atributi klase, referenca na objekat), korisničke metode i set/get metode za atibute klase, preklopiti metode *toString()* i *equals()*. Napraviti test klasu i posmatrati slučaj kad jedna Osoba može imati samo po jedan tekući račun i račun za štednju.

- Osoba je opisana sledećim podacima: JMBG, ime, prezime.
- Klasa Racun tereba da je apstraktna i opisana sledećim podacima: vlasnikom racuna (referenca ka Osobi), stanjem racuna (tipa *double*). Takođe sadrži deklaracije sledećih apstraktnih metoda:
  - *boolean podigniPare(double suma)*
  - *void uplatiPare(double suma)*

  Metoda *toString()* treba da omogući ispis trenutnog stanja sa podacima o vlasniku racuna.
- TekuciRacun nasleđuje klasu Racun i implementira apstraktne metode. TekuciRacun je opisan sledećim podacima: mesecna naknada (tipa double). TekuciRacun poseduje metodu *obracunajMesecnuNaknadu()* koja umanjuje stanje računa za mesečnu nakandu. Treba paziti da se sa računa ne može podići više novca nego što ima na stanju. Pri svakoj uplati novca banka uzima bankarsku proviziju tj. uplatu umanjuje za bankarsku proviziju od 0.01%.

- RacunStednje nasleđuje klasu Racun i implementira apstraktne metode. RacunStednje je opisan sledećim podacima: *pokrenutaStednja (tipa boolean)* i *godisnjiKoeficientStednje(tipa double)*. RacunStednje poseduje metodu *pokreniStednju()* kojom se menja stanje računa (zaključava se račun za podizanje novca) i metodu *void obustaviStednju(int meseci)* kojom se prekida štednja i izračunava novo stanje računa po formuli: stanjeRacuna + stanjeRacuna * (godisnjiKoeficient * 0.01 * brojGodina) + stanjeRacuna * (godisnjiKoeficient * 0.01 * ostatakMeseci/12). Treba praziti da se sa računa ne može podići novac ukoliko je pokrenuta štednja i ukoliko se podiže više novca nego što ima na stanju. Pri uplati se ne obračunava naknada banke.

Zadatak 2.Napisati program i klase koje modeluju rad prodavnice računara. Za sve klase potrebno je obezbediti da su svi atributi privatni, da postoje više konstuktora (bez parametara, parametri koji su atributi klase, referenca na objekat), korisničke metode i set/get metode za atibute klase, preklopiti metode *toString()* i *equals()*. Potrebno je testirati klase. Iskoristiti klase od prošlih vežbi.

- Kreirati apstraktnu klasu koja predstavlja artikl prodaje. Artikl prodaje je opisan sledećim podacima: Šifra (jedinstveno), Naziv, Cena, Raspoloživa količina, Opis.
- Definisati klasu koja predstavlja komponentu računara. Komponenta je artikl koji se prodaje i opsana je sledećim podacima: kategorija (referenca ka Kategoriji komponente).
- Definisati klasu Kategorija koja označava kategoriju komponente. Kategorija je opisana sledećim podacima: naziv kategorije, nadkategorija (referenca ka Kategoriji komponente) i podkategorija (referenca ka Kategoriji komponente). Uzeti slučaj kada nema više od jedne podkategorije (npr. K1< -- >K2< -- > K3 < -- > K4). Prilikom ispisa *toString()* metode se prikazuju samo nadkategorije.
- Definisati klasu koja predstavlja gotove konfiguracije računara. Gotova konfiguracija je Artikl koji se prodaje i opsana je sledećim podacima: komponente (lista referenci ka Komponentama).
- Kreirati klasu koja predstavlja memoriju računara. Memorija je jedna od specijalizacija komponenti računara i opsan je sledećim podacima:kapacitet.
- Kreirati klasu koja predstavlja procesor računara. Procesor je jedna od specijalizacija komponenti računara i opsan je sledećim podacima: radni takt i broj jezgara.
- Definisati klasu koja predstavlja stavke računa. Stavka računa predstavlja jedan od delova računa i opsana je sledećim podacima: redni

broj stavke na računu(jedinstveno), artikal koji se prodaje (referenca ka artiklu prodaje), jedinična cena na dan kupovine i količina.