Nasleđivanje

Eng. *Inheritance* predstavlja jedan od osnovnih koncepata Objektno-orijentisane paradigme. Ono omogućava preuzimanje i proširenje ponašanja postojeće klase. Kada klasa A nasledi klasu B, ona nasleđuje sve njene metode i atribute sa mogućnošću uvođenja svojih. Nasleđivanje ne samo da smanjuje količinu programskog koda koji je potrebno napisati, već nam omogućava prirodnije modelovanje sistema koji programiramo.

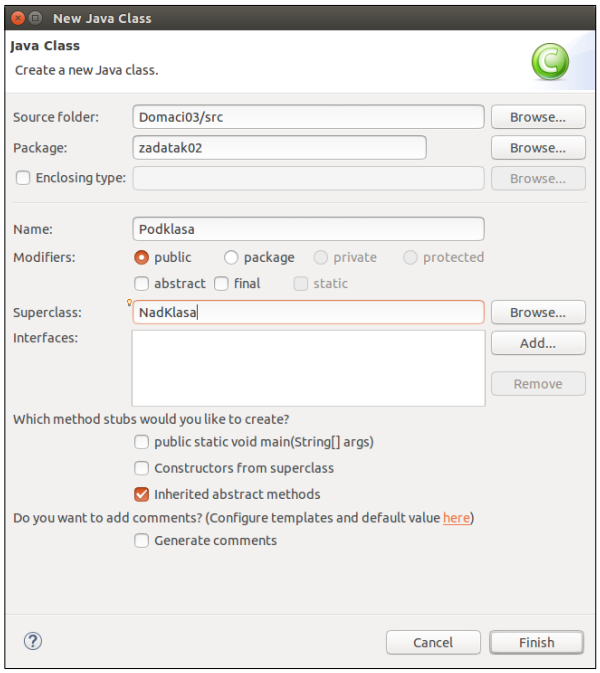
Terminologija:

Klasa koja nasleđuje – izvedena klasa, podklasa (eng. *subclass*, *child class*)

Klasa koja je nasleđena – nadklasa, roditeljska klasa, superklasa (eng. *super class*, *parent class*)

Nasleđivanje u Javi se definiše u zaglavlju (nakon imena) klase ključnom rečju **`extends`** nakon koje sledi ime klase koja se nasleđuje. Ovo je moguće dodati ručno, međutim, u Eclipse-u je moguće navesti ime klase koja se nasleđuje direktno u prozoru za kreiranje nove klase, nakon čega će potreban kod biti automatski izgenerisan.

U Java programskom jeziku, klasa može da nasleđuje najviše jednu klasu.



U Java programskom jeziku, sve klase implicitno nasleđuju klasu *Object*.

U nasljeđenoj klasi, kljčna reč **`super`** se odnosi na nadklasu i služi za pristup njenim članovima (atributima i metodama).

Modifikatori pristupa

Modifikatori pristupa su Java ključne reči koje definišu obseg vidljivosti klase, atributa ili metode. Klase mogu da imaju samo dva modifikatora pristupa:

`public` klasa je dostupna svim klasama u programu (Eclipse projekat)

`default` klasa je dostupna samo klasama u istom paketu (default modifikator se primenjuje kada ni jedan drugi nije naveden). Tzv. *package friendly* modifikator.

Metode i atributi (članovi klase) mogu da imaju jedan od sledeća 4 modifikatora pristupa.

`default`	Član klase je vidljiv u svojoj klasi kao i u svim klasama iz istog paketa.
`private`	Član klase je vidljiv samo u svojoj klasi.
`public`	Član klase je vidljiv u svim klasama iz istog programa.
`protected`	Član klase je vidljiv samo u svojoj klasi i klasama koje je nasleđuju.

Sledeći princip sakrivanja podataka, preporuka je da se atributi klase deklarišu kao **`private`** (**`protected`**, ukoliko će klasa imati naslednice), a da im se mogućnosti pristupa definišu pomoću javnih **`get`** i **`set`** metoda.

```
Student studentA = new Student();
// Ne moze vise ovako
studentA.brojIndeksa = "RS 01/2016";
// Vec ovako
studentA.setBrojIndeksa("RS 01/2016");


// Upotreba get metode
System.out.println("Ime studenta: " + studentA.getIme());
```

Napomena Upotrebom *Eclipse* alata, moguće je izgenerisati **`get`** i **`set`** metode za željene atribute klase. Da bi smo ovo postigli, potrebno je iz menija *Source* odabrati stavku *Generate Getters and Setters...* U dijalogu koji se potom otvori potrebno je odabrati atribute za koje želimo da generišemo metode i potvrditi akciju klikom na dugme *OK*.

Primer klase sa primenjenim modifikatorima pristupa, konstruktorima i get i set metodama (tzv. *POJO* klasa, Plain Old Java Object):

```
public class Automobil {

	private String marka;
	private String model;
	private int snaga;

	public Automobil() {
		this.marka = "";
		this.model = "";
		this.snaga = 0;
	}

	public Automobil(String marka, String model, int snaga) {
		super();
		this.marka = marka;
		this.model = model;
		this.snaga = snaga;
	}

	public String getMarka() {
		return marka;
	}

	public void setMarka(String marka) {
		this.marka = marka;
	}

	public String getModel() {
		return model;
	}

	public void setModel(String model) {
		this.model = model;
	}

	public int getSnaga() {
		return snaga;
	}

	public void setSnaga(int snaga) {
		this.snaga = snaga;
	}

	@Override
	public String toString() {
		return "Automobil " + this.marka + " " + this.model + ", " +
		this.snaga + "ks";
	}
}
```

Anotacija `@Override` iznad metode označava da je ta metoda preuzeta iz nadklase ili interfejsa, i redefinisana (implementirana) u izvedenoj klasi. Ova pojava se naziva redefinisanje (preklapanje) metoda (eng. method overriding).

Apstraktne klase i interfejsi

Osnovna klasa koja nema nijedan konkretan (realan) objekat, već samo predstavlja generalizaciju izvedenih klasa, naziva se apstraktnom klasom. Apstraktna klasa sadrži bar jednu apstraktnu funkciju, koja je u ovoj klasi samo deklarisana, a nije implementirana.

U Javi se apstraktne klase (ili metode) deklarišu navođenjem ključne reči **`abstract`** ispred imena, ili selektovanjem odgovarajuće *check box* komponente na dijalogu za kreiranje nove klase (slika iznad).

U primeru informacionog sistema fakulteta, klase Osoba i Zaposleni služe samo sa abstrahuju zajedničke osobine svojih izvedenih klasa. Ne postoji realna situacija u kojoj imamo potrebu za instanciranjem ovih klasa, tako da one predstavljaju savršene kandidate za apstraktne klase u ovom sistemu.

Primer apstraktne klase:

```
public abstract class Figura {

	protected String naziv;

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

}
```

Kao što se može videti, apstraktne metode nemaju implementaciju (telo funkcije). Klase koje naslede apstraktnu klasu moraju da implementiraju sve njene apstraktne metode, inače će Java kompajler javiti grešku (isto tako, grešku dobijamo ukoliko u klasi napišemo apstraktnu metodu, a samu klasu nismo proglasili za apstraktnu).

Interfejsi su poseban koncept u Javi: nisu u pitanju klase, ali interfejsi mogu da sadrže deklaracije apstraktnih metoda, konstanti i statičkih atributa.

Primer interfejsa *Instrument*:

```
public interface Instrument {
	void sviraj();
	void nastimaj();
}
```

Vidimo da interfejsi podsećaju na apstraktne klase. Veza između klasa i interfejsa je sledeća: kaže se da klasa implementira (a ne nasleđuje) interfejs. Klasa može da implementira više interfejsa istovremeno, što nije slučaj sa nasleđivanjem. Nema prepreke da klasa koja nasleđuje drugu klasu implementira i neke interfejse. Jedan interfejs može da nasledi drugi interfejs.

Primer klase Klarinet koja implementira interfejs *Instrument*:

```
public class Klarinet implements Instrument {
	@Override
	public void sviraj() {
		System.out.println("Tiruriruru!");
	}

	@Override
	public void nastimaj() {
		System.out.println("Klarinet je nastiman.");
	}
}
```

Neka pravila vezana za interfejse:

- Svaka klasa koja implementira interfejs mora da implementira sve njegove metode (ili da bude deklarisana kao apstraktna),
- Metode interfejsa su uvek implicitno apstraktne i javne,
- Metode intefejsa ne mogu biti statičke,

Primer

U informacionom sistemu jednog fakulteta potrebno je voditi evidenciju o studentima, profesorima i asistentima. Neke zajedničke osobine koje proizilaze iz toga što su svi oni osobe su: ime, prezime, JMBG, adresa i broj telefona. Čini se bezpotrebnim navoditi ove zajedničke atribute u svakoj klasi ponaosob. Dalje, ono što je zajedničko za profesore i asistente je što imaju platu koju dobijaju od fakulteta. Ovo je posledica toga što su zaposleni na toj ustanovi. Studenti, sa druge strane, pored činjenice da su osobe, kao studenti imaju svoj broj indeksa i ocene.

Upotrebnom nasleđivanja, mogli bi sve ove zajedničke osobine izdvojiti u nadklase koje ćemo naslediti specifičnim klasama.

Zadatak

Napisati Java progam koji modeluje informacioni sistem predstavljenog fakulteta uz pomoć sledećih klasa:

Osoba: *ime*, *prezime*, *JMBG*, *adresa*, *broj telefona*

Zaposleni (nasleđuje klasu *Osoba*): *plata*

Profesor (nasleđuje klasu *Zaposleni*): *broj predavanja*

Asistent (nasleđuje klasu *Zaposleni*): *broj vežbi, mentor (Profesor)*

Student (nasleđuje klasu *Osoba*): *broj indeksa*, *ocene*

Za sve klase obezbediti zaštićenost podataka, napisati konstruktore (bez parametara, sa parametrima), **`get`** i **`set`** metode i preklopiti metodu **`toString`**.

Polimorfizam

Polimorfizam usled nasleđivanja predstavlja pojavu da se objekti svih izvedenih klasa smatraju i objektima svoje nadklase. U primeru informacionog systema fakulteta, ovo nam omogućava da vodimo evidenciju o zaposlenima u jednoj jedinstvenoj kolekciji, bez obzira da li oni profesori ili asistenti:

```
Profesor profesor1 = new Profesor(...);
Asistent asistent1 = new Asistent(...);
Asistent asistent2 = new Asistent(...);

Zaposleni[] zaposleni = {profesor1, asistent1, asistent2};
```

Kao što se vidi, kreirali smo niz koji prima tip *Zaposleni* i u njega ubacili jedan objekat tipa *Profesor* i dva objekta tipa *Asistent*. Ovo inače ne bi bilo moguće (Java je strogo tipiziran jezik) da klase *Asistent* i *Profesor* nisu podklase klase *Zaposleni*. Prilikom iteracije kroz ovako kreiran niz, svi članovi se tretiraju kao instance klase *Zaposleni*, što znači da imamo pristup samo atributima i metodama iz te klase (recimo, nemamo pristup atributu *brojPredavanja* za profesore). Kako bismo mogli da pristupimo specifičnim metodama i atributima, moramo svakom članu niza promeniti tip (*type cast*). Proveru u koju konkretnu instancu treba da pretvorimo zaposlenog (profesora ili asistenta) vršimo operatorom **`instanceof`**.

```
for (int i = 0; i < zaposleni.length; i++) {
	// Ovde samo mozemo da dobijemo clanove niza kao objekte klase Zaposleni
	Zaposleni zaposlen = zaposleni[i];

	if(zaposlen instanceof Asistent) {
		// Promenu u odgovarajuci tip mozemo da izvrsimo tek nakon provere
		Asistent asistent = (Asistent)zaposlen;
		System.out.println("Ovaj zaposleni je asistent, broj vezbi: " +
					asistent.getBrojVezbi());
	}else if(zaposlen instanceof Profesor) {
		Profesor profesor = (Profesor)zaposlen;
		System.out.println("Ovaj zaposleni je profesor, broj
				predavanja: " + profesor.getBrojPredavanja());
	}
}
```

Zadatak

Napisati program koji omogućuje rad banke sa računima. Koristeći principe OOP izmodelovati klase *Racun*, *TekuciRacun*, *RacunStednje* i *Osoba*. Obezbediti da su svi atributi privatni, da postoje više konstuktora (bez parametara, sa parametrima, konstruktor kopije), korisničke metode i set/get metode za atibute klase. Napraviti test klasu.

1. **Osoba** je opisana sledećim podacima: JMBG, ime, prezime.
2. Klasa **Racun** treba da je apstraktna. Račun je opisana sledećim podacima: vlasnikom racuna (tipa `Osoba`), stanjem racuna(tipa `double`). U klasi napisati definiciju apstraktnih metoda boolean `isplata(double suma), void uplata(double suma).`
3. **TekuciRacun** nasleđuje klasu Racun i implementira apstraktne metode. Tekući račun je opisan sledećim podacima: mesecnaNaknada (tipa `double`). Klasa poseduje metodu `obracunajMesecnuNaknadu()` koja umanjuje stanje računa za mesečnu nakandu. Treba paziti da sa računa ne može da se podigne više novca nego što ima na stanju. Pri svakoj uplati novca banka uzima bankarsku proviziju tj. uplatu umanjuje za bankarsku proviziju od 0.01%.
4. **RacunStednje** nasleđuje klasu Racun i implementira apstraktne metode. Račun štednje je opisan sledećim podacima: pokrenutaStednja (tipa `boolean`) i godisnjiKoeficientStednje (tipa `double`). RacunStednje poseduje metodu `pokreniStednju()` kojom se menja stanje računa (menja se atribut pokrenutaStednja) i metodu void `obustaviStednju(int meseci)` kojom se prekida štednja i izračunava novo stanje računa po formuli:

   $$stanjeRacuna + stanjeRacuna \cdot (godisnjiKoeficient \cdot 0.01 \cdot brojGodina) + stanjeRacuna \cdot (godisnjiKoeficient \cdot 0.01 \cdot ostatakMeseci/12)$$

   Treba praziti da sa računa ne može da se podigne novac ukoliko je pokrenuta štednja i ukoliko se podiže više novca nego što ima na stanju. Pri uplati se ne obračunava naknada banke.

Zadatak za domaći

Napisati program i klase koje modeluju rad prodavnice računara. Za sve klase obezbediti da su svi atributi privatni (ili zaštićeni), da postoje više konstuktora (bez parametara, sa parametrima, konstruktor kopije), korisničke metode i set/get metode za atibute klase. Napraviti test klasu.

1. Kreirati klasu koja predstavlja korisnika. Korisnik je opisan sledećim podacima:Korisničko ime (jedinstveno),Lozinka, Ime, Prezime, Uloga (Kupac, Administrator).
2. Kreirati klasu koja predstavlja kategoriju komponente. Kategorija predstavlja tip komponente i opisana je sledećim podacima: Šifra kategorije(jedinstveno), Naziv, Opis, Nadkategorija (kategorija može, a ne mora, da bude podkategorija neke druge kategorije).
3. Kreirati klasu koja predstavlja komponentu računara. Komponenta je proizvod koji se prodaje i opsana je sledećim podacima: Šifra komponente(jedinstveno), Naziv, Cena, Raspoloživa količina, Opis, Kategorija (referenca ka kategoriji komponente).
4. Kreirati klasu koja predstavlja gotove konfiguracije računara. Gotova konfiguracija je proizvod koji se prodaje i opsana je sledećim podacima: Šifra konfiguracije(jedinstveno), Naziv, Cena, Raspoloživa količina, Opis, Komponente (lista referenci ka komponentama).
5. Kreirati 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 (može biti Komponenta ili Gotova Konfiguracija), Jedinična cena na dan kupovine, Količina.
6. Kreirati klasu koja predstavlja račun. Račun i opsan je sledećim podacima: Šifra računa(jedinstveno), Prodavac (referenca kakorisniku), Datum i Vreme, Ukupna Cena, Stavke računa (lista referenci ka stavkama računa).