Objektno orijentirano programiranje u Javi

Sadržaj

Nasljeđivanje među klasama

- Osim kreiranja novih klasa koje imaju zasebne članove, moguće je iskoristiti članske varijable i metode već postojeće klase korištenjem principa **nasljeđivanja**
- Postojeća klasa se u tom slučaju naziva **nadklasa** (engl. *superclass*), a nova klasa **podklasa** (engl. *subclass*)
- Podklasa kasnije može postati nadklasa drugim klasama koje nju nasljeđuju
- U podklasu se dodaju nove članske varijable i metode i predstavlja konkretniju implementaciju od svoje nadklase, pri čemu može mijenjati ponašanje svoje nadklase
- U Javi postoji klasa „java.lang.Object” koju izravno (kao prvu nadklasu) ili neizravno (preko drugih klasa) implicitno nasljeđuju sve klase u Javi (što nije potrebno posebno navoditi, jer se podrazumijeva)
- Java podržava samo jednostruko nasljeđivanje korištenjem ključne riječi „extends”, pri čemu svaka klasa može istovremeno nasljeđivati samo jednu nadklasu

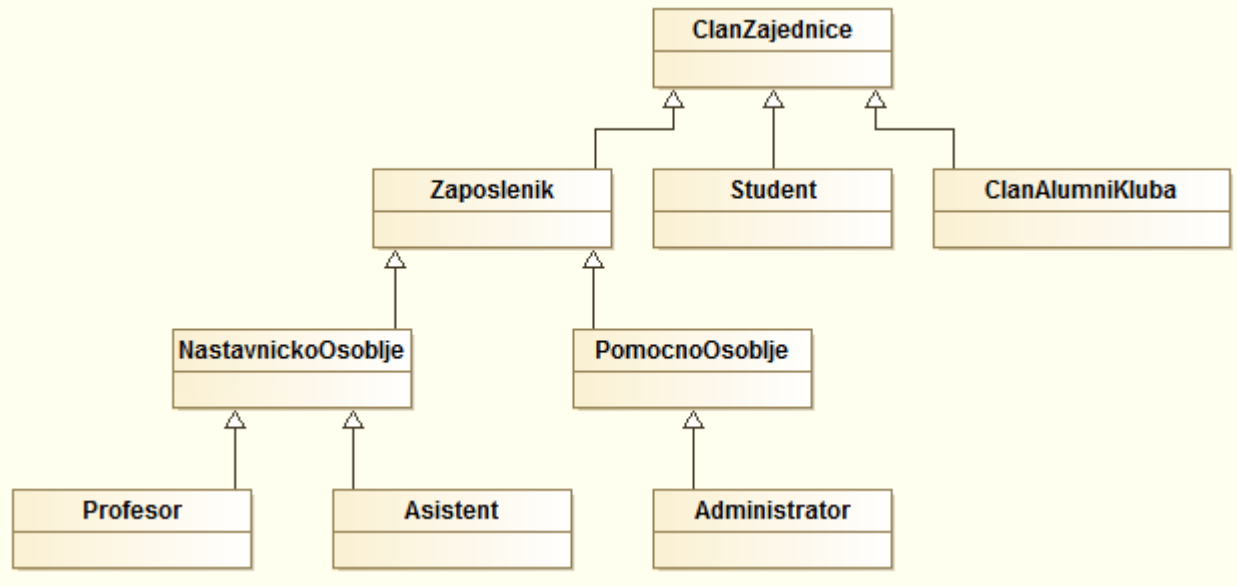
Nadklase i podklase

- Na primjer, ako se uvede osnovna klasa „BankovniKredit”, iz njega je moguće naslijediti klase „KreditZaAuto”, „KreditZaAdaptacijuNekretnine”, „HipotekarniKredit” itd.
- U tom slučaju je klasa „BankovniKredit” nadklasa, a npr. klasa „KreditZaAuto” njezina podklasa koja predstavlja konkretniju vrstu bankovnog kredita
- Još mogu postojati sljedeći primjeri nasljeđivanja:

Nadklasa	Podklase
Student	Apsolvent, DodiplomskiStudent
Oblik	Krug, Trokut, Pravokutnik, Kugla, Kocka
BankovniKredit	KreditZaAuto, KreditZaAdaptacijuNekretnine, HipotekarniKredit
Zaposlenik	NastavnickoOsoblje, PomocnoOsoblje
BankovniRacun	TekuciRacun, Ziroracun, DevizniRacun, KunskaStednja, DeviznaStednja

Hijerarhija nasljeđivanja

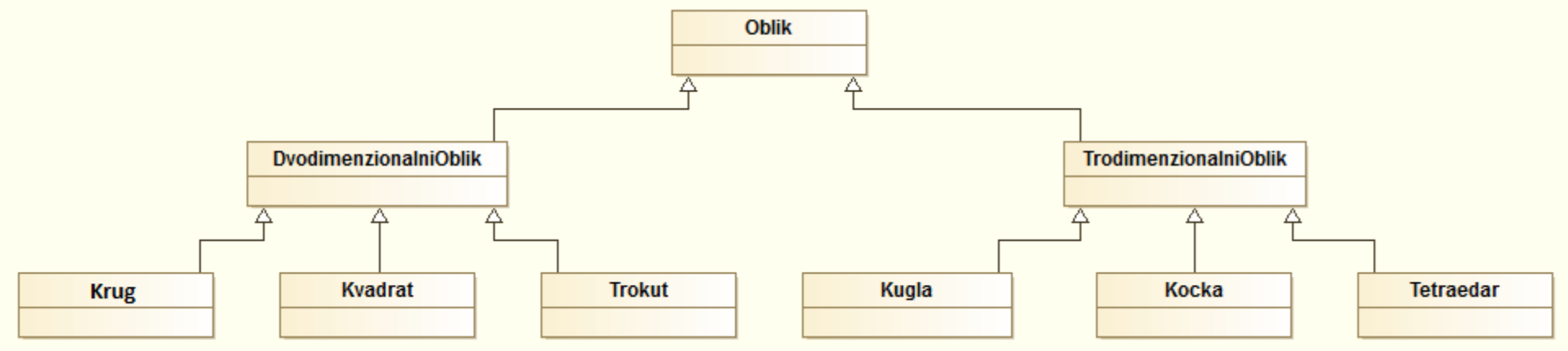
- Odnosi između klasa i njihovo nasljeđivanje često se označavaju grafičkim hijerarhijskim strukturama koje je moguće opisati i UML Class dijagramom
- Takav dijagram u slučaju opisa visokoškolske zajednice može izgledati ovako:



- Svaka „strelica” na UML dijagramu predstavlja „je” (engl. „*is a*”) vezu
- Npr. student je član zajednice, asistent je nastavničko osoblje, a administrator je pomoćno osoblje
- Klasa „ClanZajednice” je „Object”

Hijerarhija nasljeđivanja

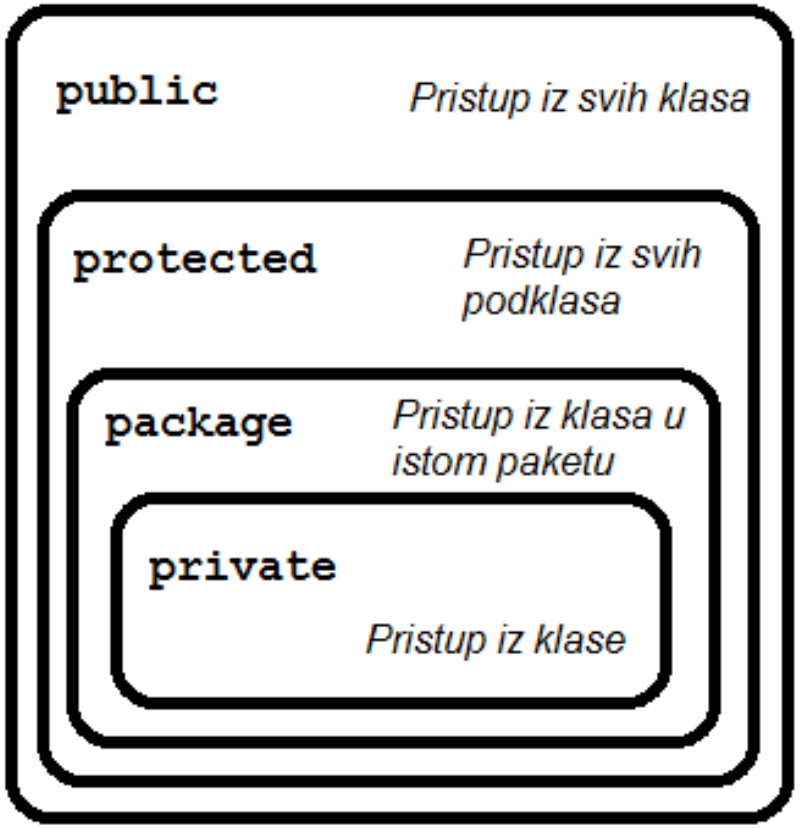
- U slučaju hijerarhije nasljeđivanja klase „Oblik”, UML Class dijagram izgleda ovako:



- Osim „je” veze postoji još i „ima” (engl. „*has a*”) veza koja definira članske varijable klase koje su objekti drugih klasa, na primjer, klasa „Knjiga” ima članske varijable „Izdavac” i „Autor”, a klasa „Zaposlenik” ima članske varijable klasa „String” koja označava njegovo prezime ili „LocalDate” koji označava njegov datum rođenja

Modifikator „protected”

- Osim modifikatora „public” i „private” postoji još i modifikator „protected”
- Modifikator „public” omogućava pristup članovima klase iz bilo koje druge klase, „private” omogućava pristup članovima klase samo unutar te iste klase, a „protected” omogućava podklasi pristup članovima nadklase i pristup svim članovima klasa koje se nalaze u istom paketu kao i klasa iz koje se tim članovima pristupa
- Ako se ne navede nijedan modifikator, podrazumijevani je „package private” koji omogućava pristup članovima klase samo ako su klase u istom paketu



Ključna riječ „super”

- Ako je iz neke klase potrebno dohvatiti članove nadklase, potrebno je koristiti ključnu riječ „super”
- Ključna riječ „super” predstavlja referencu na nadklasu, dok ključna riječ „this” predstavlja referencu na trenutnu klasu
- Na primjer, ako klasa „DvodimenzionalniOblik” ima varijablu „visina”, nju je iz klase „Trokut” moguće dohvatiti naredbom **super.visina**
- Osim pristupanja varijablama iz nadklase, ključnom riječju „super” može se pozvati i konstruktor nadklase korištenjem zagrada: **super()**

Konstruktori u podklasama

- Kreiranjem objekta podklase započinje ulančavano pozivanje podrazumijevanih konstruktora (ili konstruktora bez ulaznih parametara) svojih nadklasa
- Bez obzira navede li se ili ne, prva naredba svakog konstruktora podklase je poziv konstruktora nadklase (korištenjem naredbe „**super()**”)
- Tek nakon poziva konstruktora nadklase izvršavaju se sve ostale naredbe unutar konstruktora podklase
- Ako se ne navede takav raspored naredbi unutar konstruktora, kompajler javlja pogrešku

Nadjačavanje metoda u podklasama

- Ako je potrebno nadjačati metodu iz nadklase u podklasi, potrebno je koristiti anotaciju „@Override” te napisati identičnu definiciju metode iz nadklase

```
public class DvodimenzionalniOblik {

    private double visina;
    private double sirina;

    public double izracunajPovrsinu() {
        return visina * sirina;
    }
}
```

```
public class Krug extends DvodimenzionalniOblik {

    private double polumjer;

    @Override
    public double izracunajPovrsinu() {
        return Math.PI * Math.pow(polumjer, 2);
    }
}
```

- Anotacije predstavljaju upute kompajleru
- „@Override” označava nadjačavanje metoda, u slučaju neispravnog nadjačavanja se javlja pogreška

Klasa „Object”

- Sve klase u programskom jeziku Java izravno ili neizravno nasljeđuju klasu „java.lang.Object”
- Klasa „Object” sadržava niz metoda koje se često nadjačavaju (engl. *override*) kako bi se definiralo novo ponašanje klase, od čega su najvažnije:

Naziv metode	Opis
equals	Služi za uspoređivanje dvaju objekata i vraća „true” ako su objekti jednaki ili „false” ako nisu. U *defaultnoj* implementaciji uspoređuju se reference objekata, a ako je potrebno uspoređivati vrijednosti, onda je tu metodu potrebno nadjačati.
hashCode	Služi za definiranje algoritma izračunavanja sažetka objekta koja se koristi za optimizirano dohvaćanje objekata iz raznih struktura.
toString	Služi za definiranje String oblika objekta, pri čemu u *defaultnoj* implementaciji ispisuje oznaku reference na objekt.
clone	Služi za definiranje načina izrade kopije objekata (*shallow* ili *deep* copy).
wait, notify, notifyAll	Koriste se za rad s nitima i sinkronizaciju među njima.

Polimorfizam

- Omogućava „općenito programiranje” umjesto „specifičnog programiranja”
- Korištenjem koncepata polimorfizma može se dizajnirati sustav koji je lako proširiv
- Konkretni tipovi objekata mogu se određivati tek prilikom izvođenja programa i ne trebaju biti poznati kod pisanja i prevođenja
- Na primjer, svi objekti čije klase imaju zajedničku nadklasu mogu se spremiti u jedno zajedničko polje, s time da se kod izvođenja programa mogu pozivati specifične metode za svaku od podklasa iz kojih su kreirani objekti
- Objekt podklase može se tretirati kao objekt klase ako se nad njim izvrši „cast” operacija i u tom slučaju može koristiti samo metode iz nadklase, npr.

```
Trokut trokut = new Trokut();

((Oblik) trokut).metodaIzKlaseOblik();
```

Apstraktne klase i metode

- Apstraktne klase nisu predviđene da se iz njih kreiraju objekti, već su prvenstveno namijenjene za nasljeđivanje u drugim klasama
- Objekte iz njih nije moguće kreirati jer se radi o „nedovršenim klasama” koje se moraju dovršiti nasljeđivanjem u podklasama
- Označavaju se ključnom riječju „abstract”, npr. **public abstract class Oblik** {…}
- Apstraktne klase mogu osim „običnih” metoda sadržavati i **apstraktne metode**
- Ako klasa ima barem jednu apstraktnu metodu, automatski mora biti apstraktna (u suprotnom kompajler dojavljuje pogrešku)
- Apstraktna metoda također se označava ključnom riječju „abstract”, na primjer:

  public abstract double izracunajPovrsinu();

- Apstraktne metode su specifične po tome što **ne sadrže tijelo metode**

Apstraktne klase i metode

- Tek kad se u podklasama koje nasljeđuju apstraktnu klasu implementiraju sve apstraktne metode, tad je moguće kreirati objekte iz takvih klasa
- Prilikom implementacije apstraktnih metoda unutar podklasa potrebno je koristiti anotaciju „@Override”
- Ako klasa koja nasljeđuje apstraktnu klasu ne implementira sve apstraktne metode iz nadklase, ona također mora biti apstraktna
- Apstraktne klase ne moraju nužno sadržavati apstraktne metode, ali se svejedno moraju naslijediti da bi se mogli kreirati objekti iz njih
- Apstraktne klase mogu sadržavati i vlastiti konstruktor koji se poziva unutar konstruktora podklase

Ključna riječ „final”

- U slučaju da je potrebno spriječiti daljnje nasljeđivanje neke klase, kod njezine definicije potrebno je koristiti ključnu riječ „final”, npr.

public final class Krug extends DvodimenzionalniOblik

- Slično vrijedi i za metode, u slučaju da je potrebno spriječiti daljnje nadjačavanje neke metode u podklasama, potrebno je također koristiti ključnu riječ „final”:

public final double izracunajOpsegKruga();

- Metode koje su označene modifikatorom „private” ne mogu se pozivati izvan klase u kojoj se nalaze, pa su samim time implicitno označene i modifikatorom „final”
- Slično vrijedi i za metode zajedničke za sve objekte, označene modifikatorom „static”
- Ključnom riječju „final” označavaju se i konstantne vrijednosti, npr.

public static **final** String NAZIV_DRZAVE = ’’Hrvatska’’;

Definiranje sučelja

- Omogućavaju da klase koje se međusobno ne nasljeđuju implementiraju zajedničku skupinu metoda
- Time se postiže „standardiziranje” načina komunikacije među klasama
- Sučelja opisuju koje sve „operacije” se trebaju moći obavljati, ali ne sadrže nužno i njihovu implementaciju
- Definicija sučelja uključuje ključnu riječ „interface”, a naziv samog sučelja mora označavati neko svojstvo, npr.

```
public interface Elektricno {

  void ukljuci();

}
```

- Sučelja osim metoda mogu sadržavati i konstantne vrijednosti, a svi članovi sučelja moraju biti „public”

Implementiranje sučelja

- Svaka klasa koja implementira određeno sučelje mora implementirati sve metode koje nemaju svoje tijelo (slično kao i kod apstraktnih klasa i metoda)
- Za razliku od nasljeđivanja gdje je moguće naslijediti samo jednu klasu, svaka klasa može implementirati više sučelja
- Prilikom implementiranja sučelja klasa mora koristiti ključnu riječ „**implements**”, na primjer:

public class ElektricnaGitara extends Gitara **implements** Elektricno

- U slučaju da klasa ne implementira sve metode iz sučelja koje implementira, klasa mora biti apstraktna
- U Javi postoje i neka sučelja koja nemaju nijednu metodu i služe za označavanje određenog svojstva klase, npr. „Serializable” označava da se objekti neke klase mogu ispravno serializirati (pretvoriti u „String” oblik i npr. zapisati u datoteku ili poslati nekom udaljenom servisu putem interneta)

Proširenja funkcionalnosti sučelja u Javi 8

- Prije Jave 8 sučelja su mogla sadržavati samo javne metode bez implementacije
- Od Jave 8 sučelja mogu sadržavati i **podrazumijevane implementacije** metoda koje se označavaju s ključnom riječju „default”
- Klasa koja implementira sučelje s „default” metodom može koristiti tu podrazumijevanu implementaciju ili je nadjačati i definirati vlastitu implementaciju
- Primjer „defaultne” implementacije metode:

```
public interface PracenjeVremena {
default public String dohvatiDatumVrijeme() {
    LocalDateTime localDateTime = LocalDateTime.now();
    return localDateTime.format(DateTimeFormatter.ofPattern("dd.MM.yyyy. HH:mm:ss"));
  }
}
```

Proširenja funkcionalnosti sučelja u Javi 8

- Osim podrazumijevanih implementacija metoda u Javi 8 je omogućeno korištenje i statičkih metoda unutar sučelja
- Statičke metode sadrže zajedničku logiku za sve objekte klase koja implementira takvo sučelje i mogu se koristiti izravno iz same klase, bez potrebe za kreiranjem objekata, npr.

```
public interface Provjera {
  public static boolean provjeriPrazanString(String tekst) {
    if (tekst == null || tekst.isEmpty()) {
      return true;
    }
    else {
      return false;
    }
  }
}
```

Proširenja funkcionalnosti sučelja u Javi 8

- Java 8 još uključuje i funkcionalna sučelja koja omogućavaju korištenje **lambda izraza**
- Svako sučelje koje sadrži **samo jednu apstraktnu metodu** naziva se **funkcionalnim sučeljem**
- Moguće je napisati vlastita funkcionalna sučelja ili koristiti neka koja već postoje u Javi (npr. „Comparator”, „ActionListener”, „Runnable” ili slično)
- Kod funkcionalnog programiranja potrebno je specificirati što je potrebno napraviti, a ne kako
- Na primjer, umjesto računanja sume elemenata u cjelobrojnom polju pomoću petlje, korištenjem funkcionalnog programiranja i interne iteracije po elementima moguće je sumu izračunati korištenjem znatno manje naredbi
- Lambda izrazi omogućavaju pisanje skraćene notacije za kreiranje anonimnih metoda koje kompajler automatski prevodi u anonimne klase
- Anonimne klase su one klase čije tijelo se definira prilikom instanciranja objekta te klase

Proširenja funkcionalnosti sučelja u Javi 8

- Na primjer, ako je potrebno kreirati objekt klase koja nasljeđuje sučelje „Comparator” (koje služi za postavljanje kriterija sortiranja objekata npr. u polju), to je moguće napraviti na sljedeći način:

```
Comparator<String> c = new Comparator<String>() {
    int compare(String s, String s2) { ... }
};
```

- Sintaksa lambda funkcija se sastoji od definiranja liste parametara, nakon čega slijedi oznaka za strelicu „->” i na kraju tijelo lambda funkcije:

```
(listaParametara) -> {naredbeLambdaFunkcije}
```

- Na primjer, sljedeća lambda funkcija prima dva cijela broja i vraća njihovu sumu:

```
(int x, int y) -> {return x + y;}
```

- Moguće je izostaviti tipove parametara iz liste, kao i vitičaste zagrade u slučaju da postoji samo jedna naredba

Proširenja funkcionalnosti sučelja u Javi 9

- U Javi 9 unutar sučelja je omogućeno kreiranje i privatnih metoda unutar sučelja, kako bi se mogle koristiti unutar npr. podrazumijevanih metoda:

```
public interface Sucelje {

    private void test() {
        System.out.println("Test");
    }

    public static void test2() {
        System.out.println("Test 2");
    }

    public default void test3() {
        test();
    }
}
```

Pitanja?