Objektno-orjentisano programiranje

Sadržaj

Nasleđivanje

- Omogućava da zajedničke funkcionalnosti izdvojimo kao posebnu klasu
- Ovu klasu zatim mogu da naslede druge, specifičnije klase, pri čemu svaka dodaje ono što je za nju jedinstveno
- Klasa koja se nasleđuje je roditeljska klasa - parent, a klasa koja je nasleđuje je klasa naslednica - child

Primer nasleđivanja

- Roditeljska klasa Figura

```
public class Figura {
    public double izracunajPovrsinu(){
        return 0;
    }

    public double izracunajObim(){
        return 0;
    }
}
```

- Ovu klasu mogu da naslede klase Kvadrat i Pravougaonik
- Za nasleđivanje koristi se ključna reč **extends** i navodi se klasa koja se nasleđuje
- Primer klase Kvadrat

```
public class Kvadrat extends Figura{
```

Redefinisanje metoda

- Metode iz roditeljske klase možemo direktno koristiti ili po potrebi napisati novu funkcionalnost redefinišući tu metodu u klasi naslednici
- Za klasu Kvadrat redefinišu se metode **izracunajPovrsinu()** i **izracunajObim()** tako da se računaju vrednosti koristeći vrednost atributa stranica

Primer klase Kvadrat

```
public class Kvadrat extends Figura{
    protected double stranica;

    public Kvadrat(double stranica) {
        this.stranica = stranica;
    }

    @Override
    public double izracunajPovrsinu() {
        return stranica*stranica;
    }

    @Override
    public double izracunajObim() {
        return 4*stranica;
    }
}
```

Klasa Object

- Sve Java klase direktno ili indirektno nasleđuju klasu Object
- Klasa Object definiše osnovne metode koje imaju svi objekti u Javi a to su
  - equals(o)
  - toString()
  - hashCode()
  - getClass()
  - clone()
  - finalize()
  - wait()
  - notify()
  - notifyAll()

Metoda toString

- Definiše String reprezentaciju klase za koju je definisana
- Svaka klasa ima definisanu ovu metodu preko klase Object
- Prilikom prikaza vrednosti može se direktno proslediti referenca na objekat i automatski će biti pozvana toString metoda
- Redefinisana metoda za klasu Kvadrat

```
@Override
public String toString() {
    return "Kvadrat{stranica=" + stranica +'}';
}
```

Ključna reč super

- Ključna reč **super** je referenca na roditeljsku klasu
- Koristi se za poziv konstruktora i metoda iz roditeljske klase
- Unutar konstruktora klase naslednice poziv konstruktora roditeljske klase mora biti prva naredba

```
public Figura(String vrednost){
        System.out.println("Konstruktor sa parametrom vrednost =
            "+vrednost);
    }

public Kvadrat(double stranica)
    {
        super("Kvadrat");
        this.stranica = stranica;
    }
```

Interfejs

- Je nacrt (blueprint) klase
- Ne može biti instanciran
- Ima samo statičke konstante i apstraktne metode
- Apstraktne metode nemaju telo i moraju biti redefinisane u klasi koja implementira interfejs
- Da je neka metoda apstraktna odredjuje se dodavanjem ključne reči **abstract** u definiciji metode
- Jedna klasa može da implementira više interfejsa
- Klasa koja implementira interfejs navodi ključnu reč **implements** i naziv interfejsa
- Jedan interfejs može da nasledi drugi interfejs navodeći ključnu reč **extends**

Figura kao interfejs

```
public interface IFigura {
    public abstract double
        izracunajPovrsinu();
    public abstract double izracunajObim();
}
```

- Primer klase Kvadrat koja implementira interfejs

```
public class Kvadrat implements Figura {
    protected double stranica;

    @Override
    public double izracunajPovrsinu() { }

    @Override
    public double izracunajObim() { }
}
```

Polimorfizam

- Polimorfizam je mogućnost objekta da se pojavi u više formi
- Omogućava da prilikom definisanja tipa promenljive koristimo roditeljsku klasu ili interfejs, a da instanciramo objekat klase naslednice - child

```
Figura f=new Kvadrat(8);
```

- Koristimo kod kolekcija kada želimo da u jednoj kolekciji čuvamo objekte koji su instance različitih klasa naslednica

```
ArrayList<Figura> figure=new ArrayList<Figura>();
figure.add(new Kvadrat(5));
figure.add(new Kvadrat(3));

for(Figura figura:figure)
    System.out.println("figura = " + figura);
```

instanceof operator

- proverava da li je objekat tipa koji testiramo

```
Figura f = new Kvadrat(8);
Kvadrat k=null;
if (f instanceof Kvadrat) {
    k = (Kvadrat) f;
}
```

- kada objekat jeste tog tipa, možemo taj objekat cast-ovati u promenljivu tog tipa

Anonimne klase

- Se koriste kada je potrebno da se navede novi tip, ali koji će se koristiti samo jednom
- Definisanje i instanciranje se obavlja u istoj naredbi
- Kao osnova koristi se već postojeća klasa ili interfejs čije se metode redefinišu i po potrebi dodaju novi atributi i metode
- Za prosledjivanje vrednosti anonimnoj klasi može se koristiti promenljiva koja u definiciji ima ključnu reč **final** čime se određuje da se vrednost promenljive neće menjati tokom izvršavanja programa

Primer anonimne klase za kvadrat

```
public class TestAnonimna {
    public static void main(String[] args) {
        final double stranica=8;
        Figura f=new Figura() {

            @Override
            public double izracunajPovrsinu() {
                return stranica*stranica;
            }

            @Override
            public double izracunajObim() {
                return 4*stranica;
            }
        };
    }
}
```

Uništavanje objekata - Garbage collector

- Ne postoji destruktor
- Garbage collector radi kao poseban proces u pozadini
- Automatska dealokacija memorije
- Automatska defragmentacija memorije
- Poziva se po potrebi
  - možemo ga eksplicitno pozvati sledećim kodom: System.gc();
    ali Garbage Collector će sam "odlučiti" da li će dealocirati memoriju
  - poziv ove metode je samo sugestija GC-u da bi mogao da otpočne čišćenje

Garbage collector

- Možemo napisati posebnu metodu **finalize()**, koja se poziva neposredno pre oslobađanja memorije koju je objekat zauzimao
  - nemamo garanciju da će biti pozvana
- I pored Garbage Collector-a može doći do OutOfMemory ako ne vodimo računa

Ključna reč static

- Definiše statičke atribute i metode

```
class StaticTest {
  static int i = 47;
  static void metoda() { i++; }
}
```

- Statički atributi i metode postoje i bez kreiranje objekta
  - zato im se može pristupiti preko imena klase StaticTest.i++;
- Statički atributi imaju istu vrednost u svim objektima
  - ako promenim statički atribut u jednom objektu, on će se promeniti i kod svih ostalih objekata

Ključna reč static

- Namena statičkih metoda:
  - pristup i rad sa statičkim atributima
  - opšte metode za koje nije potrebno da se kreira objekat
- Primeri upotrebe:
  - System.out.println(); // out je staticki atribut
  - Math.random();
  - Math.sin();
  - Math.PI;
  - public static void main(String[] args) ...

Statički blok

- Statički blok se izvršava samo jednom, prilikom prvog korišćenja klase
- Unutar statičkog bloka može se pristupati samo statičkim atributima i mogu se pozivati samo statičke metode
- Unutar statičkih metoda može se samo pristupiti statičkim promenljivim i drugim statičkim metodama

Statički blok

```
public class Test {
    static int a ;
    static int b ;
    int c;
    static void f () {
        /*
        Promenljiva koja nije static ne moze se menjati unutar staticke
            metode
         */
        b = 6;
    }
    static {
        a = 5;
        f();
    }

    public int metoda()
    {
        b=7;
        return 7;
    }
}
```

Ugnježdene klase

- Unutar jedne klase može se definisati nova klasa

```
class ObuhvatajucaKlasa{ class
   UgnjezdenaKlasa{ . . . } }
```

- Na ovaj način se mogu definisati novi tipovi koji neće biti vidljivi van Obuhvatajuće klase, ako se za modifikator pristupa postavi vrednost **private** ili **protected**
- Ako se za modifikator pristupa postavi **public** pristup ugnježdenoj klasi se obavlja koristeći kvalifikaciju:
  $<ObuhvatajućiTip>.<UgnježdeniTip>$

Instanciranje ugnježdenih klasa

- Ako je za modifikator pristupa ugnježdene klase postavljen modifikator **public**:

```
ObuhvatajucaKlasa.UgnjezdenaKlasa p1=new ObuhvatajucaKlasa().new  UgnjezdenaKlasa();
```

  - Potrebno je kreirati instancu obuhvatajuće klase jer ugnježdena klasa čuva referencu na objekat obuhvatajuće klase nad kojim je kreirana

- Ako je za modifikator pristupa ugnježdene klase postavljen modifikator **public** i za ugnježdenu klasu se još doda i ključna reč **static**:

```
ObuhvatajucaKlasa.UgnjezdenaKlasa p2=new ObuhvatajucaKlasa.UgnjezdenaKlasa();
```

Lokalna klasa

- Lokalne klase se mogu kreirati unutar bloka naredbi:
  - static bloka
  - bloka naredbi metode obuhvatajuće klase

```
class ObuhvatajucaKlasa{

  public void metoda(){

      class LokalnaKlasa{

      };

      LokalnaKlasa k=new LokalnaKlasa();

   }

  //Van tela metode se vise ne moze koristiti

  // klasa LokalnaKlasa

}
```

- Lokalne klase se mogu koristiti samo unutar bloka naredbi u kojem su definisane
- Za lokalne klase se ne može postaviti modifikator pristupa