SOFTVERSKO INŽENJERSTVO

školska 2024/2025 godina

Vežba 2: Obnavljanje OOP koncepata u Javi

Cilj vežbe

Razumevanje i primena osnovnih OOP koncepata u Javi:

- klase i objekti
- nasleđivanje
- enkapsulacija
- polimorfizam
- apstraktne klase
- interfejsi
- izuzeci (exceptions)

1. Klase i objekti

Klasa predstavlja sablon za kreiranje objekata. Sadrži atribute (promenljive) i metode (funkcije) koje definišu ponašanje objekata. **Objekat** je instanca klase i može imati jedinstvene vrednosti za svoje atribute.

Klase su osnova objektno-orijentisanog programiranja, jer omogućavaju grupisanje podataka i funkcionalnosti koje se odnose na te podatke.

```
class Student {
    String ime;
    int godina;

    void predstaviSe() {
        System.out.println("Zdravo, ja sam " + ime + " i studiram " + godinu
                                                + ". godinu.");
    }
}
```

2. Nasleđivanje (Inheritance)

Nasleđivanje omogućava jednoj klasi (podklasi) da preuzme atribute i metode druge klase (nadklase). Koristi se za ponovnu upotrebu koda i hijerarhijsku organizaciju.

Prednosti:

- Smanjuje dužinu koda
- Omogućava postepeno nadograđivanje funkcionalnosti

```
class Osoba {
    String ime;

    void pozdrav() {
        System.out.println("Zdravo, ja sam " + ime);
    }
}

class Student extends Osoba {
    int brojIndeksa;

    void prikaziIndeks() {
        System.out.println("Moj broj indeksa je: " + brojIndeksa);
    }
}
```

3. Enkapsulacija (Encapsulation)

Enkapsulacija podrazumeva skrivanje podataka klase i omogućavanje pristupa isključivo putem metoda.

Koristimo private atribute i getter/setter metode za pristup. Ovaj pristup:

- Štiti podatke
- Omogućava validaciju

```
class Zaposleni {
    private double plata;

    public void setPlata(double plata) {
        if (plata > 0) {
            this.plata = plata;
        }
    }
```

```
    public double getPlata() {
        return plata;
    }
}
```

4. Apstraktne klase (Abstract Classes)

Apstraktna klasa ne može biti instancirana. Koristi se kao osnova za druge klase. Može imati:

- Apstraktne metode (bez tela)
- Konkretne metode (sa implementacijom)

Apstraktne klase se koriste kada želimo da definišemo osnovno ponašanje, a da podklase dodefinisu specifičnosti.

```
abstract class Oblik {
    String boja;

    public Oblik(String boja) {
        this.boja = boja;
    }

    public void prikaziBoju() {
        System.out.println("Boja oblika: " + boja);
    }

    public abstract double izracunajPovrsinu();
}

class Krug extends Oblik {
    double r;

    public Krug(String boja, double r) {
        super(boja);
        this.r = r;
    }

    @Override
    public double izracunajPovrsinu() {
        return Math.PI * r * r;
    }
}
```

5. Interfejsi (Interfaces)

Interfejs je ugovor koji klasa mora da ispuni. Sadrži samo potpise metoda. Koristi se za:

- Definisanje sposobnosti klasa
- Višestruko nasleđivanje (Java ne podržava višestruko nasleđivanje klasa, ali podržava višestruko implementiranje interfejsa)

```
interface Vozilo {
    void pokreni();
    void zaustavi();
}

class Auto implements Vozilo {
    @Override
    public void pokreni() {
        System.out.println("Auto se pokreće");
    }

    @Override
    public void zaustavi() {
        System.out.println("Auto staje");
    }
}
```

Interfejsi se koriste kada želimo da više nepovezanih klasa implementiraju istu funkcionalnost.

6. Polimorfizam (Polymorphism)

Polimorfizam omogućava da ista metoda ima više oblika:

- Statički (metod overloading)
- Dinamički (metod overriding)

Najčešće se koristi kroz zajednički interfejs/nadklasu:

```
class Zivotinja {
    public void zvuk() {
        System.out.println("Životinje prave zvuk.");
    }
}
```

```
class Pas extends Zivotinja {
    @Override
    public void zvuk() {
        System.out.println("Av av!");
    }
}

class Macka extends Zivotinja {
    @Override
    public void zvuk() {
        System.out.println("Mjau mjau!");
    }
}

public class Main {
    public static void main(String[] args) {
        Zivotinja z1 = new Pas();
        Zivotinja z2 = new Macka();

        z1.zvuk();  // Av av!
        z2.zvuk();  // Mjau mjau!
    }
}
```

7. Rukovanje izuzecima (Exception Handling)

Izuzeci (exceptions) u Javi predstavljaju način obrade grešaka koje se mogu javiti tokom izvršavanja programa. Java koristi try-catch blokove za hvatanje i obradu izuzetaka.

Postoje dve glavne vrste izuzetaka:

- **Checked exceptions** – moraju biti obuhvaćeni try-catch blokom ili deklarisani u metodi
- **Unchecked exceptions** – potklase klase RuntimeException

```
public class PrimerIzuzetka {
    public static void main(String[] args) {
        try {
            int a = 5 / 0;
            System.out.println("Rezultat: " + a);
        } catch (ArithmeticException e) {
            System.out.println("Greška: Deljenje nulom nije dozvoljeno.");
```

```
        } finally {
            System.out.println("Ova poruka se uvek prikazuje.");
        }
    }
}
```

Možemo takođe kreirati i **sopstvene izuzetke**:

```
class NegativnaPlataException extends Exception {
    public NegativnaPlataException(String poruka) {
        super(poruka);
    }
}

class Zaposleni {
    private double plata;

    public void setPlata(double plata) throws NegativnaPlataException {
        if (plata < 0) {
            throw new NegativnaPlataException("Plata ne može biti
                                                    negativna!");
        }
        this.plata = plata;
    }
}
```

Zadatak za samostalni rad

Tema: Sistem za obradu naloga korisnika u banci

1. Napraviti apstraktnu klasu Nalog sa:

- brojem računa (String)
- stanjem (double)
- metodom prikaziStanje()
- apstraktnom metodom izvrsiTransakciju(double iznos)

2. Napraviti klase:

- TekuciRacun: transakcija moguća ako ima dovoljno sredstava
- DevizniRacun: može ići u minus do -500 EUR

3. Napraviti interfejs VerifikacijaKorisnika sa metodom verifikuj() i primeniti ga

4. Koristiti:

- enkapsulaciju (private atributi + get/set metode)
- polimorfizam za izvršavanje transakcija na različitim vrstama naloga

5. Kreirati listu više naloga i izvršiti transakcije kroz petlju

6. (Bonus) Implementirati izuzetke:

- Ako se pokuša transakcija koja nije dozvoljena, baciti korisnički definisan izuzetak