Róbert Junas

Princípy OOP

Asociácia – používam hlavne v triedach používateľov, menovite v **skladnik.java**, kde veľa funkcií vyžaduje objekt **sklad** a **kniha** na správne fungovanie. Príklad funkcie z triedy **skladnik.java**

```
public void odoberKnihyZRegalu(Sklad s, Kniha k, int pocet, int[] pozicia){
    if(kniha != null) return;
     if(pozicia == null){
         NoveKnihy r = s.getNovyTovar();
         if(r != null && r.existujeKniha(k)) {
             int p = r.odoberKnihy(k, pocet);
             pridajKnihy(k, p);
         }
     }else{
         Regal r = s.getSekcie(pozicia[0]).getRegal(pozicia[1]);
         if(r.existujeKniha(k)){
             int p = r.odoberKnihy(k,pocet);
             pridajKnihy(k,p);
         }
     }
}
```

Agregácia – tento princíp sme použili pri spôsobe ukladania kníh v **regáli** (**Regal.java**) a **palete (NoveKnihy.java). Knihy** nie sú súčasťou jedného regálu ale môžu sa premiestňovať po všetkých regáloch a navyše **sklad.java** pracuje s katalógom **kníh**, kde sú uložene referencie na knihy.

```
public class Regal {
    protected ArrayList<Kniha> zoznamKnih;
    protected HashMap<String, Integer> pocetKnih;
protected void pridajKnihyP(Kniha k, int p){
     if(!existujeKniha(k)){
         zoznamKnih.add(k);
         pocetKnih.put(k.getISBN(),p);
     }else{
         pocetKnih.replace(k.getISBN(),pocetKnih.get(k.getISBN()) + p);
     }
}
```

Kompozícia – je súčasťou kníhkupectva ako celku, pretože **kníhkupectvo (knihkupectvo.java)** by nemalo existovať bez toho aby v sebe nemalo **sklad (sklad.java)** a **predajňu (predajna.java)**. A zase **sklad** ani **predajna** by nemala existovať bez knihkupectva. Tento vztah je aj medzi **skladom** a **sekciami**; a ešte medzi **sekciami** a **regalmi.**

```
public Knihkupectvo(){
    sklad = new Sklad();
    predajna = new Predajna();
}
```

Dedenie – umožňuje, aby funkcionalita z jednej triedy bola súčasťou druhej. Tento vzťah sme uplatnili medzi **Miestnost.java** a **Sklad.java/Predajna.java**. A medzi **Regal.java** a **NoveKnihy.java,** ktoré obidve slúžia na ukladanie kníh určitou formou. **Viacnásobne dedenie** sme využili pri triedach používateľov, kde napr. **skladník** dedí atribúty a funkcie **zamestnanca** a ten zase dedí **používateľa**, ktorý je ešte dedení zákazníkom.

```
abstract class Zamestnanec extends Pouzivatel{

public class Skladnik extends Zamestnanec{
```

Pretože niektoré triedy nepotrebujú aby sa dali vytvoriť ich inštancie, napr. **Pouzivatel.java, Zamestnanec.java** a **Miestnost.java**, tak na tieto triedy sme použili **abstrakciu**. Premennú typu **Pouzivatel** používam v **Knihkupectvo.java** vo funkcií **main.** Táto premenná ukazuje na prihláseného používateľa a spúšťa funkciu na **spracovanie vstupu**.

```
public abstract class Pouzivatel {   ...   }

 Pouzivatel p = new Zakaznik();

p.spracuj(command, knihkupectvo.sklad, knihkupectvo.predajna);
```

Enkapsulácia – všetky nestatické atribúty sú **private**, prípadne ak sa v triede používa atribút svojej rodičovskej triedy, tak v rodičovskej triede je tento atribút **protected.** Skoro všetky tieto atribúty majú nastavení **getter** ale iba niektoré majú nastavene **setter**.

```
public class Kniha {
    private String nazov, autor, isbn, jazyk, vydavatelstvo;
    private Kategoria kategoria;
    private Vazba vazba;
    private int pocetStran, rok, predaneKusy;
    private float cena;


 public Kategoria getKategoria() { return kategoria; }
 public String getISBN() { return isbn; }
 public String[] getBasicInfo() { return new String[] {nazov,autor,isbn,vydavatelstvo};}
 public String getVydavatel() {return vydavatelstvo;}
 public boolean isBestseller() { return kategoria == Kategoria.BESTSELLER; }

 public void setBestseller() { kategoria = Kategoria.BESTSELLER; }
```

Trieda i Kniha.java

```
public void setOtvorene(boolean b) { otvorene = b; }

public boolean isOtvorene() { return otvorene; }
```

Trieda ii Predajna.java

Overriding – tento princíp nám dovoľuje z dedených tried zobrať funkcie a upraviť ich na nami žiadanú funkcionalitu. Využil som ho hlavne v dedeniach triedy **používateľ**, kde sa nachádza funkcia **spracuj**, ktorá spracováva vstupné príkazy. Ďalej som ho použil v triede **NoveKnihy.java**, ktorá dedí triedu **Regal.java** , tak že funkcia **odoberKnihy** namiesto uvoľňovania miesta kontroluje či ešte zostali knihy v **NoveKnihy.**

```
public void spracuj(String s, Sklad sklad, Predajna predajna){
    if(s.equals("help")){
        System.out.println("---Vseobecne prikazy---");
        System.out.println("info-me - informacie o mne");
        System.out.println("logout - odhlasit sa");
        System.out.println("help - vypis pomocky");
        System.out.println("exit - vypni system");

    } else if(s.equals("exit")){
        exit();
    } else if(s.equals("logout")){
        Knihkupectvo.setPrihlaseny(LoggedIn.NOONE);
    } else if(s.equals("info-me")){
        this.vypisInfo();
    }
}
```

Trieda iii Pouzivatel.java

```
@Override
public void spracuj(String s, Sklad sklad, Predajna predajna){
    super.spracuj(s,sklad,predajna);
    pridajHodinu();
    if(s.equals("help")){
        System.out.println("---Zamestnanecke prikazy---");
        System.out.println("plat - zisti svoj plat");
        System.out.println("odrobene - kolko si uz odrobil");
        System.out.println("zarobene - vypocita kolko si uz zarobil");

    } else if(s.equals("plat")){
        System.out.println("Tvoj plat je: " + getPlat());
    }else if(s.equals("zarobene")){
        System.out.println("zarobil si: " + vypocitajPlat());
    }else if(s.equals("odrobene")){
        System.out.println("odrobil si: " + getOdrobene());
    }
}
```

Trieda iv Zamestnanec

Róbert Junas

Overloading – príklad overloading-u mám v triede **skladnik.java**, kde skladník môže nájsť referencie na **knihu** pomocou reťazca znakov alebo podľa katalógového čísla. Využil som aj preťažovanie konštruktorov; hlavne v **Sekcia.java**

```
public Kniha najdReferenciuNaKnihu(Sklad s, int i) { return s.getKatalog().get(i); }

public Kniha najdReferenciuNaKnihu(Sklad s, String id){
    for(Kniha kp : s.getKatalog()){
        if(kp.getISBN().toLowerCase().equals(id) || kp.getBasicInfo()[0].toLowerCase().equals(id)){
            return kp;
            }
    }
    return null;
}
```

```
public Sekcia(){init(defaultSize);}
public Sekcia(int pocetRegalov) { init(pocetRegalov); }
```