

# SPRÁVA O REALIZÁCIÍ PROJEKTU Z PREDMETU OBJEKTOVO-ORIENTOVANÉ PROGRAMOVANIE

NÁZOV PROJEKTU: POŠTOVÝ SYSTÉM

---

Lucia Rapánová, FIIT STU

OOP 2019/2020

## Pôvodný zámer projektu

Pre svoje zadanie som si vybrala správu systému pre poštové služby. Takýto systém musí byť bezpečný a spoľahlivý, pretože spravuje poštové zásielky, ktoré musia byť doručené konkrétnym ľuďom, správne a zodpovedne. Zároveň sa často pracuje aj s citlivými údajmi a peniazmi.

Na pošte sa nachádzajú rôzni zamestnanci, ktorí vykonávajú určité úlohy. Vedúci pošty má vo všeobecnosti kontrolu nad všetkými zásielkami a zamestnancami, prideliť im každé ráno peniaze, vypláca peniaze ľuďom, prijíma zásielky na podaj, vydáva zásielky, predáva poštové produkty (pohľadnice, známky, časopisy, noviny, žreby) a podobne. Pracovníci za poštovou priehradkou majú podobnú náplň práce, ale s tým, že nemajú prístup do celého systému a nemajú kontrolu nad ostatnými zamestnancami. Môžu pracovať na celý alebo na polovičný úväzok. Poštovní doručovatelia každé ráno prijímú poštu, rozdelia si ju, zásielky, ktoré majú podacie číslo, skenujú do počítača, majú pridelenú určitú hotovosť, vyplácajú dôchodky a tiež predávajú produkty. Zásielky bývajú rôznych druhov, napríklad doporučené listy, do vlastných rúk, úradné zásielky, dobierky, poistené listy, balíky a podobne. Obsahujú adresu odosielateľa a prijímateľa a prípadne ďalšie informácie, napríklad hmotnosť pri zásielkach väčších rozmerov. Zahrnúť by som chcela aj jedného pracovníka, ktorý bude spravovať výplaty pre zamestnancov (napríklad finančný manažér).

Prechod medzi používateľmi by som chcela simulovať jednoduchým spôsobom (napríklad prihlási sa jeden pracovník, má dostupné určité funkcie, potom sa prihlási druhý a podobne).

## Štruktúra a UML diagram

Väčšina vecí sa z pôvodného zámeru zachovala, ale samozrejme pribudlo grafické používateľské rozhranie a ďalšie rozšírenia projektu. V programe sa nachádza jedna hlavná trieda, odkiaľ sa program spúšťa, s názvom PostaGUI. Odtiaľ sa prechádza na ďalšie scény. Je možnosť aj prepínania sa medzi používateľmi a každý používateľ má mierne odlišnú scénu. Prihlasovanie údaje sa nachádzajú v triede CheckLogin.

V TovaryScreen sa pracuje s tovarmi. Tovary sú nadtrieda, z ktorej dedia ďalšie podtriedy, a to už konkrétne tovary - známky, pohľadnice, žreby, časopisy, noviny. Každý tovar má aj svoj druh ako typ enum. Tovary sa dajú pridávať a predávať. Ukladajú sa tiež do súboru pomocou serializácie.

Do tohto dokumentu som dala len čiastkový diagram kvôli prehľadnosti, celý diagram sa nachádza v samostatnom súbore.

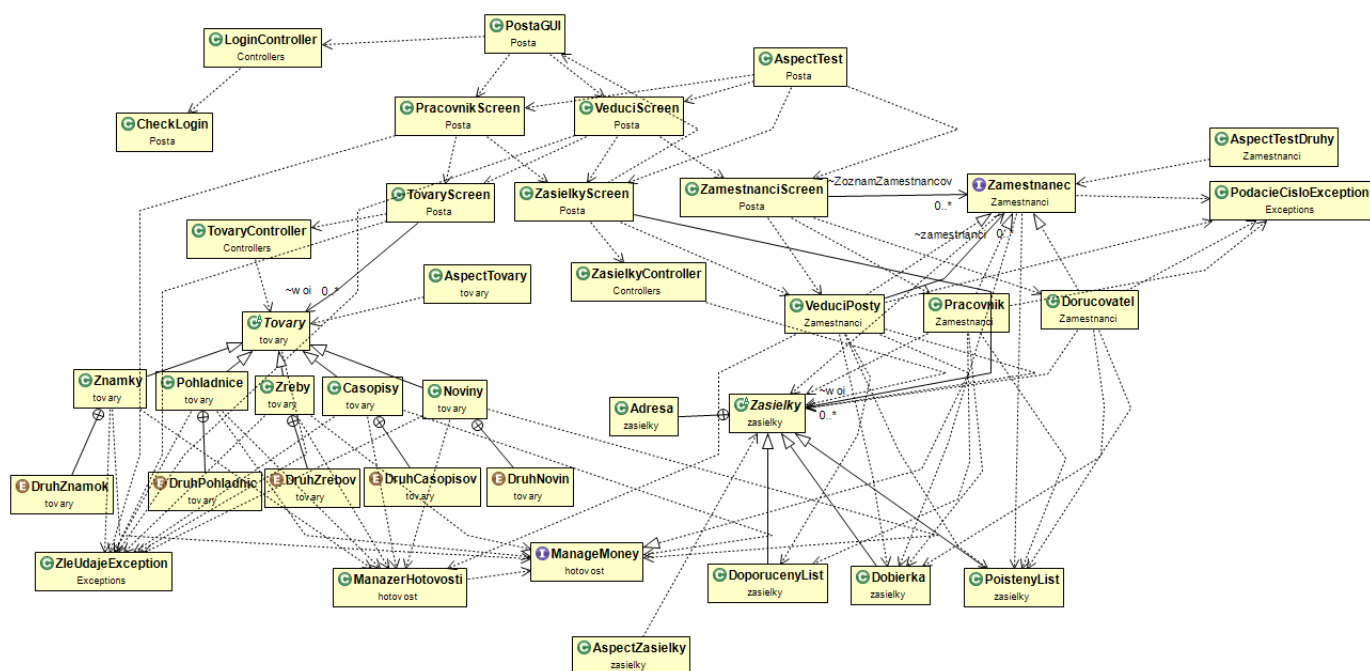


Figure 1: UML diagram tried

## Ukážky aplikácie

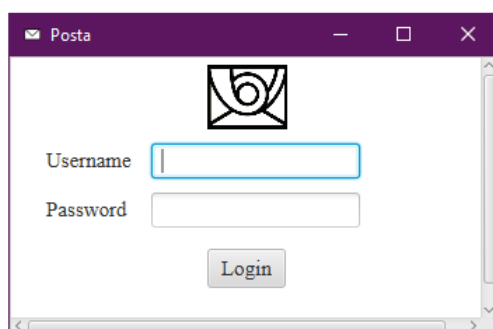


Figure 2: Prihlasovacia obrazovka

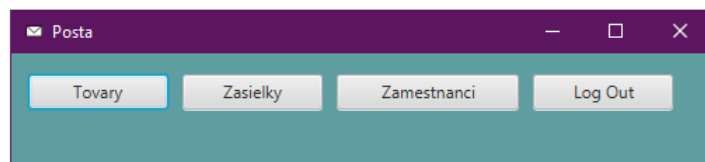


Figure 3: Menu pre vedúceho pošty

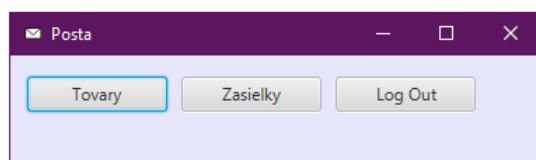


Figure 4: Menu pre pracovníka pošty

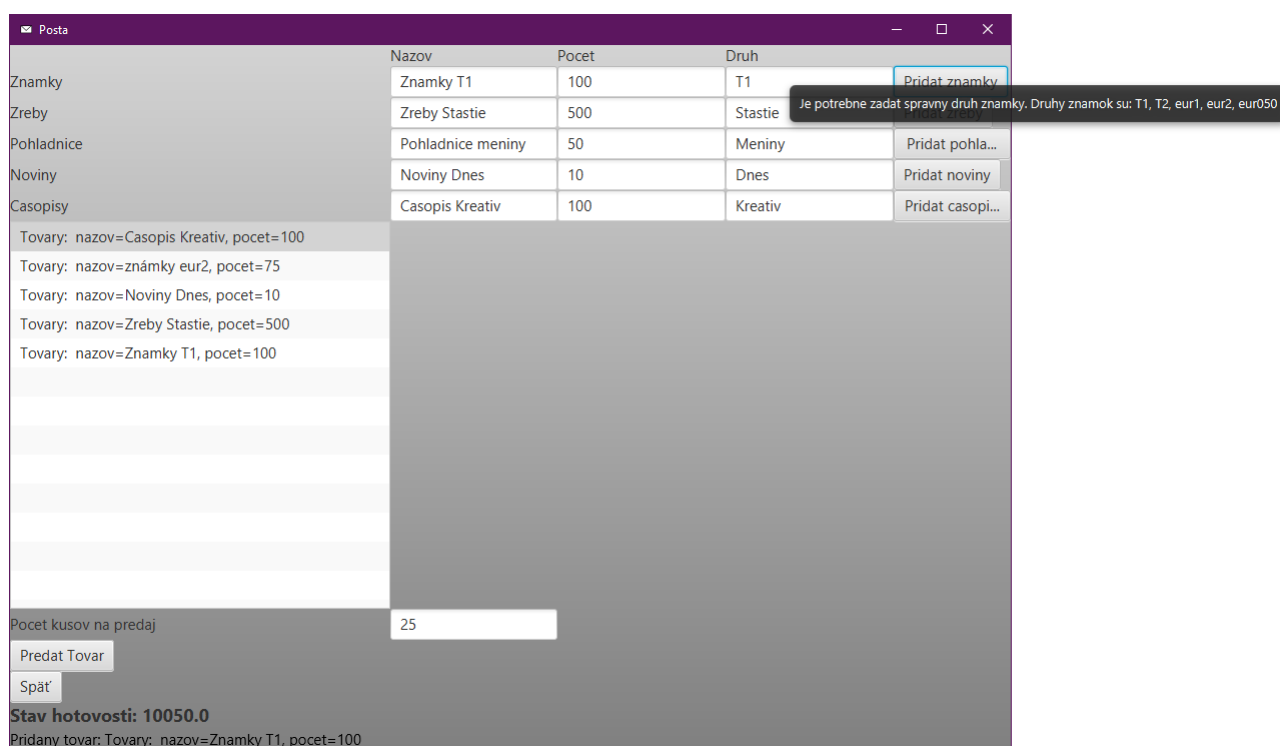


Figure 5: Tovary

**Posta**

Meno: Lucia

Priezvisko: Rapanova

Ulica: Sturova

Cislo: 10

PSC: 84104

Mesto: Bratislava

Podacie cislo: RE514799641SK

Suma:

Hmotnost:

Zapis zasielku

☒ Doporuceny List

☐ Dobierka

☐ Poisteny list

Dorucit zasiel... Spät'

Zasielky: datum =Fri May 22 21:22:59 CEST 2020, podacie cislo =RE123456


Zasielky: datum =Fri May 22 21:23:16 CEST 2020, podacie cislo =VC123456

Zasielky: datum =Fri May 22 21:23:35 CEST 2020, podacie cislo =RE123456

Figure 6: Zásielky

**Posta**

meno: Eva, priezvisko: Benkova



Dorucovatel  
Meno a priezvisko: Eva Benkova  
ID: 4579  
Mzda: 650.0

Spät' Pridel vyplatu

**Stav hotovosti: 7800.0**

Figure 7: Zamestnanci

## Kritéria hodnotenia

### Hlavné kritériá

- Použité dedenie

Dedenie bolo použité v programe v triedach Tovary a Zásielky. Tovary sú nadtrieda, od

ktorej dedia ďalšie tovary ako napríklad Noviny, časopisy, známky, pohľadnice... Od triedy Zásielky dedia trieda Doporučený list, Poistený list a Dobierka. Obe nadtriedy sú abstraktné a majú abstraktné metódy.

Listing 1: Ukážka dedenia v mojom programe

```
1 public abstract class Tovary implements java.io.Serializable {
2     ...
3     public class Znamky extends Tovary
4     public class Noviny extends Tovary
5     public class Casopisy extends Tovary
6     ...
```

### • Polymorfizmus

Polymorfizmus bol použitý práve pri týchto dvoch spomínaných triedach. Prvý polymorfizmus je použitý v metóde predatTovar v triede Tovary. Každý tovar, ktorý dedí z tejto nadtriedy, má iný spôsob predaja tovaru. Predaj tovaru závisí od druhu tovaru a ceny za tento tovar. Druhý polymorfizmus je pri triede Zásielky. Má abstraktnú metódu, ktorá kontroluje, či bolo zadané korektné podacie číslo. Pri doporučenom liste a dobierke podacie číslo začína s písmenami RE, pri poistenom liste s písmena VC.

Listing 2: Ukážka polymorfizmu pri predaji tovaru.

```
1 public class Pohladnice extends Tovary {
2     ...
3     /**
4     *kazdy tovar ma svoje druhy tovaru
5     */
6     private enum DruhPohladnic {
7         Narodeniny1("Narodeniny druh 1"),
8         Narodeniny2("Narodeniny druh 2"),
9         Vianoce("Vianoce"),
10        VelkaNoc("Velka noc"),
11        Meniny("Meniny");
12        ...
13        @Override
14        public void predatTovar(Tovary tovar, int pocet) {
15            ...
16            /**
17            * pri predaji tovaru sa kazdy druh chova inak
18            */
19            switch (tovar.getDruh()) {
20                case "Narodeniny druh 1":
21                    celkovaSuma = pocet*0.40;
22                    double result1 = manazer.add(ManazerHotovosti.↵
```

```

23         getStavHotovosti(), celkovaSuma, pridaj);
24     ManazerHotovosti.setStavHotovosti(result1);
25     break;
26     case "Narodeniny druh 2":
27         celkovaSuma = pocet*0.80;
28         double result2 = manazer.add(ManazerHotovosti.↵
29             getStavHotovosti(), celkovaSuma, pridaj);
30         ManazerHotovosti.setStavHotovosti(result2);
31         break;
32     case "Vianoce":
33         celkovaSuma = pocet*1;
34         double result3 = manazer.add(ManazerHotovosti.↵
35             getStavHotovosti(), celkovaSuma, pridaj);
36         ManazerHotovosti.setStavHotovosti(result3);
37         break;
38     ...

```

---

#### • Rozhrania

Používam rozhranie pri použití návrhového vzoru Composite a funkčné rozhranie na použitie lambda výrazu.

Listing 3: Príklad použitia rozhrania v mojom programe

```

1
2     public interface Zamestnanec {
3         ...
4         /**
5          * obsahuje metody, ako napríklad:
6          */
7         public void add(Zamestnanec zamestnanec);
8         public void Dorucit(Zasielky zasielka);
9         public Zasielky zapisZasielku(String podacieCislo, String meno, ↵
10             String priezvisko, String ulica, int cislo, int psc,
11             String Mesto) throws PodacieCisloException;
12         ...

```

---

#### • Zapúzdrenie

Zapúzdrenie bolo použité napríklad pri Tovaroch, pri adrese, pri Zasielkach, Zamestnancoch, stave hotovosti, pri prihlasovacích údajoch do aplikácie...

Listing 4: Ukážka zapúzdrenia

```

1     public class VeduciPosty implements Zamestnanec, ManageMoney {
2         private String meno;
3         private String priezvisko;

```

```

4     private int ID;
5     private double mzda;
6     private String pohlavie;
7     ...
8     public String getPohlavie() {
9         return pohlavie;
10    }
11    public void setPohlavie(String pohlavie) {
12        this.pohlavie = pohlavie;
13    }
14    @Override
15    public int getID() {
16        return ID;
17    }
18    @Override
19    public String getMeno() {
20        return meno;
21    }
22    @Override
23    public String getPriezvisko() {
24        return priezvisko;
25    }
26    ...

```

---

### • Agregácia

Najčastejšie použitie agregácie bolo to, keď nejaká iná trieda držala zoznam objektov inej triedy, napríklad Zamestanci majú zoznam objektov triedy Zásielky. Rôzne scény majú referencie na entity z iných tried.

Listing 5: Ukážka agregácie

```

1     /**
2     * Interface zamestnanec ma Array List Zasielok
3     */
4     static ArrayList<Zasielky> ar = new ArrayList<Zasielky>();
5     /**
6     * v triede VeduciPosty napríklad zapisuje zasielky do tohto zoznamu
7     */
8     ar.add(zasielka);

```

---

### • Oddelenie aplikačnej logiky od používateľského rozhrania

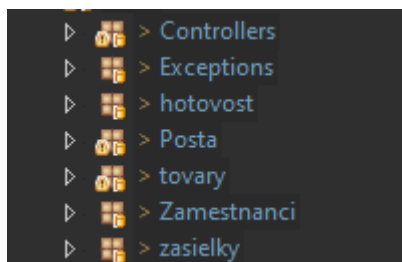
Ako používateľské rozhranie sa používa GUI. Pri scénach s GUI sa zasahuje len do zobrazovania grafických prvkov a volajú sa funkcie, ktoré majú na starosti aplikačnú logiku.

Ďalej mám napríklad triedy controllerov, ktoré spracúvajú niektoré udalosti v grafickom rozhraní,

ako napríklad serializáciu alebo vyhodnocovanie prihlasovania do aplikácia.

- **Kód organizovaný do balíkov**

Kód som rozdelila do nasledovných balíkov:



Controllers obsahujú niektoré ovládacie prvky týkajúce sa GUI. Exceptions obsahujú moje dve vlastné definované výnimky. hotovost obsahuje manažéra hotovosti a stav hotovosti v systéme. Posta obsahuje scény a grafické rozhranie. Tovary obsahujú údaje o tovaroch, Zamestnanci o zamestnancoch a zasielky o zásielkach.

- **Dokumentácia Javadoc**

Program obsahuje aj javadoc dokumentáciu, ktorá obsahuje komentáre a opisy tried a metód.

## Vedľajšie kritériá

- **Použitie návrhových vzorov**

V programe som použila návrhový vzor Composite. Znamená to, že objekty v rámci tohto vzoru budú mať nejaké spoločné správanie. Využila som to pri implementácii zamestnancov, kde interface Zamestnanci je Component, VeduciPosty je Composite a ostatní zamestnanci sú Leaves. Leaf implementuje základné správanie, Component je vždy interface pre objekty v kompozícii a Composite má leaf elementy, implementuje metódy a definuje správanie.

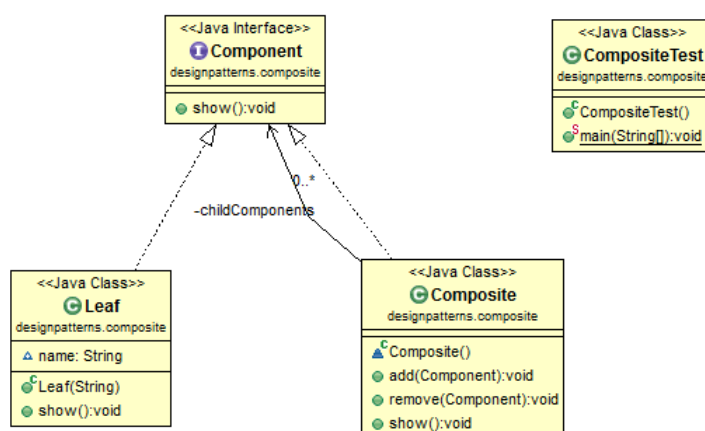


Figure 8: Návrhový vzor Composite

zdroj obrázku: <https://www.programcreek.com/2013/02/java-design-pattern-composite/>



Listing 6: Composite v mojom programe

---

```

1  /**
2  *Component
3  */
4  public interface Zamestnanec
5  /**
6  *Composite
7  */
8  public class VeduciPosty implements Zamestnanec
9  List<Zamestnanec> zamestnanci = new ArrayList<Zamestnanec>();
10 /**
11 *Listy
12 */
13 public class Pracovnik implements Zamestnanec
14 public class Dorucovatel implements Zamestnanec
15 /**
16 *Pouzitie
17 */
18 veduci1.add(dorucovatel1);
19 veduci1.add(pracovnik2);
20 veduci1.add(pracovnik1);
21 /**
22 * prideli vsetkym zamestnancom vyplatu
23 */
24 veduci1.dostanVyplatu();

```

---

#### • Vlastné výnimky

V programe som použila dve vlastné výnimky. Jednu výnimku som použila pri ošetrovaní, či bolo pri zásielkach zadané správne podacie číslo. Druhú výnimku som použila pri zisťovaní, či pri zapisovaní tovaru bol zadáný správny druh tovaru (podľa enum)

Listing 7: Ukážka vlastnej výnimky v programe

---

```

1  public class PodacieCisloException extends Exception
2  ...
3  if ((pl.CheckPodacieCislo(podacieCislo)) == false || (↵
4      CheckDuplicate(podacieCislo) == false)) {
5      throw new PodacieCisloException("Zle podacie cislo");
6  }
7  /**
8  *Pri zapise zasielok
9  */
10 catch (PodacieCisloException ex) {
11 ex.ShowAlert();
12 }

```

---

- **Grafické rozhranie oddelené od aplikačnej logiky so spracovateľom udalostí vytvorených manuálne**

Ako som už spomínala vyššie, grafické rozhranie je oddelené od používateľskej logiky. V triedach týkajúcich sa GUI prebiehajú len akcie spojené s GUI (zobrazovanie komponentov). Volajú sa tam metódy, v ktorých sa vykonáva aplikačná logika. Všetky event handlers sú vytvorené manuálne. Obsahujú aj konkrétne event handlers aj lambda výrazy.

Listing 8: Príklad oddelenia aplikačnej logiky od grafického rozhrania

```
1  buttonLogin.setOnAction(e -> {
2      /**
3      * metoda na zistenie, ktory pouzivatel sa prihlasil a ci zadal ↵
4          spravne udaje
5      */
6      if (loginController.validateUser(txtUserName.getText(), pf.getText↵
7          ()) == "pracovnik") {
8          if (pracovnikScena == null) {
9              try {
10                 /**
11                 * vybudovanie sceny
12                 */
13                 pracovnikScena = pracovnikScreen.zobrazPracovnikScreen(↵
14                     hlavneOkno, skrolScene);
15             } catch (ZleUdajeException e1) {
16                 e1.printStackTrace();
17             }
18         }
19     }
20     hlavneOkno.setScene(pracovnikScena);
21 }
```

- **RTTI**

RTTI bolo použité pri zisťovaní typu objektu a následné zobrazenie grafického prvku na základe zisteného typu

Listing 9: RTTI

```
1  if (comboBox.getValue() instanceof Dorucovatel) {
2      text.setText("Dorucovatel \nMeno a priezvisko: " + comboBox.↵
3          getValue().getMeno() + " " + comboBox.getValue().getPriezvisko↵
4          () + "\n" + "ID: " + comboBox.getValue().getID() + "\n" + "↵
5          Mzda: " + comboBox.getValue().getMzda());
6  }
```

- **Vhniezdené triedy**

Použila som jednu vhníezdenú triedu. V triede Zasielky sa nachádza vhníezdená trieda Adresa, ktorá obsahuje atribúty ako napríklad meno, priezvisko, ulica, číslo...

Listing 10: Vhniezdená trieda

```
1  public class Adresa {
2      private String meno;
3      private String priezvisko;
4      private String ulica;
5      private int cislo;
6      private int psc;
7      private String mesto;
8  }
```

- **Použitie lambda výrazov**

V programe som použila jeden vlastný lambda výraz na aktualizovanie stavu hotovosti na pošte. Použila som na to funkčné rozhranie.

Listing 11: Lambda výraz

```
1  public interface ManageMoney {
2      double pridaj(double stavHotovosti, double suma);
3  }
4  /**
5   * V triede ManazerHotovosti
6   */
7  public double add(double stav, double suma, ManageMoney manageMoney)↵
8      {
9      return manageMoney.pridaj(stav, suma);
10 }
11 /**
12 * Pri predaji tovaru
13 */
14 ManazerHotovosti manazer = new ManazerHotovosti();
15 ManageMoney pridaj = (double stavHotovosti, double suma) -> ↵
16     stavHotovosti = stavHotovosti + suma;
17 ...
18 switch (tovar.getDruh()) {
19     case "T1":
20         celkovaSuma = pocet*0.80;
21         double result1 = manazer.add(ManazerHotovosti.getStavHotovosti(),↵
22             celkovaSuma, pridaj);
23         ManazerHotovosti.setStavHotovosti(result1);
24         break;
```

```
23 ...
24 }
```

---

- **Default method implementation**

Default metódu som použila v rozhraní ManageMoney. Zariadi to, že sa na pošte objednájú peniaze. Metóda sa zavolá, ak mi napríklad nebolo dosť peňazí na vyplácanie výplat.

Listing 12: Default method

---

```
1 default void objednatPeniaze() {
2     double current = ManazerHotovosti.getStavHotovosti();
3     current += 10000;
4     ManazerHotovosti.setStavHotovosti(current);
5 }
```

---

- **Použitie aspektovo-orientovaného programovania**

V programe sa použila aspektovo-orientované programovanie na zachytenie výnimiek pri rôznych metódach. Keď vytvorím point-cut na metódu a použijem Advise, tak kdekoľvek v programe by sa metóda zavolala a zaznamela chybu, tak ju viem odchytiť pomocou aspektu.

Listing 13: Použitie aspektovo-orientovaného programovania

---

```
1 public aspect AspectTestDruhy {
2     Alert alert = new Alert(AlertType.ERROR);
3     pointcut callDorucit(): call(* Zamestnanec.Dorucit(..));
4     after() throwing (Exception e): callDorucit() {
5         alert.setContentText("Chyba: " + e + "\n" + "Nebola zvolena ←
6             ziadna zasielka na dorucenie");
7         alert.show();
8         System.out.println("Vynimka: " + e + "\n" + "Nebola zvolena ←
9             ziadna zasielka na dorucenie");
10    }
11 }
```

---

- **Serializácia**

Serializáciu som použila na ukladanie zásielok a tovarov do súboru. Metódy na ukladanie resp. mazanie sa nachádzajú v TovaryController a ZasielkyController.

Listing 14: Príklad serializácie - načítanie tovaru zo súboru

---

```
1 /**
```

```

2  * Metoda nacita tovar zo suboru do arraylistu
3  * @param woi je arraylist, do ktoreho sa uklada zoznam tovarov
4  * @return woi aktualizovany arraylist tovarov
5  */
6  public ArrayList<Tovary> nacistajTovar(ArrayList<Tovary> woi) {
7      try {
8          FileInputStream fis = new FileInputStream("serializacia\\↵
          tovary.ser");
9          ObjectInputStream ois = new ObjectInputStream(fis);
10         /**
11         * Zoznam tovarov je ulozeny v Arrayliste, ktory je ulozeny v ↵
          subore
12         */
13         woi = (ArrayList<Tovary>) ois.readObject();
14     } catch (IOException i) {
15         System.out.println("chyba pri nacistani suboru" + i);
16     } catch (ClassNotFoundException e1) {
17         e1.printStackTrace();
18     }
19     return woi;
20 }

```

---

## Zoznam hlavných verzií a najdôležitejších zmien

Najhlavnejšie verzie predstavujú tieto zmeny:

- Commit 20.3. metoda na zapisanie zasielok  
Zapisovanie zasielok bola jedna z prvých a základných funkcionalít aplikácie.
- Commit 31. 3. pridane graficke rozhranie a vstup z klavesnice  
Bolo pridané grafické používateľské rozhranie cez JavaFX a vstup z klávesnice od používateľa.
- Commit 3.4 zmena sceny, radio buttons, exceptions  
Vzniklo prepínanie scén, radio buttons na výber zasielok a prvé pridané výnimky
- Commit 3.4. composite pattern na pridelenie vyplat  
Použitý návrhový vzor Composite. Použitie jedného vzoru bolo jedným z vedľajších kritérií.
- Commit 9.4. pridany log in  
Na hlavnú obrazovku pribudol login a jeden používateľ, ktorý sa mohol prihlásiť.
- Commit 11.4. uprava enum na stringy a pridanie tovary v scene Tovary  
Vznikla nová scéna pre tovary, enum typ bol upravený tak, aby pri zapisovaní tovaru bol použitý konkrétny druh tovaru.

- Commit 9.5. zamestnanci screen

Vznikla nová scéna pre zamestnancov.

- Commit 10.5. serializacia doporučene listy

Vytvorenie serializácie zásielok a neskôr aj tovaru.

- Commit 17.5. pridane aspekty

Do programu bolo pridané aspektovo-orientované programovanie a aplikácia sú spúšťa ako AspectJ aplikácia.

- Commit 22.5. vytvorenie tovary controllera, vytvorenie zasielky controllera

Oddelenie aplikačnej logiky od GUI, vytvorenie controllerov pre ďalšie scény.

## **Zdroje obrázkov a ikon, ktoré boli použité v programe**

- Zdroj ikon zamestnancov: <https://www.freepik.com/free-photos-vectors/people>
- Zdroj ikony aplikácie: <http://www.clker.com/clipart-post-logo.html>
- Zdroj loga na Login obrazovke: <https://www.posta.sk/stranky/loga-a-piktogramy-na-stiahnutie>