

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Dispečing MHD
Objektovo-orientované programovanie

Filip Čaplák

Cvičiaci: Ing. Anna Považanová

Cvičenie: Streda 14:00

24. máj 2020

1. Zámer projektu

Projekt sa nazýva Dispečing MHD, a hlavným cieľom je simulovať kontrolu nad mestskou hromadnou dopravou. V programe vystupujú Zamestnanci ako Vodiči, Dispečeri a Kontrolný dispečeri, ktorí majú za úlohu pokutovať priestupky u vodičov. Program funguje tak, že do predpripravených liniek najprv dispečeri priradia vodičov s vozidlami, a potom vodiči môžu danú linku obsluhovať posuvom po zástavkách. Vodič môže byť skúšobný, obyčajný alebo profesionálny. Každý vodič má iné percento šance na vykonanie priestupku akým je meškanie zo zastávky alebo vysoká rýchlosť. Ak bol tento priestupok zaznamenaný, vodič nie je pokutovaný hneď, ale túto prácu musí spraviť kontrolný dispečer, ktorý udeľuje pokuty. Môžeme teda hrať za 3 rozdielne typy postáv v simulácii.

2. Porovnanie oproti verzii, ktorú som prezentoval

Vo verzii, ktorú som prezentoval mi chýbali viaceré kritériá. Mal som hotové len GUI a RTTI a štandardnú (nie vlastnú) výnimku. Nemal som napríklad lambda výraz, rozhranie... Oproti prezentovanej verzii a finálnej som ale všetko chýbajúce doplnil. Nižšie je zoznam kritérií, ktoré som splnil vo svojom programe.

3. Splnené kritéria hodnotenia

Enkapsulácia (zapuzdrenie)

V triedach sú atribúty prístupového typu protected a boli k nim zriadené gettery a settery, cez ktoré ich používajú iné triedy.

Dedenie

V našom programe sú dve hierarchie dedenia v ktorých je na vrchu trieda Clovek a Vozidlo. Konkrétne v nadtriede Clovek je použité viacúrovňové dedenie.

Agregácia

Použitá viackrát - napríklad Linka používa Zastávky, Vodic Vozidlo...

Asociácia

Vodic používa Vozidlo a zase Vozidlo používa Vodiča - obojsmerná asociácia

Lambda výrazy

V programe sa použil lambda výraz na odstránenie záznamu o priestupku z hašovacej mapy ,priestupky‘.

```
public static void removePriestupok(int idVodica)
{
    priestupky.entrySet().removeIf(entry -> entry.getKey() == idVodica);
}
```

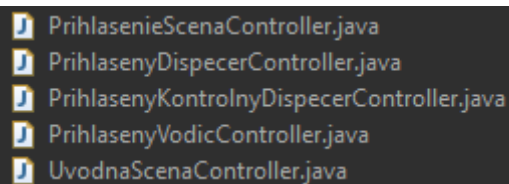
RTTI - Run-Time Type Information

Použil som pri zisťovaní typu objektu pri prihlasovacej obrazovke - užívateľ sa autentifikoval kódom, ale nevieme dopredu na akú obrazovku ho máme prepnúť a tak využijeme **instanceof** aby sme zistili, aký typ objektu je prihlásený používateľ a na základe toho ho prepne na príslušnú scénu.

```
if( prihlaseny instanceof Vodica )
{
    try
    {
        Main.scenaPrihlasenyVodic();
    }
    catch (Exception e)
    {
        invalid.setText("Vodičovi nebolo priradené vozidlo!");
    }
}
else if( prihlaseny instanceof KontrolnyDispecer )
    Main.scenaPrihlasenyKontrolnyDispecer();
else if( prihlaseny instanceof Dispecer )
    Main.scenaPrihlasenyDispecer();
```

GUI

Grafické používateľské rozhranie je implementované pomocou JavaFX, ktorý som do prostredia Eclipse IDE doinštaloval. Vytvorených bolo 5 scén, kde každá má svoj FXML súbor, ktorý obsahuje informácie o rozmiestnení jednotlivých položiek ako sú texty, tlačítka... Tieto FXML súbory sa dajú nazvať ako View nášho programu, a triedy, ktoré sa starajú o logiku týchto scén sú Controllers:



```
PrihlasenieScenaController.java
PrihlasenyDispecerController.java
PrihlasenyKontrolnyDispecerController.java
PrihlasenyVodicController.java
UvodnaScenaController.java
```

Touto organizáciou sa nám črtá MVC návrhový vzor.

Event handlers

Použili sa v grafickom rozhraní na zachytávanie stisku tlačítok, napríklad na tlačítko odhlásenia:

```
@FXML
private void btnExit(ActionEvent event) throws IOException
{
    Platform.exit();
    System.exit(0);
}
```

Vlastná výnimka

Používame ju pri pokuse o autentifikáciu - v obrazovke PrihlasenieScena je vyhadzovaná v prípade, že stlačíme tlačítko prihlásiť a v textovom poli a nachádza znak, ktorý nie je číslo. Pomáha nám k tom regulárny výraz. Zdedenú triedu výnimky sme implementovali takto:

```
public class VynimkaException extends Exception
{
    public VynimkaException(String msg)
    {
        super(msg);
    }
    public void callMethod()
    {
        System.out.println(super.getMessage());
    }
}
```

A v programe ju vyhadzujeme:

```
public void checkOnlyNumbers() throws VynimkaException
{
    String regex = ".*[a-zA-Z]+.*";
    Pattern pattern = Pattern.compile(regex);

    if(pattern.matcher(pwKod.getText()).matches())
        throw new VynimkaException("Kod moze obsahovat len cisla!");
}
```

Kde checkOnlyNumbers() sa volá už priamo v try-catch bloku handlera tlačítka.

Dve hierarchie dedenia s polymorfizmom

Trieda Vozidlo je abstraktná a obsahuje abstraktnú metódu:

```
public abstract void skontrolujPohanac();
```

Ktorá je následne prekonaná v podtriedach Autobus, Trolejbus, Elektricka vždy s iným volaním a s inou zmenou atribúty príslušného pohonu vozidla.

Trieda Zamestnanec je abstraktná a obsahuje abstraktnú metódu:

```
public abstract void skratPrestavku();
```

Je prekonaná v podtriedach Vodic, SkusobnyVodic, ProfesionalnyVodic, kde každý druh vodiča má svoj čas na ktorý sa im skráti prestávka.

Vnorená trieda

Do triedy Podnik sme vnorili triedu Autentifikator, pretože som ju považoval za súčasť Podniku, keďže autentifikuje zamestnancov daného Podniku. V tomto som našiel zmysel. Prístupujeme k nej notáciou *Podnik.Autentifikator*.

Rozhranie s návrhovým vzorom Strategy

Použil som rozhranie ktoré som nazval Strategy, v ktorom metóda priradzuje druh vozidla cestujúcemu, ktorý prišiel ako argument. Vzor strategy spúšťa metódu na základe kontextu. O tom, aká metóda sa má spustiť sa rozhoduje až počas behu programu, kedy je známe že cestujúci chce nastúpiť do určitého druhu vozidla.

```
public interface Strategy
{
    public String nastupDoVozidla(Cestujuci cestujuci);
}
```

Serializácia

Po vytvorení liniek a ich naplnením zastávkami ich serializujeme (uložíme) do súboru linkaX.ser, kde X značí číslo linky. V našom programe sú linky tvorené staticky. Týmto sme dostali uložené objekty liniek so zastávkami zvlášť do súborov.

4. Zoznam hlavných verzií programu v GitHub repozitári

V GitHub repozitári sa nachádzajú 3 verzie programu, ktoré predstavujú každé jedno odovzdanie projektu. Priečink ,PracovnaVerzia' obsahuje stav projektu, kde ešte nebolo grafické rozhranie ale bola hotová logika programu na úrovni konzolového vstupu/výstupu. Ďalší priečinok je ,GUIVerzia', ktorá bola odprezentovaná pred finálnym odovzdaním. Nachádza sa tu kompletne grafické rozhranie, ale chýbajú ešte kritéria ako oproti finálnemu. Je tu len RTTI, 2 hierarchie dedenia. Tretí priečinok sa nazýva ,FinalVersion', čo predstavuje finálnu, kompletne dokončenú verziu pripravenú na záverečné hodnotenie projektu. Táto dokumentácia opisuje finálnu verziu nachádzajúcu sa v spomínanom priečinku v repozitári.

5. Záver

Prácou na tomto projekte som si osvojil objektovo-orientované princípy a naučil sa pracovať s používateľským grafickým rozhraním, ktoré bolo pre mňa novinkou.