Róbert Junas
102970
12.4.2021

Lagatoria

1. Názov projektu je Lagatoria. Hlavnou úlohou projektu je vydávanie kníh, teda to zahŕňa napísanie textu, jeho korektúru, návrh obálky, zistenie aký je dopyt po knihe, vytlačenie kníh s tým spojené rozhodovanie sa, koľko kusov sa vytlačí a následná distribúcia výtlačkov. Ďalej sa knihy predávajú v kníhkupectve, kde si ich môže kúpiť zákazník.

2. Program je z veľkej časti funkčný. Funguje pridávanie a odoberanie ako autorov, tak aj odoberateľov, Používateľské rozhranie je rozdelené podľa vzoru MVC . Funguje vydávanie kníh ako aj ich distribúcia medzi odoberateľov. Dokážeme zmeniť stratégiu ako sa knihy vydávajú. Čo sa týka kníhkupectva, tak dokážeme objednať knihy. Premiestňovať ich do rôznych sekcií a pod.

Hlavný súbor s metódou main() je junas.robert.lagatoria.gui.Main.java

Parametre na spustenie nie sú žiadne, treba mať povolene javaFx

Pravdepodobne je potrebné si vytvoriť priečinok /res/, kde sa uloží serializacia

3. Aspoň **3 vrstvovú hierarchiu dedenia** nám tvorí dedenie používateľov. Na vrchu je interface lagatoria.utils.InputProcess ten je implementovaný abstraktnou triedou lagatoria.users.Pouzivatel, Pouzivatel je dedený triedou Zakaznik a abstraktnou triedou Zamestanec. Nakoniec triedy Skladnik, Predajca, Distributor, Manazer a aj vnorené triedy Korektor, Dizajner (v triede Vydavatelstvo.java) dedia triedu Zamestnanec.

```
public interface InputProcess {
```

```
    public abstract class Pouzivatel implements InputProcess
```

```
        public class Zakaznik extends Pouzivatel
```

```
        public abstract class Zamestnanec extends Pouzivatel{
```

```
            public class Predajca extends Zamestnanec
```

```
            public class Skladnik extends Zamestnanec {
```

```
            public class Manazer extends Zamestnanec {
```

```
            public class Distributor extends Zamestnanec {
```

4. Polymorfizmus je napríklad v tom ako sa spracovávajú vstupy v triede Pouzivatel.java sa definuje spôsob akým sa spracováva vstup a následne Zamestnanec.java prekonáva funkciu tým že s ňou ešte volá pridajHodinu(), čo pridá zamestnancovi jednu odrobenú hodinu, za použitie metódy.

```java
@Override //Pouzivatel.java
    public String spracuj(String[] args, Vydavatelstvo vydavatelstvo){
        String res = "";
        int index = 0;
            //prejde vsetky slova zadane na vstupe
            while(index < args.length) {
                //ak sa slovo nachadza v zavolatelnych funkciach
                if (inlineAkcie.containsKey(args[index])) {
                    //zavola sa funkcia
                    res += inlineAkcie.get(args[index]).process(args, Knihkupectvo.getInstance(), vydavatelstvo) + "\n";
                }
                int k;
                //najde dalsi prikaz rozdeleny |
                for (k = index + 1; k < args.length; k++) {
                    if (args[k].equals("|")) break;
                }
                index = k + 1;
            }
            return res;
    }
```

```java
@Override //Zamestnanec.java
    public String spracuj(String[] args, Vydavatelstvo vydavatelstvo){
        String res = super.spracuj(args, vydavatelstvo);
        pridajHodinu();
        return res;
    }
```

```java
\\Model.java
public class Model {
    private Pouzivatel pouzivatel;
    private Zakaznik zakaznik = new Zakaznik();
    private Predajca predajca = new Predajca("Predavac",22);
    private Skladnik skladnik = new Skladnik("Skladnik", 23);
    private Manazer manazer = new Manazer("Manazer",111,22.4);
    private Distributor distributor = new Distributor("Distributor",123,15);
...
public String spracuj(String command){
        return pouzivatel.spracuj(command.split(" "),vydavatelstvo);
    }
...
public void changeUser(LoggedIn pouzivatel){
        switch (pouzivatel) {
            case SKLADNIK:
                this.pouzivatel = skladnik;
                break;
            case PREDAJCA:
                this.pouzivatel = predajca;
                break;
            case ZAKAZNIK:
                this.pouzivatel = zakaznik;
                break;
            case MANAZER:
                this.pouzivatel = manazer;
                break;
            case DISTRI:
                this.pouzivatel = distributor;
        }
    }
...
}
```

Obrázok 1 Volanie funkcie spracuj z premennej pouzivatel

Ďalší príklad:

```
public interface Odoberatel {
    default double zaplatVydavatelovi(Kniha kniha, int pocet){
        return kniha.getCena()*0.77*pocet;
    };
}
//prekonane v Knihkupectvo.java
@Override
public double zaplatVydavatelovi(Kniha kniha, int pocet) {
     return kniha.getCena()*0.50*pocet;
}
```

Obrázok 2 Override funkcie v Knihkupetvo.java

5. Napríklad trieda Vydavatelstvo.java v sebe agreguje Manazer.java. Knihy sa agregujú v regáloch v prípade Knihkupectva. Ďalej v sebe Manazer agreguje autorov, ktorý sú pripravený písať.

```
public class Manazer extends Zamestnanec {
    ArrayList<Autor> autori = new ArrayList<Autor>();
    ...
    public void pridajAutora(ArrayList<Autor> autori){
        this.autori = (ArrayList<Autor>) autori.clone();
    }

    public void pridajAutora(Autor autor) throws AutorExistujeException {
        if(autori.contains(autor))
            throw new AutorExistujeException(autor.getMeno());
        else
            this.autori.add(autor);
    }

    public void odoberAutora(Autor autor) throws AutorNieJeNaZozname {
        if (!autori.contains(autor)) throw new AutorNieJeNaZozname(autor.getMeno());

        autori.remove(autor);
    }
    ...
}
```

Ďalší príklad:

```
public class Vydavatelstvo {

    private Korektor korektor;
    private Dizajner dizajner;
    private Manazer manazer;
    private Distributor distributor;
...
}
```

6. Používateľské rozhranie je definovane v triede View.java a aplikačná logika sa kumuluje v Model.java a Controller.java je iba na komunikáciu medzi nimi. Nakoniec sa všetky časti prepájajú v Main.java.

Róbert Junas
102970 12.4.2021

```
public void start(Stage stage) {
    Model model = new Model();
    Controller controller = new Controller(model);
    View view = new View(controller,model);
    controller.setView(view);
```

7. Máme navyše implementovanú serilizáciu na strane kníhkupectva, kde si ukladáme stav všetkých poličiek a kníh v kníhkupectve súbor s dátami ukladáme do zložky res/.

Ďalej máme implementovaný návrhový vzor observer medzi Manazer a Autor. Ďalším implementovaným návrhovým vzorom je Composite a je spojený s vytváraním kníh. Ďalej sme implementovali model Visitor a to pri implementácií Autora. Ako posledný sme implantovali Strategy vo Vydavatelstvo.java, kde sa mení metóda vydávania kníh.

Implementované máme aj vlastné výnimky: AutorExistujeException, AutorNieJeNaZozname, InvalidFormatException. Nachádzajú sa v core.utils.exceptions.

Využili sme aj multithreading pri autoroch, ktorý dokážu naraz písať knihy.

Explicitne sme využili RTTI, pri zisťovaní kam sa máju knihy a ako poslať.

Máme implementované aj vnorene triedy vyhniezdené triedy a rozhranie vo Vydavatelstvo.java.

Máme implementovanú aj default metódu na platenie vydavateľstvu v rozhraní Odoberatel.java