SLOVENSKÁ TECHNICKÁ UNIVERZITA

Fakulta informatiky a informačných technológií

v Bratislave

Objektovo orientované programovanie

Dokumentácia k semestrálnemu projektu

Adam Tomčala

Prednášajúci: doc. Ing. Valentino Vranič, Phd.

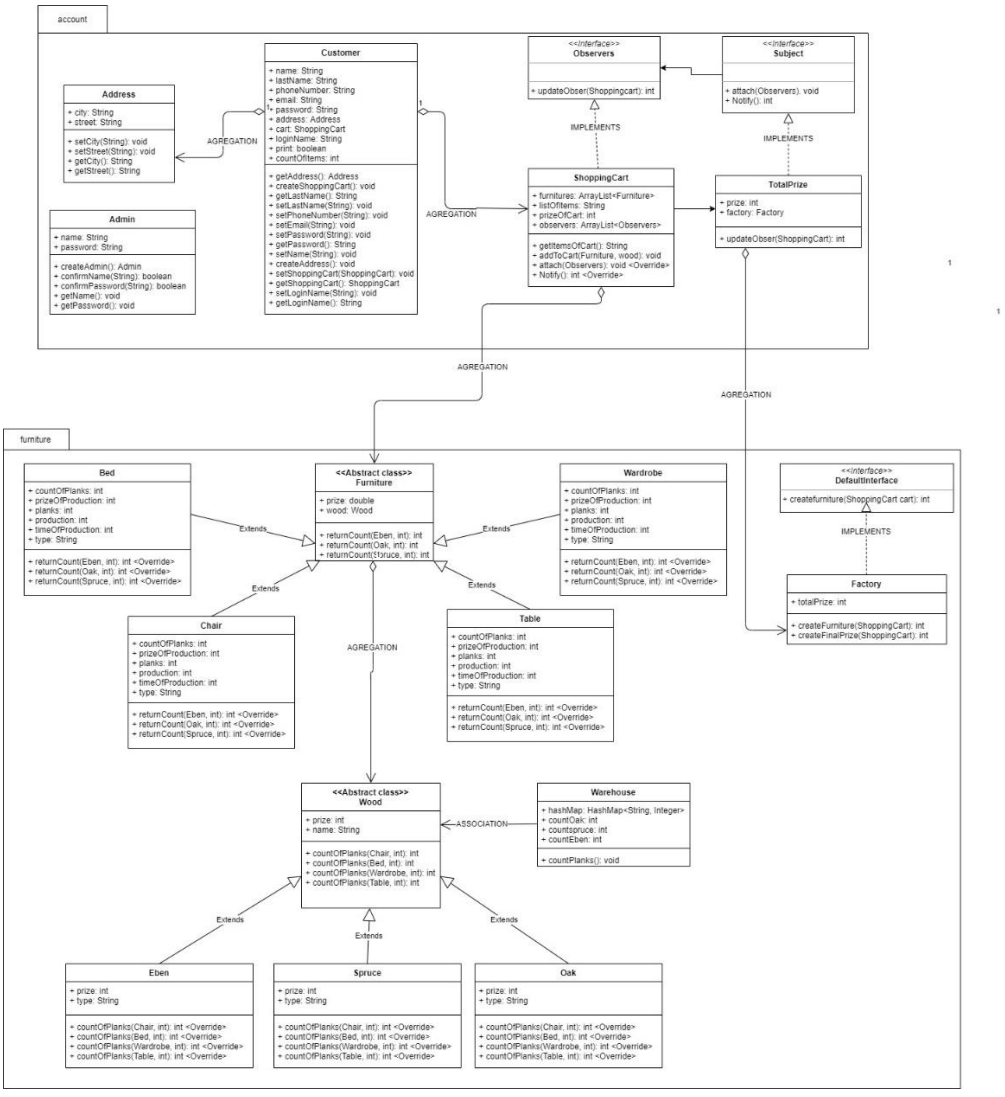
Cvičiaci: Mgr. Pavle Dakič

Čas cvičení: Utorok 16:00

Zámer projektu:

- Moja aplikáciu (WoodFiiT) je zameraná na predaj a výrobu rôznych druhov nábytku z rôznych druhov dreva. Používateľ bude môcť v aplikácií pokračovať buď ako zákazník alebo ako administrátor. Ak bude používateľ pokračovať ako zákazník, bude si môcť vybrať a nakúpiť tovar z aktuálnej ponuky. Po dokončení nákupu musí zákazník vyplniť informácie o sebe do textových polí. Po vyplnení údajov bude objednávka hotová a akceptovaná.
- Ak sa používateľ bude chcieť prihlásiť ako administrátor, bude musieť správne vyplniť prihlasovacie údaje (meno = „admin“, heslo = „12345“). Po správnom zadaní prihlasovacích údajov sa zobrazia všetky doteraz vytvorené objednávky. Ďalšiu funkcionalitu, ktorú môže admin vykonať je obnovenie množstva tovaru v sklade.

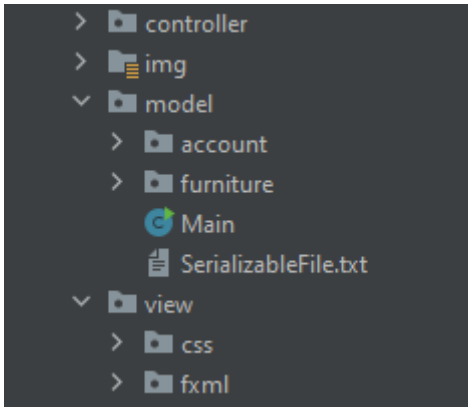
Štruktúra projektu:



- Tento diagram tried (UML) obsahuje všetky triedy logickej časti. Triedy obsahujú všetky atribúty, metódy a vzťahy medzi sebou.

Rozdelenie triedy do balíčkov:

- Pri rozdeľovaní tried do balíčkov som uplatnil návrhový vzor MVC (Model-View-Controller).
- Model - predstavuje logickú časť mojej aplikácie
- View – predstavuje vizualizáciu dát
- Controller – predstavuje časť, ktorá zachytáva informácie z view a vykonáva metódy z model-u.



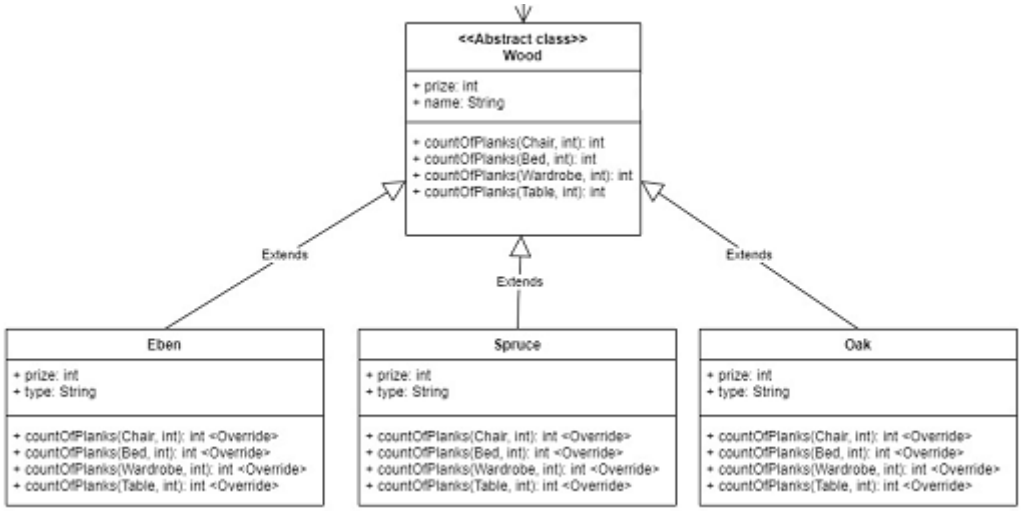
- Balíček „**controller**“ obsahuje triedy (controller-y) k jednotlivým scénam apliukácie.
- Balíček „**img**“ obsahuje všetky použité obrázky v aplikácií
- Balíček „**Model**“ obsahuje balíčky „**account**“ (balíček, v ktorom sú všetky triedy spojené s používateľom) a „**furniture**“ (balíček, v ktorom sú všetky triedy spojené s produktami, ktoré sa dajú zakúpiť). Balíček obsahuje aj triedu „**main**“ a textový súbor, ktorý slúži na serializáciu.
- Balíček „**view**“ obsahuje balíčky „**css**“ (balíček, v ktorom sú uložené css súbory) a balíček „**fxml**“ (balíček, ktorý obsahuje fxml súbory, ktoré sú vytvorené v SceneBuilder-i).

OOP princípy:

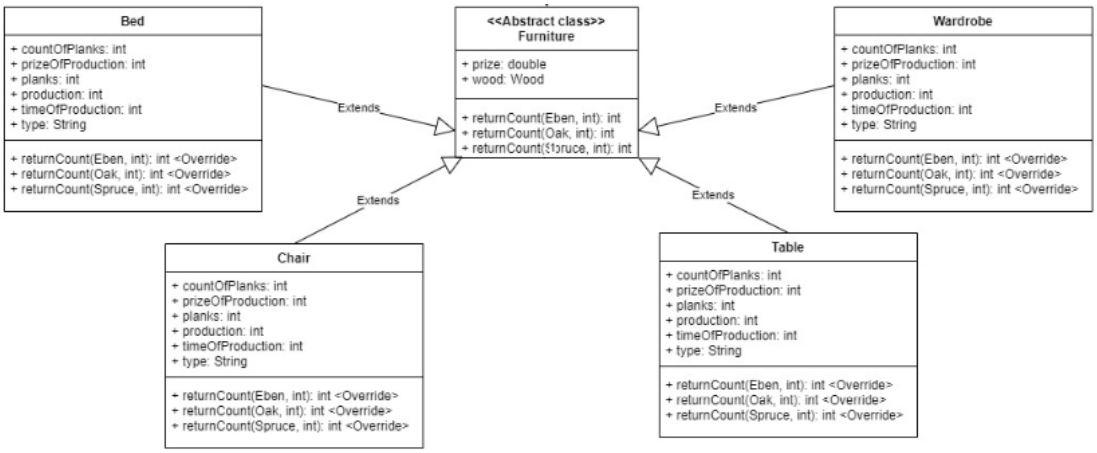
- V mojom projekte som použil nasledujúce OOP princípy:

Dedenie:

- Dedenie som použil pri triedach „**Eben**“, „**Oak**“ a „**Spruce**“. Tieto triedy predstavujú jednotlivé druhy dreva, z ktorého môže byť nábytok vyrobený. Dedia od abstraktnej triedy „**Wood**“. Zdedené triedy prekonávajú metódy rodičovskej triedy.

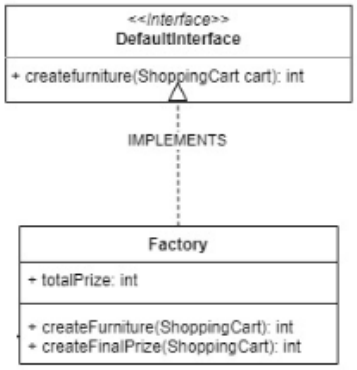


- Ďalšie dedenie som použil pri triedach „**Bed**“, „**Chair**“, „**Table**“ a „**Wardrobe**“. Tieto triedy predstavujú jednotlivé druhy nábytkov, ktoré si bude môcť zákazník zakúpiť. Dedia od abstraktnej triedy „**Furniture**“. Zdedené triedy prekonávajú metódy rodičovskej triedy.



Interface:

- V mojom projekte používam niekoľko interface – ov.
- Trieda „**Factory**“ implementuje rozhranie „**Default interface**“. Toto rozhranie obsahuje **default** metódu, ktorú následne volám práve v triede „**Factory**“.



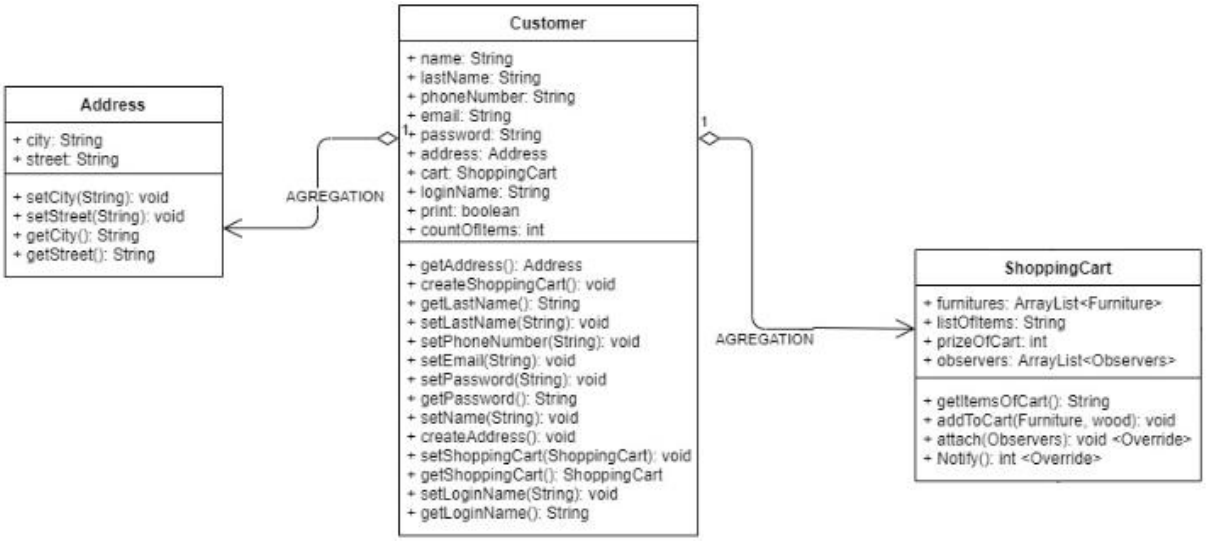
- V triede „**ShoppingCart**“ používam interface „**FunctionalInterface**“, kde implementuje metódu tohto rozhranie práve v triede „**ShoppingCart**“.

```java
@java.lang.FunctionalInterface

/**
  Functional interface that implements only one function that shows
  the total prize of customer's purchase.
*/
public interface FunctionalInterface {
    String showPrize();
}
```

```java
public String getItemsOfCart() {
    FunctionalInterface f = () -> {

return f.showPrize();
```

- Ďalšie rozhrania používam v návrhovom vzory **Observer** (komentár bude pri opise návrhové vzoru).

Agregácia:



- Agregáciu využívam v triede „**Customer**“, pretože v tejto triede sa vytváram objekt triedy „**Address**“ a objekt triedy „**ShoppingCart**“.

Zapuzdrenie:

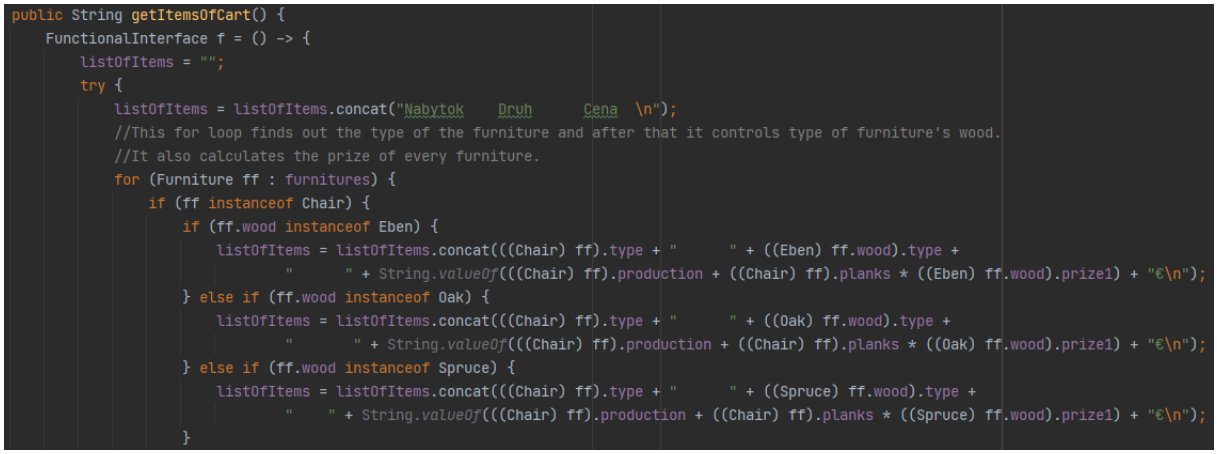
- Zapuzdrenie som použil v triede „**Admin**“, kde mám všetky atribúty triedy nastavené na **private** a pristupujem k nim pomocou getterov a setterov. Zapuzdrenie som použil aj v triede „**Customer**“, kde mám tiež atribúty triedy nastavené na private.

```
public class Customer implements Serializable{
    //information about customer
    private String name;
    private String LastName;
    private transient String phoneNumber;
    private transient String email;
    private String password;
    private Address address;
    private ShoppingCart cart;
    private String loginName;
```

- Prístup k atribútom riešim pomocou getterov a setterov.

Polymorfizmus:

- Polymorfizmus využívam napríklad v triedach „**ShoppingCart**“ alebo „**Factory**“.



```
public String getItemsOfCart() {
    FunctionalInterface f = () -> {
        listOfItems = "";
        try {
            listOfItems = listOfItems.concat("Nabytok    Druh      Cena  \n");
            //This for loop finds out the type of the furniture and after that it controls type of furniture's wood.
            //It also calculates the prize of every furniture.
            for (Furniture ff : furnitures) {
                if (ff instanceof Chair) {
                    if (ff.wood instanceof Eben) {
                        listOfItems = listOfItems.concat(((Chair) ff).type + "      " + ((Eben) ff.wood).type +
                                "      " + String.valueOf(((Chair) ff).production + ((Chair) ff).planks * ((Eben) ff.wood).prize1) + "€\n");
                    } else if (ff.wood instanceof Oak) {
                        listOfItems = listOfItems.concat(((Chair) ff).type + "      " + ((Oak) ff.wood).type +
                                "      " + String.valueOf(((Chair) ff).production + ((Chair) ff).planks * ((Oak) ff.wood).prize1) + "€\n");
                    } else if (ff.wood instanceof Spruce) {
                        listOfItems = listOfItems.concat(((Chair) ff).type + "      " + ((Spruce) ff.wood).type +
                                "    " + String.valueOf(((Chair) ff).production + ((Chair) ff).planks * ((Spruce) ff.wood).prize1) + "€\n");
                    }
```

- Screenshoot z triedy „**ShoppingCart**“ a metódy getItemsOfCart. V tejto metóde prechádzam celým ArrayList-om „furnitures“. Podľa toho, aký druh nábytku to bude, precastujem daný objekt a zoberiem si z neho potrebné informácie (každý druh nábytku má rôznu cenu, rôzny čas produkcie, atď.). Ďalej je dôležitý druh dreva, z ktorého je daný nábytok vyrobený, pretože každý druh dreva má rôznu cenu.

Splnené kritéria:

Návrhový vzor:

- V mojom projekte som použil návrhový vzor **Observer**. Návrhový vzor využíva dve rozhrania: „**Observers**“ a „**Subject**“. Rozhranie „Subject“ implementuje trieda „ShoppingCart“ a rozhranie „Observers“ implementuje trieda „Totalprize“, ktorá ešte aj dedí od triedy „**TextArea**“.
- Trieda „ShoppingCart“ je vo forme pozorovaného objektu. Vždy keď zákazník dokončí svoj nákup, zobrazí sa jeho nákupný košík aj s celkovou cenou tovaru. Na zobrazenie celkovej sumy tovaru slúži metóda notify, ktorá oboznámi všetkých pozorovateľov (v tomto prípade len trieda TotalPrize, (Text Area, ktoré slúži na zobrazenie celkovej sumy používateľovi)).

```java
@Override
public int Notify(){
    int number = 0;
    for(Observers o : observers){
        number = o.updateObser( cart: this);
    }
    return number;
}
```

- Funkcia notify oboznámi všetkých pozorovateľov a zavolá sa metóda updateObser.

```java
public int updateObser(ShoppingCart cart)  {
    this.prize = factory.createFinalPrize(cart);
    return this.prize;
}
```

- V metóde updateObser sa zavolá metóda na vypočítanie celkovej ceny nákupného košíka zákazníka.

- Okrem návrhového vzoru Observer som použil aj návrhový vzor **Singleton** v triede „Admin“, kde vytváram len jednu inštanciu triedy „Admin“.

```java
private Admin(){
    this.name = "admin";
    this.password = "12345";
}

/**
 * Creating admin like a singleton.
 * @return  New instance of Admin (Singleton).
 */
public static Admin createAdmin(){
    if(admin == null){
        admin = new Admin();
    }
    return admin;
}
```

Vlastné výnimky:

- V projekte som si vytvoril tri vlastné výnimky:
- „**EmptyFieldException**“ -> výnimka, ktorá ošetruje stav, ak zákazník nevyplní všetky potrebné TextField-y, ktoré poskytujú informácie o ňom.
- „**NotSamePasswordException**“ -> výnimka, ktorá ošetruje stav, ak zákazník nezadá rovnaké heslá pri dokončovaní objednávky.

- „**UsedName**“ -> výnimka, ktorá ošetruje stav, ak si zákazník zvolí prihlasovacie meno, ktoré je už používané.

Oddelenie GUI od aplikačnej logiky:

- Aplikačnú logiku mám v balíčku „model“ a GUI mám v balíčku „view“.

Handlers:

- Manuálne vytvorených spracovateľov udalostí (handlers) sa použil v triede „ControllerAdmin“, kde som vytvoril manuálne alert box, ktorý sa zobrazí vtedy, ak používateľ zadá nesprávne prihlasovacie meno.

```
} else {
    //using my own handlers for creating a alert box if the name is not correct.
    class Allert extends Application {
        @Override
        public void start(Stage primaryStage){
            primaryStage.setResizable(false);
            AnchorPane pane = new AnchorPane();
            Button button = new Button( text: "Späť");
            button.setLayoutX(125);
            button.setLayoutY(100);
            //handler
            button.setOnAction(event -> primaryStage.close() );

            Label label = new Label( text: "Zadali ste nesprávne meno.");
            label.setFont(new Font( name: "Arial",  size: 20));
            label.setLayoutX(30);
            label.setLayoutY(45);

            pane.getChildren().addAll(label,button);

            Scene scene = new Scene(pane,  width: 300,  height: 200);
            primaryStage.setScene(scene);
            primaryStage.showAndWait();
        }
    }
    Allert allert = new Allert();
    allert.start(new Stage());
}
```

RTTI:

- V projekte som použil RTTI v triedach „**ShoppingCart**“ a „**Factory**“, kde sa pomocou kľúčového slova **instanceof** rozhodne s akou triedou sa bude pracovať. Rozhodne sa medzi triedami, ktoré dedia od triedy „Furniture“ a triedami, ktoré dedia od triedy „Wood“, pretože každý druh nábytku má každý druh dreva má iné vlastnosti.

Vhniezdená trieda:

- Vhniezdnenú triedu som použil v triede „**ControllerAdmin**“, kde pomocou vhniezdenej triedy vytváram alert box, ktorý sa zobrazí, keď používateľ zadá nesprávne prihlasovacie meno.

```
} else {
    //using my own handlers for creating a alert box if the name is not correct.
    class Allert extends Application {
        @Override
        public void start(Stage primaryStage){
            primaryStage.setResizable(false);
            AnchorPane pane = new AnchorPane();
            Button button = new Button( text: "Späť");
            button.setLayoutX(125);
            button.setLayoutY(100);
            //handler
            button.setOnAction(event -> primaryStage.close() );

            Label label = new Label( text: "Zadali ste nesprávne meno.");
            label.setFont(new Font( name: "Arial",  size: 20));
            label.setLayoutX(30);
            label.setLayoutY(45);

            pane.getChildren().addAll(label,button);

            Scene scene = new Scene(pane,  width: 300,  height: 200);
            primaryStage.setScene(scene);
            primaryStage.showAndWait();
        }
    }
    Allert allert = new Allert();
    allert.start(new Stage());
}
```

Lambda výrazy:

- Lambda výrazy používam napríklad v triede „**ShoppingCart**“, keď v metóde **getItemsOfCart** pomocou lambda výrazu implementujem metódu fuctional interface-u.

```
public String getItemsOfCart() {
    FunctionalInterface f = () -> {
        listOfItems = "";
        try {
```

Default method implementation:

- Default method implementation som využil v interface-i „**DefaultInterface**“. V tomto rozhraní implementujem metódu s kľúčovým slovom **default**.

```
public interface DefaultInterface {

    /**
     * Methods that calculates total time of production. It iterates through
     * the all customer's shopping cart.
     * @param cart  Shopping cart , Customer's shopping cart.
     * @return int, Total time of production.
     */
    default int createFurnitureDefault(ShoppingCart cart){
```

Serializácia:

- Pomocou serializácie si uchovávam všetky informácie o zákazníkoch. Serializujem celý ArrayList<Customer>. V tomto arraylist-e sa nachádzajú všetky informácie o zákazníkoch a o ich nákupoch.
- Pri spustení programu sa mi zo súboru načíta arraylist zákazníkov.
- Po vytvorení novej objednávky sa do arraylist-u uloží nový zákazník a následne tento modifikovaný arraylist sa opäť serializuje.

```
FileInputStream file = new FileInputStream( name: "src/model/SerializableFile.txt");

if(file.available() != 0) {
    ObjectInputStream in = new ObjectInputStream(file);
    try {
        Main.customers = (ArrayList<Customer>) in.readObject();
        for (Customer c : Main.customers) {
            Main.countOfCustomers++;
        }
    } catch (EOFException exception) {
        System.out.println("Empty file exception.\n");
    } catch (IOException e) {
        System.out.println("IOException.\n");
    }
}
```

- Deserializácia

```
try {
    FileOutputStream file = new FileOutputStream( name: "src/model/SerializableFile.txt");
    ObjectOutputStream out = new ObjectOutputStream(file);
    out.writeObject(Main.customers);
    file.close();
}
```

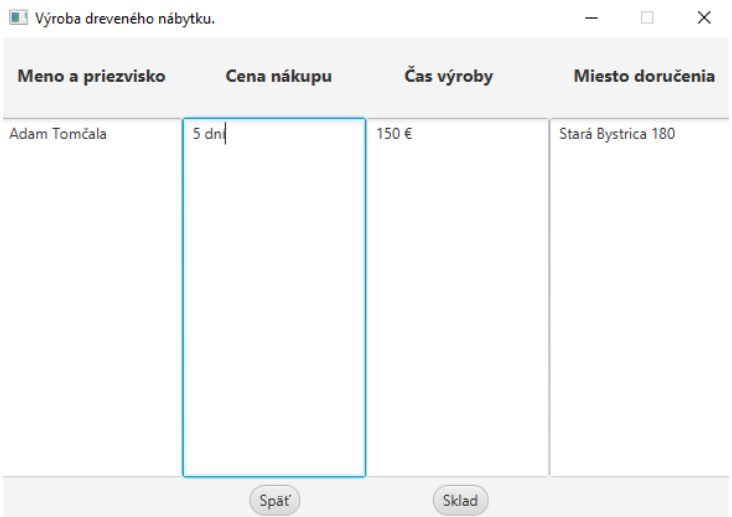
- Serializácia

Používateľská príručka:

- Po spustení aplikácie bude mať používateľ na výber ako bude pokračovať:

- Po stlačení tlačidla „Prihlásiť sa“ sa používateľovi zobrazí okno, kde bude musieť zadať prihlasovacie údaje, aby mohol pokračovať (buď ako admin alebo ako zákazník, ktorý už vykonal nejakú objednávku).

- Ak zadá správne prihlasovacie údaje od administratívneho účtu, používateľ uvidí všetky objednávky:



- Po stlačení tlačidla „Sklad“ používateľ bude môcť obnoviť zásoby dreva v sklade:

- Po stlačení jednotlivých tlačidiel sa zväčší počet daného druhu dreva o 50

- Ak používateľ na pri štarte aplikácie stlačí tlačidlo „Pokračovať“, presunie sa do časti, kde si bude môcť zakúpiť dostupný druh tovaru.

- Ak používateľ bude chcieť vidieť svoj nákupný košík, musí stlačiť tlačidlo „Košík“, ktoré zákazníka presmeruje na scénu, kde svoj košík uvidí.

- Ak zákazník bude chcieť v nákupe pokračovať, musí stačiť tlačidlo „Späť do obchodu“. Ak bude chcieť nákup dokončiť, musí stlačiť tlačidlo „Dokončiť nákup“. Po jeho stlačení sa zobrazí nasledujúca scéna:

- Ak zákazník vyplní všetky textové polia a následne stlačí tlačidlo „Potvrdiť“, objednávka bude vykonaná.

- Po vytvorení objednávky si daný zákazník bude môcť pozrieť svoj nákupný košík, ak na prihlasovacej scéne zadá svoje prihlasovacie meno a heslo, ktoré si určil predtým.

Zoznam niektorých dôležitých commitov:

- 11.4.2021 -> Pracovná verzia programu (program obsahoval základnú logiku, dedenie, polymorfizmus a agregáciu).
- 14.4.2021 -> pridanie vlastných výnimiek
- 26.4.2021 -> serializácia a dokončenie warehouse
- 5.6.2021 -> Serializácia
- 11.5.2021 -> Refactoring kódu a JavaDoc