**Restaurant management system**

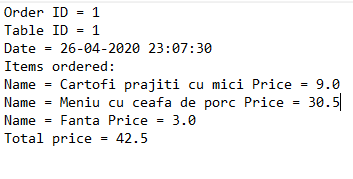
**Avram Andrei-Alexandru**

**30424**

1. **Assignment’s objective**

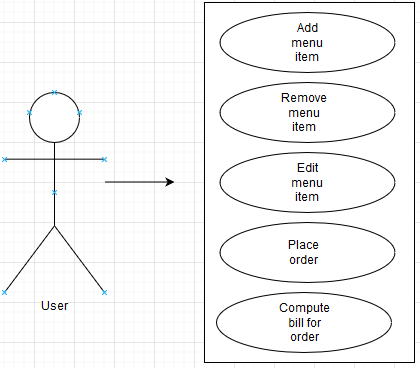
The main objective of this assignment is to design and implement a restaurant management system. This system will be capable of handling three types of users: Administrator, Chef and Waiter. For each of these users some specific operations are defined like: inserting, removing, editing a menu item, placing an order, computing the bill for an order and cooking for the chef. As a secondary objective one could be identified: implementing a mechanism for saving the menu data (serialization).

The input for the restaurant management system is supplied in a form of a file with the ‘.ser’ termination where the state of the classes that make the menu of the restaurant is saved. The output (the bill) is in the form of a simple text file where the corresponding order information are stored. Below is a sample output file.



1. **Problem analysis, modelling, scenarios, use-cases**

The possible use-cases are presented in the drawing below:



Use-case number 1: Add menu item.

1. Success scenario:

- The users enter the product name and price correctly and the product does not exist in the system.

- A new product is successfully inserted in the restaurant management system.

1. Alternative scenario 1:

* The product name and/or price are incorrect and the new product will not be added.

1. Alternative scenario 2:

* The product already exists in the system so it will not be added again.

Use-case number 2: Remove menu item:

1. Success scenario:

* The product is successfully removed.

1. Alternative scenario:

* The list of menu items does not contain the product so it will not be removed.

Use-case number 3: Edit menu item:

1. Success scenario:

* The new product name and price is correct and the product is edited successfully.

1. Alternative scenario:

* The new product name and/or price are incorrect and the product will not be edited.

Use-case number 4: Place order:

1. Success scenario:

* A new order for the table having tableID is placed.

1. Alternative scenario:

* The tableID is not an integer greater than 0 and the order will not be placed.

Use-case number 5: Compute bill for an order:

1. Success scenario:

* The bill for the order is computed and written into the output text file.

1. Alternative scenario:

* Because of some file error the order could not be generated.

This application can have many-real world use-cases. For example, it can be used by a restaurant to simplify and digitalize the process of placing orders and getting the orders to the chefs.

1. **Design (design decisions, UML diagrams, data structures, class design, interface, relations, packages, algorithms)**

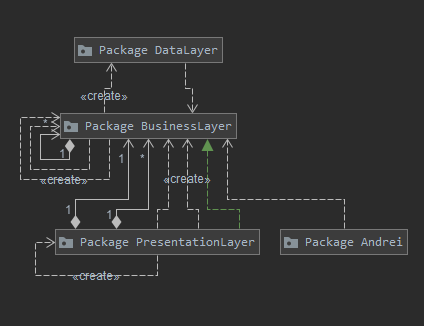
For the design of this application I have chosen to use four packages. These packages are explained briefly below:

* com.Andrei – Holds the class responsible for starting the application.
* com.BussinessLayer – Holds the classes responsible for the application logic.
* com.DataLayer – Holds the classes responsible for the input and the output.
* com.PresentationLayer – Holds the GUI classes.

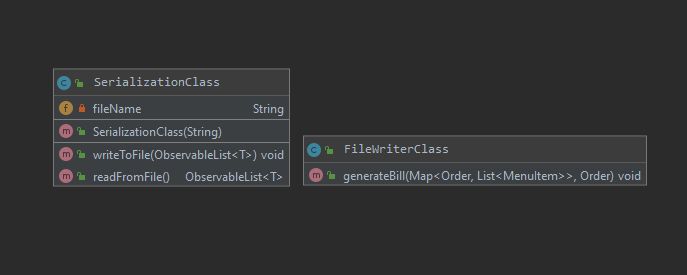
Data structures used in the application:

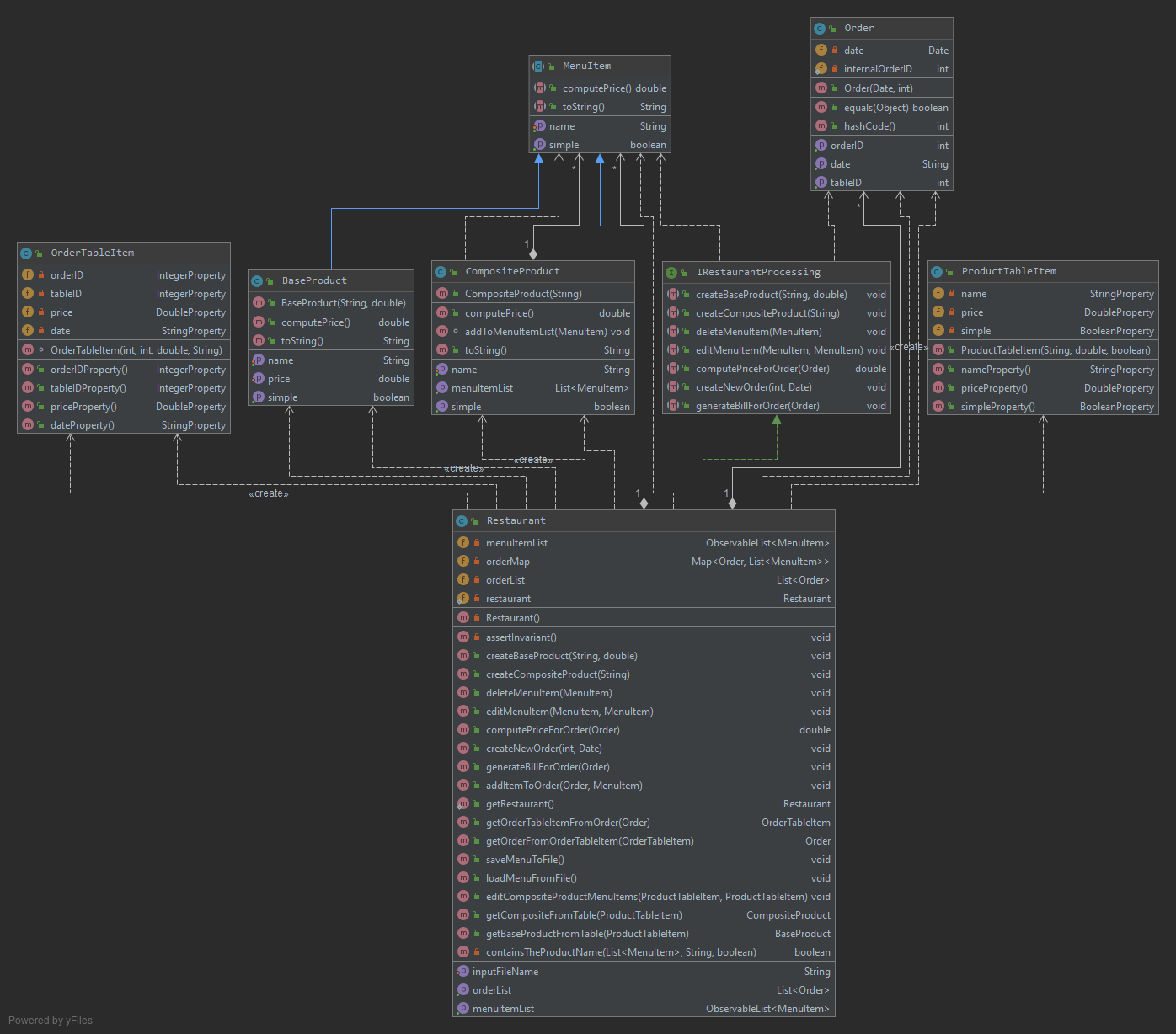
1. ArrayList to hold only the orders inside the Restaurant class.
2. HashMap to hold the relation between an order and the menu items ordered.
3. ObservableArrayList to hold the menu items.

UML diagram at the package level:

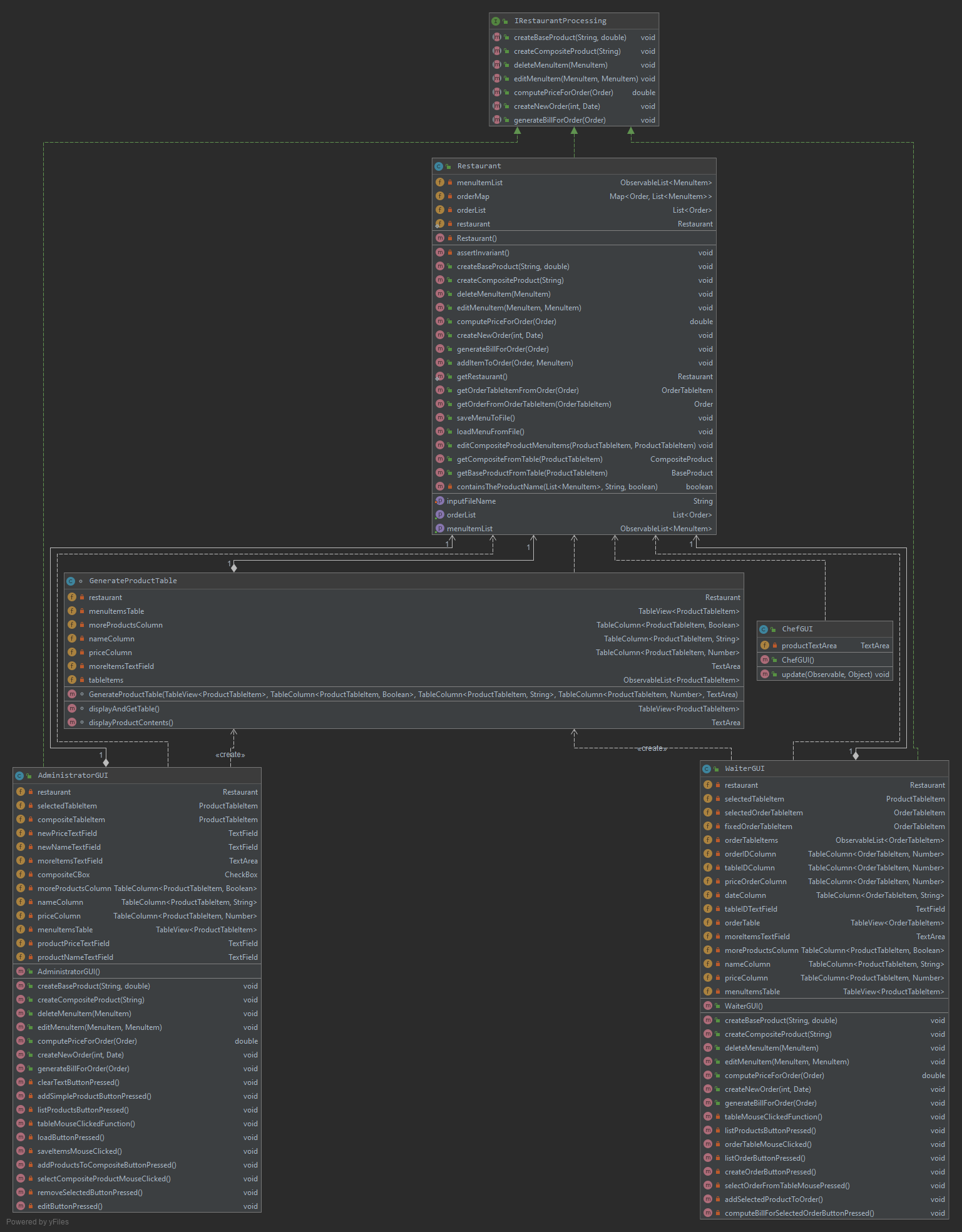


UML diagram for the DataLayer package:

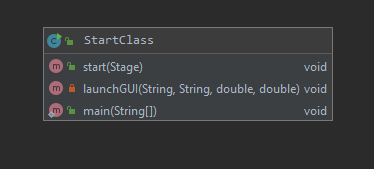


 UML diagram for the BussinessLayer package:

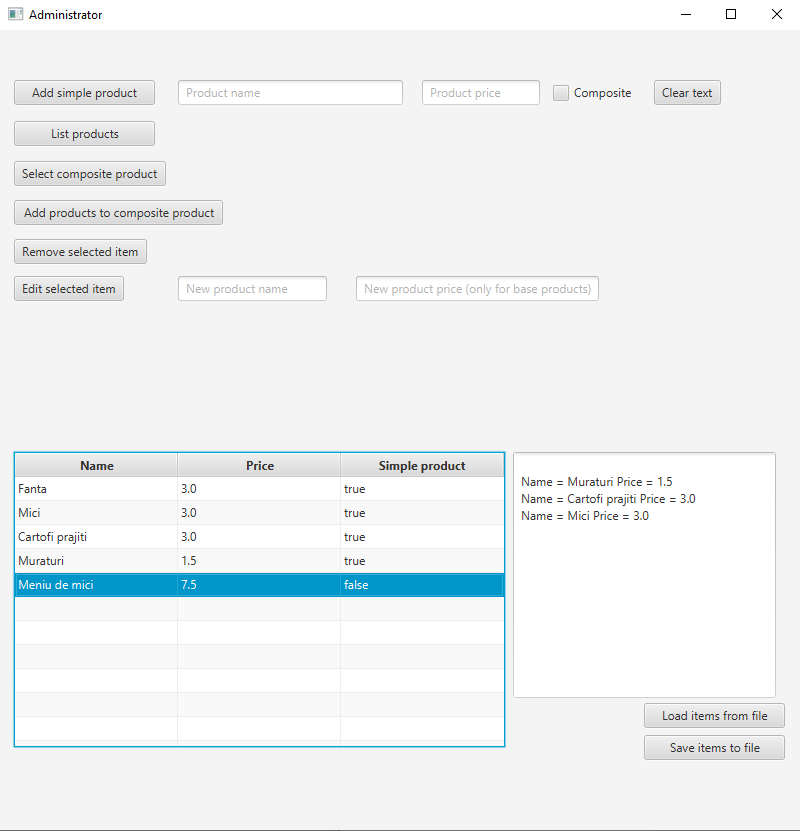
UML diagram for the presentation layer package:



UML diagram for the package Andrei:



Next, I will explain the GUI design. JavaFX was used to design the GUI. This application uses three types of users: Administrators, Chefs and Waiters. Each user has its own window. First, I will describe the administrator GUI:

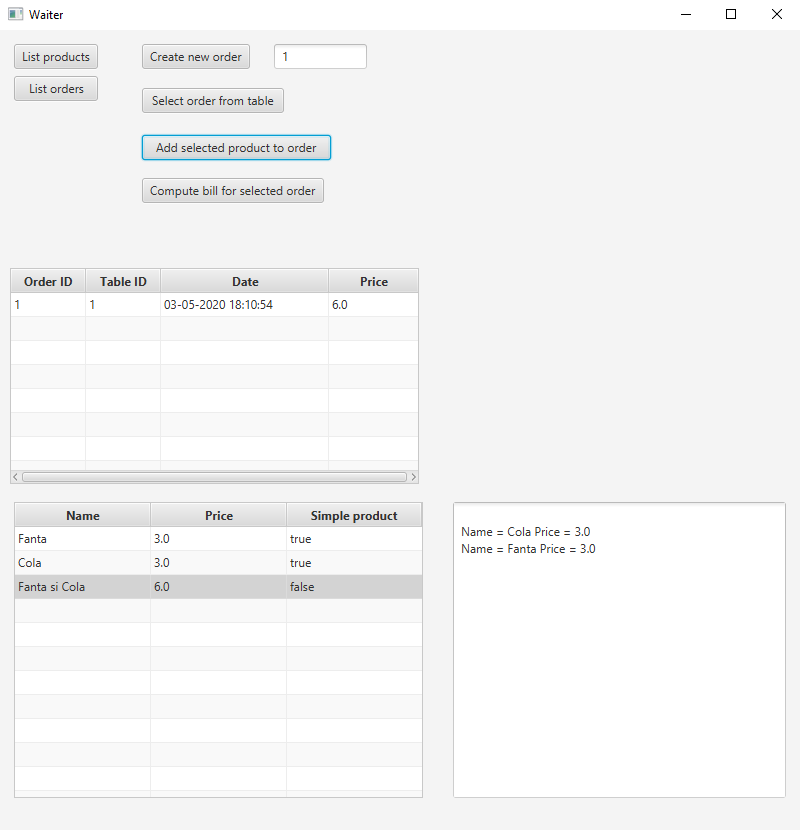


Inside the Administrator window the menu items are displayed in the TableView that is found at the bottom of the window. Here we can see the product name, price and if it is a simple product or a composite product. If the product is a composite product it will contain more base products. These base products are displayed in the TextArea that is to the right of the TableView. To see the contents of a composite product the user must left click the entry in the table he wants to see.

In the right bottom corner two buttons for saving and loading the menu items are found. The menu items are saved and read from the file supplied via the command parameters. When ‘Edit selected item’ button is pressed the selected item from the TableView will get the new name and price. The procedure to add a composite item to the menu is the following:

1. Add a simple product and check the composite checkbox.
2. Select from the TableView the composite item.
3. Press the ‘Select composite product’ button.
4. Select another product from the TableView to add to the composite item.
5. Press the ‘Add products to composite product’ button to add the selected product to the composite one.
6. Repeat for every item you want to add, the price is automatically adjusted for every item that you add.

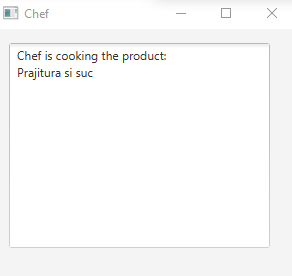
The Waiter GUI is the following:



The items that are inside the menu are stored in the same manner as in the Administrator GUI. (TableView and TextArea for composite items). To create a new order the user must enter an integer greater than 0 in the text field that is adjacent to the ‘Create new order’ button. After the button is pressed an empty order is created (it does not contain any products). To add products to the order one must select the order from the table, press the ‘Select order from table’ button and then select a product to add to the order and press the ‘Add selected product to order’ button. The price for the order is automatically computed each time a new product is added.

To compute the bill for an order, first select the order from the order table and then press the ‘Compute bill for selected order’ button. The bill will be generated inside a text file with the name ‘OrderDate.txt’ where Date is the date and time the order was placed.

The Chef GUI is the following:



Inside the chef GUI only a text area is present. Each time a composite item is added to an order the item will appear here (simulating the cooking). Please note that in this example, maybe not the best, ‘Prajitura si suc’ was a composite product.

1. **Implementation**
2. Start class

This class extends the Application class. It is the starting point of the application, containing the main method. Here the input file is read and it is passed to the Restaurant class to set the input file for the Serialization class. After the file is read the three graphical user interfaces are launched with the launchGUI method that loads the corresponding fxml file and sets the scene visible for all the graphical user interfaces.

In this implementation all the windows are open at the same time. You can work in parallel with different users without closing a window and open another one. This can work because the main class that makes the application logic is of singleton design, every class shares the same Restaurant class.

1. BaseProduct class

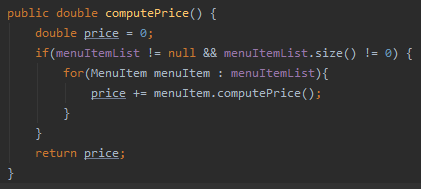
This class represents a base product in the system and it extends the MenuItem class. A base product is a product that does not contain other products. The class has fields for name, price and a Boolean field named ‘simple’. The field ‘simple’ is set to true to signal that this MenuItem is of type BaseProduct.

The methods are only getters and setters.

1. CompositeProduct class

This class represents a composite product inside the system and it extends the MenuItem class. It has two fields, a name for the product and an ArrayList to hold the products that form this product. The simple field is set to false here.

This class has only setters and getters and a method to compute the price for the composite product. This method is presented and explained below:



First the price is set to 0. We iterate through all the MenuItems that form this product and add to the price the item price that is found inside the list. In the end we return the price.

1. IRestauranProcessing interface

This is the interface that defines the main operations for the waiter and administrator user. These operations are: create base/composite product, delete a MenuItem, edit a MenuItem, compute price for an order, create a new order and generate bill for order. These methods will be implemented inside the Restaurant class. This interface has an invariant, postconditions and preconditions. The invariant of this class tests if the data structures used to hold the menu and orders are not null. For each method the precondition and postcondition will be explained in the Restaurant class explanation.

1. MenuItem class

This is an abstract class that implements the Serializable interface. It has the methods that every menu item must have and implement.

1. Order class

This class is used to store an order. It has four fields, an OrderID fields that is a private integer which holds the ID for the current order, date which holds the date this order was placed, tableID which holds the table id representing the table for which the order was placed and an internalOrderID which is a static integer that in the beginning has the value 0. The user does not have control over the order id. Each time a new order is placed the static internal order id is incremented so every new order will have a uniquely auto incremented id. This class is used to compute the key for the map that holds the orders inside the Restaurant class. In order to compute the key two methods have been overridden, the hashcode method to compute the hashcode of this class using the orderID, date and tableID fields and the equals method to test if two objects of type Order are equal.

1. OrderTableItem class

This class is used to populate the order table that is inside the waiter graphical user interface. It has four fields that mirror the order fields but are of type property so it can be added to the TableView. The fields that will go inside the table are: orderID, tableID, price and date. For each field a corresponding getter was created.

1. ProductTableItem class

This class, as the OrderTableItem class, is used to translate a MenuItem in a form that can be added to the TableView. It has three fields of type property and corresponding getters for every field. The fields that will appear inside the table are: name, price and simple (true if it is a base product and false if it is a composite product).

1. Restaurant class

This class is the class that is in charge with the restaurant logic of the application. It extends the Observable class and implements the IRestauranProcessing interface. It has 5 fields: an observableArrayList to hold the menu items, an HashMap to hold the information for order and the corresponding products that have been ordered, an ArrayList to hold only the orders, a static field of type Restaurant because this class uses the singleton design pattern and a string that represents the input file from where the menu items are read and stored. Next, I will explain the most important methods found inside this class:

* assertInvariant() method – used to test the class invariant. It asserts that the menu item list, the order list and the order map are not null.
* createBaseProduct() method – used to create a base product having the name and the price as the corresponding parameters. The product will be added only if another product with the same name is not found inside the menu list. This method has a precondition that the name is not null and a postcondition that after the product is added to the menu list the menu list size is increased by one.
* createCompositeProduct() method – used to create an empty composite product having the name as the parameter and the price is 0. It asserts the name to not be null and the menu list size increase.
* deleteMenuItem() method – is used to delete a menu item from the list that has a specific name supplied via a parameter. It asserts the name to not be null and the menu list size decrease.
* editMenuItem() method – is used to set a new name and price to a product. If the product is of type base product both the name and the price can be edited. If the product is of type composite product only the name can be changed because the price is automatically computed based on the products it contains.
* computePriceForOrder() method – is used to compute the price for an order, provided that the order exists inside the order map and it also contains products . It goes through all the products that the order contains, gets their price and returns the total value.
* createNewOrder() method – is used to create a new order having the tableID and date the same as the parametters.
* generateBillForOrder() method – is used to compute the bill for an order. It asserts the order to not be null and it calls the generateBill method found inside the FileWriterClass.

In the method addItemToOrder() when a new menu item that is of type composite item is added to an order the observer (chef) is notified to start cooking the product.

1. FileWriterClass class

This class is used to generate the text bills for orders. It has only one method generateBill() where the data that will be written to the text file is prepared.

1. SerializationClass class

This class implements the Serializable interface and is used to serialize a list of objects. This class is made generic. It has two methods, one for saving the list of objects of type T and one for reading a list of objects of type T and returning that list.

1. AdministratorGUI class

This class is used to make the GUI operations for the Administrator user. It also implements the IRestaurantProcessing interface and inside the methods the corresponding methods inside the Restaurant class are called.

1. ChefGUI class

This class is used to make the GUI operations for the Chef user. It also implements the Observer and has a method update that recives the product that the chef must cook when the observer is notified.

1. WaiterGUI class

This class is used to make the GUI operations for the Waiter user. It also implements the IRestaurantProcessing interface and inside the methods the corresponding methods inside the Restaurant class are called.

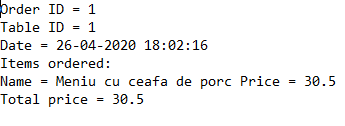
1. GenerateProductTable class

This class is used to generate a TableView to represent the menu of the restaurant in a table form.

I have created this class to avoid code duplicates because the same table must be generated inside the administrator window and inside the waiter window.

1. **Results**

After the normal use of this restaurant management system several text files will be generated depending on the number of orders that were placed. A sample of this output is presented below:



1. **Conclusions**

After completing this assignment, I have learned how to use serialization to save the internal state of objects, how to use maps in java and how to make tables in javaFX. In a future development of this application one can make some type of login system to prevent unwanted access to the administrator panel. Also, one can include a database for the menu and users details storage.

1. **Bibliography**

<https://www.geeksforgeeks.org/map-interface-java-examples/>

<https://www.tutorialspoint.com/java/java_serialization.htm>

<https://docs.oracle.com/javafx/2/ui_controls/table-view.htm>