Restaurant  
Order  
Management

Programming Techniques – Homework 4

2ND Year, 2ND Semester, Group 30422

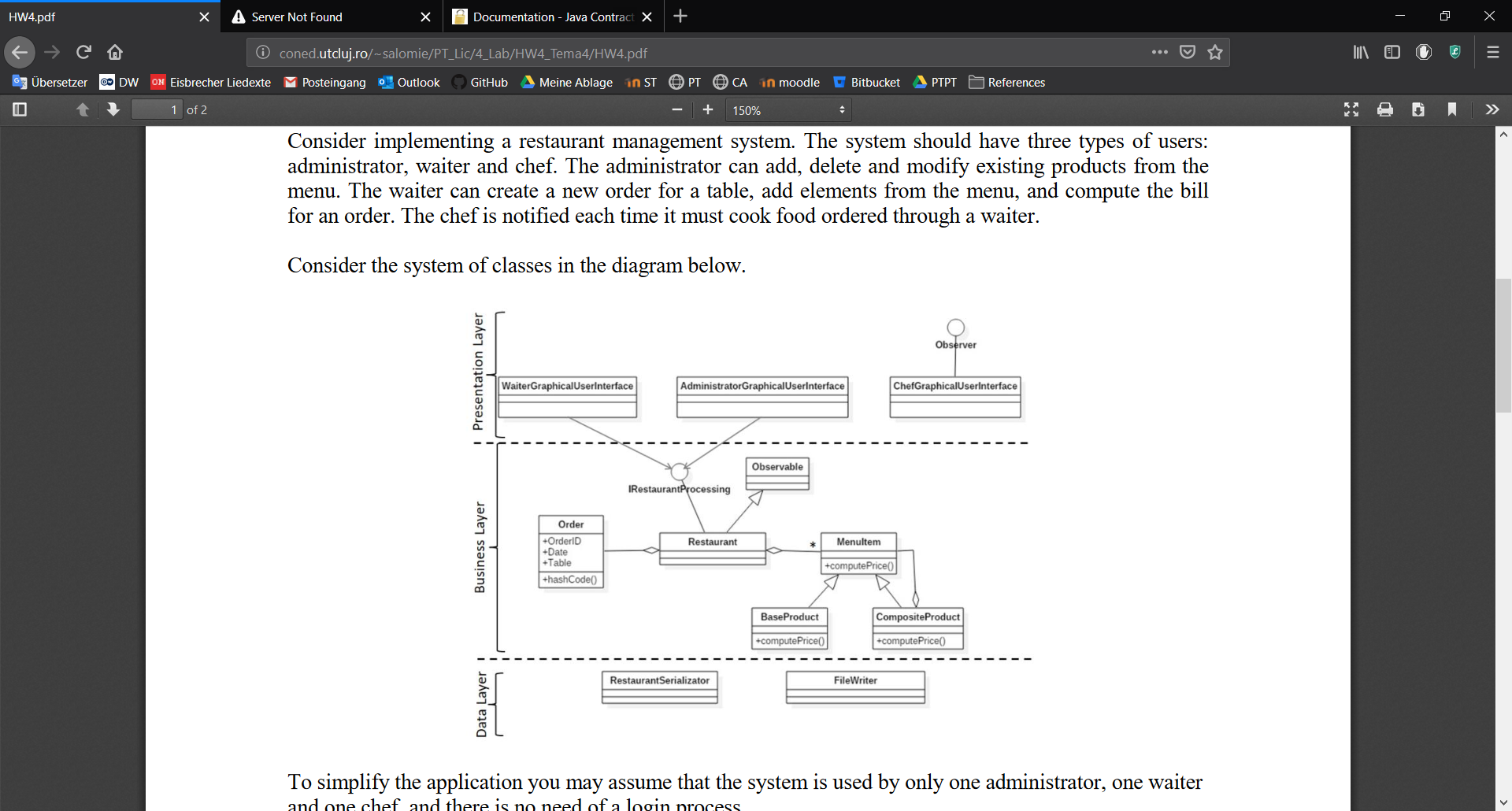
Student Name: Pop Elisa Teea

Technical University of Cluj-Napoca

Computer Science and Automation Department

Problem Definition

**Task: The application is meant to implement a restaurant management system. The system should have three types of users: administrator, waiter and chef. The administrator can add, delete and modify existing products from the menu. The waiter can create a new order, add elements from the menu and compute the bill from an order. The chef is notified each time it must cook food ordered through a waiter.  
  
 We should consider the system of classes in the following diagram:**



The stated problem can be reached by solving the following sub-problems:

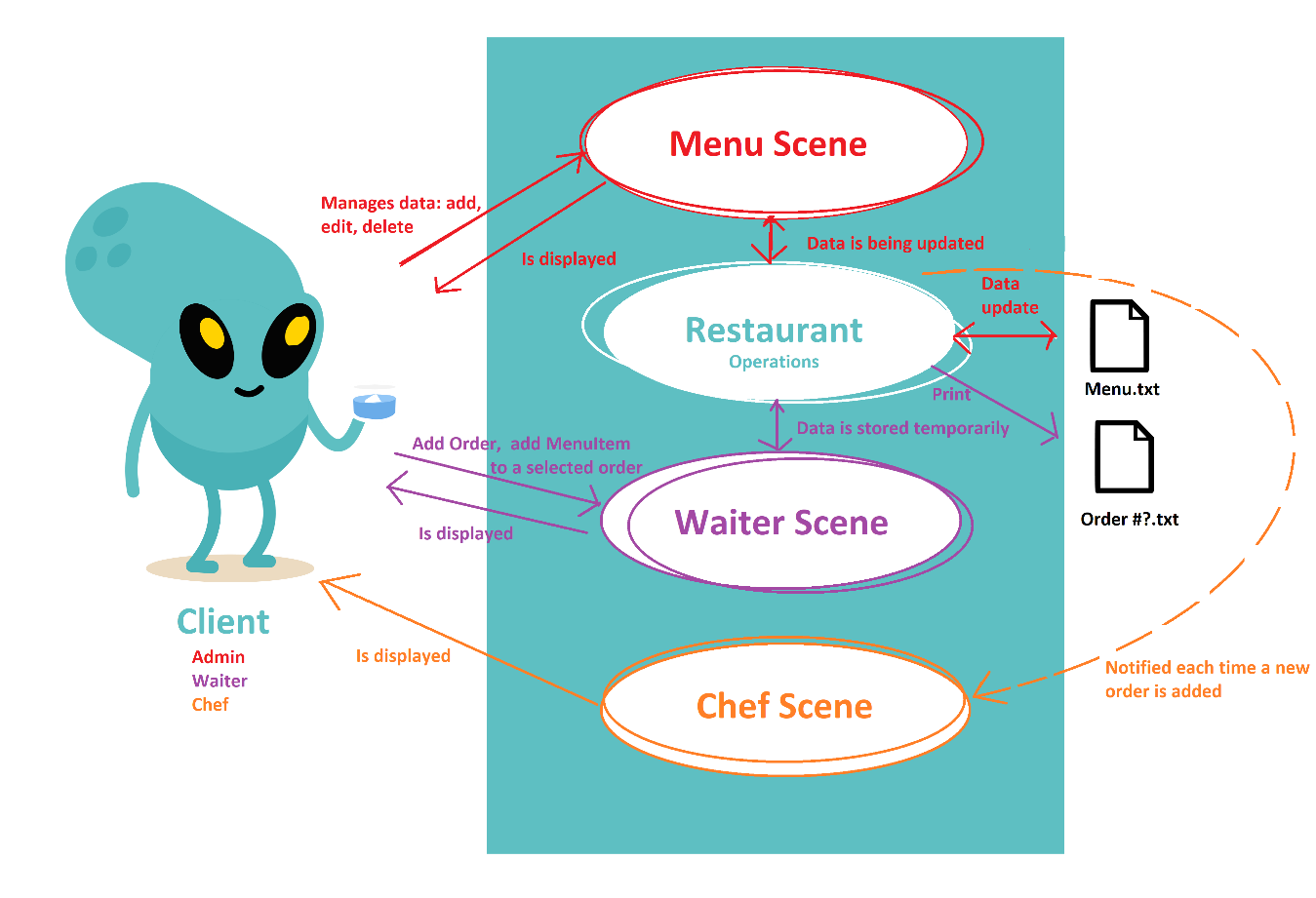
* Implementing the Data layer for serializing MenuItems
* Implementing the functionality of the given data : Business Layer
* Preparing the data so it can be shown and interacted with in the UI ( Controller + View): Presentation Layer . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

Project specifications:

* Programming language used: Java
* Project SDK: 1.8 ( java version “1.8.0\_191”)
* Project language level: 8 – Lambdas, type annotations, etc.
* GUI: JavaFX
* Program used: IntelliJ Idea
* Git : Bitbucket

Problem Analysis

Use-case diagrams . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

This application has the intention to allow the user to add, edit, delete and see the items in the menu, add, edit and print orders and observe when a new order is added. It all depends upon the user: what’s its role and what purpose this application fulfills for it.

Actors: the Client   
Use case title: Start the simulation  
Preconditions: The user has to introduce valid data for all the given fields specific to the function that is about to be used. This varies depending on the current scene.

Success scenario:  
- The user introduces the correct input for each field  
- The user chooses the right operation for the fields that were introduced  
- The change can be noticed in the displayed table.  
  
Alternative scenario:  
- The user introduces an empty, invalid id or field  
- An error box notifying the user about the current problem is shows.  
- After the user reads and closes the box, the activities can be continued normally.

Design and Programming

The project is structured using packages, according to the give diagram: bll is the business logic layer, dl is the data layer, gui is the presentation and utility provides useful methods that don not fit in other packages.

The packages

gui

* As the name suggests, this package covers all the UI-related elements.
* Due to the fact that the MVC architecture is difficult and counter-intuitive to implement with JavaFX because usually we’re working with the Scene Builder so the UI contains a Controller and a .fxml file which is the view. This is the reason why why the application only has a Controller and a View.
* For this implementation, I used multiple views due to the fact that the user can and should be able to swap between the scenes in order to see the data from all the tables. Each of these views has a Controller, which manages each specific operation there is for every different view.

bl

* BL stands for business logic layer and it contains methods that help coordinate data related to the UI, while keeping a data model to help in doing so.
* It contains classes that are necessary for displaying/ storing the necessary data, and those classes are: Order, MenuItem;

dl

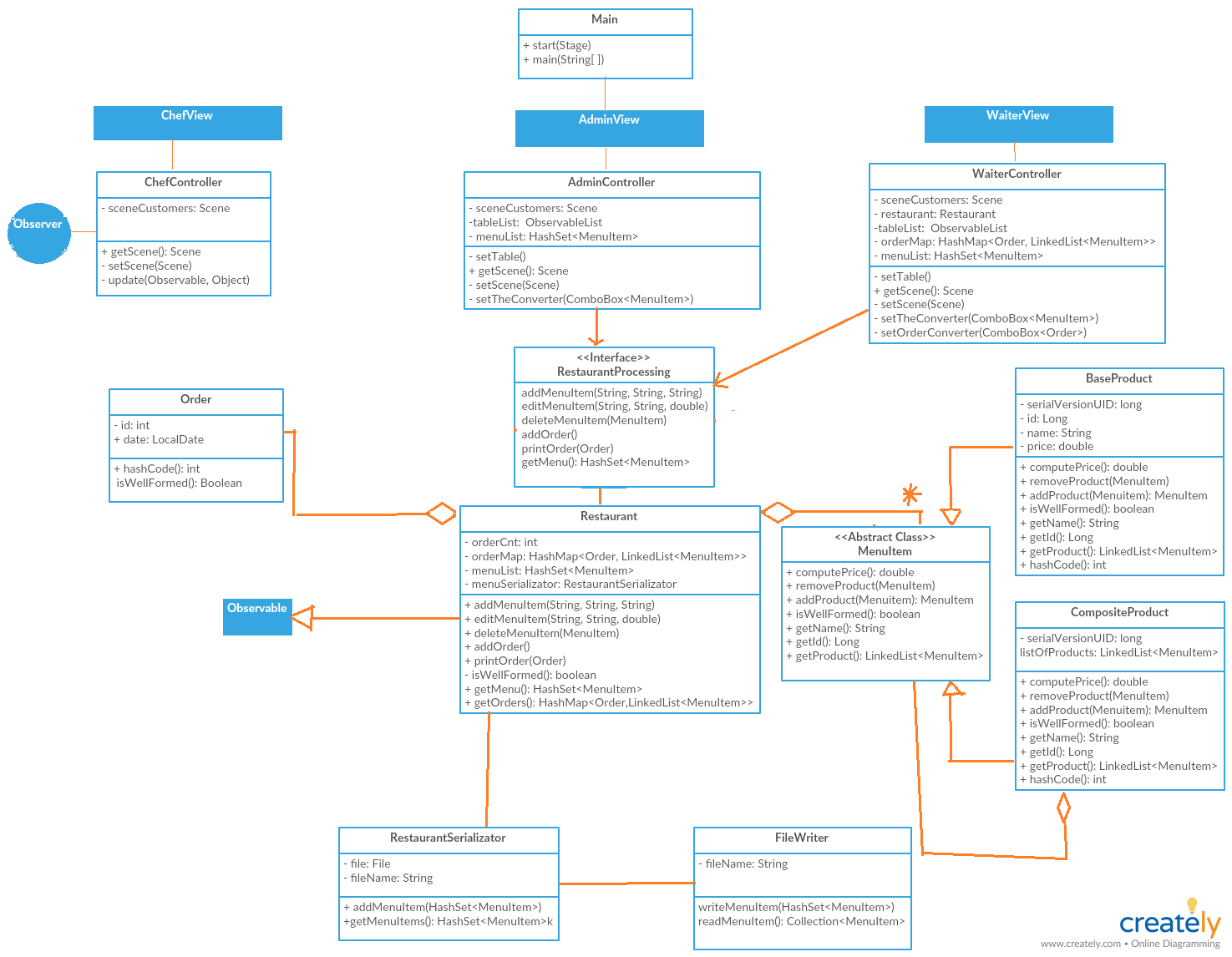
* DL stands for data layer and it contains method for serializing and deserializing the data to be stored.

main

* This package holds a sole class: Main class. This is used for setting the stage and starting the application.

utility

* This package contains various methods that do not exactly fit in the packages mentioned above.

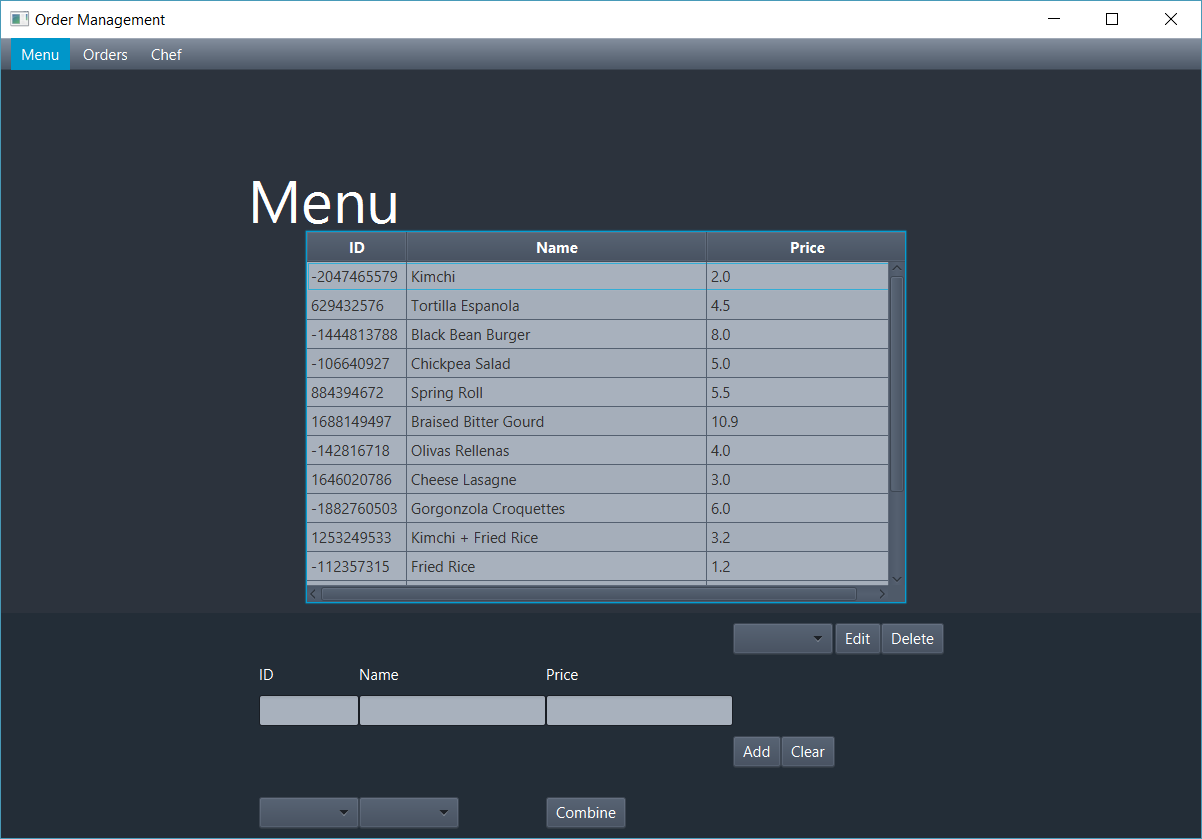


Implementation

The user interface

It consists of 3 more packages, each containing a View and a Controller. The Scenes are available to be accessed from one another, because of a ManuBar. Each scene supports the operations specific to the current scene.

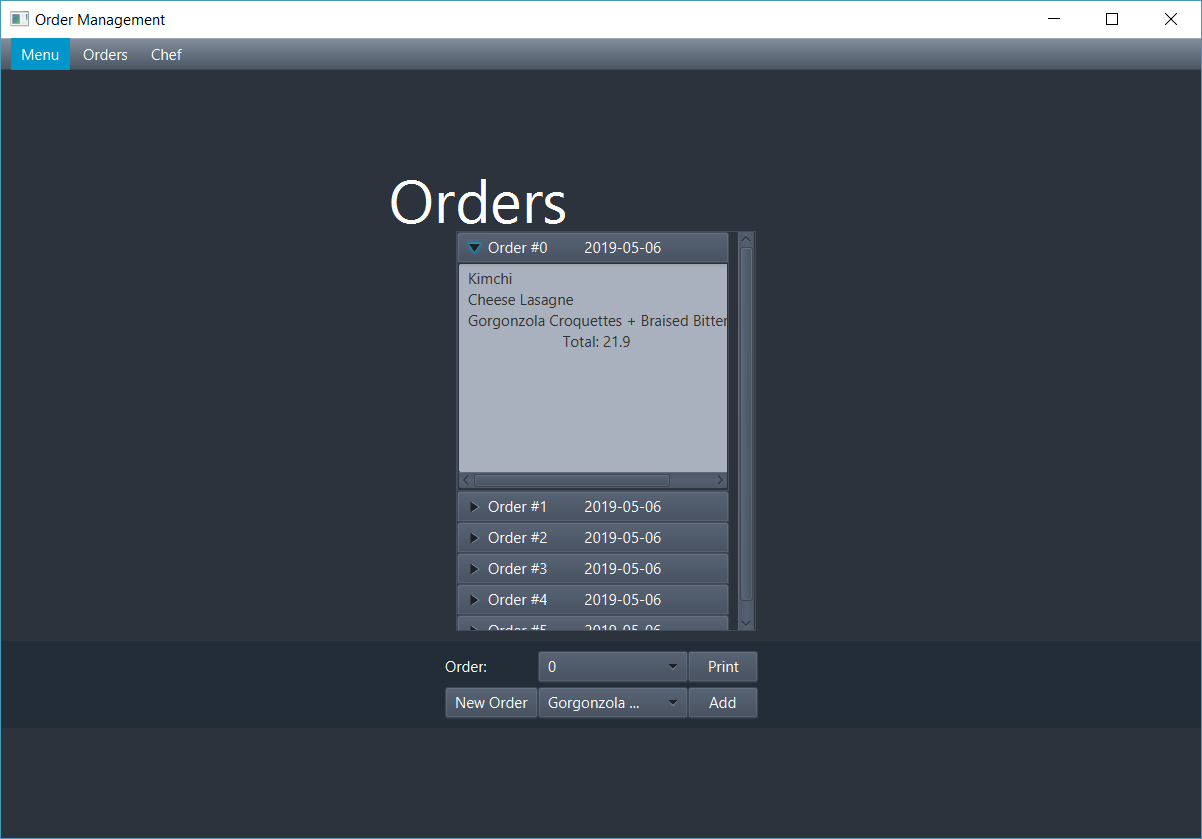
For the Menu scene, the “Add” button adds the field to the HashSet of MenuItems, the table that is always shown and updated in real time; In order to edit an item: you have to choose an item from the comboBox and click on edit. After editing the fields, click on “Add” to update the table, Delete: an element, by choosing it from the ComboBox. We can also combine two elements that already exist in the table by selecting them in the ComboBoxes from the bottom part of the interface. Pressing “Combine” will add the combined MenuItem in menu.



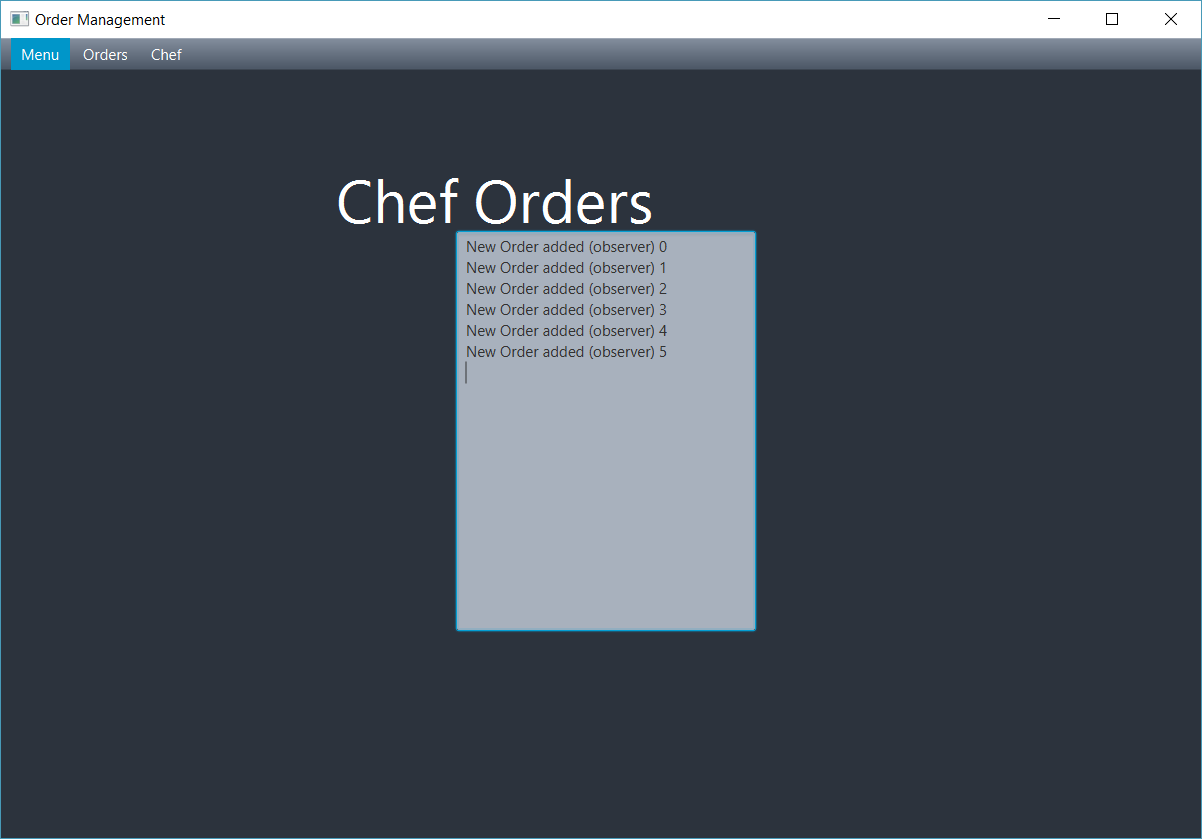
It’s a relatively simple ui, having used a ManuBar with 3 menu items, two GridPanes in which on GridPane has the table with the title label and the other one has 3 TextFields and 3 ComboBoxes with 3 Labels and 4 Buttons.

The ID label is always uneditable. The only way it can be changed is by choosing an item in the ComboBox from the upper right corner and clicking the Edit button. By doing that, all the fields are filled with all the current fields from the specified ID. You can change any field and click on the Add button: This will update the item in the Serialization file. However, each time you want to edit an item, you have to click on the Clear button afterwards because as long as the ID field is not null, the item at the current ID will keep getting edited.

The Waiter Scene contains a ScrollPane which will add a new TitledPane when you press the “Add Order” button. Afterwards, you can select an order from the upper ComboBox in order to print it, or add a new MenuItem to it by selecting an item from the lower ComboBox.



The Chef Scene contains a TextArea which Observes the Orders from Restaurant in order to notify the chef when a new one is added.



The Controller

Simply sets each action for every UI element present: updates the table, adds action listeners and has getters and setters for the View it represents.

The Restaurant

This is a very important method regarding the back-end of this application. It is the main class that stores the data and communicates with the Serialization Clases. It impelments the RestaurantProcessing interface which requires the class to override GUI methods regarding data. The following functions are implemented by this class:

* addMenuItem(String id, String name, String price) which creates a new MenuItem with the data added to the required fields and adds it to the menu, also updating the Serialization file.
* editMenuItem(String id, String name, double price) which edits the selected MenuItem with the fields required. Only the ID can not be edited. The Serialization file is updated afterwards.
* deleteMenuItem(MenuItem toDelete) which deletes the selected MenuItem and updates the Serialization file accordingly.
* addOrder() adds a new Order to the HashMap, which will be initially emptied. This method triggers the Observer.
* printOrder(Order givenOrder) creates a new text file which will contain details about the selected order. If the file already exists, it will be overwritten.
* isWellFormed() is a method used in order to check if the object is available for its class’ methods.
* getOrders() is a method that returns the HashMap containing Orders and its list of MenuItems . . . . . . . . .
* getMenu() is a method that returns the HashSet containing MenuItems.

The Order Class

* This class is used to create and store the data belonging to the order. This includes the ID and the date when the object was created. The attributes of this class are the following: id, date. Besides getters and setter, this function includes a hashCode used in order to make the Order objects to be used as a key for a HashMap.

The MenuItem Class

* This abstract class is used to create and store the data belonging to the menu table. The attributes of this class vary depending on the classes that extend it. It extends Serializable in order to be able to be stored in a simple txt file. It is part from the Composite design pattern and has either a single BaseProduct or a List of BaseProducts. The abstract methods mentioned to be overwritten are: computePrice(), getName(), getId(), removeProduct(), addProduct(), getProduct(), isWellFormed().

The BaseProduct

* This class extends MenuItem and is the base class of this Composite design pattern. The attributes of this class are the following: id, name and price.

The CompositeProduct

* This class extends MenuItem and is the composite class of this Composite design pattern. The attributes of this class are the following: LinkedList<MenuItem> listOfProducts. As the design pattern indicates, this class contains several ....BaseProduct objects and the abstract class indicates every method that can be shared by both Classes, in order to be implemented individually and used as an one.

The DoubleValidator

* This class contains a regex expression used to check if a specific field which it is used for contains a double number or not.

The FileWriter

* This class is used specifically for writing and reading Serializable objects in the . . . Serialization file. It can only read and write ALL the contents, therefore, when adding a single item, everything will be overwritten.

The RestaurantSerializator

* This class ties the Restaurant to the FileWriter: It sets the Serializable file and . . . overwrites that file whenever a new item is added (including editing and removing only a single MenuItem). It also gets the data from the Serializable file at the very beginning of the start of the application.

Results

Since there is not much to compute and the application relies heavily on the user interface, it can easily be tested and debugged directly by adding data to the interface.

Conclusions

After finishing this project, I can say I have learned a very important aspect about Java and Object Oriented Programming ( OOP ) overall, and that is the importance of . . . serialization and design patterns, specifically the observer pattern and the composite . . . pattern. I am thoroughly sure it can be further improved because I believe that an Order history would be entirely useful to the users. Also, the Chef could be provided with the list of items that will have to be prepared by the chef.

Further improvements for this program can be:   
- Logs that will help more with debugging   
- Showing the composite elements better: they can have a list of the base elements they are composed from.  
- Using a database for storing and getting information would make it much more versatile and useful.

References

<https://creately.com>

<https://stackoverflow.com/>

<http://jcontracts.sourceforge.net/addingContracts.html>

<https://www.geeksforgeeks.org/composite-design-pattern/>

<https://www.geeksforgeeks.org/serialization-in-java/>

<https://www.javaworld.com/article/2074956/icontract--design-by-contract-in-java.html>

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

Appendix

1. Cover …………………………………………………………………….. Page 1
2. Problem Definition ……………………………………………………... Page 2
3. Problem Analysis... ……………………………………………………... Page 3
4. Design and Programming ... ……………………………………………. Page 4

* The packages ..……………………………………………….…... Page 4
  + dll …... ….…………………………………………….…... Page 4
  + gui ….……………………………………….……….…... Page 4
  + bl . . …….……………………………………….………... Page 5
  + main ….…….…………………………………….…..…... Page 5
  + utility . …..….…………………………………….…..…... Page 5
  + UML ….………..………………………………….…..…... Page 5

1. Implementation…………………………………………………………… Page 6
   * The user interface ……………………………………………...… Page 6
   * The Controller ……………………………………………………. Page 8
   * The Restaurant ……………………………………………..…..... Page 8
   * The Order Class …………………………………………………. Page 8
   * The MenuItem Class ………………………………………..……. Page 8
   * The BaseProduct …………………………………….…………... Page 9
   * The CompositeProduct ……………………………...…………... Page 9
   * The DoubleValidator ………………………………………. …... Page 9
   * The FileWriter …………………………………………..…. . …... Page 9
   * The RestaurantSerializator ……………………………… ……... Page 9
2. Results ………………………………………………………………....….. Page 9
3. Conclusions ………………………………………………..…….….…….. Page 9
4. References ……………………………………………………….………... Page 10
5. Appendix ………………………………………………………….…...….. Page 11