Tehnical University of Cluj-Napoca



Facultatea de Automatică și Calculatoare

Assignment 4

Restaurant Management

Student: Loga Darius

Group: 30422

Section: CTI-E

TA: Diana Bâlc

**Table of Contents**

1. Aim of the assignment

2. Problem analysis

2.1 Use cases and scenarios

3. Design

4. Implementation

5. Results

6. Conclusions

7. Bibliography

**1. Aim of the assignment**

Here is the task that we are required to perform: Consider an application where you are required to implement or to simulate a restaurant management starting from the given class diagram and to solve some requirements. From the class diagram we can see that we have to use three packages that are explained below:

• Business Layer- contain the application logic

• Presentation layer – classes that contain the graphical user interface (**GUI**)

• Data Layer- classes that contain the Seralization Mechanism and bill generator

For the main objective we need to design and implement an application can can have three types of users and perform some actions to help with thei work.

As for the second we need to transform the real data into a problem data, implementing the Serialization mechanism, learning and making use of the Hash mapping, using the Composite and Observer Design Patter and design a User Interface to navigate easily through the application.

**2. Problem analysis**

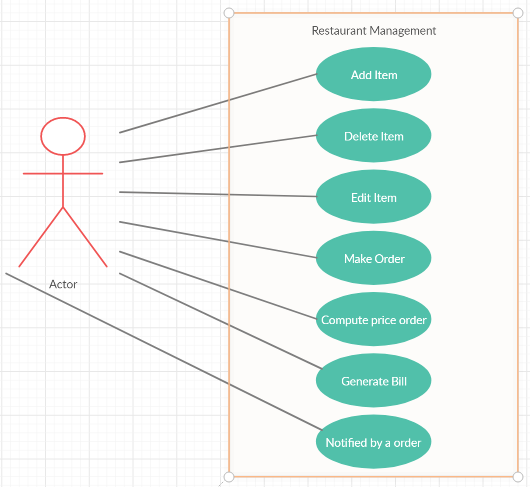
Just like in the real world, where we have restaurant that need to have a good management system, in the application we need to implement a multiuser platform, in which an administrator, a waiter and a chef can connect and look into it in a certain way: The administrator can add a new item in the menu, delete one of them or edit, but I thought that we can make group some objects and implement a case where you can add in composite those items, just like it belongs to a certain category, but not necessary. The Waiter can only add items to an order and generate the bills in a text file and a Chef that can only be notified when an order is placed to make the preparations.

We can make the assumptions that the provided input from all the actors are verified in advance and they introduce the correct version and there are no menu items with the same name nor the same ID for an order.

This is a quite simple to understand application that can be implemented with no such huge effort. There are relatively few classed, three are for the actors’ User Interface, that belongs to the presentationLayer, other five are for the implementation of the so called ”management” of the restaurant, in which have a interface that contains the methods for the Restaurant class, two classes out of that five that are really important, the Order and the MenuItem one which creates what items are in the menu, that can be base products or composite products, implemented, as well, in different classes, but to have an array where we can store the name of the items, simple or composed and their prices, single or combined and for the Order class we stored a hash function that is useful for storing the orders with the needed characteristics. The Order and the MenuItem classes implements the serialization mechanism, but also Restaurant, the control class, where it has a hash map for better to store the Orders with their assignated menu

items. I will try to better explain some of the scenarios that can happend before the actual implemtation of the whole application.

**2.1 Use cases and scenarios**

****

Upper it is the use case diagram and the actors that need to be implemented are the administrator, the waiter and the chef. To simply the implementation, only one instance of the actors are developed and no login or sign in method was provided.

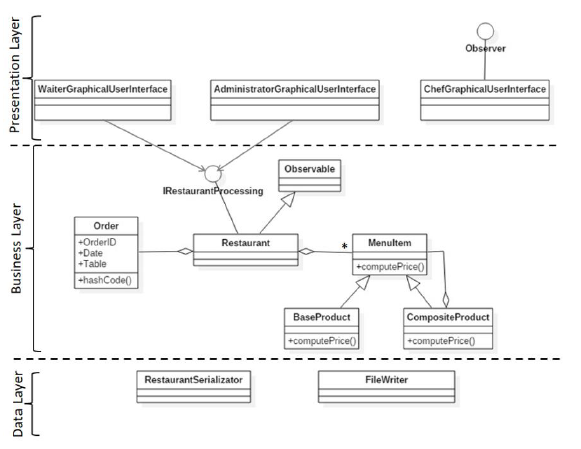
When we choose to be the administrator we can add an item to the menu, if the validations are correct, it should be because we made an assumption that this should be the case, when we press a button we should see that one item is added in a table that represents the menu itself. Same goes for the delete and the edit case, for delete, we cannot see anymore the item in the menu and the edit case only edits the price of one item that already exists in the menu. Additionally, I introduced a section where you can group base items into composite ones to make insert, for example, a pie that is made of apple and flour, if those items exists in the menu, and combine their prices to make the a composite product.

For the waiter perspective, we have an order to construct that can be made using only three inputs, one with the insertion of the ID, the number of the table and the date in which the order was taken, to easily store them in a hash map and later to associate those orders the menu items that the waiter selected. The computed price is the final price of the order when we add the base, composite products together and have the final price computed. The bill generator is a text file that shows the order’s ID, number of the table, date and the items associated with that order, like I mentioned, selected by the waiter after he served the customers.

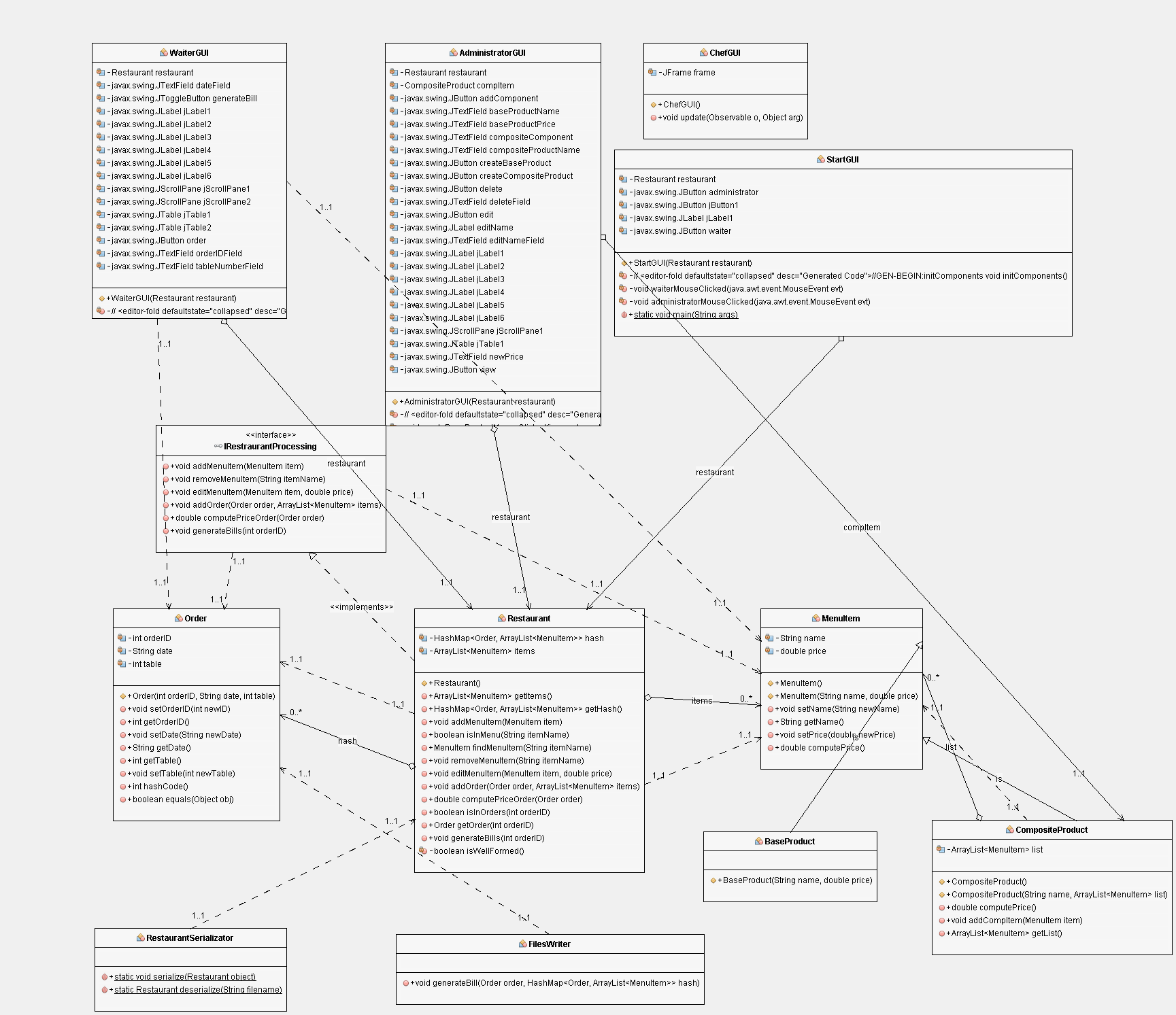
Last, but not least, there is the chef that implements the Observer class and his single case is that is notified when an order is ready to be processed by him.

3. **Design**

For the implementation of the design, like I mentioned in the first point, the Class diagram was given to us, but I generated also one from the NetBeans IDE, I will add both of them below. I designed the application in NetBeans, so I did not use the classic MVC method. This is the one given to us:



And below it is the generated one:



. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

I used the hash mapping method in one place, in the Restaurant class to store the ArrayList of the menu items that are associated with their required order and one hash function to encode my orders to be easily accessible to the hash mapping. The realtions can be identified quite easily if you can zoom in into the image but I will describe a little bit those as well. There are two aggregations realted to the Restaurant, one from the Order and one from the MenuItem that are used as arguments, like I said many times, for the HashMap type and for better storage. The Restaurant implements the IrestaurantProccessing interface and Observable class and as you can see the MenuItem has to different classes that inheritance from it because they have some characteristic in common. As for the GUI classes, those have different kind of objects from the Swing collection and only the ChefGUI class implements Observer that will help with the notification process. The remaining classes, RestaurantSerialization and FileWriter, those are used for storing or loading data using the Seralization mechanism and the other one is used for generating the bill as a text file.

The Graphical User Interface organization was quite complex because I have four classes for this implementation. One is used to choose the type of actor we are, waiter or administrator, and for each of those credentials I have Text Fields, JTables, Labels and Buttons and one JOptionPanel for the notification of the Chef, but a more detailed look I am going to discuss in the Implementation section.

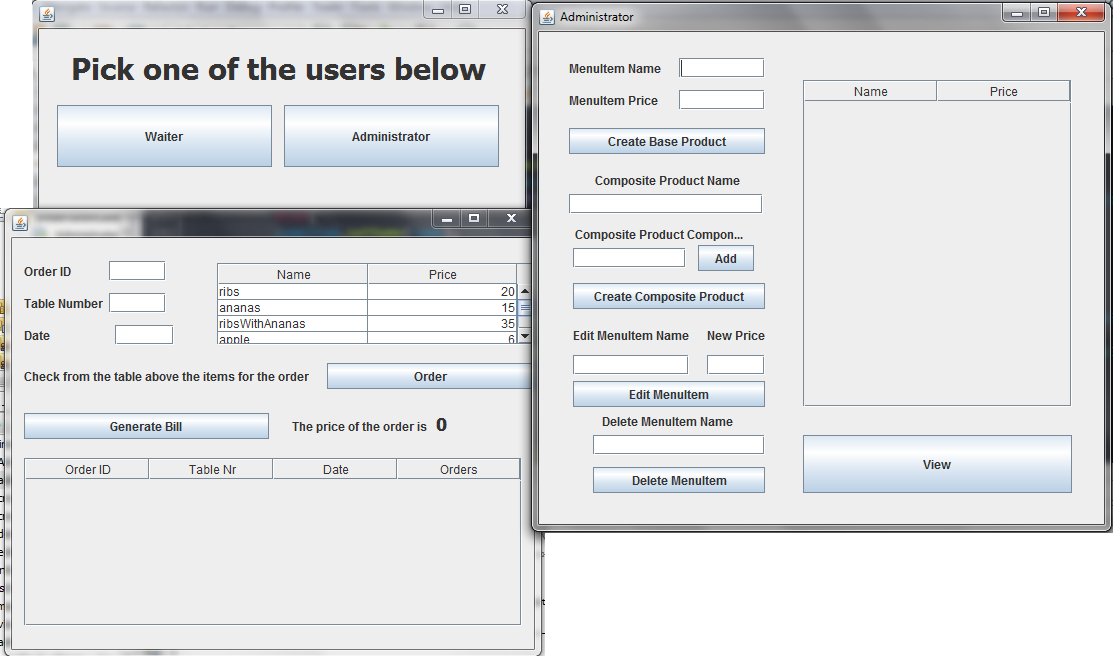
4. **Implementation**

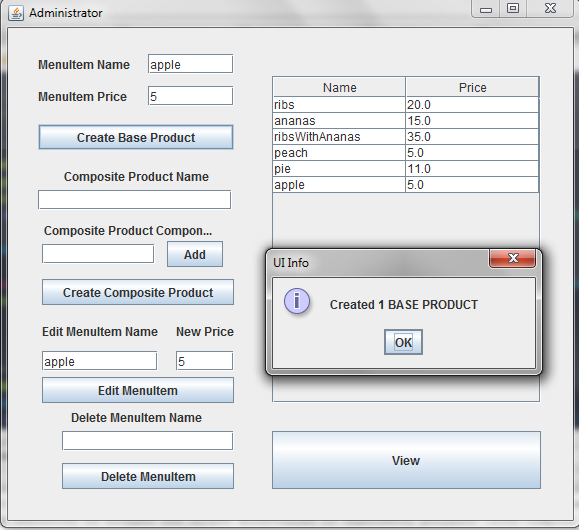
I will start in the order with the packages, so the first package is the businessLayer with the class: Restaurant, MenuItem, Order, BaseProduct and CompositeProduct. MenuItem, Order and Restaurant implement the Serializable and the BaseProduct and CompositeProduct extends the MenuItem class, like I mentioned in the design section that those inheritance some data. The BaseProduct and CompositeProduct are designed in a Composite way, that is why the ArrayList for storing menu items can have simple products and composed products that are made of multiple simple products and add their whole prices together and result in a final price and baseProduct only inheritance the name and price of a single menu item. The implementation of the IRestaurantProcessing is just an interface documented with custom tags of preconditions and postconditions that are constructed using the assert() method before the execution of a command and after it, just like in this example from the Restaurant class.



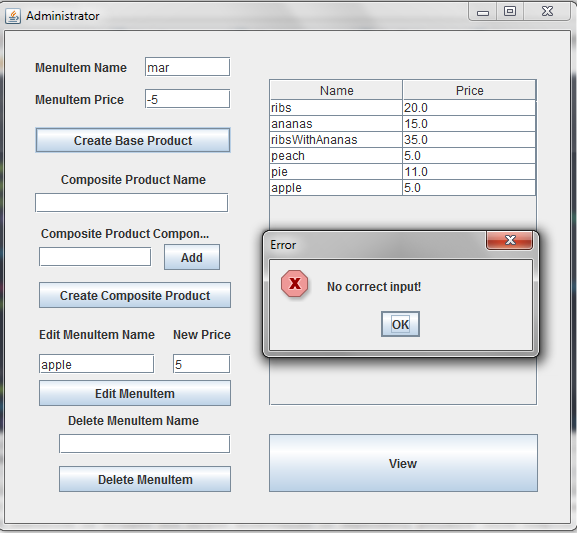
In the dataLayer, like I mentioned many times before, it used to store/load the data used in the GUI. The RestaurantSerializator is just a class that can access a file called “restaurant.ser” and write to that file or load/read data from it and can be translated into tables easily, just like a normal database can do, but with less code and no more connections. The FileWriter class is just a class that contains a method in which I can write into a text file some text associated with the correct orderID, table, date and the items for that order that can be seen in such a easy way.

Here, at the presentationLayer, the real complicated methods are being developed. In Netbeans was far less complicated to generate the Text Fields, Buttons, Labels, Tables and to edit them, but still, were a lot of those Swing object to be added. Below I will insert images that show the look of the GUI and some JOptionPanels that I will discuss. But before, the ChefGUI only implements Observer and has one method for the update that is made when we press the button that generated the bill, I will have an image for that as well. Below you can see that are all the classes, expect the Chef one, implemented and has all sorts of Swing objects.



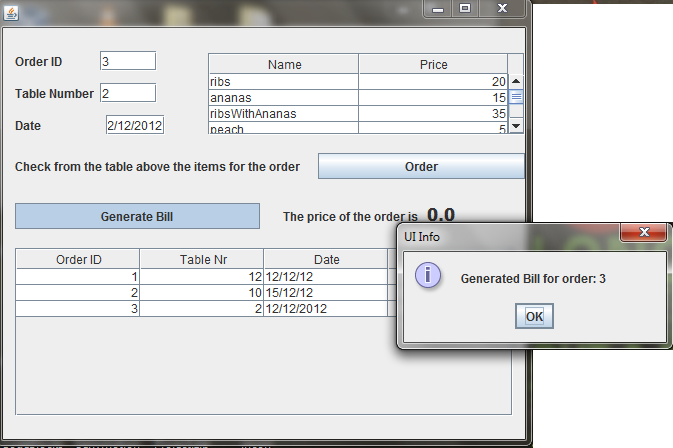


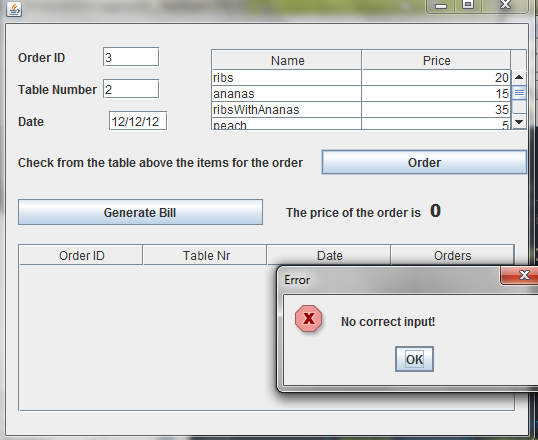
Those buttons implement the uses cases discussed at the Section 2 and by pressing them a Message is popping-up to inform us that I created a base product and hit “OK”. Same goes for the adding the base products to make a composite one and make a new composite item with prices combined and editing the MenuItem and price and for the deletation. Despite of my assumption of the correct inputs, I made some validations using regex, added this feature late and some orders still have the old format for the date, you will see an example below, for the name to have only letters, the prices to be non-zero and positive and date to be in the format dd/mm/yyyy. I will put some incorrect validations.



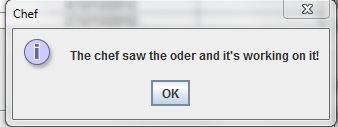
One error pop-up with a message. The waiter panel consists of two tables, one table consists of the menu items and their prices buttons and other objects of this type.

Introducing the orderId, table and date and pressing the “Order” button it shows in the second table all the info needed to be added into the bill generator, but, unfortunetly, the total price computation, on the GUI, it does not show correctly, but I have tested separately and it is working just fine.

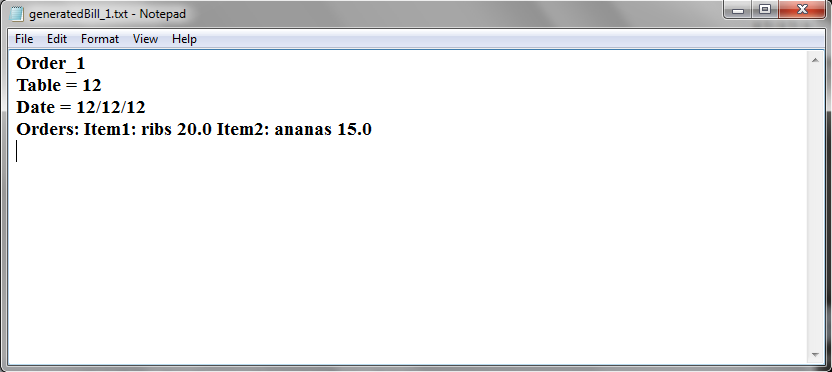




This is the error for the date because, I mentioned before, the date format was changed late and I did not want to clean the .ser file again. Below it is the message for the Chef, but not all the times it shows up in the GUI, but the implementation of it it is correct.



For the final image, a look at the generated bill that I will provide just two of them for the project.



**5. Results**

I tested before the creation of the GUI all the methods and worked, it might be the implementation of those calls of the methods that I did something wrong in order to show the correct price per order. There was no need of the validations, but I thought that a basic set is always needed to be safe. There was no problem in generating the bill after I introduced the order, but there is a little bit of lag or delay with the buttons, but it might be only on my computer. Maybe, for the future, I would like to implementa a login system and to be accessible to many actors and, perhaps, to some customers as well.

**6. Conclusions**

To sum up, implementing this project was a success, in my opinion, was creative and innovative to work with Hash mapping to easily store data at a given key, but also to design a lot of Frames form a application that serves a purpose in the real world. I think that I managed to better understand the value of generating data and save it in text files.

**7. Bibliography**

**Personal resources**

<https://stackoverflow.com/questions/15491894/regex-to-validate-date-format-dd-mm-yyyy>

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

<https://javarevisited.blogspot.com/2012/01/what-is-assertion-in-java-java.html>

<https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#tag>

<https://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html>

<http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_4/Assignment_4.pdf>

<http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_4/Assignment_4_Indications.pdf>