

FACULTY OF AUTOMATION AND COMPUTER SCIENCE

FUNDAMENTAL PROGRAMMING TECHNIQUES

ASSIGNMENT 4: RESTAURANT MANAGEMENT

Coordinator: Dr. Eng. Cristina Bianca Pop

Student: Bianca-Veronica Avram

2nd year, 2nd semester

2020

Contents

1. *Objectives*
2. *Problem analysis, modelling, scenarios, use case*
3. *Design*
4. *Implementation*
5. *Results*
6. *Conclusions*
7. *Bibliography*

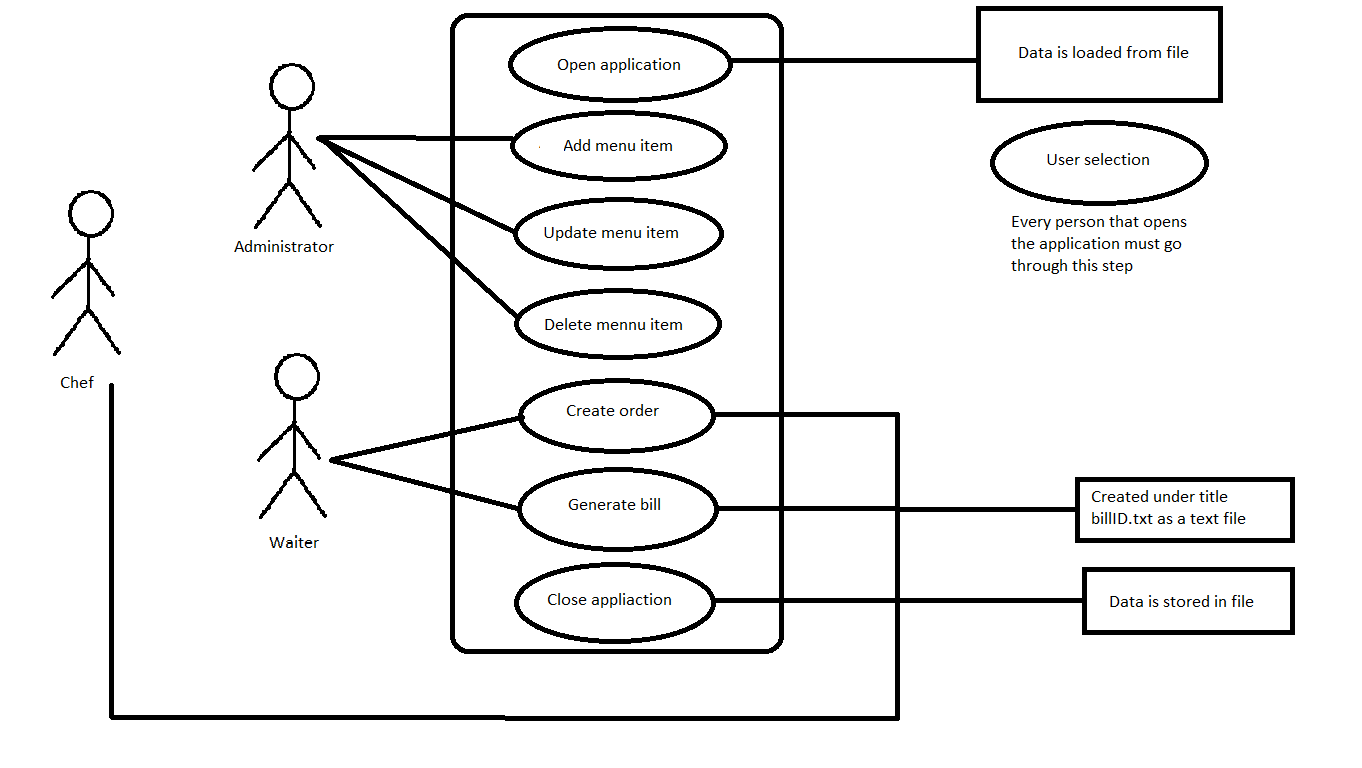
*1.Objectives*

This assignment has as main objective developing an application that manages a restaurant. There are three types of users which have different abilities inside the application: administrator, waiter and chef. The administrator works only with the menu. He can add items, delete items and edit items. The waiter interacts with the menu by placing orders. He can also generate a bill as a text file for a certain order. The chef is notified each time a new order is created and, so that he knows that he can start cooking.

As secondary objective, this assignment involved:

1. Using Composite design pattern
   * This part is described in the implementation part, namely section 3
   * This design pattern consists of 2 classes and an interface and was used for the classes BaseProduct and CompositeProduct, bot implementing MenuItem interface
   * Important to mention that the number of products in a CompositeProduct is 4 (this wass an implementation decision that is assumed by the author)
2. Implementing interface IRestaurantProcessing interface
   * That contains all the operation that can be performed on the restaurant data and are implemented in class Restaurant
   * This objective is further explained in section 4
3. Defining class Restaurant
   * This one contains the operations that can exist in the restaurant management system
   * It will be discussed in section 4
   * In implementing this point, more objectives were identified
4. Using HashMap
   * This will be presented in section 4
   * HashMap was used for the advantage of very fast memory access, namely O(1)
   * Some methods required to be overridden in order to work
5. Creating a graphical user interface
   * This is explained in section 2
   * It was practiced before, in a previous assignment
6. Using serialization
   * Widely presented in section 4
   * It is used for storing and loading the data about the restaurant, namely the menu and the orders placed in a file between successive runs of the application

*2.Problem analysis, modelling, scenarios, use case*



As described in the previous section, the application may go through these scenarios. The user first runs the application, then a frame containing a welcome message and three buttons appears. Using the buttons, the user selects the way he/she wants to “log in”(it is not properly, because there is only one user of each type and they do not require username or password). After selecting the user type, a new frame opens, corresponding to the type of user. In that frame, there are interface components that allow the user to execute their operations.

Like this, the administrator has the option of creating new menu item. When they press this button, some other components become visible and there, they can introduce the data they want to be consider a new product. Important mention here, after being content with the product created, the administrator must press the OK button, which, for some reason, can only be seen when the mouse goes over it while moving. It is situated just below the last combo box for item 4. However, when this button is pressed, the item is actually created and introduced in the menu.

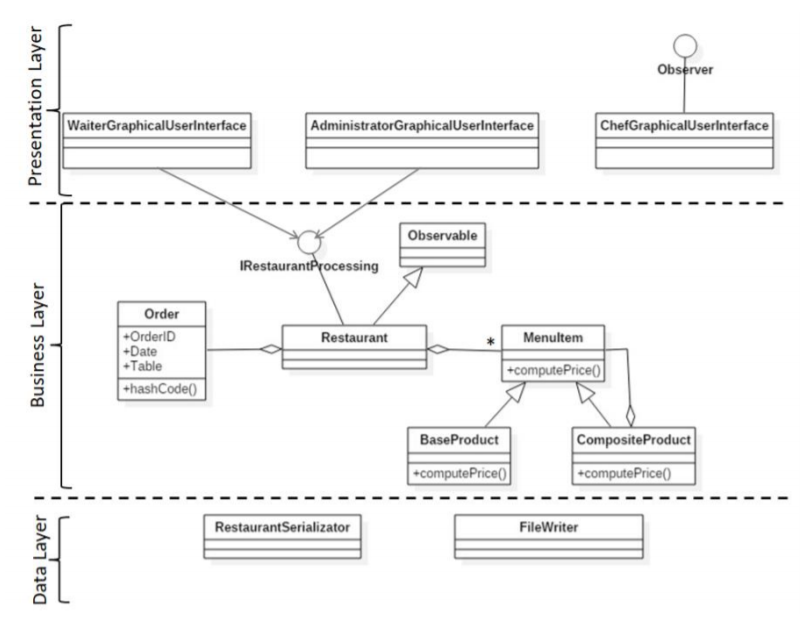
In this frame a scroll pane can be noticed. There the administrator can see and hide all the products in the menu, with name, price and componence, but only by pressing the view button. Then, by interacting with this table, the administrator can select an item from the menu and can choose to delete it or to edit it, by pressing the corresponding button. Note: these operations will only work if a selection in the table is made, otherwise, some messages are displayed. If the delete button is the option, then this happens immediately. If there is the edit option, then the components that appeared at create product are made visible again and there the info about the selected is transported to them so modifications can be made. Again, the edits will be saved after OK button is pressed.

The waiter frame is more simple. It has a table, where all the orders are displayed. It is similar to the one from the previously presented frame, but it only contains data about the order, i.e. id and date. The items ordered will be found in a combo box right below the table after a row is selected. At the stage, the waiter can press the compute bill button and a bill for the order is generated as a text file. Another thing the waiter can do is creating new orders. From the graphical point of view, this operation might seem a little bit complicated, because it requires some sort of instructions. When create order button is pressed, a combo box and two buttons appear. The combo box contains all the products in the menu an the first button is pressed when a product is chosen in order for it to be added to the current order. This step can be done multiple times and items will be added to the order. When the order is complete, the submit button is pressed and it can be noticed immediately that the order appears in the table.

Both the administrator and the waiter have a close button which is the way the application is stopped (the X implicit buttons of the frame are desactivated). When pressed, all the data from the restaurant is saved in restaurant.ser.

The frame for the chef is left empty, because he does not present a specific functionality.

*3.Design*



The figure above represents the starting point for the application design. There are three layers, which are concretely three packages. An additional package was added to contain the class with the main method. This one method contains the definition for the first frame that is opened and the action listeners for the three buttons that open the other frames, and also the invoking of the initialization methods for these frames.

Starting from the bottom, the data layer contains the necessary classes for writing the data into .ser file (and creating the file if it is the case). RestaurantSerializator also provide the starting point of the application, as it creates the object of class Restaurant on which the whole application works.

Business layer contains classes and interfaces that represent data for actually working inside the restaurant, as there are all the components of what a restaurant actually means in this context.

Here, the composite design pattern was used involving the classes BaseProduct, CompositeProduct and the interface MenuItem. These facilitate considering that there is just one type of menu item and it is stored in the restaurant collections, but it can actually be a BaseProduct od a CompositeProduct, The base product is a simple form that only contains name and price, and the composite product is based on a collection (in this case ArrayList) where many MenuItems may be stored. These MenuItems can actually be either BaseProducts or CompositeProduct. In fact, MenuItem interface has the necessary methods, which are furthermore implemented by the classes, that are just enough to keep track of the products in the menu or in an order, without really being important if it is a BaseProduct or a CompositeProduct.

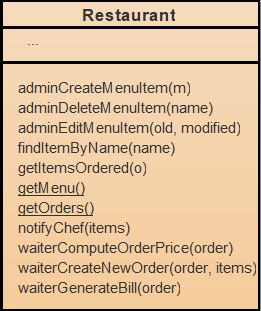
The observer design pattern is also present and it involves the classes Restaurant and ChefGUI and the interfaces Observer and IRestaurantProcessing. In Restaurant, the notify method is called in the method where a new order is created. This notification consists of adding the list of items in the order to the list of items of the chef, the items he must cook. This is done via update method which is called inside notifyChef.

In the end, there is the presentation layer that contains the frames corresponding to the three types of users. As they are GUI classes, they have more than 300 lines of code and they contain methods with more than 30 lines. All the components that can be seen on the frames were created and added to a certain place in the frame. This means that method setBounds was used for all these components.

*4.Implementation*

This section is dedicated to the actual coding and to presenting the classes composing the application and also certain implementation decisions. One that have been mentioned, but it is worth being discussed again is that a composite product can have maximum four components. This was done thinking about a realistic situation and also about the ease of implementation, because it was harder to create a table with all the menu products, without knowing the number of its columns, so it reduced to name, price, item1, item2, item3 and item4.

The classes BaseProduct, CompositeProduct, Order and the interfaces IRestaurantProcessing and MenuItem do not present any special implementation. Mainly, they contain the operations, methods, getters and getters that are also present in the requirements of the assignment. Class Restaurant will be discussed in the next paragraphs.



This class implements interface IRestaurantProcessing, so it implements the methods defined there. As fields, here are present two static ones which store the menu and the orders. They are static, as in the application there is only one menu and only one pack of orders, so in the whole application there will only be one single instance of the restaurant class.

In this class, the principal methods are the ones from the interface:

1. public void adminCreateMenuItem(MenuItem m)

This method simply adds the MenuItem given as parameter to the ArrayList<MenuItem> holding all the items in the menu of the restaurant. This operation will only be performed by the administrator, as required in the assignment text.

1. public void adminDeleteMenuItem(String name)

This method first searches the MenuItem with the name name in the menu in order to get the corresponding MenuItem instance and then deletes it from the collection. Only the administrator will be able to perform this operation.

1. public void adminEditMenuItem(MenuItem old, MenuItem modified)

This method replaces the old item in the menu with the new one.

1. public void waiterCreateNewOrder(Order order, ArrayList<MenuItem> items)

This method takes as arguments an order and the items associated to it and creates a record which is added in the HashMap.

1. public float waiterComputeOrderPrice(Order order)

This method iterates through all the items associated with the order given as parameter and calls the computePrice method for each. The result will be the sum of all prices.

1. public void waiterGenerateBill(Order order) throws IOException

This method is used by the waiter to create the text file with the bill for a certain order whit the name “bill<orderID>.txt”. It will be found in the directory where the program is run form.

Package data is more special, as it uses input and output streams ore reading to file the contents of the restaurant menu and restaurant orders. It is worth mentioning the format in which the data is saved: first, the number of menu items, then all the menu items, then the number of orders, the each order followed by the number of items ordered and the list of menu items ordered.

This method from FileWriter is the only one present in this class and creates the file “restaurant.ser” (actually the name is given as parameter). If the file exists, it will just be opened and the written. It is called each time a close button is pressed.

public static void writeFile(Restaurant restaurant, String filename) throws IOException{  
 File file = new File(filename);  
 file.createNewFile(); // if file already exists will do nothing  
 FileOutputStream streamfile = new FileOutputStream(file);  
 ObjectOutputStream out = new ObjectOutputStream(streamfile);  
 if(Restaurant.*getMenu*().size() != 0){  
 out.writeInt(Restaurant.*getMenu*().size());  
 for (MenuItem m:Restaurant.*getMenu*()) {  
 out.writeObject(m);  
 }  
 if(Restaurant.*getOrders*().size() != 0){  
 out.writeInt(Restaurant.*getOrders*().size());  
 for (Map.Entry en:Restaurant.*getOrders*().entrySet()) {  
 Order o = (Order)en.getKey();  
 out.writeObject(o);  
 int nr = Restaurant.*getOrders*().get(en.getKey()).size();  
 out.writeInt(nr);  
 for(MenuItem m : Restaurant.*getOrders*().get(en.getKey())){  
 out.writeObject(m);  
 }  
 }  
 }  
 }  
 out.close();  
 streamfile.close();  
}

This method from RestaurantSerializator is the only one from this class and is called each time the application is opened, creating the object of type Restaurant to contain the menu and the orders. It respects the sane format as at writing the file.

public static Restaurant restoreRestaurantData(String filename){  
 Restaurant restaurant = new Restaurant();  
 File file = new File(filename);  
 FileInputStream streamfile = null;  
 ObjectInputStream in = null;  
 int menusize = 0;  
 int ordernumber = 0;  
 try{  
 streamfile = new FileInputStream(file);  
 in = new ObjectInputStream(streamfile);  
 menusize = in.readInt();  
 }  
 catch (IOException ex){  
 return null;  
 }  
 for(int i = 0 ; i < menusize ; i++){  
 MenuItem m = null;  
 try{  
 m = (MenuItem) in.readObject();  
 }  
 catch (ClassNotFoundException | IOException ex){  
 return null;  
 }  
 restaurant.adminCreateMenuItem(m);  
 }  
 try{  
 ordernumber = in.readInt();  
 ArrayList<MenuItem> items = new ArrayList<>();  
 for(int i = 0 ; i < ordernumber ; i++){  
 Order o = (Order)in.readObject();  
 int count = in.readInt();  
 for(int j = 0 ; j < count ; j++){  
 MenuItem m = (MenuItem)in.readObject();  
 items.add(m);  
 }  
 restaurant.waiterCreateNewOrder(o,items);  
 }  
 in.close();  
 streamfile.close();  
 }  
 catch (IOException | ClassNotFoundException ex){  
 return restaurant;  
 }  
 return restaurant;  
}

Next on, the presentation package contains the most of the functionalities of the application. In these classes all the components of the graphic user interface are created and positioned in such a way the flow of the application makes sense. The only problem encountered in this moment is the ok button in AdministratorGUI, which is not visible from the first time, but only after the mouse passes over it. The ok button has the greatest functionality, as its action listener is responsible with saving the created product or the data about the edited one. There are actually two different buttons, but their actions are much alike.

String name = productName.getText().trim();  
if(item1.getSelectedItem().equals("None") && item2.getSelectedItem().equals("None") && item3.getSelectedItem().equals("None") && item4.getSelectedItem().equals("None")) {  
 float price = Float.*parseFloat*(productPrice.getText().trim());  
 MenuItem product = new BaseProduct(name,price);  
 restaurant.adminEditMenuItem(selected,product);  
}  
else{  
 MenuItem c1 = restaurant.findItemByName((String)item1.getSelectedItem());  
 MenuItem c2 = restaurant.findItemByName((String)item2.getSelectedItem());  
 MenuItem c3 = restaurant.findItemByName((String)item3.getSelectedItem());  
 MenuItem c4 = restaurant.findItemByName((String)item4.getSelectedItem());  
 CompositeProduct m = new CompositeProduct(name);  
 if(c1 != null)  
 m.addProduct(c1);  
 if(c2 != null)  
 m.addProduct(c2);  
 if(c3 != null)  
 m.addProduct(c3);  
 if(c4 != null)  
 m.addProduct(c4);  
 restaurant.adminEditMenuItem(selected,m);  
}

The data from the components is obtained and the objects that are of interest are creates and added to their place or edited. This snippet is from the action listener of the okEditButton.

*5.Results*

No testing framework was used in this assignment. The testing during developing was done on sample data so that items were added in the administrator frame, then the application must be reopen if the user has to be switched. Order data was the introduced in the waiter interface and the details appeared in the table. Then, when closing the application, all the data was saved on the file and ready to be reused when reopen.

*6.Conclusions*

For further implementation, more users may be created, so that there can be many waiters, many chefs and many administrators who can work on the data stored. There may be introduced even a database for storing the users, than a login graphical user interface may be implementing for this part. Also, the present graphical interface may also be improved by adding some more interactive features.

This assignment was a helpful hand in working again with java.swing, as new components, like JTable and JScrollPane, were used. Also, new methods in this context were discovered and used by the author.

Serialization is a process that have been experienced before, but it was a little bit more complicated this time, as more complex objects were serialized. Some problems were encountered at this stage, but the solution came eventually and the serialization is a success. But, there is a confusion for the author in this case. The .ser file is written in a specific way, so if data is not placed there as it was intended, hazards could happen. The file .ser must be empty in the beginning, because the application will not know how to parse it if it is not in the expected format.

Design pattern usage seemed to be a challenge in the beginning, but gradually, it seemed that it was the most easygoing problem in this application.

Working with HashMap was a new experience. Even though, so far (at least in this problem) it seems to complicate the application a lot.

Building the .jar file is always an adventure, but one was successfully obtained.

*7.Bibliography*

1. <https://howtodoinjava.com/java/date-time/compare-localdates/>
2. <https://www.geeksforgeeks.org/observer-pattern-set-2-implementation/>
3. <https://www.geeksforgeeks.org/composite-design-pattern/>
4. <http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_4/Assignment_4_Indications.pdf>
5. <https://www.javatpoint.com/java-jframe>
6. <https://javatutorial.net/java-swing-jframe-layouts>
7. <https://www.baeldung.com/java-write-to-file>
8. <https://www.codejava.net/java-se/swing/jcombobox-basic-tutorial-and-examples>
9. <https://docs.oracle.com/javase/tutorial/uiswing/components/table.html>
10. <https://javatutorial.net/jframe-buttons-listeners-text-fields>
11. <https://stackoverflow.com/questions/11999560/get-combobox-value-in-java-swing>
12. <https://www.javatpoint.com/java-joptionpane>
13. <https://www.javatpoint.com/java-fileoutputstream-class>
14. <https://www.javatpoint.com/java-fileinputstream-class>
15. <https://docs.oracle.com/javase/7/docs/api/java/io/ObjectInputStream.html>
16. <https://docs.oracle.com/javase/tutorial/displayCode.html?code=https://docs.oracle.com/javase/tutorial/uiswing/examples/components/TableRenderDemoProject/src/components/TableRenderDemo.java>