Documentation

Restaurant Management System

Student: Rusu Daniel
Group: 30422

Table of Contents:

1. Objective

Main objective of the assignment:
Consider implementing 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 for a table, add elements from the menu, and compute the bill for an order. The chef is notified each time it must cook food ordered through a waiter.

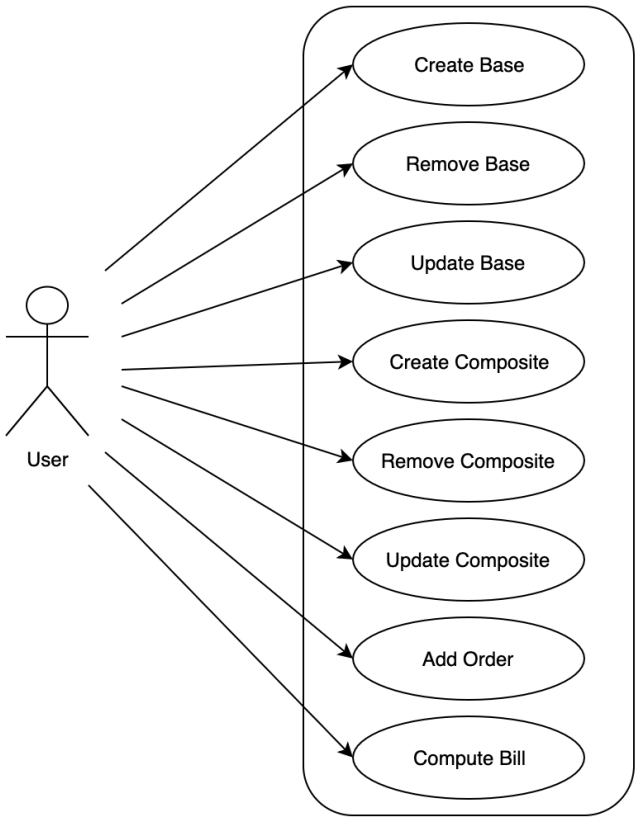
Secondary objectives:
- Creating a good-looking, fully-functional user interface
- Implementing the provided UML class diagram
- Implementing algorithms to run according to the main objective.
- Linking the algorithms to the graphical user interface, in order to display the right output to the user, and to receive the right input from the user.

All the secondary objectives presented above will be detailed in the fourth chapter of this documentation.

2. Problem analysis, modelling, use-cases

This application is built to perform operations usually used by a restaurant administrator, waiter and chef. It shows to the user the menu of the restaurant, and allows him/her (in the “Administrator” window) to modify its contents: add, remove or edit a certain item. The main goal of the application is to allow the management of a restaurant system: keep in mind the restaurant menu, edit it, allow the waiter to place an order and compute a bill, and allow to chef to be notified when a new order has been placed.

To better illustrate how the application can be used, I have built a use-case diagram:

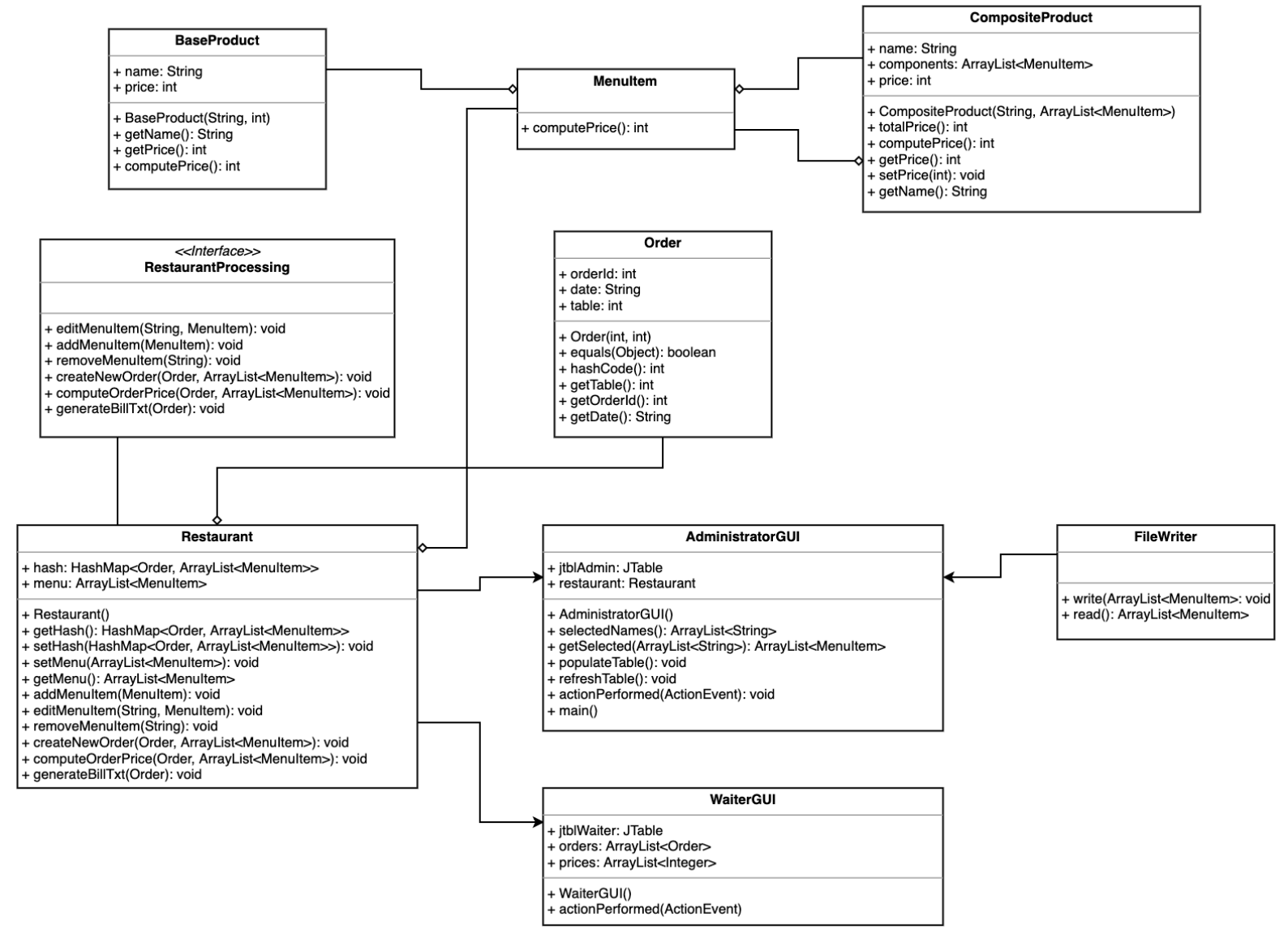


Use-cases description:

- **Create Base:** Input for the administrator window, used to create a new base product and insert it in the restaurant menu: the name and price are read from the text fields provided in the Administrator UI, then the constructor for Base Product is called, and the result is added to the restaurant “menu” list. The JTable in the UI will be updated after each button press.
- **Remove Base:** Input in the administrator window, used to remove an existing base product from the restaurant “menu” list. The user is able to delete the menu item by selecting it and pressing the “Remove Base” button: the item will be searched in the restaurant menu (by name) and removed. The JTable in the UI will be updated after each button press.
- **Update Base:** Input used in the administrator window. The process is similar to the “Remove Base” case: the item will be found by name in the restaurant “menu” list and then replaced with the ‘new’ item. In order to have a name and price, the user must provide them in the text fields that are present in the UI.
- **Create Composite:** Input used in the administrator window, in order to insert a new composite product in the restaurant “menu” list. The name of the product is read from the text field, and the components (both base and composite products) are read form the JTable (the user should first select them in the table, then press the button). Then the constructor for the class “CompositeProduct” is called, and the result is added to the restaurant “menu” list.
- **Remove Composite:** Input used in the administrator window, in order to delete a composite product from the restaurant “menu” list. It uses the same methods as the “Remove Base” input, in order to find and remove the item from the menu.
- **Update Composite:** Input used in the administrator window, in order to modify the contents of a composite product. The name of the product to be edited is read from the text field, then it is searched in the menu list and assigned the components selected in the JTable (the user should first select the items in the table, then press the button).
- **Add Order:** Input used in the waiter window, in order to create a new order. The application does this by creating a new entry in the “hash” HashMap. The key (K) is computed from the order received as parameter, and the value (V) is represented by an ArrayList<MenuItem> (the component list received as parameter). The order table and total price are placed in a JTable, in the waiter window, in order to illustrate the existing order.
- **Compute Bill:** Input used in the waiter window, in order to create a .txt representation of the bill, for the order selected in the JTable.

3. Design (Decisions, UML Diagram, Class Design, User Interface)

The software design approach is OOP style. The application uses a system of interacting objects for the purpose of solving the proposed problem.



The classes are illustrated as rectangles divided into 3 sections: the first one (up) contains the name of the class, the second (middle) contains the class fields and the third (down) contains the implemented class methods.
The relationships between different classes are represented using different types of links. There are multiple 'aggregation' relationships drawn on the diagram. For example, the ones drawn between MenuItem, BaseProduct and CompositeProduct: a menu item can contain one or more base/composite products. A composite products can contain one or more menu items. This is part of the Composite Design Pattern present in the application, and represents the "menu" list of the restaurant.
There is an "aggregation" type of link between the MenuItem class and Restaurant class, because the restaurant menu contains one or more menu items. This menu is represented in a JTable in the AdministratorGUI class, and is read from a file using the FileWriter class, hence the links between these classes on the UML diagram.

4. Implementation

The application consists of 11 classes, divided in 3 different packages: data, logic and ui. All the classes will be presented and detailed below.

• The BaseProduct Class (logic package)

This is the class used to represent base products in the restaurant menu list. It contains all the specific fields: name and price.
The most significant methods in this class are:

- BaseProduct: This is the constructor of the class. It receives as parameter a name and a price.
- getters for name and price: used in other classes, in order to provide the requested fields.
- computePrice: This method is inherited from the class MenuItem and is used to calculate the price of the base product it was called on.

• The CompositeProduct Class (logic package)

This is the class used to represent the composite product in the restaurant menu list. It contains the fields: name and components (a list of all the items used to create the composite item). This class is part of the composite design pattern, specified above.
The most significant methods in this class are:

- CompositeProduct: This is the constructor of this class. It creates a composite product based on the name and list of components received as parameters.
- totalPrice: method used to compute the total price of the components.
- computePrice: method used to set the price of the composite product. It returns the result of totalPrice
- getters used in other classes, in order to provide the requested fields.

- **The MenuItem Class (logic package)**

This is an abstract class, inherited by the BaseProduct and CompositeProduct classes. It contains no fields, the only method is computePrice, that has to be implemented in both BaseProduct and CompositeProduct

- **The Order Class (logic package)**

Class used to represent the order. It contains the specific fields: orderId, date, table.
The most important methods implemented in this class:

- Order: the constructor method of the class. It receives as parameters the orderId and the table where the order was placed. This constructor is called whenever an order is placed by the waiter, hence its importance.
- equals: this method is overridden from the Object class. It is used when interacting with the HashMap, in order to find the requested value.
- hashCode: this method is also overridden from the Object class. It is used when interacting with the HashMap, in order to compute a good hash key for storing the value.
- getters used in other classes in order to provide the requested fields.

- **The Restaurant class (logic package)**

Class used to represent the restaurant. It contains the restaurant menu list (field menu), used mainly in the AdministratorGUI class, and the restaurant hash (field hash), used mainly in the WaiterGUI class.
The most important methods implemented in this class:

- Restaurant: this method is the constructor of the class, used in the AdministratorGUI class, in order to create a restaurant to work on.
- addMenuItem: method used to add a MenuItem (received as parameter) in the “menu” list. It is used whenever the “Create Base” or “Create Composite” is pressed. The data required is read from the user interface, then a new MenuItem is created and passed as a parameter to this method.
- editMenuItem: this method is used to modify the contents of the menu list. It can receive as parameter either a base product or a composite product. It replaces the element by searching for its name (also received as parameter) and using the “set” method on the ArrayList. It is used whenever the user presses the “Update Base” or “Update Composite” button.
- removeMenuItem: method used to delete an item from the menu. It receives its name as parameter, then finds it in the menu ArrayList and removes it. It is used whenever the user presser the “Remove Base” or “Remove Composite” button.
- computeOrderPrice: method used in the WaiterGUI class, in order to calculate the total price of the order that has just been placed. It receives the order and its components (as a list of menu items), and returns an integer representing the final price. It is used whenever the user of waiter window presses the button “Add Order”.
- generateBillTxt: method used to create a .txt file, that contains some information about the order. It receives as parameter the order that it must write about: order id, table number, date and total price. It is used whenever the user of the waiter window presses the button “Compute Bill”.

5. Results

When running the application, the user can observe some of the results directly in the user interface, while some are only visible outside of the interface (the “Compute Bill” button press result). For example, whenever the “Create Base” or “Create Composite” button is pressed, the result can be observed immediately in the JTable of the Administrator window. This also happens in the Waiter window, when the “Add Order” button is pressed: the table and total price of the order are immediately displayed in the JTable.

6. Conclusions

Building this application was interesting, because of all the concepts required. I feel like I have learned a lot, and there are still a lot of things to learn and improve.

Future developments:
- Design by Contract for class Restaurant
- Observer Pattern for class ChefGUI
- Creating a login process for each of the users: Administrator, Waiter, Chef
- Creating more functionality regarding the Restaurant class.

7. Bibliography

- Pdf presentation document received on the link below:
http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/HW4_Tema4/HW4_Tutorial_Hashing_In_Java.pdf

- Java reflection tutorials:
https://www.geeksforgeeks.org/reflection-in-java/

- Java Swing tutorials:
https://docs.oracle.com/javase/tutorial/uiswing/layout/gridbag.html

https://www.javatpoint.com/java-swing

- Java composite design pattern tutorials:
https://www.geeksforgeeks.org/composite-design-pattern/