TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA
ROMANIA

Restaurant Management

Student: Bețianu Miruna – Laura
Group: 30422

Table of Contents

1. TASK **3**

2. PROJECT SPECIFICATIONS **3**

5. COMPOSITION **4**

1. PACKAGES 4
2. CLASSES 4

THE UML CLASS DIAGRAM: 4

6. IMPLEMENTATION **5**

- *BaseProduct (package business)* *5*
- *CompositeProduct (package business)* *5*
- *IRestaurantProcessing (package business)* *5*
- *MenuItem (package business)* *6*
- *Order (package business)* *6*
- *Restaurant (package business)* *6*
- *FileWriter (package data)* *7*
- *RestaurantSerializator (package data)* *7*
- *Controller (package controller)* *7*
- *AdministratorGUI (package prresentation)* *7*
- *WaiterGUI (package prresentation)* *8*
- *ChefGUI (package presentation)* *8*
- *Main (package start)* *9*

7. USER INTERFACE (HOW TO USE THE APPLICATION) **9**

8. FURTHER DEVELOPMENTS **12**

9. CONCLUSION **12**

10. BIBLIOGRAPHY **12**

1. Task

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.

2. Project specifications

1. Define the interface RestaurantProcessing containing the main operations that can be executed by the waiter or the administrator, as follows:
    - Administrator: create new menu item, delete menu item, edit menu item
    - Waiter: create new order; compute price for an order; generate bill in .txt format.
2. Define and implement the classes from the class diagram shown above:
    - Use the Composite Design Pattern for defining the classes MenuItem, BaseProduct and CompositeProduct
    - Use the Observer Design Pattern to notify the chef each time a new order containing a composite product is added.
3. Implement the class Restaurant using a predefined JCF collection which uses a hashtable data structure. The hashtable key will be generated based on the class Order, which can have associated several MenuItems. Use JTable to display Restaurant related information.
    - Define a structure of type Map<Order, Collection<MenuItem>> for storing the order related information in the Restaurant class. The key of the Map will be formed of objects of type Order, for which the hashCode() method will be overwritten to compute the hash value within the Map from the attributes of the Order (OrderID, date, etc.)
    - Define a structure of type Collection<MenuItem> which will save the menu of the restaurant. Choose the appropriate collection type for your implementation.
    - Define a method of type “well formed” for the class Restaurant.
    - Implement the class using Design by Contract method (involving pre, post conditions,

invariants, and assertions).

4. The menu items for populating the Restaurant object will be loaded/saved from/to a file using Serialization.

5. Composition

1. Packages

- business
- data
- presentation
- start

2. Classes

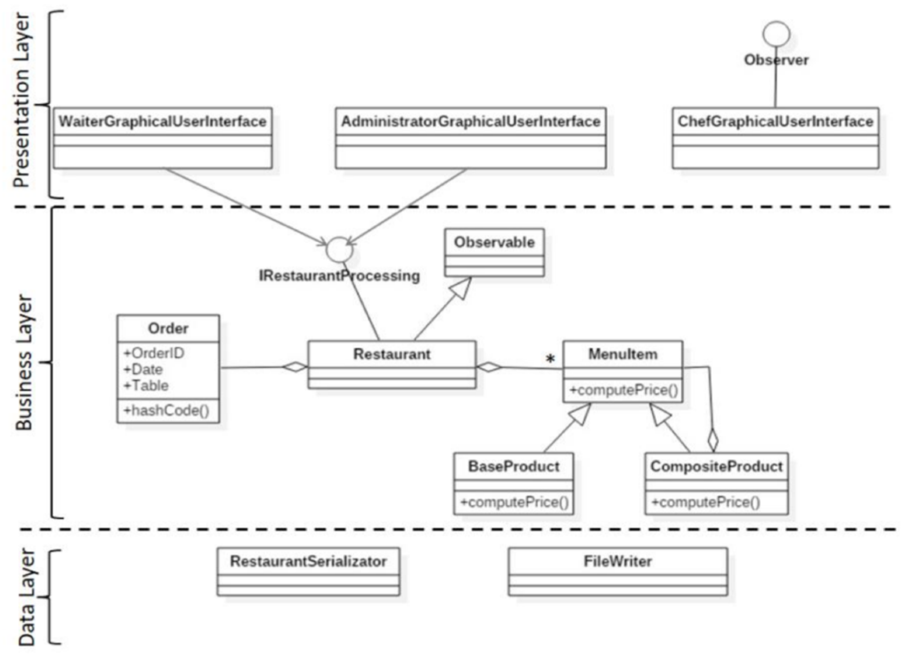
- BaseProduct
- CompositeProduct
- IRestaurantProcessing
- MenuItem
- Order
- Restaurant

- FileWriter
- RestaurantSerializator

- AdministratorGUI
- ChefGUI
- WaiterGUI
- Controller

- Main

The UML Class Diagram:



6. Implementation

BaseProduct (package business)

This class describes the object 'BaseProduct. Each base product is identified by some particular attributes: name and price.

The 'name' is the unique feature of the base products (there will be no more than one base product with a specific name).

The price is an Integer that could take only positive values.

Methods:

- setName(name): sets the name of the base product;
- getName(): returns the name of the base product;
- setPrice(price): sets the price of the base product;
- getPrice(): returns the price of the base product;
- computePrice(): returns the price of the base product;
- BaseProduct(name, price): creates the object of type Base Product having the properties received as parameters;

CompositeProduct (package business)

This class describes the object 'CompositeProduct'. Each composite product is identified by some particular attributes: name, price and a list of products(could be any object of type MenuItem).

The 'name' is the unique feature of the composite products (there will be no more than one composite product with a specific name).

The price is an Integer that could take only positive values.

Methods:

- setName(name): sets the name of the composite product;
- getName(): returns the name of the composite product;
- setPrice(price): sets the price of the composite product;
- getPrice(): returns the price of the composite product;
- setMenuItems(menuItems): sets the list of products of the composite product;
- getMenuItems(): returns the list of products of the composite product;
- computePrice(): returns the total price of the composite product;
- CompositeProduct(name, menuItems): creates the object of type Composite Product having the properties received as parameters;

IRestaurantProcessing (package business)

This interface controls the execution of the restaurant. In this class we have all the needed functions in order to control the restaurant.

Methods:

- addNewCompositeProduct(name, menuItems);
- addNewBaseProduct(name, price);

- removeProduct(menuItem);
- editProduct(menuItem, object, menuItems);
- addOrder(order, menuItems);
- computePriceOrder(order);
- getMenuItems();
- setMenuItems(menuItems);

MenuItem (package business)

This abstract class controls the menu items. In this class we have all the needed functions in order to control the operations that are necessary for the products.

Methods:

- computePrice();
- getName();
- setName();
- getMenuItems();
- setMenuItems();
- getPrice();
- setPrice ();

Order (package business)

This class describes the object ‘CompositeProduct’. Each composite product is identified by some particular attributes: id, date, table.

The ‘id is the unique feature of the composite products (there will be no more than one composite product with a specific name).

The table is an Integer that represents the table number of the order. The date is an object of type Date introduced by the waiter when wanting to place an order.

Methods:

- setOrderId(id): sets the id of the order;
- getOrderId(): returns the id of the order;
- setDate(date): sets the date of the order;
- getDate (): returns the date of the order;
- setTable(table): sets the table of the order;
- getTable(): returns the table of the order;
- Order(id, table, date): creates the object of type Order having the properties received as parameters;

Restaurant (package business)

This class describes the object ‘Restaurant. Each restaurant is identified by some particular attributes: a map of orders and a menu.

Methods:

- addNewCompositeProduct(name, menuItems): adds a new composite product to the menu.
- addNewBaseProduct(name, price): adds a new base product to the menu.
- removeProduct(menuItem): removes a menu item (base or composite) from the menu.
- editProduct(menuItem, object, menuItems): edits a menu item (base or composite) from the menu.
- addOrder(order, menuItems): adds a new order to the map of orders together with the list of menu items from that order.
- computePriceOrder(order): computes the final price of the order.
- getMenuItems(): returns the menu items of the restaurant.
- setMenuItems(menuItems): sets the menu items of the restaurant.

FileWriter (package data)

This class is the one responsible with creating the pdf bill of a specific order from the restaurant.

Methods:

- FileWriter(restaurant, order): creates a table with the menu items from the order and then generates a pdf representing the bill of the order; final price will be also shown.

RestaurantSerializator (package data)

This class is the one responsible with reading and printing from and into a file the menu items of the restaurant.

Methods:

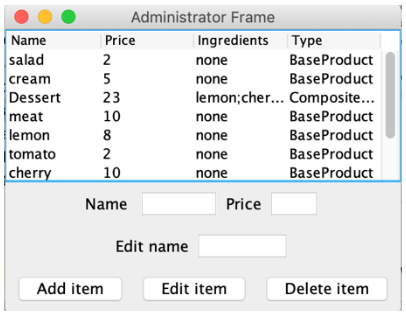
- RestaurantSerializator(fileName): constructor;
- writeData(restaurant): write the menu of the restaurant into a file;
- readData(restaurant): reads the menu of the restaurant from a file.

Controller (package controller)

This class implements the action listeners designed for the buttons declared in the frames: administrator, chef and waiter. It has a couple of inner classes for the objects needed to be used in the frames. The inner classes have as method 'actionPerformed', method from the interface 'ActionListener'.

AdministratorGUI (package presentation)

This class creates the graphical user interface of the administrator.



Name	Price	Ingredients	Type
salad	2	none	BaseProduct
cream	5	none	BaseProduct
Dessert	23	lemon;cher...	Composite...
meat	10	none	BaseProduct
lemon	8	none	BaseProduct
tomato	2	none	BaseProduct
cherry	10	none	BaseProduct

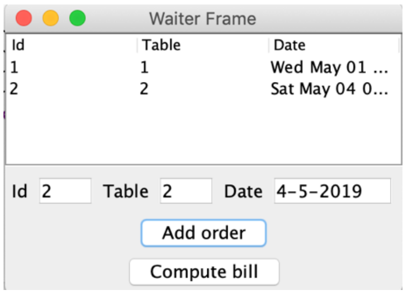
Each text field, label and the button needed in order to create this interface is created and instantiated in this class.

Methods:

- AdministratorGUI(): constructor of the class;
- newFrame(): method that opens a new frame once the button 'Add item' or 'Edit item' is pressed (without completing the price!).
- getName():returns the name of the menu item;
- getPrice(): returns the price of the menu item;
- getSecondaryName(): returns the name to edit;
- fillInTable(): fills the table with some particular values;

WaiterGUI (package presentation)

This class creates the graphical user interface of the waiter.



Id	Table	Date
1	1	Wed May 01 ...
2	2	Sat May 04 0...

Id 2 Table 2 Date 4-5-2019

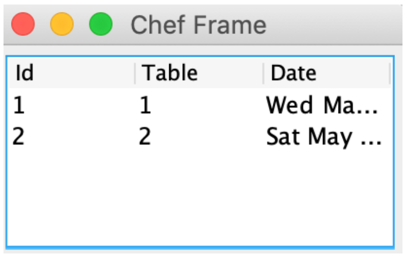
Each text field, label and the button needed in order to create this interface is created and instantiated in this class.

Methods:

- WaiterGUI(): constructor of the class;
- newFrame(): method that opens a new frame once the button 'Add order is pressed.
- getId():returns the id of the order;
- getTable(): returns the table number of the order;
- getDate(): returns the date of the order;
- getSecondaryName(): returns the name to edit;
- fillInTable(): fills the table with some particular values;

ChefGUI (package presentation)

This class creates the graphical user interface of the chef.



Chef Frame

Id	Table	Date
1	1	Wed Ma...
2	2	Sat May ...

Each text field, label and the button needed in order to create this interface is created and instantiated in this class.

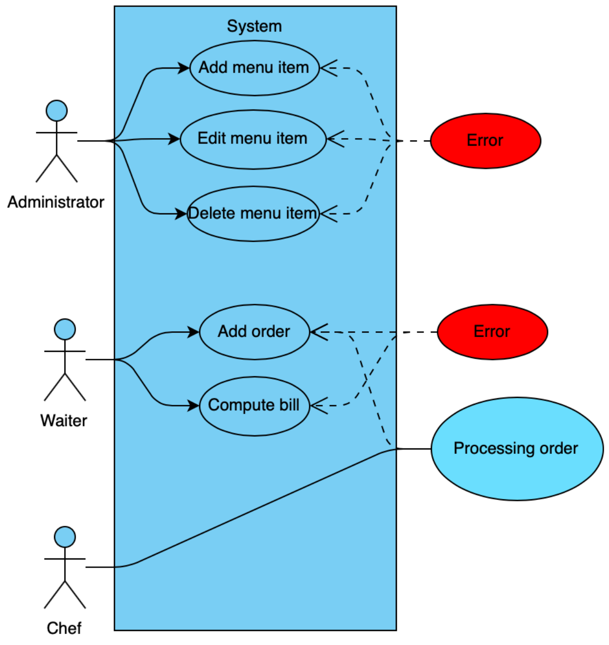
Methods:

- ChefGUI(): constructor of the class;
- update(): the chef of notified when some changes are spotted in the restaurant. If a new order is placed, the chef will be notified.
- fillInTable(): fills the table with some particular values;

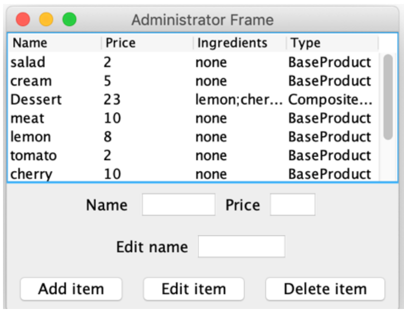
Main (package start)

This class instantiates all the objects that are needed in order to make the application work.

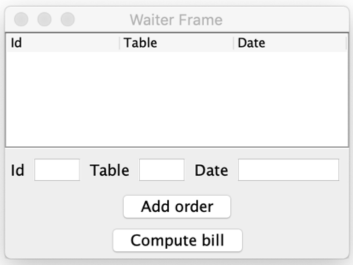
7. User Interface (how to use the application)



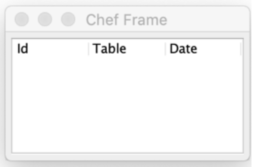
Use-Case Diagram


Administrator Frame

Name	Price	Ingredients	Type
salad	2	none	BaseProduct
cream	5	none	BaseProduct
Dessert	23	lemon;cher...	Composite...
meat	10	none	BaseProduct
lemon	8	none	BaseProduct
tomato	2	none	BaseProduct
cherry	10	none	BaseProduct


Waiter Frame

Id	Table	Date

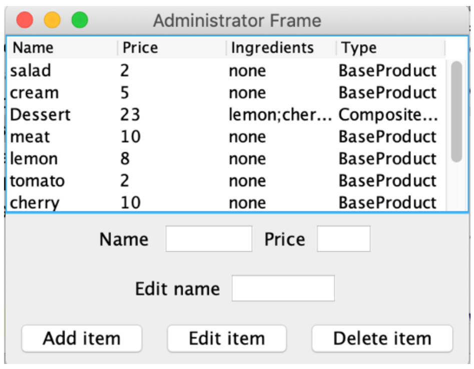

Chef Frame

Id	Table	Date

⇒ At a first glance, we observe that the graphical interface has some labels, some text fields and some buttons;

⇒ Attention: It is very important to introduce valid data in all the text boxes!

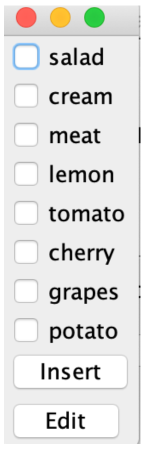
⇒ How to use the administrator frame:


Administrator Frame

Name	Price	Ingredients	Type
salad	2	none	BaseProduct
cream	5	none	BaseProduct
Dessert	23	lemon;cher...	Composite...
meat	10	none	BaseProduct
lemon	8	none	BaseProduct
tomato	2	none	BaseProduct
cherry	10	none	BaseProduct

⇒ If the admin wants to insert a base product, the only fields that are needed to be completed are name and price. After pressing 'Add item' button, the product is added in the menu and it will be shown in the table above.

⇒ If the admin wants to insert a composite product, the price will NOT be completed. If the price isn't given, a new form will pop up and the admin will have to check some check boxes from a list as given below:

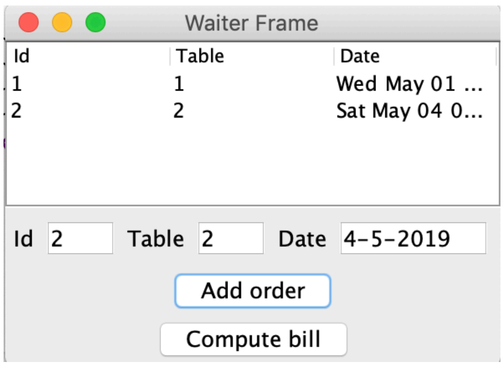


After checking the desired products, the button 'Insert' will be pressed and the composite product will be added to the menu.

⇒ If the administrator wants to edit a product, this can be done by pressing the 'Edit' button. It is done the similarly to the inserting method.

⇒ In order to delete a product, only the name of the product is required.

⇒ How to use the waiter interface:


Waiter Frame

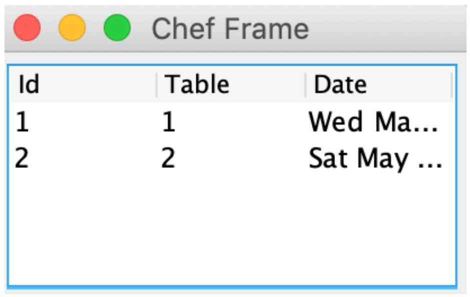
Id	Table	Date
1	1	Wed May 01 ...
2	2	Sat May 04 0...

Id 2 Table 2 Date 4-5-2019

⇒ The waiter will introduce data in the text boxes; after that, an order can be added pressing the 'Add order' button.

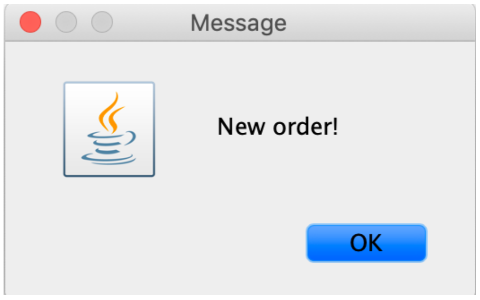
⇒ In order to compute a bill, the waiter needs to introduce a valid id from ones presented in the table above, and then hit the 'Compute bill' button. A pdf with the bill will now be stored in project's location.

⇒ How chef interface works:


Chef Frame

Id	Table	Date
1	1	Wed Ma...
2	2	Sat May ...

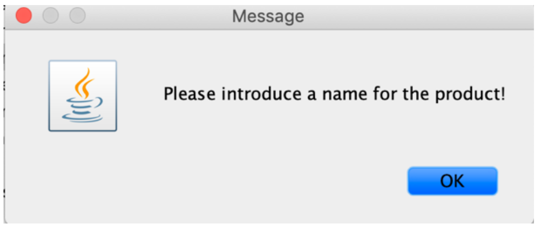
⇒ The info displayed are representing the orders. Each time a waiter introduces a new order, the chef will be notified.


Message

New order!

What happens if the user introduces an invalid input?

This:


Message

Please introduce a name for the product!

Also, all the cases of errors are considered. The waiter needs to introduce a valid date respecting a pattern, the admin cannot introduce a product will negative price and the text boxes need to be completed in most of the cases.

8. Further developments

Some of the functionalities of this application can be extended further in order to be more complex and to give the user a real – life restaurant simulation. For instance, we can develop the application so that it could have a proper restaurant simulation, like setting a time for the chef in order to prep the food and then notify the waiter if the order is ready. Also, it would be nice having the food distributed by different categories, having allergens and menus for kids.

9. Conclusion

To conclude, creating this application developed my knowledge about java programming language and how to work with the graphical interface (java.swing). Also, it helped me understanding how exceptions work by making me doing research about how to use them and how to mould them in order to be useful for my project. I consider that this project was a little bit challenging because I had never worked with programming techniques like design by pattern or composite design pattern. It improved my skills regarding working with files in Java.

10. Bibliography

http://stackoverflow.com (helped me to fix some bugs)
https://www.geeksforgeeks.org (learned about design by contract and how to work with the asserts)