Sabou Cezara
Computer Science, year 2
group 30421

Restaurant Management

1.Objectives

1.1Main objective

The main objective of this project is to create a Java Application that could provide the user the ability to simulate the management of a restaurant. It should be able to give the correct result in the Graphical User Interface

1.2 Secondary objectives

The secondary objectives derive from the main objective:

1. The ability to understand the process of communication between files and different classes
2. Being able to convert the given input into corresponding adequate candidates for processing;
3. Understanding the necessary steps to convert and process the given infromation;
4. Implementing a GUI(Graphical User Interface)

2. Problem Analysis

Restaurant management is a very common thing, and so, by taking a daily problem : deriving the steps in order for a menu item to first be created, then added to the menu, then selected by a costumer, notifying the chef about the choice and then computing the bill by the waiter, gives the user a sense of understanding about how this process actually works.

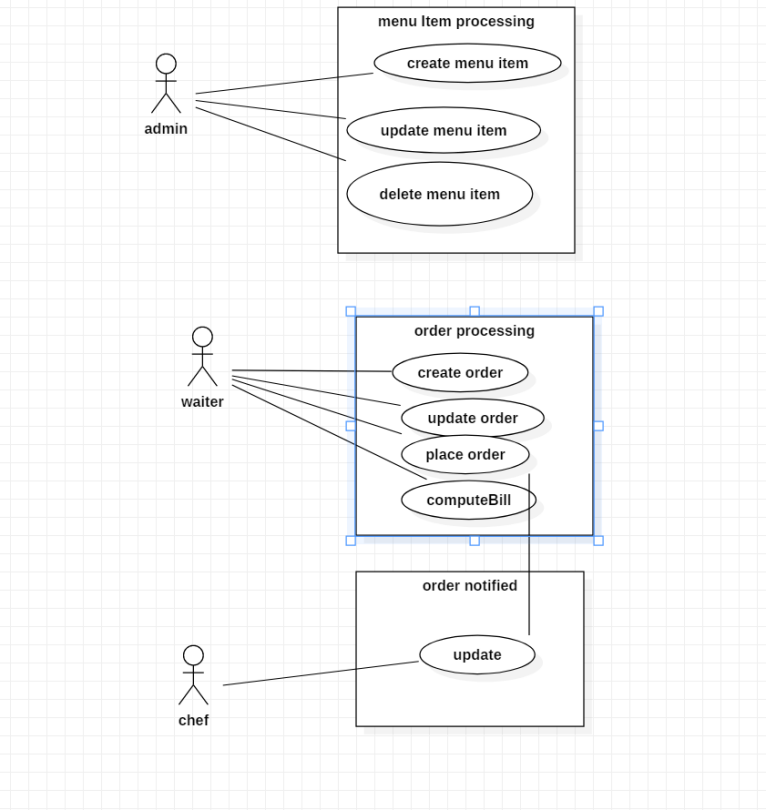
I will describe the functionality of the system in the following paragraphs.

Before manipulating menuItems, we firstly need to obtain them. We easily notice that there are very different kinds of menu items in a regular restaurant menu, but the most important ones are the products that are sold by themselves and the products that contain a subset of the menu items listed. Those are a special variety of menu items, as they contain several inside of them. However, even if this obvious difference is there, for the regular costumer, there is none. The regular costumer will not care if the product selected contains a list of product or it comes by itself. It will only care to see the price, the name, and the weight.

From the administrator's perspective though, the kind of menu item matters, and so, the creation or the modification or even the removal of a meu item will have to differ from one menu item to another. So whenever the administrator will work with this application it will know what items to choose from the list in order to create a new composite product.

After inserting the information about the menuItems, the data is processed and written into a file through serialization.

Sabou Cezara
Computer Science, year 2
group 30421



The use case diagram presents :

Use case title: admin menuItem processing

- Summary: this use case allows the admin to process menuItems
- Preconditions: administrator window opens correctly
- Main success scenario:
  - Administrator selects the menu item
  - Administrator selects the kind of operation it wants to do
  - The corresponding window opens
  - Administrator correctly inserts the information about the menu item
  - The result is displayed in the result text field and in the JTable
- Alternative sequences:
  - The input is not valid
  - Scenario returns to first step

Use case title : waiter order processing

- Summary : use case allows waiter to take an order, notify the chef and compute the bill
- Precondition: waiter window correctly opens
- Main success scenario:
  - Waiter correctly creates the order
  - Waiter adds the requested menu items from the specific table
  - Waiter places the order to notify the chef

- Waiter successfully computes the bill

Alternative sequences :

- Waiter does not alter the correct order

3.Design

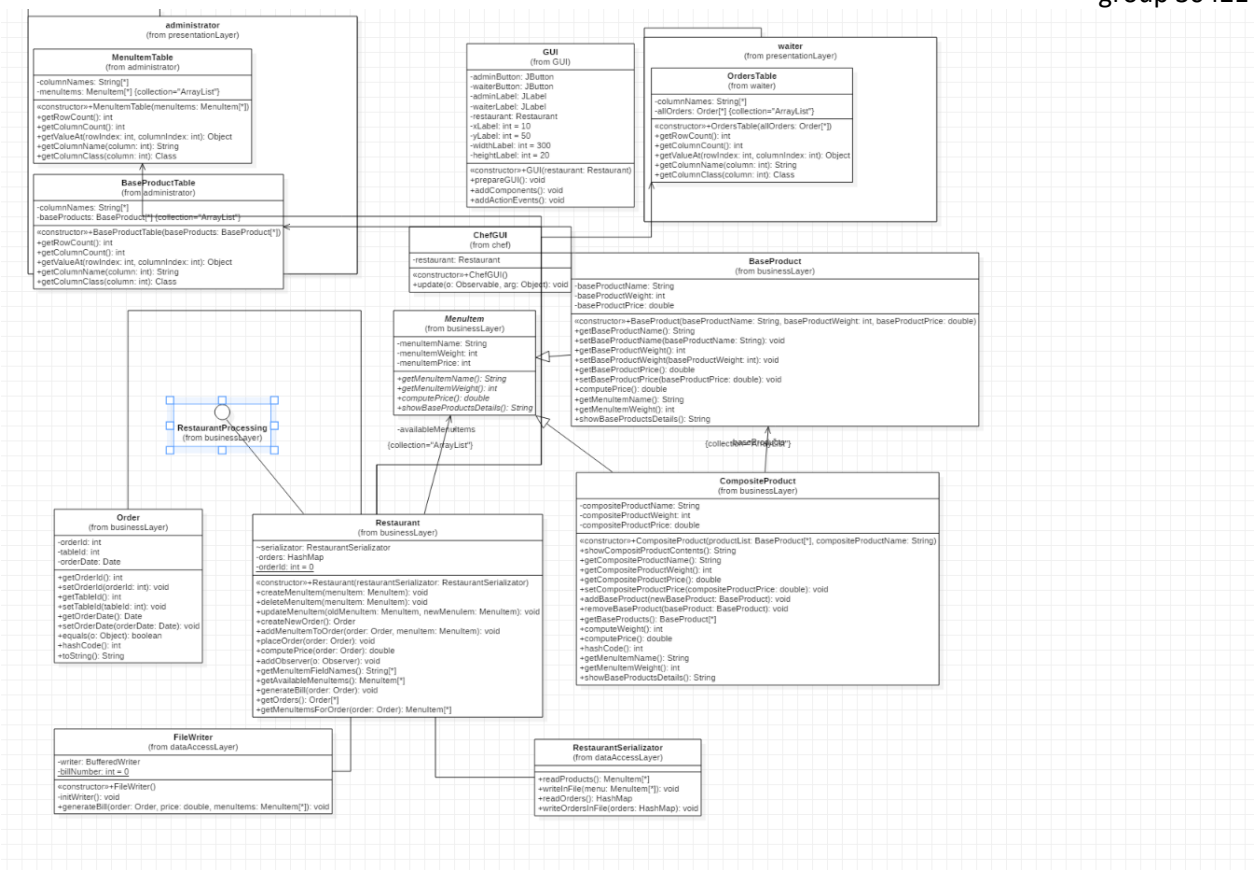
The structure of the application is based on 5 main different packages, some of them being structured in the business layer, data access layer and presentation layer.

The application is designed to take the information from the restaurant from a bitstream file through serialization, and associate orders with sets of menu items, in order to gain fast access to each order items.

Each time a piece of information is altered, the restaurant data will be re-written into the file.

I created this UML diagram in order to show the main relationships of the application and to accentuate the MVC pattern structure.

Sabou Cezara
Computer Science, year 2
group 30421



4.Implementation

In the business layer package:

4.1 Menu Item

Abstract class which actually will be inherited bu BaseProduct Class and CompositeProduct Class. They hold the information about the name, weight and price of the menu items displayed on the menu.

The composite design pattern was used in order to achieve this kind of structure.

4.2 Order

The Order class is made of simple attributes, such as: order id, order table number and order date, which will later be useful for computing the bill.

4.3 Restaurant

The Restaurant class is by far the most important one, it creates the link between the order and the menuitems.

Sabou Cezara
Computer Science, year 2
group 30421

The menu items are kept in an arraylist, and the orders represent the keys of the hashmap. In this way we are given instant access to the items in a specific order, and so it will make the access easier for the waiter.

It implements the Restaurant Processing interface, which has the purpose of being able to keep track of the implemented methods that are required for the functionality of the application. The class also extends the Observable Class, which means it will be observed by the classes in the project which extend the Observer class, each time this class calls the method: notifyAllObservers(). The methods which are present in this class are the ones at the core of this application and they tie all the other classes together. It links the presentation layer and the data access layer. There is a set of methods for the administrator: createMenuItem, UpdateMenuItem and deleteMenuItem, and also a set of methods for the waiter: createOrder, UpdateOrder, PlaceOrder and ComputeBIll. Through this class we are also given access to the separate order list and menu item list, which will help for creating the graphical user interface.

The most important thing to mention in this class is that the data is kept into a hashmap, and that the information is extracted and written into a file through serialization. I also inserted some assert invariants, in order to ensure correctness of the project.

```
//administrator
@Override
public void createMenuItem(MenuItem menuItem) {
    availableMenuItems.add(menuItem);
    serializator.writeInFile(availableMenuItems);
}

@Override
public void deleteMenuItem(MenuItem menuItem) {
    availableMenuItems.remove(menuItem);
    serializator.writeInFile(availableMenuItems);
}

@Override
public void updateMenuItem(MenuItem oldMenuItem, MenuItem newMenuIem) {
    availableMenuItems.remove(oldMenuItem);
    availableMenuItems.add(newMenuIem);
    serializator.writeInFile(availableMenuItems);
}


/**
 *
 * @return newly created order
 */
@Override
public Order createNewOrder() {
    Date date = new Date();
    Order order = new Order();
    Random random = new Random();
    order.setTableId(random.nextInt(100));
    order.setOrderDate(date);
    order.setOrderId(orderId);
    orderId ++;
    orders.put(order, new ArrayList<>());
    serializator.writeOrdersInFile(orders);
    return order;

}

public void addMenuItemToOrder(Order order, MenuItem menuItem){
    assert menuItem != null;
    if(orders.get(order) != null){
        orders.get(order).add(menuItem);
        serializator.writeOrdersInFile(orders);
    }
    else{
        ArrayList<MenuItem> menuItems = new ArrayList<>();
        menuItems.add(menuItem);
        orders.put(order,menuItems);
        serializator.writeOrdersInFile(orders);
    }
```

```
}

public void placeOrder(Order order){
    this.setChanged();
    notifyObservers(order);

}


@Override
public double computePrice(Order order) {
    double orderPrice = 0;
    assert orderPrice == 0;
    for(MenuItem menuItem : orders.get(order)){
        orderPrice += menuItem.computePrice();
    }
    assert orderPrice>0;
    return orderPrice;
}


@Override
public synchronized void addObserver(Observer o) {
    super.addObserver(o);
}

@Override
public List<String> getMenuItemFieldNames() {
    List<String> menuItemFieldNames = new ArrayList<>();
    menuItemFieldNames.add("menuItemName");
    menuItemFieldNames.add("menuItemWeight");
    menuItemFieldNames.add("menuItemPrice");
    return menuItemFieldNames;
}

@Override
public ArrayList<MenuItem> getAvailableMenuItems() {
    return availableMenuItems;
}

@Override
public void generateBill(Order order) {
    FileWriter fileWriter = new FileWriter();
    fileWriter.generateBill(order,computePrice(order),orders.get(order));
}
public ArrayList<Order> getOrders(){
    return new ArrayList<>(orders.keySet());
}

public ArrayList<MenuItem> getMenuItemsForOrder(Order order){
    return orders.get(order);
}
```

In the data access later package:

4.4 File Writer

The name of the class is pretty self explanatory, as the class' job is to take some useful information and then print it into a file.

4.5 Restaurant Serializator

This class hold four different methods : for reading the menuItems, writing the menu items, reading the orders, writing the orders into two different files. But this specific action is done through serialization, and so it is written as a stream of bytes and read in the same manner.

In the presentation layer package:

Sabou Cezara
Computer Science, year 2
group 30421

There are a few sub-packages which I chose in order to get a more organize application:

4.5 administrator package

The administrator package holds a few classes:

- the main class AdministratorGUI
- BaseProductTable which creates an abstract table model for the base products of the composite product
- MenuItemTable – used in several parts in order to give the table model of the menu Item objects
- BaseProductWindow : is the window which pops up whenever the create base product or update base product buttons are pressed;
- CompositeProduct window : the other window which opens when the create composite product or update composite product is pressed.
- The Delete button is valid for both base and composite products.

4.6 waiter package

This package has the waiterGUI as the main class and Orders Table used for table model.

This gui is a bit interesting because it has 3 tables, one for all the available menu items, one for the selected menu items, and one for all the existing orders, which are kept into a list.

The window has a few buttons:

- create order, which creates a new order but with an empty list of menu items
- delete order, which removes the order from the table and the list
- addItemToList, which allows the waiter to select an item from the menu and add it to the current order
- removeItemFromList, which in case the order was mistaken, it allows the waiter to remove the order.
- placeOrder – this button notifies the chef the order is ready to be given further to the kitchen
- computeBill – will acutally call the FileWriter stated above and write the useful information about the order, such as the id, table number, all the ordered items and the final price.

4.7. chef package

The chef package has the chef class whose purpose is simply to be notified whenever an order is placed in the waiter class, and then display an info message which states that the chef was notified

4.8 resources package

It holds all the bill text files which were created in the waiter class

4.9 main package

The main package actually is meant to simply call the main gui and instantiate the chef and the serializator.

It is very simple and it’s purpose is only to start the application

Sabou Cezara
Computer Science, year 2
group 30421

5.Results

The results are being tested with assert. Every method from the following six are tested and checked also for input correctness. The test is based on assertEquals method, which checks if the polynomial is correct and also matches it to the expected result. The match is based on the equals method present in the polynomial class, that refers to the equals method present in the monomial class. The results are correct as the information is checked at the input and also at the output.

6.Conclusions

In conclusion, the application is created in OOP style and it serves as a restaurant processing system. A Graphical User Interface is created and easy to use, in order to offer the users an easy access to restaurant processing and to give back fast and correct results.

By developing this application I gained experience in trying business layer, data access layer and presentation layer pattern, understanding its functionalities and obtaining a clearer perspective of the project. I became more familiar to the ArrayList objects and also linking the methods designed by me to the front end.

I gained information about using serialization, hashing and also involving pre and post conditions in the project, to provide correctness.

I am aware that the application is not yet in its final form, and there are some adjustments that need to be done. The input conversion could be improved, as well as the interface. I also believe that there are more efficient ways in which I could implement the methods that process the interface input. I also would add other methods and options for the user such as: giving a random time for the chef to take in order to finish the order and then notify the waiter back about the information. Arranging the application in an easier and more aesthetic way.

7.References

http://www.tutorialspoint.com/java/java_serialization.html

http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html

http://stackoverflow.com/questions/11415160/how-to-enable-the-java-keyword-assert-in-eclipse-program-wise