

Documentation

Programming Techniques, Homework 4

Restaurant Management System

Content

| | |
|-------------------------------------|-------------------------------|
| 1. Homework's Objective..... | <u>pag 2</u> |
| 2. Problem analysis, modelling..... | <u>pag 2</u> |
| 3. Design..... | <u>pag 5</u> |
| 4. Implementation..... | <u>pag 7</u> |
| 5. Results..... | <u>pag 10</u> |
| 6. Conclusions..... | <u>pag 12</u> |
| 7. Bibliography..... | <u>pag 13</u> |

1. Homework's Objective

The aim of this homework and its principal objective is to design an application for processing menu items and orders for a restaurant.

The secondary objectives need to be accomplished in order to fulfill the main objective. The secondary objectives are:

| Secondary Objectives | Description |
|--|--|
| 1. Identifying the real life problem's possible scenarios[1] | This secondary objective is mandatory in the flow of developing the system in order to identify the system's requirements for designing them afterwise |
| 2. Decisions regarding structure of application[2] | Designing the needed classes and their specific functionality |
| 3. Graphical User Interface[3] | Creating a nice and simple Graphical User Interface such that the system is accessible, interactive and easy to be understood by any user |
| 4. Assembling the components together [4] | Linking the classes and the graphical user interface together in order to complete the system's functionality |
| 5. Testing the application[5] | The results of application should be analysed in order to determine if the system is having a correct functionality or not. |

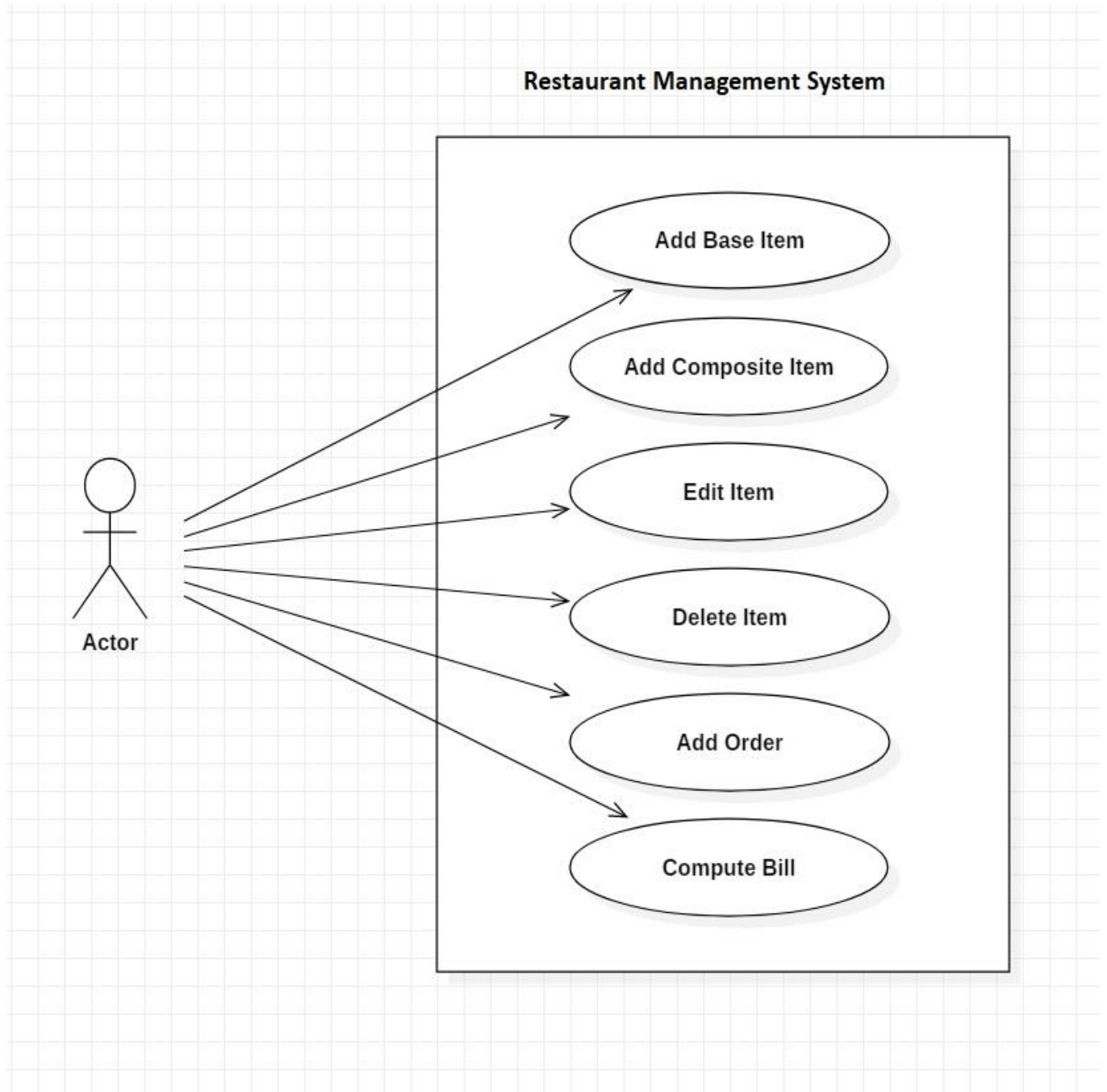
2. Problem analysis, modelling

The system manages the items from a menu of a restaurant and performs orders for a restaurant. BaseProducts and CompositeProducts can be added, removed or edited and orders can be generated. Orders can be generated using the existing products.

In order to present how the system is used, an *use-case diagram* that presents a set of scenarios that describe the “natural” behavior of the system is being created.

The scenarios are the use cases which in the current case are the operations that can be done on the products and orders.

Use-case diagram:



Use-cases description:

1. Add Base Product

- User inserts the base product properties: name and price
- System collects the data and checks to see if all the fields were inserted correctly
- System displays a message if the name or price were not inserted correctly and returns the control back to the user to insert again the base product
- System inserts the base product into the JTable if the inserted data was correct

2. Add Composite Product

- User inserts the composite product name and selects the products that he wants to contain the new composite product
- System collects the data and checks to see if all the fields were inserted correctly
- System displays a message if the name was not inserted correctly and returns the control back to the user to insert again the product
- System inserts the new composite product into the JTable if the inserted data was correct

3. Edit Product

- User insert the product properties: name and price
- System collects the data and checks to see if all the fields were inserted correctly
- System displays a message if the name or price were not inserted correctly and returns the control back to the user to insert again the base product
- System updates the product into the JTable if the inserted data was correct

4. Delete Item

- User inserts the name of the product that he wants to delete and select it in the table
- System collects the data and checks to see if the name was inserted correctly
- System displays a message if the name was not inserted correctly and returns the control back to the user to insert again the name of the product
- System deletes the product from the JTable if the inserted data was correct

5. Add Order

- User inserts the order properties: orderId, date and table and selects the items from the table of items that should take part into the order.
- System collects the data and checks to see if it was inserted correctly
- System displays a message if the data was not inserted correctly and returns the control back to the user to insert again the order related properties
- System inserts the order in the table of orders if the inserted data was correct

6. Compute Bill

- User selects the order from the table of orders
- System generates the order bill in txt format

3. Design

OOP designing of application:

The application was designed for managing the items from a menu of a restaurant and for performing orders for a restaurant. Products and Order features must be inserted from the user interface and serialized into file. They can be also edited or deleted after being inserted.

As data structures used, there was used an ArrayList of MenuItem for storing the menu items of the restaurant and HashMap for linking an order with its associated items. Also it was used an ArrayList of strings to store the name of the menu items.

As design patterns used, there was used Composite design pattern with the abstract class MenuItem, the leaf BaseProduct and the composite CompositeProduct.

```

classDiagram
    class Start {
        +main(args: String[]) void
    }
    class AdministratorGUI {
        -serialVersionUID: long = 1L (readOnly)
        +t0: JTextField
        +t1: JTextField
        +t2: JTextField
        ~models: DefaultTableModel
        ~table: JTable
        ~panel: JPanel
        ~panelOut: JPanel
        ~springLayout: SpringLayout
        ~nameLabel: JLabel
        ~quantityLabel: JLabel
        ~priceLabel: JLabel
        +addBaseButton: JButton
        +addCompositeButton: JButton
        +editButton: JButton
        +deleteButton: JButton
        +c: CompositeProduct
        ~items: MenuItem[] (collection="List")
        +constructor()+AdministratorGUI()
        +actionPerformed(e: ActionEvent): void
        +getItems(): MenuItem[]
        +makeFrame(frame_text: String): JFrame
        +makeLabels(label_text: String): JLabel
        +makeTextFields(): JTextField
        +makePrincipalPanels(panel_name: String): JPanel
        +makeButtons(button_text: String): JButton
        +getUserInput(t: JTextField): String
    }
    class Restaurant {
        -serialVersionUID: long (readOnly)
        ~map: HashMap
        +constructor()+Restaurant(items: MenuItem[])
        +constructor()+Restaurant()
        +createMenuItem(m: MenuItem): void
        +editMenuItem(m: MenuItem, names: String, items: MenuItem[]): void
        +deleteMenuItem(m: MenuItem): void
        +createOrder(o: Order, m: MenuItem[]): void
        +generateBill(o: Order, totalPrice: int, orderedProducts: String[]): void
        +getItems(): MenuItem[]
        +setItems(items: MenuItem[]): void
        +getMap(): HashMap
        +setMap(map: HashMap): void
    }
    class MenuItem {
        +computePrice(): int
    }
    class RestaurantProcessing {
    }
    class Order {
        +orderId: int
        +date: String
        +table: int
        +constructor()+Order(orderId: int, date: String, table: int)
        +hashCode(): int
        +equals(obj: Object): boolean
        +getOrderId(): int
        +getOrderDate(): String
        +getDate(): String
        +getOrderTable(): int
        +setTable(table: int): void
    }
    class CompositeProduct {
        -name: String
        -price: int
        +constructor()+CompositeProduct(name: String, products: MenuItem[])
        +constructor()+CompositeProduct()
        +addProduct(m: MenuItem): void
        +computePrice(): int
        +getName(): String
        +setName(name: String): void
        +getProducts(): MenuItem[]
        +getOrderProducts(): MenuItem[]
        +setProducts(products: MenuItem[]): void
        +getPrice(): int
        +setPrice(price: int): void
    }
    class BaseProduct {
        -price: int
        -name: String
        +constructor()+BaseProduct(name: String, price: int)
        +computePrice(): int
        +getPrice(): int
        +setPrice(price: int): void
        +getName(): String
        +setName(name: String): void
    }
    class WaiterGUI {
        -serialVersionUID: long = 1L (readOnly)
        +t0: JTextField
        +t1: JTextField
        +t2: JTextField
        ~models: DefaultTableModel
        ~table: JTable
        ~panel: JPanel
        ~panelOut: JPanel
        ~springLayout: SpringLayout
        ~idLabel: JLabel
        ~dateLabel: JLabel
        ~tableLabel: JLabel
        +addButton: JButton
        +billButton: JButton
        ~items: MenuItem[] (collection="List")
        ~map: HashMap
        +constructor()+WaiterGUI()
        +actionPerformed(e: ActionEvent): void
        +makeFrame(frame_text: String): JFrame
        +makeLabels(label_text: String): JLabel
        +makeTextFields(): JTextField
        +makePrincipalPanels(panel_name: String): JPanel
        +makeButtons(button_text: String): JButton
        +getUserInput(t: JTextField): String
    }
    Start --> AdministratorGUI
    Start --> WaiterGUI
    AdministratorGUI --> Restaurant
    AdministratorGUI --> MenuItem
    Restaurant --|> CompositeProduct
    Restaurant --|> BaseProduct
    MenuItem --|> BaseProduct
    RestaurantProcessing --> Order
    RestaurantProcessing --> CompositeProduct
    RestaurantProcessing --> BaseProduct
    WaiterGUI --> Order
    WaiterGUI --> CompositeProduct
    WaiterGUI --> BaseProduct
    
```

- There is an inheritance relationship between BaseProduct and MenuItem and between CompositeProduct and MenuItem because they extend the abstract class MenuItem
- There is an aggregation relationship between Restaurant and MenuItem because the restaurant has many menu items. Also, there is another aggregation relationship between Restaurant and Order because one restaurant can have many orders. There is an aggregation relationship between CompositeProduct and MenuItem because the composite products can contain several menu items
- The class Restaurant implements the methods from the RestaurantProcessing interface and the user interface classes use them
- There are dependency relationships between RestaurantSerializator and MenuItem, between AdministratorGUI and MenuItem, between WaiterGui and Order, between Start and AdministratorGUI and between Start and WaiterGUI because the

dependent class uses an object of type of the class which it depends on. The dependent class is the one written in the left of the “and” word above.

4. Implementation

The application is structured in 5 packages and 11 classes.

The package presentation contains the classes that defines the user interface. It also contains the action listeners for the buttons that user uses to interact with the application.

Class AdministratorGUI constructs the window related to administrator features and operations. It uses a grid layout for the window to separate the JTable[7] from the buttons and textfields. It contains JButtons for adding, editing and removing a product, JTextFields[8] for entering the product’s name and price. In the right hand side, there is a JTable containing all the products inserted along with all their features. There are also methods for creating the UI components.

The class AdministratorGUI implements also the action listeners on the buttons. When user presses the add base product button a new base product with the specified characteristics is being added to the list of MenuItems of the restaurant. When the user presses the add composite button, the list of items is traced, get the class of each item through reflexion and get the method getName through reflexion from that class, and if the name is the same with the name selected by the user in the JTable, then the new composite product is being created and the table is being updated. Also, it is stored to a file using serialization. If the user presses edit item button, then a base product is created using the specified characteristics and calls the editMenuItem method and then updates the table and serializes the new edited info to the file. If the user presses the delete item button, the list of items is traced and for each item it is obtained its class through reflexion and the method to get that item’s name. The name is compared to the one selected in the table and if it is equal, then it is deleted.

Class WaiterGUI constructs the window related to waiter operations. It uses a grid layout for the window to separate the JTable[7] from the buttons and textfields. It contains JButtons for adding an order and for generating a bill, JTextFields[8] for entering the order’s id, date and table. In the right hand side, there is a JTable

containing all the orders inserted along with all their features. There are also methods for creating the UI components.

When the user wants to add a new order, the items from the admin table are taken and the list of menu items is traced and for each item it is taken its class and the method to get its name and price through reflexion. The price of order is incremented with the price of each selected item and the selected items are added into the order list of menu items. Then the order and the list of ordered items are associated using a hash map that stores the orders and the table is being updated with the new created order. If the user presses the compute bill button, a new order with the selected order from the table is made and the keys of the hash table are being traced and if the order selected from the table is equal with the one from the map, then the list of values is traced and for each item is obtained the class and the method to get its name through reflexion, and the name is added to a list of strings containing the names of the order products. Then the method to compute the order is called and the order related info is saved into the table.

The package data contains the RestaurantSerializator class which contains methods for serializing and deserializing items from/to a file. The method serialization serializes the list of menu items received as parameter into the file restaurant.ser. The method deserialization deserializes from the file restaurant.ser into a list of menu items.

The package business contains 5 classes and an interface.

The class BaseProduct describes the attributes of a base product and compute the price for a base product with the method computePrice().

The class CompositeProduct describes the attributes of a composite product and compute the price of a composite product using the method computePrice().

```
public int computePrice() {  
    int computedPrice=0;  
    for(int i=0; i<products.size(); i++)  
    {  
        computedPrice+=products.get(i).computePrice();  
    }  
    this.price=computedPrice;  
    return computedPrice;  
}
```


The class MenuItem is an abstract class containing the abstract method computePrice();

Class Order contains the attributes of an order. Also, it overrides the methods equals and hashCode that will be used in the restaurant class.

```
@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + ((date == null) ? 0 : date.hashCode());
    result = prime * result + orderId;
    result = prime * result + table;
    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Order other = (Order) obj;
    if (date == null) {
        if (other.date != null)
            return false;
    } else if (!date.equals(other.date))
        return false;
    if (orderId != other.orderId)
        return false;
    if (table != other.table)
        return false;
    return true;
}
```

Class Restaurant implements RestaurantProcessing interface and describes the methods for administrator and waiter operations. The method createMenuItem() adds a new menu item to the list of items. The method editMenuItem edits an existing item from the list of items. The method deleteMenuItem deletes an existing item from the list of items. The method createOrder creates a new order in the hash table of orders and associates it with its menu items. The method generateBill generates the bill of an exiting order as txt.

```
public void createOrder(Order o, ArrayList<MenuItem> m)
{
    map.put(o, m);
}

/**
 * Method that generates the bill in .txt format for an order
 * @param o the order on which to generate bill
 * @param totalPrice the price of that order
 * @param orderedProducts the list containing the ordered products for that order
 */
public void generateBill(Order o, int totalPrice, ArrayList<String> orderedProducts)
{
    try {
        PrintWriter out = new PrintWriter("Orders.txt");
        out.println("Order Bill ");
        out.println("Order id: " + o.getId());
        out.println("Order made at date: " + o.getDate());
        out.println("Order made by table: " + o.getTable());
        out.println("Ordered products: " + orderedProducts);
        out.println("The total price of the order is: " + totalPrice);
        out.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}
```

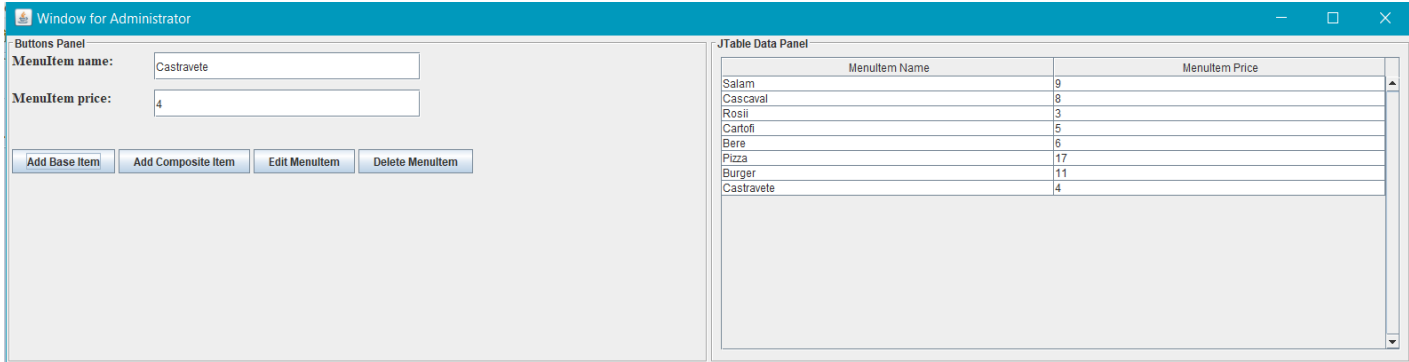
Package `business.validators` contains two classes that perform the validators. Class `NameValidator` uses the method `validate` that checks to see if the inserted name of the product is a valid name. The name is checked to contain only letters. Class `PriceValidator` uses the method `validate` that checks to see if the inserted price of the product is valid. The price is checked to contain only digits.

Package `start` contains the `Start` class which is the program driver class. It has only one method, the `main()` method that calls the constructors of classes `AdministratorGUI` and `WaiterGUI` in order to set up the windows and start the application.

5. Results

In order to see the results, operations were made. As we can see, the existing items are loaded using deserialization from file when the app starts. We can observe below the addition of a new base product:

Author: Georgescu Vlad
Group: 30424



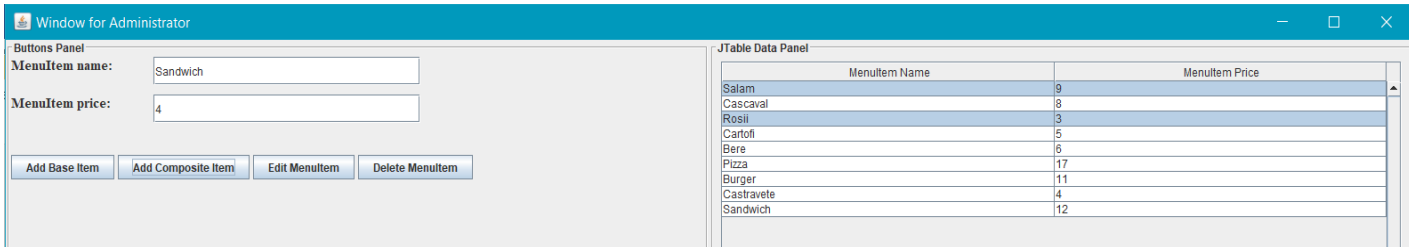
The 'Buttons Panel' contains the following fields and buttons:

- MenuItem name:
- MenuItem price:
- Buttons: Add Base Item, Add Composite Item, Edit MenuItem, Delete MenuItem

The 'JTable Data Panel' displays the following data:

| MenuItem Name | MenuItem Price |
|---------------|----------------|
| Salam | 9 |
| Cascaval | 8 |
| Rosii | 3 |
| Cartofi | 5 |
| Bere | 6 |
| Pizza | 17 |
| Burger | 11 |
| Castravete | 4 |

Let's add a new composite product, called sandwich, composed from "salam" and "rosii":



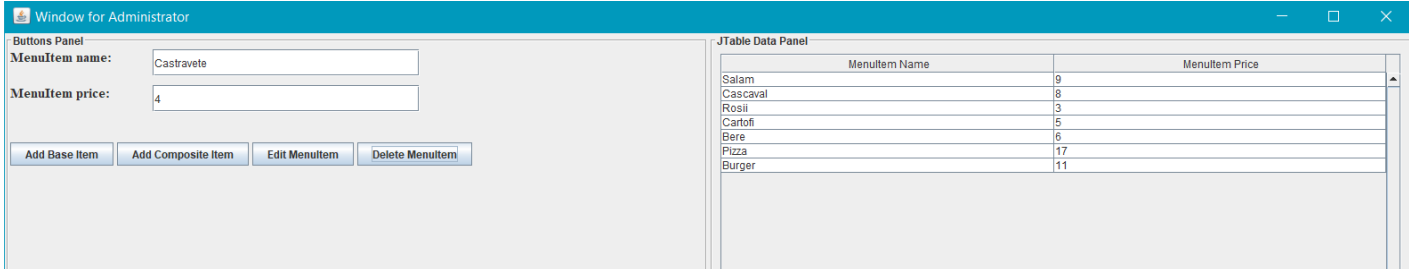
The 'Buttons Panel' contains the following fields and buttons:

- MenuItem name:
- MenuItem price:
- Buttons: Add Base Item, Add Composite Item, Edit MenuItem, Delete MenuItem

The 'JTable Data Panel' displays the following data:

| MenuItem Name | MenuItem Price |
|---------------|----------------|
| Salam | 9 |
| Cascaval | 8 |
| Rosii | 3 |
| Cartofi | 5 |
| Bere | 6 |
| Pizza | 17 |
| Burger | 11 |
| Castravete | 4 |
| Sandwich | 12 |

Let's delete now the previous two added products:



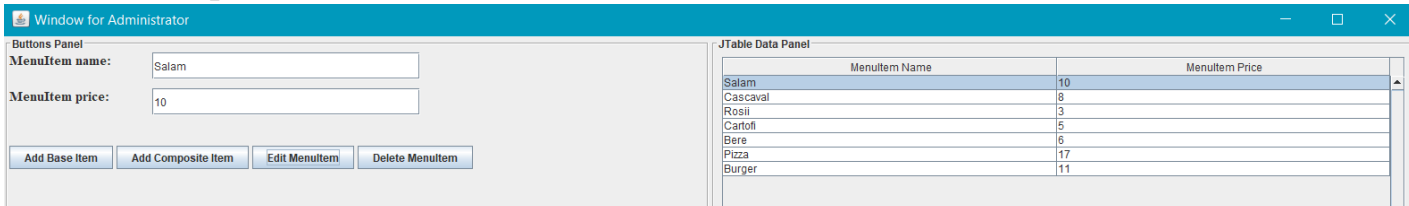
The 'Buttons Panel' contains the following fields and buttons:

- MenuItem name:
- MenuItem price:
- Buttons: Add Base Item, Add Composite Item, Edit MenuItem, Delete MenuItem

The 'JTable Data Panel' displays the following data:

| MenuItem Name | MenuItem Price |
|---------------|----------------|
| Salam | 9 |
| Cascaval | 8 |
| Rosii | 3 |
| Cartofi | 5 |
| Bere | 6 |
| Pizza | 17 |
| Burger | 11 |

Let's edit the price of "Salam" from 9 to 10:



The 'Buttons Panel' contains the following fields and buttons:

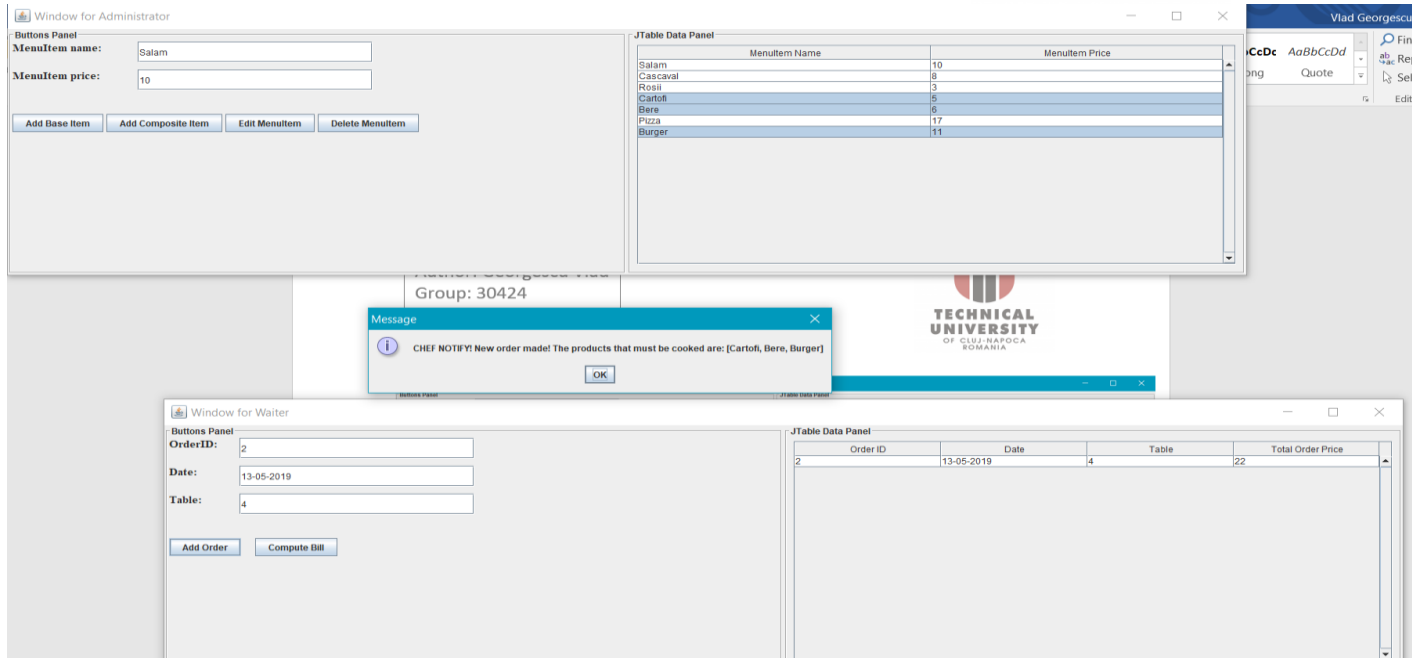
- MenuItem name:
- MenuItem price:
- Buttons: Add Base Item, Add Composite Item, Edit MenuItem, Delete MenuItem

The 'JTable Data Panel' displays the following data:

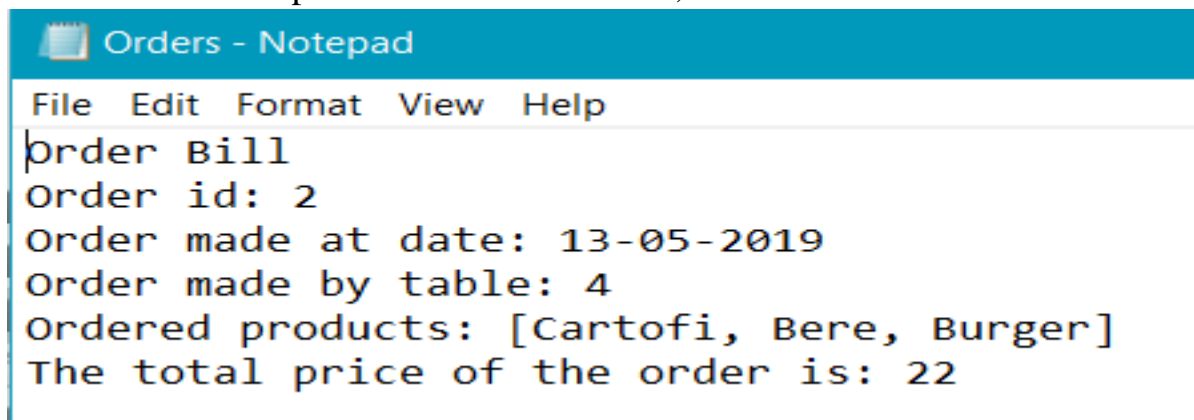
| MenuItem Name | MenuItem Price |
|---------------|----------------|
| Salam | 10 |
| Cascaval | 8 |
| Rosii | 3 |
| Cartofi | 5 |
| Bere | 6 |
| Pizza | 17 |
| Burger | 11 |

Let's now create an Order with id 2 with today's date to table 4, containing "cartofi, burger, bere". Also, the chef will be notified:

Author: Georgescu Vlad
Group: 30424



And now let's compute the bill for this order, which will be saved into a txt file:



6. Conclusions

To conclude, the system for restaurant management was very useful in accomplish new object oriented programming skills. I learnt about Composite design pattern, abstract classes, serialization and deserialization and hashing in Java. Moreover, reflexion technique was aprofunded. Moreover, this documentation was helpful because I learnt how to structure and assembly all my ideas regarding the projection of the system all together.

As future developments, I would consider implementing a new window for the chef and to notify the chef about an order that was made using the Observer Design Pattern. Also, using design by contract for the methods in the class Restaurant and implementing more validators should be taken into account.

7. Bibliography

- For creating the graphical user interface and link it to buttons:

<https://docs.oracle.com/javase/8/docs/api/javax/swing/JFrame.html> frame
<https://docs.oracle.com/javase/8/docs/api/javax/swing/JButton.html> buttons
<https://www.javatpoint.com/java-jpanel> panels
<https://stackoverflow.com/questions/10177183/java-add-scroll-into-text-area> scroll panel
<https://docs.oracle.com/javase/8/docs/api/javax/swing/JTable.html> JTable
<https://docs.oracle.com/javase/tutorial/uiswing/components/textfield.html> textfields
<https://docs.oracle.com/javase/tutorial/uiswing/layout/spring.html> layout
<https://docs.oracle.com/javase/tutorial/uiswing/examples/components/TextDemoProject/src/components/TextDemo.java> general gui example with all basic elements

- For UML Class Diagram

<https://stackoverflow.com/questions/885937/what-is-the-difference-between-association-aggregation-and-composition/34069760#34069760>
<https://stackoverflow.com/questions/1230889/difference-between-association-and-dependency>
<https://stackoverflow.com/questions/21967841/aggregation-vs-composition-vs-association-vs-direct-association>

- For application design

<https://www.geeksforgeeks.org/reflection-in-java/> Reflexion technique
http://coned.utcluj.ro/~salomie/PT_Lic/
<https://www.geeksforgeeks.org/composite-design-pattern/> Composite Design Pattern
https://www.w3schools.com/java/java_hashmap.asp Java HashMap

Author: Georgescu Vlad
Group: 30424



http://www.tutorialspoint.com/java/java_serialization.htm Serialization/Deserialization