

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

Laboratory Work – Assignment 2
Order Management Application

Name: Popa Florin

Group: 30425

Table of Contents

1. Introduction	3
1.1 Problem specification.....	3
1.2 Personal approach	3
2. Design	4
2.1 UML diagrams	4
2.2 Classes Design	7
3. Packages and Interfaces	16
4. User Interface	18
5. Conclusions	21
6. Bibliography.....	21

1. Introduction

1.1 Problem specification

The problem is defined as follows: “Consider an application OrderManagement for processing customer orders. The application uses (minimally) the following classes: Order, OPDept (Order Processing Department), Customer, Product, and Warehouse. The classes OPDept and Warehouse use a BinarySearchTree for storing orders.

- a. Analyze the application domain, determine the structure and behavior of its classes, identify use cases.
- b. Generate use case diagrams, an extended UML class diagram, two sequence diagrams and an activity diagram.
- c. Implement and test the application classes. Use javadoc for documenting the classes.
- d. Design, write and test a Java program for order management using the classes designed at question c). The program should include a set of utility operations such as under-stock, over-stock, totals, filters, etc.”

1.2 Personal approach

The program will have a graphic interface, through which the user can log in the application. The users credentials will be stored in an external file which will be accessed and verified upon pressing the “login button”. Depending on the provided credentials, the user can access two different types of interfaces: one for administration and one for regular use. Many users may be added manually in the credentials file, and only one administrator with a particular password. If an invalid user or password are written, upon pressing the “login button” an error message will be displayed on the page.

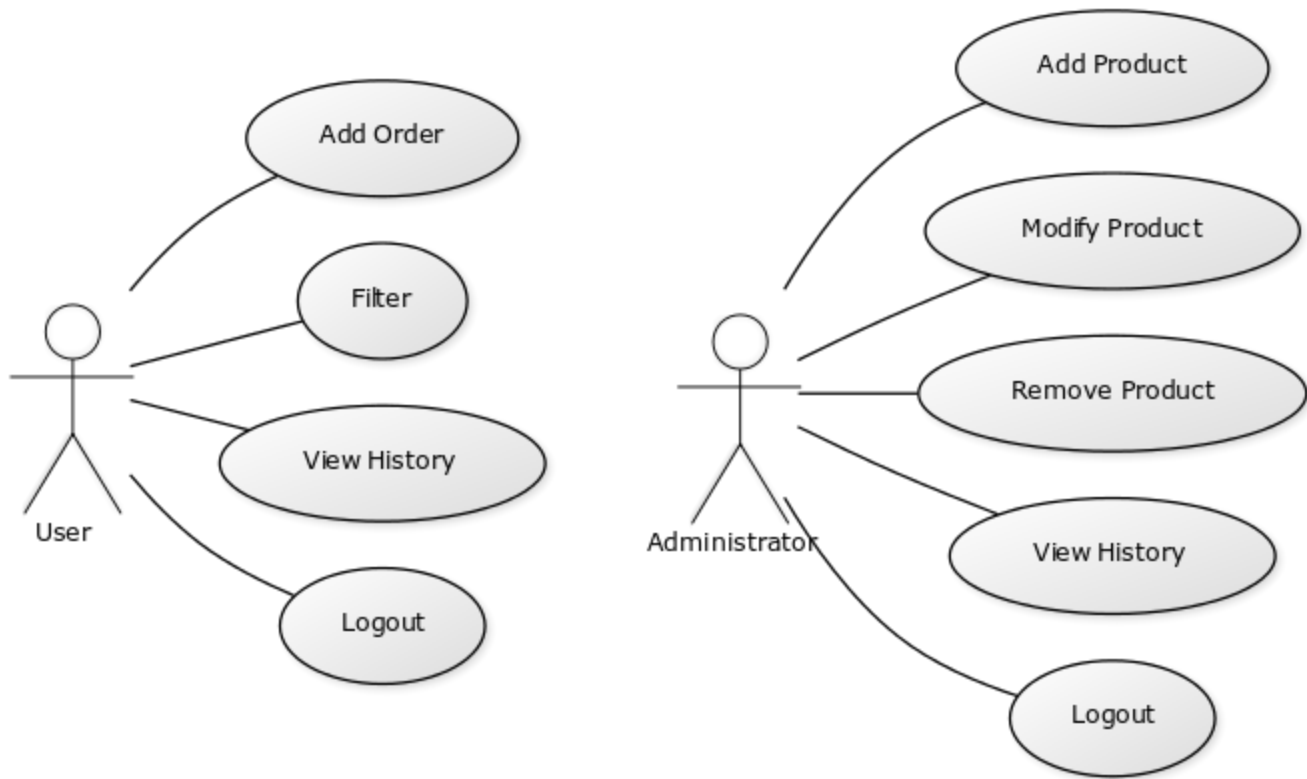
The administration interface has functions like “add product” for product addition, being provided the product’s name, price and quantity, “modify product” for quantity modification of a product, provided the new desired quantity, and “remove product” for product removal from the data structure. The manipulation of these products is done in real-time and it is showed in a table. Also, the interface provides functions like “order history” which displays all the orders made by regular users, also in a table, and “logout” for sending the user back to the login page.

The user interface displays the same table from the administrator interface, but of course, it does not have the same manipulation rights. The user is only able to see the products added by the administrator, filter them, regarding the price, and place orders. This interface provides functions like “order” used to place an order for the selected product. This function sends the user to another page where the user’s name and the desired quantity will be required for the order to be placed. If everything is correct, a validation message will be displayed and the order is placed. An order history can be displayed using the “order history” function. As well as in the administration interface, a “logout” function is present, used to send the user to the login page.

2. Design

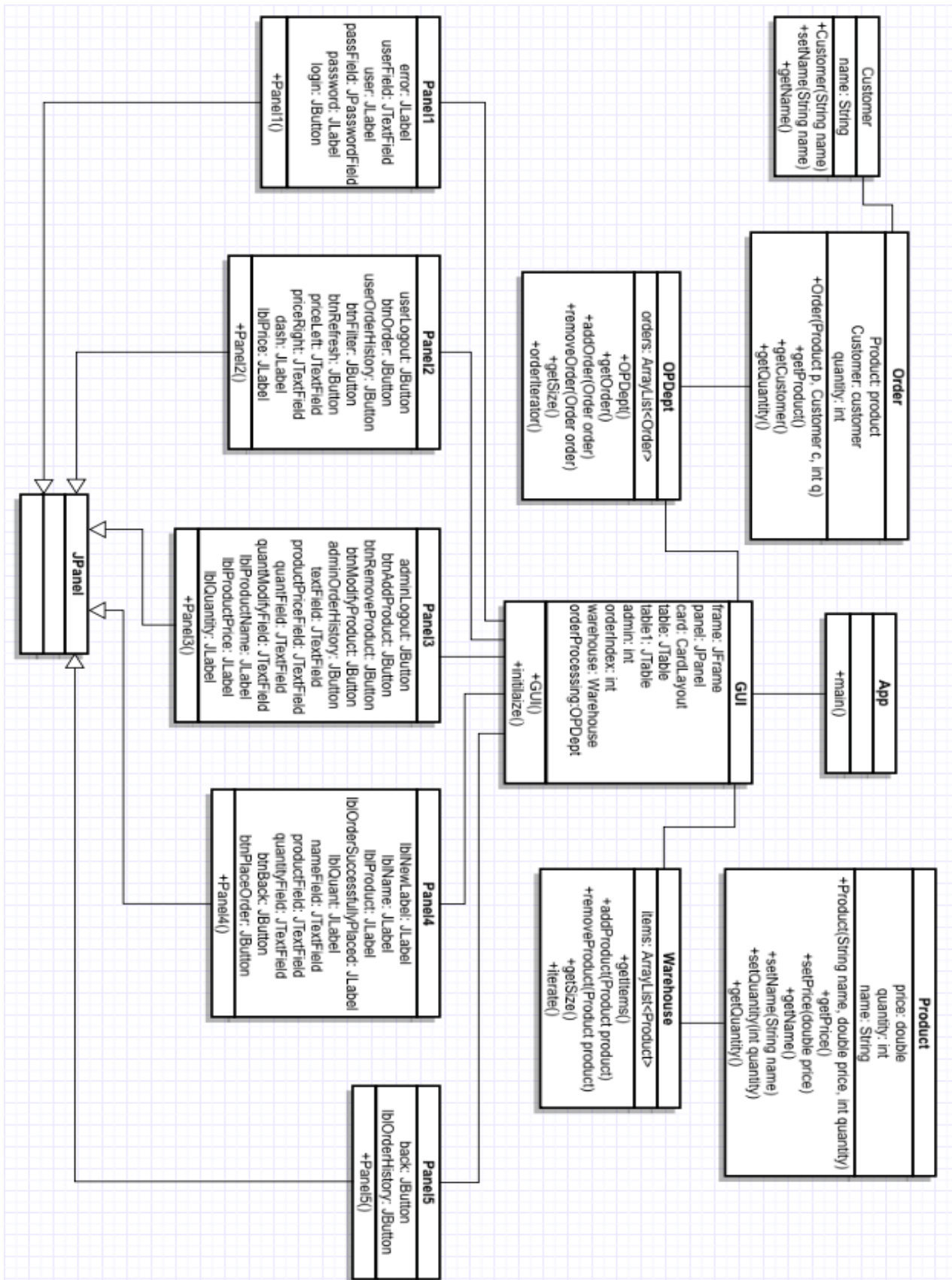
2.1 UML diagrams

a) Use Case Diagram



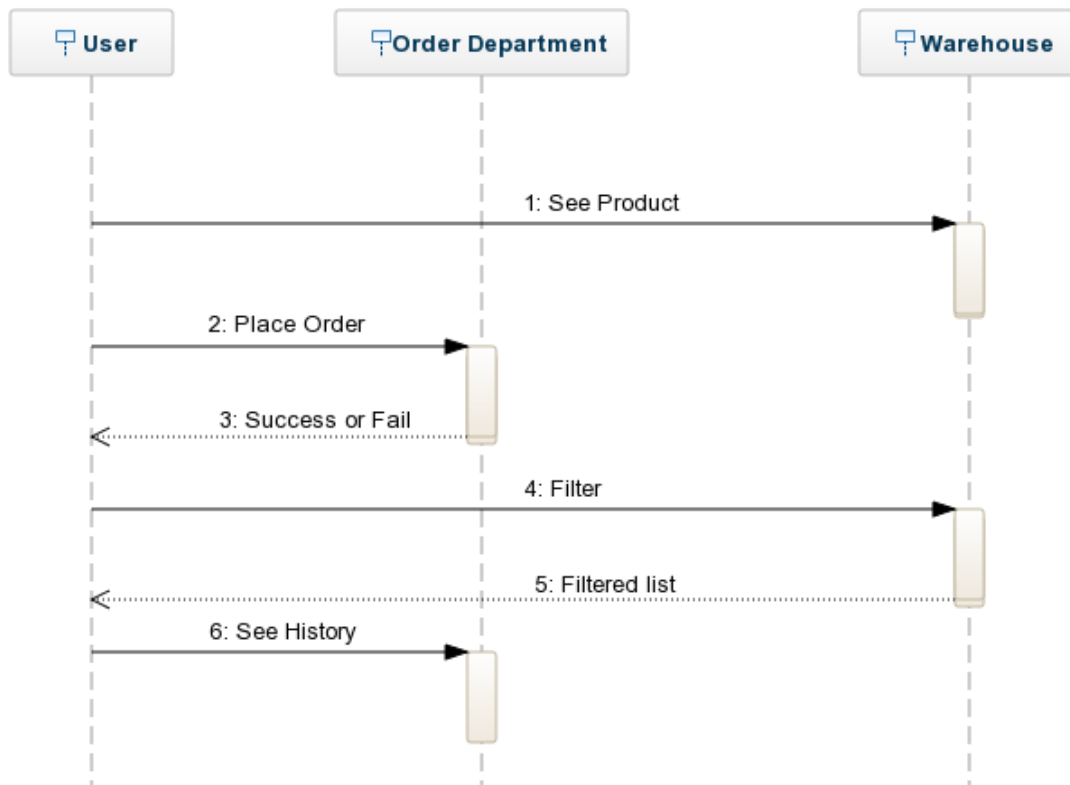
The use case diagram presents the user that interacts with the application, depending on the rights.

b) Class Diagram

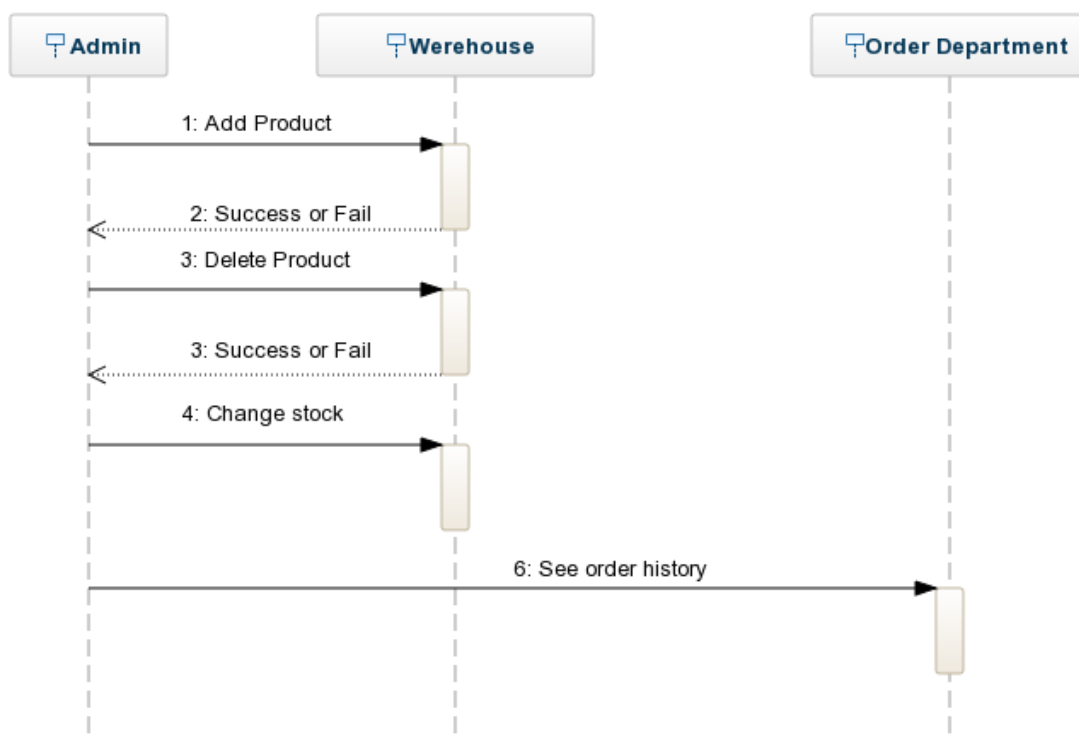


c) Sequence diagrams

Sequence diagram for the user case:



Sequence diagram for the admin case:



2.2 Classes Design

a) GUI Class

This class is designed to create a graphical user interface so the application would be easier to use. It creates a frame, where all the graphic elements are added, after which is called directly in the main function.

The GUI's constructor calls the *initialize()* method, which is the engine of the class.

The methods of the GUI class:

- *initialize()*

In this method, the frame is created, and all the interactive elements are added. The frame's layout is set to *null*. A main panel is added to the frame, that will have set a layout of type CardLayout. This type of layout easily jumps from one panel to another, helping the programmer create applications such as this. The main panel works with five different panel objects, defined in five different classes. Each panel has its own functionalities and behaves in a certain way.

In addition, the action listeners are implemented in this method for each individual button, so in conclusion, all the thinking is made here.

The attributes of the GUI class:

```
Warehouse warehouse; //Warehouse object
OPDept orderProcessing; //OPDept object
JFrame frame; //The application's frame
JPanel panel; //The main panel
CardLayout card; //CardLayout object

JTable table; //Table where the stock is displayed
JTable table1; //Table where the order history is displayed

private int admin; //A simple marker variable
private int orderIndex; //A temporary index marker
```

The interaction:

Each button assigned to the panels has a specific function upon which the application relies. Even if the action listeners for the buttons are defined in the GUI class, their functionalities will be explained in the next sections.

b) Product Class

This class is created in order to store information about a certain product. Several objects of this type will be defined in the Warehouse Class, where they will be stored in a data structure. Each individual object will have the following attributes: price, quantity and a name. This attributes are set in the constructor when the object is created, or modified later using methods defined for this purpose.

The attributes of the Product class:

```
private double price;
private int quantity;
private String name;
```

The methods of the Product class:

- ***public Product(String name, double price, int quantity)***

This is the class's constructor, where all the attributes are set. Upon creating a new object of type Product, each individual attribute will have to be given as the constructor's arguments.

- ***public double getPrice()***

This method will return the price attribute of type double.

- ***public void setPrice(double price)***

This method is used mainly to modify the value of the price attribute, given in the argument the desired value of type double.

- ***public String getName()***

This method will return the name attribute of type String

- ***public void setName(String name)***

This method is used mainly to modify the value of the name attribute, given in the argument the desired value of type String

- ***public void setQuantity(int quantity)***

This method is used mainly to modify the value of the quantity attribute, given in the argument the desired value of type int.

- ***public void getQuantity()***

This method will return the quantity attribute of type int.

c) Warehouse Class

This class is used to store objects of type Product in a data structure of type Array List. A new object of this data type is created in the class's constructor. The data stored here is manipulated using several methods defined in this class.

The attributes of the Warehouse class:

```
private ArrayList<Product> items;
```

The methods of the Warehouse class:

- ***public Warehouse()***

This is the class's constructor, where the object items of type `ArrayList<Product>` is created. When an object of type Warehouse will be created, this data structured is automatically created and the data within will be manipulated using the methods bellow.

- ***public void addProduct(Product p)***

This method is used to store in the Array List a new product, given the argument the desired "p" object of type Product.

- ***public void removeProduct(Product p)***

This method is used to remove from the Array List an existing product, give the argument the desired "p" object of type Product.

- ***public int getSize()***

This method will return the size of the Array List as value of type int.

- ***public Iterator<Product> iterate()***

This method will return an object of type Iterator that will be used to iterate through the Array List. It very useful for identifying desired objects within the data structure and then manipulate them using the methods above.

d) Customer Class

This class is created in order to store information about a certain customer. An object of this type will be defined in the GUI Class, as a result of a trigger instantiated by the "Place Order" button. After this object is created, it is given as an argument to the Order class.

The attributes of the Customer class:

```
private String name;
```

The methods of the Customer class:

- ***public Customer(String name)***

This is the class's constructor, where the name attribute is set. Upon creating a new object of type Customer, the attribute will have to be given as the constructor's arguments.

- ***public void getName()***

This method will return the name attribute of type double.

- ***public void setName(String name)***

This method is used mainly to modify the value of the name attribute, given in the argument the desired value of type double.

e) Order Class

This class is created in order to store information about a certain order. Several objects of this type will be defined in the OPDept Class, where they will be stored in a data structure. Each individual object will have the following attributes: product, customer and a quantity. These attributes are set in the constructor when the object is created, or modified later using methods defined for this purpose.

The attributes of the Order class:

```
Product product;
Customer customer;
int quantity;
```

The methods of the Order class:

- ***public Order(Product p, Customer c, int quantity)***

This is the class's constructor, where all the attributes are set. Upon creating a new object of type Order, each individual attribute will have to be given as the constructor's arguments.

- ***public void getProduct()***

This method will return the product attribute of type Product.

- ***public String getCustomer()***

This method will return the customer attribute of type Customer.

- ***public void getQuantity()***

This method will return the quantity attribute of type int.

f) OPDept Class

Being an abbreviation for order processing department, this class is used to store objects of type Order in a data structure of type Array List. A new object of this data type is created in the class's constructor. The data stored here is manipulated using several methods defined in this class.

The attributes of the OPDept class:

```
private ArrayList<Order> orders;
```

The methods of the OPDept class:

- ***public OPDept()***

This is the class's constructor, where the object orders of type ArrayList<Order> is created. When an object of type Warehouse will be defined, this data structure is automatically created and the data within will be manipulated using the methods bellow.

- ***public void addOrder(Order order)***

This method is used to store in the Array List a new order, given in the argument the desired "order" object of type Order.

- ***public void removeOrder(Order order)***

This method is used to remove from the Array List an existing order, given in the argument the desired "order" object of type Order.

- ***public int getSize()***

This method will return the size of the Array List as value of type int.

- ***public Iterator<Product> iterate()***

This method will return an object of type Iterator that will be used to iterate through the Array List. It very useful for identifying desired objects within the data structure and then manipulate them using the methods above.

g) Panel1 Class

This class, like the other four, of the same type, inherits the functionalities of the superclass JPanel. This particular class is used to display the first page when the application is ran. It embodies elements like text field, labels and buttons, used by the user to log in the application using particular credentials.

The attributes of the Panel1 class:

```
public JLabel error;
public JTextField userField;
public JLabel user;
public JPasswordField passField;
public JLabel password;
public JButton login;
```

The methods of the Panel1 class:

- ***public Panel1()***

This is the class's constructor, used to create all the objects for each attribute and given that the class is of type JPanel, the add() method is called to add each individual element to the panel.

The object error of type JLabel is used to display an error message in case the login process fails.

The objects userField and passField are used by the user to input data.

The button object login triggers the login process defined in the GUI method. This process starts by opening a .properties file in which user or admin credentials are stored. Depending the input given in the fields, if the credential given are valid, the user is taken to another panel.

h) Panel2 Class

This class, like the other four, of the same type, inherits the functionalities of the superclass JPanel. This particular class is used to display the “user” page if a successful login is made. It embodies elements like labels, text fields and buttons, used by the user to place orders, view order history, filter the products and logout of the application.

The attributes of the Panel2 class:

```
public JButton userLogout;
public JButton btnOrder;
public JButton userOrderHistory;
public JButton btnFilter;
public JButton btnRefresh;

public JTextField priceLeft;
public JTextField priceRight;

public JLabel dash;
public JLabel lblPrice;
```

The methods of the Panel2 class:

- ***public Panel2()***

This is the class’s constructor, used to create all the objects for each attribute and given that the class is of type JPanel, the add() method is called to add each individual element to the panel.

The button userLogout triggers the logout process defined in the GUI class, used to send the user back to the login page.

The button btnOrder is triggers the order process defined in the GUI class, used to send the user to the “place order page” i.e. panel 4, where the user can place the order.

The button userOrderHistory triggers the process defined in the GUI class, used to send the user to the “order history page” i.e. panel 5, where the user can view the order history.

The button btnFilter triggers the filter process defined in the GUI class, used to filter the products displayed by the arguments given in the priceLeft text field and priceRight text field.

The button `btnRefresh` triggers the refresh process defined in the GUI class, used to repopulate the product list displayed to the user, from the data structure where the products are stored. This process is used after a filtering process.

i) Panel3 Class

This class, like the other four, of the same type, inherits the functionalities of the superclass `JPanel`. This particular class is used to display the “administrator” page if a successful login is made. It embodies elements like labels, text fields and buttons, used by the administrator to add, modify and remove products, view order history, and logout of the application.

The attributes of the `Panel3` class:

```
public JButton adminLogout;
public JButton btnAddProduct;
public JButton btnRemoveProduct;
public JButton btnModifyProduct;
public JButton adminOrderHistory;

public JTextField textField;
public JTextField productPriceField;
public JTextField quantField;
public JTextField quantModifyField;

public JLabel lblProductName;
public JLabel lblProductPrice;
public JLabel lblQuantity;
```

The methods of the `Panel3` class:

- ***public Panel3()***

This is the class’s constructor, used to create all the objects for each attribute and given that the class is of type `JPanel`, the `add()` method is called to add each individual element to the panel.

The button `adminLogout` triggers the logout process defined in the GUI class, used to send the user back to the login page.

The button `btnAddProduct` triggers the add product process defined in the GUI class, used to create a new object of type `Product`, store it in the data structure defined in the `Warehouse` class and display it in the table.

The button `btnRemoveProduct` triggers the remove product process defined in the GUI class, used to remove an object of type `Product`, stored in the data structure defined in the `Warehouse` class. This action removes the product from the table as well.

The button `btnModifyProduct` triggers the modify product process defined in the GUI class, used to modify the quantity attribute of an object of type `Product`, stored in the data structure defined in the `Warehouse`. The new quantity that is to be assigned, is given by the administrator in the `quantModifyField` text field.

The button `adminOrderHistory` triggers the process defined in the GUI class, used to send the user to the “order history page” i.e. panel 5, where the user can view the order history.

j) Panel4 Class

This class, like the other four, of the same type, inherits the functionalities of the superclass `JPanel`. This particular class is used by the user to place orders. It embodies elements like labels, text fields and buttons, used by the user to input the required order credentials, place the order and return to the previous page.

The attributes of the `Panel4` class:

```
public JLabel lblNewLabel;
public JLabel lblName;
public JLabel lblProduct;
public JLabel lblOrderSuccessfullyPlaced;
public JLabel lblQuant;

public JTextField nameField;
public JTextField productField;
public JTextField quantityField;

public JButton btnBack;
public JButton btnPlaceOrder;
```

The methods of the `Panel4` class:

- ***public Panel4()***

This is the class’s constructor, used to create all the objects for each attribute and given that the class is of type `JPanel`, the `add()` method is called to add each individual element to the panel.

The object `lblOrderSuccessfullyPlaced` of type `JLabel` is used to display a message in case the order process succeeds.

In the `nameField` text field, the user has to input the name on which the order is placed. This name will be given as an argument for the `Customer Class` constructor.

In the `productField` text field, the name of the selected product is already set. Hence that it is not modifiable.

In the `quantityField` text field, the user has to input the desired quantity which will be given as an argument to the `Order Class` constructor.

The button `btnBack` triggers the process defined in the GUI class, used to send the user to the previous page

The button `btnPlaceOrder` triggers the process defined in the GUI class, used to create an object of type `Customer` and an object of type `Order`. The `Customer` object, alongside the `Product` selected and the quantity, are given as arguments to the `Order Class` constructor. After, the `Order` object is stored in the data structure from the `OPDept` class.

k) Panel5 Class

This class, like the other four, of the same type, inherits the functionalities of the superclass JPanel. This particular class is used by the user or the administrator to view the order history. It embodies elements like labels and buttons, used by the user to return to the previous page.

The attributes of the Panel5 class:

```
public JButton back;  
public JLabel lblOrderHistory;
```

The methods of the Panel5 class:

- ***public Panel5()***

This is the class's constructor, used to create all the objects for each attribute and given that the class is of type JPanel, the add() method is called to add each individual element to the panel.

The button object back triggers the process defined in the GUI class, used to send the user to the previous page, whether it was the user page or the administrator page.

3. Packages and Interfaces

A Java package is a mechanism for organizing Java classes into namespaces. Java packages can be stored in compressed files called JAR files, allowing classes to download faster as a group rather than one at a time. Programmers also typically use packages to organize classes belonging to the same category or providing similar functionality. A package provides a unique namespace for the types it contains. Classes in the same package can access each other's package-access members.

A package allows a developer to group classes (and interfaces) together. These classes will all be related in some way – they might all have to do with a specific application or perform a specific set of tasks.

For this application the following packages are imported, each of them having a certain role for the proper working of the application. We import them in the GUI Class (most of them relate to the user interface properties):

- 1) `import java.awt`: Contains all of the classes for creating user interfaces and for painting graphics and images. A user interface object such as a button or a scrollbar is called, in AWT terminology, a component. The `Component` class is the root of all AWT components.
 - a. `java.awt.Font`: The `Font` class represents fonts, which are used to render text in a visible way. A font provides the information needed to map sequences of characters to sequences of glyphs and to render sequences of glyphs on `Graphics` and `Component` objects.
- 2) `import java.awt.event` : Used in interacting with input devices such as the mouse and keyboard
 - a) `java.awt.event.ActionEvent`: A semantic event which indicates that a component-defined action occurred. This high-level event is generated by a component (such as a `Button`) when the component-specific action occurs (such as being pressed). The event is passed to every `ActionListener` object that registered to receive such events using the component's `addActionListener` method.
 - b) `java.awt.event.ActionListener`: The listener interface for receiving action events. The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's `addActionListener` method. When the action event occurs, that object's `actionPerformed` method is invoked.
- 3) `import javax.swing`: Swing is a GUI widget toolkit for Java. It is part of Oracle's Java Foundation Classes (JFC) – an API for providing a graphical user interface (GUI) for Java programs.

Swing was developed to provide a more sophisticated set of GUI components than the earlier Abstract Window Toolkit (AWT). Swing provides a native look and feel that emulates the look and feel of several platforms, and also supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform. It has more powerful and flexible components than AWT. In addition to familiar components such as buttons, check boxes and labels, Swing provides several advanced components such as tabbed panel, scroll panes, trees, tables, and lists.

- a) `javax.swing.JButton`: Buttons can be configured, and to some degree controlled, by Actions. Using an Action with a button has many benefits beyond directly configuring a button.
- b) `javax.swing.JFrame`: An extended version of `java.awt.Frame` that adds support for the JFC/Swing component architecture.
- c) `javax.swing.JLabel`: A display area for a short text string or an image, or both. A label does not react to input events. As a result, it cannot get the keyboard focus. A label can, however, display a keyboard alternative as a convenience for a nearby component that has a keyboard alternative but can't display it.

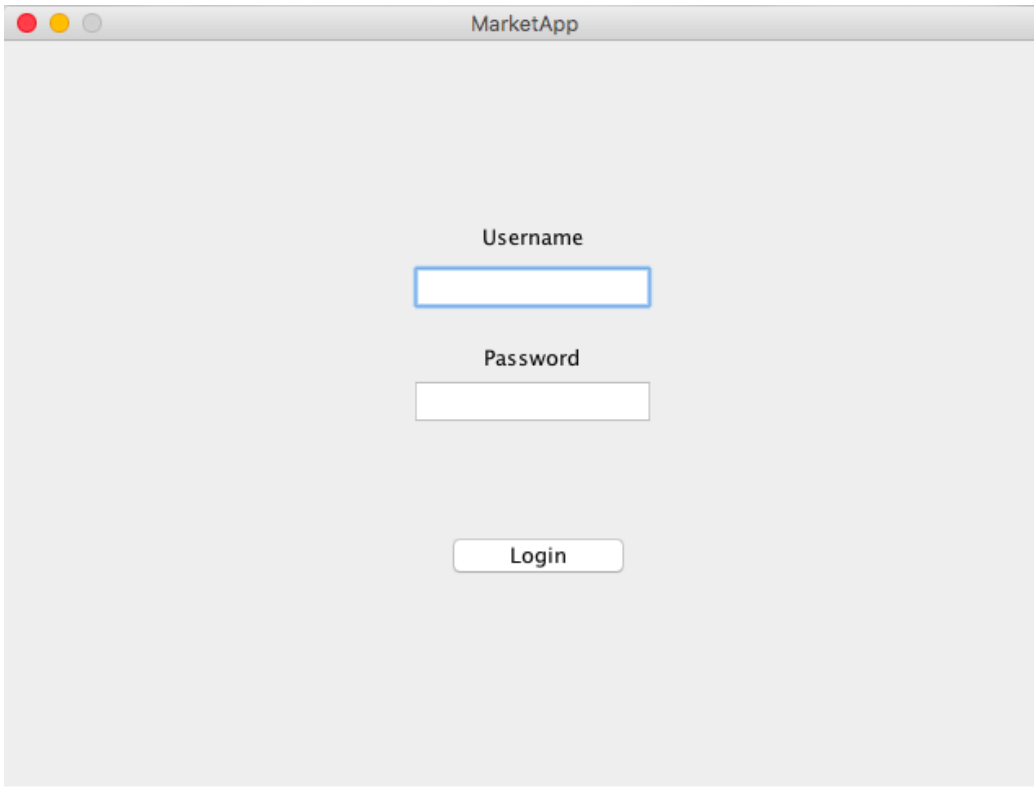
A `JLabel` object can display either text, an image, or both. You can specify where in the label's display area the label's contents are aligned by setting the vertical and horizontal alignment. By default, labels are vertically centered in their display area. Text-only labels are leading edge aligned, by default; image-only labels are horizontally centered, by default.

- d) `javax.swing.JPanel`: The `JPanel` class provides general-purpose containers for lightweight components. By default, panels do not add colors to anything except their own background; however, you can easily add borders to them and otherwise customize their painting.

In many types of look and feel, panels are opaque by default. Opaque panels work well as content panes and can help with painting efficiently, as described in Using Top-Level Containers. You can change a panel's transparency by invoking the `setOpaque` method. A transparent panel draws no background, so that any components underneath show through.

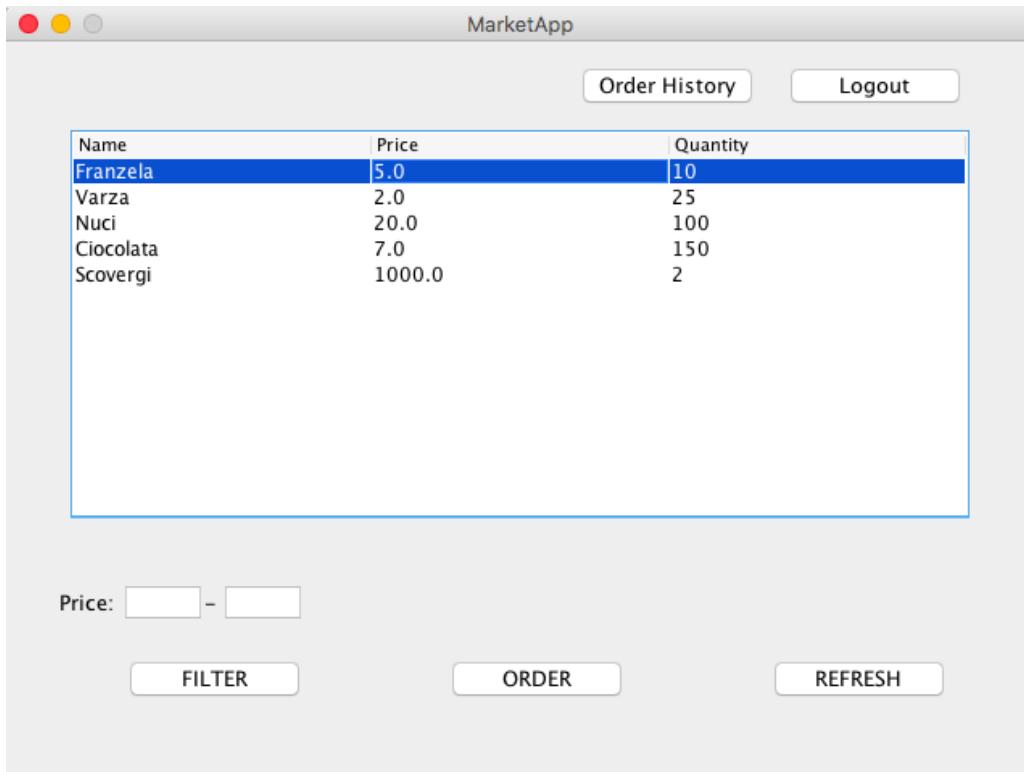
- e) `javax.swing.JTable`: The `JTable` is used to display and edit regular two-dimensional tables of cells. The `JTable` has many facilities that make it possible to customize its rendering and editing but provides defaults for these features so that simple tables can be set up easily.

4. User Interface



A screenshot of a web application window titled "MarketApp". The window has a light gray background. In the center, there is a login form. It consists of two text input fields: the top one is labeled "Username" and the bottom one is labeled "Password". Below these fields is a button labeled "Login".

Login page (Panel 1)



A screenshot of a web application window titled "MarketApp". The window has a light gray background. At the top right, there are two buttons: "Order History" and "Logout". Below these buttons is a table with three columns: "Name", "Price", and "Quantity". The table contains five rows of data. The first row, "Franzela", is highlighted in blue. Below the table, there is a "Price:" label followed by two input fields separated by a minus sign. At the bottom, there are three buttons: "FILTER", "ORDER", and "REFRESH".

Name	Price	Quantity
Franzela	5.0	10
Varza	2.0	25
Nuci	20.0	100
Ciocolata	7.0	150
Scovergi	1000.0	2

User page (Panel 2)

Order History Logout

Name	Price	Quantity
Franzela	5.0	10
Varza	2.0	25
Nuci	20.0	100
Ciocolata	7.0	150
Scovergi	1000.0	2

Product Name:

Product Price:

Quantity:

ADD PRODUCT MODIFY PRODUCT REMOVE PRODUCT

Administrator page (Panel 3)

BACK

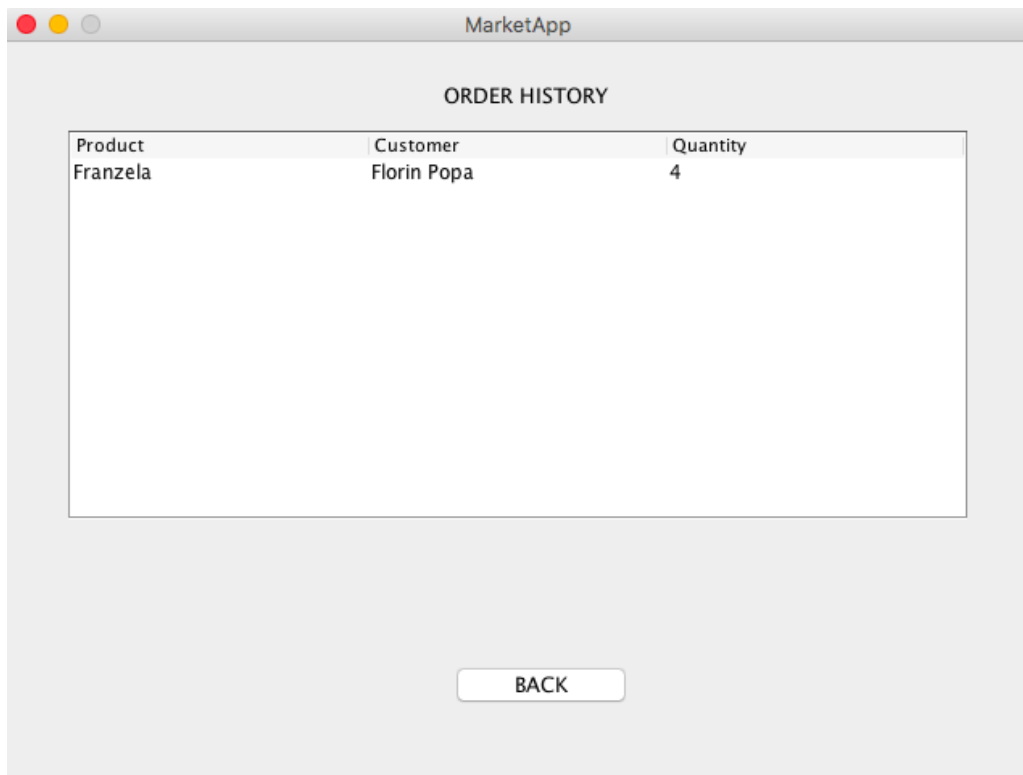
Provide the information in the fields bellow:

Name: Product:

Quantity:

PLACE ORDER

Order placement (Panel 4)



Order history (Panel 5)

Application Window

When running the application, the “MarketApp” window will open and it will provide to the user the possibility of giving inputs and choosing the operations to be executed. This window is constructed in the GUI class using some predefined classes and instructions.

The user interface is based on the properties of the above mentioned packages. All the objects we need are declared as attributes of the GUI class and they are initialized in the constructor of this class.

The listener interface is for receiving action events. The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's `addActionListener` method. When the action event occurs, that object's `actionPerformed` method is invoked. In this case the only events that occur are when the user clicks on one of the operation buttons from the graphical interface.

5. Conclusions

Developing this application was good challenge. It was a very good practice as a laboratory assignment. Of course, many features could be implemented in future versions, such as providing the user with the possibility of creating a new account, changing the data structure with an SQL database, add graphic images to improve esthetics, etc.

6. Bibliography

- <https://docs.oracle.com/>
- <http://stackoverflow.com>
- <https://wiki.eclipse.org>