UNIVERSITATEA TEHNICĂ CLUJ-NAPOCA

Programming Techniques

Laboratory Work - Homework 2

Name: Bologa Marius Vasile

Group: 30425

1.Project objectives

Task: 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.

To achieve this objective , we use object - oriented programming principles , applying programming language Java using Eclipse IDE. The application works in a basic way , such that we have 2 users. One the admin which manages the stock of the products and it is able to see all the orders of the clients, and the other one is te user, which can see the stock of products, and can order any product given in the stock.

2. Problem analysis , modeling scenarios , use cases

2.1 Problem analysis

The analysis of a problem starts from examining the real model or the model we confront with in the real world and passing the problem through a laborious process of abstractization. Hence we identify our problem domain and we try to decompose it in modules easy to implement. Starting from the concept of e-commerce, I have started to underline the main, objects presented in the process of buying a product like, the customer, the product, order, stock etc. In order to be able to design and build a software program that performs and satisfies all of the specifications and requirements presented above it is very important to understand all the opperations needed in order to hava a sustainable application.

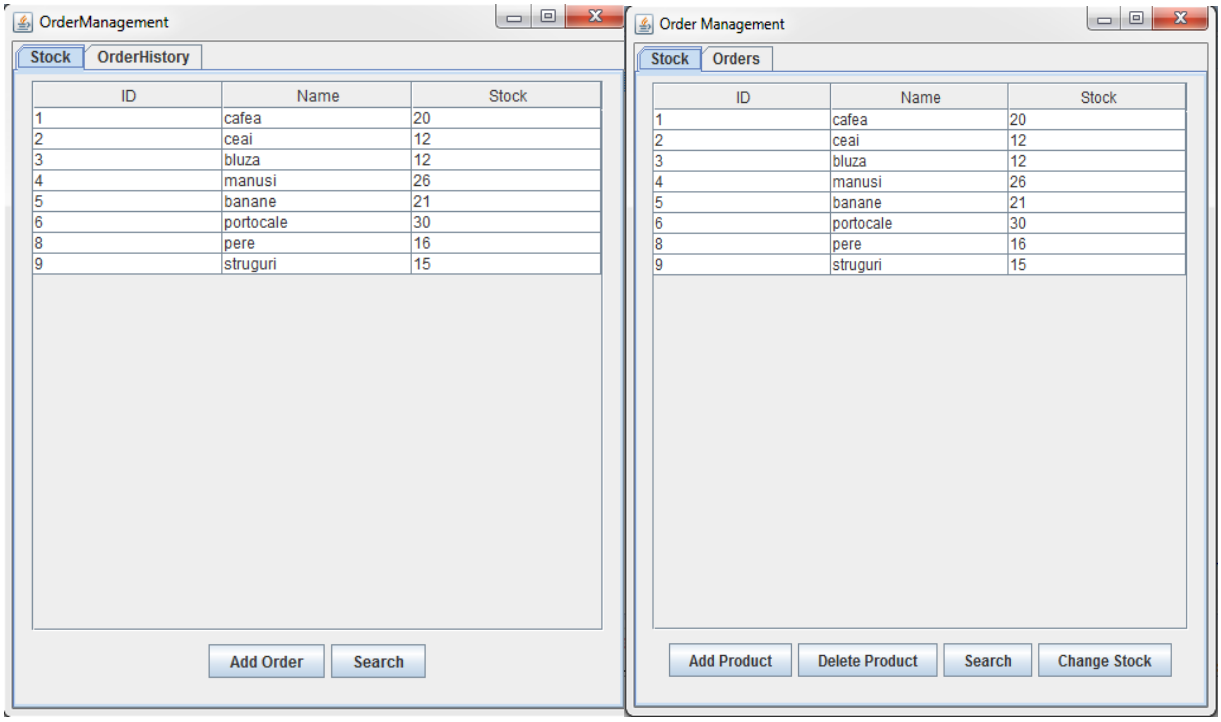
2.2 Modeling

First of all, we need to find some means to store the history orders and the stock. The specification says thatwe should have a minimum number of classes, and I have started from that point to built my application . In order to save the stock, and the order history, I have used the interface Serializable, in order to have my object written as a sequence of bytes when the program ends, and to restore the stock and the orders at their previous status, when the app was

closed. In order to do that, all the classes that contributed in a way or another to the built of the other classes that implements the interface Serializable, must implement by their own the interface. If we want that a field from a class not to be serialized we must declare it as „transient". Why should I use BST in order to keep the elements from the stock and order department sorted? Well in order to find them in a quick way, we use the TreeSet collection, adn sort the products and the orders by id. So when we will need to search the object with a name or an id, then we know that each product has it's own unique id, so we will know exactly it's position in the tree.

2.3 Scenario and Use cases

The use cases are strongly related to the user. How is it better to design the interface in order to be as user-friendly as possible, both for the user and the admin ? Hence, I have decided upon the following models :



(User frame) (Admin frame)

The admin hits the button add product, and a window for adding a product is opened. There he introduces, the name of the product, the id, and the quantity to be on the stock, and then he clicks „OK". If he wants to delete a product, he must select the row where product is situated and the click delete product. In order to search for a product he must hit the button search, and introduce the name of the product he wants to find. Also if he wants to change the quantity of a product, then he hits the button change stock and introduces the value by which the number of a specific object is changed.

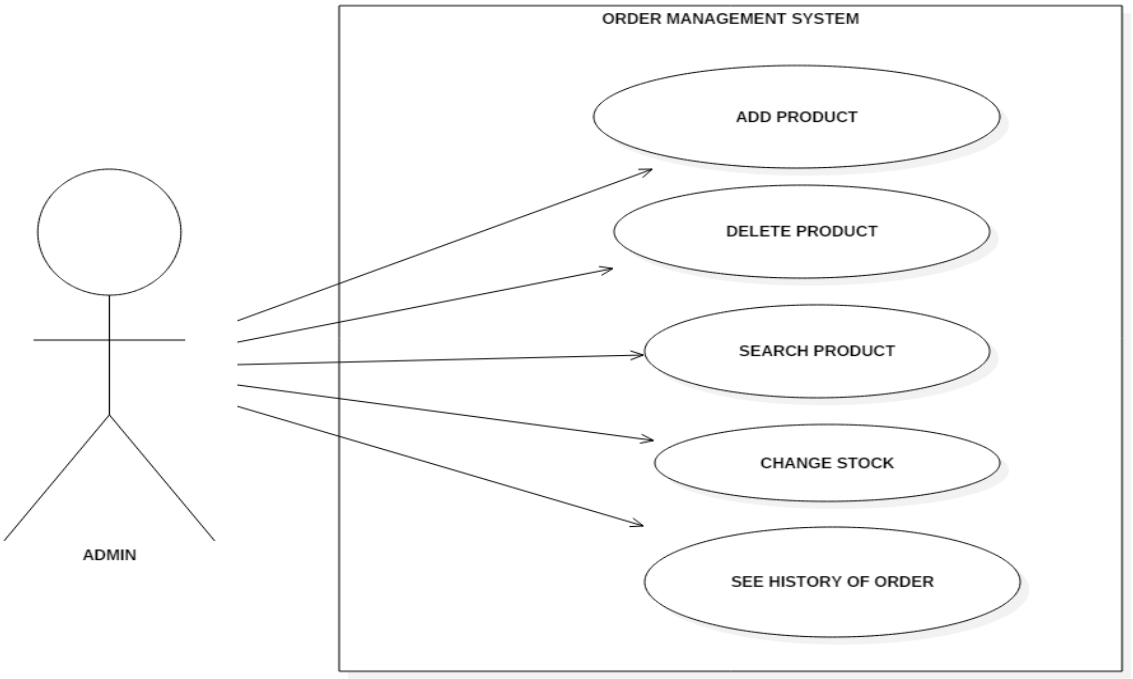
As an user, he can search for a product he just introduces there the name, and there will pop-up a window that will inform the user on what id is the product, and the quantity. Also he can add an order, by simply put the right data in the add order window.

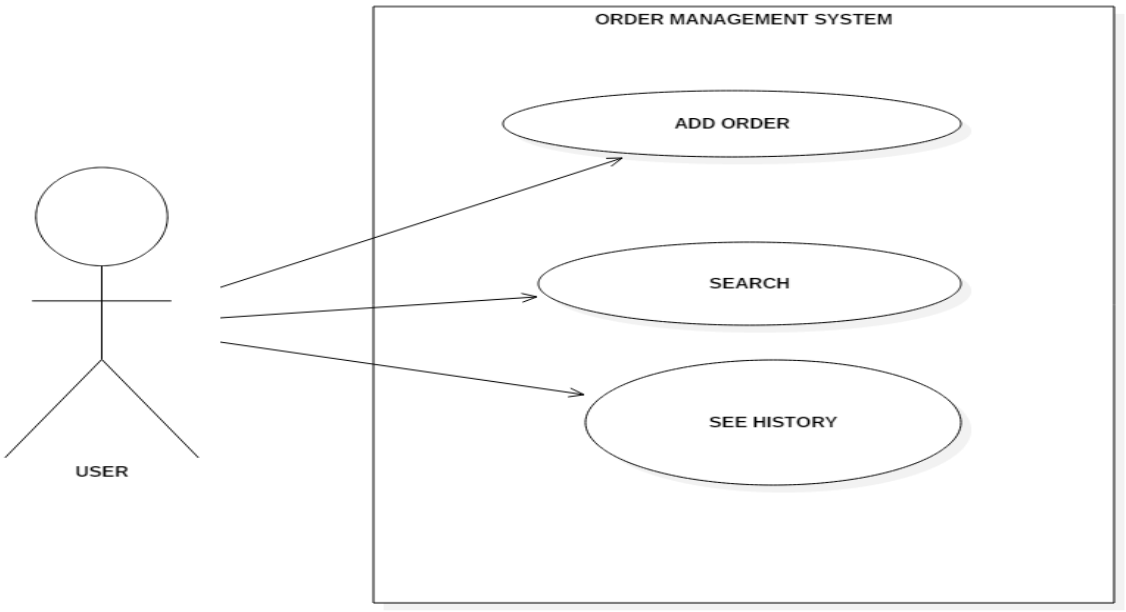
3. Projection and Implementation

In what the implementation is concerned this project was developed in Eclipse and it was only tested in this environment. However the program should maintain its portability. Concerning the code implementation I did not make use of laborious algorithms, but I have rather stayed faithful to the classical implementation of an order management system.

3.1 UML diagrams

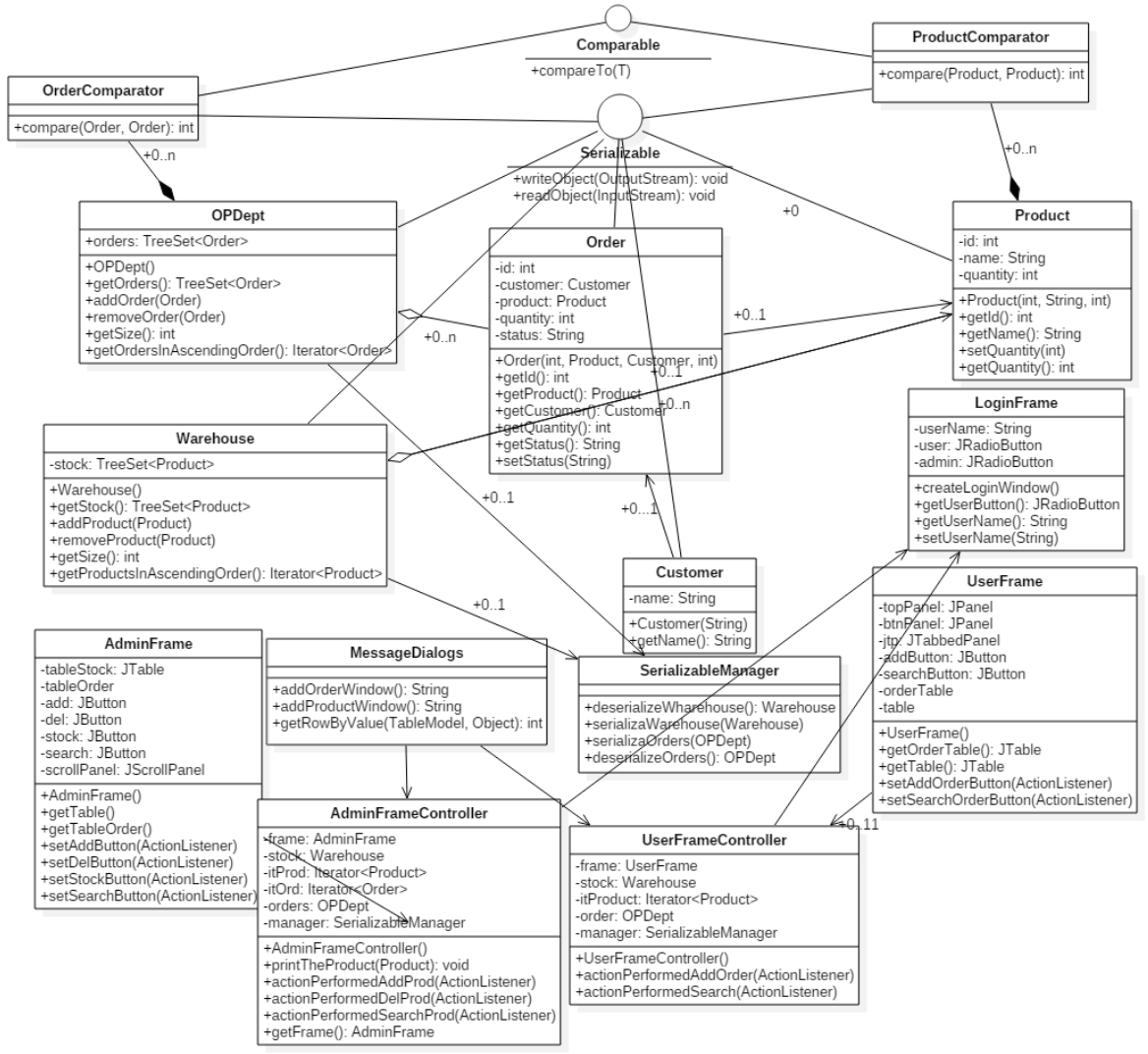
3.1.1 Use case diagram





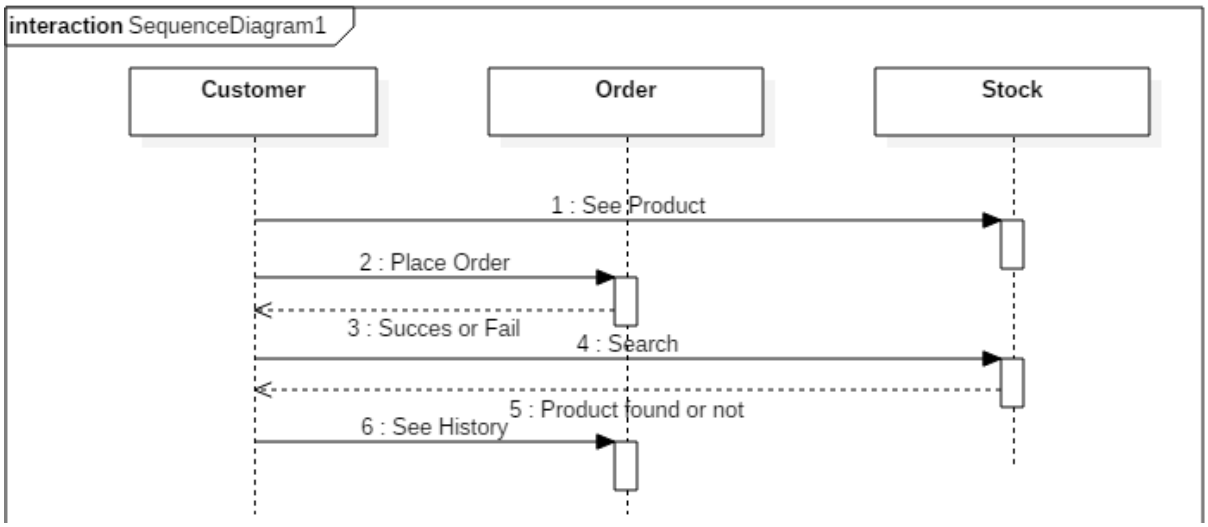
The use case diagram is nothing but the system functionalities written in an organized manner, presentig the actor, which is the admin in this case and the operations provided by the system for the user.

3.1.2 Class diagram

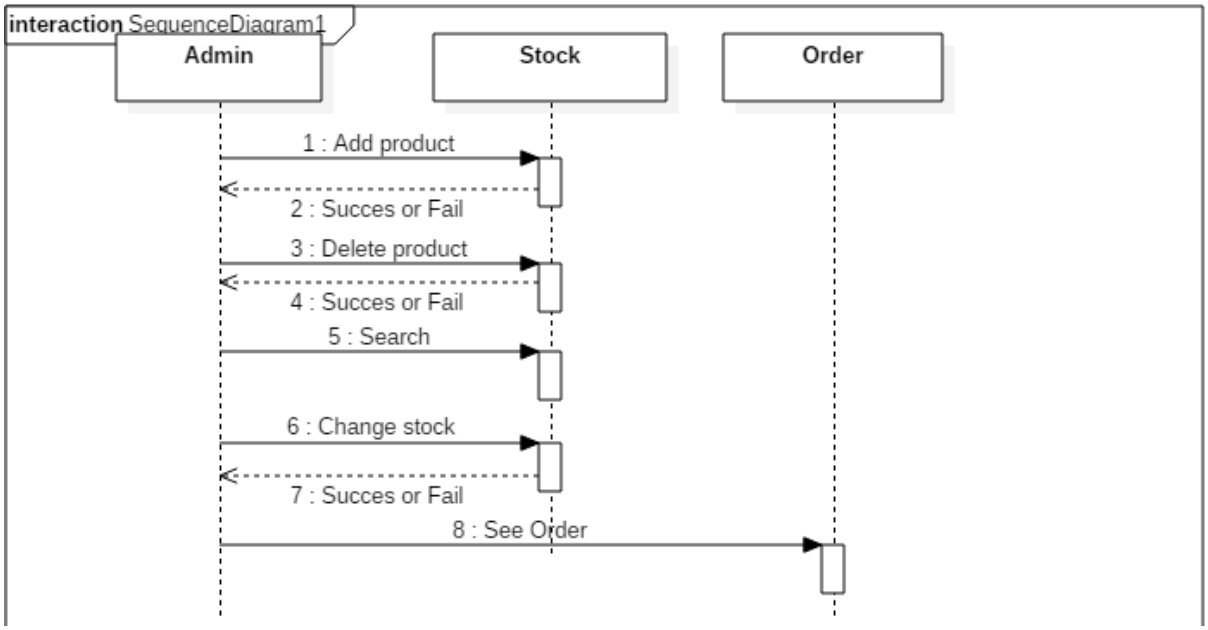


I have choosen besides the mandatory classes, 9 classes, that helped me in implementing the task. One of the main relationship presented here is the dependiency. A nother useful relation shiop represented in the class diagram is the agregation warehouse - product and the aggregation order department- order. So by this the product, and the order will exists even if the warehouse or opdept will be destroyed. Also we have composition between product comparator and product, and order comparator and order. Using this relation ship the comparators will be destroyed as soon as those 2 classes are destroyed.

3.1.3 Sequence diagram

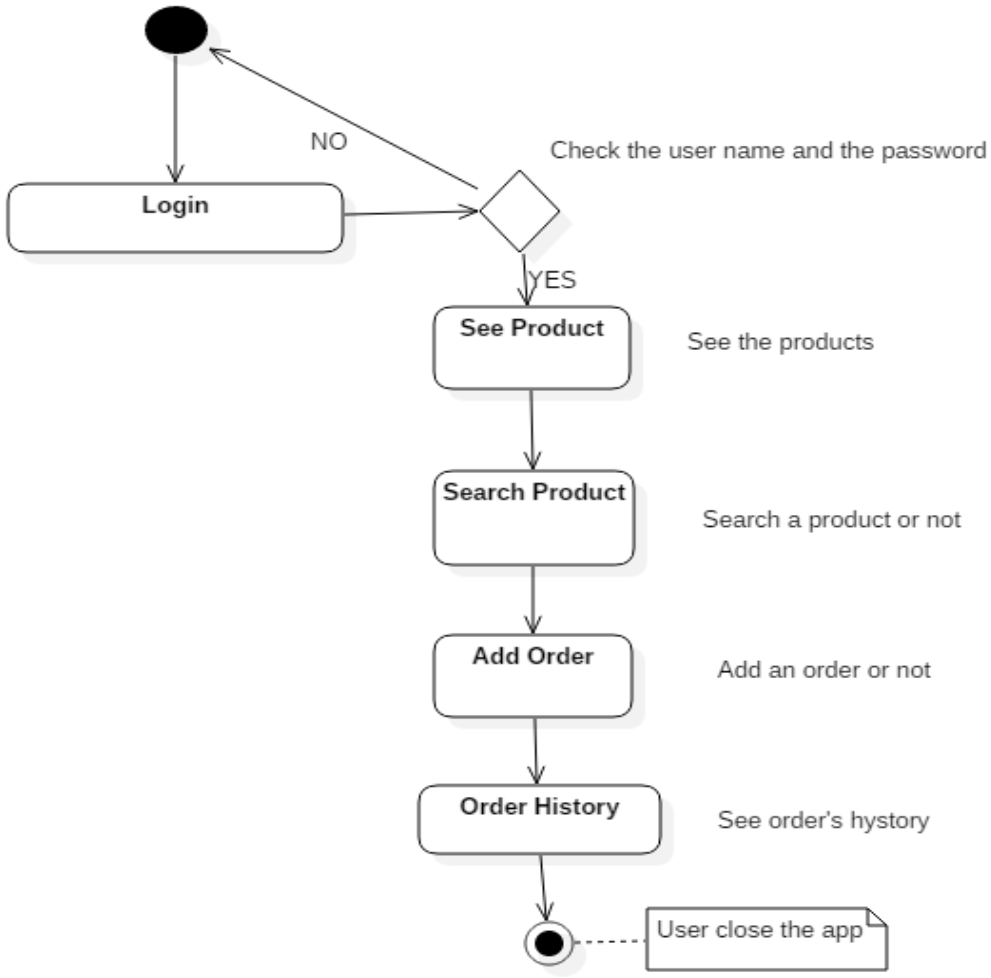


This is the sequence diagram for the user use case.



This is the sequence diagram for the admin use case.

3.1.4 Activity diagram



3.2 Data structures

The data structures used at this problem are either primitive data types such as integers or more complex data structures such as a TreeSet or new created objects such as Product,Order, Customer,Warehouse or OPDept. The TreeSet was introduced, to have a binary search tree representation, to keep the orders in the OPDept, and the products in the warehouse sorted.

3.3 Class projection

Class projection refers mainly to how the model was thought, how the problem was divided in sub-problems, each sub-problem representing more or less the introduction of a new class. First I will start by mentioning exactly how my problem was divided into packages and afterwards each package with its own classes. I begin by creating the four packages I

used: the first one being called „view" the second one being called „model", as a third one is a subpackage of the model package and the fourth one „controllers". I named them intuitively because the first one handles the interface part, the part that deals with the user, the second one handles the implementation part, the thirs package is the one that provides the comparators by which the elements in the warehouse and opdept are placed in the BST and the fourth one, is the one that contains the controllers of the interfaces. I will begin with the interface, then I will continue with the models, comparators and controller part.

3.4 Interface – package view

LoginFrame.java - extends JPanel , with which we made a login window for the application -some elements of the interface are: labels , text fields, radio buttons, buttons and JPasswordField. This class has only 3 instance variables, presented below:

private static JRadioButton *userButton*: a JradioButton to choose to login in as a user
private JRadioButton adminButton: a JradioButton to choose to login in as an admin
private static String *userName*: a String to keep the name of the user that has logged in.

The main method of this class „createLoginWindow", constructs a Jpanel that has one text area where the user can introduce his name, a password field, and 2 buttons „OK" and „Cancel".We also have setters and getters for the instance variable:
public static JRadioButton getUserButton(): get the status of the user button
public void setUserName(String userName): set the username
public static String getUserName():get the name of the user
public void setAdminButton(JRadioButton adminButton): set the status of the user button

The most important thing from this class is the main function that calls the function of creating a new login window, from where the application starts:

```
public static void main(String[] args) {
                LoginFrame p = new LoginFrame();
                p.createLoginWindow();}
```

MessageDialogs.java – this class contains the main dialog windows that are created in order to get for exemple the name of the object searched, or all the details needed for an order to be computed. The class does not contain instance variables, and have only 2 methods:
public static String[] addProductPanel(): this method is used for creating a Joptionpane that will collect the data that is put into the fields of the pane, into an array of strings, from where we construct a product,and add the new object to the Warehouse and to a table.

public static String[] addOrderWindow(): this method, like the one presented above is used for creating a Joptionpane that will collect the data that is put into the fields of the pane, into an array of strings, from where we construct an order, and add the new object to the OPDept and to a table.

UserFrame.java- this class represents the GUI for the user. The class contains the following instance variables:

private JPanel topPanel: used only for settle the components into the frame
private JTabbedPane jtp: used for going easy between the frames
private JPanel btnPanel: used to settle the buttons in to the bottom of the page
private JButton addOrder, searchButton: the main buttons representing the actions a user can do
private static JTable *table* :a table where we have the stock of the warehouse
private JScrollPane scrollPane : used especially for the tables to see the data easier
private static JTable *orderTable :* a table where we have all the orders of the orders departament

Constructor of this class places all the components enumerated above on the frame.
We have also getters and setters for the instance variables, and also we set the Listeners for the buttons.

```
public static JTable getTable()
public void setTable(JTable table)
public final void setAddOrderButtonActionListener(final ActionListener a)
public final void setSearchButtonActionListener(final ActionListener a)
public static JTable getOrderTable()
public static void setOrderTable(JTable orderTable)
```

AdminFrame.java- this class represents the GUI for the admin. The class contains the following instance variables:
private JTabbedPane jtp : used to navigate easy between the frames
private JScrollPane scrollPane: used especially for the tables to see the data easier
private static JTable *tableStock:* a table where we have the stock of the warehouse
private static JTable *tableOrder:* a table where we have all the orders of the orders departament
private JButton addButton, delButton, searchButton, seeButton, searchButton1,stock: buttons for the main operations that an admin can operate.

Constructor of this class places all the components enumerated above on the frame.
We have also getters and setters for the instance variables, and also we set the Listeners for the buttons.
public static JTable getTable()
public void setTable(JTable tableStock)
public final void setAddActionListener(**final** ActionListener a)
: add action listener for the add product button
public final void setDelButtonActionListener(**final** ActionListener a)
: add action listener for the delete product button
public final void setSearchButtonActionListener(**final** ActionListener a)
: add action listener for the search product button
public final void setStockButtonActionListener(**final** ActionListener a)
: add action listener for the stock product button
public final void setSeeActionListener(**final** ActionListener a)
: add action listener for the see product button
public final void setSearch1ButtonActionListener(**final** ActionListener a)
: add action listener for the search order product button
public static JTable getTableOrder()
public void setTableOrder(JTable tableOrder)

3.5 Models

3.5.1 Models

Customer.java –the class represents the user that has logged in with it's name and the password.

The class has only one instance variable a String that is actually the name of the user.There is just one method besides the constructor, that gets the name of the user. Implements serializable, in order to be saved, and to recover the information regarding the user.

Order.java- this class is one of the most important classes in this project. It contains the following instance variables:

private int ID: to uniquely identify an order
private Product product: the product that was ordered
private Customer customer: the user that bought a product
private int quantity: the quantity of order
private transient String status: and the status, that is transient, so the variable will not be include in serialization process

The constructor of the Order class:

```
public Order(int id, Product prod, Customer cust, int quant) {

		this.ID = id;
		this.product = prod;
		this.customer = cust;
		this.quantity = quant;
	}
```

The class implements serializable, in order to be saved after the application is closed. We also have getters and setters for the instance variables.

Product.java- the class is one of the 5 mandatory classes for creating the application. The class implements serializable, in order to be saved after the application is closed.

It contains the following instance variables:

private int ID: to uniquely identify the product
private String name: the name of the product
private int quantity: the quantity of the product

The constructor of this class is:

```
public Product(int id, String name, int quantity) {
		this.ID = id;
		this.name = name;
		this.quantity = quantity;
	}
```

We also have getters and setters for the instance variables.

OPDept.java- this class contains a collection of orders. To be more specific a TreeSet. This is the only instance variable of this class. The constructor of the class initialize the TreeSet and sets the criterion for storing into the tree the orders.

There are also some other methods that help using tree sets:

public TreeSet<Order> getOrders() : get the orders
public void setOrders(TreeSet<Order> orders): set the orders
public void addOrder(Order order): add an order to the tree set
public void removeOrder(Order order): remove an order from the tree set
public int getSize(): get the number of the elements from the tree set
public Iterator<Order> getOrdersInAscendingOrder(): gets an iterator for the collection

Warehouse.java- this class represents the stock of the products that the shop owns. The stock is represented as a tree set of orders. The constructor of the class initialize the TreeSet and sets the criterion for storing into the tree the products.

There are also some other methods that help us handling tree sets:

public TreeSet<Product> getStock(): gets the stock of products
public void setStock(TreeSet<Product> stock): sets the stock of products
public void addProduct(Product product): add a product to the stock
public void removeProduct(Product product): remove a product from the stock
public int getSize(): returns the number of products
public Iterator<Product> getProductsInAscendingOrder():gets an iterator for the collection

3.5.2 Models.Comparators

ProductComparator.java &OrderComparator.java –this classes are the criterion by which the elements in the tree sets are stored. By using the comparators, when we try to introduce an element to the tree set, the object will be introduces on a well konw position, and using this it will be easy to search or to extract an object.

Both classes implement two interfaces: Serializable, used especially when we serialize the objects product and order, and Comparator that has only one functions that needs to be implemented –compare- that gives us the order of storing the objects.

```
public class ProductComparator implements Comparator<Product>,Serializable {
	public int compare(Product product1, Product product2) {

	}

}
```

3.6 Controllers

SerializableManager.java- the role of this class is to save the classes that implement the Serializable interface when the program ends, and to restore them when the application starts again. This class has no instance variables, but has 4 methods:

public Warehouse deserializeWharehouse(): this method will take from a file input stream the contents and creates an object of type warehouse that is returned by the method

public void serializaWarehouse(Warehouse wh): this method will take as an argument the class Warehouse that needs to be serialized, and write it's content to an output file stream

public void serializaOrders(OPDept ord) : this method will take as an argument the class OPDept that needs to be serialized, and write it's content to an output file stream

public OPDept deserializeOrders():this method will take from a file input stream the contents and creates an object of type OPDept that is returned by the method

AdminFrameController.java- this class creates the frame AdminFrame that we created in the package View and adds ActionListener to the buttons to handle the events given. To do that, we create classes AddPorductListener, DeleteProductListener, SearchListener, SeeListener, StockListener that implements ActionListener. This implies the implementation of the methos actionPerformed.

```
public AdminFrameController() {
frame.setAddActionListener(new AddButtonActionListener());
frame.setDelButtonActionListener(new DeleteButtonActionListener());
frame.setSearchButtonActionListener(new SearchButtonActionListener());
frame.setSeeActionListener(new SeeButtonActionListener());
frame.setSearch1ButtonActionListener(new Search1ButtonActionListener());
frame.setStockButtonActionListener(new StockButtonActionListener());
this.stock=manager.deserializeWharehouse();
this.orders=manager.deserializeOrders();}
```

UserFrameController.java- this class creates the frame UserFrame that we created in the package View and adds ActionListener to the buttons to handle the events given.To do that, we create classes AddOrderListener, SearchListener that implements ActionListener. This implies the implementation of the methos actionPerformed.

```
public UserFrameController() {
frame.setAddOrderButtonActionListener(new AddOrderButtonActionListener());
frame.setSearchButtonActionListener(new SearchProductButtonActionListener());
this.stock = manager.deserializeWharehouse();
this.order = manager.deserializeOrders();
}
```

4.Using and testing the application

In order to use the application run the app from the Eclipse. There will be a login window opened where the user will choose how to log in, as a user or as an admin. There will be

needed a username and a password. Depending on the way the user chooses to log in then a frame is available. On the frame will be displayed the stock, and the admin has the right to change it, but the user can only search for a product into the stock.

5. Results

The application is an user friendly and useful application to perform basic management operations such as: addind a product/order, deleting a product/order, search for a product/order, change the stock, see history. Output results of the application are actually the results of operations performed on the stock.The information and the data are displayed in a table, as can be seen from the picture that shows the graphical user interface.The results of every action that it is done upon the stock can be seen in the tables. Even though being limited, this application can be considered a good and helpful tool in understanding the operations that stand behind an order management system.

6. Conclusion and future developments

Achieving such a program may be hard both in terms of algorithms, graphical structure. The best is to represent the customers and products as a tree type structure because this kind of structure makes it easier for some operations to be done: add, remove or search for an element from the structure.

For a better performance there should be implemented all cases where exceptions can occur and the application stops working due to an error made by the user or the admin. Also, this project could be implemented with a data base in the backround, to be more close to the reality and usefulness of this kind of applications.

7. Bibliography

http://stackoverflow.com/
http://docs.oracle.com/
http://users.utcluj.ro/~jim/OOPE/