Technical University of Cluj-Napoca

Assignment 2

Order Management

Name: Tudor Alexandru Haba

Group:30425

Table of contents

1. Introduction

1.1 Task objectives . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .page3

1.2 Personal approach . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . page3

2. Problem description

2.1 Modelling. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . page3

2.2 Scenarios . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . page4

2.3Use cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . page4

3. Projection

3.1 UML diagrams. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . page6

\* a) Use Case Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .page6

\* b) Class Diagram. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .page8

\* c) Sequence Diagram. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .page8

\* d) Activity Diagram. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .page11

3.2 Data structures. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . page11

3.3 Class projection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . page12

3.4 The algorithms. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . page14

4. Implemetation and Testing. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . page15

5. Results. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . page15

6. Conclussions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . page15

1. Introduction

1.1 Task objectives

The task is: Consider an application Order Management for processing customer orders. The application uses (minimally) the following classes: Order, O P Dept (Order Processing Department), Customer, Product, and Warehouse. The classes O P Dept and Warehouse use a Binary Search Tree for storing orders.

1.2 Personal approach

I have tried to design each of the capabilities of the application from the easiest to the most difficult ones. I have solved the addition of new products, the deletion of products, searching and so on.

2. Problem description

An Order Management system has to be simple and easy to understand because it will most probably be used by people that are not necessarily accustomed to technology/computers. All online shops (i.e amazon ,emag) have an OrderManagement system that is very compact and easy to understand/ use.

Items within an eShop have to be easy to search for; ordering of the items has to be quick and simple; this is the basis of a good Order Management system.

The Order Management as a theoretical problem seems a bit tangled at first but by taking a reasonably good approach I have been able to solve most of the required tasks. I have started with the Graphical User Interface because that is the backbone of this assignment. By using a simple and efficient Graphical User Interface I thought I could tackle all the issues that might come as they came.

2.1 Modeling

This time the thinking is pretty straight forward because I knew what data straucture to use and why. By using Binary Search Trees the sorting of the items is pretty easy beceause Java already has this data structure implemented and it only needs to be called.

I will not lie, at first I have used simple lists just to be sure I can add items correctly and display them in the console and then subsequently in the Graphical User Interface.

When I was sure that I could add / delete items correctly and safely in / from the list I switched to trees.

2.2 Scenarios

This time the number of classes increased significantly. I still use the Main class that has the job of instantiating an object of type Authenticate. This class has to job of creating the login screen that will allow normal users , customers to log on and do their shopping and tracking their history. It also allows the more privileged administrators to watch over the stock of products, add new products in stock, delete products from stock, changing the amount of items in stock (maybe a new shipment arrived).

The login screen is a user / password combo that will only allow a person with a correct combination to enter and place orders/track them.

The classes that branch of from Authentificate are CustomerOP and AdminOp. Those clases are part Graphical User Interface and part logic class. I will talk about them in detail later.

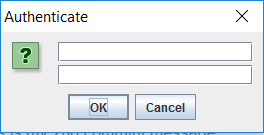
I have a class Product which is a framework for any products in the shop. All products have a distinct name, a price, a colour and number of items on stock.

The products are stored in the class Warehouse. Also some of the pre existing products, the ones that are almost always on stock are stored there.

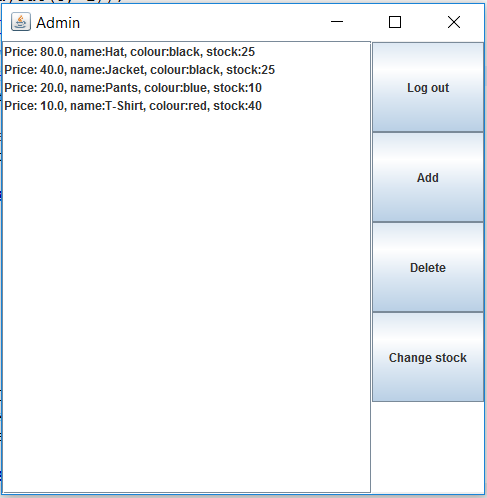
2.3 Use cases

The use cases are the ones the user will benefit from so they have to be simple enough so that the user won’t lose him/herself in but they have to be complex enough to give the user the ability to solve polynomial operations with relative ease.

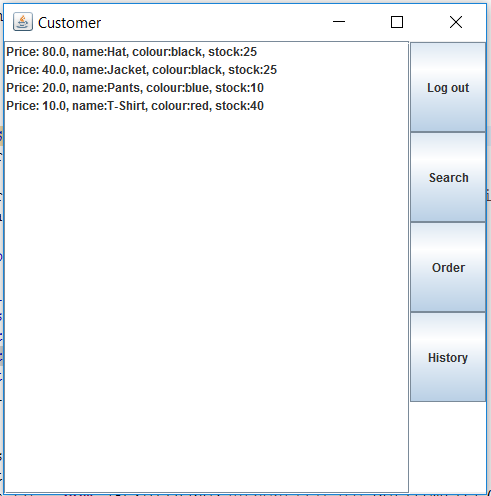
This is the interface I came up with for login:



The next Graphical User Interfaces are either the administrator or the user ones.



The next Graphical User Interfaces are either the administrator or the user ones.



Anyone can see that they are simple straightforward panel that will do theit respective jobs. In the Admin Graphical User Interface there are 4 buttons: Log out ( does what is says really ) , add ( adds new items to the warehouse ), delete ( deletes items from the warehouse ) , change stock ( changes the stock of a given item ).

For the user Graphical User Interface a customer can : log out ( self - explanatory ) , search ( searches for a given item in the product binary search tree ) , order\* and history\* .

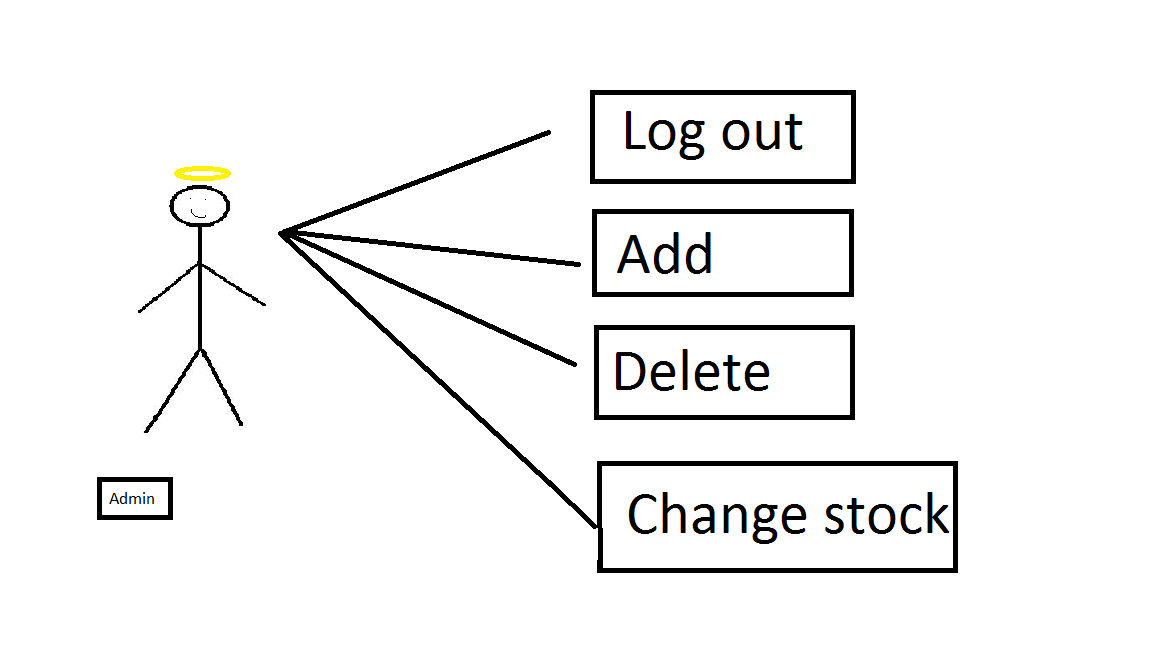
\*both are just theoretical

3. Projection

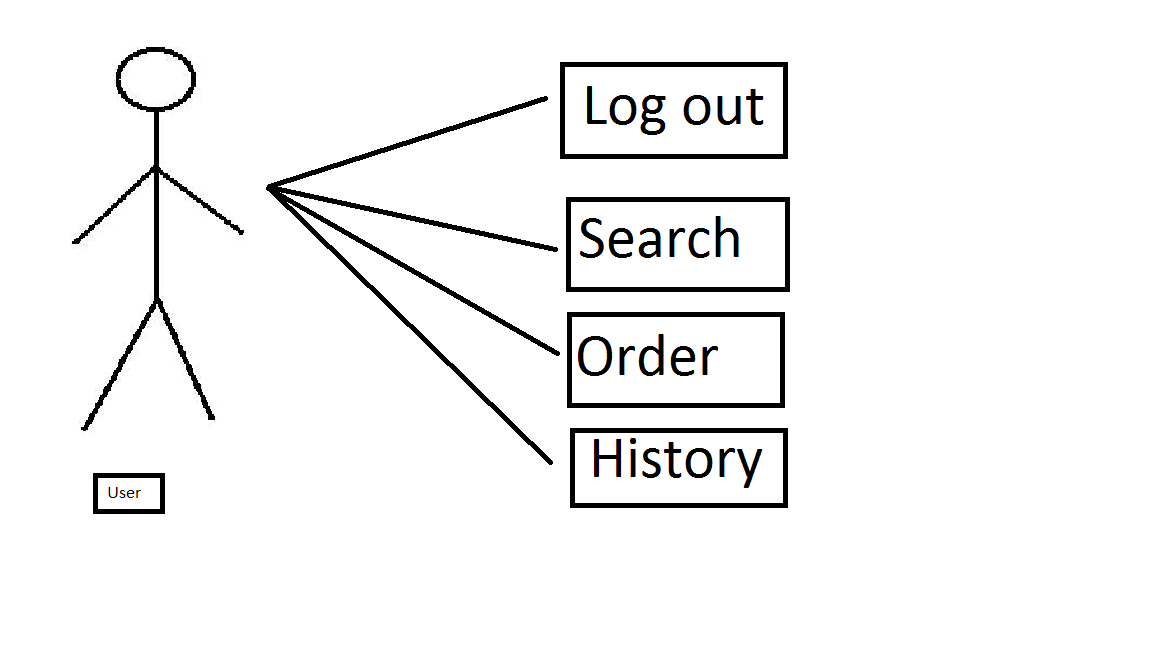
3.1 UML diagrams

a)Use Case Diagram:

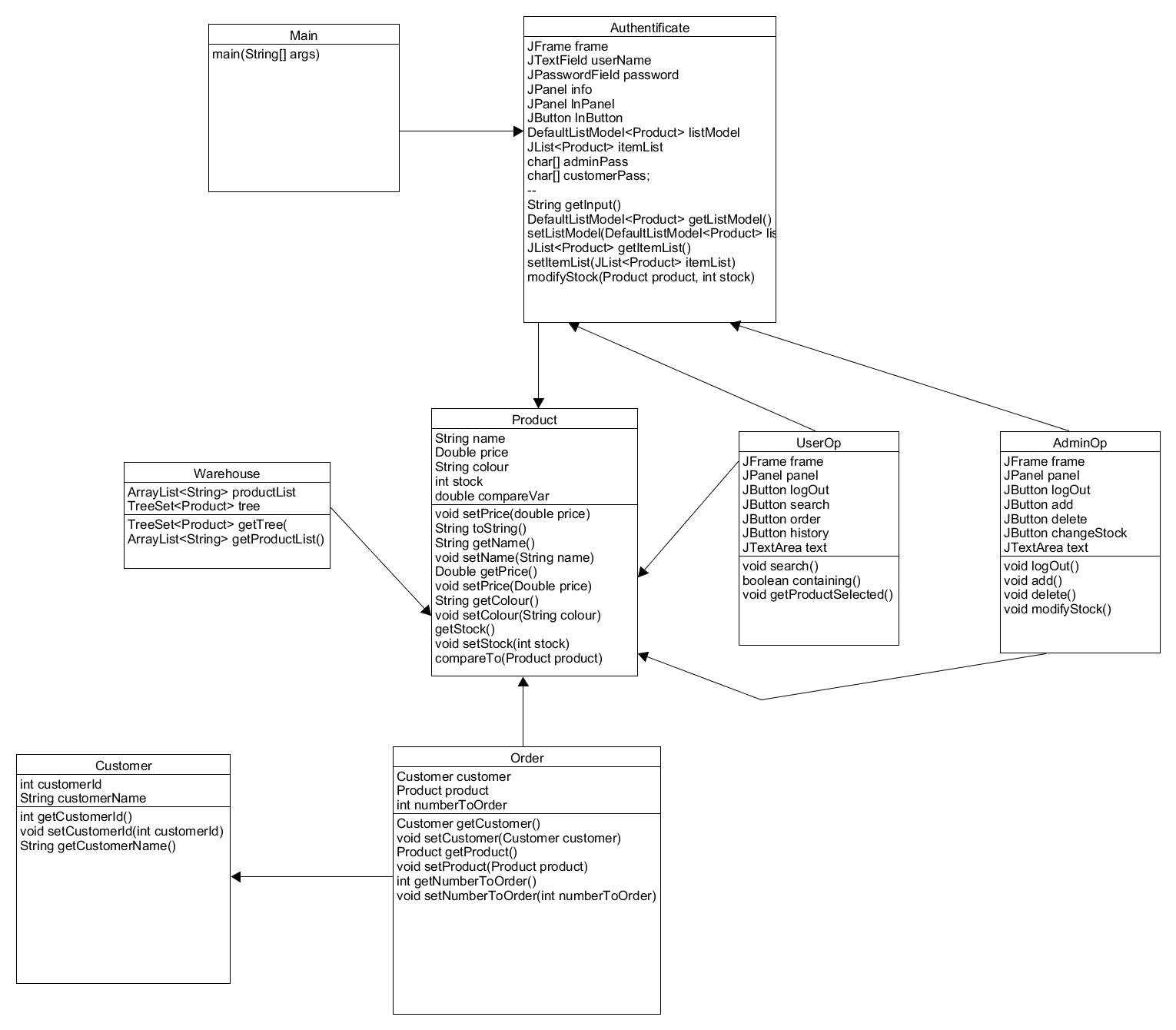
For the administrator part of the Graphical User Interface:



For the user part of the Graphical User Interface:

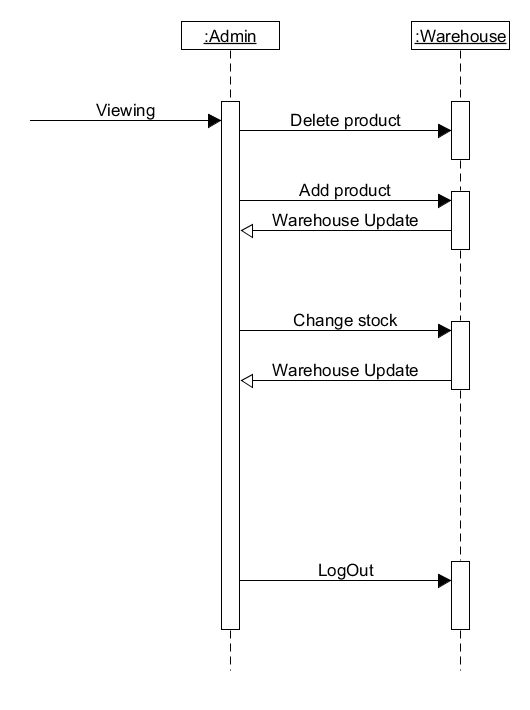


b) Class Diagram:

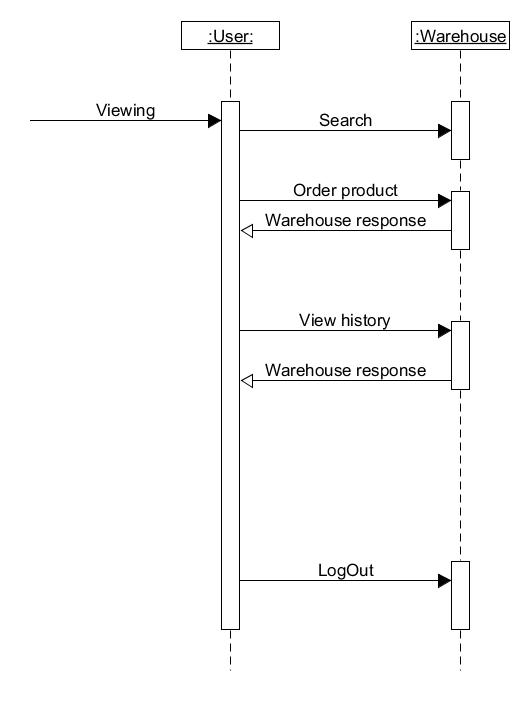


c) Sequence diagrams:

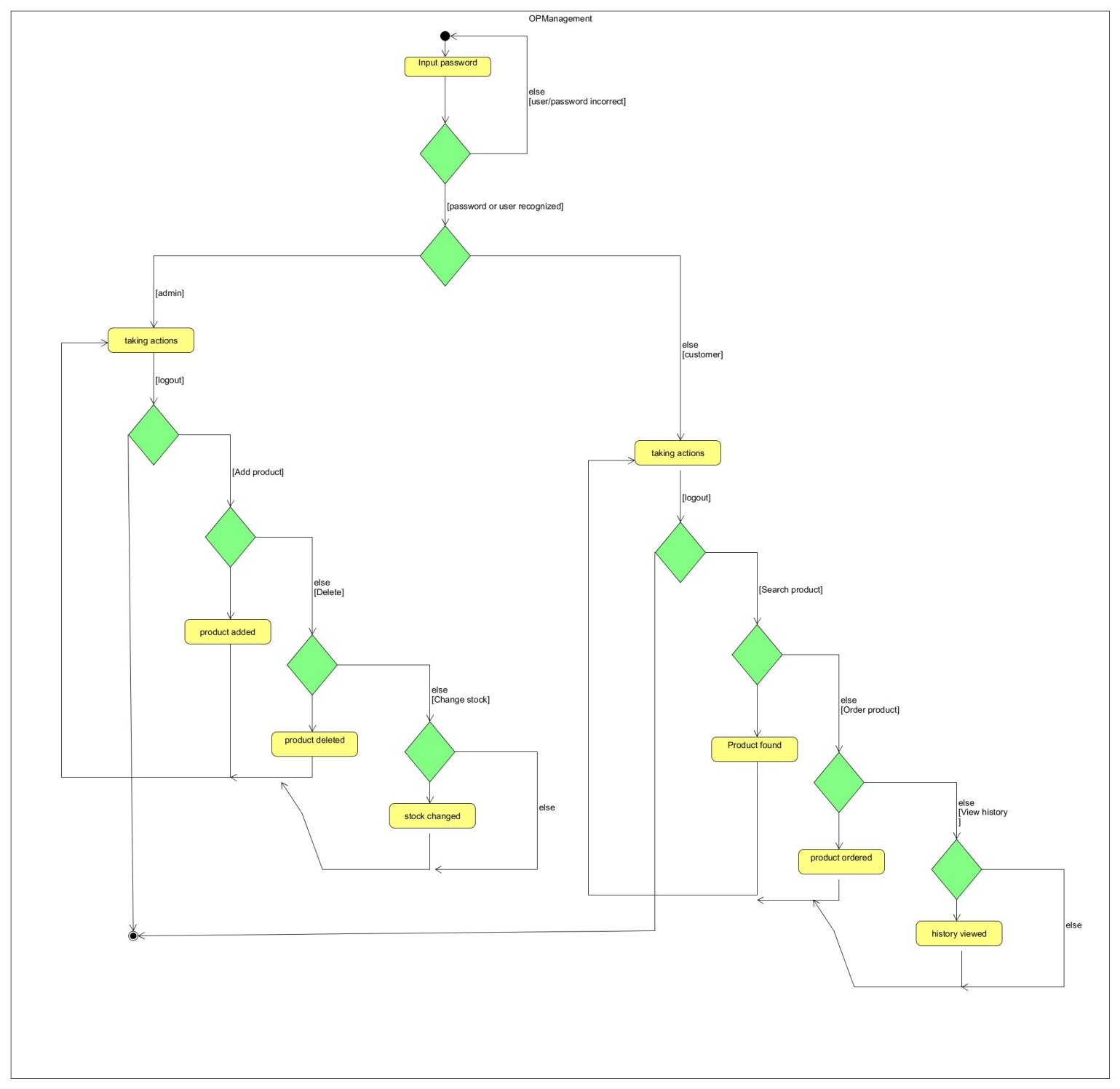
-admin sequence diagram:



-user sequence diagram:



c) Activity diagrams:



3.2 Data structures

I have used both classic ArrayLists and treeSets for completing the assignment. The treeSet is the main datastructure used though. The ArrayLists are used only when the customer searches for a certain product.

3.3 Class projection

I have used classes that were mandatory and a couple more for the Graphical User Interface.

The classes that I have used are: AdminOp.java, Authentificate.java, Customer ,java, CustomerOp.java, Main.java, OPDepartment.java, order.java, Product.java, UserGUI.java, warehouse.java.

Now I will explain in detail what I have done in each class ( + methods ) :

**Main.java**

This class contains the main method where an object of type Authentificate ( ) is instantiated .

**Authentificate.java**

This is the class where the main the Graphical User Interface is implemented . It is also the place where objects of type AdminOp and CustomerOp are instantianted.

The Authentificate class contains the hard copies of the user/password combo of both admin and customer.

This class creates and login window where a person can input their username and password to utilize the system. IF the username/password are incorrect then a message dialog will apper and the program will close for safety reasons. If the username & password are correct then depending onbeing a admin user or customer user a different panel will appear.

The methods used in this class are:

private String getInput ( ) - this method is used to input the password and username. If the passwords and usernames match it will return them in the main method; if not a “mistake” string will be return and it will signal the end of the program.

public DefaultListModel<Product> getListModel() - getter for the listModel.

public void setListModel(DefaultListModel<Product> listModel) -setter for the listModel.

public JList<Product> getItemList()- getter for JList produt.

public void setItemList(JList<Product> itemList) -setter for JList produt.

public void modifyStock(Product product, int stock) – modifies the stock of a given product.

**AdminOp.java**

This class creates the G U I for the the Admin user. It creates buttons with different capabilities then the one a normal user would have. The viewing of the products is the same although.

The methods used in this class are:

private void logout ( ) – self explanatory.

private void add(Warehouse warehouse, Authentificate authentificate)- this methods adds new products to the setTree. The attributes of the product ( type , color , price ,stock ) are introduced through a number of textfields.

private void delete ( Authentificate authentificate ) – deletes the selected product from the setTree.

private void modifyStock(Authentificate authentificate) – modifies the stock of the selected product with a number introduced through a textfield. The number has to be integer. The change in the stock appears in real time.

**CustomerOp.java**

The customerOp class is very similar to the AdminOp class because it has a similar look and feel to it . The buttons ( most of them ) have different capabilities though. The buttons are: LogOut, Search, Order , History.

The methods used in this class are:

private void search(Warehouse warehouse, Authentificate authentificate)- this methods does an excellent search of products by their name. It works alongside 2 other methods for doing this job.

public boolean containing(ArrayList<String> list, String name) - this is the method that uses the ArrayList that I was writing about earlier. If this Boolean return true that means it has found a match in the tree to a product with the same name as the on introduced.

public void getProductSelected(String name, Warehouse warehouse, Authentificate authentificate) - this magical method checks if the name in the seach bar corresponds to any products in the tree. If so then the index will go directly to the the product that matches it.

**Product.java**

In this class it is defined what a product is and how it will look like. I chose to give each product 4 attributes. The attributes are: name, price, colour, stock. I hav use lots of getters and setters in this class so I could utilize each attribute.

This class also contains the compareTo overridden method for the correct creation of the tree.

The methods that I have used in this class are:

public TreeSet< Produc > getTree ( ) – getter for the tree

3.4 The algorithms

I will describe the algorithms that i have used for making this assignment:

I will start from the AdminOp algorithms:

For adding more products to the tree I have used a button and then a JoptionPane to introduce the data from keyboard:

**private** **void** add(Warehouse warehouse, Authentificate authentificate) {

add.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

JTextField nume = **new** JtextField ( ) ;

JTextField pret = **new** JTextField( ) ;

JTextField colour = **new** JtextField ( );

JTextField stock = **new** JtextField ( );

Object[] ob = { nume , pret , colour , stock };

JOptionPane.*showConfirmDialog*(**null**, ob,

"Add”JOptionPane.***OK\_CANCEL\_OPTION***) ;

Product product = **new** Product(nume.getText(),Double.*parseDouble*(pret.getText()), colour.getText(),Integer.*parseInt*(stock.getText( ) ) );

authentificate.getListModel().addElement(product);

warehouse.getProductList().add(product.getName( ));

} } ) ; }

The deletion of a product is much simpler showing us once again that destroying is a lot easier than building:

**Private** **void** delete( Authentificate authentificate ) {

delete.addActionListener(**new** ActionListener( ) {

**public** **void** actionPerformed(ActionEvent e) {

**int** selectedIndex = authentificate.getItemList( ).getSelectedIndex( );

**if** ( selectedIndex ! = -1 ) {

authentificate.getListModel( ).remove(selectedIndex );

}}});}

Modifiyng the stock was more difficult that i would have imagined but at the end i was able to do it:

**private** **void** modifyStock(Authentificate authentificate) {

changeStock.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

JTextField stock = **new** JTextField();

Object[] ob = { stock };

JOptionPane.*showConfirmDialog*(**null**, ob, "ModifyStock", JOptionPane.***OK\_CANCEL\_OPTION***);

authentificate.modifyStock((Product) authentificate.getItemList().getSelectedValue(),

Integer.*parseInt*(stock.getText( ) ) ) ; } } ) ; }

The CustomerOp algorithm that i have utilized for searching through a tree with the help of a list is made by 2 methods and it is the following:

**private** **void** search(Warehouse warehouse, Authentificate authentificate) {

search.addActionListener ( **new** ActionListener ( ) {

**public** **void** actionPerformed(ActionEvent e ) {

JTextField product = **new** JtextField ( );

Object [ ] ob = { product };

JOptionPane.*showConfirmDialog*( **null**, ob, "Search", JOptionPane.***OK\_CANCEL\_OPTION***);

getProductSelected(product.getText( ), warehouse, authentificate ) ; } } ) ; }

The second part of the algorithm looks like this:

**public** **void** getProductSelected ( String name, Warehouse warehouse, Authentificate authentificate ) {

**for** ( Product product : warehouse.getTree( )) {

**if** ( product.getName( ).contains( name )) {

**int** index = authentificate . getListModel ( ) .indexOf(product);

authentificate . getItemList( ) . setSelectedIndex ( index ) ; } } }

4.Implementation and testing

I have implemented this program in Eclipse and I have used algorithms that could do with some improvement. The data strucyures I have used are quite solid this time but I will say that the algoithms could do with a bit of improving.

The input is not checked by try and catch so if the user does something strange errors can appear. The graphical user interface cannot really generate errors because it does not let the user go that crazy. Checking for all possible bugs will be done in a future version.

Also I have not been able to make it possible for the customer to order or view the history but that could be done in a future version.

5.Results

This is a good enough application to see the input from keyboard directly to textfield with the capability to search through a setTree, delete / add products . Because it is made in Java, the application is highly portable on any operating system that has Java JDK.

6. Conclusions

This has been a rather interesting assignment to work on. Even though it took a while I believe I have done a decent job. The Graphical User Interface has been quite a challenge this time but also quite fun.