**Technical University of Cluj-Napoca**

**Faculty of Automation and Computer Science**

**Computer Science Department**

**Programming Techniques**

**Second Homework: Order Management System**

**Mihoc Nicolae**

**Group: 30425**

Table of Contents

*1. Problem Specification . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3*

*1.1 Analysis of the problem . . . . . . . . . . . . . . . . . . . . . . 3*

*1.2 Visualization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3*

*2. Projection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .4*

*2.1 Class Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5*

*2.1 Sequence Diagrams . . . . . . . . . . . . . . . . . . . . . . . . . . 7*

*2.2 Activity Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8*

*3. Design and Implementation . . . . . . . . . . . . . . . . . . . . . . . . . 9*

*3.1 Data Structures . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10*

*3.2 Packages . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10*

*3.3 Classes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .11*

*4. Using & Testing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .18*

*5. Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18*

*6. Future Development . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18*

*7. Bibliography . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19*

**1. Problem Specification**

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.

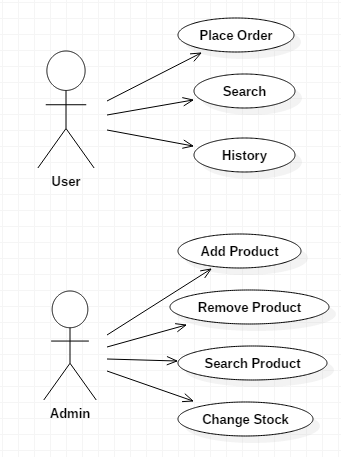
d. Design, write and test a Java program for order management using the classes designed at question c). The program should include a set of utility operations such as under-stock, over-stock, totals, filters, etc.

**1.1 Analysis of the problem**

We are required to implement an application that behaves as an online store these days. The hypothesis is simple: we have a customer who can buy and an administrator who manages the store. Each one of these persons can do specific things. For example, the customer can visualize the existing products or search for a desired product and once he has found what he was looking for he can place a order and the application will tell him if the order request was succesful or there were not enough products in stock and the order could not be placed. Also if the customer desires he can visualize his/hers order history to remember what he/she bought before. The administrator has the job of managing the store. He ca see the existing products, modify the quantity in stock if it is at 0 or he thinks more products are necessary, search in the whole warehouse for a specific product and add or remove products at any time.

**1.2 Visualization**

To help us see better I will present next the user case diagram from my perspective so visualize the whole process from these 2 angles: customer and admin.

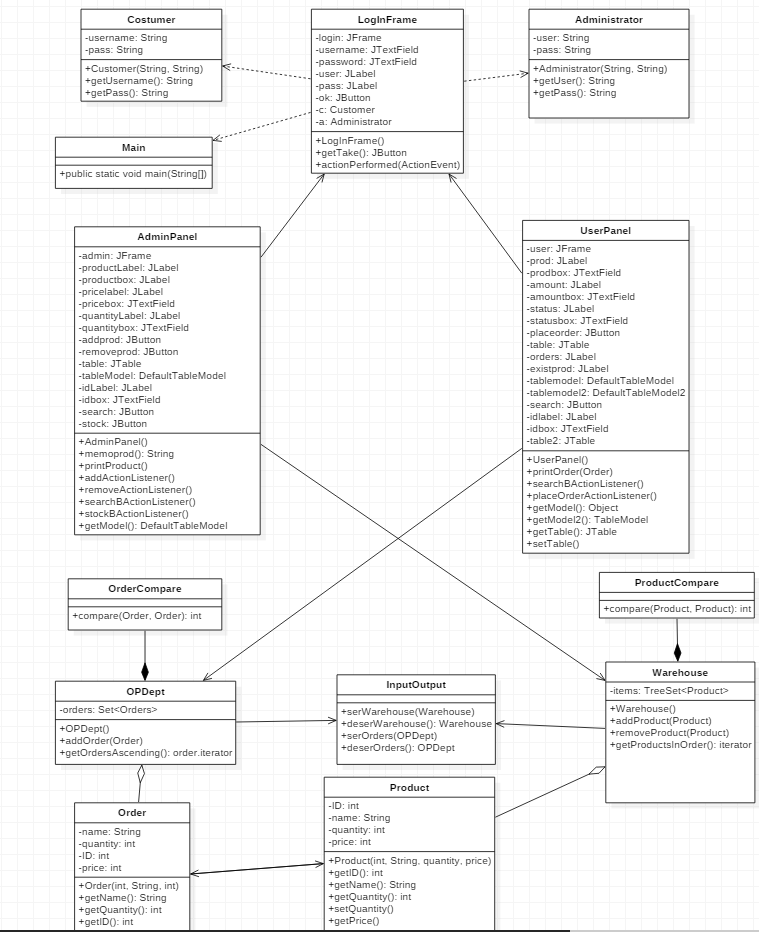


**2. Projection**

Because the use case diagram does not give us to many details about the whole process in addition some more diagrams must be presented.

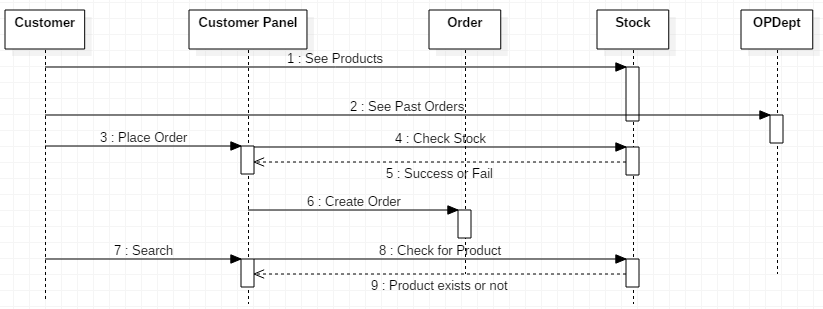
**2.1 Class Diagram**

To better understand what we are dealing with we need the Class Diagram because is the most complete diagram of them all. In this type of diagram all the classes are represented and also all the links between them which makes it easier to follow form the beggining to the end.



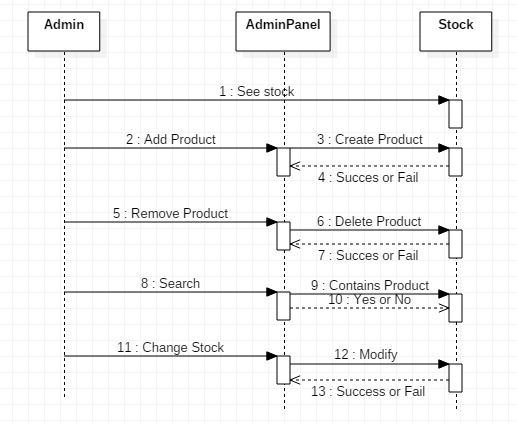
**2.2 Sequence Diagrams**

Sequence diagrams show us the order , how the proccesses are interacting with each other and the participating objects. In this case I made a sequence diagram for the customer and a sequence diagram for the administrator.



In this diagram we can see what happens “in the back” when the customer places a command in the application or searches for something etc.

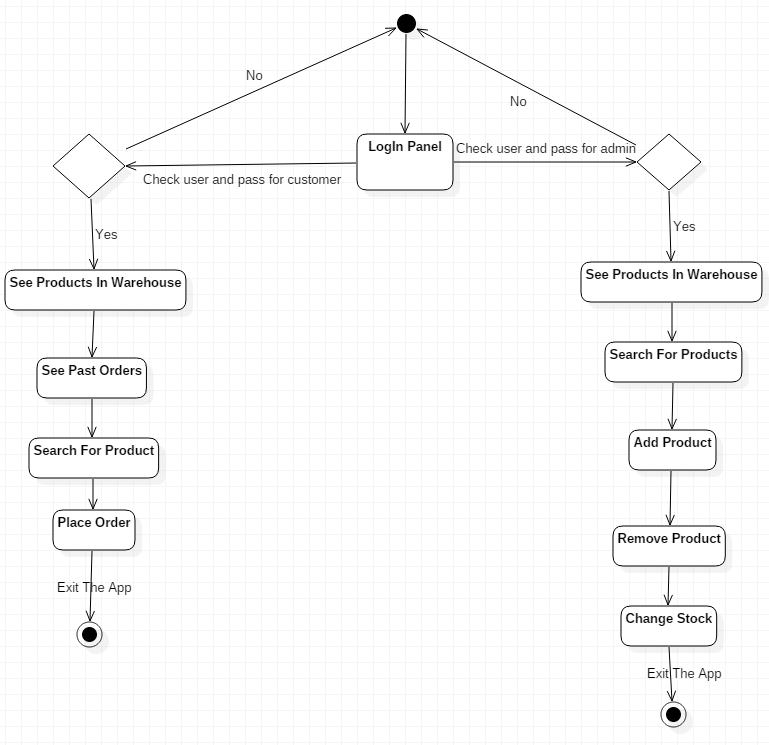
On the other hand we have the administrator diagram which shows us what happens when de admin lurks in the warehouse and changes products. He also interacts with the application just like the customer but with a different account just for him.



**2.3 Activity Diagram**

The final diagram is an activity diagram which shows the execution step by step with all the changes that can happen at any step inspired by the use case diagram.

On one side is the customer and on the other side is the administrator. The principle is the same: each one of them enters their login data in a panel and if the login was succesful they can do stuff, if not they get back to the initial login state and can go no further. After that they end the session by closing the application.



**3. Design and Implementation**

In this section I will describe step by step what have I done to solve the proposed problem starting with the data structures and up to the tiny details of the classes I implemented.

**3.1 Data Structures**

Like in any other problem first we need the primitive type such as integers or strings and after that the ones I will use to get the job done. For this problem I will use the implemented TreeSet from Java to help be store the orders and the products in an arranged way so that it will be easy to find. This will be implemented only for the Warehouse class and the OPDept class. Besides this I will need special objects from the classes Product, Order, Warehouse to be able to link them all together.

**3.2 Packages**

Like in any project I need to implements first some packages and in them there will be the classes that specialize for some specific task. I think 3 packages will be enough for this task.

First package is called “Design” and is the packages that contains the mandatory classes from the problem specification + 2 more classes which I considered to be necessary.

Second package is called “Read\_Write”. This package contains only one class that is responsible to serialize and deserialize the data from the app.

Third package is called “Comparators” and it contains 2 classes that implement the comparator interface and they handle the order in the warehouse for products and in OPDept for orders.

Fourth package is called Frames and it contains 3 classes which handle the graphical user interface for each case and the action listeners.

**3.3 Classes**

In this section I will describe in detail what each class does and what it contains.

The first package contains the classes: Administrator, Customer, Main, OPDept, Order, Product and Warehouse.

The class Administrator has:

Attributes:

-private String user;

-private String password;

Methods:

-public Administrator(String user, String password): the constructor of the class that creates the object with these fields at initialization.

-public String getUser(): returns the user.

-public String getPass(): returns the password.

The class Customer has:

Attributes:

-private String username;

-private String password;

Methods:

-public Customer(String username, String password): the constructor which initilizes the object Customer.

-public getUsername(): returns the username.

-public getPassword(): returns the password.

The class OPDept has:

Attributes:

-private TreeSet<Order> orders;

Methods:

-public OPDept(): the constructor to initialize the TreeSet for storing the Orders based on the Comparator class.

-public void addOrder(Order o): the method to add an order

-public Iterator<Order> getOrdersAscending(): returns the iterator for the orders.

This class implements also the Serializable interface to help the application to remember all the orders after the session closed.

The class Order has:

Attributes:

-private String name;

-private int quantity;

-private int ID;

-private int price;

Methods:

-public Order(int ID, String name, int quantity, int price): creates the Order object with the specified properties.

-public String getName(): returns the name.

-public int getQuantity(): returns the quantity.

-public int getID(): returns the id.

-public int getPrice(): returns the price.

-public void setPrice(int price): method to set the desired price.

The class Product has:

Attributes:

-private int ID;

-private String name;

-private int quantity;

-private int price;

Methods:

-public Product(int ID, String name, int quantity, int price): constructor that creates the general description of a product.

-public String getName(): returns the name.

-public int getQuantity(): returns the quantity.

-public int getID(): returns the id.

-public int getPrice(): returns the price.

-public void setQuantity(int quantity): method to set the desired quantity.

The class Warehouse has:

Attributes:

-private TreeSet<Product> items;

Methods:

-public Warehouse(): the constructor initializes the TreeSet which will contain products in sorted order because of the Comparator class.

-public void addProduct(Product i): method to add a product to the set.

-public void removeProduct(Product i): method to remove a product from the set.

-public Iterator<Product> getProductsInOrder(): method that helps me to iterate through products in the warehouse.

This class also implements the Serializable interface to remember the products in the warehouse after the session ended.

The class Main is the one that allows me to do all the things in this project and it extends Jframe and creates the first frame when launched. Also I used this class to add an order and a product for the first time to test if the serialization/deserialization was working.

The second package contains only the class Input\_Output. This class has only the job to serialize or deserialize the data introduced. This means that it stores the orders and the products in 2 different files which are restored every time the customer or the administrator open the application.

This class has 4 methods:

-public void serWarehouse(Warehouse w): this method writes the existing data in a file called “warehouse.ser” stored in the project folder.

-public Warehouse deserWarehouse(): this method initializes an empty warehouse and reads the input from the same file which is recived as input and also iterates the items in the warehouse and places them in a table which I have in the graphical user interface.

-public void serOrders(OPDept order): the method writes the order data in a file called “orders.ser” stored in the project folder.

-public OPDept deserOrders(): like in the previous case an empty order processing department is created and the data is read from the file and placed back in the coresponding place. Also the orders are iterated and placed in a table in the graphical user interface.

The third package contains 2 classes: OrderCompare and ProductCompare. These classes which are used in the Warehouse class and OPDept class have here the definition and are used to store in order the products or orders. They implement the Comparator interface and also the Serializable interface and return an integer value. The integer can be -1 when the first product/order is smaller than the second, 0 if they are equal and 1 if the first is bigger (based on the ID).

The fourth and final package has 3 classes which are the ones responsible with all that we see on screen and the actions that happen when something is done in them.

In this problem we need to decide if the person that wants to use the application is the administrator or a customer. To decide between them I implemented a class called LogIn Frame which based on the username and password entered opens the Customer Panel or the Admin Panel.

The class LogInFrame implements the interface ActionListener and has the following attributes:

-private JFrame login: the object for the frame.

-private JTextField username: field for username.

-private JpasswordField password: field for password.

-private Jlabel user, Jlabel pass;

-private Jbutton take: a button that is not visible but when the customer logins is setSelected() to know when the table is opened in the customer panel or in the admin panel

-private Customer c, Administrator a: 2 object that inherit the properties from the constructors of the classes Customer and Administrator to decide at login

Methods:

-public LogInFrame(): constructor that contains all the details of the frame. I chose not to have any layout so i placed manually all the objects inside this frame using the command setBounds(int x, int y, int width, int height) . Here i also add actionListener to the Jbutton and set some more options like: visibility, location and close operation.

-public void actionPerfomed(ActionEvent event): when the button is pressed the credentials are verified against the one set by me in the constructor. If they match the customer or the admin may enter each one with his different panel and if not nothing happens.

-public static Jbutton getButton(): method to get the buttons state.

The next class is the UserPanel class which handles the part of the customer in this problem. After the log in was succesful the customer panel opens. In this class I have the following attributes:

-private Jframe user: frame object.

-private Jlabel prod, amount, status, orders, existprod, idlabel: just labels for my boxes and tables.

-private JtextField prodbox, amountbox, statusbox, idbox: boxes for different parts.

-private Jbutton placeorder, search;

-private Jtable table, DefaultTableModel tableModel;

-private Warehouse w, Product p, InputOutput x, OPDept y: special object from other classes which I need in the methods

Methods:

-public UserPanel(): creates the customer panel which has no layouts and all that stuff is placed with setBounds. It also contains 2 tables: one for orders and one for existing products , initializations of the special objects mentioned and action listeners for the buttons.

-public void printOrder(Order o): method to print the orders in the table

-class SearchBActionListener: iterates through products and when it finds a product in the warehouse that coressponds with the one entered in the box it returns a succes message otherwise a fail one

-class PlaceOrderActionListener: iterates through products and checks it he name and quantity matchers before placing and order. If the quantity is good the order is placed and a message is printed in the status box that was a successful order or that could be a fail if there are not enough products in stock. The order is printed in the table.

-getModel(), getModel2(): method for table format.

-getTable(), setTable(): get and set the tables;

The AdminPanel class is the class that opens the frame which contains the Admin Panel. It is similar with the Customer as design.

Attributes:

-private Jframe admin: frame object.

-private Jlabel productlabel, pricelabel, quantitylabel, idlabel;

-private JtextField productbox, pricebox, quantitybox, idbox;

-private Jbutton addprod, removeprod, search, stock;

-private Warehouse items, InputOutput x, Product p;

-private DefaultTableModel tableModel, table;

Methods:

-public AdminPanel(): constructor for frame, no layout, objects added with bounds, table definition, actionListeners for buttons and initialization of the special object

-public void printProduct(Product p): prints the name, ID, quantity, price in the table of products.

-class AddActionListener: adds in the warehouse a product with the information described in the constructor and prints the product in the table.

-class RemoveActionListener: the admin enters an id and this iterates through the warehouse to find a match and when it does, if the admin selects the coresponding row that product is removed from the warehouse and table at the same time.

-class StockBActionListener: check if a product with the same ID and name exists in the warehouse and modifies the stock.

-public static int getRow(TableModel model, Object o): method that moves through the table and gets the value at the coresponding row.

-class SearchActionListener: searches in the warehouse for a object with the same name as in the product box and returns a message in case of success or fail.

**4. Using & Testing**

It took me a while to figure out all the aspects of the problem like the fact that the table was not updating when a quantity was changed or with the serialization/deserialization that I never used before but if went fine in the end. This application is easy to use for anybody and simulates the real word shopping.

**5. Results**

In the end I obtained a mini application for shopping and a chance to see both sides of an activity that these days is very common and of high importance. It is not easy and are alot of aspects to take into considerations but is a small step for a bigger application in the future.

**6. Future Development**

My application is far from a perfect one and also is not very complex having only basic operations and details. The improvements are infinite. Some of them may be:

-the admin to accept the order or reject it in his panel;

-the admin to be able to view the orders of the clients and message them with the status of a command;

-the costumer to create a personalized profile;

As the world and the technology improves there will be more and more tricks to add to an app like these.

**7.Bibliography**

<http://stackoverflow.com/>

http://tutorialspoint.com/