**TECHNICAL UNIVERSITY OF CLUJ-NAPOCA**

**Laboratory Work:**

**Order Management**

**Name: Nistor Rares Constantin**

**Group: 30425**

**Content:**

1. **Introduction**

**1.1 Homework objective . . . . . . . . . . . . . . . . . . . . . . . . . Page 3**

1. **Problem solving. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Page 3**
2. **Projection**

**3.1 UML diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Page 3**

**3.2 Data structures . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Page 9**

**3.3 Class projection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Page 9**

**3.4 User interface. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Page 11**

**3.5 Packages . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Page 14**

**3.6 Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Page 14**

1. **Implementation and testing. . . . . . . . . . . . . . . . . . . . . . Page 16**
2. **Results . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Page 17**
3. **Conclusions. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Page 17**
4. **Bibliography. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .Page 18**
5. **Introduction**
   1. **Homework objective**

The objective of the first homework is the following: “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.” In other words we must create a program to simulate an online shop. The program must contain a customer interface with its specific actions and a admin interface, again with its specific actions.

1. **Problem solving**

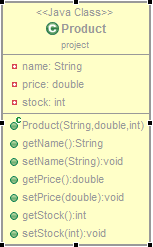
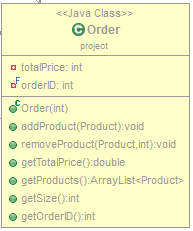
To solve this problem I started by thinking about the operations that an admin and a normal user can do in this application and how to separate those actions from one another. Then I started to think how I can implement those operations using methods and what king of data should I use. After that I started making class diagrams to see how the project should look like. The most interesting part of this project was to save the data in the warehouse, or to show the past orders. For this task I was recommended to use Serializable. The scenarios and use cases are strongly related. The customer can create an order in which he introduces products from the warehouse and can also remove products. The admin os the one who takes care of the warehouse he can add, remove or modify different properties of the products in the warehouse.

1. **Projection**

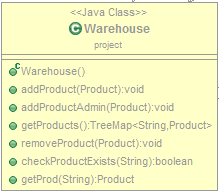
**3.1 UML diagram**

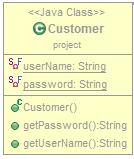
**Class diagram**

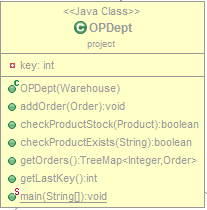
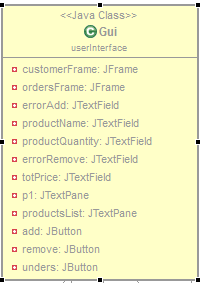
The class diagram show what classes the program contains, what methods and variables they have and how they are connected. Because some classes have a lot of methods or variable they will not be fully shown.

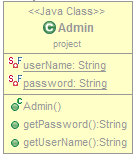








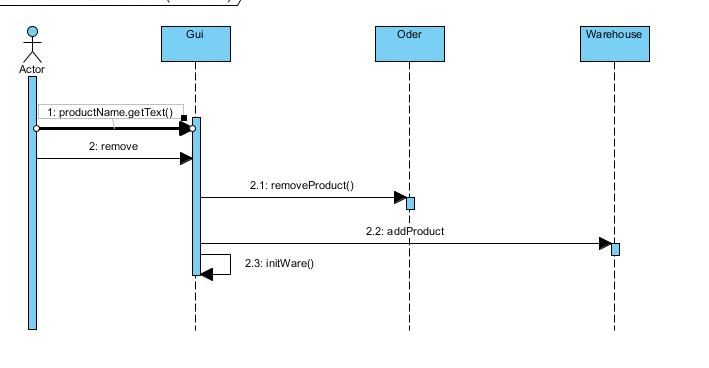




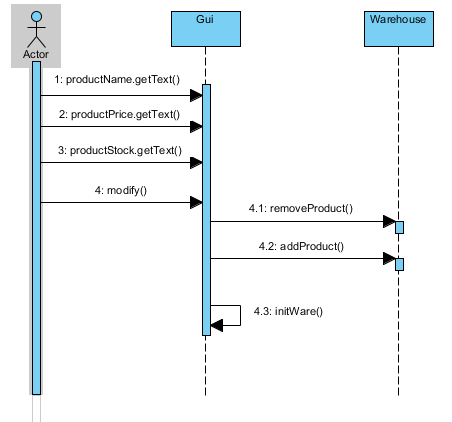
**Sequence diagram**

The sequence diagram shows the methods that are called and from what class when the user performs a action.

The first diagram shows what happens when the users want to remove a product from the order.



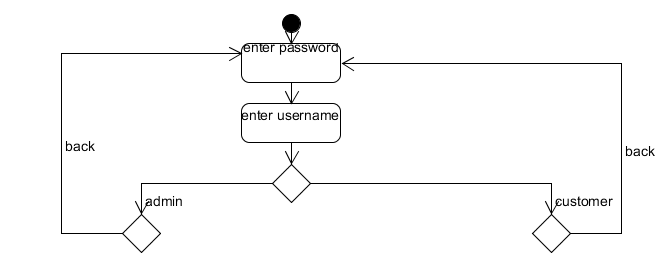
The next sequence diagram show what happens when the admin wants to modify a product from the warehouse.



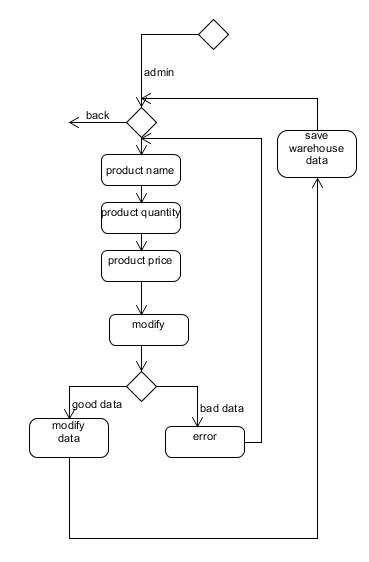
**Activity diagram**

The activity diagram captures the dynamic behavior of the system. The next activity diagram will be split in 3 parts because it is too big.

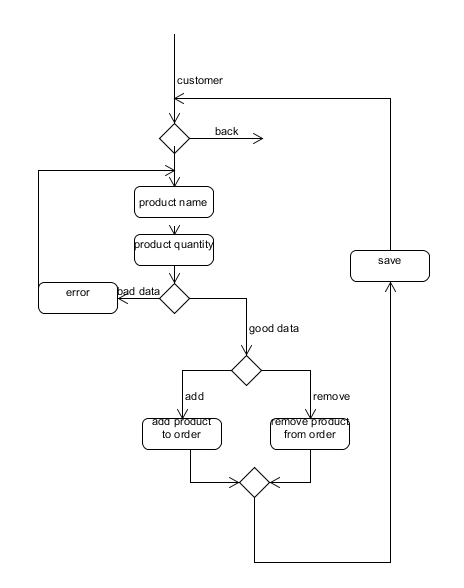
The first part is the login part where the user introduces his name and password to login.



If the user was an admin he can modify products then save the modifications or go back to the login page.



In the last part, in case the user was a customer, he can add or delete items to its order, then save the order or can go back to the login page.



**Use case diagram:**

Remove from order

Add to order

Customer

Order history

Admin

Modify product

**3.2 Data structure**

To save data I used data structures like ArrayList, integers, strings and TreeMap. The integers and strings were used to store different information’s about the products; the ArrayLists were used to store products in orders, and TreeMaps (Binary Search Tree) were used to store products in the warehouse and orders in the Order processing department.

**3.3 Class projections**

Now we will talk about how the program is structured. There are seven classes: Admin, Customer, Product, Order, Warehouse, OPDept and Gui. We will now talk a little about each class.

**Admin:**

The admin class has just 2 variables (userName and password) and 2 methods (getUserName and getPassword). The username and password are hardcoded. This class is used for the login screen to login like an admin.

**Customer:**

This class is like the Admin class with the same 2 methods and 2 variables. And, like before, the username and password variables are hardcoded. This class is also used in the login screen to login like a customer.

**Product:**

This class has 3 variables (name, stock, and price), all attributes of a product, and it has 6 methods to set and get the values of the variables.

**Order:**

The order class is a bit more complex that the previous classes and it has an ArrayList of products, a Customer object (to see who the owner of the order is), and 2 more variables, totalPrice which represents the total cost of all the objects in the order (object \* quantity), and a ID (we will see why this is needed later). As methods this class can add products to the order, remove products from the order, return the total price of the order, return the nr of different products of the order and return an ArrayList of all the products in the order.

**Warehouse:**

This class is contains a TreeMap of products, this is the place where the orders get their products from. Besides the TreeMap there are no other variables is this class. It has methods to check if a product exist, compare it’s quantity to another given value, add products to the warehouse, remove products from the warehouse and return a TreeMap with all the products that are found in the warehouse.

**OPDept**

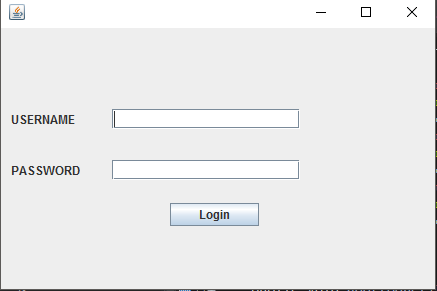
This class is the one that links the Order class and the Warehouse Class togheter. OPDept, like Warehouse, has a TreeMap but this time of orders. OPDept stores orders in the TreeMap and verifies if a product from the order is in the Warehouse or it’s quantity in the warehouse is greater that the quantity in the order. It also returns a TreeMap with all the orders stored.

**Gui**

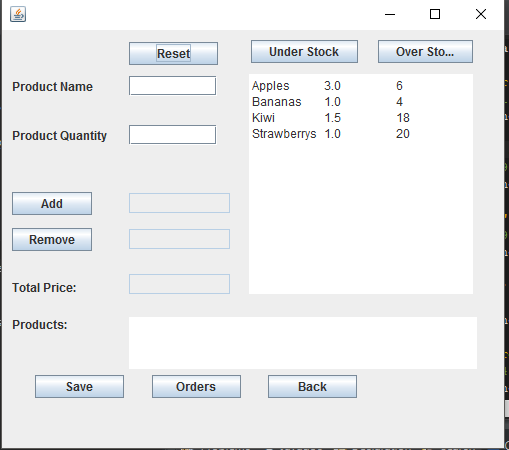
This is by far the biggest and the most complex Class of all. This class has 3 windows, one for the login, one for the admin, one for the user and one to show the orders. We will talk a lot more about this at the next point.

**3.4 User interface**

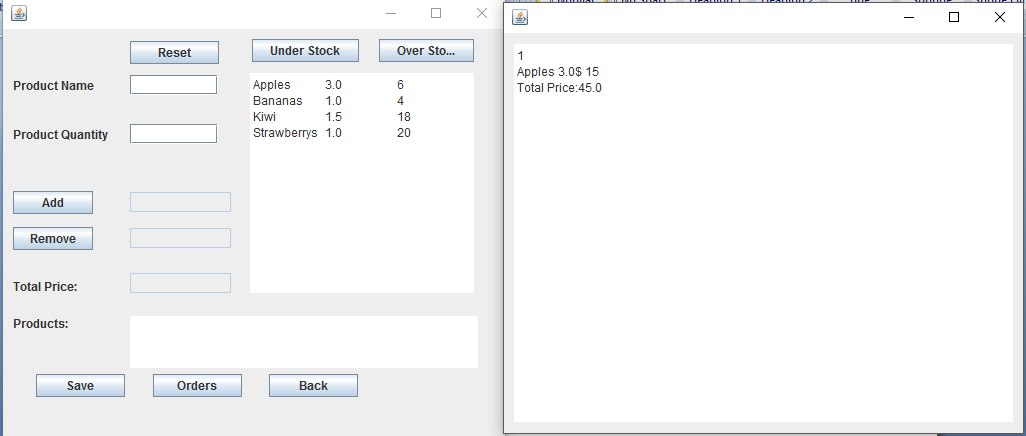
The user interface starts with the login screen.



Depending on the USERNAME and PASSWORD (which can be long to a customer or a admin) you can go to the customer’s window.

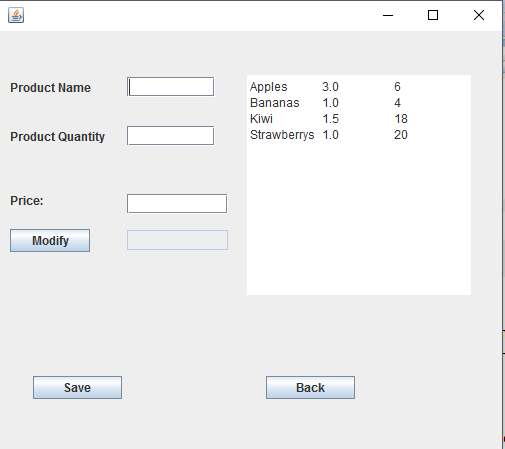


This is the customer’s window. In this window the customer can add a product to its order by completing the Product Name and the Product Quantity fields and pressing the add button or can remove a product by completing the Product Name field and pressing the remove button. When adding a product in the order if the product doesn’t exist in the warehouse, or the quantity is too high or in the quantity field there user introduced a text not a number an error message will be displayed. When removing a product from the order, if the product doesn’t exist in the order again an error message will be displayed. The Total Price field displays the total price of the order. In the right side of the window are the products in the warehouse with their price and quantity. The Over Stock button will change the display of the warehouse so that only the products that have a overstock will be displayed, the under stock button will change de display so that only the under stock products are shown and the reset button will change the display so that all the products are shown again. The Products field shows the products that you currently have in your order, by adding or removing products the display will change. The save button is used when your command is finished and you want to sent it.



The Orders button will make a new window appear which will show the user its previous command what product they contained, in what quantity the price of each product and the total price of the order. The Back sends the user back to the login screen.

And now the admin window:



This is the window where the admin can change information about the products, their name, quantity and price. To do this the user must complete the Product Name field, the Price field and the Product Quantity field. If one of these fields is wrongly completed an error message will be displayed. After the user has completed the fields he must press the modify button. After pressing the button if the product doesn’t exist a new product with the given price and quantity will be created, if the product already exits then its price and quantity will change to the new ones, if the quantity is 0 the that product will be removed. After finishing the changes to the warehouse the save button must be pressed so that the changes will be saved. And like in the user window the back button return the user to the login screen.

**3.5 Packages**

This project has only 2 packages the user Interface package where the Gui class is and the project package where all the other classes are (Product, Warehouse, Order, OPDept, Admin, User).

**3.6 Algorithms**

We will now talk about some of the algorithms that were used in the methods. The methods in the Customer, Admin, Warehouse and Product classes are pretty simple and straight forward so they don’t need to be explained.

In the Order class we will talk a little about the addProduct and removeProduct methods.

-addProduct: this method has as parameter a product. It searches for that product and if it finds it the the new product quantity will be added over the old product quantity and the total price will be recalculate, if it doesn’t find it then the product will be added to the order and again the total price will be recalculated.

-removeProduct: this method has as parameters the product and the position where I is found in the ArrayList. This method removes the product from the order and recalculates the total price.

In the Warehouse class it should be mentioned that the TreeMap is filled in the constructor with data previously saved in a binary file called data.bin.

The OPDept class has the following methods that we will talk about: checkProductStock and checkProductExists.

-checkProductExists: this method has as parameter a product and it checks if this product is found in the warehouse.

- checkProductStock: this method also has as parameter a product but this time verifies if the quantity of the product in the warehouse is greater that the quantity of the given product.

The two functions are usually used together. Also the constructor of the OPDept class loads the TreeMap with orders from a file.

Although the Gui class is the biggest one most of its methods just call other methods (or use objects to call other methods) so we will talk about how the previously described methods are used.

* Adding products to an order: to add a product to a order first we check using the checkProductExists if that product exits in the warehouse, then we check if the product in the warehouse has enough quantity for the order, if not we display a error message, with the checkProductStock method, if it does we use the add method from the Order class.
* Removing products from an order: first we check if the product is found in the order if it is we use the removeProduct method from the order Class
* Saving the order: when we save a order we add the order in the OPDept Treemap then we save the content of that Treemap in a binary file called data1.bin, we also save the TreeMap from the warehouse in a binary file called data.bin. These 2 files are the files from which the constructors of the classes OPDept and Warehouse read the data.
* Display previous orders. To display previous order a new JFrame is created with a JPane. In that JPane we display the content of the TreeMap belonging to the OPDept class.
* Back to the login: we do this by simply making the current frame invisible, using setvisible(fale) and the login frame visible using setvisible(true)
* Login in: to login we check if the data introduced in the fields is the same with the data in the Customer object, if it is we the make the login window invisible and the customer window visible. If the data is equal with the data in the admin object then the admin window is made visible and the login window invisible
* Modify data: to modify data in the admin window we check if the data in the fields is introduced correct if it is then:
  + - If the quantity is 0 we remove the product using removeProduct
    - If the quantity is greater than 0 we delete the previous object with the same name and then create a new one with the data from the input.
* Save data: to save data in the warehouse we act like befor and save the data from the TreeMap in the file data.bin. The data is saved using the Serializable interface.

1. **Implementation and Testing**

Unlike the previous project, this time I tried to make the code as OOP like as possible. Although the implementation is not without its flows it is much better than the previous one. I tried to implement the algorithms as simple as possible and to respect as much as possible the OOP principles. The most interesting and challenging part in this project was to use the Serializable interface. The testing part consisted in giving the programs as many inputs as possible in many different ways and giving it on purpose wrong inputs to see if the program will crash and if not what will happen then. The implementation of the graphic user interface is not so good because it has a lot of code and methods but because of the short time I had I could not divide it in multiple much smaller interfaces . And based on these tests made warning messages for every wrong input and stop the program from crashing. I hope I covered all the possible problems that can occur. For what I can tell so far there are no bugs or errors and the program behaves like it should.

1. **Results**

The result of this homework is a user friendly application acts like an online shop with a login page where you can login either as a customer or as an admin and to simulate the activities that the 2 users can perform. The graphic interface is a easy interface to work on and understand.

1. **Conclusions**

In conclusion this homework helped me to improve my thinking. I learned new things and was remained about things I forgot and discovered how to work with the Seializable interface, even if just a little. Also this homework helped to see that I need to learn a lot more about how to work with java or about the OOP principles and how to write a good code.

For example here are some changes that I would make in the project.

* + - * A more clean graphic user interface class
      * Dividing some classes (Gui for eaxample) in more simple and easy to understand classes
      * Adding new operations like having an account, adding money to that account and verifying if the cost of the order is smaller than the money in the account
      * Making a create user window where the user can create its own username and password
      * Verifying if the username is available

1. **Bibliography**

For this homework I used different to help me understand concepts that I did not yet understood or to help me create different diagrams.

Some of these sites are:

-<http://stackoverflow.com/>

-https://www.youtube.com/

And the lectures were also of huge help.