

Technical University of Cluj-Napoca

Computer Science and Automation Faculty

Computer Science Department

2nd semester 2015-2016

Programming Techniques Homework 2

Pascaru Cosmina –Roxana

Group: 3042

CONTENTS

• Assignment Objectives	3
• Problem Analysis. Modeling	4
• Usage scenarios	5
• Design	7
▪ UML Diagram	10
▪ UseCase Diagrams	11
▪ Sequence Diagrams	12
▪ Activity Diagram	14
▪ Classes	15
• Gui Class	15
• Person, Admin, Customer, ShopCustomers Classes .	18
• Order and OPDep Classes	19
• Product, ProductStock and Warehouse Classes	20
• StockTableModel Class	21
▪ Packages.	22
• Results.	24
• Conclusions. Further implementations	24
• Bibliograpfy	24

1 Assignment Objectives

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.

The application also provides a graphical user interface which allows different views and operations for regular customers and for the admin.

2 Problem Analysis. Modeling

The application simulates, in a way, working with a database of customers, products and orders, but instead of using an already implemented database, it uses serialization in order to keep track of new and old information.

The complexity of the assignment is solved by using various classes to implement the subjects of this task, such as: the customer, admin, order, product, product stock, etc.

All these classes would have to be serializable, i.e. implement the Serializable interface, so that they can be used to create a database.

Other classes are used to store groups of subjects. The class ShopCustomers is used to store all the customers of the shop. It also uses a BinarySearchTree. The classes OPDept and Warehouse are used to store orders and products. They also use BinarySearchTrees which facilitates a faster search for a specific object.

The class Order uses an array list of products to store the products of an order, as a customer can add more products to his/her order.

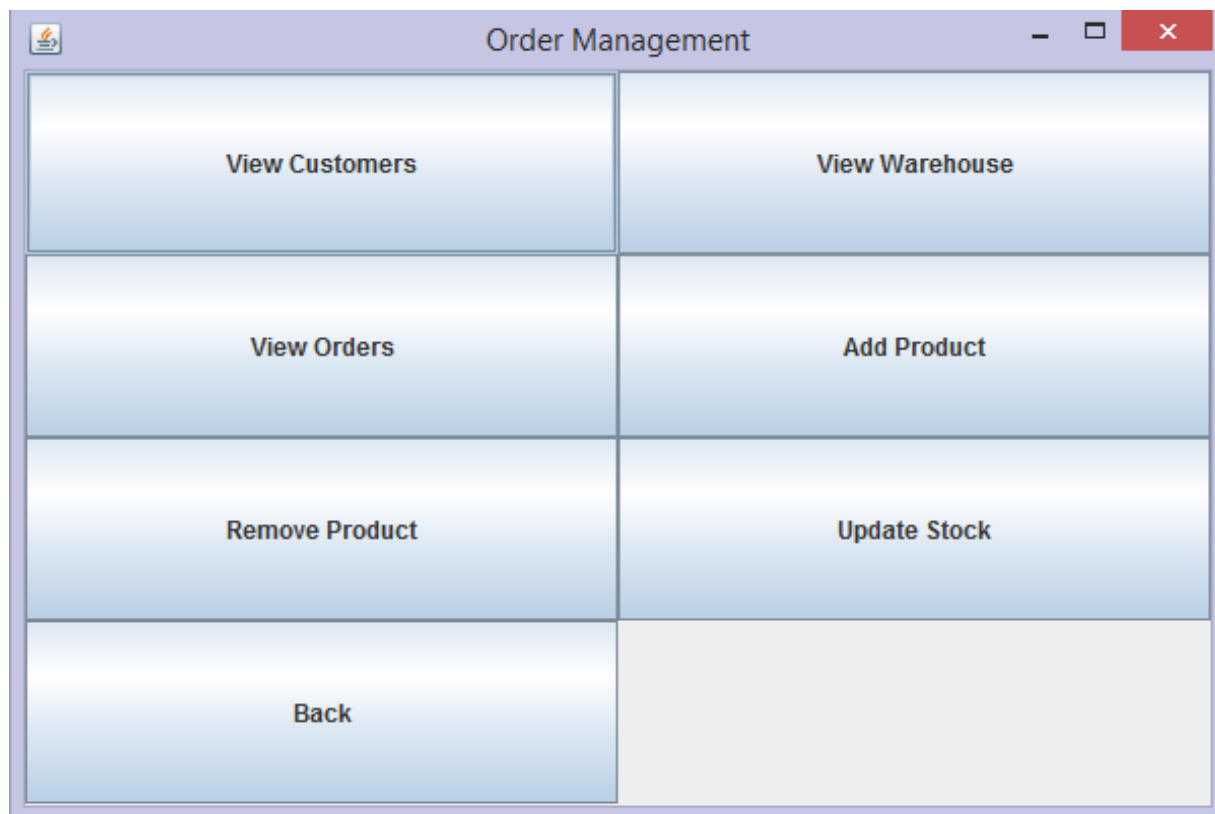
The graphical user interface provides two access modes: one for the regular customers and one for the admin. This means that each of them will see a different side of the application. The customers will only be able to see the available products in stock, add or remove products from their order and see their order and order history, if it exists. The admin will be able to see all the databases associated to the application (warehouse products, orders, customers) and also modify them. The admin can therefore add or remove products from the warehouse and update the stock.

At the start of the application, each customer must complete a profile before making orders (if they do not already have one). After they do this, they are automatically saved in the database.

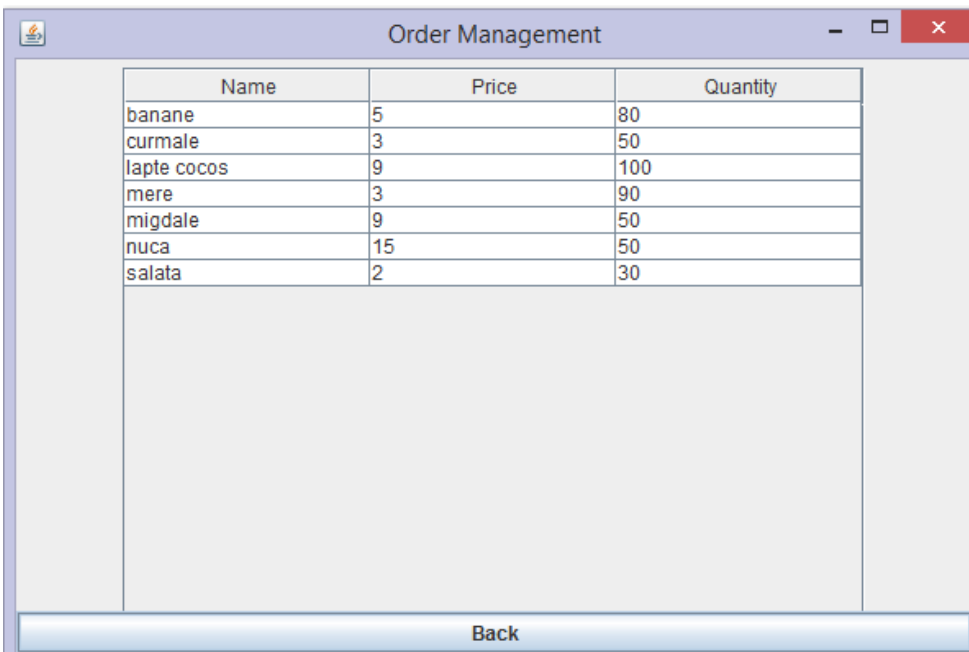
3 Usage scenarios

The application simulates an online shop which provides different use cases for customers and the admin.

At the start of the application the user is prompted with a login screen which asks for a username and a password. If they belong to the administrator of the shop, then a screen with the possible actions that the admin can perform appears. These actions include viewing the customers of the shop, the orders and the products in the warehouse, adding or removing products from the warehouse or updating the number of products in stock.



If the admin wants to add a product which already exists in the warehouse, or delete a product which does not exist in the warehouse, then a warning message will be prompted. The same happens when updating the stock;



The screenshot shows a window titled "Order Management" with a standard Windows-style title bar (minimize, maximize, close buttons). Inside the window is a table with three columns: "Name", "Price", and "Quantity". The table contains seven rows of data. Below the table is a large empty rectangular area, and at the bottom of the window is a button labeled "Back".

Name	Price	Quantity
banane	5	80
curmale	3	50
lapte cocos	9	100
mere	3	90
migdale	9	50
nuca	15	50
salata	2	30

If the username and password belong to an already registered customer, then he/she will be able to directly see the products in the warehouse, add or remove products from their order, see the products

from their last order, view their current order and the total price they have to pay and finally, confirm the order which will then be added to the database.

If the customer wants to remove a product which was no added to the order, a warning message is displayed.

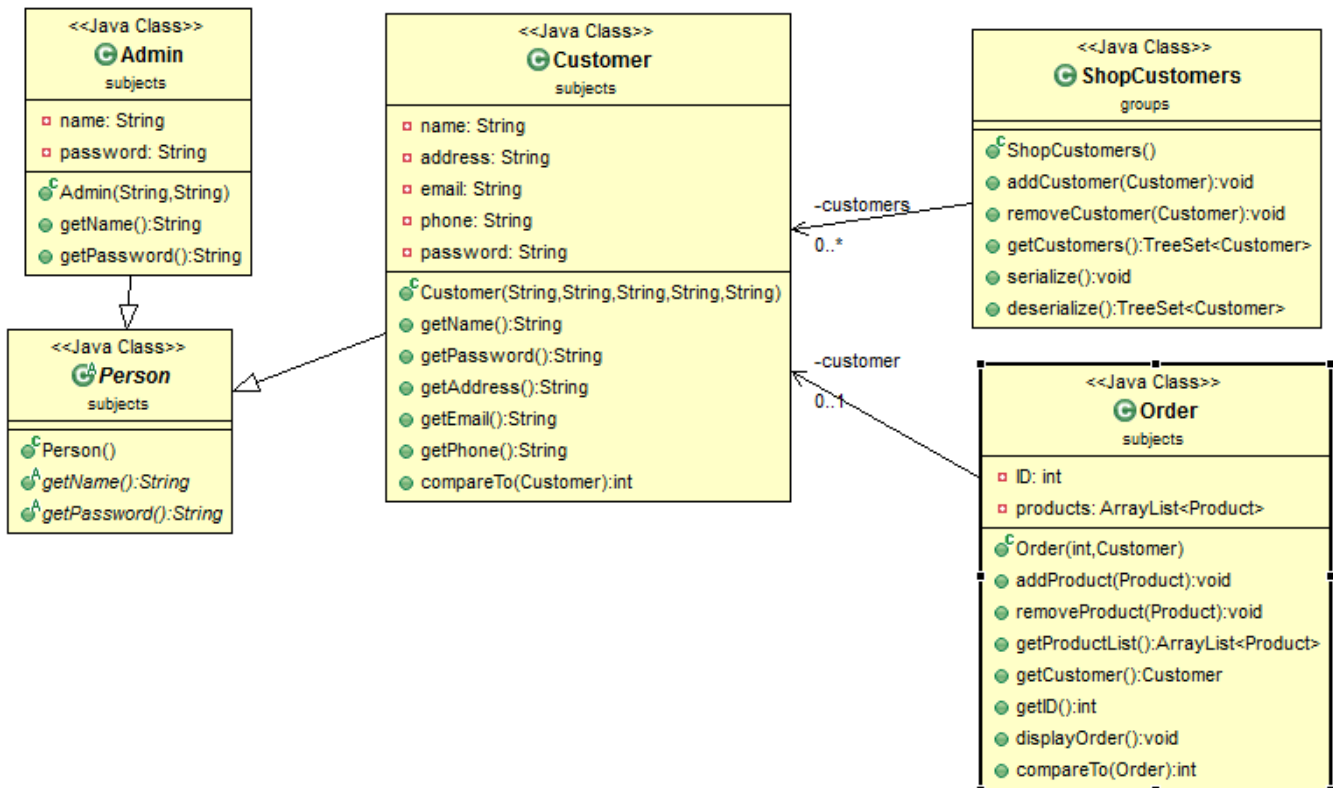
If the username and password belong to a new customer, then he/she will have to complete his/her profile by adding an address, an email, and a phone number. Then he/she will be able to place an order. If they want to view their order history, no history will appear.

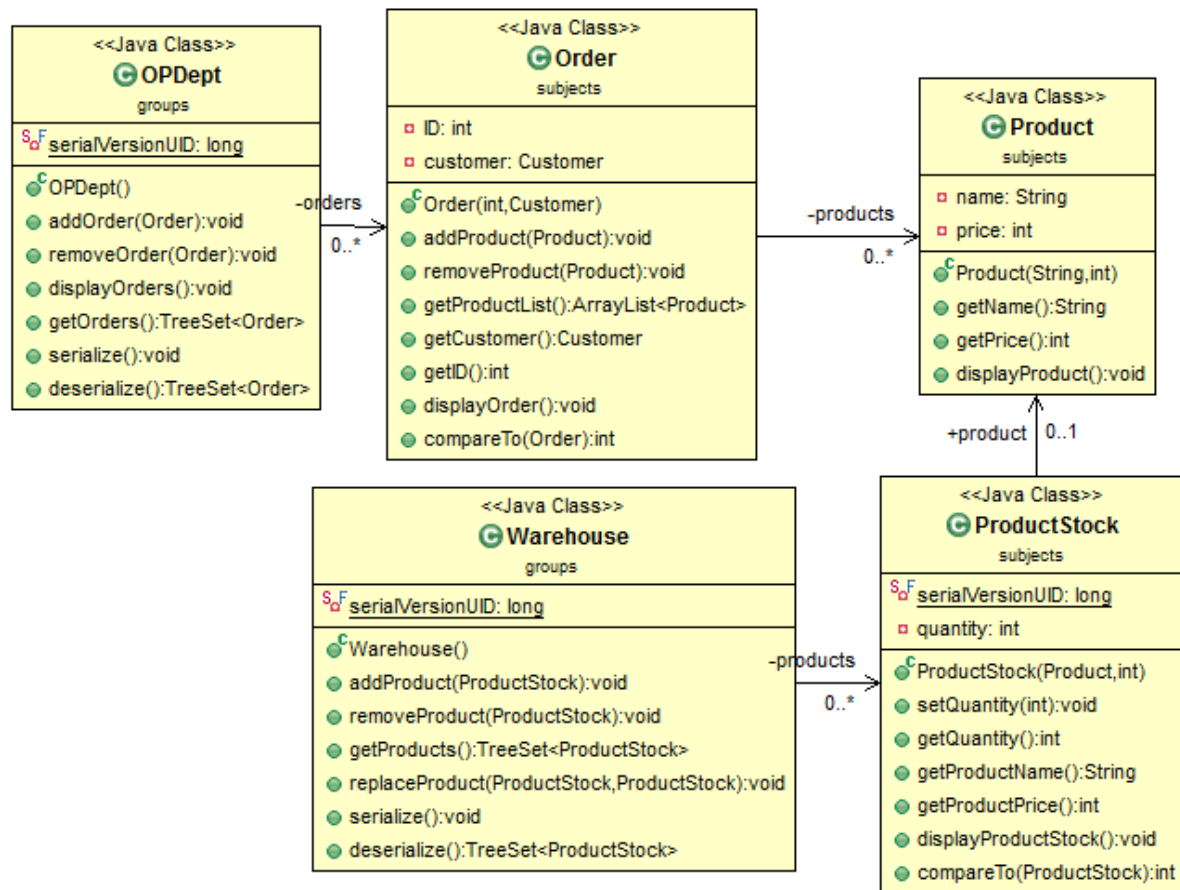
4 Design

The project is divided into 5 packages and 21 classes. The subjects package contains the classes representing the admin, customer, order, product and product stock.

The package groups is made of classes which contain sets of subjects. For example the class OPDept (order processing department) contains a TreeSet of orders. The class ShopCustomers contains a TreeSet of all the customers of the shop. The class warehouse keeps track of all the products in the warehouse. It also uses a TreeSet to store information.

The classes Admin and Customer extend the abstract class Person. The classes Order and ShopCustomers use the class Customer. Every order has a customer associated to it and a shop has many customers.

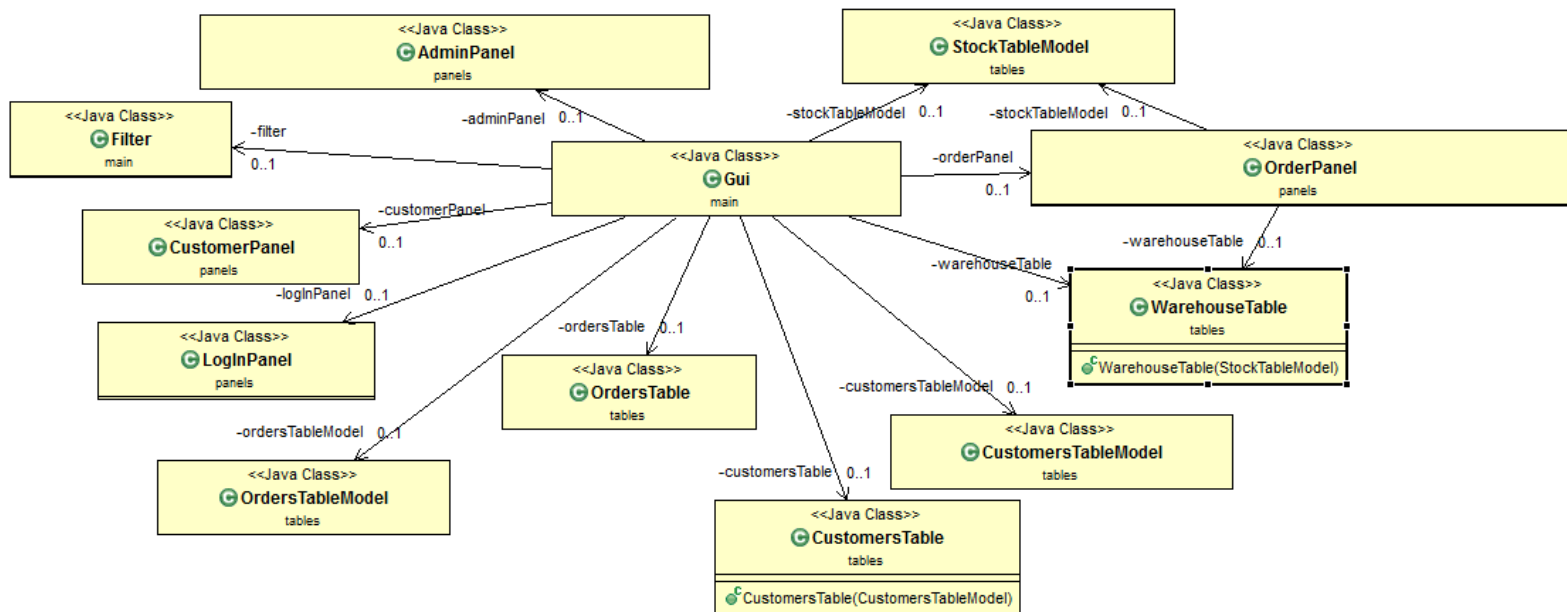




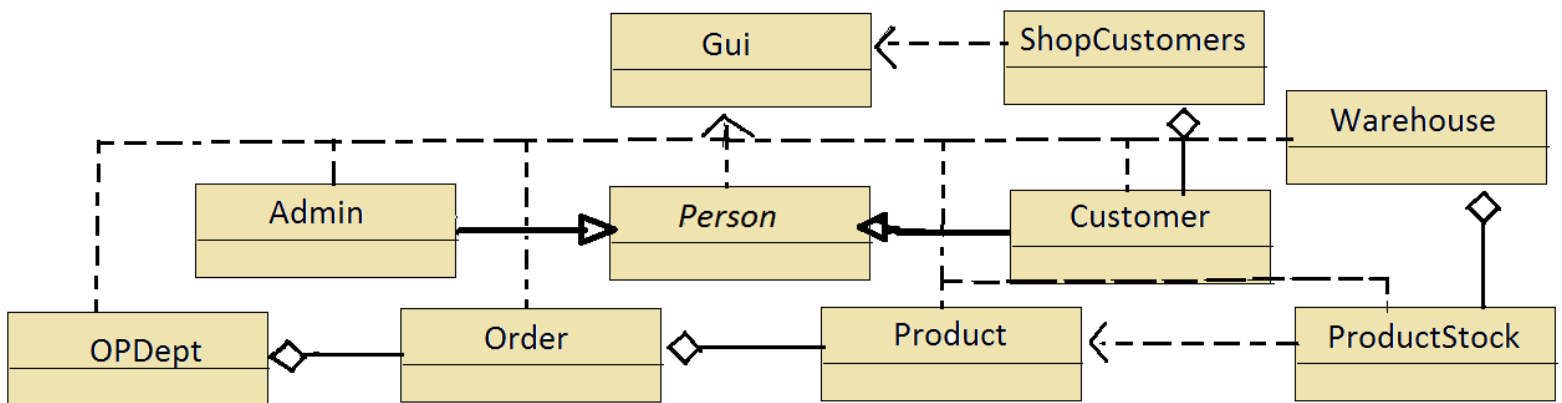
The class OPDept processes orders, which means it uses the class Order. An order will also have one or more products, so the class Order also uses the class Product. Product stock keeps track of the number of products in stock. A warehouse will host many product stocks which means the class Warehouse will use the class ProductStock.

The Gui class coordinates all the panels and tables in the project. It also contains the logic that makes the application work. It uses a different panel for the admin and for the regular customers. Each of them present different functionalities. The LoginPanel checks for the authentication of the admin or of already registered customers.

The tables created in the classes WarehouseTable, CustomersTable and OrdersTable are used to display the data saved by serialization. These classes use TableModels to add or remove data from the tables.

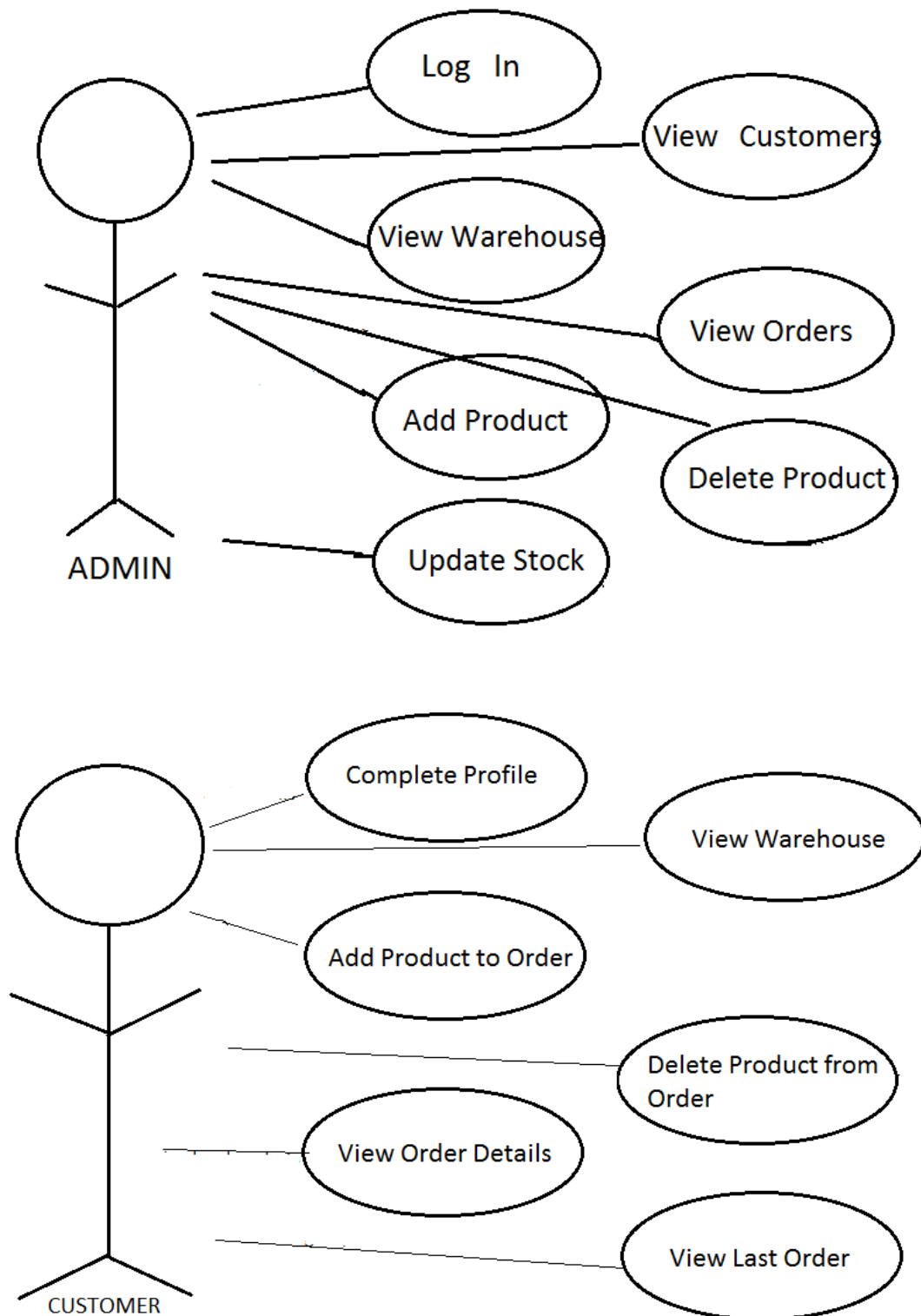


UML Diagram



The UML diagram shows the relationships between classes. The Gui class uses almost all the classes, including the classes designed for the graphical user interface. The classes Admin and Customer extend the abstract class Person. ShopCustomers uses a TreeSet of customers, so there is an aggregation relationship between the two classes. Similarly, the class Warehouse uses a TreeSet of ProductStock, the class Order uses an arraylist of Products and the class OPDept uses a TreeSet of orders.

4.1 UseCase Diagrams

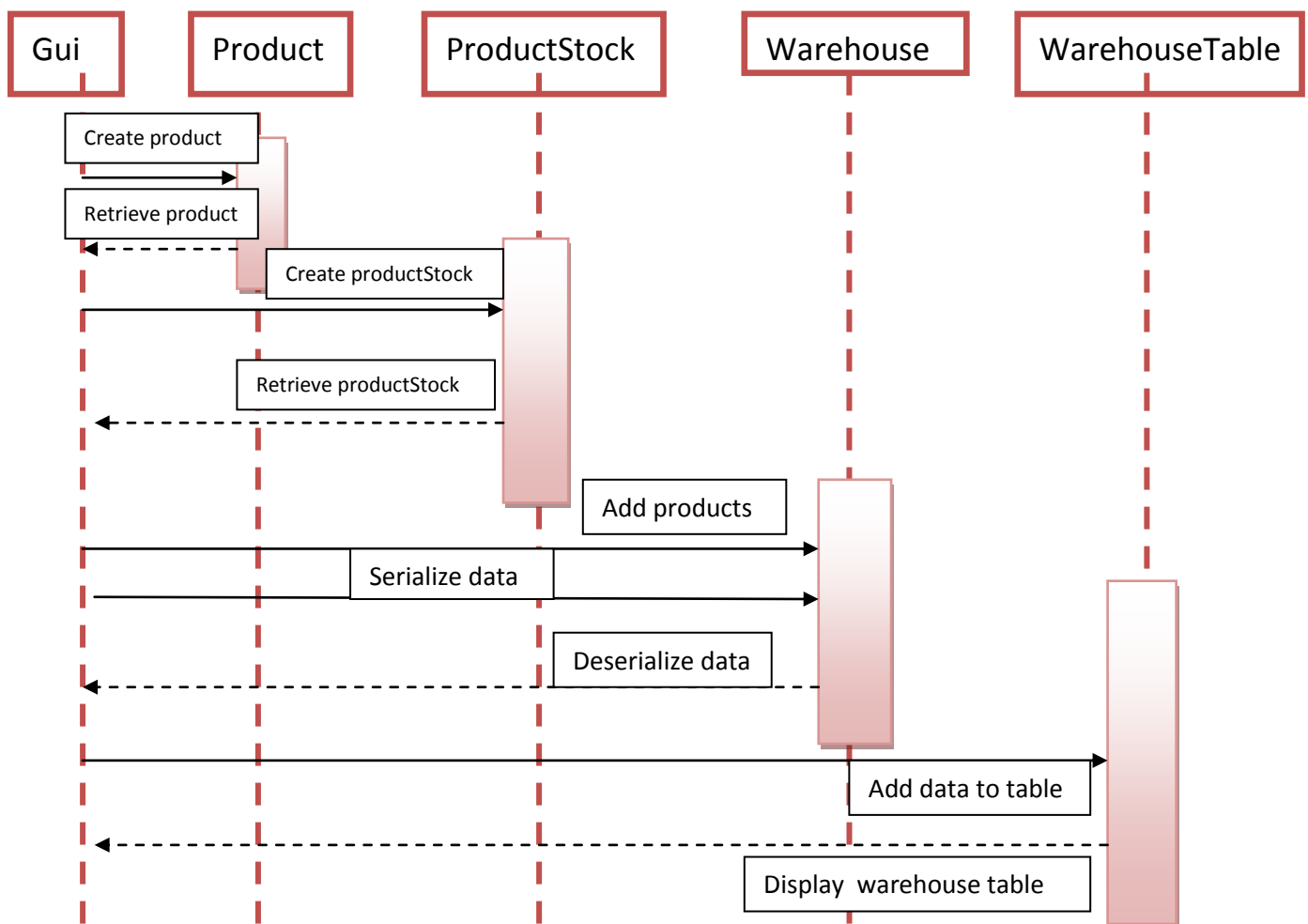


The first use case diagram describes the cases that appear when the user logs in as the administrator of the shop. A different panel from the one of the regular customer appears and the admin can perform different operations. He/She can view all the customers of the shop and their orders, all the products in the warehouse, add or delete products from the warehouse and update the stock of an already existent product.

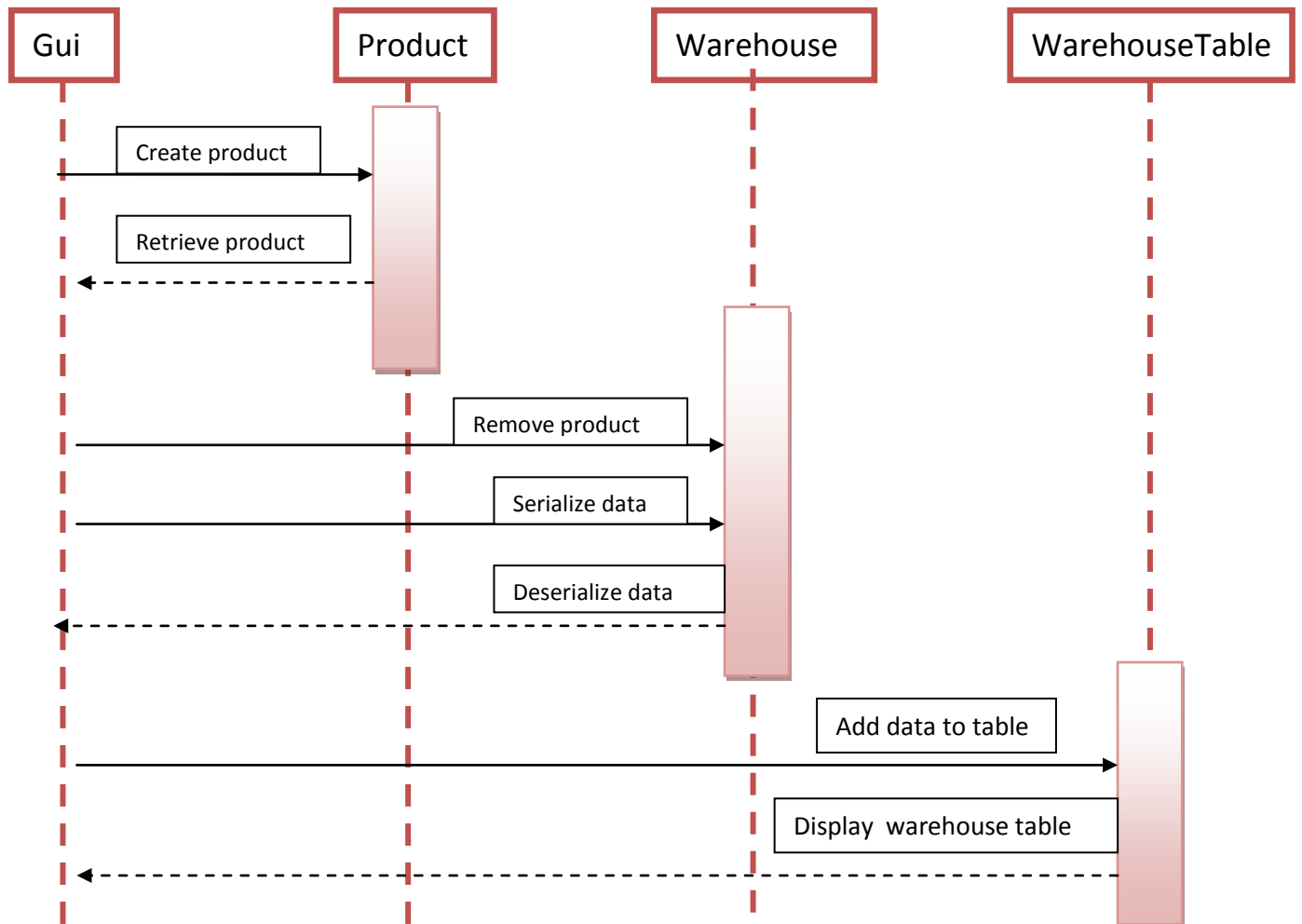
The second use case diagram describes the actions that a regular customer can perform. Unlike the admin, the customer cannot view the other clients or their orders, but he/she can view the products in the warehouse, add products to their order, remove products from their order, view the order and the total price they have to pay, and if they have placed an order before, they can view the products they ordered on their last order.

4.2 Sequence Diagram

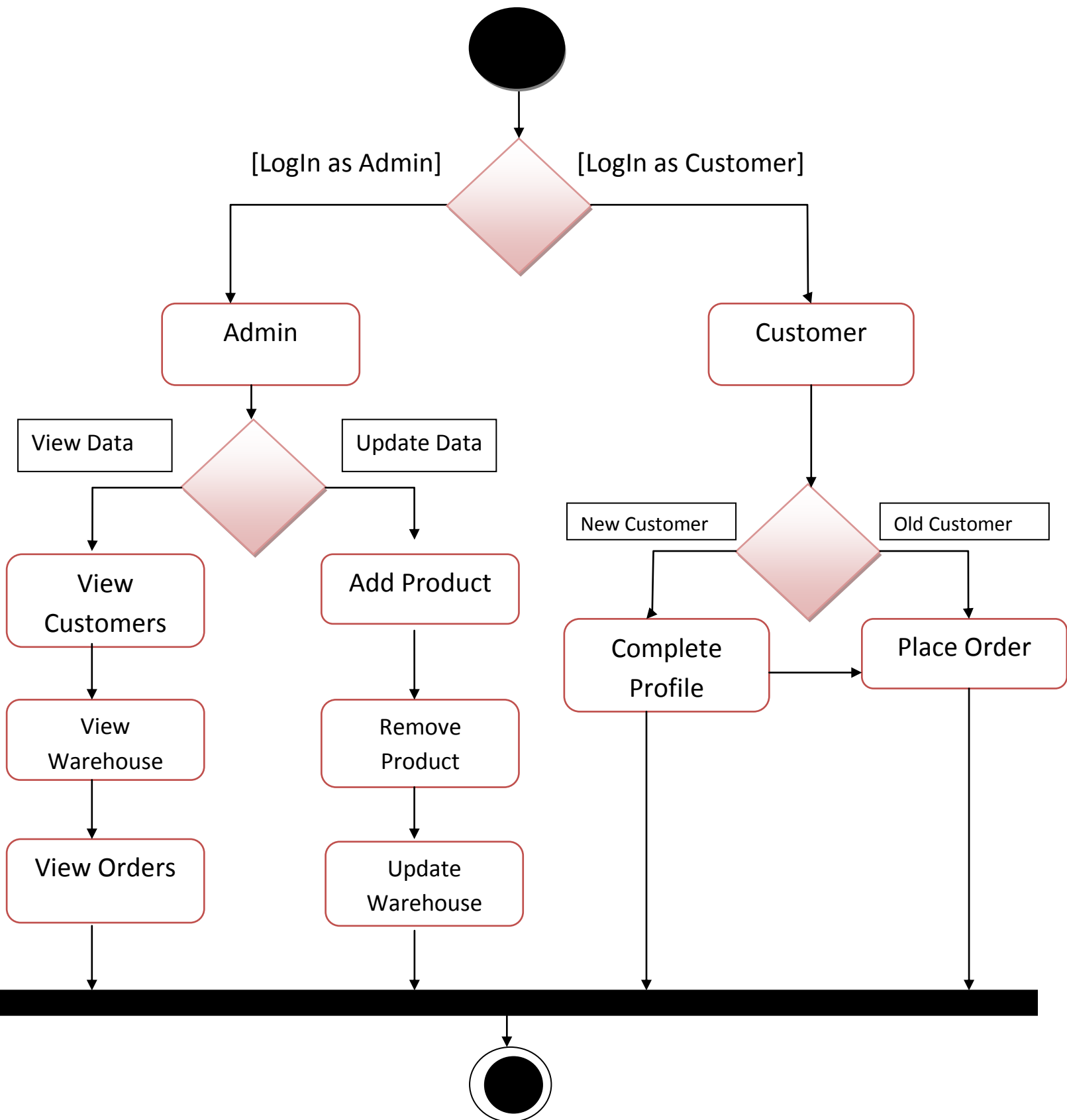
Adding products to the Warehouse



Removing products from the Warehouse

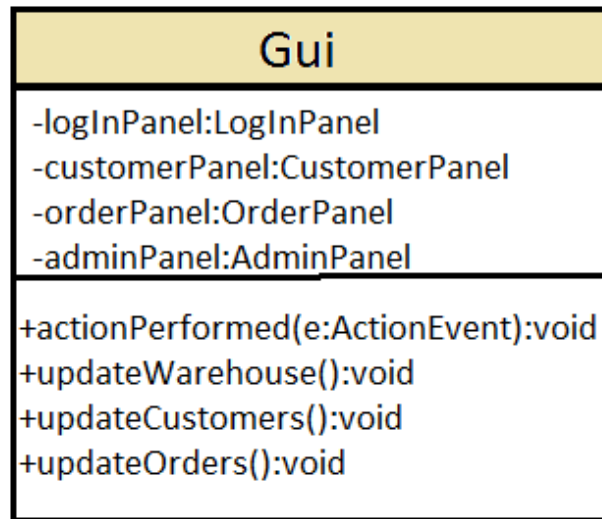


4.3 Activity Diagram



4.4 Class Diagrams

4.4.1 Gui Class



The Gui class is called from the main method. It brings together all the classes in the application and it uses them in the graphical user interface it creates. The Gui class contains a frame to which are added progressively the corresponding panels. The panels are defined as separate classes, but their functionalities are integrated in the Gui class. The Cui class sends the corresponding buttons to the constructors of the panel classes and checks for events in the method `actionPerformed` which is overridden from the `ActionListener` interface.

Attributes

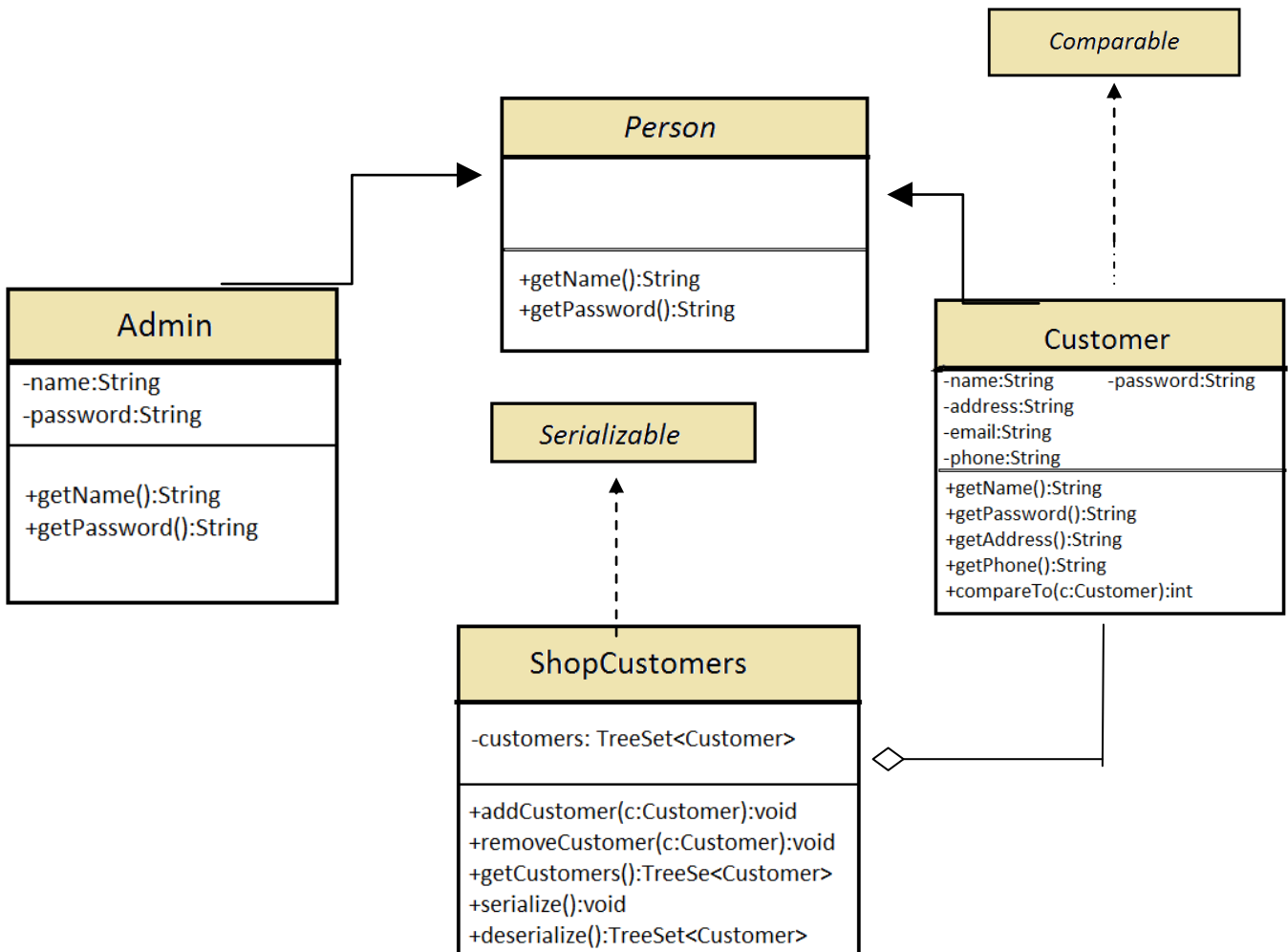
- login Button
 - it first checks if the entered username and password belong to the admin
 - If not, it checks to see if the entered customer already exists in the database
 - If not, it displays the details panel which will ask for the user's information
- save button
 - It saves the new customer into the database
- addProduct button
 - Prompts a `JOptionPane` message dialog for the user to enter the searched product; if it exists It is added to the order, if not, a message is displayed
- removeProduct button

- Prompts a JOptionPane message dialog for the user to enter the products they want to remove from the order; if the product is not in the order, a message is displayed, else the product is removed from the order
- viewOrder button
 - goes through all the products from the order, gets their name and price and then displays a JOptionPane message dialog containing the products from the order and the total price
- viewHistory button
 - if the customer has ordered from the shop before, then it displays a JOptionPane message dialog with the products they last bought and their total price
- confirm button
 - adds the new order to the existing TreeSet of orders from the OPDept class
- viewCustomers button
 - only available to the admin
 - it adds a table containing all the customers to the panel
- viewWarehouse button
 - only available to the admin
 - it adds a table containing all the products in the warehouse to the panel; the table contains all the information about each product
- viewOrders button
 - only available to the admin
 - it adds a table containing all the orders to the panel; the table displays the ID of the order and the customer that placed it
- addStock button
 - prompts a JOptionPane input dialog to get the name of the product the admin wants to enter
 - it then searches for that product in the warehouse; if it already exists, a message is displayed; else the admin has to enter the rest of the data for that product (category, price, quantity)
- removeStock button
 - prompts a JOptionPane input dialog to get the name of the product the admin wants to remove from the warehouse
 - it searches for the product in the warehouse; if it doesn't exist a message is displayed; if it does, it is removed
- updateStock button
 - the user enters the product to be updated; if it doesn't exist in the warehouse a message is displayed; if it does, then a JOptionPane input dialog is prompted asking for the admin to enter the new quantity

Methods

- `updateWarehouse()`
 - deletes all data from the `stockTableModel`
 - deserializes the warehouse
 - adds the new products from the deserialized data to the table model
 - creates a new table with the products from the warehouse
- `updateCustomers()`
 - works similarly to the `updateWarehouse` method
 - creates a new table of customers from the deserialized data stored in `ShopCustomers`
- `updateOrders()`
 - creates a new table of orders from the deserialized data stored in `OPDept`

4.5.2 Person, Admin, Customer and ShopCustomers Classes

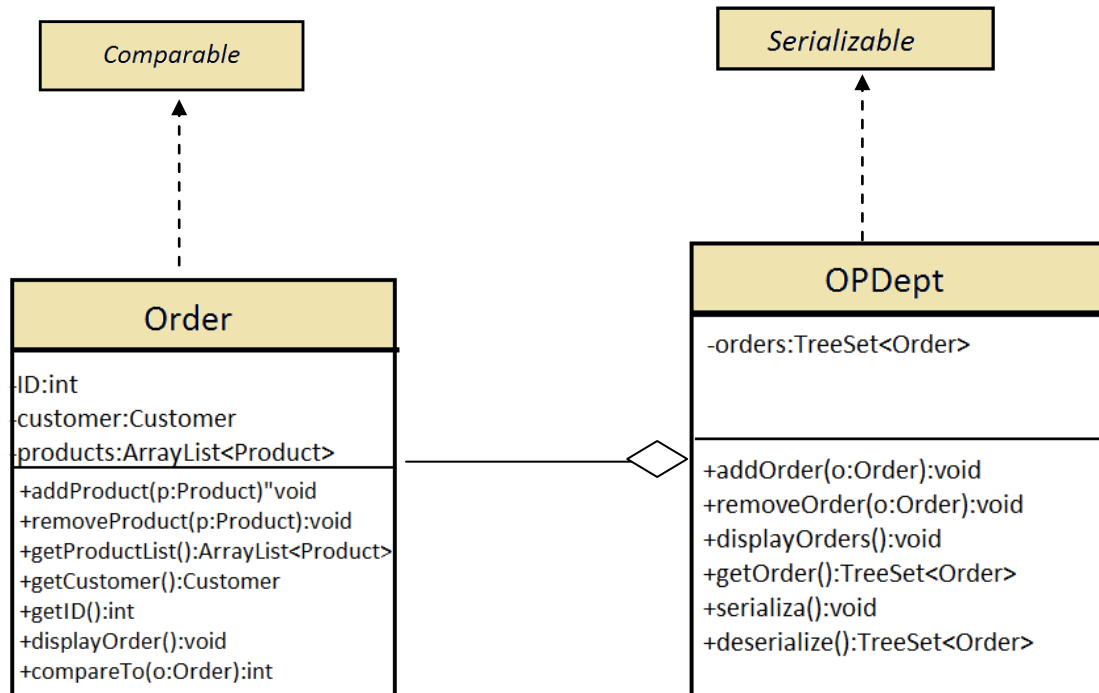


The classes Admin and Customer extend the abstract class Person which has the abstract methods getName() and getPassword(); they are overridden in the Admin and Customer classes.

The class ShopCustomers contains a TreeSet of customers. It contains methods to add and remove customers and also to serialize the data stored in the TreeSet.

The class Customer implements the Comparable interface and it overrides the method compareTo() that the interface has. This method is used for ordering the customers in alphabetical order in the TreeSet form the ShopCustomers class.

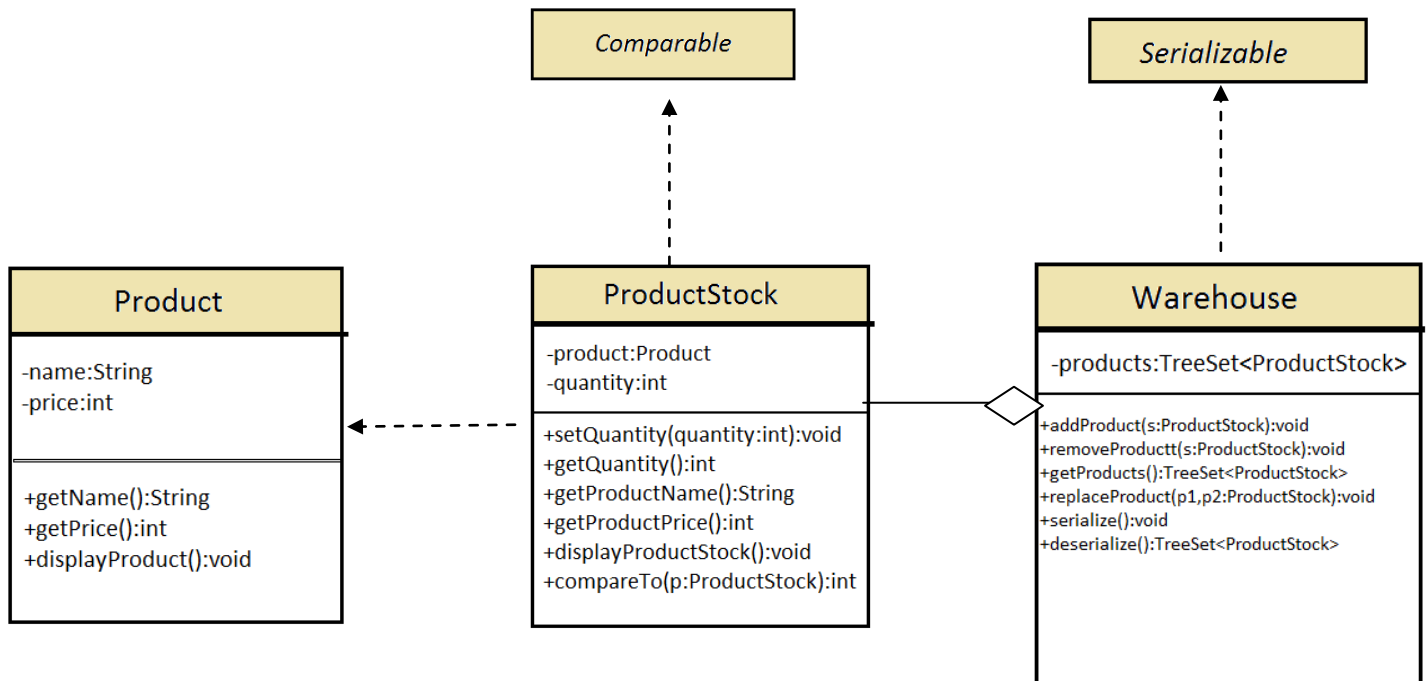
4.5.3 Order and OPDept Classes



The class `Order` implements the `Comparable` interface and overrides the method `compareTo()`. It uses the class `Customer` and `Product` as it stores an `ArrayList` of `Products`. The class `Order` has methods for adding or removing products from the `ArrayList`, as well as for getting the customer and the ID of the order.

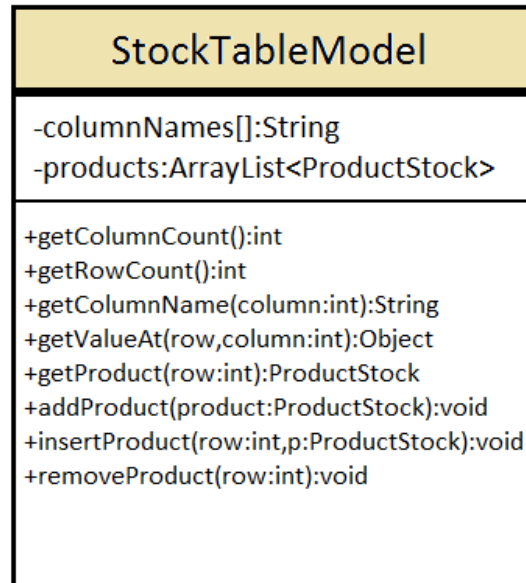
The class `OPDept` uses a `TreeSet` of `Orders`. The orders are stored in alphabetical order. The comparison is done by the `compareTo()` method declared in the class `Order`. The class `OPDept` has methods for adding and removing orders from the `TreeSet`. It implements the `Serializable` interface. The `serialize()` and `deserialize()` methods are used for storing the `TreeSet` of orders in a separate file so the data can be saved even after the application was closed.

4.5.4 Product, ProductStock and Warehouse Classes



The class ProductStock has 2 attributes: product and quantity. It is used to store the stock quantity of each existent product. It has the compareTo() method which allows for the products to be sorted in alphabetical order. It implements the Comparable interface. The class Warehouse implements the Serializable interface. It uses a TreeSet of ProductStock to store all the products in the warehouse. It has the methods serialize() and deserialize() which are used to save the TreeSet in a separate file. The method replaceProduct() is used for the stock update operation performed by the admin.

4.5.5 StockTableModel Classe



The StockTableModel class is used to create the data that will later be added to the class WarehouseTable which creates a table with all the products from the warehouse.

It has methods for adding or removing products from the table, as well as for getting the total number of rows (i.e. products in stock) from the table.

4.5.6 Packages

The project is composed of 5 packages: main, subjects, groups, panels and tables.

The main package contains:

- MainClass Class
 - The class which contains the main method; the class where the application starts
- Gui Class
 - Is the class which creates the graphical user interface
 - It uses different swing and awt components
- Filter Class
 - Contains methods used to search for objects in collections
 - The method `filterProductStock()` searches for a given ProductStock in the Warehouse
 - It returns a Boolean values: true if the product was found and false if not
 - The method `filterOrder()` searches for a given product in the list of the Order class
 - It return true if the order was found and false otherwise
 - The method `getFoundProductStock()` returns the ProductStock that was found by the `filterProductStock()` method
 - The method `getFoundProduct()` return the product that was found by the `filterOrder()` method

The subjects package contains:

- Admin Class
- Customer Class
- Order Class
- Person Abstract Class
- Product Class
- ProductStock Class

The groups package contains the classes that have ArrayLists or TreeSets of a class from the subjects package. The package contains:

OPDept Class

- Has a TreeSet of orders
- Implements the serializable

- ShopCustomers Class
 - Contains a TreeSet of customers
 - It implements the Serializable interface
- Warehouse Class
 - Contains a TreeSet of ProductStock
 - It implements the Serializable interface

The panels package contains all the classes that extends JPanel. These classes are used by the Gui class to create the graphical user interface. The package contains:

- AdminPanel Class
 - Creates the panel which is visible when the user that logs in is the admin of the shop
- CustomerPanel Class
 - It creates the panel that is visible when a new customer logs in
 - It is the panel that will help the customer complete their profile
- LoginPanel Class
 - It creates the first visible panel that appears when the application is run
 - It gets the username and the password of the customer
- OrderPanel Class
 - It creates the panel that is visible after the customer completed his/her profile
 - It offers the customer the possibility to add or remove products from his/her order, to view the order and the total price or to see the history of their orders

The tables package contains the classes that create the tables for the warehouse, customers and orders. It contains both the tabs and the table models:

- CustomersTable Class
- CustomersTableModel Class
- OrdersTable Class
- OrdersTableModel Class
- StockTableModel Class
- WarehouseTable Class

Imported packages:

- java.awt.Color;
- java.awt.Dimension;
- java.awt.Font;
- java.awt.GridLayout;
- java.awt.event.ActionEvent;
- java.awt.event.ActionListener;
- javax.swing.JButton;
- javax.swing.JFrame;
- javax.swing.JLabel;
- javax.swing.JOptionPane;
- javax.swing.JPanel;
- javax.swing.JTextArea;
- javax.swing.JTextField;

5 Results

The application is user friendly, it is easy to use by both the customers and the admin. It is not complex, so it doesn't require any special knowledge. The graphical user interface resembles one suitable for mobile applications.

The application can be successfully used as a managing tool for the warehouse or the products available.

6 Conclusions. Further implementations

In conclusion, the present state of the application is functional and ready to use, but it can easily be upgraded to include even more functionalities, or to have a more appealing aspect. Instead of storing data in files, a database could be used. This would offer even more flexibility and it could help the application to extend and manage even larger amounts of data.

7 Bibliography

- [1] https://en.wikipedia.org/wiki/Activity_diagram
- [2] <http://usna86-techbits.blogspot.ro/2012/11/uml-class-diagram-relationships.html>
- [3] <https://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html>
- [4] <https://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html>
- [5] <http://www.javapractices.com/topic/TopicAction.do?Id=45>
- [6] <http://stackoverflow.com/questions/21626439/how-to-implement-the-java-comparable-interface>
- [7] <http://stackoverflow.com/questions/21626439/how-to-implement-the-java-comparable-interface>