Programming Techniques
HOMEWORK 3

Student: Tudorica Andrei
Group: 30421

0. Contents

1. Objective of the homework

Consider an order management application for processing customer orders for a warehouse. Relational databases are used to store the products, the clients and the orders. Furthermore, the application uses (minimally) the following classes:

1. Domain specific classes: Order, Customer and Product
2. Business Logic (warehouse-specific processing) classes: OrderProcessing, WarehouseAdministration, ClientAdministration
3. Presentation classes: GUI related classes
4. Data access classes: Database access related classes Other classes and packages can be added to implement the full functionality of the application.

Requirements

a. Analyze the proposed application, determine the structure and behavior of its classes and draw an extended UML class diagram.

b. Design, implement and test the application classes. Use javadoc for documenting classes.

c. Define, design and implement a system of utility programs (examples: reports for under-stock, totals, filters, etc.).

d. Design and implement a comprehensive demo driver for the order management application.

2. Problem Analysis

Now a days, as storages get bigger and bigger and harder to manage and organize, store management systems become vital in organizing depots and products in these depots. This project includes projecting a system to help store managers to have easier access to their data, manage this data, by adding customers, editing customers, deleting customers, or printing a receipt for a certain customer,having a list of customers that dynamically updates by the changes done by the user. Adding a product, removing a product and editing a product, by having a list of products that dynamically updates by the changes made by the manager. The app also allows the manager to add or delete orders using a list of orders also dynamically updated.

3. Modelling

The process of modelling is defined as the process of modularising a big problem into smaller ones, that are easier to understand and debug. This also helps making an abstract idea more clear. In software development, modelling is essential in order to build an application that has a strong background structure.

Our problem, as stated above in the “Problem Statement” paragraph, includes a storage management system, for storage managers. For the minimal case I developed here, we only use 3 types of elements : customers, orders, and products. The orders table behave like a many to many table between the customer table and the product table. This application works on a database containing the table system i just described, that is why, in a data first approach of development, we need first need to design the database structure. I use, in this case, 3 tables, previously described : customer, order and product, each of them having columns that will be described in what follows. After the database design phase, comes the class design phase, where i used a class mapping that looks like the database. To make a binding between this two we, need a DAO layer. DAO stands for data access object, and it is an object that gives an abstract interface to a database, in our case, but it can be applied to other kinds of persistence mechanisms. The DAO provides some specific operations on the database, but it doesn’t expose other information on the database. In the dao layer i used a couple of classes that extend my AbstractDAO class. The abstract DAO class is developed using reflection.
Reflection is a Java programming language specific feature that gives a Java application the possibility to introspect upon itself, to examine itself, and manage its internal features and elements. For example a Java application can obtain names for all its members and methods, or it can also make code injections. This is a feature other programming languages like c, c plus plus, python, et cetera don’t have.

4. Use cases

The developed software application is a basic one, therefore it only simulates the queues, and optimizes them in a greedy way. The app is useful though for organizing simple storages, like, for example, the ones of a supermarket, giving the manager the possibility to easily add, edit or delete elements in the tables he needs to.

This app could be useful , for instance , in a supermarket that makes deliveries. It would have the table of clients that buy stuff from the online shop. Also there is the table showing the elements the supermarket has in store at a certain moment. Clients can place orders, choosing what product they want to buy, and generating the receipt when their shopping session is over.

4.1. Scenarios

Let's consider an example of how the application works;

IMPORTANT NOTES:

- The ID input text boxes may only contain integer numbers.
- The Quantity input text boxes may only contain integer numbers.
- The Price input text boxes may only contain double numbers.
- The other input text boxes may contain integers and strings. Anything that is introduced in these boxes will be used as a name.
- The application does not allow new orders containing products for which the quantity is 0.
- All the changes made to the tables is instantly updated in the database too.

Lets consider a scenario where a user wants to add a client, add two products and place an order that the client orders 2 or each products and generates a receipt.
Steps:

- The user successfully launches the application.
- In the Customers tab he enters the name of the client he wants to add, in the first text box in the panel.
- In the Customers tab he enters the surname of the client he wants to add, in the second text box in the panel.
- He presses the button which has " Add Customer " written on it.
- The client is added in the list below.
- The user keeps in mind the id of that client.
- The user goes to the Product panel.
- In the Product tab he enters the name of the first product he wants to add, in the first text box in the panel.
- In the Product tab he enters the quantity of the first product he wants to add, in the second text box in the panel, respecting the notes given in the " IMPORTANT NOTES " section .
- In the Product tab he enters the price of the first product he wants to add, in the third text box in the panel, respecting the notes given in the " IMPORTANT NOTES " section .
- He presses the button which has " Add Product " written on it.
- The first product will be added to the list below.
- In the Product tab he enters the name of the second product he wants to add, in the first text box in the panel.
- In the Product tab he enters the quantity of the second product he wants to add, in the second text box in the panel, respecting the notes given in the " IMPORTANT NOTES " section .

- In the Product tab he enters the price of the second product he wants to add, in the third text box in the panel, respecting the notes given in the “ IMPORTANT NOTES ” section .
- He presses the button which has “ Add Product ” written on it.
- The second product will be added to the list below.
- The user also keeps in mind the ids of these two products, from the table.
- The user goes to the orders tab.
- The user adds the id of the customer in the first box.
- The user adds the id of the first product in the second box.
- The user clicks the “ Add Order ” button.
- The order will be added in the table below.
- To add the second instance of this product, the user needs to pres the same button one more time.
- The order will be added in the table below.
- The user changes the id in the second text box with the id of the second product.
- The user clicks “ Add order ”.
- The order will be added in the table below.
- The user clicks “ Add order ” again.
- The order will be added in the table below.
- The user goes to the Customer panel, introduces the id of the customer in the text box besides the button with “ Generate receipt ” written on it.
- The user clicks the “ Generate Receipt ” button.
- The receipt is created.
- The user exits the application.

5. Design

5.0. The default package

The default package contains the graphic user interface class and a table model class. The graphic user interface class contains the description of the G U I. That is:

- A JFrame for the entire app.
- A tabbed pane, for switching between customers , products and orders.
- 3 panels for each section : customers , products and orders.

- The customer's panel contains 4 other panels and the table of contents of the customers table: add customer panel, edit customer panel, delete customer panel and generate receipt; adding a customer requires a name and a surname for that client. Editing a client requires the ID of the customer to be edited along side with the new name and new surname. For deleting a customer we need to input its id. sam e goes for generating receipts, we need to input the id.
- The products panel contains 3 other panels and the table of contents of the products table: add product panel, edit product panel and delete product panel; adding a product requires a name, a quantity and a price for that product. Editing a product requires the ID of the product to be edited along side with the new name, new quantity and new price. For deleting a product we need to input its id.
- The orders panel contains 2 other panels and the table of contents of the orders table: add order and delete order. In order to add a new order we need to know the id of the customer buying and the id of the product being bought. One order is a pair between a client and a product taking out a quantity of 1. If one wants to buy more instances of the same product he needs to place more orders on the same product. When the receipt for a certain customer is generated all the orders are summed up to total the price.
- All the 3 tables, the one for customers, the one for orders and the one for products are created using a table model. The table model class contains an empty constructor and a method called createModel that receives as an argument an arrayList of generic class Type T. This class is called using an array list of customers for the customers table, using an array list of orders for the orders table and using an array list of products for the products table. The method takes the fields of the class of the array list elements and generates the model of the table using these fields. This method uses reflection in order to get the fields of the class.

5.1. The Model package

The model package contains the definition of the classes used for data processing that is the customer class, the order class and product class.

- The customers class has 4 fields: the id of the customer, the name of the customer, the surname of the customer and the is deleted fields. It also contains 2 constructors, an empty one and une with all the fields. It also contains getters and setters for each variable in the class.
- The orders class has 2 fields: the id of the order, the id of the customer that is buying, the id of the product that is being bought and the is deleted fields. It

also contains 2 constructors, an empty one and une with all the fields. It also contains getters and setters for each variable in the class.

- The product class has 4 fields: the id of the product, the name of the product, the quantity of the product, the price of the product and the is deleted fields. It also contains 2 constructors, an empty one and une with all the fields. It also contains getters and setters for each variable in the class.

5.2. The bussinessLayer package

The business layer contains only one class, the manager, which is the binding between the other 2 layers of the application the graphical user interface and the data base. The data layer in our case is represented by the DAO classes. This class contains 3 array list:the customers array list, the orders array list and the products array list. It contains a constructor with no arguments, that initializes the 3 array lists, the customers array list, the orders array list and the products array list by calling the find all function in the Customers DAO for the customers list, the Order DAO for the orders list and the Products DAO for the products list.
The manager class contains methods for Adding elements to each type of list by adding the objects to the local array list and also adding them to the database by calling the insert object method in the Customers DAO for the customers list, the Order DAO for the orders list and the Products DAO for the products list.
The manager class also contains methods for editing elements in each type of list by getting the id as a parameter, searching the element in the arrays by id, editing the objects to the local array list and also editing them in the database by calling the update object method in the Customers DAO for the customers list, the Order DAO for the orders list and the Products DAO for the products list.
The manager class also contains methods for deleting elements from each type of list by getting the id as a parameter, searching the element in the arrays by id, editing the is deleted property of the elements the objects to the local array list and also editing them in the database by calling the update object method in the Customers DAO for the customers list, the Order DAO for the orders list and the Products DAO for the products list.
The last but not least, the manager class contains the class for generating the receipt for a certain customer. It gets the id of the customer, looks for the orders that have the customer id equal to the current customer. For each order like this it takes the product id and sums up the price of these products. The method prints the data to a file with the name “ receipt for XXX YYY ” where XXX is the name of the customer and YYY is the surname of the customer;

5.3. The dataAccessLayer package

The data access layer consists of the DAO files. DAO stands for data access object, and it is an object that gives an abstract interface to a database, in our case, but it can be applied to other kinds of persistence mechanisms. The DAO provides some specific operations on the database, but it doesn't expose other information on the database. In the data access layer I used a couple of classes that extend my AbstractDAO class. Each class in the model has a special DAO class that extends Abstract DAO. The abstract DAO implements generically the insert update and add method getting elements of generic type T. The abstract DAO class is developed using reflection. Every other dao calls for the super class when it tries to add update.

6. Results

The result of the development of this application, that lasted for two weeks together with writing the documentation, is a software easy to use that makes it easy for the user

7. Further development

- Adding different user levels on the application.
- Adding more details on customers.
- Adding more details on products;
- Displaying customer details and product details in the order page.
- Adding the possibility to add more products in the same order.

8.Conclusions

During the development of this project i learned and improved of several skills:

- A better use of Eclipse
- A better use of the Java coding language
- A better understanding and usage of the object oriented programming paradigms
- More organized and clear code

9. Bibliography

1. Java general documentation https://docs.oracle.com/javase/7/docs/api/
2. Lecture and laboratory guidelines http://www.coned.utcluj.ro/~salomie/PT_Lic/3_Lab/

3. Word counter https://wordcounter.net/