Technical University of Cluj-Napoca
Faculty of Automation and Computer Science

Programming Techniques

Homework Assignment 3

Warehouse simulation

Student: Crisan Adrian Gabriel
Group: 30433

Table of contents

1 . Problem specification

2 . Problem analysis, modelling, scenarios, use-cases

    2.1 Problem analysis, modelling

    2.2 Scenarios, use cases

3 . Design

    3.1 UML Class Diagram

    3.2 Package Diagram

    3.3 Database Diagram

    3.4 Classes design

    3.5 User Interface

4. Using and testing the application

5. Conclusions

1. Problem specification

Consider an application OrderManagement 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:

- Model classes -represent the data models of the application (for example Order, Customer, Product)
- Business Logic classes -contain the application logic (for example OrderProcessing, WarehouseAdmin, ClientAdmin)
- Presentation classes –classes that contain the graphical user interface
- Data access classes -classes that contain the access to the database Other classes and packages can be added to implement the full functionality of the application.

2. Problem analysis, modelling, scenarios, use cases

2.1 Problem analysis, modelling

For this application I used the paradigms of Object Oriented Programming in Java. Object-oriented programming (OOP) refers to a type of computer programming (software design) in which programmers define not only the data type of a data structure, but also the types of operations (functions) that can be applied to the data structure. In this way, the data structure becomes an object that includes both data and functions. In addition, programmers can create relationships between one object and another. For example, objects can inherit characteristics from other objects. The main concepts of Object Oriented Programming are Encapsulation, Abstraction, Inheritance and Polymorphism.

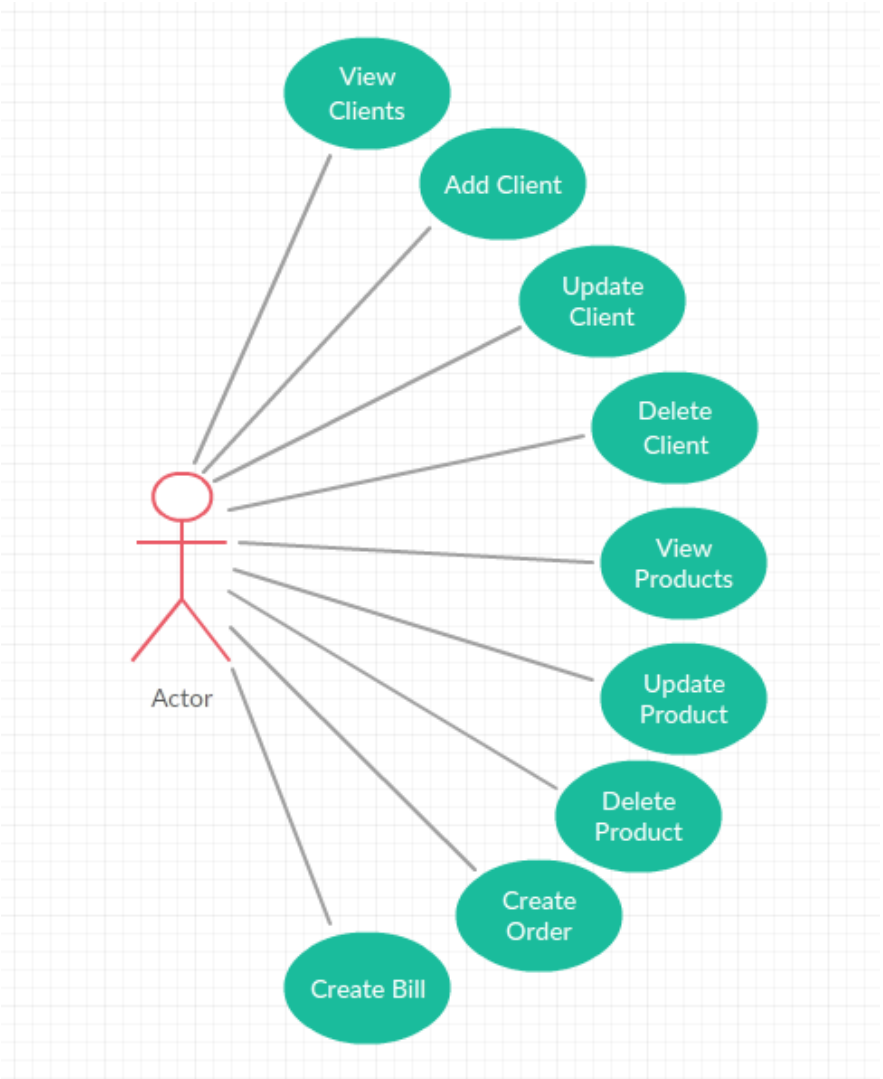
Inheritance provides a powerful and natural mechanism for organizing and structuring your software. Classes inherit state and behavior from their superclasses, and you can derive one class from another using the simple syntax provided by the Java programming language.

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class.

2.2 Scenarios, use cases

Use case diagrams are usually referred to as behavior diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors). Each use case should provide some observable and valuable result to the actors or other stakeholders of the system.



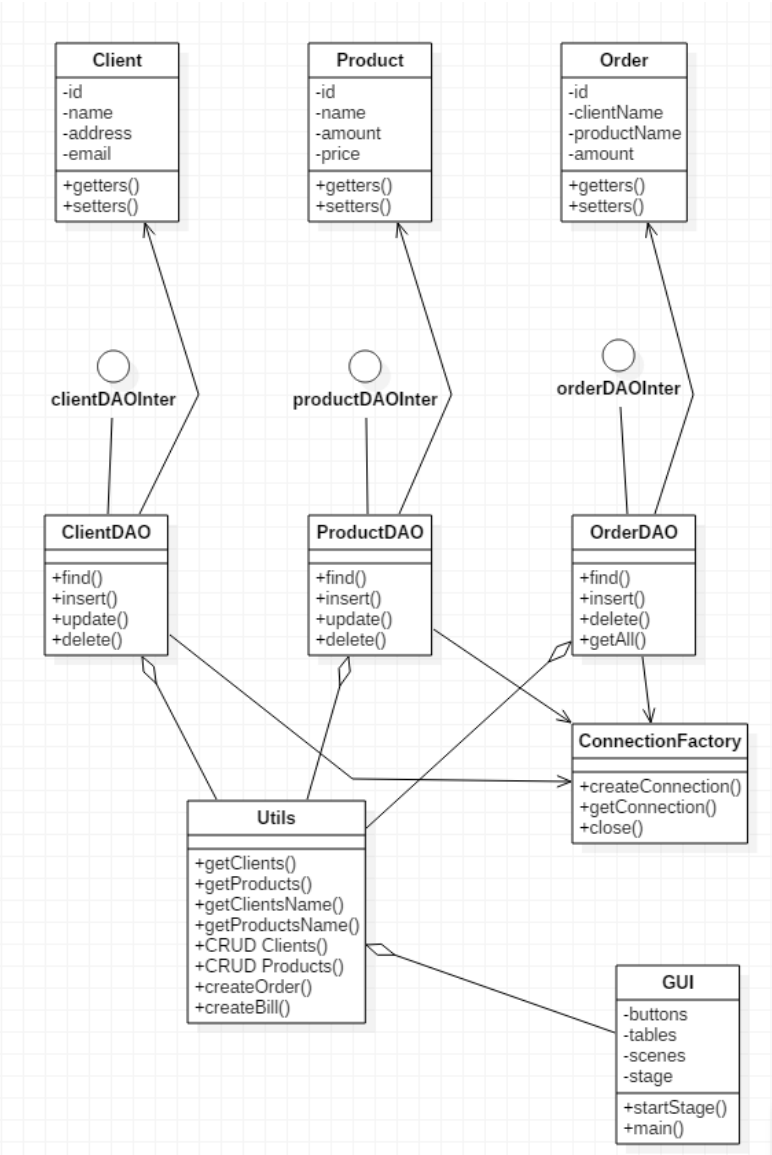
The program will have a basic input interface with buttons, textboxes, combo boxes and table views. One must provide a valid input, so there will be no non-positive values in the textboxes.

3. Design

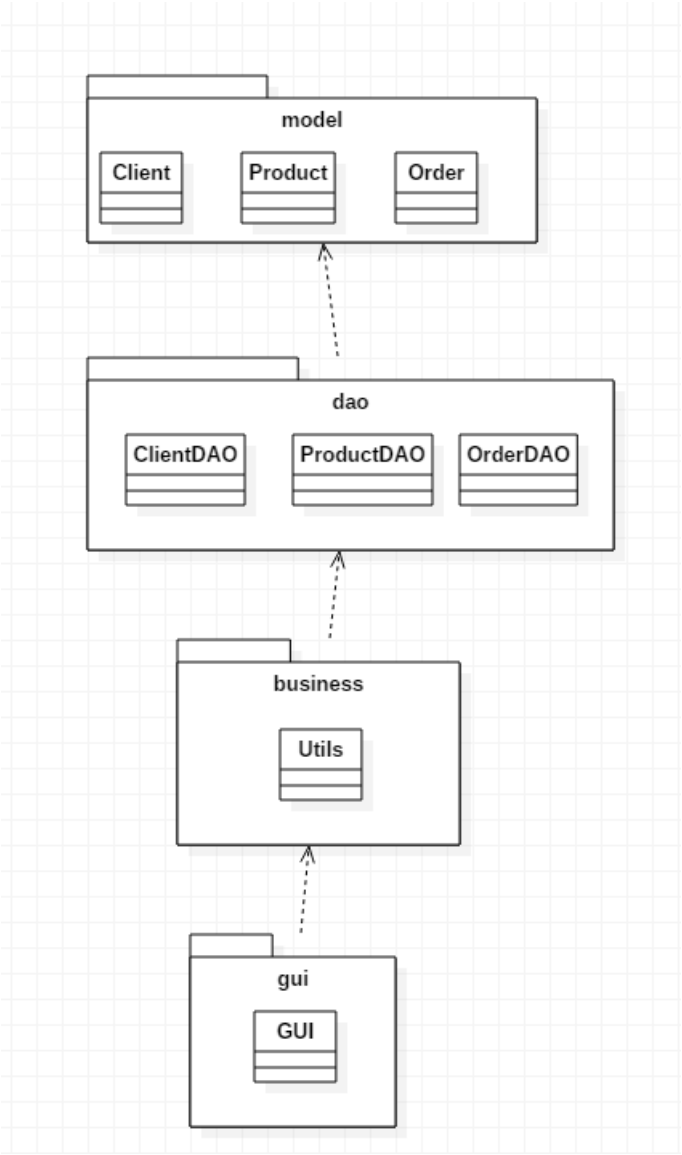
3.1 UML Class Diagram

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

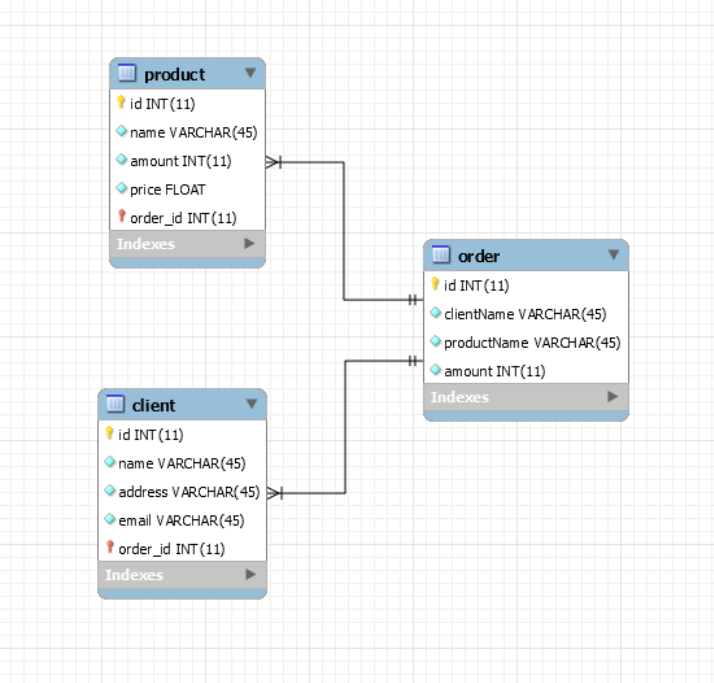
The class diagram is the main building block of object-oriented modelling. It is used both for general conceptual modelling of the systematics of the application, and for detailed modelling translating the models into programming code



3.2 UML Package Diagram



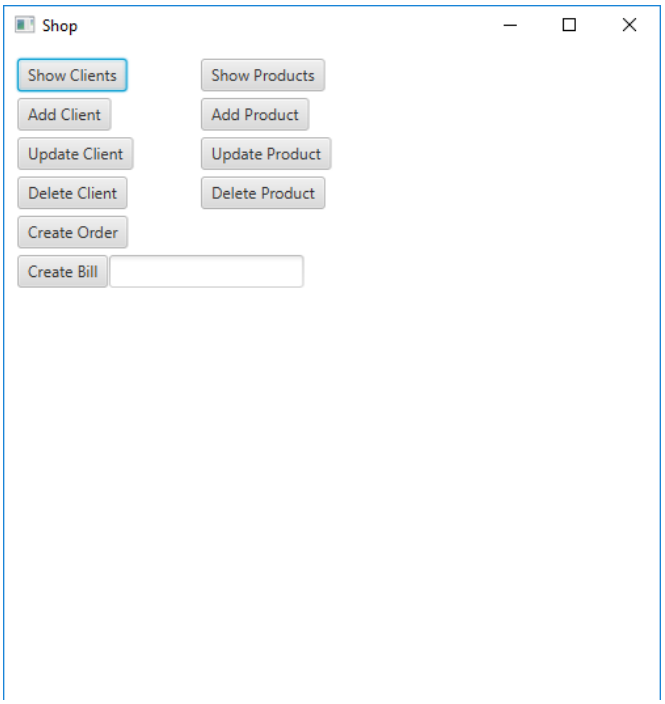
3.3 Database Diagram



3.4 Classes design

1. GUI Class

The “GUI” class implements the interaction with the user, also known as GUI or Graphical User Interface. The window that pops up at runtime:



2. The Client Class: **public class Client**

This class is the representation of a client.

The Fields of the Client Class:

- A private int field “id” which is used to identify the clients.
- A private String field ”name” corresponding to the name of the client.
- A private String field “address” corresponding to the address of the client.
- A private String field “email” corresponding to the email of the client.

The Constructors of the Client Class: **public Client**(**int** id, **String** name, **String** address, **String email**) {}

public Client() {}

- It will initialize the above mentioned fields

The Methods of the Client Class:

- Setters and Getters method:

3. The Product Class: **public class Product**

Same as Client class.

4. The Order Class: **public class Order**

Same as Client Class.

5. The ClientDAO Class: **public class ClientDAO implements clientDAOInter**

The class that retrieves data from the database for the Client model, implements an interface in order to make the code more flexible. Contains methods that implement the CRUD operations on the Client table from the database.

The methods from the ClientDAO class:

- **public Client findClient(String clientName) –** finds a client with the given name
- **public void insertClient(Client client) –** inserts a new client in the database

- **public void updateClient(String address, String name) –** updates the address of the given client
- **public void deleteClient(String clientName) –** deletes the given client
- **public ObservableList<Object> getClients() –** retrieves all the clients from the database in a list
- **public ObservableList<String> getClientsName() –** retrieves all the names of the clients from the database in a list

6. The ProductDAO class: **public class ProductDAO implements productDAOInter**
   Same as ClientDAO class.
7. The OrderDAO class: **public class OrderDAO implements orderDAOInter**
   Same as ClientDAO class.

8. The ConnectionFactory class: **public class ConnectionFactory**
   Realizes the connection to the database and returns it in order to access it for data retrieval.
9. The Utils class: **public class Utils**
   An utility class that contains all the business logic.
   The methods of the Utils class:
   - **public void insertClient(int id, String name, String address, String email) –** inserts a new client in the database
   - **public void insertProduct(int id, String name, int amount, float price) –** inserts a new product in the database
   - **public void updateClient(String name, String address) –** updates the address of a client
   - **public void updateProduct(String name, int amount) –** updates the amount of a product
   - **public void deleteClient(String name) -** deletes the given client
   - **public void deleteProduct(String name) –** deletes the given product
   - **public String findClient(String name) –** returns the name of a client if found or null otherwise
   - **public String findProduct(String name) –** returns the name of a product if found or null otherwise
   - **public String createOrder(String clientName, String productName, int amount) –** creates a new order if possible, if not it returns an under stock error message

- **public void createBill(ArrayList<Order> orders) throws Exception** - creates a new PDF bill for the selected client that contains all the orders that he made and their total price.

3.5 User Interface

As I've previously mentioned, the user interface is pretty much basic and easily-understood, even by non-familiarized people. When running the application, a window will open and it will provide to the user different options. This window is constructed in the GUI class using some predefined classes and instructions.

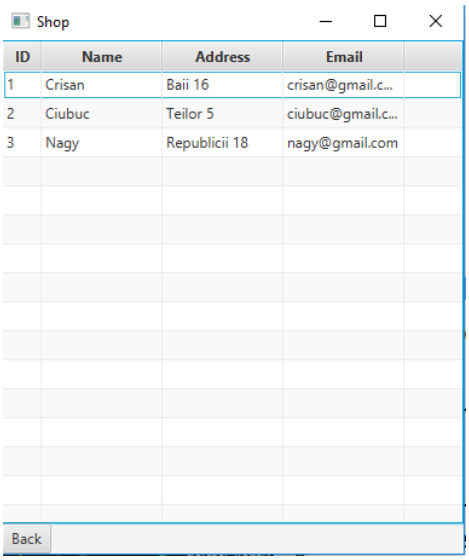
4. Using and testing the application

Rules for the input data:

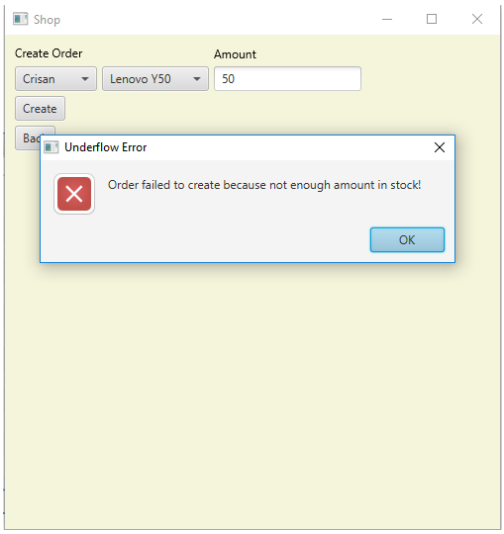
- The input ids, amounts and prices must be given as positive integers (or floats for the price), respecting the intuitive labels existent in the graphical user interface.

Testing example:

1. The view of a table



2. Underflow error



5. Conclusions

In the end, being able to implement something so dynamically and simulating such an often encountered situation of everyday life while combining everything with the concepts and rules of Object-oriented programming was a challenge with a self-rewarding result. As a further development, the first thing that should be a better validation system.

The problem specification presented itself as an interesting subject with many possibilities. Working with databases turned out to impose a lot of additional concepts and ways of implementation on top of the basic Object-oriented paradigms.

There is still a lot of space for improvements in order to make the program more efficient. Some of the algorithms could be optimized and maybe implemented in a different way. Some constrains can be imposed on the input, meaning that more exceptions can be handled with and there can be created more statistics for the simulation. The graphical user interface can be improved. There could be added some colors or given a different layout of the components in the frame.