

Order Management

Student: Bonta Vlad-Valentin

Group:30424

Contents

1. Objective of the assignment.	3
2. Problem analysis.	3
2.1. Analysis of the problem.	3
2.2. Modelling.	3
2.3. Scenarios.	3
2.4. Use cases.	3
3. Design (UML diagrams, data structures, class implementation, interfaces, relations, packages, algorithms, graphical user interface) . . .	4
4. Implementing and testing.	11
5. Results	11
6. Conclusions, what has been learned, further developments	12
7. Bibliography	13

1.Objective of the assignment

The objective of this assignment was finding an optimal and clear solution to design a system for processing the customer orders. In order to store the data, we can use a binary search tree or a database. Such application are used by online shops to store the customers accounts and have a list of available products and to track the orders.

2.Problem analysis

2.1. Analysis of the problem

A customer order is a demand of a customer to the the store/ warehouse the product that he needs. An order consists of a unique identifier of the customer and another unique identifier of the requested product, the quantity of that product. Also, an order may consist of more products.

2.2. Modelling

Having these in mind, I have implemented a solution using a database that holds my data and a Java program with an interface that simulates the customers and products functions like creating, editing and deleting and an user can make his own “shopping cart” consisting of one or more orders. For this implementation, the informations that are required for user and the product are few because of the main purpose of the project: a simulation.

Speaking of the database, it is implemented very simple in SQLite, which is a relational database management system and the connection between the application and the database is made using jdbc-driver ^[1]. It has only 3 tables to store the data: Customer, Product and Order, each with the same columns like in the app.

2.3. Scenarios

The desired scenario is when the user enters the customers and products in the right format and makes quite simple orders. But this scenario may not happen all the time. For example, he can introduce different data types as input and for these cases, the application should be able to handle those exceptions.

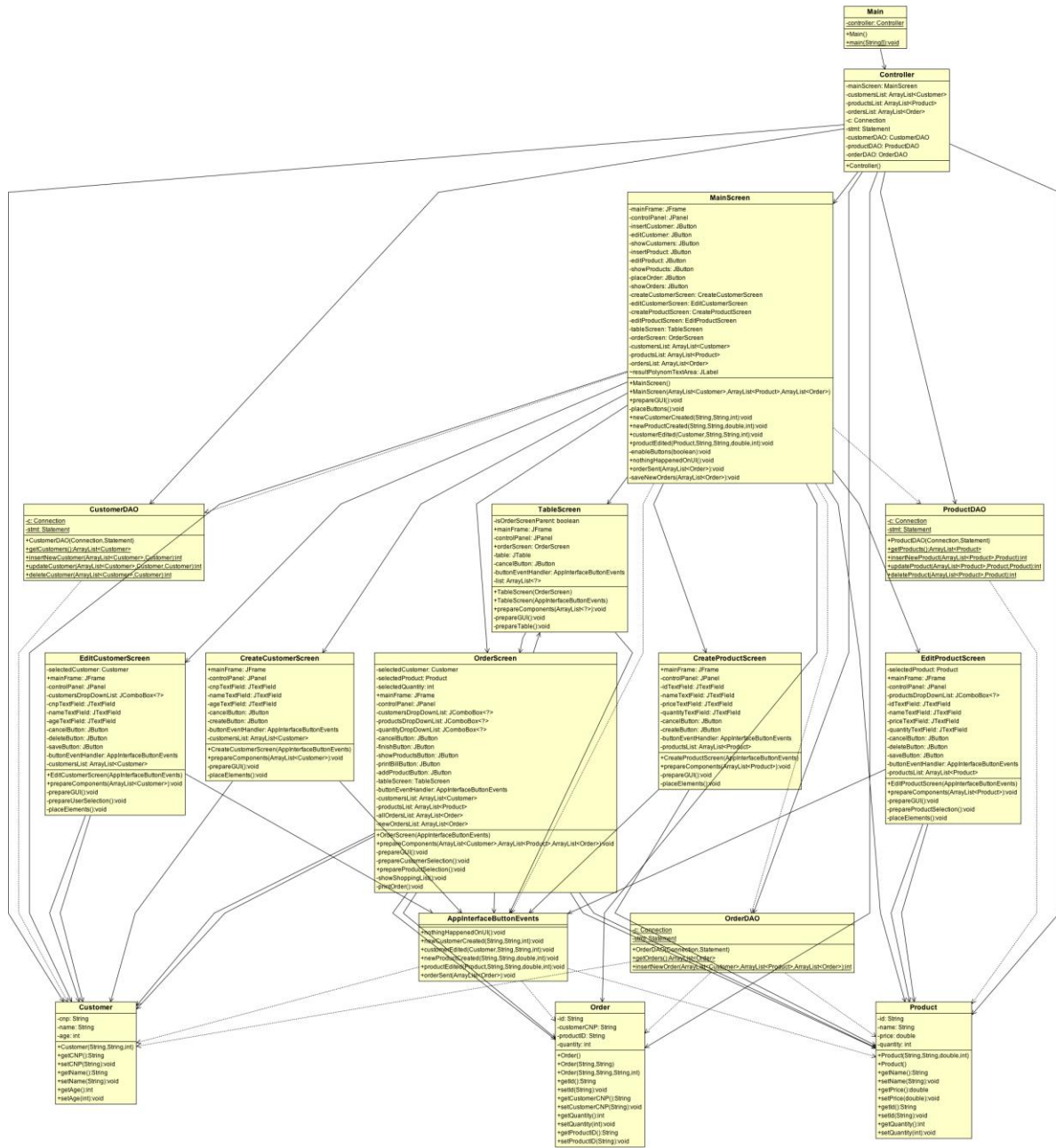
2.4. Use cases

A use case defines the interactions between external actors and the system under consideration to accomplish a goal. Actors must be able to make decisions.^[2] According to this definition, we can identify the main actor as the user of application.

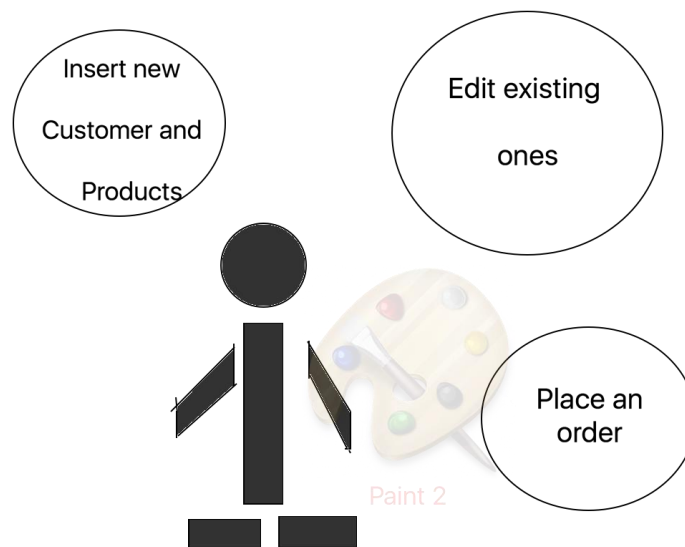
Given that the user is the main actor, we can highlight some of the ways he can interact with the application: the user can create customers and products using the provided interface, after that, he can edit them, and the most important part, it can create orders choosing from the existing users and products. Another important aspect of the application is that the user can save the data into text files. Right now, the table with the orders are written not very nice in the text file but it can be improved.

3.Design

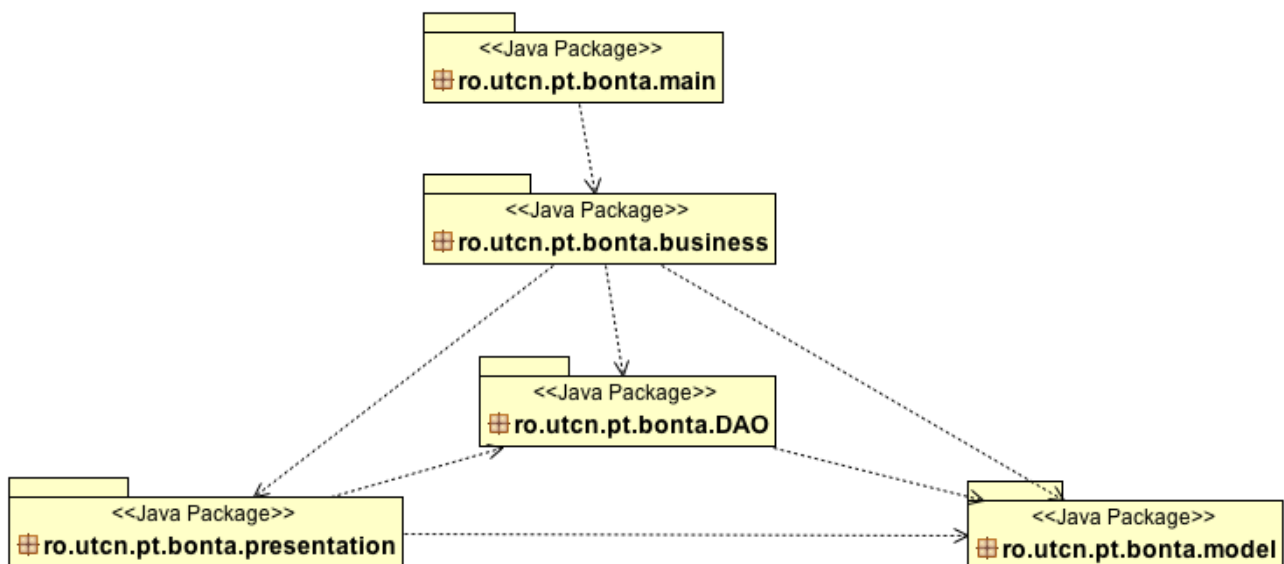
3.1.a UML Class Diagram



3.2.b Use case diagram



3.4.PackageDiagram



The package diagram shows how the packages are organised.

3.2. Data Structures

Customer: The Customer class has as attributes the description of a customer: it's unique field, the CNP of the customer, his name and the age. A valid customer consists of valid CNP having only digits and not empty, name should not be empty and the age should be numerical and not

empty also. For current implementation, the CNP field do not require 10 digits because of the testing purposes.

Product: The product class has as attributes the description of a product: an unique field, an ID, the product name, price being a double and the quantity. A valid product has a unique id that is not null, it's name can not be null, the price is numeric and positive, the quantity should be numeric and positive as well;

Order: The order class has as attribute the description of an order: an unique identifier, called ID, the cap of the customer that make the order, the id of the product and the quantity. A valid order has a valid connection with user and product by theirs unique identifiers and the quantity that the customer requests should be available on the stock.

3.3. Algorithms

Considering the assignment and the implementation of the solution using a database, no complex algorithms were applied.

3.4. Class Design

Classes used in project are split into classes regarding the purpose of each class. Using the data

(default package)	BusinessLogic	DAO	Model	Presentation
Main	Controller	CustomerDAO	Customer	AppInterfaceButtonsEvents
		ProductDAO	Product	CreateCustomerScreen
		OrderDAO	Order	CreateProductScreen
				EditCustommerScreen
				EditProductScreen
				MainScreen
				OrderScreen
				TableScreen

Using Maven it structures the project such as you see a clear differentiation between implementation classes and test classes. The resources are also well organized, such that implementation and tests have different folders for resources.

Using a database, the application is structured in layers: Business Logic Layer for the controller part, Model Layer having the classes for Customer, Product and Order modelling, the DAO layer which is the responsible class for handling the model from app with the model from the database. The last layer is the Presentation Layer which makes the app more user friendly.

Controller

The Controller class is used as the controller of the application. It manages the interface and the flow of the app.

Controller
-mainScreen: MainScreen
-customersList: ArrayList<Customer>
-productsList: ArrayList<Product>
-ordersList: ArrayList<Order>
-c: Connection
-stmt: Statement
-customerDAO: CustomerDAO
-productDAO: ProductDAO
-orderDAO: OrderDAO
+Controller()

MainScreen

The MainScreen class is responsible for the whole interface. It manages all the screens and the exceptions that can appear. It implements the interface AppInterfaceButtonEvents which is used as a delegate between the MainScreen class and other screens. The other screens announce the MainScreen when the user entered the data for a customer/product or an order was made.

MainScreen
-mainFrame: JFrame
-controlPanel: JPanel
-insertCustomer: JButton
-editCustomer: JButton
-showCustomers: JButton
-insertProduct: JButton
-editProduct: JButton
-showProducts: JButton
-placeOrder: JButton
-showOrders: JButton
-createCustomerScreen: CreateCustomerScreen
-editCustomerScreen: EditCustomerScreen
-createProductScreen: CreateProductScreen
-editProductScreen: EditProductScreen
-tableScreen: TableScreen
-orderScreen: OrderScreen
-customersList: ArrayList<Customer>
-productsList: ArrayList<Product>
-ordersList: ArrayList<Order>
~resultPolynomTextArea: JLabel
+MainScreen()
+MainScreen(ArrayList<Customer>,ArrayList<Product>,ArrayList<Order>)
+prepareGUI():void
-placeButtons():void
+newCustomerCreated(String,String,int):void
+newProductCreated(String,String,double,int):void
+customerEdited(Customer,String,String,int):void
+productEdited(Product,String,String,double,int):void
-enableButtons(boolean):void
+nothingHappenedOnUI():void

Customer

The Customer class handles the model of a customer with simple data about him. Every field is private and has its own getter and setter. The database model of this class is handled by the CustomerDAO class.

Customer
-cnp: String -name: String -age: int
+Customer(String,String,int) +getCNP():String +setCNP(String):void +getName():String +setName(String):void +getAge():int +setAge(int):void

Product

The Product class handles the model of a product with simple data about it. Every field is private and has its own getter and setter. The database model of this class is handled by the ProductDAO class.

Product
-id: String -name: String -price: double -quantity: int
+Product(String,String,double,int) +Product() +getName():String +setName(String):void +getPrice():double +setPrice(double):void +getId():String +setId(String):void +getQuantity():int +setQuantity(int):void

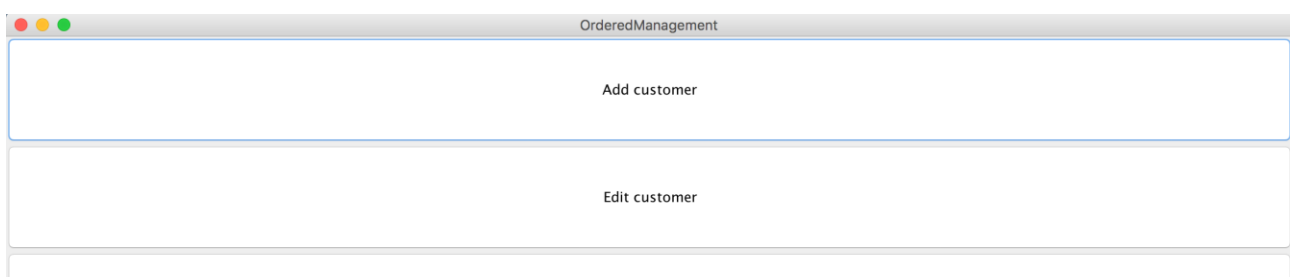
Order

The Order class handles the model of an order with simple data about it. Every field is private and has its own getter and setter. The database model of this class is handled by the OrderDAO class.

Order
-id: String -customerCNP: String -productID: String -quantity: int
+Order() +Order(String,String) +Order(String,String,String,int) +getId():String +setId(String):void +getCustomerCNP():String +setCustomerCNP(String):void +getQuantity():int +setQuantity(int):void +getProductID():String +setProductID(String):void

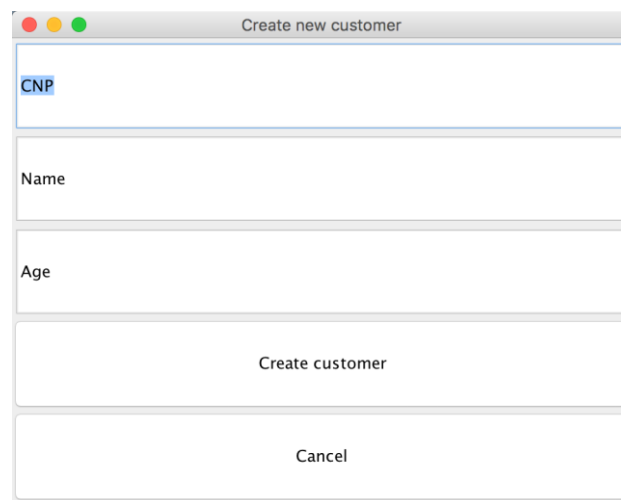
3.5. User Interface

This application is a user oriented application, so it is important to design an interface that is user friendly. Taking this into consideration, the interface should be intuitive and simple to use. The User Interface was built entirely with Java Swing elements: JFrame with a GridLayout and JPanels added to this frame. Buttons, labels and text field were added on these panels to make the interface more readable for the user. The size of the main window is set to 1200/1200 pixels.



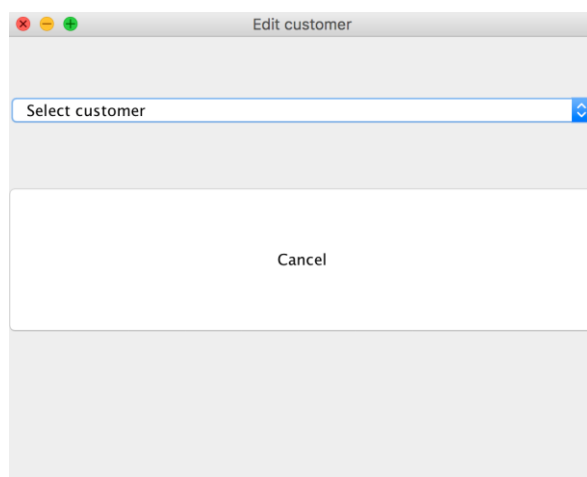
The main window has 8 buttons which takes the user to another frames that will let him work with the app. It can create new customers or products, edit existing ones, including the deletion of the, he can place an order or show all the orders from the database. The user is not allowed to delete or edit a made order because these orders should remain “forever” in the database.

The screen for creating the user asks user to enter the cap, name an age of the new customer. The app tells user if the given data is not right, for example, the user tries to insert text into age field the app tells user that the field should be filled with a positive number. For the product creation screen, only the fields are different.



The screenshot shows a window titled "Create new customer". It contains three text input fields: the first is labeled "CNP" and has the text "CNP" entered; the second is labeled "Name"; and the third is labeled "Age". Below these fields are two buttons: "Create customer" and "Cancel".

The Edit customer screen is presented in the next image. It has the fields of the Customer and are filled with the data of the chosen customer. Here you can choose to delete the selected customer. Of course, it requires another validation of the user input. The same thing is for the Product model but with it's own fields.



The screenshot shows a window titled "Edit customer". It features a dropdown menu at the top with the placeholder text "Select customer". Below the dropdown is a "Cancel" button.

The View customers screen presents in a JTable the customers that are in the database. For the JTable I used reflection ^[2] to take the fields of the customer class and set them as headers of

the table, and iterating through the objects of the Customer class I was able to populate the table. The same thing is done for the Product class.

cnp	name	age
23	Marius	23
1231	Vlad	20
24356	Name	2323

The order screen is made very simple. The user has to choose between existing users, select a product and the necessary quantity. After he selected all the needed products and there are enough products, the user can choose to print the file in a text file or to save his order.

4. Implementation and testing

The application was implemented using Java programming language and Eclipse IDE for Java developers. The project was created using Maven and SQLite-jdbc for the database connection. The implementation is very simple and it did not require complex algorithms and data structures. Using Object-Oriented Programming concepts and a Model View Controller approach, the solution for the assignment is simple and every used class handles its purpose very simple. MVC makes the code easier to understand because the model part is formed from the Customer, Product and Order, the screens are in the presentation package and are separated by their screen purpose. The requests used are part from the DAO classes and the connection to the database is made at beginning.

The implementation for the reflection of the JTable is pretty simple. The next code presents how the reflection was made:

```
for (Field field : allFields) {  
  
    if (Modifier.isPrivate(field.getModifiers())) {  
  
        privateFields.add(field);  
  
        columnNames[index] = field.getName();  
  
        System.out.println(columnNames[index]);  
  
        index++;  
  
    }  
  
}
```

During and also after the design and implementation of the assignment, the functionality has to be tested. In order to do this, I used System.err.println(String x) function which prints the string on the standard error output. Another tool that I used is the debugger from Eclipse. It let you trace the execution of code line by line helping you finding minor problems in implementation. In order to verify if the requests to the database are correctly and are really

updating the database. SQLite gives the opportunity to have fast access to the database and querying it to check if data is updated recording to the requests.

Another testing should be done on the input from the user. Like I have said, the user may not suit to the best scenario and he would insert some invalid data for the Customer or Product info. In order to let the user know what was wrong with his requests. Several validation have been implemented for the user input and using a pop-up the app sends the suitable message for the error.

5. Results

I consider I have reached a good implementation of the OrderManagement assignment. It has a simple interface which solves the problem of order management.

SQLite is pretty simple to use. A request can be made very intuitively, for example:

```
sql = "INSERT INTO Customer VALUES
      (" + customer.getCNP() + ", "
      + customer.getName() + ", "
      + customer.getAge() + ");";
```

6. Conclusions, what has been learned, further developments

This assignment gives me a chance to revise my knowledge about the databases and requests between it and the application. Maven helped me to connect the app to the driver.

```
<dependency>
  <groupId>org.xerial</groupId>
  <artifactId>sqlite-jdbc</artifactId>
  <version>3.8.11.2</version>
</dependency>
```

The documentation written in Javadoc let you navigate through classes and see the methods and attributes of each class. Javadoc is very simple to use, inserting comments and then get an interactive web page to look into.

```
/**
 * Controller.java - Controller : brain of the app. It makes the
 * connection to the database and manages the interface.
 * @author Vlad Bonta
 */
```

For further developments I could think of various improvements and new features such as:

- Add more fields to the Customer and Order models, like : email, address, respectively year of fabrication, details about the manufacturer

- Track the number of orders of every customer and print for them his historic of orders
- Give the possibility to create accounts for customers as the buyers and different account to the sellers
- Improve the aspect of the user interface
- Make the app more user friendly

7. Bibliography

[1] <http://theopentutorials.com/tutorials/java/jdbc/jdbc-mysql-create-database-example/>

[2] <http://tutorials.jenkov.com/java-reflection/index.html>

<http://www.tutorialspoint.com/sqlite/>