**Documentation Homework 3**

**Order Management**

**Name: Popa Adriana**

**Group: 30423**

**Laboratory assistant: Antal Marcel**

1. **Objectives**

Consider an application OrderManagement for processing clientorders for a warehouse. Relational databases are used to store the products, the clients and the orders. Furthermore, the application should be structured in packages using a and should use (minimally) the following classes:

* Model classes -represent the data models of the application
* Business Logic classes -contain the application logic
* Presentation classes –GUI related classes
* Data access classes -classes that contain the access to the database

1. **Dimensions of the problem**
2. **General overview**

This application should be able to fulfil all the requirements in order to display, modify, and keep track of orders, clients and products. These are stored in a relational MySQL database, along with the information about the users which have access to the system. This way, all the data is easier to retrieve and access from different computers.

1. **Input and Output**

When talking about the input in the application, the user can choose to manage 3 tables, Client, Product, Order. All tables have the options defined as the CRUD operations, Create (insert entry) , Read ( show all entries ), Update (modify entry) and Delete, and the user can introduce the values in the specific fields for any of the 3 different tables.

For example, for the Client table the user can introduce an ID for the client ,the name of the client, the Email address and the phone number. All fields have to obey the standard rules such as phone number has to have 10 digits ,the name of the client can contain only letters and spaces, the email address has to contain the @ and the . symbols .

For the following tables, the input is similar but with different fields.

1. **Scenarios and use cases**

* Title: Add Client

Resume: The user can add a new Client by introducing the client details such as id, name, email address and phone number in the specified fields and then he/she should press the Insert button. If the client is created successfully, a pop-up will appear and say that the client was created, and in case of error another pop-up will state that there was an error somewhere.

Actors: User

* Title: Delete Client

Resume: The user can remove a client by specifying the client ID in the dedicated field and then pressing the Delete button. If the operation was successful a pop-up will appear and state that. In case of error another pop-up will state the error.

Actors: User

* Title: Update Client

Resume: The user can edit the details of an existing client. He/ She should fill all the fields representing client details with the new values and press the Update button. If the operation was successful a pop-up will appear and state that. In case of error another pop-up will state the error.

Actors: User

* Title: Add Product

Resume: The user can add products in order to be bought by the clients, the products have to be specified via Id, name, price and stock, meaning the total amount of pieces of that particular product. Each order decreases the stock because people buy those products from our market. Any cancelled/ deleted order will “put back” the amount of products involved in that order. If the operation was successful a pop-up will appear and state that. In case of error another pop-up will state the error.

Actors: User

* Title: Delete Product

Resume: The user can remove a product by introducing the ID of the product he/ she wants to delete and then press the delete button. If the operation was successful a pop-up will appear and state that. In case of error another pop-up will state the error.

Actors: User

* Title: Update Product

Resume: The user can edit the details of an existing product. He/She has to input all the details of the product he/she wants to modify and then press the Update button. If the operation was successful a pop-up will appear and state that. In case of error another pop-up will state the error.

Actors: User

* Title: Show Product

Resume: The user can display all the products pressing the Show All button. If a product is modified/ created/ deleted and the user wants to see that modification he/she has to press the Show All button again.

Actors: User

* Title: Show Client

Resume: The user can display all the clients pressing the Show All button. If a client is modified/ created/ deleted and the user wants to see that modification he/she has to press the Show All button again.

Actors: User

* Title: Show Order

Resume: The user can display all the order pressing the Show All button. If an order is modified/ created/ deleted and the user wants to see that modification he/she has to press the Show All button again.

Actors: User

* Title: Insert Order

Resume: The user can insert a new order by specifying the id of the order, selecting the client id from the list of clients, selecting the product id and introducing the quantity that will be bought by the client. If the operation was successful a pop-up will appear and state that. In case of error another pop-up will state the error.

Actors: User

* Title: Delete Order

Resume: The user can remove an order by introducing the ID of the order he/she wants to delete and then press the delete button. If the operation was successful a pop-up will appear and state that. In case of error another pop-up will state the error.

Actors: User

* Title: Update Order

Resume: The user can edit the details of an existing order. He/ She has to input all the details of the order he/she wants to modify and then press the Update button. If the operation was successful a pop-up will appear and state that. In case of error another pop-up will state the error.

Actors: User

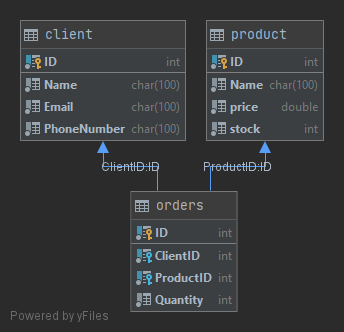
* Title: Generate Bill

Resume: The user can decide if he wants to generate a bill for a specified order. He/ She has to input the id of the order he/ she wants the bill for and press the Generate Bill button. If it is generated successfully a pop-up will appear and state that.

Actors: User

**3. Project design**

1. **Database Diagram**

****

1. **Package Diagram**

Diagram

Description automatically generated

Java packages help in organizing multiple modules and group together related classes and interfaces.

I devided my classes into 7 packages:

* Application – contains a single class, which contains the customary main() method
* Model –  contains the classes which model the problem
  + Client
  + Order
  + Product
* Business Logic – contains the logic of the application
* Validators – contain validators for model entries
* Connection – contains the class the connects to database
* Data Access – contains the classes that connect with the database
* Presentation – contains the classes which create the GUI
* Control – it interconnects the model and the view

1. **Class Diagram**

Text

Description automatically generated

1. **Data structures**

## Java ArrayList

The ArrayList class is a resizable array, which can be found in the java.util package.

The difference between a built-in array and an ArrayList in Java, is that the size of an array cannot be modified (if you want to add or remove elements to/from an array, you have to create a new one). While elements can be added and removed from an ArrayList whenever you want.

**4. Implementation**

1. **The Model** 
   * Client Class

Constructors:

public Client(int ID, String name, String email, String phoneNumber)

public Client(){}

Getters and setters:

public int getID()  
public void setID(int ID)   
public String getName()  
public void setName(String name)   
public String getEmail()  
public String getPhoneNumber()  
public void setEmail(String email)   
public void setPhoneNumber(String phoneNumber)

* Orders Class

Constructors:

public Orders(int ID, int clientID, int productID, int quantity)public Orders(){}

Getters and setters:

public int getID()  
public void setID(int ID)   
public int getClientID()  
public int getProductID()  
public int getQuantity()  
public void setClientID(int clientID)   
public void setProductID(int productID)   
public void setQuantity(int quantity)

* Product Class

Constructors:

public Product(int ID, String name, double price, int inStock)public Product(){}

Getters and setters:

public int getID()  
public void setID(int ID)   
public String getName()  
public void setName(String name)   
public double getPrice()  
public int getStock()  
public void setStock(int inStock)   
public void setPrice(double price)

1. **Connection**

* Connection Factory

Constructors:

private ConnectionFactory()

Methods:

private Connection createConnection() - create a connection

public static Connection getConnection() - return the connection

public static void close(Connection connection) - close the connection

public static void close(Statement statement) - close the statement

public static void close(ResultSet resultSet) - close the result set

1. **Data Access**

* Abstract DAO

Constructors:

public AbstractDAO() – gets the type of the object

Methods:

public String createSelectQuery() - create a simple select query

public String[] returnColumns(T t) - return the name of the columns for the table view

public T findById(int id) - return the object having as ID id

public List<T> createObjects() - create a list with all object from a table

public void executeQuery(StringBuilder query) - execute a given query

public T insert(T t) - insert an object in the table

public T update(T t) – update an object in the table

public T delete(T t) – delete an object from the table

* Client DAO

Methods:

public List<Client> createClientsList()- *Call functions from AbstractDAO* ***@return*** *list of clients*public void insertNewClient(Client c) - *Call insert function from AbstractDAO*  
public void updateClient(Client c) - *Call update function from AbstractDAO* public void deleteClient(Client c) - *Call delete function from AbstractDAO*public String[] returnColumns(Client c) - *Gets the columns names for the table*public Client findClient(int id) - *Get the client with specified id*

* Order Dao

Methods:

public List<Orders> createOrdersList() - *Get the list of all orders in the data base*public void insertNewOrder(Orders o) - *Call insert from AbstractDAO*  
public void updateOrder(Orders o) - *Call update from AbstractDAO*public void deleteOrder(Orders o) - *Call delete from AbstractDAO*public Orders findOrder(int id) - *Call findById from AbstractDAO*public String[] returnColumns(Orders o) - *Call returnColumns from AbstractDAO*

* Product DAO

Methods:

public List<Product> createProductsList() - *Get the list of all products in the data base*public void insertNewProduct(Product p) - *Call insert from AbstractDAO*public void updateProduct(Product p) - *Call update from AbstractDAO*  
public void deleteProduct(Product p) - *Call delete from AbstractDAO*public String[] returnColumns(Product p) - *Call returnColumns from AbstractDAO*public Product findProduct(int id) - *Call findById from AbstractDAO*

1. **Validators**

* Validator Class

Methods:

boolean validate(T t) - *General validator*

* Email Validator

Methods:

public boolean validate(Client t) - *Validate Email*

* InStock Validator

Methods:

public boolean validate(Product t) - *Validate Stock* of the product

* Phone Validator

Methods:

public boolean validate(Client t) - *Validate Phone number*

* Price validator

Methods:

public boolean validate(Product t) – validate price for the product

* Quantity Validator

Methods:

public boolean validate(Orders t) – validate the quantity the order

1. **Business Logic**

* ClientBLL

Constructor:

public ClientBLL(ClientController clientController)

Methods:

public boolean validateID(Client t) - *Check if id is unique*

public boolean validateClient(Client t) - *\*Use the validators to check the client*public void insertClient(Client t) - *validate and insert client* public void updateClient(Client t) - *validate and update client*public void deleteClient(Client t) - *validate and delete client*public List<Client> getClients() - *Get the list of clients*public String[] returnColumns(Client c) - *Get column names*  
public Client findClient(int id) - *Get client with specified id*

* OrderBLL

Constructors:

public OrderBLL(OrderController orderController)

Methods:

public boolean validateID(Orders t) - *Check if id is unique*public boolean validateOrder(Orders t) - *Use the validators to check the order*public void insertOrder(Orders t) - *validate and insert order and product stock*public void updateOrder(Orders t) - *validate and update order and product stock*public void deleteOrder(Orders t) - *validate and delete order and product stock*public Orders findOrder(int id) - *Get order with specified id*public List<Orders> getOrders() - *Get the list of orders*public String[] returnColumns(Orders o) - *Get column names*

* ProductBLL

Constructor:

public ProductBLL(ProductController productController)

Methods:

public boolean validateID(Product t) - *Check if id is unique*public boolean validateProduct(Product t) - *Use the validators to check the product*public void insertProduct(Product t) - *validate and insert product*public void updateProduct(Product t) - *validate and update product*public void deleteProduct(Product t) - *validate and delete product*public List<Product> getProducts() - *Get the list of products*public String[] returnColumns(Product p) - *Get column names*public Product findProduct(int id) - *Get product with specified id*

1. **Presentation**

* Main controller

Methods:

public void goToClientWindow(ActionEvent event) – go to client interface

public void goToProductWindow(ActionEvent event) – go to product interface

public void goToOrderWindow(ActionEvent event) – go to order window

* Client Controller

Methods:

public void insertButton() - *When button pressed insert client in the table*

public void updateButton() - *When button pressed update client in the table*

public void deleteButton() - *When button pressed delete client in the table*

public void show() - *Display the table of clients*

private ObservableList<Client> getObservableClients() - *Create observable list of clients*

* Order Controller

Methods:

public void showError(String s,String s1) - *Display error message*

public void showFinalizationMessage(String s,String s1) - *Display success message*

public void initialize() - *Initialize stage*

public void getClientsID() - public void getClientsID()

public void getProductsID() - *Get ID's of the products in the data base*

public void generateBill() - *Generate a .txt of the file with the bill*public void show() - *Display the table of orders*private ObservableList<Orders> getObservableOrders() - *Generate observable list of orders*

public void goToMainWindow(ActionEvent event) - *Go back to main stage*

* Product Table

Methods:

public void showError(String s,String s1) - *Display error message*

public void showFinalizationMessage(String s,String s1) - *Display success message*

public void insertButton() - *When button pressed insert product in the table*

public void updateButton() - *When button pressed update product in the table*

public void deleteButton() - *When button pressed delete product in the table*

public void goToMainWindow(ActionEvent event) - *Go back to main stage*

public void show() - *Display the table of products*

1. **Application**

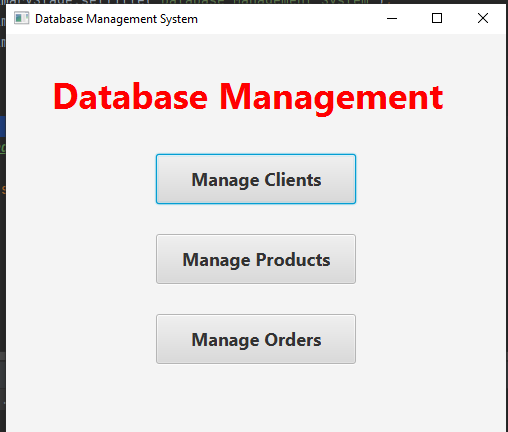
* Main

Methods:

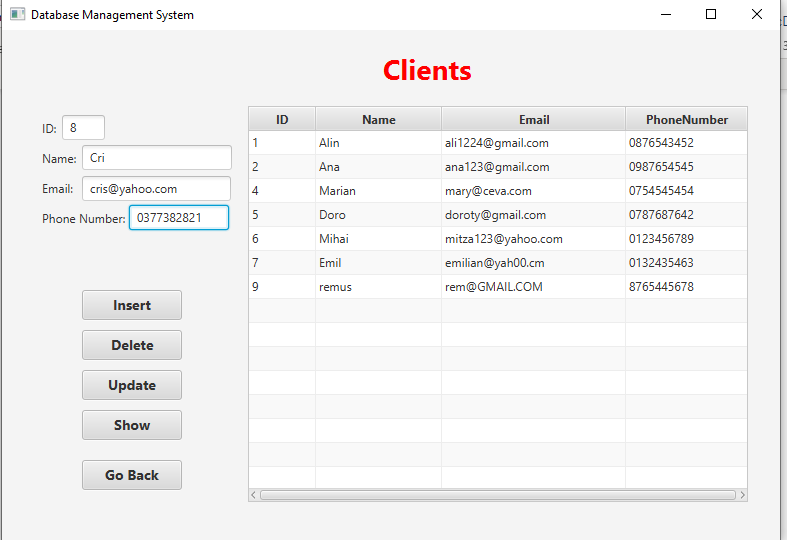
public void start(Stage primaryStage) - *Start the application*

public static void main(String[] args)- *Main of the program*

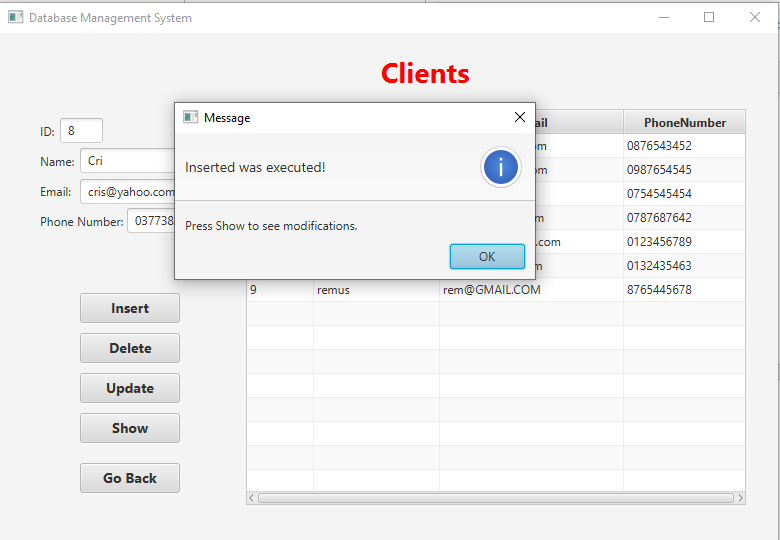
**5 .Results**

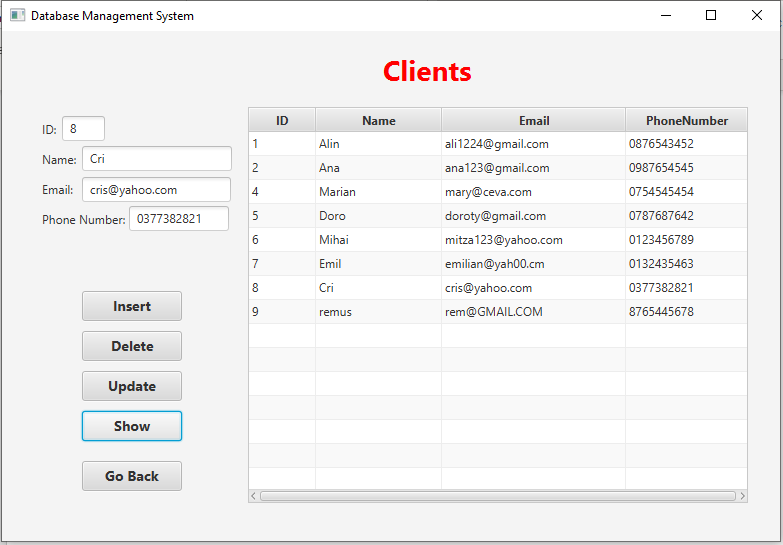


Before insertion

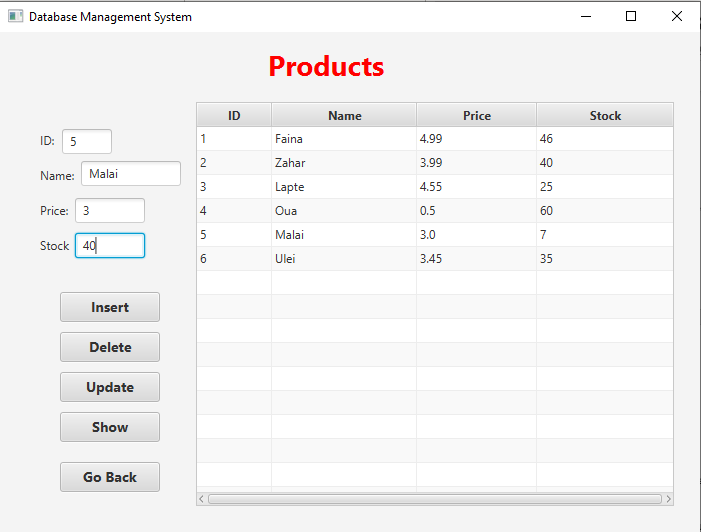


After insertion

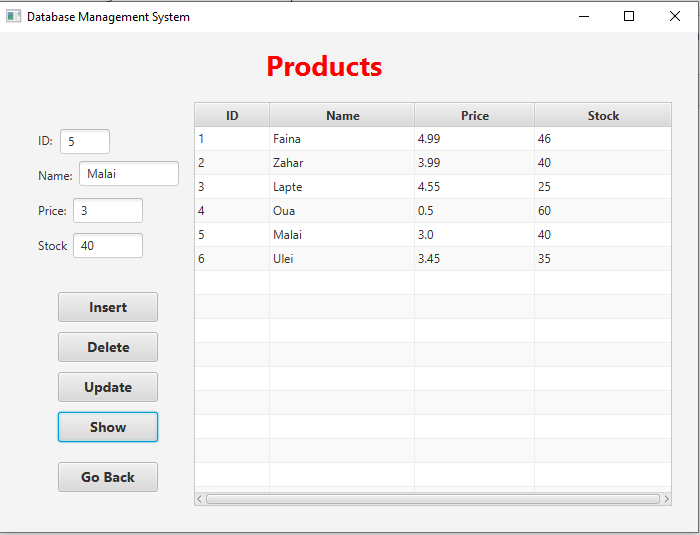


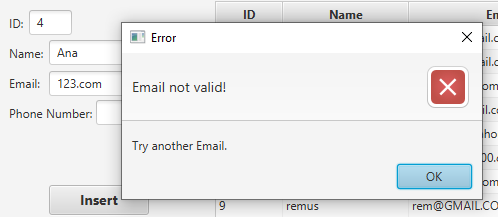


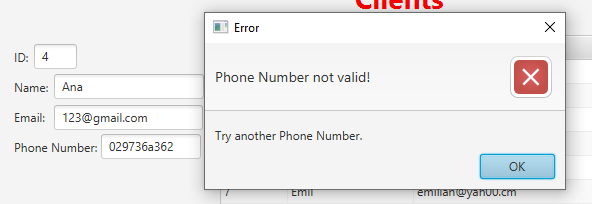
Before Update

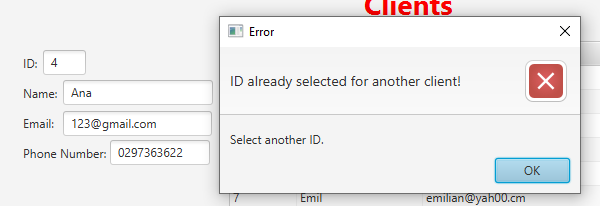


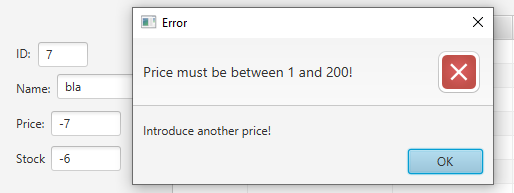
After update

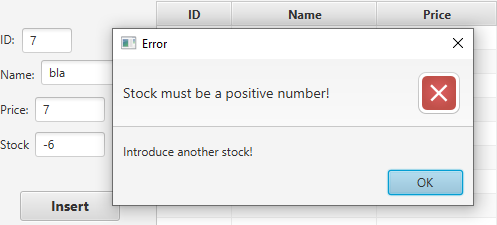




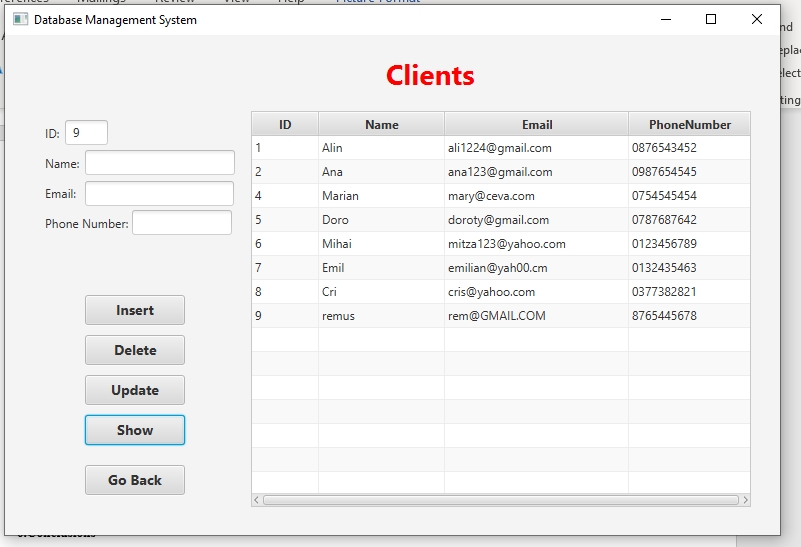




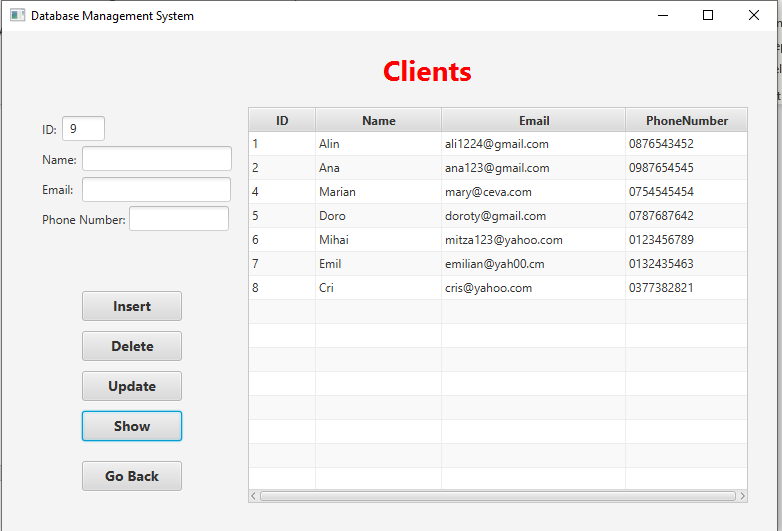




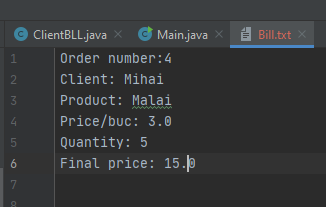
Before Delete



After delete



Bill generated



**6.Conclusions**

This project was a good exercise in remembering the OOP concepts learned in the first semester, but also learning new ones, I found it very useful and challenging at first. There are a few learned things which I would present next.

First of all , time management is very, very, very, crucial, because a good organizational spirit helps you see things gradually and making things from time helps you a lot.

Secondly, modelling the problem in a right way from the beginning helps you to implement it faster.

Thirdly, I arrived at the conclusion that facing problems with your code and trying to make it work by yourself, through the mean of research, has the benefit of learning new concepts and a better use of the known ones.

**7.Bibliography**

* + Object-Oriented Programming - Lecture Slides of prof. Marius JOLDOS
* Programming Techniques – Lectures of prof. Ioan SALOMIE
* <https://www.jetbrains.com/help/idea/export-data.html#productivity-tips>
* <https://stackoverflow.com/questions/1901164/get-type-of-a-generic-parameter-in-java-with-reflection>
* <https://www.softwaretestinghelp.com/javadoc-tool/>
* <https://stackoverflow.com/questions/21659691/error-1452-cannot-add-or-update-a-child-row-a-foreign-key-constraint-fails>
* <https://stackoverflow.com/questions/1901164/get-type-of-a-generic-parameter-in-java-with-reflection>
* https://stackoverflow.com/questions/35848332/javafx-use-reflection-with-custom-annotation-to-create-tablecolumn-cant-creat