Database Application for Processing Customer Orders

Fejér Alpár

Group 30424

Objective

The goal of this project is to develop an application that processes customer orders of various products . The application must support the following operations : add product , delete product , update information about currently existing product , JTable for products , add new customer , delete customer information , update customer information , JTable with all the customers informations , placing and processing customer orders on the existing products and invoice for existing order .

The project works with a graphical user interface , which allows the user to input new data and save it , update information about current products / customers . It also provides the user with a JTable view of every table in the database . Errors are written in the result box in case of incorrect input variables .

Problem analysis

The application has a large variety of input data with a strong relational connection. Thus , figuring out a way to store the data is paramount for solving the problem . A binary search tree data modeling is a good way to store the data for this application . A relational database with multiple tables is also a good choice .

The graphical user interface is an important component of the project , as the user only cares of the simple usage and easy understanding of the program . The graphical user interface in this project is made to fit this description . With only the necessary fields and buttons present , the interface is as simple as it can be .

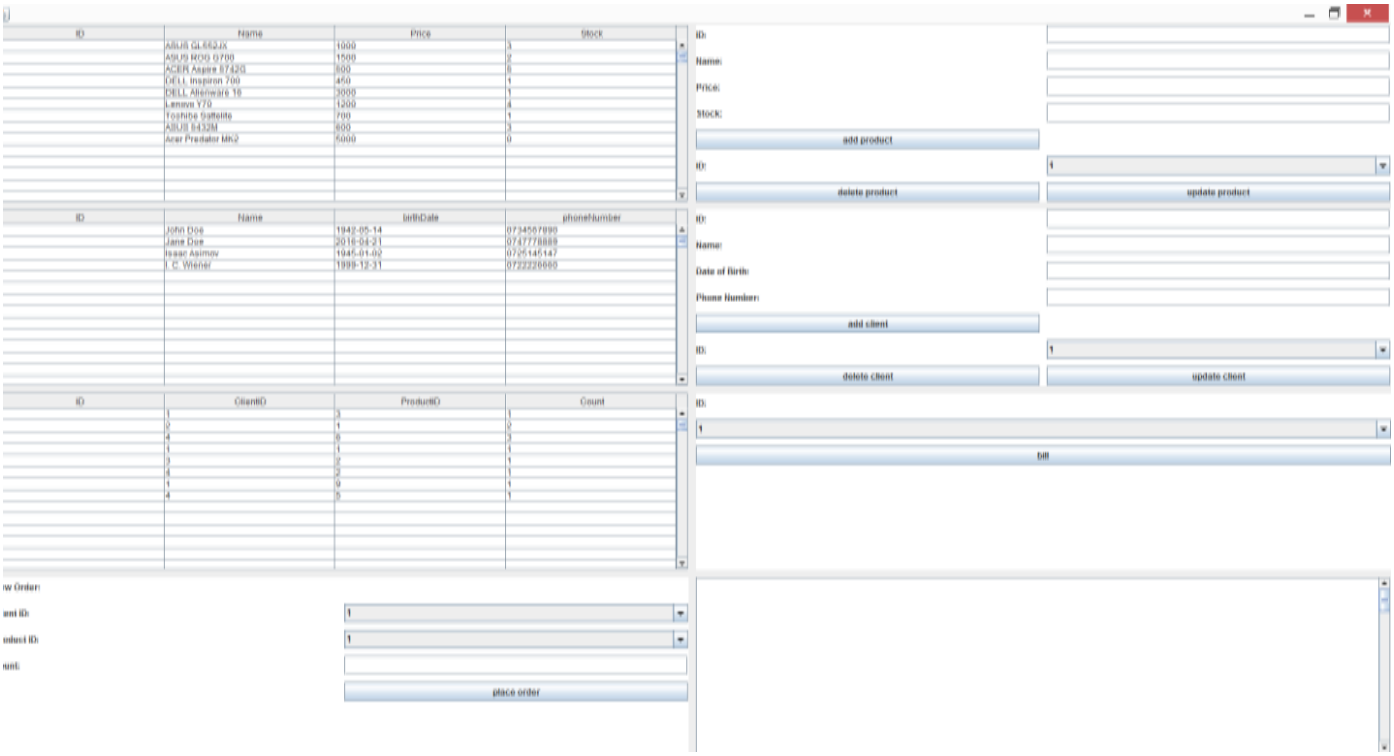
Modeling

A relational database with three tables is used in modeling the data in this application . The tables are : products , which holds information about the products on which the orders are executed . The columns of this table are , as follow : ID , an UNIQUE key , by which the data is identified and also sorted , Name , the name of the product , Price and Stock . The second table is for holding information about the customers , namely clients , with four columns : ID , Name , birthDate and phoneNumber . The third table is the most important one , as it holds information about the orders made by the customers on products . It links the other tables together , with three ID columns and a Count column .

All the tables have UNIQUE constraints , after their own ID’s , names , and a composite Primary key ensures that two orders with same input data are not processed or saved .

The other important structural component of the application is the graphical user interface . It has all the tables from the database and fields for every column to update records . For the sake of simplicity , no additional frame is present in the project , the user can see all the necessary fields when opening the program . When issuing a new order , the product and client ids need to be used , and the user chooses them from combo boxes to ensure that orders won’t contain inexisting products , or be order by someone that is not on the customer list .

User Interface



The graphical user interface is made to be simple , easy to use and to understand , and with as few distractions as possible . On the left side , there are the JTables showing all the data in the database , and also the buttons , combo boxes and text fields needed to issue a new order on the existing data .

The GUI consists of eight JPanels . All of these are used for a single purpose each. The first panel holds the products table in form of a JTable.

```
ptPanel.setLayout(new GridLayout(1,1));
pTable.getModel().addTableModelListener(pTable);
pTable = getProductTable();
psTable = new JScrollPane(pTable);
ptPanel.add(psTable);
```

The second JPanel is for inserting , deleting , and updating data in the products table .'

The first three pair of JPanels are similar in function .

The seventh JPanel is for issuing new orders , and the user can select from the existing customer and products ID's . The order can have a number of the same products , which is given by the count column , read from the text field labeled accordingly .
The last JPanel only has a JText Area , and is used for writing messages to the user , like Underflow on the product stock .

All of the buttons have a common actionlistener .

```
public void actionPerformed(ActionEvent e) {
        if (e.getSource() == addP){
                ProductBL newProduct = new ProductBL(new
Product(Integer.parseInt(pID.getText()),pName.getText(),Integer.parseInt(pPrice.getText(
)),Integer.parseInt(pStock.getText())));
                newProduct.addData();
                @SuppressWarnings("unused")
                GUI newGUI = new GUI();
                this.dispose();
        }
        if (e.getSource() == delP){
                ProductBL newProduct = new ProductBL(new Product());
                newProduct.deleteData((int)bpID.getSelectedItem());
                @SuppressWarnings("unused")
                GUI newGUI = new GUI();
                this.dispose();
        }
        if (e.getSource() == upP){
                ProductBL newProduct = new ProductBL(new
Product(Integer.parseInt(pID.getText()),pName.getText(),Integer.parseInt(pPrice.getText(
)),Integer.parseInt(pStock.getText())));
                newProduct.updateData((int)bpID.getSelectedItem());
                @SuppressWarnings("unused")
                GUI newGUI = new GUI();
                this.dispose();
        }
        if (e.getSource() == addC){
                ClientBL newClient = new ClientBL(new
Client(Integer.parseInt(cID.getText()),cName.getText(),cDate.getText(),cPhone.getText()));
                newClient.addData();
                @SuppressWarnings("unused")
                GUI newGUI = new GUI();
                this.dispose();
        }
        if (e.getSource() == delC){
                ClientBL newClient = new ClientBL(new Client());
                newClient.deleteData((int)bcID.getSelectedItem());
                @SuppressWarnings("unused")
                GUI newGUI = new GUI();
```

```
                this.dispose();
            }
            if (e.getSource() == upC){
                ClientBL newClient = new ClientBL(new
Client(Integer.parseInt(cID.getText()),cName.getText(),cDate.getText(),cPhone.getText()));
                newClient.updateData((int)bcID.getSelectedItem());
                @SuppressWarnings("unused")
                GUI newGUI = new GUI();
                this.dispose();
            }
            if (e.getSource() == order){
                OrderBL newOrder = new OrderBL(new
Order(0,(int)bcID2.getSelectedItem(),(int)bpID2.getSelectedItem(),Integer.parseInt(count.
getText())));
                int success = newOrder.addData();
                if(success == 1){


                @SuppressWarnings("unused")
                GUI newGUI = new GUI();
                this.dispose();
                }
                else if (success == -1) message.setText("Understock");
            }
            if (e.getSource() == bill){
                OrderBL newOrder = new OrderBL();
                newOrder.bill((int) oID.getSelectedItem());
            }
```

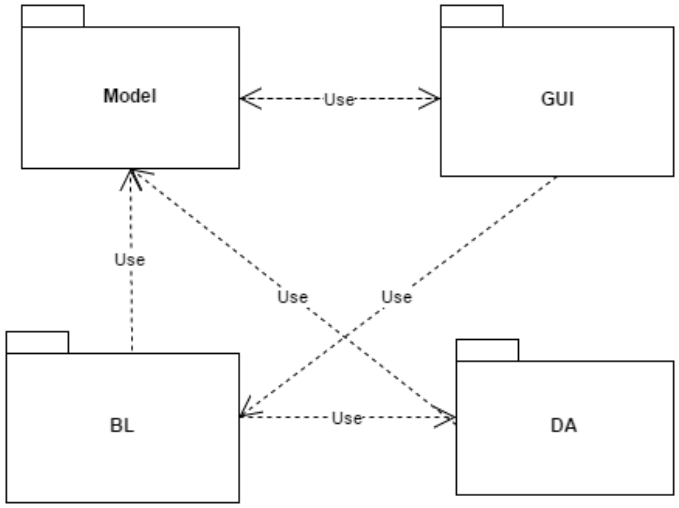

```
}
```

Classes and packages

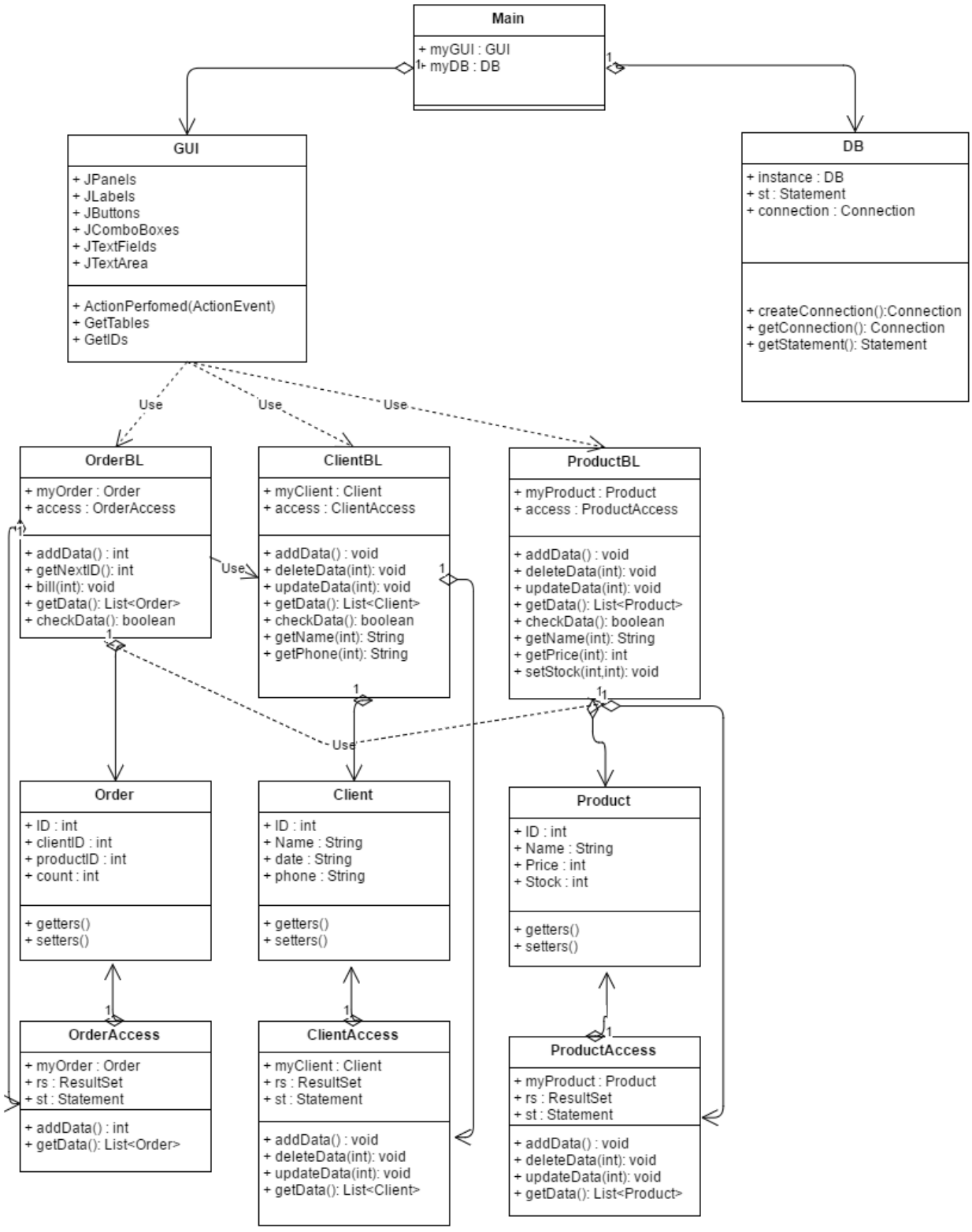
This section contains the description of the used classes and packages in more detail .

Packages

There are a total of four packages in the java project , namely: model , gui , bussinesslogic , and dataAccess for the DAO classes ..



As it can be seen from the diagram , the model holds the main function of the program . This function first calls every JUnit test from the polynomial .test package sequentially to assure that the program will work correctly . Only when all the tests yield a positive result , the user interface is shown and the user can begin using the program .


Main
+ myGUI : GUI
+ myDB : DB
GUI
+ JPanels
+ JLabels
+ JButtons
+ JComboBoxes
+ JTextFields
+ JTextArea
+ ActionPerfomed(ActionEvent)
+ GetTables
+ GetIDs
DB
+ instance : DB
+ st : Statement
+ connection : Connection
+ createConnection():Connection
+ getConnection(): Connection
+ getStatement(): Statement
Use
Use
Use
OrderBL
+ myOrder : Order
+ access : OrderAccess
+ addData() : int
+ getNextID(): int
+ bill(int): void
+ getData(): List<Order>
+ checkData(): boolean
ClientBL
+ myClient : Client
+ access : ClientAccess
+ addData() : void
+ deleteData(int): void
+ updateData(int): void
+ getData(): List<Client>
+ checkData(): boolean
+ getName(int): String
+ getPhone(int): String
ProductBL
+ myProduct : Product
+ access : ProductAccess
+ addData() : void
+ deleteData(int): void
+ updateData(int): void
+ getData(): List<Product>
+ checkData(): boolean
+ getName(int): String
+ getPrice(int): int
+ setStock(int,int): void
Use
Use
Order
+ ID : int
+ clientID : int
+ productID : int
+ count : int
+ getters()
+ setters()
Client
+ ID : int
+ Name : String
+ date : String
+ phone : String
+ getters()
+ setters()
Product
+ ID : int
+ Name : String
+ Price : int
+ Stock : int
+ getters()
+ setters()
OrderAccess
+ myOrder : Order
+ rs : ResultSet
+ st : Statement
+ addData() : int
+ getData(): List<Order>
ClientAccess
+ myClient : Client
+ rs : ResultSet
+ st : Statement
+ addData() : void
+ deleteData(int): void
+ updateData(int): void
+ getData(): List<Client>
ProductAccess
+ myProduct : Product
+ rs : ResultSet
+ st : Statement
+ addData() : void
+ deleteData(int): void
+ updateData(int): void
+ getData(): List<Product>

Model

This package contains the entities to all of the tables in the database .

It holds three classes , namely Product , Client , Order and Main .

Product

This is the entity class for the products table . It has the same fields as a record from the products table would have .

```
public class Product {

    private int id;
    private String name;
    private int price;
    private int stock;

    //rest of the class

}
```

It also has an overloaded constructor , for user – given and default data .

Client

This is the entity class for the clients table , which holds the information about the customers .

```
public class Client {

    private int id;
    private String name;
    private String birthDate;
    private String phone;

    // rest of the class
}
```

Encapsulation is very important in object oriented programming , and more so in applications involving databases , to ensure the integrity of the data , so everything is done with getters and setters .

```
    public int getID(){
```

```
        return this.id;
    }
    public void setID(int id){
        this.id = id;
    }

    public String getName(){
        return this.name;
    }
    public void setName(String name){
        this.name = name;
    }

    public String getBirthDate(){
        return this.birthDate;
    }
    public void setBirthDate(String date){
        this.birthDate = date;
    }

    public String getPhone(){
        return this.phone;
    }
    public void setPhone(String phone){
        this.phone = phone;
    }
```

Orders

It is the entity class for the orders table from the database . This is the link between the other tables from the database , so it has fields from both of those tables , and additional information .

```
public class Order {

    private int id;
    private int clientID;
    private int productID;
    private int count;


    //rest of the class

}
```

Main

The Main function is compact , used only for creating the link between the database and the application , getting the connection for further use and initializing the Graphical User Interface .

```
public class Main {

	public static void main(String[] args){

		@SuppressWarnings("unused")
		DB myDB = new DB();
		DB.getConnection();
		@SuppressWarnings("unused")
		GUI myGUI = new GUI();
}
```

BussinessLogic

This package holds the classes that are the link between the DAO classes and the user interface .

This class holds all the logic that goes into the project . Every input data is validated here , before it is sent to the DAO classes . It has three classes , one for each table . It uses the previously modeled entity classes .

ProductBL

The data from the input is checked before it is sent to the DRO classes . In case of products, the following constraints need to be met . All the other constraints (like uniqueness) are enforced by the structure of the database .

```
	private boolean checkData(){
		if (myProduct.getID() < 1) return false;
		if (myProduct.getPrice() <=0) return false;
		if (myProduct.getStock() < 0) return false;
		return true;
	}
```

OrderBL

The business logic for the orders need to use the business logic from the other classes , as this is the table that connects the other ones . One change to the orders table cannot be made without a change to the products classes . (If a customer orders an amount of one product , either an Understock message should be printed , or the stock of that product must be updated) .

```
public int addData(){
		if (this.checkData()){
			if (myOrder.getCount() > ProductBL.getCount(myOrder.getProductID()))
				return -1;
			else{
				access = new OrderAccess(myOrder);
				access.addData();

	ProductBL.setStock(myOrder.getProductID(),myOrder.getCount());
				return 1;
			}
		}
		else return 0;
	}
```

A required task is also creating invoices to .txt files from current orders .

```
public void bill(int id){
		List<Order> orders = this.getData();
		for (Order o : orders)
			if (o.getID() == id){
				try{
				FileWriter fileWriter = new FileWriter("bill.txt");
				BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);
				bufferedWriter.write("Order ID: ");
				bufferedWriter.write(String.valueOf(id));
				bufferedWriter.newLine();
				bufferedWriter.write("Client Name: ");
				bufferedWriter.write(ClientBL.getName(o.getClientID()));
				bufferedWriter.write(" Client phone number: ");
				bufferedWriter.write(ClientBL.getPhone(id));
				bufferedWriter.newLine();
				bufferedWriter.write("Product name: ");
				bufferedWriter.write(ProductBL.getName(id));
				bufferedWriter.write(" Unit price: ");
				bufferedWriter.write(String.valueOf(ProductBL.getPrice(id)));
				bufferedWriter.write(" Count: ");
				bufferedWriter.write(String.valueOf(o.getCount()));
				bufferedWriter.newLine();
				bufferedWriter.write("Total: ");

	bufferedWriter.write(String.valueOf(o.getCount()*ProductBL.getPrice(id)));
				bufferedWriter.close();
				}
				catch (IOException e){
					System.out.println(e.getMessage());
				}
```

```
                }
        }
```

DataAccess

This package is the direct link between the java application and the MySQL database .

For every operation , a SQL query must be created as a prepared statement to interact with the MySQL database .

There are four classes in this package , one for each table in the database , and one for the database itself .

DB

```
import java.sql.*;

public class DB {
   //static reference to itself
   private static DB instance = new DB();
   protected static Statement st;
   protected static Connection connection;
   public static final String URL = "jdbc:mysql://localhost:3306/ecommerce";
   public static final String USER = "root";
   public static final String PASSWORD = "";
   public static final String DRIVER_CLASS = "com.mysql.jdbc.Driver";

   //private constructor
   public DB() {
      try {
         Class.forName(DRIVER_CLASS);
      } catch (ClassNotFoundException e) {
         e.printStackTrace();
      }
   }

   private Connection createConnection() {

      try {
         connection = DriverManager.getConnection(URL, USER, PASSWORD);
         st = connection.createStatement();
      } catch (SQLException e) {
         System.out.println("ERROR: Unable to Connect to Database.");
      }
      return connection;
```

```
    }

    public static Connection getConnection() {
        return instance.createConnection();
    }

    public static Statement getStatement(){
            return st;
    }
}
```

In the first class , a connection is created with the help of the URL to the database , the username and password of the owner of the database , and a driver Class , called JDBC .

ProductAccess

```
public void addData(){
            try {
                    String query = " insert into products (id, name, price, stock)"
                            + " values (?, ?, ?, ?)";
                    PreparedStatement addQuery = DB.connection.prepareStatement(query);
                    addQuery.setInt(1, myProduct.getID());
                    addQuery.setString(2, myProduct.getName());
                    addQuery.setInt(3, myProduct.getPrice());
                    addQuery.setInt(4, myProduct.getStock());
                    addQuery.execute();
            }
            catch (SQLException e){
                    System.out.println(e.getMessage());
            }
        }

        public void deleteData(int myID){
                try{
                        String query = "delete from products where id = ?";
                        PreparedStatement deleteQuery =
DB.connection.prepareStatement(query);
                        deleteQuery.setInt(1, myID);
                        deleteQuery.execute();
                }
                catch (SQLException e){
                        System.out.println(e.getMessage());
                }
        }

        public void updateData(int myID){
                try{
```

```
                String query = "update products set id=?, name=?,price=?,stock=? where
id = ?";
                PreparedStatement updateQuery =
DB.connection.prepareStatement(query);
                updateQuery.setInt(1, myProduct.getID());
                updateQuery.setString(2, myProduct.getName());
                updateQuery.setInt(3, myProduct.getPrice());
                updateQuery.setInt(4, myProduct.getStock());
                updateQuery.setInt(5, myID);
                updateQuery.execute();
            }
            catch (SQLException e){
                System.out.println(e.getMessage());
            }
        }

        public List<Product> getData(){
            List<Product> products = new ArrayList<Product>();

            try{
            String query = "select * from products order by id";
            rs = st.executeQuery(query);
            while(rs.next()){
                Product newProduct = new Product();
                newProduct.setID(rs.getInt("id"));
                newProduct.setName(rs.getString("name"));
                newProduct.setPrice(rs.getInt("price"));
                newProduct.setStock(rs.getInt("stock"));
                products.add(newProduct);
            }
            }
            catch (SQLException e){
                System.out.println(e.getMessage());
            }
            return products;
        }
```

In order to do operations on the database , SQL queries are needed to be built . This is why only the Data Access classes should have access to the database from the project .

The queries are build as prepared statements , and executed , with the only exception being the select * queries , which are executed on the database statement field . This yields a ResultSet type Object , which is then decomposed and added to the data according to the modeling of the table .

Layers

The most important aspect of a database application project are the layers . The first one is the presentation layer , which contains the graphical user interface and communicates with the user .

One level down there is the businessLogic layer , where everything is tested so that the inputs are always correct when they got to the database . It is also responsible for calculating needed information from the data , like the bill for a chosen order , so it has to link all tables together .

The third level is the intermediate one between the application and the MySQL database . It half in java and half in SQL language . This layer is responsible for translating from one language to the other . It is also considered as a repository , as all data has to go trough it to go from the user interface to the database or vice versa .

GUI

This holds the user interface . It helps the user use the program without understanding it . It is a crucial part of the project .

The user interface is extremely easy to use , with no unnecessary buttons or complications. The inputs for inserting or updating elements are in text fields and to avoid linking inexistent data , the ID's are chosen from combo boxes .

Further Development

The project meets the criteria for the given task , but some improvements can still be made . The program doesn't catch exceptions for incorrect input , like writing a character string in an integer field , and this result in an error , which is only visible in the console , and not in the user interface .

The project can also be developed on a much more complex database .

Bibliography

https://www.youtube.com/watch?v=BCqW5XwtJxY

http://stackoverflow.com/