Technical University of Cluj-Napoca

Programming Techniques

Laboratory-Assignment 3



Teacher: prof. Ioan Salomie

Teacher Assistant: Viorica Chifu

Student:Stioiu Denis-Adrian

Group:30424

1. **Objective**

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 should be structured in packages using a layered architecture presented in the support material and use at least the following classes:

1. Model classes: the data models of the application
2. Business Logic Classes: implement the application logic
3. Presentation Logic: implement the user input/output
4. Data access classes: implement the access to the database
5. **Problem analysis, scenarios, use cases**
6. General Overview

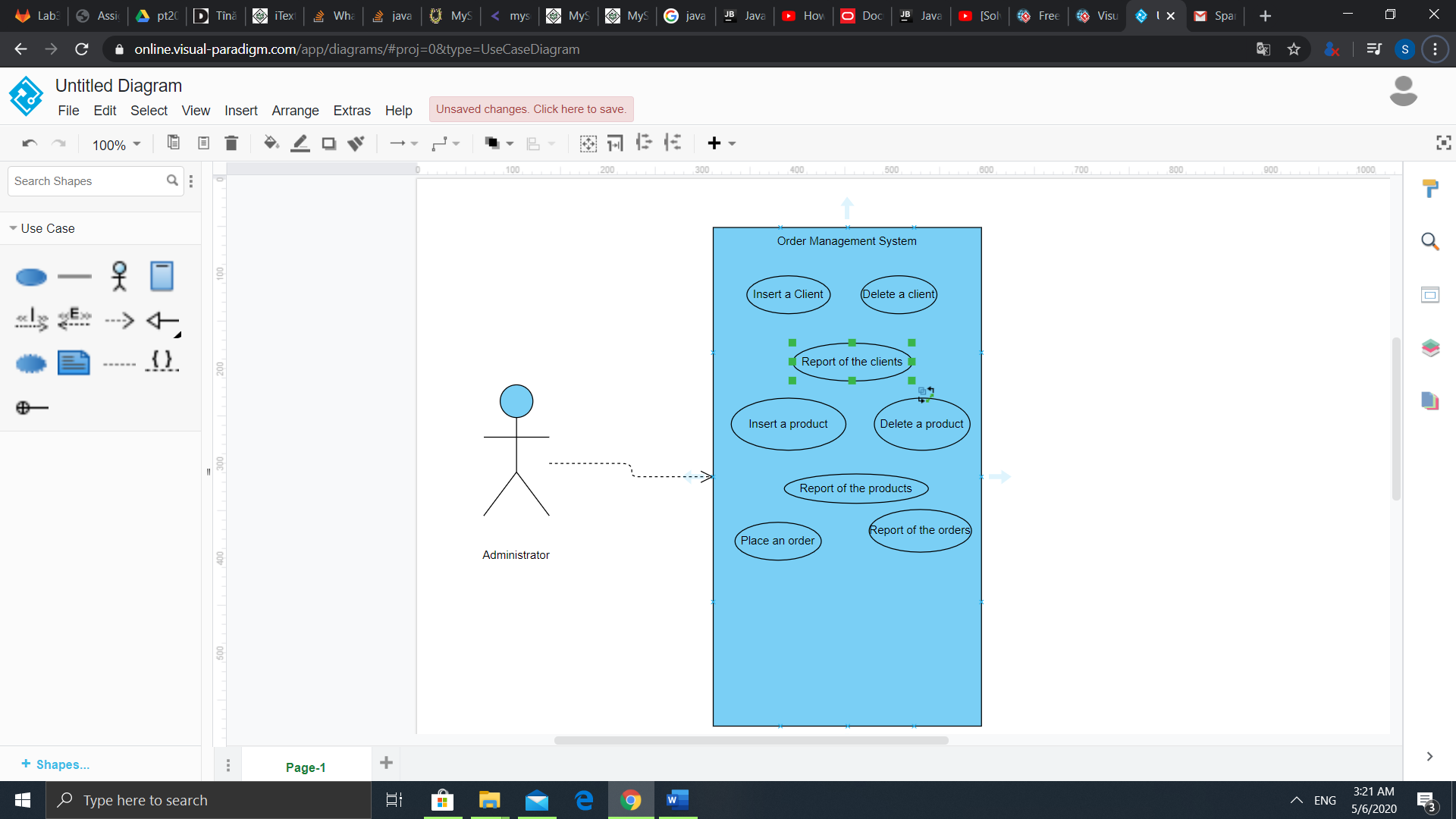
The application is able to display, modify and keep track of the clients, products, and orders. These are stored in a MySql database, along with the information about the users which have access to the system. This way, all the data is easier to retrieve and access from different computers.

1. Input and Output

The user can choose to manage four tables: Client, Product, Order, and OrderItem. All the tables have the options defined as the CRUD operations: **C**reate( insert and entry), **R**ead (show all entries), **U**pdate (update an entry), and **D**elete (delete an entry).

For example, in the Client table: the user introduces a name and address, which are checked using RegEx to be valid, and then inserts them in the table. An id of the client is auto incremented and generated by the table. The conditions for the input are that both the name and the address has to contain only letters, spaces and “-“.

1. Use Cases



* 1. Insert a client

The user can insert a new client by introducing the client’s name and address, and an id will be automated generated. If the name and address do not correspond to the valid format( they are composed only of letters, spaces and “-“) then an exception will be thrown. If the client is already present in the database, a notification will be sent to the user and the entry will be ignored. For this application we will consider that there cannot be two clients with the same name and address.

* 1. Delete a client

The user can delete a client form the table by using its name and address as arguments. If the client is not present in the table, than the operation will not take place.

* 1. Insert a product

The user can insert a product by introducing its name, quantity and price in the table, and its id will be automated generated. If a product with the same name is present in the database, then we will update only its quantity.

* 1. Delete product

For this application is considered that each product name is unique, and thus we can delete a product from the table only by typing its name as argument for the delete command

* 1. Report all clients

The user has the option to see all the entries in the Client table, displayed in a tabular form pdf. The pdf will contain the time when the report was generated in case its entries will be modified.

* 1. Report all products

The user has the option to see all the entries in the Product table, displayed in a tabular form pdf. The pdf will contain the time when the report was generated in case its entries will be modified.

* 1. Order a product

The user can process an order for the client. In the order will be specified the name of the client, his/her address, the name of the bought product, the quantity ordered, and the total price. In case the ordered quantity is greater than the available quantity, then the order will not be processed, and an alert pdf will be generated. If a client orders all the products, then that entry will be deleted. At the end of the order a bill will be generated.

* 1. Report order

The user has the option to see all the entries in the Order table, displayed in a tabular form pdf. The pdf will contain the time when the report was generated in case its entries will be modified.

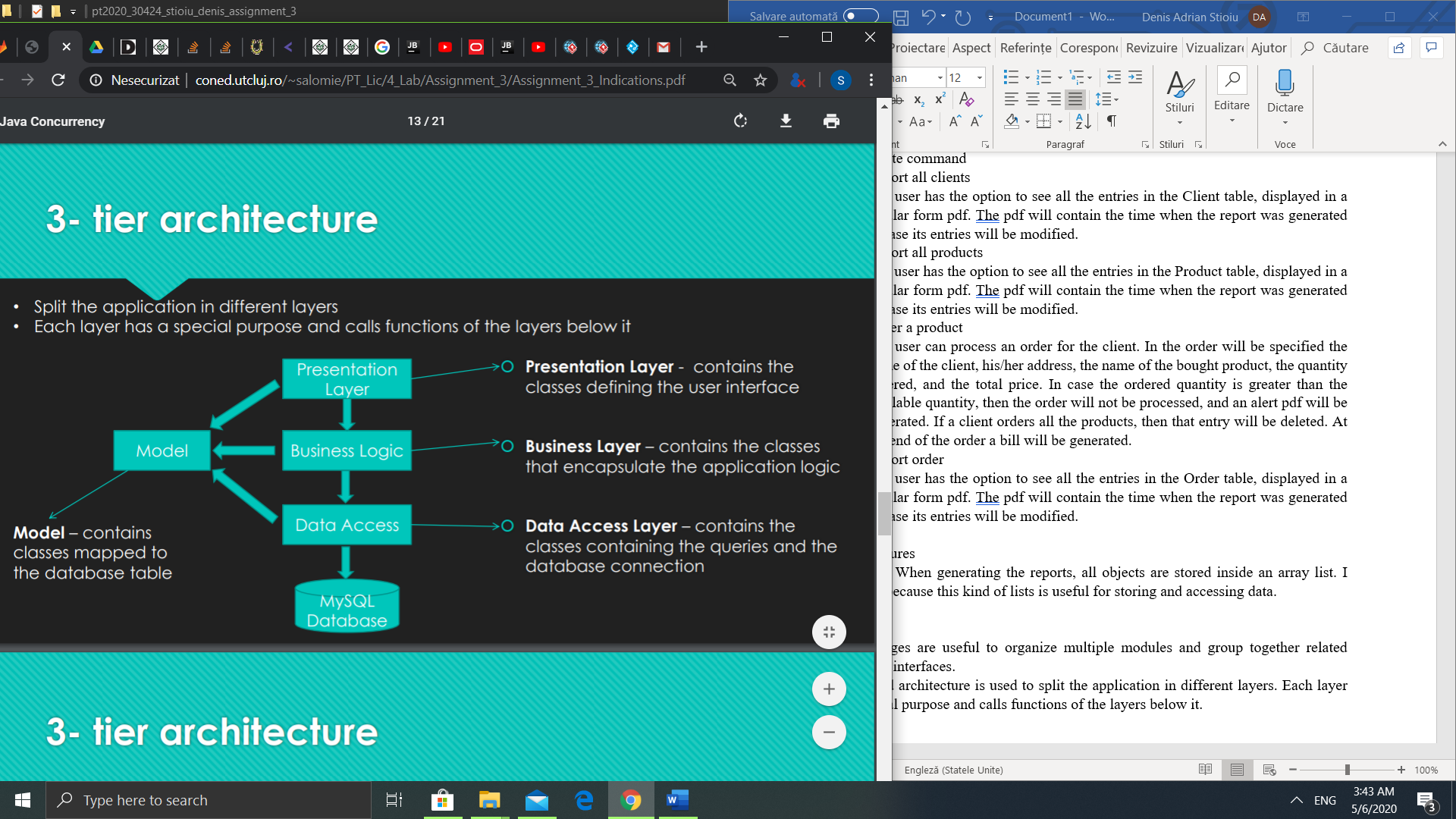
1. Data Structures

Array List: When generating the reports, all objects are stored inside an array list. I chose this because this kind of lists is useful for storing and accessing data.

1. Packages

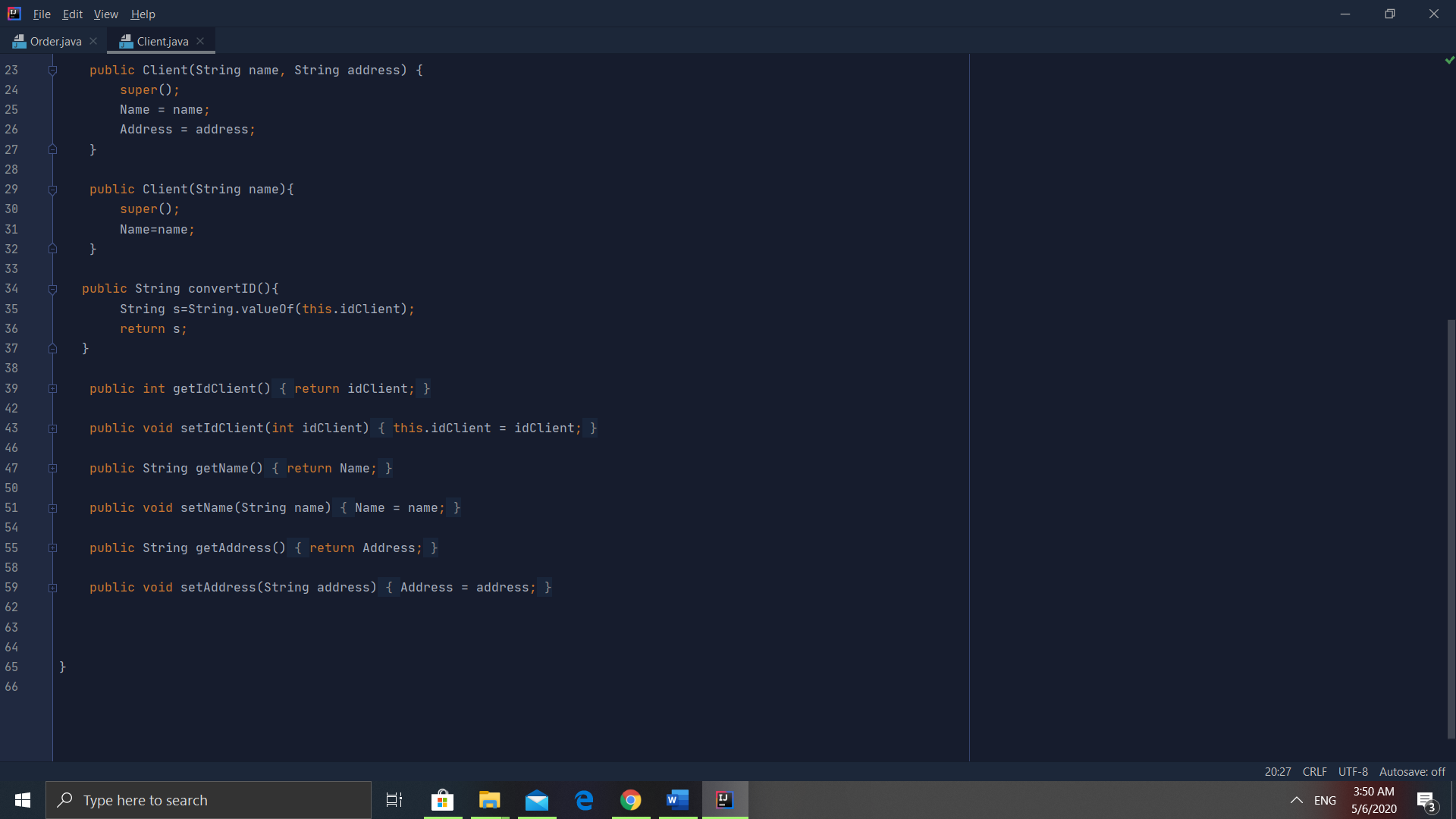
Java packages are useful to organize multiple modules and group together related classes and interfaces.

The layered architecture is used to split the application in different layers. Each layer has a special purpose and calls functions of the layers below it.



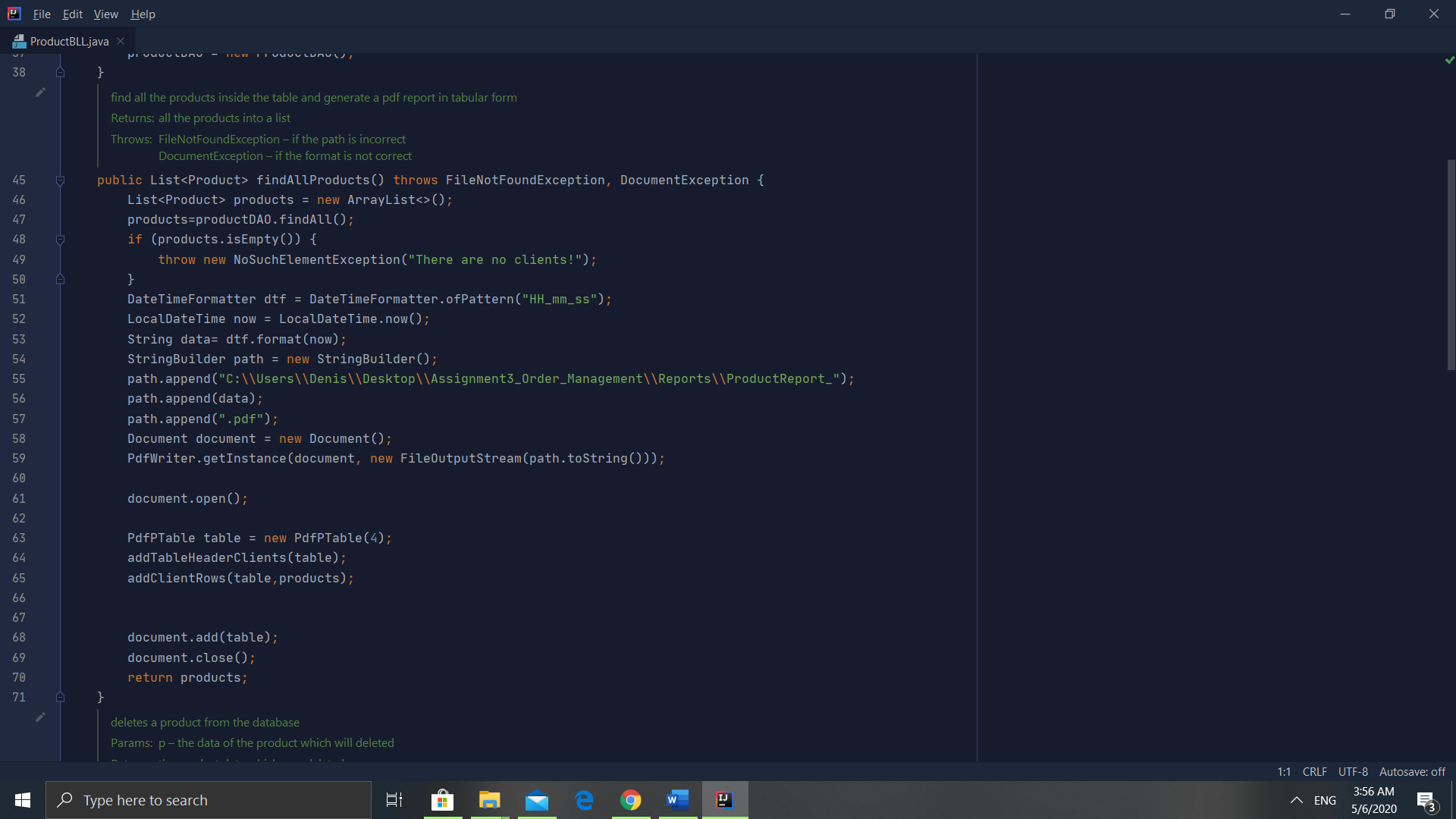
The model classes are the data models of the application. In this case, each class corresponds to table. This classes, called entities, are used to extract data from the DB table. The class must has its fields exactly the same type as the columns from the corresponding table. This classes also contain getters, setters and constructors. I added a method for each class, which I called converters that are used to convert the number types.

Example:



In the business logic classes I have the validators, a package with classes uses the validate the entries for insertion. These validators use regex and check that the names of both clients and products, and the addresses of the clients contain only letters. The products also have a validator to check that quantities and prices are numbers, and they are positive. In case the input is not valid an exception will be thrown.

The business classes contain the logic of the application, where the operations needed for the instructions to take place are made. For example, when reporting all the products inside the table:



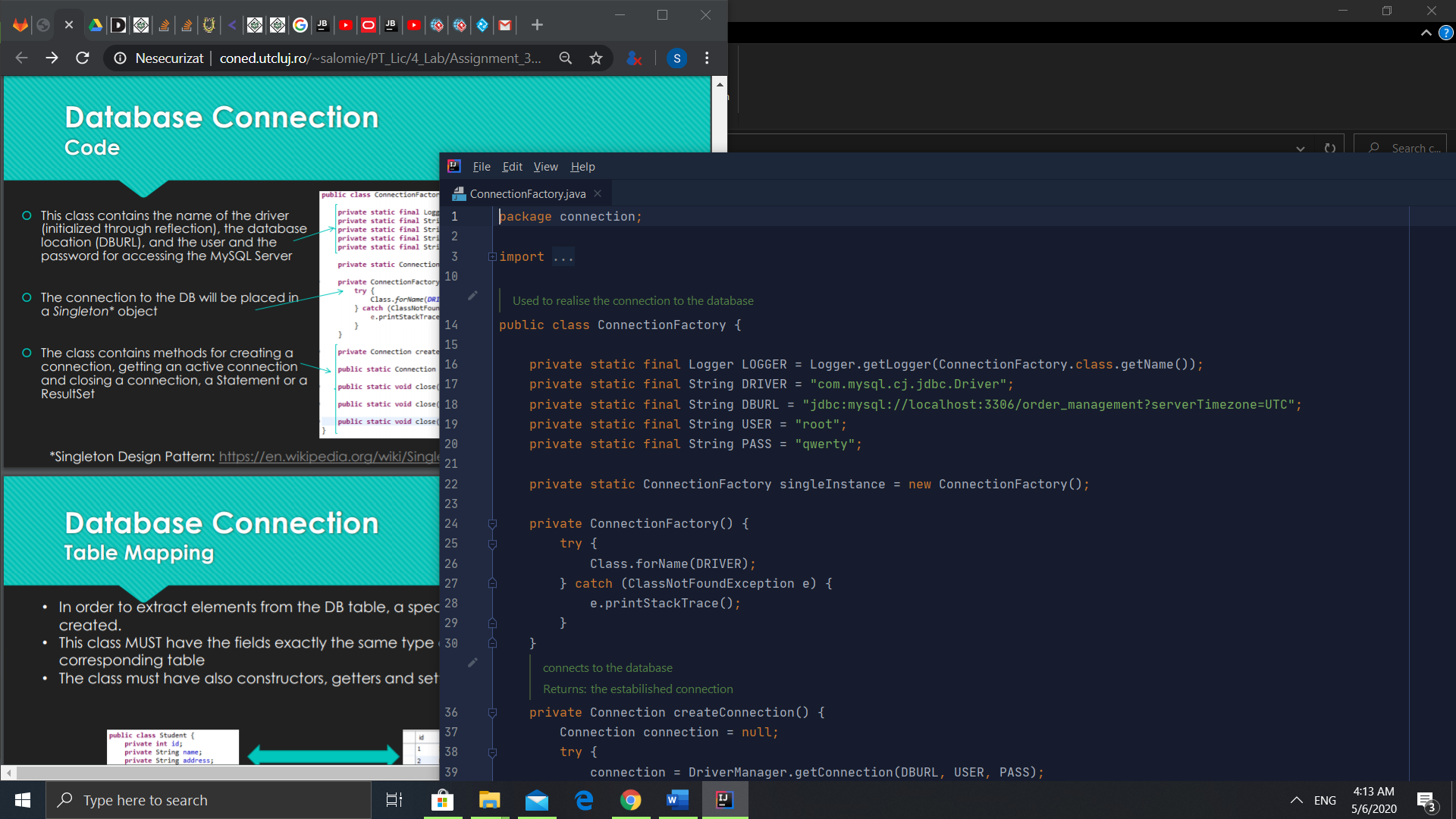
We initialize an empty array list that will be filled with product using the operation findAll. If the list is empty, meaning that there are no products in the table, a warning will be sent. The report will be called ProductReport\_HH\_MM\_SS, have in the name the exact moment the command was performed in order to differentiate reports realized in different times. The function addTableHeader will create the header of the table, with four columns in our case, and the title of each column will the field of the class. The function addClientRows add the data from the array list to the table.

The presentation classes in will process the input of the program. This is met in the Start package where the main class implements a parser to process the data, going through each line of the document, and implementing the operations from the business layer. The instructions from text file are conversed to lower case and splitted to extract the needed values. The code for the parser is the following one:

String inputLine = "";  
try {  
 Scanner sc = new Scanner(new File(args[0]));  
 while (sc.hasNextLine()) {  
 inputLine = sc.nextLine();  
 if (inputLine.toLowerCase().equals("report client")) {  
 List<Client> clients = clientBLL.findAllClients();  
 } else if (inputLine.toLowerCase().equals("report product")) {  
 List<Product> products = productBLL.findAllProducts();  
 } else if (inputLine.toLowerCase().equals("report order")) {  
 List<Order> orders = orderBLL.findAllOrders();  
 List<OrderItem> orderItems = orderItemBll.findAllOrders();  
 } else {  
 String[] command = inputLine.split(":");  
 if (command.length != 2) {  
 throw new IOException("Wrong input format!");  
 }  
 if (command[0].toLowerCase().equals("insert client")) {  
 String[] clientDetails = command[1].split(",");  
 if (clientDetails.length != 2) {  
 throw new IOException("Too many client details! Only the name and address please!");  
 }  
 Client c = new Client(clientDetails[0], clientDetails[1]);  
 clientBLL.insertClient(c);  
 } else if (command[0].toLowerCase().equals("delete client")) {  
 String[] clientDetails = command[1].split(",");  
 if (clientDetails.length < 1 || clientDetails.length > 2) {  
 throw new IOException("Wrong client format!");  
 }  
 if (clientDetails.length == 2) {  
 Client c = new Client(clientDetails[0], clientDetails[1]);  
 clientBLL.deleteClient(c);  
 } else {  
 Client c = new Client(clientDetails[0]);  
 clientBLL.deleteClientAfterName(c);  
 }  
 } else if (command[0].toLowerCase().equals("delete product")) {  
 String productDetails = command[1];  
 Product p = new Product(productDetails);  
 productBLL.deleteProduct(p);  
  
 } else if (command[0].toLowerCase().equals("insert product")) {  
 String[] productDetails = command[1].split(",");  
 if (productDetails.length != 3) {  
 throw new IOException("Too many/few product details! Only the name and address please!");  
 }  
 for (int i = 0; i < productDetails[1].length(); i++) {  
 if (productDetails[1].charAt(i) == ' ')  
 productDetails[1] = new StringBuilder(productDetails[1]).deleteCharAt(i).toString();  
 }  
 int quantity = Integer.parseInt(productDetails[1]);  
 double price = Double.parseDouble(productDetails[2]);  
 Product p = new Product(productDetails[0], price, quantity);  
 productBLL.insertProduct(p);  
  
 } else if (command[0].toLowerCase().equals("order")) {  
 String[] orderDetails = command[1].split(",");  
 if (orderDetails.length != 3) {  
 throw new IOException("Insufficient arguments!");  
 }  
 for (int i = 0; i < orderDetails[2].length(); i++) {  
 if (orderDetails[2].charAt(i) == ' ')  
 orderDetails[2] = new StringBuilder(orderDetails[2]).deleteCharAt(i).toString();  
 }  
 int quantity = Integer.parseInt(orderDetails[2]);  
 Product p = productBLL.findProductByName(orderDetails[1]);  
 OrderItem oi = new OrderItem(p.getIdProduct(), quantity);  
 Client c = clientBLL.findClientByName(orderDetails[0]);  
 Order o = new Order();  
 double price;  
 int idOrderItem;  
 if (p.getQuantity() < quantity) {  
 orderBLL.insufficientQuantity(c.getName(), p.getName(), p.getQuantity(), quantity);  
 } else {  
 oi = orderItemBll.insertOrderItem(oi);  
 idOrderItem = orderItemBll.findID(oi);  
 o.setIdClient(c.getIdClient());  
 o.setOrderItem(idOrderItem);  
 o = orderBLL.insertOrder(o);  
 price = p.getPrice() \* quantity;  
 orderBLL.generateBill(o, c.getName(), p.getName(), price, quantity);  
 p.setQuantity(- quantity);  
 if (p.getQuantity() == 0) {  
 productBLL.deleteProduct(p);  
 }  
 productBLL.updateProduct(p);  
 }  
 }  
 }  
 }  
} catch (IOException e) {  
 e.printStackTrace();  
}

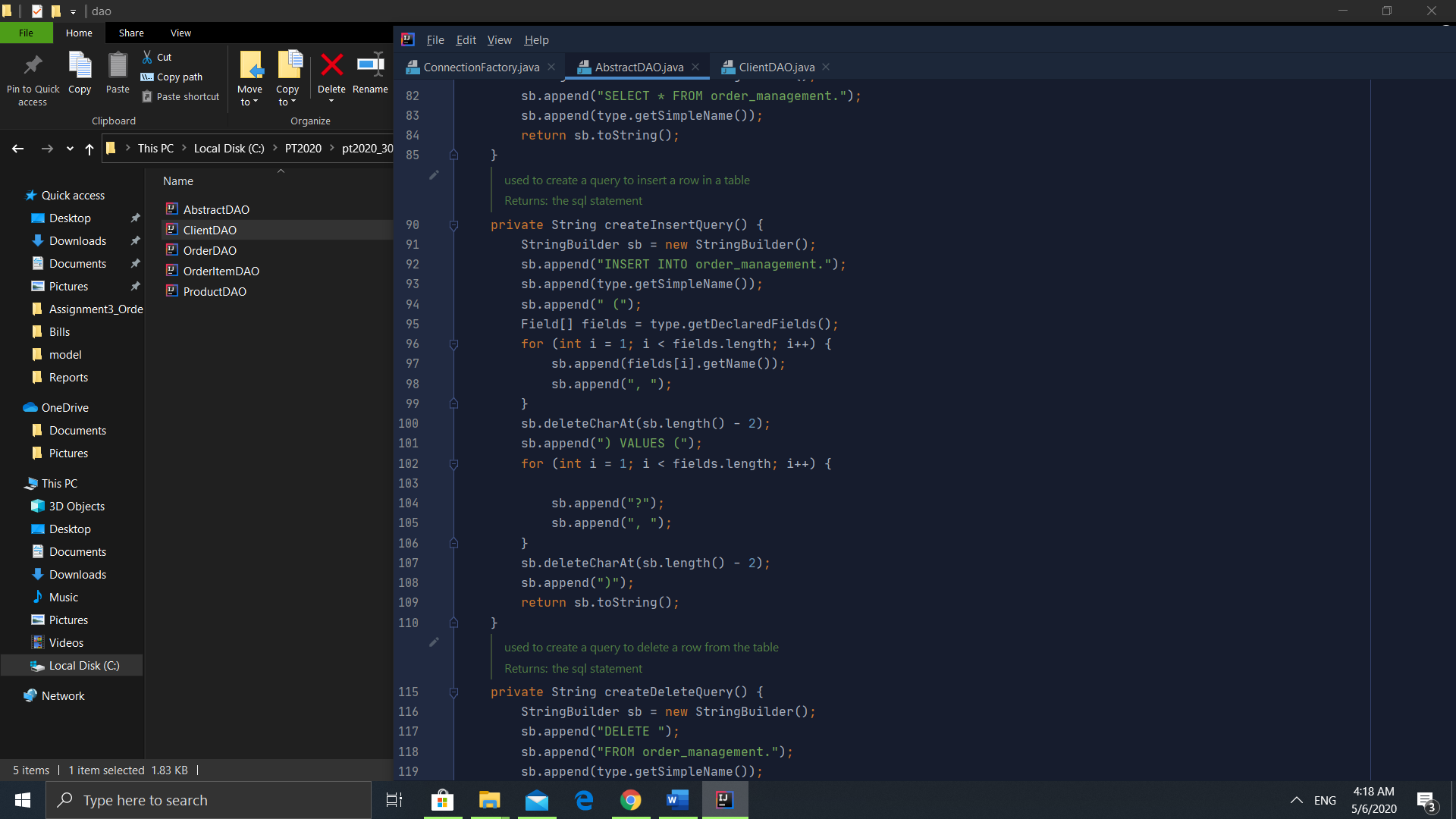
The data access layer contains the packages DAO and Connection are used to realize the connection with the database and MySql server. I used reflection techniques in order to create a generic class that contains the methods for accessing the DB. The Queries for accessing the DB for a specific object that corresponds to a table will be generated dynamically through reflection.

The Connection factory class contains the name of the driver( initialized through reflection), the database location( DBURL), and the user and the password for accessing the MySql Server. The connection to the DB will be placed in a Singleton object. The singleton pattern is a software design pattern that restricts the instantiation of a class to one single instance. It is useful when exactly one object is needed to coordinate across the system.

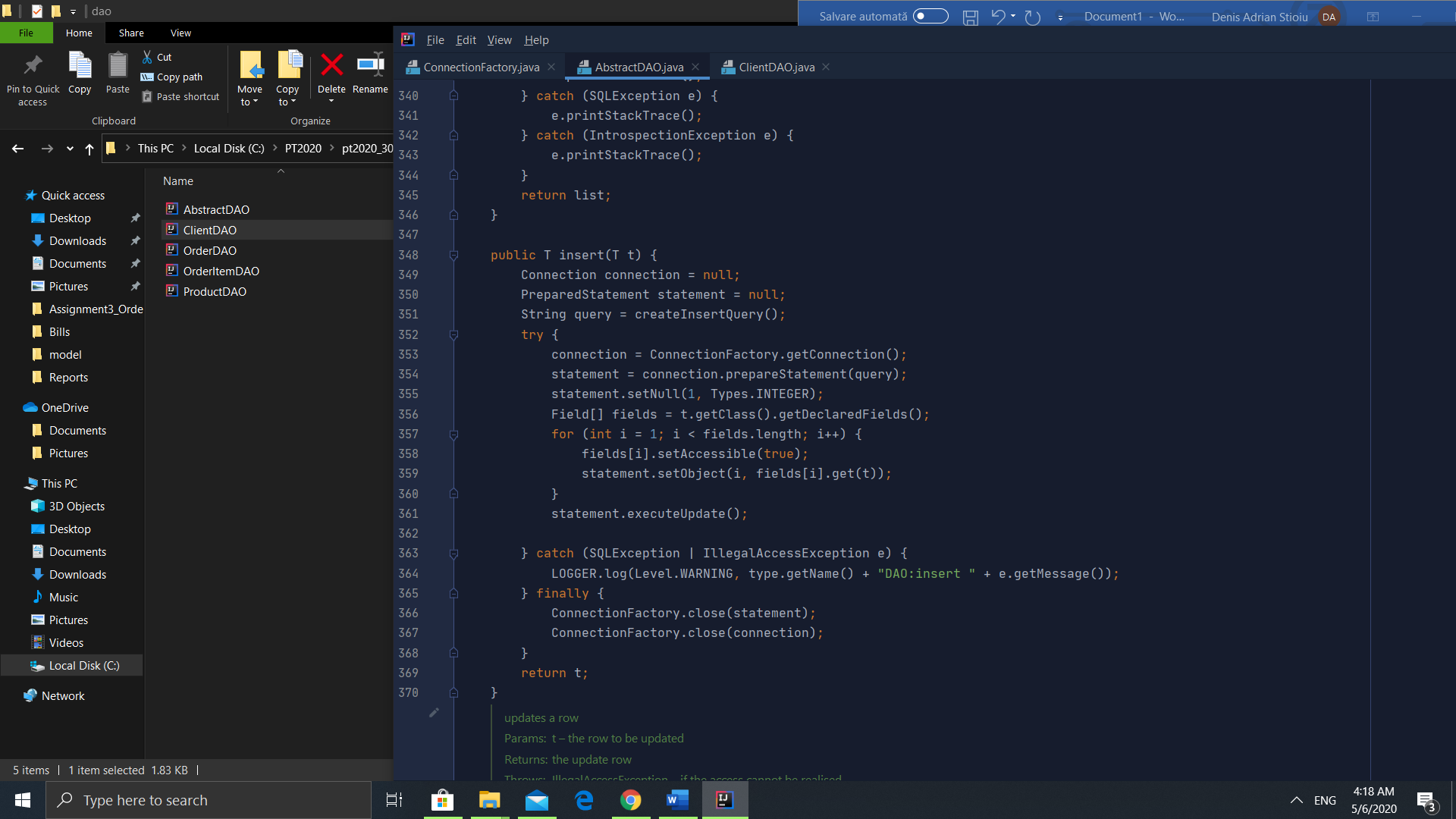


In the DAO package there are present the abstract class which creates the queries and the db instructions, and crud classes that implenement the instructions sent through reflection.

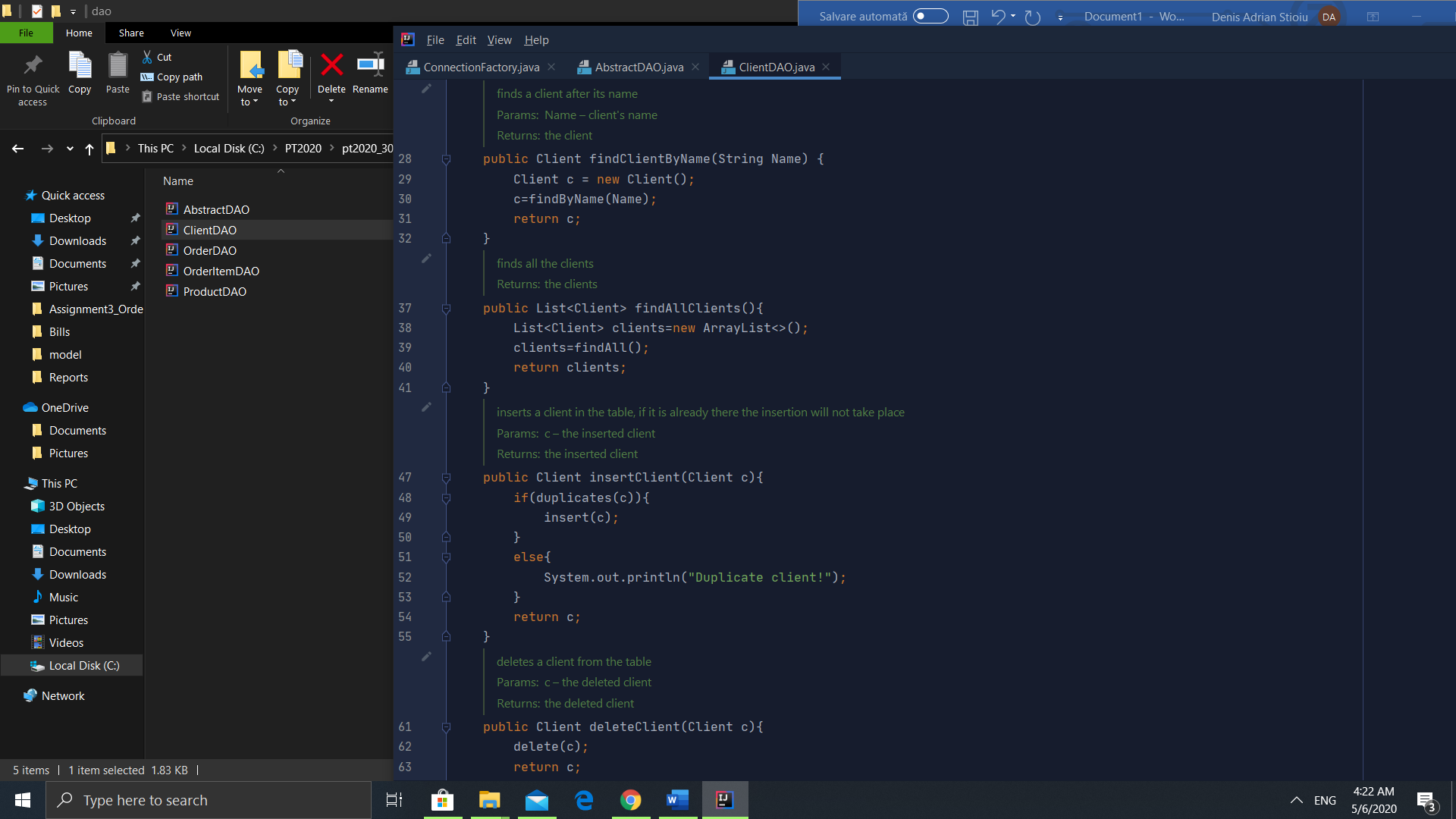
Example: The Insert Query in Abstract DAO:



And the insertion operation:



As we can see the query is generated, the we connect to the database, prepare the statement, and insert the element of T in the database. We complete the statement related to the fields of the class. For example for the class ClientDAO the insert query will be: INSERT INTO order\_management.Client (Name, Address) Values(?, ?) where these values will be received as an argument. The insertion in the ClientDAO looks like this:



If the client is already in the database, it will not be inserted.

1. Class Design

The idea of splitting the program into classes is based on the general rule of divide et impera.

* + 1. Start
       1. Start: Starts the application, and implements the parser
    2. Connection
       1. ConnectionFactory: Creates a statement in order to connect to the database. It is also responsible for closing the connection/ statements
    3. DAO
       1. AbstractDAO: contains generic methods to insert a row, delete a row, find the id of a row, find the name of the row, update a cell of the row, select all rows from the table.
       2. ClientDAO: extends the AbstractDAO and uses the generic methods to implement CRUD operations on the Client table
       3. ProductDAO: extends the AbstractDAO and uses the generic methods to implement CRUD operations on the Product table
       4. OrderDAO: extends the AbstractDAO and uses the generic methods to implement CRUD operations on the Order table
       5. OrderItemDAO: extends the AbstractDAO and uses the generic methods to implement CRUD operations on the OrderItem table
    4. Model
       1. Client: Models the client table from the database
       2. Product: Models the product table from the database
       3. Order: Models the order table from the database
       4. OrderItem: Models the orderItem table from the database, it is a table where the items and the quantities commanded are stored
    5. BLL
       1. Validators: Validate the inputs
       2. ClientBLL: Performs the instructions for the Client table, and generates a tabular form pdf when using the report command
       3. ProductBLL: Performs the instructions for the Product table, and generates a tabular form pdf when using the report command
       4. OrderBLL: Generates the bills, the understock warning, and the table for report, plus insertion( creating an order)
       5. OrderItemBLL: Inserts into the order item table, plus generates a pdf with tabular form for the report.

Each class and method is commented with javadocs, generated as html links, in order to help the user understand the code better.

1. **Conclusion**

This project was a good opportunity to learn how to work with databases using java, and to perfect my OOP skills. I learned about the layered architecture, how to use generics, the singleton pattern, and javadocs. I learned how to generate javadocs, in order to help to understand better the application. Working with creation of pdf was very interesting as it was my first time, and I find this new knowledge very useful. Also, I remembered the SQL and Database information from the first semester.

1. **Bibliography**

Materials offered during the courses and the laboratories

<https://www.baeldung.com/java-pdf-creation>

<https://www.baeldung.com/javadoc>