**Order Management**

***-Project documentation-***

Student: Cocoş Tudor-Adrian

Group: 30423

**Contents:**

1. Objective of the assignment
2. Problem analysis
   1. Assumptions
   2. Modelling
   3. Scenarios
   4. Use cases
3. Design
   1. Design choices
   2. UML diagram
   3. Class design
   4. Algorithms
   5. User Interface
4. Implementation
5. Testing
6. Results
7. Conclusions
8. Bibliography

**1) Objective of the assignment**

The main objective of this assignment is to design and implement an application for processing customer orders for a warehouse, using relational databases to store de products.

The secondary objectives are the steps needed to be taken to achieve the main objective. First of all, we need to analyse the concept of warehouse management and what that implies in terms of individual entities such as products, clients and orders. After that we need to break down the problem to implementable features like classes and methods needed. The next steps include class design, method implementation, User Interface development and testing. Finally, the results are evaluated, and we can think of further improvements that can be made to the existing project.

**2) Problem analysis**

*a) Assumptions*

Given the assignment, we assume that the application should have multiple views, one for each type of entity (clients, products, orders) with common types of actions that can be taken for every one of them for the sake of simplicity.

*b) Modelling*

To make the data as simple as possible there are a few fields to describe each entity such as name, address, email, mobile phone number and age for each customer, a name, quantity and price for each product, while an order is defined by a client id, product id, the desired quantity of that product and the value of the order which is given by the quantity ordered multiplied by the price of the product.

*c) Scenarios*

The main scenario is when it is only deal with the appropriate data and when there are no logical or data integrity problems that might be faced. This includes making no mistakes when inputting the fields of any entity for any type of action that is performed on the respective entity. As a result, the right data will be placed and represented in the correct spot at all times.

Other scenarios are the ones that begin with the “what if?” question after deciding on the main scenario. “What if … the inputted phone number is shorter or longer than 10 digits?”, “What if … the inputted price of the product is less than 0?”. “What if … the inputted order quantity exceeds the quantity of the product that is available in the warehouse?”. All these scenarios might come up and we need to ensure of the quality of the data that we want to send and store in the database as to not have faulty data later when we have to work with it: having less stock or none of the product that was ordered and not being able to honour the request of a client is not a desired scenario to deal with.

*d) Use cases*

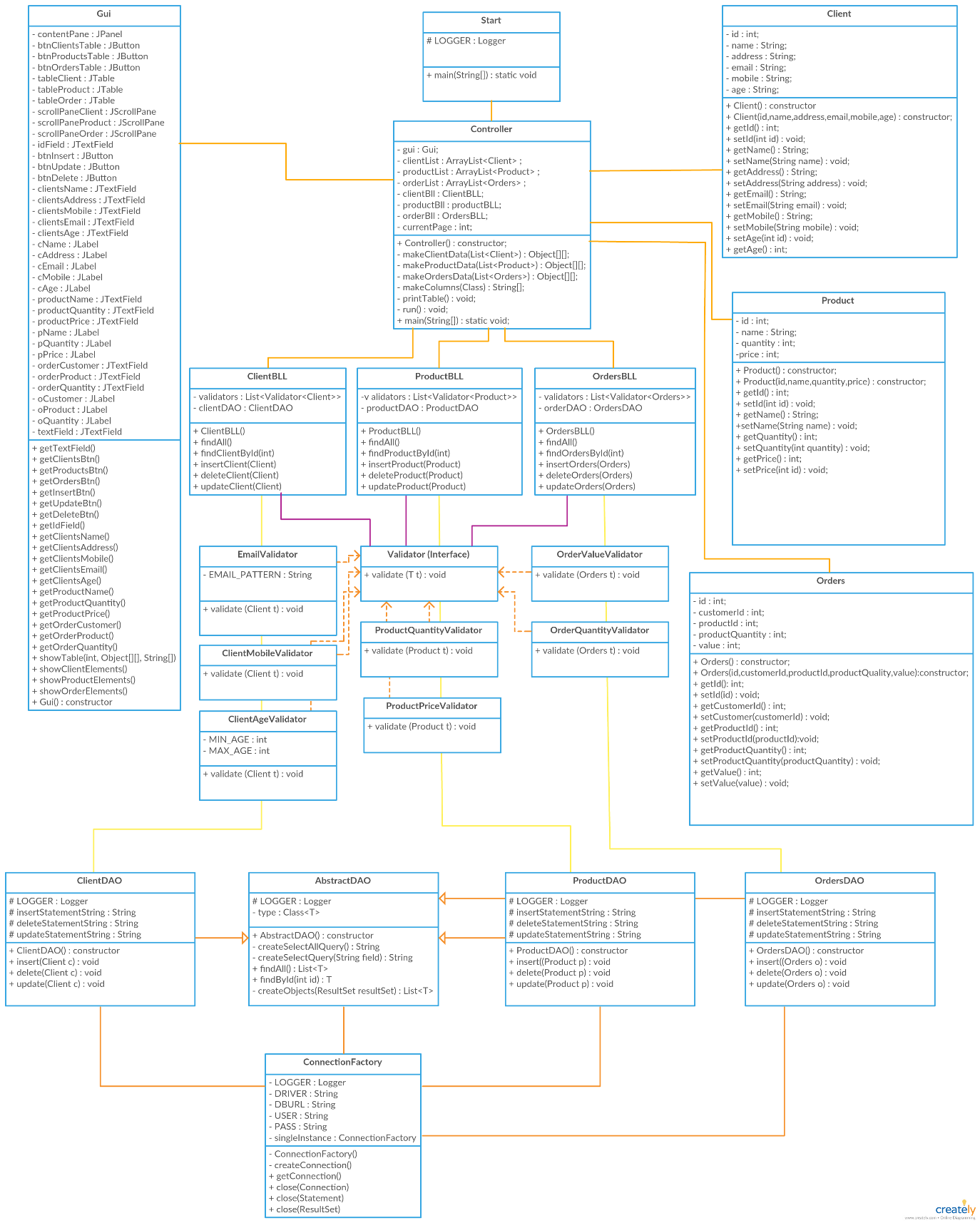
The application may be used to input the personal data of a new client, updating some of his personal data or, if he or she is no longer a client of the warehouse, to be able to delete his account. When considering the products of the warehouse, the user should be able to insert a new product to the database, update its stock or price, or even its name if it has to and finally to delete the product if that product is no longer sold by the company. As far as the orders go, one should be able to make a new one and insert it in the database. By doing so, the quantity of the chosen product in the warehouse should be lowered by the quantity that was ordered. The placed order should be able to be deleted and the product of that ordered to be placed back in the warehouse, restocking the remaining products of that type. An order, however, should not be updated, in my opinion, because when we think in real life terms, having ordered a certain quantity of a certain good means that a receipt has been issued with that certain amount and value, if it implies delivery that amount of the product is being packaged and delivered and no one should be able to modify an order that has been made. If the client or the seller decides to shut down a deal, then the order should be deleted, and a new order should be made.

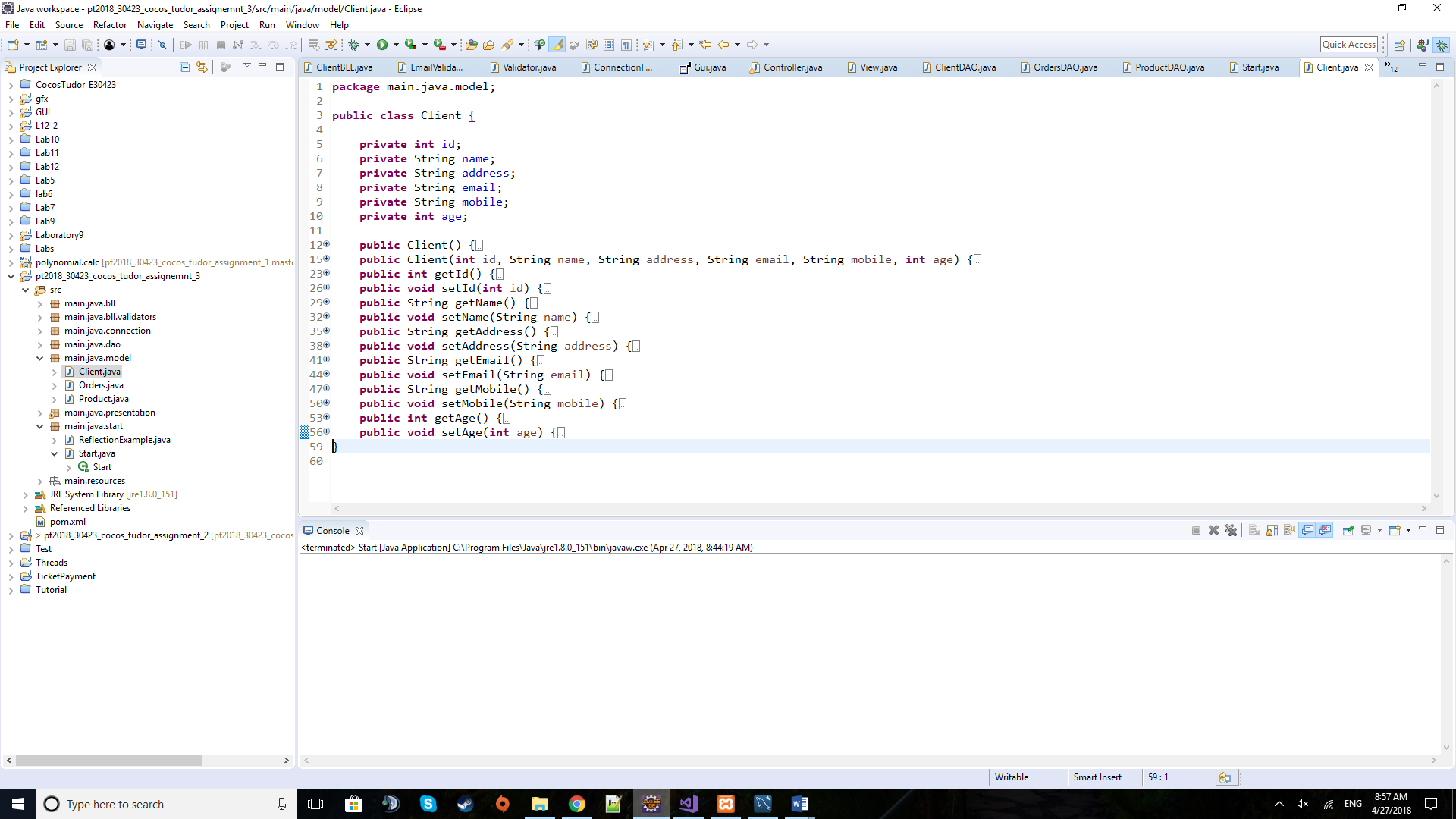
**3) Design**

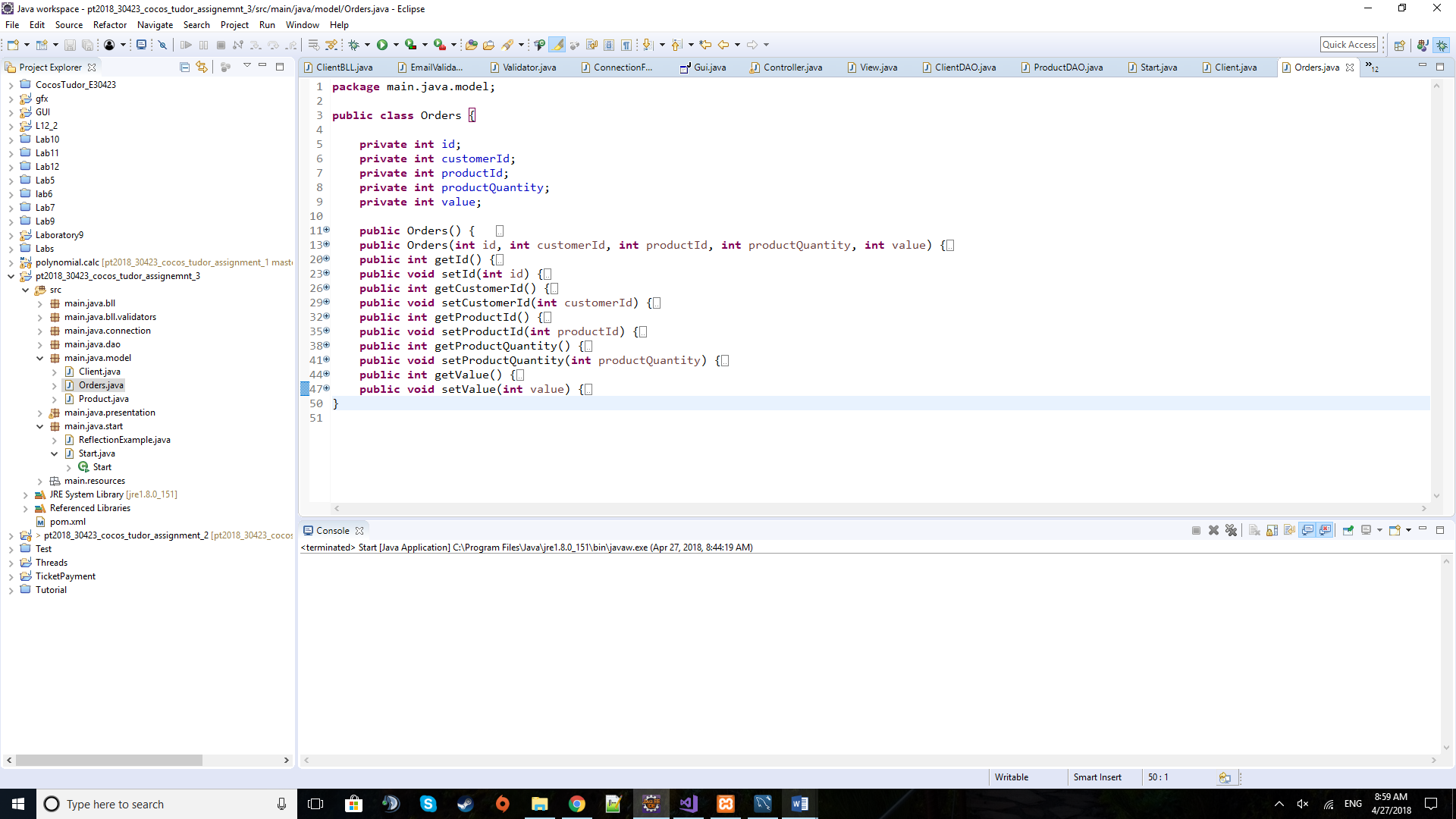
*a) Design choices*

The design of the application starts with the entities that need to have both a database representation and also a class representation in our program. These entities are the Clients, Products and Orders data tables and classes with their according table columns or fields. As presented in the Data Modelling section, each client should have an id, a name, an address, an email, a mobile phone number and an age to better differentiate between them. A product consists of an id, its name, the quantity available in the warehouse and its price. The order should be identified by its id, the id of the customer and product, the quantity of the ordered product and a value given by the quantity that was ordered, multiplied by the price of the product. These classes constitute the Model classes of our application and are followed by the Business Logic classes, which are the ones that deal with their respective model counterparts. Here there are also some validators defined such as: price validator (in order not to have negative prices), age or email validators, etc. They are used to validate the operations that the user wants to perform and the input that he or she wants to input. If everything is alright the data is sent to the Data access classes that interact with the database by means of the SQL queries. All of the actions performed by the application are being driven by the ones of the user so he or she should always have a representation of the tables that are found in the database with a live feed of their content. The Controller class of this layer is the one managing the components of the GUI and the actions that they are responsible for. It will also print the receipt once a new order has been issued.

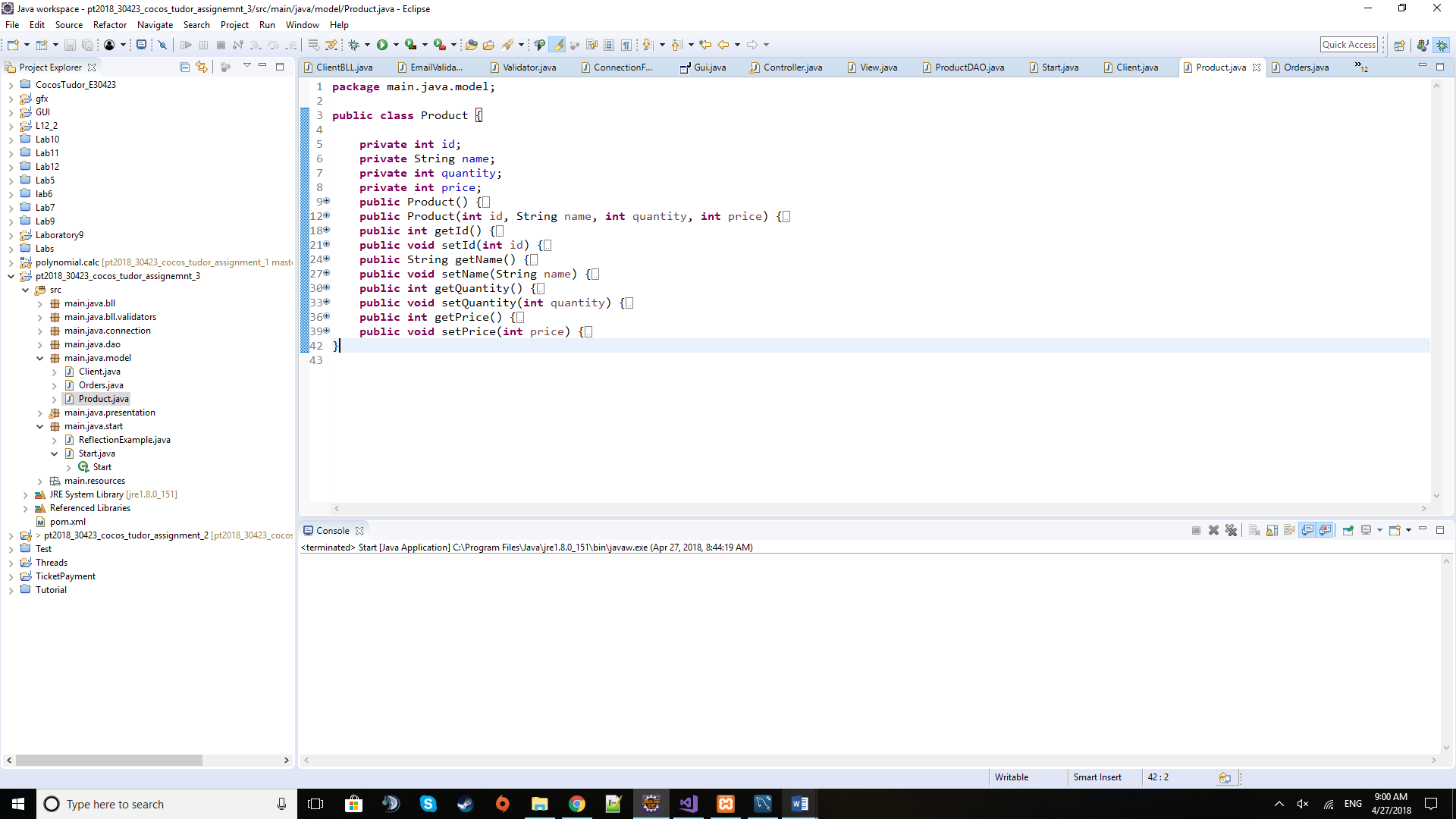
*b) UML Diagrams*

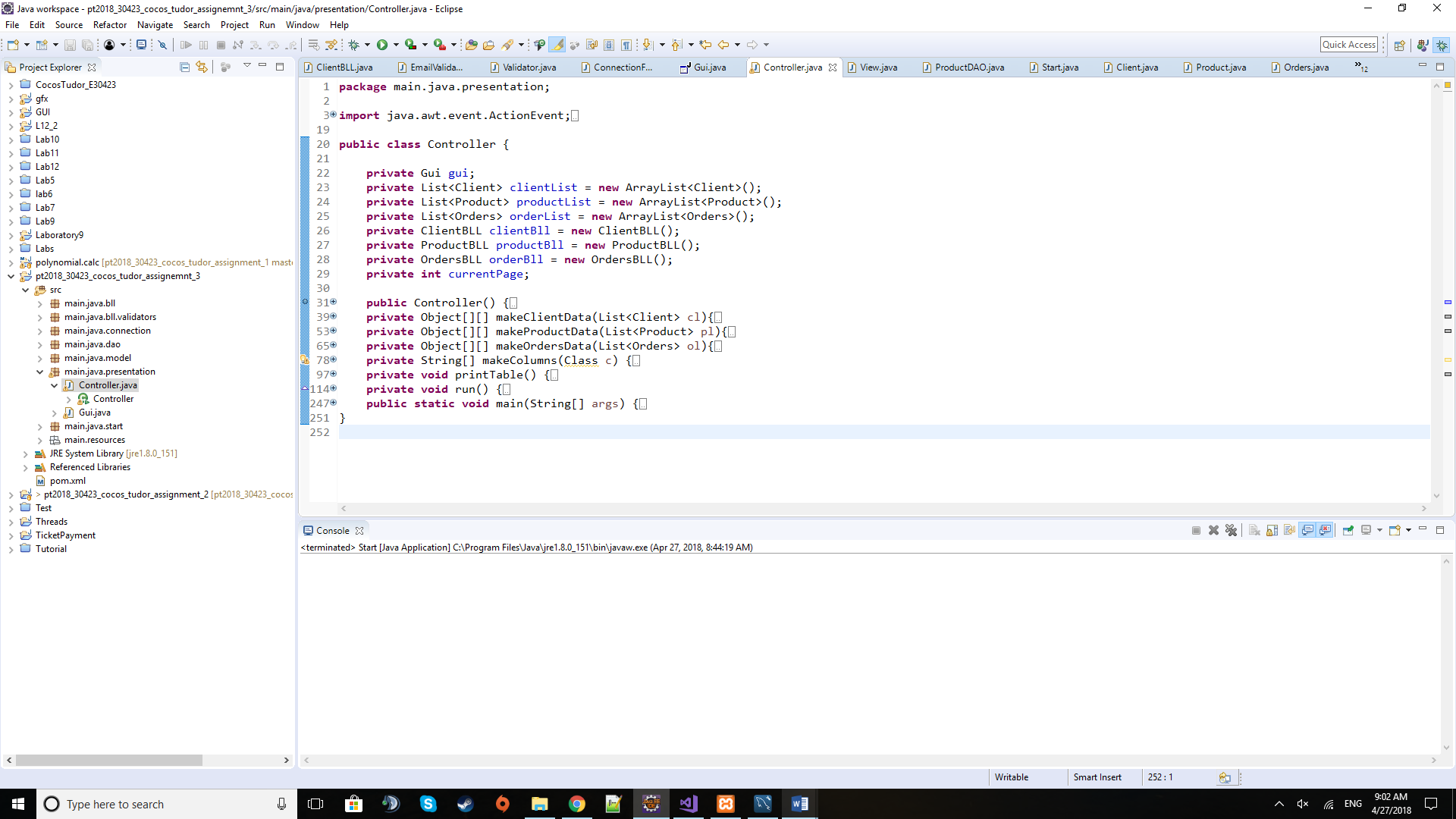
**

*c) Class design*

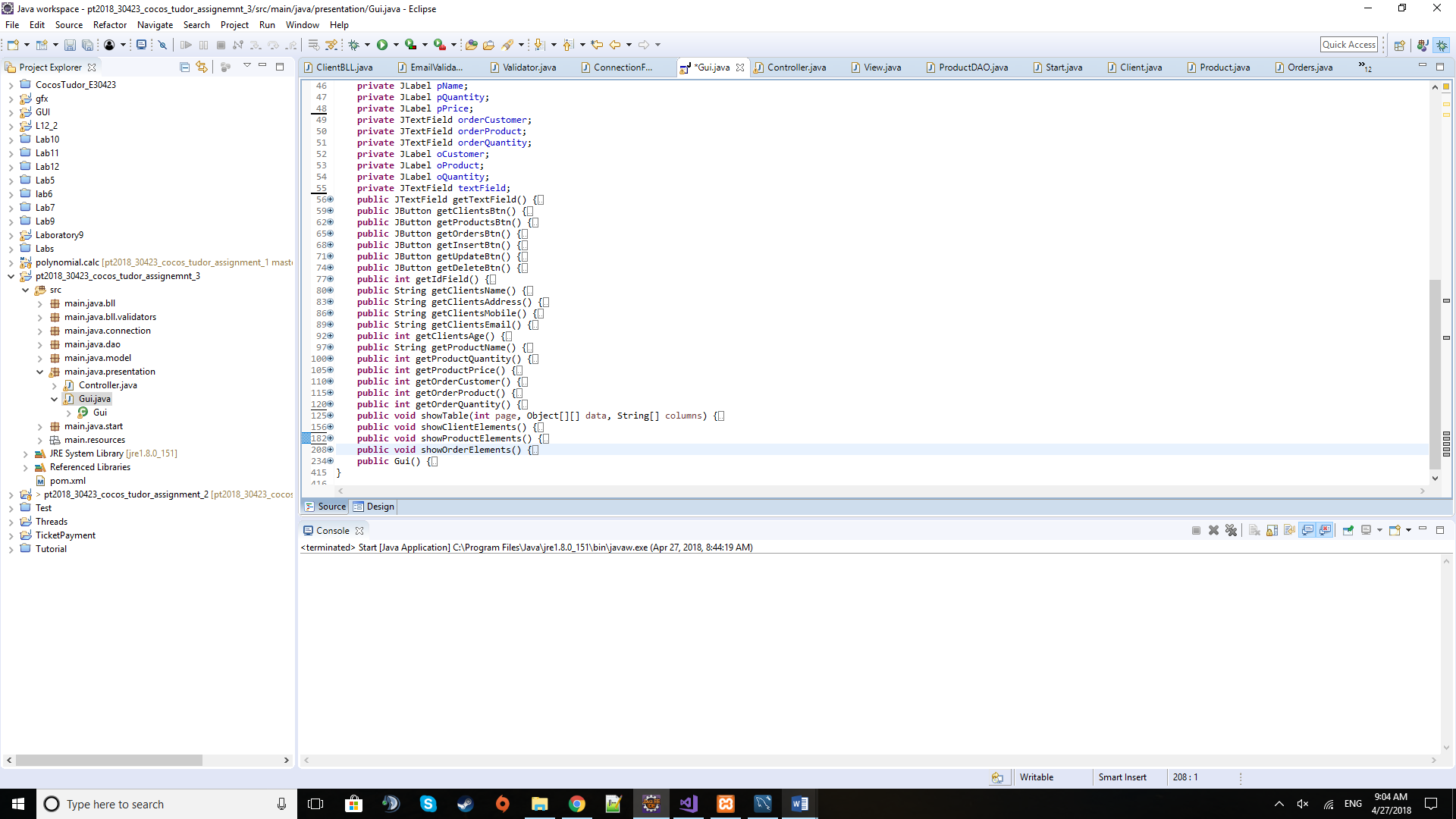
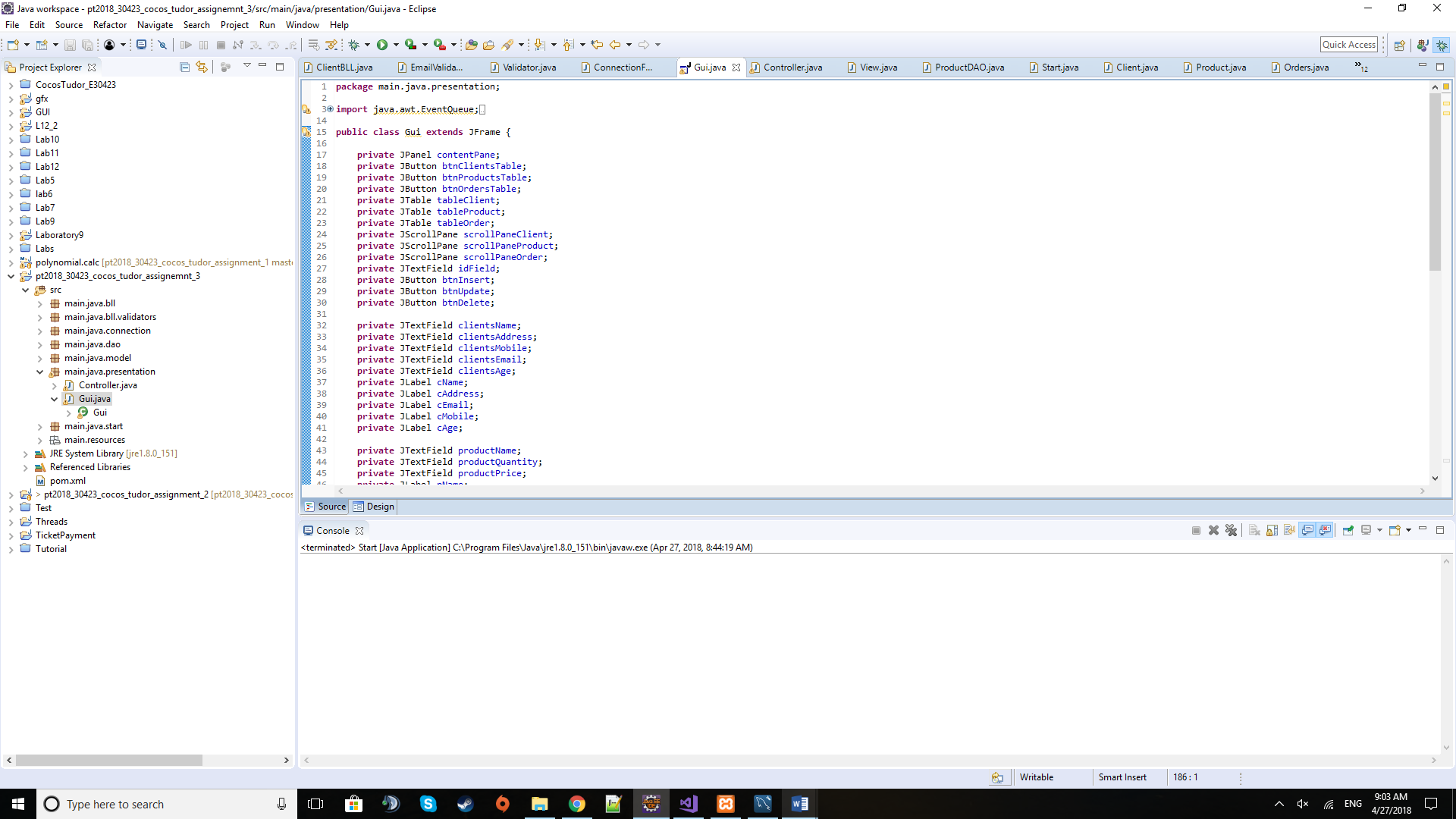
***Client class***

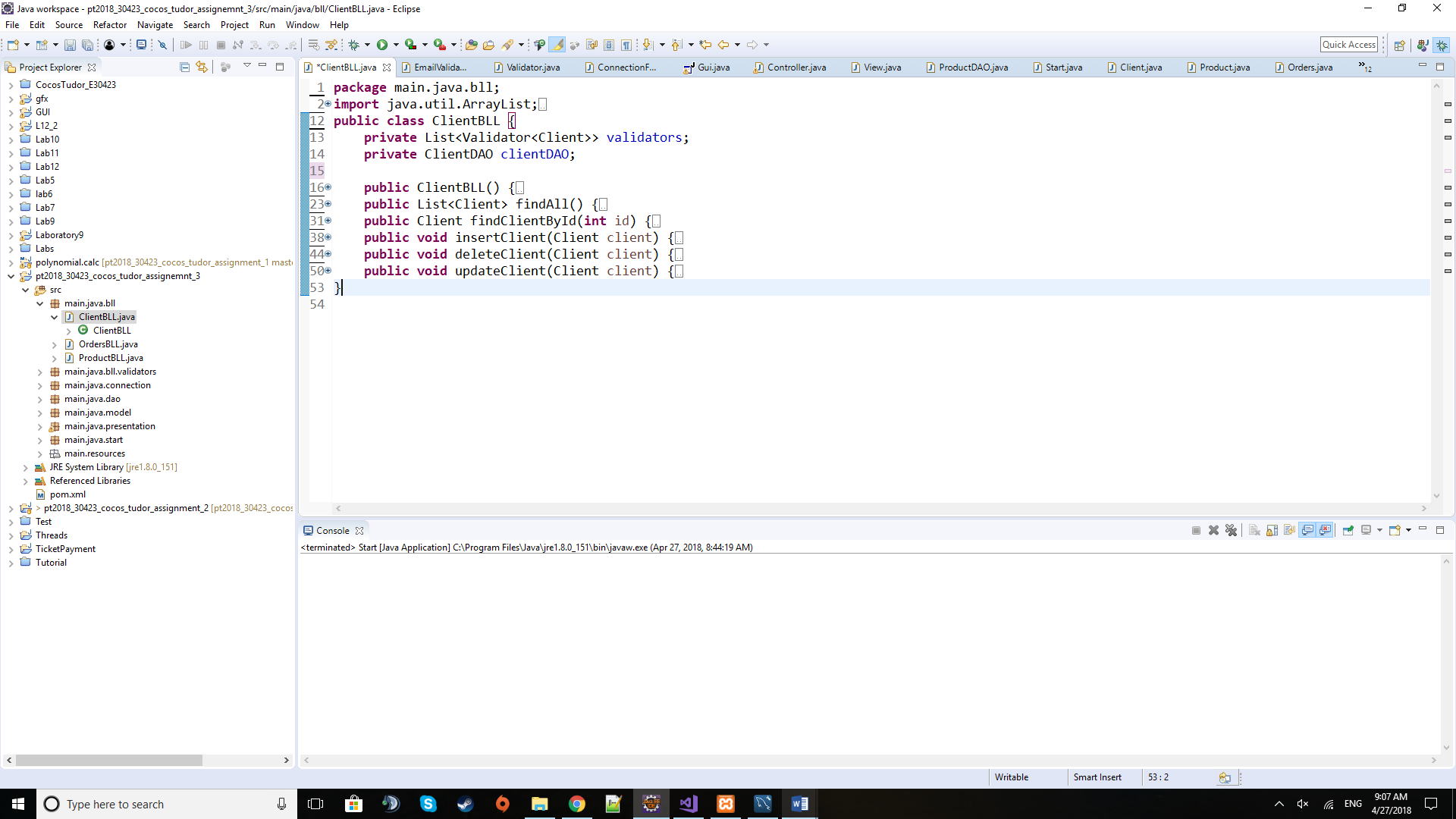
***Orders class***



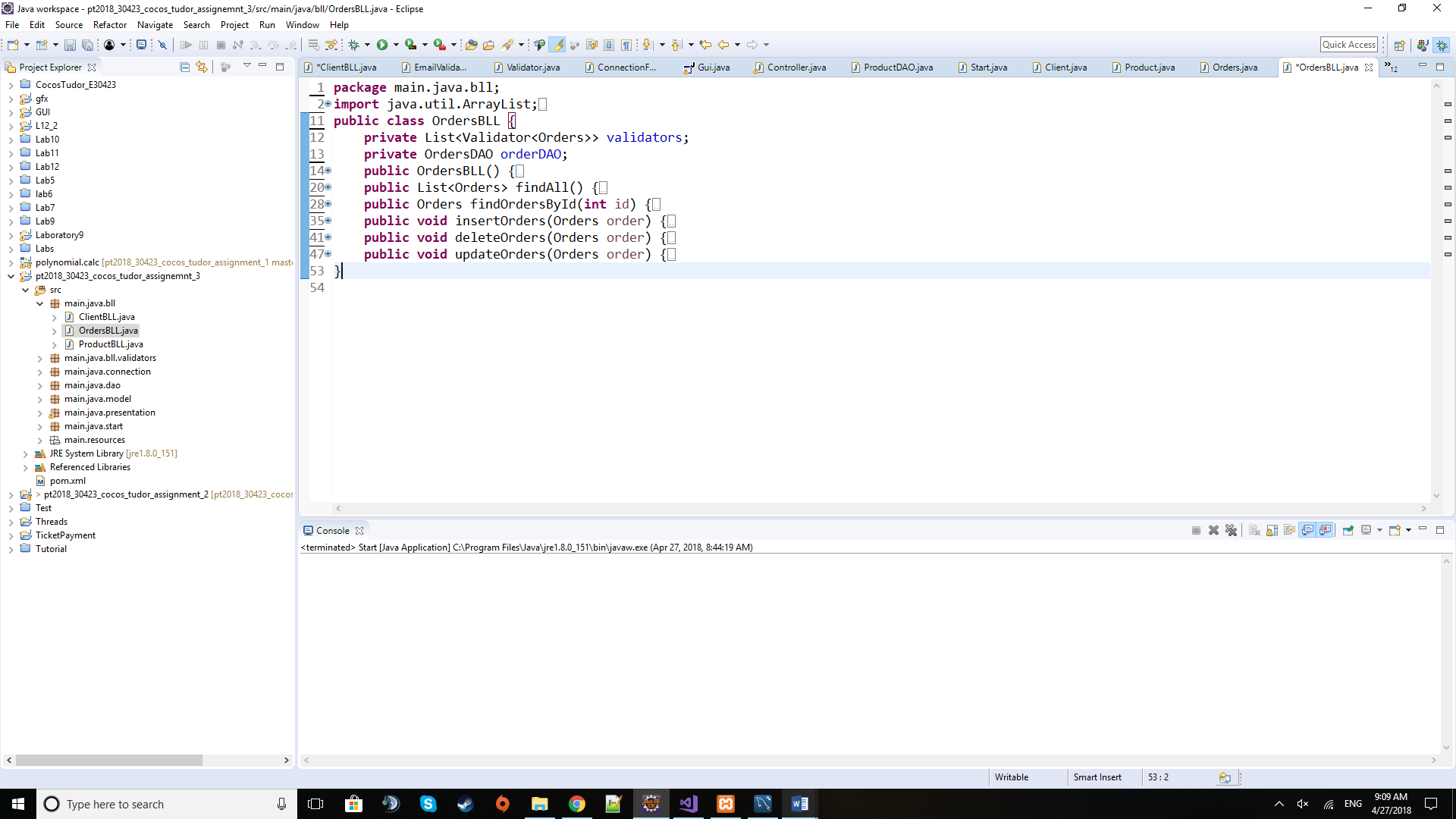
***Product class***

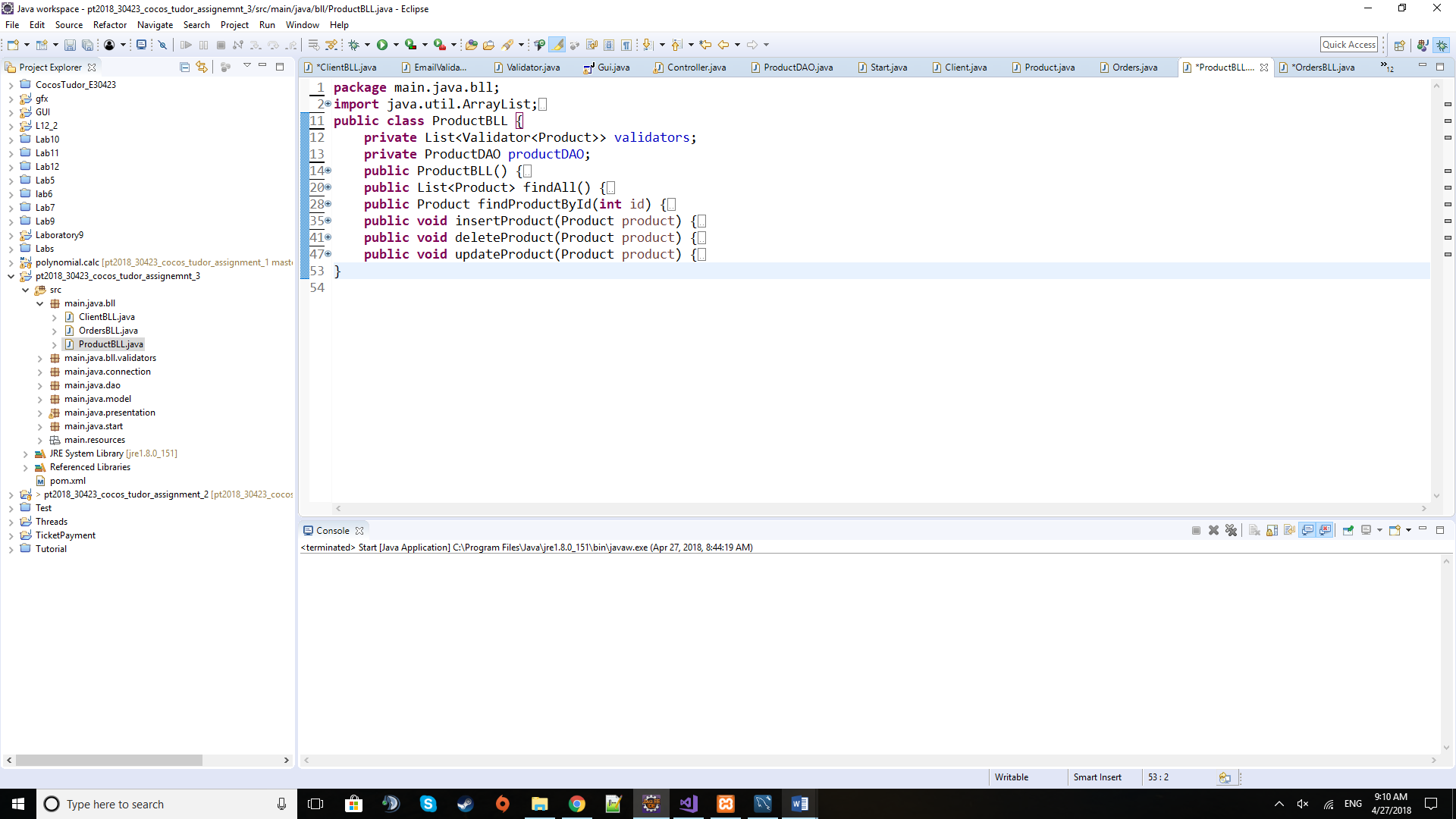
***Controller class***



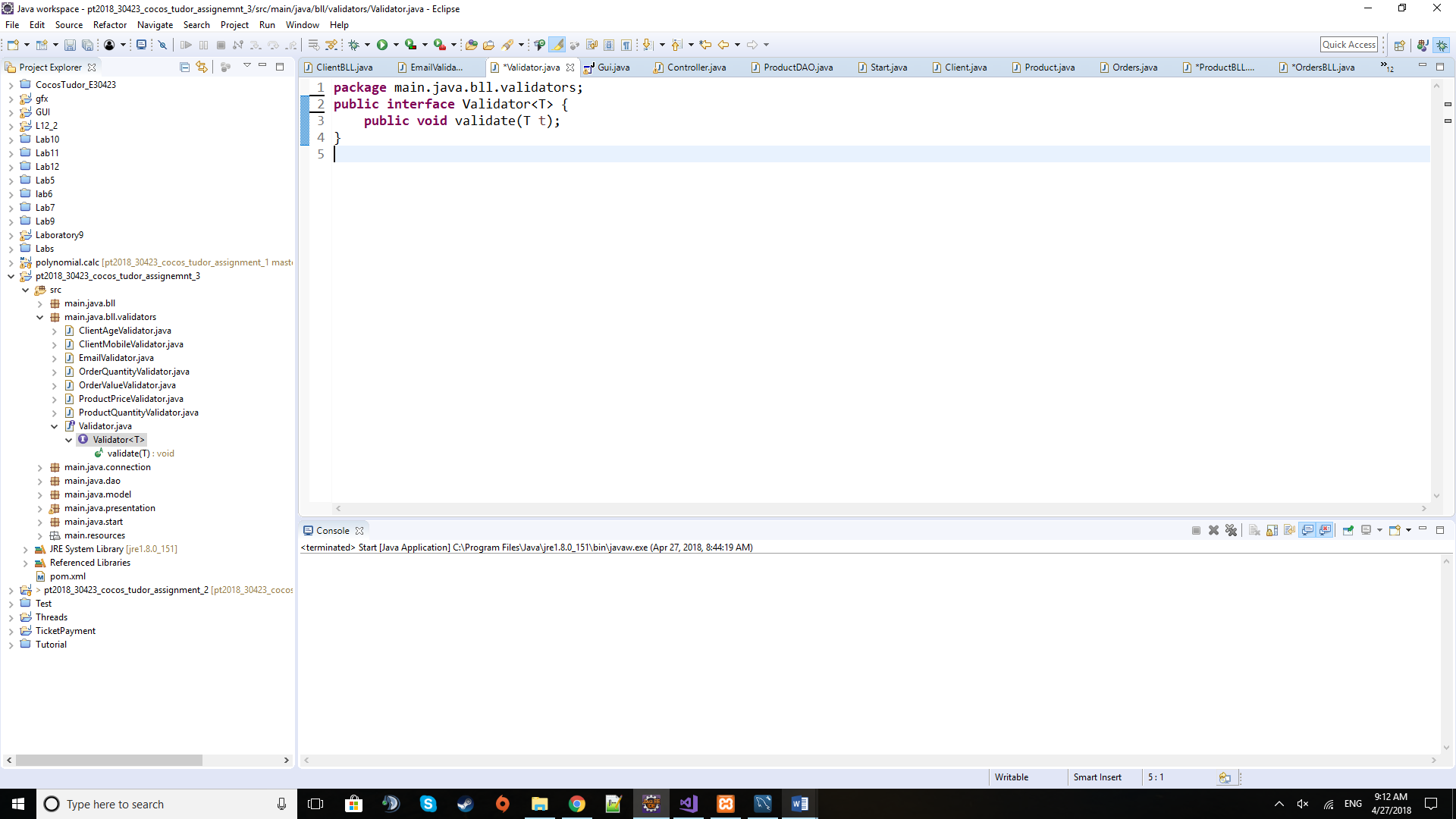
***GUI class***

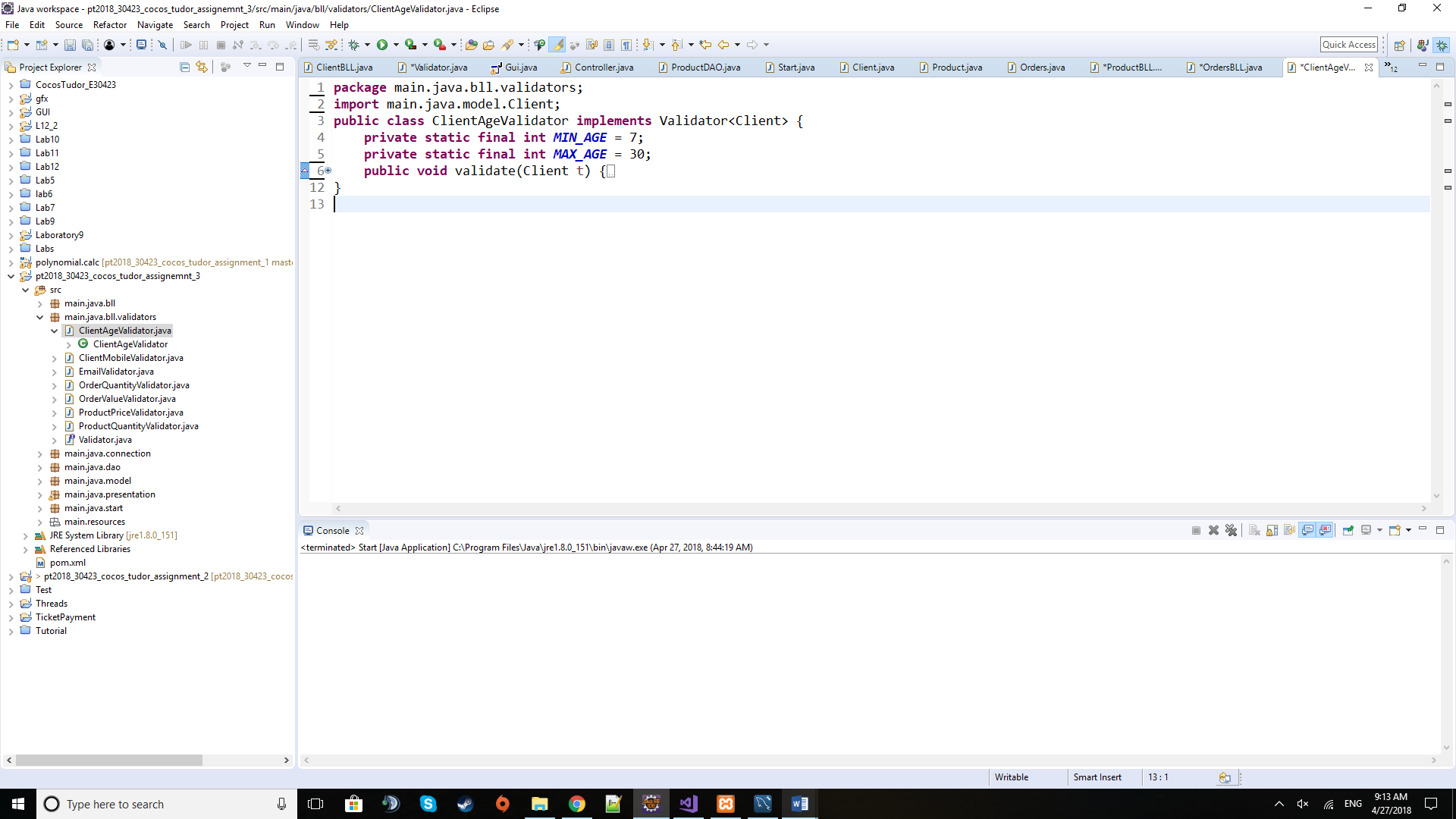
***Client Business Logic Layer (ClientBLL) class***



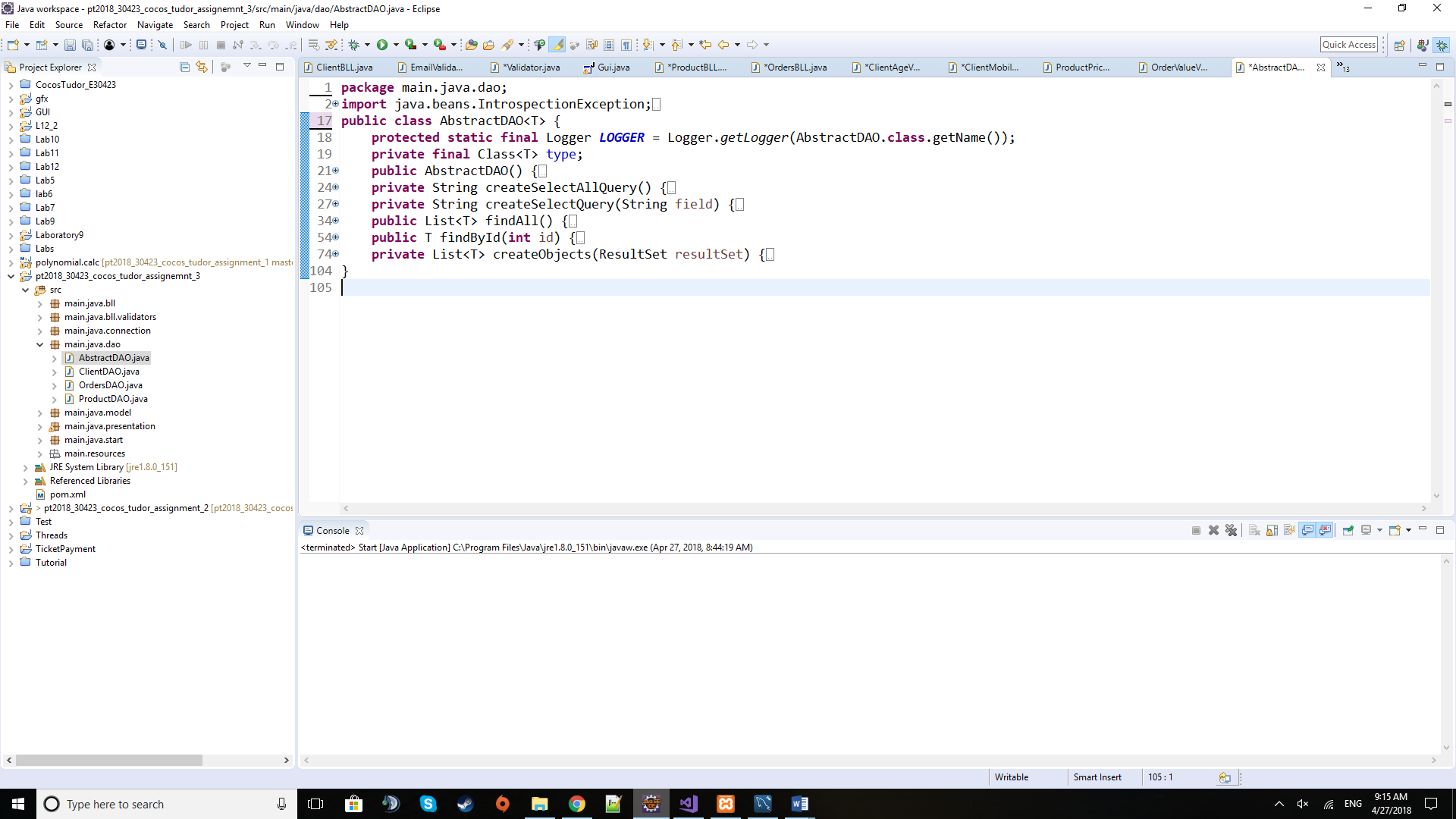
***Orders Business Logic Layer (OrdersBLL) class***

***Product Business Logic Layer (ProductBLL) class***

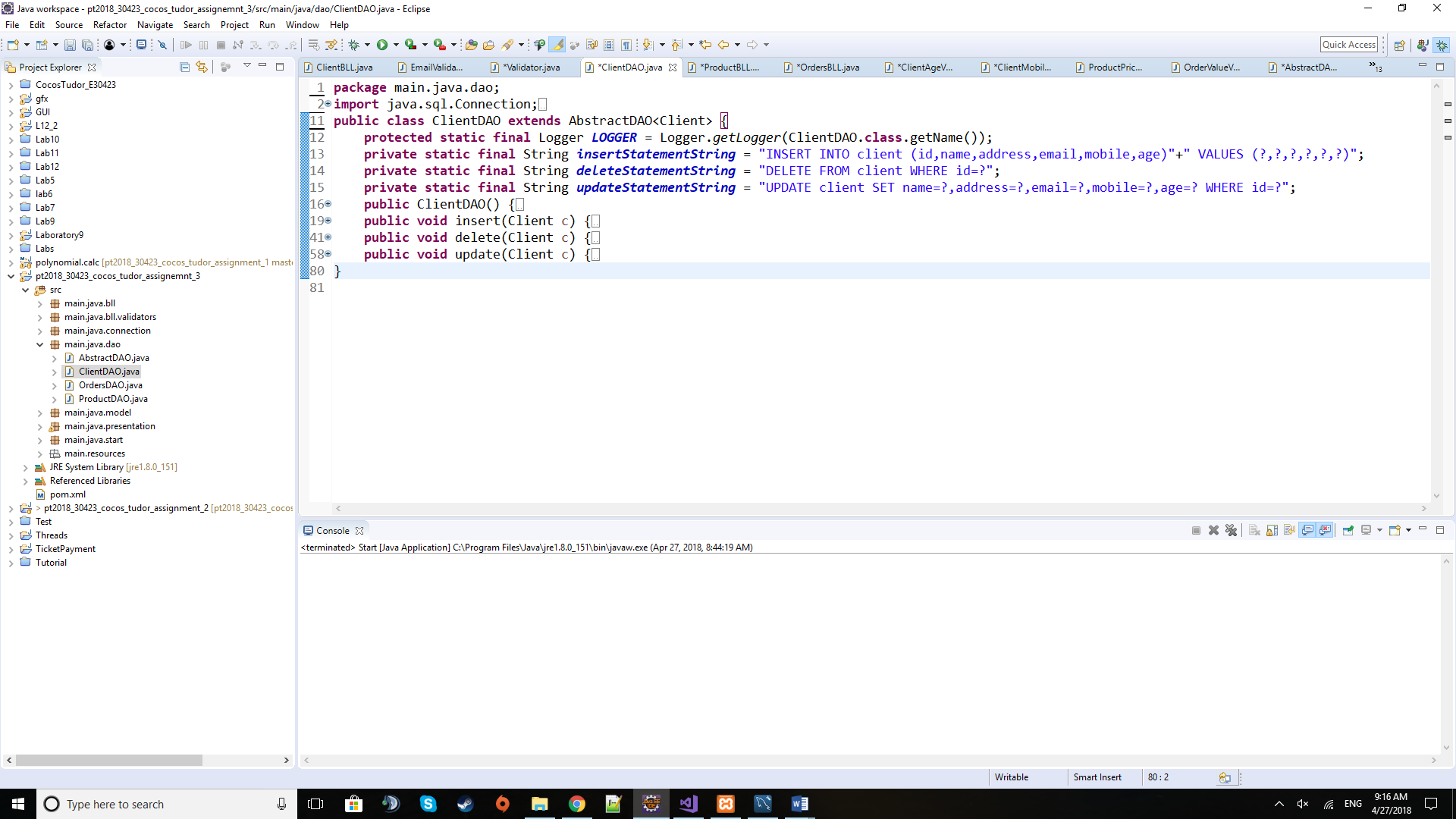




***Validator interface and one Validator class***

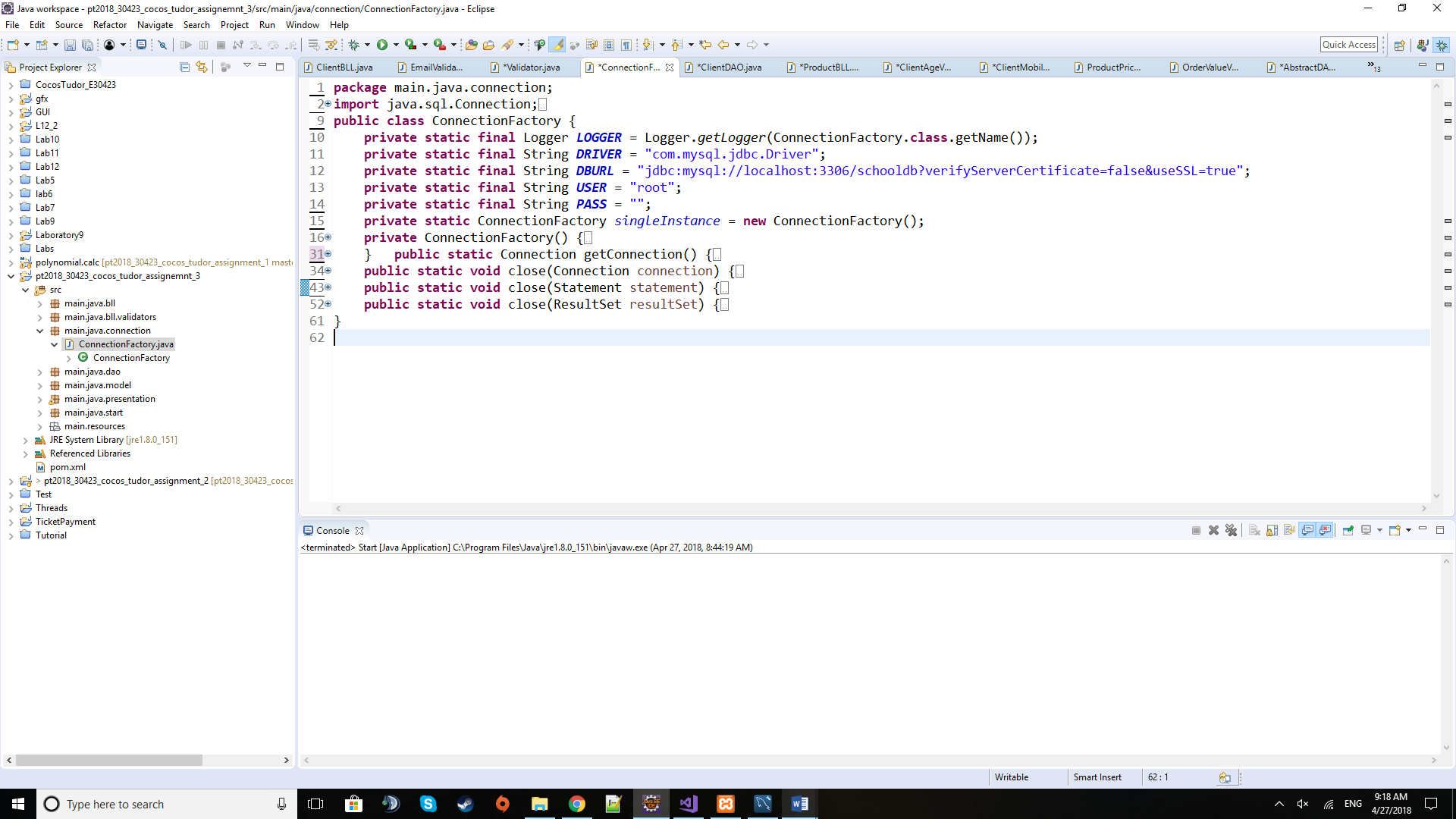


***AbstractDAO class***

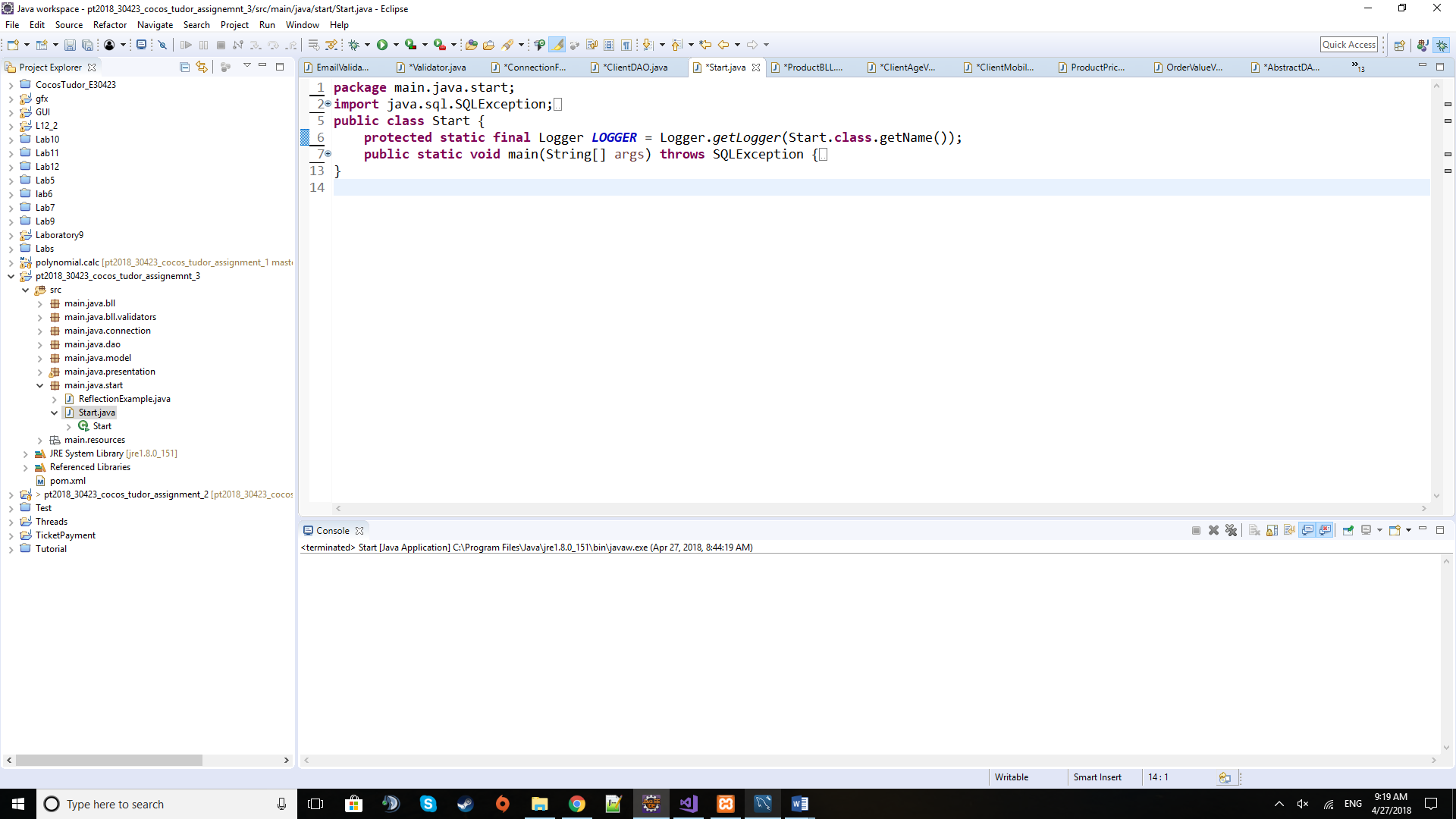


***ClientDAO class***

***-similarly, the OrderDAO and ProductDAO classes for orders and products, respectively-***



***ConnectionFactory class***



***Start class***

*d) Algorithms*

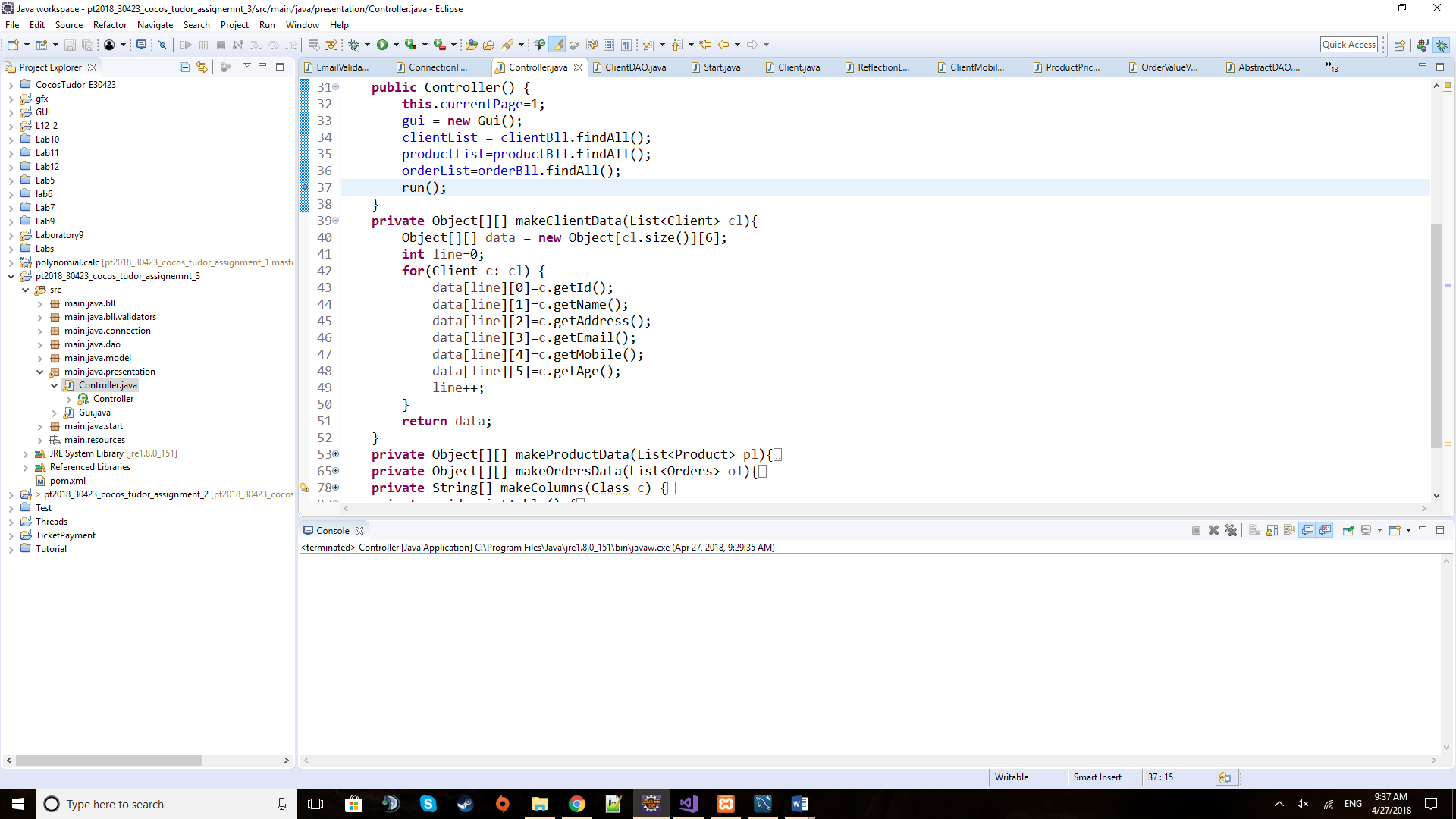
As for algorithms there are no special mathematical-based algorithms like in the polynomial processing assignment. The usual ones imply iterating through lists of clients or products, but the special ones this time are the ones used for reflection of class fields. The algorithms will be presented and explained in the Implementation section below.

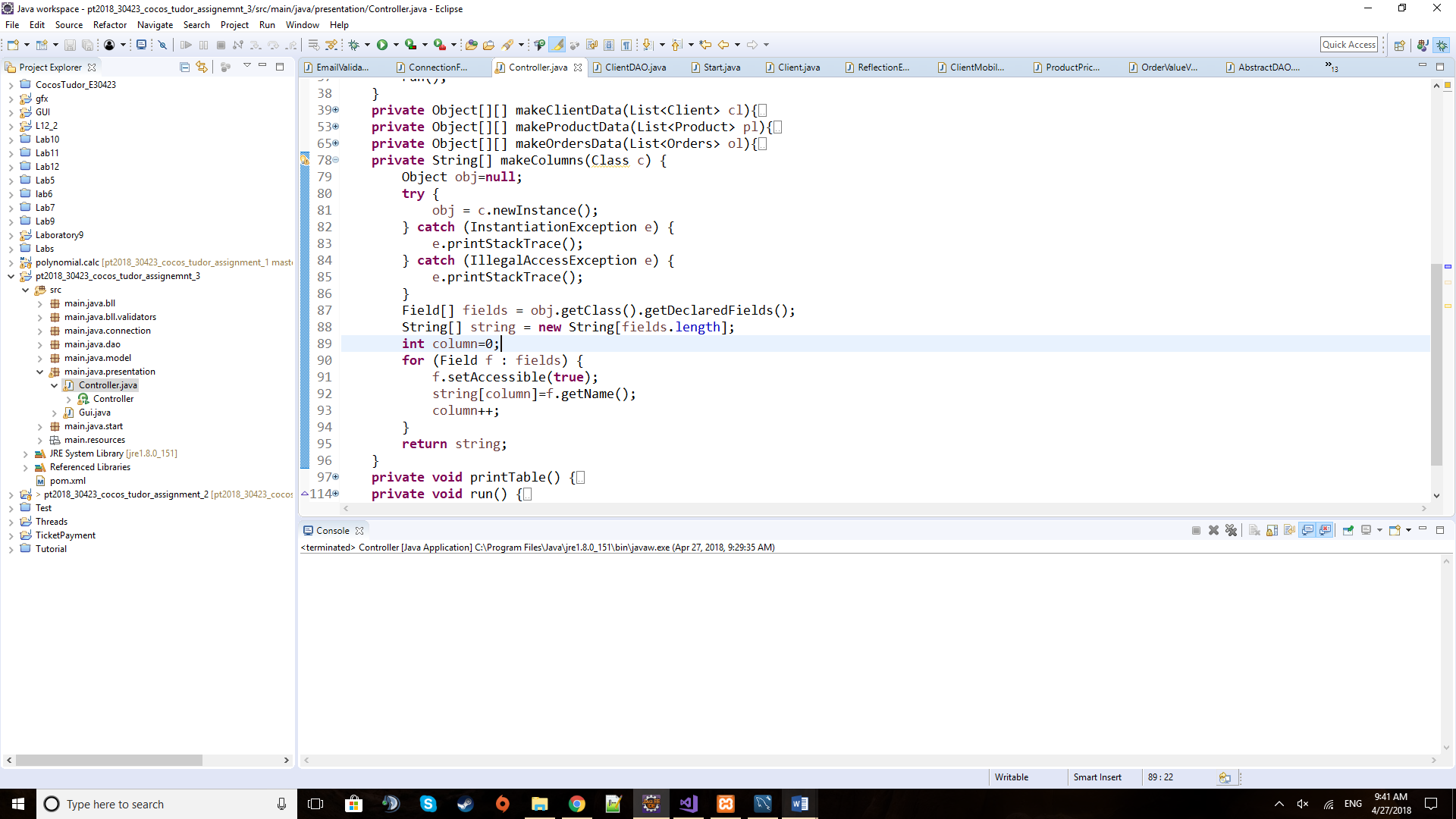
*e) User Interface*

This is the window that the user is prompted to when the application is started. It contains several types of graphical elements, each with its own specific use. At the top there are three buttons that are used to switch between table views: Clients Table, Products Table and Orders Table. If Clients Table is selected, for example, on the middle the fields for client name, address, email, mobile and age are displayed and can be filled in. Also, the button on the right (Insert, Update and Delete) operate only with clients now. If we press the Products Table button, the table view switches to the Product Table and the fields for product name, quantity and price are displayed while the action buttons on the right will now work for operations on products. The id field on the left is always displayed but just like the buttons on the right, it will know on what page we are at a given moment and will be used for clients / products / orders table.

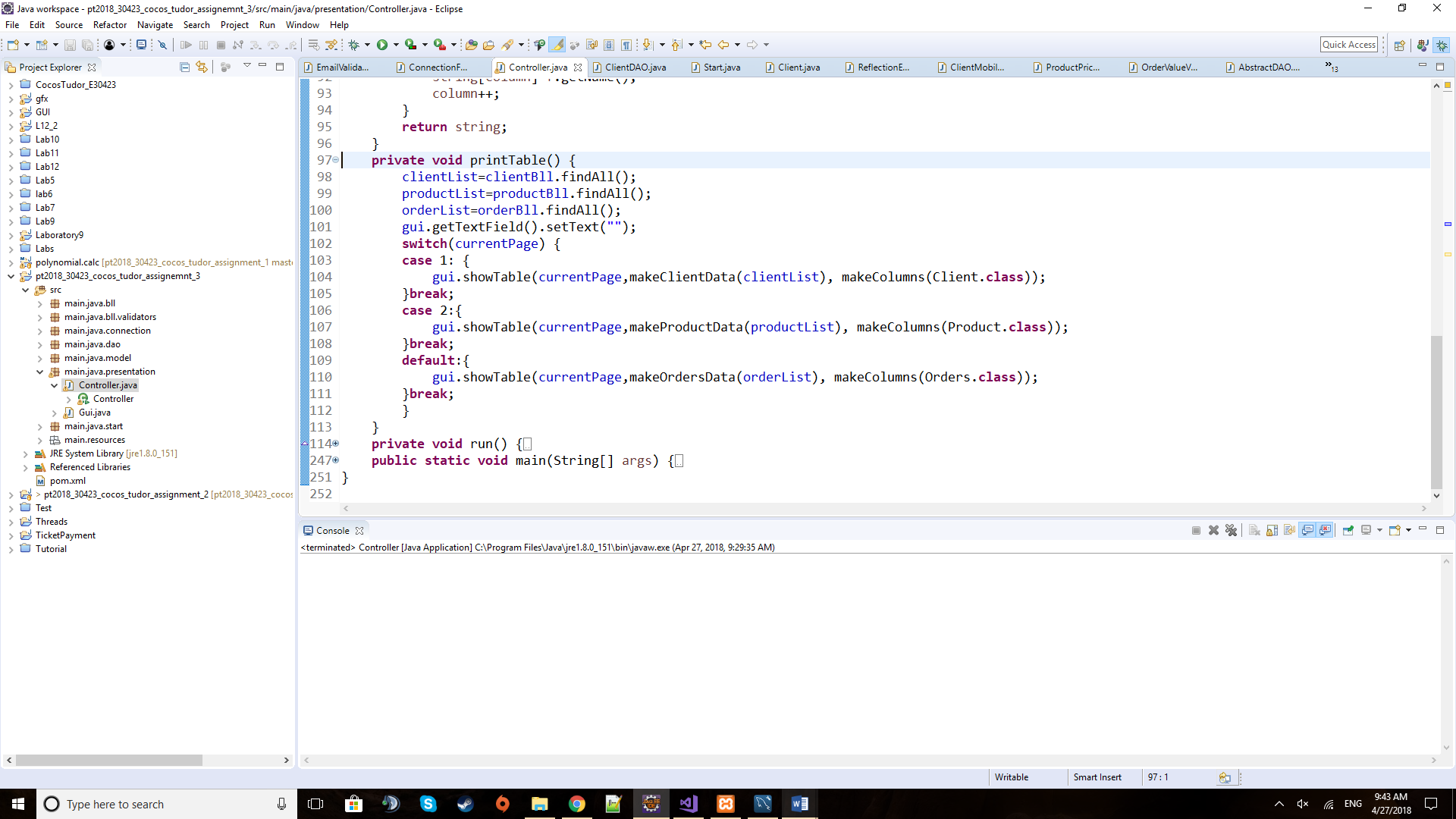
**4) Implementation**

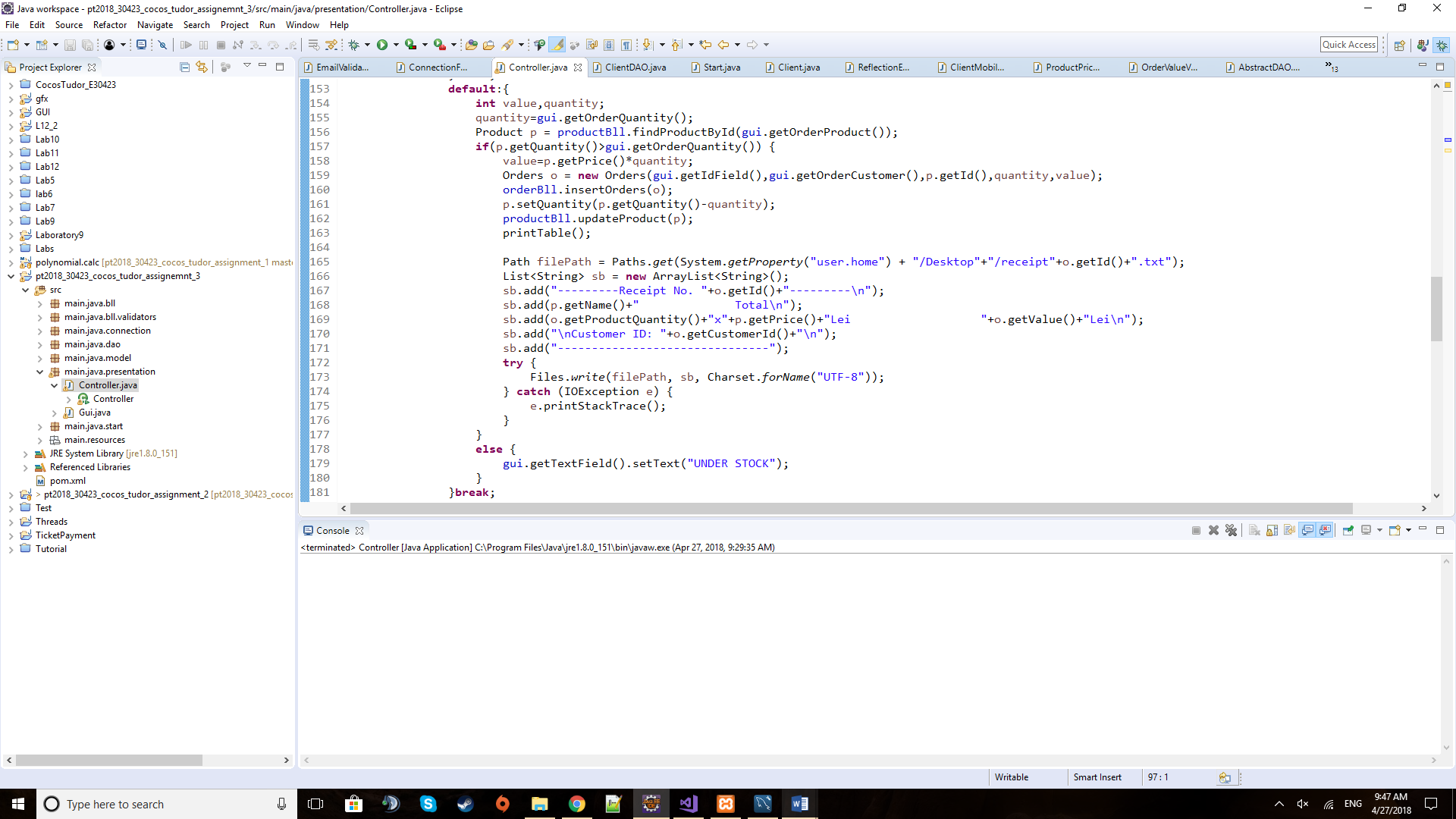
I will describe here the most relevant pieces of code and especially those that are not that common and straight-forward (unlike getters, setters, and some basic constructors).



The makeClientData is part of the Controller class and takes the list of clients and returns a matrix of objects: each row is a client while the columns are filled with the contents of their fields. In a similar fashion operate the makeProductData and makeOrdersData methods of the same class.

The makeColumns method takes a class given as parameter and returns an array of strings that contain the name of the fields of the class.



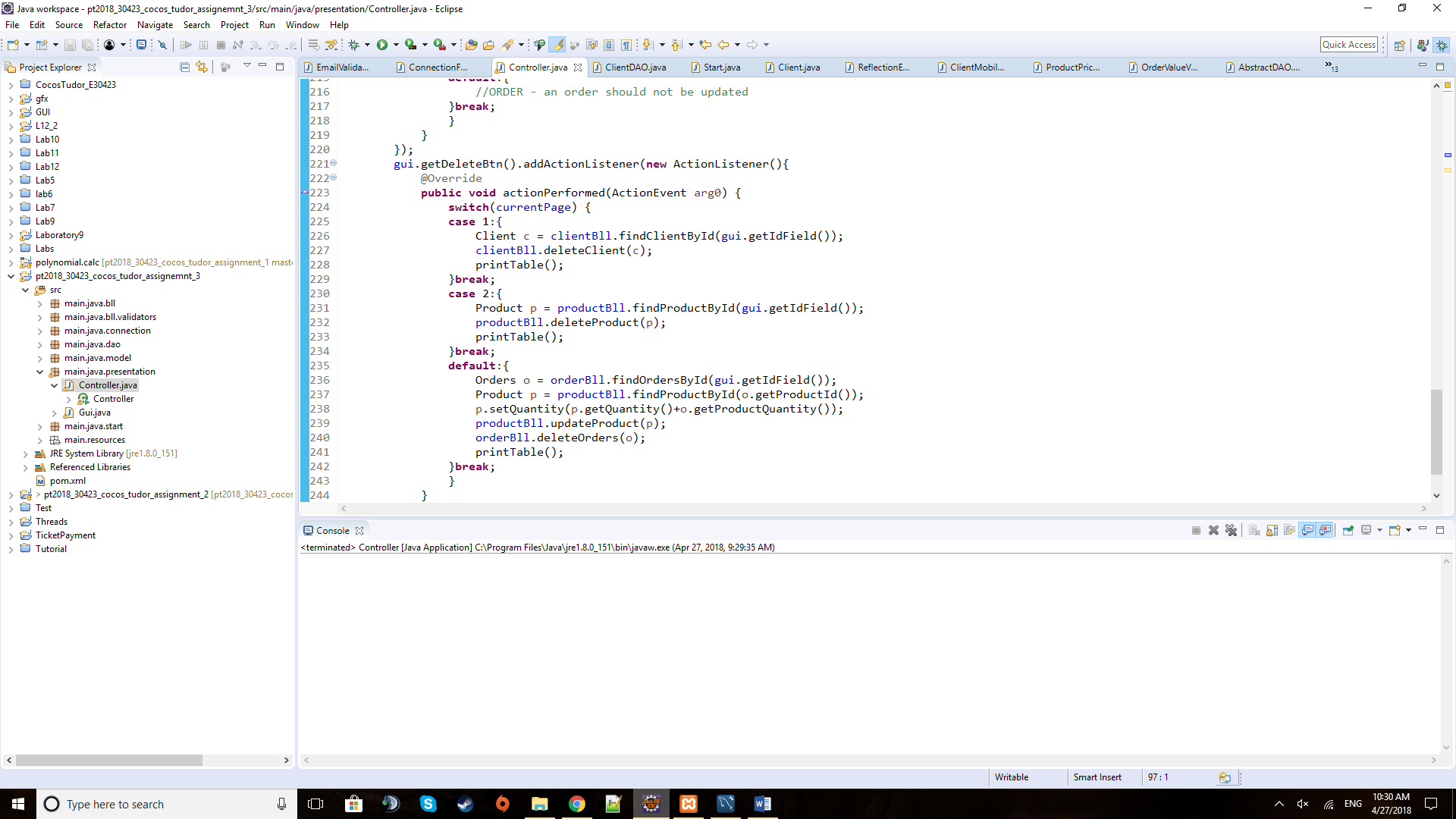
 PrintTable is used to generate the list of clients that are sent afterwards to the GUI with their respective data matrices and column headers in order to display the table of the current page.

When pressing the Insert button this is what the event handler does for orders as the clients and products are pretty straightforward. When an order is made, it will determine what product is ordered and compute the value of the order by that product’s price and the quantity that is ordered. This is done once it is checked that we can order such amount of the product and it is not under stocked. After updating both product and order tables, it will print a receipt on the desktop for the respective order containing the id of the order, the name of the product, the quantity of what was ordered, the price per unit and the total amount to be paid for the order. It will also provide the id of the client and just that, in order not to disclose his or her full identity.

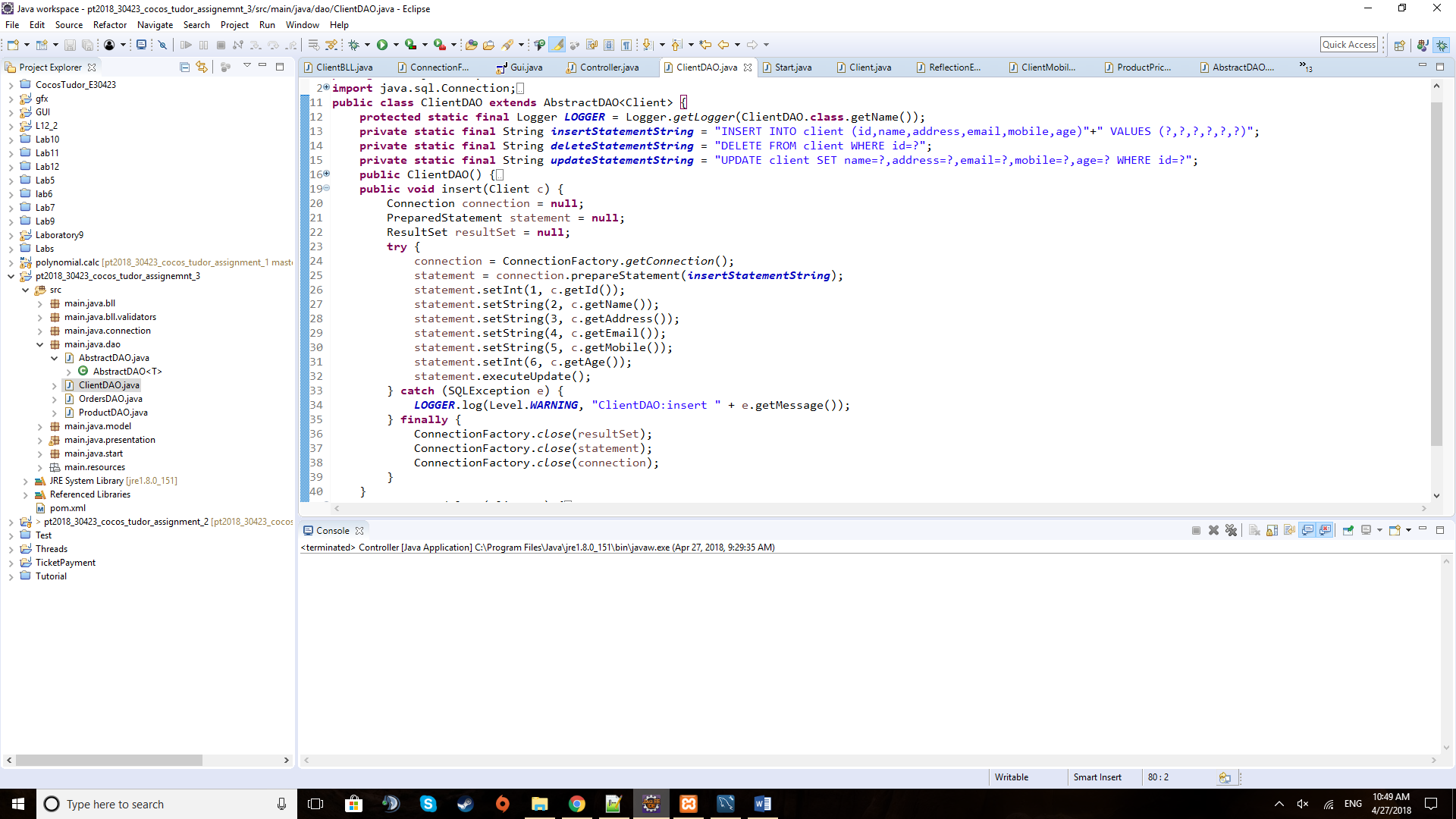


When pressing the Update button, depending on the current page that we are on, it will update either the client or the product. I considered useless and potentially harmful to update an existing order. It’s like going to the market and after the shopkeeper weighs the produce and waits for the payment, you tell him to add another 2 kilos of that product and after doing that you change it again and again. By doing that you block the sale process. An order should only be made or removed.

The method takes the instance of the client or product to be modified and checks the text fields in the form for new information that must be placed and updated in the given instance.



The above piece of code is part of the method used when pressing the Delete button, in the case of deleting an order. The other two, deleting a client or a product, are basic. This one takes care to remove the order but to also restock the product from the order.

 In ClientDAO class the Insert method operates using the string statement declared above and filling the ‘?’ with the right field values. Similarly, the delete and update is done and also the ones for the other types of entities like Products and Orders.

**5) Testing**

The testing for this project was done by running the application and observing the evolution of the tables depending on the actions taken and the data inputted in the text fields. I also checked every time for the same changes to be applied to the database tables and, indeed, they happened as planned. The testing was done on the whole length of the development after each method and feature was implemented.

**6) Results**

The results of the simulation are available at any time during the running of the application, being displayed in the appropriate tables for a certain page. Also, after performing an action, the table is reinitialized and displayed live. The data can always be found in the database tables too.

**7) Conclusions**

This assignment proved to be extremely useful to learn how to interact with a database using the Java programming language and OOP paradigms. Just like the previous assignment, the one with threads, opened up a new chapter in the development of my programming skills as this is a huge domain with tremendous amounts of useful features.

Similar to this application, an online shop may be created by using a database with clients, products and orders. But the applicability of databases in applications is infinite.

The warehouse management application can be developed further by increasing the number of fields for each entity or, for example, redoing the GUI in order to differentiate between a client user and an administrator user (the shop admin) and such the client being able to create only his customer account and only update his fields. He or she should also be able to insert an order or if the time allows to delete an order to cancel it. The shop administrator should be able to edit the product table in anyway and also by some rules and guidelines to delete the user accounts (inactive user or other reasons) or to delete an order (the customer does not want to pay for example, etc).

**8) Bibliography**

Tutorial to understand the issue and start working on the project the following presentation was really useful: <http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/HW3_Tema3/Tema3_HW3_Indications.pdf>

For any issues regarding exceptions: <https://stackoverflow.com/>

For details about the methods and fields of predefined classes: <https://docs.oracle.com/javase/8/docs/api/overview-summary.html>

An example used to generate a simple JTable and place it in a JScrollPane: <https://www.thoughtco.com/how-to-create-a-simple-table-2033894>

For the UML diagram I used an online application: <https://creately.com/blog/diagrams/uml-diagram-types-examples/>