Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

Semester 2021-2021

Fundamental Programming Techniques

- Laboratory Assignment no. 4 -

Food Delivery Management System

Student: Maria Gliga-Hambeț

Group: 30423

Supervising teacher: Cristina Pop

# TABLE OF CONTENTS

[1. Objectives 3](#_Toc97399318)

[2. Problem analysis, modelling, use cases 4](#_Toc97399319)

[3. Design 5](#_Toc97399320)

[4. Implementation 8](#_Toc97399321)

[5. Results 15](#_Toc97399322)

[6. Conclusions 17](#_Toc97399323)

[7. Bibliography 17](#_Toc97399324)

1. **Objectives**

The main purpose of this project is to design and later on implement a food delivery management system for a catering company, for instance, with a user-friendly graphical interface, which could be easily used in order to simulate real life scenarios. The user can be of 3 different kinds, all of which are able to perform various actions regarding the products and orders of the restaurant. What the project manages to achieve is managing a complex system which takes into account many scenarios.

In order to achieve the desired result, the whole project can be divided into the following, smaller tasks:

* Analyze the problem and identify requirements
* Design the orders management application
* Implement the orders management application
* Test the orders management application

1. **Problem analysis, designing, scenarios, use cases**
   1. **Problem analysis**

The food delivery management system application simulates a scenario that is met in various fields in everyday life, the management of a catering company, for example, or any sort of domain that requires the placement of orders, done by individuals. What should be achieved in the end are the options to add, update and delete clients, products and orders. As I have said, since the purpose of the application is a pretty general one, the need to choose an architecture which is as flexible as possible. For this reason, we make use of inheritance in order to handle different structures of the classes we are modelling in our application (in our case we have BaseProduct, CompositeProduct but we can extend to other classes too).

* **Functional requirements:**

- The application should allow the user (admin) to add a new client

- The application should allow the user (admin) to add a new product

- The application should allow the user (admin) to modify an already existing product

- The application should allow the user (admin) to delete an existing product

- The application should allow the user (admin) to import the initial products

- The application should allow the user (client) to add a new order of one or multiple products

- The application should check the validation of the operations the user is trying to do in order to keep the use of this application real to our catering company

- The application should allow the user (admin) to create reports, in order to get more data about the orders easily

* **Non-Functional requirements:**

- The application should be intuitive and easy to use by the user

- The application should be modifiable and allow for future development

In order to make a closer to life experience, we must take into account the following facts:

* The user is announced if the username or password are incorrect when logging in
* Produces are only present once in the data base, username must be unique
  1. **Designing**

In our application we are using Composite Design Pattern in order to have more flexibility in manipulating our basic items in our application (Base and Composite products). Internally they are designed differently but in their usual use case they are considered as the same object, so we need to integrate this design pattern in our application. These values are stored as byte streams in a file. I have also though the graphical interfaces in a more flexible way. As many of them were similar, I used the same class, but manage to change the information as it was needed.

* 1. **Scenarios and use cases**

When the simulation is started, the user must log in to their account. They can create a new account or log in to their existing one. If the username or password are incorrect, the user is announced. Depending on the type of user, the scenarios are very different, as the purpose of the 3 types of users are different. chooses what part of the application he wants to interfere with: the management of clients, products or orders. The admin can add, modify or delete a product. He can also import the initial set of products, search based on multiple criteria or generate order reports. The client can also search for products, but in terms of orders he can only add to the current order and place the order in the end.

**Use case description**

**Use Case**: Register a client

**Primary Actor**: User

**Main success scenario**:

1. The user accesses the first interface where you can either login or register as a client.

2. The application will display an interface where we have a form to add data (username and password).

3. The user inserts the data for creating the new client.

4. The user click on the “Register” button.

5. A new window pops up, also asking for the full name of the user

6. The user click on “Register” again and he is now logged in to the application and his account is stored among the users.

**Alternative sequence**: Invalid input given by the user

- The user inserts a username which already exists in the file

- The user didn’t add data to all the fields in the form

- The application creates a pop-up window to inform the user that an error has occurred

- The scenario returns back to step 3

**Use Case**: Modify a product

**Primary Actor**: Admin

**Main success scenario**:

1. The user (admin) accesses the interface where he can interact with the list of products.

2. The user selects one of the rows of data from the table

3. The user clicks on the “Modify” button

4. A new window pops up which allows the user to change any data about the product

5. The user click on the “Modify” button again

6. The application modifies the updated product with the new data and the user is brought back

7. The content of the product table is refreshed in real time so that the user can see the changes in the files.

In order to delete a product, the scenario is a lot like the one of modifying a product, but after selecting it and pressing a button, the action is completed.

**Use case**: Create a composite Product

**Primary Actor**: Admin

**Main success scenario**:

1. The user accesses the interface where he can interact with the list of products.

2. The user selects a product from the table and pushes the “Add to menu” button and keeps on doing so until he has added all the wanted products

4. The user inserts a title for the new composed product.

5. The user click on the “Finish menu” button.

6. The application modifies the updated product with the new data and the user is brought back

7. The content of the product table is refreshed in real time so that the user can see the changes in the files.

**Use case**: Generate reports

**Primary Actor**: Admin

**Main success scenario**:

1. The user accesses the interface where he can interact with the list of products.

2. The user selects clicks on the “Generate reports” button.

4. The user is presented with a new window which allows him to complete the wanted limitations for the reports and then later on generating them by pressing the according buttons.

**Use case**: Place an order

**Primary Actor**: Client

**Main success scenario**:

1. The user accesses the interface where he can interact with the list of products.

2. The user selects a product from the table and pushes the “Add to order” button and keeps on doing so until he has added all the wanted products

4. The user inserts a title for the new composed product.

5. The user click on the “Place order” button.

6. The application modifies the updated product with the new data and the user is brought back

7. The content of the product table is refreshed in real time so that the user can see the changes in the files.

**Use case**: Search for a product

**Primary Actor**: Client

**Main success scenario**:

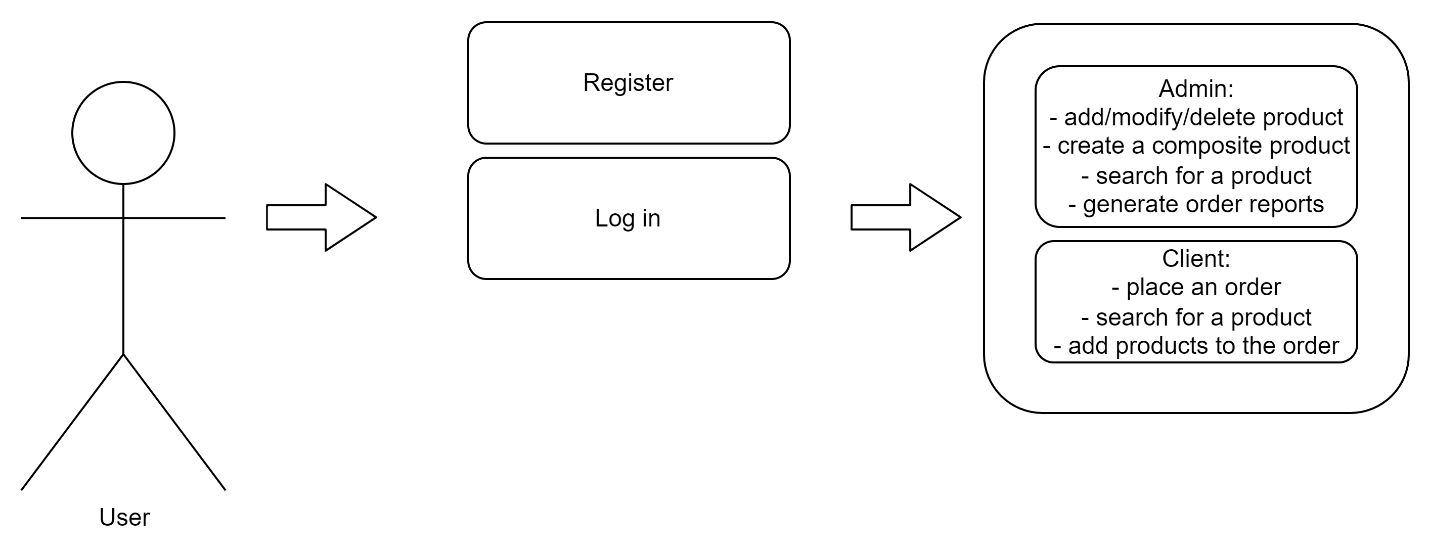
1. The user accesses the interface where he can interact with the list of products.

2. The user clicks on the “Search” button

4. The user is presented with a new window which allows him to add any new wanted criteria

5. The user click on the “Search for products” button again.

6. The application shows the user a table with all the products that follow the wanted specifications.



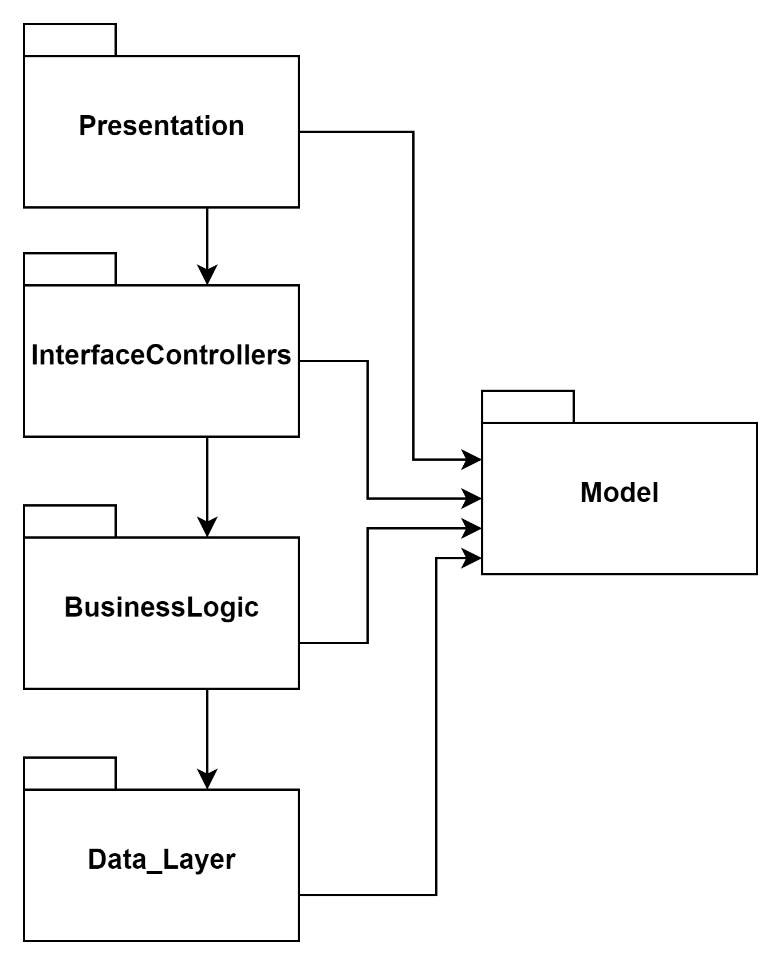
1. **Design**

The structure is formed of main 2 designs: the class structure and the files where we store these objects in a serial manner. The files are menu\_items.ser where we store the products (base or composite products), orders.ser where we store our orders and users.ser where store our users (admin, employee or client). The main-program architecture which is a layered one. We adopted a MVC architecture (Model–view–controller) which is a software design pattern commonly used for developing user interfaces that divide the related program logic into three interconnected elements.

Going to lower level, our design in package-level structure contains the following packages:

* 1. **Packages and relationship between them**

The architecture used in this project is a layered architecture, made out of five packages: BusinessLogic, DataAccess, Model, Connection and Presentation (this package is, as well, divided into smaller packages in order to keep things more organized. Therefore, it consists of the Client, Order and Window packages and a standalone class, the StartingWindow one). Their roles are as it follows:

* The BusinessLogic holds all the decisions which represent the internal behavior of our application are defined here. We can identify all the operations and others methods which we use in order to transform data from input in specific data structures (defined in Model package)
* The DataLayer contains we define the actions that determine the exchange of data between the application and the files (the data is structured according to the Model package classes).
* The Presentation package contains the frames of the graphical interface.
* The InterfaceControllers contains all the controllers for the frames mentioned above. They also hold part of the decision-making process of the application.
* The Model holds the basic classes we will be working with in the whole application, those being, in our case: Client, Product and Orders.

1. **Implementation**

**The Business\_Layer** consists of an interface IDeliveryService and a class implementing it. Therefore, one of the most important classes of the project is the DeliveryService class, as it manages the basic operations when it comes to making the connections between other packages and classes. It interacts with the serializer, holding methods for writing and reading from our files. It also handles functions such as generating the order reports using streams and other methods in order to transmit information in a more adequate way - I added a getData method which returns the data of the tables in the form of a matrix of type String, in order to interact more easily with the graphical user interface (the JTable requires this particular type for the received data).

In order to hold the data, I have chosen HashSet and HashMap in order not to have duplicates, keeping the data consistent.

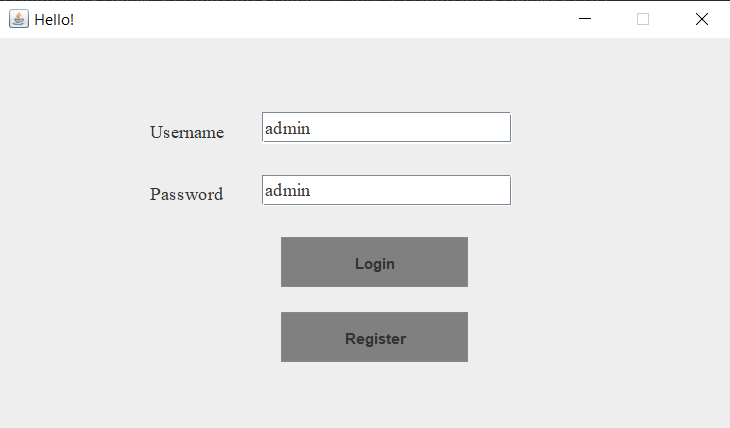
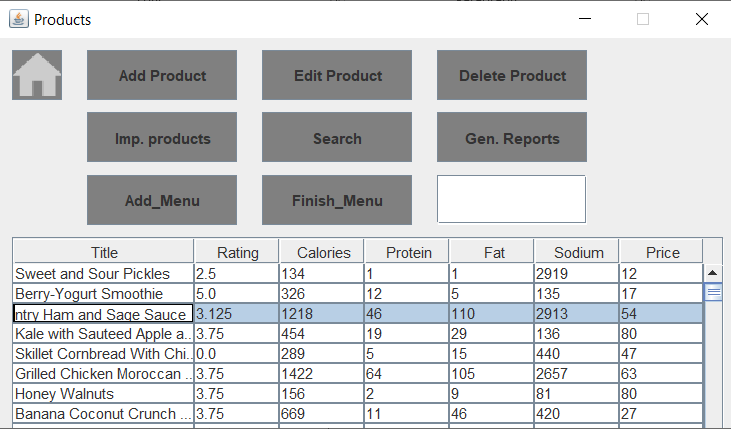
**The Model** package consists of the classes which are used all throughout the project:

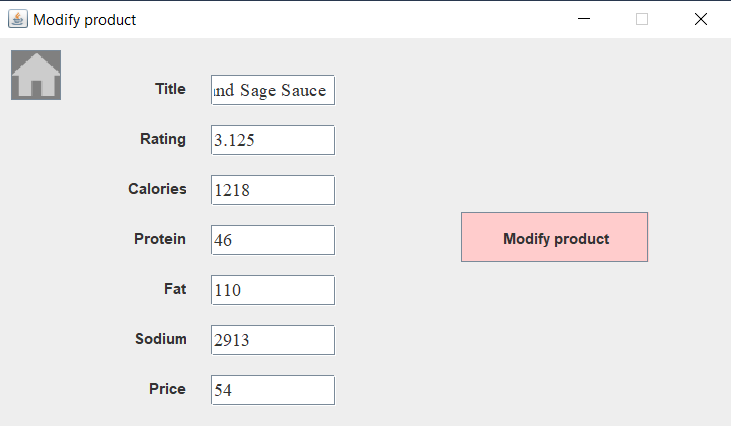
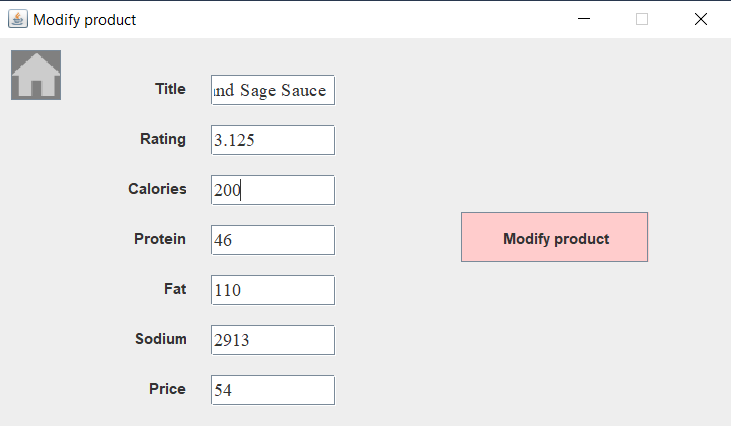
* The User class defines the user of our application, which is defined by userID, username, password, name and type, where type is the type of user
* The userType is an enumeration of the 3 possible types of users: admin, client and employee.
* The BaseProduct and The CompositeProduct classes both extend the MenuItem, which basically describes a product that we can have inside an order

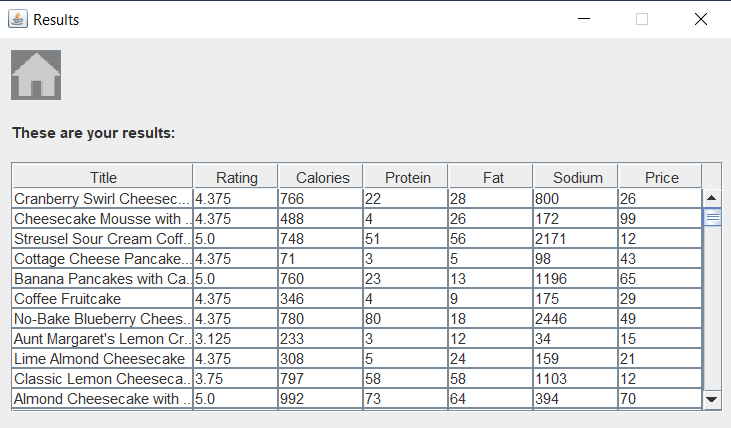
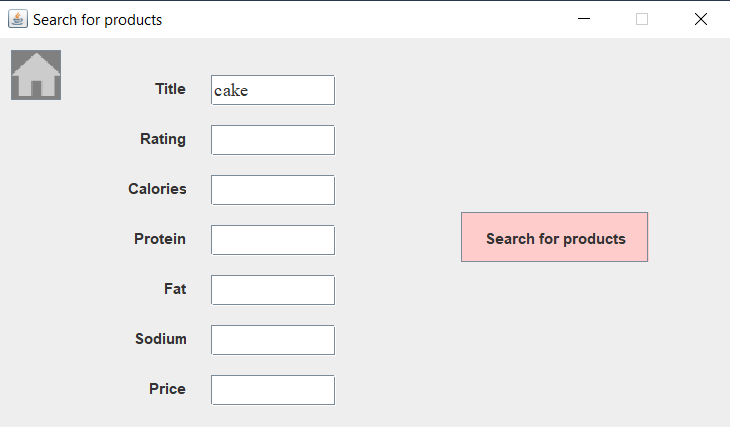
**The Data\_Layer** consists of 2 classes, which both interact with files form outside the project. The Serializator holds methods to write and read from serialized files, while FileWrite creates, writes and then closes a normal text file, used for writing the bills and the reports.

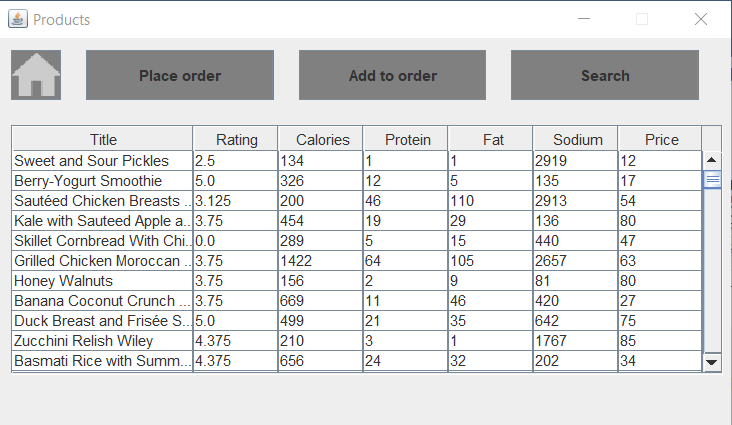
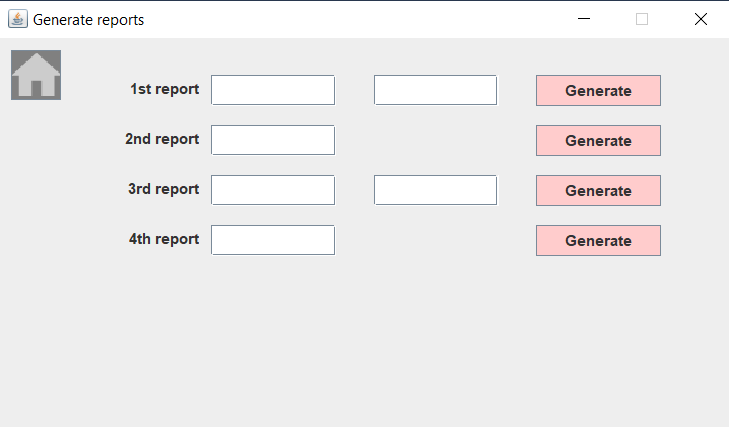
**The Presentation** package contains all the frames of the graphical user interface. Some of them are more generalized and reused, in order to make it more efficient. Their controllers are in the **InterfaceControllers** package, those interacting with the DeliveryService class and the FileWriter.

1. **Results**

In order to demonstrate at least parts of the way the application works, I followed some of the user cases and screenshotted the results:







1. **Conclusions**

The whole project has enabled me to learn how to work with serialized files and streams. Moreover, to see the usefulness in working in an as general as possible manner as possible, in order to later on be able to extent the project without a lot of extra work. Compared to the previously realized projects, in terms of the graphical interface, the current project also enabled me to learn how to switch in between windows and update the displayed information.

What is more, I have deepened my knowledge on how to work with a layered architecture.

Future developments for the project:

* Add more checks throughout the project
* Add a functionality which allows the status of the orders – the employees to be able to categories them as done, in the process or not yet started
* I would like to see what ways there are to update the data displayed to the user in order to make the application work faster

1. **Bibliography**

* <https://dsrl.eu/courses/pt/>
* <https://www.baeldung.com/java-observer-pattern>
* <https://howtodoinjava.com/java8/java-stream-distinct-examples/>