

**Fundamental Programming Techniques**

**Assignment 4: Food Delivery Management System**

**Student:** Diana-Maria Jurcan

**Group:** 30422

**Laboratory professor:** Marcel Antal

**Course professor:** Cristina Bianca Pop

# CUPRINS

1. Assignment objective……………………………………….3
2. Problem analysis, modelling, scenarios, usage cases……….4
3. Design……………………………………………………….5
4. Implementation………………………………………………9
5. Results……………………………………………....……...17
6. Conclusions………………………………………………...20
7. Bibliography……………………………………………......20
8. **Assignment objective**

The main objective of this theme is to design and implement an application that simulates the functionalities of a food delivery management system that allows the connection of three types of users (administrator, employee, client), being dedicated to placing and fulfilling food orders and having an intuitive graphical interface to facilitate interaction with any type of user.

The secondary objectives of this topic are presented in the table below:

|  |  |  |
| --- | --- | --- |
| Secondary Objective | Description | Section |
| 1. Analysis of the problem | * Involves the analysis and determination of all the requirements that the application must meet | Analysis |
| 1. Identification of the scenarios | * Involves the identification and analysis of all scenarios in which the application could be found depending on the input values entered by the user (exception, proper operation, etc.) | Analysis |
| 1. Identification of the use cases | * Involves identifying situations in which the app justifies its functionality | Analysis |
| 1. Design | * Involves going through the stages of synthesizing and designing the app according to the OOP paradigm, by making UML diagrams, designing classes, imagining the GUI, etc. | Design |
| 1. App implementation | * Involves the actual development of classes and their related methods in code | Implementation |
| 1. Testing | * Involves testing the functionality of the app in all possible operating situations summarized in the use cases | Results |

1. **Analysis**

The problem to be solved is the fact that in the current reality, there has been significant

increase in home delivery policies, especially in the restaurant industry, which brings the need to develop customer ordering systems and their receipt by employees in order to meet the demand and delivery of the desired products to the buyer's home.

Therefore, following the analysis of this issue, the following framework of functional requirements has been identified:

▪ The application must be able to connect three types of users.

▪ The application must inform the user if the input entered is invalid, i.e., for the fields

where numeric values are expected, other characters are entered, the time slot is

invalid, etc.

▪ The application must be able to allow each administrator user to perform basic

operations on the products available to the company: importing the initial set of

products, adding new products, deleting existing products, editing existing

products, creating composite products, generating reports for designing sales buying statistics.

▪ The application must allow each customer user type: create a new account, connect

with an existing account, view all component products, place an order and filter

available products according to various selection criteria.

* The application must allow each employee to view all orders placed to

honor them and notify each employee when placing a new order.

The framework of the non-functional requirements outlined around the problem to be solved is:

▪ The application must have an intuitive graphical interface.

▪ The application must be easy to use for any type of user, regardless of their field

of activity.

**Modelling**

The application was designed on the model of a system with two inputs and two outputs, the inputs being represented by the username and password associated with the account, information required to connect to the application, while the outputs are represented by the invoice generated after placing and registering the order to be honored, but also to visualize the information through the graphical interface.

The problem model is thus translated into an object-oriented model, based on the User, MenuItem, BaseProduct, CompositeProduct, Order and DeliveryService classes, through which the inputs, outputs and functionalities associated with the application will be defined and transposed in the object model, defined on the basis of the real-world model.

1. **Design**

* **Use case diagram**

The use case diagram for the Administrator user is shown below:

**Diagram

Description automatically generated**

The usage case diagram for the Client user is presented below:

**Diagram

Description automatically generated**

The use case diagram for the Employee type user is shown below:

**Diagram

Description automatically generated**

Use cases are similar in terms of the flow of steps performed by the user and that of the answers provided by the implemented application, which is why a single overview of the steps performed in both the success scenario and the limit scenarios will be developed.

The main actor: User (Administrator / Client / Employee)

**Success scenario:**

1. The user correctly enters the information necessary for the desired functionality in the text fields present in the graphical interface or selects from the tables provided in the graphical interface of the application the useful and necessary information for the functionality in question.

2. The user presses the button associated with the functionality that the application wants to perform.

3. The application allows you to view the information associated with the functionality in the graphical interface or generates an output file corresponding to the results of performing the operation desired by the user.

Pressing the button also serializes the information within the application and saves it in the appropriate file, necessary after deserialization.

**Alternative scenarios (possible errors):**

The information entered by the user is incorrect or inconsistent in terms of the functionalities implemented.

1. The user is notified by the appearance of a window with an error message corresponding to the exception generated.

2. The scenario returns to the step of entering the information necessary to achieve the desired functionality at the moment by the user logged in to the application.

* **UML Diagram**

Unified Modeling Language or UML for short is a standard language for describing models and software specifications. UML was originally developed to represent the complexity of object-oriented programs, the foundation of which is the structuring of programs into classes, and their instances (also called objects).

However, due to its efficiency and clarity in the representation of abstract elements, UML is used beyond the IT domain. On the next page you can see the UML diagrams generated with IntelliJ.

A screenshot of a computer

Description automatically generated with medium confidence

A picture containing graphical user interface

Description automatically generated

In the UML diagram above are presented all the classes, with the attributes and the relations between them, the main packages are:

* Package business:
* Class BaseProduct
* Class CompositeProduct
* Class DeliveryService
* Class IDeliveryServiceProcessing
* Class MenuItem
* Class Order
* Class User
* Class UserType
* Package data:
* Class FileReaderWriter
* Class Serializator
* Package main:
* Class Main
* Package presentation:
* Class AdministratorGUI
* Class ClientGUI
* Class EmployeeGUI
* Class LogInGUI
* Class MenuItemsTable

1. **Implementation**

**Data structures**

The most important data structures that have been used are the following:

* Map<Order, Collection<MenuItem>>: hash map of pairs order and menu items for placed orders
* Set<MenuItem>: set of menu items in delivery service class
* Set<User>: set of users that are signed up
* List<MenuItem>: list of menu items in composite product class

**Packages, classes and methods:**

In the following, all the important classes and methods will be briefly presented:

* **Package business:**
* **Class BaseProduct:**
* Contains the values for rating, price, calories, protein, fat, sodium of each base product taken from file products.csv.
* Important methods: getters and setters for each field, showProductDetails which will print out each field, computePrice which will be overridden and the overriding of the functions toString, hashCode and equals.
* **Class CompositeProduct:**
* Contains a list of menu items from which the composite product will be made and the price of each composite product.
* The main methods are: addProduct which adds a base product to a composite product, removeProduct which removes a product, and showBaseProductsDetails which shows the details of the base products that are included in the composite. Also, getters for rating, fat, protein, sodium, calories are overridden here. As well as computePrice, toString, hashCode and equals. The method searchByName will also be of importance because it will return the title of the base product or the composite according to which needs to be shown.
* **Class DeliveryService:**
* Contains a set of menu items, a map of orders + collection (menu item) and a user set
* In the constructor, we will use the method deserializeAll which will deserialize all the info from the given file. setChanged will notify that the set of elements has been changed and the observers will be notified meaning the employee will be notified and the info of the order will be written in their window.
* Main methods are: addInitialUsers which will add an admin and an employee for the first time in the application, registerNewClient which adds a new client, findByName which searches a product according to its name, logInUser, serializeAll, deserializeAll, findUser which finds a user according to its username, newOrder which places a new order notifies the observer and creates the bill, readProductsFromCsv which will get our info from the txt files which will be serialized and all the other methods which will add/delete/update a product and the methods which make the corresponding reports.
* **Interface IDeliveryServiceProcessing:**
* Contains the possible functionalities that are implemented in the DeliveryService Class
* The methods are readProductsFromCSV, addProduct, deleteProduct, updateProduct, filterProducts which will filter the products according to different criteria, registerNewClient which register a new client which signed up into the app, logInUser, newOrder which will register a new order that was made and methods for all the reports that need to be made.
* **Abstract Class MenuItem:**
* Contains the title field, and the abstract getters for price, rating, calories, fat, sodium, protein.
* **Class Order:**
* Contains fields for client ID, order ID, and the order date.
* Main methods implemented here are getters and setters, getOrderHour, convertDateToLocalDate, getOrderDay, getOrderMonth, getOrderYear which all will be helpful in the making of the reports. Also, the methods equals, hashCode, toString and compareTo are overridden here.
* **Class User:**
* Contains user id, username, password and userType fields.
* Main methods are overridden hashCode, toString and equals methods along with getters and setters for the user fields.
* **Enum UserType:**
* Contains the types of users the application can have: administrator, employee, client.
* **Package data:**
* **Class FileReaderWriter:**
* In this class, we mostly have some important methods which have to do with reading and writing information regarding products and orders
* Contains methods like readFromCSV which uses streams to read each field we have to get regarding our products, createBill which writes the info needed regarding an Order in a txt file and methods for generating the String used for writing in a txt file the report information for each one.
* **Class Serializator:**
* Contains two methods for serialization and deserialization of objects of generic type T from a given file. Throws IOExceptions for unwanted cases.
* **Package main:**
* **Class Main:**
* Here we have the start of the application.
* **Package presentation:**
* **Class AdministratorGUI**
* **Class ClientGUI**
* **Class EmployeeGUI**
* **Class LogInGU**
* **Class MenuItemsTable**

**Graphical User Interface**

The application has an intuitive graphical interface, developed using Swing and built on four JFrames:

* a window dedicated to connecting users within the implemented system
* a second window for Administrator functionalities, with a tab dedicated to each type of functionality that was implemented
* a third window to facilitate the operations allowed for Client-type users, with tabs attached to each type of functionality
* a fourth window, dedicated to Employee-type users, which allows viewing the orders placed by customers.

Windows were developed using a JTabbedPane main panel in which various sub-panels were added, implemented using JPanel. Thus, the graphical interface consists of editable text fields for entering information for the registration of a new customer or product by the user, the provision of reporting information by the Administrator and the setting of filtering criteria by the connected customer.

Labels are made using JLabel, while text fields are based on JTextField components. Also, in the graphical interface of the application you can find many buttons buttons, made using JButton components. Tables containing available products have been placed in JScrollPanel scrolling panels to allow them to be viewed in their entirety.

Below, we can see the Log In and Sign Up window:

**Graphical user interface, text, application

Description automatically generated**Graphical user interface, text, application, email

Description automatically generated

And here we have the Administrator window with each tab:

Graphical user interface, text, application, Teams

Description automatically generatedGraphical user interface, application, table

Description automatically generated

Table

Description automatically generated

Table

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Next, we have the Employee Window, which will show a notification for each order:

Application

Description automatically generated with low confidence

And the Client Window with its tabs:

Graphical user interface, text, application, Teams

Description automatically generated

Table

Description automatically generated

Table

Description automatically generated

Next, the following messages will be displayed if the user input is incorrect:

Graphical user interface, application, Word

Description automatically generated Graphical user interface, application, Word

Description automatically generated Graphical user interface, application, Word

Description automatically generated

Graphical user interface, application, Word

Description automatically generated Graphical user interface, application, Word

Description automatically generated Graphical user interface, application, Word

Description automatically generated

The graphical interface facilitates the use of the simulator by any type of user and provides a better interaction of the user with the object model of the solution.

1. **Results**

The application was tested to cover all use cases, both in the success scenario and in the failure scenario. The application reacted correctly, having the expected behavior in all test cases provided, complied with the framework of functional requirements imposed by the problem specification, in the generated .txt file can be found for each order information about the customer who placed the order, product information ordered and the total payment, as can be seen from the figure below:

Text

Description automatically generated

A picture containing text

Description automatically generated

The files containing the administrator-generated reports also reflected the correctness of the application's behavior in saving data by serialization and loading it through deserialization, the reports providing accurate and statistically relevant information about customers, orders, products, which can be seen in the figures below:

A screenshot of a computer

Description automatically generated with medium confidence

Graphical user interface, text

Description automatically generated

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

Finally, the operations that result in the display of information in the graphical interface associated with the application have proved correct in behavior on all test cases provided, as can be seen in the following figure:

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, table

Description automatically generated

The testing of the project was performed on a wide range of tests, the application having a positive response to the multitude of test cases provided, among which were also exceptional situations that were intercepted and treated correctly. Also, at each test step, the consistency of the results written in the file containing the invoice attached to the order with those retained through serialization was verified.

1. **Conclusions**

In conclusion, it can be stated that this topic comes to solve the problem of necessity development of a computer management system for companies whose object of work is the delivery of food products, being implemented according to the paradigm of object-oriented programming and being designed as a mini-application that can be used by any type of user, having a pattern architectural that allows the separation of the object model from the graphical interface with which the user interacts, each level of internal logic being well defined. For the project developer, its implementation proved useful, resulting in a better mastery of manipulating and developing Object Oriented applications, by practicing the practical implementation of concepts specific to this programming paradigm, but also gaining new knowledge about saving data through serialization. , Possible further developments of the application include the display of a window containing the details and component products of a composite product when selecting it in order to place an order, but also the possibility to cancel the order placed within a certain time after registration its.

1. **Bibliography**

* <https://stackoverflow.com/>
* <https://docs.oracle.com/javase/tutorial/uiswing/>
* <https://www.vogella.com/tutorials/JUnit/article.html>
* <https://www.baeldung.com/junit-5>
* <https://www.geeksforgeeks.org/>
* <https://en.wikipedia.org/wiki/Use_case>
* <https://google.github.io/styleguide/javaguide.html>
* <https://www.lucidchart.com/>
* <https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>
* <https://www.tutorialspoint.com/java/util/timer_schedule_period.htm>
* <https://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html>
* <https://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html>
* <https://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html#axzz6wacloiOB>
* <https://www.tutorialspoint.com/java/java_serialization.htm>
* <https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>
* <https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html>