DOCUMENTATION

ASSIGNMENT No. 4

STUDENT: Mureșan Salomeea

GROUP: 30424

# CONTENTS

[1. Assignment objective 3](#_Toc95297885)

[2. Problem analysis, modeling, scenarios, use cases 3](#_Toc95297886)

[3. Design 3](#_Toc95297887)

[4. Implemention 3](#_Toc95297888)

[5. Results 3](#_Toc95297889)

[6. Conclusion 3](#_Toc95297890)

[7. Bibliography 3](#_Toc95297891)

# Assignment objective

Consider a food delivery management system for a catering company. The client can order products from the company’s menu. The system should have three types of users that log in using a username and a password: administrator, regular employee, and client.

The administrator can:

* Import the initial set of products which will populate the menu from a .csv file.
* Manage the products from the menu: add, delete, modify products and create new products composed of several products from the menu (an example of composed product could be named “daily menu 1” composed of a soup, a steak, a garnish, and a dessert).
* Generate reports about the performed orders considering the following criteria: o time interval of the orders – a report should be generated with the orders performed between a given start hour and a given end hour regardless the date. o the products ordered more than a specified number of times so far. o the clients that have ordered more than a specified number of times so far and the value of the order was higher than a specified amount. o the products ordered within a specified day with the number of times they have been ordered.

The client can:

* Register and use the registered username and password to log in within the system. • View the list of products from the menu.
* Search for products based on one or multiple criteria such as keyword (e.g., “soup”), rating, number of calories, proteins, fats, sodium, prices.
* Create an order consisting of several products – for each order the date and time will be persisted, and a bill will be generated that will list the ordered products and the total price of the order.

The employee is notified each time a new order is performed by a client so that it can prepare the delivery of the ordered dishes.

# Problem analysis, modeling, scenarios, use cases

1. Use Case Diagram

Diagram

Description automatically generated

Use case: **LOG IN**

Primary actor: user (Admin, Client or Employee)

Success scenario steps:

1. The user introduces their username
2. The user introduces the password
3. The user clicks on a button having written on it “LOG IN”
4. The application checks if the data given by the user is correct and if the two fields are both completed
5. The calculator opens the window needed for the type of user (admin, client, employee)

Use case: **MANAGE PRODUCTS**

Primary actor: Admin

Success scenario steps:

1. The user introduces the title, rating, calories, proteins, fats, sodium and price for the new menu
2. The user clicks on a button having the symbol containing the words “modify”, “delete” or “add”
3. The app checks the input text if it is correct
4. The app executes the operation selected by the user
5. The product is modified in the list of menu items

Use case: **GENERATE REPORTS**

Primary actor: Admin

Success scenario steps:

1. The admin chooses which reports they want to execute
2. The admin completes the required fields
3. The user clicks on a button having written on it “GENERATE REPORTS”
4. The application checks if the data given by the user is correct and if all fields are completed in order to generate a correct report
5. The calculator opens the window containing a .txt file (or more) containing the wanted reports

Use case: **SEARCH PRODUCTS**

Primary actor: Client

Success scenario steps:

1. The client completes one or more fields based on the search criteria they want to find one or more disponible products
2. The client clicks on a button having written on it “SEARCH PRODUCTS”
3. The application searches for the products wanted by the client
4. The application will show all the products wanted by the client

Use case: **PLACE ORDER**

Primary actor: Client

Success scenario steps:

1. The user adds to cart some products from the products table
2. The user clicks on a button having written on it “PLACE ORDER”
3. The application checks if the shopping cart of the client is empty
4. The application will generate a bill for the client, which is shown on the screen in a new window

Use case: **SEE ORDERS IN REAL TIME**

Primary actor: Employee

Success scenario steps:

1. The employee introduces their username
2. The employee introduces the password
3. The employee clicks on a button having written on it “LOG IN”
4. The application checks if the data given by the user is correct and if the two fields are both completed
5. The calculator opens the window for employees which contains a text area where every order is shown

# Design

* 1. Package diagram

1. businessLogic package contains the classes

AccessRights: an enumeration of 3 access rights, each one corresponding to one type of user (ADMINISTRATOR, EMPLOYEE, CLIENT); BaseProduct: extends MenuItem; CompositeProduct: extends MenuItem; MenuItem: an abstract class; DeliveryService: it holds the “brain” of the application, being able to control everything that every client does; Observable; Orders; User

1. dataAccess package contains the classes

FileReader: it holds methods used for reading the .csv files of products and users; FileWriter: used for writing bills; Serializator: doesn’t work, just being cute and useless

1. GUI package contains the classes

AdminGUI: represents the visualization of the user interface for a user admin; ClientGUI: represents the visualization of the user interface for a user client; EmployeeGUI: represents the visualization of the user interface for a user employee; Observer: this one doesn’t work either, my project is filled with useless stuff; LogInGUI: this is the first face of the project and it is also the only user interface that is colored and pretty.

* 1. Class diagram

Diagram, schematic

Description automatically generated

# Implementation

In this part of the documentation, each class will be discussed and have its functionalities presented.

* 1. View

View is responsible for arranging the window/s that will be seen by the user, essentially designing the layout of the application. In order to create a user-friendly interface, I used Group Layout. Group Layout works with the horizontal and vertical layouts separately. The layout is defined for each dimension independently. You do not need to worry about the *vertical* dimension when defining the *horizontal* layout, and vice versa, as the layout along each axis is totally independent of the layout along the other axis.

When focusing on just one dimension, you only have to solve half the problem at one time. This is easier than handling both dimensions at once. This means, of course, that each component needs to be defined twice in the layout. If you forget to do this, Group Layout will generate an exception. Group Layout uses two types of arrangements: sequential and parallel, combined with hierarchical composition.

1. With **sequential** arrangement, the components are simply placed one after another, just like Box Layout or Flow Layout would do along one axis. The position of each component is defined as being relative to the preceding component.

layout.setHorizontalGroup(layout.createSequentialGroup()  
 .addGroup(layout.createParallelGroup(GroupLayout.Alignment.*LEADING*)  
 .addComponent(firstInputLabel)  
 .addComponent(secondInputLabel)  
 .addComponent(freeSpaceLabel)  
 .addComponent(resultLabel))  
.addGroup(layout.createParallelGroup(GroupLayout.Alignment.*LEADING*)  
 .addComponent(firstInputTextField)  
 .addComponent(secondInputTextField)  
 .addComponent(freeSpaceLabel)  
 .addComponent(resultValueLabel))  
.addGroup(layout.createParallelGroup(GroupLayout.Alignment.*LEADING*)  
 .addComponent(addButton)  
 .addComponent(multiplyButton)  
 .addComponent(freeSpaceLabel))  
.addGroup(layout.createParallelGroup(GroupLayout.Alignment.*LEADING*)  
 .addComponent(subtractButton)  
 .addComponent(divideButton)  
 .addComponent(freeSpaceLabel))

.addGroup(layout.createParallelGroup(GroupLayout.Alignment.*LEADING*)  
 .addComponent(differentiateButton)  
 .addComponent(integrateButton)  
 .addComponent(helpButton)));

1. The second way places the components in **parallel**—on top of each other in the same space. They can be baseline-, top-, or bottom-aligned along the vertical axis. Along the horizontal axis, they can be left-, right-, or center-aligned if the components are not all the same size.

layout.setVerticalGroup(layout.createSequentialGroup()  
 .addGroup(layout.createParallelGroup(GroupLayout.Alignment.*BASELINE*)  
 .addComponent(firstInputLabel)  
 .addComponent(firstInputTextField)  
 .addComponent(addButton)  
 .addComponent(subtractButton)  
 .addComponent(differentiateButton))

.addGroup(layout.createParallelGroup(GroupLayout.Alignment.*BASELINE*)  
 .addComponent(secondInputLabel)  
 .addComponent(secondInputTextField)  
 .addComponent(multiplyButton)  
 .addComponent(divideButton)  
 .addComponent(integrateButton))

.addGroup(layout.createParallelGroup(GroupLayout.Alignment.*BASELINE*)  
 .addComponent(freeSpaceLabel)  
 .addComponent(freeSpaceLabel)  
 .addComponent(freeSpaceLabel)  
 .addComponent(freeSpaceLabel))  
 .addComponent(helpButton)

.addGroup(layout.createParallelGroup(GroupLayout.Alignment.*BASELINE*)  
 .addComponent(resultLabel)  
 .addComponent(resultValueLabel)));

Each button has set an action command, so the controller will be able to complete the tasks it needs to do:

addButton.setActionCommand("ADD");  
subtractButton.setActionCommand("SUBTRACT");  
multiplyButton.setActionCommand("MULTIPLY");  
divideButton.setActionCommand("DIVIDE");  
differentiateButton.setActionCommand("DIFFERENTIATE");  
integrateButton.setActionCommand("INTEGRATE");  
helpButton.setActionCommand("HELP");  
  
addButton.addActionListener(this.controller);  
subtractButton.addActionListener(this.controller);  
multiplyButton.addActionListener(this.controller);  
divideButton.addActionListener(this.controller);  
differentiateButton.addActionListener(this.controller);  
integrateButton.addActionListener(this.controller);  
helpButton.addActionListener(this.controller);

* 1. User

This class represents a user. A monomial is, roughly speaking, a person that uses the application. Clearly, the class attributes are:

* userID: a unique identifier that is different for each user. It is an integer bigger than zero. It only contains numbers from 0 to 9. It is not bounded by any restrictions
* username: a unique username for each client, administrator or employee. It is a string and it doesn’t matter what kinds of characters it contains, as long as the user remembers it to be able to log in the application
* password: for each client, administrator or employee. It is a string and it doesn’t matter what kinds of characters it contains, as long as the user remembers it to be able to log in the application
* accessRights: can be CLIENT, EMPLOYEE or ADMINISTRATOR. These are set when adding the user to the .csv file and cannot be changed by anyone other than the administrator that has access to the application.
  1. Composite Product

This class represents a Menu Item as a collection of Base Products.

* subMenuItems: a list (ArrayList) of Menu Items that put together results the composite product in question
  1. Menu Item

This class represents a Menu Item, meaning a product available for purchase

* Title: the name of the menu item. Can be seen by clients
* Times ordered: how many times the product has been ordered. It can only be seen by the admin when generating reports

This has to be done in order to be able to sort the list of monomials with only one single line:

@Override  
public boolean equals (Object object) {  
 if (this == object) return true;  
 if (object == null || this.getClass() != object.getClass()) return false;  
 MenuItem menuItem = (MenuItem) object;  
 return this.title.equals(menuItem.title);  
}  
  
@Override  
public int hashCode () {  
 return Objects.*hash*(this.title);  
}

* 1. Operations

This class implements each of the 5 operations that can be done on 2 polynomials.

ADDITION:

public static String add(Polynomial first, Polynomial second) throws ParseException {  
  
 for(Monomial m1 : first.getMonomials()){  
 boolean thereIs = false;  
 for (Monomial m2: second.getMonomials()) {  
 if(m1.getPower()==m2.getPower()){  
 thereIs = true;  
 m2.setCoefficient(m2.getCoefficient().intValue()+m1.getCoefficient().intValue());  
 }  
 }  
 if(!thereIs){  
 second.addMonomial(m1);  
 }  
 }  
 second.simplify();  
 return second.polynomialToString();  
}

This code goes through both of the polynomials, finds the monomials with the same power and then performs the addition in m1. If, when going through the second polynomial list of monomials, the monomial will be added to the first polynomial, keeping the sign of the coefficient unchanged.

SUBTRACTION:

public String subtract(Polynomial first, Polynomial second) throws ParseException {  
 for(Monomial m2 : second.getMonomials()){  
 m2.setCoefficient((-1)\*m2.getCoefficient().doubleValue());  
 }  
 return *add*(first, second);  
}

The subtraction operation is based on addition, but this time, we have to change the signs of the coefficients of each monomial, so if the coefficient is +5, it will be transformed in -5 and vice versa.

MULTIPLICATION

public String multiply(Polynomial first, Polynomial second) throws ParseException {  
 Polynomial result = new Polynomial("");  
 for(Monomial m1 : first.getMonomials()){  
 for (Monomial m2: second.getMonomials()) {  
 Monomial mResult = new Monomial("");  
 mResult.setPower((m1.getPower()+m2.getPower()));  
 mResult.setCoefficient(m1.getCoefficient().intValue()\*m2.getCoefficient().intValue());  
 result.getMonomials().add(mResult);  
 }  
 }  
 result.simplify();  
 return result.polynomialToString();  
}

This method multiplies the *first monomial from* the first polynomial with all the other monomials from the second polynomial, and *then the second*, and so on. After all monomials have been multiplied, the result will be simplified (using the method .simplify()).

DIFFERENTIATE

public String differentiate(Polynomial polynomial) throws ParseException {  
 for(Monomial m : polynomial.getMonomials()){  
 m.setCoefficient(m.getCoefficient().intValue()\*m.getPower());  
 m.setPower((m.getPower()-1));  
 }  
 polynomial.simplify();  
 return polynomial.polynomialToString();  
}

The differentiation method will be applied only to the first polynomial. Considering the input coefficients and powers integers, the differentiation will be easy to compute:

We know that (xa)` = a\*xa-1, so we apply the same formula to each monomial of the input polynomial.

INTEGRATION

public String integrate(Polynomial polynomial) throws ParseException {  
 for(Monomial m : polynomial.getMonomials()){  
 m.setCoefficient((m.getCoefficient().doubleValue()\*(1.0/(m.getPower()+1))));  
 m.setPower((m.getPower()+1));  
 }  
 polynomial.simplify();  
 return polynomial.polynomialToString();  
}

The integration method will be applied only to the first polynomial. Considering the input coefficients and powers integers, the integration will be easy to compute:

We know that (xa)` = a\*xa-1, so we apply the same formula to each monomial of the input polynomial.

# Conclusions

I cannot believe I’ve actually finished this project. I had so many mental breakdowns and wishes to quit university just because I felt like I couldn’t focus anymore on working on this pain in the ass. I wanted to do anything else and here I am now, at almost 2 a.m. writing this documentation because I don’t want to retake this class. Please give me a passing grade. Please and thank you. I’ll go study for Mr. Cret’s exam now.

# Bibliography

* <https://dsrl.eu/courses/pt/materials/PT2021-2022_Assignment_4.pdf>
* <https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>
* <https://www.jetbrains.com/help/idea/configuring-code-style.html#import-export-schemes>
* <https://dsrl.eu/courses/pt/>
* <https://www.tutorialspoint.com/swing/swing_jtextarea.htm>
* <https://www.javatpoint.com/java-jtextarea>
* <https://www.geeksforgeeks.org/java-swing-jtextarea/>
* <https://www.geeksforgeeks.org/jlabel-java-swing/?ref=lbp>
* <https://www.geeksforgeeks.org/remove-elements-from-a-list-that-satisfy-given-predicate-in-java/?ref=lbp>
* <https://www.geeksforgeeks.org/iterate-map-java/?ref=leftbar-rightbar>
* https://www.geeksforgeeks.org/treemap-in-java/?ref=lbp