**DOCUMENTATION HOMEWORK 4**

**Delivery Service**

**Chindea Tudor Eugen**

**Grupa 30423**

**Profesor Laborator Assistant Antal Marcel**

Contents

[1. Functional Requirements 3](#_Toc72484810)

[2. Objectives 3](#_Toc72484811)

[2.1. Principal Objective: 3](#_Toc72484812)

[2.2. Secondary Objectives: 3](#_Toc72484813)

[3. Problem Analysis And Use Cases 4](#_Toc72484814)

[3.1 Analysis 4](#_Toc72484815)

[3.2 Use Cases 4](#_Toc72484816)

[4. Design 6](#_Toc72484817)

[4.1. Data structures 6](#_Toc72484818)

[4.2. Class diagram 6](#_Toc72484819)

[4.3. Lambda expressions 6](#_Toc72484820)

[5. Implementation 7](#_Toc72484821)

[6. Testing 8](#_Toc72484822)

[7. Conclusions and Future Development 8](#_Toc72484823)

[8. Bibliography 8](#_Toc72484824)

# Functional Requirements

Build a delivery service manager which can compute the following operations:

* Store clients in a serializable file. Sign up and login clients. Display products and let the clients place orders.
* Administrator window to generate reports, update delete and create new products.

# Objectives

## Principal Objective:

Propose, build and test a system that is capable of storing users, reports, bills and menu items. Users must be able to log in. Each user based on the user type will have a different window opened for the application. The three user types are the client, employee, administrator.

New clients can create new accounts and log in with them, the account will be stored. Clients can search for products based on multiple fields and place orders.

Administrators can generate different kinds of reports and can manage the products.

## Secondary Objectives:

|  |  |  |
| --- | --- | --- |
| **Secondary Objectives** | **Description** | **Chapter** |
| Development of use cases and scenarios | The application should be able to deal with many types of input and many scenarios. The input should be easy and natural for the user to type. It could be used for any food delivery service. | 3 |
| Choosing the data structures | The right data structures enable an easy manipulation of the data stored in the tables. Three basic classes will be implemented and all operations will be done on them. The data is serialized to have it stored for the next time the program is opened | 4 |
| Division into classes | Dividing the methods into separate classes makes the code easier to understand. The DeliveryService class represents the class in which all data important manipulation methods are written and executed. | 4 |
| Algorithm development | The algorithm implementation was simple once the data was stored properly using a composite design pattern. | 4 |
| Solution implementation | Serialization is used for storing data. Each user type has different functionalities. | 5 |
| Testing | For the product management and the menu items testing can be done in the user interface to add, delete, modify and create menu items. The user functions can be tested in the U. I. | 6 |

# Problem Analysis And Use Cases

## 3.1 Analysis

The composite pattern is meant to allow treating individual objects and compositions of objects, or “composites” in the same way. It can be viewed as a tree structure made up of types that inherit a base type, and it can represent a single part or a whole hierarchy of objects. We can break the pattern down into:

component – is the base interface for all the objects in the composition. It should be either an interface or an abstract class with the common methods to manage the child composites.

leaf – implements the default behavior of the base component. It doesn't contain a reference to the other objects.

composite – has leaf elements. It implements the base component methods and defines the child-related operations.

client – has access to the composition elements by using the base component object. Hash table based implementation of the Map interface. This implementation provides all of the optional map operations, and permits null values and the null key. (The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls.) This class makes no guarantees as to the order of the map; in particular, it does not guarantee that the order will remain constant over time.

This implementation provides constant-time performance for the basic operations (get and put), assuming the hash function disperses the elements properly among the buckets. Iteration over collection views requires time proportional to the "capacity" of the HashMap instance (the number of buckets) plus its size (the number of key-value mappings). Thus, it's very important not to set the initial capacity too high (or the load factor too low) if iteration performance is important.

A lambda expression is a short block of code which takes in parameters and returns a value. Lambda expressions are similar to methods, but they do not need a name and they can be implemented right in the body of the method.

Introduced in Java 8, the Stream API is used to process collections of objects. A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result.

The features of Java stream are –

A stream is not a data structure instead it takes input from the Collections, Arrays or I/O channels.

Streams don’t change the original data structure, they only provide the result as per the pipelined methods.

Each intermediate operation is lazily executed and returns a stream as a result, hence various intermediate operations can be pipelined. Terminal operations mark the end of the stream and return the result.

## 3.2 Use Cases

Use Case Name: Place an Order.

Actors: Person using the application.

Triggers: Person presses the button to place the order

Preconditions: The user has printed the items and has selected the items he wants to order one by one.

Post Conditions: The data will be stored in a hashmap for future operations that can be done by the administrator. The user

Normal Flow:

1. The user prints the items he wants to view. He can print all items by the click of a button or he can filter the items by any field he wants
2. The user selects each item he wants to buy one by one by clicking on each row in the table and by pressing the select button.
3. The user presses the button to store the order and a bill will be created in .txt format with the data of the order. The order will be stored in a hashMap

Use Case Name: Updating a product.

Actors: Administrator using the application.

Triggers: The user types in the new values and presses the button to update a product.

Preconditions: The user must click on the item he wants to update. The user has typed the data he wants to update in his product. If a field is empty, the old data will be kept.

Post Conditions: The data will be stored in a new object that is added to the end of the list. The old object will be deleted

Normal Flow:

1. The user has clicked on the item he wants to update
2. The user will type in any of the next textboxes the new data he wants to have for the given product
3. The user presses the button to update the product table. The old item will be deleted and a new one will be added with the update values

Use Case Name: Filter data.

Actors: Client using the application.

Triggers: The client types in the fields by which he wants to filter the data and presses the filter button.

Preconditions: If no fields are enterd, the entire table is displayed.

Post Conditions: If any data can be displayed, the tableview will contain it. Normal Flow:

1. The client enters filtering criteria.
2. The client type the button to print the data.

# Design

## Data structures

The data will be stored in a form such that it would be easy to be interpreted. We use ArrayLists to store the users and the menuItems. For the orders we use Hashmaps. To generate the reports, streams are also used for the lambda expressions. A Composite design pattern is used on the menuItems and three different classes are created.

## Class diagram

Diagram

Description automatically generated

## Lambda expressions

The first report uses the date of each order. A list of orders is generated with the corresponding time. We go through each order and check the time

The second report uses lambda expressions to find the items that were used more than x times. X is typed in by the administrator. At the begging a list of all items is created. The second expression creates a list only with the items needed. This is done by checking the frequency of each item and adding it if it is higher then our desired number. Duplicates are entered in the list but we print in the report each element one time.

The third report finds the clients that have ordered a minimum number of times and all orders are over a given value. First we use lambda expressions to make a list of all orders with total price over a given value. The second lambda expression checks the second list of orders and takes the clients Names that appear more than the minimum number of orders. Duplicates might appear but they are not printed in the report.

The fourth report shows the products ordered on a specific day and how many times each product was ordered. The lambda expressions used here are for selecting the orders made on the specific day. The frequency of each product will be computed afertwards.

# Implementation

The program is composed of different packages. The Business package contains the Interface implemented by the delivery Service with all the functions . It contains the MenuItems and the users . Has the function used to read the input data from a csv file and has the functions used to create new menu items, to modify and delete existing menu items. The user data and meu items extend the serializable class.

The Data package is used for a filewriter and for the serializer. The functions more used are for serializing and deserializing all the data when objects are modified ore the program is opened or closed. The filewrite is used to creae the bill, to create reports.

The presentation package includes the controllers, in which all button actions are done and the entire user interface is coded. The AdministratorUI has function for udating, deleting and creating new items. It has a tableview showing the modifications at each step. The Administrator can also create 4 different types of reports. These reports will be printed in txt files. Each report I screated by using lambda expressions and stream processing.

Graphical user interface, table

Description automatically generated

The ClientUI class controlls the login function, the registration function for new clients and all the functions a client has in its menu. If a client wants to login with invalid username and password, a message will appear on screen. The client once it is loged in can place multiple orders and can search through the menu by filtering the item he sees based on all possible fields. The client can place orders of one or more items, items can be selecte dmore times and tanfter confirming the order, tha data will be stored in a hashmap. The content of the hashmap will pe serialized when the program is close and deserialized when the program is opened. A bill is generated printing the date of the order, the name of the client and all the items orederd, with individual price and the total price of all items.

# Testing

For each function testing can be done in the user interface. Tha data for modifying the menu will appear in the tableview after each modification and the orders and clients data can be seen in txt files. All buutons are functional and are used in the program. The main functions that can be tested are the report creation, the menu item manipulation and the client registration. We can also see the filtering criteria working in the clients user interface.

# Conclusions and Future Development

This application is useful for working with java serialization, lambda expressions and stream processing, hasmaps, Composite design patterns. This application can be used by any restaurant that wants to have a user interface and that wants to use a delivery service. The user interface is easy to use, more complicated for the administrator but we assume he gets used to it. The serialization is a useful method for storing data after each application call and each data modification. Future development can be done by cresting an employee window that receives update every time an order is processed.

# Bibliography

1. Laboratory and Course material
2. <https://www.geeksforgeeks.org/stream-in-java/https://www.w3schools.com/mysql/mysql_ref_functions.asp>
3. <https://www.w3schools.com/java/java_lambda.asp>
4. <https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html>
5. https://www.baeldung.com/java-composite-pattern