CANKAYA UNIVERSITY

Software Engineering Department

Name and Surname	: Barış Cem Bayburtlu
Identity Number	: 202228009
Course	: SENG271
Experiment	: Experiment 2
Subject	: Project Analysis & Design Draft
Data Due	: Thursday, 7 December 2023, 11:59 PM
E-Mail	: c2228009@student.cankaya.edu.tr

REQUIREMENT ANALYSIS

PROBLEM

We are going to build an **online shopping site** called **MyBazaar** for this programming assignment. This site **has a number of features** that we think can be complicated, such as **registering users**, **items to sell**, **shopping concepts**, and **general actions** of an online shopping site. Our program is going to **get the necessary information from files** and **automatically handle them** in our code.

GOALS

This project will give us **the knowledge** that will allow us to better **understand the relationship between the files** themselves, and will teach us **how to create a medium sized project** using the **Java** language. Although the documentation is **longer** than the **previous** programming assignment, it allows us to do most things in a **more understandable way** thanks to the **detailed explanation**. Although it may seem **very detailed**, I think that **implementing** most things will **not take that long**. We will also learn **file manipulation/splitting file line by line** with Java and **integrate it into the code** which we need in order to get our **Users, Commands and more**.

INPUTS

The files that we want to **include** in our program in this project are the **files that we need to pass as arguments** when we run the file. The files are **being seperated** with **<TAB>,** so there are a lot of things we need to check when **adding files into our program**. We need to type **check the strings** we splitted with <TAB> and **convert them to array** one by one. Let's take the **user input file** as an **example**. If the first string of the array is ADMIN, there should be a **total of 6 elements** in this array (**ADMIN**, **name**, **email**, **birthDate**, **salary**, **password**). If there are **extra elements**, or if the first string is not **ADMIN**, **TECH**, or **CUSTOMER**, there is **a problem in that line** of the file we were given. Since we think **there is a problem**, we will **continue to look** at the other lines as if that line **did not exist**. I said that there is a **type check** that needs to be done **on all array elements**, for example, we need to make sure that **salary** is a **float**.

OUTPUTS

First of all, **all output** is written **directly to the console**, i.e. we do not **create** a **new file for** the **output**, but we receive the **commands through a file**. There are **special outputs** for all events in the **Event/Actions** section. For example, when **adding a new client**, if the **name of the Admin** who added it or the **Admin's password is wrong**, we will show this, or **if everything is correct**, an output like "The client you specified **has been added**" can be written. As **another option**, if you want to view the **information of a** specific **customer**, if the **admin's name is wrong** in the command section, it will **send** us an **output accordingly**.

HANDLING ERRORS

I want to start with the **members** of the program, since we will need to **write a debugging mechanism** for each file. After all, they all work in the **same way**, but we have to **take care of different types** for each of them. We definitely need to **check if the files** we requested at the very beginning **have been given or not**. If not, there **will be no situation** where we will **create error handling**, and we can directly **close the program** by saying that the **requested data** has **not** been **given** to the program. I can give an **example** of this problem with the changes that exist **when adding Admin**, **Customer** and **Technician**. For example, we read the file where the members are **written line by line** and we come **across a case** where the **first word** at the beginning is not ADMIN, TECHNICIAN or CUSTOMER. Then we can tell the **program that this line is written incorrectly**. Let's say that this line skipped this part of the program extraction and we are **trying to add an admin** to the program. First we need to **start the line with** the keyword **ADMIN**. If there are **not 6 elements**, we can say that this **line is wrong**, because **we need** exactly **6 elements** (keyword **ADMIN**, **name**, **email**, **date of birth**, **balance** and **admin password**). Same for the events/actions part, for example, let the customer **try to buy something** from the **customer's shopping cart**. If the **shopping cart is empty**, the customer **will not be able to buy** anything, so we need to check that.

DESIGN

ALGORITHMS

I will **store** all members **in the system** and give us via file via **ArrayList**. I will probably open **two different ArrayLists** as **Customers** and **Employees**, I am **not very sure** about this but I think it will be **better regarding performance**. I plan to use a **HashTable** for the **products** that will be given from the file, because **each product already has an ID**, I think it will be **easy to store the products** with **key/get logic** using these IDs and I think it will make a **great contribution to us** in terms of **performance**.

PROS and CONS

ARRAYLIST

PROS:

Dynamic Size: Can dynamically grow or shrink in size.
Random Access: Allows fast access to elements using indexes.
Type Safety: Supports type parameterization, reducing runtime errors.

CONS:

Resizing Overhead: Dynamic resizing can incur performance overhead.
Memory Wastage: May allocate more memory than needed due to resizing.
Slower Insertions/Deletions: Insertions and deletions in the middle are slower compared to LinkedList.

HASHTABLE

PROS:

Fast Retrieval: O(1) avg time complexity for search and insert operations.
Efficient Data Retrieval: Well-suited for scenarios with large datasets.
Dynamic Size: Can dynamically resize to maintain performance.
Versatility: Widely used in various applications like databases, caches, and language implementations.

CONS:

Resizing Overhead: Dynamic resizing can incur performance overhead.
Memory Wastage: May allocate more memory than needed due to resizing.
Slower Insertions/Deletions: Insertions and deletions in the middle are slower compared to LinkedList.