12174521 Assignment 2 - Reflection I

Existing Class Changes

As this app is based on moneyless payment and credit top-up which is provided by another app, I don't plan to have any coin/currency class or coinset. Credit will be a field in the new Client class and will be kept track of in that class in Euros. Also products has a 'price' field which is also be in euros.

Commented [UW1]: I did infact get rid of the coin class and as planned I added a price field into products and a credit field into client which use euros

The VendingMachine() class remains but will be changed. The fields will now be a an object of ReadUserData class which will read all the user Data into the vending Machine from csv files as an ArrayList. A WriteUserData class which will write the data back to the files. An arraylist of locations as the vending machine has locations to store the products and an Object of ReadWriteProducts which will read the csv file of products to an ArrayList and store new number of products on 'shutdown'.

Commented [UW2]: The Vending machine class remained(called VM). I put the read user data class and write user data class into one class called ReadWriteUserData as it made more sense and meant less need to pass data through parameters needlessly.I also added the string fields for all the filepaths to make it easier for a user to adapt to putting in their own files, rather than having to search code to see where to change hardcoded paths.

Some methods in the Vending Machine class will remain. Such as getProductList(), addProduct(), and removeProduct() for when products are purchased or 'reloaded'.

Commented [UW3]: Yes these method were created. I also added a shutdown method thatwrites all the data in the user arraylist and product arraylist back to the csv files.

The Vending Machine Menu will remain but be will become an interface and 2 menus will be created. One for Client user and one for Admin user. These classes will contain GUI components, to create the 'look' of the menu and the menu will call the different methods of the Vending machine class for its logic to perform the operations.

Commented [UW4]: I was not able to achieve this in the way that I had planned. Due to the structure of my GUI, I was unable to find a way to create a menu class that I could then add as a field in the user class that could be used within the vm class to correctly display data. Due to the GUI being set up with a left and right side and the need for certain actions for buttons such as buy and reload, it was impossible for me to find a way to pass data back to the menu within the user class and have uptodate product data from the VM class. As a work around, instead I added a 'type' field to user class and use this on login to call getClientMenu() and getAdmin Menu() methods in the GUI class to change the GUI according to the type of user logging in. It is not as tidy as I wanted but functions perfectly well.

Product class with remain, with added field of locationId and exisiting methods of getters and setters.

Commented [UW5]: Yes product class remained very much as I expected.

New Classes

I plan to create an abstract User class as a superclass that will have 2 subclasses: Client and Admin. These classes will have fields of username, (credit for client) and password. The user class will have an authenticate method used by both subclasses. They will both have a get Menu method which will allow a different menu to be loaded on login based on the type of user(client or admin).

Commented [UW6]: I created the superclass user though this was not an abstract class, it had no abstract methods. It worked very well to make an arraylist of users that could then authenticate and identify which type of user it was for data reading and writing and to get the correct GUI display for the user.

Location Class will be created to keep track of where all products are loaded in the machine. Methods such as add/remove products.

Commented [UW7]: Location class was created and stores all the products in the vending machine.
I had to add a products count for the locations as if a product is 'topped-up' I needed a way to get the quantity of 'all' the products in the location and also writes this back to the file on shutdown.

ReadUserData class will contain an arraylist of all user both client and admin(created by reading in csv files from readClass. this will have a login method that used with the authenticate method in user will enable ability to find user and type of user on login.

Finally, there will be a GUI class that will contain the GUI components that will take user input and contain object of VMto provide logic.

Classes Overview

1. Class User() - protected String username; protected String password; protected Menu Menu; protected String type

authenticate(), getName(), setName(), getPassword(), setPassword, gettype(), getMenu()

3. Client Class extends User - private double credit

getCredit(), buy()

4. Admin Class extends User

(Fields and methods as in User)

5. Class Location() - private String position, private int productsCount, private Product product.

resetProduct(), addProduct(), uploadProductList(), getPosition(), setPosition(), getProductsCount, getProduct()

6.Class ReadUserData - private static ArrayList<User>users;

readFile(), getClientCredit(), getArray(), Login(), getUserName(), getUserPassword()

7.WriteUserData - private ReadUserData object;

writeClientData()

Put into one class ReadWriteUserData()

8.ReadWriteProduct - private ArrayList<Product>products

readProduct(), writeProducts(), getProduct(), getProductName(), getProductLocation(), getProductPrice(), getProductQuantity()

9.Class VendingMachine - private ReadWriteUserData object, private ReadWriteProducts object, private ArrayList Locations

login(), buy(), removeProduct(), addProduct(), displayProducts(), displayLocations(), getProductCount(), getProductPrice(), getProductName(), getProductQuantity(), getProductLocation(), getUser().=, getUserType(), getUserCredit(), shutdown()

Commented [UW8]: As mention I created a read and write class rather than 2 seperate classes. I also created a read and write product class.
Finally, there will be a GUI class that will contain the GUI components that will take user input and contain object of VendingMachine to provide logic.

Commented [UW9]: My GUI does create an object of vending machine, I altered the VM class a little to have a constructor that takes in file paths that make it easier for the user to be able to specify their own files and paths rather than using the hardcoded ones. In this way if an object on VM is created, then the filenames need to be inputted as parameters and there are no other changes for files that need to be made in code.

Commented [UW10]: I put read and write user data class into one ReadWriteUserData() class.

10. VendingFX - private VendingMachine object

login(), getUserMenu, getAdminMenu(), logout(), shutdown(), buy(), addProduct()