Ryerson University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	Object Oriented Eng Analysis Design
Course Number:	COE 528
Semester/Year	W2024

Instructor:	Boujemaa Guermazi

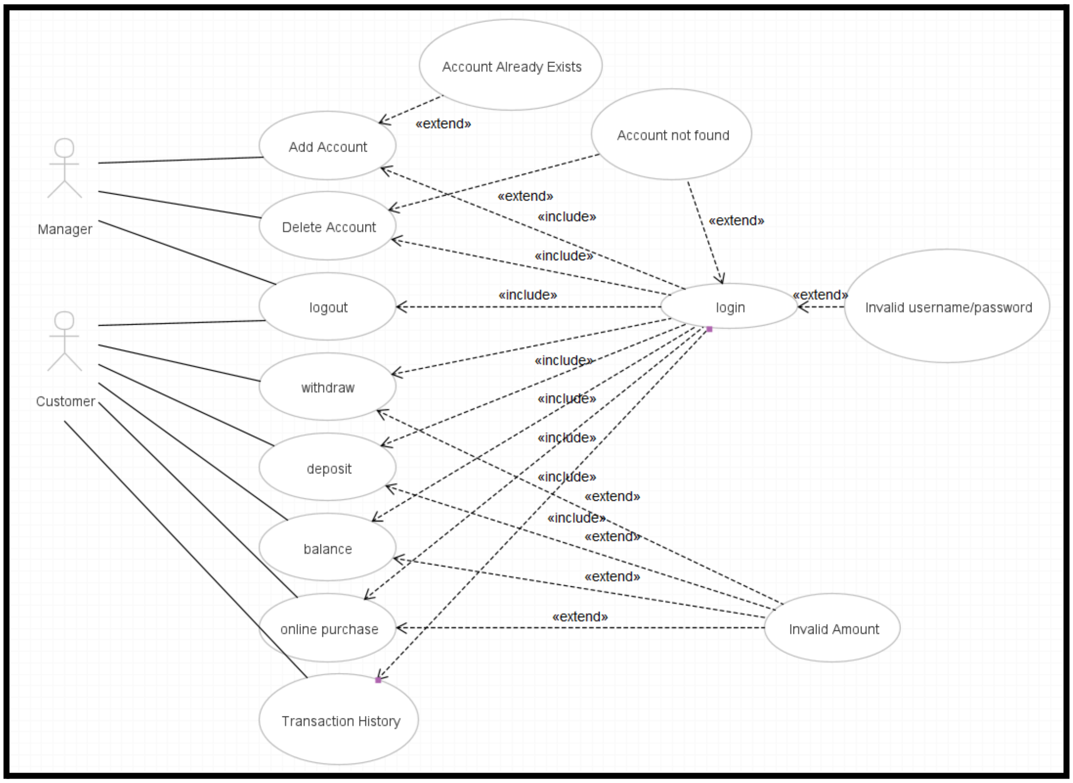
Assignment/Lab Title:	Term Project: Winter 2024

Submission Date:	March 25, 2024
Due Date:	March 25, 2024

Student's Last Name	Student's First Name	Student #	Section	Signature
Kamboj	Divjot Singh	501165887	12	DSK

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: http://www.ryerson.ca/senate/current/pol60.pdf

Use Case Diagram

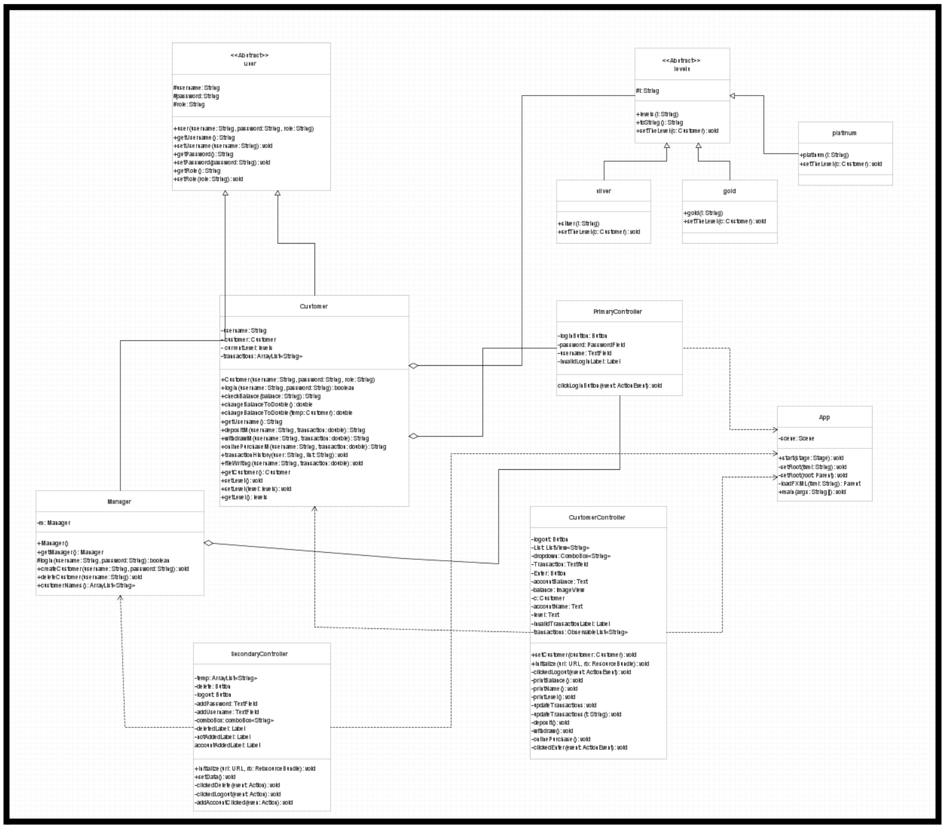


Above is a diagram which depicts the use cases according to the certain actors. The customer and the manager are both actors and have access to login. Upon login as the manager you are giving the set cases of being able to Add Account, Delete Account and logout. The Exit conditions for Add account is successfully being able to add an account or the account already existing. The exit conditions for delete account would be successfully deleting an existing account or the account not existing in the first place. For logout the exit condition is clicking the logout button and entering the login screen. When logging in as the customer you are given the set cases of being able to withdraw, deposit, online purchases, check balance and view transaction history. All of these have an exit condition when an invalid amount has been entered into the system. The login itself has exit cases which include correctly logging in as either manager or customer and have an invalid username/password.

Use case name:	Add Account
Participating actor:	Manager
Flow of events:	1. User logs in as a Manager successfully 2. The Manager enters a username and password into the according text fields and clicks the add account button.

	3. A label will appear and indicate whether the account was added successfully or not.
Entry Condition:	● The user is logged in as Manager
Exit Condition:	● The Manager has received an acknowledgment and the submitted account has been successfully added, OR ● The manager has received an acknowledgment and the submitted account has not been added successfully.
Quality requirements:	● The username and password can only contain letters and numbers ● No two customers can have the same username

Class Diagram



<<Abstract>>
user
#username: String
#password: String
#role: String
+user(username: String, password: String, role: String)
+getUsername(): String
+setUsername(username: String): void
+getPassword(): String
+setPassword(password: String): void
+getRole(): String
+setRole(role: String): void
<<Abstract>>
levels
#l: String
+levels(l: String)
+toString(): String
+setTheLevel(c: Customer): void
platinum
+platinum(l: String)
+setTheLevel(c: Customer): void
silver
+silver(l: String)
+setTheLevel(c: Customer): void
gold
+gold(l: String)
+setTheLevel(c: Customer): void
Customer
-username: String
-customer: Customer
-currentLevel: levels
-transactions: ArrayList<String>
+Customer(username: String, password: String, role: String)
+login(username: String, password: String): boolean
+checkBalance(balance: String): String
+changeBalanceToDouble(): double
+changeBalanceToDouble(temp: Customer): double
+getUsername(): String
+depositM(username: String, transaction: double): String
+withdrawM(username: String, transaction: double): String
+onlinePurchaseM(username: String, transaction: double): String
+transactionHistory(user: String, list: String): void
+fileWriting(username: String, transaction: double): void
+getCustomer(): Customer
+setLevel(): void
+setLevel(level: levels): void
+getLevel(): levels
PrimaryController
-loginButton: Button
-password: PasswordField
-username: TextField
-invalidLoginLabel: Label
clickLoginButton(event: ActionEvent): void
App
-scene: Scene
+start(stage: Stage): void
-setRoot(fxml: String): void
-setRoot(root: Parent): void
-loadFXML(fxml: String): Parent
+main(args: String[]): void
Manager
-m: Manager
+Manager()
+getManager(): Manager
#login(username: String, password: String): boolean
+createCustomer(username: String, password: String): void
+deleteCustomer(username: String): void
+customerNames(): ArrayList<String>
CustomerController
-logout: Button
-List: ListView<String>
-dropdown: ComboBox<String>
-Transaction: Textfield
-Enter: Button
-accountBalance: Text
-balance: ImageView
-c: Customer
-accountName: Text
-level: Text
-invalidTransactionLabel: Label
-transactions: ObservableList<String>
+setCustomer(customer: Customer): void
+initialize(url: URL, rb: ResourceBundle): void
-clickedLogout(event: ActionEvent): void
-printBalance(): void
-printName(): void
-printLevel(): void
-updateTransactions: void
-updateTransactions(t: String): void
-deposit(): void
-withdraw(): void
-onlinePurchase(): void
-clickedEnter(event: ActionEvent): void
SecondaryController
-temp: ArrayList<String>
-delete: Button
-logout: Button
-addPassword: TextField
-addUsername: TextField
-comboBox: comboBox<String>
-deletedLabel: Label
-notAddedLabel: Label
accountAddedLabel: Label
+initialize(url: URL, rb: RebsourceBundle): void
+setData(): void
-clickedDelete(event: Action): void
-clickedLogout(event: Action): void
-addAccountClicked(even: Action): void

The image above showcases the class diagram for the bank account application. This class diagram consists of 11 different classes. The user class is an abstract class which has two child classes, these include the Manager and Customer classes. These classes define the types of users that will be used in the application. The levels class is an abstract class which has 3 child classes that include silver, gold and platinum. These are the levels that the user can have according to the balance in their account. These three classes are the state-level classes. The three state-level classes are subclasses to the levels class and integrate an is-a relationship with the superclass. The three different controller classes which include primaryController, secondaryController, and customerController depend on the App class. The App class is the main class which controls the implementations by the GUI. The primaryController depends on the abstract user class, the secondaryController depends on the Manager class, and the customerController depends on the Customer class.

Overview

The abstract user class is the class that was selected to address point number 2 mentioned on the COE528 (Winter 2024) - project documentation.

State Design Pattern

The abstract class levels in the representation of the state design pattern. It has three subclasses which include silver, gold, and platinum. These states inherit the different levels and fees accordingly Depending on the customer's account balance the levels which are the states will change accordingly.

Resources

https://stackoverflow.com/
https://www.w3schools.com/