Toronto Metropolitan University

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

Course Title:	COE
Course Number:	528
Semester/Year (e.g.F2016)	Winter/2024

Instructor:	Professor. Boujemaa

Assignment/Lab Number:	Project
Assignment/Lab Title:	GUI Final Banking App

Submission Date:	2024-03-24
Due Date:	2024-03-24

Student LAST Name	**Student FIRST Name**	**Student Number**	**Section**	**Signature***
Badie	Andrew	501192204		Andrew Badie

*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: https://www.torontomu.ca/content/dam/senate/policies/pol60.pdf

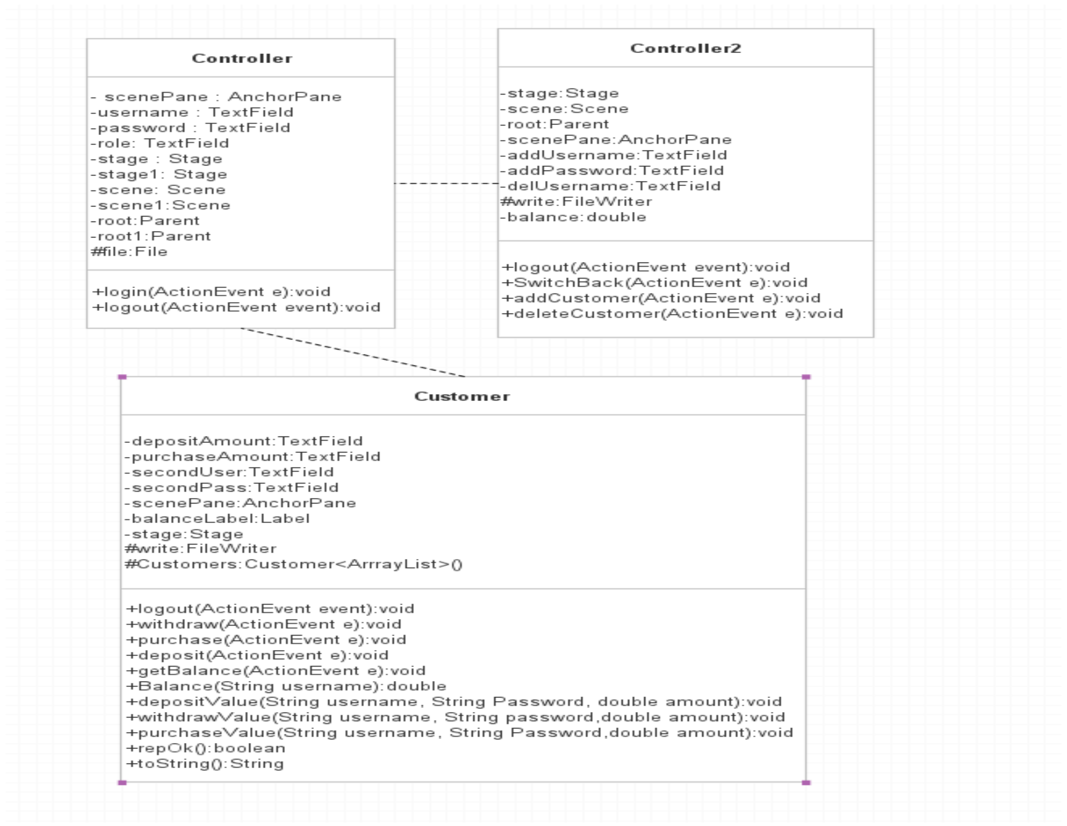
COE528 Project

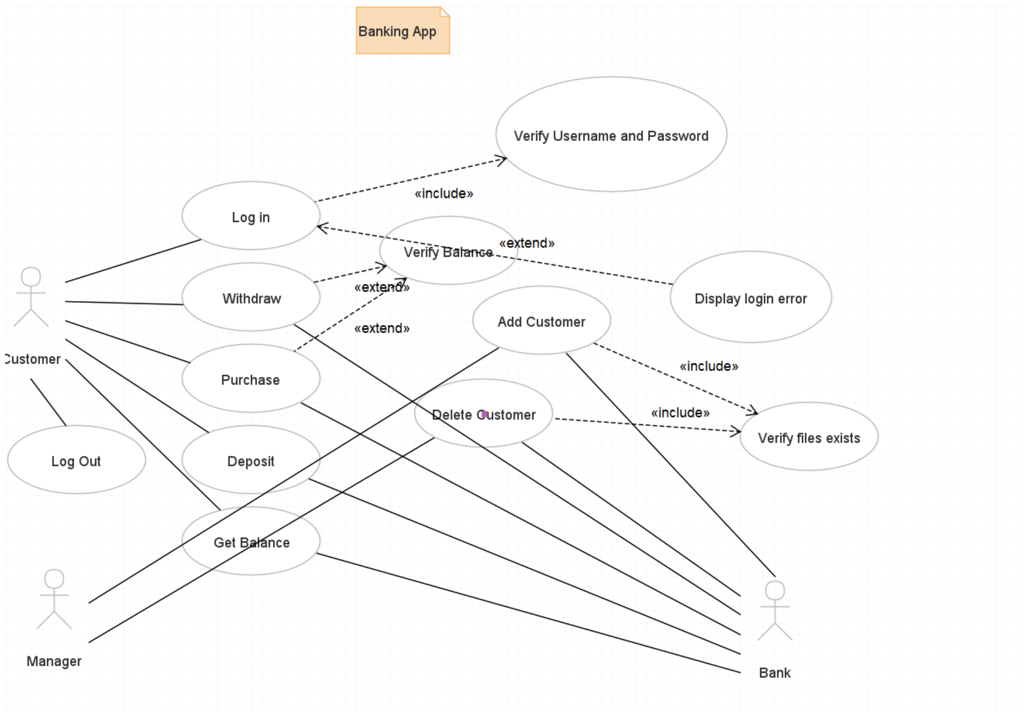
The use case diagram displays all the interactions between the external sources of the system. The three external users are the customer, manager, and the bank. The diagram shows that the roles and the interactions the customer partakes in are money transactions with his account, login and logout. The diagram also displays by the extending arrows, that for some of the methods such as withdraw and purchase. The difference between the extend and include arrow are that include is part of the method no matter the circumstance, however, the extend at times functions because of conditions that are being broken. This diagram also displays how the manager has access to add a customer and/or delete a customer, depending on the availability of the file.

The class diagram reveals also how the Contoller.java is the start of all of these java files by the dotted lines. Each of the access modifiers, type and the names of the variables are located in the second row. The third row is for the method, rather than the instance variables of the code. Each of the three classes have their own table.

I selected Customer.java to be the file that was chosen for part 2. This file's purpose as mentioned in the overview and abstraction function is that the methods written in this class are the actions the customer has authority to do. All the textfields require not be null and to be equal to their specific value type. Many of the methods rely on the ActionEvent of the method to perform the function. The methods include depositing money, withdrawing, getting the balance, and purchasing. The deposit method adds money to the account you already have and the withdrawal method subtracts for the customer's balance. The get balance method returns the double cast of the balance on the scene, and etc. The rest of the methods are similar to the deposit, but with a small catch.

The state design pattern was implemented in the code. The customer had three levels (states) he was assigned. There are conditions for each level. The customer through the Cusstomer.java code can deposit, withdraw, or purchase changing his/her level. Also, the Customer and the controller2 java files are then created. The purpose of the customer is dependent on the level of the customer. The parts that form the state design pattern are seen in how the Customer and Controller 2 class are connected to the Controller class by dashed lines. The other part is in the class diagram where it mentions the methods of purchase. This method is in charge of replacing and updating the levels of the customers.


Controller
- scenePane : AnchorPane
-username : TextField
-password : TextField
-role: TextField
-stage : Stage
-stage1: Stage
-scene: Scene
-scene1:Scene
-root:Parent
-root1:Parent
#file:File
+login(ActionEvent e):void
+logout(ActionEvent event):void
Controller2
-stage:Stage
-scene:Scene
-root:Parent
-scenePane:AnchorPane
-addUsername:TextField
-addPassword:TextField
-delUsername:TextField
#write:FileWriter
-balance:double
+logout(ActionEvent event):void
+SwitchBack(ActionEvent e):void
+addCustomer(ActionEvent e):void
+deleteCustomer(ActionEvent e):void
Customer
-depositAmount:TextField
-purchaseAmount:TextField
-secondUser:TextField
-secondPass:TextField
-scenePane:AnchorPane
-balanceLabel:Label
-stage:Stage
#write:FileWriter
#Customers:Customer<ArrrayList>()
+logout(ActionEvent event):void
+withdraw(ActionEvent e):void
+purchase(ActionEvent e):void
+deposit(ActionEvent e):void
+getBalance(ActionEvent e):void
+Balance(String username):double
+depositValue(String username, String Password, double amount):void
+withdrawValue(String username, String password,double amount):void
+purchaseValue(String username, String Password,double amount):void
+repOk():boolean
+toString():String


Banking App
Verify Username and Password
«include»
Log in
Verify Balance
«extend»
Display login error
Withdraw
«extend»
«extend»
Add Customer
Customer
Purchase
«include»
Delete Customer
«include»
Verify files exists
Log Out
Deposit
Get Balance
Manager
Bank