Ryerson University

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

Course Title:	Object Oriented Eng Analysis Design
Course Section:	COE528
Semester/Year (e.g. F2017)	W2024

Instructor	Boujemaa Guermazi

Assignment/Lab Number:	N/A
Assignment/Lab Title:	Project

Submission Date:	23/03/2024
Due Date:	23/03/2024, 23:59

Student LAST Name	**Student FIRST Name**	**Student Number**	**Section**	**Signature***
Muruganantham	Haran	501166674	04	[illegible]

*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

UML Use Case Diagram

The use case diagram for this implementation of the project shows there are two different actors. The first is the manager, and the second is the customer. For both actors, they initially use the login method, which authenticates their username and password entries and then changes them to their respective scenes. Furthermore, once in their respective scene, both actors can use the back method to take them back to the login scene, effectively logging them out. Also, because both methods have an end goal of changing the scene, these methods include the change method. The manager has two methods that only they can perform, and these are create and delete. Create allows the manager to generate a new customer file with an inputted username and password, and delete enables the manager to delete the customer file as long as the username and password match up. The customer has four methods that only they can perform. These methods are balance, deposit, withdraw, and online purchase. The first method, balance, will return the user's current balance and status level. The following three methods will perform their respective transaction and then ensure the balance, status, and user files have been updated. These three methods include the checkStatus and updateFile method to update the status and file.

Name	deposit
Participating Actors	customer
Flow of Events	1. The customer enters an double amount in the text field 2. The customer selects the deposit option 3. An alert will pop up to show the transaction success or failure
Entry Condition	customer has logged in
Exit Condition	- Successful transaction - Failed transaction
Exceptions	IOException
Quality Requirements	N/A

UML Class Diagram

The class diagram gives a better representation of how all the classes work together to achieve the goal of the functioning bank application. The main component of this application is in the Bank class. This is where the initial scene is loaded and where the scene change method occurs. The initial scene that gets loaded is the login scene, which works with the loginController class. Here, the entries for username and password are processed and authenticated. Once logged in, the user will go to either the manager or customer scene, depending on who they are. Also, these two scenes work in tandem with their respective controller class. If logged in as a manager, the user can use the create and delete method with the given username and password entry. If logged in as a customer, the loginController will create an instance of the customer object containing the customer's info. This object is created statically, allowing it to further pass through to the customerController. Due to this, there is an abstract relationship between loginController and the customer because loginController has a customer. Also, there is an inheritance relationship between loginController and customerController. The user can perform various transactional methods or check their balance as a customer. Furthermore, for both manager and customer scenes, there is an option to go back to the login scene, which effectively logs them out.

Customer Class Description

This implementation includes an overview clause, abstraction function, rep invariant, and necessary clause for the customer class. The overview clause briefly describes the class's goal and whether the class is mutable or not. The abstraction function shows how the instance variables relate to an abstract instance of an object of the class. Furthermore, the abstraction function generated the class's toString method parameters. The rep invariant describes the bounds for the various parameters used in the object constructor. This was used to create the repOk method, which tests the parameters to ensure they are safe for the constructor. Finally, the necessary clause comments such as requires, modifies, and effects were used in all of the methods to describe what the method does and how it interacts with the object.