COE528 Banking Application Report
Riya Dave|| 501189965 || Section 8 ||

Introduction:
The purpose of this lab project was to program, design and implement a banking application that can complete certain tasks based on the type of user. In addition, the graphical user interface was created by JavaFX and the entire project was coded on the IDE NetBeans 19.

Use Case Diagram:

Use Case Description

1. Use case name: Get Account Level	
2. Participating actors:	Initiated by Customer Communicates with Bank DataBase
3. Flow of events	1. Customer activates the "Get Account Level" on the banking app 2. Bank Database responds by first checking the corresponding balance of the customer 3. Bank Database reviews the balance of the customer, and selects the correct account level 4. Bank Databasesends out the correct
4. Entry Condition	1. Customer is logged into Bank Application
5. Exit condition	1. Customer has received the Account Level they are at 2. Customer had received a message explaining why the action cannot be processed at the moment
6. Quality Requirements	None

In this Use Case Diagram, it models the behaviour of the Banking Application and the user's interactions with it. In this diagram, there are 3 actors: Customer, Manager and Bank System. The Customer is the actor that interacts the most with the bank app, they can log in, log out, check their balance, and account level, make an online purchase, and deposit/withdraw money. The following interactions that it has are towards the use cases of what actions it can conduct such as "Transaction" which is extended by "Withdraw Money" and "Deposit Money". In addition to "Withdraw Money" it also "includes" the verification of sufficient funds within the bank account. The second actor is the Manager, who can log in, log out, create a new customer, and delete a customer. The interactions it is linked to are the use cases of "Add Customer", "Delete Customer" and "Logout". The last use case it

is connected to is “Login”, which has an “includes” relation that verifies the login credentials and “extends” to an error message if those credentials are false. Lastly, the Bank Database is the actor that the bank application is getting information from. The difference between the Bank Database and the Bank application is that the application is the system that the user interacts with and the system retrieves information from the database. Hence, the verifications of any action that the customer and manager do are gone through the database first.

Class Diagram:

The following class diagram is a structural representation of the Banking Application program and its classes. Each class is made through the following attributes, modifiers and methods that they possess. There are 8 classes in total that make up this banking application and the following relationships between the classes can help interpret the flow of information on how this application works.

Relationship: Inheritance

Starting with the bankAccount abstract class, this class is the parent class to the 3 other concrete state classes or “Account Levels” (Silver, Gold, and Platinum). The concrete classes inherit the functionality from this abstract class and the methods necessary for them to work such as “onlinePurchase” and “getAccountLevel”. The abstract class provides the common interface that all child classes must implement

Relationship: Aggregation

This type of relationship is described as the “has a” relationship and the arrow is currently between the bankAccount class and the Customer class. To put it in simpler terms, the bankAccout has a reference type variable of Customer, and Customer one as well. Hence, each Customer “has” a bank account once they log in or when they are created.

Relationship: Composition

The 3 classes in this type of relationship are the popUpBox, Manager, and Customer, which are highly dependent on the Bank class and are mainly what compose of the banking application. These classes cannot exist independently from the Bank class and must be all together in order for the application to function properly.

Q. Mention the class you have selected to address point number 2 (mentioned above)

The class that I have chosen to address the Overview clause, the abstraction function and rep invariant, along with its implementation (toString method, repOk method) and the requires, modifies, and effects clauses is the bankAccount class

Q. Refer to your UML class diagram and indicate the part(s) that form the State design pattern.

In the UML class diagram above, the parts that form the State design pattern and enable to the Customer to switch from one state to another (ex. Gold to Silver), is due to the abstract class of the BankApplication and the child classes of GoldState, SilverState and PlatinumState.The behavior of an object can change at runtime by changing its state. In this case, changing the state of a bank account from Gold to Silver or Platinum results in different behaviors for the onlinePurchase method.The bankAccount abstract class defines the common interface that all concrete state classes (GoldState, SilverState, and PlatinumState) must implement. This ensures that all state classes adhere to a common contract.

References:

The following references in MLA were used to assist in the completion of this project and help troubleshoot or debug any issues within the code, or with the JavaFX in particular. In emphasis to the JavaFX, since this was a fairly new concept, most of the references here are geared towards learning how to use the tool properly in order to create a functioning GUI. In addition, the professor's lecture notes for this course were used as well.

1. "JavaFX Login Form Tutorial Using Scene Builder | Javafx and Scene Builder Tutorial | 2020 Version." *YouTube*, YouTube, 23 Dec. 2020, www.youtube.com/watch?v=HBBtlwGpBek&t=71s.
2. "Release: Javafx 2.2.40." *Getting Started with JavaFX: About This Tutorial | JavaFX 2 Tutorials and Documentation*, 30 Aug. 2013, docs.oracle.com/javafx/2/get_started/jfxpub-get_started.htm.
3. "A Simple Javafx Application without FXML." *YouTube*, YouTube, 11 Nov. 2020, www.youtube.com/watch?v=zRLbrkXMcd8.