Assignment 2 Java ATM Program

Introduction to Software Development

**SWE4101**

Date of Submission: 01 November, 2021.

Table of Contents

[1.0 Introduction: 2](#_Toc86703255)

[2.0 Requirement Analysis: 3](#_Toc86703256)

[3.0 Design Analyse: 4](#_Toc86703257)

[**3.1 Customer:** 4](#_Toc86703258)

[**3.2 Bank:** 4](#_Toc86703259)

[**3.3 ATM:** 4](#_Toc86703260)

[**3.4 Transaction:** 5](#_Toc86703261)

[**3.5 Account:** 5](#_Toc86703262)

[**3.6 UML class diagram:** 5](#_Toc86703263)

[**3.7 Use case diagram:** 6](#_Toc86703264)

[4.0 Discussion of the solution: 7](#_Toc86703265)

[**4.1 ATM:** 7](#_Toc86703266)

[**4.2 User:** 8](#_Toc86703267)

[**4.3 Bank:** 9](#_Toc86703268)

[**4.4 Account:** 10](#_Toc86703269)

[**4.5 Transaction:** 11](#_Toc86703270)

[5.0 Testing: 12](#_Toc86703271)

[**5.1 Test with correct details:** 12](#_Toc86703272)

[**5.2 Testing User menu option no 3:** 12](#_Toc86703273)

[**5.3 Testing User menu option no 2:** 13](#_Toc86703274)

[**5.4 Testing User menu option no 4:** 14](#_Toc86703275)

[**5.5 Testing User menu option no 1:** 14](#_Toc86703276)

[**5.6 Testing to withdraw more than account balance:** 15](#_Toc86703277)

[**5.7 Testing User menu option no 5:** 16](#_Toc86703278)

[6.0 Error handling and exception: 17](#_Toc86703279)

[**6.1 Incorrect input:** 17](#_Toc86703280)

[**6.2 Invalid choice:** 17](#_Toc86703281)

[**6.3 Unresolve exception:** 19](#_Toc86703282)

[7.0 Conclusion: 20](#_Toc86703283)

[Reference: 21](#_Toc86703284)

[Word count: 1966 22](#_Toc86703285)

# **Introduction:**

In this stage of the learning; we implement some core programming concept with a real life system that has been most widely used, the ATM machine. The assignment brief clearly outlined the requirement with assumptions. Whilst, the aim is to produce a system closed to a real life ATM fortunately; java have built in class and library available to make that possible.

We are using Eclipse IDE for Java Developers (includes Version: 2021-09 (4.21.0). The project is done using macOS Big Sur Version 11.4 MacBook Air Processor 1.6 GHz Dual-Core Intel Core i5.

# **Requirement Analysis:**

Before commencing design, the software development process starts by presenting a requirements document that specifies the overall purpose of the ATM program and what it should do. Throughout the case study, we refer to the requirements document to decide precisely what functionality the system must include.

Automated teller machine (ATM) used by customers in order to perform basic financial transactions. Each user can have one or many account at the bank. ATM users should be able to:

* Use credentials to log in
* View account history.
* Withdraw funds from the account.
* Transfer funds between the account.

In this case the PC monitor to represents ATM's screen, and the keyboard to obtian the ATM's input as keypad. ATM session starts with authenticating a user; as the user insert the bank card followed by four digit personal identification number (PIN), in order to execute financial transactions, the ATM must interact with the bank's account information database however, for simplicity we are using array list instead of database. System like an ATM complicated security problems that go well beyond the scope of this course.

# **Design Analyse:**

Design is a pivotal element that will reveal the blueprint of the project. In this phase a class diagram introduces to identify the objects and relationship that direct the dependency among various functions working together as a program. A class is a blueprint for objects, and an object is an instance of a class. When the individual objects are created, they inherit all the variables and methods from the class. (W3schools.com, 2021)

Class diagram mainly drawn by extracting keywords and noun phrases from the assignment description that models an ATM system. Prior to that let's list out the classes with its property and relationships using less technical jargon.

## **3.1 Customer:**

The centre of the attention lies on the user who is a register bank customer. The customer class represents the end user of the system. This customer has a name, card number, and pin. For this person to be considered a customer, they must have an account. In fact, it’s a bi-directional association that is drawn using a blank line no arrows. Customer may have 1 to many accounts.

## **3.2 Bank:**

The bank class act as the consortium. In practice bank has a location and a unique id, it has been simplified by not using any location or multiple branch in this case. Bank also manages several accounts. There's a clear indication of association here as account 'part of' a bank aggregation.

## **3.3 ATM:**

The ATM class represents a ATM as an entity. In practice, an ATM has a location and is managed by a bank. Following the same analogy let’s simplified with no location even though managed by bank. This class seeks user input; handle the credentials by validating user. The system then prompts to the user menu to conduct the basic functionality based on that final output will be displayed. From programming perspective this is the class where the main method going to run; fundamentally the entry point of the program.

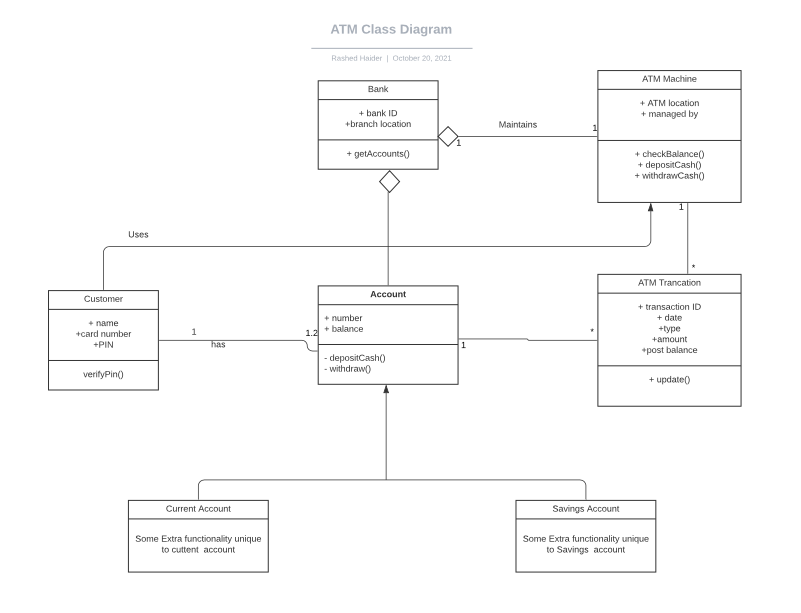
## **3.4 Transaction:**

ATM transaction is fundamentally play the role ledger keeping class. A valid transaction should hold the basic information like transaction ID, transaction type, amount, and any relevant note that user like to leave as a reference. This class will return an update amount to the relevant account.

## **3.5 Account:**

The account class represents a bank account. Common attributes of bank accounts include account number, account type, balance. Banks offers numerous types of accounts: a current account a savings account. This indicates the superclass inherits two sub class. We'll denote this by using a solid black line with an unfilled arrow going into the account class. accounts include account number, balance, etc. We have two inherited class for account namely Current Account Savings Account. The account class has get the update method from transaction class and updates current balance.

## **3.6 UML class diagram:**



Further, a simple use case diagram can be illustrated to draw the relationship between user and system. The tool is however, taught in another module.

## **3.7 Use case diagram:**

View account balance

Withdraw

Transfer Funds

Exit

Deposit Funds

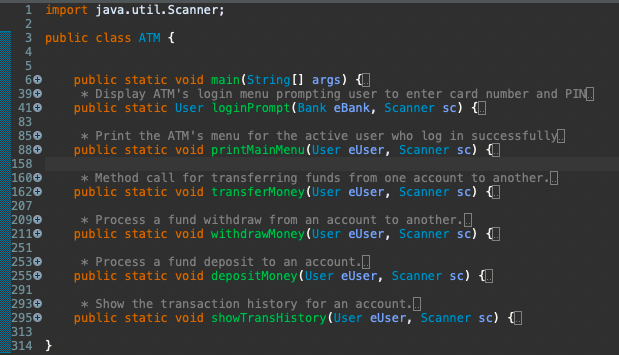
# **4.0 Discussion of the solution:**

Keeping the above analogy, implementation are now ready to commence into the java programing language.

To simplify the objective and requirement, a console program will be made. For saving the data, array list will be used. As the discussion expands the focus on the built in java class followed by methods and variable used in the class. Throughout the code appropriate commenting is been used that will require little explanation.

## **4.1 ATM:**

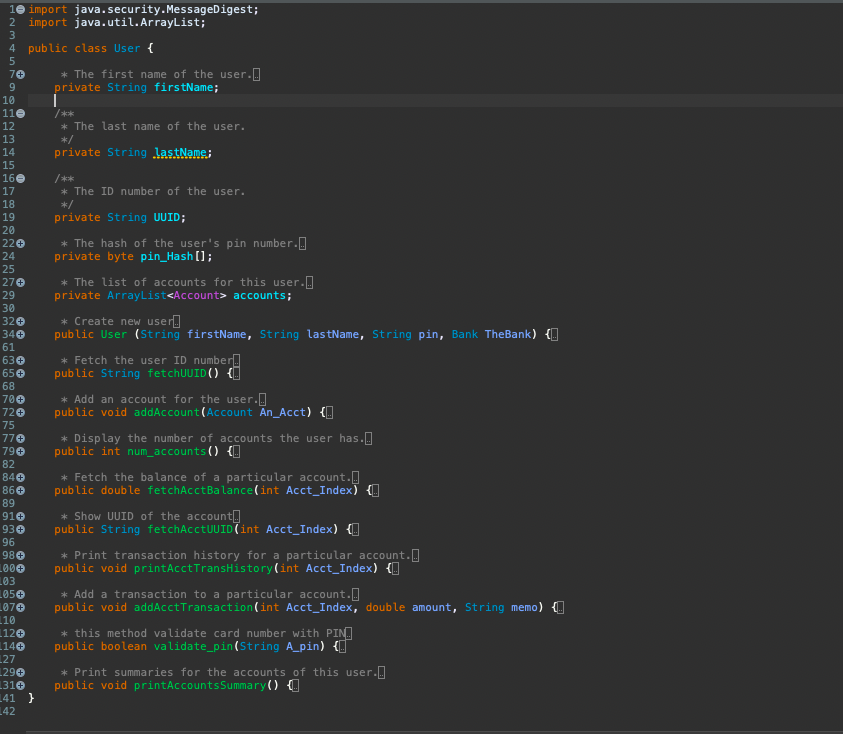
ATM is class where main method is located; they entry point of the program. Scanner class is imported to get the user input. The log in method validate the card and PIN and upon successful entry the program proceed to the main menu function. User make selection to and switch statement used to execute the operation.



## **4.2 User:**

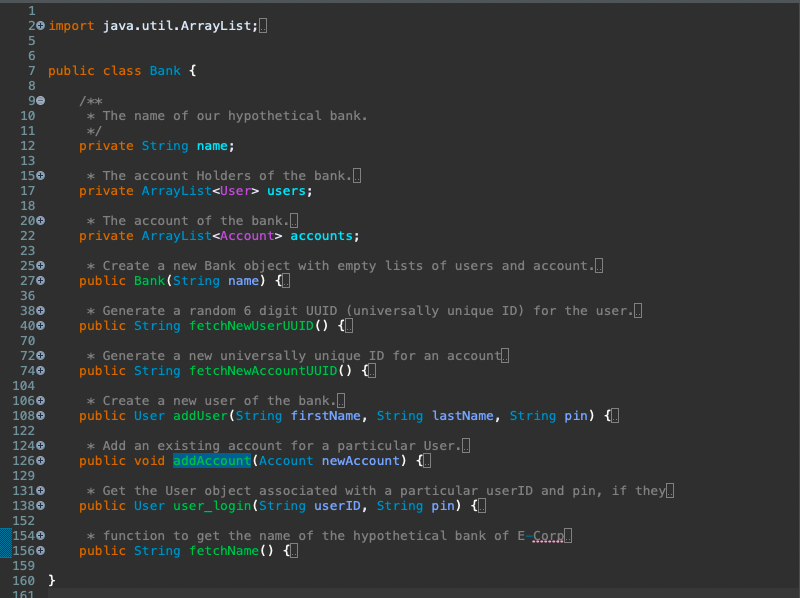
Refer as customer in the class diagram. Starting with the built in class for security is MessageDigest class; that provides applications the functionality of a message digest algorithm, in this case MSG5 hash. Message digests or are secure one-way hash functions that take arbitrary-sized data and output a fixed-length hash value. (Oracle.com, 2020).

Array list also imported to store the user and its corresponding data, user details as well as accounts. The built in class UUID used, that is a class that represents an immutable universally unique identifier (UUID). The idea is to camouflage the PIN to the bank. When decaling PIN variable we used byte data type since it only four digit number; should not occupy unnecessary memory. This class is called in the main method to access the information of the user created for the session.



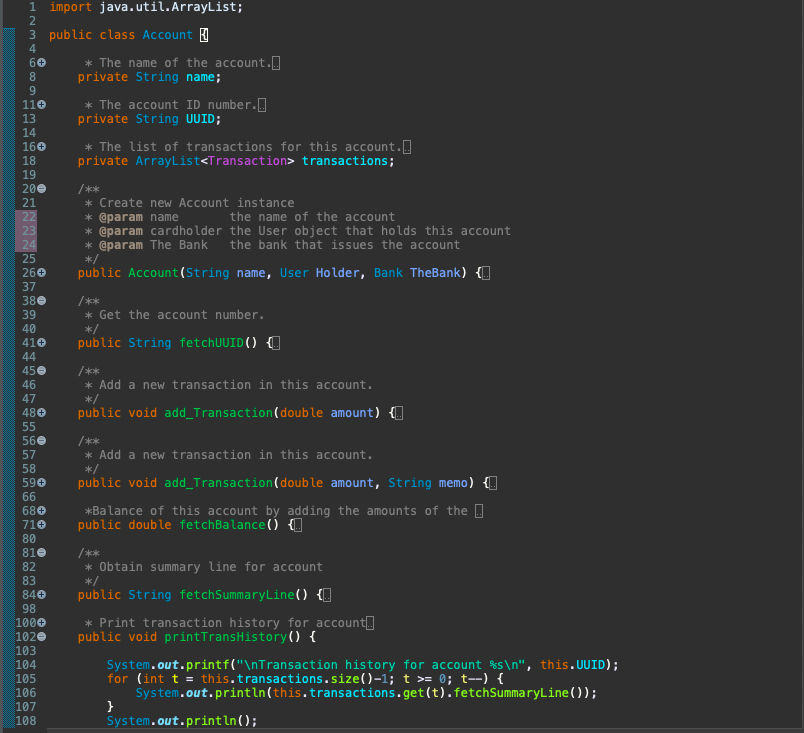
## **4.3 Bank:**

In this class the bank object has been created followed by the UUID generator class for the 10 digit account number and create a new user to the bank. In essence, the user and account array list combine here.



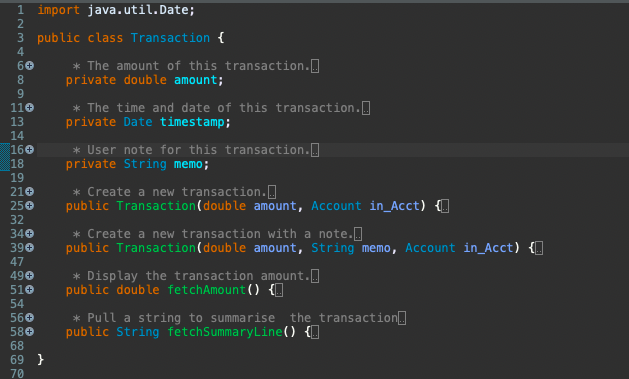
## **4.4 Account:**

This class connects user and the bank and does the function as instructed by user input.



## **4.5 Transaction:**

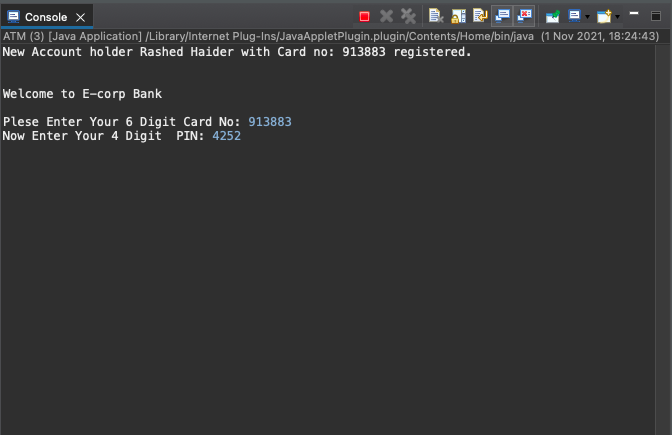
As name implies this class stores the value of the transaction that has a built in date class imported in order to get the timestamp of the transaction. This class also stores the amount and update the amount to the summary line that is called in the main method.



# **5.0 Testing:**

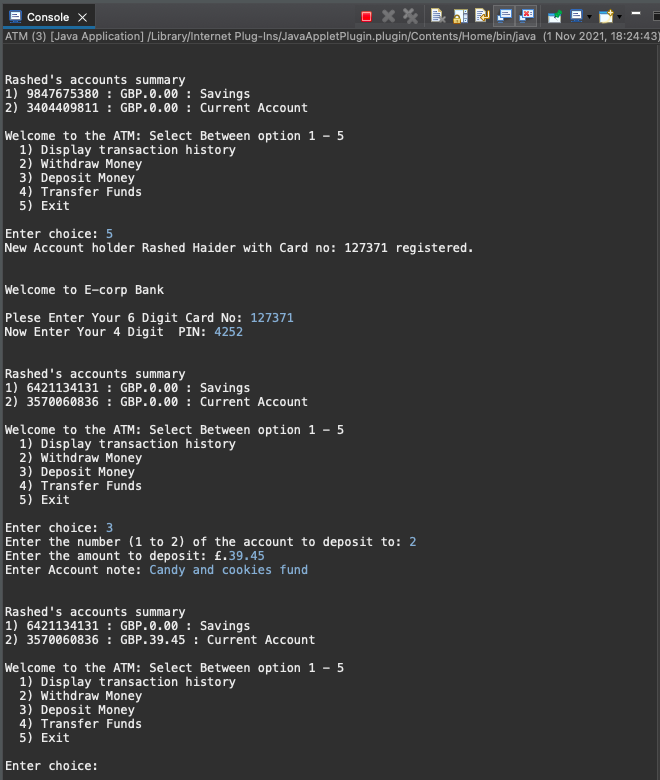
Testing starts by running the program to check all the functionalities.

## **5.1 Test with correct details:**



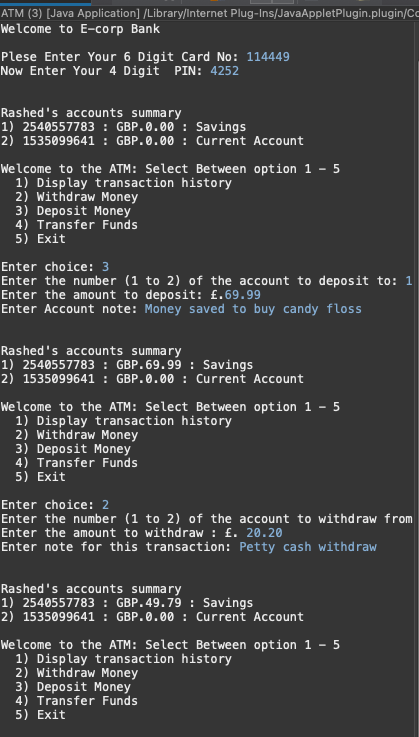
Upon correct entry we moved to the main menu option. Again for testing purpose correct input is entered to see how system behaves. First let's deposit some money and then we check the transaction history. The transaction history shown with time stamp and note for the transaction

## **5.2 Testing User menu option no 3:**



## **5.3 Testing User menu option no 2:**

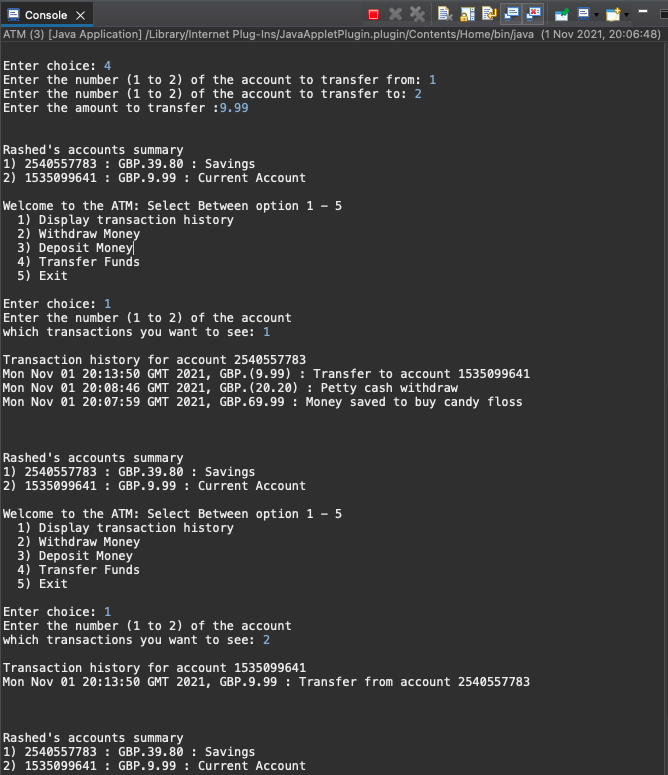
Start by depositing some money and followed by withdraw method. Note that upon completion of any transaction the system is giving us the account status followed by the main menu option.



## **5.4 Testing User menu option no 4:**

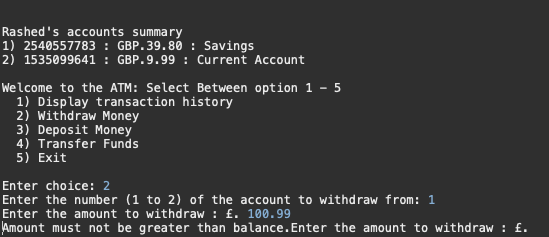
Next let's test the transfer fund functions and see if that’s shows in the transaction history option 1.

## **5.5 Testing User menu option no 1:**

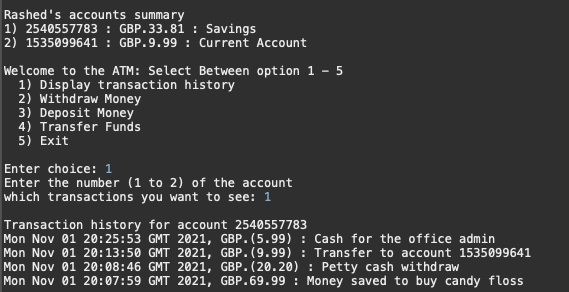


## **5.6 Testing to withdraw more than account balance:**

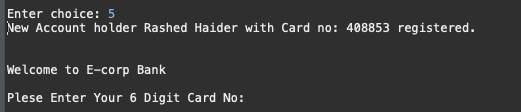
Now let's try to test the program by attempting to withdraw more than an account holds to see the result in the following screenshot:



This time let's make withdraw less than the balance and select option 1 to display transaction history with most recent in the top.



## **5.7 Testing User menu option no 5:**

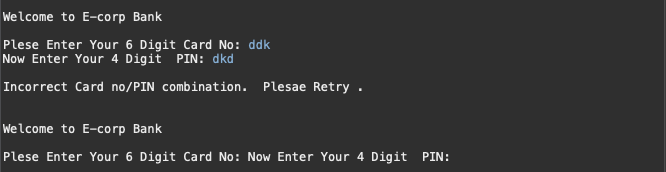
Finally we have option 5 that will log out the user from the program and the console will loop back to generate another 6 digit card no as follows:

# **6.0 Error handling and exception:**

A well-structured error handling defines how well the program is written. In this phase, all the possible scenario and how the program behaves clearly explained.

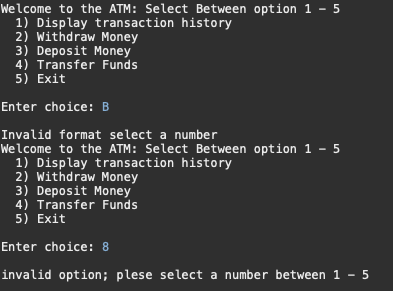
## **6.1 Incorrect input:**

If user types the wrong input the following result is observed.

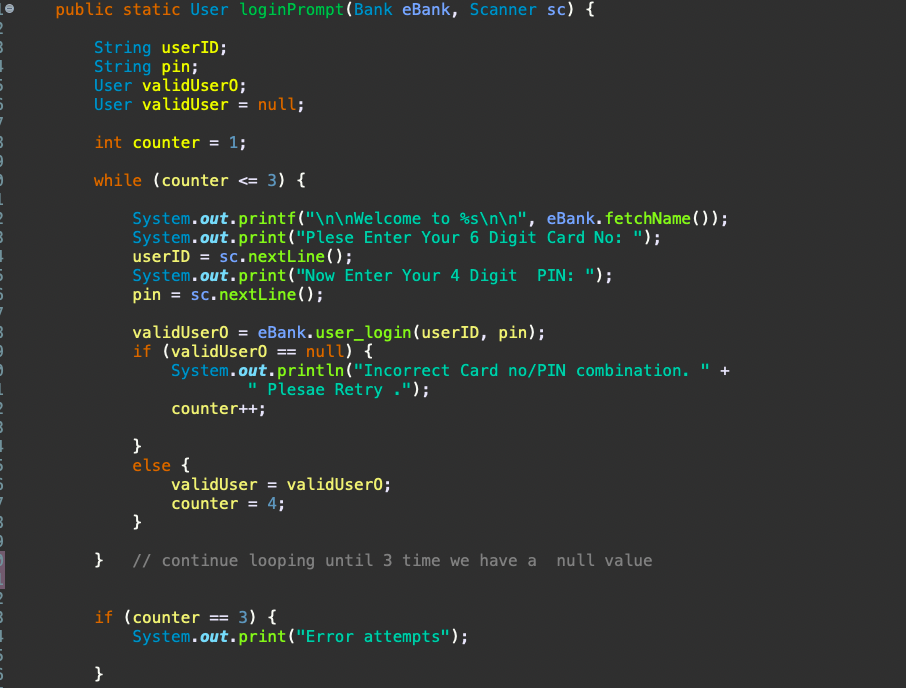


## **6.2 Invalid choice:**

In the main menu we have two exception can be distinguish between invalid format and invalid option as follows:



Below the code snippet how it works in the ATM class to first to the log in menu and then to the main menu.

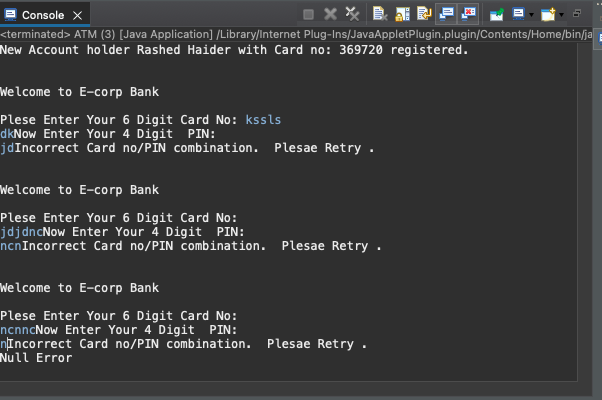






## **6.3 Unresolve exception:**

Due to the time limitation user log in attempt of 3 times has not been resolved and the program shows null error and crush.



# **7.0 Conclusion:**

The key take away from this course is the skewed learning curve of java programming that at the end engraved fundamental concept of object oriented programing from beginner to intermediary level. As much the desire to create a GUI ATM; the change of plan leads us to create the console app. Despite trying several times to install java swing window builder with all its essentials components; does not show its panels on design pane therefore, did not allow to drag and drop. The IDE as well as the language Java itself requires some strict convention and procedure that could make easily confused.

However, doing this detail project with over 35 methods used in between 5 class and many times had to deal with error and did have go back and forth to identify and rectify the problem as much as possible. This project thus enforces the learning at the end; and giving the impression that the level programing skill improved significantly.

# **Reference:**

1. Eclipse.org. (2021). *Eclipse Installer 2021-09 R | Eclipse Packages*. [online] Available at: https://www.eclipse.org/downloads/packages/installer [Accessed 1 Aug. 2021].
2. Lee, A. (2020). *Java Naming Conventions*. *YouTube*. Available at: https://www.youtube.com/watch?v=QazDDPtSulY [Accessed 18 Sep. 2021].
3. Oracle.com. (2021). *Java Platform SE 8*. [online] Available at: https://docs.oracle.com/javase/8/docs/api/ [Accessed 31 Oct. 2021].
4. Oracle.com. (2021). *Arrays (The JavaTM Tutorials > Learning the Java Language > Language Basics)*. [online] Available at: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html [Accessed 21 Sep. 2021].
5. Visual-paradigm.com. (2021). *UML Class Diagram Tutorial*. [online] Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/ [Accessed 30 Oct. 2021].
6. W3schools.com. (2021). *Java OOP (Object-Oriented Programming)*. [online] Available at: https://www.w3schools.com/java/java\_oop.asp [Accessed 1 Oct. 2021].

# **Word count: 1966**