**Classes:**

**Account:** This class takes care of the accounts that the user will be able to create. Accounts are the most important part of a software system designed for a bank because it enables the users to create a personalized space in order to carry out financial operations.

**TransferMoney:** This class serves the purpose of giving the user a way of sending and receiving money from to and from another user. It also gives the option to wire money and to pay bills directly from the account

**Deposit:** The deposit class gives the user the option to deposit cash into the account, as well as cashing checks.

**DisplayInfo:** the class has the purpose of giving the user the visual cues in order to keep them informed about the operations that have been carried out as well as keeping them updated on the balance and the amount of transactions.

**CardInformation:** cardInformation class helps the user set up a credit/debit card and gives all the information related to these cards. It also gives the option to check credit scores and history.

**Security:** The security features in this class enable the user to freeze the cards, as well as the account. It can also help the user access help

**Rewards:** The rewards class gives the user access to certain rewards that the bank offers. It enables the user to view promotions and programs, as well as applying these promotions to the balance or to cards.

**Investments:** The final class is the investment class. This class has the purpose of helping the user find a financial plan that suits their needs, enables them to get in touch with an expert, it makes it possible to invest straight from the app and to keep track of all investments made.

**Methods:**

USER:  
● deposit()

○ Displays the deposit page to the user ● transfer()

○ Displays the transfer page to the user ● invest()

• ○ Displays the investing page to the user ● card()

• ○ Displays the user’s card information to the user ● reward()

○ Displays the reward programs page to the user REWARDS:

● viewPrograms()  
○ Allows for the user to view a list of current programs offered by bank

● viewPromotions()

• ○ Allows for the user to view details of current promotions offered by bank  
● apply()

• ○ Allows for the user to apply for their selected choice of promotion  
INVESTMENTS:  
● financialPlan()

○ Creates financial suggestions for their future monetary goals ● contactExpert()

○ Allows for the user to communicate with a financial expert for advising ● viewInvestments()

○ Allows the user to view their current list of investments and suggested investments

● invest()  
○ Allows the user to invest an amount of money to their selected stock

CARD INFORMATION: ● payCard()

○ Allows the user to pay off their credit balance ● checkCredit()

○ Allows for the user to see their current credit score ● creditHistory()

○ Allows for the user to view their credit history ● requestCard()

DISPLAY INFO:  
● showBalance()

○ Allows for the user to view their current balance ● showTransactions()

○ Allows for the user to view history of previous transactions DEPOSIT:

● cashCheck()  
○ Allows for the user to deposit amount of check into balance

TRANSFER MONEY: ● directDeposit()

○ Will deposit amount from sender into balance ● wireMoney()

○ Initiates transfer of funds to recipient ● thirdParty()

○ Creates a transaction with a party other than recipient ● payBills()

• ○ Allows ACCOUNT:  
● login()

• ○ Allows  
● createAcc()

○ Allows

● logout()  
○ Allows

● closeAcc()  
○ Allows

● viewSettings() ○

SECURITY:  
● freezeCard()

○ Allows ● freezeAcc()

○ Allows

for the user to setup an automatic payment to recurring bills

for the user to log into their account for the user to create an account  
for the user to log out of their account for the user to close their account

for the user to freeze their selected card for the user to freeze their entire account

**Attributes:**

USER:  
● Name:

• ○ Type: string

• ○ Stores the user’s full name

• ● AccountID:

• ○ Type: int

• ○ Stores the user’s account number

• ● Address:

• ○ Type: string

• ○ Stores the user’s full address

• ● PhoneNum:

• ○ Type: int

• ○ Stores the user’s phone number  
REWARDS:  
● Program:

• ○ Type: string

• ○ Stores the rewards program offered INVESTMENTS:  
● InvName

• ○ Type: string

• ○ Stores the

● InvAmount:

• ○ Type: float

• ○ Stores the CARD INFORMATION:  
● cNum;

• ○ Type: int

• ○ Stores the

• Credit:

• ○ Type: int

• ○ Stores the DISPLAY INFO:  
● Balance:

• ○ Type: float

• ○ Stores the

• ● TrAmount:

• ○ Type: float

• ○ Stores the

• ● TrDescription:

investment name in order to differentiate it from others amount of an investment

number of a card user’s credit score

amount of money present in the account

amount of a transaction

• ○ Type: string

• ○ Stores the description of a transaction DEPOSIT:  
● DeAmount:

• ○ Type: float

• ○ Stores the amount of money deposited in the account TRANSFER MONEY:  
● Amount:

• ○ Type: float

• ○ Stores the amount of money transferred ACCOUNT:  
● AccountID:

• ○ Type: int

• ○ Stores the Identification Number of the account

• ● User:

• ○ Type: string

• ○ Stores the login of a user

• ● Pass:

• ○ Typer: string

• ○ Stores the password related to the account SECURITY:  
● AccID:

• ○ Type: int

• ○ Stores the account Identification in the security system

• ● CardNum:

• ○ Type: int

• ○ Stores the numbers of cards in the security system

**Unit testing:**

**TEST 1:** Testing if correct balance is displayed

Method: showBalance()

Steps:

-Create a user, Bob, with an initial balance of $1000 using new User("Bob", 1000).

-Call the showBalance(bob) method to retrieve Bob's balance.

-Assert that the returned balance (x) matches the expected balance of $1000.

Code:

//create new user

User = new User (“bob”, 1000);

y = 1000

x = user.showBalance();

//assert that the displayed balance is equal to that of actual balance

Assert (x == 1000)

Expected Result: Correct balance is displayed

**TEST 2:** Testing if the card is frozen

Method: freezeCard()

Steps:

-Create a user, Alice, with an initial balance of $1000 using new User("Alice", 1000).

-Call the user.freezeCard method to retrieve Alices' balance.

-Attempt to make a much purchase by calling user.mockPurchase and assert that it comes back as false.

Code:

//create new user

user = new User("Alice", 1000);

//call freezeCard method to deactivate card

user.freezeCard();

//call the mockPurchase method to make fake purchase and assert that it comes back as false

Assert ( user.mockPurchase() == false )

Expected Result: the mock purchase failed

**Functional testing:**

**TEST 1:** Testing transferring money between two different accounts

Methods: directDeposit(), showBalance()

Steps:

* Create two users, one named Bob with $100 in his account User(“Bob”, 100) and another user named Joe with $100 in his account User(“Bob”, 100)
* User bob uses the deposit method and selects Joe to send $20 directDeposit(“Joe”, $20)
* Both Bob and Joe use the showBalance() method to check if their balance correlates to the money given and received

Expected Result: Bob seeing $80 in his account and Joe seeing $120 in his account

**TEST 2:** Testing cashing in a check

Methods: cashCheck(), showBalance()

Steps:

* Create a user named Steve with an initial balance of $100 User(“Steve”, 100)
* Use the cashCheck() method, which takes two inputs, both a manual input of the dollar amount of the check (in this case $50), and a picture taken by the user of the check cashCheck(50, pictureInput)
* The user will use showBalance() to check if the money has been deposited into his account

Expected Result: Steve will be able to see $150 in his account

**System testing:**

**TEST 1:** Response Time

Objective: To verify that the app responds to user actions within the specified response time.

Steps:

-Perform actions like checking the account balance and fund transfers.

-Measure the time taken for the app to respond.

Code:

int iterations = 100; // Number of iterations to test

// Start the timer

long startTime = System.currentTimeMillis();

// Perform the operation to test

for (int i = 0; i < iterations; i++) {

user.showBalance();

}

// Stop the timer

long endTime = System.currentTimeMillis();

// Calculate the average response time

long responseTime = (endTime - startTime) / iterations;

//assert if response time is less than or equal to 2

Assert (responseTime <= 2)

Expected Result: The app responds within 2 seconds for the tested actions.

**TEST 2:**