System Testing Phase

This section describes every test case for every class that we have. The big titles indicate the class name being tested, and the smaller ones indicate the test method name.

ClientAccount

test_changeEmail

This test case, Test_changeEmail, is for the changeEmail method of the ClientAccount class.

Check to verify if emails can be altered properly:

The email address "tahshin89@gmail.com" should be changed.

Output: The method changes the email address associated with the client account to "tahshin89@gmail.com" and outputs True.

Verify that email cannot be converted to an empty string:

Type a blank string "" to get False as the method's output.

test_changePhone

The account class's changePhone() method is tested by this function. This techn ique's goal is to update the phone number linked to the account.

Base case :
When a legitimate phone number is input, the base case determines if the proce dure can successfully alter the phone number linked to the account.
The method is first used with the argument "6471239999", and the returned res ult is then saved in the result variable. In order to confirm that the outcome of t he method call is True, an assertion is then made using the assertTrue() method.
The following action verifies that the account's phone number has been correctl

y updated. The newPhone variable is given the account object's phone_no attri bute. The assertEqual() method is used to make a claim about whether the new Phone value corresponds to the given phone number, "6471239999".

Edge case:
The edge case determines whether the method may accept an invalid phone nu mber as input, more precisely None.
Initialization begins with calling the method with the argument None, and the r esulting result is then saved in the result variable. In order to confirm that the o utcome of the method call is False, an assertion is then made using the assertFal se() method.

test_changeAddress

Functionality tested: Changing the address of an account

When the changeAddress() method is invoked with a new address as input, this test case determines whether the account's address has been appropriately updated. Additionally, it examines if the function returns True following an address update. Additionally, it checks an edge case when the function is given an empty string as an input to determine if it returns False.

Base case:

When a valid address is input, the base case evaluates if the method can successfully change the account's associated address.

First, the method is invoked with the '"60 Shuter St"' parameter, and the 'result' variable is used to record the method's return value. The 'assertTrue()' method is then used to make an assertion to confirm that the result of the method call is 'True'.

The next action verifies that the account's linked address has been correctly updated. The 'newAddr' variable is given the 'address' attribute of the 'account' object. The 'assertEqual()' method is used to make a claim about whether the specified address **"60 Shuter St""" and the 'newAddr' value match.

Edge case:

The edge case examines how well the technique can handle an invalid address as an input, more precisely an empty string.

Prior to storing the returned result in the result variable, the procedure is first invoked with the input "". In order to confirm that the outcome of the method call is False, an assertion is then made using the assertFalse() method.

test_applyLoan

The applyLoan method of the ClientAccount class is tested using the test_applyLoan function. This method is in charge of making a new loan object and adding it to the client's account.

Base case:

"Car Loan" is the loan type and a loan amount of 1000 are specified.

The present date and a future date are used as the start and end dates, respectively.

The required arguments are passed when the applyLoan method is called.

To confirm that a new loan object has been created and added to the account, the total number of loans in the client's account are checked.

The newly formed loan object's amount, type, start date, and end date are examined to make sure they correspond to the provided values.

Edge case:

"Car Loan" is the loan type and "0.01" is the loan amount.

The present date and the following day are set as the start and end dates, respectively.

The required arguments are passed when the applyLoan method is called.

To confirm that a new loan object has been created and added to the account, the total number of loans in the client's account are checked.

The newly formed loan object's amount, type, start date, and end date are examined to make sure they correspond to the provided values.

test_openSavingsAccount

Checks that the Account class's openSavingsAccount method operates as intended. By using this method, a fresh savings account should be created and added to the list of savings accounts connected to the account.

Base case:

The list of savings accounts' starting length is kept in a variable. The length of the list of savings accounts is then checked to see if it has grown by two before calling the openSavingsAccount function twice. Both calls to openSavingsAccount are guaranteed to return True by the test.

Edge case:

The list of savings accounts' starting length is kept in a variable. The openSavingsAccount method is used five times, and each time, it is verified that the number of savings accounts has grown by five..

test_openCreditAccount

The procedure opens a new credit account with a predetermined initial balance for the account holder.

Process:

A fresh credit account is opened with a 1,000 (base case) opening balance.

The approach gives True results.

An update is made to the account's list of credit accounts.

The new credit account's balance is confirmed.

A new credit account is attempted to be opened with a 0 initial amount (edge case).

The technique gives a False result.The list of credit accounts on the account stays the same.

Base case:

the method must return True, adding the new credit account with the right balance to the account's list of credit accounts.

Edge case:

The method must return False, and the list of credit accounts for the account must not change.

test_addPayee

The addPayee() method of the account class is examined by this function. A payee (a person or a company) is added to the list of payees connected to the account using the addPayee() method.

Base case:

The payee1 variable is initially assigned a Payee object with the name "Slava" and an empty string for the account number. The assertIn() method is used to make an assertion that payee1 has been added successfully to the list of payees connected to the account when the addPayee() method is used with payee1 as a parameter.

Edge case:

The names "Fardin" and "Whatever" are used to generate a tuple payee2, which contains the payment information. The outcome of the addPayee() function call is saved in the result variable when payee2 is sent as an argument. The assertFalse() function is used to make an assertion that the method call's outcome is False.

test_acceptEtransfer

Tests the Account class's acceptEtransfer function.

Produces a sample Etransfer object and adds it to the list of incomingEtransfers for the Account.

Uses the example Etransfer object as input when using the acceptEtransfer method of the Account class.

Makes the claim that the method returns True and deletes the Etransfer object from the list of incomingEtransfers.

test_requestMoney

The requestMoney() and fulfillRequest() methods of the account class are tested by this function. The fulfillRequest() function carries out the request by sending the desired sum from the second account to the first account.

The requestMoney() method makes a request for money to another account.

First, a second ClientAccount object is created and filled out with information for the second account.

The second account is then added to the AccountInterface database using the main.AccountInterface.clientAcc.append(account2).

In the most extreme scenario, the requestMoney() function is used to send a request for $500 to account2 from the self.account, and the resulting request object is sent to request1. The assertIn() function is used to make a claim and confirm that'request1' is correct.

test_makeLoanPayment

This test case examines the operation of the ClientAccount class's makeLoanPayment method. It determines if a loan payment from the chequing account is possible, and if so, updates the amount in the account as necessary.

-Setup the test data first.

-Use the parameters listed to call the makeLoanPayment method.

- Verify whether the method call's outcome is True.

-Verify that the chequing account balance has been adjusted to $500.

Test results:

The amount in the checking account is $1,000. "Education Loan" is the loan kind, with a 500 dollar loan amount, a start date of today, and an end date of April 3, 2024.

The loan, the chequing account, and a payment of 500 are passed to the makeLoanPayment method.

Expected result:

The outcome of the call to the makeLoanPayment method must be True.

The chequing account balance must be changed to 500.

test_cancel_balance_account

The cancelBalanceAccount() method of the ClientAccount class is being tested as part of this unit test.

Base case:

The client has a credit account in this test case, and the method is called with the credit account as a parameter. The test determines whether the credit account is effectively cancelled and returns True.

Edge case:

A BalanceAccount class object is passed as an argument to the method in this test case, which is an invalid parameter. The method's return value is verified by the test.

Test result:

The method must return True in the base case and False in the edge case for the test to succeed.

AutoPayment

testChangeAmount

The changeAmount method in the AutoPayment class is tested using this test case to determine whether it is functioning properly. The function is used to modify the payment amount.

Base case: The amount is changed using a valid integer value of 500.

Edge case: The amount is changed by using the integer value 0.

Test Procedure:

A new AutoPayment object is made with a payee and payment rate that are already set.

To modify the payment amount, the changeAmount function is used with a valid integer value of 500.

Using the assertEqual method, the current payment amount is contrasted with the anticipated value of 500.

To modify the payment amount, the changeAmount method is once more invoked with the integer value 0.

To confirm that the payment amount stayed the same, the new payment amount is compared to the first payment amount using the assertNotEqual method.

Expected Outcome:

The first assertEqual function ought to succeed, signifying that the payment sum was updated to the anticipated value of 500.

The second assertNotEqual function ought to succeed, proving that the payment sum remained constant even though an improper value of 0 was entered.

testChangeRate

The changeRate method in the AutoPayment class is tested using this test case to determine whether it is functioning properly. The function is used to modify the auto-payment's payment rate.

Base case:

The payment rate is changed using a legitimate payment rate of 50 days.

Edge case:

The payment rate is changed using an incorrect payment rate of 0 days.

Test Procedure:

A fresh AutoPayment object with a predetermined payee and payment sum is generated.

To modify the payment rate, the changeRate function is used with a 50-day valid payment rate.

Using the assertEqual method, the current payment rate is contrasted with the anticipated value of 50 days.

To modify the payment rate, the changeRate function is called once again with an incorrect payment rate of 0 days.

To confirm that the payment rate stayed the same, the new payment rate is compared to the first payment rate using the assertNotEqual method.

Expected Outcome:

The first assertEqual function ought to succeed, signifying that the payment rate was modified to take the anticipated value of 50 days.

The second assertNotEqual function ought to succeed, proving that the payment rate remained constant even when a false payment rate of 0 days was applied.

Card

testLock

The lockCard and unlockCard functionalities of the Card class are verified using this test case. The card object can be locked or unlocked using these operations, accordingly.

Test Technique:

The setUp function is used to generate a fresh Card object.

To lock the card object, the lockCard method is used.

To ensure that the locked property of the card object is set to True, use the assertTrue function.

To unlock the card object, the unlockCard method is used.

The locked attribute of the card object is checked to see if it is set to False using the assertFalse function.

Expected Outcome:

The lockCard method should successfully set the locked property of the card object to True, as indicated by the first assertTrue function passing.

The unlockCard method should successfully set the locked property of the card object to False, as seen by the second assertFalse function passing.

Loan

testPay

The pay function in the Loan class is tested using this test case to determine whether it is functioning properly. The feature is utilised to make loan repayment instalments.

Test Results:

The Loan object is created with a loan amount of $10,000, a car loan type, and dates for the beginning and conclusion of the loan.

Base-case 1: A legitimate 2000 loan payment is made.

Base-case 2 : The loan receives a proper 8,000 payment.

Edge case: A loan payment of $11,000 is made in error.

Test Procedure:

Using the setUp function and the supplied data, a new Loan object is created.

With a payment value of 11,000, which is more than the loan balance, the pay function is invoked.

To determine if the return result of the pay function matches the anticipated value of -1, which denotes that the payment amount is illegal, the assertEqual function is utilised.

A valid payment value of 2000 is sent to the pay function, which is invoked.

The assertEqual method is used to confirm that the final payment amount, 8000, is correct.

With 8000 as the payment value—another legal payment amount—the pay function is once more invoked.

The assertEqual function is used to verify that the remaining payment amount after making the payment is 0.

Expected Outcome:

Indicating that the pay function returns -1 for an incorrect payment amount, the initial assertEqual function ought to succeed.

The second assertEqual function should succeed, proving that 8000 is the balance due following a successful payment of 2000.

The third assertEqual function should succeed, proving that there is nothing left to pay after a genuine payment of 8000 has been made.

TellerAccount

testRegisterClient

The registerClient method in the TellerAccount class is tested using this test case to determine whether it is functioning properly. The feature is used to add new customers to the bank's database.

Test Results:

The teller username "iamateller", teller name "Nikita", and teller password "teller123" are initialised in the TellerAccount object.

Base case:

The bank registers a new customer named "john98" with a special username.

Edge case:

A new client registration attempt is performed using the same login as the outgoing client.

Test Technique:

The setUp function is used to generate a new TellerAccount object with the supplied information.

The information for a new client with the distinctive username "john98" is passed to the registerClient method.

To confirm that the new client is present in the clientAcc list of registered clients, the assertIn method is utilised.

With the same username "john98" as the last client, the registerClient method is called once more.

To confirm that the return value of the registerClient method is None, indicating that the registration was unsuccessful, use the assertIsNone function.

Expected Outcome:

The initial assertIn function ought to succeed, signifying that the registration of the new client was successful.

The second assertIsNone function should succeed, indicating that a username conflict prevented the registration from being successful.

testFindClient

The findClient method in the TellerAccount class is tested using this test case to determine whether it is functioning properly. A registered client may be located using the function and their account number.

Test Results

The teller username "iamateller", teller name "Nikita", and teller password "teller123" are initialised in the TellerAccount object.

Base case:

A new customer registers with the bank using the login "john98," and the bank uses the customer's account number to identify them.

Edge case:

An effort is made to track down a customer who has an incorrect account number.

Test Technique:

The setUp function is used to generate a new TellerAccount object with the supplied information.

The registerClient method is used to register a new customer with the bank.

The account number of the registered client is used to invoke the findClient method.

To ensure that the delivered client object and the registered client object are same, use the assertEqual method.

With an incorrect account number, the findClient method is performed once more.

To confirm that the returned result is None, indicating that the client was not located, the assertIsNone method is used.

Expected Outcome:

The first assertEqual function should succeed, proving that the findClient method uses the account number of the client to return the right client object.

The second assertIsNone function should succeed, proving that a bad account number causes the findClient method to return None.

UserAccount

testUpdate

The testUpdate() method checks whether the update() method adds a new notification to the notifications list of the UserAccount object.

testLogin

The testLogin() method checks the login() method's functionality by testing four different scenarios: wrong login and password, wrong login, wrong password, and successful login. The test asserts that the login() method returns False for the first three scenarios and True for the successful login scenario.

testChangePassword

The testChangePassword() method tests whether the changePassword() method changes the password of the UserAccount object. It checks for three scenarios: wrong password, wrong username, and successful password change. The test asserts that the password has not been changed for the first two scenarios and that the password has been successfully changed for the third scenario.

BalanceAccount

setUp

Base case: Creates a new instance of a client account and teller account

Edge case: The account setup will fail if any of the cases are invalid or incorrect, missing an attribute or instances are not set properly

testAddObserver

Base case: Assertion checks if the observer is added to the chequing account

Edge case: Make sure if the string has correctly input

testRemoveObserver

Base case: Removes a client account from the list of chequing account

Edge case: Account only can be removed if it is on the list of clients

testNotifyObservers

Base case: Runs the notification functionality for the clients

Edge case: Clients are required to be registered to the chequing account beforehand

testSetNotifAmount

Base case: Sets the amount for notifying the clients for the chequing account

Edge case: Not going to work if the account balance goes below negative 10

testDeposit

Base case: It runs the functionality of depositing to the chequing account

Edge case: Check whether the balance of the chequing account is still old balance

testWithdraw

Base case: It stores the balance of the chequing account in the variable old balance

Edge case: Withdrawing a higher value then the balance will fail the withdraw transaction

testNewTransaction

Base case: It updates the chequing account depending on the amount of wire transfer

Edge case:

testTransferBetweenAccount

Base case: It transfers between a chequing and a saving account of a single client

Edge case: If the balance is lower than the transfer amount then it will not complete the transaction

testSetupAutoPayment

Base case: It requires choosing a payee and the payment will be generated

Edge case: The payee or account name requires to be valid and the amount has to be more than the balance .

testSendWireTransfer

Base case: Transferring the amount from one chequing account to another client's chequing account

Edge case: Transfer will be unsuccessful if the amount is higher than the balance

testSendEtransfer

Base case: Processes an electronic transfer from one registered chequing account

Edge case: The transfer amount needs to be greater than the available balance and payee information has to be completed correctly.

testMakePurchase

Base case: Can process a purchase according to the spending from the chequing account and will be added to the transaction list

Edge case: Amount of purchase has to be sufficient to cover the account balance and the inputs of the vendor's name and amount has to be accurate

testWithdrawalATM

Base case: Clients can use their accounts to withdraw portion of their balance from the chequing account

Edge case: If the account balance is zero and the amount are invalid then the withdraw will fail