Software Test Plan for BankApp

Team members: Alma, Joel, Bryce

GitHub: https://github.com/boylebryce/BankApp

Introduction

The BankApp program is a simulation of a financial institution that allows users to interact with various elements of a bank, such as ATMs and branches. Users can create new bank accounts, make withdraws and deposits, view transactions, add authorized users, and more. The bank system is simulated by a suite of classes that handle issues such as verifying transactions, loading ATMs with cash and receipt paper, and flagging fraudulent account activity.

Accuracy and reliability are essential to any monetary system. We have developed a test plan to ensure that the BankApp program accurately models a real banking system. The test plan ensures that each component of the BankApp program functions as intended without crashing or producing errors. The test plan covers common use cases of a bank simulation as well as unit tests for the software components of the program. Each test case and unit test is designed to be implemented using the JUnit framework to allow for automated testing.

Test Cases

The following test cases document anticipated use cases for the BankApp program and the expected results. Test cases allow us to model high level testing of user interactions with the BankApp program, ensuring that the public interface reliably produces accurate results. The user can choose to interact with the ATM interface or with the Branch interface when running the program, so our test cases are based on which interface is being used.

Test Case Template

#.

Scenario:
Preconditions:

-

Steps:

1.

Expected results:

ATM Test Cases

1.

Scenario: Customer accesses an ATM
Preconditions:

- Customer has credentials to be authenticated by ATM
- Customer account is not locked

Steps:

1. Customer selects to go to ATM
2. Customer submits account credentials to ATM

Expected results: ATM displays successful login message, and then displays what actions the customer can take.

2.

Scenario: Customer makes a valid deposit at the ATM
Preconditions:

- Customer successfully logged in to ATM
- ATM displaying available actions

Steps:

1. Customer selects deposit option
2. Customer enters deposit amount and deposit information, and submits deposit

Expected results: ATM displays successful deposit message, new balance, and returns to displaying available actions.

3.

Scenario: Customer makes a fraudulent deposit at the ATM
Preconditions:

- Customer successfully logged in to ATM
- ATM displaying available actions

Steps:

1. Customer selects deposit option
2. Customer enters deposit amount and deposit information, and submits deposit

Expected results: ATM displays error message and ends customer session.

4.

Scenario: Customer makes a valid withdrawal from an ATM
Preconditions:

- Customer successfully logged in to ATM
- ATM displaying available actions
- Customer has enough balance to process withdrawal
- ATM has enough cash to process withdrawal

Steps:

1. Customer selects withdrawal option
2. Customer enters withdraw amount and submits withdraw

Expected results: ATM displays successful withdrawal message, new balance, and returns to displaying available actions.

5.

Scenario: Customer makes an invalid withdrawal from an ATM – Insufficient balance
Preconditions:

- Customer successfully logged in to ATM
- Customer does NOT have enough balance to process withdrawal
- ATM displaying available actions
- ATM has enough cash to process withdrawal

Steps:

1. Customer selects withdrawal option
2. Customer enters withdraw amount and submits withdraw

Expected results: ATM displays withdraw error: insufficient balance message and returns to displaying available actions.

6.

Scenario: Customer makes an invalid withdrawal from an ATM – ATM low cash
Preconditions:

- Customer successfully logged in to ATM
- Customer has enough balance to process withdrawal
- ATM displaying available actions
- ATM does NOT have enough cash to process withdrawal

Steps:

1. Customer selects withdrawal option
2. Customer enters withdraw amount and submits withdraw

Expected results: ATM displays withdraw error: ATM low cash message and returns to displaying available actions.

7.

Scenario: Customer makes a valid transfer between two accounts at an ATM
Preconditions:

- Customer successfully logged in to ATM
- Customer has at least two accounts
- Customer has enough balance in the account being transferred from
- ATM displaying available actions

Steps:

1. Customer selects transfer option
2. Customer enters transfer-from account, transfer-to account, and amount
3. Customer submits transfer

Expected results: ATM displays updated balance for both accounts and returns to displaying available actions.

8.

Scenario: Customer makes an invalid transfer between two accounts at an ATM
Preconditions:

- Customer has successfully logged in to ATM
- Customer has at least two accounts
- Customer does NOT have enough balance in transfer-from account
- ATM displaying available actions

Steps:

1. Customer selects transfer option
2. Customer enters transfer-from account, transfer-to account, and amount
3. Customer submits transfer

Expected results: ATM displays invalid transfer – insufficient balance message and returns to displaying available actions.

9.

Scenario: Customer ends ATM session
Preconditions:

- Customer has successfully logged in to ATM
- ATM displaying available actions

Steps:

1. Customer selects end session option and chooses a receipt option (print / Email)

Expected results: ATM generates receipt and provides it to the customer through the specified method. ATM ends session with customer.

Branch Test Cases

10.

Scenario: Customer accesses a branch
Preconditions:

- Customer has credentials to be authenticated by branch
- Customer account is not locked

Steps:

1. Customer selects to go to a branch
2. Customer provides credentials to branch entity

Expected results: Branch entity displays successful login message, and then displays what actions the customer can take.

11.

Scenario: Customer makes a valid deposit at the branch
Preconditions:

- Customer successfully authenticated by branch
- Branch returned available actions

Steps:

1. Customer selects deposit option
2. Customer enters deposit amount and deposit information, and submits deposit

Expected results: Branch returns successful deposit message, new balance, and list of available actions.

12.

Scenario: Customer makes a fraudulent deposit at the branch
Preconditions:

- Customer successfully authenticated by branch
- Branch returned available actions

Steps:

1. Customer selects deposit option
2. Customer enters deposit amount and deposit information, and submits deposit

Expected results: Branch returns error message and ends customer session.

13.

Scenario: Customer makes a valid withdrawal at a branch
Preconditions:

- Customer authenticated by branch
- Branch returned available actions
- Customer has enough balance to process withdrawal
- Branch has enough cash to process withdrawal

Steps:

1. Customer selects withdrawal option
2. Customer enters withdraw amount and submits withdraw

Expected results: Branch returns successful withdrawal message, new balance, and list of available actions.

14.

Scenario: Customer makes an invalid withdrawal from a branch – Insufficient balance
Preconditions:

- Customer authenticated by branch
- Customer does NOT have enough balance to process withdrawal
- Branch returned available actions
- Branch has enough cash to process withdrawal

Steps:

1. Customer selects withdrawal option
2. Customer enters withdraw amount and submits withdraw

Expected results: Branch returns withdrawal error: insufficient balance and list of available actions.

15.

Scenario: Customer makes an invalid withdrawal from a branch – Branch low cash
Preconditions:

- Customer authenticated by branch
- Customer has enough balance to process withdrawal
- Branch returned available actions
- Branch does NOT have enough cash to process withdrawal

Steps:

1. Customer selects withdrawal option
2. Customer enters withdraw amount and submits withdraw

Expected results: Branch returns withdrawal error: branch low cash and list of available actions.

16.

Scenario: Customer makes a valid transfer between two accounts at a branch
Preconditions:

- Customer authenticated by branch
- Customer has at least two accounts
- Customer has enough balance in the account being transferred from
- Branch returned available actions

Steps:

1. Customer selects transfer option
2. Customer enters transfer-from account, transfer-to account, and amount
3. Customer submits transfer

Expected results: Branch returns updated balances for both accounts and list of available actions.

17.

Scenario: Customer makes an invalid transfer between two accounts at a branch
Preconditions:

- Customer authenticated by branch
- Customer has at least two accounts
- Customer does NOT have enough balance in transfer-from account
- Branch returned list of available actions

Steps:

1. Customer selects transfer option
2. Customer enters transfer-from account, transfer-to account, and amount
3. Customer submits transfer

Expected results: Branch returns invalid transfer – insufficient balance error and list of available actions.

18.

Scenario: Customer leaves branch (ends session)
Preconditions:

- Customer authenticated by branch
- Branch returned available actions

Steps:

1. Customer selects end session option and chooses a receipt option (print / Email)

Expected results: Branch generates a receipt of all transactions made during the session and returns the receipt to the customer via the selected method.

19.

Scenario: Customer opens a new account (checking / savings / credit)
Preconditions:

- Customer authenticated by branch
- Customer has less than three accounts
- Branch returned available actions

Steps:

1. Customer selects create new account action
2. Customer selects which type of account to create (checking / savings/ credit)
3. Customer enters a desired PIN number for the new account

Expected results: Branch returns new account information: account number, card number, and chip number. Branch returns list of available actions.

20.

Scenario: Customer adds authorized user to an existing account
Preconditions:

- Customer authenticated by branch
- Account does not already have an authorized user
- Branch returned available actions

Steps:

1. Customer selects add authorized user action
2. Customer selects the account to add an authorized user to
3. Customer submits name of authorized user

Expected results: Branch returns success message and list of available actions.

21.

Scenario: Customer adds an authorized user to an existing account that already has an authorized user
Preconditions:

- Customer authenticated by branch
- Account already has an authorized user
- Branch returned available actions

Steps:

1. Customer selects add authorized user action
2. Customer selects the account to add an authorized user to

Expected results: Branch returns error message: account already has an authorized user. Branch returns list of available actions.

22.

Scenario: Customer removes authorized user from an existing account
Preconditions:

- Customer authenticated by branch
- Account has an authorized user
- Branch returned available actions

Steps:

1. Customer selects remove authorized user action
2. Customer selects the account to remove the authorized user from

Expected results: Branch returns success message and list of available actions.

23.

Scenario: Customer removes authorized user from an existing account that does not have an authorized user

Preconditions:

- Customer authenticated by branch
- Account does not have an authorized user
- Branch returned list of available actions

Steps:

1. Customer selects remove authorized user action
2. Customer selects the account to remove the authorized user from

Expected results: Branch returns error: account does not have an authorized user message. Branch returns list of available actions.

24.

Scenario: Customer changes PIN number

Preconditions:

- Customer authenticated by branch
- Customer has at least one account

Steps:

1. Customer selects change PIN action
2. Customer selects which account to update the PIN for
3. Customer enters the new PIN

Expected results: Branch returns success message and list of available actions.

25.

Scenario: Customer unlocks a locked account

Preconditions:

- Customer authenticated by branch
- Customer has a locked account

Steps:

1. Customer selects unlock a locked account action
2. Customer provides additional authentication information

Expected results: Branch returns success message and list of available actions.

26.

Scenario: Customer updates personal info
Preconditions:

- Customer authenticated by branch

Steps:

1. Customer selects update personal info action
2. Customer enters new personal info

Expected results: Branch returns success message and list of available actions.

Unit Tests

This section describes the requirements for unit tests that will be used to test the functionality of each discrete component of the BankApp project programmatically. The project will use JUnit as its testing framework to facilitate automated testing.

1. Each class in the source code for the BankApp program will have a test driver composed of unit tests.
    a. A test driver for a class will have the same name as the class, followed by “Test”. For example, a class “Customer” will have the test driver class “CustomerTest”.
2. A test driver will test all public methods exposed by the class that it is testing.
    a. Test drivers will use the suite of JUnit assertion methods to ensure that methods successfully modify the state of the program as intended.
    b. For each assertion that compares an expected value to the result of a method, the expected value should be the LEFT argument, and the result of the method should be the RIGHT argument.
3. Each method should be tested in multiple ways to ensure completeness of testing.
    a. Tests should focus on boundary cases and extreme values that may cause unexpected behavior, such as positive and negative values, the max value represented by a data type, or the boundaries of an array.
4. Each test for a method should be handled in its own function within the test driver.
    a. Each test function should have a descriptive name that differentiates it from other tests being done on the same method.
        i. Example: addPositiveValueToATMCash(), addNegativeValueToATMCash()
5. All test functions should have a timeout value of 1000 to ensure no methods hang for an unusual amount of time.
6. All methods that throw exceptions should be tested for those exceptions using JUnit’s built in exception testing functionality.
7. All test drivers should utilize the @Before and @After structures to build and tear down objects that are used in tests to ensure that each test function is running on a fresh instance of the objects.
    a. This ensures that tests are self-contained and cannot rely on the sequence of operations performed in other tests. Each test function should therefor only be concerned with its own assertions.