CSE341 - Software Testing, Validation, and Verification

Banking System Testing

23P0251 Lara Sherif Fathy

23P0230 Farida Waleed El Sayed

23P0379 Andrea Aziz Fathy

23P0124 Fady Joseph Guirgis

23P0234 Lina Tamer Yousri

23P0325 Madonna Emad Labib

Project Overview

This project implements a simplified banking system designed to demonstrate and evaluate different software testing techniques, including unit testing, state-based testing, integration testing, and GUI-based testing.

The system models core banking operations such as deposits, withdrawals, account status management, and transaction processing. The architecture follows a layered design, separating business logic from control logic and user interface concerns to improve testability and maintainability.

Project structure

Account: Represents a bank account entity, Maintains balance and account status
TransactionProcessor: Contains transaction decision logic
AccountService: Acts as a business layer, sends transaction handling to the processor
ClientController: Handles client-side transaction requests
CreditService: for the extra feature

- Demonstrates exception handling and boundary testing
- Tested separately via black-box and edge-case tests

User Interface Implementations

To address GUI-based testing requirements and mitigate GUI testing challenges discussed in lectures, two UI approaches were implemented:
1. HTML-Based Dashboard (Static GUI)
A simple HTML interface was used to:

- Analyze UI structure and component presence
- Validate input constraints and error messages

This approach supports black-box GUI testing, focusing on expected behavior rather than internal implementation. It assumes java is used as backend for html
2. JavaFX-Based GUI (Dynamic GUI)
A minimal JavaFX application was implemented to:

- Exercise real GUI components (buttons, labels, text fields)
- Trigger GUI events (e.g., button clicks)
- Demonstrate state-dependent behavior (e.g., disabling buttons when an account is suspended or closed)
- Connect GUI actions to the controller and service layers

This JavaFX interface enables automated GUI-based functional testing using JUnit-based tests, without requiring a fully complex UI framework.
Together, these approaches provide sufficient coverage for GUI-based testing objectives while keeping the implementation minimal and focused on testing concepts rather than UI complexity.

GitHub link: https://github.com/lara710/Testing

Section A: Black-Box Testing

Test Case Table

Test Case ID	Input	Precondition	Expected Output	Notes
BB01	deposit(-100)	Account = Verified, Balance = 100	false	Invalid amount
BB02	withdraw(50)	Account = Verified, Balance ≥ 50	true	Valid withdrawal
BB03	withdraw(500)	Account = Verified, Balance = 100	false	Overdraft prevention
BB04	deposit(100)	Account = Closed	false	Closed account restriction
BB05	withdraw(50)	Account = Suspended	false	Suspended account restriction
BVA-D-01	deposit(0)	Account = Verified	false	Lower boundary
BVA-D-02	deposit(0.01)	Account = Verified	true	Just above boundary
BVA-W-01	withdraw(balance)	Account = Verified	true	Exact balance
BVA-W-02	withdraw(balance +1)	Account = Verified	false	Just above boundary

Equivalence Partitioning

Partition	Input Range	Expected Behavior
Invalid	amount ≤ 0	Deposit rejected
Valid	amount > 0	Deposit accepted
Invalid (state)	Closed account	Deposit rejected

Partition	Input Range	Expected Behavior
Valid	amount ≤ balance	Withdrawal accepted
Invalid	amount > balance	Withdrawal rejected
Invalid (state)	Suspended / Closed	Withdrawal rejected

Boundary Value Analysis:

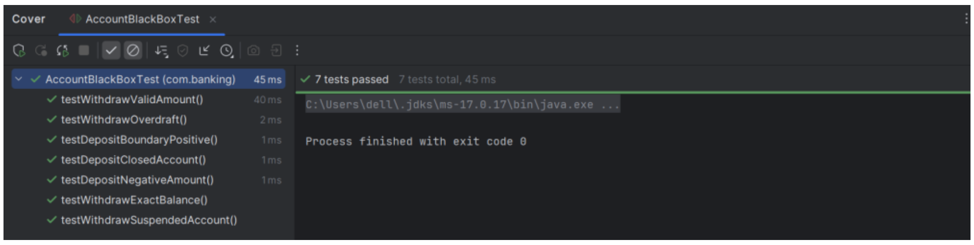
Test	Amount	Expected Result
Lower boundary	0	Reject
Below boundary	-100	Reject
Just above boundary	0.01	Accept

Test	Amount	Expected Result
Exact balance	balance	Accept
Just above balance	balance + 1	Reject

Validate behavior against requirements:

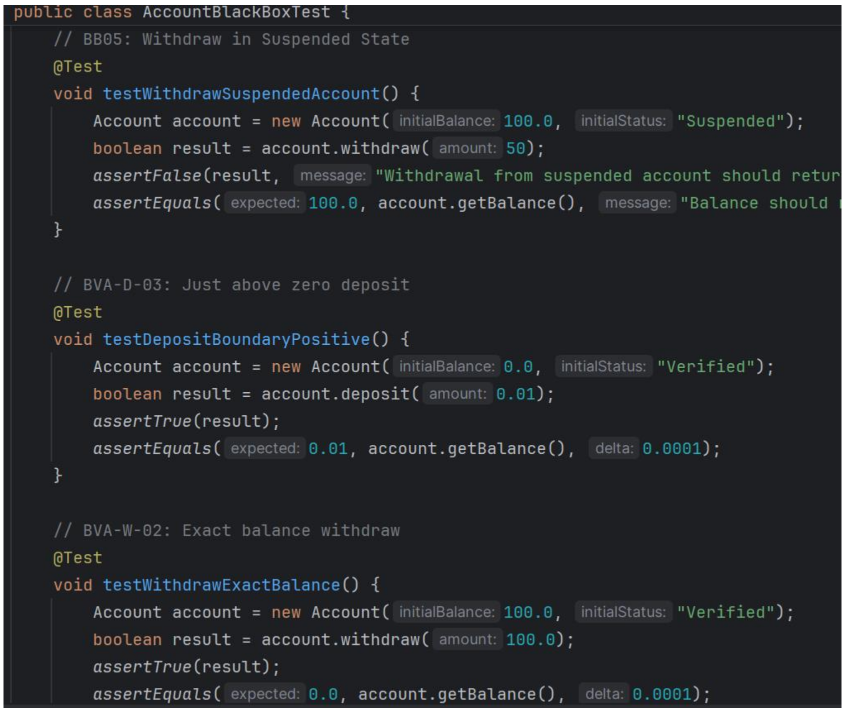
- Invalid deposits are rejected
- Overdrafts are prevented
- State restrictions are enforced
- Valid operations correctly update balance

Report:



```
✓ AccountBlackBoxTest (com.banking)   45 ms
    ✓ testWithdrawValidAmount()       40 ms
    ✓ testWithdrawOverdraft()          2 ms
    ✓ testDepositBoundaryPositive()    1 ms
    ✓ testDepositClosedAccount()       1 ms
    ✓ testDepositNegativeAmount()      1 ms
    ✓ testWithdrawExactBalance()
    ✓ testWithdrawSuspendedAccount()

✓ 7 tests passed  7 tests total, 45 ms
C:\Users\dell\.jdks\ms-17.0.17\bin\java.exe ...

Process finished with exit code 0
```

Implementation example:



```java
public class AccountBlackBoxTest {
    // BB05: Withdraw in Suspended State
    @Test
    void testWithdrawSuspendedAccount() {
        Account account = new Account( initialBalance: 100.0,  initialStatus: "Suspended");
        boolean result = account.withdraw( amount: 50);
        assertFalse(result,  message: "Withdrawal from suspended account should retur
        assertEquals( expected: 100.0, account.getBalance(),  message: "Balance should
    }

    // BVA-D-03: Just above zero deposit
    @Test
    void testDepositBoundaryPositive() {
        Account account = new Account( initialBalance: 0.0,  initialStatus: "Verified");
        boolean result = account.deposit( amount: 0.01);
        assertTrue(result);
        assertEquals( expected: 0.01, account.getBalance(),  delta: 0.0001);
    }

    // BVA-W-02: Exact balance withdraw
    @Test
    void testWithdrawExactBalance() {
        Account account = new Account( initialBalance: 100.0,  initialStatus: "Verified");
        boolean result = account.withdraw( amount: 100.0);
        assertTrue(result);
        assertEquals( expected: 0.0, account.getBalance(),  delta: 0.0001);
```

Section B: White-Box Testing

Analysis

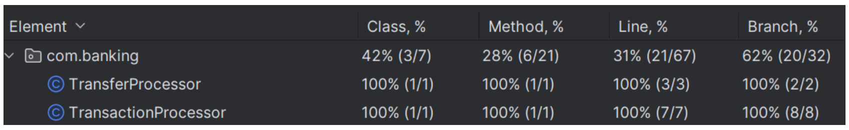
White-box testing was applied to the TransactionProcessor, deposit(), and withdraw() methods to ensure full branch coverage. Test cases were designed based on the internal control flow and conditional logic, covering null checks, transaction type routing, account status validation (Verified, Suspended, Closed), negative amounts, overdraft conditions, and successful execution paths. All decision outcomes were exercised using JUnit tests, achieving complete branch coverage of the implemented logic. TransferProcessor reuses TransactionProcessor logic, preserving separation of concerns.

Test Case Table

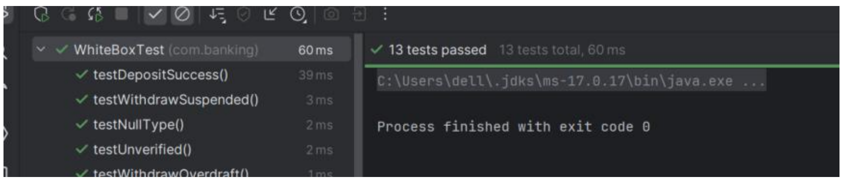
Test Case ID	Input Conditions	Expected Output	Branch
WB-T-01	account = null	false	Null account check
WB-T-02	type = null	false	Null transaction type
WB-T-03	type = "TRANSFER"	false	Unknown transaction type branch
WB-T-04	status = Unverified "	false	Unverified account
WB-D-01	status = Closed, amount = 50	false	Closed account deposit rejection
WB-D-02	status = Verified, amount = -10	false	Negative deposit amount
WB-D-03	status = Verified, amount = 50	true	Valid deposit success path
WB-W-01	status = Closed, amount = 10	false	Closed account withdrawal
WB-W-02	status = Suspended, amount = 10	false	Suspended account withdrawal
WB-W-03	status = Verified, amount > balance	false	Overdraft prevention

Test ID	Method	Branch	Expected Result
WB-TR-01	transfer()	Withdraw = T → Deposit = T	Transfer succeeds
WB-TR-02	transfer()	Withdraw = F	Transfer fails

Report



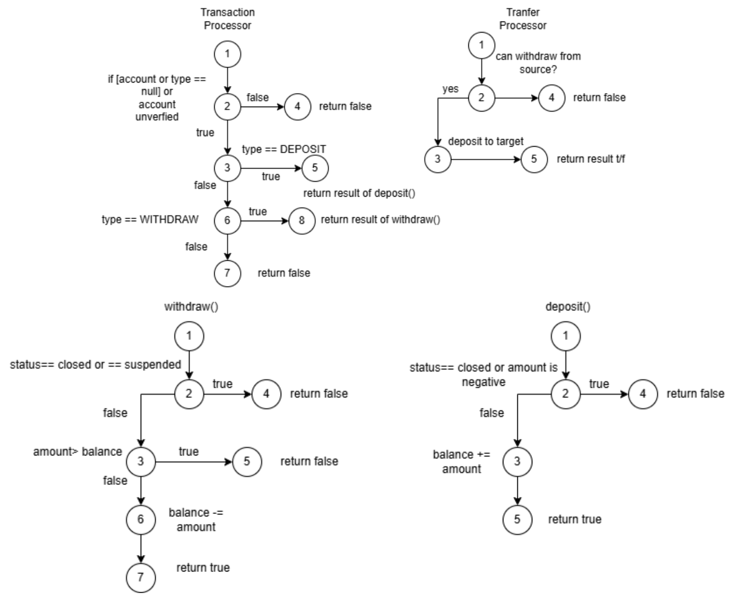
Element	Class, %	Method, %	Line, %	Branch, %
com.banking	42% (3/7)	28% (6/21)	31% (21/67)	62% (20/32)
TransferProcessor	100% (1/1)	100% (1/1)	100% (3/3)	100% (2/2)
TransactionProcessor	100% (1/1)	100% (1/1)	100% (7/7)	100% (8/8)



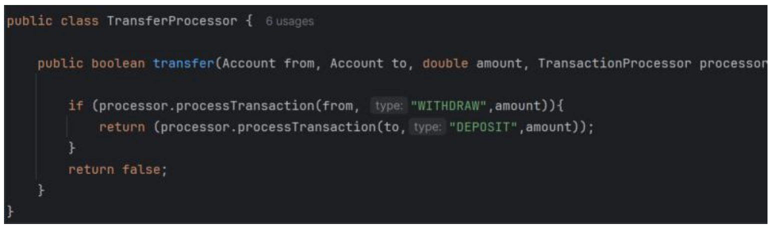
```
✓ WhiteBoxTest (com.banking)      60 ms
    ✓ testDepositSuccess()        39 ms
    ✓ testWithdrawSuspended()      3 ms
    ✓ testNullType()               2 ms
    ✓ testUnverified()             2 ms
    ✓ testWithdrawOverdraft()      1 ms
```

```
✓ 13 tests passed  13 tests total, 60 ms
C:\Users\dell\.jdks\ms-17.0.17\bin\java.exe ...

Process finished with exit code 0
```

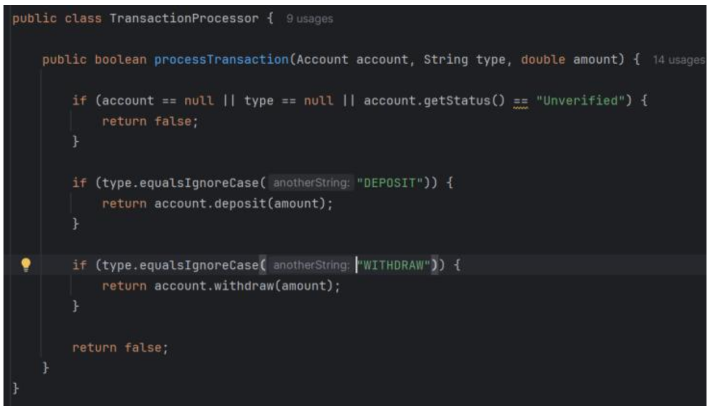
CFG:



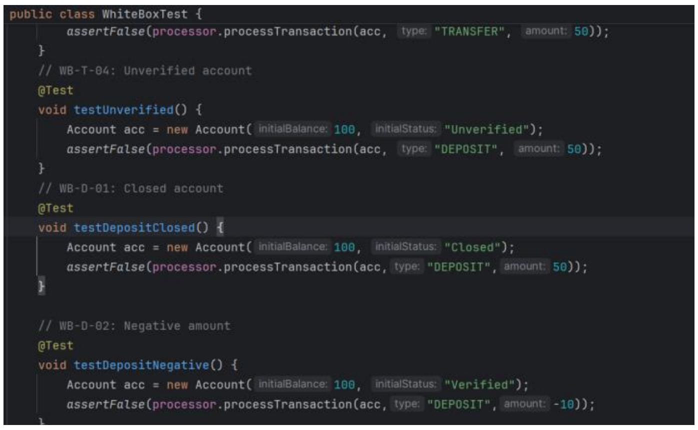
Implementation example:



```
public class TransferProcessor {  6 usages

    public boolean transfer(Account from, Account to, double amount, TransactionProcessor processor

        if (processor.processTransaction(from,  type: "WITHDRAW",amount)){
            return (processor.processTransaction(to,  type: "DEPOSIT",amount));
        }
        return false;
    }
}
```



```
public class TransactionProcessor {  9 usages

    public boolean processTransaction(Account account, String type, double amount) {  14 usages

        if (account == null || type == null || account.getStatus() == "Unverified") {
            return false;
        }

        if (type.equalsIgnoreCase( anotherString: "DEPOSIT")) {
            return account.deposit(amount);
        }

        if (type.equalsIgnoreCase( anotherString: "WITHDRAW")) {
            return account.withdraw(amount);
        }

        return false;
    }
}
```



```
public class WhiteBoxTest {
        assertFalse(processor.processTransaction(acc,  type: "TRANSFER",  amount: 50));
    }
    // WB-T-04: Unverified account
    @Test
    void testUnverified() {
        Account acc = new Account( initialBalance: 100,  initialStatus: "Unverified");
        assertFalse(processor.processTransaction(acc,  type: "DEPOSIT",  amount: 50));
    }
    // WB-D-01: Closed account
    @Test
    void testDepositClosed() {
        Account acc = new Account( initialBalance: 100,  initialStatus: "Closed");
        assertFalse(processor.processTransaction(acc,  type: "DEPOSIT",  amount: 50));
    }

    // WB-D-02: Negative amount
    @Test
    void testDepositNegative() {
        Account acc = new Account( initialBalance: 100,  initialStatus: "Verified");
        assertFalse(processor.processTransaction(acc,  type: "DEPOSIT",  amount: -10));
```

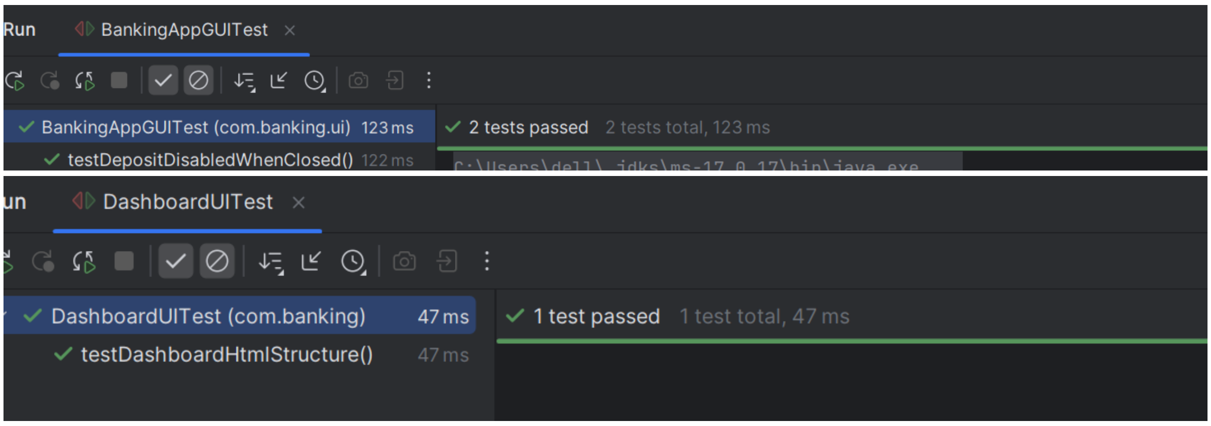
Section C: UI Testing

Approaches:
This section validates the correctness of the user interface for the banking system. Both the JavaFX desktop application and the HTML/JavaScript web dashboard were tested to ensure proper input validation, state-dependent behavior, and user feedback. Automated tests were used to verify button enable/disable logic and client-side validation rules, while static analysis was applied to confirm the presence of required UI logic. These tests ensure that invalid actions are prevented and that the UI consistently reflects the account state.

Tests

Test ID	Scenario	Input / State	Expected Result
UI-01	Status label rendering (checked manually)	Account = Verified	Status label displays “Verified”
UI-02	Withdraw disabled when Suspended	Status = Suspended	Withdraw button disabled
UI-03	Deposit disabled when Closed	Status = Closed	Deposit button disabled
UI-04	Input validation	Amount ≤ 0 or NaN	Error message shown
UI-05	Insufficient funds	Withdraw > balance	“Insufficient funds” shown
UI-06	Withdraw disabled when Suspended	Status = Suspended	Withdraw button disabled
UI-07	Deposit disabled when Closed	Status = Closed	Deposit button disabled
UI-08	State behavior logic	Status = Suspended	Action blocked

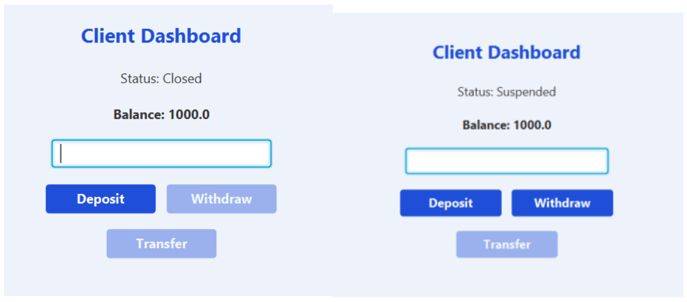
Report:



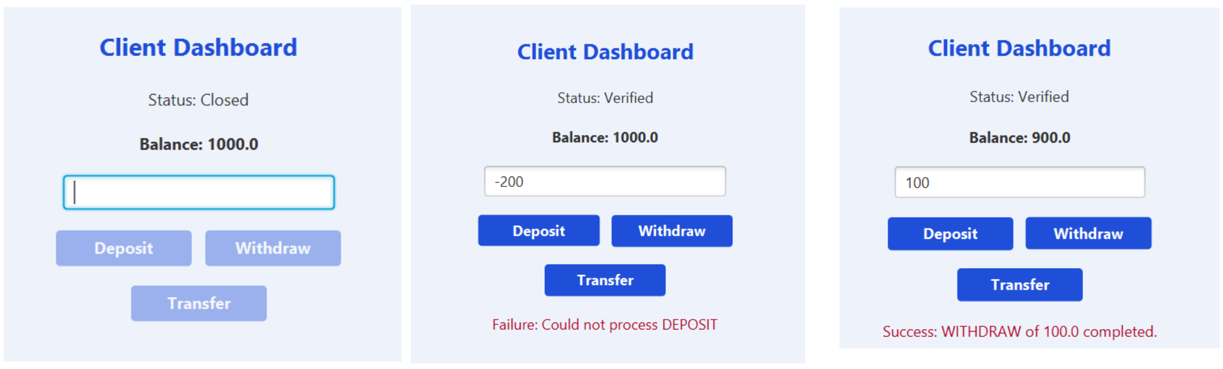
```
Run   BankingAppGUITest ×
BankingAppGUITest (com.banking.ui) 123 ms    ✓ 2 tests passed  2 tests total, 123 ms
  ✓ testDepositDisabledWhenClosed() 122 ms    C:\Users\dell\.jdks\ms-17.0.17\bin\java.exe

un   DashboardUITest ×
✓ DashboardUITest (com.banking)  47 ms    ✓ 1 test passed  1 test total, 47 ms
  ✓ testDashboardHtmlStructure()  47 ms
```

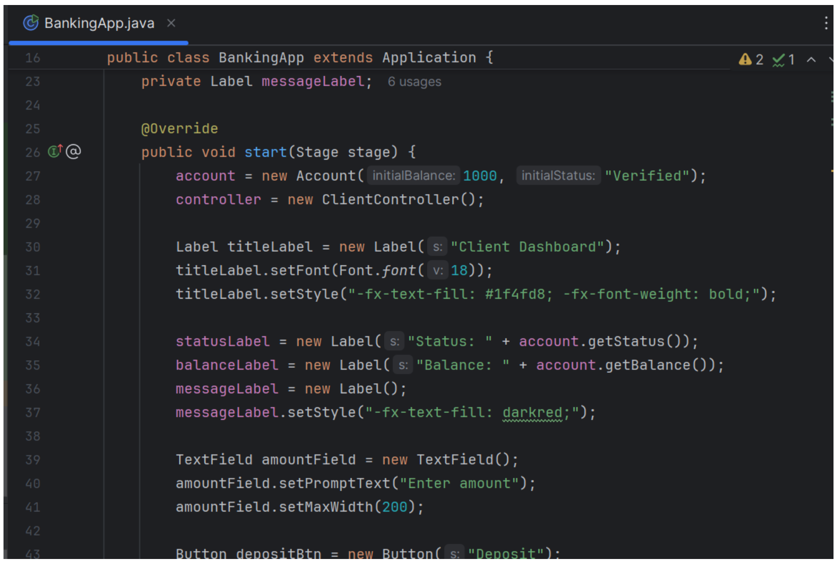
UI bug list

Bug	Description	Status
1	Withdraw button active when Suspended	Fixed
2	Invalid amount not validated	Fixed
3	Closed account allowed deposit	Fixed

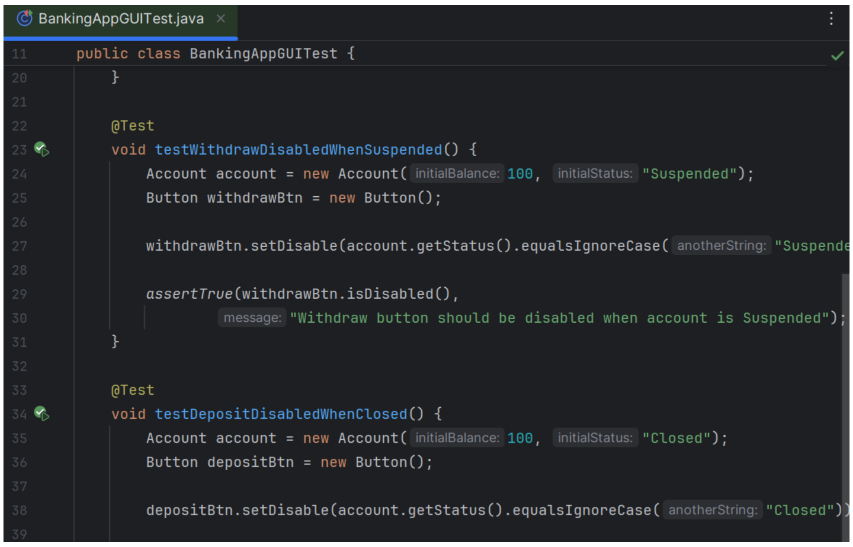


Implementation example:

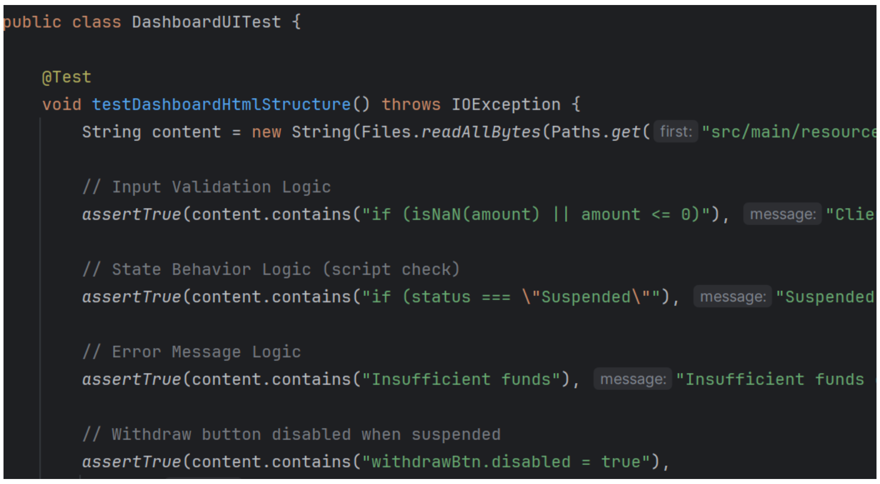




```java
public class BankingApp extends Application {
    private Label messageLabel;  6 usages

    @Override
    public void start(Stage stage) {
        account = new Account( initialBalance: 1000,  initialStatus: "Verified");
        controller = new ClientController();

        Label titleLabel = new Label( s: "Client Dashboard");
        titleLabel.setFont(Font.font( v: 18));
        titleLabel.setStyle("-fx-text-fill: #1f4fd8; -fx-font-weight: bold;");

        statusLabel = new Label( s: "Status: " + account.getStatus());
        balanceLabel = new Label( s: "Balance: " + account.getBalance());
        messageLabel = new Label();
        messageLabel.setStyle("-fx-text-fill: darkred;");

        TextField amountField = new TextField();
        amountField.setPromptText("Enter amount");
        amountField.setMaxWidth(200);

        Button depositBtn = new Button( s: "Deposit");
```



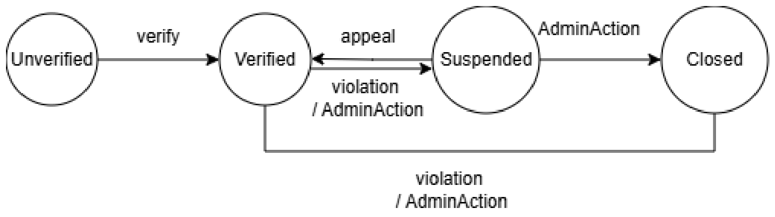
```java
public class BankingAppGUITest {
    }

    @Test
    void testWithdrawDisabledWhenSuspended() {
        Account account = new Account( initialBalance: 100,  initialStatus: "Suspended");
        Button withdrawBtn = new Button();

        withdrawBtn.setDisable(account.getStatus().equalsIgnoreCase( anotherString: "Suspende

        assertTrue(withdrawBtn.isDisabled(),
                 message: "Withdraw button should be disabled when account is Suspended");
    }

    @Test
    void testDepositDisabledWhenClosed() {
        Account account = new Account( initialBalance: 100,  initialStatus: "Closed");
        Button depositBtn = new Button();

        depositBtn.setDisable(account.getStatus().equalsIgnoreCase( anotherString: "Closed")
```



```java
public class DashboardUITest {

    @Test
    void testDashboardHtmlStructure() throws IOException {
        String content = new String(Files.readAllBytes(Paths.get( first: "src/main/resource

        // Input Validation Logic
        assertTrue(content.contains("if (isNaN(amount) || amount <= 0)"),  message: "Clie

        // State Behavior Logic (script check)
        assertTrue(content.contains("if (status === \"Suspended\""),  message: "Suspended

        // Error Message Logic
        assertTrue(content.contains("Insufficient funds"),  message: "Insufficient funds

        // Withdraw button disabled when suspended
        assertTrue(content.contains("withdrawBtn.disabled = true"),
```

Section D: State-Based Testing

State chart:



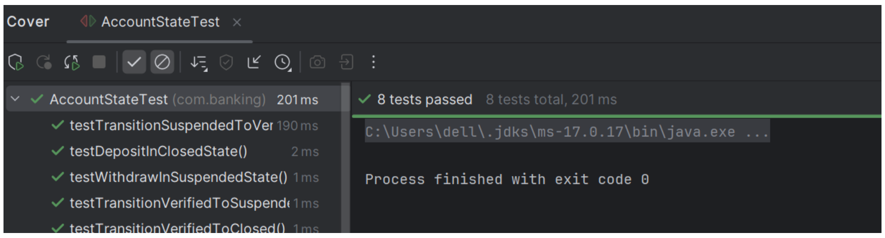
State matrix:

State	Allowed Actions	Illegal Actions	Transitions
Unverified	—	Deposit, Withdraw, Transfer	Verify -> Verified
Verified	Deposit, Withdraw, Transfer	—	Violation/ AdminAction -> Suspended / closed
Suspended	View and Deposit	Withdraw, Transfer	AdminAction -> Closed Appeal -> Verifed
Closed	View only	Deposit, Withdraw, Transfer	

Test table:

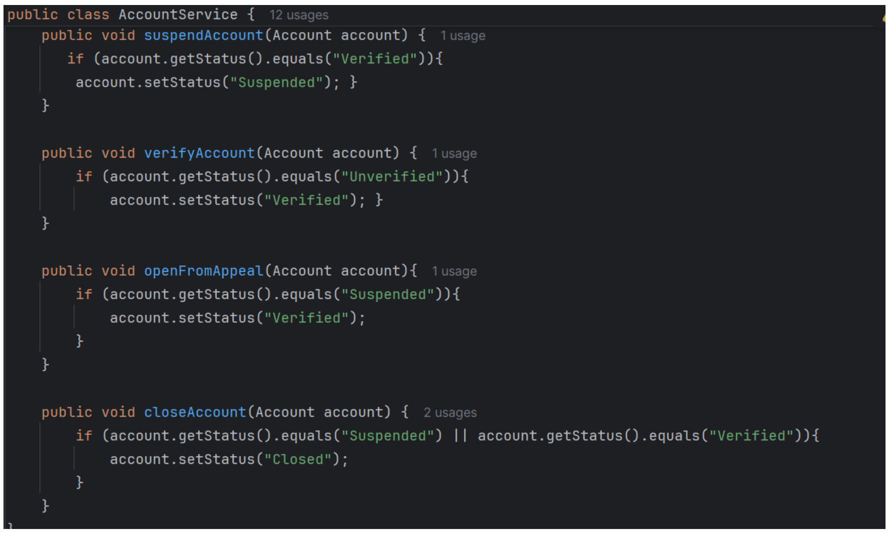
Test ID	Initial State	Action / Transition	Input	Expected Result
ST-01	Closed	Deposit	+100	Transaction fails
ST-02	Suspended	Withdraw	50	Transaction fails
ST-03	Verified	Deposit & Withdraw	+50 / −50	Both succeed
ST-04	Unverified → Verified	Verify account	—	State becomes Verified, withdraw allowed
ST-05	Verified → Suspended	Suspend account	—	State becomes Suspended, withdraw blocked
ST-06	Suspended → Verified	Appeal accepted	—	State restored to Verified, withdraw allowed
ST-07	Suspended → Closed	Close account	—	State becomes Closed, withdraw blocked
ST-08	Verified → Closed	Close account	—	State becomes Closed, withdraw blocked

Report:

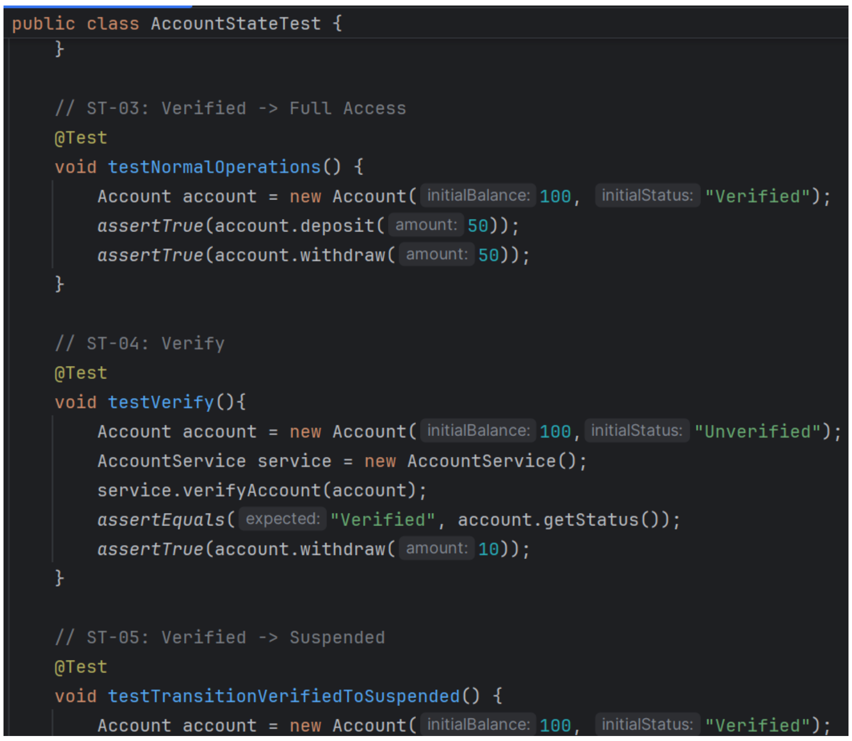


```
Cover    AccountStateTest ×

AccountStateTest (com.banking)  201 ms      ✓ 8 tests passed  8 tests total, 201 ms
  ✓ testTransitionSuspendedToVer 190 ms     C:\Users\dell\.jdks\ms-17.0.17\bin\java.exe ...
  ✓ testDepositInClosedState()     2 ms
  ✓ testWithdrawInSuspendedState() 1 ms     Process finished with exit code 0
  ✓ testTransitionVerifiedToSuspend 1 ms
  ✓ testTransitionVerifiedToClosed() 1 ms
```

Implementation example:



```java
public class AccountService {  12 usages
    public void suspendAccount(Account account) {  1 usage
       if (account.getStatus().equals("Verified")){
        account.setStatus("Suspended"); }
    }

    public void verifyAccount(Account account) {  1 usage
        if (account.getStatus().equals("Unverified")){
            account.setStatus("Verified"); }
    }

    public void openFromAppeal(Account account){  1 usage
        if (account.getStatus().equals("Suspended")){
            account.setStatus("Verified");
        }
    }

    public void closeAccount(Account account) {  2 usages
        if (account.getStatus().equals("Suspended") || account.getStatus().equals("Verified")){
            account.setStatus("Closed");
        }
    }
}
```



```java
public class AccountStateTest {
    }

    // ST-03: Verified -> Full Access
    @Test
    void testNormalOperations() {
        Account account = new Account( initialBalance: 100,  initialStatus: "Verified");
        assertTrue(account.deposit( amount: 50));
        assertTrue(account.withdraw( amount: 50));
    }

    // ST-04: Verify
    @Test
    void testVerify(){
        Account account = new Account( initialBalance: 100, initialStatus: "Unverified");
        AccountService service = new AccountService();
        service.verifyAccount(account);
        assertEquals( expected: "Verified", account.getStatus());
        assertTrue(account.withdraw( amount: 10));
    }

    // ST-05: Verified -> Suspended
    @Test
    void testTransitionVerifiedToSuspended() {
        Account account = new Account( initialBalance: 100,  initialStatus: "Verified");
```

Section E: Test-Driven Development (TDD)

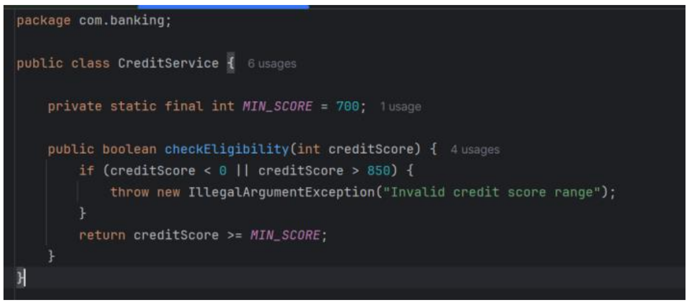
Test cases:

Test ID	Description	Input	Expected Output
TDD-01	Approve high credit score	750	true
TDD-02	Reject low credit score	500	false
TDD-03	Boundary approval	700	True
TDD-04	Boundary rejection	699	false
TDD-05	Invalid score range	-1, 900	Exception

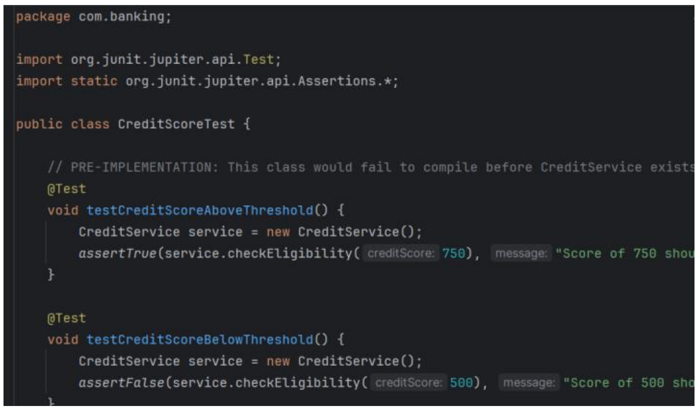
Stub code

```
public class CreditService {
    public boolean checkEligibility(int creditScore) {
        return false;
    }
}
```

Full Implementation:

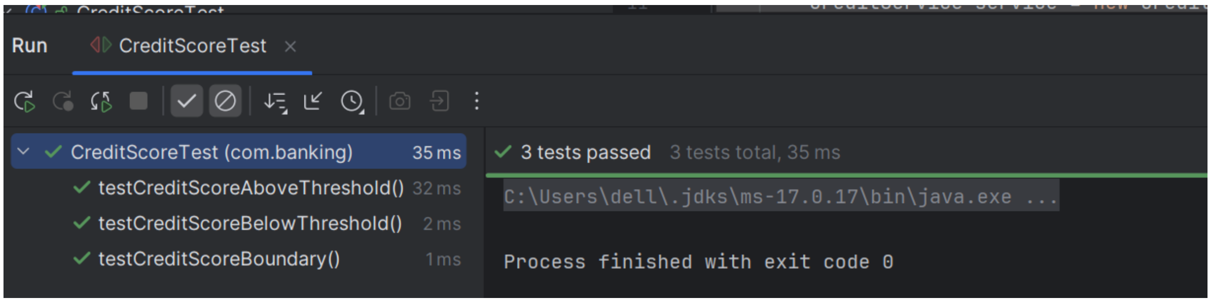


```
package com.banking;

public class CreditService {  6 usages

    private static final int MIN_SCORE = 700;  1 usage

    public boolean checkEligibility(int creditScore) {  4 usages
        if (creditScore < 0 || creditScore > 850) {
            throw new IllegalArgumentException("Invalid credit score range");
        }
        return creditScore >= MIN_SCORE;
    }
}
```



```
package com.banking;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class CreditScoreTest {

    // PRE-IMPLEMENTATION: This class would fail to compile before CreditService exists
    @Test
    void testCreditScoreAboveThreshold() {
        CreditService service = new CreditService();
        assertTrue(service.checkEligibility( creditScore: 750),  message: "Score of 750 shou
    }

    @Test
    void testCreditScoreBelowThreshold() {
        CreditService service = new CreditService();
        assertFalse(service.checkEligibility( creditScore: 500),  message: "Score of 500 sho
    }
```

Report (after implementation)



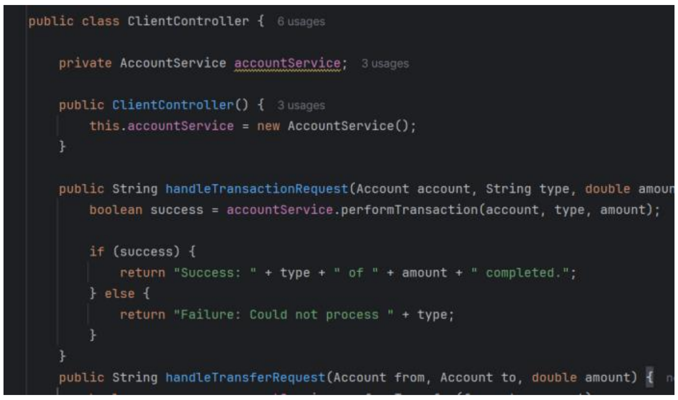
```
✓ CreditScoreTest (com.banking)      35 ms     ✓ 3 tests passed  3 tests total, 35 ms
   ✓ testCreditScoreAboveThreshold() 32 ms     C:\Users\dell\.jdks\ms-17.0.17\bin\java.exe ...
   ✓ testCreditScoreBelowThreshold()  2 ms
   ✓ testCreditScoreBoundary()        1 ms     Process finished with exit code 0
```

Section F: Simple Integration Testing

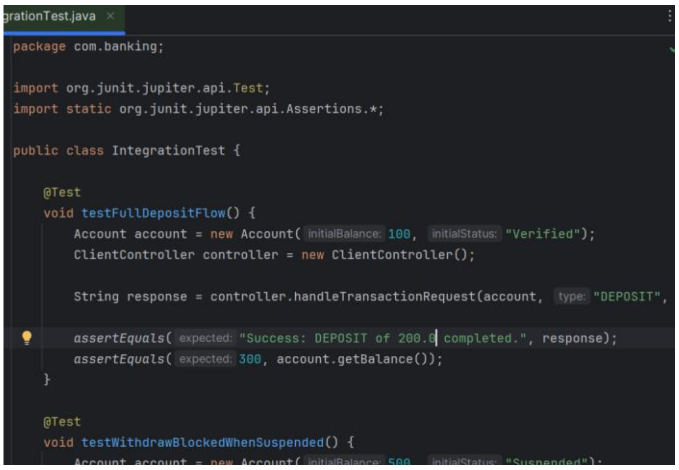
System Components Involved

- GUI Layer (Simulated): User actions such as clicking *Deposit*, *Withdraw*, or *Transfer*
- ClientController: Receives requests and returns user-facing response messages
- AccountService: Coordinates transaction and transfer logic
- TransactionProcessor / TransferProcessor: Executes business rules
- Account: Updates balance and enforces state constraints

Implementation:

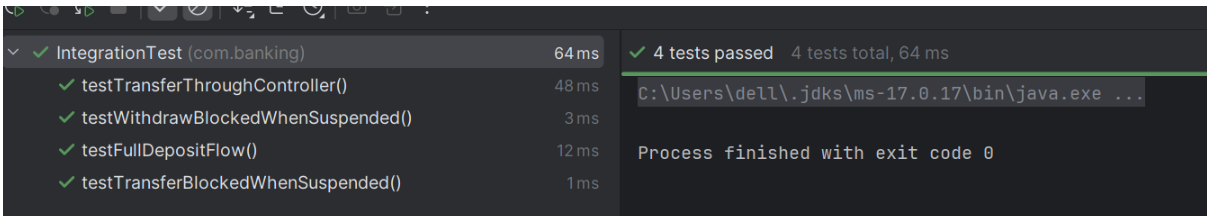


```java
public class ClientController {  6 usages

    private AccountService accountService;  3 usages

    public ClientController() {  3 usages
        this.accountService = new AccountService();
    }

    public String handleTransactionRequest(Account account, String type, double amoun
        boolean success = accountService.performTransaction(account, type, amount);

        if (success) {
            return "Success: " + type + " of " + amount + " completed.";
        } else {
            return "Failure: Could not process " + type;
        }
    }
    public String handleTransferRequest(Account from, Account to, double amount) {
```



```java
grationTest.java ×
package com.banking;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class IntegrationTest {

    @Test
    void testFullDepositFlow() {
        Account account = new Account( initialBalance: 100,  initialStatus: "Verified");
        ClientController controller = new ClientController();

        String response = controller.handleTransactionRequest(account,  type: "DEPOSIT",

        assertEquals( expected: "Success: DEPOSIT of 200.0 completed.", response);
        assertEquals( expected: 300, account.getBalance());
    }

    @Test
    void testWithdrawBlockedWhenSuspended() {
        Account account = new Account( initialBalance: 500,  initialStatus: "Suspended");
```

Report:



```
✓ IntegrationTest (com.banking)              64 ms    ✓ 4 tests passed  4 tests total, 64 ms
    ✓ testTransferThroughController()         48 ms    C:\Users\dell\.jdks\ms-17.0.17\bin\java.exe ...
    ✓ testWithdrawBlockedWhenSuspended()       3 ms
    ✓ testFullDepositFlow()                   12 ms    Process finished with exit code 0
    ✓ testTransferBlockedWhenSuspended()       1 ms
```