Chapter 3

JUnit Testing for Parabank Logic

3.1 Overview

JUnit is a widely used testing framework for Java that facilitates unit testing to ensure each component of the codebase performs as intended. This helps in early bug detection, better code reliability, and smoother integration with development workflows.

In this project, JUnit is employed to validate backend functionalities of a simulated banking service, including login, registration, fund transfer, and bill payment.

3.2 Why JUnit?

- Facilitates fast and repeatable tests.
- Supports automated testing workflows.
- Easy integration with NetBeans and other Java IDEs.
- Provides clear feedback through assertions.

3.3 Tested Functionalities

The following features were tested using JUnit:

1. User Login (success and failure)
2. User Registration (valid and invalid)
3. Fund Transfer (success and exception case)
4. Bill Payment (sufficient and insufficient funds)

3.4 JUnit Test Class

The complete test class used for verifying the banking logic:

3.5 Execution in NetBeans

To run the JUnit test in NetBeans:

- Right-click on the test class file.
- Click **Run File**.
- NetBeans will compile and execute the tests and show results in the output window.

```
Running BankServiceTest
Testsuite: BankServiceTest
Testcase: testLoginSuccess PASSED
Testcase: testLoginFail PASSED
Testcase: testValidRegistration PASSED
Testcase: testInvalidRegistration PASSED
Testcase: testTransferSuccess PASSED
Testcase: testTransferFail PASSED
Testcase: testBillPayment PASSED
BUILD SUCCESSFUL (total time: 1 second)
```

3.6 Conclusion

Using JUnit enabled structured and automated testing of core functionalities. The test outcomes provided clear feedback and ensured that logic like login validation, fund transfer, and bill payment met expected behaviors. This practice boosts software reliability and facilitates cleaner codebase management in Java applications.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class BankServiceTest {

    private BankService service;

    @BeforeEach
    public void setUp() {
        service = new BankService();
    }

    @Test
    public void testLoginSuccess() {
        assertTrue(service.login("admin", "1234"));
    }

    @Test
    public void testLoginFail() {
        assertFalse(service.login("user", "wrong"));
    }

    @Test
    public void testValidRegistration() {
        assertTrue(service.register("test@example.com", "pass123"));
    }

    @Test
    public void testInvalidRegistration() {
        assertFalse(service.register("invalidEmail", "123"));
    }

    @Test
    public void testTransferSuccess() {
        assertEquals(500.0, service.transferFunds(1000.0, 500.0));
    }

    @Test
    public void testTransferFail() {
        Exception exception = assertThrows(IllegalArgumentException.class, () -> {
            service.transferFunds(100.0, 200.0);
        });
        assertEquals("Insufficient balance", exception.getMessage());
    }

    @Test
    public void testBillPayment() {
        assertTrue(service.payBill(500.0, 400.0));
        assertFalse(service.payBill(100.0, 200.0));
    }
}
```

Figure 3.1: J-unit code