Software Testing, Validation and Verification

Online Banking System

Mohamed Abdelaty Mohamed	20P9634
Mohaned Hesham Nasr	20P2204
Abdallah Mohamed Elkhalafawy	20P4739
Mahamad Hany Mahros Elshabory	20P9302
Abdelrahman Mahmoud Agha	20P6300
David Waseem Wasfy	18P3391

Submitted to Assis. Prof. Dr. Islam Mahmoud Elmadah

Online Banking System

Users can buy items and pay bills as well as transfer money to other accounts. They must be able to see statements of their bank accounts, get notified for each transaction.

PROJECT COMPONENT :

-Account test cases

getBalance Implementation

```
    public double getBalance(){
        return balance;
```

Test Cases

```
    @Test
    public void getBalanceTest1() {
        Account acc = new Account(1500);
        assertEquals(1500, acc.getBalance(), 2);
    }
    @Test
    public void getBalanceTest2() {
        Account acc = new Account(1);
        assertEquals(1, acc.getBalance(),2);
    }
    @Test
    public void getBalanceTest3() {
        Account acc = new Account(0);
        assertEquals(0, acc.getBalance(),2);
    }
    @Test
    public void getBalanceTest4() {
        Account acc = new Account(-1);
        assertEquals(0, acc.getBalance(),2);
    }
    @Test
    public void getBalanceTest5() {
        Account acc = new Account(-100);
        assertEquals(0, acc.getBalance(),2);
    }
```

checkAmount Implementation

```java
    private boolean checkAmount(double inputAmount){
        if(inputAmount <= 0)
            return false;
        return balance >= inputAmount;
    }
```

Test cases

```java
@Test
public void checkAmountTest1() throws NoSuchMethodException, InvocationTargetException, IllegalAccessException {
    Account account = new Account(1000);

    // Use reflection to access the private method withdrawTransaction()
    Method checkAmountMethod = Account.class.getDeclaredMethod("checkAmount", double.class);
    checkAmountMethod.setAccessible(true);

    // Invoke the private method
    boolean result = (boolean)checkAmountMethod.invoke(account, 100.0);

    assertTrue(result);
}
@Test
public void checkAmountTest2() throws NoSuchMethodException, InvocationTargetException, IllegalAccessException {
    Account account = new Account(1500);

    // Use reflection to access the private method withdrawTransaction()
    Method checkAmountMethod = Account.class.getDeclaredMethod("checkAmount", double.class);
    checkAmountMethod.setAccessible(true);

    // Invoke the private method
    boolean result = (boolean)checkAmountMethod.invoke(account, 0);

    assertFalse(result);
}
@Test
public void checkAmountTest3() throws NoSuchMethodException, InvocationTargetException, IllegalAccessException {
    Account account = new Account(1500);

    // Use reflection to access the private method withdrawTransaction()
    Method checkAmountMethod = Account.class.getDeclaredMethod("checkAmount", double.class);
    checkAmountMethod.setAccessible(true);

    // Invoke the private method
    boolean result = (boolean)checkAmountMethod.invoke(account, -100.0);

    assertFalse(result);
```

```java
public void checkAmountTest4() throws NoSuchMethodException, InvocationTargetException, IllegalAccessException {
    Account account = new Account(1500);

    // Use reflection to access the private method withdrawTransaction()
    Method checkAmountMethod = Account.class.getDeclaredMethod("checkAmount", double.class);
    checkAmountMethod.setAccessible(true);

    // Invoke the private method
    boolean result = (boolean)checkAmountMethod.invoke(account, -1);

    assertFalse(result);
}
@Test
public void checkAmountTest5() throws NoSuchMethodException, InvocationTargetException, IllegalAccessException {
    Account account = new Account(1500);

    // Use reflection to access the private method withdrawTransaction()
    Method checkAmountMethod = Account.class.getDeclaredMethod("checkAmount", double.class);
    checkAmountMethod.setAccessible(true);

    // Invoke the private method
    boolean result = (boolean)checkAmountMethod.invoke(account, 1);

    assertTrue(result);
}
@Test
public void checkAmountTest6() throws NoSuchMethodException, InvocationTargetException, IllegalAccessException {
    Account account = new Account(1500);

    // Use reflection to access the private method withdrawTransaction()
    Method checkAmountMethod = Account.class.getDeclaredMethod("checkAmount", double.class);
    checkAmountMethod.setAccessible(true);

    // Invoke the private method
    boolean result = (boolean)checkAmountMethod.invoke(account, 1600);

    assertFalse(result);
```

Withdraw Implementation

```java
    private boolean withdraw(double inputAmount) {
        if (checkAmount(inputAmount)) {
            balance -= inputAmount;
            withdrawTransaction(inputAmount);
            return true;
        } else{
            return false;
        }
    }
```

Test cases

```java
@Test
public void withdrawTest1() throws NoSuchMethodException, InvocationTargetException, IllegalAccessException {
    Account account = new Account(1500);

    // Use reflection to access the private method withdrawTransaction()
    Method withdrawMethod = Account.class.getDeclaredMethod("withdraw", double.class);
    withdrawMethod.setAccessible(true);

    boolean result = (boolean) withdrawMethod.invoke(account, 100);

    assertTrue(result);

    assertEquals(1400, account.getBalance(),2);
}
@Test
public void withdrawTest2() throws NoSuchMethodException, InvocationTargetException, IllegalAccessException {
    Account account = new Account(1500);

    // Use reflection to access the private method withdrawTransaction()
    Method withdrawMethod = Account.class.getDeclaredMethod("withdraw", double.class);
    withdrawMethod.setAccessible(true);

    boolean result = (boolean) withdrawMethod.invoke(account, 1);

    assertTrue(result);

    assertEquals(1499, account.getBalance(),2);
}
@Test
public void withdrawTest3() throws NoSuchMethodException, InvocationTargetException, IllegalAccessException {
    Account account = new Account(1500);

    // Use reflection to access the private method withdrawTransaction()
    Method withdrawMethod = Account.class.getDeclaredMethod("withdraw", double.class);
    withdrawMethod.setAccessible(true);

    boolean result = (boolean) withdrawMethod.invoke(account, 0);
```

```java
    assertEquals(1500, account.getBalance(),2);
}
@Test
public void withdrawTest4() throws NoSuchMethodException, InvocationTargetException, IllegalAccessException {
    Account account = new Account(1500);

    // Use reflection to access the private method withdrawTransaction()
    Method withdrawMethod = Account.class.getDeclaredMethod("withdraw", double.class);
    withdrawMethod.setAccessible(true);

    // Invoke the private method
    boolean result = (boolean) withdrawMethod.invoke(account, -1);

    assertFalse(result);

    assertEquals(1500, account.getBalance(),2);
}
@Test
public void withdrawTest5() throws NoSuchMethodException, InvocationTargetException, IllegalAccessException {
    Account account = new Account(1500);

    // Use reflection to access the private method withdrawTransaction()
    Method withdrawMethod = Account.class.getDeclaredMethod("withdraw", double.class);
    withdrawMethod.setAccessible(true);

    // Invoke the private method
    withdrawMethod.invoke(account, -100.0);

    boolean result = (boolean) withdrawMethod.invoke(account, -100.0);

    assertFalse(result);

    assertEquals(1500, account.getBalance(),2);
}
@Test
public void withdrawTest6() throws NoSuchMethodException, InvocationTargetException, IllegalAccessException {
    Account account = new Account(1500);
```

```java
@Test
public void withdrawTest6() throws NoSuchMethodException, InvocationTargetException, IllegalAccessException {
    Account account = new Account(1500);

    // Use reflection to access the private method withdrawTransaction()
    Method withdrawMethod = Account.class.getDeclaredMethod("withdraw", double.class);
    withdrawMethod.setAccessible(true);

    // Invoke the private method
    withdrawMethod.invoke(account, -100.0);

    boolean result = (boolean) withdrawMethod.invoke(account, 1501.0);

    assertFalse(result);

    assertEquals(1500, account.getBalance(),2);
}
```

withdrawTransaction Implementation

```java
    private void withdrawTransaction(double amount){
        if(transactions[9] == null){
            transactions[transactionIndex] = "W" + decfor.format(amount);
            transactionIndex++;
        }else{
            for(int i = 0; i < 9; i++)
                transactions[i] = transactions[i+1];
            transactions[9] = "W" + amount;
        }
    }
```

Test cases

```java
@Test
public void withdrawTransactionTest() throws NoSuchMethodException, InvocationTargetException, IllegalAccessException {
    Account account = new Account(1500);

    // Use reflection to access the private method withdrawTransaction()
    Method withdrawTransactionMethod = Account.class.getDeclaredMethod("withdrawTransaction", double.class);
    withdrawTransactionMethod.setAccessible(true);

    // Invoke the private method
    withdrawTransactionMethod.invoke(account, 50.0);

    String[] statements = account.getStatements();

    assertEquals("W50.00", statements[0]);
}
```

payFor implementation

```
public boolean payFor(Item item) {
    double price = item.getPrice();
    if (withdraw(price))
        return true;
    else
        return false;
}

public boolean payFor(Bill bill) {
    double price = bill.getPrice();
    if (withdraw(price)) return true;
    else return false;
}
```

Test cases

```
    @Test
    public void payForTest1() {
        Account acc4 = new Account(1500);
        assertFalse(acc4.payFor(new Item(0)));
        assertEquals(1500, acc4.getBalance(),2);
    }
    @Test
    public void payForTest2() {
        Account acc4 = new Account(1500);
        assertTrue(acc4.payFor(new Item(150)));
        assertEquals(1350, acc4.getBalance(),2);
    }
    @Test
    public void payForTest3() {
        Account acc4 = new Account(1500);
        assertFalse(acc4.payFor(new Item(-50)));
        assertEquals(1500, acc4.getBalance(),2);
    }
    @Test
    public void payForTest4() {
        Account acc4 = new Account(1500);
        assertTrue(acc4.payFor(new Item(1))); //0 -> should be false//
        assertEquals(1499, acc4.getBalance(),2);
    }
    @Test
    public void payForTest5() {
        Account acc5 = new Account(1500);
        assertFalse(acc5.payFor(new Bill(-1)));
        assertEquals(1500, acc5.getBalance(),2);
    }
    @Test
    public void payForTest6() {
        Account acc4 = new Account(1000);
        assertFalse(acc4.payFor(new Item(0)));
        assertEquals(1000, acc4.getBalance(),2);
    }
    @Test
    public void payForTest7() {
```

```java
@Test
public void payForTest7() {
    Account acc4 = new Account(1500);
    assertTrue(acc4.payFor(new Item(100)));
    assertEquals(1400, acc4.getBalance(),2);
}
@Test
public void payForTest8() {
    Account acc4 = new Account(1500);
    assertFalse(acc4.payFor(new Item(-100)));
    assertEquals(1500, acc4.getBalance(),2);
}
@Test
public void payForTest9() {
    Account acc4 = new Account(1500);
    assertTrue(acc4.payFor(new Item(1))); //0 -> should also be false//
    assertEquals(1499, acc4.getBalance(),2);
}
@Test
public void payForTest10() {
    Account acc5 = new Account(1500);
    assertFalse(acc5.payFor(new Item(-1)));
    assertEquals(1500, acc5.getBalance(),2);
}
@Test
public void payForTest11() {
    Account acc5 = new Account(1500);
    assertFalse(acc5.payFor(new Item(1501)));
    assertEquals(1500, acc5.getBalance(),2);
}
@Test
public void payForTest12() {
    Account acc5 = new Account(1500);
    assertFalse(acc5.payFor(new Bill(1700)));
    assertEquals(1500, acc5.getBalance(),2);
}
```

Deposit Implementation

```java
    public void deposit(double inputAmount) {
        if (inputAmount > 0) {
            balance += inputAmount;
            depositTransaction(inputAmount);
        }
    }
```

Test cases

```
@Test
public void depositTest1() {
    Account acc3 = new Account(1000);
    acc3.deposit(100);
    assertEquals(1100, acc3.getBalance(),2);
}
@Test
public void depositTest2() {
    Account acc3 = new Account(1500);
    acc3.deposit(-100);
    assertEquals(1500, acc3.getBalance(),2);
}
@Test
public void depositTest3() {
    Account acc3 = new Account(1500);
    acc3.deposit(0);
    assertEquals(1500, acc3.getBalance(),2);
}
@Test
public void depositTest4() {
    Account acc3 = new Account(1500);
    acc3.deposit(100);
    assertEquals(1600, acc3.getBalance(),2);
}
@Test
public void depositTest5() {
    Account acc3 = new Account(1500);
    acc3.deposit(-5);
    assertEquals(1500, acc3.getBalance(),2);
}
```

DepositTransactions Implementation

```java
    private void depositTransaction(double amount){
        if(transactions[9] == null){
            transactions[transactionIndex] = "D" + decfor.format(amount);
            transactionIndex++;
        }else{
            for(int i = 0; i < 9; i++)
                transactions[i] = transactions[i+1];
            transactions[9] = "D" + amount;
        }
    }
```

Test cases

```java
@Test
public void depositTransactionTest() throws NoSuchMethodException, InvocationTargetException, IllegalAccessException {
    Account account = new Account(1500);

    // Use reflection to access the private method withdrawTransaction()
    Method depositTransactionMethod = Account.class.getDeclaredMethod("depositTransaction", double.class);
    depositTransactionMethod.setAccessible(true);

    // Invoke the private method
    depositTransactionMethod.invoke(account, 100.0);

    String[] statements = account.getStatements();

    assertEquals("D100.00", statements[0]);
}
```

getStatements Implementation

```java
    public String[] getStatements(){
        return transactions;
    }
```

Test cases

```java
@Test
public void getStatementsTest() {
    Account acc8 = new Account(1500);

    assertEquals("W50.00",list[1]);
    assertEquals("W200.00",list[2]);

}
```

transferMoney implementation

```java
    public boolean transferMoney(User user,double amount){
        if (withdraw(amount)) {
            user.acceptMoney(amount);
            return true;
        } else{
            return false;
        }
    }
```

Test cases

```java
@Test
public void TransferMoneyTest1() {
    Account acc7 = new Account(1500);
    User user = new User("Abdallah", "Elkhalafawy", "ak", "01020011111",1500 ,"0202020", "2020100");
    assertFalse(acc7.transferMoney(user, 0));
    assertEquals(1500, user.getAccount().getBalance(),2);
    assertEquals(1500, acc7.getBalance(),2);
}
@Test
public void TransferMoneyTest2() {
    Account acc7 = new Account(1500);
    User user = new User("Abdallah", "Elkhalafawy", "ak", "01020011111",1500 ,"0202020", "20201500");
    assertTrue(acc7.transferMoney(user, 1));
    assertEquals(1501, user.getAccount().getBalance(),2);
    assertEquals(1499, acc7.getBalance(),2);
}
@Test
public void TransferMoneyTest3() {
    Account acc7 = new Account(1500);
    User user = new User("Abdallah", "Elkhalafawy", "ak", "01020011111",1500 ,"0202020", "2020100");
    assertFalse(acc7.transferMoney(user, -1));
    assertEquals(1500, user.getAccount().getBalance(),2);
    assertEquals(1500, acc7.getBalance(),2);
}
@Test
public void TransferMoneyTest4() {
    Account acc7 = new Account(1500);
    User user = new User("Abdallah", "Elkhalafawy", "ak", "01020011111",1500 ,"0202020", "2020100");
    assertFalse(acc7.transferMoney(user, -500));
    assertEquals(1500, user.getAccount().getBalance(),2);
    assertEquals(1500, acc7.getBalance(),2);
}
@Test
```

```java
@Test
public void TransferMoneyTest5() {
    Account acc7 = new Account(1500);
    User user = new User("Abdallah", "Elkhalafawy", "ak", "01020011111",1500 ,"0202020", "2020100");
    assertTrue(acc7.transferMoney(user, 500));
    assertEquals(2000, user.getAccount().getBalance(),2);
    assertEquals(1000, acc7.getBalance(),2);
}
@Test
public void TransferMoneyTest6() {
    Account acc7 = new Account(1400);
    User user = new User("Abdallah", "Elkhalafawy", "ak", "01020011111",1500 ,"0202020", "2020100");
    assertFalse(acc7.transferMoney(user, 1500));
    assertEquals(1500, user.getAccount().getBalance(),2);
    assertEquals(1400, acc7.getBalance(),2);
}
```

User Test Cases

acceptMoney Implementation

```java
public void acceptMoney(double amount){
    account.deposit(amount);
}
```

Test cases

```java
@Test
public void acceptMoneyTest(){
    User user4 = new User("Abdallah", "Elkhalafawy", "ak", "01020011111", 1500, "akkk", "12345678912345");
    user4.acceptMoney(100);
    assertEquals(1600, user4.getAccount().getBalance(), 2);
}
```

Getters test cases

```java
public class UserTest {
    @Test
    public void getNationalIdTest(){
        User user1 = new User("Abdallah", "Elkhalafawy", "ak", "01020011111", 1500, "akkk", "12345678912345");
        assertEquals("12345678912345", user1.getNationalId());
    }

    @Test
    public void getPasswordTest(){
        User user2 = new User("Abdallah", "Elkhalafawy", "ak", "01020011111", 1500, "akkk", "12345678912345");
        assertEquals("akkk", user2.getPassword());
    }

    @Test
    public void getAccountTest(){
        User user3 = new User("Abdallah", "Elkhalafawy", "ak", "01020011111", 1500, "akkk", "12345678912345");
        assertEquals(1500 ,user3.getAccount().getBalance(),2);
    }

    @Test
    public void acceptMoneyTest(){
        User user4 = new User("Abdallah", "Elkhalafawy", "ak", "01020011111", 1500, "akkk", "12345678912345");
        user4.acceptMoney(100);
        assertEquals(1600, user4.getAccount().getBalance(), 2);
    }

    @Test
    public void getUsernameTest(){
        User user5 = new User("Abdallah", "Elkhalafawy", "ak", "01020011111", 1000, "akkk", "12345678912345");
        assertEquals("ak", user5.getUsername());
    }
```

Bill Test Cases

```java
@Test
public void getPriceTest1() {
    Bill bill = new Bill(1500);
    assertEquals(1500, bill.getPrice(),2);
}

@Test
public void getPriceTest2() {
    Bill bill = new Bill(1);
    assertEquals(1, bill.getPrice(), 2);
}

@Test
public void getPriceTest3() {
    Bill bill = new Bill(0);
    assertEquals(0, bill.getPrice(), 2);
}

@Test
public void getPriceTest4() {
    Bill bill = new Bill(-1);
    assertEquals(0, bill.getPrice(), 2);
}

@Test
public void getPriceTest5() {
    Bill bill = new Bill(-1000);
    assertEquals(0, bill.getPrice(), 2);
}
```

Item Test Cases

```java
public class ItemTest {
    @Test
    public void getPriceTest1() {
        Item item = new Item(1000);
        assertEquals(1000, item.getPrice(), 2);
    }

    @Test
    public void getPriceTest2() {
        Item item = new Item(1);
        assertEquals(1, item.getPrice(), 2);
    }

    @Test
    public void getPriceTest3() {
        Item item = new Item(0);
        assertEquals(0, item.getPrice(), 2);
    }

    @Test
    public void getPriceTest4() {
        Item item = new Item(-1);
        assertEquals(0, item.getPrice(),2);
    }

    @Test
    public void getPriceTest5() {
        Item item = new Item(-1000);
        assertEquals(0, item.getPrice(),2);
    }
```

The First Test Suite

```
@RunWith(Suite.class)
@Suite.SuiteClasses({
        BillTest.class,
        ItemTest.class,
        AccountTest.class,
        UserTest.class
})

public class UnitTestSuite {
```

Example 1: Making a new Account (Constructor) using a negative value.

```
@Test
public void getBalanceTest4() {
    Account acc = new Account(-1);
    assertEquals(0, acc.getBalance(),2);
}
```

Before Fix

```
public Account(double balance) {
        this.balance = 0;
    idCounter++;
    transactionIndex = 0;
}
```

After

```java
public Account(double balance) {
    if(balance > 0)
        this.balance = balance;
    else
        this.balance = 0;
    idCounter++;
    transactionIndex = 0;
}
```

Added if Statements to check balance is not negative

Example 2: In Deposit / Pay for Item when the number is negative it shouldn't be accepted

```java
}
@Test
public void depositTest2() {
    Account acc3 = new Account(1500);
    acc3.deposit(-100);
    assertEquals(1500, acc3.getBalance(),2);
}
@Test
```

Deposit Function

```java
public void deposit(double inputAmount) {

        balance += inputAmount;
        depositTransaction(inputAmount);

}
```

After

```
public void deposit(double inputAmount) {
  if(inputAmount>0){
      balance += inputAmount;
      depositTransaction(inputAmount);
  }
```

Added If to only accept a positive Value(Fixed this issue for all functions that take a value (Pay Bill, Transfer Money, Buy Item))

Integration Testing

Account Integration Testing.

-Testing these functions together.

- deposit
- payFor
- transferMoney
- getStatement
- withdraw
- checkBalance
- depositTransaction
- withdrawTransaction

```java
public void deposit_payForTest() {
```

```java
public void deposit_payFor_transferTest() {
```

```java
public void deposit_payFor_transfer_statementsTest() {
```

```java
public void deposit_payFor_transfer_statements_withdrawTest() throws NoSuchMethodException, InvocationTargetException, IllegalAccessException {
```

```java
@Test
public void deposit_payFor_transfer_statements_withdraw_checkBalanceTest() throws NoSuchMethodException, InvocationTargetException, IllegalAccessException {
    Account acc = new Account(1000);
```

```java
@Test
public void deposit_payFor_transfer_statements_withdraw_checkBalance_depositTransactionTest() throws NoSuchMethodException, InvocationTargetException, IllegalAccessExcepti
    Account acc = new Account(1000);
```

```java
@Test
public void deposit_payFor_transfer_statements_withdraw_checkBalance_depositTransaction_Test() throws NoSuchMethodException, InvocationTargetException, IllegalAccessExcept
    Account acc = new Account(1000);
```

Tests passed: 7 of 7 tests – 34 ms

AccountIntegrationTest (com.example.tes	34 ms
deposit_payFor_transfer_statements_w	29 ms
deposit_payFor_transfer_statements_wi	2 ms
deposit_payFor_transfer_statements_wi	1 ms
deposit_payForTest	0 ms
deposit_payFor_transferTest	0 ms
deposit_payFor_transfer_statements_wi	1 ms
deposit_payFor_transfer_statementsTes	1 ms

```
C:\Users\mosta\.jdks\openjdk-18.0.1.1\bin\java.exe ...
--AccountIntegrationTest-- test case started
--AccountIntegrationTest-- test case ended
--AccountIntegrationTest-- test case started
--AccountIntegrationTest-- test case ended
--AccountIntegrationTest-- test case started
--AccountIntegrationTest-- test case ended
--AccountIntegrationTest-- test case started
--AccountIntegrationTest-- test case ended
--AccountIntegrationTest-- test case started
--AccountIntegrationTest-- test case ended
--AccountIntegrationTest-- test case started
--AccountIntegrationTest-- test case ended
--AccountIntegrationTest-- test case started
--AccountIntegrationTest-- test case ended
--AccountIntegrationTest-- Class Testing ended

Process finished with exit code 0
```

User Integration Testing

Testing these functions together.

- getNationalId
- getPassword
- getUserName
- getAccount
- acceptMoney

```java
public class UserIntegrationTest {
    @Test
    public void getNationalId_PasswordTest(){
        User user1 = new User("Abdallah", "Elkhalafawy", "ak", "01020011111",1500 ,"0202020", "2020100");
        assertEquals("2020100", user1.getNationalId());
        assertEquals("0202020", user1.getPassword());
    }

    @Test
    public void getNationalId_Password_UsernameTest(){
        User user1 = new User("Abdallah", "Elkhalafawy", "ak", "01020011111",1500 ,"0202020", "2020100");
        assertEquals("2020100", user1.getNationalId());
        assertEquals("0202020", user1.getPassword());
        assertEquals("ak", user1.getUsername());
    }

    @Test
    public void getNationalId_Password_Username_AccountTest(){
        User user1 = new User("Abdallah", "Elkhalafawy", "ak", "01020011111",1500 ,"0202020", "2020100");
        assertEquals("2020100", user1.getNationalId());
        assertEquals("0202020", user1.getPassword());
        assertEquals("ak", user1.getUsername());
        assertEquals(1500 ,user1.getAccount().getBalance(),2);
    }

    @Test
    public void getNationalId_Password_Username_Account_acceptMoneyTest(){
        User user1 = new User("Abdallah", "Elkhalafawy", "ak", "01020011111",1500 ,"0202020", "2020100");
        assertEquals(1500 ,user1.getAccount().getBalance(),2);
        user1.acceptMoney(100);
        assertEquals("2020100", user1.getNationalId());
        assertEquals("0202020", user1.getPassword());
        assertEquals("ak", user1.getUsername());
        assertEquals(1600, user1.getAccount().getBalance(), 2);
    }
```

```
Tests passed: 4 of 4 tests – 22 ms
UserIntegrationTest (com.example.testing 22 ms
    getNationalId_Password_Username_A 21 ms
    getNationalId_Password_Username_Ac 0 ms
    getNationalId_Password_UsernameTes 0 ms
    getNationalId_PasswordTest 1 ms

C:\Users\mosta\.jdks\openjdk-18.0.1.1\bin\java.exe ...
--UserIntegrationTest-- test case started
--UserIntegrationTest-- test case ended
--UserIntegrationTest-- test case started
--UserIntegrationTest-- test case ended
--UserIntegrationTest-- test case started
--UserIntegrationTest-- test case ended
--UserIntegrationTest-- test case started
--UserIntegrationTest-- test case ended
--UserIntegrationTest-- Class Testing ended

Process finished with exit code 0
```

Classes Integration Testing

Testing these classes together

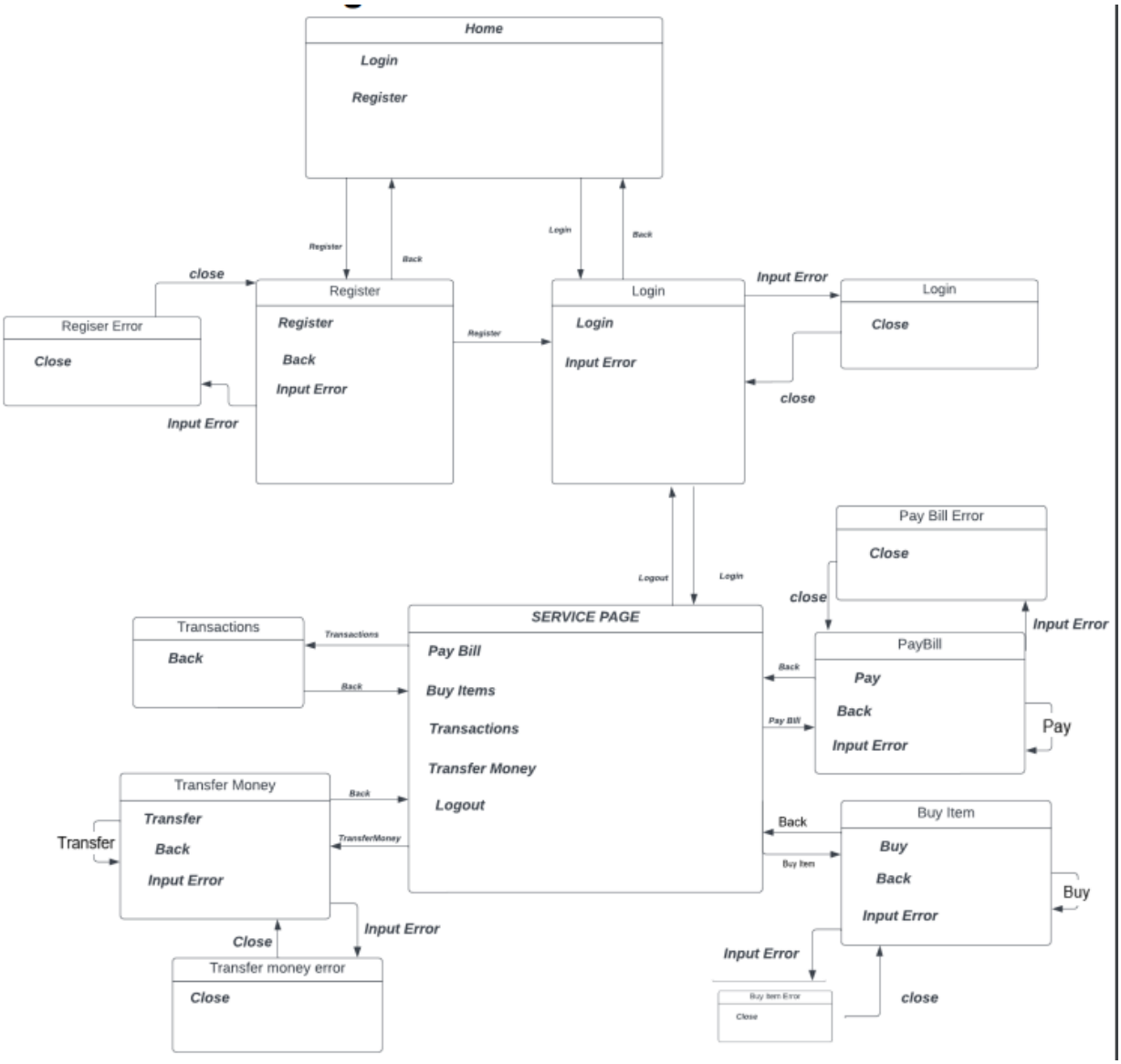
- Item
- Bill
- User
- Account

```java
public class ClassesIntegrationTest {
    @Test
    public void Item_BillTest(){
        Bill bill = new Bill(100);
        assertEquals(100, bill.getPrice(),2);

        Item item = new Item(250);
        assertEquals(250, item.getPrice(),2);
    }

    @Test
    public void Item_Bill_UserTest(){
        Bill bill = new Bill(100);
        assertEquals(100, bill.getPrice(),2);

        Item item = new Item(250);
        assertEquals(250, item.getPrice(),2);

        User user1 = new User("Abdallah", "Elkhalafawy", "ak", "01020011111",1500 ,"0202020", "2020100");
        assertEquals(1500 ,user1.getAccount().getBalance(),2);
        user1.acceptMoney(100);
        assertEquals("2020100", user1.getNationalId());
        assertEquals("0202020", user1.getPassword());
        assertEquals("ak", user1.getUsername());
        assertEquals(1600, user1.getAccount().getBalance(), 2);
    }
```

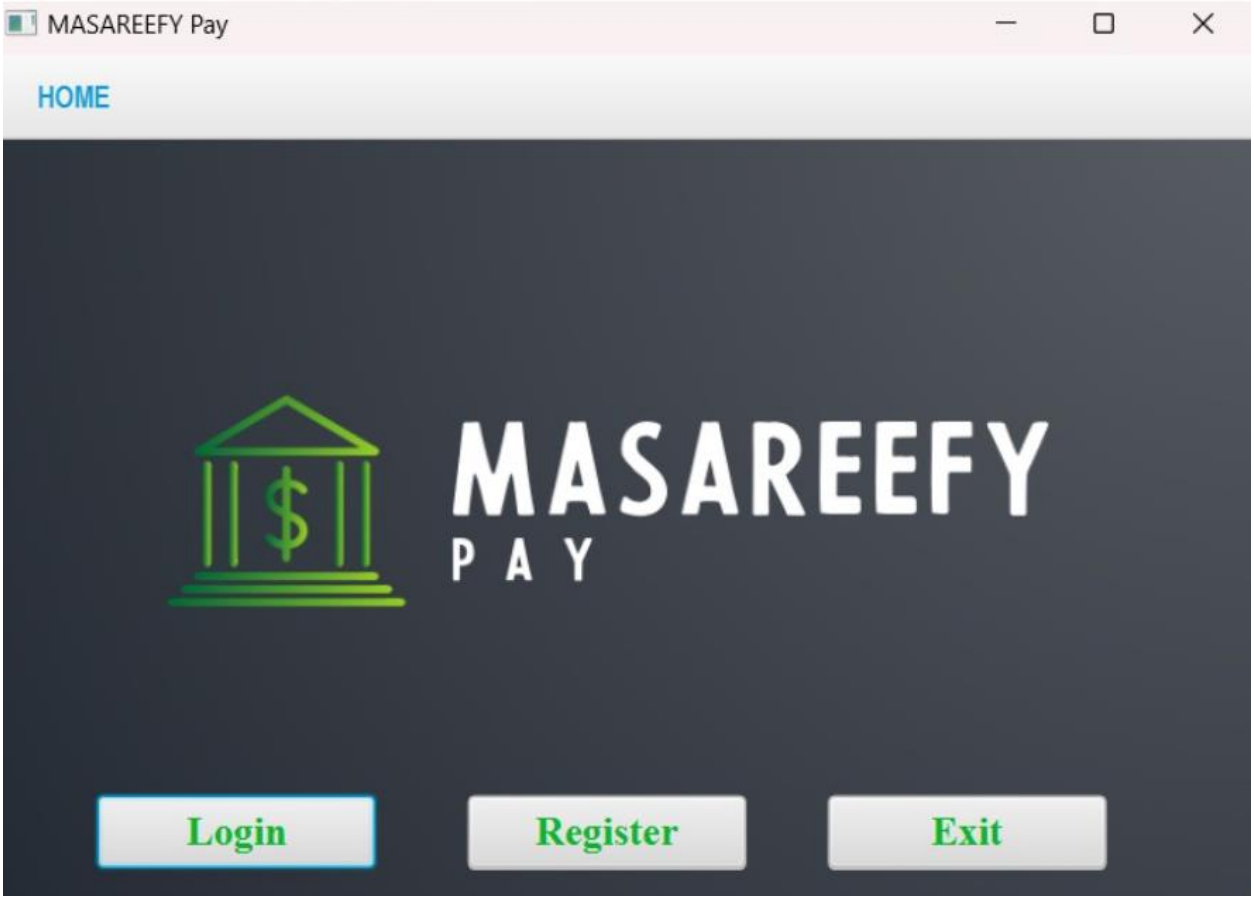
```java
@Test
public void Item_Bill_User_AccountTest() throws NoSuchMethodException, InvocationTargetException, IllegalAccessException {
```

```
Run: ClassesIntegrationTest
Tests passed: 3 of 3 tests – 35 ms
ClassesIntegrationTest (com.example.test 35 ms
    Item_Bill_UserTest 18 ms
    Item_Bill_User_AccountTest 16 ms
    Item_BillTest 1 ms

C:\Users\mosta\.jdks\openjdk-18.0.1.1\bin\java.exe ...
--ClassesIntegrationTest-- test case started
--ClassesIntegrationTest-- test case ended
--ClassesIntegrationTest-- test case started
--ClassesIntegrationTest-- test case ended
--ClassesIntegrationTest-- test case started
--ClassesIntegrationTest-- test case ended
--ClassesIntegrationTest-- Class Testing ended

Process finished with exit code 0
```

GUI State Diagram



Gui State Testing (Manual)

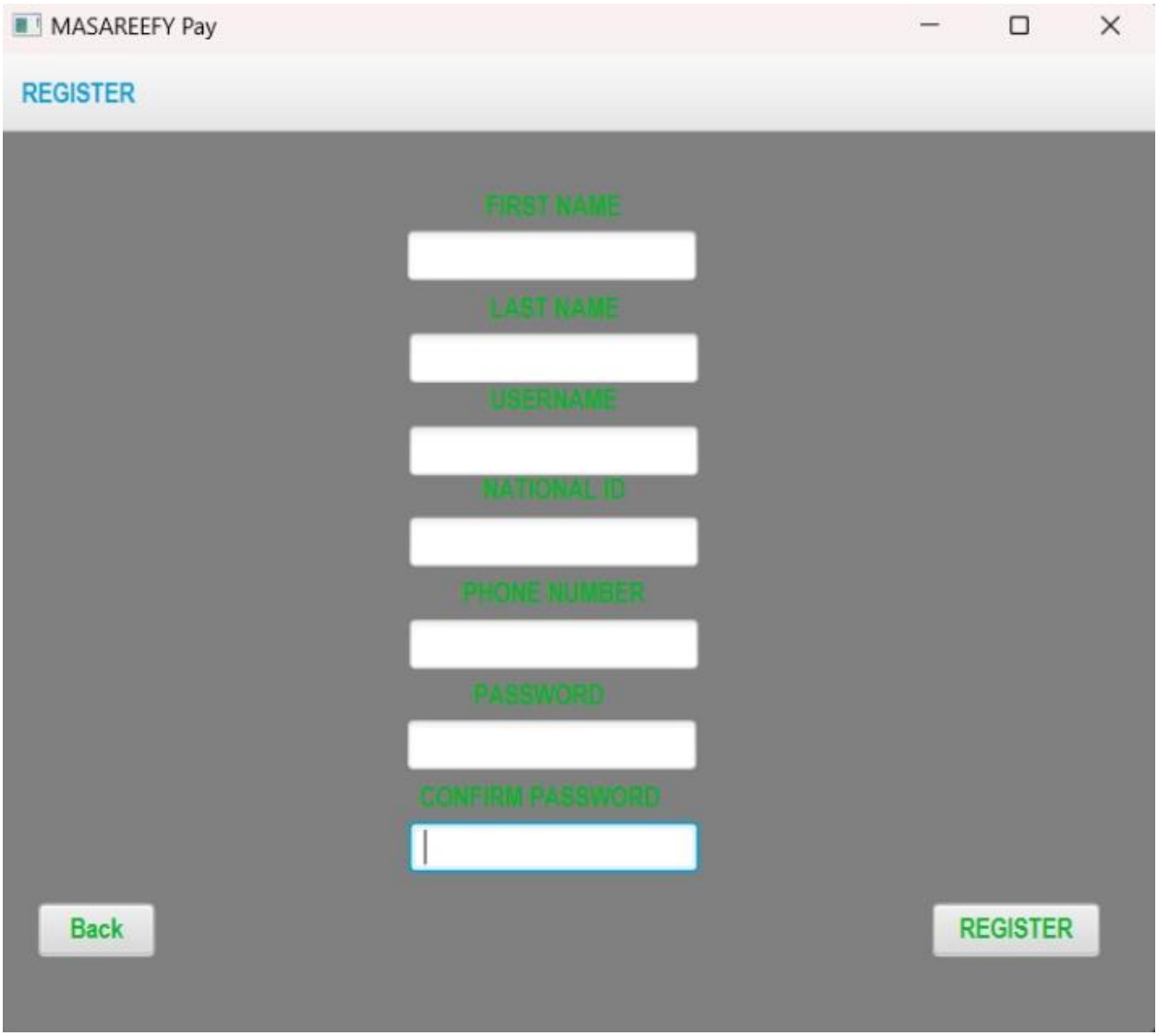


Home : Login: Goes to Login State

Home : Register: Goes to Register State

Home : exit : exit home state

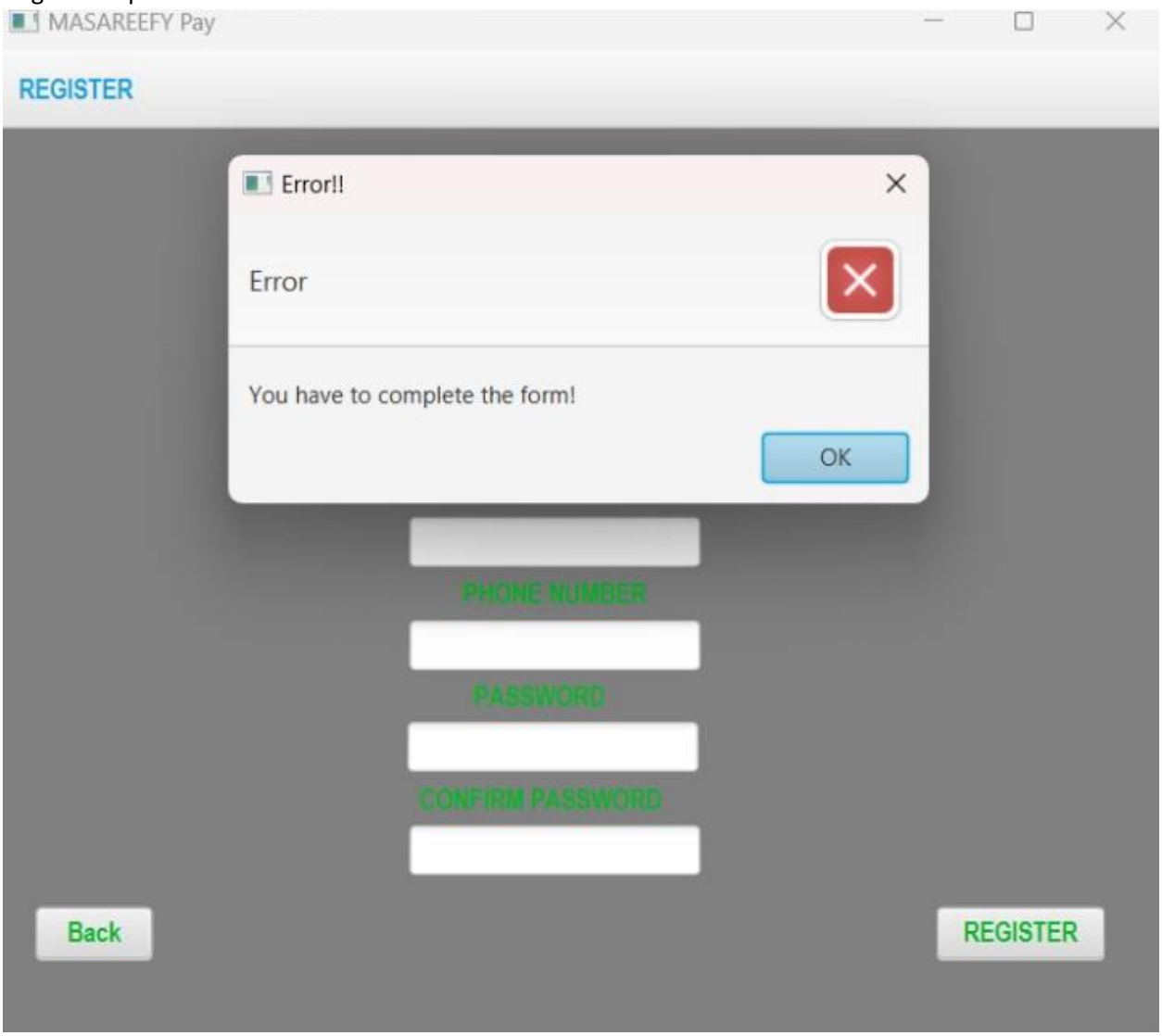
Register State



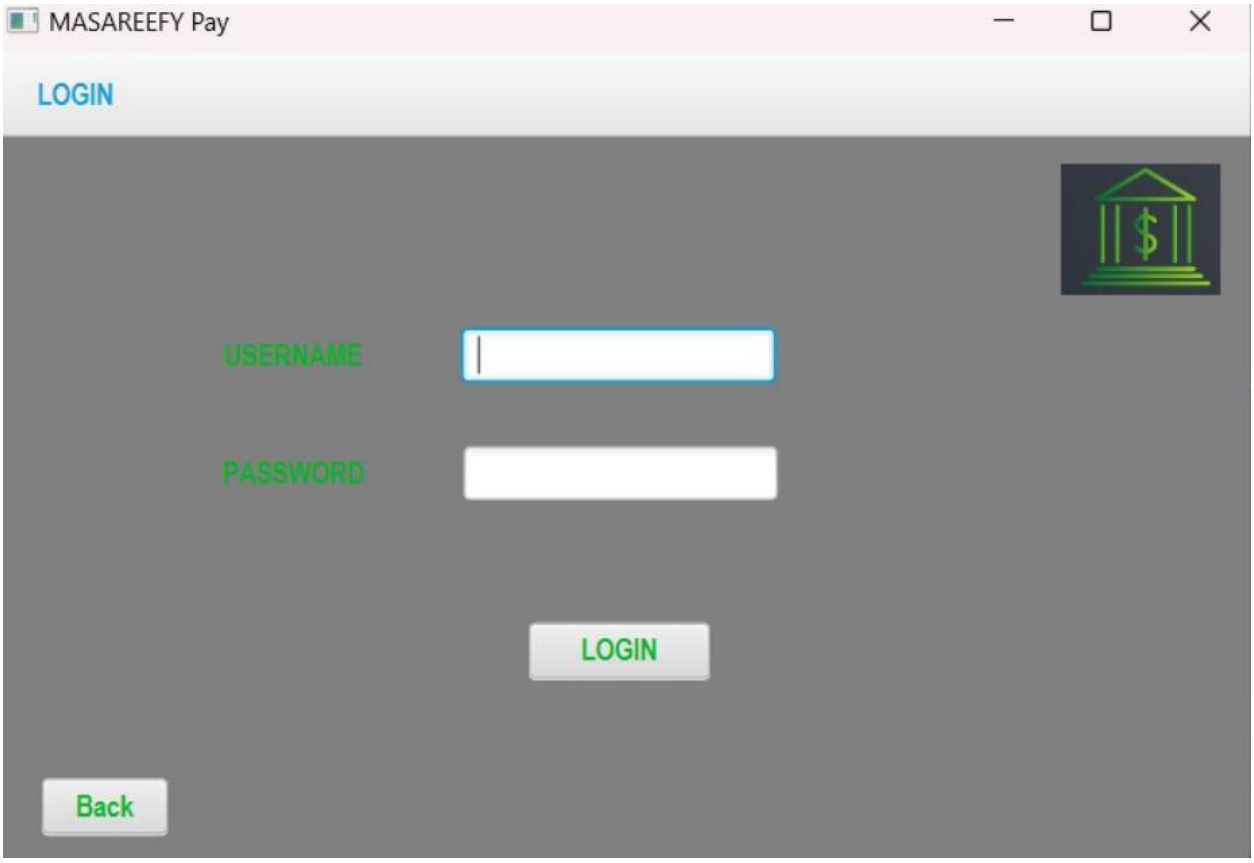
Register :Register: Goes to Login State

Register :Back: Goes to Home State

Register :Input Error:

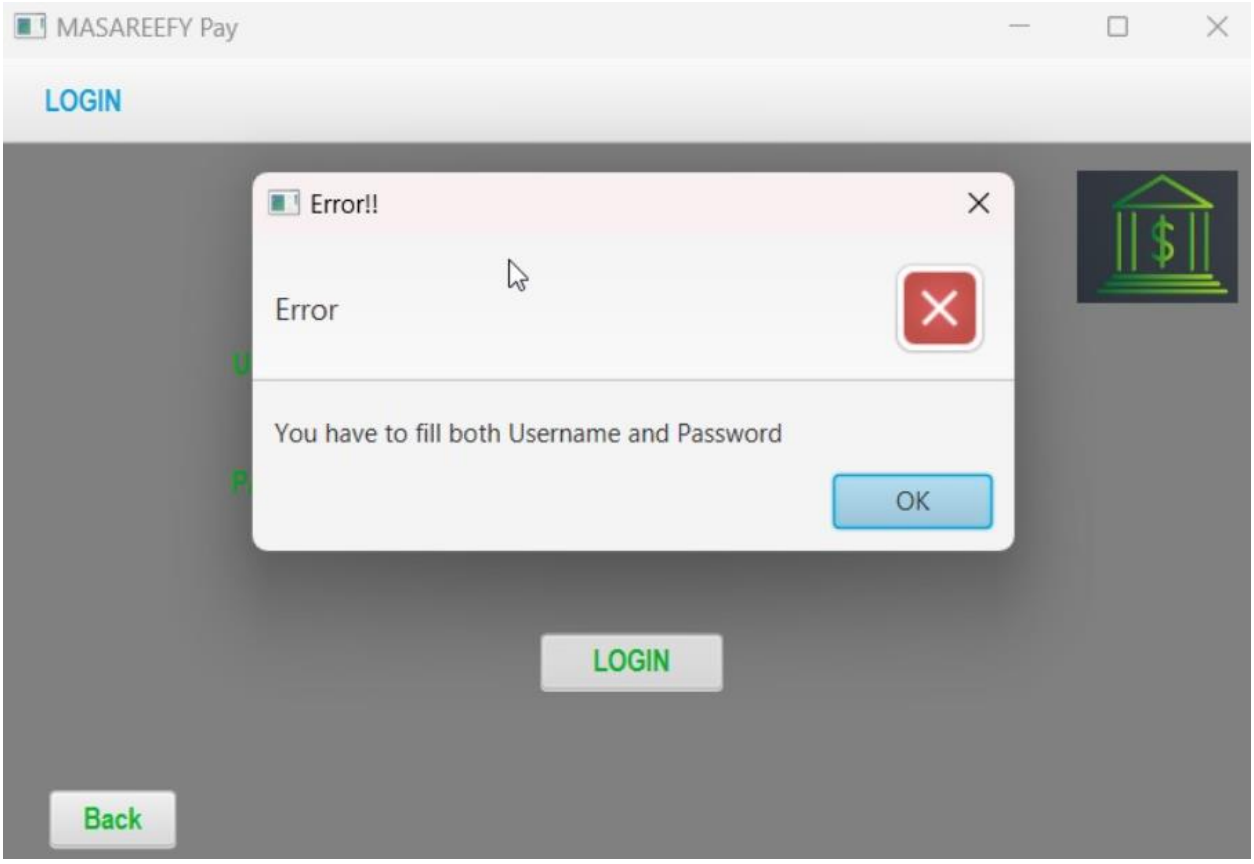


Login state

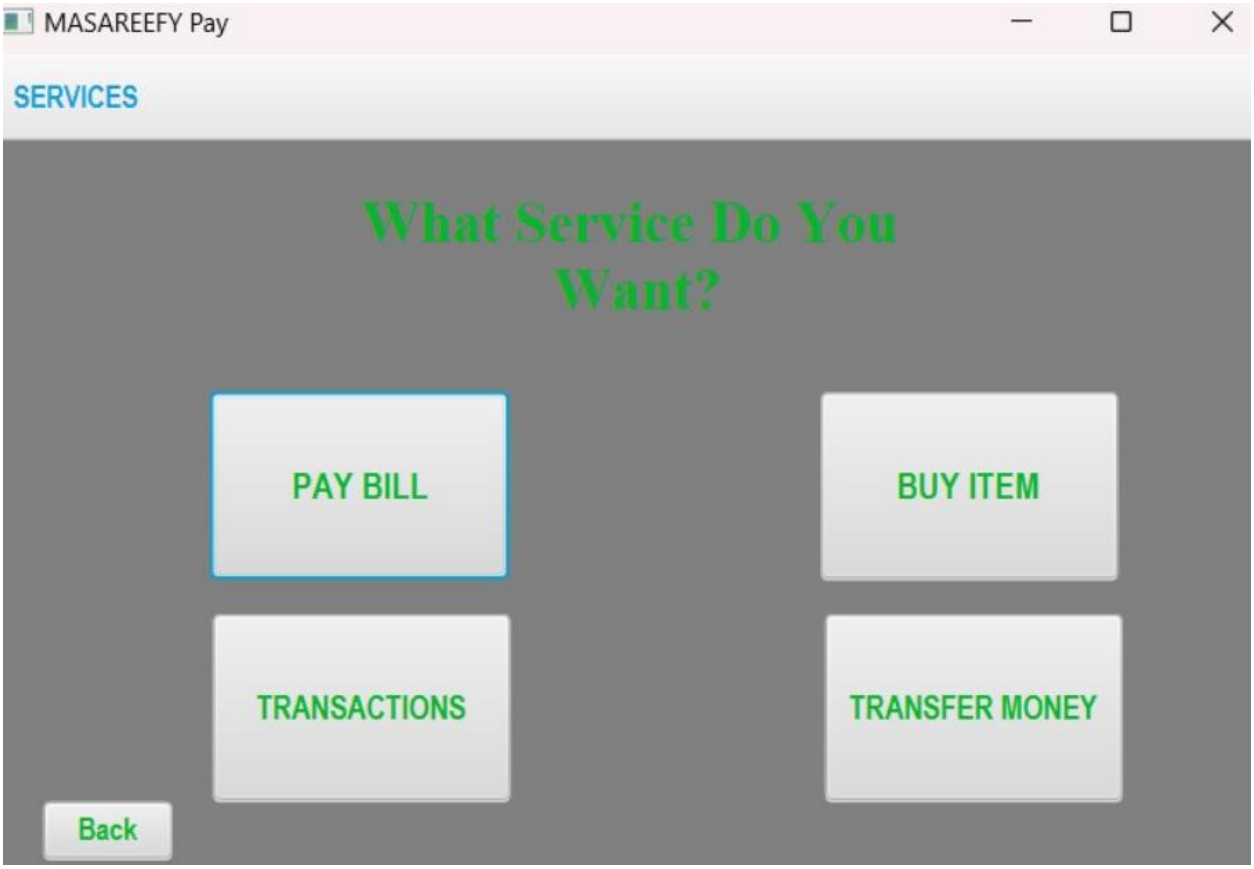


Login :Login: Goes to Service Page State

Login:Input Error: If Field Empty / Wrong (Username-Password)



Service Page State



Serivce Page : Pay Bill: Goes to Pay Bill State

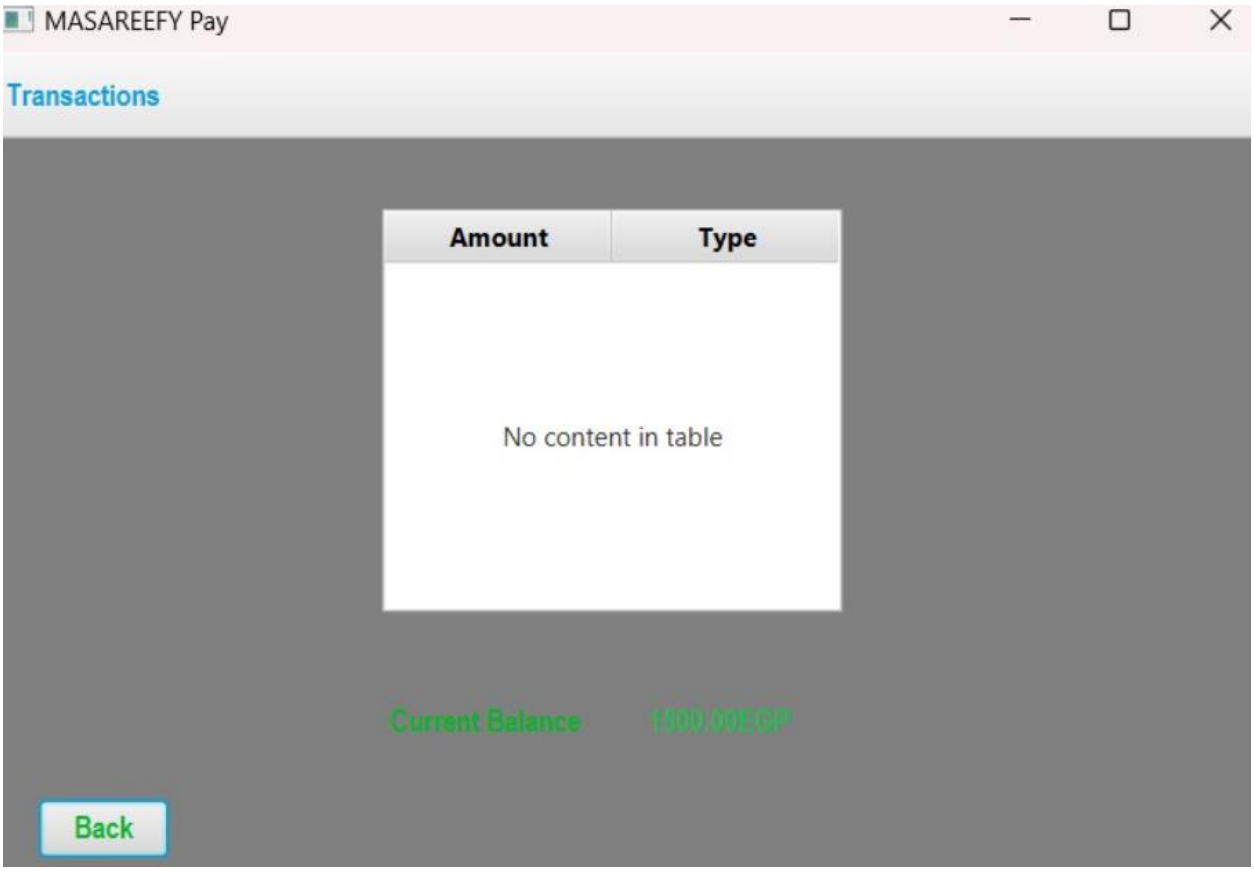
Serivce Page : Buy Items: Goes to Buy Items State

Serivce Page : Transactions: Goes to Transactions State

Serivce Page : Transfer Money: Goes to Transfer Money State

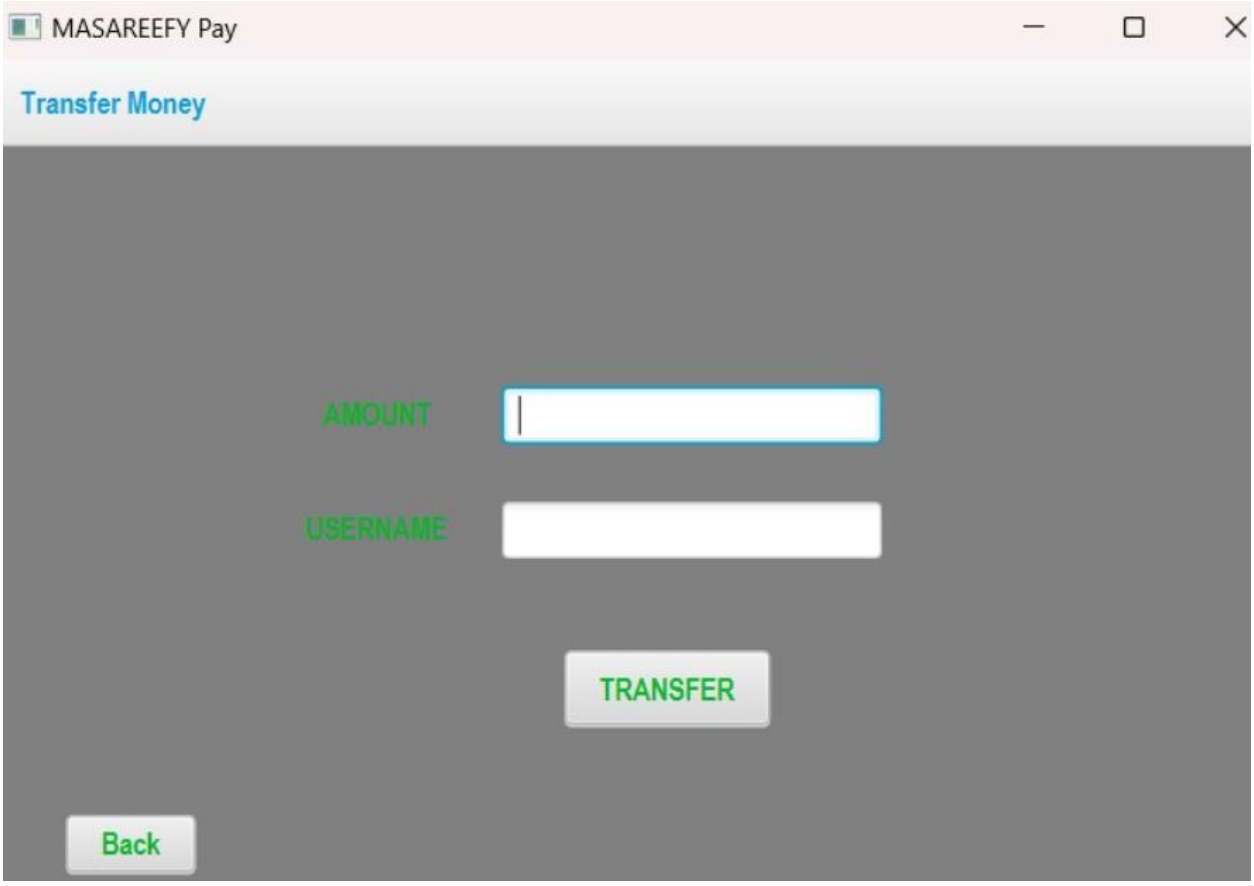
Serivce Page : back: Goest to Home State

Transactions State



Transaction :Back: Goes to Service Page State

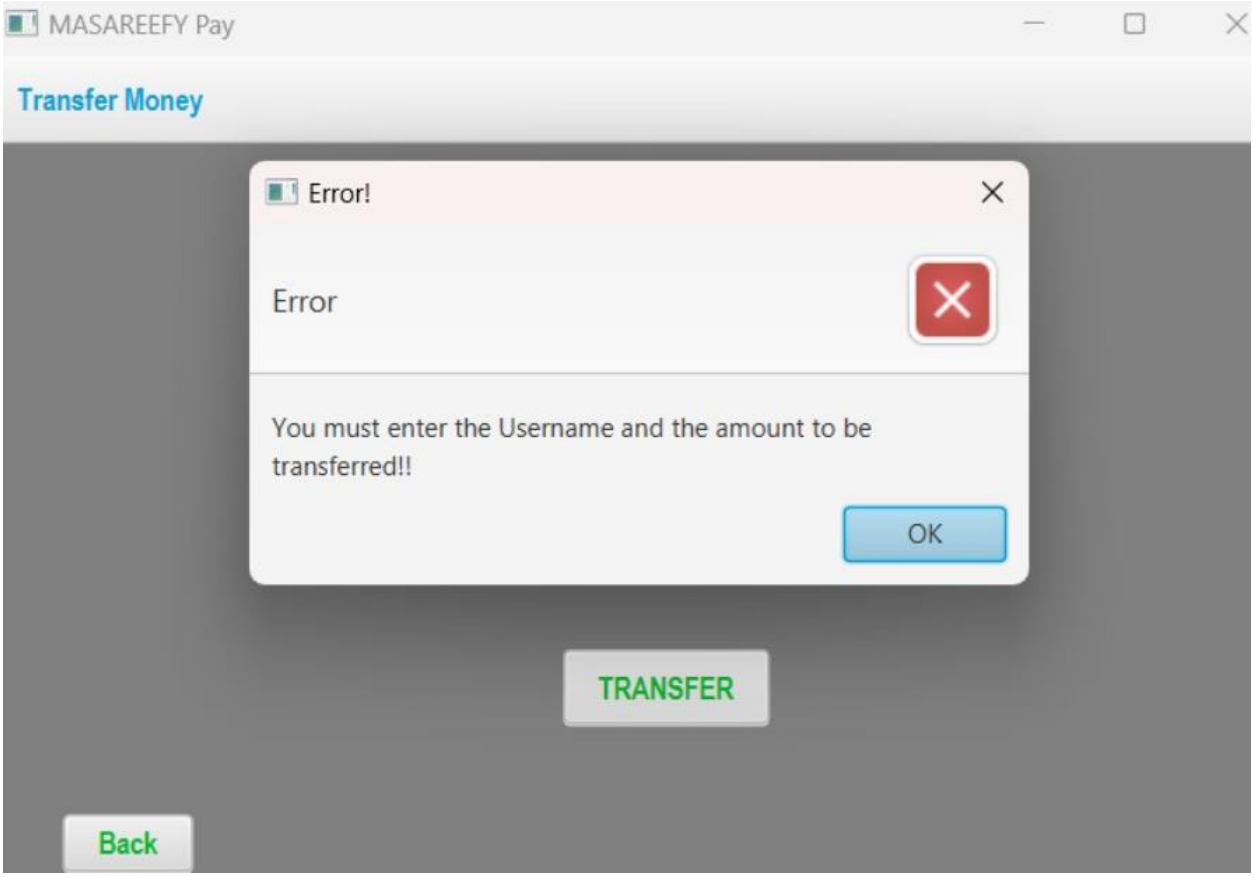
Transfer Money State



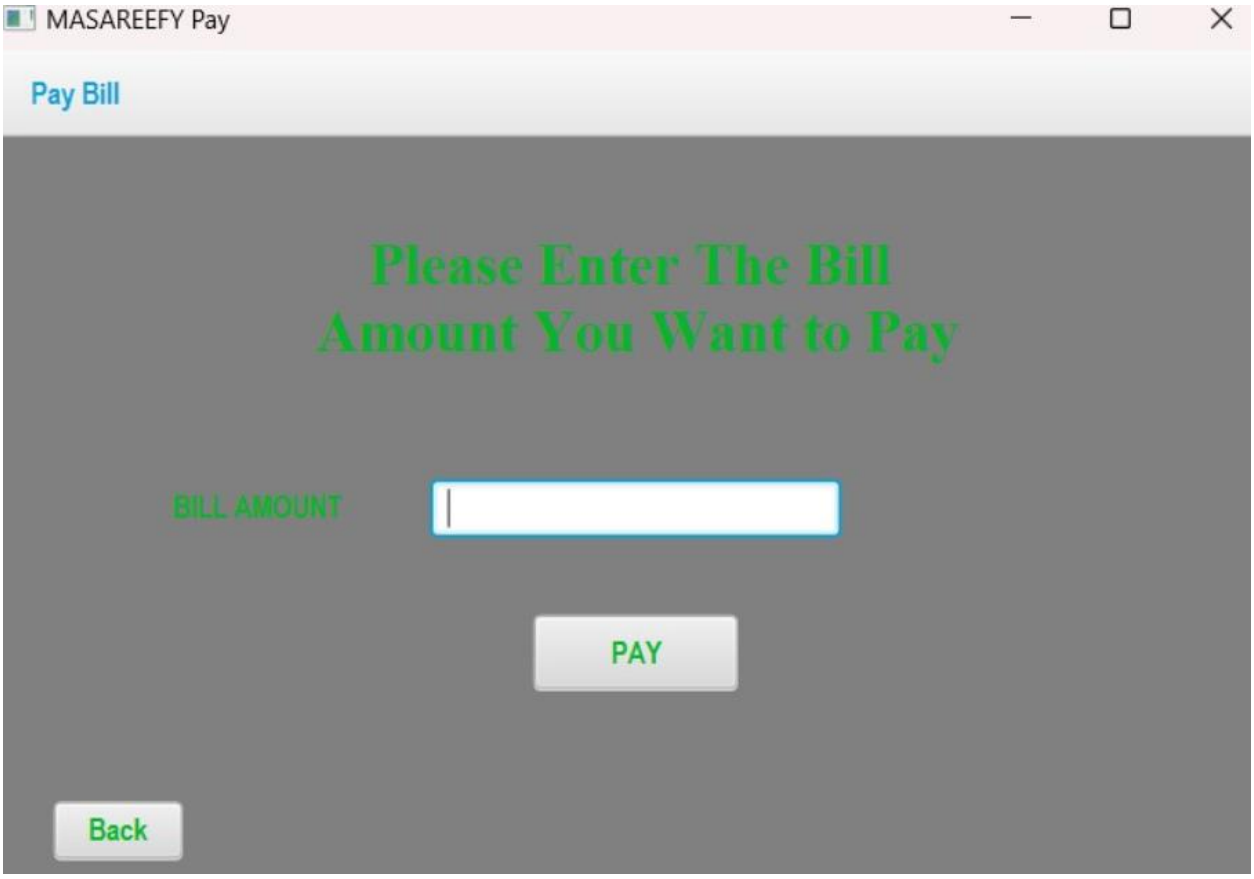
TransferMoney :Transfer: Does Transaction if Balance is enough and stays in the same state Else goes to Input Error State

TransferMoney :Back: Goes to Service Page State

TransferMoney: Input Error: If Balance is not enough for amount or username incorrect to empty



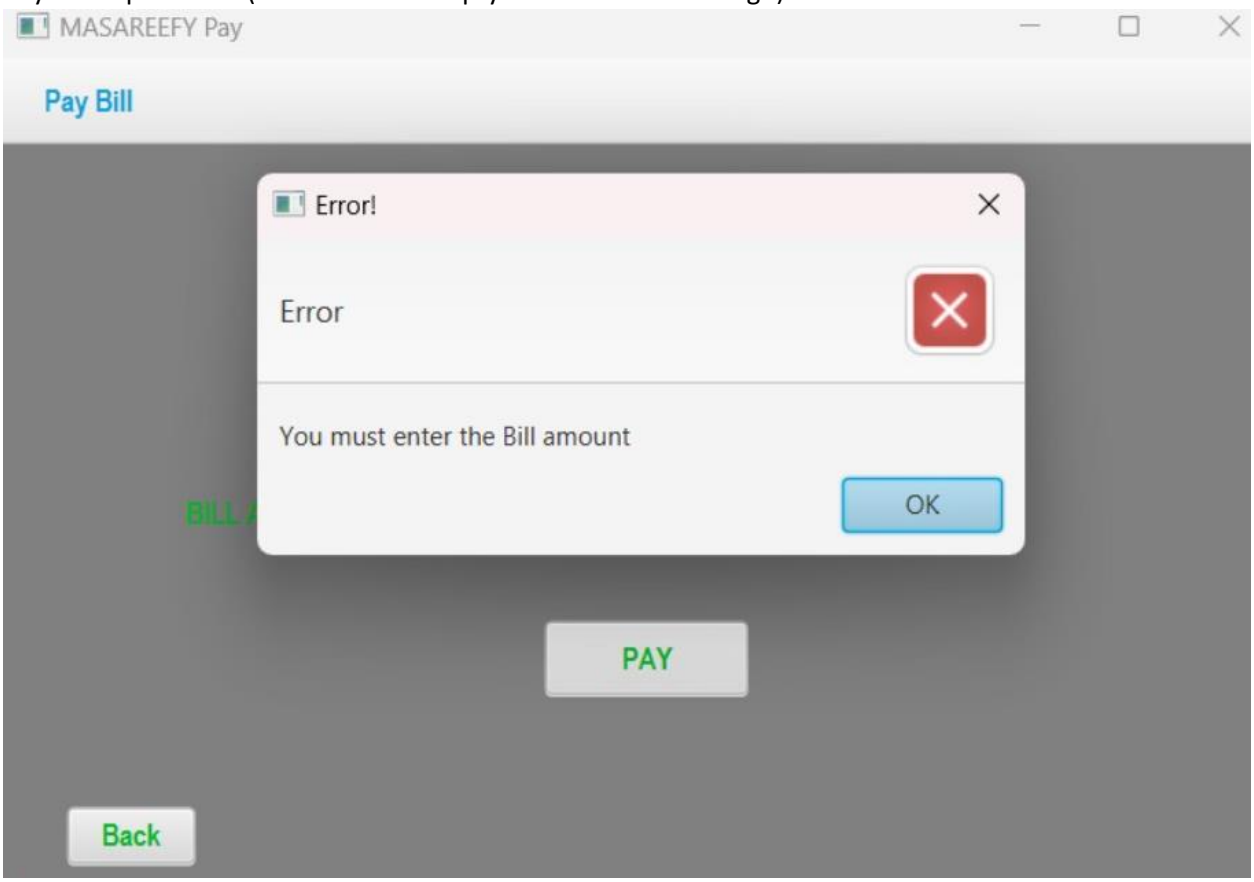
Pay Bill State



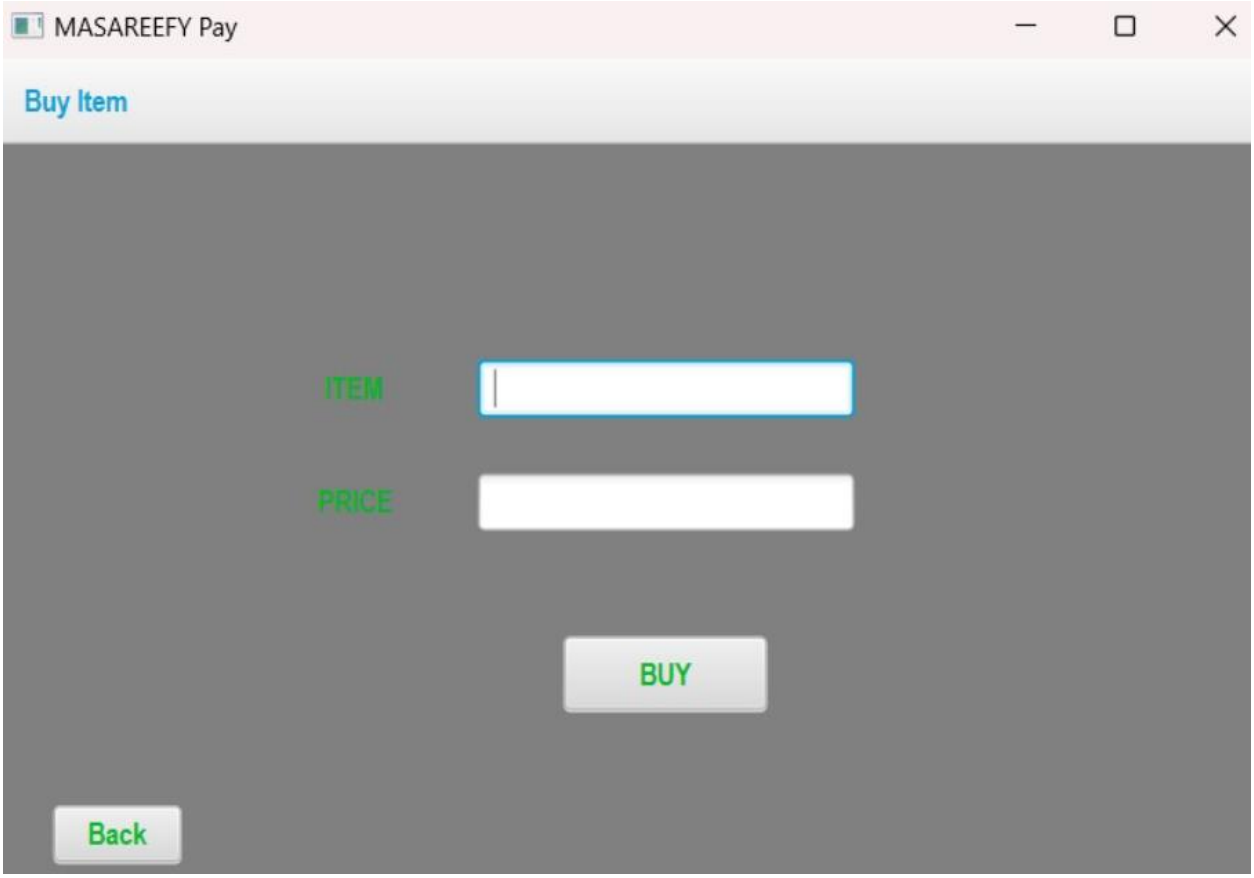
Pay Bill :Pay: if Balance>=Bill, Pays bill and stays in the same state else goes to Input Error State

Pay Bill :Back: Goes to Service Page State

Pay Bill :Input Error: (if Bill Amount Empty or Balance not enough)

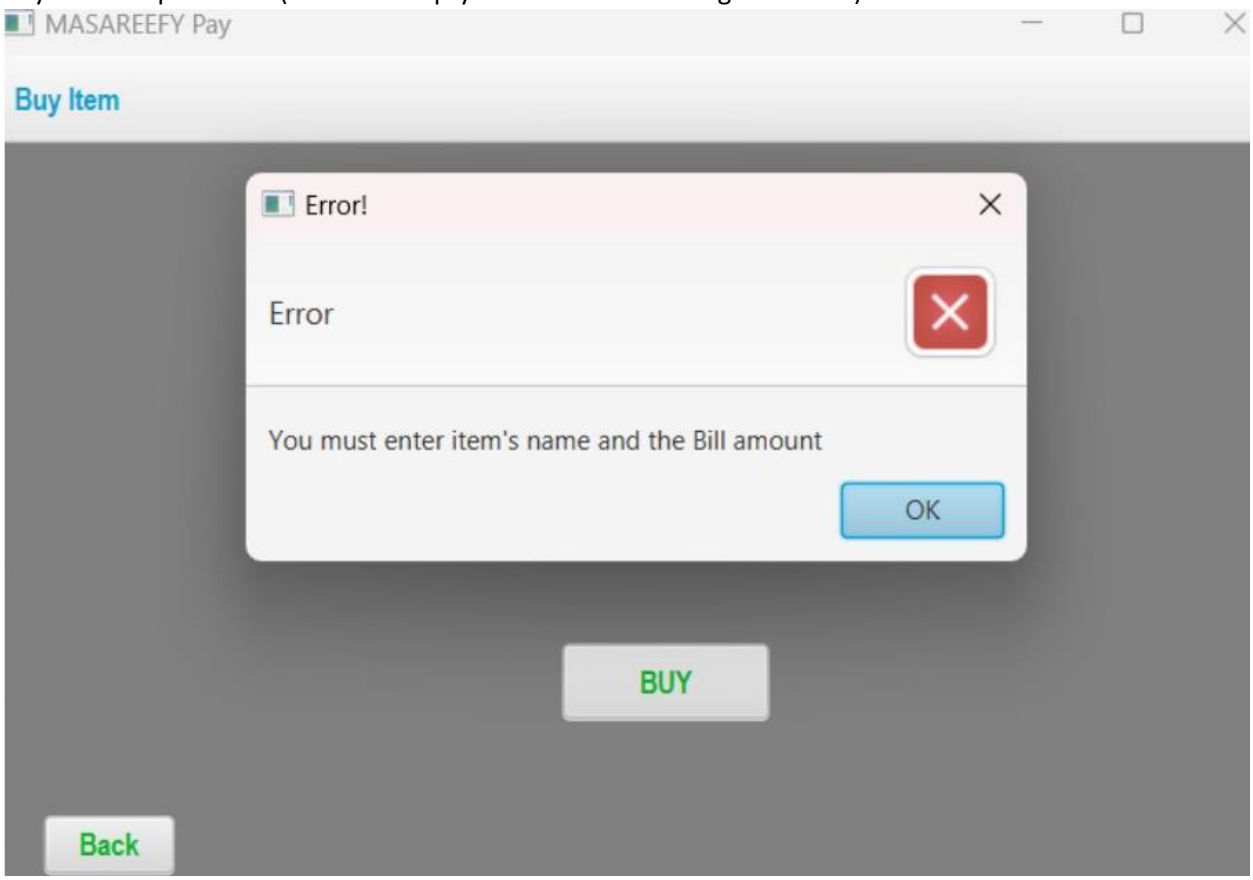


Buy Items State



Buy Items :Buy: if Balance>=Price deducts money, stays in the same state else goes to Input Error State
Buy Items :Back: Goest to Service Page State

Buy Items :Input Error: (If field is empty or Balance not enough for Price)



Performance / Load Testing using JMeter (1000 Users Test) Ramp up time 3 seconds

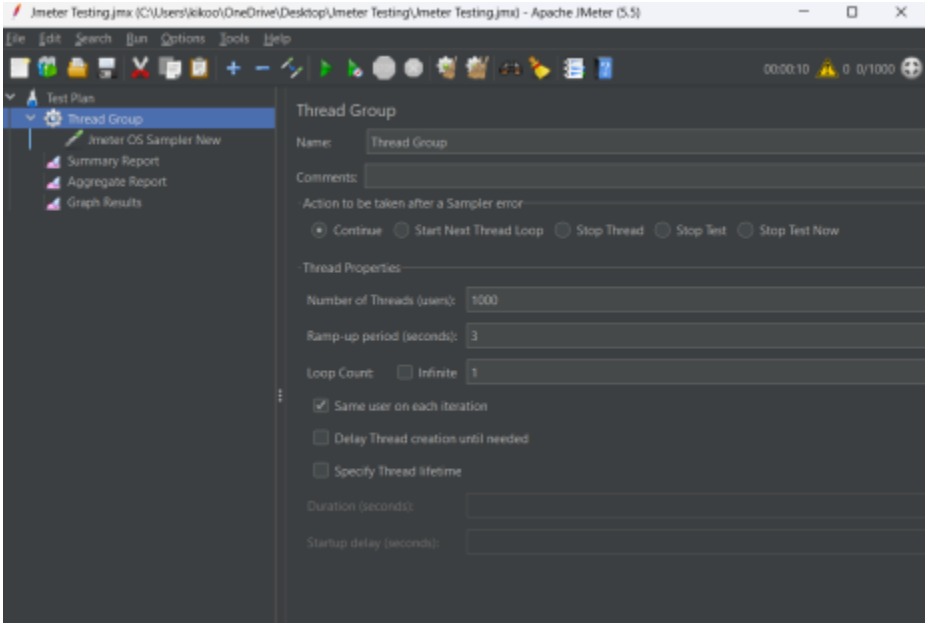
In JMeter, when you configure multiple threads and a ramp-up period, it does create separate instances of the code example for each thread. Each thread runs independently, simulating concurrent users accessing your banking system. While it's true that JMeter is executing multiple instances of the code example on your computer, the purpose of load testing is to assess the performance and scalability of your banking system under realistic usage scenarios. By simulating concurrent users with multiple threads, JMeter puts a load on your system and measures its response. It helps you identify potential bottlenecks.

JMeter was used OS Process Sampler to test multiple Instances of Online Banking System.

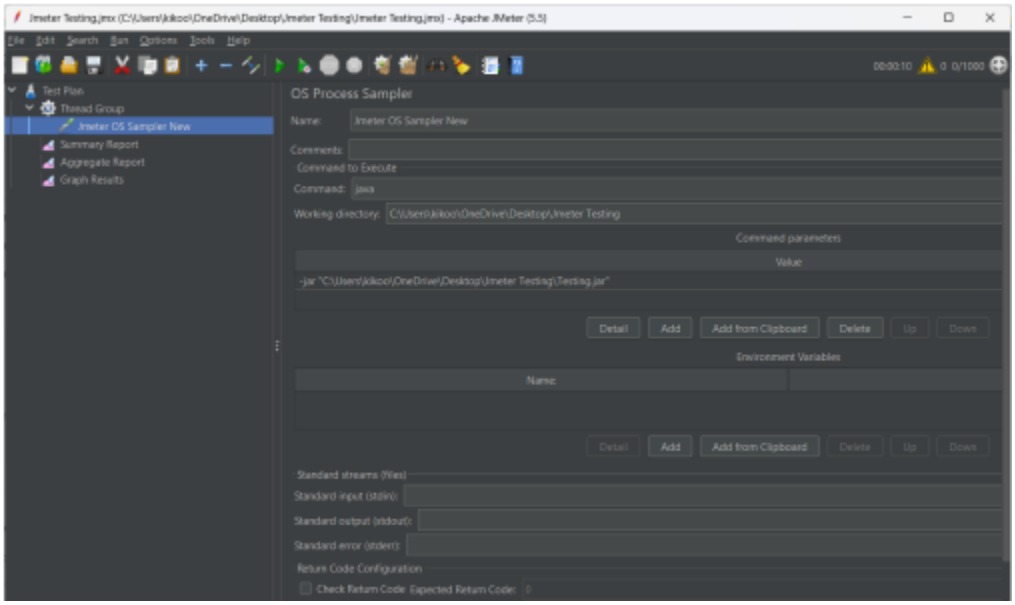
JMeter Test Example Code

```
public class BankingSystemCLI {
  public static void main(String[] args) {
      User user = new User("John", "Doe", "johndoe", "1234567890",
"password", "123456789");
      double amount = 500.0;
      // Perform deposit
      user.getAccount().deposit(amount);
      // Perform withdrawal
      double withdrawalAmount = 200.0;
      user.getAccount().withdraw(withdrawalAmount);
      // Transfer money to another user
      User recipientUser = new User("Jane", "Smith", "janesmith",
"9876543210", "password", "987654321");
      double transferAmount = 300.0;
      user.getAccount().transferMoney(recipientUser, transferAmount);
      // Get account balance
      double balance = user.getAccount().getBalance();
      assertEquals(1000.00,balance,2);
  }
```

Thread Settings / Ramp up time

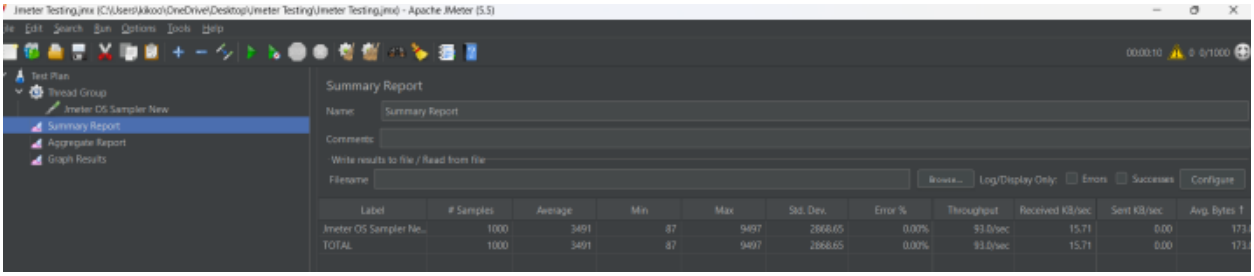


Jmeter Os Sampler Settings

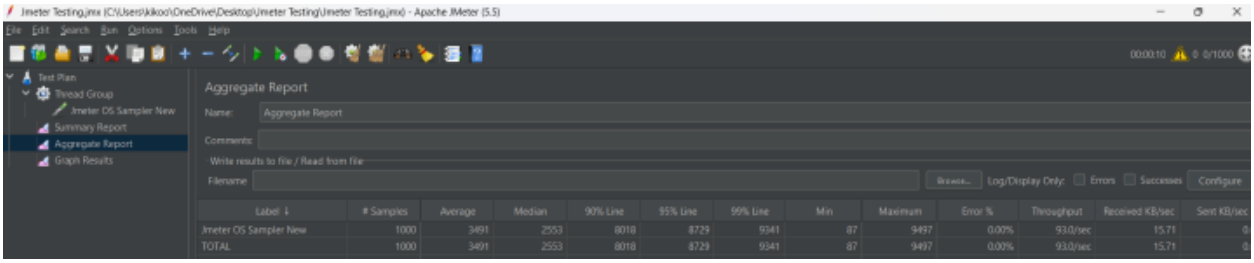


Load Testing Reports / Graphs

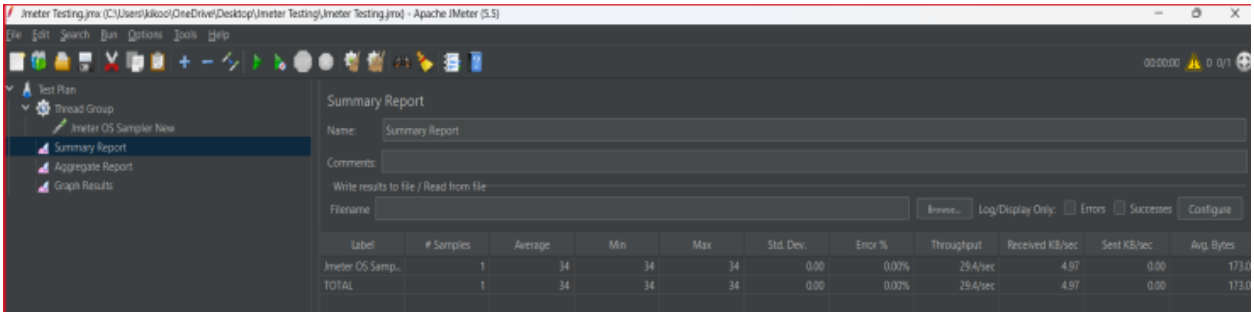
Summary Report



Aggregate



Performance



Report Graph