



Software Testing, Validation and Verification Online Banking System

Mohamed Abdelaty Mohamed	20P9634
Mohaned Hesham Nasr	20P2204
Abdallah Mohamed Elkhalfawy	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.

- Account Test Cases.....	
getBalance Implementation	
Test Cases.....	
checkAmount Implementation.....	
Test Cases.....	
Withdraw Implementation.....	
Test Case	
withdrawTransaction Implementation.....	
Test Case	
payFor implementation	
Test Cases.....	
Deposit Implementation.....	
DepositTransactions Implementation.....	
Test Case	
getStatements Implementation.....	
Test Case.....	
transferMoney implementation.....	
Test Case.....	
User Test Cases.....	
acceptMoney Implementation	
Test case	
Getters test cases	
Bill Test Cases	
Item Test Cases.....	
The First Test Suite.....	
How the Test Cases were Used to Debug Code	
Example 1: Making a new Account (Constructor) using a negative value.	
Example 2: In Deposit / Pay for Item when the number is negative it shouldn't be accepted	
Integration Testing.....	
Account Integration Testing.....	
Testing these functions together.....	
User Integration Testing	
Testing these functions together.....	
Classes Integration Testing.....	
Testing these classes together.....	

GUI State Diagram	
Gui State Testing (Manual).....	
Home State	
Home :Login: Goes to Login State	
Home :Register: Goes to Register 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	
Servive Page : Pay Bill: Goes to Pay Bill State	
Servive Page : Buy Items: Goes to Buy Items State.....	
Servive Page : Transactions: Goes to Transactions State.....	
Servive Page : Transfer Money: Goes to Transfer Money State.....	
Servive 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 seconds41 JMeter was used OS Process Sampler to test multiple Instances of Online Banking System. JMeter Test Example Code	
Thread Settings / Ramp up time	
Jmeter Os Sampler Settings.....	
Load Testing Reports / Graphs	
Summary Report Aggregate.....	

Performance
Report Graph.....

PROJECT COMPONENT :

-Account test cases

getBalance Implementation

```
67     public double getBalance(){  
68         return balance;  
69     }
```

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

```
44     private boolean checkAmount(double inputAmount){
45         if(inputAmount <= 0)
46             return false;
47         return balance >= inputAmount;
48     }
```

Test cases

```
@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);
}
```

```

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

```

50     private boolean withdraw(double inputAmount) {
51         if (checkAmount(inputAmount)) {
52             balance -= inputAmount;
53             withdrawTransaction(inputAmount);
54             return true;
55         } else{
56             return false;
57         }
58     }

```

Test cases

```
@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);
}
```

```

        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);
    }
    @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

```
22     private void withdrawTransaction(double amount){
23         if(transactions[9] == null){
24             transactions[transactionIndex] = "W" + decfor.format(amount);
25             transactionIndex++;
26         }else{
27             for(int i = 0; i < 9; i++){
28                 transactions[i] = transactions[i+1];
29                 transactions[9] = "W" + amount;
30             }
31         }
```

Test cases

```
@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() {
```

```
@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

```
60     public void deposit(double inputAmount) {  
61         if (inputAmount > 0) {  
62             balance += inputAmount;  
63             depositTransaction(inputAmount);  
64         }  
65     }
```

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

```
33 private void depositTransaction(double amount){
34     if(transactions[9] == null){
35         transactions[transactionIndex] = "D" + decfor.format(amount);
36         transactionIndex++;
37     }else{
38         for(int i = 0; i < 9; i++)
39             transactions[i] = transactions[i+1];
40         transactions[9] = "D" + amount;
41     }
42 }
```

Test cases

```
@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

```
94 public String[] getStatements(){
95     return transactions;
96 }
```

Test cases

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

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

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

```

@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

```

```

@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

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

Test cases

```
@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

```
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

```
@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

```
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

```

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

```

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

```

Deposit Function

```

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

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

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

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

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

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

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

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

```
Tests passed: 7 of 7 tests - 34 ms  
✓ AccountIntegrationTest (com.example.test: 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  
--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`


```

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\.jdk\openjdk-18.0.1\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

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

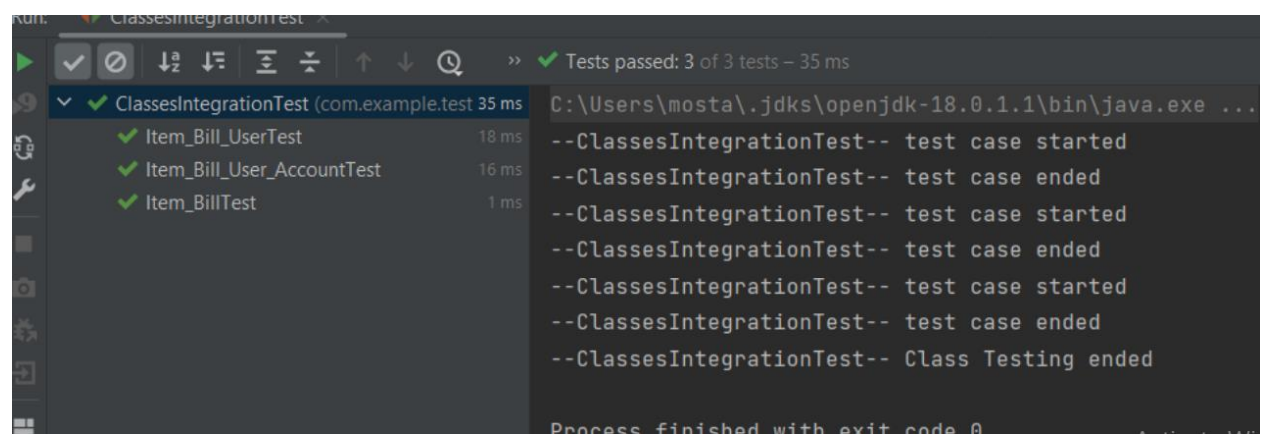
        Item item = new Item(250);
        assertEquals(250, item.getPrice(),2);
    }

    @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);
    }
}
```

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



run. ClassesIntegrationTest

Tests passed: 3 of 3 tests – 35 ms

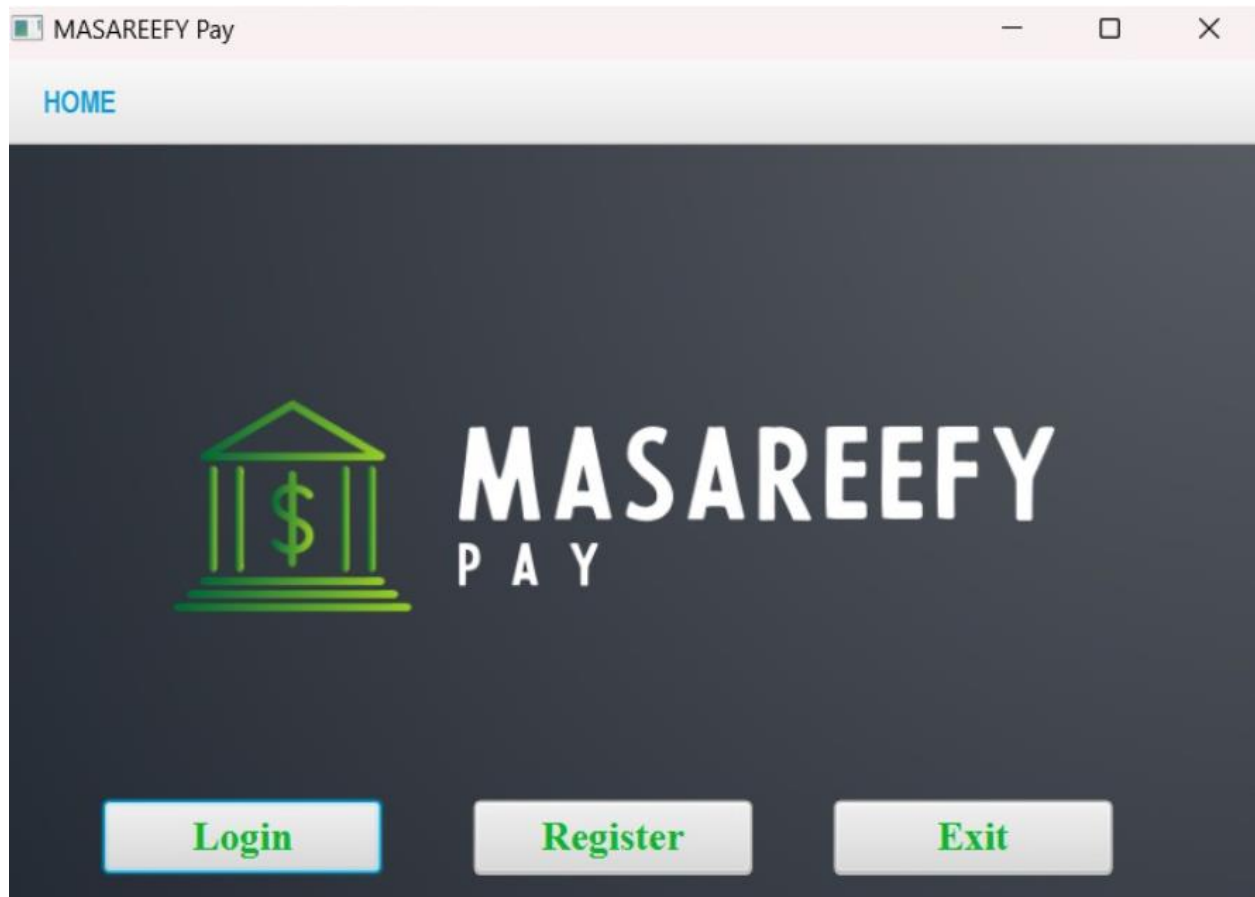
Test Case	Duration
Item_Bill_UserTest	18 ms
Item_Bill_User_AccountTest	16 ms
Item_BillTest	1 ms

Process finished with exit code 0

GUI State Diagram



Gui State Testing (Manual)

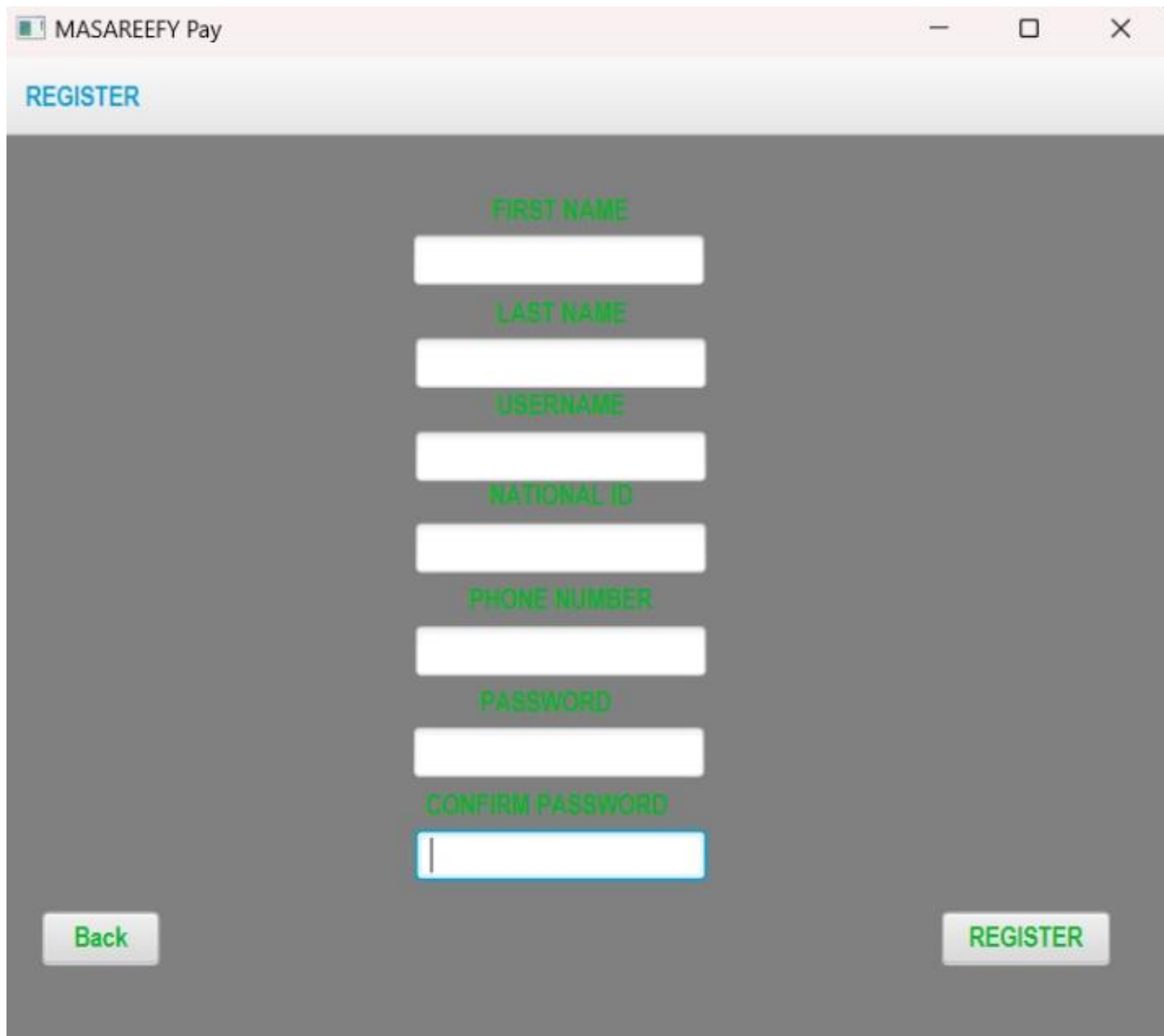


Home : Login: Goes to Login State

Home : Register: Goes to Register State

Home : exit : exit home state

Register State



The image shows a web application window titled "MASAREEFY Pay" with standard window controls (minimize, maximize, close). Below the title bar, the word "REGISTER" is displayed in blue. The main content area has a dark gray background and contains a vertical stack of seven white input fields. Each field is preceded by a green label: "FIRST NAME", "LAST NAME", "USERNAME", "NATIONAL ID", "PHONE NUMBER", "PASSWORD", and "CONFIRM PASSWORD". The "CONFIRM PASSWORD" field is currently active, indicated by a blue border and a vertical cursor. At the bottom left is a "Back" button and at the bottom right is a "REGISTER" button, both with green text on a light gray background.

MASAREEFY Pay

REGISTER

FIRST NAME

LAST NAME

USERNAME

NATIONAL ID

PHONE NUMBER

PASSWORD

CONFIRM PASSWORD

Back

REGISTER

Register :Register: Goes to Login State

Register :Back: Goes to Home State

Register :Input Error:

MASAREEFY Pay

REGISTER

Error!

Error

You have to complete the form!

OK

PHONE NUMBER

PASSWORD

CONFIRM PASSWORD


Back

REGISTER

Login state

MASAREEFY Pay

LOGIN



USERNAME

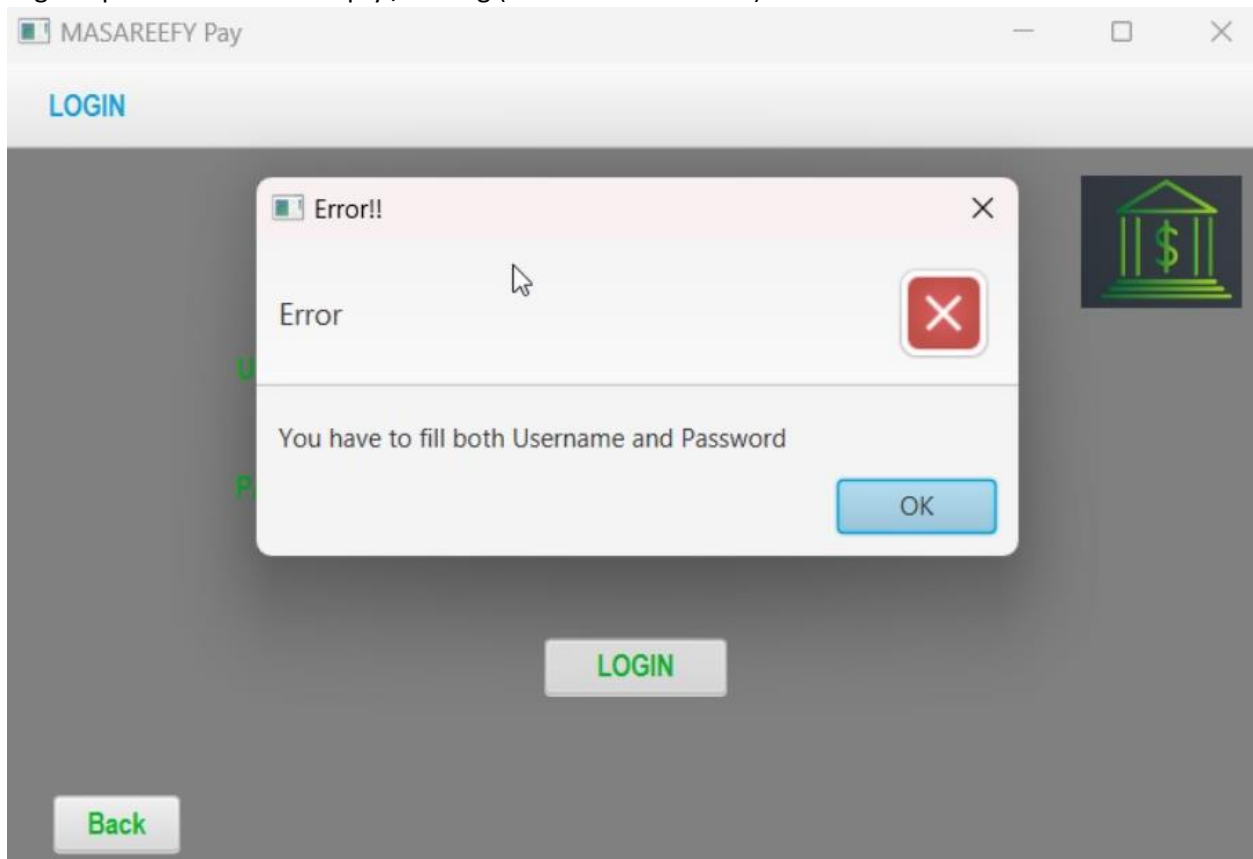
PASSWORD

LOGIN

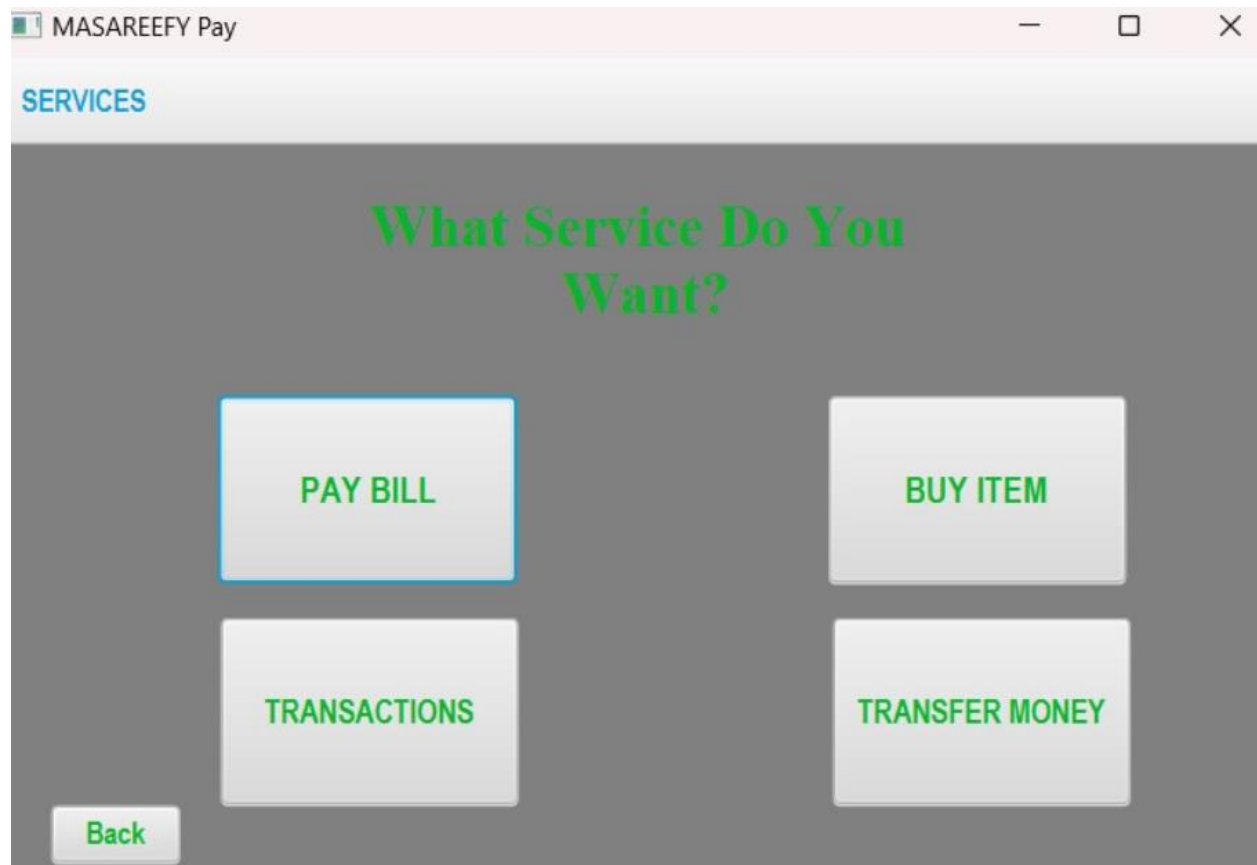
Back

Login :Login: Goes to Service Page State

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



Service Page State



Service Page : Pay Bill: Goes to Pay Bill State

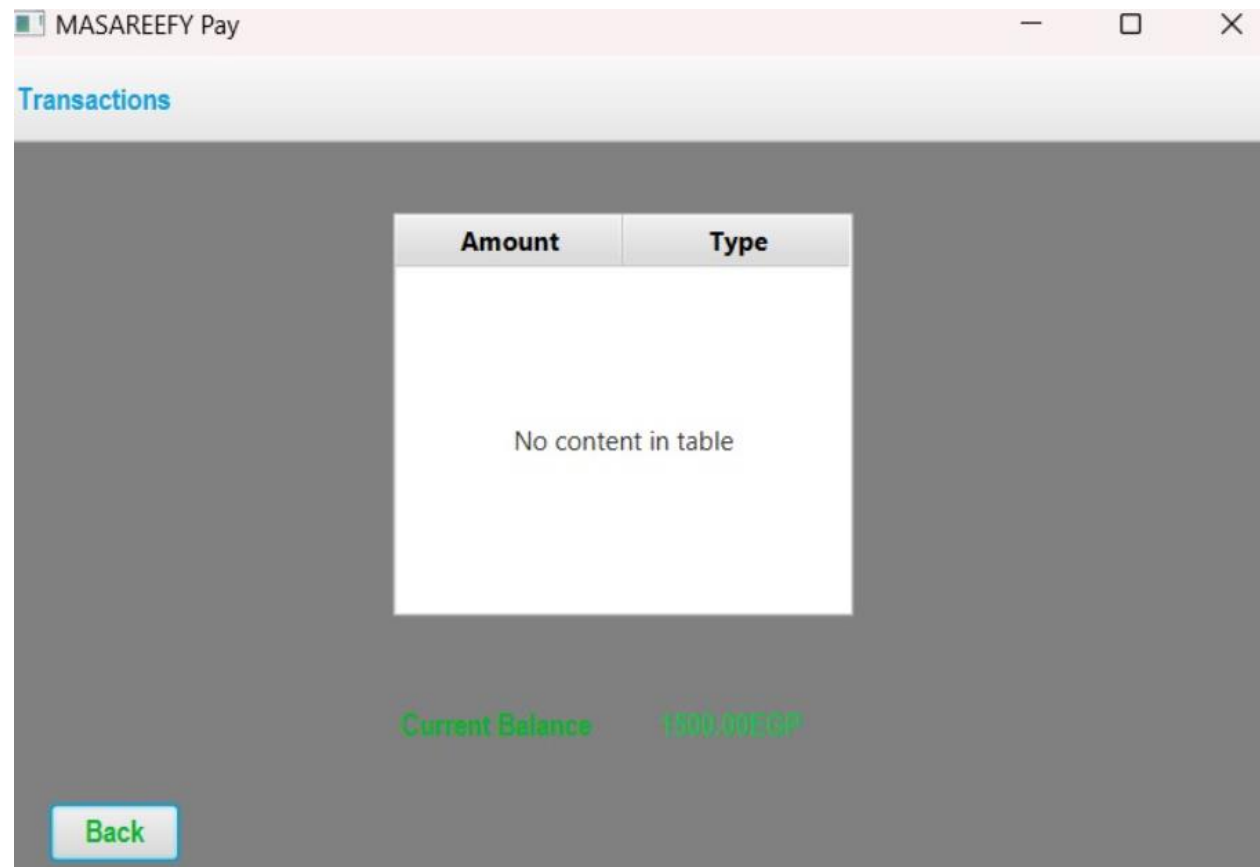
Service Page : Buy Items: Goes to Buy Items State

Service Page : Transactions: Goes to Transactions State

Service Page : Transfer Money: Goes to Transfer Money State

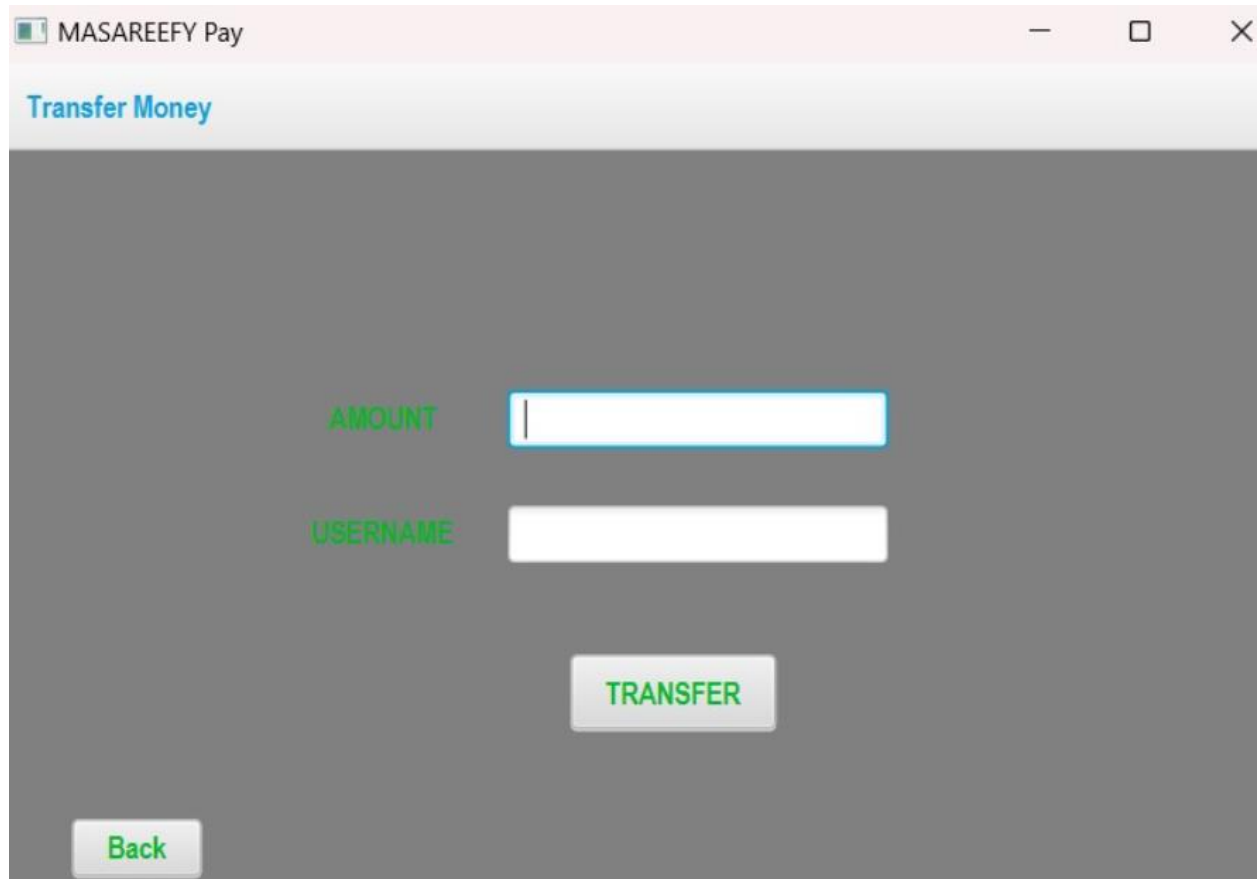
Service Page : back: Goest to Home State

Transactions State



Transaction :Back: Goes to Service Page State

Transfer Money State

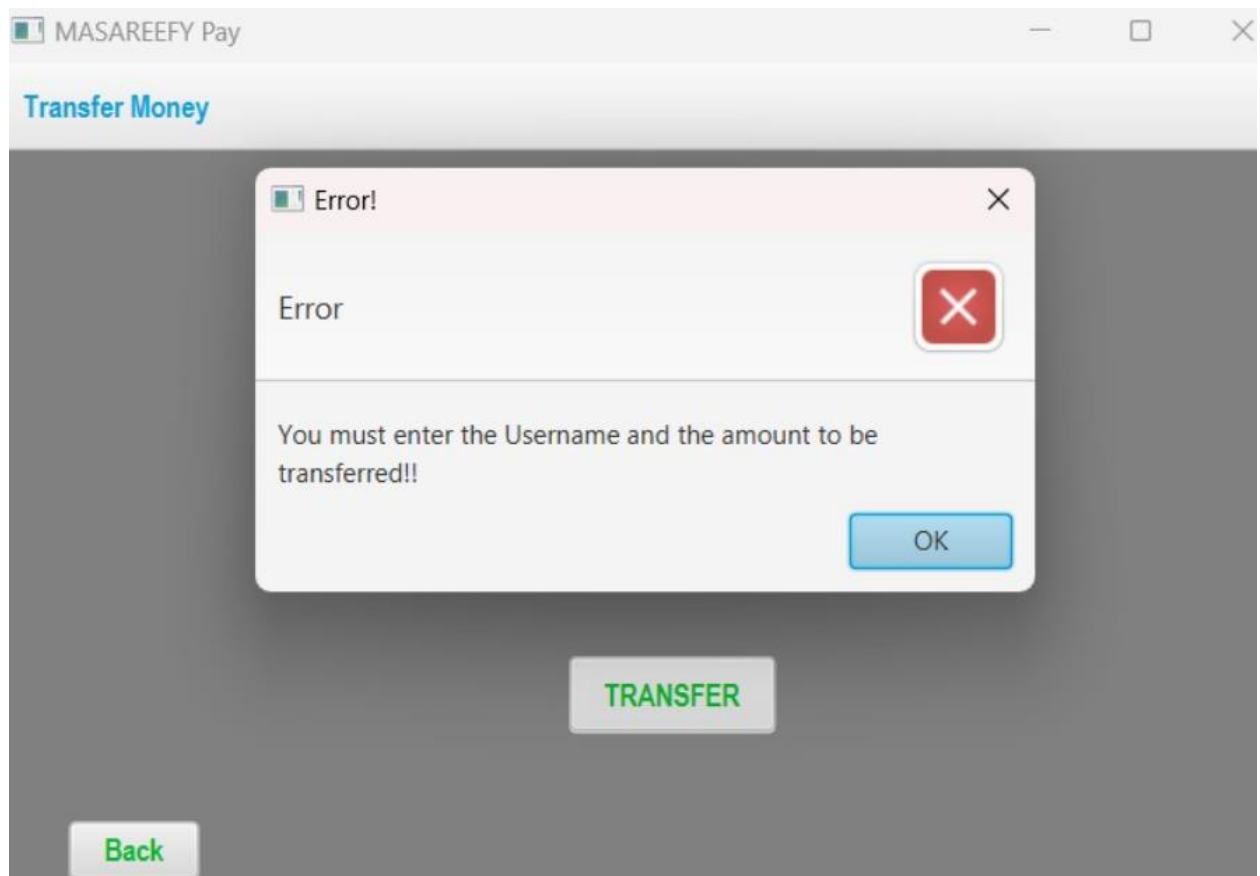


The screenshot shows a web application window titled "MASAREEFY Pay" with standard window controls (minimize, maximize, close). Below the title bar is a header with the text "Transfer Money" in blue. The main content area has a dark gray background and contains two input fields with green labels: "AMOUNT" and "USERNAME". The "AMOUNT" field is a small text input, and the "USERNAME" field is a wider text input. Below these fields is a large, light gray button with the text "TRANSFER" in green. In the bottom left corner, there is a smaller, light gray button with the text "Back" in green.

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

MASAREEFY Pay

Pay Bill

Please Enter The Bill Amount You Want to Pay

BILL AMOUNT

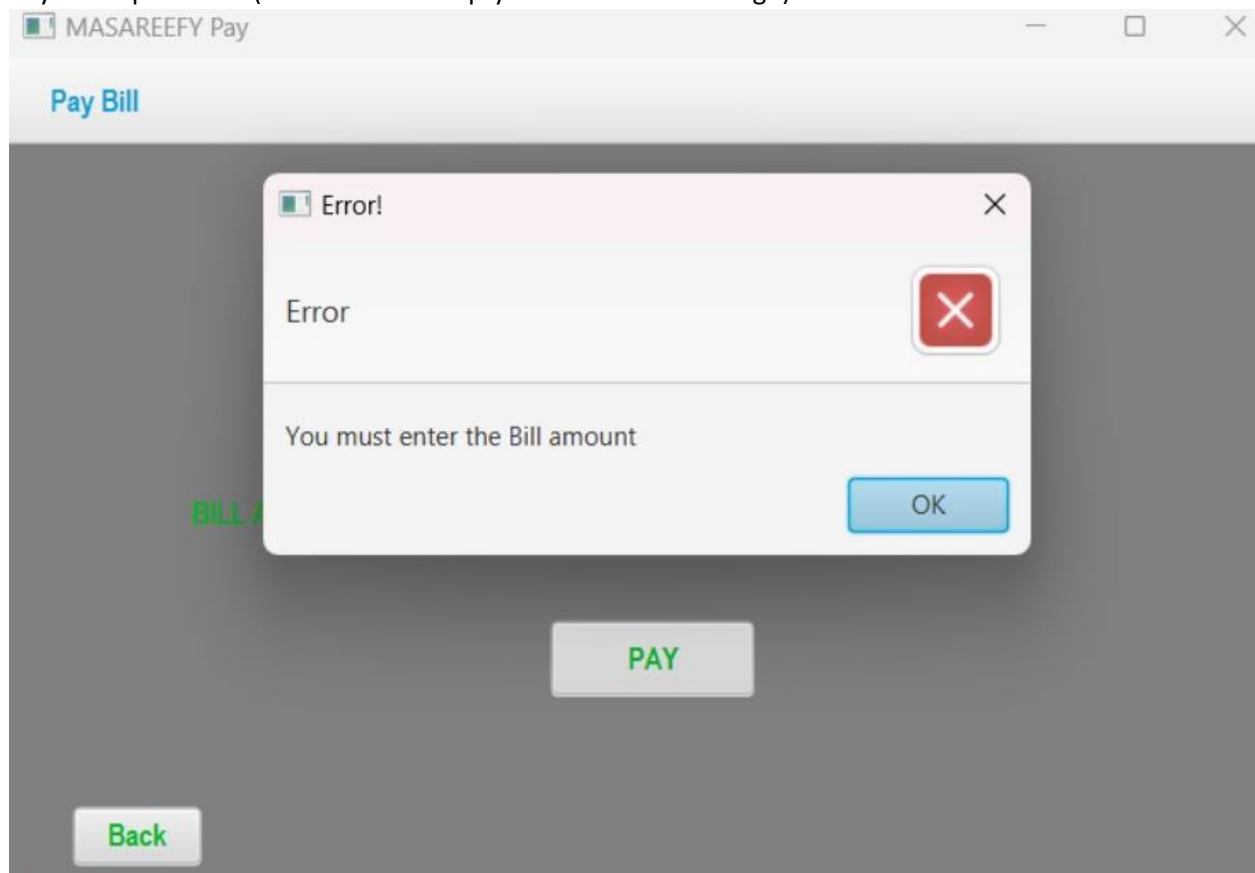
PAY

Back

Pay Bill :Pay: if Balance \geq 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)



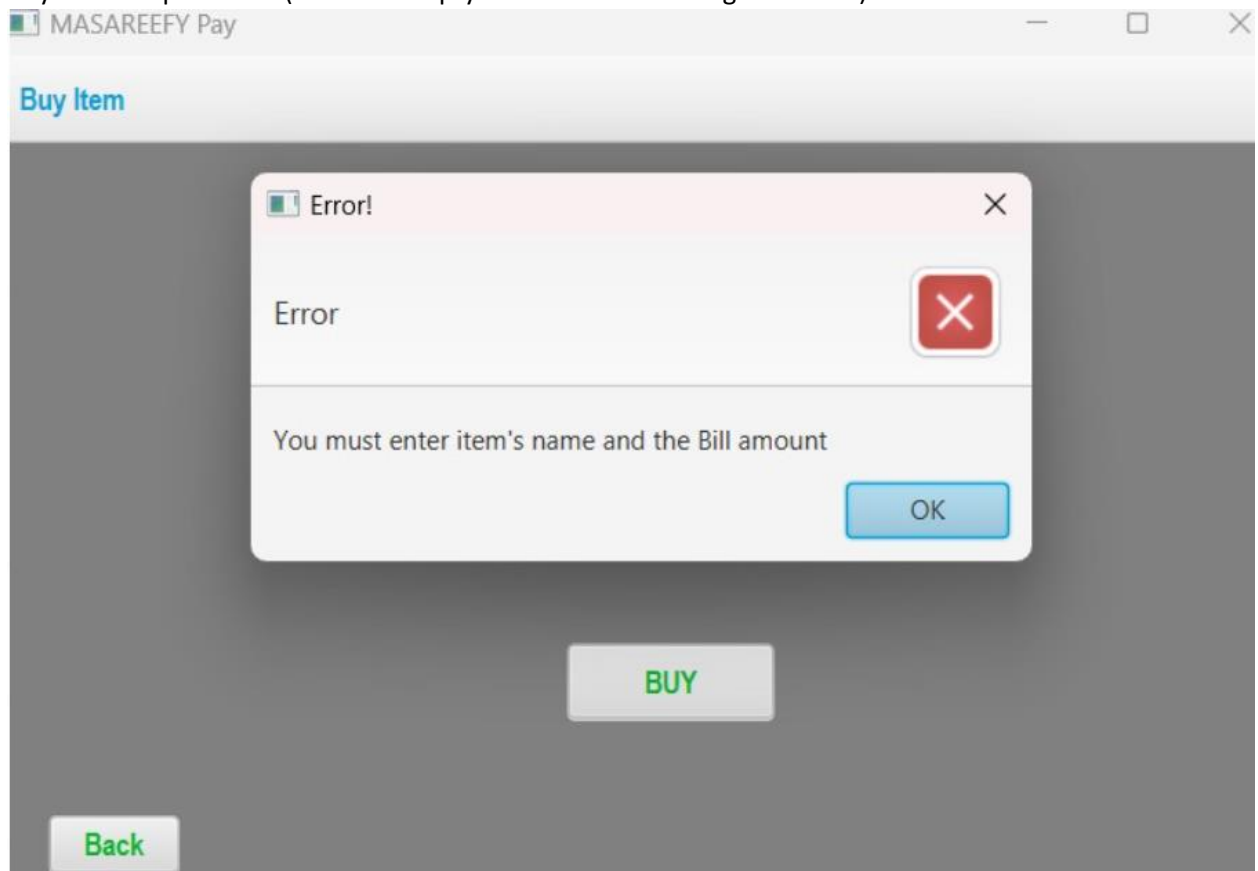
Buy Items State

The screenshot shows a web application window titled "MASAREEFY Pay". Inside the window, there is a header bar with the text "Buy Item". The main content area has a dark gray background. It contains two input fields: the first is labeled "ITEM" in green text and has a blue border; the second is labeled "PRICE" in green text and has a white border. Below these fields is a green "BUY" button. In the bottom left corner, there is a green "Back" button.

Buy Items :Buy: if $\text{Balance} \geq \text{Price}$ 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

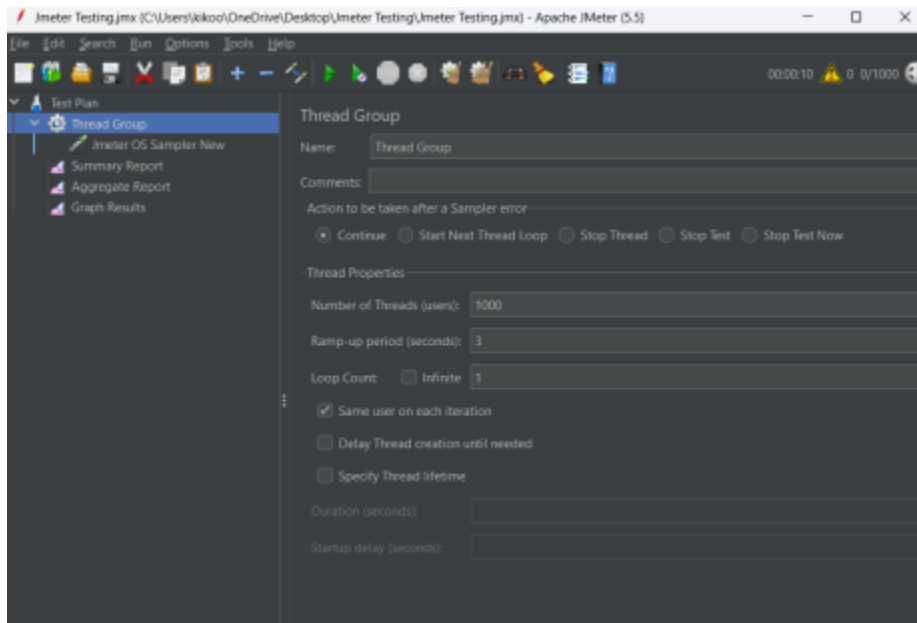
In JMeter, when you configure multiple threads and a ramp-up period, it does create separate instances of the code example for each thread. Each thread runs independently, simulating concurrent users accessing your banking system. While it's true that JMeter is executing multiple instances of the code 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.

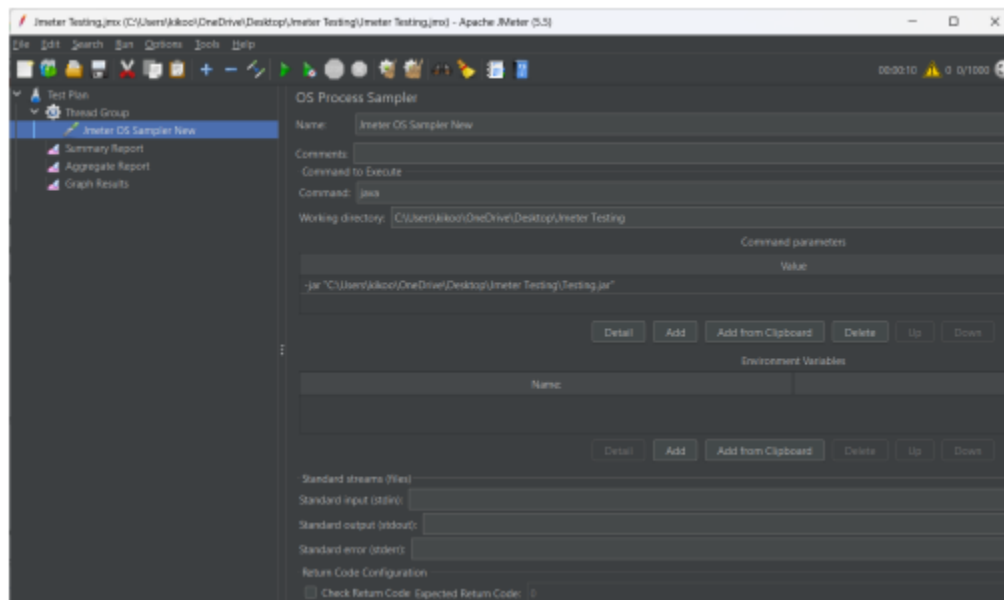
JMeter Test Example Code

```
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

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename: ☐ Log/Display Only ☐ Errors ☐ Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Jmeter OS Sampler New	1000	3491	87	9497	2868.65	0.00%	93.0/sec	15.71	0.00	173.0
TOTAL	1000	3491	87	9497	2868.65	0.00%	93.0/sec	15.71	0.00	173.0

Aggregate

Aggregate Report

Name: Aggregate Report

Comments:

Write results to file / Read from file

Filename: ☐ Log/Display Only ☐ Errors ☐ Successes

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
Jmeter OS Sampler New	1000	3491	2553	8018	8729	9341	87	9497	0.00%	93.0/sec	15.71	0.00
TOTAL	1000	3491	2553	8018	8729	9341	87	9497	0.00%	93.0/sec	15.71	0.00

Performance

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename: ☐ Log/Display Only ☐ Errors ☐ Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Jmeter OS Sampler New	1	34	34	34	0.00	0.00%	29.4/sec	4.97	0.00	173.0
TOTAL	1	34	34	34	0.00	0.00%	29.4/sec	4.97	0.00	173.0

Report Graph

