SOFTWARE TESTING PROJECT

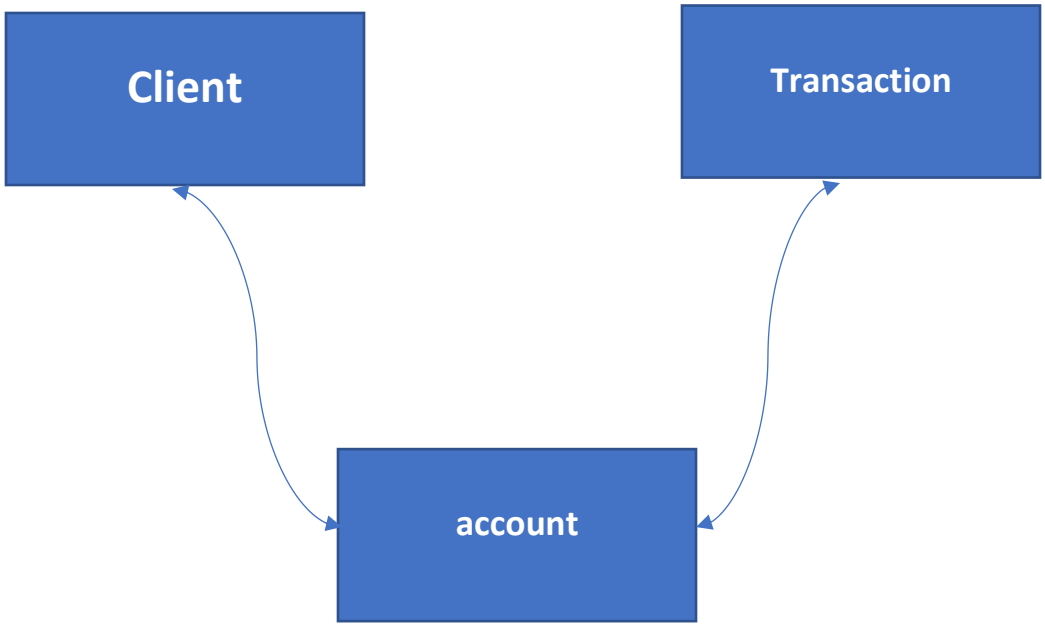
Online Banking System

Team Members:

- Malak Ahmed Yehia 2001350
- Youssef Emad Eldin 20P3844
- Sherwette Mohamed Khali Barakat 20P8105
- Donia Sameh Farouk 20P3424
- Mohamed Ibrahim Elsayed 20P8449

Overview:

A banking system contains three main classes client, transaction, account. Where when needing to add a new client an account is formed and at each transaction the balance at account is changed according to the transaction.



1. Client class:

```java
import java.util.ArrayList;

public class Client {
    private String Name;
    private String phoneNo;
    private String Address;
    private String Password;
    private Account account;


    public Client(String Name, String Password){
        this.Name = Name;
        this.Password = Password;
        createAccount();
    }
    public Client(String Name, String Password, String phoneNo){
        this.Name = Name;
        this.Password = Password;
        this.phoneNo = phoneNo;
        createAccount();
    }
    public Client(String Name, String Password, String phoneNo, String Address){
        this.Name = Name;
        this.Password = Password;
        this.phoneNo = phoneNo;
        this.Address = Address;
        createAccount();
    }



    private void createAccount(){
        this.account = new Account(Name);
    }

    public Account getAccount() {
        return account;
    }

    public String getName() {
        return Name;
    }

    public String getPhoneNo() {
        return phoneNo;
    }

    public String getAddress() {
        return Address;
    }

    public String getPassword() {
        return Password;
    }
}
```

2. Account Class:

```
import java.util.ArrayList;

public class Account {
    private String Name;
    private int AccountNo;
    private double Balance;
    private static int counter=0;
    private ArrayList<String> transactions_list= new ArrayList<String>();

    public Account(String Name){
        this.Name = Name;
        this.AccountNo = counter;
        counter++;
        this.Balance = 0;
        System.out.println("Account Created");
    }

    public double getBalance() {
        return Balance;
    }

    public String updateBalance(double balance) {
        if ((Balance+balance< 0) && (balance < 0)){
            System.out.println("Invalid Transaction");
            return("Failed");
        }
       else if(balance >= 'a' && balance <= 'z'|| (balance >= 'A' && balance <= 'Z')){
            return("Failed");
        }

        else{
            Balance += balance;
            return ("Successful");
        }
    }

    public void addTransaction(String transact)
    {
        this.transactions_list.add(transact);
    }

    public ArrayList<String> getTransactions_list() {
        return transactions_list;
    }

    public void printTransactions()
    {
        int val = 0;
        while(transactions_list.size() > val)
        {
            System.out.println(transactions_list.get(val));
            val++;
        }
    }

    public String getName() {
        return Name;
    }

    public int getAccountNo() {
        return AccountNo;
```

```
    }
}
```

3.Transaction:

```
import java.util.Random;
public class Transaction {
    private String type;
    //private String time;
    private double amount;
    private Account account;
    private Account transferAccount;
    private String payCode;
    static double[] prices = {100,200,300,400,500,600,700,800,900,1000};


    public Transaction(String type, double amount, Account account)
    {
        this.type = type;
        //this.time = time;
        this.account = account;
        this.amount = amount;


        checkType();
    }

    public Transaction(String type,Account account, String code)
    {
        this.type = type;

        //this.time = time;
        this.account = account;

        this.payCode = code;

        checkType();


    }
    public Transaction(String type, double amount, Account account,Account
transferAccount)
    {

        this.type = type;
        //this.time = time;
        this.account = account;
        this.transferAccount = transferAccount;
        this.amount = amount;
        checkType();
    }



    public void checkType(){
        if(type == "Withdraw"){
            withdraw();
        }
        else if(type == "Deposit"){
            deposit();
        }
        else if(type == "Transfer"){
```

```
            transferMoney();
        }
        else if(type == "Pay")
        {
            payOnline();

        }
    }

    public void payOnline()
    {
        String check = account.updateBalance(-prices[payCode.charAt(0) - '0']);
        if(check == "Successful")
        {
            this.account.addTransaction("Item with code "+ this.payCode + " was purchased
$" + prices[payCode.charAt(0) - '0'] + " Successfully");
        }
        else
        {
            this.account.addTransaction("Unable to purchase item with code "+
this.payCode);
        }

    }

    public void withdraw()
    {
        String check;
        check = account.updateBalance(-amount);
        if(check == "Successful")
        {
            this.account.addTransaction(type + " $" + amount + " Successful");
        }
        else
        {
            this.account.addTransaction(type + " $" + amount + " Failed");
        }


    }

    public void deposit(){
        account.updateBalance(amount);
        this.account.addTransaction(type + " $" + amount + " Successful");
    }

    public void transferMoney(){
        if(account.getBalance()<amount){
            System.out.println("Can't proceed transfer");
            this.account.addTransaction(type + " $" + amount + " Failed");

        }
        else{
            account.updateBalance(-amount);
            transferAccount.updateBalance(amount);
            this.account.addTransaction(type + " $" + amount + " Successful");
            this.transferAccount.addTransaction("$" + amount + " were transferred to you
from " + account.getName());
            System.out.println(amount + " Transferred Successfully!");
        }
    }

}
```

Testing:

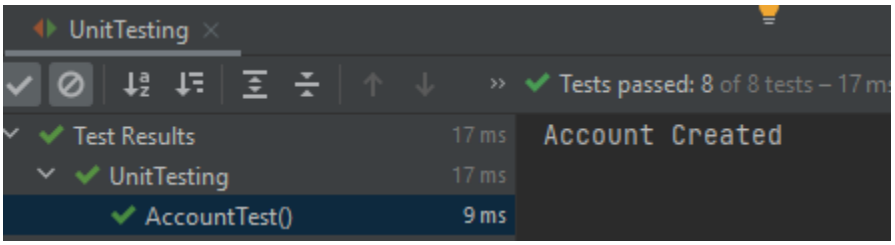
1. **Unit Testing:**

   Each function was tested individually to make sure that the foundations are correct.

   The following is sample from each class testing:

Account class test:

```
class UnitTesting {
    @Test
    public void AccountTest() {
        Account client1 = new Account( Name: "Seif");
        client1.updateBalance(866);
        assertEquals( expected: 866,client1.getBalance());
        assertEquals( expected: "Seif",client1.getName());
        client1.updateBalance(10);
        assertEquals( expected: 876,client1.getBalance());
    }
```



An account was created successfully

A balance updated to 866 and to it 10, assertequal assured that each function worked correctly.

```java
@Test
public void AccountTest2() {
    Account client1 = new Account( Name: "Seif");
    client1.updateBalance(866);
    assertNotEquals( unexpected: 86,client1.getBalance());
    client1.updateBalance(10);
    assertNotEquals( unexpected: 76,client1.getBalance());
}

@Test
public void AccountTest3() {
    Account client1 = new Account( Name: "Seif");
    client1.updateBalance(-866);
    assertEquals( expected: 0,client1.getBalance());
    client1.updateBalance(10);
    assertNotEquals( unexpected: 76,client1.getBalance());
}
```

```java
@Test
public void AccountTest4() {
    Account client1 = new Account( Name: "Seif");
    client1.updateBalance('p');
    assertEquals( expected: 0,client1.getBalance());
}
```

Tests passed: 8 of 8 tests – 17 ms

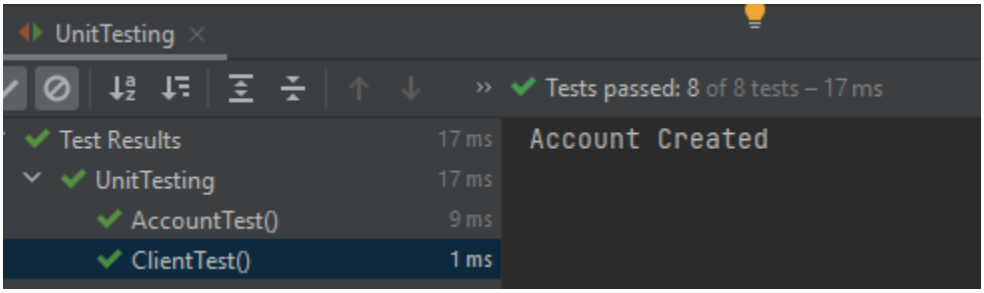
Test Results	17 ms
UnitTesting	17 ms
AccountTest()	9 ms
ClientTest()	1 ms
TransactionTest2()	3 ms
TransactionTest3()	1 ms
TransactionTest4()	1 ms
TransactionTest()	1 ms
AccountTest2()	1 ms
AccountTest3()	

```
Account Created
Invalid Transaction
```

This test assured that the system doesn’t accept negative transactions.

Client class test:

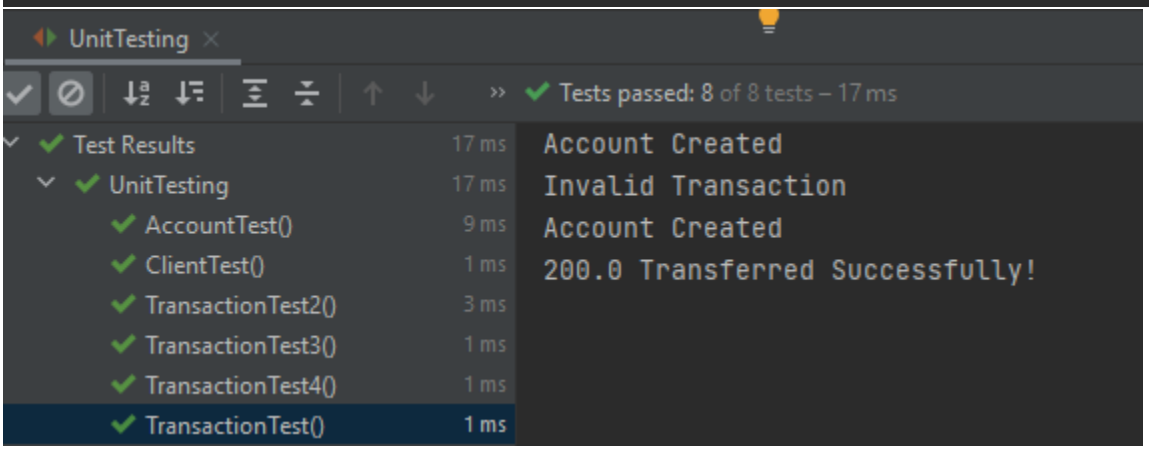
```java
@Test
public void ClientTest(){
    Client client1=new Client( Name: "Yara", Password: "12345", phoneNo: "0129232434", Address: "NasrCity");
    assertEquals( expected: "Yara",client1.getName());
    assertEquals( expected: "12345", client1.getPassword());
    assertEquals( expected: "0129232434",client1.getPhoneNo());
    assertEquals( expected: "NasrCity",client1.getAddress());
}
```



An account was created successfully with the given information.

Transaction test:

```java
@Test
public void TransactionTest(){
    Client client1 = new Client( Name: "Seif", Password: "****");
    Transaction trans1=new Transaction( type: "Withdraw", amount: 200,client1.getAccount());
    assertEquals( expected: 0,client1.getAccount().getBalance());

    Transaction trans2 = new Transaction( type: "Deposit", amount: 200,client1.getAccount());
    assertEquals( expected: 200,client1.getAccount().getBalance());

    Client client2 = new Client( Name: "Yassin", Password: "******");
    Transaction trans3 = new Transaction( type: "Transfer", amount: 200,client1.getAccount(),client2.getAccount());
    assertEquals( expected: 0,client1.getAccount().getBalance());
    assertEquals( expected: 200,client2.getAccount().getBalance());

    Transaction trans4=new Transaction( type: "Withdraw", amount: 100,client2.getAccount());
    assertEquals( expected: 100,client2.getAccount().getBalance());
}
```



It tested the odd cases that could occur in the system as trying to withdraw from an empty account.

it tested the money transfer between an account and another.

```java
@Test
public void TransactionTest2(){ //tests the printed out transactions made by a certain account
    ArrayList<String> transactions_list1= new ArrayList<String>(){{
        add("Deposit $400.0 Successful");
        add("Deposit $600.0 Successful");
        add("Withdraw $150.0 Successful");
        add("Transfer $150.0 Successful");
    }
    };
    Client client8 = new Client( Name: "Seif Sameh", Password: "*****");
    Client client9 = new Client( Name: "Yassin Mahgoub", Password: "*****");

    Transaction trans = new Transaction( type: "Deposit",  amount: 400, client8.getAccount());
    Transaction trans2 = new Transaction( type: "Deposit",  amount: 600, client8.getAccount());
    Transaction trans3 = new Transaction( type: "Withdraw",  amount: 150, client8.getAccount());
    Transaction trans4 = new Transaction( type: "Transfer",  amount: 150, client8.getAccount(), client9.getAccount());
    assertEquals(transactions_list1,client8.getAccount().getTransactions_list());

}
```

UnitTesting

Tests passed: 8 of 8 tests – 17 ms

Test Results	17 ms
UnitTesting	17 ms
AccountTest()	9 ms
ClientTest()	1 ms
TransactionTest2()	3 ms

```
Account Created
Account Created
150.0 Transferred Successfully!
```

```java
@Test
public void TransactionTest3(){
    Client client1 = new Client( Name: "Seif", Password: "****");
    Transaction trans1=new Transaction( type: "Withdraw", amount: 200,client1.getAccount());
    assertEquals( expected: 0,client1.getAccount().getBalance());

    Transaction trans2 = new Transaction( type: "Deposit", amount: -200,client1.getAccount());
    assertEquals( expected: 0,client1.getAccount().getBalance());

    Transaction trans6 = new Transaction( type: "Deposit", amount: 200,client1.getAccount());
    assertEquals( expected: 200,client1.getAccount().getBalance());

    Client client2 = new Client( Name: "Yassin", Password: "******");
    Transaction trans3 = new Transaction( type: "Transfer", amount: 400,client1.getAccount(),client2.getAccount());
    //wont proceed transfering money because its not availble in the client's account
    assertEquals( expected: 200,client1.getAccount().getBalance());

    /*Transaction trans4=new Transaction("Withdraw",100,client2.getAccount());
    assertEquals(100,client2.getAccount().getBalance());*/
}
```

UnitTesting

Tests passed: 8 of 8 tests – 17 ms

Test Results	17 ms
UnitTesting	17 ms
AccountTest()	9 ms
ClientTest()	1 ms
TransactionTest2()	3 ms
TransactionTest3()	1 ms

```
Account Created
Invalid Transaction
Invalid Transaction
Account Created
Can't proceed transfer
```

```java
@Test
public void TransactionTest4(){
    Client client1 = new Client( Name: "Seif", Password: "****");
    Transaction trans1=new Transaction( type: "Withdraw", amount: 200,client1.getAccount());
    assertEquals( expected: 0,client1.getAccount().getBalance());

    Transaction trans2 = new Transaction( type: "Deposit", amount: -200,client1.getAccount());
    assertEquals( expected: 0,client1.getAccount().getBalance());

    Transaction trans6 = new Transaction( type: "Deposit", amount: 200,client1.getAccount());
    assertEquals( expected: 200,client1.getAccount().getBalance());

    Client client2 = new Client( Name: "Yassin", Password: "******");
    Transaction trans3 = new Transaction( type: "Transfer", amount: 400,client1.getAccount(),client2.getAccount());
    //wont proceed transfering money because its not availble in the client's account
    assertEquals( expected: 200,client1.getAccount().getBalance());

    Transaction trans4 = new Transaction( type: "Pay",client1.getAccount(), code: "0091");
    assertEquals( expected: 100,client1.getAccount().getBalance());
}
```

UnitTesting

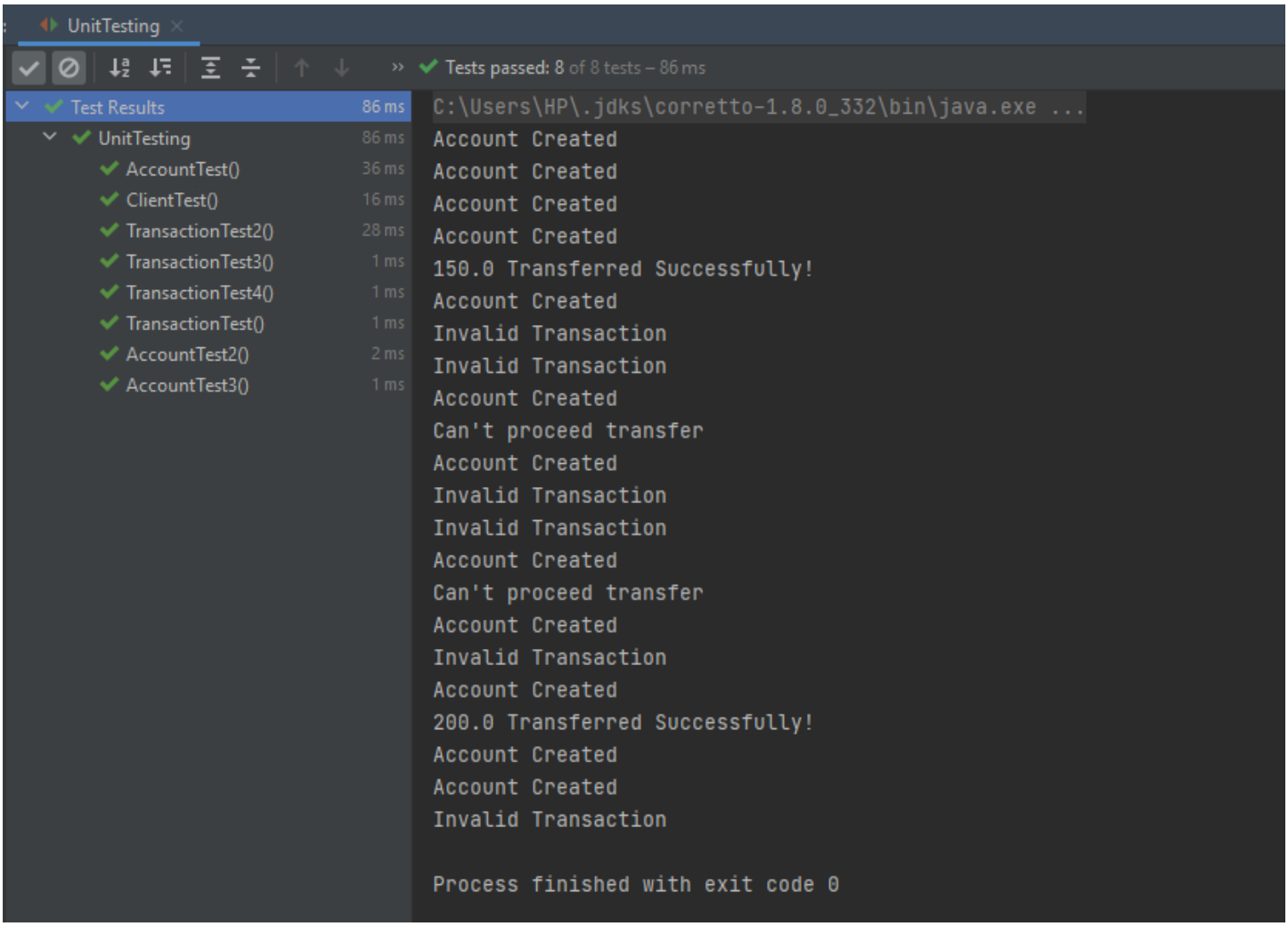
Tests passed: 8 of 8 tests – 17 ms

Test Results	17 ms
UnitTesting	17 ms
AccountTest()	9 ms
ClientTest()	1 ms
TransactionTest2()	3 ms
TransactionTest3()	1 ms
TransactionTest4()	1 ms

```
Account Created
Invalid Transaction
Invalid Transaction
Account Created
Can't proceed transfer
```

This test suit tested other cases as trying to transfer from an empty account to other account.

Output:


UnitTesting
Tests passed: 8 of 8 tests – 86 ms
Test Results 86 ms
UnitTesting 86 ms
AccountTest() 36 ms
ClientTest() 16 ms
TransactionTest2() 28 ms
TransactionTest3() 1 ms
TransactionTest4() 1 ms
TransactionTest() 1 ms
AccountTest2() 2 ms
AccountTest3() 1 ms
C:\Users\HP\.jdks\corretto-1.8.0_332\bin\java.exe ...
Account Created
Account Created
Account Created
Account Created
150.0 Transferred Successfully!
Account Created
Invalid Transaction
Invalid Transaction
Account Created
Can't proceed transfer
Account Created
Invalid Transaction
Invalid Transaction
Account Created
Can't proceed transfer
Account Created
Invalid Transaction
Account Created
200.0 Transferred Successfully!
Account Created
Account Created
Invalid Transaction
Process finished with exit code 0

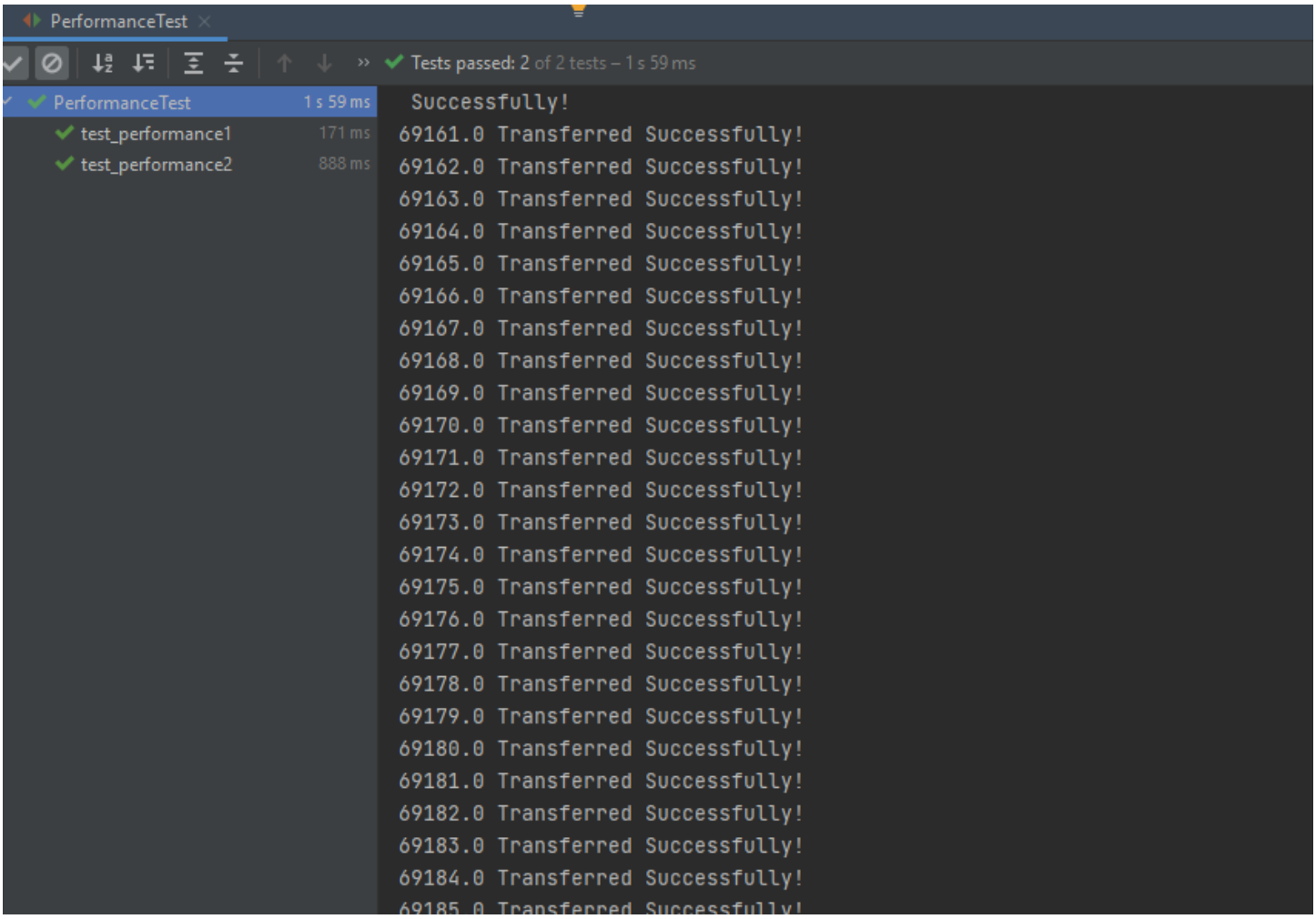
Performance testing:

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

public class PerformanceTest {
    //Client clients[];

    @Test(timeout=550)
    public void test_performance1(){
        Client client1 = new Client("Seif","****");
        Client client2 = new Client("Seif2","****");
        for (int i=20; i<100000;i++){
            Transaction trans1 = new Transaction("Deposit",i,client1.getAccount());
            Transaction trans3 = new Transaction("Deposit",i,client2.getAccount());
        }
        for (int i=20; i<100000;i++){
            Transaction trans2 = new Transaction("Withdraw",i,client1.getAccount());
            Transaction trans4 = new Transaction("Withdraw",i,client2.getAccount());
        }
    }

    @Test(timeout=3000)
    public void test_performance2(){
        Client client1 = new Client("Seif","****");
        Client client2 = new Client("Seif2","****");
        for (int i=20; i<100000;i++){
            Transaction trans1 = new Transaction("Deposit",i,client1.getAccount());
            Transaction trans3 = new Transaction("Deposit",i,client2.getAccount());
        }
        for (int i=20; i<100000;i++){
            Transaction trans5 = new
Transaction("Transfer",i,client1.getAccount(),client2.getAccount());
        }
    }
}
```

the aim of performance testing to make sure that neither a conflict nor fault would appear when too many clients uses the system or many orders where given to the system.

We tested the deposit ,withdraw and transferring money from a client to other and no fault appeared.

output:


PerformanceTest
Tests passed: 2 of 2 tests – 1 s 59 ms
PerformanceTest 1 s 59 ms
test_performance1 171 ms
test_performance2 888 ms
Successfully!
69161.0 Transferred Successfully!
69162.0 Transferred Successfully!
69163.0 Transferred Successfully!
69164.0 Transferred Successfully!
69165.0 Transferred Successfully!
69166.0 Transferred Successfully!
69167.0 Transferred Successfully!
69168.0 Transferred Successfully!
69169.0 Transferred Successfully!
69170.0 Transferred Successfully!
69171.0 Transferred Successfully!
69172.0 Transferred Successfully!
69173.0 Transferred Successfully!
69174.0 Transferred Successfully!
69175.0 Transferred Successfully!
69176.0 Transferred Successfully!
69177.0 Transferred Successfully!
69178.0 Transferred Successfully!
69179.0 Transferred Successfully!
69180.0 Transferred Successfully!
69181.0 Transferred Successfully!
69182.0 Transferred Successfully!
69183.0 Transferred Successfully!
69184.0 Transferred Successfully!

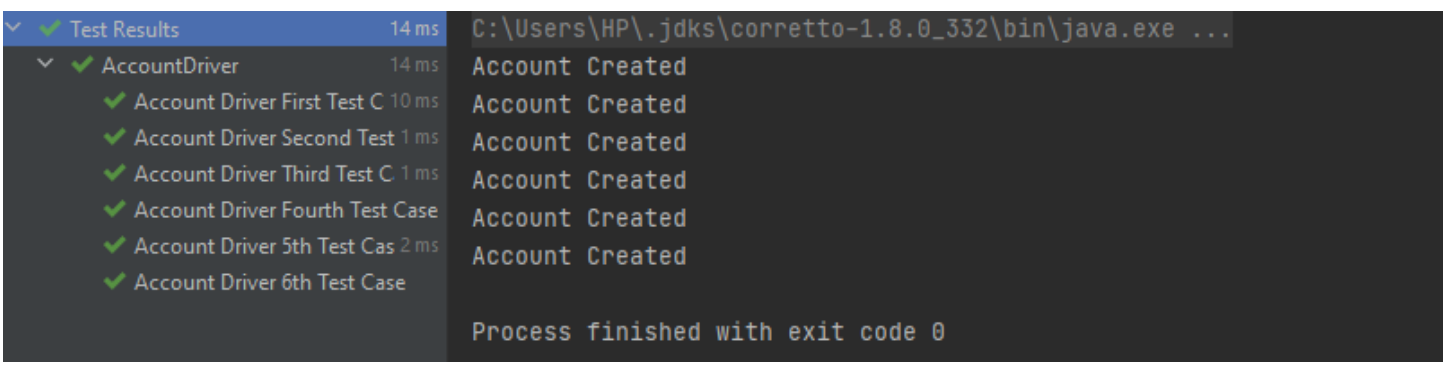
integration testing:

we adopted a bottom up approach as starting at the bottom of the hierarchy again means that the critical modules are generally built and tested first and therefore any errors or mistakes in these forms of modules are identified early in the process.

Account driver:

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

// BOTTOM-UP Approach with Account as actual code and drivers Client and Transaction
class AccountDriver{
    class driver{
        public boolean format(Account account){
            account.updateBalance(-200);
            account.updateBalance(300);
            if(account.getBalance()==1100){
                return true;
            }
            return false;
        }
        public boolean format1(Account account){
            account.addTransaction("Deposit $200 Successful");
            account.addTransaction("Withdraw $100 Successful");
            if(account.getTransactions_list().get(0)=="Deposit $200 Successful" &&
account.getTransactions_list().get(1)=="Withdraw $100 Successful"){
                return true;
            }
            return false;
        }
        public boolean format2(Account account,String Name){
            if(account.getName() == Name){
                return true;
            }
            return false;
        }
    }

    Account a1;
    driver d1;

    @Test
    @DisplayName("Account Driver First Test Case")
    public void testAccountDriver1(){
        a1 = new Account("Name");
        a1.updateBalance(1000);
        d1 = new driver();
        assertTrue(d1.format(a1));
    }
    @Test
    @DisplayName("Account Driver Second Test Case")
    public void testAccountDriver2(){
        a1 = new Account("Name");
        a1.updateBalance(2000);
        d1 = new driver();
        assertFalse(d1.format(a1));
    }
    @Test
```

```java
    @DisplayName("Account Driver Third Test Case")
     public void testAccountDriver3(){
         a1 = new Account("Name");
         a1.updateBalance(1000);
         d1 = new driver();
         assertTrue(d1.format1(a1));
     }
     @Test
     @DisplayName("Account Driver Fourth Test Case")
     public void testAccountDriver4(){
         a1 = new Account("Name");
         a1.addTransaction("Deposit $400.0 Successful");
         d1 = new driver();
         assertFalse(d1.format1(a1));
     }
     @Test
     @DisplayName("Account Driver 5th Test Case")
     public void testAccountDriver5(){
         a1 = new Account("Name");
         d1 = new driver();
         assertEquals(true,d1.format2(a1,"Name"));
     }
     @Test
     @DisplayName("Account Driver 6th Test Case")
     public void testAccountDriver6(){
         a1 = new Account("Name");
         d1 = new driver();
         assertEquals(false,d1.format2(a1,"NAME"));
     }

}
```

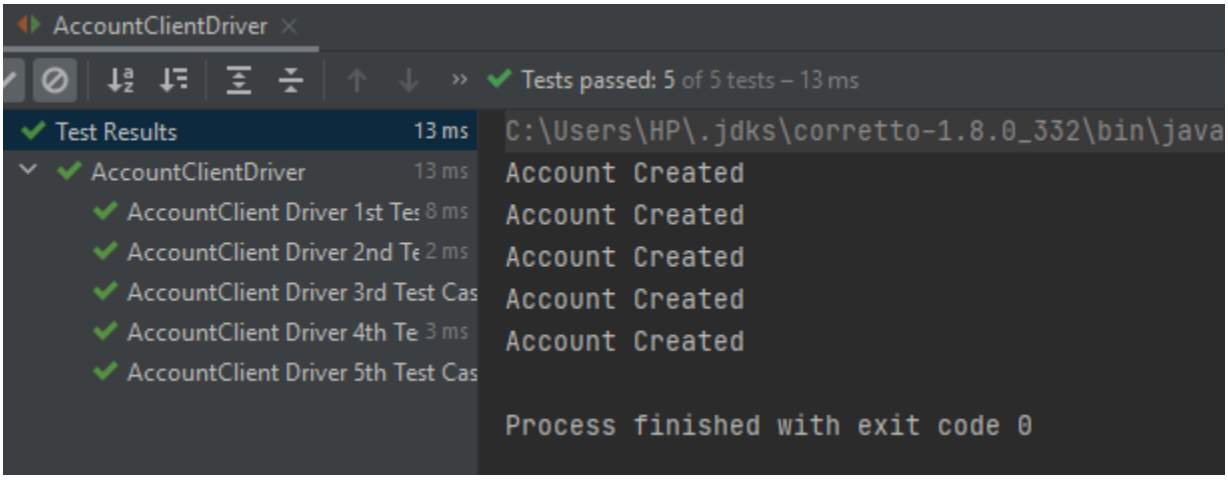


We created a driver for Account class to make that the base of the program, is invoked correctly with no faults.

Account client driver:

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

// BOTTOM-UP Approach with Account and Client as actual code and drivers Transaction
//Creation of Client implies creation of Account
public class AccountClientDriver {
    class driver{
        public boolean format(Client client){
            client.getAccount().updateBalance(2000);
            client.getAccount().updateBalance(-500);
            if(client.getAccount().getBalance()==1500){
                return true;
            }
            return false;
        }
        public boolean format1(Client client){
            client.getAccount().addTransaction("Deposit $2000 Successful");
            client.getAccount().addTransaction("Withdraw $500 Successful");
            if(client.getAccount().getTransactions_list().get(0)=="Deposit $2000
Successful" && client.getAccount().getTransactions_list().get(1)=="Withdraw $500
Successful"){
                return true;
            }
            return false;
        }
        public boolean format2(Client client,String Name){
            if(client.getAccount().getName() == Name){
                return true;
            }
            return false;
        }
    }

    Client c1;
    AccountClientDriver.driver d1;

    @Test
    @DisplayName("AccountClient Driver 1st Test Case")
    public void testAccountClientDriver1(){
        c1 = new Client("Name","Password");
        d1 = new AccountClientDriver.driver();
        assertTrue(d1.format(c1));
    }
    @Test
    @DisplayName("AccountClient Driver 2nd Test Case")
    public void testAccountClientDriver2(){
        c1 = new Client("Name","Password");
        c1.getAccount().updateBalance(1000);
        d1 = new AccountClientDriver.driver();
        assertFalse(d1.format(c1));
    }
    @Test
    @DisplayName("AccountClient Driver 3rd Test Case")
    public void testAccountClientDriver3(){
        c1 = new Client("Name","Password");
        c1.getAccount().addTransaction("Deposit $400.0 Successful");
        d1 = new AccountClientDriver.driver();
```

```java
        assertFalse(d1.format1(c1));
    }
    @Test
    @DisplayName("AccountClient Driver 4th Test Case")
    public void testAccountClientDriver4(){
        c1 = new Client("Name","Password");
        d1 = new AccountClientDriver.driver();
        assertEquals(true,d1.format2(c1,"Name"));
    }
    @Test
    @DisplayName("AccountClient Driver 5th Test Case")
    public void testAccountClientDriver5(){
        c1 = new Client("Name","Password");
        d1 = new AccountClientDriver.driver();
        assertEquals(false,d1.format2(c1,"NAME"));
    }
}
```



We made a driver for both Account and Client to invoke each class functions and make sure that their integration didn't cause any error.

Account client transaction:

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

public class AccountClientTransactionTester {
    Client client1;
    Client client2;
    Transaction trans;
    Transaction trans2;
    Transaction trans3;

    @BeforeEach
    public void setUp() {
        client1 = new Client("Seif Sameh","*****");
        client2 = new Client("Yassin Mahgoub","*****");
    }

    @Test
    @DisplayName("Test Case 1")
    public void test1(){
        trans = new Transaction("Deposit", 400, client1.getAccount());
        assertEquals(400,client1.getAccount().getBalance());
    }
    @Test
    @DisplayName("Test Case 2")
    public void test2(){
        Transaction trans1 = new Transaction("Deposit", 600, client1.getAccount());
        trans2 = new Transaction("Withdraw", 150, client1.getAccount());
        assertEquals(450,client1.getAccount().getBalance());
    }
    @Test
    @DisplayName("Test Case 3")
    public void test3(){
        trans = new Transaction("Deposit", 600, client1.getAccount());
        trans2 = new Transaction("Withdraw", 150, client1.getAccount());
        trans3 = new Transaction("Transfer", 150, client1.getAccount(),
client2.getAccount());
        assertEquals(300,client1.getAccount().getBalance());
    }
    @Test
    @DisplayName("Test Case 4")
    public void test4(){
        trans = new Transaction("Deposit", 600, client1.getAccount());
        trans2 = new Transaction("Withdraw", 150, client1.getAccount());
        trans3 = new Transaction("Transfer", 150, client1.getAccount(),
client2.getAccount());
        assertEquals(150,client2.getAccount().getBalance());
    }
    @Test
    @DisplayName("Test Case 5")
    public void test5(){
        trans = new Transaction("Deposit", 600, client1.getAccount());
        trans2 = new Transaction("Pay",client1.getAccount(),"30091");
        assertEquals(200,client1.getAccount().getBalance());
    }

    @AfterEach
    public void tearDown(){
        client1 = null;
```

```
        client2 = null;
    }

}
```

AccountClientTransactionTester

Tests passed: 5 of 5 tests – 12 ms

Test Results	12 ms
AccountClientTransactionTest	12 ms
Test Case 1	10 ms
Test Case 2	
Test Case 3	1 ms
Test Case 4	
Test Case 5	1 ms

```
C:\Users\HP\.jdks\corretto-1.8.0_332\bin\java.exe ...
Account Created
Account Created
Account Created
Account Created
Account Created
Account Created
150.0 Transferred Successfully!
Account Created
Account Created
150.0 Transferred Successfully!
Account Created
Account Created

Process finished with exit code 0
```

Tests passed: 5

last stage in integration testing we tested the whole system and the connection between the classes and their functions.

Faults appeared during testing:

The withdraw function used to withdraw negative money amounts leading to an increase in the balance.

Before modification:

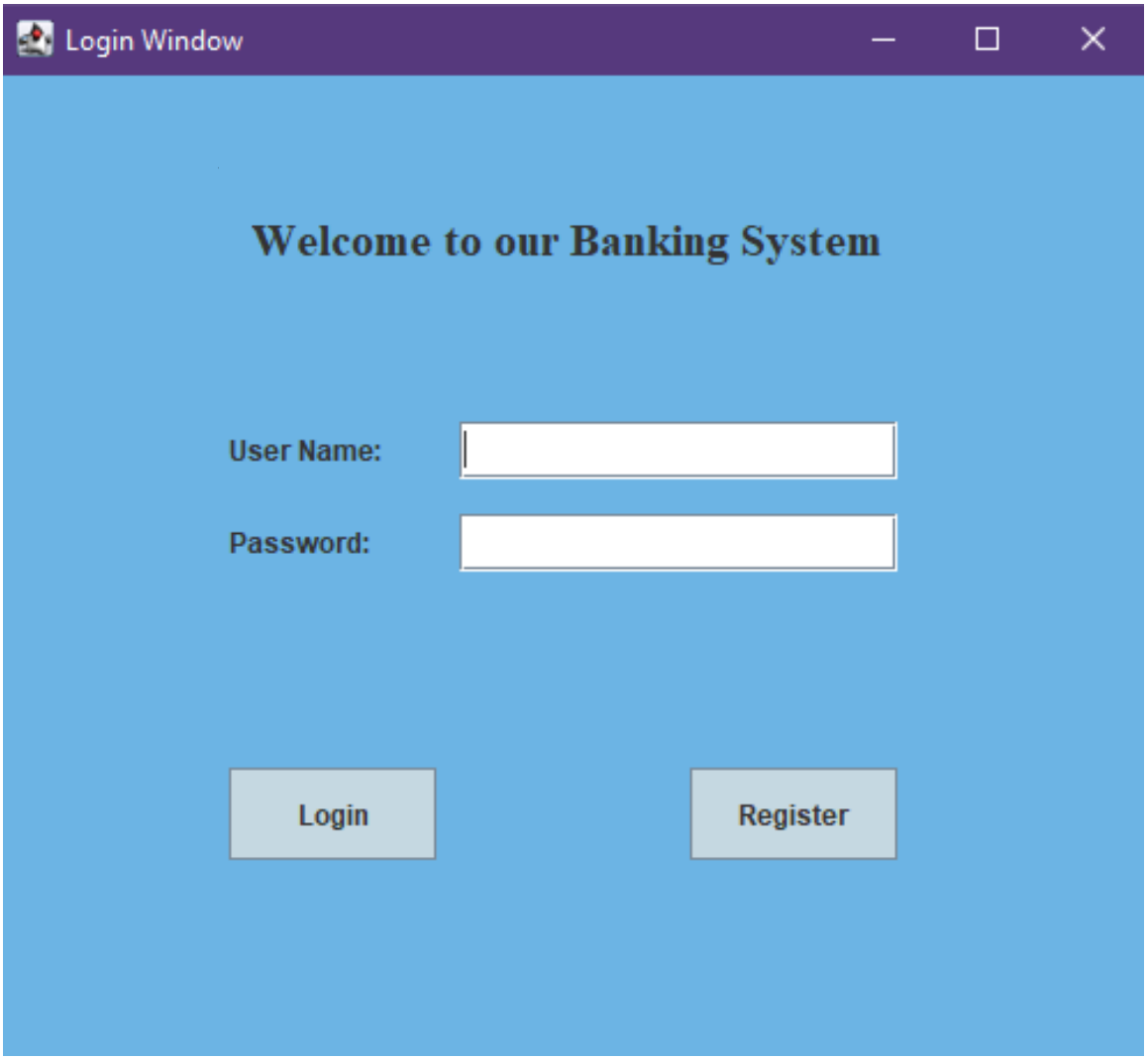
```
    public void withdraw()
    {
        account.updateBalance(-amount);
    }
```

After modification:

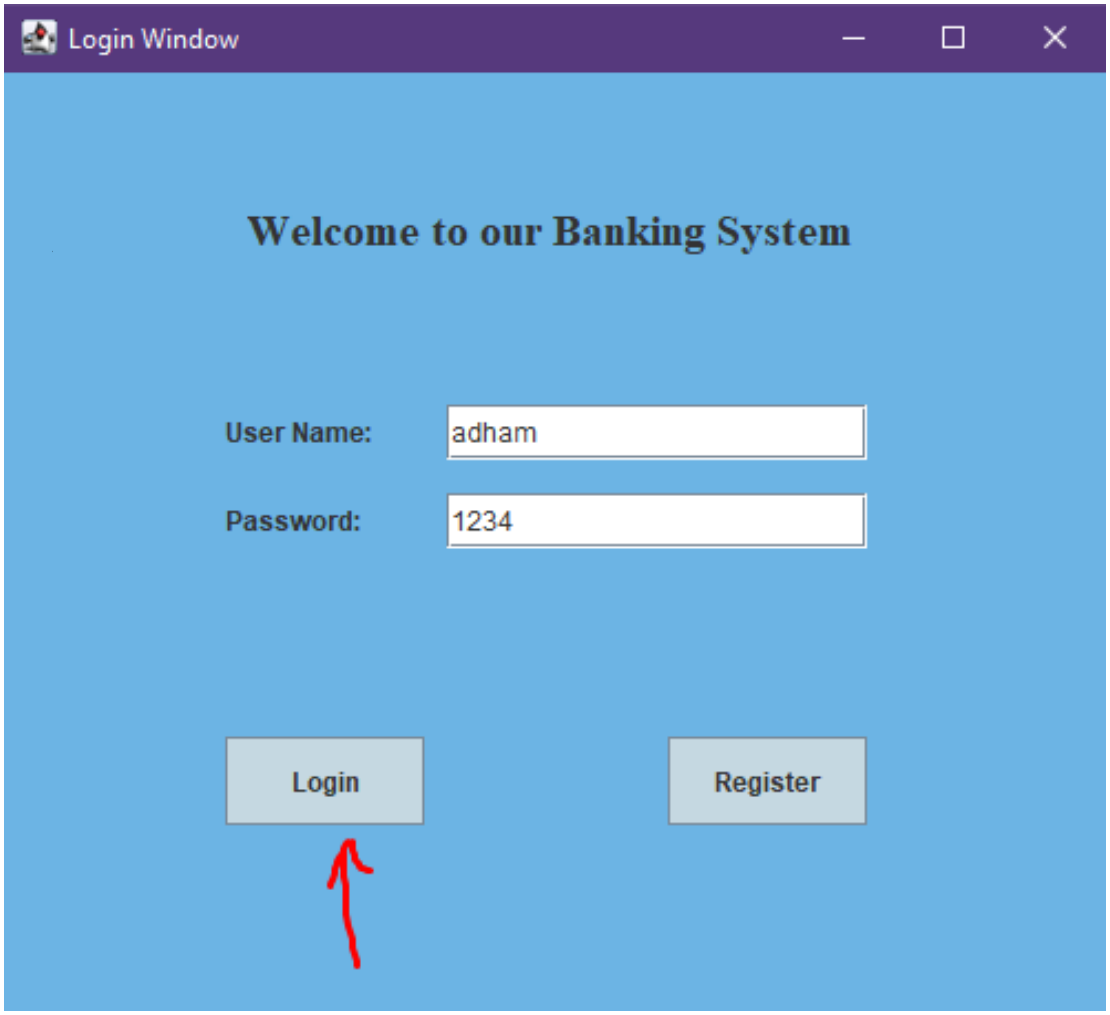
```
public void withdraw()
{
    String check;
    check = account.updateBalance(-amount);
    if(check == "Successful")
    {
        this.account.addTransaction(type + " $" + amount + " Successful");
    }
    else
    {
        this.account.addTransaction(type + " $" + amount + " Failed");
    }
```

Gui Testing

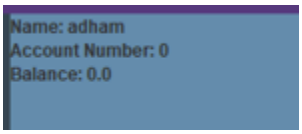
Login screen, user enters his username and password to log into their account.

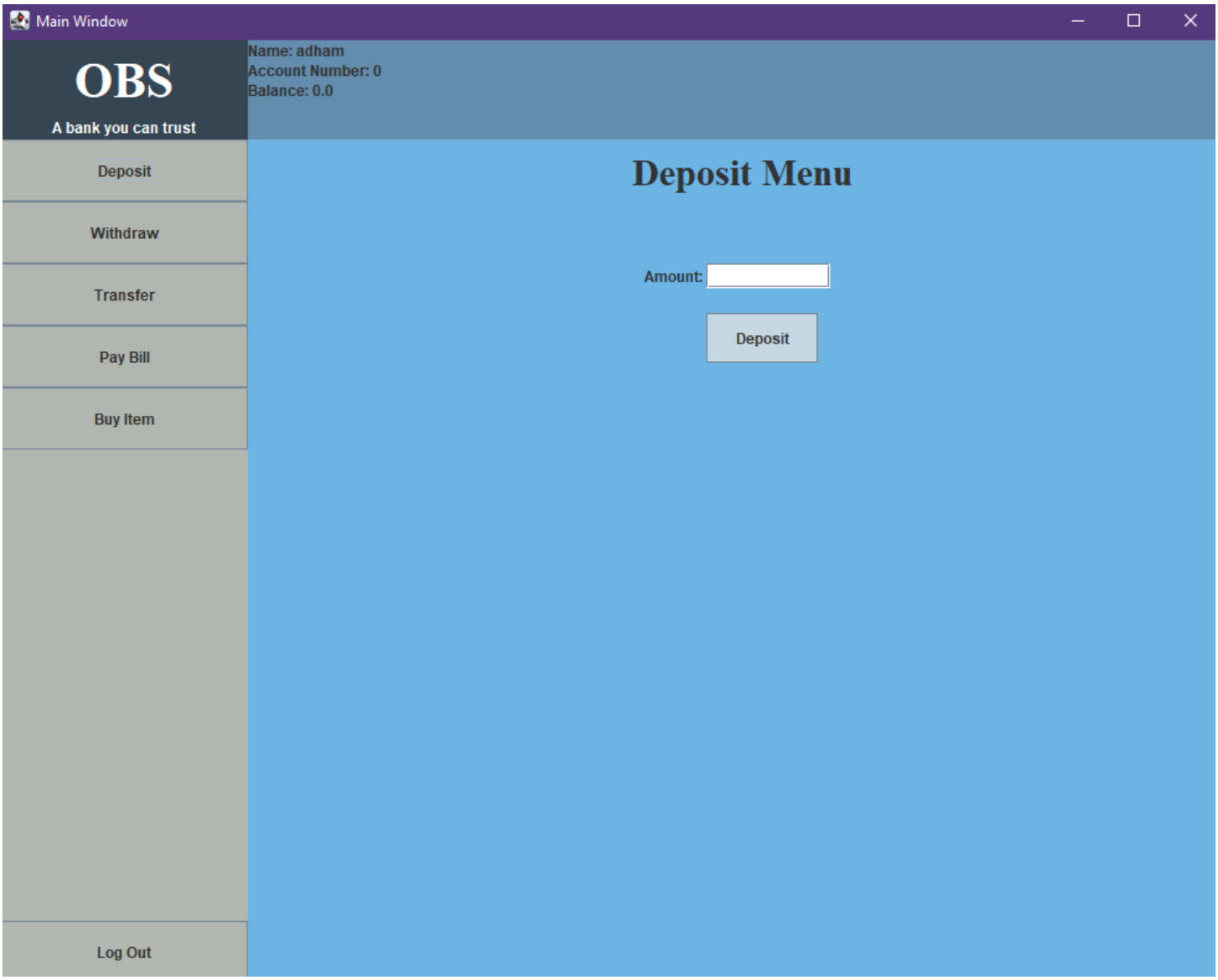


Example


Login Window
Welcome to our Banking System
User Name:
adham
Password:
1234
Login
Register

Next page after logging in is the main window which includes the functionalities of the system

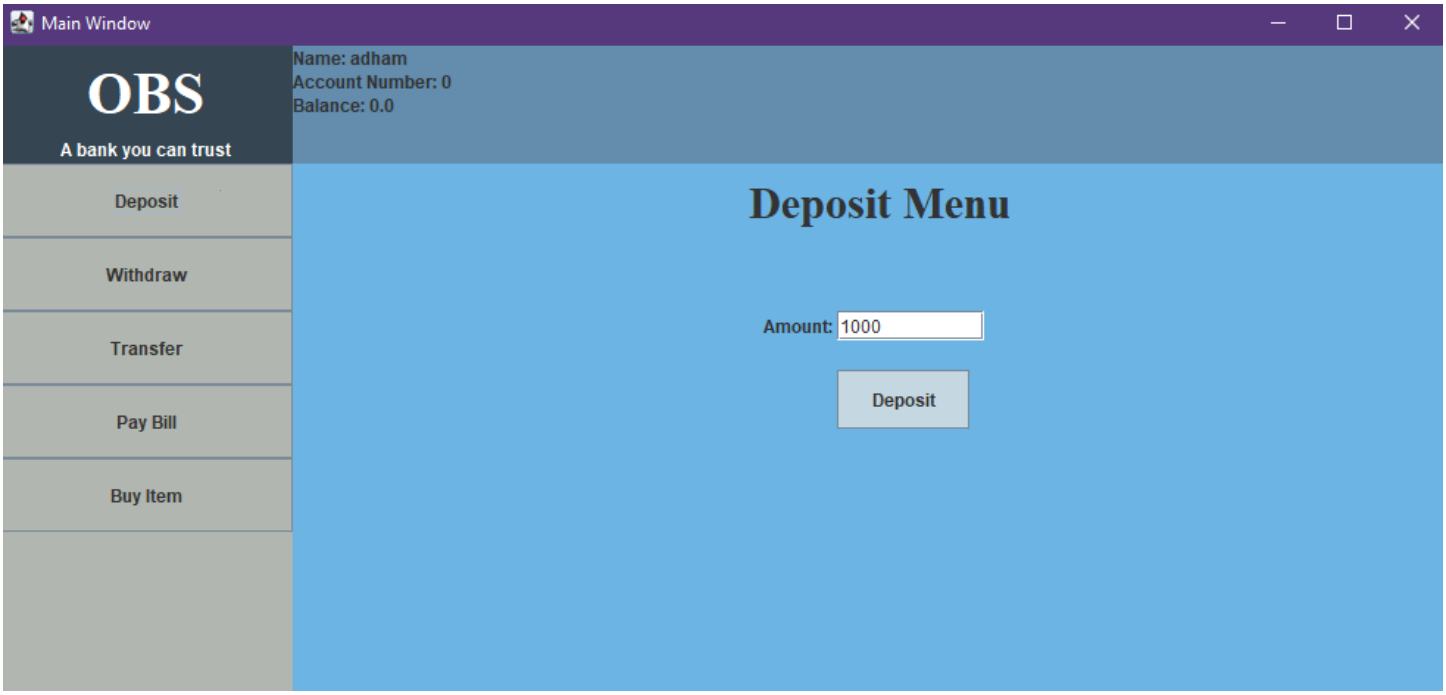


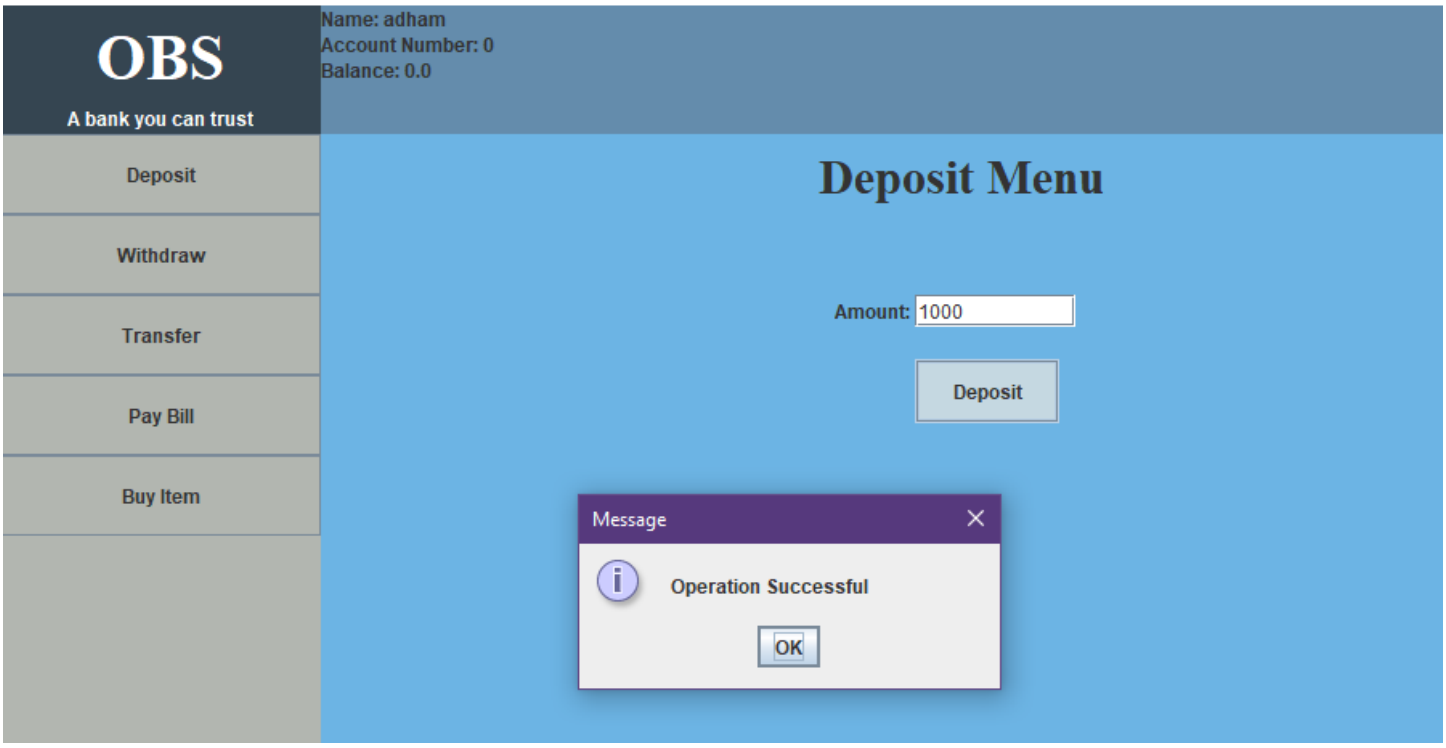


User can deposit or withdraw or transfer money, pay bill and buy an item

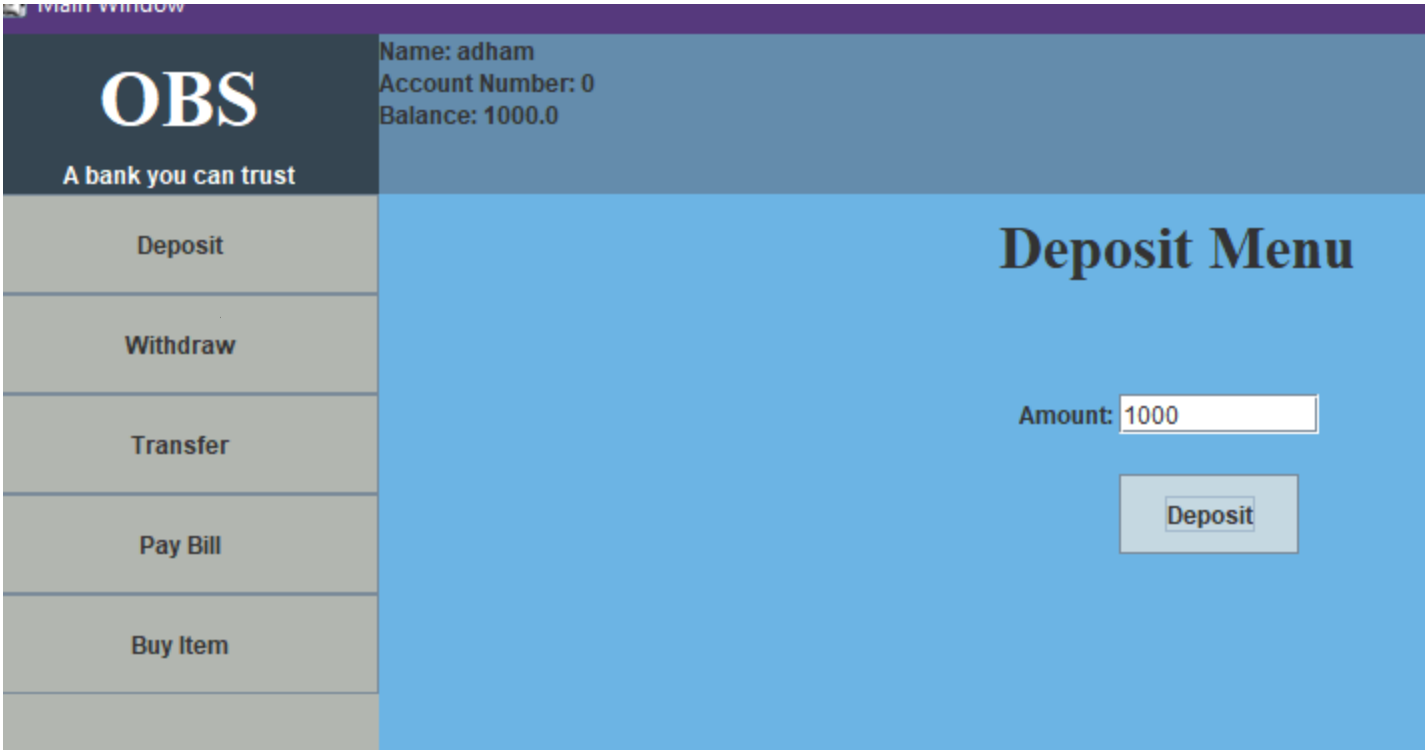
He can also log out of his account.

Testing deposit

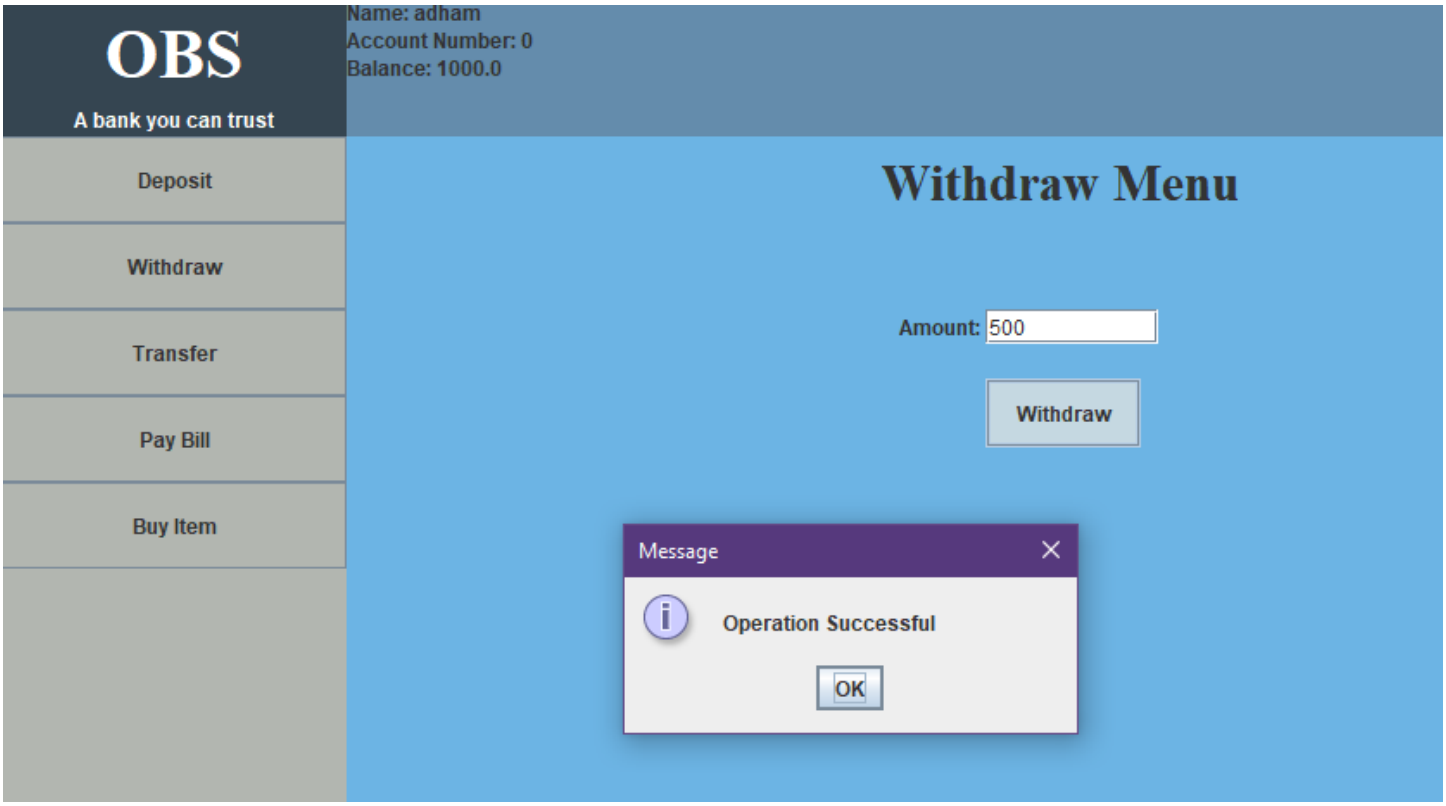




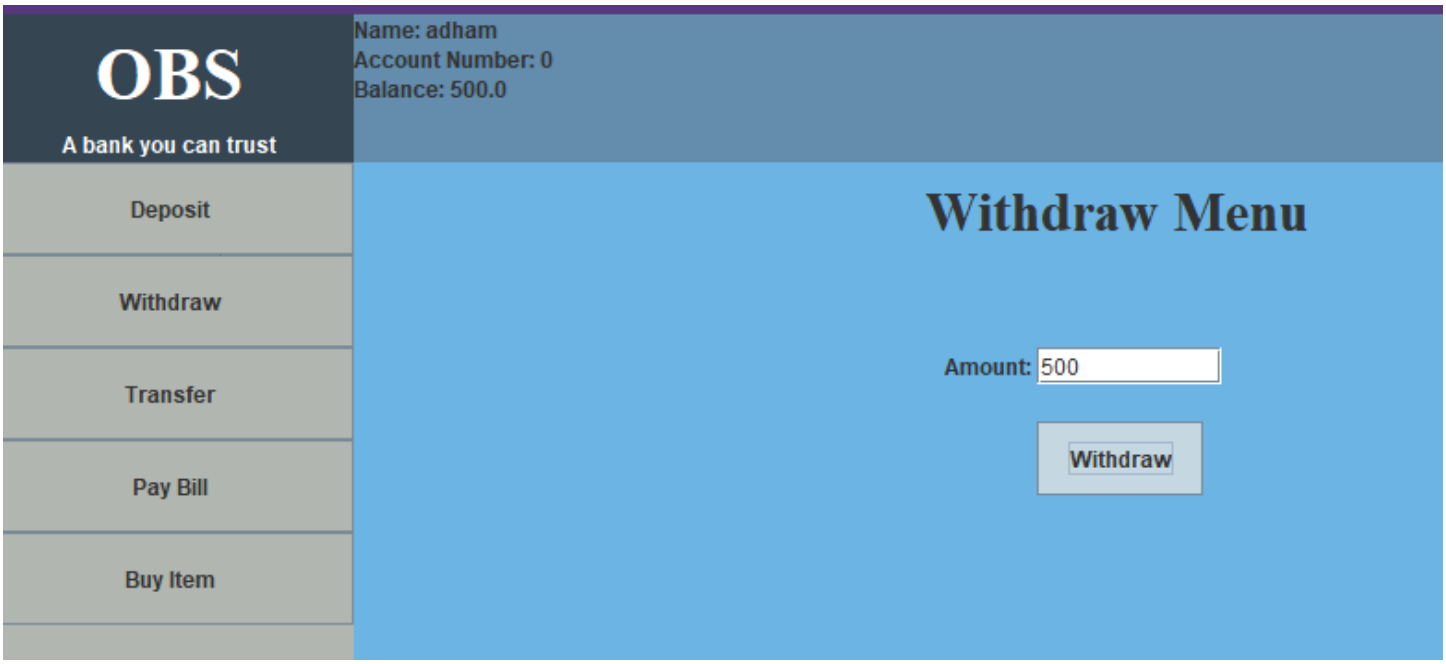
Success and balance info changes



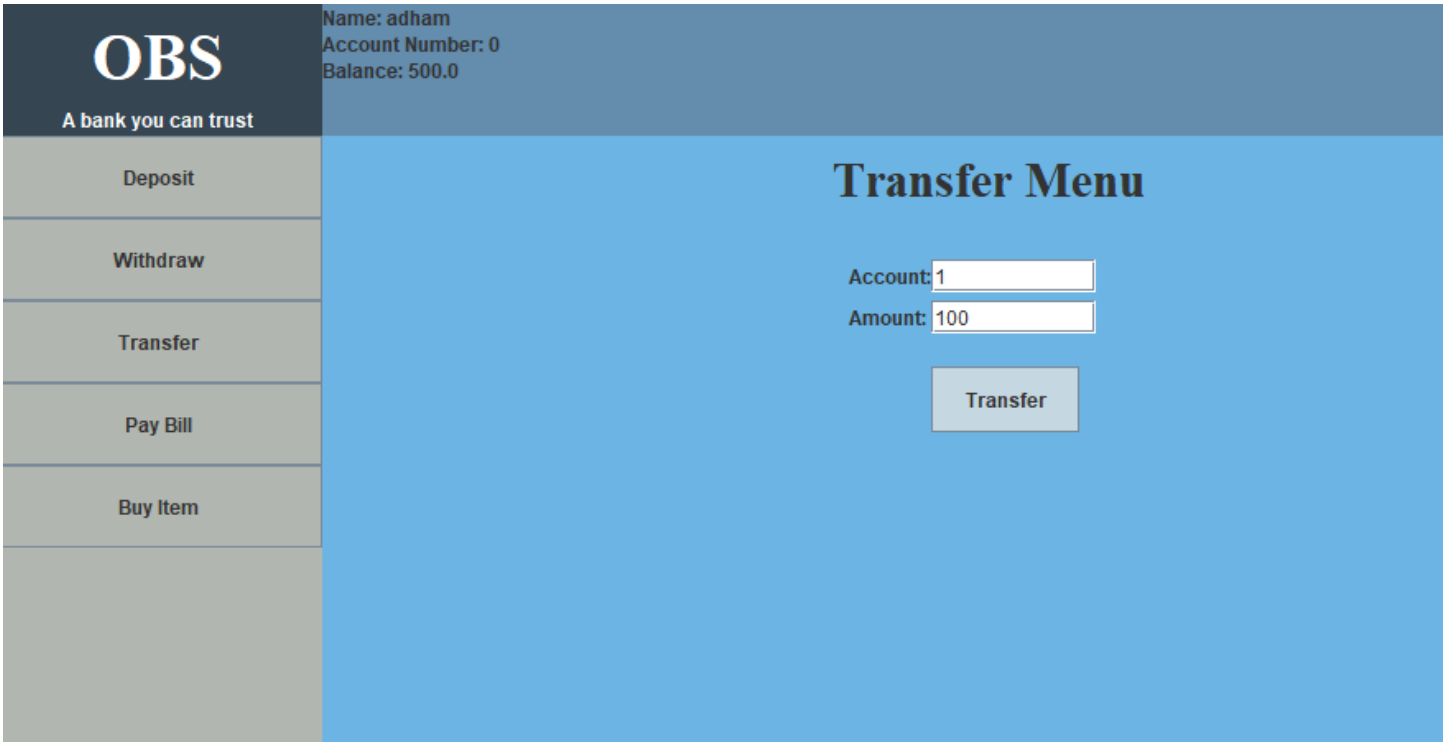
Testing withdraw

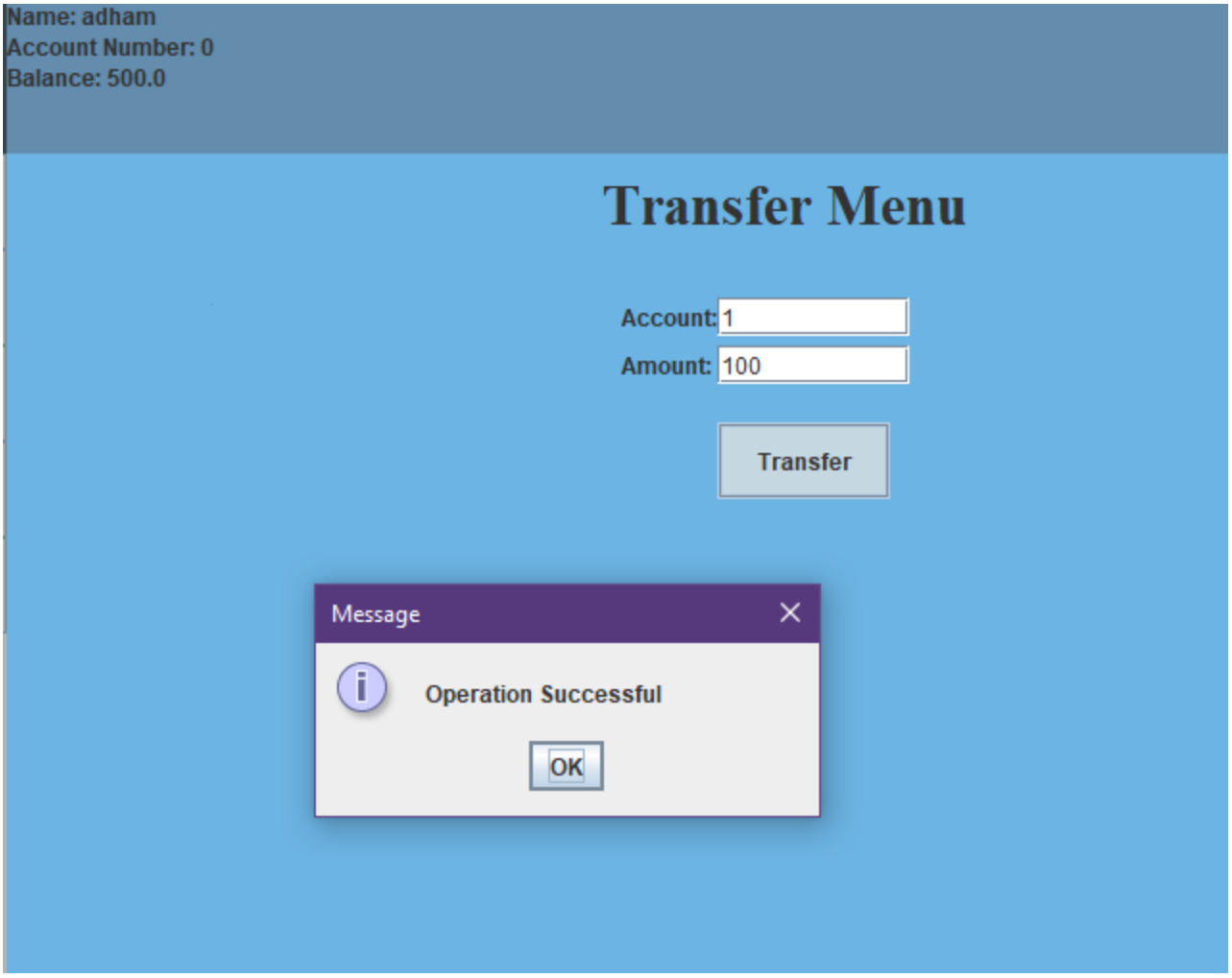


Success and balance changes



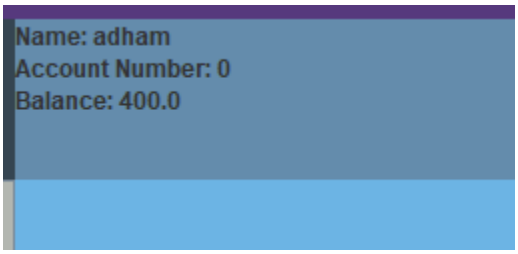
Testing transfer



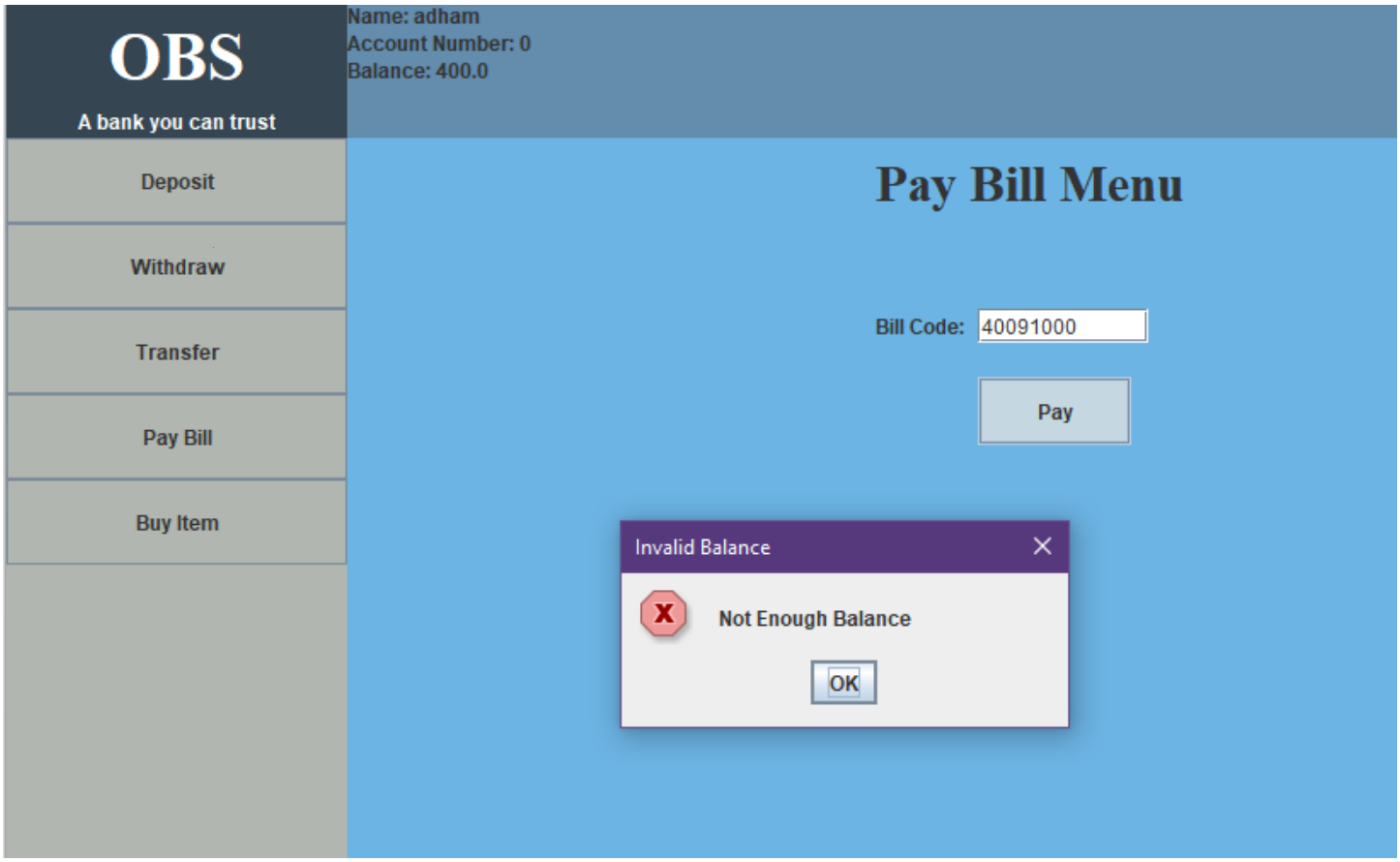


Successfully transfers 100 to accountNo 1 already created.

New balance

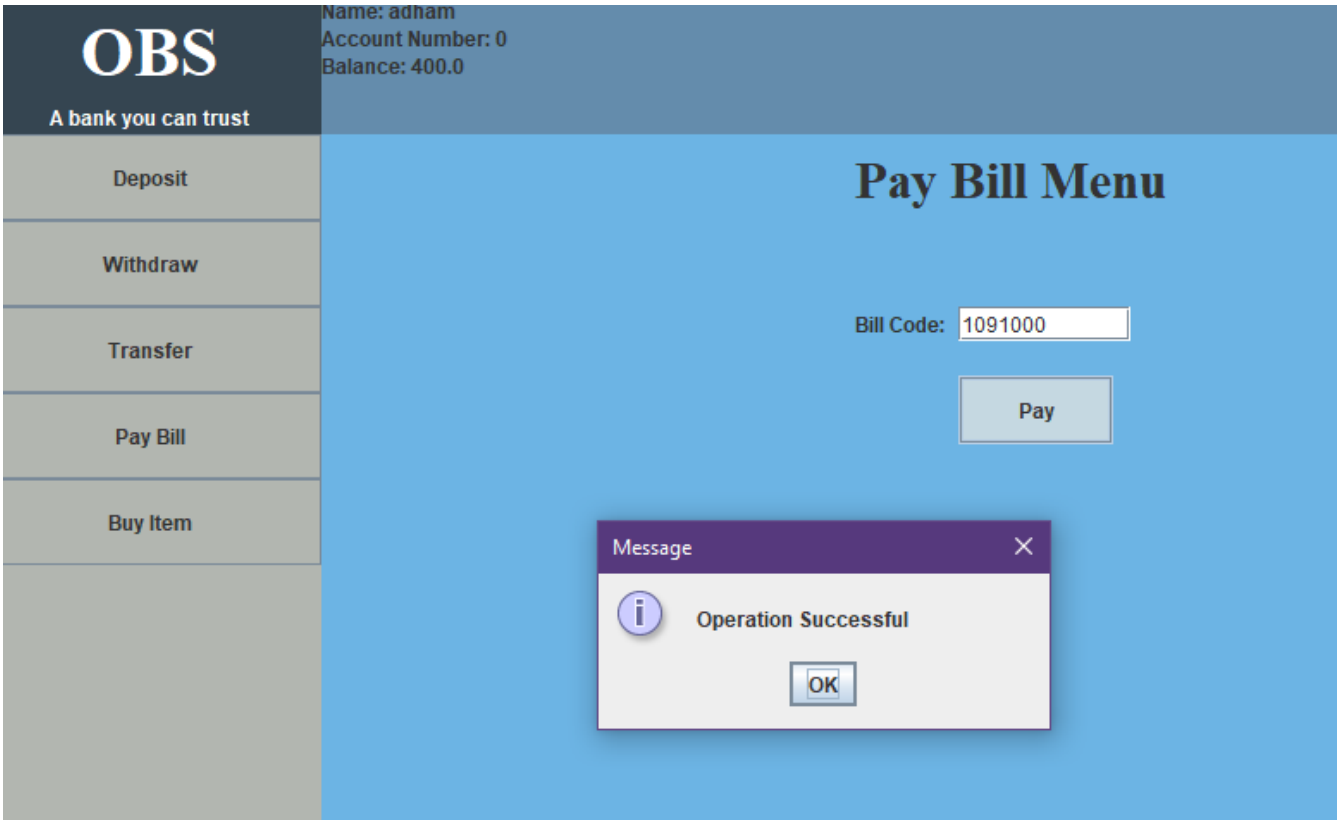


Testing pay pill

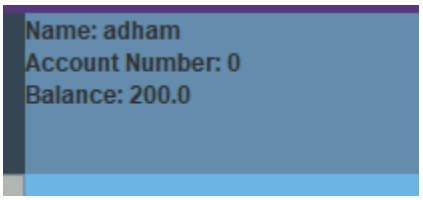


Not enough balance for the required bill code

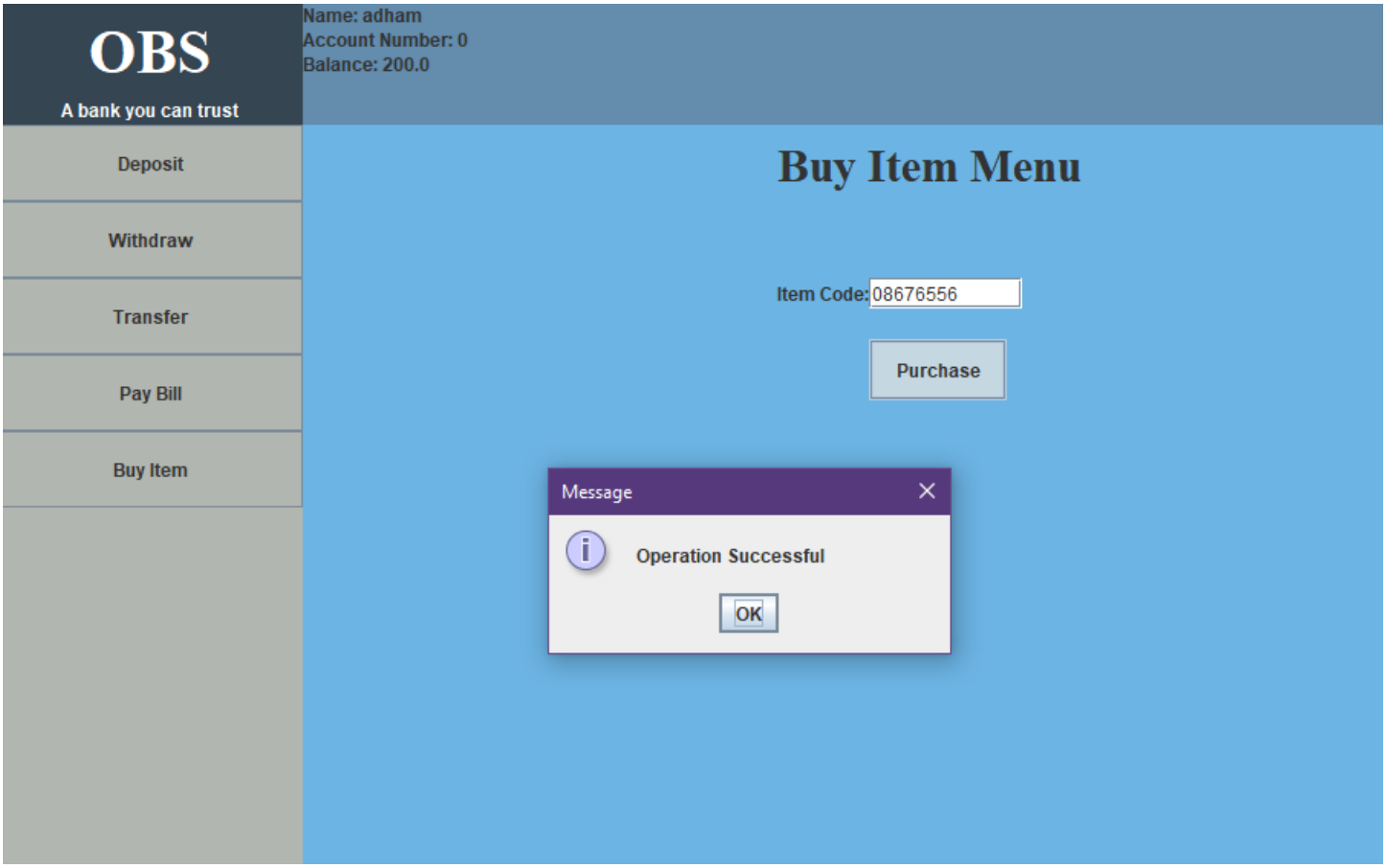
Successful if enough balance



New balance

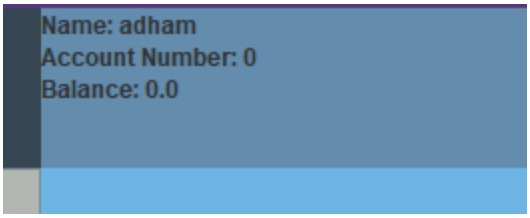


Testing buy item

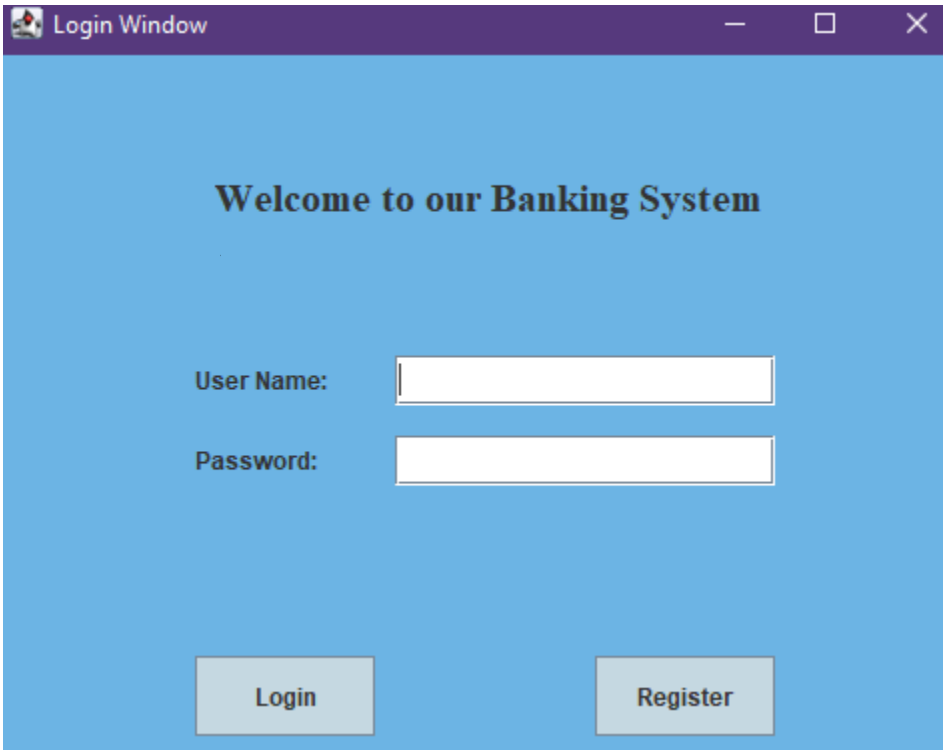


Success

Remaining balance



Testing logging out



Logged out successfully