Software Testing Project

Team Members:

* Yassin Khaled Mostafa Attia 19P9597
* Yara Mostafa Ibrahim 19p1120
* Gannah Allah Mohamed 19p9610
* Ziad Ashraf Ahmed Kasem 19p7095
* Seifeldin Sameh 19p3954
* Adham Ehab Salman Selim 19p4388

GitHub Repo:

https://github.com/DedRec/Testing\_project.git

Online Banking System

**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.

**Transaction**

**Client**

**account**

Client class:

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:**

Text

Description automatically generated

Graphical user interface, application

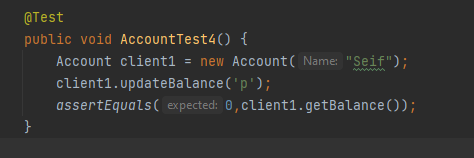
Description automatically generated

An account was created successfully

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

Text

Description automatically generated



Graphical user interface, text

Description automatically generated

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

**Client class test:**

Text

Description automatically generated

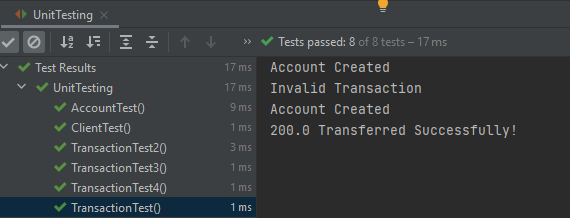
Graphical user interface, application

Description automatically generated

**An account was created successfully with the given information.**

**transaction test:**

Text

Description automatically generated 

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.

Text

Description automatically generated Graphical user interface, text, application, chat or text message

Description automatically generated

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

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

**Output:**

Text

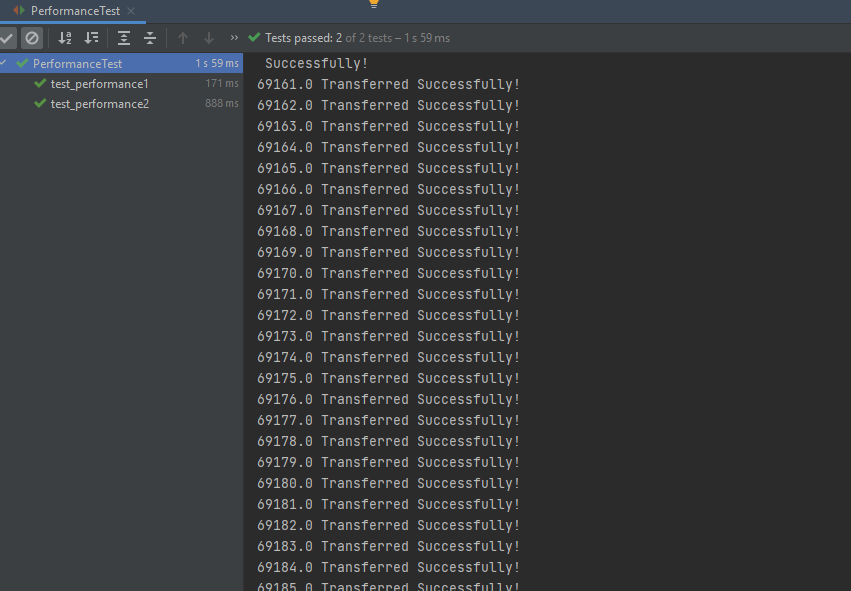
Description automatically generated

**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:**

**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  
 @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"));  
 }  
  
}

Text

Description automatically generated

**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();  
 *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"));  
 }  
}

Graphical user interface, text

Description automatically generated

**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;  
 }  
  
}

Text

Description automatically generated

**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:

A picture containing graphical user interface

Description automatically generated

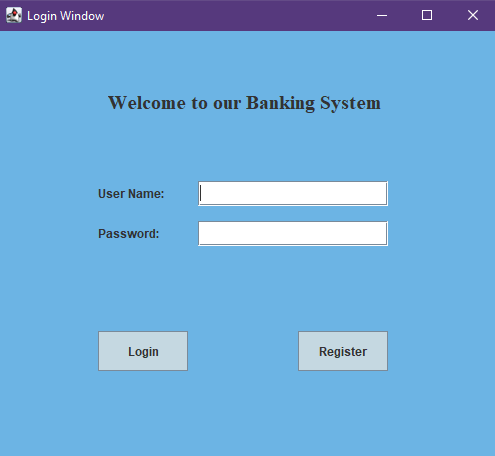
After modification:

A screenshot of a computer

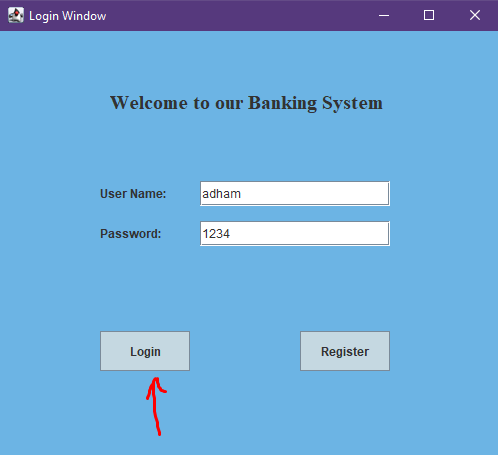
Description automatically generated with medium confidence

**Gui Testing**

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

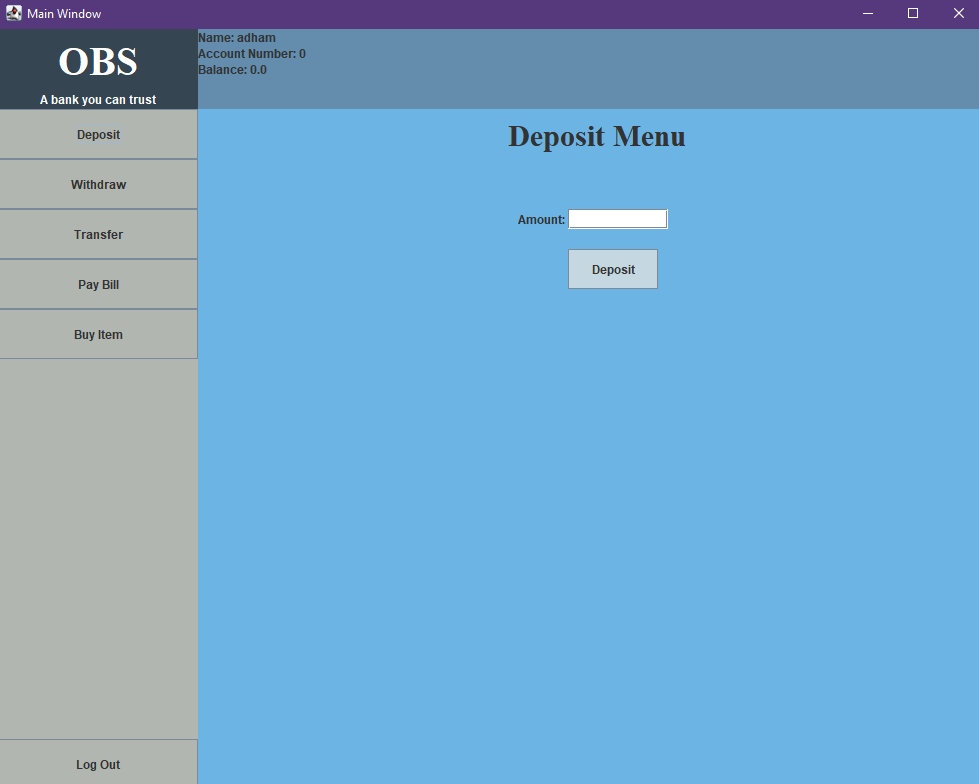


Example



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

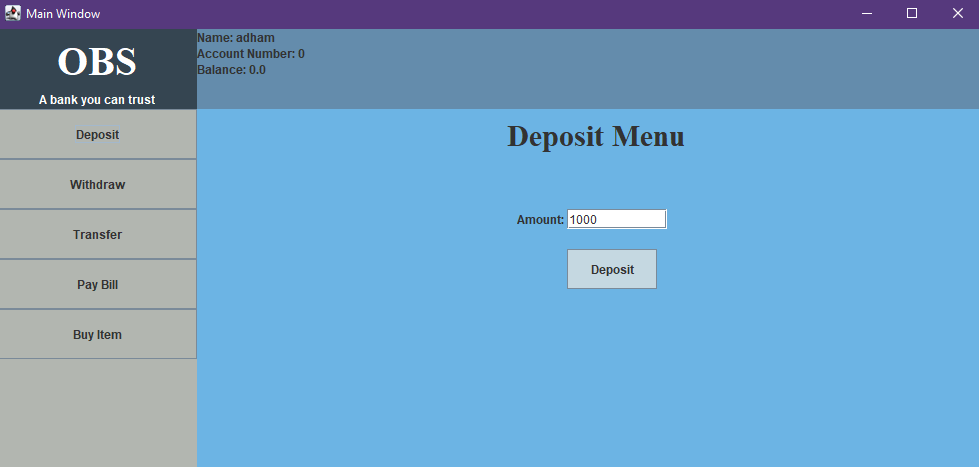


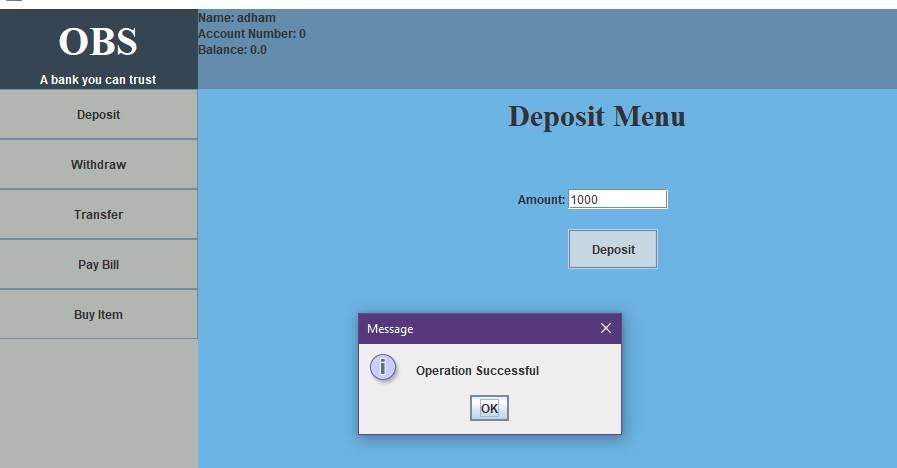


User can deposit or withdraw or transfer money, pay bill and buy an item

He can also log out of his account.

Testing deposit

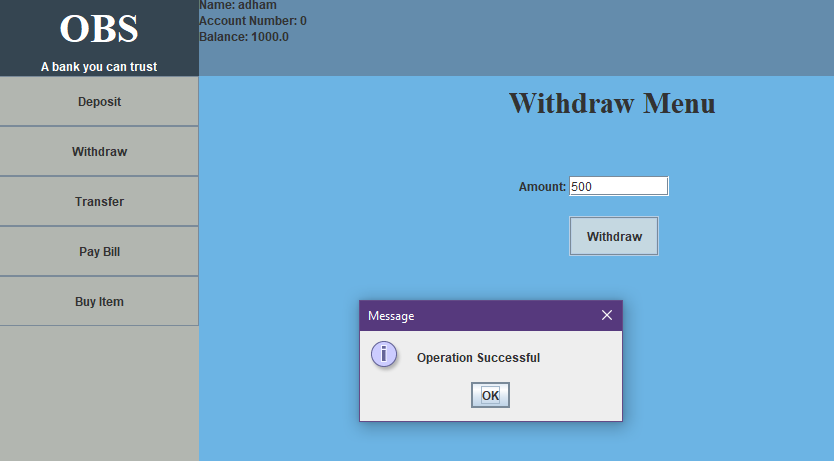




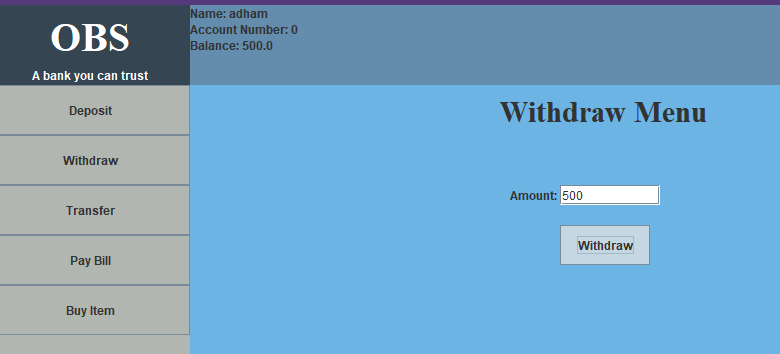
Success and balance info changes



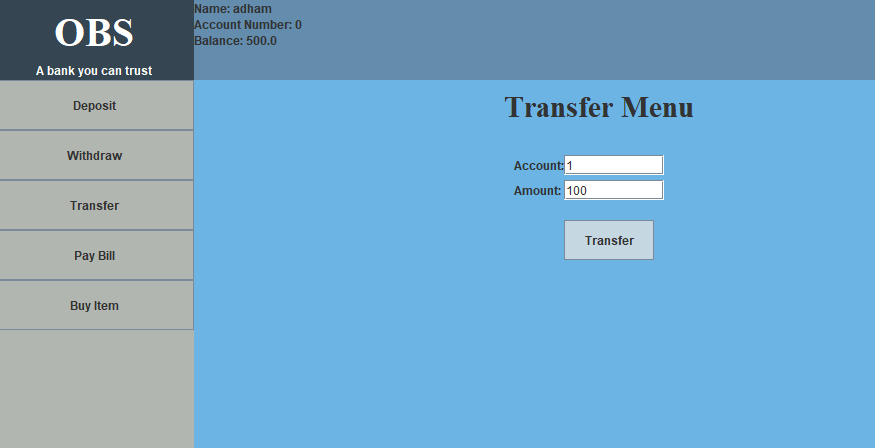
Testing withdraw

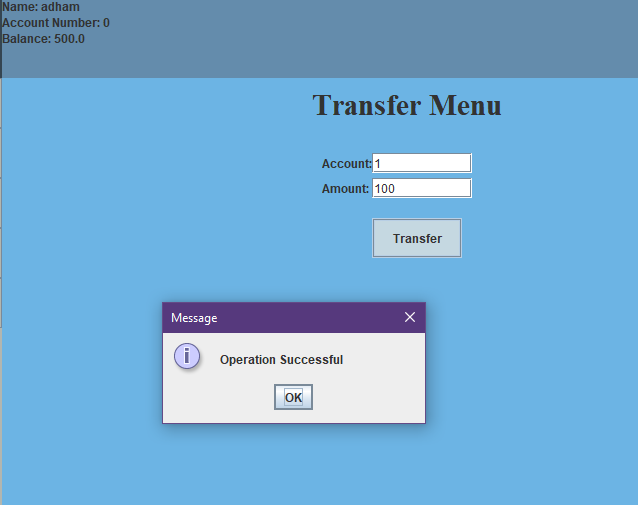


Success and balance changes



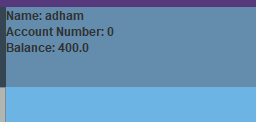
Testing transfer



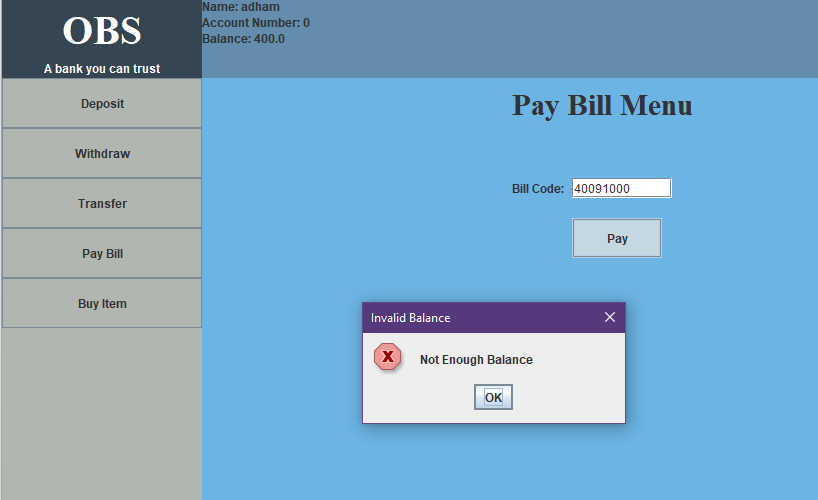


Successfully transfers 100 to accountNo 1 already created.

New balance

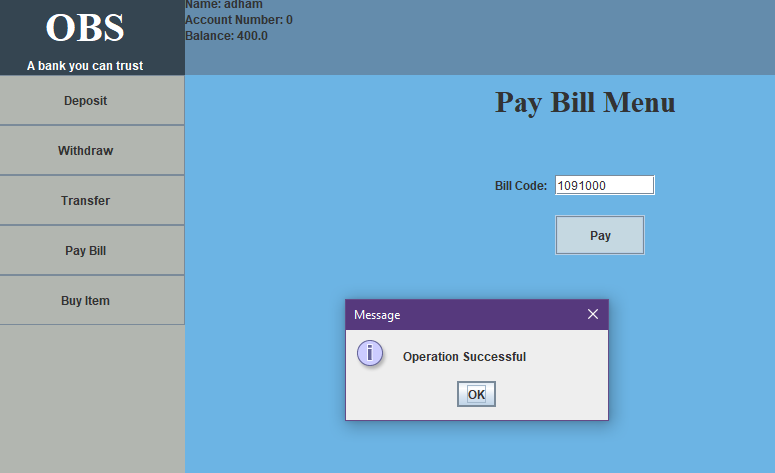


Testing pay pill

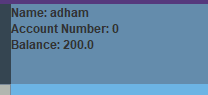


Not enough balance for the required bill code

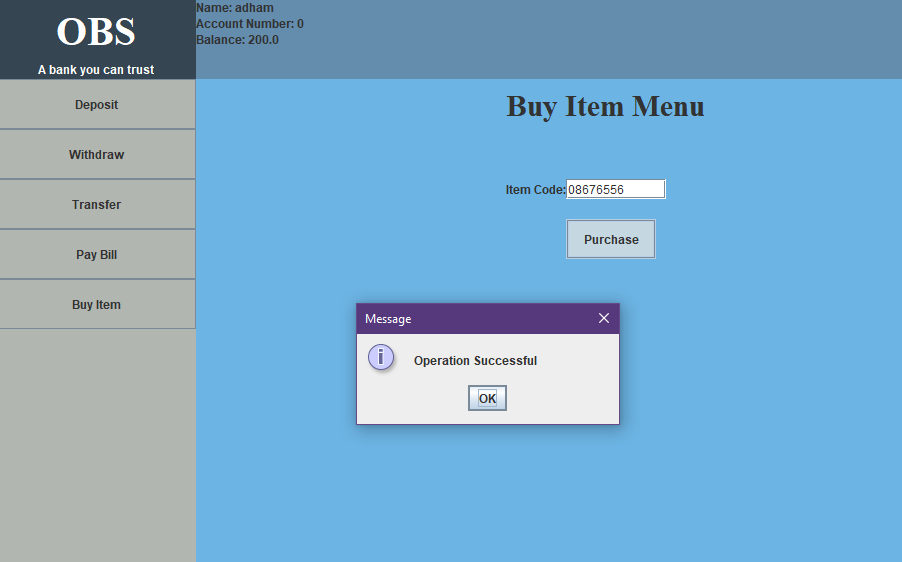
Successful if enough balance



New balance

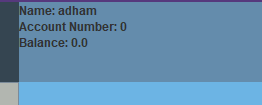


Testing buy item

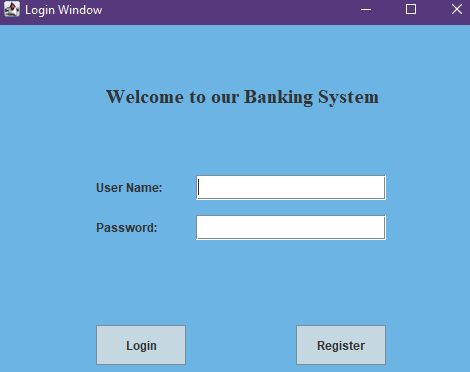


Success

Remaining balance



Testing logging out



Logged out successfully