CSE338: Software Testing, Validation, and Verification

CESS Spring 2022 Major Task: Banking Application

Name	ID
Salma Ihab Abdelmawgoud	19P8794
Noorhan Hatem Ibrahim Mohamed	19P5821
Abdelhay Nader Abdelhay Abouzayed	18P9033
Serag Eldin Mohamed Ali Mohamed Hussein	19P1183
Mohamed Mohamed Saeed Mohamed	16p6015

TABLE OF CONTENTS

Introduction

Our team has decided upon a banking desktop application, developed fully using Java. The functional requirements are roughly for the user to be able to log in, sign in, and use his account: deposit, withdraw, transfer, and purchase items using a credit card. He can also check his balance and view his bank statement anytime. Some non-functional requirements include but not limited to: fast response time for actions, no crashing when unusual action is taken. All this is done through an intuitive GUI. The application must be developed using Test-driven development (TDD) methodology and with refactoring concepts in mind, and there must be unit, integration, system, performance tests, and optionally more, in this project.
Test-driven development is a style of programming in which three activities are done at once: coding and testing (through writing unit tests) and design (through refactoring code). The developer is forced to consider the requirements of how the functions should act, before coding them. Note that the unit tests will be automated.

Unit and Integration Testing

TDD steps:
Step 1) Red: writing a test that fails
Step 2) Green: passing the test with as simple code as possible
Step 3) Refactor: removing simple code and refactoring it to fit ‘simple criteria’
The following are some screenshots showing the steps of writing our unit tests and basic application. We have 5 classes: Bank, Item, Transaction, UserAccount, CreditCard.

Bank Class

Initializing Test and Object Classes

```
 8  public class BankTest {
 9
10      @Before
11      public void setUp() throws Exception {
12      }
13
14      @Test
15      public void test() {
16          Bank bank = new Bank();
17      }
18
19  }
20
```

Bank cannot be resolved to a type

9 quick fixes available:

- Create class 'Bank'
- Create record 'Bank'
- Create interface 'Bank'
- Change to 'BankTest' (dummyBank)
- Change to 'Book' (java.awt.print)
- Create enum 'Bank'
- Add type parameter 'Bank' to 'BankTest'

Press 'F2' for focus

```
2
3  public class Bank {
4
5  }
6
```

Test 1: addAccount and Integration with UserAccount deposit function

Step 1: test fails

```
    @Test
    public void test() {
        Bank bank = new Bank();
        UserAccount user = new UserAccount();
        bank.addAccount(user);
    }

}
```

The method addAccount(UserAccount) is undefined for the type Bank

2 quick fixes available:

- Create method 'addAccount(UserAccount)' in type 'Bank'
- Add cast to 'bank'

Press 'F2' for focus

```
public class Bank {

    public void addAccount(UserAccount user) {
        // TODO Auto-generated method stub
        this.userlist.put(user.getUserName(),user);

    }

}
```

userlist cannot be resolved or is not a field

2 quick fixes available:

- Create field 'userlist' in type 'Bank'
- Create constant 'userlist' in type 'Bank'

Press 'F2' for focus

```java
    @Test
    public void test() {
        Bank bank = new Bank();
        UserAccount user = new UserAccount();
        user.setUserName("ahmed113");
        user.deposit(2000);
        bank.addAccount(user);
        assertEquals(2000,bank.getAccount("ahmed113").checkBalance())
    }

}
```

The method getAccount(String) is undefined for the type Bank

3 quick fixes available:

- Change to 'addAccount(..)'
- Create method 'getAccount(String)' in type 'Bank'
- Add cast to 'bank'

Press 'F2' for focus

```java
public class Bank {

    private Map<String,UserAccount> userlist;

    public void Bank() {
        userlist = new HashMap<>();
    }

    public void addAccount(UserAccount user) {
        // TODO Auto-generated method stub
        this.userlist.put(user.getUserName(),user);

    }

    public UserAccount getAccount(String string) {
        // TODO Auto-generated method stub
        return this.userlist.get(string);
    }

}
```

Step 2: test passes

Runs: 1/1 Errors: 0 Failures: 0

dummyBank.BankTest [Runner: JUnit 4] (0.002 s)

test (0.002 s)

```java
import static org.junit.Assert.*;

public class BankTest {

    @Before
    public void setUp() throws Exception {
    }

    @Test
    public void test() {
        Bank bank = new Bank();
        UserAccount user = new UserAccount();
        user.setUserName("ahmed113");
        user.deposit(2000);
        bank.addAccount(user);
        assertEquals(2000,bank.getAccount("ahmed113").checkBalance(),0);
    }

}
```

Test 2: Store Bank data in serializable

Step 1: test fails

```java
@Test
public void test2() {
    Bank bank = new Bank();
    UserAccount user = new UserAccount();
    Bank bank2= new Bank();
    user.setUserName("ahmed113");
    user.deposit(2000);
    bank.addAccount(user);
    bank.serializeDataOut();
    bank2.serializeDataIn();
}
```

```java
public void serializeDataOut() throws Exception {
    // TODO Auto-generated method stub
     File fileOne=new File("userlist");
     FileOutputStream fos=new FileOutputStream(fileOne);
     ObjectOutputStream oos=new ObjectOutputStream(fos);

     oos.writeObject(this.userlist);

     oos.flush();
     oos.close();
     fos.close();
}

public void serializeDataIn() throws Exception {
    // TODO Auto-generated method stub
     File toRead=new File("userlist");
     FileInputStream fis=new FileInputStream(toRead);
     ObjectInputStream ois=new ObjectInputStream(fis);

     this.userlist=(HashMap<String,UserAccount>)ois.readObject();

        ois.close();
        fis.close();


}
```

```java
@Test
public void test2() throws Exception {
    Bank bank = new Bank();
    UserAccount user = new UserAccount();
    Bank bank2= new Bank();
    user.setUserName("ahmed113");
    user.deposit(2000);
    bank.addAccount(user);
    bank.serializeDataOut();
    bank2.serializeDataIn();
    assertEquals(bank.getUserList(),bank2.getUserList());
}
```

Step 2: test passes

Runs: 2/2 Errors: 0 Failures: 0

dummyBank.BankTest [Runner: JUnit 4] (0.020 s)
- test (0.003 s)
- test2 (0.017 s)

Failure Trace

```java
import static org.junit.Assert.*;

public class BankTest {

    @Before
    public void setUp() throws Exception {
    }

    @Test
    public void test() {
        Bank bank = new Bank();
        UserAccount user = new UserAccount();
        user.setUserName("ahmed113");
        user.deposit(2000);
        bank.addAccount(user);
        assertEquals(2000,bank.getAccount("ahmed113").checkBalance(),0);
    }
    @Test
    public void test2() throws Exception {
        Bank bank = new Bank();
        UserAccount user = new UserAccount();
        Bank bank2= new Bank();
        user.setUserName("ahmed113");
        user.deposit(2000);
        bank.addAccount(user);
        bank.serializeDataOut();
        bank2.serializeDataIn();
        assertEquals(2000,bank2.getAccount("ahmed113").checkBalance(),0);
    }
```

Step 3: refactoring

```java
import static org.junit.Assert.*;

public class BankTest {
    Bank bank;
    Bank bank2;
    UserAccount user;

    @Before
    public void setUp() throws Exception {
        bank = new Bank();
        user = new UserAccount();
        user.setUserName("ahmed113");
        user.deposit(2000);
        bank.addAccount(user);
    }

    @Test
    public void test() {
        assertEquals(2000,bank.getAccount("ahmed113").checkBalance(),0);
    }
    @Test
    public void test2() throws Exception {
        bank2= new Bank();
        bank.serializeDataOut();
        bank2.serializeDataIn();
        assertEquals(2000,bank2.getAccount("ahmed113").checkBalance(),0);
    }
```

Item Class

Initializing Object Class Constructor

```
public class Item {

    public Item(int i) {
        // TODO Auto-generated
        this.price=i;
    }

}
```

Test 1: Setters and Getters

Step 1: setName and getPrice Red

```
    @Test
    public void test() {
        Item item = new Item(1000);
        item.setName("Pillow");

    }

}
```

The method setName(String) is undefined for the type Item

2 quick fixes available:

- Create method 'setName(String)' in type 'Item'
- Add cast to 'item'

Press 'F2' for focus

```
    }

    @Test
    public void test() {
        Item item = new Item(1000);
        item.setName("Pillow");
        assertEquals(1000,item.getPrice(),0)
    }

}
```

The method getPrice() is undefined for the type Item

2 quick fixes available:

- Create method 'getPrice()' in type 'Item'
- Add cast to 'item'

Press 'F2' for focus

Step 2: setName and getPrice Green

```java
public class Item {

    private double price;
    private String name;

    public Item(double i) {
        // TODO Auto-generated construc
        this.price=i;
    }

    public void setName(String string)
        // TODO Auto-generated method s
        this.name=string;

    }

    public double getPrice() {
        // TODO Auto-generated method s
        return this.price;
    }
```

Step 1: getName Red

```java
}

@Test
public void test() {
    Item item = new Item(1000);
    item.setName("Pillow");
    assertEquals(1000,item.getPrice(),0);
    assertEquals("Pillow",item.getName())
}
```

The method getName() is undefined for the type Item

3 quick fixes available:

- Change to 'getPrice(..)'
- Create method 'getName()' in type 'Item'
- Add cast to 'item'

Press 'F2' for focus

Step 2: getName Green

```java
public class Item {

    private double price;
    private String name;

    public Item(double i) {
        // TODO Auto-generated constructor stub
        this.price=i;
    }

    public void setName(String string) {
        // TODO Auto-generated method stub
        this.name=string;

    }

    public double getPrice() {
        // TODO Auto-generated method stub
        return this.price;
    }

    public String getName() {
        // TODO Auto-generated method stub
        return this.name;
    }



}
```

Note: sometimes, the refactoring step is unnecessary due to risk of over abstraction of code. In this example, we are just writing setters and getters to demonstrate how TDD may work. In the more complicated classes, we have made sure to refactor the functions to be more abstract and suitable for reuse.

Transaction Class

Test 1: Setters and Getters

Step 1: Red

```java
public class TransactionTest {

    @Before
    public void setUp() throws Exception {
    }

    @Test
    public void test() {
        Transaction t = new Transaction();
        t.setDescription();
        t.setType();
        t.setAmount();
        t.getDescription();
        t.getType();
        t.getAmount();
    }

}
```

Step 2: writing functions to pass + green test

```java
public class Transaction {
    enum TransactionType{
        DEPOSIT,WITHDRAWAL,PURCHASE;
    }

    private String description;
    private TransactionType type;
    private double amount;

    public void setDescription(String s) {
        // TODO Auto-generated method stub
        this.description=s;

    }

    public void setType(int i) {
        // TODO Auto-generated method stub
        switch(i) {
        case 0:
            this.type=TransactionType.DEPOSIT;
            break;
        case 1:
            this.type=TransactionType.WITHDRAWAL;
            break;
        case 2:
            this.type=TransactionType.PURCHASE;
            break;
        }



    }

    public void setAmount(double i) {
        // TODO Auto-generated method stub
        this.amount=i;

    }
```

dummyBank.TransactionTest [Runner: JUnit 4] (0.000 s)

```java
public class TransactionTest {

    @Before
    public void setUp() throws Exception {
    }

    @Test
    public void test() {
        Transaction t = new Transaction();
        t.setDescription("PAYROLL");
        t.setType(0);
        t.setAmount(100);
        assertEquals("PAYROLL",t.getDescription());
        assertEquals("DEPOSIT",t.getType().toString());
        assertEquals(100,t.getAmount(),0);
    }

}
```

Step 2: writing function to pass

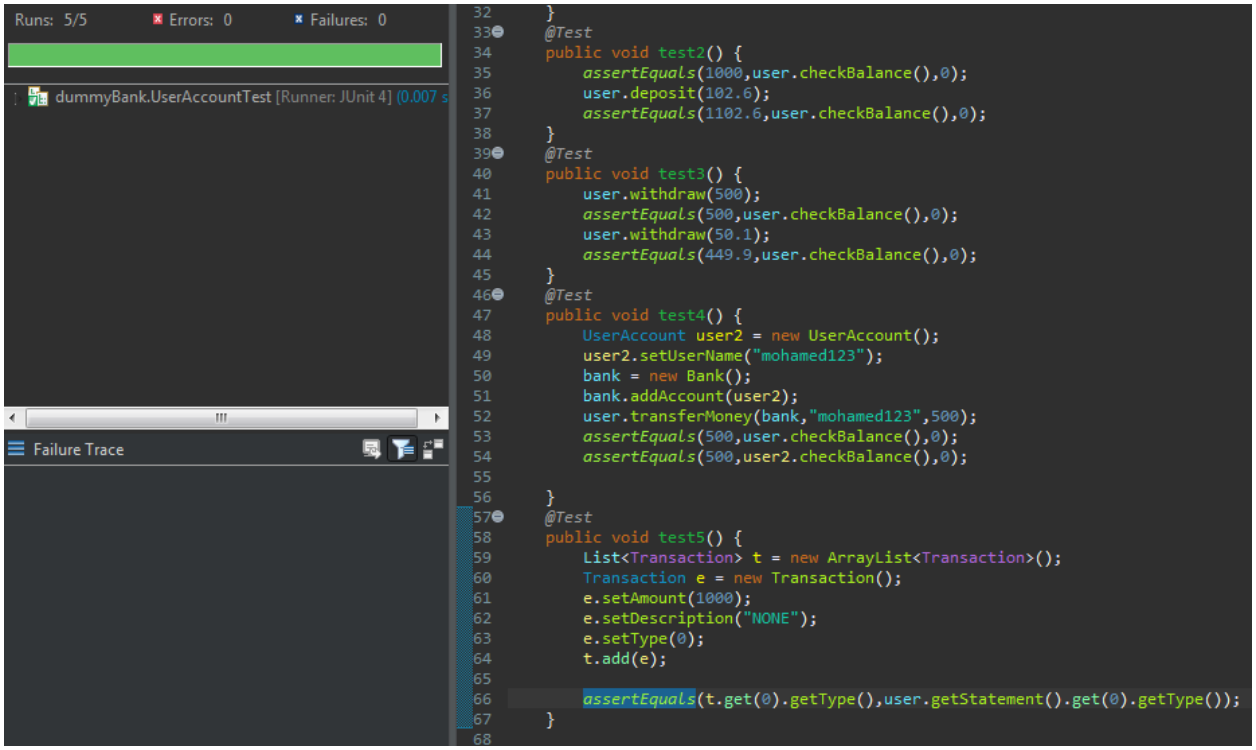
```java
    public Date getDate() {
        // TODO Auto-generated method stub
        return this.date;

    }
```

Step 2: Green

```java
@Test
public void test() {
    Transaction t = new Transaction();
    t.setDescription("PAYROLL");
    t.setType(0);
    t.setAmount(100);
    assertEquals("PAYROLL",t.getDescription());
    assertEquals("DEPOSIT",t.getType().toString());
    assertEquals(100,t.getAmount(),0);
    assertEquals(new Date(),new Transaction().getDate());
}
```

Step 3: refactoring passing Transaction functions (moving them to a more suitable class aka UserAccount)




```
    }
    @Test
    public void test2() {
        assertEquals(1000,user.checkBalance(),0);
        user.deposit(102.6);
        assertEquals(1102.6,user.checkBalance(),0);
    }
    @Test
    public void test3() {
        user.withdraw(500);
        assertEquals(500,user.checkBalance(),0);
        user.withdraw(50.1);
        assertEquals(449.9,user.checkBalance(),0);
    }
    @Test
    public void test4() {
        UserAccount user2 = new UserAccount();
        user2.setUserName("mohamed123");
        bank = new Bank();
        bank.addAccount(user2);
        user.transferMoney(bank,"mohamed123",500);
        assertEquals(500,user.checkBalance(),0);
        assertEquals(500,user2.checkBalance(),0);

    }
    @Test
    public void test5() {
        List<Transaction> t = new ArrayList<Transaction>();
        Transaction e = new Transaction();
        e.setAmount(1000);
        e.setDescription("NONE");
        e.setType(0);
        t.add(e);

        assertEquals(t.get(0).getType(),user.getStatement().get(0).getType());
    }
```

User Account Class

Test 1: Setters and Getters

Step 1: Red test class

```
public class UserAccountTest {

    @Before
    public void setUp() throws Exception {
    }

    @Test
    public void test() {
        UserAccount user = new UserAccount();
    }

}
```

UserAccount cannot be resolved to a type

8 quick fixes available:

- Create class 'UserAccount'
- Create record 'UserAccount'
- Create interface 'UserAccount'
- Change to 'UserAccountTest' (dummyBank)
- Create enum 'UserAccount'
- Add type parameter 'UserAccount' to 'UserAccountTest'
- Add type parameter 'UserAccount' to 'test()'

```
    @Test
    public void test() {
        UserAccount user = new UserAccount();
        user.setFirstName("ahmed");
    }

}
```

The method setFirstName(String) is undefined for the type UserAccount

2 quick fixes available:

- Create method 'setFirstName(String)' in type 'UserAccount'
- Add cast to 'user'

Step 2: writing simple code into object class to pass test

```
public class UserAccount {

    public void setFirstName(String string) {
        // TODO Auto-generated method stub

    }

}
```

```
public class UserAccount {

    public void setFirstName(String string) {
        // TODO Auto-generated method stub
        this.firstName=string;
    }

}
```

firstName cannot be resolved or is not a field

1 quick fix available:

- Create field 'firstName' in type 'UserAccount'

Step 1: Red getter

```
    @Test
    public void test() {
        UserAccount user = new UserAccount();
        user.setFirstName("ahmed");
        assertEquals("ahmed",user.getFirstName());
    }

}
```

The method getFirstName() is undefined for the type UserAccount

2 quick fixes available:

- Create method 'getFirstName()' in type 'UserAccount'
- Add cast to 'user'

Press 'F2' for focus

Step 2: writing getter to pass + Green

```java
public class UserAccount {

    private String firstName;

    public void setFirstName(String string) {
        // TODO Auto-generated method stub
        this.firstName=string;
    }

    public String getFirstName() {
        // TODO Auto-generated method stub
        return this.firstName;
    }

}
```

Finished after 0.041 seconds

Runs: 1/1 Errors: 0 Failures: 0

dummyBank.UserAccountTest [Runner: JUnit 4] (0.002 s

test (0.002 s)

```java
import static org.junit.Assert.*;

public class UserAccountTest {

    @Before
    public void setUp() throws Exception {
    }

    @Test
    public void test() {
        UserAccount user = new UserAccount();
        user.setFirstName("ahmed");
        assertEquals("ahmed",user.getFirstName());
    }

}
```

Step 1: Red setters

```java
public class UserAccountTest {

    @Before
    public void setUp() throws Exception {
    }

    @Test
    public void test() {
        UserAccount user = new UserAccount();
        user.setFirstName("ahmed");
        assertEquals("ahmed",user.getFirstName());
        user.setLastName("Mohamed");
        user.setUserName("ahmed113");
        user.setPassword("123456");
    }

}
```

Step 2: Green setters + Step 1: Red getters

```java
public class UserAccountTest {

    @Before
    public void setUp() throws Exception {
    }

    @Test
    public void test() {
        UserAccount user = new UserAccount();
        user.setFirstName("ahmed");
        assertEquals("ahmed",user.getFirstName());
        user.setLastName("Mohamed");
        user.setUserName("ahmed113");
        user.setPassword("123456");
        assertEquals("Mohamed",user.getLastName());
        assertEquals("ahmed113",user.getUserName());
        assertEquals("123445",user.getPassword());
    }

}
```

Step 2: writing getters + Green getters assertions

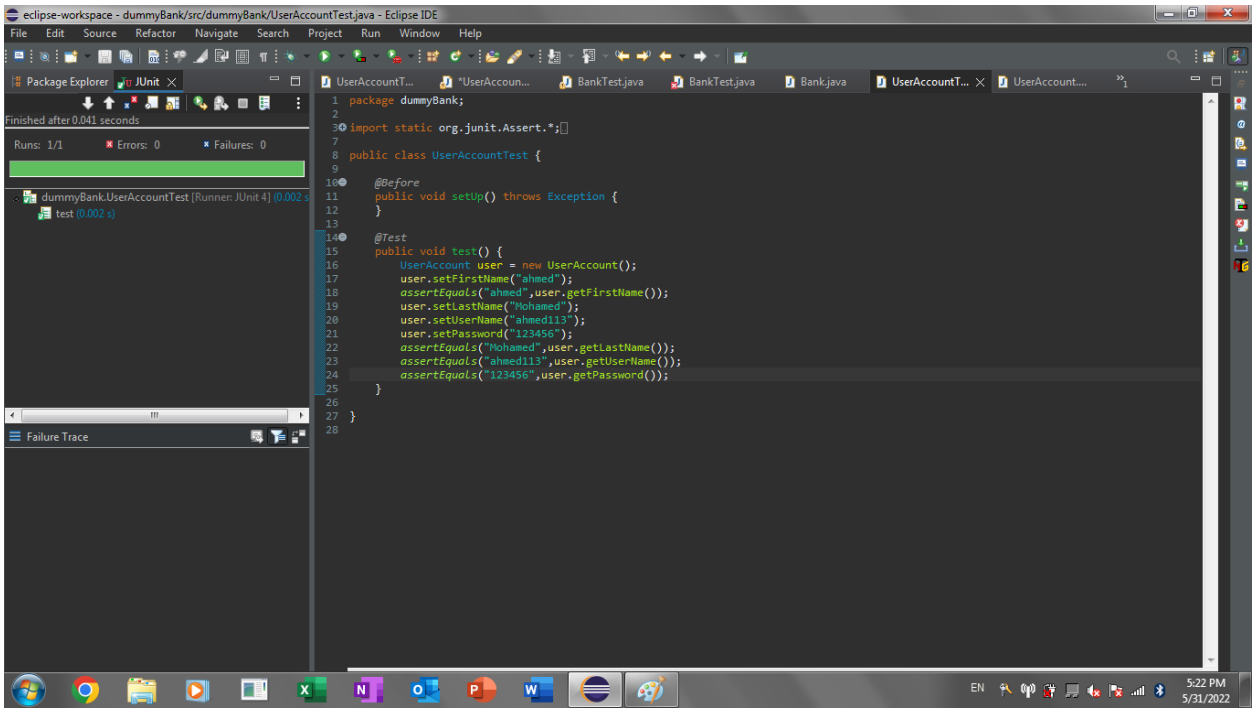
```java
    }

    public String getLastName() {
        // TODO Auto-generated method stub
        return this.lastName;
    }

    public String getUserName() {
        // TODO Auto-generated method stub
        return this.userName;
    }

    public String getPassword() {
        // TODO Auto-generated method stub
        return this.password;
    }

}
```



Test 2: Deposit and checkBalance

Step 1: Red deposit test

```
    @Test
    public void test2() {
        UserAccount user = new UserAccount();
        user.setFirstName("ahmed");
        user.setLastName("Mohamed");
        user.setUserName("ahmed113");
        user.setPassword("123456");
        user.deposit(1000);
    }

}
```

The method deposit(int) is undefined for the type UserAccount

2 quick fixes available:

- Create method 'deposit(int)' in type 'UserAccount'
- Add cast to 'user'

Press 'F2' for focus

Step 2: writing deposit function

```
    public void deposit(int i) {
        // TODO Auto-generated method stub
        this.balance+=i;

    }

}
```

balance cannot be resolved or is not a field

1 quick fix available:

- Create field 'balance' in type 'UserAccount'

Step 1: Red checkBalance function

```
    @Test
    public void test2() {
        UserAccount user = new UserAccount();
        user.setFirstName("ahmed");
        user.setLastName("Mohamed");
        user.setUserName("ahmed113");
        user.setPassword("123456");
        user.deposit(1000);
        asserEquals(1000,user.checkBalance());
    }

}
```

The method checkBalance() is undefined for the type UserAccount

2 quick fixes available:

- Create method 'checkBalance()' in type 'UserAccount'
- Add cast to 'user'

Press 'F2' for focus

Step 2 and 3: Green checkBalance test + separating (refactoring) assert functions

Runs: 2/2 Errors: 0 Failures: 0

dummyBank.UserAccountTest [Runner: JUnit 4] (0.002 s)

test (0.002 s)

test2 (0.000 s)

Failure Trace

```java
import static org.junit.Assert.*;

public class UserAccountTest {

    @Before
    public void setUp() throws Exception {
    }

    @Test
    public void test() {
        UserAccount user = new UserAccount();
        user.setFirstName("ahmed");
        assertEquals("ahmed",user.getFirstName());
        user.setLastName("Mohamed");
        user.setUserName("ahmed113");
        user.setPassword("123456");
        assertEquals("Mohamed",user.getLastName());
        assertEquals("ahmed113",user.getUserName());
        assertEquals("123456",user.getPassword());
    }
    @Test
    public void test2() {
        UserAccount user = new UserAccount();
        user.setFirstName("ahmed");
        user.setLastName("Mohamed");
        user.setUserName("ahmed113");
        user.setPassword("123456");
        user.deposit(1000);
        assertEquals(1000,user.checkBalance(),0);
    }
```

Runs: 2/2 Errors: 0 Failures: 0

dummyBank.UserAccountTest [Runner: JUnit 4] (0.002 s

Failure Trace

```java
import static org.junit.Assert.*;

public class UserAccountTest {
    UserAccount user;

    @Before
    public void setUp() throws Exception {
        user = new UserAccount();
        user.setFirstName("ahmed");
        assertEquals("ahmed",user.getFirstName());
        user.setLastName("Mohamed");
        user.setUserName("ahmed113");
        user.setPassword("123456");
        user.deposit(1000);
    }

    @Test
    public void test() {
        assertEquals("Mohamed",user.getLastName());
        assertEquals("ahmed113",user.getUserName());
        assertEquals("123456",user.getPassword());
    }
    @Test
    public void test2() {
        assertEquals(1000,user.checkBalance(),0);
        user.deposit(102.6);
        assertEquals(1102.6,user.checkBalance(),0);
    }
```

Test 3: Withdraw

Step 1: Red withdraw test

```
    @Test
    public void test3() {
        user.withdraw(500);
    }

}
```

The method withdraw(int) is undefined for the type UserAccount

2 quick fixes available:

- Create method 'withdraw(int)' in type 'UserAccount'
- Add cast to 'user'

Press 'F2' for focus

Step 2: Green withdraw test + wrote withdraw function

dummyBank.UserAccountTest [Runner: JUnit 4] (0.003 s)

- test (0.002 s)
- test2 (0.001 s)
- test3 (0.000 s)

Failure Trace

```
@Before
public void setUp() throws Exception {
    user = new UserAccount();
    user.setFirstName("ahmed");
    assertEquals("ahmed",user.getFirstName());
    user.setLastName("Mohamed");
    user.setUserName("ahmed113");
    user.setPassword("123456");
    user.deposit(1000);
}

@Test
public void test() {
    assertEquals("Mohamed",user.getLastName());
    assertEquals("ahmed113",user.getUserName());
    assertEquals("123456",user.getPassword());
}
@Test
public void test2() {
    assertEquals(1000,user.checkBalance(),0);
    user.deposit(102.6);
    assertEquals(1102.6,user.checkBalance(),0);
}
@Test
public void test3() {
    user.withdraw(500);
    assertEquals(500,user.checkBalance(),0);
    user.withdraw(50.1);
    assertEquals(449.9,user.checkBalance(),0);
}
```

```
public void withdraw(double i) {
    // TODO Auto-generated method stub
    this.balance-=i;

}
```

Test 4: transferMoney

Step 1: Red transferMoney function

```
    @Test
    public void test4() {
        UserAccount user2 = new UserAccount();
        user2.setUserName("mohamed123");
        Bank bank = new Bank();
        bank.addAccount(user2);
        user.transferMoney(bank,"mohamed123",500);

    }
```

Step 2: Writing transferMoney function

```java
    public void transferMoney(Bank bank, String string, double i) {
        // TODO Auto-generated method stub
        this.balance-=i;
        bank.getAccount(string).deposit(i);

    }

}
```

Runs: 4/4 Errors: 0 Failures: 0

dummyBank.UserAccountTest [Runner: JUnit 4] (0.003 s

test (0.001 s)

test2 (0.000 s)

test3 (0.000 s)

test4 (0.001 s)

Failure Trace

```java
        user.setFirstName("ahmed");
        assertEquals("ahmed",user.getFirstName());
        user.setLastName("Mohamed");
        user.setUserName("ahmed113");
        user.setPassword("123456");
        user.deposit(1000);
    }

    @Test
    public void test() {
        assertEquals("Mohamed",user.getLastName());
        assertEquals("ahmed113",user.getUserName());
        assertEquals("123456",user.getPassword());
    }
    @Test
    public void test2() {
        assertEquals(1000,user.checkBalance(),0);
        user.deposit(102.6);
        assertEquals(1102.6,user.checkBalance(),0);
    }
    @Test
    public void test3() {
        user.withdraw(500);
        assertEquals(500,user.checkBalance(),0);
        user.withdraw(50.1);
        assertEquals(449.9,user.checkBalance(),0);
    }
    @Test
    public void test4() {
        UserAccount user2 = new UserAccount();
        user2.setUserName("mohamed123");
        Bank bank = new Bank();
        bank.addAccount(user2);
        user.transferMoney(bank,"mohamed123",500);
        assertEquals(500,user.checkBalance(),0);
        assertEquals(500,user2.checkBalance(),0);

    }
```

Test 5: Integration Test with Transaction class

Runs: 5/5 Errors: 0 Failures: 0

dummyBank.UserAccountTest [Runner: JUnit 4] (0.007 s

Failure Trace

```java
    }
    @Test
    public void test2() {
        assertEquals(1000,user.checkBalance(),0);
        user.deposit(102.6);
        assertEquals(1102.6,user.checkBalance(),0);
    }
    @Test
    public void test3() {
        user.withdraw(500);
        assertEquals(500,user.checkBalance(),0);
        user.withdraw(50.1);
        assertEquals(449.9,user.checkBalance(),0);
    }
    @Test
    public void test4() {
        UserAccount user2 = new UserAccount();
        user2.setUserName("mohamed123");
        bank = new Bank();
        bank.addAccount(user2);
        user.transferMoney(bank,"mohamed123",500);
        assertEquals(500,user.checkBalance(),0);
        assertEquals(500,user2.checkBalance(),0);

    }
    @Test
    public void test5() {
        List<Transaction> t = new ArrayList<Transaction>();
        Transaction e = new Transaction();
        e.setAmount(1000);
        e.setDescription("NONE");
        e.setType(0);
        t.add(e);

        assertEquals(t.get(0).getType(),user.getStatement().get(0).getType());
    }
```

Test 6: Integration Test with Item Class and purchases function

Step 1: Red Item object instantiation test

```java
    @Test
    public void test6() {
        Item item = new Item();
        user.purchase(item)
    }
```

Step 2: Green Item + Step 1: purchases function

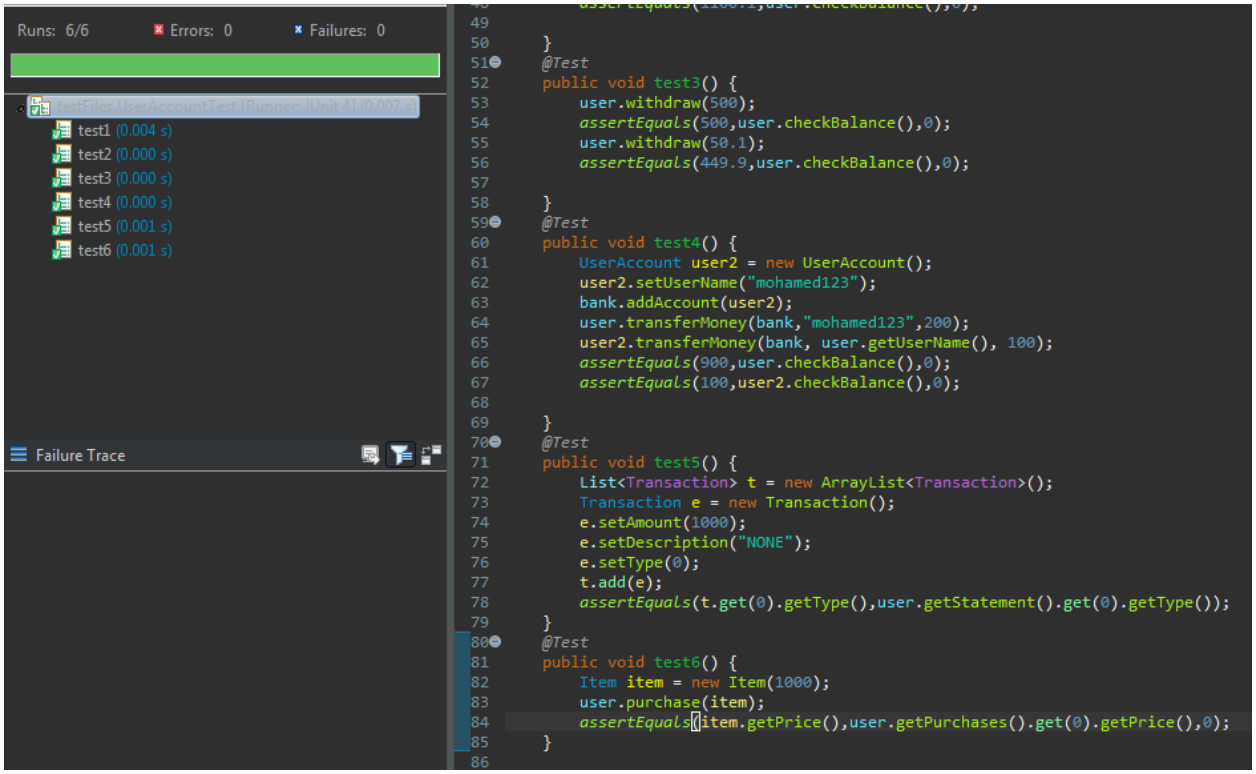
```java
    @Test
    public void test6() {
        Item item = new Item(1000);
        user.purchase(item);
        assertEquals(item.getPrice(),user.getPurchases())
    }
```

Step 2: Writing getPurchases

```java
    public List<Item> getPurchases() {
        // TODO Auto-generated method stub
        return this.purchases;
    }

}
```

All tests now pass…



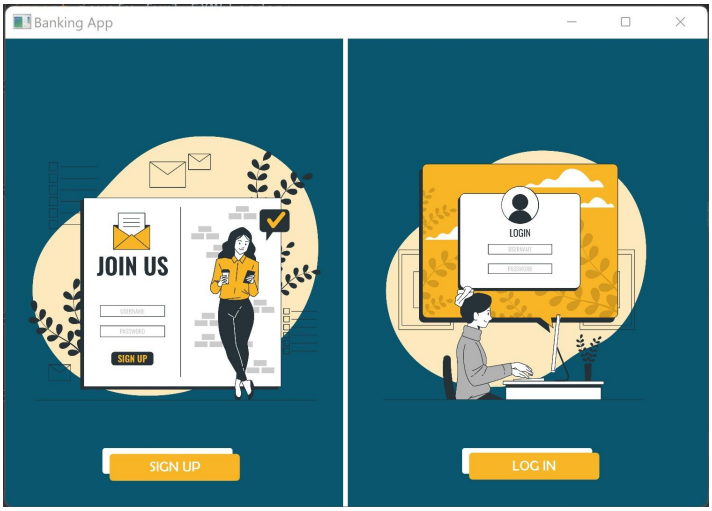
GUI Testing

This kind of testing can be done manually or automated using any testing framework such as TestFX. Our TestFX code is now redundant since it was decided that manual tests were easier to run without many dependencies or libraries and easier for anyone to replicate. Regardless, here are some manually tested cases.

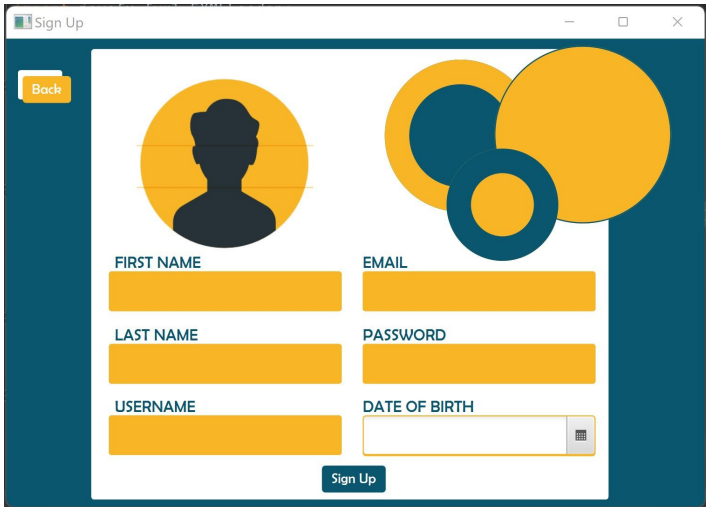
- Ensure buttons work as intended.
- Ensure error messages pop up when incorrect action taken.
- Ensure buttons that cause window to change work.

These test cases ensure our GUI works well in general.

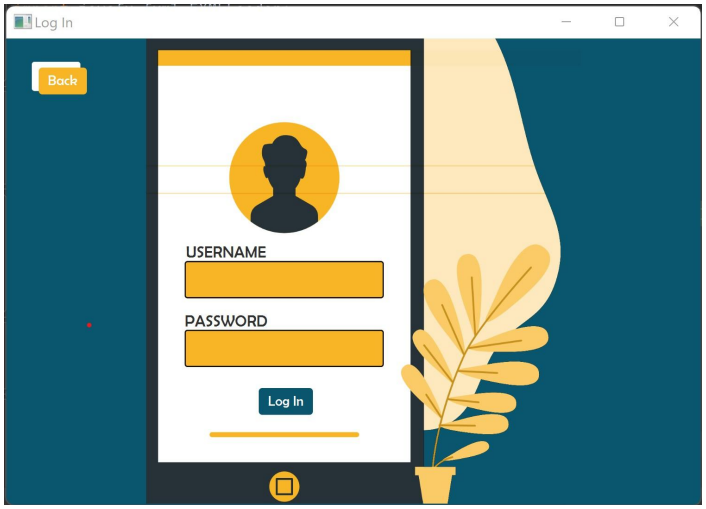
First GUI Window



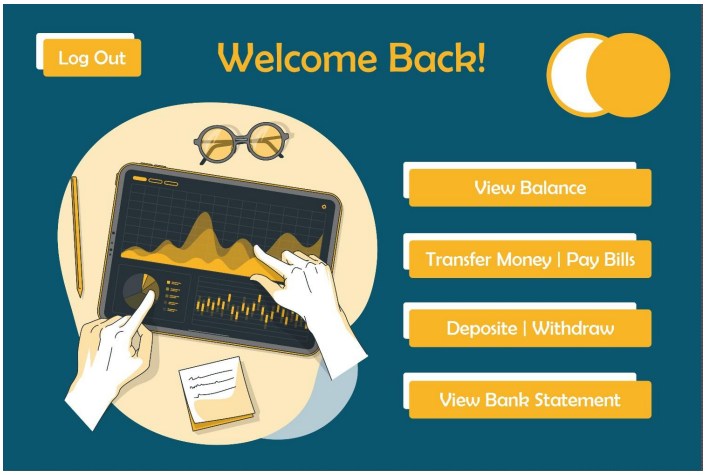
SignUp Window



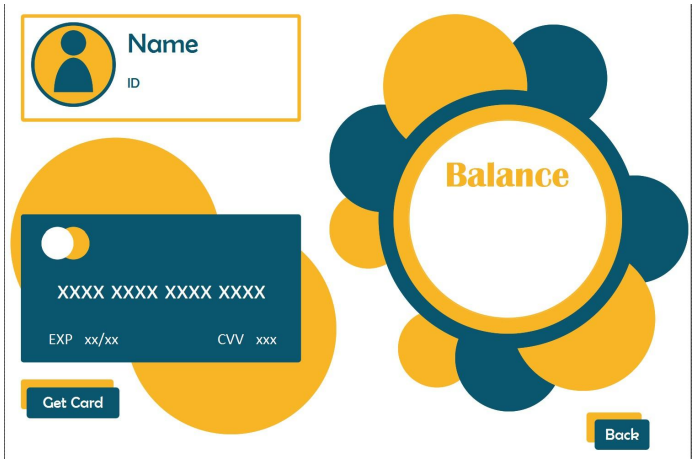
Login Window



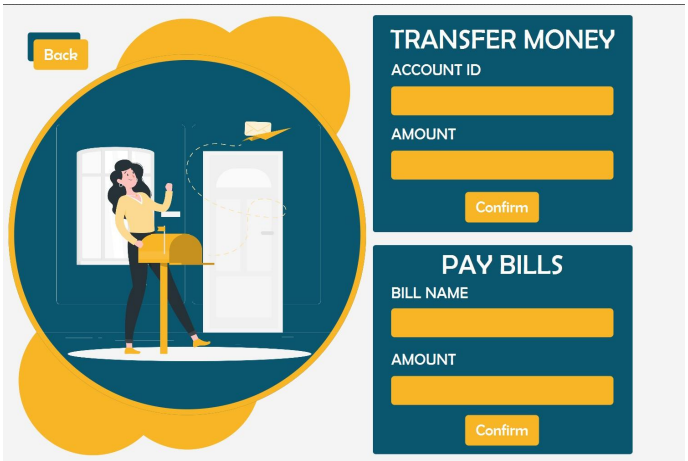
Account Dashboard After Login



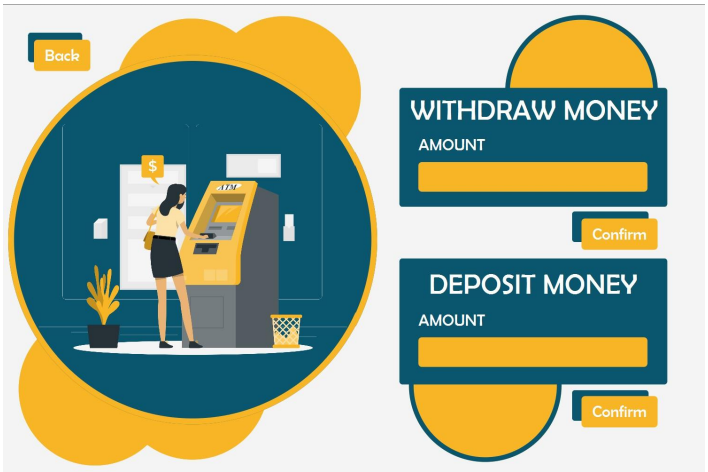
View Balance (This is the scenebuilder view. When the app runs, “Name” and“ID” will be replaced with the user’s name and ID respectively. His/Her balance amount will appear under Balance inside the circle on the right and his/her credit card information will appear in the card on the left.)



Transfer Money | Pay Bills



Deposit | Withdraw

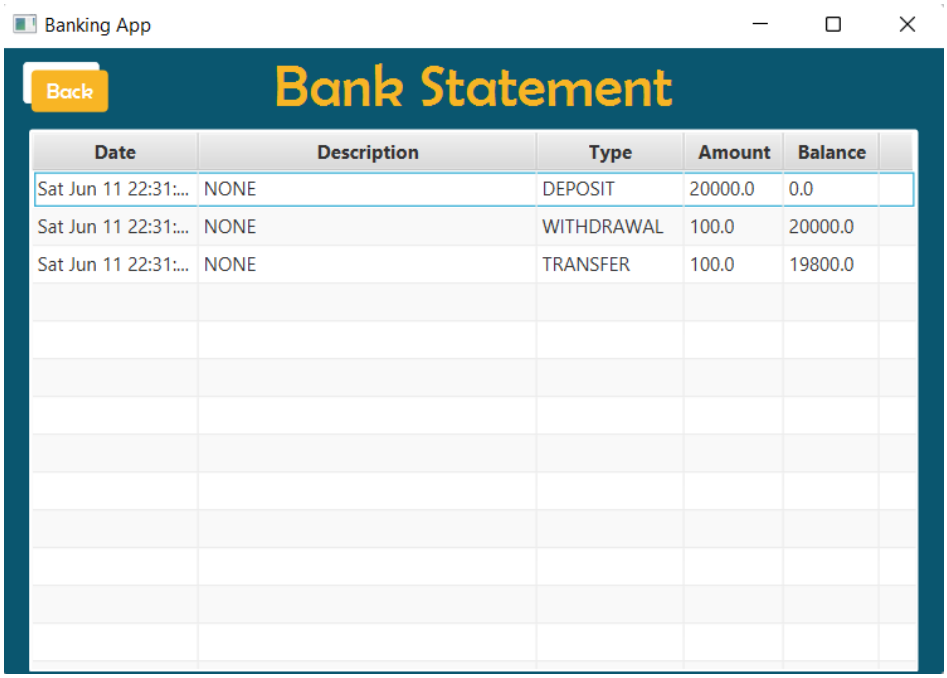


View Bank Statement (will be filled by the logged in user's transactions)

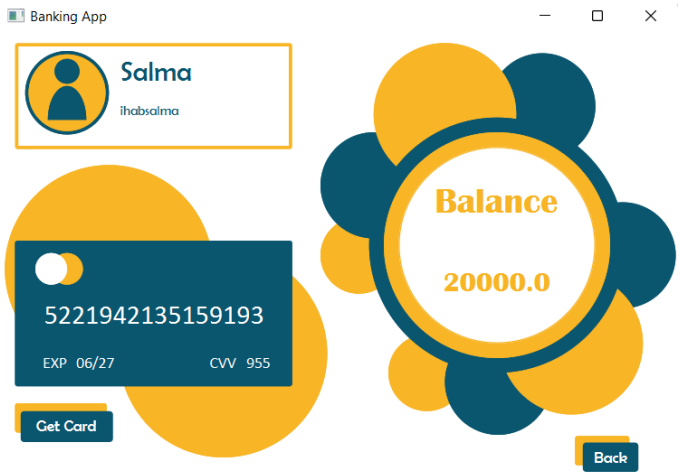
System Testing

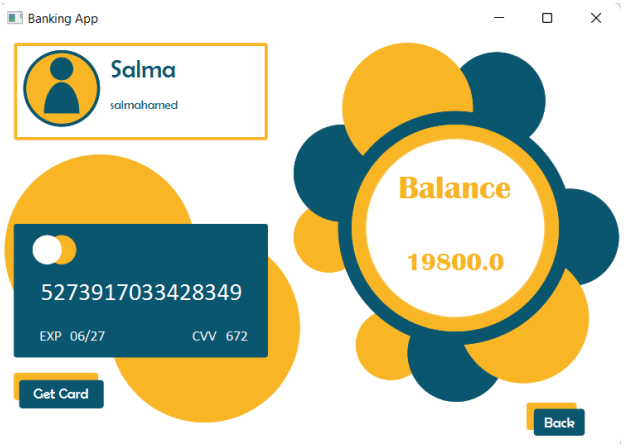
System testing tests the whole software end-to-end. Thus, we now need to run the project and check for some real-life scenarios.

- After signing up for an account, we can log in and out of the account more than once. (home screen->go to online bank->sign up-> enter details and click sign up-> enter details and log in-> back-> login-> enter details and login-> back-> login -> enter details and login)
- After logging in, we can deposit 20000$, withdraw 100$, transfer 100$, and check bank account for the remaining balance equal 19800$.



- After checking balance, we can log out and back in with no problem.
- After signing out, we can sign up for a new account, and log out and into our old account.





Performance Testing

Performance testing is the practice of testing the quality of capability of a software. It involves non-functional testing of the system. This extends to throughput, response time, latency and scalability.

To simulate load testing, we will create many accounts in the one running instance, and see how the program behaves. We can also use the IntelliJ performance plugin to check start-up as follows.

Startup Time Cost per Plugin

Plugin	Startup Time (ms)
Kotlin	25025
Android	18410
Code With Me	14398
Package Search	8712
Database Navigator	7180
Maven	6599
IntelliLang	4034
Space	3452
Configuration Script	2592
Groovy	2577
Shared Indexes	1708
WSL File System Support	1185
Windows 10 Light Theme	1175
Git	1118
Grazie	1116
EditorConfig	1034
Markdown	953
GitHub	867
Machine Learning Code Com...	820
Task Management	779
Ant	765
Gradle	764
Plugin DevKit	703
Lombok	629
IDE Features Trainer	562
JUnit	543
Mercurial	537
XPathView + XSLT	517
Properties	473
JavaFX	467
Gradle-Java	426
Subversion	378

Disable Selected Plugins OK Cancel

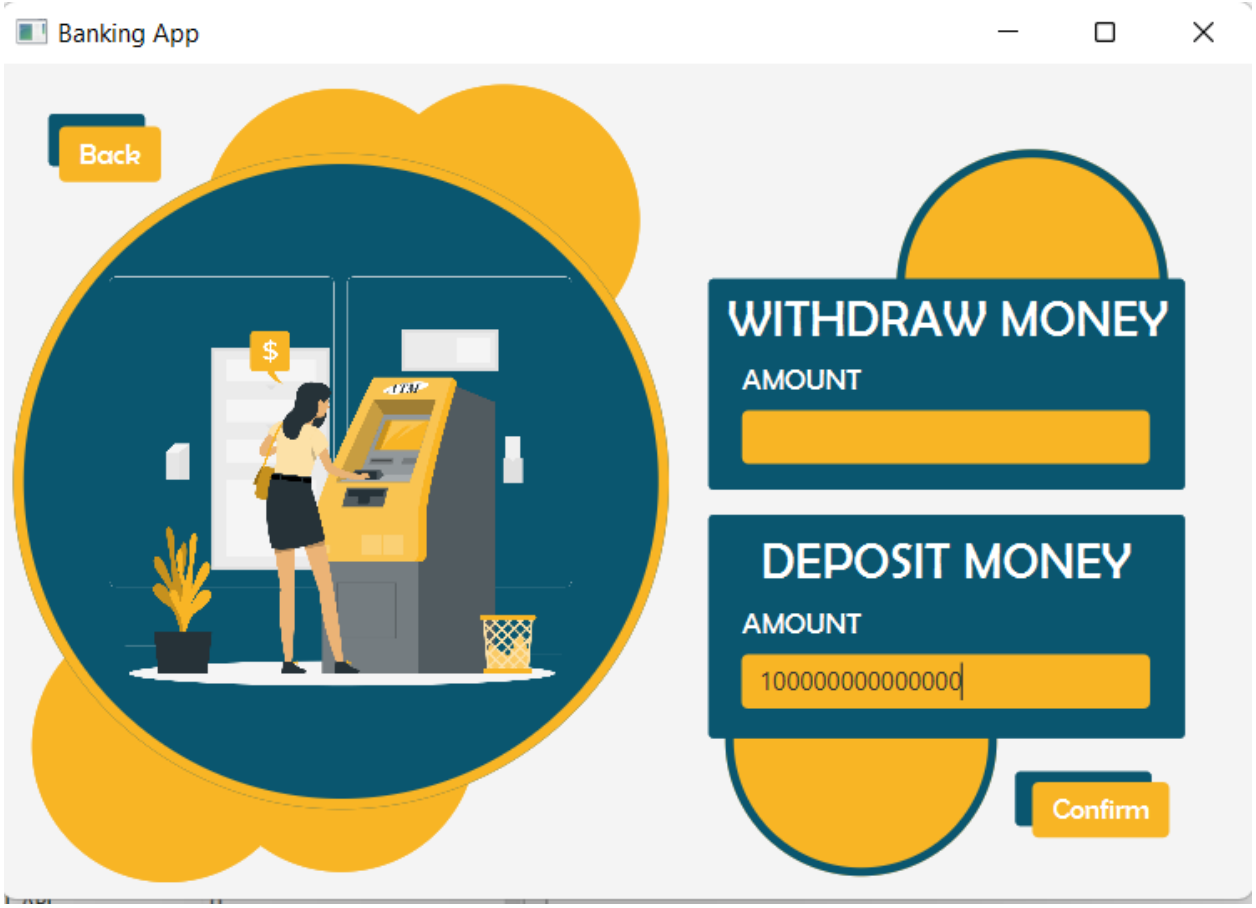
Startup Time Cost per Plugin

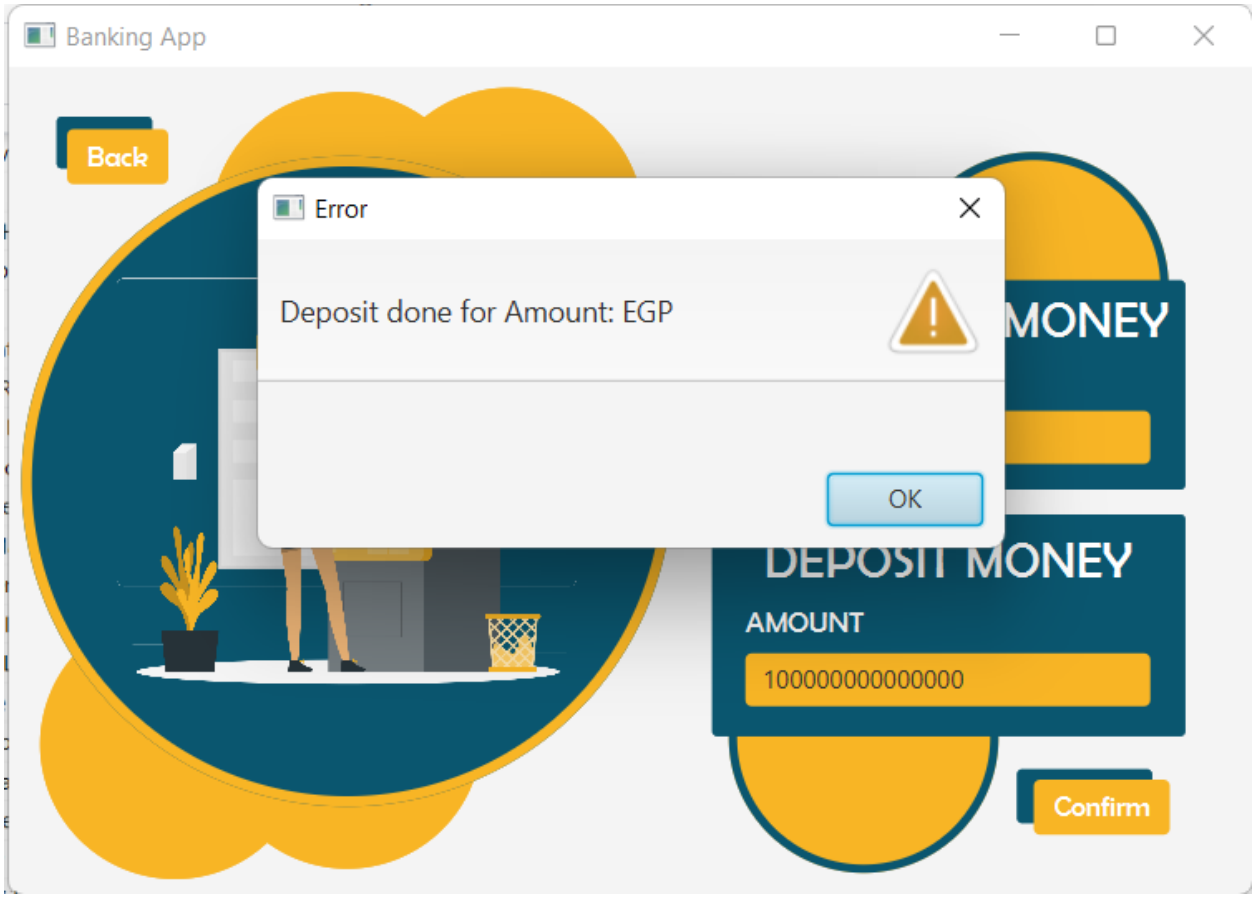
Plugin	Startup Time (ms)
Qodana	375
TextMate Bundles	331
Java Internationalization	303
Images	299
JetBrains Repository Search	296
TestNG	281
Shell Script	244
UI Designer	221
WebP Support	192
Copyright	191
Code Coverage for Java	164
YAML	141
Perforce Helix Core	130
XSLT Debugger	127
Terminal	111
Eclipse Interoperability	101
Settings Repository	73
JetBrains Marketplace Licensi...	54
Java Bytecode Decompiler	49
Gradle Dependency Updater I..	45
Gradle-Maven	32
Dependency Management Ap..	21
com.intellij.tracing.ide	15
Machine Learning in Search E...	5
Bytecode Viewer	5
Smali Support	2
Java Stream Debugger	2
ChangeReminder	1
Projector Libraries for Code ...	1
Gradle DSL API	0
JetBrains maven model api cl...	0
IDE Features Trainer: Git Less...	0

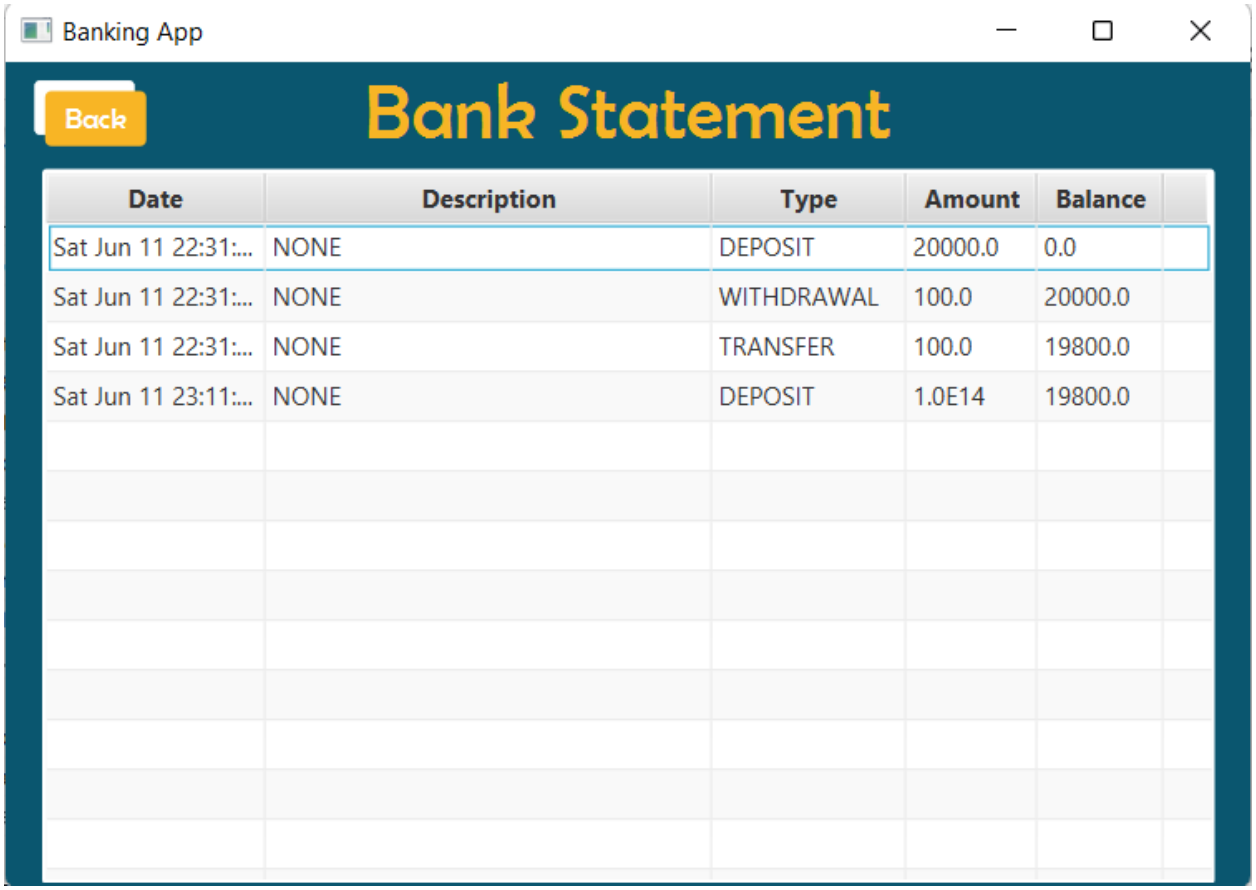
Disable Selected Plugins OK Cancel

Stress Testing

Insert a large number to deposit, the app will not crash and the bank statement works as usual.

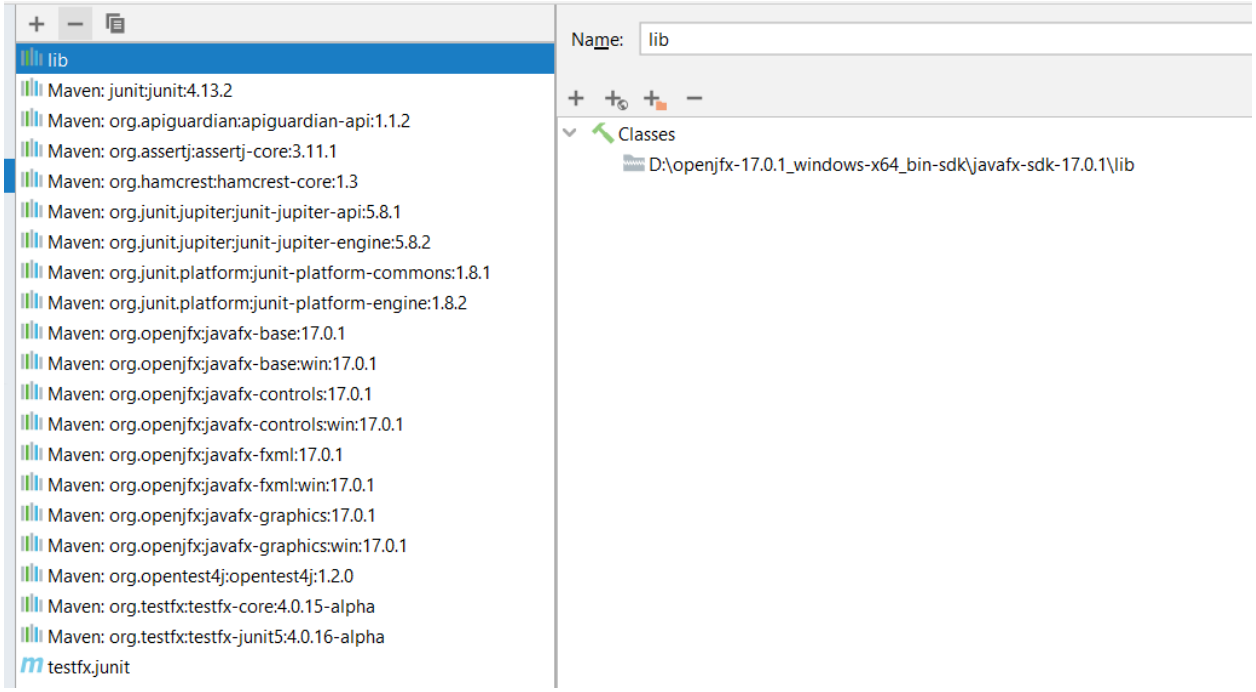

Banking App
Back
WITHDRAW MONEY
AMOUNT
DEPOSIT MONEY
AMOUNT
100000000000000
Confirm


Banking App
Back
Error
Deposit done for Amount: EGP
OK
MONEY
DEPOSIT MONEY
AMOUNT
100000000000000
Confirm

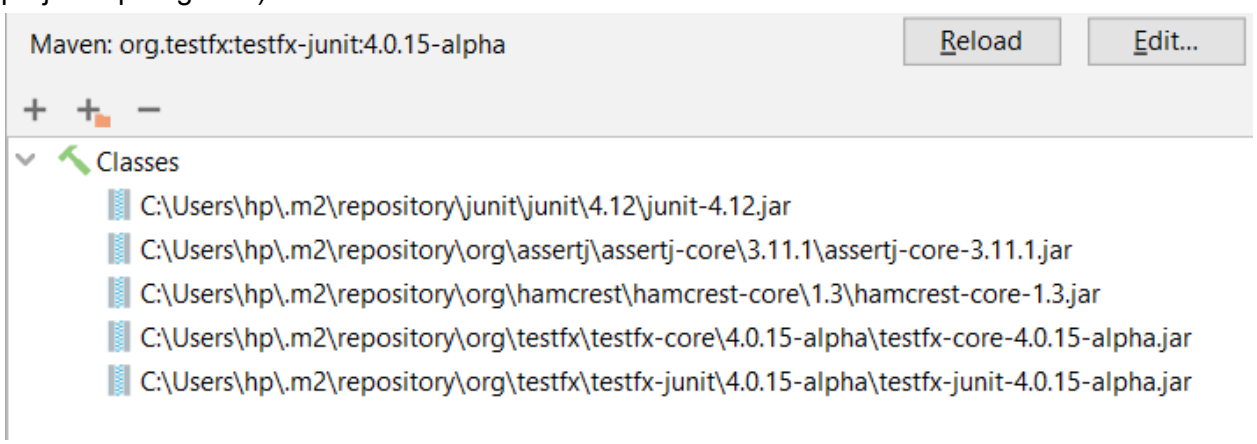


Tools Used in Development

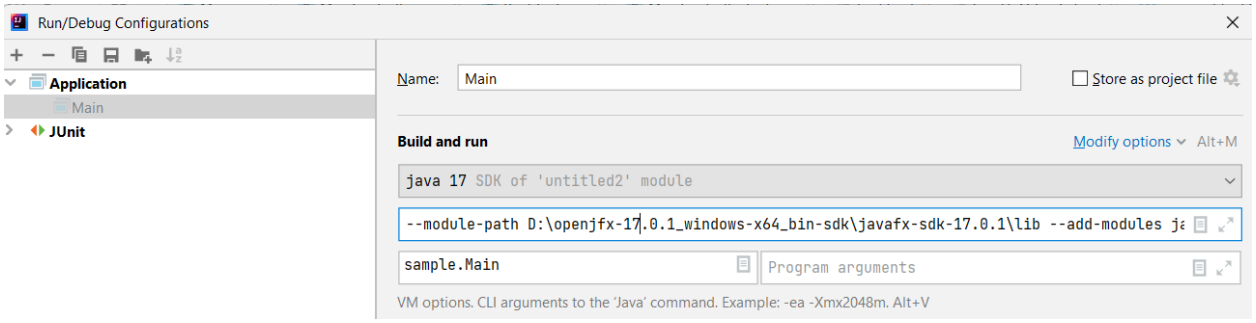
- IntelliJ IDEA as IDE
- Java SDK 17
- JavaFX and SceneBuilder for GUI
- Maven to create project with dependencies
- Junit for unit tests
- TestFX for GUI testing
- Jmeter for performance testing
- **Libraries:**



'lib' is javafx lib, and 'testfx' is the following: (More details about dependencies in pom.xml file in project zip or github)



- **Run Configuration**: this is important to setup to run a JavaFX project in IntelliJ



Closing Notes on Reuse and Refactoring

The Item class unit tests, while useful, did not make it to the final version of the app since the purchases with credit card option were removed as it was not able to integrate with GUI. Therefore, Item class was unnecessary in the end, but perhaps in a future update, the user will be able to make purchases with his credit card. The fxml files for the purchases page also exist in the project folder and can easily be reintegrated once regression testing has proven they will not break the app again.