Assignment 3
Unit Testing - JUnit
Points: 50

Preparation/Readings:

JUnit: http://junit.org
JUnit Tutorial: http://courses.cs.washington.edu/courses/cse143/11wi/eclipse-tutorial/junit.shtml

---

Instructions:

Follow the steps detailed below. Make sure your provide good Javadoc comments for all your classes and methods.

Step 1: Create BankAccount class

Create a new eclipse project titled Lab-JUnit. Create *BankAccount* class under package core. It should have the following attributes and methods:

- *accountNumber* - String
- *accountHolder* - String
- *balance* - double (balance without interest added)
- *accountType* - int (Account type can be either 1 - Savings, 2 - Award Savers, 3 - Checking, or 4 for Credit Card, anything other than these values should set to 0)
- Two constructors:
  - default constructor (no parameters): Set balance=0, accountType=0, accountNumber="none", accountHolder="Unknown"
  - Other constructor accepts user-entered values for all attributes
- Gets and sets for all attributes (You don't have to type up all the get and set methods. Use Eclipse to generate them automatically. Click on Source → Generate Getters and Setters).
- Method *calculateTotalBalance* () - returns the balance of the account after the interest has been applied. Formula for calculating the balance is:
  *totalBalance = balance + (balance * interestRate/100)*.
  The interest rate depends on the accountType:
  - Savings: interest is 0.5%
  - Award Savings: interest is 4.5%
  - Checking: interest is 1.0%

- CreditCard: interest is 15%
- Other: return 0

- Method *getInterestRate* that returns the interest rate depending on the type of account

---

Step 2: Create a TestCase

Create a TestCase class titled *BankAccountTest* (New → Java → JUnitTestCase) under package *test*. Check appropriate method stubs you would like to create. You can create *setUp*(), *tearDown*() methods are run before and after, respectively, each test case is run. You can create *setUpBeforeClass*(), *tearDownAfterClass*() methods only run when the test case class is first instantiated and terminated respectively. You can browse the application to the class that you wish to test and select appropriate methods that you plan to test, or this could be left blank if you will generate the class while creating the test. Test *calculateTotalBalance* and *getInterestRate* methods of *BankAccount* class.

Below is a test case template that tests a List class. This test class demonstrates the basic functionality of *setUp*() and *tearDown*() methods, and gives example test cases.

```
import org.junit.Assert.*;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

public class SampleTest {

private List emptyList;

@BeforeClass
public static void setUpBeforeClass() throws Exception {
}

@AfterClass
public static void tearDownAfterClass() throws Exception {
}

/**
* Sets up the test fixture (Called before every test case method.)
*/
public void setUp() throws Exception {
        emptyList = new ArrayList();
}
```

```
/**
* Tears down the test fixture (Called after every test case method.)
*/
public void tearDown() throws Exception {
        emptyList = null;
}

@Test
public void testSomeBehavior() {
        assertEquals("Empty list should have 0 elements", 0, emptyList.size());
}

@Test
public void testAnotherBehavior() {
        assertTrue(emptyList.size()  == 0);
}

}
```

Step 3: Create a TestSuite

A TestSuite is a simple way of running one program that, in turn, runs all test cases at one time. Create a TestSuite class titled *LabTestSuite* under package *test*. You have to provide names of test classes you would like to include in the test suite using the annotation @SuiteClasses (Test1.class, Test2.class, Test3.class).

Below is a test suite template. This test suite demonstrates the basic functionality of the *suite*() method, which is what you add each of the test cases to the suite in. This should all be generated for you by Eclipse if you use the eclipse wizard to create the Test Suite as explained above. Do not forget to change the class name to *LabTestSuite*.

```
import org.junit.runners.Suite.SuiteClasses;
import org.junit.runners.Suite;
import org.junit.runner.RunWith;

@RunWith(Suite.class)
@SuiteClasses({ BankAccountTest.class })

public class LabTestSuite {

}
```

Execute the test cases by right clicking on LabTestSuite, Select Run → JUnitTestCase. Make sure the tests run successfully.

Step 4: Create Customer class

Create another class titled *Customer* under package *core*. It should have the following attributes and methods:

- *name* - String
- *streetAddress* - String
- *city* - String
- *state* - String
- *zip* - String
- *age* - int
- Constructor that accepts name and address values and sets the attributes.
- Gets and sets for all attributes (You don't have to type up all the get and set methods. Use Eclipse to generate them automatically. Click on Source → Generate Getters and Setters).
- Method *displayAddress* () - returns a string that has the complete formatted address with street address, city, state, and zip (you can choose how you want to format it).
- Method *displayAddressLabel* () - returns a string that has the Customer's name along with the complete formatted address (you can choose how you want to format it).

Also create a TestCase class titled *CustomerTest* to test *displayAddress* and *displayAddressLabel* methods. Add this test case to *LabTestSuite* and run the tests. Make sure all tests are successful.

Assertion Statement Reference

This is a list of the different types of assertion statements that are used to test your code. Any Java data type or object can be used in the statement. These assertions are from the JUnit API. Refer to the API for more details.

assertEquals(expected, actual)
assertEquals(message, expected, actual)
assertEquals(expected, actual, delta) - used on doubles or floats, where delta is the difference in precision
assertEquals(message, expected, actual, delta) - used on doubles or floats, where delta is the difference in precision
assertFalse(condition)
assertFalse(message, condition)
assertNotNull(object)
assertNotNull(message, object)

assertNotSame(expected, actual)
assertNotSame(message, expected, actual)
assertNull(object)
assertNull(message, object)
assertSame(expected, actual)
assertSame(message, expected, actual)
assertTrue(condition)
assertTrue(message, condition)
fail()
fail(message)
failNotEquals(message, expected, actual)
failNotSame(message, expected, actual)
failSame(message)

Deliverables:

Submit the following on Blackboard:
• Compress your Eclipse project titled as ASURiteID-Assignment3.zip

---