COMS/SE 319: Software Construction and User Interface
Fall 2018

LAB Activity 8 – TDD & JUnit Testing on Eclipse

Task 1: JUnit Testing in Eclipse

Introduction

- Testing: Process of checking the functionality of the application whether it is working as per requirements.

- Unit testing –
  - Testing at developer level.
  - Testing of single entity (class or method)

- Testing is very essential to every software company to give a quality product to their customers.

- Unit testing can be done in two ways
  - Manual testing
  - Automated testing

What is JUnit ?

- JUnit is a unit testing framework for the Java Programming Language.

- Approach is like "test a little, code a little, test a little, code a little..."

- Increases programmer productivity and stability of program code that reduces programmer stress and the time spent on debugging.

What is a unit test case?

- A Unit Test Case is a part of code which ensures that another part of code (method) works as expected.

- A formal written unit test case is characterized by a known input and by an expected output, which is worked out before the test is executed.
- More details, please go through: http://www.tutorialspoint.com/junit/

Testing on Eclipse

Package Hierarchy

It is a good practice to have separate folders for source and test code (for development/deployment).

1. Create a source folder called “test” in the project. This should be in the same level as src.
2. Create test classes in packages mirroring original classes.

For example:
Suppose you have a class cs319.lab1. MyClass
Create packages cs319 and cs319.lab1 in test and create test class in cs319.lab1 under test folder.

Running Tests

1. Make sure JUnit is on build path.

right click on project->properties-> Java Build Path, click on libraries tab, click on Add Library, click on JUnit, click next, select JUnit 4 and Finish.

2. Right click on individual file and “run as” JUnit 4 tests

Or, right click on “test" directory or package and “run as” open run Dialog and customize. Make sure JUnit is on build path.

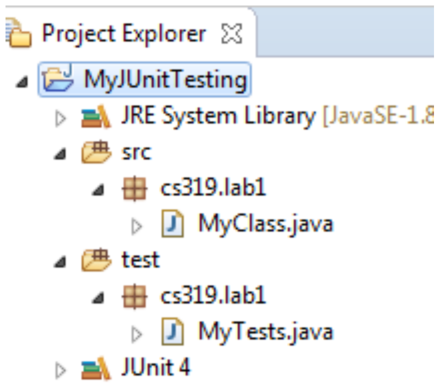
Example 1:

Create a Java project called MyJUnitTesting.

a. Create a class called **MyClass** in package cs319.lab1 under src. Copy the given code.

b. Create a class called **MyTests** in package cs319.lab1 under test.

c. Add JUnit on build path.



d. Create an object of MyClass in MyTests and add a test case to test the findMax method as follows:

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

public class MyTests {

      @Test
      public void findMaxShouldReturnMaxOfThree()
      {
            MyClass tester = new MyClass(); // MyClass is tested

        // assert statements
        assertEquals("Max of (3,2,1) must be 3", 3,
tester.findMax(3,2,1));
        assertEquals("Max of (1,3,2) must be 3", 3,
tester.findMax(1,3,2));
        assertEquals("Max of (1,2,3) must be 3", 3,
tester.findMax(1,2,3));
      }

}
```

e. Run the test case get the result. This will open the "JUnit" view that presents the testing progress and results.

f. Next, modify the findMax method as follows (introduce a bug) and save the MyClass file.

```
public int findMax(int x, int y, int z)
{
      int max;

      if((x>y) && (x>z))
      {
            max = x;
      }
      else if(y>x)
      {
            max = y;
      }
      else
      {
            max = z;
      }

      return max;
}
```

g. Run the test case and get the result.

Using Assertion (Assertion class)

- All the assertion are in the Assert class.

  `public class Assert extends java.lang.Object`

- This class provides a set of assertion methods useful for writing tests. Only failed assertions are recorded.
- Some of the important methods of **Assert** class are:

Methods & Description
void assertEquals(boolean expected, boolean actual) Check that two primitives/Objects are equal
void assertTrue(boolean expected, boolean actual) Check that a condition is true
void assertFalse(boolean condition) Check that a condition is false
void assertNotNull(Object object) Check that an object isn't null.
void assertNull(Object object) Check that an object is null
void assertSame(boolean condition) The assertSame() methods tests if two object references point to the same object
void assertNotSame(boolean condition) The assertNotSame() methods tests if two object references not point to the same object
void assertArrayEquals(expectedArray, resultArray); The assertArrayEquals() method will test whether two arrays are equal to each other.

Example2:

Create a new class called **TestAssertions** in test folder. and check each of the assertions given in above table as follows.

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

import org.junit.Test;

public class TestAssertions {

    @Test
    public void testAssertions() {
       //test data
       String str1 = new String ("abc");
       String str2 = new String ("abc");
       String str3 = null;
       String str4 = "abc";
       String str5 = "abc";
       int val1 = 5;
       int val2 = 6;
       String[] expectedArray = {"one", "two", "three"};
       String[] resultArray =  {"one", "two", "three"};

       //Check that two objects are equal
       assertEquals(str1, str2);

       //Check that a condition is true
       assertTrue (val1 < val2);

       //Check that a condition is false
       assertFalse(val1 > val2);

       //Check that an object isn't null
       assertNotNull(str1);

       //Check that an object is null
       assertNull(str3);

       //Check if two object references point to the same object
       assertSame(str4,str5);

       //Check if two object references not point to the same object
       assertNotSame(str1,str3);
       //Check whether two arrays are equal to each other.
       assertArrayEquals(expectedArray, resultArray);
    }
}
```

Quiz Question:
assertSame(str4 , str5);
With the above code in the TestAssertions class, will the test pass or fail?

Task 2: Test Driven Development (TDD)

Test-Driven Development starts with designing and developing tests for every small functionality of an application. In TDD approach, first the test is developed which specifies and validates what the code will do.

TDD cycle

1. Write a test.
2. Make it run.
3. Change code to make it right (Refactor).
4. Repeat process.

Exercise 1

You have provided a java file BankAccount. BankAccount is production code, it should go to the src folder. AccountTest is the test code which should be included in test folder of your project. Additional to the getter and setter function, there will be a deposit function which will deposit a certain amount to the balance.

Step1:

Create a java project **TDDExcercise1** and create a test folder in the same level of src.

Step2:

Create a package cs319.lab1 in both src and test and copy given source files. Make sure you can run the first test given without adding any TODOs.

Step3:

Start from AccountTest. Complete one test at a time and add production code you need in BankAccount.

Quiz Question:

```
@Test public void testDeposit() {

Account a = new Account("Customer");

a.deposit(100);

assertEquals(100, a.getBalance());

account.deposit(50);
assertEquals(150, a.getBalance());

}
```

Will this test pass?

<u>Exercise 2</u>

You have provided three java files **EmployeeDetails**, **EmpBusinessLogic** and **TestEmployee**. EmployeeDetails and EmpBusinessLogic are production code. Therefore should go in src folder. TestEmployee is the test code which should be included in test folder of your project.

<u>Step1</u>:

Create a java project called **TDDExcercise2** and create a **test** folder in the same level of src**.**

<u>Step2:</u>

Create a package cs319.lab2 in both src and test and copy given source files. Make sure you can run the first test given without adding any TODOs.

<u>Step3:</u>

Start from TestEmployee. Complete one test at a time and add production code you need in both EmployeeDetails and EmpBusinessLogic.

NOTE: Please try to follow TDD cycle. After completing each test, run the code to see whether it passes or fail. If fails, make the production code right in order to pass the test.