Programming 1

Week 14 – Introduction to Unit Testing with JUnit

What is Unit Testing

- Method of testing that verifies the individual units of the code is working properly.
- Test the smallest unit in source code
- A unit test is a piece of code that exercises a small, specific area of functionality, e.g., a particular method in a particular context.
- A unit test seeks to demonstrate that the code does what the developer expects.
- If testing indicates that code works as expected, we proceed to assemble and test the whole system (Integration testing).

Types of unit testing

- There are two ways to perform unit testing:

  1) Manual Testing

  If you execute the test cases manually without any tool support, it is known as manual testing. It is time consuming and less reliable.

  2) Automated Testing

  If you execute the test cases by tool support, it is known as automated testing. It is fast and more reliable.

Why Unit Testing

- Verifies if the unit is working
- Make sure the unit is working even after late changes in source code
- The most used testing framework in java is **Junit**:
    - Is a unit test framework in java
    - Widely used and commonly become standard unit test framework

Using Junit Testing

- JUnit framework uses annotations to identify methods that specify a test.
- A JUnit test is a method contained in a class which is only used for testing. This is called a Test class.
- To define that a certain method is a test method, annotate it with the **@Test** annotation. This method executes the code under test.
- Tests do not have a return type - they are void functions
- Upon success, a test will do nothing
- Upon failure, the test will throw an AssertionError
- This error is handled by JUnit, no extra work for the programmer!

How Tests Pass or Fail

- In JUnit tests, the programmer asserts a condition
- If the assertion is true, the test passes
- If the assertion is false, the test fails
- JUnit provides many assert functions

Annotation in Junit Testing

- Let's see the annotations that can be used while writing the test cases.
- **@Test** annotation specifies that method is the test method.
- **@BeforeClass** annotation specifies that method will be invoked only once, before starting all the tests.
- **@Before** annotation specifies that method will be invoked before each test.
- **@After** annotation specifies that method will be invoked after each test.
- **@AfterClass** annotation specifies that method will be invoked only once, after finishing all the tests

A JUnit test class

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

public class name {
    ...

    @Test
    public void name() {  // a test case method
        ...
    }
}
```

- A method with `@Test` is flagged as a JUnit test case.
  - All `@Test` methods run when JUnit runs your test class.

Assert class

- You use an **assert** method, provided by JUnit framework, to check an expected result versus the actual result.
- The `org.junit.Assert` class provides methods to assert the program logic.

The common methods of Assert class are as follows:

- **assertEquals(expected, actual)**: checks that two primitives/objects are equal. It is overloaded.
- **assertTrue(boolean condition)**: checks that a condition is true.
- **assertFalse(boolean condition)**: checks that a condition is false.
- **assertNull(Object obj)**: checks that object is null.
- **assertNotNull(Object obj)**: checks that object is not null.

- You should provide meaningful messages in assert statements. That makes it easier for the user to identify and fix the problem.
- This is especially true if someone looks at the problem, who did not write the code under test or the test code.

Simple JUnit Demo

Suppose that we wish to carry out unit testing on the following Java program, which uses static methods to perform arithmetic operations on two integers.

```
public class Calculator {
    public static int add(int number1, int number2) {
    return number1 + number2;
    }
    public static int sub(int number1, int number2) {
    return number1 - number2;
    }
   public static int mul(int number1, int number2) {
    return number1 * number2;
    }
    public static int divInt(int number1, int number2) {
     return number1 / number2;
     }
```

Create Unit Test

- Choose this menu in NetBeans
  - Tools > Create/Update Tests
- Or just simply press Ctrl + Shift + U.
- A window dialogue will appear, choose suitable options, Or you can leave it as is.
- Test case will automatically build inside the test package folder.

Unit Testing

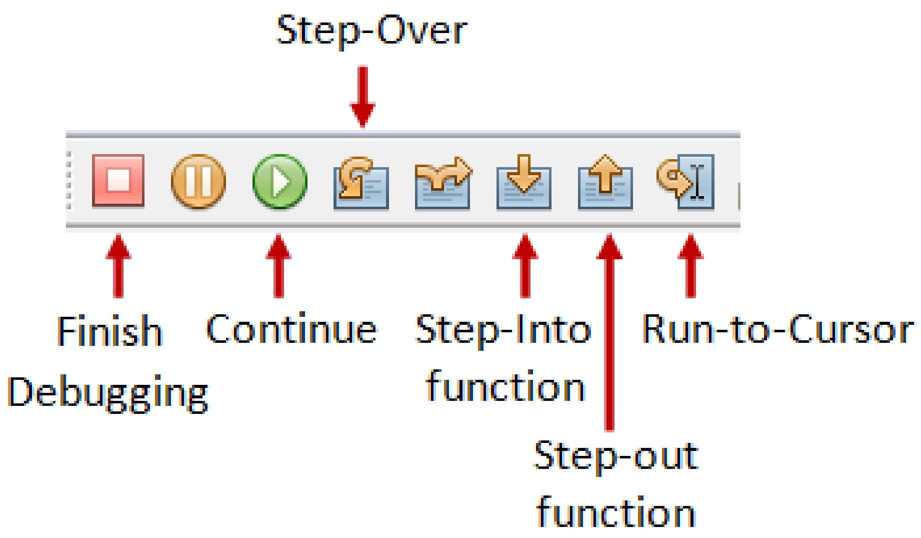
- Assign the variable value for the test case.
- Remove the fail() method in return valued method test.
- Run the test class using Shift + F6.
- See the test result

Create JUnit Test Suite Demo

What is Debugging?

- When things don't work correctly, it can be difficult to figure out exactly what the problem is so that we can fix it. NetBeans, like most other programming environments, provides a particular tool for this called a **debugger**.
- **Debugging** can be defined as the process used for examining the code for errors. Debugging is carried out by setting breakpoints in code and then using debugger to run it. We can execute our code one line at a time and examine the state of our application in order to discover any problems.

Icons..


Step-Over
Finish Debugging
Continue
Step-Into function
Step-out function
Run-to-Cursor