JUnit Testing

JUnit is an open-source Unit Testing Framework for JAVA. It is useful for Java Developers to write and run repeatable tests. Erich Gamma and Kent Beck initially developed it. It is an instance of xUnit architecture.

Developers who are following test-driven methodology must write and execute unit test first before any code. Once you are done with code, you should execute all tests, and it should pass. Every time any code is added, you need to re-execute all test cases and makes sure nothing is broken.

Typical import statements:

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

`assert.XXX` Method	What it's used for
`assertArrayEquals("message", arr1, arr2);`	Asserts the equality of arrays `arr1` and `arr2`
`assertEquals("message", obj1, obj2);`	Asserts that two objects `obj1` and `obj2` (by calling `obj1.equals(obj2)`)
`assertEquals("message", val1, val2);`	Asserts the equality of primitive values `val1` and `val2`
`assertEquals("message", val1, val2, tolerance);`	Asserts the equality of floating-point values `val1` and `val2` within positive `tolerance`
`assertTrue("message", condition);`	Asserts that the condition is true
`assertFalse("message", condition);`	Asserts that the condition is false
`assertNotEquals("message", obj1, obj2);`	Asserts that objects `obj1` and `obj2` are **not** equals
`assertNotEquals("message", val1, val2);`	Asserts that primitive values `val1` and `val2` are **not** equals
`assertNotEquals("message", val1, val2, tolerance);`	Asserts that floating-point values `val1` and `val2` are **not** equal to within positive `tolerance`
`assertNotSame("message", obj1, obj2);`	Asserts that objects `obj1` and `obj2` are **not** the same object
`assertNull("message", obj);`	Asserts the `obj` is a null reference
`assertSame("message", obj1, obj2);`	Asserts that objects `obj1` and `obj2` refer to the same object

Compiler Annotation	What it's used for
`@Test`	The **`public void`** method can be run as a test case
`@Test(expected=ExceptionType.class)`	The corresponding **`public void`** test method is expected to throw an exception of type ExceptionType. If it doesn't, or if another exception type is thrown, the test fails.
`@BeforeClass`	The **`public static void`** method will be run **once** before any of the test methods. This is used primarily for some expensive setup task like connecting to a database.
`@AfterClass`	If any expensive resources were allocated by a @BeforeClass method, they should be deallocated by the **`public static void`** method designated with @AfterClass. This method will be run after all the tests have been run, and is guaranteed to run even if the BeforeClass method throws an exception.
`@Before`	A **`public void`** method that is run before **each** test case is run. This can be used to initialize data before the tests are run.
`@After`	A **`public void`** method that is run after **each** test case is run. This can be used to deallocate resources after a test.
`@Ignore`	Temporarily disable individual tests or the entire test class. Can put @Ignore in front of class declaration to disable entire class or in front of each @Test annotation in class to disable individual tests.
`@Ignore("Need to finish test")`	Temporarily disable test but record why test is being ignored

Simple Example

```
// MyEven.java

public class MyEven
{
    public static boolean isEven(int number)
    {
        return (number % 2 == 0);
    }
}



// MyEvenTest.java

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

public class MyEvenTest
{
    @Test
    public void testIsEven()
    {
        assertEquals("10 is an even number", true, MyEven.isEven(10));
        assertEquals("11 is an odd number", false, MyEven.isEven(11));

        // Alternative to assertEquals
        assertTrue("10 is an even number", MyEven.isEven(10));
        assertFalse("11 is an odd number", MyEven.isEven(11));
    }
}
```

Longer Example

```
// RationalTest.java

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

public class RationalTest
{
    @BeforeClass
    public static void testRational()
    {
        System.out.println("\nThis program performs unit "
            + "testing");
        System.out.println("on the Rational class "
            + "using JUnit.");
    }

    @Test
    public void testConstructor()
    {
        Rational number = new Rational(0, 1);
        assertEquals("number = 0/1", "0/1", number.toString());
        number = new Rational(3, 1);
        assertEquals("number = 3/1", "3/1", number.toString());
        number = new Rational(2, 6);
        assertEquals("number = 2/6 = 1/3", "1/3", number.toString());
        number = new Rational(12, 9);
        assertEquals("number = 12/9 = 4/3", "4/3", number.toString());
        number = new Rational(1, -6);
        assertEquals("number = 1/-6 = -1/6", "-1/6", number.toString());
        number = new Rational(-5, -9);
        assertEquals("number = -5/-9 = 5/9", "5/9",  number.toString());
    }

    @Test(expected=IllegalArgumentException.class)
    public void testConstructorException()
    {
        Rational number = new Rational(5, 0);
    }

    @Test
    public void testAccessors()
    {
        Rational number = new Rational(3, 1);
        assertEquals("numerator = 3", 3, number.getNumerator());
        assertEquals("denominator = 1", 1, number.getDenominator());

        number = new Rational(2, 3);
        assertEquals("numerator = 2", 2, number.getNumerator());
        assertEquals("denominator = 3", 3, number.getDenominator());
    }
    @Test
    public void testSetValue()
    {
        Rational number = new Rational(1, 1);
        number.setValue(5, 4);
        assertEquals("setvalue(5, 4)", "5/4", number.toString());

        number.setValue(-4, -12);
        assertEquals("setvalue(-4, -12) = 1/3", "1/3", number.toString());
    }

    @Test(expected=IllegalArgumentException.class)
    public void testSetValueException()
    {
        Rational number = new Rational(1, 1);
        number.setValue(8, 0);
    }

    @Test
    public void testGetValue()
    {
        Rational number = new Rational(5, 4);
        assertEquals("getValue() == 5.0/4", 1.25, number.getValue(), 1e-6);
        number = new Rational(3, 1);
        assertEquals("getValue() == 3.0/1", 3.0, number.getValue(), 0);
    }

    @Test
    public void testEquals()
    {
        Rational number1 = new Rational(1, 5);
        Rational number2 = new Rational(1, 5);
        assertNotSame("number1 and number2 are separate objects",
            number1, number2);
        assertEquals("1/5 == 1/5", number1, number2);
        number2.setValue(1, 6);
        assertNotEquals("1/5 != 1/6", number1, number2);
        number2.setValue(2, 5);
        assertNotEquals("1/5 != 2/5", number1, number2);
    }

    @AfterClass
    public static void testEnd()
    {
        System.out.println("\nEnd of testing for "
            + "the Rational class.");
    }
}
```

Setting Up JUnit on the Unix System

We are using JUnit 4.1.2. On the Unix system, we need two Java archive (JAR) files in our classpath. You can put the location of those files in your classpath by doing **either** of the following:

- Copy `~lwilson/.bash_profile` to your **home** directory, so that you will have the CLASSPATH settings, **OR**
- Copy just the two lines of that file that contain CLASSPATH information to your own `.bash_profile` file in your **home** directory.

To make the change take effect for your current login session, execute the following command while in your home directory: `source .bash_profile`

Compiling and Running a Test Program on the Unix System

Once your CLASSPATH is set correctly, you can compile your JUnit test program just as you normally would:

```
javac MyEvenTest.java
```

Running the program is a little harder, because we must specify the “runner” used to execute our test cases.

```
java org.junit.runner.JUnitCore MyEvenTest
```

To simplify this process of running our JUnit tests, I’ve created a Unix script called `runJUnit` can that be called as follows:

```
runJUnit MyEvenTest
```

Note that this script should be used ONLY for running JUnit test cases, not your regular Java programs!

Common question: How do I test the private methods of a class?

Answer: Often, you can test them suitably by testing the `public` methods that call them. If you absolutely must test a private method, you can temporarily give the method to be tested *package* visibility as follows:

Comment out private visibility, but make it clear
that it should be private when testing completed

```
/* private -> TESTING */  void reduce()
{
    // code for Rational's reduce method
}
```