**Unit Testing with JUnit**

1. **Unit tests**

A unit test is a piece of code written by a developer that executes a specific functionality in the code which is tested.

Unit tests target small units of code, e.g. a method or a class.

Unit tests ensure that code works as intended. They are also very helpful to ensure that the code still works as intended in case you need to modify code for fixing a bug or extending functionality.

Typically unit tests are created in their own project or their own source folder to avoid that the normal code and the test code is mixed.

1. **JUnit**

**JUnit** is a unit testing framework for the Java Programming Language. JUnit has been important in the development of test-driven development.

JUnit features:

* Assert statements: used to assert a specific result.
* Test suite: Help us to organize and run the tests.
* Test exception: to test whether an exception occurs or not.

1. **Assert Statements**

JUnit provides a number of assert statements (full list found [here](http://junit.sourceforge.net/javadoc_40/index.html) under the Assert class. The assertEquals() method takes two parameters. The first parameter is the value we expect the method will return. The second parameter is what the actual value is. Let's look at the following assert statement:  
      assertEquals(6, 3\*2);  
In this assert, we expect the result of 3\*2 to be 6, so we put that as the first parameter. The second parameter is then the result of what 3\*2 is.

You could also add another parameter to specify the message when error occurs. For example: assertEquals(“This is not equal ”,6, 3\*3), the value we expect is 6 but the actual value return is 9. So the test case will fail with error “This is not equal”.

1. **Test Suite**

A TestSuite is a Composite of Tests. It runs a collection of test cases. Here is an example using the dynamic test definition.

TestSuite suite= new TestSuite();

suite.addTest(new MathTest("testAdd"));

suite.addTest(new MathTest("testDivideByZero"));

You could also combine a test suite into another one. For example:

TestSuite suite= new TestSuite(TestGame.class);

suite.addTestSuite(TestPlayer.class);

suite.addTestSuite(TestScore.class);

When you run this test suite, it will run every test method in included class.

Alternatively, a TestSuite can extract the tests to be run automatically. To do so you pass the class of your TestCase class to the TestSuite constructor.

TestSuite suite= new TestSuite(MathTest.class);

1. **Test exception**

This test is used to handle whether an exception occurs. For example:

*Fail*(“Expected NumberFormatException”)

String a= “abc”;

Int b=Integer.*parseInt*(a);

As we see, String “abc” cannot parse to Integer. So NumberFormatException would occurs and we declare fail() method to handle it.

1. **Server-side test**

The tests described above are intended to assist with testing client-side code. The test case wrapper GWTTestCase will launch either a development mode session or a web browser to test the generated JavaScript. On the other hand, server-side code runs as native Java in a JVM without being translated to JavaScript, so it is not necessary to run tests of server-side code using GWTTestCase as the base class for your tests. Instead, use JUnit's TestCase and other related classes directly when writing tests for your application's server-side code. That said, you may want both GWTTestCase and TestCase coverage of code that will be used on both the client and the server.

1. **Asynchronous Testing**

GWT's [JUnit](http://www.junit.org) integration provides special support for testing functionality that cannot execute in straight-line code. For example, you might want to make an [RPC](http://www.gwtapps.com/doc/html/com.google.gwt.doc.DeveloperGuide.RemoteProcedureCalls.html) call to a server and then validate the response. However, in a normal JUnit test run, the test stops as soon as the test method returns control to the caller, and GWT does not support multiple threads or blocking. To support this use case, [GWTTestCase](http://www.gwtapps.com/doc/html/com.google.gwt.junit.client.GWTTestCase.html) has extended the TestCase API.

The two key methods are [GWTTestCase.delayTestFinish(int)](http://www.gwtapps.com/doc/html/com.google.gwt.junit.client.GWTTestCase.html#delayTestFinish%28int%29) and [GWTTestCase.finishTest()](http://www.gwtapps.com/doc/html/com.google.gwt.junit.client.GWTTestCase.html#finishTest%28%29). Calling delayTestFinish() during a test method's execution puts that test in asynchronous mode, which means the test will not finish when the test method returns control to the caller. Instead, a *delay period* begins, which lasts the amount of time specified in the call to delayTestFinish(). During the delay period, the test system will wait for one of three things to happen:

1. If finishTest() is called before the delay period expires, the test will succeed.
2. If any exception escapes from an event handler during the delay period, the test will error with the thrown exception.
3. If the delay period expires and neither of the above has happened, the test will error with a [TimeoutException](http://www.gwtapps.com/doc/html/com.google.gwt.junit.client.TimeoutException.html).

The normal use pattern is to setup an event in the test method and call delayTestFinish() with a timeout significantly longer than the event is expected to take. The event handler validates the event and then calls finishTest().

1. **Integrate Junit into GWT project**

Some simple steps to integrate Junit into GWT project:

* Add junit library into classpath.
* In Java Build Path, move GWT SDK ahead of App Engine SDK.
* At test folder, add a package (remember to add same package with gwt module). Declare source path for this package in Gwt Module.
* To test a RPC calling, you need to add <servlet path=*'/greet'* class=*'com.demo.junit.server.GreetingServiceImpl'*/> into GWT Module.
* Create a class in testing package which extends gwtTestcase (GWTTestCase only be used to test client-side).
* Write a simple test method which method’s name start with test (ex: testServer(), testSimple() ).
* Run the class with Eclipse command => Run as => GWT JUnit test.