**Unit Testing**

A unit test is a piece of code written by a developer that tests a specific functionality in the code which is tested. Unit tests can ensure that functionality is working and can be used to validate that this functionality still works after code changes.

### How does JUnit work?

JUnit is a library packed in a jar file. Among other things it contains a tool (called test runner) to run your test files. It is not an automated testing tool: you still have to write your test files by hand. JUnit does give you some support so that you can write those test files more conveniently.

Suppose you have a class *C* that you want to test. We will write the tests in a new class; let’s call it *Ctest*. This *Ctest* is our test file. Actually, test class is a better name. We will typically group the tests in *Ctest* in a bunch of methods called test methods.

To actually test *C* we need to execute its test class *Ctest*. This is done by calling JUnit’s test runner tool; we pass the name *Ctest* to it. That’s all. JUnit will then execute *Ctest* for you.

JUnit will report how many of the test methods in *Ctest* succeed, and how many fail. The detail of each failure will be reported; this will be in the form of a print of Java’s stack trace leading to the location of the failure.

**JUNIT features includes:**

1. test runners for running tests (JUnitCore.runClasses(MyClassTest.class);)
2. test fixtures for sharing test data (@Before, @After)
3. Assertions for testing expected results.

### Compiling and executing your test

Open a console. You first need to compile the test class, and then you can execute it. The commands are (in Windows):

javac -cp .;<full path to JUnit.jar> SubscriptionTest.java  
  
java  -cp .;<full path to JUnit.jar> org.junit.runner.JUnitCore SubscriptionTest

**Annotations**

|  |  |
| --- | --- |
| Annotation | Description |
| @Test public void method() | Annotation @Test identifies that this method is a test method. |
| @Before public void method() | Will perform the method() before each test. This method can prepare the test environment, e.g. read input data, initialize the class) |
| @After public void method() | Test method must start with test |
| @BeforeClass public void method() | Will perform the method before the start of all tests. This can be used to perform time intensive activities for example be used to connect to a database |
| @AfterClass public void method() | Will perform the method after all tests have finished. This can be used to perform clean-up activities for example be used to disconnect to a database |
| @Ignore | Will ignore the test method, e.g. useful if the underlying code has been changed and the test has not yet been adapted or if the runtime of this test is just to long to be included. |
| @Test(expected=IllegalArgumentException.class) | Tests if the method throws the named exception |
| @Test(timeout=100) | Fails if the method takes longer then 100 milliseconds |

### Assert statements

|  |  |
| --- | --- |
| Statement | Description |
| fail(String) | Let the method fail, might be usable to check that a certain part of the code is not reached. |
| assertTrue(true); | True |
| assertsEquals([String message], expected, actual) | Test if the values are the same. Note: for arrays the reference is checked not the content of the arrays |
| assertsEquals([String message], expected, actual, tolerance) | Usage for float and double; the tolerance are the number of decimals which must be the same |
| assertNull([message], object) | Checks if the object is null |
| assertNotNull([message], object) | Check if the object is not null |
| assertSame([String], expected, actual) | Check if both variables refer to the same object |
| assertNotSame([String], expected, actual) | Check that both variables refer not to the same object |
| assertTrue([message], boolean condition) | Check if the boolean condition is true. |

**Q: What Happens If a JUnit Test Method Is Declared as "private"?**

**Ans**: If a JUnit test method is declared as "private", the compilation will pass ok. But the  
execution will fail. This is decause JUnit requires that all test methods must be declared  
as "public".

**What Happens If a JUnit Test Method Is Declared to Return "String"?**

If a JUnit test method is declared to return "String", the compilation will pass ok. But the execution will fail. This is decause JUnit requires that all test methods must be declared to return "void"