COMP220 Test Driven development and Junit University of Liverpool

Author Sebastian Coope

In test driven development the tests are written before the target code is written. The tests themselves are written using the specification which should describe the behaviour of the code.

The tests act as a goal which the code must satisfy before the product can be released. Initially all the tests fail, the code is then improved and as more tests pass this shows progress in the development of the code.

Test driven development has a number of benefits
The testing always gets done
The testing is repeated many times (as it is automated) and early features of the code are testing again and again (as they will be tested every time the code is built)
It is easier to modify the code with confidence as there is re-assurance that any bugs introduced will be picked up by the tests.
The code written tends to be simpler and shorter as the code is only written to satisfy the tests.

JUnit

JUnit is a unit testing is a framework for Java programming language.

To mark a method as a test method the @Test annotation is put just before the method declaration. The annotation adds meta information to the class file for each method. The test runner then uses this meta tag to identify the test methods.

Junit provides a range of methods called assertions, the assertion can be used to test if two values are equal, a value is true or false, or a value is within a certain tolerance of another value. If the assertion fails then the test itself is reported as failed.

Example of an assertion for testring multiply.

```
assertEquals(“Test Multiply failed”,2*2,4);
```

Test fixtures

Test fixtures are resources such as instance of object, files, variables or database tables. Junit provides a special annotation which marks code used to set up the test fixtures for a particular test. The code to create the test fixtures is always marked with a @Before annotation.

Sometimes a series of tests require a fixed set of objects before than can be run. These resources are called a test fixtures and are always created in code which has the @Before annotation. If a test fixture needs to be removed before the next test starts this can be done by marking code with the @After annotation.

Test Methods

Test methods in the Junit test class are prefixed with @Test notation.

Things to note are:

1) The method must be public if the method is not private the test class will fail to load and there will be an initialisation error when the Junit framework tries to load it
2) The testing within the method is done through a series of assertions if all assertions pass then the test passes, if ANY assertions fail, the test fails, the framework will display ALL failing assertions including a stack trace showing where they failed
3) If there is an error (for example a null pointer exception) this is called a test Error and not a test Failure. Error and failure are different, error means the test did not run successfully, failure means the test did run but the assertion failed.

Types of assertion

assertSame(String message,Object object1, Object object2)

The assertion passes is object1 and object2 refer to the same object in memory, this means they have store the same reference.

So

```
String string1=”Hello”;

String string2=”Hello”;

assertSame(“check strings”,s1,s2);
```

Will fail as s1 and s2 are not the same object even though they contain the same data.

assertEquals(String message,Object object1, Object object2)

The assertion passes if object1 and object2 contain the same data, however this depends on the classes of the objects concerned overriding the boolean equals method. Here is an example.

```
public class Person {
        String surname=””;
        String foremane=””;

        public Boolean equals(Object o) {

                Person person=(Person)o;

                return(  (p.surname.equals(surname)) && (p.forename.equals(forename))  );

        }

}
```

Comparing numbers

Integers as primitives can be simply compared.

assertEquals(String message,int number1, int number2);

With floating point values it is more complicated, here is the call

assertEquals(String message,float number1, float number2,float delta);

This will pass as long as | (number1-number2) | < delta

If you forget the delta the call will be

public static void assertEquals(String message, double expected, double actual)

This will compile BUT this is a deprecated call and will cause an API error when the test is run.