JUnit 4.x in Eclipse, a Quick Tutorial

Wishnu Prasetya

November 28, 2008

This tutorial is for a bit more experienced Java programmers. I will assume:

- you know how to write JUnit tests (else see e.g. my *JUnit 4.x Quick Tutorial*).
- you are an Eclipse user; so you would know e.g. how to add a jar to your project's build path.

Setting up an example

Start a new project first; call it e.g. `MyProject`. Create in this project the following java class (which should be called `Subscription.java`). This will be the class we are going to test.

```
public class Subscription {

    private int price  ; // subscription total price in euro-cent
    private int length ; // length of subscription in months

    /**
     * A constructor to create a subsription.
     */
    public Subscription(int p, int n) {
        price = p ;
        length = n ;
    }

    /**
     * Calculate the monthly subscription price in euro,
     * rounded up to the nearest cent.
     */
    public double pricePerMonth() {
        if (length<=0 || price<=0) return 0 ;
        double r = (double) price / (double) length ;
        double fraction = Math.IEEEremainder(r,1.0) ;
        if (fraction > 0)return Math.floor(r) + 1 ;
        else return Math.floor(r) ;
    }

    /**
     * Call this to cancel/nulify this subscription.
     */
    public void cancel() { length = 0 ; }

}
```

An instance of the above class represents a subscription to something (e.g. newspaper, but it doesn't really matter here). Each subscription has its total price, stored in the variable `price`. This price is in Euro-cent. It also has the length of the subscription, given in months.

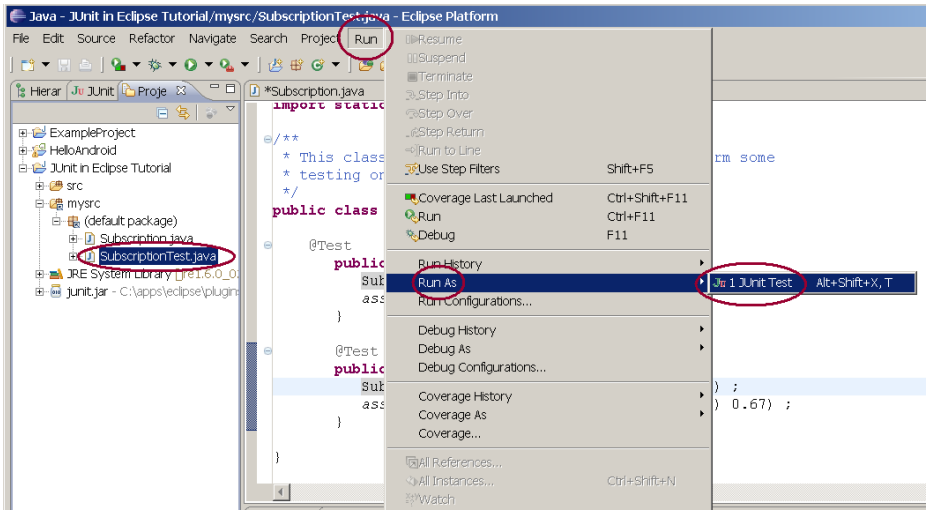

Figure 1: *Runing your test class.*

Creating a test class

First you need to include JUnit's jar into your project build path. Download JUnit if it is not already in your computer. Else look in Eclipse's `plugins` subdirectory; it may already contain JUnit's jar.

Now create the following class, which will act as our test class. It contains some simple tests against the `Subscription` class above.

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

public class SubscriptionTest {

     @Test
     public void test_returnEuro() {
         Subscription S = new Subscription(200,2) ;
         assertTrue(S.pricePerMonth() == (double) 1) ;
     }

     @Test
     public void test_roundingup() {
         Subscription S3 = new Subscription(200,3) ;
         assertTrue(S3.pricePerMonth() == (double) 0.67) ;
     }
}
```

Executing your test from Eclipse IDE

To execute the above test class, just *run* it (via the 'Run' menu). Choose to run it as a JUnit test. See Figure 1.

After running your test class, you will get a report that looks like in Figure 2. You can see there that we have two failures, implying both tests in `SubscriptionTest` fail. Below that you get an overview of all test methods in `SubscriptionTest`, each will be marked by whether it fails of succeeds. If you point to a failing test, on the pane below you can see which line exactly in your test class causes the failure; you see in the screen shot that it says line 13.

Measuring your test coverage

After fixing your class eventually you will get no more failures. So, your class pass your tests. But how good are your tests? The adequacy of tests is usually measured in terms of coverage. *Line coverage* is the precentage of the lines in your methods that were executed by your tests. Similarly, *branch coverage* is the precentage of the branches in your methods that were traversed by the tests. 100 % coverage does not imply absence of errors, but it still indicates that you have tested toroughly. Furthermore, low coverage implies you have not tested enough.

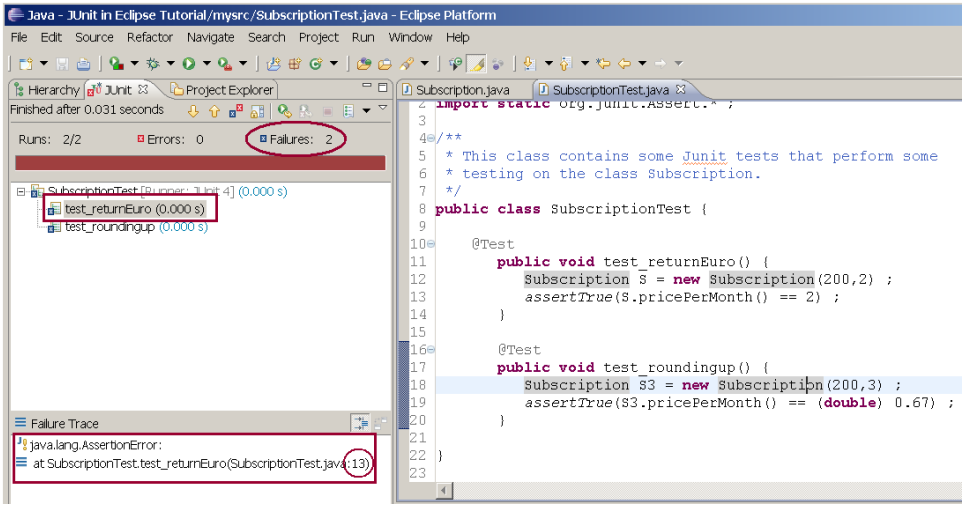

Figure 2: *Runing your test class.*

In Eclipse you can easily measure your coverage. First you need to install the *EclEmma* plugin; it's very easy (an instruction on how to do this can be found in its website). I will just assume that you have it.

Emma is a tool for measuring coverage. The EclEmma plugin adds Emma functionality to your Eclipse. Emma measures line coverage, but it also gives indication if a line containing e.g. an if-then decision has its branches fully or only partially covered.

EclEmma provides this button. See also the screen shot in Figure 3; it's circled in blue.

Select the class `Subscription` (e.g. by clicking on it from your project explorer), then hit the button. This will cause Eclipse to `SubscriptionTest`. You will get your usual JUnit report (the left pane in Figure 3), but notice that the source code of `Subscription` is now collored (the right pane in Figure 3).

A *green* line means it is fully covered; *red* means it is not covered; and *yellow* means it is partially covered. As you can see in Figure 3 our simple tests do not cover the method `cancel` and only partially cover the lines (and the branches) in `pricePerMonth`. So, they were quite incomplete.

At the bottom pane you can also get a summary of the precentage of the line coverage per method in your target class.

End.

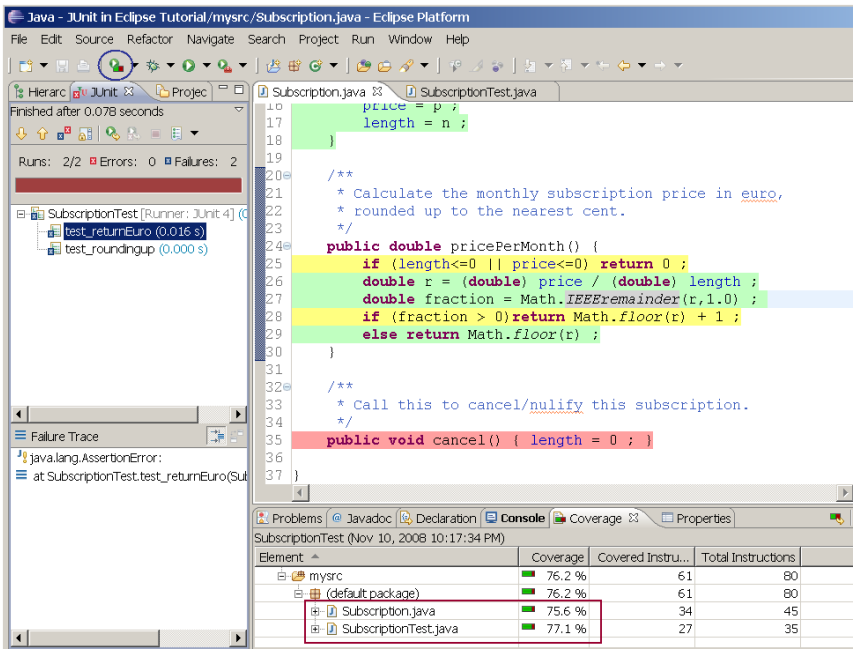

Figure 3: *Measuring the coverage of tests.*