Testing using TestNG in Java

> **Please note**
>
> There is a more up-to-date tutorial using Selenium providing additional capabilities. Please check it instead.

Overview

In this tutorial, we will create a Java Test class with multiple Test Cases, implemented in Java.

The tutorial shows two different approaches for providing additional meta-information for each Test method, one of them will require developing specific code for extending TestNG.

Note that providing additional information on the Test results is optional.

Description

The automated tests validate a Calculator class and exploits some TestNG features such as the ability of validating the same Test against multiple input values, and also the possibility of linking Tests with requirements in Jira using attributes on the test results, amongst other things.

Calculator.java

```
package com.xpand.java;

public class Calculator
{
// Square function
public static int Square(int num)
{
    return num*num;
}
// Add two integers and returns the sum
public static int Add(int num1, int num2 )
{
    return num1 + num2;
}
// Add two integers and returns the sum
public static double Add(double num1, double num2 )
{
    return num1 + num2;
}
// Multiply two integers and retuns the result
public static int Multiply(int num1, int num2 )
{
    return num1 * num2;
}

public static int Divide(int num1, int num2 )
{
    return num1 / num2;
}

// Subtracts small number from big number
public static int Subtract(int num1, int num2 )
{
    if ( num1 > num2 )
    {
    return num1 - num2;
    }
    return num2 - num1;
    }
}
```

Example 1: Using TestNG standard capabilities

This example uses the standard built-in TestNG capabilities, without making use of TestNG's advanced features such as annotations (which in turn would require specific code to handle them).

The test class provides some methods marked with the @Test annotation and in some of them additional meta-information is added to the results using the *ITestResult* object.

Xray is able to process TestNG's *ITestResult* custom attributes named "test", "requirement" and "labels". Whenever specified, they will allow Xray to identify an existing Xray Test issue to report results to, the requirement to link to and additional labels to add to the Test issue, respectively.

CalcTest.java

```
package com.xpand.java;

import org.testng.Assert;
import org.testng.annotations.Test;
import org.testng.annotations.BeforeSuite;
import org.testng.annotations.AfterSuite;
import org.testng.annotations.DataProvider;
import org.testng.Reporter;
import org.testng.reporters.XMLReporter;
import org.testng.ITestResult;
import com.xpand.annotations.Xray;

public class CalcTest {

    @BeforeSuite
    public void setUp() throws Exception {

    }

    @AfterSuite
    public void tearDown() throws Exception {
    }



    @DataProvider
    public Object[][] ValidDataProvider() {
        return new Object[][]{
            {  1, 2, 3 },
            {  2, 3, 4 },  // error or the data itself :)
            { -1, 1, 0 }
        };
    }


    @Test(dataProvider = "ValidDataProvider")
    public void CanAddNumbersFromGivenData(final int a, final int b, final int c)
    {
        Assert.assertEquals(Calculator.Add(a, b), c);
        ITestResult result = Reporter.getCurrentTestResult();
        result.setAttribute("requirement", "CALC-1234");
        result.setAttribute("test", "CALC-2");
    }


        @Test
    public void CanAddNumbers()
    {
        Assert.assertEquals(Calculator.Add(1, 1),2);
        Assert.assertEquals(Calculator.Add(-1, 1),0);
        ITestResult result = Reporter.getCurrentTestResult();
```

```
        result.setAttribute("requirement", "CALC-1234");
        result.setAttribute("test", "CALC-2");
        result.setAttribute("labels", "core addition");
    }


    @Test
    public void CanSubtract()
    {
        Assert.assertEquals(Calculator.Subtract(1, 1), 0);
        Assert.assertEquals(Calculator.Subtract(-1, -1), 0);
        Assert.assertEquals(Calculator.Subtract(100, 5), 95);
        ITestResult result = Reporter.getCurrentTestResult();
        result.setAttribute("requirement", "CALC-1235");
        result.setAttribute("labels", "core");
    }


    @Test
    public void CanMultiplyX()
    {
        Assert.assertEquals(Calculator.Multiply(1, 1), 1);
        Assert.assertEquals(Calculator.Multiply(-1, -1), 1);
        Assert.assertEquals(Calculator.Multiply(100, 5), 500);
        ITestResult result = Reporter.getCurrentTestResult();
        result.setAttribute("requirement", "CALC-1236");
    }


    @Test
    public void CanDivide()
    {
        Assert.assertEquals(Calculator.Divide(1, 1), 1);
        Assert.assertEquals(Calculator.Divide(-1, -1), 1);
        Assert.assertEquals(Calculator.Divide(100, 5), 20);
        ITestResult result = Reporter.getCurrentTestResult();
        result.setAttribute("requirement", "CALC-1237");
    }


    @Test
    public void CanDoStuff()
    {
        Assert.assertNotEquals(true, true);
        ITestResult result = Reporter.getCurrentTestResult();
    }

}
```

Example 2: Using TestNG annotation capabilities

This examples uses some of the advanced TestNG capabilities, namely the annotation mechanism.

We'll use a specific "Xray" annotation in order to quickly, and in a more elegant way, provide additional meta-information (i.e. "test", "requirement" and "labels") to the Test result, without having to use the *ITestResult* object in the Test method's code. Whenever specified, they will allow Xray to identify an existing Xray Test issue to report results to, the requirement to link to and additional labels to add to the Test issue, respectively.

CalcTest.java

```
package com.xpand.java;

import org.testng.Assert;
import org.testng.annotations.Test;
import org.testng.annotations.BeforeSuite;
import org.testng.annotations.AfterSuite;
```

```
import org.testng.annotations.DataProvider;
import org.testng.Reporter;
import org.testng.reporters.XMLReporter;
import org.testng.ITestResult;
import com.xpandit.testng.annotations.Xray;

public class CalcTest {

    @BeforeSuite
    public void setUp() throws Exception {

    }

    @AfterSuite
    public void tearDown() throws Exception {
    }


    @DataProvider
    public Object[][] ValidDataProvider() {
        return new Object[][]{
            {  1, 2, 3 },
            {  2, 3, 4 },  // error or the data itself :)
            { -1, 1, 0 }
        };
    }


    @Test(dataProvider = "ValidDataProvider")
    @Xray(requirement = "CALC-1234", test = "CALC-1")
    public void CanAddNumbersFromGivenData(final int a, final int b, final int c)
    {
        Assert.assertEquals(Calculator.Add(a, b), c);
    }


        @Test
    @Xray(requirement = "CALC-1234", test = "CALC-2", labels = "core addition")
    public void CanAddNumbers()
    {
        Assert.assertEquals(Calculator.Add(1, 1),2);
        Assert.assertEquals(Calculator.Add(-1, 1),0);
    }


    @Test
    @Xray(requirement = "CALC-1235", labels = "core")
    public void CanSubtract()
    {
        Assert.assertEquals(Calculator.Subtract(1, 1), 0);
        Assert.assertEquals(Calculator.Subtract(-1, -1), 0);
        Assert.assertEquals(Calculator.Subtract(100, 5), 95);
    }


    @Test
    @Xray(requirement = "CALC-1236")
    public void CanMultiplyX()
    {
        Assert.assertEquals(Calculator.Multiply(1, 1), 1);
        Assert.assertEquals(Calculator.Multiply(-1, -1), 1);
        Assert.assertEquals(Calculator.Multiply(100, 5), 500);
    }


    @Test
    @Xray(requirement = "CALC-1237")
    public void CanDivide()
    {
        Assert.assertEquals(Calculator.Divide(1, 1), 1);
        Assert.assertEquals(Calculator.Divide(-1, -1), 1);
```

```
        Assert.assertEquals(Calculator.Divide(100, 5), 20);
    }


    @Test
    public void CanDoStuff()
    {
        Assert.assertNotEquals(true, true);
    }

}
```

This requires some additional side code, that uses TestNG extension mechanisms, in order to process the custom "Xray" annotation shown above. This code is not part of Xray and is only provided for reference; feel free to adapt and customize it to your needs.

> **Please note**
>
> Instead of adding the listener code shown ahead, you can include a specific maven plugin that implements extensions for TestNG related to Xray.

We'll define an interface that will be used by a custom TestNG listener (i.e. implementing *ITestListener*).

com/xpandit/testng/annotations/Xray.java

```
package com.xpandit.testng.annotations;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 * A Custom Annotation to inject additional information into a TestNG Test
 */
@Retention(RetentionPolicy.RUNTIME)
public @interface Xray {

    String requirement() default "";

    String test() default "";

    String labels() default "";

}
```

The XrayListener must be defined as a service so TestNG is able to load it at runtime.

src/test/resources/META-INF/services/org.testng.ITestNGListener

```
com.xpand.annotations.XrayListener
```

XrayListener class is responsible for processing "Xray" annotation, get the values of some specific attributes and them as attributes to the *ITestResult* object.

com/xpandit/testng/annotations/XrayListener.java

```
package com.xpandit.testng.annotations;

import java.lang.reflect.Method;

import org.testng.IInvokedMethod;
import org.testng.IInvokedMethodListener;
import org.testng.ITestContext;
import org.testng.ITestListener;
import org.testng.ITestNGMethod;
import org.testng.ITestResult;
import static java.lang.System.out;
import static java.lang.System.err;

/**
 * The listener interface for receiving Xray events.
 * The Listener can be automatically invoked when TestNG tests are run by using ServiceLoader mechanism.
 * You can also add this listener to a TestNG Test class by adding
 * <code>@Listeners({com.xpand.java.XrayAnnotationListener.class})</code>
 * before the test class
 *
 * @see Xray
 */
public class XrayListener implements IInvokedMethodListener, ITestListener  {

    boolean testSuccess = true;


    /* (non-Javadoc)
     * @see org.testng.IInvokedMethodListener#beforeInvocation(org.testng.IInvokedMethod, org.testng.
ITestResult)
     */
    public void beforeInvocation(IInvokedMethod method, ITestResult testResult) {
        if(method.isTestMethod() && annotationPresent(method, Xray.class) ) {
            testResult.setAttribute("requirement", method.getTestMethod().getConstructorOrMethod().getMethod().
getAnnotation(Xray.class).requirement());
            testResult.setAttribute("test", method.getTestMethod().getConstructorOrMethod().getMethod().
getAnnotation(Xray.class).test());
            testResult.setAttribute("labels", method.getTestMethod().getConstructorOrMethod().getMethod().
getAnnotation(Xray.class).labels());
        }
    }


    private boolean annotationPresent(IInvokedMethod method, Class clazz) {
        boolean retVal = method.getTestMethod().getConstructorOrMethod().getMethod().isAnnotationPresent(clazz)
? true : false;
        return retVal;
    }


    /* (non-Javadoc)
     * @see org.testng.IInvokedMethodListener#afterInvocation(org.testng.IInvokedMethod, org.testng.ITestResult)
     */
    public void afterInvocation(IInvokedMethod method, ITestResult testResult) {
        if(method.isTestMethod()) {
            if( !testSuccess ) {
                testResult.setStatus(ITestResult.FAILURE);
            }
        }
    }

    public void onTestStart(ITestResult result) {
        // TODO Auto-generated method stub

    }

    public void onTestSuccess(ITestResult result) {
        // TODO Auto-generated method stub
```

```
    }

    public void onTestFailure(ITestResult result) {
        // TODO Auto-generated method stub

    }

    public void onTestSkipped(ITestResult result) {
        // TODO Auto-generated method stub

    }

    public void onTestFailedButWithinSuccessPercentage(ITestResult result) {
        // TODO Auto-generated method stub

    }

    public void onStart(ITestContext context) {

    }

    public void onFinish(ITestContext context) {
        // TODO Auto-generated method stub

    }

}
```

In Maven's `pom.xml` file, we need to make sure test result related attributes are added to the generated XML report. This can be done by setting `generateTestResultAttributes` to "true".

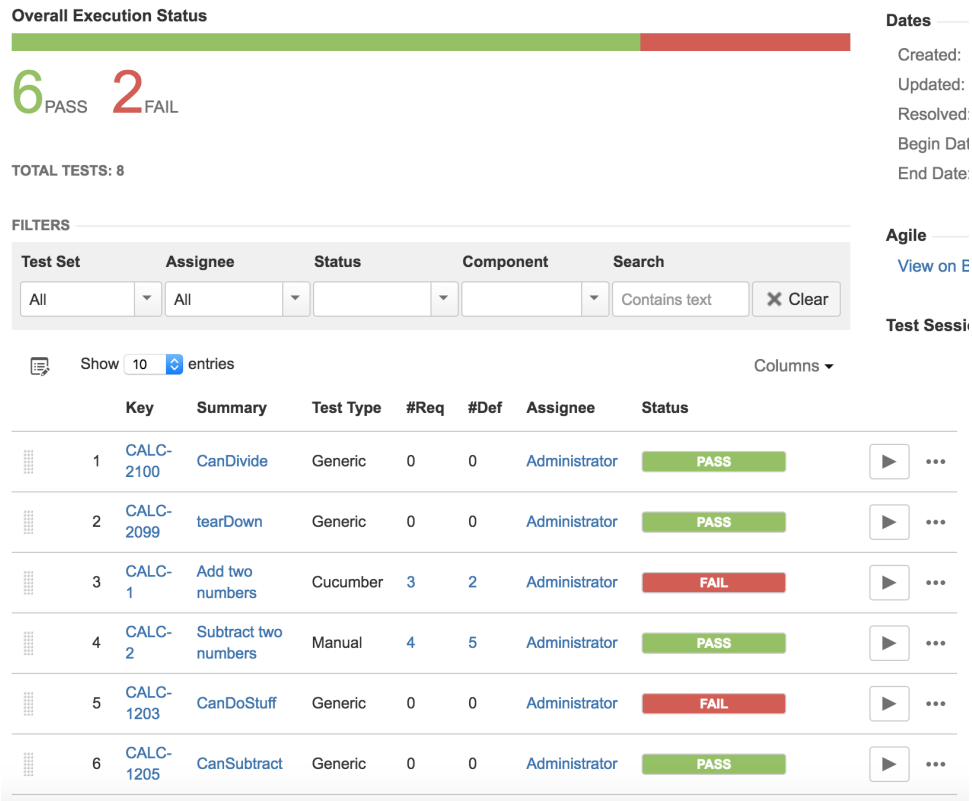
pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.xpand.java</groupId>
    <artifactId>xpand-test</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>

        <!--XRay Properties -->

        <!--IN PROFILE ~.m2/settings.xml-->
        <!--<xray.jiraURL></xray.jiraURL>
        <xray.resultsFormat>JUNIT</xray.resultsFormat>
        <xray.username>admin</xray.username>
        <xray.password>123qwe</xray.password>-->

        <xray.projectKey>CALC</xray.projectKey>
        <!--
        <xray.testExecKey></xray.testExecKey>
        <xray.testPlanKey></xray.testPlanKey>
        <xray.testEnvironments></xray.testEnvironments>
        <xray.revision></xray.revision>
        -->

        <xray.surefire.location>${basedir}/target/surefire-reports</xray.surefire.location>
        <!--End Xray Properties -->

    </properties>
```

```
    <build>
        <plugins>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <debug>true</debug>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>

            <plugin>

                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>2.20.1</version>

                <configuration>
                    <testFailureIgnore>true</testFailureIgnore>

                    <suiteXmlFiles>
                      <suiteXmlFile>testng.xml</suiteXmlFile>
                    </suiteXmlFiles>

                    <properties>
                        <property>
                            <name>reporter</name>
                            <value>org.testng.reporters.XMLReporter:generateTestResultAttributes=true,
generateGroupsAttribute=true</value>
                        </property>
                    </properties>

                </configuration>
             </plugin>




        </plugins>
    </build>

    <dependencies>
        <dependency>
          <groupId>org.testng</groupId>
          <artifactId>testng</artifactId>
          <version>7.4.0</version>
          <scope>test</scope>
        </dependency>

        <dependency>
                <groupId>com.xpandit.xray</groupId>
                <artifactId>xray-maven-plugin</artifactId>
                <version>1.0.0</version>
                    <scope>test</scope>
            </dependency>
    </dependencies>

    <repositories>
                   <repository>
                          <id>xpandit</id>
                       <name>xpand-releases</name>
                    <url>http://maven.xpand-it.com/artifactory/releases</url>
                    <releases>
                            <enabled>true</enabled>
                    </releases>
            </repository>
    </repositories>


    <reporting>
```

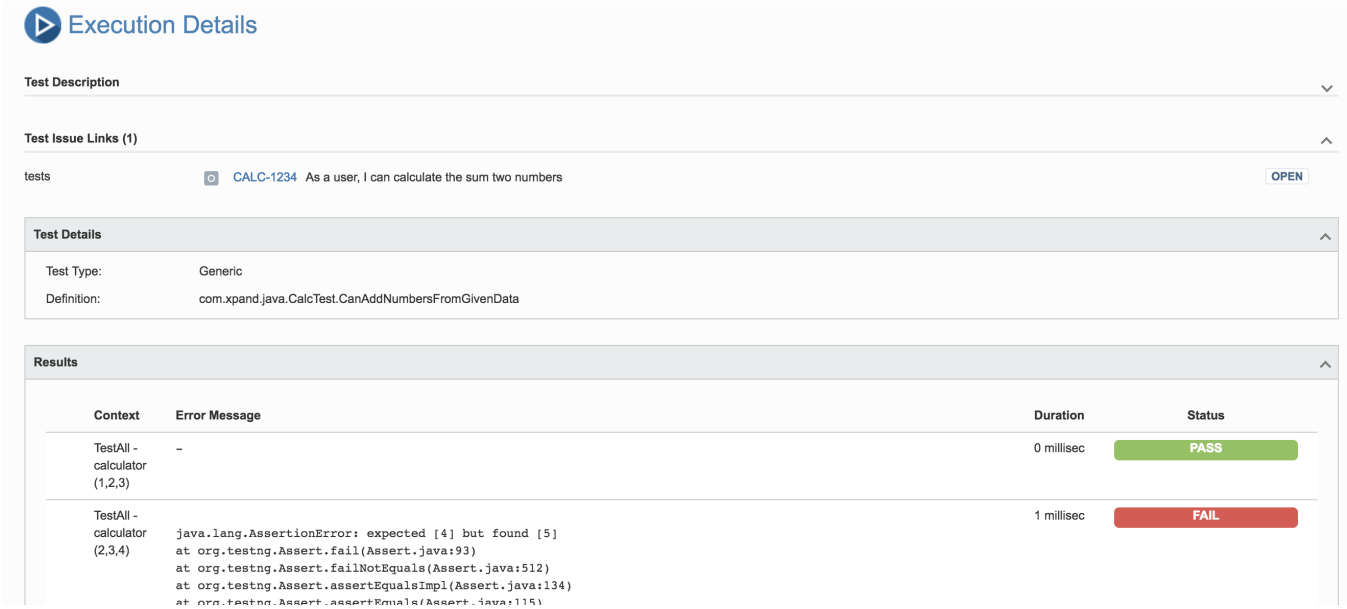
```
        <plugins>
            <plugin>
                <artifactId>maven-surefire-report-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>com.xpandit.xray</groupId>
                <artifactId>xray-maven-plugin</artifactId>
                <version>1.1.0</version>
            </plugin>
        </plugins>
    </reporting>


</project>
```

After successfully running the tests and generating the TestNG XML report (e.g. testng-results.xml), it can be imported to Xray (either by the REST API or through the **Import Execution Results** action within the Test Execution).



TestNG's tests are mapped to Generic Tests in Jira, and the **Generic Test Definition** field contains the namespace, the name of the class, and the method name that implements the Test case.

The Execution Details of the Generic Test contains information about the context, which in this case corresponds to the Test case method, along with the different input values that were validated.



It can also be seen that the Test "CanAddNumbersFromGivenData" was automatically linked to the sum requirement (i.e., the user story "CALC-1234").

References

- http://testng.org/doc/documentation-main.html
- https://github.com/bitcoder/xray-testng-extensions
- Taking advantage of TestNG XML reports