TESTNG CUSTOM REPORTER

In this section, we will cover, with an example, the method of writing your custom reporter and attaching it to TestNG. To write a custom reporter class, our extension class should implement the IReporter interface. Let's go ahead and create an example with the custom reporter.

Create Test Case Class

Create a java class, say, SampleTest.java in **C:\ > TestNG_WORKSPACE**.

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

public class SampleTest {
    @Test
    public void testMethodOne(){
        Assert.assertTrue(true);
    }

    @Test
    public void testMethodTwo(){
 Assert.assertTrue(false);
    }

    @Test(dependsOnMethods={"testMethodTwo"})
        public void testMethodThree(){
        Assert.assertTrue(true);
    }
}
```

The preceding test class contains three test methods out of which *testMethodOne* and *testMethodThree* will pass when executed, whereas *testMethodTwo* is made to fail by passing a *false* Boolean value to the Assert.assertTrue method, which is used for truth conditions in the tests.

Create Custom Reporting Class

Create another new class named CustomReporter.java in **C:\ > TestNG_WORKSPACE**.

```
import java.util.List;
import java.util.Map;

import org.testng.IReporter;
import org.testng.ISuite;
import org.testng.ISuiteResult;
import org.testng.ITestContext;
import org.testng.xml.XmlSuite;

public class CustomReporter implements IReporter{
    @Override
    public void generateReport(List xmlSuites, List suites,
        String outputDirectory) {
        //Iterating over each suite included in the test
        for (ISuite suite : suites) {
            //Following code gets the suite name
            String suiteName = suite.getName();
     //Getting the results for the said suite
     Map suiteResults = suite.getResults();
     for (ISuiteResult sr : suiteResults.values()) {
         ITestContext tc = sr.getTestContext();
         System.out.println("Passed tests for suite '" + suiteName +
                 "' is:" + tc.getPassedTests().getAllResults().size());
         System.out.println("Failed tests for suite '" + suiteName +
                 "' is:" +
                 tc.getFailedTests().getAllResults().size());
```

```
            System.out.println("Skipped tests for suite '" + suiteName +
                  "' is:" +
                  tc.getSkippedTests().getAllResults().size());
         }
          }
     }
}
```

The preceding class implements the *org.testng.IReporter* interface. It implements the definition for the method *generateReport* of the *IReporter* interface. The method takes three arguments:

- xmlSuite, which is the list of suites mentioned in the testng XML being executed.
- suites, which contains the suite information after the test execution. This object contains all the information about the packages, classes, test methods, and their test execution results.
- outputDirectory, which contains the information of the output folder path, where the reports will be generated.

Create testng.xml

Create testng.xml in **C:\ > TestNG_WORKSPACE** to execute test case*s*.

```
<?xml version="1.0" encoding="UTF-8"?>
<suite name="Simple Reporter Suite">
  <listeners>
    <listener class-name="CustomReporter" />
  </listeners>

  <test name="Simple Reporter test">
    <classes>
      <class name="SampleTest" />
    </classes>
  </test>
</suite>
```

Compile the SampleTest, CustomReporter classes using `javac`.

```
C:\TestNG_WORKSPACE>javac CustomReporter.java SampleTest.java
```

Now, run testng.xml.

```
C:\TestNG_WORKSPACE>java -cp "C:\TestNG_WORKSPACE" org.testng.TestNG testng.xml
```

Verify the output.

```
===============================================
Simple Reporter Suite
Total tests run: 3, Failures: 1, Skips: 1
===============================================

Passed tests for suite 'Simple Reporter Suite' is:1
Failed tests for suite 'Simple Reporter Suite' is:1
Skipped tests for suite 'Simple Reporter Suite' is:1
```

The preceding example shows a simple custom reporter, which prints the number of failed, passed, and skipped tests on the console for each suite included in the said test execution. Reporter is mainly used to generate the final report for the test execution. The extension can be used to generate XML, HTML, XLS, CSV, or text format files depending upon the report requirement.