TESTNG CUSTOM LOGGING

We had earlier read about the different options that TestNG provides for logging and reporting. Now, let's learn how to start using them. To start with, we will write a sample program in which we will use the ITestListener interface for logging purposes.

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 Logging Class

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

```
import org.testng.ITestResult;
import org.testng.TestListenerAdapter;

public class CustomListener extends TestListenerAdapter{
    private int m_count = 0;

    @Override
    public void onTestFailure(ITestResult tr) {
        log(tr.getName()+ "--Test method failed\n");
    }

    @Override
    public void onTestSkipped(ITestResult tr) {
        log(tr.getName()+ "--Test method skipped\n");
    }

    @Override
    public void onTestSuccess(ITestResult tr) {
        log(tr.getName()+ "--Test method success\n");
    }

    private void log(String string) {
        System.out.print(string);
        if (++m_count % 40 == 0) {
     System.out.println("");
        }
```

```
    }

}
```

The above class extends *TestListenerAdapter*, which implements *ITestListener* with empty methods. Hence, no need to override other methods from the interface. You can implement the interface directly, if you prefer so.

Create testng.xml

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

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

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

Compile the SampleTest, CustomListener classes using `javac`.

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

Now, run the testng.xml.

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

Verify the output.

```
testMethodOne--Test method success
testMethodTwo--Test method failed
testMethodThree--Test method skipped

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

We created a custom logger class, which implements the ITestListener interface and attached itself to the TestNG test suite as a listener. Methods of this listener class are invoked by TestNG when test started, at test fail, at test success, and so on. Multiple listeners can be implemented and added to the test suite execution, TestNG will invoke all the listeners that are attached to the test suite.

Logging listeners are mainly used when we need to see the continuous status of the test execution when the tests are getting executed.