**TestNG Listeners:-**

1. Listener is a feature available in TestNG which is used to monitor the test execution in runtime and it generates events whenever testing test script is getting pass or fail.
2. Listener feature contains several interfaces and annotations linke
3. **ItestListener**-> INTERFACE
4. **ITestResult**-> INTERFACE
5. **Reporter**-> class
6. **@Listeners**-> annotations

**@Listeners**:-

It is an advanced annotation available in TestNG which is used to monitor the testscript execution during runtime and generates event wherever test script is getting pass or fail

**ITestListner**:-

1. It is an inbuilt interface available in TestNG
2. As per the rule Listner Implementation class should implements ITestListner Interface and should override onTestFailure method
3. Listener implementation class is used to receive the failure event from @Listener annotations present in every testscript file and then perform appropriate action based on result (pass and fail)
4. In real framework Listener implementation class will be used to take screenshot whenever test script is getting failed to achieve this we have to write screenshot program inside onTestFailure()

**ITestResult**:

It is an interface available in TestNG and it is used as an argument for every Listener implementation class on test failure method which is used to receive the execution result during runtime.

**Reports**: -

1. Whenever we execute 1000’s of test script in a batch to track the execution reporting feature will be very helpful to know the status of the application and thses reports acts as a proof of document for automation execution process
2. Without reporting we cannot conclude the result
3. Whenever we get a new build we should execute all automation testscripts and we should generate automation reports
4. This automation report will be shared with customers
5. There are two types of reports:-
6. **High level reports**
7. **Low level reports**
8. **High level report**:-
9. Testng will automatically generate high level report which contains number of test scripts passed, skipped, failed, time
10. High level report helps us to know the status of the application and we can submit this report to the customer
11. To get graphical reports like pie chart bar chart we go for third party reporting tools like extent reports, log4j, reporting etc..
12. **Low Level Reports**:-
13. This help us to debug the testscripts whenever any test scripts getting failed, looking at this report we can guess the failure
14. TestNg also provide low level report feature
15. To generate low level report, we should take a help of

Reporter.log (“message”, true);

**Generate Extent Report**:

ItestListener Implementation for extent report

------------------------------------------------------------------

public class **ItestlistenerIMP** implements **ITestListener** {

ExtentReports report;

ExtentTest test;

@Override

public void onFinish(ITestContext arg0) {

report.flush();

}

@Override

public void onStart(ITestContext arg0) {

ExtentHtmlReporter htmlReport=new ExtentHtmlReporter(new File(".\\ExtentReport\\report.html"));

htmlReport.config().setDocumentTitle("Extent Report");

htmlReport.config().setTheme(Theme.DARK);

htmlReport.config().setReportName("Functional Test");

report=new ExtentReports();

report.attachReporter(htmlReport);

report.setSystemInfo("TestURL", "https://example.com");

report.setSystemInfo("Platform", "Windows 10");

report.setSystemInfo("Reporter Name", "Nithesh");

}

@Override

public void onTestFailedButWithinSuccessPercentage(ITestResult arg0) {

// TODO Auto-generated method stub

}

@Override

public void onTestFailure(ITestResult result) {

test.log(Status.FAIL, result.getMethod().getMethodName()+" is failed");

test.log(Status.FAIL, result.getThrowable());

try {

String path = WebDriverUtiity.takeScreenshot(BaseClass.staticDriver, result.getMethod().getMethodName());

test.addScreenCaptureFromPath(path);

} catch (Throwable e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

@Override

public void onTestSkipped(ITestResult result) {

test.log(Status.SKIP, result.getMethod().getMethodName()+" is skipped");

test.log(Status.SKIP, result.getThrowable());

}

@Override

public void onTestStart(ITestResult result) {

test=report.createTest(result.getMethod().getMethodName());

}

@Override

public void onTestSuccess(ITestResult arg0) {

test.log(Status.PASS, arg0.getMethod().getMethodName()+" is passed");

}

}

Adding testcases with listener

-----------------------------------------------------------------

public class TestCase1 extends BaseClass{

@Test

public void testcase1() {

System.out.println("Testcase1 is executed");

}

}

------------------

public class Testcase2 extends BaseClass{

@Test

public void testcase2() {

driver.findElement(By.name("user\_name")).sendKeys("Admin");

driver.findElement(By.name("user\_password")).sendKeys("manager");

Assert.assertTrue(false);

}

}

------------------

public class Testcase3 extends BaseClass {

@Test

public void testcase3() {

test=report.createTest("testcase3");

System.out.println("testcase3 is executed");

}

}

------------------

Running the suite by adding listener tags

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">

<suite name="Suite">

<listeners>

<listener class-name="extentReportCreation.ItestlistenerIMP"/>

</listeners>

<test thread-count="5" name="Test">

<classes>

<class name="extentReportCreation.TestCase1"/>

<class name="extentReportCreation.Testcase2"/>

<class name="extentReportCreation.Testcase3"/>

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

----------------------------------------------