TestNG Listeners

Ostan Dsouza

CodeCraft Technologies

10/06/2020

Table of Contents

Chapter 1: **Introduction**

TestNG provides the @Listeners annotation which listens to every event that occurs in a selenium code. Listeners are activated either before the test or after the test case. It is an interface that modifies the TestNG behaviour. Listeners "listen" to the event defined in the selenium script and behave accordingly. It is used in selenium by implementing Listeners Interface. In simpler terms, If the event matches the event for which we want the listener to listen, it executes the code, which ultimately results in modifying the default behaviour of TestNG.

TestNG Listeners in Selenium WebDriver can be implemented at two levels:

1. **Class level:** In this, you implement listeners for each particular class, no matter how many test cases it includes.
2. **Suite level:** In this, you implement listeners for a particular suite which includes several classes of test cases.

Section 1.1: **Types of Listeners**

There are many types of listeners which allows you to change the TestNG's behaviour.

Below are the few TestNG listeners:

- IAnnotationTransformer
- IAnnotationTransformer2
- IConfigurable
- IConfigurationListener
- IExecutionListener
- IHookable
- IInvokedMethodListener
- IInvokedMethodListener2
- IMethodInterceptor
- IReporter,
- ISuiteListener
- ITestListener

Section 1.1.1: **ITestListener**

ITestListener is the most adopted TestNG listener in Selenium WebDriver. It provides you with an easy to implement interface through a normal Java class, where the class overrides every method declared inside the ITestListener. By using this TestNG listener in Selenium

WebDriver, you can change the default behaviour of your test by adding different events to the methods. It also defines a new way of logging or reporting.

The following are some methods provided by this interface:

- **onStart:** This method is invoked before any test method gets executed. This can be used to get the directory from where the tests are running.
- **onFinish:** This method is invoked after all tests methods gets executed. This can be used to store information of all the tests that were run.
- **onTestStart:** This method is invoked before any test methods are invoked. This can be used to indicate that the particular test method has been started.
- **onTestSkipped:** This method is invoked when each test method is skipped. This can be used to indicate that the particular test method has been skipped.
- **onTestSuccess:** This method is invoked when any test method succeeds. This can be used to indicate that the particular test method has successfully finished its execution.
- **onTestFailure:** This method is invoked when any test method fails. This can be used to indicate that the particular test method has failed. You can create an event for taking a screenshot which will show where the test has been failed.
- **onTestFailedButWithinSuccessPercentage:** This method is invoked each time the test method fails but is within the success percentage mentioned. To implement this method, we use two attributes as a parameter of test annotation in TestNG, i.e. successPercentage and invocationCount. The successPercentage takes the value of percentage of successful tests and invocationCount denotes the number of times that a particular test method executes. **For example:** @Test(successPercentage =60, invocationCount =5). In this annotation, the success percentage is 60% and the invocation count is 5, which means that out of 5 times, if the test method gets passed at least 3 times ((¾)*100= 60), it will be considered as passed.

Section 1.1.2: **IRetryAnalyzer**

There may be many reasons for a Test case getting failed, may be due to element not found or time out exception or stale element exception etc. Normally in automation after executing scripts/tests, we will check for the results and if the test fails because of above reasons we will re-run then again.

Instead of that we can ask testNG to execute the failed test cases again for X (we can define) number of times and check for the updated results.

To achieve this, we need to implement TestNG IRetryAnalyzer, which basically takes a numeric value is passed to the annotation which specifies the number of times a failed test needs to be rerun.

Due to static nature of Annotations, recompilation is needed when you want to change values. You can override this behavior at runtime with **IAnnotationTransformer** listener.

IAnnotationTransformer is a TestNG listener which allows you to modify TestNG annotations and configure them further during runtime.

Section 1.1.3: **IAnnotationTransformer**

Annotations are static in nature by design, so any change in the values require recompilation of source files. Since TestNG relies heavily on annotations, it would be nice if one can override its behavior at runtime. This is exactly what TestNG allows you to do using its annotation transformation framework. IAnnotationTransformer is an interface that provides a "transform" method which gets invoked by TestNG to modify the behaviour of the test annotation method in our test class. The transform method provides various parameters:

- **annotation:** The annotation that is read from the test class.
- **testClass:** If the annotation found on a class, this parameter would represent that same class.
- **testConstructor:** If the annotation found a constructor, this parameter would represent that same constructor.
- **testMethod:** If the annotation found a method, this parameter would represent that same method.

Section 1.1.4: **IReporter**

This TestNG listener in Selenium WebDriver provides an interface which helps you to customize the test report generated by TestNG. It provides the generateReport method which gets invoked after the execution of all the suites. The method further contains three parameters:

- **xmlSuite:** it provides you with a list of multiple suites presented in the testng xml file that goes under execution.
- **suites:** This object represents a great deal of information about the classes, packages, test execution result, along with all the test methods. Basically, it represents detailed information around the suite after the final execution.
- **outputDirectory:** contains the output folder path where the report gets generated.

Section 1.1.5: **IMethodInterceptor**

IMethodInterceptor listener alters or modifies the list of tests that will be executed by TestNG. By using the IMethodInterceptor listener, we run the intercept method, which returns the list of methods that will be run by TestNG. Now the TestNG will run the methods in the same order that returned in the list. The IMethodInterceptor listener in TestNG contains only one method:

This method is used:

- to return the list of IMethodInstance, after the execution of TestNG.

- to sort the list of test methods.

intercept: This method returns the list of methods that will be executed by TestNG.

Chapter 2: Usages

Below is the detailed explanation on how these listeners are used in current framework.

Section 2.1.1: ITestListener

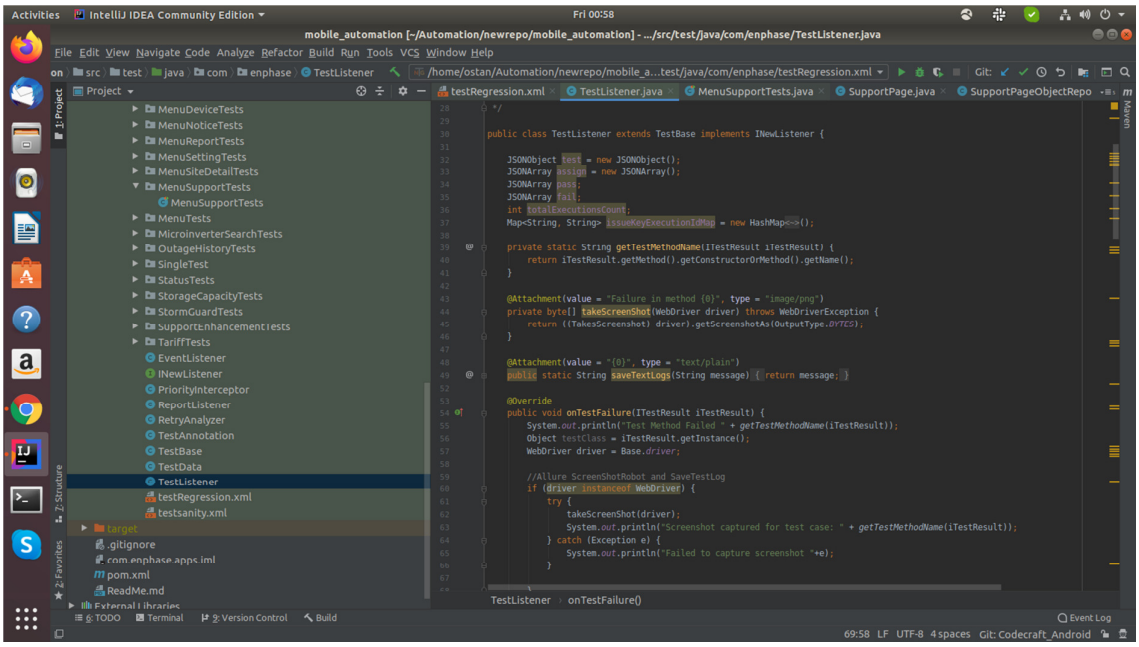
Capture Failure Screenshots:

We achieve this by making use of **OnTestFailure** provided by ITestListener, so that when testcase is failed only then screenshot will be captured.

Zephyr Integration:

We make use of all different stages provided by ITestListener to achieve this.

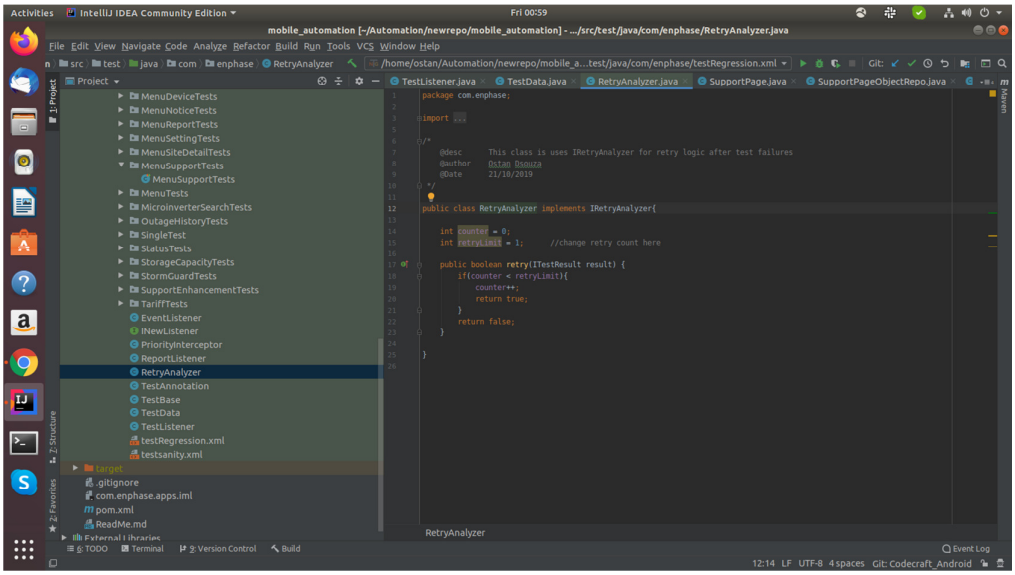
a) *onStart*: In this stage all the global variables are declared.
b) *OnTestStart*: In this stage we add testcases to the already create cycle.
c) *OnTestFailure*: In this stage we get the execution Id's for all testcases and store execution Id's failed testcases in an array variable.
d) *OnTestSuccess:* In this stage we get the execution Id's for all testcases and store execution Id's success testcases in an array variable.
e) *OnTestSkipped*: In this stage we get the execution Id's for all testcases and store execution Id's skipped testcases in an array variable.
f) *OnFinish*: In this stage all the testcases are executed by making use of execution Id's from failed, success and skipped stages.



Section 1.1.2: IRetryAnalyzer

Retry Failed Testcases:

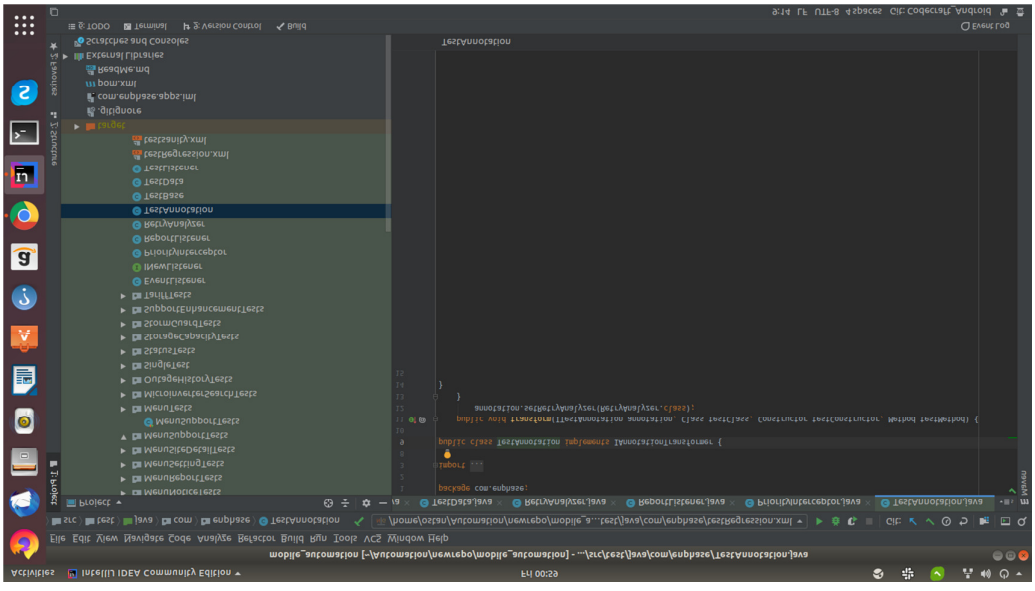
We achieve this by making use of IRetryAnalyser listener that helps us re-run failed testcases based on the input specified to this. This helps in eliminating flaky test failures.



Section 2.1.3: IAnnotationTransformer

Assigning retry flag for all TC:

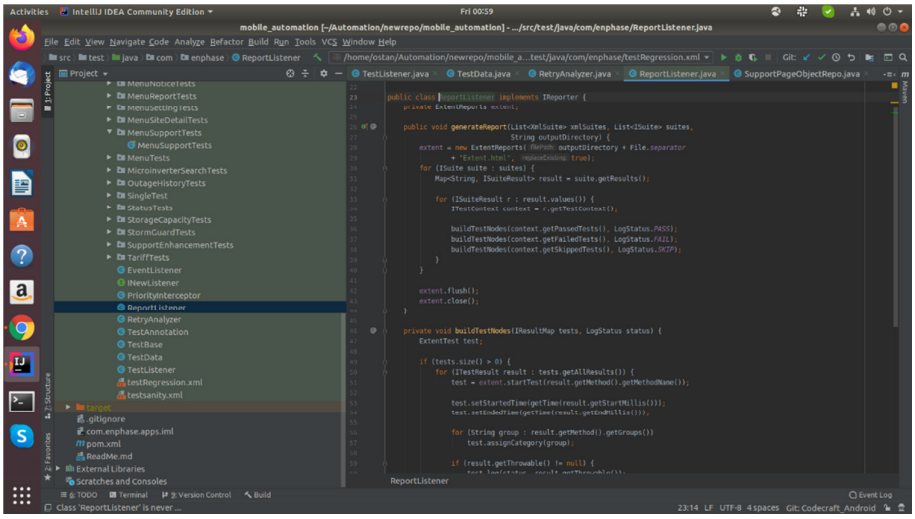
Along with IRetryAnalyser listener, we make use of IAnnotationTransformer listener to extender retry functionality for all testcases in testing.xml. As this listener programmatically adds retry annotation to your test methods during run time.



Section 2.1.4: **IReporter**

Creation of extent report:

We use IReport listener to listener you need to implement if you want to generate a report after all the suites are run. This TestNG listener in Selenium WebDriver provides an interface which helps you to customize the test report generated by TestNG. It provides the generate report method which gets invoked after the execution of all the suites.



Section 1.1.5: **IMethodInterceptor**

prioritizing test cases:

This listener alters or modifies the list of tests that will be executed by TestNG. By using the IMethodInterceptor listener, we run the intercept method, using which we can run testcases in the order they're written initially instead of testcases being run on random order.