**JAVA PROGRAMING AND SELENIUM**

**WEEK 7 – TESTNG AND JUNIT**

**10/22/2022**

**-   Compare Junit vs TestNG**

|  |  |  |
| --- | --- | --- |
| **Basis of** | **Junit** | **TestNG** |
| Developed by | JUnit was developed by Kent Beck, David Saff, Erich Gamma. Erich Gamma, and Kris Vasudevan. | TestNG is a testing framework that was developed by Cédric Beust. |
| Open-Source | JUnit is an open-source framework used to trigger and write tests. | TestNG is a Java-based framework that is an upgraded option for running tests. |
| Parallel test Runs | JUnit does not support to run parallel tests. | TestNG can run parallel tests. |
| Supports Annotation | It does not support advanced annotation. | It supports advanced annotation. |
| Dependency tests | The dependency tests are missing in JUnit. | Dependency tests are present in TestNG |
| Grouping tests | Grouping tests together is not possible in JUnit. | Tests can be grouped together and run parallel. |
| Ease of Use | Running tests need a certain dependency on JUnit. | Writing tests and configuring them is easy in TestNG than JUnit. |

**-  What is TestNG?**  
TestNG is also a Java framework that facilitate to perform software tests in Java. It is a framework that runs the tests in classes. It makes classes for corresponding test and then process them. TestNG is an advanced framework that overcomes limitations found in JUnit. It is also being considered a flexible tool to performs tests as it uses the same Classes to run its all tests and manages threads to run procedures which makes overall functioning of the checking tests fast.  
  
**-   Why TestNG is popular?**

TestNG offers an easy way to run parallel execution. You can simply select the option and choose whether to run parallel execution based on methods, classes, or tests. You can also specify the thread count. When you run the XML file, it will automatically execute the tests in parallel

**- How TestNG is used?**

Writing a test is typically a three-step process:

* Write the business logic of your test and insert TestNG annotations in your code.
* Add the information about your test (e.g. the class name, the groups you wish to run, etc…) in a testng.xml file or in build.xml.
* Run TestNG.xml.

**·   Explain  Maven Dependencies**

In Maven, a dependency is just another archive (JAR, ZIP, and so on) which our current project needs in order to compile, build, test, and/or run. These project dependencies are collectively specified in the pom.xml file, inside of a <dependencies> tag.

When we run a maven build or execute a maven goal, these dependencies are resolved and then loaded from the [local repository](https://howtodoinjava.com/maven/change-local-repository-location/).

If these dependencies are not present in the local repository, then Maven will download them from a remote repository and cache them in the local repository.

**·  Explain TestNG Annotation**

Annotations generally represent a note or a comment on a diagram, etc., to convey its meaning. TestNG also uses them for the exact reason. TestNG annotations are the code written inside your source test code logic to manage the flow of the execution of tests? Therefore, it is vital to annotate your processes in TestNG to execute the tests. TestNG will ignore the process that does not contain an annotation since it won’t know when to run it.

Annotations in TestNG can be comprehended as distinct pieces of code integrated within the logic of a program to manage the sequential execution of the test methods. In other words, TestNG Annotations assist in deciding the sequence of the tests made to execute. These annotations are extracted from Java language and are a critical component of Test Next Generation (TestNG), an automation framework that Selenium uses.

A TestNG annotation begins from the symbol “@,” and the annotation name follows after that. Apart from the header file and ‘@’ symbol, you require nothing to execute TestNG annotations.

**Benefits Of TestNG Annotations**

* TestNG annotations aid in the creation of a comprehensive and complete test report that includes information on the number of test cases run, passed test cases, failed test cases, and skipped test cases.
* It enables the simple grouping of multiple test cases using the testing.xml file. Moreover, one can prioritize the execution of test cases.
* TestNG Annotations can implement the same test case multiple times by using the keyword, ‘invocation count.’
* Annotations in TestNG are easily understandable.
* TestNG Annotations provide simple coding of tests to eliminate the requirement for a static primary method.
* TestNG assists in the grouping of test cases and their execution.
* TestNG offers the ability to prioritize test cases based on the demands of the programmer.
* TestNG helps to execute the test cases in a parallel manner.
* TestNG provides an ideal way of report generation according to the test results.
* TestNG is strongly typed, which specifies the capability of solid error checking.
* TestNG can also link to external data sources.
* To execute test cases, there is no requirement to follow a pattern or format.
* Additional parameters can be passed to TestNG annotations.
* In the case of TestNG annotations, there is no requirement to extension of any test classes.
* TestNG Annotations are highly typed, which means that errors are detected at compile time.

**· Before Test and Method**

|  |  |
| --- | --- |
| [@BeforeTest](https://www.javatpoint.com/testng-beforetest-annotation) | The @BeforeTest annotated method will be executed before the execution of all the test methods of available classes belonging to that folder. |
| [@BeforeMethod](https://www.javatpoint.com/testng-beforemethod-annotation) | The @BeforeMethod annotated method will be executed before each test method will run. |

**· After Test and Method**

  
  
**- @Test** 

|  |  |  |
| --- | --- | --- |
| **@Test** | | **Marks a class or a method as part of the test.** |
|  | alwaysRun | If set to true, this test method will always be run even if it depends on a method that failed. |
|  | dataProvider | The name of the data provider for this test method. |
|  | dataProviderClass | The class where to look for the data provider. If not specified, the data provider will be looked on the class of the current test method or one of its base classes. If this attribute is specified, the data provider method needs to be static on the specified class. |
|  | dependsOnGroups | The list of groups this method depends on. |
|  | dependsOnMethods | The list of methods this method depends on. |
|  | description | The description for this method. |
|  | enabled | Whether methods on this class/method are enabled. |
|  | expectedExceptions | The list of exceptions that a test method is expected to throw. If no exception or a different than one on this list is thrown, this test will be marked a failure. |
|  | groups | The list of groups this class/method belongs to. |
|  | invocationCount | The number of times this method should be invoked. |
|  | invocationTimeOut | The maximum number of milliseconds this test should take for the cumulated time of all the invocationcounts. This attribute will be ignored if invocationCount is not specified. |
|  | priority | The priority for this test method. Lower priorities will be scheduled first. |
|  | successPercentage | The percentage of success expected from this method |
|  | singleThreaded | If set to true, all the methods on this test class are guaranteed to run in the same thread, even if the tests are currently being run with parallel="methods". This attribute can only be used at the class level and it will be ignored if used at the method level. Note: this attribute used to be called sequential (now deprecated). |
|  | timeOut | The maximum number of milliseconds this test should take. |
|  | threadPoolSize | The size of the thread pool for this method. The method will be invoked from multiple threads as specified by invocationCount. Note: this attribute is ignored if invocationCount is not specified |

**· Setup methods**  
When you build the test class, there will be certain Java methods annotated with @Test, which tells TestNG that the method is a test and should be run. Those tests will run in random order by default except, if you use a dependent method, a sequential naming scheme, or a priority attribute. That will force the tests to run in a specific order.

For all the methods in a suite of tests, there will be common actions that need to be executed before each suite, test, groups, class, or methods, and instead of calling the same setup method in each class or test, for instance, it makes sense to do them in one place. Using the TestNG setup annotations will allow users to execute a routine in a central place.

**·    Write a Simple Example  
  
  
·    What’s TestNG Suite?**

A test suite is a collection of test cases intended to test a behavior or a set of behaviors of software program. In TestNG, we cannot define a suite in testing source code, but it is represented by one XML file, as suite is the feature of execution. It also allows flexible configuration of the *tests* to be run. A suite can contain one or more tests and is defined by the <suite> tag.

<suite> is the root tag of your testng.xml. It describes a test suite, which in turn is made of several <test> sections.

**·    How to parameterize TestNG Tests?**

Parameterized tests allow developers to run the same test over and over again using different values.

TestNG lets you pass parameters directly to your test methods in two different ways −

* With testng.xml
* With Data Providers

**·    TestNG Pros and Cons**

TestNG is an open-source testing framework and inspired by the other two prominent frameworks – JUnit and NUnit with some new additional functionalities. Deriving from its name where NG stands for Next Generation, this framework doesn’t disappoint to prove its strength in testing. The process of testing here is also not very complex where we can test the scenarios by requesting through the framework to test the databases or maybe the front ends. An HTML report can also be extracted which is quite useful for the tests being performed.

**Pros:**

* Support for parallel testing
* Supports log generation
* Capable of creating post-testing HTML reports
* Underlying test cases can be grouped together
* Test case execution’s priority can be set

**Cons:**

* Setting up TestNG takes more time
* Not recommended if you don’t need to prioritize the test cases.

**-  How to generate TestNG Report, show steps**

Add ReportNG dependencies

Create a java class

Create a Test Class

In Test class follow 3 steps process

1) Declare Class

2) Instantiate Object

3) Add Test Methods with Annotation @Test

testAdd()

testMul()

Import Package

* Save the Test program
* Run the Test Program

**- How to generate code coverage Report, show steps**

Add Codecoverage plugin::**JaCoCo**

Add the following section in pom.xml before the </project>

<build> <plugins> <plugin> **<groupId>org.jacoco</groupId> <artifactId>jacoco-maven-plugin</artifactId> <version>0.7.7.201606060606</version> <executions>** <execution> <goals> <goal>prepare-agent</goal> </goals> </execution> <execution> <id>report</id> <phase>prepare-package</phase> <goals> <goal>report</goal> </goals> </execution> </executions> </plugin> </plugins> </build>

Now perform **maven clean**

If needed to add goa**l site:site :: this can help you to generate reports**

**After few run, you will see the code coverage option in the menu**

Now see the code coverage report

It shows GREEN -- the lines are tested , RED -- Unit test has not touched (means testing is needed)

If you double click on Student class, you will see RED and GREE color to show - what line is validated by TestnNG

Now add a condition, if the code coverage i**s not above 60% then build should fail**

We can update the **<build> section with the following with line coverage rules**

<build> <plugins> <plugin> <groupId>org.jacoco</groupId> <artifactId>jacoco-maven-plugin</artifactId> <version>0.7.7.201606060606</version> <executions> <execution> <goals> <goal>prepare-agent</goal> </goals> </execution> <execution> <id>report</id> <phase>prepare-package</phase> <goals> <goal>report</goal> </goals> </execution> <execution> <id>jacoco-check</id> <goals> <goal>check</goal> </goals> <configuration> <rules> <rule> <element>PACKAGE</element> **<limits> <limit> <counter>LINE</counter> <value>COVEREDRATIO</value> <minimum>0.60</minimum> </limit> </limits>** </rule> </rules> </configuration> </execution> </executions> </plugin> </plugins> </build>

Perform Maven Clean & build, it will fail

Change the coverage requirements to 0.30 and then perform maven clean and build

T Now create multiple TesNG Test cases then create TestSuite

Create a new **TestNG TestSuite** -- which can run all Testcases together

This time Build will be successful

**1) Create a Rectangle class with Fields (length, width ) operation getArea() , getPerimeter()**

Verify the class using the TestNG or Junit framework

2) **Create a School class , has Students and Instructors**

\*\* Has A Relationship (means use ArrayList in School class)