**TestNG:**

* TestNG is a unit testing framework which is inspired from Junit.
* TestNG stands for Test Next Generation.
* Basically, TestNG used to control the flow of execution.
* Sequential and parallel execution is done without writing code in TestNG.
* To use TestNG in our code, we must have dependencies (Jar File – 6.14.2) and plugin (install TestNG from eclipse marketplace).

**Sequential Execution:**

* Static is not used in TestNG.
* Return type is not supported by TestNG.
* TestNG Annotation are used to control the next method to be executed in the test script - @Test.

**Parallel Run:**

* A process where multiple tests are executed simultaneously/in parallel in different thread processes.
* To run code parallelly then we have to go to XML file. This file will have attribute called thread count in the top section. This thread count is responsible for parallel execution.
* If thread count is -1 then it will take default value as 5.
* If thread count is 1 then at a time one browser is executed based on the computer spec.

**Priority:**

* TestNG priority attribute helps in executing the test cases in that order.
* @Test (Priority = -1)
* Priority will take negative, positive and zero as its value. Basically, an int type.
* The default value is 0.
* Suppose if two methods have same priority then it will go by ASCII order.

**Depends On:**

* “dependsOnMethods” is an attribute comes inside Test annotation.
* The name itself says that it will depend on a test that should be executed first before running it.
* @Test(dependsOnMethods = “Signup”). Here, Signup is the parent method.
* We can able to specify the priority attribute as well. But it will take higher preference to dependsOnMethods. E.g., @Test(dependsOnMethods = “Signup”, priority = 1).
* If we fail parent methods by exception then automatically depended methods will be skipped.
* Another way to assign the method name is to give class name with method followed by package name. E.g., package.class.method.

**Invocation Count and Invocation timeout:**

* Invocation count defines the number of times a test method is performed.
* Invocation timeout is similar to wait which used to determine the maximum time permitted for a test method to execute.
* Generally, invocation count is also an attribute which makes the test to run multiple times according the value specified to it.
* @Test(invocationCount = 3, invocationTimeout = 2000)
* Invocation timeout will work only when there is mentioned the invocation count.

**Always Run, Enable, Description:**

* These all three are also attributes.
* Always Run is used when we want to run a test even its depended methods fail. (Hint: dependsOnMethods)
* If the value of enabled is true then it will execute the test. false – test case will not be executed
* @Test(alwaysRun = true). Generally, this is used to avoid skip. By default, it is false.
* @Test(enabled = false). By default, it is true.
* @Test(description = “This method is used to do the signup”)
* High priority goes to enabled when both alwaysRun and enabled are mentioned. (Interview Q.)

**ThreadPoolSize, Timeout and Expected Exception:**

* These all three are also attributes.
* @Test(threadPoolSize=x) annotation defines the number of threads to be used while running a test method.
* Timeout is similar to wait which used to determine the maximum time permitted for a test method to execute.
* Expected Exception specify the type of exceptions that are expected to be thrown by a test during execution.
* Thread Pool Size works only when there is invocation count.
* @Test(invocationCount = 3, threadPoolSize = 2)
* @Test(timeOut=5000).
* @Test ( expectedExceptions = { IOException.class, NullPointerException.class } )

**Groups:**

* Groups allow you to perform groupings of different test methods.
* There are two steps to execute groups. First step is to provide the group attribute in the Test annotation. Second step is to write code in xml file as below.
* @Test(groups= (“smoke”,”reg”}).
* <groups>

<run>

<include name = “smoke”> </include>

<exclude name = “reg”></exclude>

</run>

</groups>

* We can implement this group tag either suite level or test level according to our preference.
* Exclude has the high priority.

**Parameters:**

* Parameters is an annotation which used to send the test data to specified method.
* Parameter concept allows us to run the same test over and over again with different set of test data.
* @Test

@Parameters("myName")

public void parameterTest(String myName) {

System.out.println("Parameterized value is : " + myName);

}

* <parameter name = "myName" value="manisha"/>

<parameter name = "myName" value="Ram"/> 🡪 Test level (Will run two times)

**Data Provider:**

* Data provider is a method annotation.
* @DataProvider(name = "test1")

public static Object[][] primeNumbers() {

return new Object[][] {{2, true}, {6, false}, {19, true}, {22, false}, {23, true}};

}

* @DataProvider(name = “login”, indicies = {3}) 🡪 Third set will be executed.
* Indicies = {1,3) 🡪 Three set of data will be executed.
* @DataProvider(parallel = true)
* Data Provider error will occur mismatch when we use both data provider and parameter.
* Higher priority – Data provider.
* @Test(dataProvider = "test1")

public void testPrimeNumberChecker(Integer inputNumber, Boolean expectedResult) {

System.out.println(inputNumber + " " + expectedResult);

Assert.assertEquals(expectedResult, primeNumberChecker.validate(inputNumber));

}