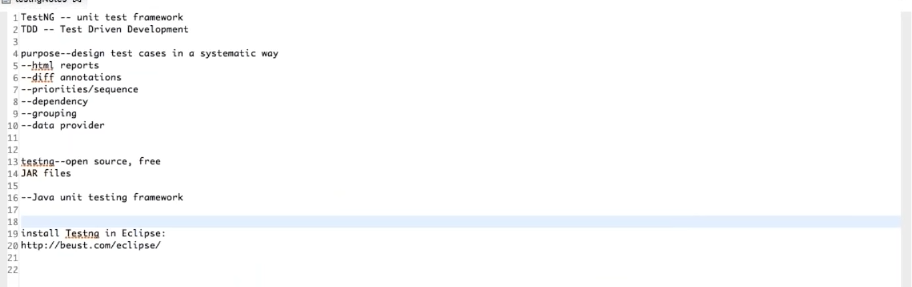
**TestNG – Test Next Generation**

TestNG is a testing framework for the Java programming language created by Cédric Beust and inspired by JUnit and NUnit. Using testing we can achieve from unit testing to integration testing.

* It is also referred as Unit testing framework
* Its Based on TDD , Test Driven Development. First text cases will be written based on the requirement , before application is fully developed and the application will be repeatedly tested against all test cases.
* TestNg is a java framework.



**TestNG Annotation** is a piece of code which is inserted inside a program to control the execution flow of test methods

**Annotations**

Postcondition Annotations – after Precondition Annotations – Before

**@BeforeSuite** - The annotated method will be run before all tests in this suite have run.

**@AfterSuite** - The annotated method will be run after all tests in this suite have run

**@BeforeTest:** The annotated method will be run before any test method belonging to the classes inside the <test> tag is run.

**@AfterTest:** The annotated method will be run after all the test methods belonging to the classes inside the <test> tag have run.

**@BeforeGroups:** This method is guaranteed to run shortly before the first test method that belongs to any of these groups is invoked.

Eg: @BeforeGroups("Group name")

**@AfterGroups:** This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked.

**@BeforeClass:** The annotated method will be run before the first test method in the current class is invoked.

**@AfterClass:** The annotated method will be run after all the test methods in the current class have been run.

**@BeforeMethod:** The annotated method will be run before each test method.

**@AfterMethod:** The annotated method will be run after each test method.

**@Test** : Marks a class or a method as part of the test. Also called **test annotation**

**@DataProvider** : Marks a method as supplying data for a test method.

**@Factory** : Marks a method as a factory that returns objects that will be used by TestNG as Test classes. The method must return Object[].

**@Listeners** : Defines listeners on a test class.

**@Parameters :** Describes how to pass parameters to a @Test method.

**We can have multiple annotation of the same type in a class including @BeforeMethod, @AfterClass**

**@Beforemethod**

**@Beforemethod**

**@test**

**@Aftermethod**

**@Aftermethod**

sequence

@BeforeSuite

@BeforeTest

@BeforeClass

@BeforeMethod

@Test1

@Tes2

@AfterMethod

@AfterClass

@AfterTest

@BeforeSuite

Execution combination----

@BeforeMethod

@Test1

@AfterMethod

@BeforeMethod

@Test2

@AfterMethod

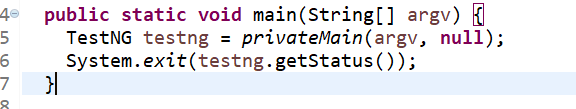


**TestNG invocation** – can be done in below ways,

* Testing.xml file (Eclipse)
* Ant
* Command line

We need to execute testNG through xml (extensible markup language) file .

**We don’t need to write main() because , main() is already defined in the TestNG.class**



**Testing.xml file**

**Suite tag –** is the root tag of your testng.xml, will display the suite name. Should be only one tag per xml file.

* Mandatory attribute ‘name’ and optional attributes like verbose, parallel, thread-count

<suite name=’suite1’ verbose=’1’> …..</suite>

Verbose - it is used to define the amount of logging to be performed on the console.

**Test tag –** used to execute the classes mentioned in this tag.will display the Test name. Can be more than one test tag per xml file. Mandatory attribute ‘name’ and optional attributes like preserve-order

preserve-order – it determine the order of execution of the test methods if it is false then random order will be maintained . By default it is true so execution will be in the order that we created.

**Groups tag –** used to specify which group need to be included or excluded during run. Can be specified inside the test tag or outside the test tag.

<groups>

<run>

<exclude name="brokenTests" />

<include name="checkinTests" />

</run> </groups>

**Classes tag :** specified inside the test tag

|  |
| --- |
| <classes> |

|  |
| --- |
| <class name="test.IndividualMethodsTest"> |

|  |
| --- |
| <methods> |

|  |
| --- |
| <include name="testMethod" /> |

|  |
| --- |
| </methods> |

|  |
| --- |
| </class> |

|  |
| --- |
| </classes> |

**Packages tag:** specified inside the test tag

|  |
| --- |
| <packages> |

|  |
| --- |
| <package name="test.sample" /> |

|  |
| --- |
| </packages> |

|  |
| --- |
| **Sample xml file** |
| **Test Method (@Test)**  Test case Method will be defined under this. **If the method have return type then testng will ignore this method as testng method** and it won’t execute this method. If we need to include this then following attribute need to be specified suite or test tag  <suite allow-return-values="true"> |

**@Test public void a(){ return t;} …. Then this method won’t be executed until above syntax is used.**

**@Test public void b(){….}**

**Test Groups –** used to execute particular set of tests mentioned under particular group.

- Groups are specified in your testng.xml file and can be found either under the <test> or <suite> tag. Groups specified in the <suite> tag apply to all the <test> tags underneath

-  if you specify group "a" in <suite> and "b" in <test>, then both "a" and "b" will be included.

Test syntax

@Test(groups = { "functest", "checkintest" })

Xml syntax

|  |
| --- |
| <groups> |

|  |
| --- |
| <run> |

|  |
| --- |
| <include name="functest"/> |

|  |
| --- |
| </run> |

|  |
| --- |
| </groups> |

**Using regular expressions**

@Test(groups = {"linux.checkintest"} )

@Test(groups = { "windows.checkintest" })

@Test(groups = { "windows.checkintest2" })

|  |
| --- |
| <groups> |

|  |
| --- |
| <run> |

|  |
| --- |
| <include name="windows.\*"/> |

|  |
| --- |
| </run> |

|  |
| --- |
| </groups> |

This will only execute the all windows groups .\*used to specify any

**Method groups –**exclude or include individual methods

|  |
| --- |
| <classes> |

|  |
| --- |
| <class name="example1.Test1"> |

|  |
| --- |
| <methods> |

|  |
| --- |
| <include name=".\*enabledTestMethod.\*"/> |

|  |
| --- |
| <exclude name=".\*brokenTestMethod.\*"/> |

|  |
| --- |
| </methods> |

|  |
| --- |
| </class> |

|  |
| --- |
| </classes> |

**Groups of groups -** Groups can also include other groups. These groups are called "MetaGroups".



**Exclusion groups –** we can also exclude groups

|  |
| --- |
| <groups> |

|  |
| --- |
| <run> |

|  |
| --- |
| <include name="checkintest"/> |

|  |
| --- |
| <exclude name="broken"/> |

|  |
| --- |
| </run> |

|  |
| --- |
| </groups> |

**Partial groups –** can able to add groups at class level and also for method

|  |
| --- |
| @Test(groups = { "checkin-test" }) |

|  |
| --- |
| public class All { |

|  |
| --- |
|  |

|  |
| --- |
| @Test(groups = { "func-test" ) |

|  |
| --- |
| public void method1() { ... } |

|  |
| --- |
|  |

|  |
| --- |
| public void method2() { ... } |

|  |
| --- |
| } |

Here method2 belongs to class lvl grp but not belongs to method grp func-test

**Parameters –** We can able to make Parameterized methods in testng. Using @Parameters annotation we can able to pass values to the arguments mentioned in the method.

-this annotation can be placed before any @Test, @Before/After or @Factory annotation

- The XML parameters are mapped to the Java parameters in the same order as they are found in the annotation, and TestNG will issue an error if the numbers don't match.

-Parameters are scoped, can be mentioned under <suite> or <test>

-@Optional can be used to set default value if parameters value is not mentioned in the xml file(means the method argument by default will accept value from @Optional if parameters value is not mentioned)

|  |
| --- |
| @Parameters({ "first-name" }) |

|  |
| --- |
| @Test |

|  |
| --- |
| public void testSingleString(String firstName) { |

}

**XML**

|  |
| --- |
| <suite name="My suite"> |

|  |
| --- |
| <parameter name="first-name"  value="Cedric"/> |

|  |
| --- |
| <test name="Simple example"> |

**@Optional**

|  |
| --- |
| @Parameters("db") |

|  |
| --- |
| @Test |

|  |
| --- |
| public void testNonExistentParameter(@Optional("mysql") String db) { ... } |

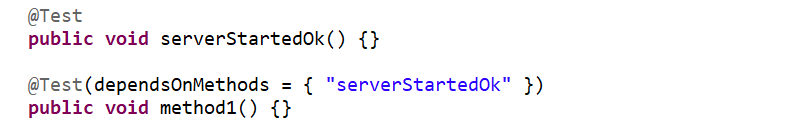
**‘mysql’ value will be set to db string if db parameter is empty.**

**Dependencies –** Some time in testing one method may depend on another method/methods, this can be achieved by dependencies attributes in testNG. Dependencies are also used to achieve order ness in execution.

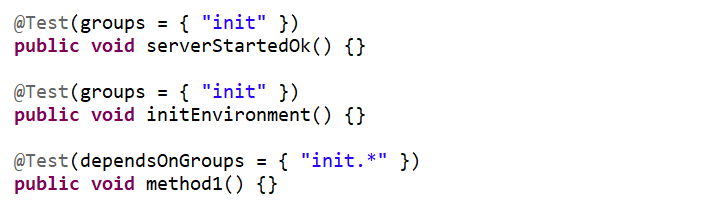
**Hard Dependencies –** the method will be **skipped**  if any one of the dependent method fails.

**Soft Dependencies -** the method will not be **skipped**  even if any one of the dependent method fails, this can be achieved using **alwaysRun=’true’** . Soft dependencies mainly used to achieve execution order , the method will be executed only after all the dependent method ran.

**dependsOnMethods –** used when all tests are within in the same class



**dependsOnGroups –** mostlyused when tests are in different class. If we need to execute only one method of the overloaded versions , then we can use dependsOnGroups , since dependsOnMethods uses method name it is not possible to run any one method of overloaded versions



**Class level annotations -** @Test can be usedon class level . The effect of a class level @Test annotation is to make all the public methods(only public methods) of this class to become test methods even if they are not annotated

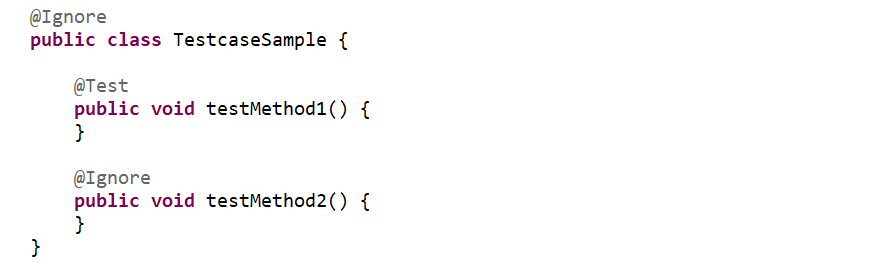


test1, test3 will be executed but test2 is a private method will not be executed. Private method needs @test to execute.

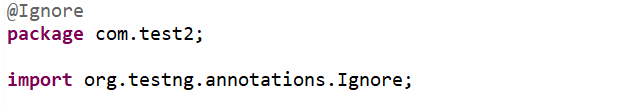
**Ignoring tests**

testNG let us to ignore @test methods in a Class , in a package and its sub-package or a particular method

at class lvl and method lvl

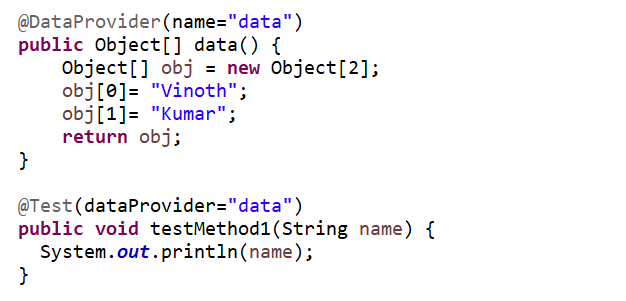


To specify at package lvl we need to create **Create package-info.java** (right-click -> New -> Package: This offers you a checkbox Create package-info.java)

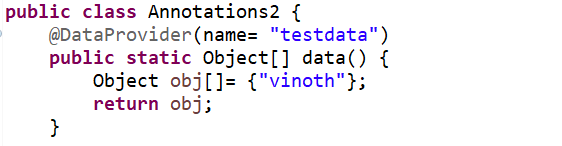


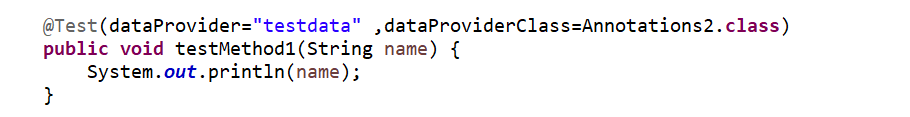
**DataProvider -**  A Data Provider is a method on your class that returns an array of array of objects to the called test method. DataProvider allows us to run the **test method** multiple time with different set of data.

Below test method will be executed 2 time since dp will return 2 objects.

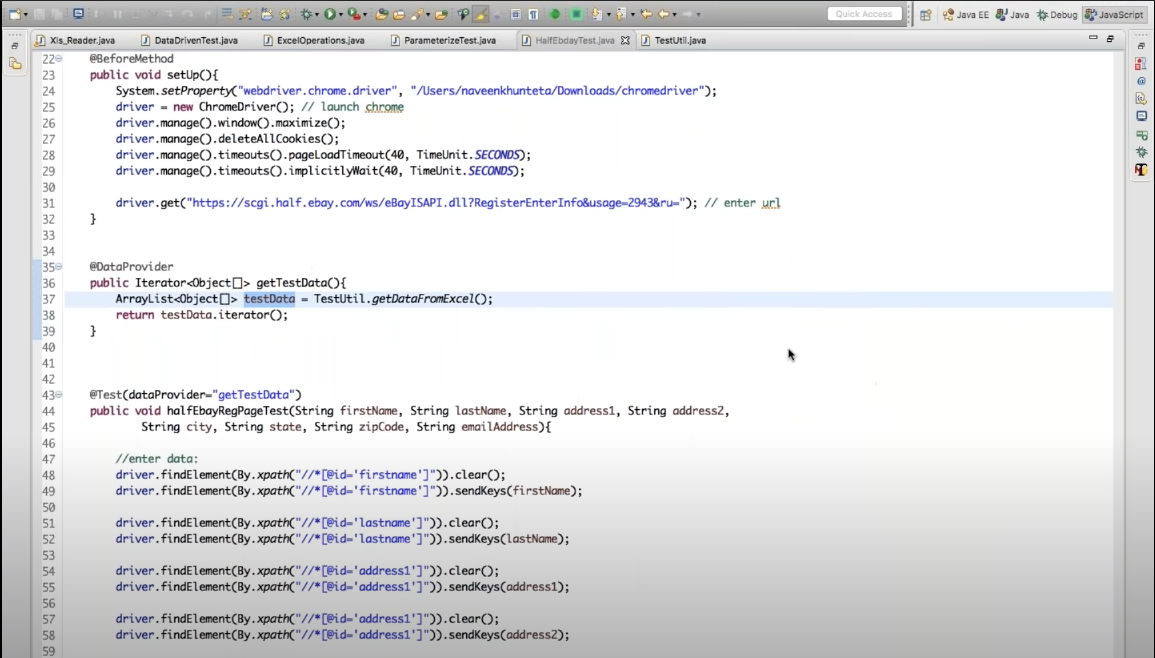


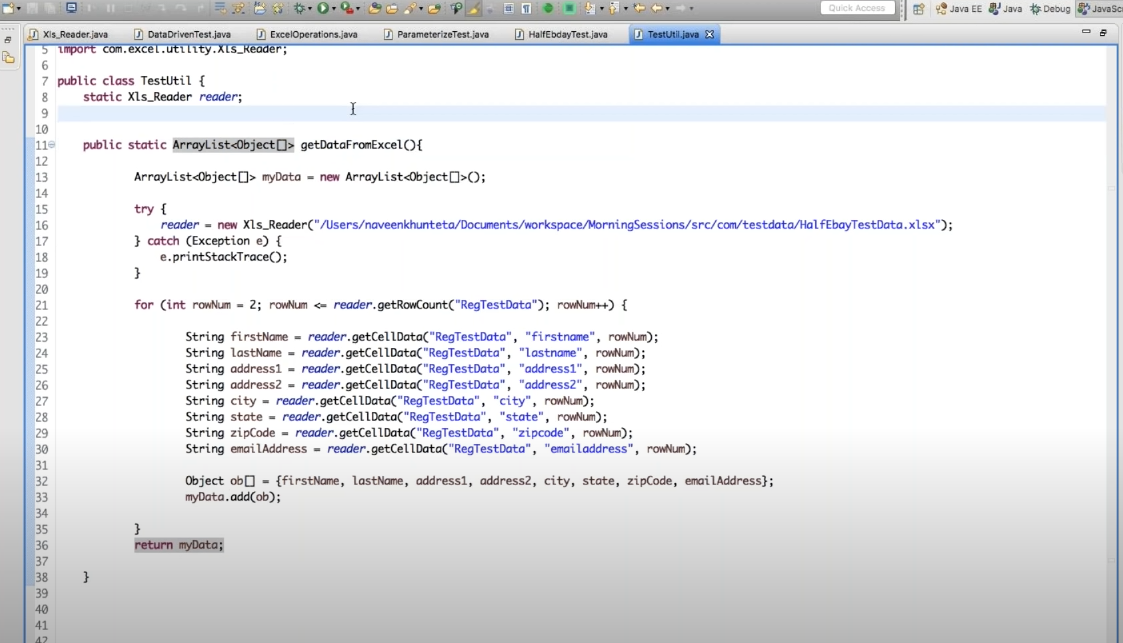
Dataprovider method can also be declared under different class and can be accessed by test method which is in different class, using dataProviderClass attribute (dataProviderClass =Methodname.class) and the Dataprovider method should be static





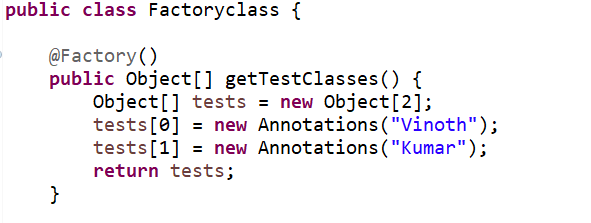
Accessing data from excel using data provider

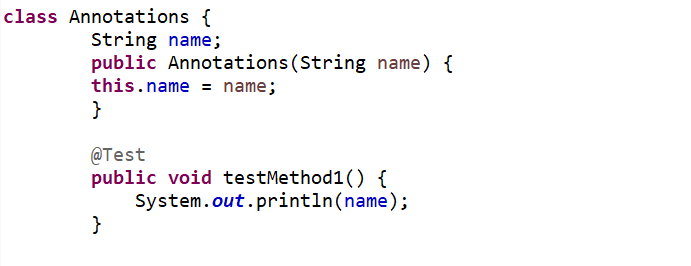




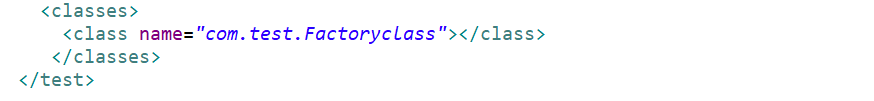
**Factories**

Factories allow you to create tests dynamically. execute the test methods present in the same test class using different instances of the respective class .We need to create factory test in separate class



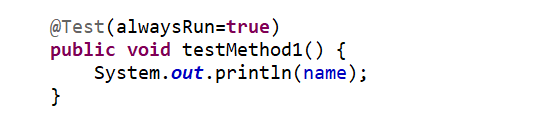


In the xml file we just need to call the factory class

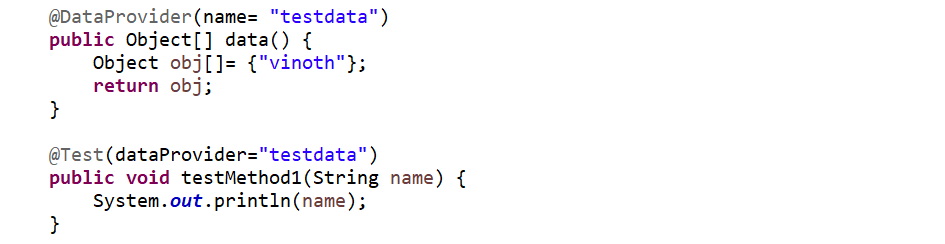


**@Test – Attributes**

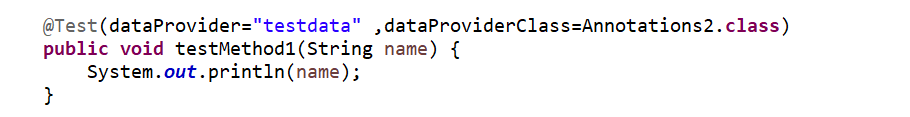
**alwaysRun –** It will run the test even if the dependent method got failed/skipped {used for soft dependency)



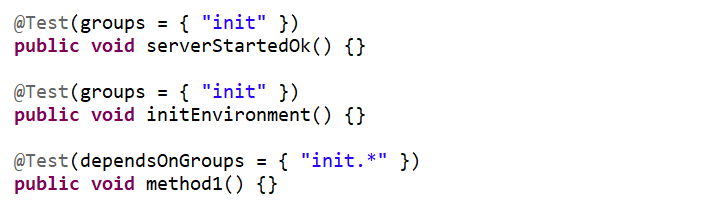
**dataProvider –** used to specify the name of the Dataprovider



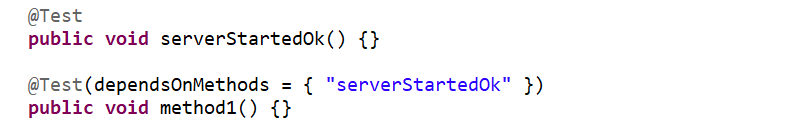
**dataProviderClass –** used to specify the name of the class in which the dataprovider is created



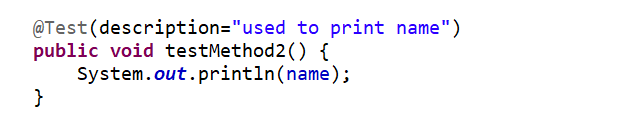
**dependsOnGroups –** The list of groups this method depends on.



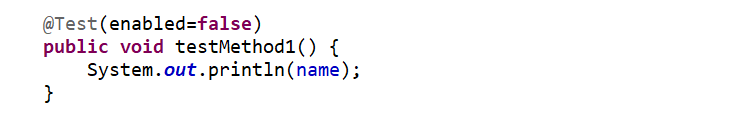
**dependsOnMethods –** The list of methods this method depends on.



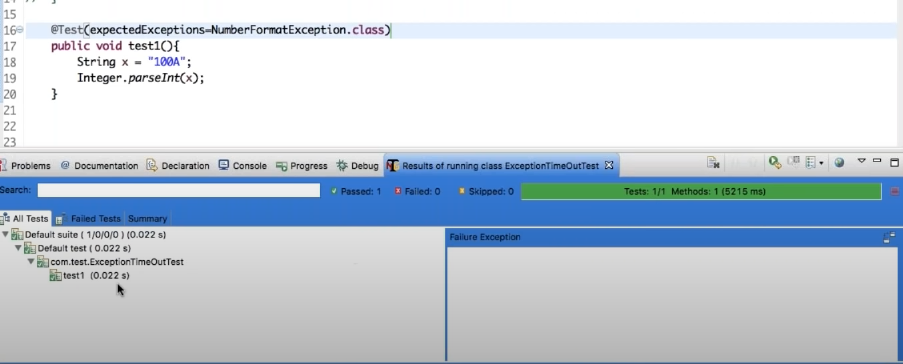
**description -** description for this method



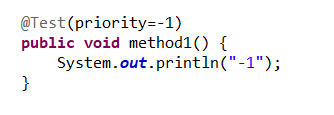
**enabled –** Whether methods on this class/method are enabled (**how you avoid certain tc in testng**)



**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

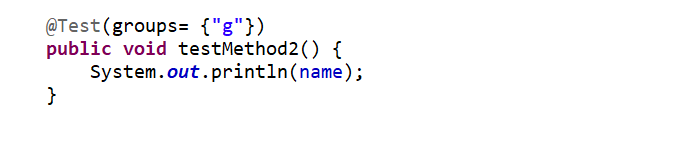


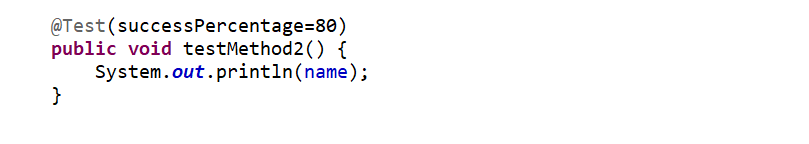
**priority** - using Priority we can alter the sequence of the test execution .Its scope applied across classes. The priority can be set as an integer value (-2,-1,0,1) and lower this integer value, higher is the priority. -1 will be executed first compared to 0,1.



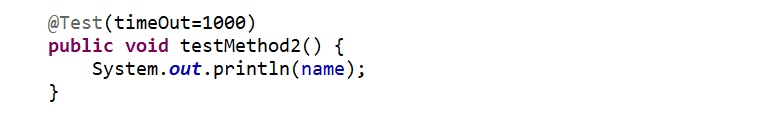


**groups -** The list of groups this class/method belongs to

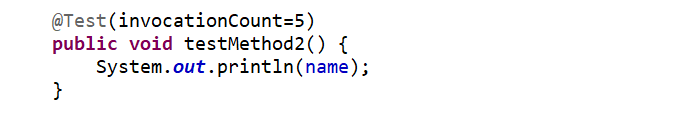


**successPercentage –** The percentage of success expected from this method

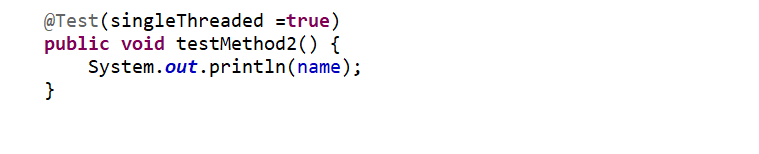
**timeOut -** The maximum number of milliseconds this test should take. Test will be failed beyond the timeout.



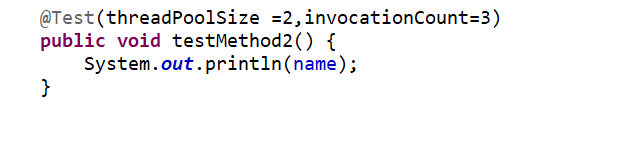
**invocationCount -** The number of times this method should be invoked



**singleThreaded** - If set to true, all the methods on this test class are guaranteed to run in the same thread



**threadPoolSize –** The size of the thread pool for this method. The method will be invoked from multiple threads as specified by invocationCount. This attribute is ignored if invocationCount is not specified



**Parallelism**

Suite lvl – we can able to run different suites in parallel in separate thread.

**Parallel tests, classes and methods**



parallel="methods" - TestNG will run all your test methods in separate threads

parallel="tests" - TestNG will run all the methods in the same <test> tag in the same thread, but each <test> tag will be in a separate thread

parallel="classes" - TestNG will run all the methods in the same class in the same thread, but each class will be run in a separate thread.

parallel="instances" - TestNG will run all the methods in the same instance in the same thread, but two methods on two different instances will be running in different threads.

**TestNG Listeners –** testNG listeners areinterfaces which allow you to modify the TestNG behavior

IAnnotationTransformer

IAnnotationTransformer2

IHookable

IInvokedMethodListener

IMethodInterceptor

IReporter

ISuiteListener

ITestListener

We can implement listeners in one of the following ways,

Using Command line

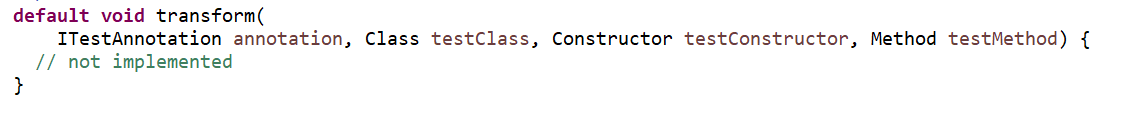
Using Ant

Using <listeners> in xml

Using annotation @ Listeners

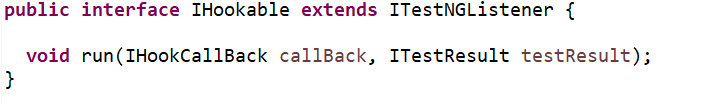
**Annotation Transformers –** using this we can able to modify the content of the annotation at runtime.

**-IAnnotationTransformer -** is a interface using this we can able to change the content of @test annotation. Which extends ITestNGListener interface and have default method called transform with empty body(not implemented)

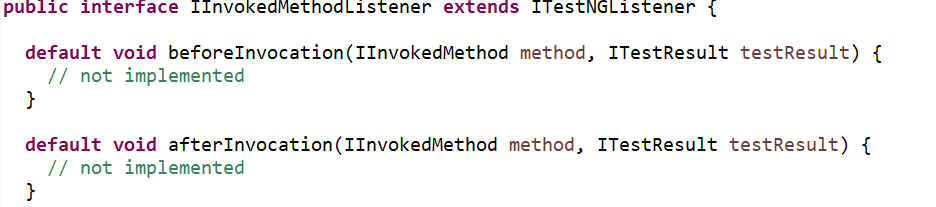


**-IAnnotationTransformer2 -** is a interface using this we can able to change the content of @Factory @DataProvider annotation. Which extends ITestNGListener interface and have default method called transform with empty body(not implemented)

**IHookable –** is a interface which extends ITestNGListener interface and have run(). If a test class implements this interface, its run() method will be invoked instead of each @Test method found



**IInvokedMethodListener -** – is a interface which extends ITestNGListener interface .A listener that gets invoked before and after a method is invoked by TestNG. This listener will only be invoked for configuration and test methods.



**IMethodInterceptor -** is a interface which extends ITestNGListener interface. used to alter the list of test methods that TestNG is about to run.

Once TestNG has calculated in what order the test methods will be invoked, these methods are split in two groups:

* Methods run sequentially. These are all the test methods that have dependencies or dependents. These methods will be run in a specific order.
* Methods run in no particular order. These are all the methods that don't belong in the first category. The order in which these test methods are run is random and can vary from one run to the next (although by default, TestNG will try to group test methods by class)

In order to control the methods in second category , this will be used.

**ISuiteListener -** Listener for test suites. is a interface which extends ITestNGListener interface. Have two methods onStart(), onFinish().

**ITestListener -** Listener for test running. is a interface which extends ITestNGListener interface. This can be used to override the methods of ITestListener as per our requirement.

Methods in ITestListener, or methods in extent report

**onTestStart(ITestResult result)** -  is invoked only when any test method gets started

**onTestSuccess():** method is executed on the success of a test method.

**onTestFailure():** method is invoked when test method fails.

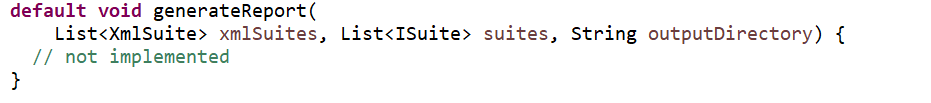
**onTestSkipped():** run only when any test method has been skipped.

**onTestFailedButWithinSuccessPercentage():** is invoked each time when the test method fails but within success percentage.

**onStart():** is executed on the start of any test method.

**onFinish():** Invoked after all the tests have run

**IReporter -** interface can be implemented to generate a report. is a interface which extends ITestNGListener interface. Have only one method generateReport() , will be invoked after all the suite have run.



**Rerunning Failed tests**

Once the tests failed , then a file called **testng-failed.xml** will be created under the **test-output** folder. This file will have all the test methods that got failed and will contain all the necessary dependent methods. We can simply use this file to re run the failed cases.



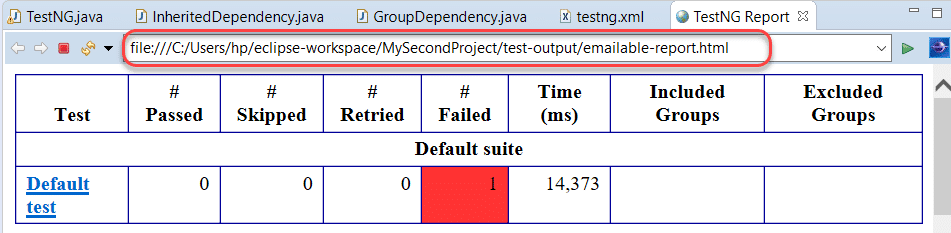
**Retrying the failed method**

We can able to automatically retry the test method if it fails using **IRetryAnalyzer,** it is a Interface.

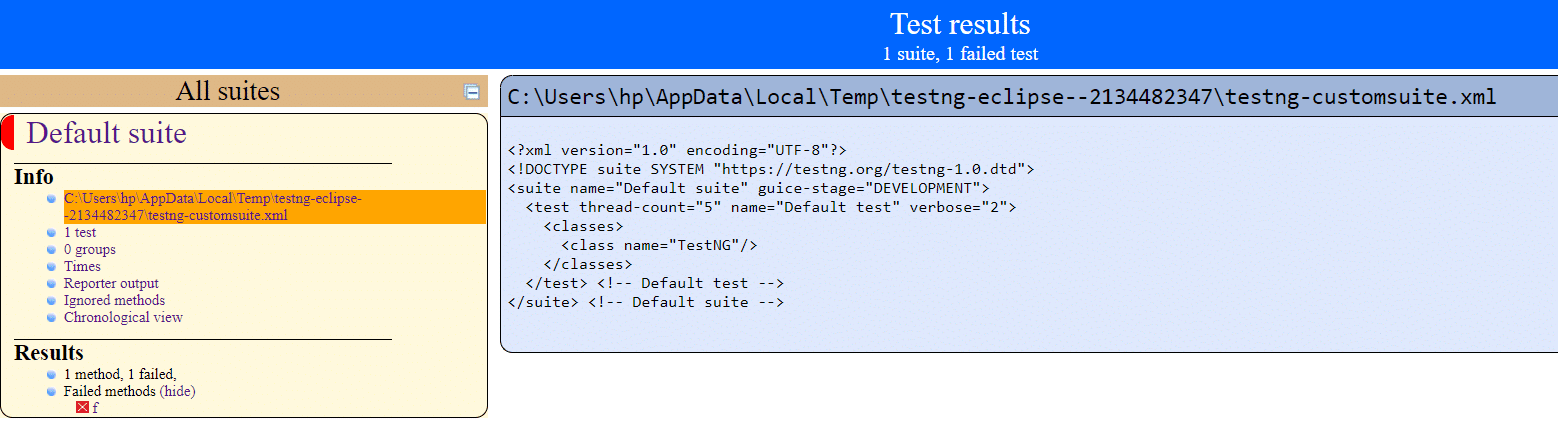
**Reports in TestNg**

Two types of reports will be generated after the execution in **test-output** folder by default.

Emailable-report.html



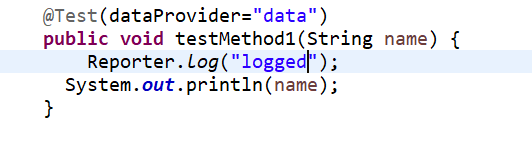
Index.html **-** left pan have index view while right pan gives the detailed view for the selected item in the left pan.



**Reporter class - *TestNG reporter is an inbuilt class in TestNG, which helps in logging the messages in the output reports.***

***We can able to log messages using below syntax***

Reporter.log(“message”); **Import Required:***import org.testng.Reporter;*



Questions,

1. What is TestNG?
2. What are the annotations used in the TestNG?
3. What is the sequence of execution of all the annotations in TestNG?
4. How to set the priorities in TestNG?
5. Define grouping in TestNG?
6. What is dependency in TestNG? (When we want to run the test cases in a specific order)
7. How to pass the parameter in test case through testng.xml file?
8. What is the importance of testng.xml file?

It defines the order of the execution of all the test cases.

It allows you to group the test cases and can be executed as per the requirements.

It executes the selected test cases.

In TestNG, listeners can be implemented at the suite level.

It allows you to integrate the TestNG framework with tools such as Jenkins.

1. How can we disable the test case from running? (enabled=false)
2. What is the use of @Listener annotation in TestNG?
3. What is the use of @Factory annotation?
4. What is the difference between @Factory and @DataProvider annotation? **@DataProvider:** It is annotation used by TestNG to execute the test method multiple numbers of times based on the data provided by the DataProvider.

**@Factory:** It is annotation used by the TestNG to execute the test methods present in the same test class using different instances of the respective class.

1. TestNG test suite? collection of tests
2. What are the categories of annotations in TestNG?

Post-condition Annotation

Test Annotation

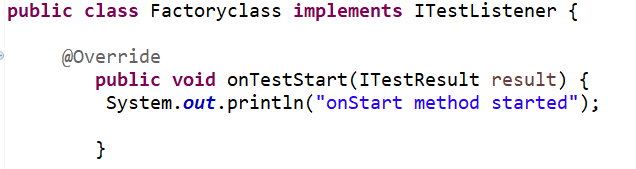
Pre-condition Annotation

1. What are the types of reports generated in TestNG by default?
2. What are priorities in TestNG?
3. What is parameterization in TestNG?

parameterization runs a test method multiple times with different values. Another name for this process is data-driven testing in TestNG. We can acquire Parameterization in TestNG in two ways: XML and Dataprovider

1. What are the optional parameters in TestNG?
2. On what levels can we apply parallel testing in TestNG? Methods,tests,classes,instances
3. **How are listeners declared in TestNG?**

**-**first need to write a class by extending the any Listener Interface that we want.



-And we need to put the listener annotation along with the class that we created.

