Automation Testing

Working with Attributes of @Test

A Day in the Life of an Automation Test Engineer

In the previous lesson, John had learned and understood the various annotations and configuration files in TestNG.

He must now learn how to use the test groups, which offer the most flexibility in dividing the tests and use the TestNG.xml. It enables him to create and manage multiple test classes, define test suites and tests, perform parallel testing with TestNG, and import configuration classes and files.

To achieve the above, he will learn the order of executions and dependencies, Test groups and TestNG.xml, and parallel testing TestNG.

Learning Objectives

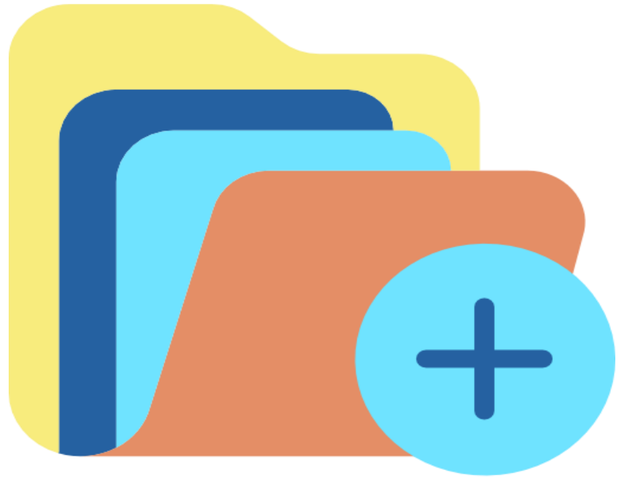
By the end of this lesson, you will be able to:

- Analyze the importing of configuration classes and files
- Analyze Test groups and TestNG.xml
- Discuss order of executions and dependencies
- Describe the parallel testing TestNG

Importing @Configuration

Importing Additional Configuration Classes

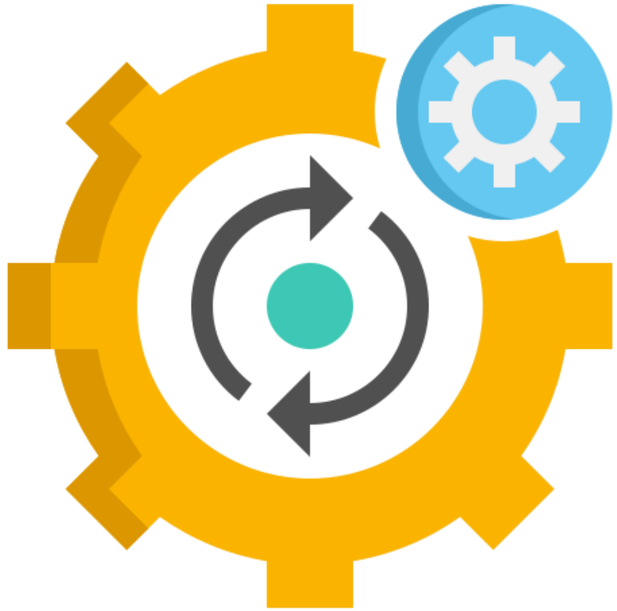
The **@Import** annotation can be used to import additional configuration classes.



Users do not have to put all the @Configuration classes into one class.

Importing XML Configuration

The **@ImportResource** annotation is used to import one or more XML configuration files.



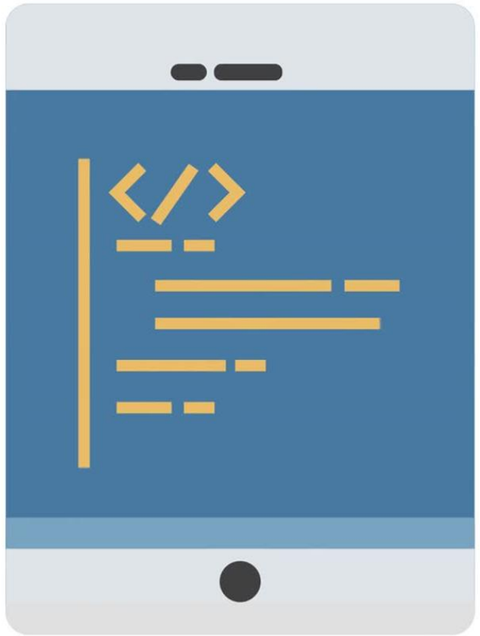
Limitation for @Configuration Classes

- Static declarations are required for any nested configuration classes.
- Configuration classes must be non-final.
- Configuration classes must be non-local.
- Additional configuration classes cannot be created using @Bean methods.

Key Attributes of @Test

Test Annotation

The @Test annotation is one of TestNG's most basic annotations, and it designates a method as a test method.



The @Test annotation on the test method informs TestNG that this method is a **test** and should be executed when the user runs the class.

Key Attributes of @Test

Some of the key attributes supported by Test annotation are:

Attributes	Description
dataProviderClass	Used to call the DataProvider method from another class
expectedExceptions	Used for exception testing
invocationCount	Used to execute a method any number of times

@Test at Class Level

@Test at Class Level

At the class level, all the methods of class will be treated as @test annotated methods by default.

Users can also define groups like **@Test(groups = { "regression" })** at the class level and add groups at the individual method level.

Test Groups and TestNG.xml

Test Groups

Grouping of test methods is required when users want to access the test methods of different classes.

It gives flexibility to the users by grouping the test methods and eliminating the need to recompile test cases when users run two different sets of test cases back-to-back.

Test Groups and TestNG.xml

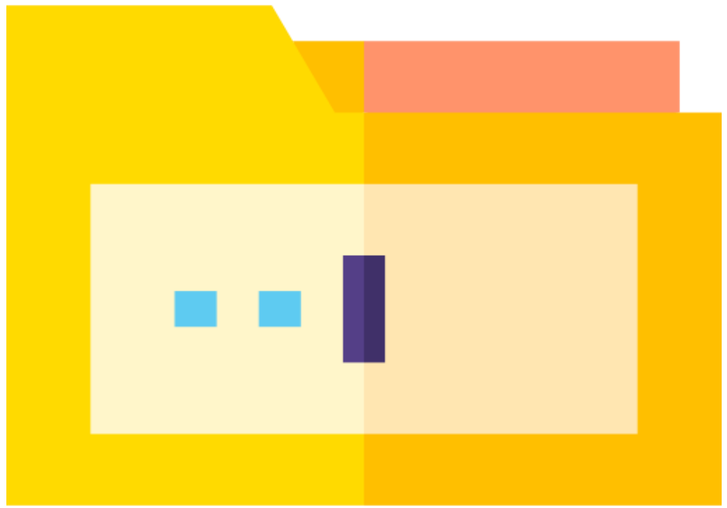
Groups are specified in the testng.xml file with **<groups>** tag.

Groups can be specified either in the **<suite>** tag or **<test>** tag.

Test Groups: Naming Conventions

Test Groups Naming Conventions

Test name should be presented as a statement or fact of life that expresses workflows and outputs.



TestNG: Order of Executions and Dependencies

TestNG Order of Execution

The execution procedure of the TestNG test methods:

01 Create a Java class file name TestngAnnotation.java

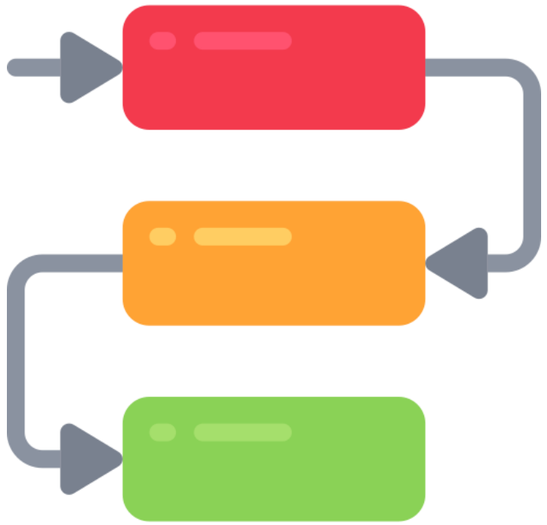
02 Create the file testng.xml in in/work/testng/src to execute annotations

03 Compile the Test case class using javac

04 Run the testng.xml, which will run the test case defined in the Test Case class

TestNG Dependencies

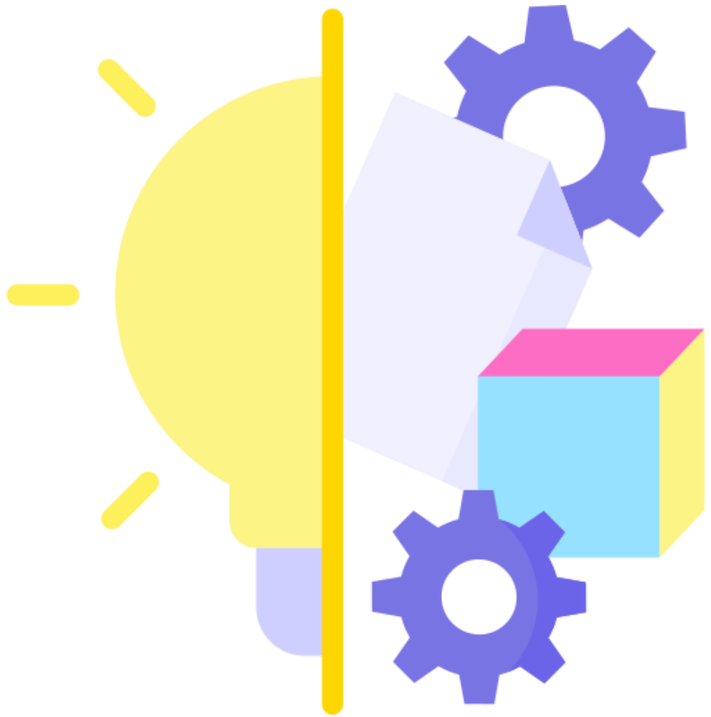
Test dependencies are abilities to make test methods dependent on one or more other test methods.



It is one of the most useful features in the TestNG framework.

Dependencies with Annotations

The @Test annotations such as dependOnMethods and dependOnGroups attributes can be used to specify dependencies in TestNG..



Types of Dependencies

Dependencies can be divided into two categories:

01

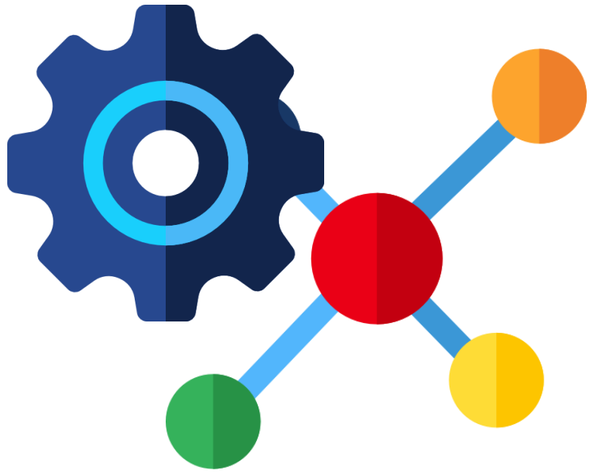
Hard dependencies

02

Soft dependencies

Hard Dependencies

All the methods on which the user relies should run and succeed for the user to run.



If at least one failure occurs in the dependencies, the user will not be invoked and will be marked as skip in the report.

Soft Dependencies

Soft dependencies are useful when users want to make sure the test methods run in a specific order, but their success is not entirely dependent on the success of other methods.

Even if some of the methods fail, the user will be able to run the methods.

Dependencies in XML

Users can specify group dependencies in the testng.xml file by using the <dependencies> tag.

Example:

```
<test name="Mysuite">
  <groups>
    <dependencies>
      <group name="c" dependson="a  b" />
      <group name="z" dependson="c" />
    </dependencies>
  </groups>
</test>
```

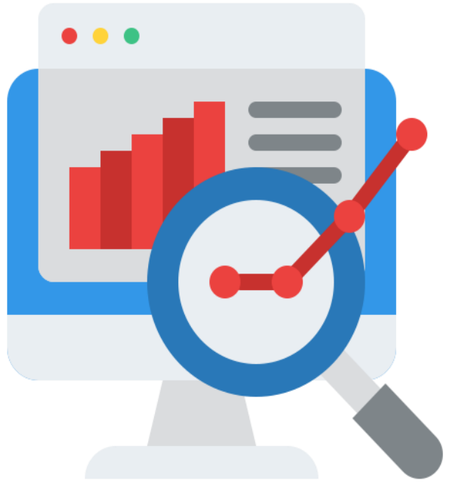
Parallel Test Execution Capability

Parallel Testing

Parallel testing is a method of leveraging automation testing capabilities by running the same tests in multiple environments, real device combinations, and browser configurations at the same time.

Parallel Testing Using TestNG

The comprehensive feature set of TestNG makes parallel testing simple to set up.



The auto-defined XML file in the TestNG framework allows testers to set the parallel attributes to methods, tests, or classes by leveraging the allowance for multithreading in Java.

Advantages of Parallel Testing

Parallel tests can significantly speed up the process by running the same test against multiple configurations at the same time.

Cloud-based test grids enable high-concurrency test execution, lowering the overall cost per test.

Large test suites can be broken down into smaller, independent jobs that can be run in parallel to get results faster, improving the pipelines performance.

Key Takeaways

- Configuration classes must be provided as classes with a generated subclass that allows for runtime enhancements.
- All public methods declared within the class will be marked as TestNG test methods when using the @Test annotation.
- TestNG groups allow users to perform groupings of different test methods.
- The overarching goal of parallel testing is to reduce time and resource constraints.