**TestNG Framework:**

TestNG is a robust testing framework that brings flexibility and advanced features to your Selenium automation suite.

- Key Features:

1. **Annotations**: TestNG offers annotations like `@Test`, `@BeforeMethod`, and `@AfterMethod` to control the execution flow and setup/teardown operations.[Organising, grouping the tests]

2. **Parallel Execution**: With TestNG, you can execute tests in parallel, making the most of your testing infrastructure and speeding up test runs.

3. **Grouping and Prioritization**: Organize your test methods into groups and prioritize their execution, allowing for selective testing.

4. **Dependency Management**: Define test method dependencies to ensure specific test methods run only when their prerequisites pass.

5. **Data-Driven Testing**: Utilize data providers to run the same test with different sets of data, enhancing test coverage.

6. **Listeners and Reporting**: Incorporate custom listeners for generating detailed and customised test reports.

7. **Configuration Management**: Use XML configuration files to manage suite-level configurations and parameterization.

- Setting Up TestNG Framework:

1. Environment Setup: Install Java and your preferred Integrated Development Environment (IDE) like Eclipse or IntelliJ IDEA.

2. Selenium WebDriver: Download and set up Selenium WebDriver for the desired browser(s).

3. TestNG Installation: Add TestNG to your project as a dependency using tools like Maven or by downloading the JAR files directly.

4. Creating Test Cases: Write test cases using TestNG annotations. Organize them into test classes.

5. Test Suite Configuration: Create an XML suite configuration file to define test classes, parallel execution, and other settings.

6. Running Tests: Right-click on the XML suite file and run as TestNG suite. Observe the results in the TestNG console or generated reports.

Order of TestNG annotation execution.

1. **@BeforeSuite**: This annotation is executed before any tests or test suites in the suite.

2. **@BeforeTest**: This annotation is executed before any tests within a `<test>` tag in the XML suite file.

3. **@BeforeClass**: This annotation is executed before the first test method in a test class.

4. **@BeforeMethod**: This annotation is executed before each test method in a test class.

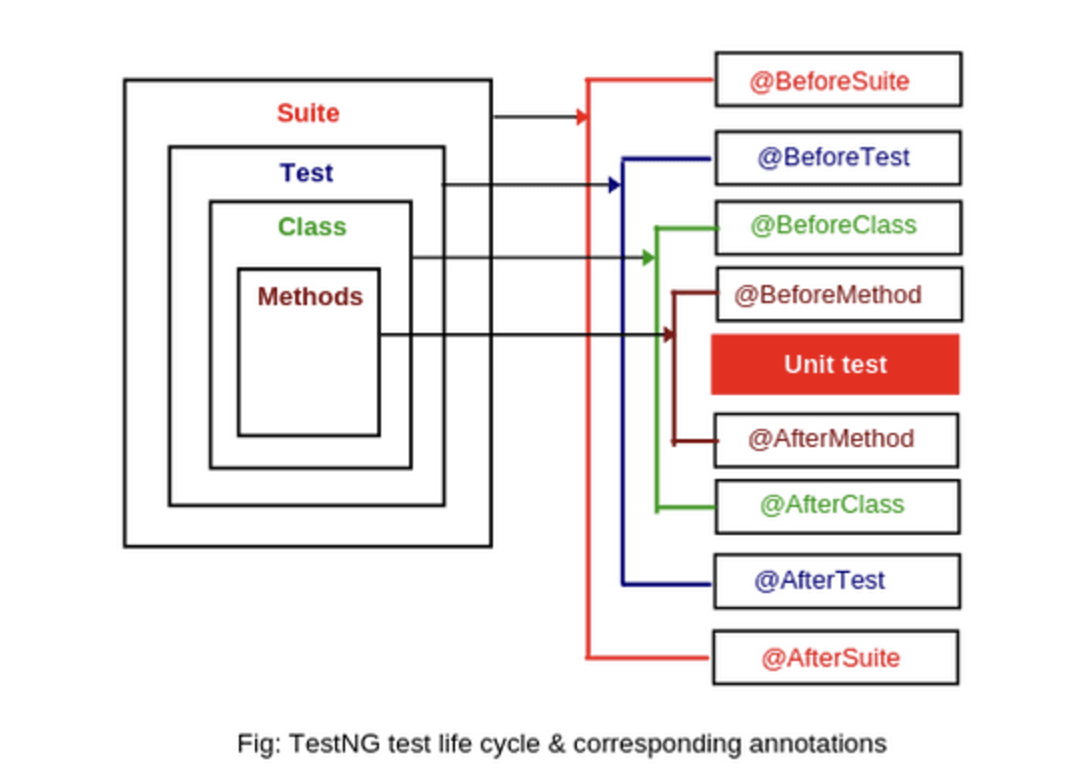
5. **@Test**: This annotation is executed for each individual test method. The order of execution is typically determined by the ordering of the test methods within the test class.

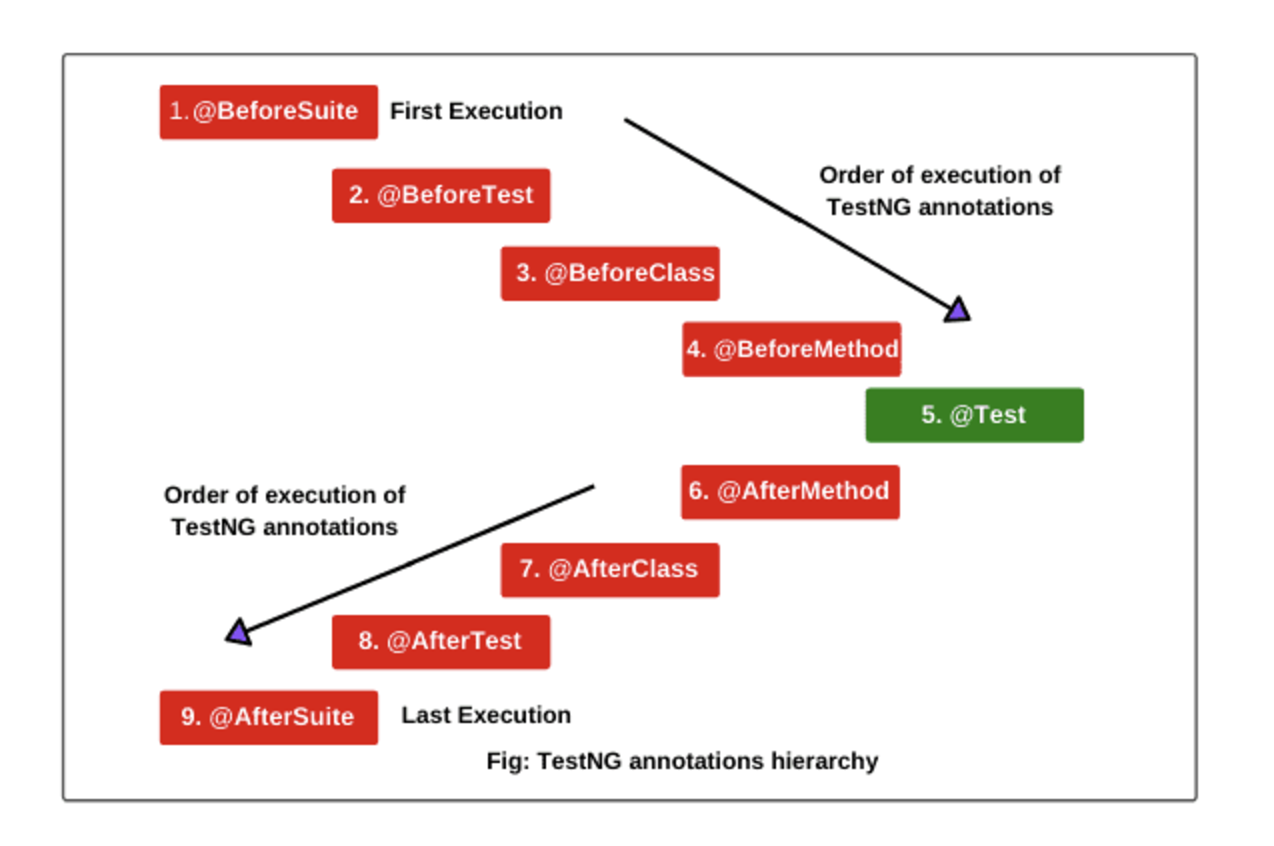
6. **@AfterMethod**: This annotation is executed after each test method in a test class.

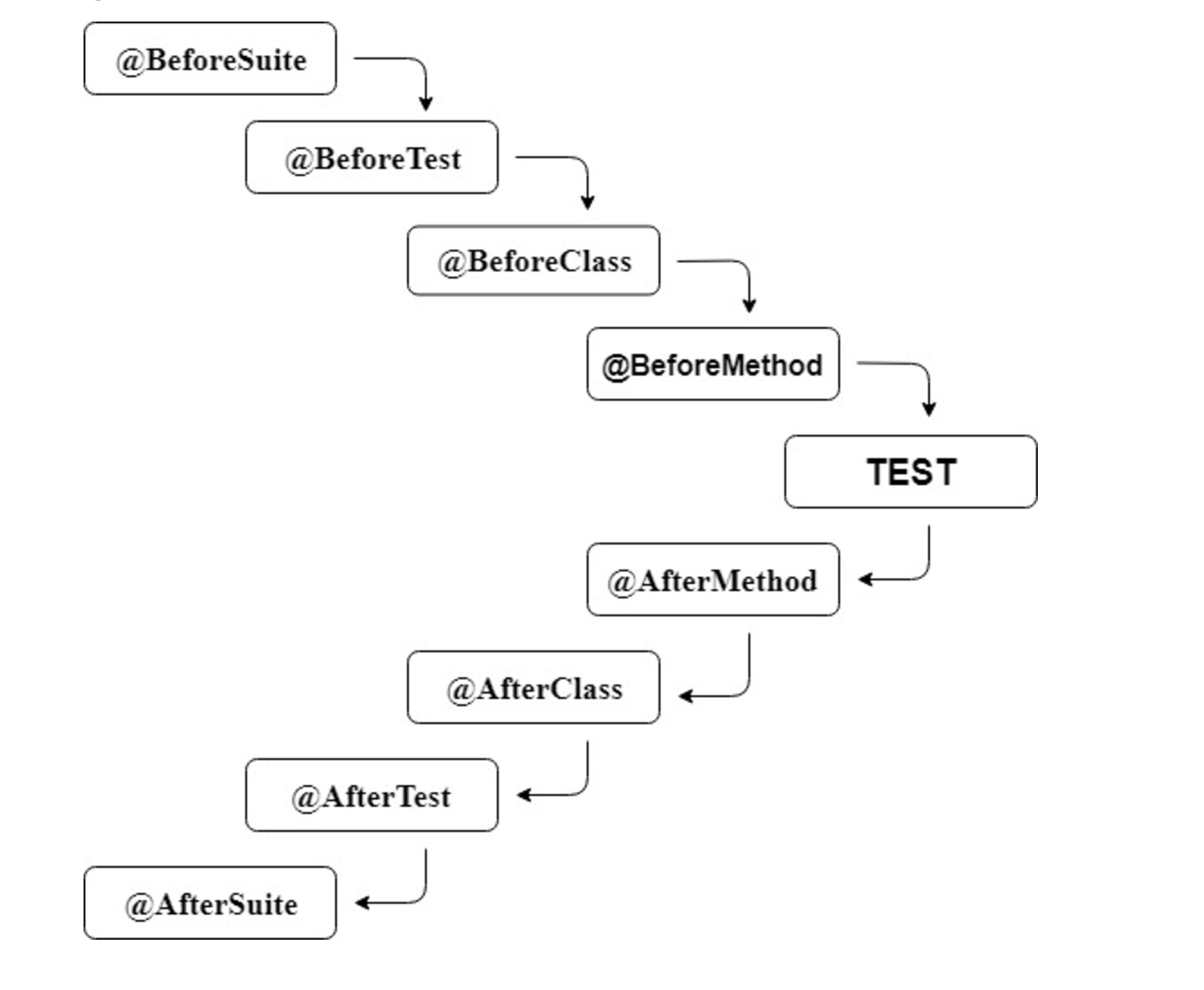
7. **@AfterClass**: This annotation is executed after all the test methods in a test class have completed.

8. **@AfterTest**: This annotation is executed after all the tests within a `<test>` tag in the XML suite file have completed.

9. **@AfterSuite**: This annotation is executed after all tests and test suites in the suite have completed.







It's important to note that TestNG provides flexibility in configuring the order of execution using various attributes like `priority`, `dependsOnMethods`, and `dependsOnGroups` in the annotation declarations. These attributes allow you to explicitly specify the order in which methods or groups should be executed.

Additionally, if you are using XML configuration for TestNG, you can define the `<test>` tags and their execution order within the `<suite>` tag to further control the execution flow.

Remember that the order of execution is influenced by the dependencies and priorities defined by the annotations and their attributes.

In TestNG, the grouping concept allows you to categorize and organize test methods into logical groups based on certain criteria. This is useful for selectively running specific groups of tests, controlling the execution order of tests within a group, and applying configuration settings or behaviours to groups of tests.

### **How Grouping Works in TestNG**

Grouping in TestNG is achieved using the groups attribute within the @Test annotation or the <groups> tag in the TestNG XML configuration file. Test methods can be assigned to one or more groups, and you can then execute tests based on these groupings.

### **Example of Grouping with @Test Annotation**

package grouping;

import org.testng.annotations.Test;

public class Grouping

{

@Test(groups= {"sanity", "regression", "smoke"})

public void display()

{

System.out.println("All 3 testing type");

}

@Test(groups= {"sanity", "smoke"})

public void show()

{

System.out.println("All 2 sanity and smoke testing type");

}

@Test(groups= {"regression", "smoke"})

public void print()

{

System.out.println("All 2 smoke and regression testing type");

}

@Test(groups= {"sanity"})

public void animal()

{

System.out.println("1 sanity testing type");

}

@Test(groups= {"smoke"})

public void car()

{

System.out.println("1 smoke testing type");

}

@Test(groups= {"regression"})

public void bike()

{

System.out.println("1 regression testing type");

}

}

### **Executing TestNG Tests Based on Groups**

You can execute tests based on groups using TestNG XML configuration or programmatically in your test suite setup.

### **1. TestNG XML Configuration**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "<https://testng.org/testng-1.0.dtd>">

<suite name="Grouping">

<test thread-count="5" name="TestGrouping1">

<groups>

<run>

<include name="sanity"/>

<include name="smoke"/>

<include name="regression"/>

or

<exclude name="sanity"/>

<exclude name="smoke"/>

<exclude name="regression"/>

</run>

</groups>

<classes>

<class name="grouping.Grouping" />

</classes>

</test> <!-- TestGrouping1 -->

</suite> <!-- Grouping -->

### **Benefits and Use Cases of Grouping**

* Selective Test Execution: Grouping allows you to run specific categories of tests (e.g., smoke tests, regression tests) based on your requirements.
* Execution Order: You can define the execution order of test methods within a group, ensuring that certain tests are run before others.
* Configuration Management: Apply different configuration settings or behaviors (e.g., parallel execution, data providers) to specific groups of tests.
* Reporting and Analysis: TestNG reports include information about groups, making it easier to analyze test results based on categories.

By leveraging the grouping concept in TestNG, you can effectively manage and organize your test suite, enabling targeted and controlled test execution based on logical groupings of test methods. This enhances flexibility, maintainability, and scalability of your automated testing efforts.

Implementing Data Driven Testing using Selenium WebDriver?

In order to achieve Test Parameterization, TestNG offers 3 solutions:

1. Parameters
2. Data Providers
3. Factories

@Parameters:

1. This annotation is used on a test method to specify the names of the parameters which the test would require and where to find their values.
2. The values can be provided through an XML file or at runtime via the command line.

Example:



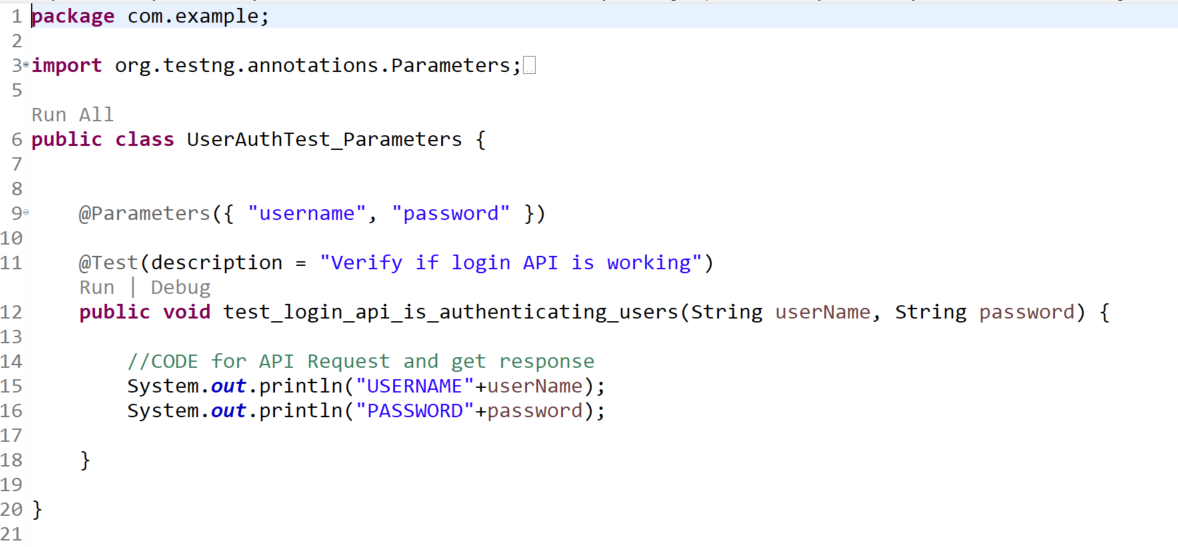
Explanation:

<parameter name="username" value="admin"></parameter>

<parameter name="password" value="password"></parameter>

* These are parameters for the test suite. They allow passing values to the test cases or classes within the suite.
* In this case, the username and password are being passed as parameters with their respective values.

Test Class:



Explanation:

1. The @Test annotation is used to mark the test\_login\_api\_is\_authenticating\_users method as a test case.
2. The @Parameters annotation is used to specify the input parameters for the test case, which are the username and password.
3. These are the name of the name of the parameter passed to the tests from testng.xml file
4. In the test\_login\_api\_is\_authenticating\_users method, the input parameters are passed as arguments to the method as userName and password, respectively.
5. Output:



Supported Values:

You can pass various types of parameters to your test method including:

1. String: This is the most common data type for passing parameters. You can pass a string value to your test method.
2. Boolean: You can pass a Boolean value to your test method to represent a true or false condition.
3. Numeric Values: You can pass an int,byte,short,long,double,float value to your test method.
4. Char: You can pass a character value to your test method.

Real world Usage:

1. We often use testng parameters only to pass simple key value data to the tests. Example Environment information, endpoint or URL
2. If you wish to pass complex data set to the test its better to choose Data Provider over the Parameters.

And That's all you need to know about how to use parameters in TestNG. Now Lets move to second feature for data driven testing which is our Data Providers!!

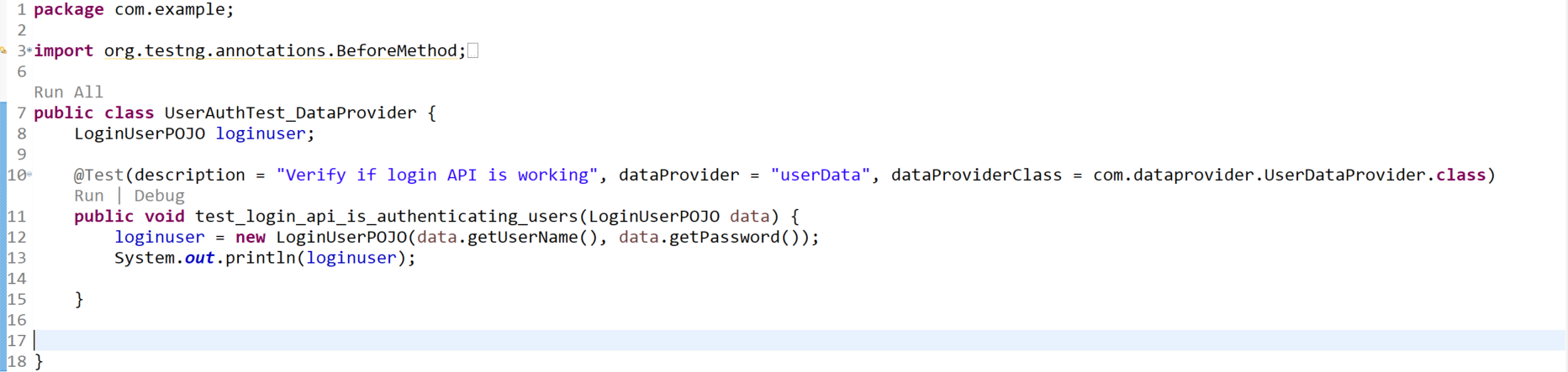
@DataProvider: This annotation is used on a method that returns a one/two-dimensional array of objects or even Iterators. When working with a 2D Array The first dimension represents the test data, while the second dimension represents the arguments to the test method



Code Explanation:

* The data provider method is named userDataProvider and it returns an array of LoginUserPOJO objects.
* Each LoginUserPOJO object represents a user's login credentials, consisting of a username and a password.
* The LoginUserPOJO class is likely a POJO (Plain Old Java Object) that has fields for the username and password and getter/setter methods for accessing those fields.
* The data provider returns an array of 5 LoginUserPOJO objects, representing 5 different user login scenarios.
* TestNG can use this data provider to feed the test method with test data so that the same test method can be executed multiple times with different data sets.

The data provider can be used in a TestNG test class like this:



Code Explanation:

This code is an example of how to use TestNG's data provider feature to supply test data to a test method.

Benefit:

By separating the test data from the test logic, you can write more efficient and readable tests.

Important Thought:

Often time I have seen that people pass 10,15 parameters to a test method. This practice is bad and should be avoided as much as you can.

You would ask then what is the ideal number of parameters for a method?

1. The number of parameters that a method should have is a matter of debate and largely depends on the specific use case and context. In general, a method should have the minimum number of parameters required to perform its intended functionality.
2. Having too many parameters can make a method hard to read and understand, and can make it more difficult to maintain and test. It can also indicate that the method is trying to do too many things and may benefit from being split into smaller, more focused methods.
3. On the other hand, having too few parameters can also be a problem, as it may lead to code duplication or the need to use global variables or other mechanisms to pass additional information to the method.

As a rule of thumb, it's often recommended to aim for no more than three to five parameters for a method. However, this is not a hard and fast rule and there may be cases where more parameters are necessary. Ultimately, the goal should be to write code that is easy to read, understand, maintain!

As a rule of thumb, it's often recommended to aim for no more than three to five parameters for a method.

Lets move on to the last type of Data Parameterization is Factory

In TestNG, factories and data providers are two distinct concepts that serve different purposes.

Factories in TestNG are used to create new instances of test classes.

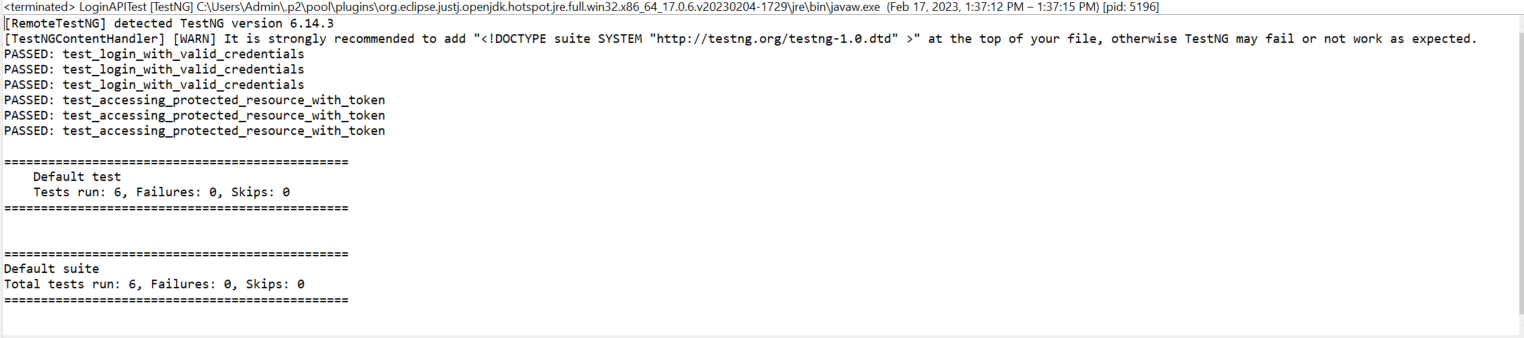
1. @Factory: This annotation is used on a method that returns an array of objects [Test Instances]. Each object in the array represents a new instance of the test class. TestNG will then create a new instance of the test class for each object in the array.
2. Factories allow you to create tests dynamically. Imagine you want to create a test method that will access a page on a Web site several times, and you want to invoke it with different values:

Sample Code





Output:



One of the main advantages of using the factory methods is that we can pass parameters to test classes while initializing them.

These parameters can then be used across all the test methods present in the said test classes.

### **Difference between Data Provider and Factories:**

1. A factory in TestNG is a component that is responsible for creating and managing test instances at runtime.
2. On the other hand, a data provider in TestNG is used to supply test data to test methods. This can be useful when you need to test a single method with different sets of input data.

Using both factory and DataProviders for test data parameterization can make your tests more flexible and easier to maintain, as you can separate the test data generation logic from the test method implementation.

## **What are Listeners in Selenium?**

Listeners in Selenium are classes that implement specific interfaces in order to track events that occur during the execution of **Selenium tests.** You can use them to perform additional actions or log information when specific events happen, such as a test case starting or ending, or a test step passing or failing.

In Selenium, a listener is an object that “listens” for certain events to occur during the execution of a test script, such as the start or completion of a test case, or the detection of an error. Hence, when an event of interest occurs, the listener can perform a specific action, such as logging information about the event or taking a screenshot.

Here are a few examples of how listeners can be used in Selenium:

1. **Screenshot on failure**: You can use a listener to take a screenshot of the browser window when a test case fails. This can be useful for debugging and identifying the cause of the failure.
2. **Logging**: You can also use them to log information about the test execution, such as the start and end time of each test case, and the test case status (pass/fail). This can be useful for generating detailed test execution reports.
3. **Email notification**: They can be used to send an email notification when a test execution completes. This can be useful for alerting the development team of the test execution results.
4. **Performance Metrics**: A listener can also be used to measure and record the execution time of each test method. It can be used to log test-wise or suite-wise performance metrics, which will be helpful in identifying performance bottlenecks.

## **Types of Listeners in Selenium**

In Selenium, there are two types of listeners.

* TestNG listeners.
* WebDriver listeners.

We use TestNG listeners to perform operations before and after a test method is executed. On the other hand, we use WebDriver listeners to listen to WebDriver events like navigating to a new URL and clicking an element.

It’s worth noting that there are several built-in listeners which are provided by Selenium. In addition to this, developers can also create their own custom listeners by implementing the interfaces or classes and registering them to listen to the events. Let’s have a look at both these types of listeners in detail.

## **What are WebDriver Listeners in Selenium?**

WebDriver listeners in Selenium are event listeners that allow you to listen to events emitted by the WebDriver. Consequently, these listeners provide an extra layer of functionality. This allows you to capture and process events, such as navigating to a new URL or clicking on an element.

### **Built-in WebDriver Listeners**

Selenium provides several built-in WebDriver listeners. We can use them to capture and process events emitted by the WebDriver. These include:

* **WebDriverEventListener**: This is an interface that contains methods for listening to WebDriver events.
* **AbstractWebDriverEventListener**: It is an abstract class that you can subclass to implement the WebDriverEventListener interface.
* **EventFiringWebDriver**: This class wraps up a WebDriver instance and enables the firing of WebDriver events.
* **WebDriverListenerManager**: A class that manages the registered listeners and notifies them of the events.

### **Use Cases of WebDriver Listeners**

WebDriver listeners are particularly useful in cases where you need to capture and process events emitted by the WebDriver. Some common use cases include:

* Taking screenshots of test failure.
* Logging test results.
* Capturing performance metrics.
* Analyzing browser logs.

In summary, WebDriver listeners in Selenium allow you to capture and process events emitted by the WebDriver. These listeners provide an extra layer of functionality, making it easier to capture and process events in your tests.

## **What are TestNG listeners in Selenium?**

As discussed earlier, we use TestNG listeners in Selenium to perform operations before and after we execute a test method. We can provide additional functionality to TestNG tests, such as capturing screenshots or logging test results. Let’s have a look at built-in TestNG listeners in Selenium and their use cases.

### **Built-in TestNG Listeners**

As discussed, we have several built-in TestNG listeners in Selenium. These can be used to perform operations before and after a test method is executed. These include:

* **IInvokedMethodListener**: This listener is invoked before and after every test, the method is invoked.
* **IAnnotationTransformer**: It is used to modify test method annotations.
* **IMethodInterceptor**: This listener is used to add an interceptor to the test method invocation.
* **IHookable**: It is used to execute any code before and after a test method, test configuration method, or suite.
* **IConfigurationListener**: This listener is invoked before and after a configuration, and the method is invoked.
* **IExecutionListener**: It is invoked at the beginning and end of a test run.
* **ISuiteListener**: This listener is invoked before and after a suite is run.
* **ITestListener**: It is invoked before and after a test is run and on test success or failure.

### **Use Cases of TestNG Listeners**

TestNG listeners are particularly useful in cases where you need to perform operations before and after a test method is executed. Some common use cases include:

* Taking screenshots of test failure.
* Logging test results.
* Capturing performance metrics.
* Analyzing browser logs.
* Interceptors for test methods.

In summary, TestNG listeners in Selenium allow you to perform operations before and after a test method is executed. Furthermore, with the use of these listeners in Selenium, it’s easier to add additional functionalities to your test cases.

**Thumb Rule For Parallelism**

* If you have more than one class, then do parallelism at **Class** Level
* If you have only one class(with lots of @Test), then do parallelism at **methods** level

Running groups:

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">

<suite name="Suite">

<test thread-count="5" name="Test">

<groups>

<run>

<include name="smoke"> </include>

<exclude name="regression"> </exclude>

</run>

</groups>

<classes>

<class name="pageobject.demowebshop.TestNGDemoApp1"/>

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

Paramet

*suite name = “TestNG suite”>*

*<test name=”ExplicitWait”>*

*<parameter name=”firstName″ value=”Tony″ />*

*<parameter name=”LastName value=”Harris″ />*

*<classes>*

*<class name=”org.example.ExplicitWait” />*

*</classes>*

*</test>*

*</suite>*

*@Test*

*@Parameters({“firstName″”, “LastName”})*

*Public void firstTest(String firstName, String lastName ){*

*}*