**TestNG Data Providers**

In TestNG, a **DataProvider** is a feature that allows you to run a test method multiple times with different sets of data. This is particularly useful for data-driven testing, where you want to validate a method or function against multiple inputs and expected outputs.

**Key Concepts of Data Providers**

1. **@DataProvider Annotation**:
   * The @DataProvider annotation is used to define a method that provides data for test methods.
   * The method marked with @DataProvider must return an Object[][], where each row in the array represents a set of parameters to be passed to the test method.
2. **@Test with DataProvider**:
   * To use data from a DataProvider, a test method should specify dataProvider as an attribute in the @Test annotation.
   * Example: @Test(dataProvider = "dataProviderName")
3. **Name Attribute**:
   * By default, the name of the DataProvider method is used to refer to it in @Test.
   * You can assign a custom name using @DataProvider(name = "customName").
4. **Parallel Execution**:
   * DataProvider supports parallel execution by setting the parallel attribute to true, which can improve test performance.

**Practical Example**

import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;

public class DataProviderExample {

@DataProvider(name = "testData")

public Object[][] createData() {

return new Object[][] {

{"Alice", 25},

{"Bob", 30},

{"Charlie", 35}

};

}

@Test(dataProvider = "testData")

public void testMethod(String name, int age) {

System.out.println("Name: " + name + ", Age: " + age);

// Add assertions or test logic here

}

}

**Explanation**:

* The createData method returns an array of data with name and age pairs.
* The testMethod is run three times, once for each data set provided by createData.

**Advantages of Data Providers**

* **Reusable Data**: Data can be reused across multiple tests.
* **Data-Driven Testing**: Makes it easy to run tests with multiple data sets, which is essential for thorough testing.
* **Parameterization Flexibility**: Enables parameterization without relying on external files.

**Use Cases for Data Providers**

* Validating form input fields with various data.
* Testing login functionality with different usernames and passwords.
* Checking the behaviour of a function under multiple inputs to ensure stability and correctness.

### 1. ****TestNG Parameterization with**** @DataProvider ****and**** dataProvider ****Attribute****

The @DataProvider annotation in TestNG allows you to create a data provider method that supplies data for test methods. The test method can then refer to this provider using the dataProvider attribute.

import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;

public class DataProviderExample {

@DataProvider(name = "loginData")

public Object[][] createData() {

return new Object[][] {

{ "user1", "password1" },

{ "user2", "password2" }

};

}

@Test(dataProvider = "loginData")

public void loginTest(String username, String password) {

System.out.println("Username: " + username + ", Password: " + password);

}

}

In this example, loginData provides two sets of username and password, which loginTest will receive as parameters.

### 2. ****Naming Data Provider Method using**** name ****Attribute****

The name attribute of @DataProvider assigns a custom name to the data provider, which is referenced in the test method’s dataProvider attribute.

@DataProvider(name = "customName")

public Object[][] dataProviderMethod() {

return new Object[][] { { "test1" }, { "test2" } };

}

@Test(dataProvider = "customName")

public void testMethod(String data) {

System.out.println("Data: " + data);

}

Here, the @DataProvider method is named "customName", which the testMethod references in the dataProvider attribute.

### 3. ****Parameterizing Test Method with Array Parameter****

You can also pass arrays as parameters to your test method via @DataProvider.

@DataProvider(name = "arrayDataProvider")

public Object[][] arrayDataProvider() {

return new Object[][] {

{ new String[] { "data1", "data2" } },

{ new String[] { "data3", "data4" } }

};

}

@Test(dataProvider = "arrayDataProvider")

public void arrayTest(String[] dataArray) {

System.out.println("Data: " + String.join(", ", dataArray));

}

In this example, the test method arrayTest receives an array parameter from arrayDataProvider.

### 4. ****Separating Data Provider into a Separate Class Using**** dataProviderClass ****Attribute****

You can move your data provider methods to a separate class and specify it with the dataProviderClass attribute.

import org.testng.annotations.DataProvider;

public class DataProviderClass {

@DataProvider(name = "externalData")

public static Object[][] provideData() {

return new Object[][] { { "external1" }, { "external2" } };

}

}

Then, reference it in the test class:

import org.testng.annotations.Test;

public class SeparateDataProviderExample {

@Test(dataProvider = "externalData", dataProviderClass = DataProviderClass.class)

public void testWithExternalData(String data) {

System.out.println("Data: " + data);

}

}

This setup keeps your test code clean and organized, especially when multiple test classes share the same data providers.

### 5. ****Multiple Data Provider Methods in Multiple Classes****

You can have multiple data provider methods in different classes and specify the class and provider names as needed in your tests.

@Test(dataProvider = "provider1", dataProviderClass = Class1.class)

public void test1(String data) {

System.out.println("Data from provider1: " + data);

}

@Test(dataProvider = "provider2", dataProviderClass = Class2.class)

public void test2(String data) {

System.out.println("Data from provider2: " + data);

}

Each test method here uses data from different classes and data provider methods.

### 6. ****Why We Have to Use Object Array to Return Data****

TestNG requires data providers to return Object[][] because it provides flexibility for tests with parameters of different types. The outer Object[] array represents each test invocation, and the inner Object[] array represents the parameters for a single test invocation.

For example, with an Object[][] return type, you can provide data with mixed types:

@DataProvider(name = "mixedDataProvider")

public Object[][] mixedDataProvider() {

return new Object[][] {

{ "username", 12345, true },

{ "anotherUser", 67890, false }

};

}

### 7. ****Different Return Types of Data Provider****

Although Object[][] is commonly used, a data provider can also return Iterator<Object[]>, Iterator<Object>, or Stream<Object[]>.

**Using Iterator<Object[]>:**

@DataProvider(name = "iteratorDataProvider")

public Iterator<Object[]> iteratorDataProvider() {

List<Object[]> data = new ArrayList<>();

data.add(new Object[] { "data1" });

data.add(new Object[] { "data2" });

return data.iterator();

}

**Using Stream<Object[]>:**

@DataProvider(name = "streamDataProvider")

public Stream<Object[]> streamDataProvider() {

return Stream.of(new Object[] { "data1" }, new Object[] { "data2" });

}

Returning Stream is useful when you want to take advantage of Java 8+ functional programming capabilities.

### 8. ****Jagged Arrays and Data Providers****

Jagged arrays are arrays of arrays where inner arrays may have different lengths. In @DataProvider, this is particularly useful when your test method requires a variable number of parameters.

@DataProvider(name = "jaggedArrayProvider")

public Object[][] jaggedArrayProvider() {

return new Object[][] {

{ "data1" },

{ "data2", "extraData2" },

{ "data3", "extraData3", "additionalData3" }

};

}

@Test(dataProvider = "jaggedArrayProvider")

public void testWithJaggedArray(Object... params) {

System.out.println("Parameters: " + Arrays.toString(params));

}

The testWithJaggedArray method can accept a variable number of arguments using Object... params, allowing you to handle jagged arrays in your data providers flexibly.