TestNG @Parameters Example

In **TestNG**, the **`@Parameters`** annotation is used to pass values from an external XML file (typically **`testng.xml`**) to your test methods or configuration methods.

✅ Use Case:

To run the same test with different values (like browser name, environment, URL, etc.) without hardcoding them in the test class.

🧩 Example of `@Parameters` in TestNG

1. testng.xml

```xml
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="MyTestSuite">
  <test name="MyTest">
    <parameter name="browser" value="chrome"/>
    <parameter name="url" value="https://example.com"/>
    <classes>
      <class name="tests.MyTestClass"/>
    </classes>
  </test>
</suite>
```

2. Java Test Class

```java
package tests;

import org.testng.annotations.Parameters;
import org.testng.annotations.Test;

public class MyTestClass {

  @Test
  @Parameters({"browser", "url"})
  public void testLogin(String browser, String url) {
    System.out.println("Browser: " + browser);
    System.out.println("URL: " + url);
    // You can now launch the browser and navigate to the URL
  }
}
```

🔧 Notes:

- The parameter names in the Java method should match exactly with those in **`testng.xml`**.
- Parameters can also be used with **`@BeforeTest`**, **`@BeforeClass`**, etc.

- If a required parameter is missing, TestNG will throw an exception unless a default value is provided using **`@Optional`**.

💡 Using `@Optional`

```
java


@Parameters({"browser"})
@Test
public void testWithOptional(@Optional("firefox") String browser) {
  System.out.println("Running on browser: " + browser);
}
```

This will use **`"firefox"`** if no parameter is provided in **`testng.xml`**.

Would you like an example with multiple classes or using it with cross-browser testing?

Great! Let's walk through a full **cross-browser testing setup** using `@Parameters` from `testng.xml` with **multiple classes**.

🔪 Scenario: Cross-Browser Login Test

We'll create:

1. A `testng.xml` to pass different browser values
2. A `BaseTest` class to handle browser setup
3. Two test classes (`LoginTest` and `DashboardTest`)

🗂️ 1. testng.xml

```xml
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="CrossBrowserSuite" parallel="tests" thread-count="2">
  <test name="ChromeTest">
    <parameter name="browser" value="chrome"/>
    <classes>
      <class name="tests.LoginTest"/>
      <class name="tests.DashboardTest"/>
    </classes>
  </test>

  <test name="FirefoxTest">
    <parameter name="browser" value="firefox"/>
    <classes>
      <class name="tests.LoginTest"/>
      <class name="tests.DashboardTest"/>
    </classes>
  </test>
</suite>
```

🧱 2. BaseTest.java

```java
package tests;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.*;

public class BaseTest {
    protected WebDriver driver;

    @Parameters("browser")
    @BeforeClass
    public void setUp(String browser) {
        if (browser.equalsIgnoreCase("chrome")) {
            driver = new ChromeDriver();
        } else if (browser.equalsIgnoreCase("firefox")) {
            driver = new FirefoxDriver();
        } else {
            throw new IllegalArgumentException("Browser not supported: " + browser);
        }
```

```
        driver.manage().window().maximize();
        driver.get("https://example.com");
    }

    @AfterClass
    public void tearDown() {
        if (driver != null) {
            driver.quit();
        }
    }
}
```

✅ 3. LoginTest.java

```
java

package tests;

import org.testng.annotations.Test;

public class LoginTest extends BaseTest {

    @Test
    public void testLoginFunctionality() {
        System.out.println("Login Test - Title: " + driver.getTitle());
    }
}
```

✅ 4. DashboardTest.java

```
java

package tests;

import org.testng.annotations.Test;

public class DashboardTest extends BaseTest {

    @Test
    public void testDashboardLoads() {
        System.out.println("Dashboard Test - URL: " + driver.getCurrentUrl());
    }
}
```

🏁 Execution Summary

Test	Browser	Classes Executed
ChromeTest	Chrome	LoginTest, DashboardTest
FirefoxTest	Firefox	LoginTest, DashboardTest

✅ Both browsers run in parallel, and each test class uses the correct WebDriver instance.

💡 Tips:

- Install required browser drivers (**`chromedriver`**, **`geckodriver`**) and set the path.
- Use **`@BeforeMethod`** instead of **`@BeforeClass`** if you want a fresh browser for each test method.

Would you like this example modified to work with **DataProviders** or **grid setup** next?

Excellent! Let's now cover the **`@DataProvider` annotation** in TestNG — used for **data-driven testing** (running the same test method multiple times with different sets of data).

🔍 What is `@DataProvider`?

- **`@DataProvider`** supplies test data to a **`@Test`** method.
- It returns a 2D **`Object[][]`** array.
- It allows running a test multiple times with different input values.

🔪 Example: Login Test with Multiple Credentials

✅ 1. Using `@DataProvider` in the Same Class

```
java

import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

public class LoginTest {

    @DataProvider(name = "loginData")
    public Object[][] getLoginData() {
        return new Object[][] {
            {"admin", "admin123"},
            {"user1", "user123"},
            {"guest", "guest123"}
        };
    }

    @Test(dataProvider = "loginData")
    public void testLogin(String username, String password) {
        System.out.println("Username: " + username + ", Password: " + password);
        // simulate login using these credentials
    }
}
```

📃 Output:

```
yaml

Username: admin, Password: admin123
Username: user1, Password: user123
Username: guest, Password: guest123
```

🧱 2. Using `@DataProvider` from a Separate Class

a. Create a Data Provider Class

```
java

package utils;

import org.testng.annotations.DataProvider;
```

```
public class DataProviders {

    @DataProvider(name = "userData")
    public static Object[][] provideUserData() {
        return new Object[][] {
            {"alice", "pass1"},
            {"bob", "pass2"}
        };
    }
}
```

b. Use it in your Test Class

```
java

import org.testng.annotations.Test;

public class UserTest {

    @Test(dataProvider = "userData", dataProviderClass = utils.DataProviders.class)
    public void testUserLogin(String username, String password) {
        System.out.println("Testing login for " + username);
    }
}
```

🛠️ Additional Tips

🔁 DataProvider with Indexes:

You can selectively run specific data sets using:

```
xml

<method-selectors>
  <method-selector>
    <selector-class>
      <name>org.testng.internal.MethodSelector</name>
      <priority>0</priority>
    </selector-class>
  </method-selector>
</method-selectors>
```

🧩 DataProvider with Complex Types:

```
java

@DataProvider(name = "userObjects")
public Object[][] provideObjects() {
    return new Object[][] {
        {new User("john", "1234")},
        {new User("jane", "abcd")}
    };
}
```

🔄 Difference Between `@Parameters` vs `@DataProvider`

Feature	@Parameters	@DataProvider
Source	**`testng.xml`**	Java method
Data Type	Strings only	Any Object
Use Case	Pass config/environment data	Data-driven test inputs
Flexibility	Limited	Highly flexible

Would you like a hands-on lab where we test login functionality using **`@DataProvider`** + Selenium?

Perfect! You're referring to **data-driven testing**, where we **run the same test multiple times with different data**. In TestNG, this is elegantly handled using the `@DataProvider` annotation.

✅ Goal: Run the same test with multiple sets of data

Let's simulate this with a **login test** that takes different username/password combinations.

🔪 Full Working Example: Login Test with Multiple Credentials

📂 Project Structure

```
css

src/
├── tests/
│   └── LoginTest.java
```

🧱 Step 1: `LoginTest.java`

```
java

package tests;

import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

public class LoginTest {

    // DataProvider method returning multiple data sets
    @DataProvider(name = "loginCredentials")
    public Object[][] loginData() {
        return new Object[][] {
            {"admin", "admin123"},
            {"user1", "user123"},
            {"guest", "guest123"},
            {"invalid", "wrongpass"}
        };
    }

    // Test method that uses the data from the DataProvider
    @Test(dataProvider = "loginCredentials")
    public void testLogin(String username, String password) {
        System.out.println("Testing login with Username: " + username + ", Password: " +
password);
        // Add your actual login test logic here
        // e.g., driver.findElement(...).sendKeys(username)...
    }
}
```

🔁 What Happens When You Run This Test?

TestNG will automatically **run the `testLogin()` method 4 times**, once for each set of username/password pairs.

📄 Sample Output:

```
pgsql

Testing login with Username: admin, Password: admin123
Testing login with Username: user1, Password: user123
Testing login with Username: guest, Password: guest123
Testing login with Username: invalid, Password: wrongpass
```

🧠 Benefits of Using `@DataProvider`

Feature	Benefit
Reusability	Centralize test data for multiple tests
Scalability	Easily add more test data without code changes
No test duplication	One test method, many test runs

💡 Want to Combine It with Selenium?

Here's how you'd use this with Selenium to test an actual login page:

```
java

@Test(dataProvider = "loginCredentials")
public void testLogin(String username, String password) {
    WebDriver driver = new ChromeDriver();
    driver.get("https://example.com/login");

    driver.findElement(By.id("username")).sendKeys(username);
    driver.findElement(By.id("password")).sendKeys(password);
    driver.findElement(By.id("login")).click();

    // Assertions here (e.g., check for login success or error)

    driver.quit();
}
```

Would you like to take this further with:

- Reading data from **Excel or CSV**?
- Passing **complex objects (e.g., POJOs)**?
- Using `@DataProvider` with **parallel test execution**?

Let me know your use case!