TestNG Documentation for Selenium with Java

TestNG (Test Next Generation) is a testing framework inspired by JUnit and NUnit, but with enhanced features. It simplifies testing in Selenium-Java projects with annotations, grouping, parameterization, and parallel execution.

1. Setup TestNG

Add to Maven pom.xml:

```
<dependency>
  <groupId>org.testng</groupId>
  <artifactId>testng</artifactId>
  <version>7.8.0</version>
</dependency>
```

2. TestNG Annotations

1. Suite-Level Annotations

Annotation	Description	Example
@BeforeSuite	Runs once before all tests in the suite.	@BeforeSuite public void setupSuite() { ... }
@AfterSuite	Runs once after all tests in the suite.	@AfterSuite public void teardownSuite() { ... }

2. Test-Level Annotations

Annotation	Description	Example
@BeforeTest	Runs before each <test> block in the XML file.	@BeforeTest public void setupTest() { ... }
@AfterTest	Runs after each <test> block in the XML file.	@AfterTest public void teardownTest() { ... }

3. Class-Level Annotations

Annotation	Description	Example
@BeforeClass	Runs once per class before any test method in the class.	@BeforeClass public void setupClass() { ... }
@AfterClass	Runs once per class after all test methods in the class.	@AfterClass public void teardownClass() { ... }

4. Method-Level Annotations

Annotation	Description	Example
@BeforeMethod	Runs before each test method in a class.	@BeforeMethod public void setupMethod() { ... }
@AfterMethod	Runs after each test method in a class.	@AfterMethod public void teardownMethod() { ... }

5. Test Case Annotation

Annotation	Description	Example
@Test	Marks a method as a test case.	@Test public void loginTest() { ... }

Execution Order

Annotations run in this hierarchy: @BeforeSuite → @BeforeTest → @BeforeClass → @BeforeMethod → @Test → @AfterMethod → @AfterClass → @AfterTest → @AfterSuite

3. Test Execution Order

Control order using priority:

```
@Test(priority = 1)
public void launchBrowser() { ... }


@Test(priority = 2)
public void loginTest() { ... }
```

4. Grouping Tests

Group tests using groups parameter:

```
@Test(groups = "smoke")
public void homePageTest() { ... }

@Test(groups = "regression")
public void checkoutTest() { ... }
```

Run groups via testng.xml:

```
<groups>
<run>
   <include name="smoke"/>
</run>
</groups>
```

5. Parameters

XML Parameters:

```
@Test
@Parameters({"url", "username"})
public void loginTest(String url, String username) {
   driver.get(url);
   // Use username
}
```

In testng.xml:

```
<parameter name="url" value="https://example.com"/>
<parameter name="username" value="admin"/>
```

DataProvider:

```
@DataProvider(name = "loginData")
public Object[][] provideData() {
   return new Object[][] {
         {"user1", "pass1"},
         {"user2", "pass2"}
   };
}

@Test(dataProvider = "loginData")
public void dataDrivenTest(String user, String pass) {
   // Test with user/pass
}
```

DataProvider in Separate Class:

```
public class DataProviders {
   @DataProvider(name = "loginData")
   public static Object[][] loginData() {
      return new Object[][] {
            {"user1", "pass1"},
            {"user2", "pass2"}
      };
   }
}

//In test class:
@Test(dataProvider = "loginData", dataProviderClass = DataProviders.class)
public void loginTest(String user, String pass) {
   // Test implementation
}
```

6. Dependencies

Run tests conditionally:

```
@Test
public void setup() { ... }

@Test(dependsOnMethods = "setup")
public void mainTest() { ... } // Runs only if setup() passes

//Skip dependent tests on failure:
@Test(dependsOnMethods = "setup", alwaysRun = true)
public void cleanupTest() { ... } // Runs even if setup() fails
```

7. Assertions

Use TestNG assertions:

```
import static org.testng.Assert.*;

@Test
public void assertionTest() {
   assertEquals(actual, expected);
   assertTrue(condition);
   assertNotNull(object);
   assertFalse(condition);
   assertEquals(actual, expected, "Custom error message");
}
```

Soft Assertions

For multiple validations without stopping on the first failure:

```
import org.testng.asserts.SoftAssert;

@Test
public void softAssertExample() {
   SoftAssert softAssert = new SoftAssert();

   softAssert.assertEquals(1, 2, "Numbers should be equal"); // Fails but continues
   softAssert.assertTrue(5 > 3, "5 should be greater than 3"); // Passes

   //Important: Call assertAll() at the end to report all failures:
   softAssert.assertAll();
}
```

8. Parallel Execution

Run tests in parallel via testng.xml:

```
<suite name="ParallelSuite" parallel="methods" thread-count="3">
<test name="ParallelTest">
  <classes>
    <class name="com.example.TestClass"/>
  </classes>
</test>
</suite>
```

Parallel options:

- methods: Run test methods in parallel
- classes: Run test classes in parallel
- tests: Run <test> tags in parallel
- instances: Run different instances of test classes in parallel

9. Listeners

Implement listeners for logging/reporting:

```
public class CustomListener implements ITestListener {
    @Override
    public void onTestSuccess(ITestResult result) {
        System.out.println("Test passed: " + result.getName());
    }

    @Override
    public void onTestFailure(ITestResult result) {
        // Capture screenshot for Selenium tests
        System.out.println("Test failed: " + result.getName());
    }

    @Override
    public void onStart(ITestContext context) {
        System.out.println("Starting test: " + context.getName());
    }

    @Override
    public void onFinish(ITestContext context) {
        System.out.println("Finishing test: " + context.getName());
    }
}
```

Add to testng.xml:

```
<listeners>
<listener class-name="com.example.CustomListener"/>
</listeners>
```

Or add at the class level:

```
@Listeners(com.example.CustomListener.class)
public class MyTestClass {
    // Test methods
}
```

10. TestNG Reports

TestNG generates HTML reports in test-output/ by default. Use emailable-report.html for summaries.

Enhanced Reporting with ExtentReports

Add ExtentReports dependency:

```
<dependency>
  <groupId>com.aventstack</groupId>
  <artifactId>extentreports</artifactId>
  <version>5.0.9</version>
</dependency>
```

Create ExtentReports listener:

```
public class ExtentReportListener implements ITestListener {
    private ExtentReports extent;
    private ThreadLocal<ExtentTest> test = new ThreadLocal<>();

    @Override
    public void onStart(ITestContext context) {
        String reportPath = "reports/extent-report.html";
        extent = new ExtentReports();
        extent.attachReporter(new ExtentHtmlReporter(reportPath));
    }

    @Override
    public void onTestStart(ITestResult result) {
        String methodName = result.getMethod().getMethodName();
        ExtentTest testReport = extent.createTest(methodName);
        test.set(testReport);
    }

    @Override
    public void onTestSuccess(ITestResult result) {
        test.get().pass("Test passed");
    }

    @Override
    public void onTestFailure(ITestResult result) {
        test.get().fail(result.getThrowable());
        // Add screenshot capture code here for Selenium tests
    }

    @Override
    public void onFinish(ITestContext context) {
        extent.flush();
    }
}
```

11. Expected Exceptions

Testing exceptions:

```
@Test(expectedExceptions = IllegalArgumentException.class)
public void exceptionTest() {
   throw new IllegalArgumentException("Expected exception");
}

//With specific message check:
@Test(expectedExceptions = IllegalArgumentException.class,
      expectedExceptionsMessageRegExp = "Expected.*")
public void exceptionMessageTest() {
   throw new IllegalArgumentException("Expected exception");
}
```

12. Timeouts

Set timeouts for tests:

```
//Test fails if it takes longer than 5000ms
@Test(timeOut = 5000)
public void timeoutTest() {
   // Test implementation
}
```

13. Ignoring Tests

Skip tests:

```
// Skip this test
@Test(enabled = false)
public void skippedTest() {
   // This test will not run
}
```

14. Factory Pattern

Create test instances dynamically:

```
public class ParameterizedTest {
    private String param;

    public ParameterizedTest(String param) {
        this.param = param;
    }

    @Test
    public void testParameter() {
        System.out.println("Parameter: " + param);
    }
}

public class TestFactory {
    @Factory
    public Object[] createTests() {
        return new Object[] {
                new ParameterizedTest("value1"),
                new ParameterizedTest("value2")
        };
    }
}
```

15. Sample TestNG XML File

```xml
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="SampleSuite">
  <test name="RegressionTests">
    <parameter name="browser" value="chrome"/>
    <groups>
      <run>
        <include name="regression"/>
        <exclude name="broken"/>
      </run>
    </groups>
    <classes>
      <class name="com.example.LoginTests"/>
      <class name="com.example.CheckoutTests"/>
    </classes>
  </test>

  <test name="SmokeTests">
    <parameter name="browser" value="firefox"/>
    <classes>
      <class name="com.example.SmokeTests"/>
    </classes>
  </test>
</suite>
```

16. Running Tests

- IDE: Right-click test class or XML file → Run
- Maven: mvn test
- Command Line: java -cp "path/to/testng.jar:path/to/your/classes" org.testng.TestNG testng.xml
- Gradle: ./gradlew test

17. Configuration Failures

Handle setup and teardown failures:

```java
@Test(configFailurePolicy = ConfigurationFailurePolicy.CONTINUE)
public void testMethod() {
  // This will run even if @BeforeMethod fails
}
```

18. Retry Failed Tests

Create a retry analyzer:

```
public class RetryAnalyzer implements IRetryAnalyzer {
    private int count = 0;
    private static final int MAX_RETRY = 2;

    @Override
    public boolean retry(ITestResult result) {
        if (!result.isSuccess()) {
            if (count < MAX_RETRY) {
                count++;
                return true; // Retry the test
            }
        }
        return false; // No more retries
    }
}

//Apply to a test
@Test(retryAnalyzer = RetryAnalyzer.class)
public void flakyTest() {
    // This test will retry up to MAX_RETRY times if it fails
}
```

19. Integration with Selenium

Sample Selenium test using TestNG:

```java
public class LoginTest {
    private WebDriver driver;
    @BeforeClass
    public void setUp() {
        WebDriverManager.chromedriver().setup();
        driver = new ChromeDriver();
        driver.manage().window().maximize();
        driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
    }

    @Test
    public void validLogin() {
        driver.get("https://example.com/login");
        driver.findElement(By.id("username")).sendKeys("validUser");
        driver.findElement(By.id("password")).sendKeys("validPass");
        driver.findElement(By.id("loginButton")).click();
        String actualTitle = driver.getTitle();
        Assert.assertEquals(actualTitle, "Dashboard", "Login failed");
    }

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

20. Best Practices

1. Use priority sparingly – prefer dependencies for test flow
2. Group related tests (e.g., smoke, regression)
3. Use DataProviders for data-driven tests
4. Combine with Page Object Model (POM) for Selenium
5. Keep test methods focused on a single functionality
6. Use meaningful test method names (e.g., verifyLoginWithValidCredentials)
7. Use soft assertions when multiple validations are needed
8. Implement listeners for better logging and reporting
9. Use parametrization to reduce duplicate test code
10. Add proper exception handling for setup and teardown methods