Selenium TestNG Advanced Concepts

---

1. Loggers - Log4J2

Why? Logging helps track execution, debug issues, and generate detailed runtime information.

Setup Log4J2

- Add dependency (Maven `pom.xml`):

```
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.23.1</version>
</dependency>
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.23.1</version>
</dependency>
```

log4j2.xml (config file)

```
<Configuration status="WARN">
  <Appenders>
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout pattern="%d [%t] %-5level %logger{36} - %msg%n"/>
    </Console>
    <File name="FileLogger" fileName="logs/app.log">
      <PatternLayout>
        <Pattern>%d [%t] %-5p %c - %m%n</Pattern>
      </PatternLayout>
    </File>
  </Appenders>

  <Loggers>
    <Root level="info">
      <AppenderRef ref="Console"/>
      <AppenderRef ref="FileLogger"/>
    </Root>
  </Loggers>
</Configuration>
```

Usage in Selenium Test

```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class LogExample {
    private static final Logger log = LogManager.getLogger(LogExample.class);

    @Test
    public void testLogin() {
        log.info("Starting login test");
        driver.get("https://example.com");
        log.debug("Navigated to application");
        log.warn("This is a warning example");
        log.error("This is an error example");
        Assert.assertTrue(true);
        log.info("Test completed");
    }
}
```

---

2. Reporting in Selenium WebDriver

Reports show test results (pass/fail/skipped) and execution logs.

Types:

- **TestNG Default Report** (simple HTML)
- **Extent Report** (detailed HTML with screenshots)
- **Allure Report** (attractive dashboard style)

---

3. TestNG Annotations

Annotations control test execution flow.

Syntax:

```
@BeforeSuite
@BeforeTest
@BeforeClass
@BeforeMethod
@Test
@AfterMethod
@AfterClass
@AfterTest
@AfterSuite
```

Example:

```
@BeforeClass
public void setup() {
    driver = new ChromeDriver();
}

@Test
public void testGoogle() {
    driver.get("https://google.com");
    Assert.assertEquals(driver.getTitle(), "Google");
}

@AfterClass
public void teardown() {
    driver.quit();
}
```

---

4. Data-Driven Testing

Test cases use **external data (Excel/DB/CSV)**.

Excel + Apache POI Example:

```
@DataProvider(name = "LoginData")
public Object[][] getData() throws Exception {
    return new Object[][] {
        {"admin","admin123"},
        {"user","user123"}
    };
}

@Test(dataProvider = "LoginData")
public void testLogin(String username, String password) {
    driver.findElement(By.id("username")).sendKeys(username);
    driver.findElement(By.id("password")).sendKeys(password);
    driver.findElement(By.id("login")).click();
}
```

---

5. Grouping of Tests

Execute specific groups of test cases.

```
@Test(groups = {"smoke"})
public void test1() { }

@Test(groups = {"regression"})
public void test2() { }
```

Run via `testng.xml`:

```
<test name="RegressionSuite">
  <groups>
    <run>
      <include name="regression"/>
    </run>
  </groups>
  <classes>
    <class name="tests.SampleTest"/>
  </classes>
</test>
```

---

6. Parallel Testing

Run multiple tests simultaneously.

```
<suite name="Suite" parallel="tests" thread-count="2">
  <test name="ChromeTest">
    <parameter name="browser" value="chrome"/>
    <classes><class name="tests.ParallelTest"/></classes>
  </test>
  <test name="FirefoxTest">
    <parameter name="browser" value="firefox"/>
    <classes><class name="tests.ParallelTest"/></classes>
  </test>
</suite>
```

---

7. Parameterization

Passing values at runtime via `testng.xml`.

```
<test name="ParamTest">
  <parameter name="url" value="https://google.com"/>
  <parameter name="browser" value="chrome"/>
  <classes>
    <class name="tests.ParamTest"/>
  </classes>
</test>
@Parameters({"url","browser"})
@Test
public void openSite(String url, String browser) {
    if(browser.equals("chrome")) driver = new ChromeDriver();
    driver.get(url);
}
```

---

8. Dependency Test

Execute one test only if another passes.

```
@Test
public void startApp() { System.out.println("App started"); }

@Test(dependsOnMethods = {"startApp"})
public void login() { System.out.println("Logged in"); }
```

---

9. Multi-threaded Testing Support

TestNG supports parallel test execution by threads.

```
@Test(threadPoolSize = 3, invocationCount = 6, timeOut = 1000)
public void testMultiThread() {
    System.out.println("Thread ID: " + Thread.currentThread().getId());
}
```

---

10. Assertions

Used to validate conditions.

```
@Test
public void testTitle() {
    driver.get("https://google.com");
    Assert.assertEquals(driver.getTitle(), "Google");
    Assert.assertTrue(driver.findElement(By.name("q")).isDisplayed());
}
```

---

11. Cross Browser Testing

Run same tests on multiple browsers.

```
@Parameters("browser")
@BeforeTest
public void setup(String browser) {
    if(browser.equals("chrome"))
        driver = new ChromeDriver();
    else if(browser.equals("firefox"))
        driver = new FirefoxDriver();
}
```

---

12. Test Suites

Collection of test cases.

```
<suite name="Suite">
  <test name="Regression">
    <classes>
      <class name="tests.LoginTest"/>
      <class name="tests.HomeTest"/>
    </classes>
  </test>
</suite>
```

---

13. Data Providers

Advanced data-driven test execution.

```
@DataProvider(name="data")
public Object[][] getData() {
    return new Object[][] { {"admin","admin123"}, {"user","user123"} };
}

@Test(dataProvider="data")
public void testLogin(String user, String pass) { ... }
```

---

14. TestNG Listeners

Listeners help customize reporting.

```
public class MyListener implements ITestListener {
    public void onTestFailure(ITestResult result) {
        System.out.println("Test Failed: " + result.getName());
    }
}
<listeners>
  <listener class-name="tests.MyListener"/>
</listeners>
```

---

15. Extent Report

Advanced HTML report with screenshots.

```
ExtentReports extent;
ExtentTest test;

@BeforeTest
public void startReport() {
    extent = new ExtentReports();
    ExtentSparkReporter spark = new ExtentSparkReporter("report.html");
    extent.attachReporter(spark);
}

@Test
public void demoReport() {
    test = extent.createTest("Google Test");
    driver.get("https://google.com");
    test.pass("Opened Google");
}

@AfterTest
public void endReport() {
    extent.flush();
```

```
}
```

---

16. Selenium Automation Framework

A complete framework generally includes:

- **Base Class** (WebDriver setup, teardown)
- **Page Object Model (POM)** (separates UI locators & actions)
- **Utilities** (Excel reader, Config reader, Screenshot)
- **TestNG for execution**
- **Log4J2 for logging**
- **ExtentReports for reporting**
- **Data-driven + Cross-browser support**
- **CI/CD integration (Jenkins, GitHub Actions)**