# **Advanced Selenium Framework Topics**

## **🧪 TestNG Framework Basics**

| Concept | Description |
| --- | --- |
| What is TestNG? | A testing framework (like JUnit) designed for unit, functional, integration, and end-to-end tests in Java. Provides annotations, grouping, prioritization, parallelism, reporting. |
| Why use TestNG? | Clear test flow with annotations  Flexible test configuration  Easy grouping & prioritization  Built-in reports  Parallel execution |
| How to use? | Add TestNG to your project (Maven: testng dependency)  Write test classes with @Test methods  Run via TestNG runner or XML config |
| Where is it used? | Automation frameworks (Selenium + TestNG)  API testing  Unit/integration testing for Java apps |

## **📝 Common Annotations**

| Annotation | Purpose |
| --- | --- |
| @Test | Marks a method as a test |
| @BeforeMethod | Runs before each @Test method |
| @AfterMethod | Runs after each @Test method |
| @BeforeClass | Runs once before first test in the class |
| @AfterClass | Runs once after all tests in the class |
| @BeforeSuite | Runs before entire test suite |
| @AfterSuite | Runs after entire test suite |

## **⚡ Features**

✅ Group tests @Test(groups = {"smoke", "regression"})  
✅ Prioritize tests (priority=1)  
✅ Parameterize (@Parameters)  
✅ Run tests in parallel @Test(threadPoolSize = 3, invocationCount = 10)

✅ Dependencies @Test(dependsOnMethods = "loginTest")  
✅ Custom reports

### **Key Features**

* Parallel Execution: @Test(threadPoolSize = 3, invocationCount = 10)
* Grouping Tests: @Test(groups = {"smoke", "regression"})
* Dependencies: @Test(dependsOnMethods = "loginTest")

### **Core Components**

Java

import org.testng.annotations.\*;

@BeforeSuite

public void setupSuite() {

// One-time initialization

}

@BeforeTest

public void setupTest() {

// Test-level setup

}

@BeforeMethod

public void setup() {

// Method-level setup

driver = new ChromeDriver();

}

@Test(priority = 1)

public void loginTest() {

// Test logic

}

@AfterMethod

public void teardown() {

driver.quit();

}

## **✅ Assertions & Parameterization**

### **Assertion Types**

java

// Hard Assertions

Assert.assertEquals(actual, expected);

Assert.assertTrue(condition);

Assert.assertNotNull(object);

// Soft Assertions

SoftAssert softAssert = new SoftAssert();

softAssert.assertEquals(actual, expected, "Message");

softAssert.assertAll(); // Triggers validation

### **Parameterization Methods**

1. @DataProvider

java

@DataProvider(name = "loginData")

public Object[][] provideData() {

return new Object[][] {

{"user1", "pass123"},

{"user2", "pass456"}

};

}

@Test(dataProvider = "loginData")

public void testLogin(String username, String password) {

loginPage.enterCredentials(username, password);

}

2. XML Parameters

xml

<parameter name="browser" value="chrome"/>

java

@Parameters("browser")

@BeforeMethod

public void setup(String browser) {

// Initialize specified browser

}

## **🏗️ Page Object Model (POM)**

### **Implementation Pattern**

java

public class LoginPage {

// Elements

@FindBy(id = "username") WebElement usernameField;

@FindBy(id = "password") WebElement passwordField;

// Constructor

public LoginPage(WebDriver driver) {

PageFactory.initElements(driver, this);

}

// Methods

public void login(String user, String pass) {

usernameField.sendKeys(user);

passwordField.sendKeys(pass);

}

}

### **Best Practices**

* Separate page classes for each screen/module
* Return page objects from methods:

java

* public HomePage clickLogin() {

loginButton.click();

return new HomePage(driver);

}

* Use LoadableComponent pattern for critical pages

## **📊 Extent Reports Integration**

### **Setup & Configuration**

java

// Initialize reporter

ExtentSparkReporter spark = new ExtentSparkReporter("report.html");

ExtentReports extent = new ExtentReports();

extent.attachReporter(spark);

// Test logging

@Test

public void testWithReporting() {

ExtentTest test = extent.createTest("Login Test");

test.log(Status.INFO, "Entering credentials");

try {

loginPage.login("admin", "admin123");

test.pass("Login successful");

} catch (Exception e) {

test.fail("Test failed: " + e.getMessage());

test.addScreenCaptureFromPath(takeScreenshot());

}

}

@AfterSuite

public void generateReport() {

extent.flush();

}

### **Advanced Features**

* Custom themes: spark.config().setTheme(Theme.DARK);
* Categories: test.assignCategory("Smoke");
* Screenshots: test.addScreenCaptureFromPath("screenshot.png");

## **🔗 Framework Architecture**

text

test-project/

├── src/

│ ├── main/

│ │ └── java/

│ │ ├── pages/ # Page classes

│ │ ├── utilities/ # Helpers

│ │ └── base/ # TestBase

│ └── test/

│ ├── java/

│ │ ├── tests/ # Test classes

│ │ └── listeners/ # TestNG listeners

│ └── resources/

│ ├── testng.xml # Test suites

│ └── config.properties

├── reports/ # Extent reports

└── pom.xml # Maven config

## **🚀 Pro Tips**

1. Combine POM with Factory Pattern for dynamic page creation
2. Use RetryAnalyzer for flaky tests:

java

* @Test(retryAnalyzer = RetryAnalyzer.class)
* Leverage Event Listeners for advanced reporting:

java

1. public class TestListener implements ITestListener {

public void onTestFailure(ITestResult result) {

ExtentTestManager.getTest().fail(result.getThrowable());

}

}

This comprehensive guide covers essential advanced Selenium framework concepts with practical implementation examples. For production-grade frameworks, consider adding:

* Dependency Injection (Guice/Spring)
* BDD Integration (Cucumber)
* Visual Testing (Applitools)