**Java Selenium Document**

**1. Install Java**

**Step-by-Step Instructions:**

1. **Download Java Development Kit (JDK):**
   * Visit the [Oracle JDK download page](https://www.oracle.com/java/technologies/javase-downloads.html).
   * Choose the latest version suitable for your operating system (Windows, macOS, Linux).
2. **Install JDK:**
   * Run the downloaded installer.
   * Follow the on-screen instructions to complete the installation.
   * During installation, note the directory where JDK is being installed.
3. **Set JAVA\_HOME Environment Variable:**
   * Open System Properties (Right-click on 'This PC' or 'My Computer' -> Properties -> Advanced system settings).
   * Click on 'Environment Variables'.
   * Under System Variables, click 'New' and add:
     + Variable name: JAVA\_HOME
     + Variable value: Path to your JDK installation directory (e.g., C:\Program Files\Java\jdk-14.0.1).
   * Find the 'Path' variable, select it, and click 'Edit'. Add a new entry: %JAVA\_HOME%\bin.
4. **Verify Installation:**
   * Open Command Prompt and type java -version and javac -version.
   * If installed correctly, it will display the installed version of Java.

**2. Install Selenium Java**

**Step-by-Step Instructions:**

1. **Download Selenium Java Client:**
   * Visit the Selenium downloads page.
   * Download the Selenium Java Client Library.
2. **Add Selenium to Your Project:**
   * If you're using Maven, add the following dependency to your pom.xml:

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>4.1.0</version> <!-- Check for the latest version -->

</dependency>

* + If you're not using Maven, extract the downloaded Selenium zip file and add the JAR files to your project’s classpath.

**3. Install and Create Project in IntelliJ (POM, TestNG Setup)**

**Step-by-Step Instructions:**

1. **Install IntelliJ IDEA:**
   * Download IntelliJ IDEA from the [JetBrains website](https://www.jetbrains.com/idea/download/).
   * Install the software by following the on-screen instructions.
2. **Create a New Project:**
   * Open IntelliJ IDEA.
   * Click on 'New Project'.
   * Select 'Maven' and ensure that the Project SDK is set to the JDK you installed earlier.
   * Click 'Next' and provide a GroupId and ArtifactId for your project.
3. **Setup POM for Selenium and TestNG:**
   * Open the pom.xml file and add the following dependencies:

<dependencies>

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>4.1.0</version> <!-- Check for the latest version -->

</dependency>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>7.4.0</version> <!-- Check for the latest version -->

<scope>test</scope>

</dependency>

</dependencies>

1. **Install TestNG Plugin:**
   * Go to 'File' -> 'Settings' -> 'Plugins'.
   * Search for 'TestNG' and install it.
   * Restart IntelliJ IDEA if prompted.

Sure, here is a detailed report with specific guidance on each of the topics:

**4. Parallel Execution (Grid, Parameter, Read JSON File, configs.properties)**

**Step-by-Step Instructions:**

1. **Setup Selenium Grid:**
   * Download the Selenium Server standalone jar from the Selenium downloads page.
   * Start the hub:

java -jar selenium-server-standalone-<version>.jar -role hub

* + Start a node and register it with the hub:

java -jar selenium-server-standalone-<version>.jar -role node -hub http://localhost:4444/grid/register

1. **Parameterization in TestNG:**
   * Use @Parameters annotation:

@Test

@Parameters({"browser", "url"})

public void testMethod(String browser, String url) {

// Test code

}

* + Define parameters in testng.xml:

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

<parameter name="url" value="http://example.com"/>

1. **Reading JSON File:**
   * Add json-simple dependency to pom.xml:

<dependency>

<groupId>com.googlecode.json-simple</groupId>

<artifactId>json-simple</artifactId>

<version>1.1.1</version>

</dependency>

* + Read JSON file in your test:

JSONParser parser = new JSONParser();

Object obj = parser.parse(new FileReader("path/to/your/file.json"));

JSONObject jsonObject = (JSONObject) obj;

1. **Using config.properties File:**
   * Create a config.properties file in your project directory.
   * Add properties like browser=chrome and url=http://example.com.
   * Load properties in your test:

Properties prop = new Properties();

FileInputStream ip = new FileInputStream("path/to/config.properties");

prop.load(ip);

String browser = prop.getProperty("browser");

String url = prop.getProperty("url");

**Parallel Test Methods: Running Test Methods in the Same Class in Parallel**

**Step-by-Step Instructions:**

1. **TestNG XML Configuration:** Modify your testng.xml file to enable parallel execution at the method level:

<suite name="Suite" parallel="methods" thread-count="5">

<test name="Test">

<classes>

<class name="com.example.YourTestClass"/>

</classes>

</test>

</suite>

1. **Define Test Methods:** Define multiple test methods in the same class:

public class YourTestClass {

@Test

public void testMethod1() {

// Test code

}

@Test

public void testMethod2() {

// Test code

}

}

**Parallel Test Classes: Running Different Test Classes in Parallel**

**Step-by-Step Instructions:**

1. **TestNG XML Configuration:** Modify your testng.xml file to enable parallel execution at the class level:

<suite name="Suite" parallel="classes" thread-count="5">

<test name="Test">

<classes>

<class name="com.example.TestClass1"/>

<class name="com.example.TestClass2"/>

</classes>

</test>

</suite>

1. **Define Test Classes:** Create different test classes:

public class TestClass1 {

@Test

public void testMethod1() {

// Test code

}

}

public class TestClass2 {

@Test

public void testMethod2() {

// Test code

}

}

**Parallel Test Instances: Running Instances of the Same Test Class in Parallel**

**Step-by-Step Instructions:**

1. **TestNG XML Configuration:** Modify your testng.xml file to enable parallel execution at the instance level:

<suite name="Suite" parallel="instances" thread-count="5">

<test name="Test">

<classes>

<class name="com.example.YourTestClass"/>

</classes>

</test>

</suite>

1. **Create Multiple Instances:** Use a factory method to create multiple instances of your test class:

public class YourTestClass {

@Factory

public Object[] createInstances() {

return new Object[] { new YourTestClass(), new YourTestClass() };

}

@Test

public void testMethod() {

// Test code

}

}

**Parallel Test Suites: Running Different Test Suites in Parallel**

**Step-by-Step Instructions:**

1. **TestNG XML Configuration:** Create multiple testng.xml files, each representing a suite. For example, create suite1.xml and suite2.xml with different tests.
2. **Run Suites in Parallel:** Use TestNG’s command-line options to run the suites in parallel:

java -cp "path/to/testng.jar:path/to/your-tests.jar" org.testng.TestNG -suitethreadpoolsize 2 suite1.xml suite2.xml

**Parallel Execution with Data Providers: Running Tests in Parallel with Different Data Sets**

**Step-by-Step Instructions:**

1. **Define Data Provider:** Define a data provider method that supplies data sets to your test methods:

@DataProvider(name = "dataProvider", parallel = true)

public Object[][] dataProviderMethod() {

return new Object[][] { {"data1"}, {"data2"} };

}

@Test(dataProvider = "dataProvider")

public void testMethod(String data) {

// Test code using data

}

1. **TestNG XML Configuration:** Ensure your testng.xml file is set to run methods in parallel:

<suite name="Suite" parallel="methods" thread-count="5">

<test name="Test">

<classes>

<class name="com.example.YourTestClass"/>

</classes>

</test>

</suite>

**Parallel Execution with Factory Methods: Using @Factory to Create Multiple Test Instances with Different Data and Running Them in Parallel**

**Step-by-Step Instructions:**

1. **Define Factory Method:** Use @Factory to create test instances with different data:

public class YourTestClass {

private String data;

public YourTestClass(String data) {

this.data = data;

}

@Factory

public static Object[] factoryMethod() {

return new Object[] { new YourTestClass("data1"), new YourTestClass("data2") };

}

@Test

public void testMethod() {

// Test code using data

}

}

1. **TestNG XML Configuration:** Ensure your testng.xml file supports parallel execution:

<suite name="Suite" parallel="methods" thread-count="5">

<test name="Test">

<classes>

<class name="com.example.YourTestClass"/>

</classes>

</test>

</suite>

**Parallel Execution with TestNG Suites from Command Line: Running Test Suites in Parallel from the Command Line**

**Step-by-Step Instructions:**

1. **Create Multiple Suite XML Files:** Create different testng.xml files representing different suites.
2. **Run Suites in Parallel:** Use TestNG’s command-line options to run multiple suites in parallel:

java -cp "path/to/testng.jar:path/to/your-tests.jar" org.testng.TestNG -suitethreadpoolsize 5 suite1.xml suite2.xml suite3.xml

**Parallel Execution with Docker and Selenium Grid: Setting Up a Distributed Testing Environment with Docker and Running Tests in Parallel Across Multiple Containers**

**Step-by-Step Instructions:**

1. **Setup Docker and Selenium Grid:**
   * Install Docker from the Docker website.
   * Pull the Selenium Hub image:

docker pull selenium/hub

* + Pull the Selenium Node images (e.g., Chrome):

docker pull selenium/node-chrome

1. **Start Selenium Grid:**
   * Start the Selenium Hub:

docker run -d -p 4444:4444 --name selenium-hub selenium/hub

* + Start the Selenium Nodes and link them to the Hub:

docker run -d --link selenium-hub:hub selenium/node-chrome

1. **Configure Tests to Run on Selenium Grid:**
   * Modify your WebDriver setup to use the remote WebDriver pointing to the Selenium Hub:

WebDriver driver = new RemoteWebDriver(new URL("http://localhost:4444/wd/hub"), DesiredCapabilities.chrome());

1. **Run Tests:** Execute your tests as usual, and they will run in parallel on the Selenium Grid.

**ThreadLocal with TestNG: Using ThreadLocal to Manage WebDriver Instances in Parallel Tests**

**Step-by-Step Instructions:**

1. **Define ThreadLocal WebDriver:** Use ThreadLocal to manage WebDriver instances:

public class BaseTest {

private static ThreadLocal<WebDriver> driver = new ThreadLocal<>();

public WebDriver getDriver() {

return driver.get();

}

@BeforeMethod

public void setUp() {

WebDriver webDriver = new ChromeDriver();

driver.set(webDriver);

}

@AfterMethod

public void tearDown() {

getDriver().quit();

driver.remove();

}

}

1. **Use WebDriver in Tests:** Inherit from BaseTest and use getDriver() to access the WebDriver instance:

public class YourTestClass extends BaseTest {

@Test

public void testMethod() {

getDriver().get("http://example.com");

// Test code

}

}

**5. Report on Installing and Using Allure and Extent Report with TestNG and Listeners**

**2. Installing Allure Report**

**2.1 Prerequisites**

* Java Development Kit (JDK)
* Maven
* TestNG

**2.2 Installing Allure Command-Line Tool**

1. **Install Scoop (Windows) or Homebrew (macOS/Linux):**
   * For Windows: Open PowerShell as Administrator and execute:

iwr -useb get.scoop.sh | iex

scoop install allure

* + For macOS/Linux: Open Terminal and execute:

brew install allure

1. **Verify Installation:**

allure --version

**2.3 Maven Dependency**

Add the following dependencies to your pom.xml file:

<dependencies>

<!-- Allure TestNG Integration -->

<dependency>

<groupId>io.qameta.allure</groupId>

<artifactId>allure-testng</artifactId>

<version>2.13.8</version>

</dependency>

</dependencies>

**3. Installing Extent Report**

**3.1 Maven Dependency**

Add the following dependencies to your pom.xml file:

<dependencies>

<!-- Extent Reports -->

<dependency>

<groupId>com.aventstack</groupId>

<artifactId>extentreports</artifactId>

<version>5.0.9</version>

</dependency>

<!-- Selenium and TestNG dependencies -->

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>3.141.59</version>

</dependency>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>7.4.0</version>

</dependency>

</dependencies>

**4. Setting Up Allure Report**

**4.1 Allure Configuration**

Create an Allure listener class to integrate Allure with TestNG:

import io.qameta.allure.testng.AllureTestNg;

import org.testng.annotations.Listeners;

@Listeners({AllureTestNg.class})

public class AllureTest {

@Test

public void testMethod() {

// Your test code

}

}

**4.2 Running Tests and Generating Allure Reports**

1. Run your tests using TestNG.
2. Generate Allure reports by executing:

allure serve allure-results

**5. Setting Up Extent Report**

**5.1 Extent Report Configuration**

Create an ExtentReportListener class to integrate Extent Report with TestNG:

import com.aventstack.extentreports.ExtentReports;

import com.aventstack.extentreports.ExtentTest;

import com.aventstack.extentreports.reporter.ExtentHtmlReporter;

import com.aventstack.extentreports.reporter.configuration.Theme;

import org.testng.\*;

public class ExtentReportListener implements ITestListener {

private static ExtentReports extent;

private static ExtentHtmlReporter htmlReporter;

private static ExtentTest test;

@Override

public void onStart(ITestContext context) {

htmlReporter = new ExtentHtmlReporter("extent-report.html");

htmlReporter.config().setTheme(Theme.STANDARD);

htmlReporter.config().setDocumentTitle("Extent Report");

htmlReporter.config().setReportName("Test Execution Report");

extent = new ExtentReports();

extent.attachReporter(htmlReporter);

extent.setSystemInfo("OS", "Windows 10");

extent.setSystemInfo("Browser", "Chrome");

}

@Override

public void onTestStart(ITestResult result) {

test = extent.createTest(result.getMethod().getMethodName());

}

@Override

public void onTestSuccess(ITestResult result) {

test.pass("Test passed");

}

@Override

public void onTestFailure(ITestResult result) {

test.fail(result.getThrowable());

}

@Override

public void onTestSkipped(ITestResult result) {

test.skip("Test skipped");

}

@Override

public void onFinish(ITestContext context) {

extent.flush();

}

}

**5.2 Using Extent Report Listener**

Annotate your test class with the ExtentReportListener:

import org.testng.annotations.Listeners;

import org.testng.annotations.Test;

@Listeners(ExtentReportListener.class)

public class ExtentReportTest {

@Test

public void testMethod() {

// Your test code

}

}

**6. Running Tests and Generating Reports**

**6.1 Running Tests**

Execute your test suite using TestNG. The reports will be generated automatically.

**6.2 Viewing Reports**

* **Allure Report:** Run the command allure serve allure-results to view the Allure report in your browser.
* **Extent Report:** Open the extent-report.html file generated in your project directory to view the Extent report.