2022

Java Selenium and Apache JMeter Test Automation

Contents

Pre-requisites

Java Programming Setup
Java Programming Crash Course

Introduction to Selenium

a. Selenium is a test automation framework for web application testing.
b. Selenium can interact with the browser and the elements inside the browser.
c. Selenium can be used by multiple programming languages like Java, Python and Ruby, but selenium web drivers are downloaded specific to each language (i.e. Selenium Web Driver for Java).
d. Selenium remains popular because:
    i. Largest User Base group support
    ii. Open Source
    iii. De-facto for web automation testing
    iv. Multi-Browser compatibility

Setup and Verify Java, Selenium and Chrome

- Check java

\> java --version
\> where java

- We can use java 1.8 (java 8) or the latest as of date java 16 (as of June 2021)

Java 8 https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html
Java 16 https://www.oracle.com/java/technologies/javase-jdk16-downloads.html

Java Version history https://en.wikipedia.org/wiki/Java_version_history

- Download and Install Eclipse

Eclipse IDE (version as of writing: 2021-6) https://www.eclipse.org/downloads/

- Download and install Google Chrome
- Selenium Web Driver

Selenium client and webdriver https://www.selenium.dev/downloads/

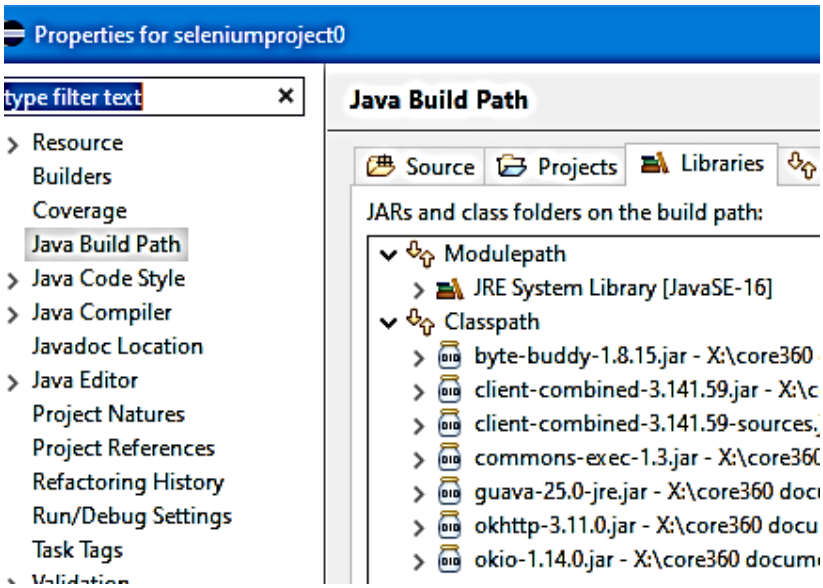
Download Selenium for Java 3.14, extract it to a folder in C:\seleniumjava314

- We will setup selenium in an eclipse workspace.

i. Launch eclipse → choose/create a workspace folder (ex. “TestAutomation”) → create a new java project (ex. “project1”).
ii. Rclick on src → new package (ex. “selenium”)
iii. Rclick project (“project1”) → properties → java build path → libraries tab → classpath → add external jar → select the 2 jar files:

client-combined.jar
client-combined-sources.jar

and all the other jar files in the libs folder of the extracted selenium download like as follows:



Writing first test

- Verify Selenium installation and writing our first test.
    a. Rclick your created package in eclipse (i.e. “selenium”) → add new public class (ex. “TestSelenium”)
    b. Code:

TestSelenium.class

```
package selenium;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class TestSelenium {

	public static void main(String[] args) {
		WebDriver driver = new ChromeDriver();
		driver.get("https://www.google.com");
	}

}
```

    c. Rclick→run java application, and see error regarding not specifying the path to the web driver.

To fix this, we need to download chrome driver in the selenium web site under Third Party, Bindings and other Plugins:

ChromeDriver (version 92 as of writing):
https://sites.google.com/a/chromium.org/chromedriver/

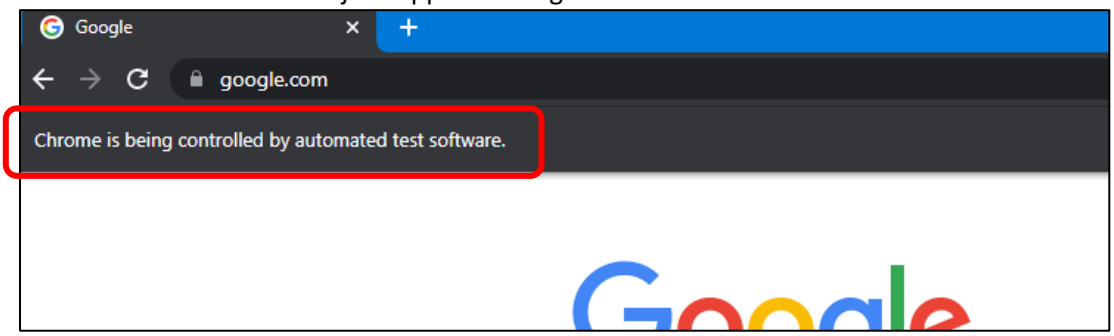
Note: Take some time to check and read the docs for the chrome driver since oftentimes, there is a specific chrome driver for chrome versions.

After downloading, extract the chrome driver into a folder location and get the path to its executable like:
C:\chromedriver\chromedriver.exe

d. Then modify your code:

```
package selenium;

import org.openqa.selenium.*;
import org.openqa.selenium.chrome.*;

public class TestSelenium {

	public static void main(String[] args) {
		System.setProperty("webdriver.chrome.driver", "C:\\chromedriver\\chromedriver.exe");
		WebDriver driver = new ChromeDriver();
		driver.get("https://www.google.com");
		driver.quit();
	}

}
```

e. Then run the java application again.



f. Try exploring more by adding the following lines:

```
package selenium;

import org.openqa.selenium.*;
import org.openqa.selenium.chrome.*;

public class TestSelenium {

	public static void main(String[] args) {
		System.setProperty("webdriver.chrome.driver", "C:\\chromedriver\\chromedriver.exe");
		WebDriver driver = new ChromeDriver();
		driver.get("https://www.google.com");

		try {
```

```
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }  // Let the user actually see something!

        WebElement searchBox = driver.findElement(By.name("q"));
        searchBox.sendKeys("ChromeDriver");
        searchBox.submit();

        try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }  // Let the user actually see something!

        driver.quit();
    }

}
```

Using Gecko Driver

- Try Selenium with Firefox using GeckoDriver

Get gecko driver here:
https://github.com/mozilla/geckodriver/releases

extract to a suitable location and use on the java project as follows:

```
package selenium;

import org.openqa.selenium.*;
import org.openqa.selenium.firefox.*;;

public class TestSelenium {

    public static void main(String[] args) {
        System.setProperty("webdriver.chrome.driver", "C:\\geckodriver\\geckodriver.exe");
        WebDriver driver = new FirefoxDriver();
        driver.get("https://www.google.com");

        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }  // Let the user actually see something!
```

```
		WebElement searchBox = driver.findElement(By.name("q"));
	searchBox.sendKeys("ChromeDriver");
	searchBox.submit();

	try {
			Thread.sleep(5000);
		} catch (InterruptedException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}  // Let the user actually see something!

	//driver.quit();
	}

}
```

Introduction to web drivers

- Introduction to Web Driver
    a. Web drivers are browser representations
    b. Most browsers have their own web drivers that can be use like an api for development and testing purposes.
    c. Selenium Web Driver makes direct calls to the browser thru each browser's native support for automation.
    d. How direct calls from Selenium to the browsers are called varies depending on the browser used.

- To use the web driver, we instantiate a web driver object from the browser's web driver class. Then we can access the methods and properties of that browser.

```
package selenium;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class test2 {

	public static void main(String[] args) {

		System.setProperty("webdriver.chrome.driver", "C:\\chromedriver\\chromedriver.exe");
		WebDriver driver = new ChromeDriver();
		driver.get("https://www.google.com");

		try {
			Thread.sleep(5000);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}  // Let the user actually see something!
```

```
            WebElement searchBox = driver.findElement(By.name("q"));
        searchBox.sendKeys("ChromeDriver");
        searchBox.submit();



        try {
                    Thread.sleep(5000);
            } catch (InterruptedException e) {
                    e.printStackTrace();
            }  // Let the user actually see something!


            String title = driver.getTitle();
            String url = driver.getCurrentUrl();
            System.out.println("title:" + title + "\n" + "URL:" + url);


    }

}
```

- Other driver methods here:
https://www.selenium.dev/selenium/docs/api/java/org/openqa/selenium/WebDriver.html

Java Note: to auto fix indention and spacing, use ctrl+shift+f

Introduction to TestNG

a. TestNG is a testing framework for the Java Programming Language.
b. The design goal of TestNG is to cover a wider range of test categories: unit, functional, end-to-end, integration, etc. . With more power and ease of use than Junit/NUnit.
c. What is smoke testing?

A smoke test is a quick run through of a site; it focuses on critical functionality to ensure the site can perform basic features. The primary features are often called red routes in the software industry.

It only takes a couple of minutes to complete, up to ten minutes at most. What is great about smoke tests is you can perform them either daily or every other day.

'Smoke testing' came to software testing from a similar hardware test -where the device passed if it did not catch fire (or smoked) the first time it was turned on.

For software purposes, an example of smoke testing could be for a hotel reservation site. In this smoke test example, the tester would ensure the user will be able to sign up, change your password, create a booking, and be notified.

d. What is regression testing?

A regression test is an in-depth, thorough examination of a site. It tests all of the complex user stories and detailed nuances of the site, therefore; they may take many hours to complete.

Performing a regression test ensures any changes made did not negatively impact any of the functionality of the site.

A regression test will cover every feature, new and old, along with bug fix checks to make sure bugs did not reappear in the software.

e. If you have multiple tests (a combination of smoke and regression tests), TestNG can create groups of tests by using annotations:

```
@Test(groups={"regression"})
@Test(groups={"smoke"})
```

f. TestNG also has "Test Lifecycle Management"

Open browser→run tests→close browser

TestNG can use annotations like
@Before
@After
@Test
Etc…

- Setup and using TestNG
  a. Adding TestNG in Eclipse
     i. Install via Eclipse Marketplace

  Help→eclipse marketplace (search for "testing for eclipse by cedric beust")
  https://marketplace.eclipse.org/content/testng-eclipse

     ii. Eclipse will restart after installation. To verify, you can open the eclipse marketplace again and check the installed tab.
     iii. You can also try creating a new TestNG Class

  Rclick package→new→other→testNG→testNG class (ex. "MyTestNGClass")

  Note: Can you see the TestNG annotations when creating the TestNG Class?

     iv. The testNG annotations can be a replacement for the main method.

WebElements

1. WebElements
   a. Most used API for Selenium
   b. Gets a specific element on the web page
   c. Using webelement, we can create references to web form controls on the page (ex, textboxes and buttons)
2. Create a simple html web project for the webelement demonstration

Find Element

It is used to find an element and returns a first matching single WebElement reference, that can be used for future element actions

```
WebDriver driver = new FirefoxDriver();
driver.get("http://www.google.com");

// Get search box element from webElement 'q' using Find Element
WebElement searchBox = driver.findElement(By.name("q"));
searchBox.sendKeys("webdriver");
```

Find Elements

Similar to 'Find Element', but returns a list of matching WebElements. To use a particular WebElement from the list, you need to loop over the list of elements to perform action on selected element.

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import java.util.List;

public class findElementsExample {
   public static void main(String[] args) {
      WebDriver driver = new FirefoxDriver();
      try {
         driver.get("https://example.com");
         // Get all the elements available with tag name 'p'
         List<WebElement> elements = driver.findElements(By.tagName("p"));
         for (WebElement element : elements) {
            System.out.println("Paragraph text:" + element.getText());
         }
      } finally {
         driver.quit();
      }
   }
}
```

Find Element from Element

It is used to find a child element within the context of parent element. To achieve this, the parent WebElement is chained with 'findElement' to access child elements

```
WebDriver driver = new FirefoxDriver();
driver.get("http://www.google.com");
WebElement searchForm = driver.findElement(By.tagName("form"));
WebElement searchBox = searchForm.findElement(By.name("q"));
searchBox.sendKeys("webdriver");
```

Find Elements from Element

It is used to find the list of matching child WebElements within the context of parent element. To achieve this, the parent WebElement is chained with 'findElements' to access child elements

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import java.util.List;

public class findElementsFromElement {
   public static void main(String[] args) {
      WebDriver driver = new ChromeDriver();
      try {
         driver.get("https://example.com");

         // Get element with tag name 'div'
         WebElement element = driver.findElement(By.tagName("div"));

         // Get all the elements available with tag name 'p'
         List<WebElement> elements = element.findElements(By.tagName("p"));
         for (WebElement e : elements) {
            System.out.println(e.getText());
         }
      } finally {
         driver.quit();
      }
   }
}
```

Is Element Enabled

This method is used to check if the connected Element is enabled or disabled on a webpage. Returns a boolean value, True if the connected element is enabled in the current browsing context else returns false.

```
//navigates to url
driver.get("https://www.google.com/");

//returns true if element is enabled else returns false
boolean value = driver.findElement(By.name("btnK")).isEnabled();
```

WebElement Methods

Demonstrate other webelement methods:

click()	Click this element.
getAttribute(java.lang.String name)	Get the value of the given attribute of the element.
getCssValue(java.lang.String propertyName)	Get the value of a given CSS property.
getDomAttribute(java.lang.String name)	Get the value of the given attribute of the element.
getDomProperty(java.lang.String name)	Get the value of the given property of the element.
getSize()	What is the width and height of the rendered element?
getText()	Get the visible text

isDisplayed()	Is this element displayed or not?
isSelected()	Determine whether or not this element is selected or not.
isEnabled()	sIs the element currently enabled or not?
sendKeys()	Use this method to simulate typing into an element.
submit()	If this current element is a form, or an element within a form, then this will be submitted to the remote server.

Quick example:

```
@Test
public void f() throws Exception{

        cdriver.get("https://www.youtube.com");

        Thread.sleep(5000);
        WebElement searchBox = cdriver.findElement(By.id("search"));
        searchBox.sendKeys("bts");

        WebElement searchbtn = cdriver.findElement(By.id("search-icon-legacy"));
        searchbtn.click();

        Thread.sleep(5000);

}
```

Resource Locators

1. Resource locators
    a. Used to point to a specific resource
    b. Commonly used resource locators: id, name, xpath, css, etc
    c. On scenarios where id or name may be inconsistent or not available, we may use xpath
    d. To find xpath, you may use chrome developer tool or firebug

XPath (XML Path)
Syntax for finding elements in a web page.

Example:
input[@title=”search” and @type=”text”]

Using Chrome Developer Tool to find xpath

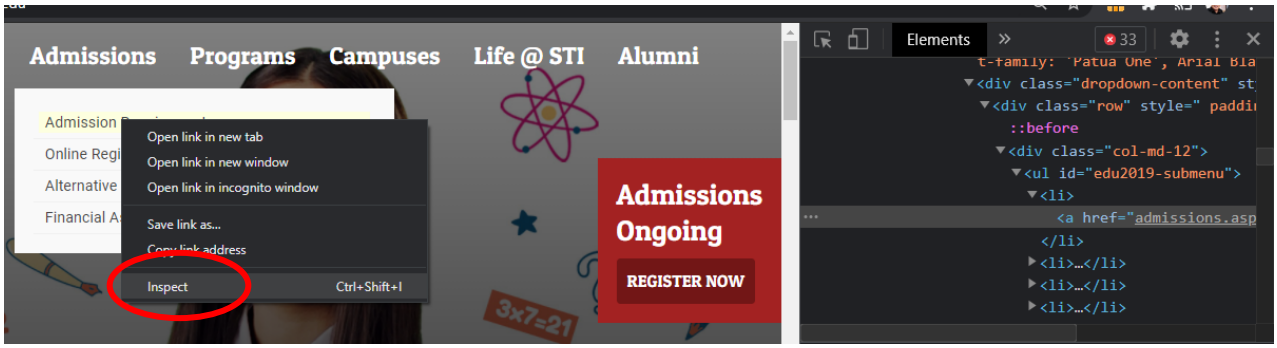
a. Launch the web page in chrome
b. Click on chrome options (the ellipsis […] on upper-right)→more tools→developer tools
c. Now hover mouse over the web element→rclick inspect to inspect the dom and css attributes of the web element.
d. To check and find for the existence (or for duplicates), developer tool→console→type the following:

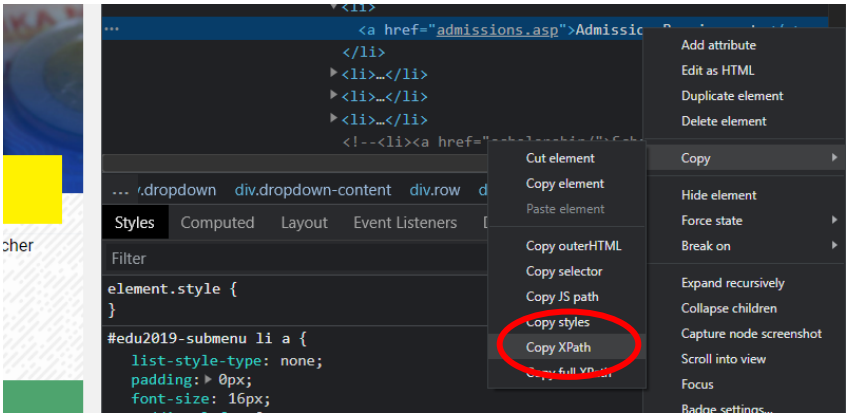
    > document.getElementById(<id here>);
    > document.getElementsByName(<name here>);

Get all hyperlink href in a page
> var links = document.links;

for(var i=0;i<=links.length;i++){
console.log(links[i].href + "\n");
}

e. To get the xpath, inpect element, rclick on the code in the elements tab in the chrome developer tool→copy xpath





Sample xpath:
//*[@id="edu2019-submenu"]/li[1]/a

Sample usage:

```
@Test
public void testByXpath() {
        driver.get("https://testautomation.co/webelements/");
        driver.findElement(By.xpath("//*[@id="edu2019-submenu"]/li[1]/a")).click();
}
```

Waits

1. Waits
    a. Using waits in Test Automation
        i. Waits hold automated task (code) for a certain amount of time before processing to the next step.
        ii. These are important since sometimes you will not be able to access webelements if they are not loaded into the page yet. Timing may cause errors like “NoSuchElementException” or “Element not found”.
        iii. Some pages load faster than others. Some UI elements also takes longer to load and appear on page.

        iv. Lazy loading is common in most JS frameworks where UI elements may load at different times after the page renders fully.

            Sample (delayed explicit button) :
            https://testautomation.co/webelements/

    b. Implicit waits
        i. Implicit waits are created in the webdriver and is usable throughout the life cycle of the webdriver.

```
driver.manage().timeouts().implicitlyWait(TimeOut,TimeUnit.SECONDS);

//for example
driver.manage().timeouts().implicitlyWait(10,TimeUnit.SECONDS);
```

    c. Explicit waits
        i. Waits not only by time, but by expected conditions.
        ii. Conditions like, wait until a button is clickable or if a component is displayed, before doing the next step.

```
WebDriverWait wait = new WebDriverWait(driver, 10);
WebElement delayedBtn = wait.until(ExpectedConditions.elementToBeClickable(By.id(“btn1”)));
```

    d. Fluent waits
        i. Enhanced explicit wait
        ii. How often to check expected condition(s)?

2. Activities

Implementing Implicit wait

```
public class ImplicitTestNG {
        WebDriver driver = null;

        @Test
        public void testImplicit() {
                driver.get("https://testautomation.co/webelements/");
                driver.findElement(By.id("link2courses1")).click();

        }
```

```
	@BeforeMethod
	public void beforeMethod() {
		System.setProperty("webdriver.chrome.driver", "C:\\chrome\\...\\chromedriver.exe");
		driver = new ChromeDriver();
		driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

	}



	@AfterMethod
	public void afterMethod() {
		// driver.quit();
	}

}
```

Implementing Explicit wait

```
public class ExplicitTestNG {

	WebDriver driver = null;

	@BeforeMethod
	public void beforeMethod() {
		System.setProperty("webdriver.chrome.driver", "C:\\chrome\\...\\chromedriver.exe");
		driver = new ChromeDriver();
	}

	@Test
	public void testExplicit() {
		driver.get("https://testautomation.co/webelements/");
		WebDriverWait wait = new WebDriverWait(driver,10);
		wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("explibutton")));
		driver.findElement(By.id("explibutton")).click();

	}

	@AfterMethod
	public void afterMethod() {
		// driver.quit();

	}

}
```

Action Class

Setup
Before using the Actions class, ensure you have Selenium set up in your Java project. This typically involves adding Selenium WebDriver dependencies to your project's build file (e.g., Maven or Gradle).

Importing Necessary Classes
To use the Actions class, you need to import it, along with other necessary Selenium classes:

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.interactions.Actions;
```

Initializing WebDriver and Actions
First, create a WebDriver instance, and then instantiate the Actions class with this driver.

```
WebDriver driver = new ChromeDriver(); // Make sure to set the system property for your driver
driver.get("http://example.com"); // Replace with your target URL
Actions actions = new Actions(driver);
```

Performing Actions
The Actions class provides a builder pattern to queue up a series of actions, which are not performed until you call .perform() at the end.

Moving to an Element

```
WebElement targetElement = driver.findElement(By.id("targetElementId"));
actions.moveToElement(targetElement).perform();
```

Clicking

```
actions.click().perform(); // Clicks at the current mouse location
actions.click(targetElement).perform(); // Moves to the element and clicks
```

Right-Click (Context Click)

```
actions.contextClick(targetElement).perform();
```

Drag and Drop

```
WebElement source = driver.findElement(By.id("sourceId"));
WebElement target = driver.findElement(By.id("targetId"));

actions.dragAndDrop(source, target).perform();
```

Typing

```
actions.sendKeys(targetElement, "Text to enter").perform(); // Sends keys to the element
actions.sendKeys("Text").perform(); // Sends keys to the current focused element
```

Complex Gestures

You can chain multiple actions together:

```
actions.moveToElement(element)
    .click()
    .keyDown(Keys.SHIFT)
    .sendKeys("hello")
    .keyUp(Keys.SHIFT)
    .doubleClick()
    .contextClick()
    .perform();
```

Closing the Driver

Don't forget to close your WebDriver session when you're done:

```
driver.quit();
```

Assertions

1. Assertions
    a. Assertions are ways to check that you are getting the expected values back.
    b. They are means for users to validate tests.
    c. Assertions results are based on the comparison of actual and expected results. A test passes only when the assertions show no exception.
    d. If you use assertions on your tests, the test(s) will be aborted if the assert fails. Next test case will start executing if you are running a test suite.
    e. Two types of assertions:
        i. Hard Assertion

            Default assert mechanism built into TestNG (org.testng.assert) package. We use it when a test has to stop immediately after the assertion fails.

        ii. Soft Assertion

            This is a custom assert mechanism supported by TestNG (org.testng.asserts.Softassert) package. We use it when a test has to continue execution even after an assertion fails in the sequence.

    f. Commonly used assertion methods:

```
//assertEquals(expected,actual)
Assert.assertEquals("Google",driver.getTitle());

//assertNotEquals(expected,actual)
Assert.assertNotEquals("",driver.getTitle());

//assertion passes when the Boolean expression returns true
Assert.assertTrue(driver.getTitle().length > 0);

//assertion passes when the Boolean expression returns true
Assert.assertFalse(driver.getTitle().length == 0);
```

```
		//test passes if the object tested is null
		Assert.assertNull(objectname);

		//test passes if the object tested is not null
		Assert.assertNotNull(objectname);
```

Sample code:

```
public class HardAssertionExample {
	WebDriver driver;

	@Test
	public void testHardAssertion() {

		driver.get("https://www.google.com");
		String titleOfGoogleHomePage = driver.getTitle();

String expectedTitleOfGoogleHomePate = "Google";
String searchTerm = null;
//searchTerm = "TestAutomation.co";
Assert.assertEquals(titleOfGoogleHomePage, expectedTitleOfGoogleHomePate);

		WebElement searchBox = driver.findElement(By.xpath("//input[@title='Search' and @type='text']"));
		Assert.assertNotNull(searchTerm);
		searchBox.sendKeys(searchTerm);
		WebElement submitSearchBtn = driver.findElement(By.name("btnK"));
		submitSearchBtn.submit();

	}

	@BeforeMethod
	public void beforeMethod() {
		System.setProperty("webdriver.chrome.driver", "C:\\...\\chromedriver.exe");
		driver = new ChromeDriver();

	}

	@AfterMethod
	public void afterMethod() {
		// driver.quit();
	}

}
```

Activity for Assertion:

a. Form submission
b. Links and navigation checking

Single-class-multiple-test case scenario

1. Demonstrate single-class-multiple-test case scenario.

   Note(s):
   a. TestNG executes multiple tests in a class in order of their function names (alphabetically).
   b. You can override this test sequence by using priority attribute in the test annotation. Lowest priority executes first.

      Example:

```
@Test(priority=1)
public void bTest(){ … }

@Test(priority=2)
public void aTest(){ … }
```

   c. Using @Test(timeOut=milliseconds). This is the test attribute that sets the maximum number of milliseconds the test should take.
   d. Use comma (“,”) to separate attributes of the @Test annotation.

      Ex.
      @Test(priority=1,timeOut=5000,description=”form submission test”)

   e. @Test *groups* and *dependsonGroups* attributes

      Ex.

```
package TestNG;

import org.testng.annotations.Test;

public class Group {

    @Test(groups="Smoke")
    public void Test2(){
            System.out.println("Smoke group method");
    }

    @Test(groups="Regression")
    public void Test3(){
            System.out.println("Regression group method");
    }

    @Test(dependsOnGroups = {"Smoke","Regression"})
    public void DependsOnGroups(){
            System.out.println("Execution of dependsOnGroups");
    }
}
```

```
Smoke group method
Regression group method
Execution of dependsOnGroups
PASSED: Test2
PASSED: Test3
PASSED: DependsOnGroups

===============================================
    Default test
    Tests run: 3, Failures: 0, Skips: 0
===============================================


===============================================
Default suite
Total tests run: 3, Failures: 0, Skips: 0
===============================================
```

In the above result, we can see the execution of the DependsOnGroups method depends on the list of the groups listed. If any group doesn't execute then DependsOnGroups method will also not execute.

f. @Test *dependsOnMethods* attribute

Ex.

```
package TestNG;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Test;

public class MethodDependency {
	@Test
	public void test1() {
	  System.out.println("test1()");
	}

	@Test
	public void test2() {
	  System.out.println("test2()");
	}

	@Test(dependsOnMethods={"test1", "test2"})
	public void DependsOnMethod() {
	  System.out.println("DependsOnMethod()");
	}
}
```

```
test1()
test2()
DependsOnMethod()
PASSED: test1
PASSED: test2
PASSED: DependsOnMethod

===============================================
    Default test
    Tests run: 3, Failures: 0, Skips: 0
===============================================


===============================================
Default suite
Total tests run: 3, Failures: 0, Skips: 0
===============================================
```

In the above result, we can see the dependency of the method DependsOnMethod depends on the list of methods. If any method fails then DependsOnMethod will not execute.

Check if a webelement exists

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class WebElementExistenceTestWithTestNG {
   private WebDriver driver;

   @BeforeTest
   public void setUp() {
      System.setProperty("webdriver.chrome.driver", "path/to/chromedriver.exe");
      driver = new ChromeDriver();
      driver.get("https://example.com");
   }

   @Test
   public void testElementExistence() {
      By locator = By.id("elementId");
      List<WebElement> elements = driver.findElements(locator);
      Assert.assertFalse(elements.isEmpty(), "Element does not exist on the page");
   }

   @AfterTest
   public void tearDown() {
      if (driver != null) {
```

```
        driver.quit();
    }
  }
}
```

Test links and navigation

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class LinkNavigationTest {
  private WebDriver driver;

  @BeforeTest
  public void setUp() {
    System.setProperty("webdriver.chrome.driver", "path/to/chromedriver.exe");
    driver = new ChromeDriver();
    driver.get("https://example.com");
  }

  @Test
  public void testLinkNavigation() {
    // Find and retrieve all links on the page
    List<WebElement> links = driver.findElements(By.tagName("a"));

    // Iterate through the links
    for (WebElement link : links) {
      String urlToNavigate = link.getAttribute("href");

      // Click on the link to navigate
      link.click();

      // Get the current URL after navigation
      String currentUrl = driver.getCurrentUrl();

      // Define the expected URL you want to verify
      String expectedUrl = "https://example.com/some_page"; // Replace with the expected URL

      // Use TestNG assertions to check if the navigation is correct
      Assert.assertEquals(currentUrl, expectedUrl, "Link navigation is incorrect for URL: " + urlToNavigate);

      // You can navigate back to the original page if needed
      driver.navigate().back();
```

```
    }
  }

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

Login Test

```
@Test
public void testLogin() {
  driver.findElement(By.id("username")).sendKeys("yourUsername");
  driver.findElement(By.id("password")).sendKeys("yourPassword");
  driver.findElement(By.id("loginButton")).click();

  // Verify that the user is logged in
  WebElement welcomeMessage = driver.findElement(By.id("welcomeMessage"));
  Assert.assertTrue(welcomeMessage.isDisplayed(), "Login was not successful.");
}
```

Search Functionality Test

```
@Test
public void testSearchFunctionality() {
  driver.findElement(By.id("searchBox")).sendKeys("searchQuery");
  driver.findElement(By.id("searchButton")).click();

  // Verify search results
  List<WebElement> searchResults = driver.findElements(By.className("searchResult"));
  Assert.assertFalse(searchResults.isEmpty(), "No search results found.");
}
```

Form Submission Test

```
@Test
public void testFormSubmission() {
  driver.findElement(By.id("name")).sendKeys("John Doe");
  driver.findElement(By.id("email")).sendKeys("john@example.com");
  driver.findElement(By.id("submitButton")).click();

  // Verify form submission success
  WebElement confirmationMessage = driver.findElement(By.id("confirmationMessage"));
  Assert.assertTrue(confirmationMessage.isDisplayed(), "Form submission was not successful.");
}
```

Handling Alerts and Pop-ups

```
@Test
public void testAlertHandling() {
   driver.findElement(By.id("triggerAlertButton")).click();

   Alert alert = driver.switchTo().alert();
   String alertText = alert.getText();
   alert.accept();

   // Verify alert message
   Assert.assertEquals(alertText, "This is an alert message.", "Alert message doesn't match the expected value.");

   // Continue with further testing after handling the alert
}
```

Test if images appear

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class ImageTest {
   private WebDriver driver;

   @BeforeTest
   public void setUp() {
      System.setProperty("webdriver.chrome.driver", "path/to/chromedriver.exe");
      driver = new ChromeDriver();
      driver.get("https://example.com");
   }

   @Test
   public void testImageValidation() {
      // Find all image elements on the page
      List<WebElement> images = driver.findElements(By.tagName("img"));

      for (WebElement image : images) {
         // Get image attributes
         String src = image.getAttribute("src");
         String alt = image.getAttribute("alt");
         int width = Integer.parseInt(image.getAttribute("width"));
         int height = Integer.parseInt(image.getAttribute("height"));

         // Verify image attributes
```

```
        Assert.assertFalse(src.isEmpty(), "Image URL is empty.");
        Assert.assertFalse(alt.isEmpty(), "Image alt text is empty.");
        Assert.assertTrue(width > 0, "Image width is not greater than zero.");
        Assert.assertTrue(height > 0, "Image height is not greater than zero.");

        // Check image visibility (you may need to implement a method to check visibility)
        // Assert.assertTrue(isImageVisible(image), "Image is not visible.");
      }
    }

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

Test loading time of pages

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class PageLoadTimeTest {
    private WebDriver driver;
    private long startTime;
    private long endTime;
    private long pageLoadTimeThreshold = 5000; // Set the threshold in milliseconds

    @BeforeTest
    public void setUp() {
      System.setProperty("webdriver.chrome.driver", "path/to/chromedriver.exe");
      driver = new ChromeDriver();
      driver.manage().timeouts().pageLoadTimeout(30, java.util.concurrent.TimeUnit.SECONDS);
    }

    @Test
    public void testPageLoadTime() {
      startTime = System.currentTimeMillis();
      driver.get("https://example.com");
      endTime = System.currentTimeMillis();

      long pageLoadTime = endTime - startTime;

      // Use TestNG assertions to check if the page load time is within the threshold
```

```
        Assert.assertTrue(pageLoadTime < pageLoadTimeThreshold, "Page load time exceeds the threshold.");
    }

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

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class PageLoadPerformanceTest {
    private WebDriver driver;
    private String url = "https://example.com";

    @BeforeTest
    public void setUp() {
        System.setProperty("webdriver.chrome.driver", "path/to/chromedriver.exe");
        driver = new ChromeDriver();
    }

    @Test
    public void measurePageLoadTime() {
        long startTime = System.currentTimeMillis();
        driver.get(url);
        long endTime = System.currentTimeMillis();

        long pageLoadTime = endTime - startTime;
        System.out.println("Page load time: " + pageLoadTime + " ms");
    }

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

Checking database impact with selenium front-end changes

Step 1: Add JDBC Driver to Your Project

First, you need to add the JDBC driver for your specific database to your project's dependencies. If you're using Maven, add the dependency for your database driver in the pom.xml. For example, for MySQL:

```
<dependency>
   <groupId>mysql</groupId>
   <artifactId>mysql-connector-java</artifactId>
   <version>8.0.23</version>
</dependency>
```

Step 2: Establish a Database Connection

To connect to your database, use the DriverManager.getConnection() method with your database's URL, username, and password.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DatabaseTest {
   private static final String DATABASE_URL = "jdbc:mysql://localhost:3306/yourDatabase";
   private static final String USERNAME = "yourUsername";
   private static final String PASSWORD = "yourPassword";

   public Connection connectToDatabase() {
      Connection connection = null;
      try {
         // Load the JDBC driver
         Class.forName("com.mysql.cj.jdbc.Driver");
         // Establish the connection
         connection = DriverManager.getConnection(DATABASE_URL, USERNAME, PASSWORD);
      } catch (ClassNotFoundException | SQLException e) {
         e.printStackTrace();
      }
      return connection;
   }
}
```

Step 3: Execute a Query

Once connected, you can execute SQL queries to interact with your database. This can be useful for setting up test data or verifying that your web application is correctly modifying the database.

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public void executeQuery(Connection connection) {
   try {
      Statement statement = connection.createStatement();
      ResultSet resultSet = statement.executeQuery("SELECT * FROM yourTable");
```

```
        while (resultSet.next()) {
            // Process the result set
            String data = resultSet.getString("columnName");
            System.out.println(data);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Step 4: Integrate with Selenium

After setting up your database connection and query execution, you can integrate these with your Selenium tests. For instance, after performing an action with Selenium that should update the database, you can run a query to verify the changes.

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class SeleniumDatabaseTest {
    public static void main(String[] args) {
        WebDriver driver = new ChromeDriver();
        driver.get("http://example.com");

        // Your Selenium test actions here

        DatabaseTest dbTest = new DatabaseTest();
        Connection connection = dbTest.connectToDatabase();

        // Verify database state
        dbTest.executeQuery(connection);

        driver.quit();
    }
}
```

Closing the Connection

Always ensure you close the database connection once your operations are complete to free up resources.

```
if (connection != null) {
    try {
        connection.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

This approach allows you to validate the backend state of your application in coordination with front-end tests performed by Selenium, giving you a more comprehensive test coverage.

Using Apache JMeter for Database, API and App Testing

Lab Exercise and Demonstration: Apache JMeter- Basic Scenario

a. Create a test plan
b. Create a thread group
   - Threads (users) 10
   - Ramp up (60 secs)
   - Loop count 1

c. Add a http cache manager to the test plan
   Rclick test plan → add → config element → http cache manager
   - Clear cache each iteration

d. Add http cookie manager to the test plan
   Rclick test plan → add → config element → http cookie manager
   - Clear cookie each iteration

e. Add http request defaults to the test plan
   Rclick test plan → add → config element → http request defaults

   [Basic Tab]
   - Protocol: http
   - Server name / ip: www.blazedemo.com
   - Port: 80

   [Advanced Tab]
   - Retrieve all embedded resources: Enabled
   - Parallel download: Enabled (2)

f. Add view results tree to the test plan
   Rclick test plan → add → listener → view results tree

g. Add http request to the thread group
   Rclick thread group → add → sampler → http request
   - Leave as is as it will use the http request default config element
   - Rename the http request as needed

h. Add constant timer to the http request sampler
   Rclick http request sampler → add → timer → constant timer
   - Thread delay: 10000

i. Add **another** http request to the thread group
   Rclick thread group → add → sampler → http request
   - Rename the http request as needed
   - Servername/ip: www.blazedemo.com
   - Path: /reserve.php

j. Add constant timer to the http request sampler (on item "i")
Rclick http request sampler → add → timer → constant timer
- Thread delay: 3000

k. Go to view results tree and run the test. Note that each thread will run two http request.
l. Try to interpret the result
- In the view results tree, click on the first result.
  i. The sampler result shows some request-response stats like response time
  ii. Request tab shows request data information
  iii. Response data shows page code

m. Add aggregate report
Rclick test plan → add → listener → aggregate report

We would want to see the average response time

Lab Exercise and Demonstration: Utilizing Assertions in Apache JMeter

a. Create a test plan
b. Install JSON plugin
Options → Plugins Manager → Available plugins → JSON Plugin → Restart JMeter

c. Create a thread group
[use defaults in this demo]
  a. Threads (users) 1
  b. Ramp up (1 sec)
  c. Loop count 1

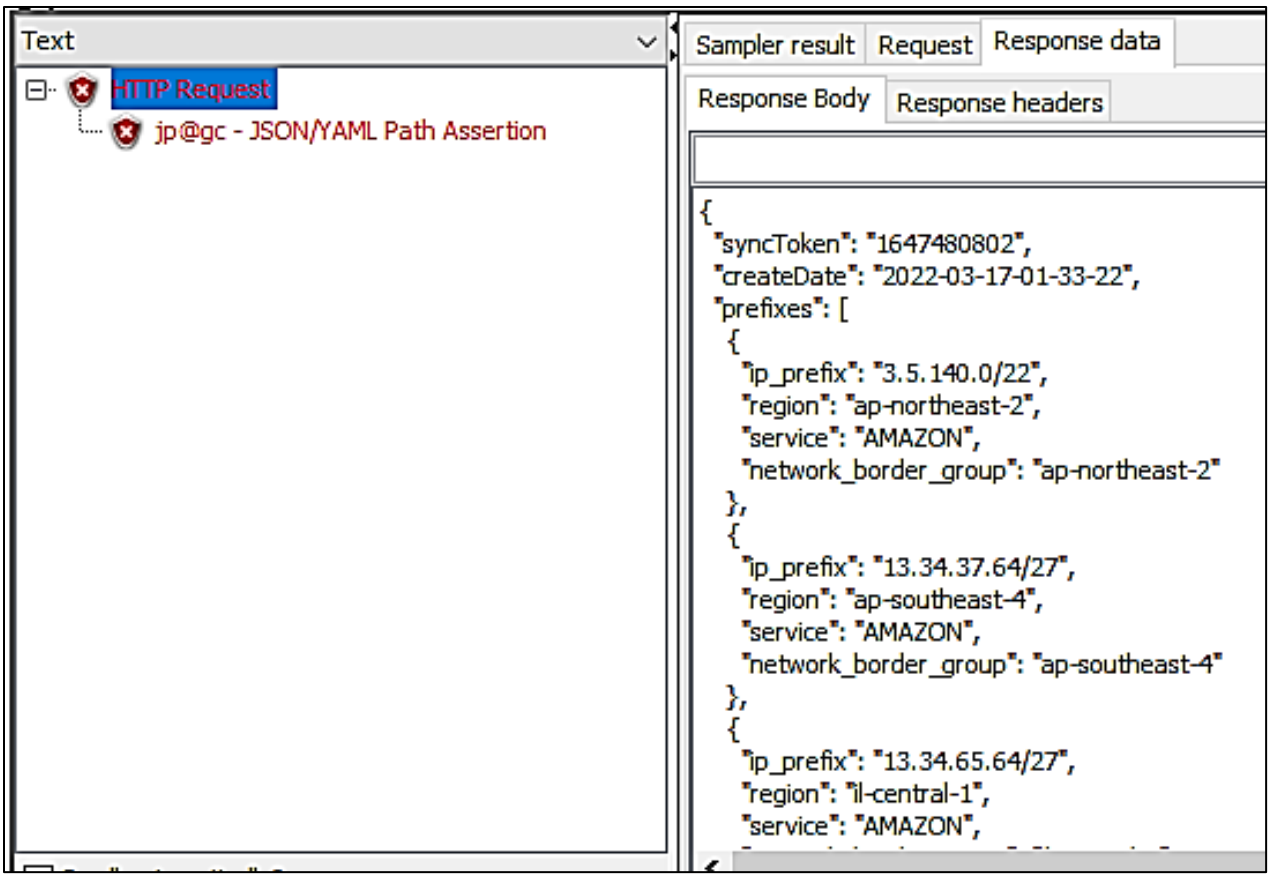
d. Add a http request to the thread group
Rclick thread group → add → sampler → http request
- Rename: HTTP JSON 1
- Protocol: https
- Servername/ip: ip-ranges.amazonaws.com
- Method: Get
- Path: /ip-ranges.json

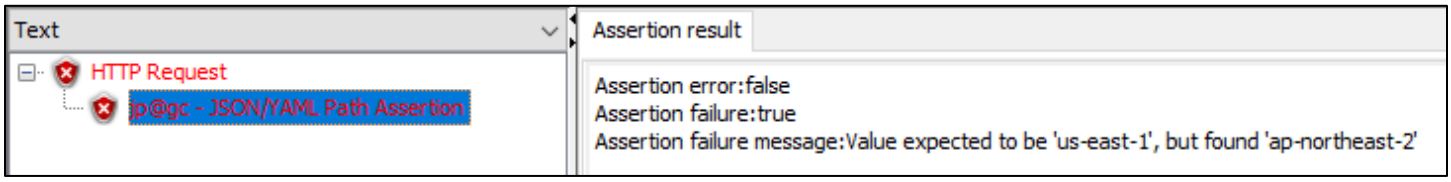
e. Add assertion to the http request sampler
Rclick http request → add → assertion → JSON/YAML Path Assertion

- Assert Json Path Exist: $.prefixes[0].region
- Additionally assert value: enable
- Match as regular expression: disable
- Expected value: us-east-1

f. Add view results in tree
Rclick thread group → add → listeners → view results in tree

g. Run the test
You may get a failing test if the first prefix is not the expected value





h. Try modifying the test to see a passing test

Use a response assertion to see if a text is contained in the response.

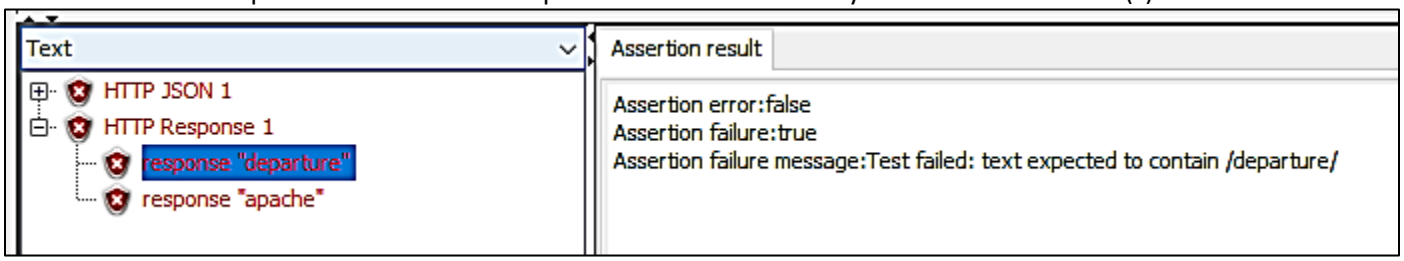
i. You may use the same test plan and the same thread group for this demo

j. Add another http sampler
Rclick thread group → add → samplers → http sampler
- Rename: HTTP Response 1
- Protocol: https
- Servername/ip: ip-ranges.amazonaws.com
- Method: Get
- Path: /ip-ranges.json

k. Add a response assertion to the sampler in item "j"
Rclick http request sampler → add → assertions → response assertion
- Apply to: Main sample only
- Field to test: text response
- Pattern matching rules: contains
- Patterns to test: departure

l. Add **another** response assertion to the sampler in item "j"
Rclick http request sampler → add → assertions → response assertion
- Apply to: Main sample only
- Field to test: text response
- Pattern matching rules: contains
- Patterns to test: apache

m. Run the test plan and see the two response assertion fail. Analyze the assertion result(s).



n. Modify the pattern to test to demo a passing assertion (ex. "AMAZON")
o. (optional) demo response assertion [response code] and duration assertion

Testing database servers using JMeter GUI:

- navigate inside extracted JMeter folder→bin→jmeter.bat
- We will need:
  - Database name (might as well add sample table with data)
  - Server name
  - Port number
  - Username, Password (with permission to perform crud operations)

- JMeter
  - rclick Test Plan → add → threads → thread group
    - provide thread group name
    - add mysql-connector.jar to lib folder of jmeter
      ***jmeter_folder*/lib/<add mysql-connector.jar here>**
    - save
  - rclick Test Plan→add→config element → jdbc connection configuration

    variable name for created pool
    example: "db_variable"

    database url
    jdbc:mysql://servername:port/databaseName
    jdbc:mysql://servername:port/databaseName?
    useTimezone=true&serverTimezone=GMT%2B8

    jdbc driver class
    org.mariadb.jdbc.Driver
    com.mysql.jdbc.Driver

provide the username and password

- write and execute a sql query
    - rclick thread group→add→sampler→jdbc request
        - in the "variable name of pool declared in jdbc connection configuration" enter the variable name you set earlier
        - type in a sql query
    - rclick thread group→add→listener→view result tree
    - rclick thread group→add→listener→view result in table
    - click the play(or start) button on the menu and observe the results
    - you may add more thread to simulate multiple users
    - rclick thread group→add→sampler→jdbc request
        - set variable name of the jdbc connection again
        - try using an update query (you can also use an *insert* statement here)

Using Selenium Grid

Selenium Grid Standalone Mode

Step 1: Setting Up Selenium Grid
Selenium Grid 4 simplified its architecture by combining the roles of the Hub and Nodes into a single server that can act as both. You can start it in one of three modes: Hub, Node, or Standalone (which combines Hub and Node). For simplicity and to cover a typical use case, we'll run it in Standalone mode.

Download the Selenium Server: Go to the Selenium downloads page and download the latest version of the Selenium Server (Grid).

Start the Selenium Server in Standalone Mode:
Open a terminal or command prompt and navigate to the directory where you downloaded the Selenium Server jar file. Run the following command to start it in Standalone mode:

```
java -jar selenium-server-<version>.jar standalone
```

Replace <version> with the version number of the downloaded jar file.

This starts Selenium Server as both the Hub and Node, allowing you to run tests directly against it without further configuration.

Step 2: Writing a Test with Selenium Grid in Java
Ensure you have Selenium WebDriver and the WebDriver for the browsers you intend to test against added to your project's dependencies. If you're using Maven, your pom.xml file should include dependencies similar to these:

```
<dependencies>
  <dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>4.0.0</version> <!-- Use the latest version -->
  </dependency>
</dependencies>
```

Next, write a Java test class. This class will initialize a RemoteWebDriver instance pointing to your Selenium Grid server, then perform a simple test (e.g., opening a page and checking the title).

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;
import java.net.MalformedURLException;
import java.net.URL;

public class GridTest {

   public static void main(String[] args) throws MalformedURLException {
      // Define desired capabilities for the browser.
      DesiredCapabilities capabilities = new DesiredCapabilities();
      capabilities.setBrowserName("chrome"); // You can change this to "firefox", "edge", etc.

      // Initialize the WebDriver instance using the RemoteWebDriver and pointing it to the Grid server.
      WebDriver driver = new RemoteWebDriver(new URL("http://localhost:4444"), capabilities);

      // Your test code here. For example, open a web page and check its title.
      driver.get("http://example.com");
      System.out.println("Title of the page is: " + driver.getTitle());

      // Quit the driver session.
      driver.quit();
   }
}
```

In this example, you create a RemoteWebDriver that communicates with the Selenium Grid. The URL http://localhost:4444 is where the Grid is listening for incoming requests. You need to specify the desired capabilities for the browser you want to use. This example uses Chrome, but you can change the setBrowserName parameter to use other browsers.

Step 3: Run Your Test

Compile and run your Java test. The test will send commands to the Selenium Grid, which in turn will execute the test actions in the specified browser.

Running Parallel tests

```
package package1;

import java.net.MalformedURLException;
import java.net.URL;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;
import org.testng.annotations.Test;

public class Test1 {
```

```
	@Test
	public void test1() {
			try {
				System.setProperty("webdriver.chrome.driver",
						"C:\\chromedriver-win64\\chromedriver.exe");
				DesiredCapabilities capabilities = new DesiredCapabilities();
				capabilities.setBrowserName("chrome"); // You can change this to "firefox", "edge", etc.

				// Initialize the WebDriver instance using the RemoteWebDriver and pointing it to the
Grid server.
				WebDriver driver = new RemoteWebDriver(new URL("http://localhost:4444"),
capabilities);

				driver.get("https://psa.gov.ph/");
				Thread.sleep(3000);

				driver.quit();
			} catch (Exception e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
	}

	@Test
	public void test2() {
			try {
				System.setProperty("webdriver.chrome.driver",
						"C:\\chromedriver-win64\\chromedriver.exe");
				DesiredCapabilities capabilities = new DesiredCapabilities();
				capabilities.setBrowserName("chrome"); // You can change this to "firefox", "edge", etc.

				// Initialize the WebDriver instance using the RemoteWebDriver and pointing it to the
Grid server.
				WebDriver driver = new RemoteWebDriver(new URL("http://localhost:4444"),
capabilities);

				driver.get("https://psa.gov.ph/");
				Thread.sleep(3000);

				driver.quit();
			} catch (Exception e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
	}

	@Test
	public void test3() {
			try {
```

```
                                System.setProperty("webdriver.chrome.driver",
                                                "C:\\chromedriver-win64\\chromedriver.exe");
                                DesiredCapabilities capabilities = new DesiredCapabilities();
                                capabilities.setBrowserName("chrome"); // You can change this to "firefox", "edge", etc.

                                // Initialize the WebDriver instance using the RemoteWebDriver and pointing it to the
Grid server.
                                WebDriver driver = new RemoteWebDriver(new URL("http://localhost:4444"),
capabilities);

                                driver.get("https://psa.gov.ph/");
                                Thread.sleep(3000);

                                driver.quit();
                        } catch (Exception e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
        }



}
```

Add/update the highlighted sectionsof your testNG xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Suite" parallel="methods" thread-count="3">
  <test thread-count="3" name="Test">
    <classes>
      <class name="package1.Test1"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

Step 4: Expanding Your Grid (Optional)

For more complex setups, such as running tests in parallel across different browsers and operating systems, you may want to set up additional nodes. This involves starting new instances of the Selenium Server in Node mode and registering them with your Hub (or Standalone server). However, in Standalone mode, for simplicity, the server acts as both the Hub and Node, suitable for development or small-scale testing environments.

```
java -jar selenium-server-<version>.jar node --detect-drivers true --publish-events tcp://hub-address:4442 --subscribe-events tcp://hub-address:4443
```

Replace <version> with the version number of your jar file and hub-address with the address of your Hub or Standalone server.

Selenium Grid Hub and Node Mode

To use Selenium Grid in Hub mode, you need to set up a central Hub that will manage WebDriver sessions and direct tests to run on various Nodes. Nodes are essentially test environments with different browser and system configurations. This setup allows you to run tests across multiple browsers, versions, and operating systems simultaneously. Below, I'll guide you through setting up the Hub and connecting Nodes to it.

Step 1: Download the Selenium Server
First, ensure you have downloaded the latest version of the Selenium Server (Grid) from the Selenium downloads page. This server will act as both the Hub and the Nodes.

Step 2: Start the Hub
Open a terminal or command prompt.
Navigate to the directory where you've downloaded the Selenium Server jar file.
Start the Hub using the following command:

```
java -jar selenium-server-<version>.jar hub
```

Replace <version> with the version number of the Selenium Server jar file you downloaded. This command starts the Hub on your local machine.

The Hub will start listening for Nodes to register and for test requests. By default, the Hub listens on port 4444.

Step 3: Set Up Nodes
After starting the Hub, you need to set up Nodes that will register with this Hub. You can start Nodes on the same machine as the Hub for testing purposes or on different machines to simulate different environments.

Open a new terminal or command prompt window for each Node you wish to start.
Navigate to the directory where the Selenium Server jar file is located.
Start a Node and register it with the Hub using the following command:

```
java -jar selenium-server-<version>.jar node --detect-drivers true

java -jar selenium-server-<version>.jar node --detect-drivers true --publish-events tcp://<HUB_IP>:4442 --subscribe-events tcp://<HUB_IP>:4443
```

- Replace <version> with the version number of your Selenium Server jar file.
- Replace <HUB_IP> with the IP address of the machine where the Hub is running. If you're running the Node on the same machine as the Hub, you can use localhost.

This command starts a Node that automatically detects available drivers (e.g., ChromeDriver, GeckoDriver) and registers itself with the Hub. The --detect-drivers true option tells the Node to automatically detect compatible WebDriver binaries in the system's PATH. If you're running specific browsers, ensure their corresponding WebDriver binaries are installed and properly set up in your system's PATH.

Step 4: Run Tests Through the Hub
With the Hub and Nodes set up, you can now run your Selenium tests. Configure your test scripts to connect to the Hub using the RemoteWebDriver and specify the desired capabilities for your test. Here’s a simple Java example that connects to the Hub and requests a Chrome browser session:

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;
import java.net.URL;
```

```
public class SeleniumGridTest {
   public static void main(String[] args) throws Exception {
      DesiredCapabilities capabilities = new DesiredCapabilities();
      capabilities.setBrowserName("chrome");

      WebDriver driver = new RemoteWebDriver(new URL("http://<HUB_IP>:4444"), capabilities);

      driver.get("http://example.com");
      System.out.println("Title of the page is: " + driver.getTitle());

      driver.quit();
   }
}
```

Replace <HUB_IP> with the IP address of your Hub. This test initializes a WebDriver that connects to the Selenium Grid Hub, requests a Chrome browser session, opens a web page, prints its title, and then quits.

Tips for Using Hub Mode

- Monitor the Hub: The Hub's console provides valuable information about registered Nodes, available sessions, and ongoing tests. You can access the Hub console by navigating to http://<HUB_IP>:4444 in a web browser.
- Scale Your Grid: To enhance your testing capabilities, add more Nodes with different configurations. This allows you to run tests across a wide range of environments.
- Maintain Your Grid: Regularly update the Selenium Server and browser drivers on both the Hub and Nodes to ensure compatibility and to take advantage of new features and bug fixes.
- By setting up Selenium Grid in Hub mode and connecting multiple Nodes, you significantly increase your test coverage and efficiency, enabling parallel execution of tests across various environments.

Selenium IDE

Selenium IDE (Integrated Development Environment) is an easy-to-use tool for creating automated tests directly in your browser. It's designed for rapid test development and is particularly useful for creating quick bug reproduction scripts and exploratory testing scripts. Selenium IDE supports both Chrome and Firefox browsers. Here's a basic guide on how to use Selenium IDE:

Step 1: Install Selenium IDE

- For Chrome: Visit the Chrome Web Store and search for Selenium IDE, then click "Add to Chrome" to install it.
- For Firefox: Visit the Firefox Browser Add-Ons page and search for Selenium IDE, then click "Add to Firefox" to install it.

Step 2: Open Selenium IDE

After installation, you can open Selenium IDE from your browser's extensions menu. The first time you open it, you'll be greeted with a welcome screen and options to create a new project or open an existing one.

Step 3: Create a New Project

- Create a New Project: Give your project a name. This is where all your tests for a specific website or application will be stored.
- Enter the Base URL: This is the starting point for your tests. Usually, it’s the homepage of the website you want to test.

Step 4: Record a Test

- Click 'Record a new test in a new project': After setting up your project, you'll be prompted to create and name your first test.
- Navigate in the Browser: As you interact with the web application (clicking links, filling out forms, etc.), Selenium IDE records your actions as steps in a test.
- Stop Recording: Once you've completed the actions you want to test, click the "Stop Recording" button in Selenium IDE.

Step 5: Playback Your Test

To see if your test works as expected, click the "Run" button in Selenium IDE. It will execute the steps you recorded in the browser. Watch as it automatically performs each action you recorded.

Step 6: Save Your Project

It's a good practice to save your project after creating or modifying tests. This way, you can easily revisit and run your tests in the future.

Step 7: Edit Your Test

- Add Assertions: To make your tests more robust, you can add assertions. For example, you can verify that a specific element is present on the page after an action is performed.
- Use the Command Editor: If you need to modify or add new steps manually, you can use the Command Editor. Each step consists of a Command (the action), Target (where the action takes place), and Value (any input needed for the command).

Step 8: Export Your Test

Selenium IDE allows you to export your tests in various programming languages like Python, Java, C#, etc. This is useful if you want to integrate your tests into a larger test suite or CI/CD pipeline.

Additional Tips

- ✓ Use the 'Select Target in Page' tool: This tool allows you to click on an element in your browser window and automatically generates the target selector for it in Selenium IDE.
- ✓ Explore Plugins: Selenium IDE supports plugins that can extend its functionality. Check the available plugins to see if there’s anything that could be useful for your testing needs.
- ✓ Practice with Different Commands: Familiarize yourself with different commands available in Selenium IDE to make your tests more flexible and powerful.