**What is Selenium?**

* Selenium is automation tool which is used to automate web based application for software automation process.
* It contains several classes, commands which are useful to handle different web element.

**Advantages of Selenium:**

1. It is open source automation tool.

2. It supports multiples programming languages such as java, python, C sharp etc.

3. Cross browser testing is possible

4. Cross platform testing is also possible

**Disadvantages of Selenium:**

1. We can’t automate desktop based application.

2. We can’t automate captcha code using selenium.

3. We can’t read barcode using selenium tool.

4. Ad-hoc testing (we know application but can’t have test data) can’t be performed.

**Selenium Architecture:**

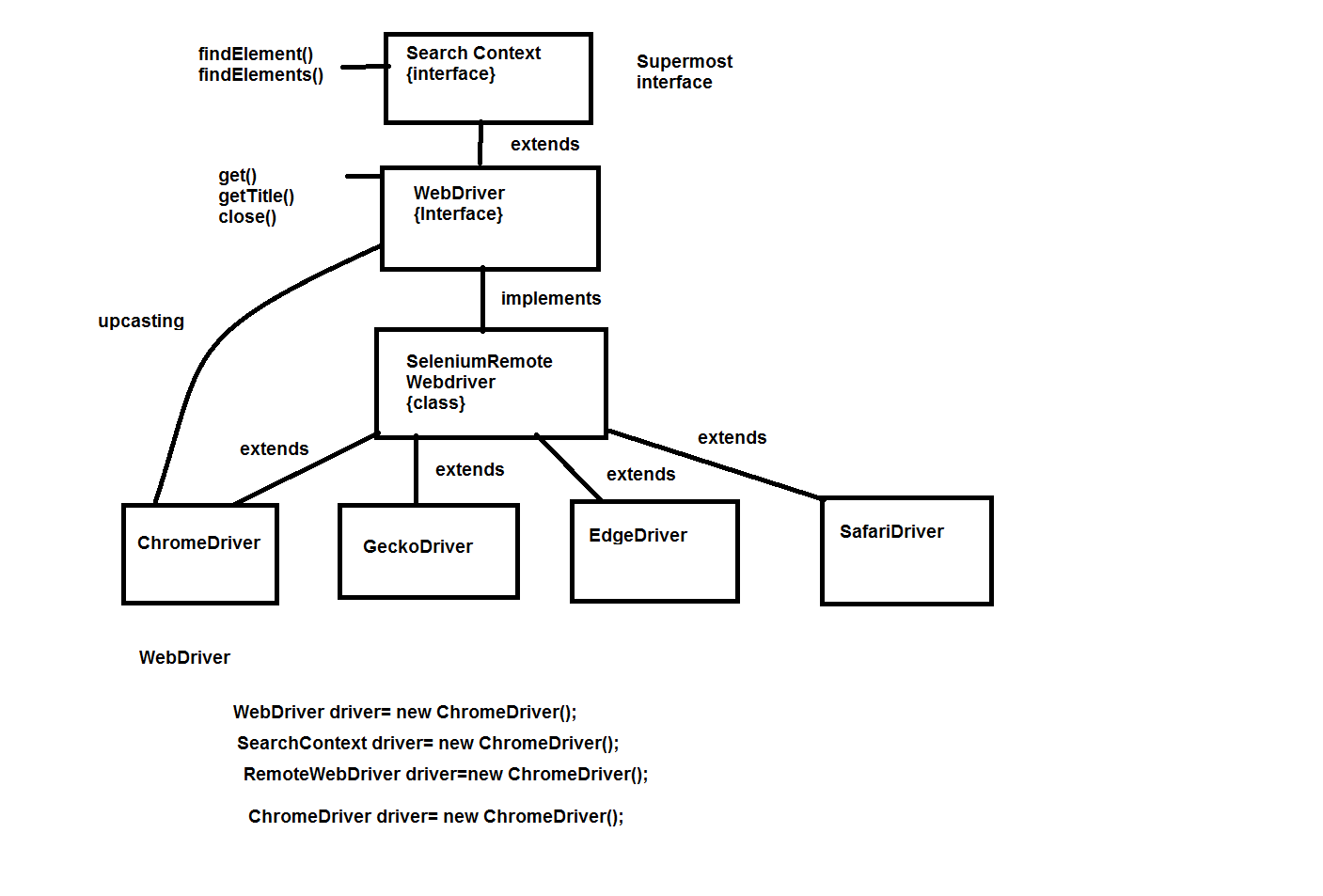
1. Search Context: It is a super most interface in selenium. It consists of all abstract methods and that methods are inherited to the Webdriver interface.

2. Webdriver: It is an interface present in selenium which consists of two types of abstract methods that is abstract methods of search context and his own abstract methods.

3. Selenium Remote Webdriver: It a class which implements all the abstract methods of both the interfaces that is search context and Webdriver. This implementation class is extended to the different browsers such as chrome. Firefox, internet explorer etc.

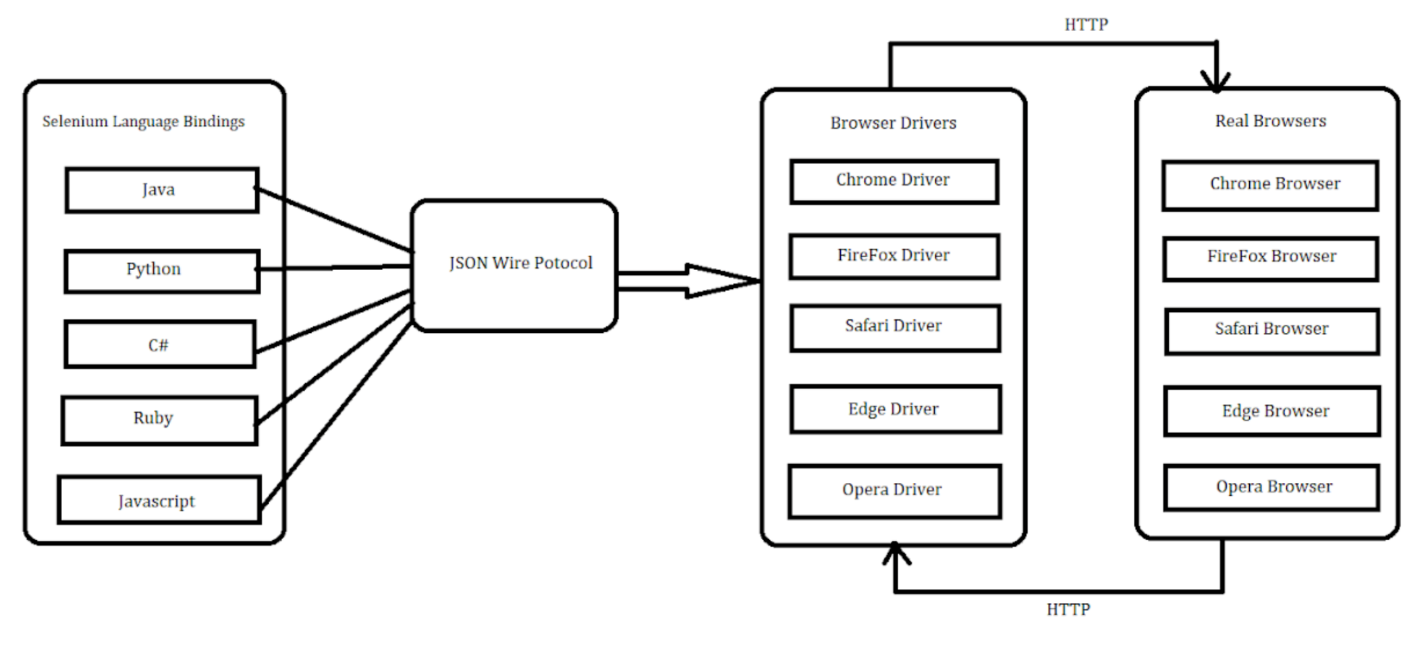
4. Browser driver: For compatibility testing we need to use runtime polymorphism to perform up casting in selenium. For example, to open a browser using script, we create an object of chrome driver with reference of Webdriver interface.

WebDriver driver = new ChromeDriver ();

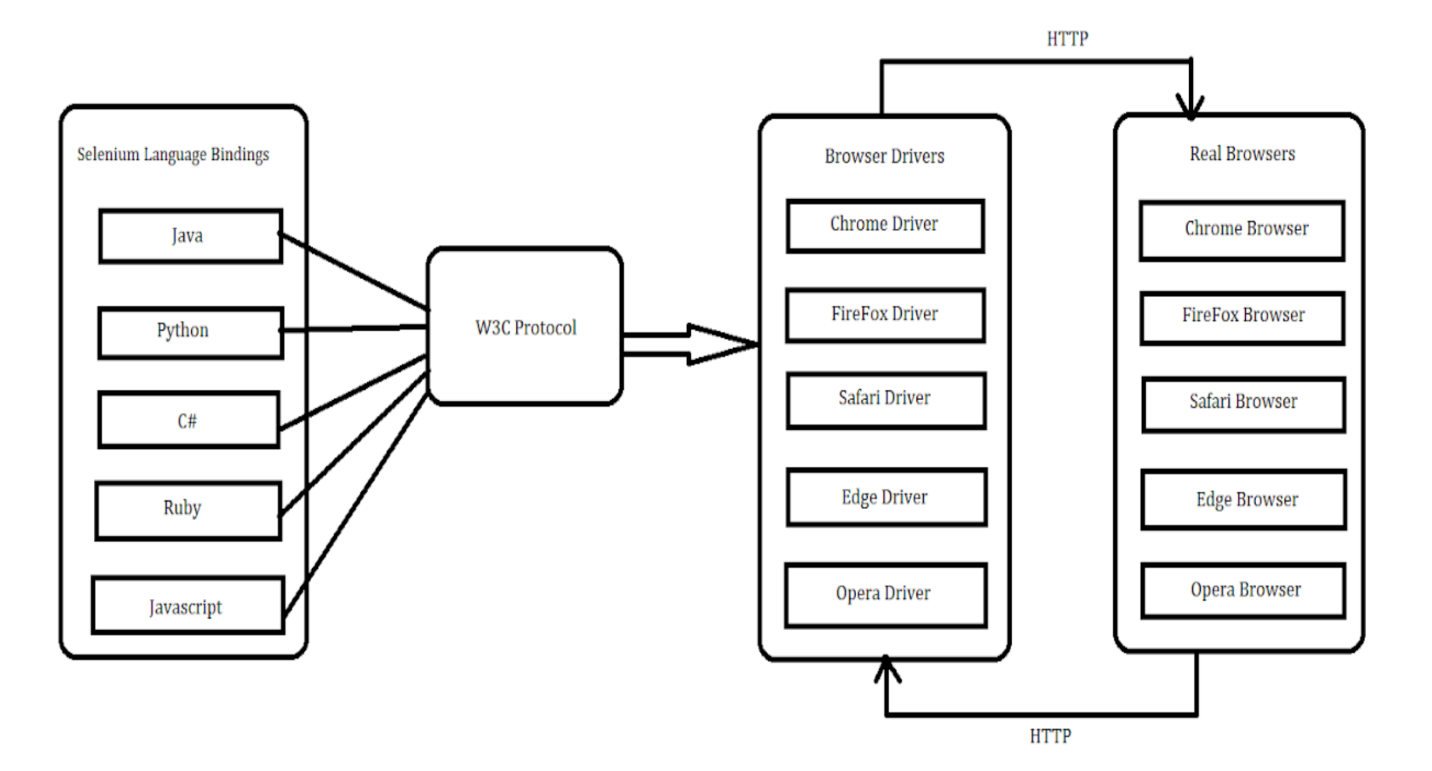
****

|  |
| --- |
| **Generalization:**   * Extracting all the important common properties & declaring it in super class (i.e. super interface) & providing implementation/definition according to subclass specification is called Generalization. * Generalization file can be normal java class or abstract class or Interface, but only Interface is recommended. |

**Selenium 3:**



**Selenium 4:**



**What are the different Locators in Selenium with Syntax?**

1. ID

driver.findElement(By.id(“id\_value”));

1. Name

driver.findElement(By.name(“name\_value”));

1. Class Name

driver.findElement(By.className(“nameofClass”));

1. Tag Name

driver.findElement(By.tagName(“tagname”));

1. Link Text

driver.findElement(By.linkText(“link\_text”));

1. Partial Link Text

driver.findElement(By.partialLinkText(“part\_of\_link\_text”));

1. Css Selector

**ID selector:**

driver.findElement(By.cssSelector(“tag#id”));

driver.findElement(By.cssSelector(“#id”));

driver.findElement(By.cssSelector(“tag#id[attribute=’value’]”));

**Class Selector:**

driver.findElement(By.cssSelector(“.class”));

driver.findElement(By.cssSelector(tag.class));

driver.findElement(By.cssSelector(“tag.class[attribute=’value’]”));

**Attribute Selector:**

driver.findElement(By.cssSelector(“[attribute=’value’]”));

driver.findElement(By.cssSelector(“tag[attribute=’value’]”));

driver.findElement(By.cssSelector(“tag[attribute=’value’][attribute2=’value2’]”));

**Substring selector:**

Starts with: driver.findElement(By.cssSelector(“tagname[attribute^=’start\_value’]”));

Ends with: driver.findElement(By.cssSelector(“tagname[attribute$=’end\_value’]”));

Contains: driverfindElement(By.cssSelector(“tagname[attribute\*=’substring’]”);

node:nth-child(n) – selects nth child of node

eg - div:nth-child(2)

<ul id="fruits">

<li>Apple</li>

<li>Banana</li>

<li>Mango</li>

<li>Orange</li>

</ul>

WebElement thirdFruit = driver.findElement(By.cssSelector("#fruits li:nth-child(3)")); >>Mango

If the parent has **mixed tags**, you might prefer :nth-of-type():

<div>

<p>First</p>

<span>Ignore me</span>

<p>Second</p>

</div>

* p:nth-child(2) ❌ won’t work (because 2nd child is <span>)
* p:nth-of-type(2) ✅ selects the second <p>, which is what you want.

**🧪 Java Example: nth-of-type**

java

CopyEdit

WebElement secondPara = driver.findElement(By.cssSelector("p:nth-of-type(2)"));

**Select even/odd items**

driver.findElements(By.cssSelector("li:nth-child(even)")); // 2nd, 4th, 6th...

driver.findElements(By.cssSelector("li:nth-child(odd)")); // 1st, 3rd, 5th...

1. Xpath

**Absolute:**

it is the direct way to find the element, but the disadvantage of the absolute XPath is that if there are any changes made in the path of the element then that XPath gets failed.

The key characteristic of XPath is that it begins with the single forward slash(/) ,which means you can select the element from the root node.

e.g. /html/body/div[2]/div[1]/div/h4[1]/b/html[1]/body[1]/div[2]/div[1]/div[1]/h4[1]/b[1]

**Relative:**

starts from the middle of HTML DOM structure. It starts with double forward slash (//). It can search elements anywhere on the webpage, means no need to write a long xpath and you can start from the middle of HTML DOM structure. Relative Xpath is always preferred as it is not a complete path from the root element.

Xpath=//input[@name='uid']

**Using AND & OR**

=//tag[@attribute=’value’ and @attribute=’value’]

=//tag[@attribute=’value’ or @attribute=’value’]

**Xpath methods**

**Contains()**

Contains() is a method used in XPath expression. It is used when the value of any attribute changes dynamically, for example, login information.

=//tag[contains(@attribute,’value’)]

=//tag[contains(text(),'here')]

**starts-with()**

In this method, the starting text of the attribute is matched to find the element whose attribute value changes dynamically. You can also find elements whose attribute value is static (not changes).

=//tag[starts-with(@attribute,’value’)]

**Text()**

 It helps to find the exact text elements and it locates the elements within the set of text nodes. The elements to be located should be in **string** form.

=//tag[text()=’value’]

**XPath axes methods**

XPath Axes are the methods used to find dynamic elements, which otherwise not possible to find by normal XPath method

e.g ancestors, following, child, Preceding, following-sibling, Parent, Self, Descendant,

**Difference between Xpath and css selector?**

|  |  |
| --- | --- |
| **XPath Selector** | **CSS Selector** |
| Xpath is bidirectional you can traverse elements from parent to child or child to parent | CSS selector is unidirectional you can only traverse from parent to child |
| XPath is slower in terms of speed and performance | CSS selector is comparatively faster |
| XPath allows the construction of text-based selectors | CSS doesn’t allow to construct the text-based selectors |
| XPath should start with / or // followed by a tag name or wildcards like \* | CSS allows to directly direct some of the attribute-based selectors such as # for id and . for classes |
| XPath provides Axes to solve complicated selector problems | CSS doesn’t have any Axes methods |
| Xpath is less readable as it grows | CSS Selectors are more readable |

**Syntax for opening browser:**

WebDriver driver= new ChromeDriver(); OR WebDriver driver= new EdgeDriver();

System.setProperty(“webdriver.edge.driver”,”Path of driver exe file”);

**WebDriver Methods:**

**driver.close();**

Close the current window, quitting the browser if it's the last window currently open.

**driver.quit();**

Quits this driver, closing every associated window – closes all tabs opened by browser

**Driver.findElement(By by);**

Find the first matching [WebElement](eclipse-javadoc:%E2%98%82=CucumberPOM/C:%5C/Users%5C/pratiksha.dilip.kale%5C/.m2%5C/repository%5C/org%5C/seleniumhq%5C/selenium%5C/selenium-api%5C/4.26.0%5C/selenium-api-4.26.0.jar=/maven.pomderived=/true=/=/maven.pomderived=/true=/=/maven.groupId=/org.seleniumhq.selenium=/=/maven.artifactId=/selenium-api=/=/maven.version=/4.26.0=/=/maven.scope=/compile=/%3Corg.openqa.selenium(WebDriver.class%E2%98%83WebDriver~findElement~Lorg.openqa.selenium.By;%E2%98%82WebElement) using the given By locator.

WebElement element=driver.findElement(By by);

Returns exception as elementNotFound

**Driver.findElements(By by);**

Find all elements within the current page using the given mechanism.

List<WebElement> elements= driverfindElement(By by);

Returns empty list if elements are not found

**driver.get(String url);**

Load a new web page in the current browser window. This is done using an HTTP POST operation, and the method will block until the load is complete

**driver.getCurrentUrl();**

Get a string representing the current URL that the browser is looking at.

**driver.getPageSource();**

Get the source of the last loaded page.

**driver.getTitle();**

Get the title of the current page.

**Navigation commands in Selenium:**

Navigation commands in Selenium WebDriver perform operations that involve navigating through web pages and controlling browser behavior. These commands provide an efficient way to manage a browser's history and perform actions like going back and forward between pages and refreshing the current page.

The Navigation Command provides four methods: to(), back(), forward(), and refresh(). These methods allow the WebDriver to perform the following operations:

**1. to() Command**

**L**oads a new web page in the current browser window. It accepts a string parameter that specifies the URL of the web page to be loaded.  
driver.navigate.().to(String url);

**2. back() Command**

Moves back one step in the browser’s history stack. It does not accept any parameters and does not return anything.

driver.navigate().back();

**3. forward() Command**

Moves forward one step in the browser’s history stack. It does not accept any parameters and does not return anything.

driver.navigate().forward();

**4. refresh() Command**

Reloads the current web page in the browser window. It does not accept any parameters and does not return anything.  
driver.navigate().refresh();

**Waits in Selenium:**

Dynamic Web Applications often load asynchronously making it difficult to predict whether a particular element we want to interact with is available for interaction or not.

Selenium Waits is a mechanism that helps to address this issue, it instructs the automation script to wait for a certain condition to be met before we move on to the next stage.

Without proper synchronization, tests may fail due to the element not being available or ready when we are trying to interact with it.

**1. Implicit Waits**

Implicit Wait is a type of Wait in Selenium which instructs the Web Driver to wait for a certain amount of time before throwing an exception if an element is not found. It is applied globally to each, and every element location calls for the entire session. If the implicit wait is set for 10 seconds, the Web Driver will wait for 10 second before throwing an error and during that duration as soon as the element is found it will return the elements reference.

driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(5));

**2. Explicit Waits**

Explicit wait is a type of wait in Selenium which instructs the Web Driver to wait until a certain condition is met or maximum time is elapsed. Unlike Implicit Waits, Explicit waits are not applied globally they are more specific and allows the Web Driver to wait for a certain condition for a particular element before throwing an error.

WebDriverWait wait=new WebDriverWait(driver, Duration.ofSeconds(5));

wait.until(ExpectedConditions.visibilityOfElementLocated(webElement)).click();

**Expected Conditions**

alertIsPresent()

elementSelectionStateToBe()

elementToBeClickable()

elementToBeSelected()

frameToBeAvaliableAndSwitchToIt()

invisibilityOfTheElementLocated()

invisibilityOfElementWithText()

presenceOfAllElementsLocatedBy()

presenceOfElementLocated()

textToBePresentInElement()

textToBePresentInElementLocated()

textToBePresentInElementValue()

titleIs()

titleContains()

visibilityOf()

visibilityOfAllElements()

visibilityOfAllElementsLocatedBy()

visibilityOfElementLocated()

**3. Fluent Wait**

The Fluent Wait in Selenium is used to define maximum time for the web driver to wait for a condition, as well as the frequency with which we want to check the condition before throwing an “ElementNotVisibleException” exception. It checks for the web element at regular intervals until the object is found or timeout happens.

WebDriverWait fwait=(WebDriverWait)

new FluentWait(driver).

withTimeout(Duration.ofSeconds(30)).

pollingEvery(Duration.ofSeconds(5)).

ignoring(NoSuchElementException.class);

WebElement El1 = fwait.until(new Function<WebDriver, WebElement>()

{

public WebElement apply(WebDriver driver)

{

return driver.findElement(username);

}

});

El1.click();

**Handle windows in selenium:**

Handling multiple windows in Selenium is done using window handles. One navigates between several browser tabs or windows.

To get the ID of Windows we will use the following method.

String ID=driver.getWindowHandle(); To get the ID of the parent window (main window)

Set<String> Ids= driver.getWindowHandles(); To get the ID of child windows (new window)

**What does window handle mean in Selenium?**

*Window handle uniquely identifies browser windows or tabs where Selenium is performing. It allows us to switch our control across multiple windows or tabs while working with automation.*

**How to get the handle of the current window?**

*Get the handle of the currently active window using* driver.getWindowHandle().

**How to get a list with all the opened window handles?**

*Obtain the list of handles for all open windows and tabs using the* driver.getWindowHandles();.

**How do I jump to a different window or tab?**

*First of all, get all the window handles by using driver.getWindowHandles(); then do* driver.switchTo().window(handle) *with the handle you need.*

**Can I launch multiple windows and tabs simultaneously?**

*Yes, you will handle several windows or tabs. You just need to switch them when you want with their handles. Just keep a tab on the handle so you can quickly switch back and forth.*

***Example: Handling Multiple Windows***

*Here is an example of handling multiple windows using Selenium WebDriver:*

import java.util.Iterator;

import java.util.Set;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

public class WindowHandleDemo {

public static void main(String[] args) throws Exception {

System.setProperty("webdriver.chrome.driver", "Path to the driver");

WebDriver driver = new ChromeDriver();

driver.manage().window().maximize();

// Load the website

driver.get("http://www.naukri.com/");

// Get the handle of the parent window

String parentWindowHandle = driver.getWindowHandle();

Set<String> allWindowHandles = driver.getWindowHandles();

// Iterate through child windows

Iterator<String> iterator = allWindowHandles.iterator();

while (iterator.hasNext()) {

String childWindowHandle = iterator.next();

if (!parentWindowHandle.equals(childWindowHandle)) {

driver.switchTo().window(childWindowHandle);

System.out.println(driver.switchTo().window(childWindowHandle).getTitle());

driver.close();

}

}

// Switch back to the parent window

driver.switchTo().window(parentWindowHandle);

}

}

**Handle frames in selenium:**

**What are iframes in Selenium?**

Displaying a webpage as a part of another webpage this parameter we called iframe. If the element is present on the main page, then we can directly interact and we can perform the actions but when the element is present on the iframe we cannot directly interact with those elements. In such cases, we need to change the focus of selenium from the main page to the iframe.

**Difference between frame and iframe in Selenium**

In Selenium, both frame and iframe elements allow you to embed content or even another web page within the main page. However, there are slight differences also exist which are as follows:

* **Frame:**A **frame is an HTML element** that **divides**the **webpage**into **different**sections, each having its **own content**. It becomes an **older**element in **HTML**which is now replaced by more modern methods for **creating layouts**.
* **iFrame:**An **iframe** ( inline frame) is more commonly used today. It allows us to **embed external content or another webpage inside our main HTML document.** For example, an embedded **YouTube video is often placed inside an iframe.**

**How to Switch Over the Elements in iframes using Web Driver Commands?**

**Using the SwitchTo().frame function:**

The SwitchTo().frame() function is important while working with frames or iframes in Selenium. Before we can interact with elements inside a frame , we must first tell Selenium which frame we want to focus on.

**Syntax:**

driver.switchTo().frame("frameName")

Once switch into a frame, Selenium will start interacting with the elements inside that frame only.

**1. With the help of the index:**

We can switch to a frame using its index. The index starts from 0, means the first frame on the page has an index 0, the second frame has an index 1 and so on.

**Syntax:**

driver.switchTo().frame(index);

driver.switchTo().frame(0);*//Switch to the first frame*

**2. With the help of id/name:**

If a frame has a**name or ID attribute**we can simply switch to it by using that name or ID.

**Syntax:**

driver.switchTo().frame("frameNameOrID");

*here we need to identify the id value or name value from the HTML codes of the iframe.*

This method is more reliable than switching by index because **name and IDs**typically remain static , even if the structure of page changes.

driver.switchTo().frame("loginFrame"); *// to switch to thr frame with the name ' loginFrame'*

**3. With the help of web element**

**Syntax:**

driver.switchTo().frame(webelement); *here we need to declare a webelement by using attributes of iframe.*

**How to Switch back to the Main iframe?**

Suppose we need to change the focus of selenium from iframe to the main page or parent iframe for which there are two different ways.

* driver.switchTo().parentFrame(); - We use this method to change the focus of selenium from one iframe to parent iframe (previous iframe)
* driver.switchTo().defaultContent(); - We use this method to change the focus of selenium from one iframe to the main page.

**Can selenium interact with elements inside an iframe without switching to it first?**No, selenium cannot directly interact with the elements inside a frame or iframe.we must switch to that frame using driver.switchTo().frame() before interacting with any elements inside.

**What happens if I try to access a frame that doesn't exist?**If the frame does not exist, selenium simply throws a NoSuchFrameException.

**Can I Switch to a nested frame inside another frame?**Yes, we can switch to a nested frame by first switching to the parent frame then switching to the nested frame.

**Is Switching by index safe to use?**Switching by index can be risky if the structure of the webpage changes .It's better to use a frame name or Id for more stable and maintainable code.

**How to Scroll an Element into View in Selenium?**

There are 2 different approaches to scroll an element into view in Selenium, which are as follows:

1. [Using JavascriptExecutorDriver](https://www.geeksforgeeks.org/javascriptexecutor-in-selenium/)
2. [Using Actions](https://www.geeksforgeeks.org/how-to-select-multiple-elements-using-actions-class-in-selenium-using-java/)

**Using JavascriptExecutorDriver**

The JavaScriptExecutorDriver is used when certain actions cannot be performed through the usage of Selenium in-built functions, and we need some external class to perform such actions. One such action is scrolling of elements into view, that cannot be performed by Selenium in-built functions. In this approach, we will see how we can scroll an element into view using JavascriptExecutorDriver in Selenium.

Syntax:

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript("arguments[0].scrollIntoView();", element);

Here,

**element**: It is the element that must be brought into view by scrolling.

**Using Actions**

The Actions is used when the users want web driver to perform numerous actions one after another. This method is great but has certain limitations with it as the **moveToElement** function which is used in it moves the mouse cursor to the center of the element. It doesn't make sure if the element is visible within the viewport or not. Thus, limiting the webdriver to perform further action on the element. In this approach, we will see how we can scroll an element into view using Actions in Selenium.

**Syntax:**

Actions a = new Actions(driver);

a.moveToElement(element).perform();

Here,

* **element**: It is the element that must be brought into view by scrolling.

*In conclusion, Selenium can handle only the elements that are visible within the viewport. Thus, scrolling of elements is very crucial for the elements currently out of viewport so that further actions of that element could be performed successfully. The technique of scrolling through JavascriptExecutorDriver is the recommended approach as it gives users the choice to control the web page through Javascript code and is comparatively faster than the other approach.*

* 1. **Is the performance of the Selenium script affected by scrolling of elements into view?**

***Ans:***Yes, scrolling an element into view affects the performance in Selenium tests, while it is very minimal. Still, it is recommended not to scroll elements much as it may increase the execution time.

* 1. **Is there any way to determine if the element is visible or not in the viewport?**

***Ans:***Yes, we can use the built-in function of Selenium, i.e., isDisplayed to check if the particular element is visible or not within the viewport.

* 1. **How To send Capital letters through selenium?**

Actions a = new Actions(driver);

WebElement element = driver.findElement(By.id(“id”));

Actions SriesOfActions = a.moveToElement(element).

KeyDown(element, Key.SHIFT).

sendKeys(element, “hello”).

KeyUP(element, key.SHIFT).

build();

SeriesOfActions.perform();

* 1. **How To mouse hover over a web Element ?**

Actions a = new Actions(driver);

a.moveToElement(By.id(“id”)).perform();

**Methods of Action class:**

* **Mouse Actions in Selenium:**

1. doubleClick(): Performs double click on the element
2. clickAndHold(): Performs long click on the mouse without releasing it
3. dragAndDrop(): Drags the element from one point and drops to another
4. moveToElement(): Shifts the mouse pointer to the center of the element
5. contextClick(): Performs right-click on the mouse
6. click():Perform click on the element
7. release():

* **Keyboard Actions in Selenium:**

1. sendKeys(): Sends a series of keys to the element
2. keyUp(): Performs key release
3. keyDown(): Performs keypress without release

**Assertions in Selenium:**

In Selenium, assertions are used for verification or checkpoints in the test case. If assertion is not used in the test case, it's not possible to determine whether the test case is passed or failed. Assertion will be used to generate the test execution reports. And if all the test steps present in the test cases passed successfully, then assertion will not impact the test. It will report only when a test case is failed.

To use the Assertion in Web Driver, you need to download the Testng jar file and add it to the eclipse.

**There are two types of Assertion:**

Hard Assertion

Soft Assertion

**Hard Assertion:**

Hard Assertion is an Assertion that throws the **AssertException** when the test case is failed. In the case of Hard Assertion, you can handle the error by using a catch block like a java exception. Suppose we have two test cases in a suite. The first test case in a suite has an assertion that fails, and if we want to run the second case in a suit, then we need to handle the assertion error. A Hard Assertion contains the following methods:

* AssertEquals
* AssertNotEquals
* AssertTrue
* AssertFalse
* AssertNull
* AssertNotNull

**AssertFalse()**

Assertion verifies the boolean value returned by a condition. If the boolean value is false, then assertion passes the test case, and if the boolean value is true, then assertion aborts the test case by an exception. Syntax of AssertFalse() method is given below:

Assert.AssertFalse(condition);

**AssertTrue()**

Assertion verifies the boolean value returned by a condition. If the boolean value is true, then assertion passes the test case, and if the boolean value is false, then assertion aborts the test case by an exception. Syntax of AssertTrue() method is given below:

Assert.AssertTrue(condition);

**AssertEquals()**

AssertEquals() is a method used to compare the actual and expected results. If both the actual and expected results are same, then the assertion pass with no exception and the test case is marked as "passed". If both the actual and expected results are not the same, then the assertion fails with an exception and the test case is marked as "failed". Syntax of an AssertEquals() method is given below:

Assert.assertEquals(actual,expected);

Assert.assertEquals(actual,expected, assertion error msg);  - error msg will be printed if assertion fails

**AssertNotEquals()**

It is opposite to the function of AssertNotEquals() method. AssertNotEquals() is a method used to compare the actual and expected results. If both the actual and expected results are not the same, then the assertion pass with no exception and the test case is marked as "passed". If both the actual and expected results are same, then the assertion fails with an exception and the test case is marked as "failed". Syntax of an AssertNotEquals() method is given below:

AssertNotEquals(actual,expected,message);

**AssertNull()**

AssertNull() is a method that verifies whether the object is null or not. If an object is null, then assertion passes the test case, and the test case is marked as "passed", and if an object is not null, then assertion aborts the test case and the test case is marked as "failed". Syntax of AssertNull() method is given below:

Assert.assertNull(object);

**AssertNotNull()**

AssertNotNull() is a method that verifies whether the object is null or not. If an object is not null, then assertion passes the test case and test case is marked as "passed", and if an object is null, then assertion aborts the test case and test case is marked as "failed". Syntax of AssertNotNull() method is given below:

Assert.assertNotNull(object);

**SoftAssertion**

Till now, we have learnt about the Hard Assertion in Web Driver using Testng framework. In hard assertion, if an assertion fails then it aborts the test case otherwise it continues the execution. Sometimes we want to execute the whole script even if the assertion fails. This is not possible in Hard Assertion. To overcome this problem, we need to use a soft assertion in testng.

softAssert softAssert = new SoftAssert();

In the end, we have to call the **assertAll**() method that is used to throw the exceptions and results at the end of the test.

*softAssert.assertAll();*

Soft Assert also supports all the methods used by Hard Asserts such as,

* AssertEquals
* AssertTrue
* AssertFalse
* AssertNotEquals

**Alerts in Selenium:**

**What are Alerts in Selenium?**

Alert in Selenium is a message/notification box that notifies the user about some information or asks for permission to perform a certain kind of operation. It may be used for warning purposes as well.

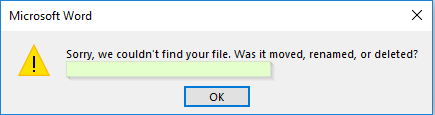
**Types of Alerts in Selenium**

There are three types of Alert in [Selenium](https://www.browserstack.com/selenium), described as follows:

1. Simple Alert
2. Prompt Alert
3. Confirmation Alert

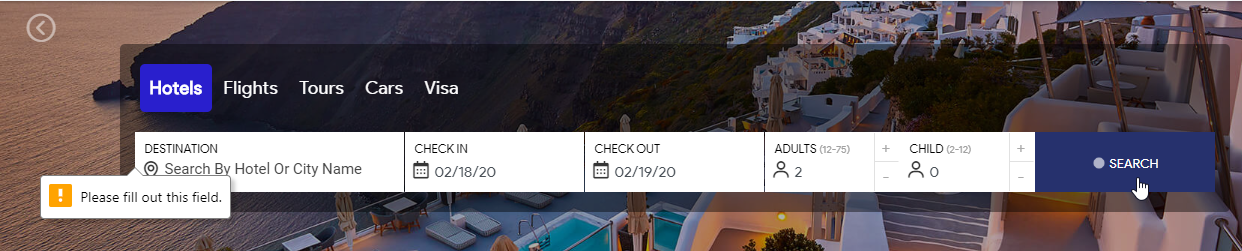
**1. Simple Alert**

This alert is used to notify a simple warning message with an ‘OK’ button, as shown in the below snapshot.



**2. Prompt Alert**

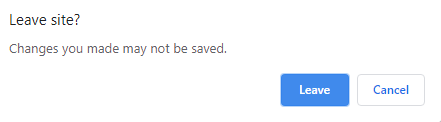
This alert will ask the user to input the required information to complete the task. In the below snapshot, you can see that without entering the destination for Hotel, you are not allowed to hit the search button.



Now, this input can be entered with the help of **sendKeys(“input\_text”)** method in [Selenium Webdriver](https://www.browserstack.com/guide/selenium-webdriver-tutorial).

**3. Confirmation Alert**

This alert is basically used for the confirmation of some tasks. For Example: Do you wish to continue a particular task? Yes or No? The snapshot below depicts the same.



**How to Handle Alerts in Selenium?**

Handling alerts manually is a tedious task. To reduce human intervention and ease this task, Selenium provides a wide range of functionalities and methods to handle alerts.

The following methods are useful to handle alerts in Selenium:

**1. Void dismiss():** This method is used when the ‘Cancel’ button is clicked in the alert box.

driver.switchTo().alert().dismiss();

**2. Void accept():** This method is used to click on the ‘OK’ button of the alert.

driver.switchTo().alert().accept();

**3. String getText():** This method is used to capture the alert message.

driver.switchTo().alert().getText();

**4. Void sendKeys(String stringToSend):** This method is used to send data to the alert box.

driver.switchTo().alert().sendKeys("Text");

**How to handle dropdowns in selenium:**

**Select class in Selenium WebDriver**

The 'Select' class in Selenium WebDriver is used for selecting and deselecting option in a dropdown. The objects of Select type can be initialized by passing the dropdown webElement as parameter to its constructor.

WebElement dropDown = driver.findElement(By.id("testingDropdown"));

Select dropdown = new Select(dropDown);

**How to select an option from drop-down menu?**

WebDriver provides three ways to select an option from the drop-down menu.

1. selectByIndex - It is used to select an option based on its index, beginning with 0.

dropdown.selectByIndex(5);

2. selectByValue - It is used to select an option based on its 'value' attribute.

dropdown.selectByValue("Database");

3. selectByVisibleText - It is used to select an option based on the text over the option.

dropdown.selectByVisibleText("Database Testing");

**How do I select an option from a drop-down using Selenium with Java?**

Use the Select class. Create an instance with new Select(driver.findElement(By.id("dropdownId"))), then use methods like selectByVisibleText("Option 2"), selectByIndex(1), or selectByValue("option2").

**Can Selenium handle multi-select drop-downs?**

Yes, you can handle multi-select drop-downs using the Select class by checking if the drop-down allows multiple selections with isMultiple(), and then selecting multiple options.

**How do I interact with a custom drop-down (not using the <select> tag)?**

For custom drop-downs, locate the drop-down element and click to reveal options, then select the desired option by finding and clicking on it using XPath or CSS selectors.

**What should I do if the drop-down options are dynamically loaded?**

Use WebDriverWait to wait for the options to be visible or present before interacting with them, ensuring that the drop-down is fully loaded.

**How can I handle drop-downs within an iframe?**

Switch to the iframe using driver.switchTo().frame("iframeId") before interacting with the drop-down, and switch back to the default content using driver.switchTo().defaultContent() after you're done.

**getOptions( ) : List<WebElement>**

This method gets all the options belonging to the Select tag. It takes no parameter and returns List<WebElements>

Select objSelect = new Select(driver.findElement(By.id("Search-box")));

List <WebElement> elementCount = objSelect.getOptions();

System.out.println(elementCount.size());

**deselectAll()**

This method clears all the selected entries. This is only valid when the drop-down element supports multiple selections.

Syntax:

objSelect.deselectAll();

**TRY**

**DeselectByIndex() void -Indexing starts with 0**

**DeselectByValue() -void**

**DeselectByVisibleText()- void**

**How to Select multiple items in Dropdown with the Select command?**

The multiple select attribute is a boolean expression. This method specifies that multiple options can be selected at once. These options vary for different operating systems and browsers.

* For Windows: Press the control (ctrl) button to select multiple options.
* For Mac: Hold down the command button to select multiple options.

Use the isMultiple method to select multiple commands.

*isMultiple(): boolean* – This method informs whether the Select element supports multiple selection options at the same time or not. This method accepts nothing and returns a boolean value (true/false).

Syntax:

objSelect.isMultiple();

**if (select.isMultiple()) {**

**// Select multiple options**

**select.selectByValue("apple");**

**select.selectByVisibleText("Banana");**

**select.selectByIndex(2); // Index starts from 0**

**}**

**getAllSelectedOptions() -List<WebElement>**

- This method is used to get only selected options present in the listBox.

**getFirstSelectedOption()- WebElement**

- This method is used to display first selected option in multiselected listBox.

**Handle checkbox in selenium:**

A checkbox in the DOM is defined using the input tag with the type as ‘**checkbox’**.

Checkboxes are the graphical user interface elements that allow users to select or deselect an item from a list of options, they will enable users to make binary choices either selecting or deselecting an item from a list of options.

**How to Select multiple options in Checkbox using Selenium**

There may be a group of checkboxes which have the same name/class/id attribute and therefore choosing the checkbox with respect to their value attribute is the best option if we need to click on a single checkbox.

And if there is a need to select multiple checkboxes which have the same name/id/class attribute but different attribute values, creating different web elements and then clicking on it one by one would be cumbersome. Instead, we can create a list of web elements and iterate through it sequentially and click on it.

Let us see how we can select all checkboxes in the above example with the below Selenium code.

* First, we will create a List of web elements to store all the checkboxes.

List<WebElement> chkboxes=driver.findElements(By.cssSelector("input[type='checkbox']"));

* Then we need to find out the size of the web element list.

int size=chkboxes.size();

* Then using the for loop and **get()** method we will iterate through all the check boxes and click on it using the **click()** method.

chkboxes.get(i).click();

**Validations on a checkbox using Selenium**

To validate pre and post conditions of checkbox’s state, Selenium provides certain methods which are as follows:

* **isEnabled():** A pre-validation for checkbox click event to check whether the checkbox is enabled or disabled on the web page. This method returns true in case element is enabled otherwise it returns false.
* **isDisplayed():** A pre-validation for checkbox click event to check whether the checkbox is displayed on the web page or not. It returns true if the desired element is displayed on DOM otherwise it returns false.
* **isSelected():** A post-validation after the checkbox click event to check whether the checkbox is selected or not. It returns true if the element is selected, else false for deselected.

Suppose there is a scenario where the user needs to first check the visibility of the checkbox and based on the visibility it needs to be selected. So in such a condition we can use **isDisplayed()** method as a precondition and on the basis of its result either true or false, the checkbox can be selected.

**How to assert that a checkbox is checked?**

If there is a need to validate whether the checkbox is selected after selecting the desired checkbox, we can use **isSelected()** method which returns true if the checkbox is selected and false in either case. The Selenium code to test this scenario is as follows:

**How to Deselect Checkbox in Selenium**

At the start of this article we learnt that the **click()** method of WebElement interface is used to click a checkbox. Similarly, to deselect a checkbox, the same **click()** method is used.

However, in web automation it is a good practice to check the state of the checkbox before deselecting it, otherwise clicking to deselect the checkbox will select the checkbox if the checkbox was not in a selected mode previously.

**Handle radio button in selenium:**

A Radio Button is an [HTML](https://en.wikipedia.org/wiki/HTML) element, which allows the user to select only one of the options given. We generally organize Radio buttons rally organized in a group that contains mutually exclusive options. We can select only one option out of the options given.

Radio buttons are the GUI element that allows users to select a single option out of several mutually exclusive options.

* Radio buttons are denoted by the <input> HTML tags having "type" as "radio"
* Moreover, we can locate the Radio button elements by id, name, XPath, or CSS selector.
* Also, we can select Radio buttons by using the click() method.
* Selenium also offers validation methods like isSelected(), isEnabled() and isDisplayed(). We can use these methods before performing any operation to make sure radio buttons are in the correct state.

**How to perform validations on Radio Buttons using Selenium WebDriver?**

***Selenium WebDriver*** provides certain methods that can pre and post validate the states of ***Radion Buttons***. Few of these methods are:

* ***isSelected():****Checks whether a radio button is selected or not.*
* ***isDisplayed()****: Checks whether a radion button is displayed on the web page or not.*
* ***isEnabled()****: Checks whether a radion button is enabled or not*

We can use these methods to validate the current state of the radio buttons. E.g., *to validate that after clicking the radio button, whether it's selected or not, we can use the "****isSelected()****" method. Similarly, before clicking a radio button, we can validate that whether the radio button displays on the page and is in enabled status. After validating, only then click on the radio button. So, we can do such pre validations using the "****isDisplayed()****" and "****isEnabled()****" methods*.

**Handle date picker in Selenium:**

[**How to select date from Datepicker in Selenium | BrowserStack**](https://www.browserstack.com/guide/datepicker-in-selenium)

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.edge.EdgeDriver;

import com.github.dockerjava.api.model.Driver;

public class DatePicker {

public static void main(String[] args) {

String Day="25";

String m\_y="March 2025";

WebDriver driver = new EdgeDriver();

System.setProperty("webdriver.edge.driver", "C:\\\\Users\\\\pratiksha.dilip.kale\\\\Downloads\\\\edgedriver\_win64\\\\msedgedriver.exe");

driver.get("https://www.goibibo.com/flights/?utm\_source=bing&utm\_medium=cpc&utm\_campaign=DF-Brand-EM&utm\_adgroup=Only%20Goibibo&utm\_term=!SEM!DF!B!Brand!RSA!150035352!1211662109442708!!e!goibibo!c!&msclkid=fad8e358a4b8174e41e184d2c8e135d4");

driver.findElement(By.cssSelector(".sc-12foipm-8")).click();

System.out.println("clicked");

while (true)

{

System.out.println("found1 "+ driver.findElement(By.cssSelector(".DayPicker-Months>div>:nth-child(1)")).getText());

System.out.println("found2 "+ driver.findElement(By.cssSelector(".DayPicker-Months>:nth-child(2)>div>:nth-child(1)")).getText());

if (driver.findElement(By.cssSelector(".DayPicker-Months>div>:nth-child(1)")).getText().equalsIgnoreCase(m\_y))

{

System.out.println("write a code to click on day" );

driver.findElement(By.xpath("//div[contains(@aria-label,'Tue Mar 25 2025')]")).click();

System.out.println("day selected: ");

break;

}

else if (driver.findElement(By.cssSelector(".DayPicker-Months>div>:nth-child(2)")).getText().equalsIgnoreCase(m\_y))

{

System.out.println("write a code to click on day" );

driver.findElement(By.xpath("//div[contains(@aria-label,'Tue Mar 25 2025')]")).click();

System.out.println("day selected" );

break;

}

else

{

System.out.println("click on next element");

driver.findElement(By.xpath("//span[@aria-label='Next Month']")).click();

}

}

//driver.close();

}

}

**Take Screenshot in Selenium:**

To capture screenshots in Selenium, one must utilize the method TakesScreenshot. This notifies WebDriver that it should take a screenshot in Selenium and store it.

File file = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);

String screenshotBase64 = ((TakesScreenshot)driver).getScreenshotAs(OutputType.BASE64);

e.g.

String screenshotName = sc.getName().replace(" ", "\_");

**byte** [] sourcePath= ((TakesScreenshot)driver).getScreenshotAs(OutputType.***BYTES***);

sc.attach(sourcePath, "image/png", screenshotName);

 **Type Casting:**  
((TakesScreenshot) driver)  
This is **explicit casting**, where you're casting the WebDriver instance to a TakesScreenshot interface. This tells the compiler that your driver can perform screenshot operations.

 **Method Chaining:**  
.getScreenshotAs(OutputType.FILE)  
Once the cast is done, you're immediately calling the method getScreenshotAs() from the TakesScreenshot interface.

**Print all links on the page:**

* Navigate to the webpage.
* Get the list of WebElements with the TagName “a”.
* List<WebElement> links=driver.findElements(By.tagName(“a”));
* Iterate through the List of WebElements.
* Print the link text.

**public** **class** **Geeks** {

WebDriverManager.chromedriver().setup();

WebDriver driver = **new** ChromeDriver();

driver.manage().window().maximize();

driver.get("https://www.geeksforgeeks.org/");

*// Get all the available Links*

List<WebElement> links

= driver.findElements(By.tagName("a"));

*// Iterating through all the Links and printing link*

**for** (WebElement link : links) {

System.out.println(link.getText());

} driver.close(); }

**Axes Methods:**

Different XPath Axes Used In Selenium Testing

There are thirteen different axes listed below. However, we’re not going to use all of them during Selenium testing.

1. ancestor: These axes indicate all the ancestors relative to the context node, also reaching up to the root node.
2. ancestor-or-self: This one indicates the context node and all the ancestors relative to the context node and includes the root node.
3. attribute: This indicates the attributes of the context node. It can be represented with the “@” symbol.
4. child: This indicates the children of the context node.
5. descendent: This indicates the children, grandchildren, and their children (if any) of the context node. This does NOT indicate the Attribute and Namespace.
6. descendent-or-self: This indicates the context node and the children, and grandchildren and their children (if any) of the context node. This does NOT indicate the attribute and namespace.
7. following: This indicates all the nodes that appear after the context node in the HTML DOM structure. This does NOT indicate descendent, attribute, and namespace.
8. following-sibling: This one indicates all the sibling nodes (same parent as the context node) that appear after the context node in the HTML DOM structure. This does NOT indicate descendent, attribute, and namespace.
9. namespace: This indicates all the namespace nodes of the context node.
10. parent: This indicates the parent of the context node.
11. preceding: This indicates all the nodes that appear before the context node in the HTML DOM structure. This does NOT indicate descendent, attribute, and namespace.
12. preceding sibling: This one indicates all the sibling nodes (same parent as the context node) that appear before the context node in the HTML DOM structure. This does NOT indicate descendent, attribute, and namespace.
13. self: This one indicates the context node.

[**XPath Axes For Dynamic XPath In Selenium WebDriver**](https://www.softwaretestinghelp.com/xpath-axes-tutorial/) **– Read for more details**

**POM (Page Object Model):**

**What is Page Object Model in Selenium?**

Page Object Model, also known as POM, is a [design pattern in Selenium](https://www.browserstack.com/guide/design-patterns-in-selenium) that creates an object repository for storing all web elements. It helps reduce code duplication and improves test case maintenance.

In Page Object Model, consider each web page of an application as a class file. Each class file will contain only corresponding web page elements. Using these elements, testers can perform operations on the website under test.

**Advantages of Page Object Model**

* **Easy Maintenance**: POM is useful when there is a change in a UI element or a change in action. An example would be a drop-down menu is changed to a radio button. In this case, POM helps to identify the page or screen to be modified. As every screen will have different Java files, this identification is necessary to make changes in the right files. This makes test cases easy to maintain and reduces errors.
* [**Code Reusability**](https://www.browserstack.com/guide/importance-of-code-reusability): As already discussed, all screens are independent. By using POM, one can use the test code for one screen, and reuse it in another test case. There is no need to rewrite code, thus saving time and effort.
* **Readability and Reliability of Scripts**: When all screens have independent java files, one can quickly identify actions performed on a particular screen by navigating through the java file. If a change must be made to a specific code section, it can be efficiently done without affecting other files.

**1. StaleElementReferenceException**

**What is it?**

A StaleElementReferenceException occurs in Selenium when the WebElement reference you previously found is no longer valid. This can happen if:

* The page refreshes or navigates to a new page.
* The DOM is updated dynamically (e.g., JavaScript modifies the elements).
* The element is removed and re-added to the DOM

**2. NullPointerException**

**What is it?**

A NullPointerException occurs when your code tries to access an object that hasn't been initialized. In Selenium, this typically happens when:

* WebDriver is not initialized.
* The findElement() method returns null.
* The WebElement is not found in the DOM.

|  |  |
| --- | --- |
| **Page Object Model** | **Page Factory** |
| Finding web elements using **By** | Finding web elements using **@FindBy** |
| POM does not provide lazy initialization | Page Factory does provide lazy initialization |
| Page Object Model is a design pattern | PageFactory is a class that implements the Page Object Model design pattern. |
| In POM, one needs to initialize every page object individually | In PageFactory, all page objects are initialized by using the **initElements()** method |

**Read Table:**

**Sample HTML Table:**

<table id="userTable">

<thead>

<tr><th>Name</th><th>Email</th><th>Role</th></tr>

</thead>

<tbody>

<tr><td>Alice</td><td>alice@example.com</td><td>Admin</td></tr>

<tr><td>Bob</td><td>bob@example.com</td><td>Editor</td></tr>

<tr><td>Charlie</td><td>charlie@example.com</td><td>Viewer</td></tr>

</tbody>

</table>

**✅ 1. Get the second row (Bob)**

WebElement secondRow = driver.findElement(By.cssSelector("#userTable tbody tr:nth-child(2)"));

System.out.println(secondRow.getText());

// Output: Bob bob@example.com Editor

**✅ 2. Get Bob's Role only (3rd column of 2nd row)**

WebElement bobsRole = driver.findElement(By.cssSelector("#userTable tbody tr:nth-child(2) td:nth-child(3)"));

System.out.println(bobsRole.getText());

// Output: Editor

**✅ 3. Get all rows (loop through them)**

List<WebElement> allRows = driver.findElements(By.cssSelector("#userTable tbody tr"));

for (WebElement row : allRows) {

System.out.println(row.getText());

}

**✅ 4. Get every first column cell (Names only)**

List<WebElement> names = driver.findElements(By.cssSelector("#userTable tbody td:nth-child(1)"));

for (WebElement name : names) {

System.out.println(name.getText());

}

// Output: Alice, Bob, Charlie

**TestNG:** *this is a totally different framework but we are using cucumber BDD*

* TestNG is an automation testing framework widely used across many projects. NG means “Next Generation” and it is influenced by JUnit and it follows the annotations (@) – unit testing framework (used by developers)- TDD
* Testers also can use it.
* Purpose – design test cases in proper systematic way.
* Open source
* Available in the form of jar files
* Only applicable in java
* Html report generation
* Annotations
* Priorities/sequence can be set for test cases
* Define dependencies
* Define grouping
* Data provider
* No need of main class

**What are TestNG Annotations?**

The Concept Annotations is introduced in [Java](https://www.geeksforgeeks.org/java/)1.5 (jdk5). TestNG annotations are special codes we add to our test programs to decide the order in which our test methods run. These annotations give extra information about our methods or classes and start with the **'@'**symbol. They're like special symbols that we put before each method in our test code. If a method doesn't have these annotations, it won't be run when we execute our tests. TestNG uses these tags to create a strong and organized testing framework.

**Types and Hierarchy of TestNG Annotations**

These are the types of TestNG Annotations

* [@BeforeSuite](https://www.geeksforgeeks.org/testng-annotations-beforesuite/)
* [@AfterSuite](https://www.geeksforgeeks.org/testng-annotations-aftersuite/amp/)
* [@BeforeTest](https://www.geeksforgeeks.org/testng-annotations-beforetest/)
* [@AfterTest](https://www.geeksforgeeks.org/testng-aftertest-annotation/)
* [@BeforeClass](https://www.geeksforgeeks.org/testng-annotations-beforeclass/)
* [@AfterClass](https://www.geeksforgeeks.org/testng-afterclass-annotations/)
* [@BeforeMethod](https://www.geeksforgeeks.org/testng-beforemethod-annotations/)
* [@AfterMethod](https://www.geeksforgeeks.org/testng-aftermethod-annotations/)
* [@BeforeGroups](https://www.geeksforgeeks.org/testng-beforegroups-annotations/)
* [@AfterGroups](https://www.geeksforgeeks.org/testng-annotations-aftergroups/)

Let's Elaborate each one by one:

1. BeforeSuite**:** @BeforeSuite is one of the TestNG Annotations. As the name defines, @BeforeSuite is executed before the execution of all the test cases inside a TestNG Suite.
2. AfterSuite: @AfterSuite is one of the TestNG Annotations. As the name defines, @AfterSuite is executed after the execution of all the test cases inside a TestNG Suite.
3. BeforeTest**:**@BeforeTest is one of the TestNG Annotations. As the name defines, @BeforeTest is executed before the execution of all the @test annotated methods inside a TestNG Suite
4. AfterTest: @AfterTest is one of the TestNG Annotations. As the name defines, @AfterTest is executed after the execution of all the @test annotated methods inside a TestNG Suite.
5. BeforeClass: @BeforeClass is one of the TestNG Annotations. As the name defines, @BeforeClass is executed before all the methods of the current class start their execution.
6. AfterClass: @AfterClass is one of the TestNG Annotations. As the name defines, @AfterClass is executed after all the methods of the current class finish their execution.
7. BeforeMethod: @BeforeMethod is one of the TestNG Annotations. As the name itself defines, @BeforeMethod is executed before each test method within a test class. Suppose there are n test methods within a test class, then n times @BeforeMethod annotated method will be invoked.
8. AfterMethod: @AfterMethod is one of the TestNG Annotations. As the name defines, @AfterMethod is executed after each test method within a test class. Suppose there are n test methods within a test class, then n times @AfterMethod annotated method will be invoked.
9. BeforeGroups**:**@BeforeGroups is one of the TestNG Annotations. When you annotate a method with @BeforeGroups, TestNG ensures that this method is invoked before any test method belonging to the specified groups is executed.
10. AfterGroup: @AfterGroups is one of the TestNG Annotations. As the name defines, @AfterGroups should be executed after all the test methods belonging to a specified group have been run.

**1. Priority keyword**= to define priority of test case

e.g. @Test(priority=1)

Value can be +ve, -ve,0,duplicate

+ve= 1….10

-ve=-1……-10

**2. Groups keyword**= to categorize test cases into groups

e.g @Test(priority=1, groups=regression)

**3.dependsOnMethods** keyword= to provide dependency

@Test

public void LoginTest()

{//some code }

@test (dependsOnMethod=”Login Test”)

Public void HomePageTest()

{ // some code }

* If login test passed then only homepage test will be executed otherwise it will be skipped.

**4.invocationCount keyword** = to execute test cases multiple times

@Test(invocationCount=10)

public void LoginTest()

{//some code }

Will be executed 10 times

**5.expectedExeption keyword =** used to write exception that may occur in the test case execution but test case should not fail if exception occurs

e.g. @Test(expectedException=numberFormatException.class)

public void test1()

{

String a=”100H”;

Integer.parseInt(a); // this line will throw numberformatexception

}

**6.** **enabled:**

-disable test method

-@Test(enabled=false)

**7. TimeOut:**

-we are giving TimeOut to test method which is time consuming to execution

@Test(timeOut=60000)

**Test Suite:**

* It is xml file which contains all the test classes’ name which need to be executed.
* It is use to execute all/multiple Test classes.
* Syntax:

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">  <suite name=*"Suite"*>  <test name=*"Test1"*>  <classes>  <class name=*"collection.ex2"*/>  <class name=*"collection.ex3"*/>  <class name=” POM\_Pagefactory.Testclass”/>  </classes>  </test> <!-- Test -->  </suite> <!-- Suite --> |
|  |

- Steps to Create Test Suit

1. Click/Select the Classes (Multiple).

2. Right Click on selected classes Click on TestNG then Click Convert to TestNG

3. New Popup will open Rename the XML file and Click on Finish.

4. Refresh the Project (Click F5)

5. At the Bottom of Project List there is test-output folder.

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">  <suite name="Suite">  <test thread-count="5" name="Test">  <classes>  <class name="testNGKeywords.enabledKeyword"/>  <class name="testNGKeywords.priorityKeyword"/>  <class name="testNGKeywords.dependsOnMethodsKeyword"/>  <class name="testNGKeywords.invocationCountKeyword"/>  <class name="testNGKeywords.timeOutKeyword"/>  </classes>  </test> <!-- Test -->  </suite> <!-- Suite --> |

**Testing.xml**

**to execute only failed test cases in cucumber BDD**

**Interview Questions:**

**How to check if page got opened or not in selenium?**

**How to get all the values from checkbox?**

**How to check color of button in selenium? Also check if color is changing upon mouse hover ?**

**WebTable**

**Without using “select” how will you select drop down value?**

Ans: Using Actions class we can do that. Identify the element. Using the click method of Actions class click the dropdown and then click down arrows using the method sendKeys(keys.Down) and then click the desired element.

**Challenges faced in Automation:**

Important Interview question: Tell any 2

1. Screenshots taken in browsers other than “Firefox” will not cover the entire webpage. Only the area with the element of focus will be covered. Solution: Use Actions class to move (scroll) the webpage up/down and take screenshots.
2. Dynamically changing xpath: Solution: Used partial xpath to handle it.
3. Need to start scripting before the application is ready. That is we have mockup screen alone. Solution: Using Properties file concept and handled it. Once the application is ready we will simply change the code in the properties file rather than changing script.
4. Need to upload a document from my computer as part of sanity testing which would need windows application interactions. We can achieve it using tools like Sikuli or Autoit. But in client machines we should not install such tools. Solution: There is a method “Robot.awt” under “Java.AWT” package. \*Concept: store the file path using the “setContents” method to store in clipboard. Then using sendKeys method send “Ctrl+V” and “Enter.”
5. Reading contents from a PDF file. Solution: Two jar files – “PDF Box” to read PDF and “Font box” to read/analyze” the contents in the PDF. (Limitation: It cannot read images/tables. Only text will be read.)
6. Run execution in the background or Hidden mode. Solution: HTML Unit driver is used to run scripts in the background. That is we don’t want any browser to run in the foreground. [Limitation of HTML unit driver: it cannot take screenshot. Mouse hover, Key board actions cannot be performed].

**can list have duplicate records**

**Implicit and explicit wait**

**how to find hidden elements on webpage**

**how to find broken link on webpage**

**can we handle windows based popups in selenium**

**what is Scenario and step**

**how to exclude scenarios from execution**

**reporting and logging**

**what is then keyword in BDD**

**how to provide priority to feature file**

**how to do cloning in bitbucket**

**Agile ceremonies**

**how to revert the merge changes**

**What is Automation ROI?**

Automation ROI = How much time/money/effort you save or gain through automation compared to how much you spend building and maintaining it.