**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Selenium**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. **Difference in driver.get() and driver.navigate().to()**

driver.get() is used to navigate particular URL(website) and wait till page load.

driver.navigate() is used to navigate to particular URL and does not wait to page load. It maintains browser history or cookies to navigate back or forward.

1. **Navigation methods in selenium**

driver.navigate().to(URL);

driver.navigate().forward();

driver.navigate().back();

driver.navigate().refresh();

1. **What is POM?**

***Page Object Model or POM*** is a design pattern or a framework that we use in Selenium using which one can create an object repository of the different web elements across the application. To simplify, in the Page Object Model framework, we create a class file for each web page. This class file consists of different web elements present on the web page. Moreover, the test scripts then use these elements to perform different actions. Since each page's web elements are in a separate class file, the code becomes easy to maintain and reduces code duplicity.

1. **Implicit wait**

Implicit Wait directs the Selenium WebDriver to wait for a certain measure of time before throwing an exception. Once this time is set, WebDriver will wait for the element before the exception occurs. Once the command is in place, Implicit Wait stays in place for the entire duration for which the browser is open. It’ s default setting is 0.

SYNTAX: driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

1. **Explicit wait- use of explicit wait, how to ignore exceptions using explicit wait**

* By using the Explicit Wait command, the WebDriver is directed to wait until a certain condition occurs before proceeding with executing the code.
* Setting Explicit Wait is important in cases where there are certain elements that naturally take more time to load. If one sets an implicit wait command, then the browser will wait for the same time frame before loading every web element. This causes an unnecessary delay in executing the test script.
* Explicit wait is more intelligent, but can only be applied for specified elements. However, it is an improvement on implicit wait since it allows the program to pause for dynamically loaded Ajax elements.

In order to declare explicit wait, one has to use “ExpectedConditions”. The following Expected Conditions can be used in Explicit Wait.

* alertIsPresent()
* elementSelectionStateToBe()
* elementToBeClickable()

SYNTAX:

WebDriverWait wait = new WebDriverWait(driver,30);

wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath("//div[contains(text(),'COMPOSE')]")));

1. **Fluent wait**

* Fluent Wait in Selenium marks the maximum amount of time for Selenium WebDriver to wait for a certain condition (web element) becomes visible. It also defines how frequently WebDriver will check if the condition appears before throwing the “**ElementNotVisibleException**”.
* To put it simply, Fluent Wait looks for a web element repeatedly at regular intervals until timeout happens or until the object is found.
* Fluent Wait commands are most useful when interacting with web elements that can take longer durations to load. This is something that often occurs in Ajax applications.
* While using Fluent Wait, it is possible to set a default polling period as needed. The user can configure the wait to ignore any exceptions during the polling period.
* Fluent waits are also sometimes called smart waits because they don’t wait out the entire duration defined in the code. Instead, the test continues to execute as soon as the element is detected – as soon as the condition specified in .**until(YourCondition)** method becomes true

SYNTAX:

//Declare and initialise a fluent wait

FluentWait wait = new FluentWait(driver);

//Specify the timout of the wait

wait.withTimeout(5000, TimeUnit.MILLISECONDS);

//Sepcify polling time

wait.pollingEvery(250, TimeUnit.MILLISECONDS);

//Specify what exceptions to ignore

wait.ignoring(NoSuchElementException.class)

//This is how we specify the condition to wait on.

//This is what we will explore more in this chapter

wait.until(ExpectedConditions.alertIsPresent());

This command operates with two primary parameters: timeout value and polling frequency. The above code defines time out value as 50 seconds and polling frequency as 3 seconds. It directs WebDriver to wait for no more than 50 seconds to verify a specific condition. If the condition occurs during those 50 seconds, it will execute the next step in the test script. If not, it will return “**ElementNotVisibleException**”.

**A few other associated commands are:**

* **PageLoadTimeout Command**

This command establishes the time WebDriver must wait for a page to completely load before triggering an error. In case the timeout set is negative, the page load time can be indefinite.

**Syntax:**

driver.manage().timeouts().pageLoadTimeout(100, SECONDS);

* **SetScriptTimeout Command**

This command establishes the time WebDriver will wait for an asynchronous script to finish executing before triggering an error. Like the previous command, the script will run indefinitely if the timeout is set to a negative value.

**Syntax:**

driver.manage().timeouts().setScriptTimeout(100,SECONDS);

* **Sleep Command**

The Sleep command is rarely used because it is quite ineffective. It causes WebDriver to wait for a specific time (and does not let it run faster even if the specified condition is met). In fact, Selenium wait commands are considered the smarter, more effective alternative to the Sleep command.

**Syntax:**

thread.sleep(1000);

1. **Examples for expected conditions in explicit wait**

. The following Expected Conditions can be used in Explicit Wait.

* alertIsPresent()
* elementSelectionStateToBe()
* elementToBeClickable()
* elementToBeSelected()
* frameToBeAvaliableAndSwitchToIt()
* invisibilityOfTheElementLocated()
* invisibilityOfElementWithText()
* presenceOfAllElementsLocatedBy()
* presenceOfElementLocated()
* textToBePresentInElement()
* textToBePresentInElementLocated()
* textToBePresentInElementValue()
* titleIs()
* titleContains()
* visibilityOf()
* visibilityOfAllElements()
* visibilityOfAllElementsLocatedBy()
* visibilityOfElementLocated()

1. **Default timeout of pageLoadTimeout.**

The page load timeout is set to**-1** by default. This means that Selenium will wait indefinitely for the page to load. The WebDriver specification, which was derived from Selenium has settled on the following values: For implicit waits: 0 seconds.

1. **Use of desired capabilities.**

The RemoteWebdriver is used to instantiate the webdriver instance.

The DesiredCapabilities object is used to set preferences about the browser.for example: chrome or Firefox

DesiredCapabilities cap = DesiredCapabilities.internetExplorer();

cap.setBrowserName("internet explorer");

String Node ="http://10.82.189.5:5558/wd/hub";

driver = new RemoteWebDriver(new URL(Node), cap);

**10. Use of chromeOptions, firefoxOptions ,ieOptions, etc.**

The **Chromeoptions Class** is a concept in Selenium WebDriver for manipulating various properties of the Chrome driver. The Chrome options class is generally used in conjunction with Desired Capabilities for customizing Chrome driver sessions. It helps you perform various operations like opening Chrome in maximized mode, disable existing extensions, disable pop-ups, etc. Or to accept ssl certificates

Below are the list of available and most commonly used arguments for ChromeOptions class

* **start-maximized**: Opens Chrome in maximize mode
* **incognito:**Opens Chrome in incognito mode
* **headless:** Opens Chrome in headless mode
* **disable-extensions**: Disables existing extensions on Chrome browser
* **disable-popup-blocking**: Disables pop-ups displayed on Chrome browser
* **make-default-browser:** Makes Chrome default browser
* **version**: Prints chrome browser version
* **disable-infobars:** Prevents Chrome from displaying the notification ‘Chrome is being controlled by automated software

**11. How to disable notification in selenium**.

Create an object of ChromeOptions class

ChromeOptions options = **new** ChromeOptions();

Call the method addArguments()

options.addArguments("disable-infobars");

Pass the object of ChromeOptions class in ChromeDriver.

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

**12. Difference between getText() and getAttribute()**

the getText () method gives the text present between the start and end html tags (which is not hidden by CSS) and the getAttribute () method identifies the key value pairs within the html tags.

getAttribute (“text”) on title returns the title .

**13. How to extract css property value**

driver.findElement(bySearchButton).getCssValue("font-size"));

**14. Difference between driver.close() and driver.quit()**

**driver.close() will close only current window/tab. But driver.quit() will exit all tabs.**

• Driver.close(): It is used to close the browser or page currently in focus.

close() is a webdriver command which closes the browser window which is currently in focus.

During the automation process, if there are more than one browser window opened, then the close() command will close only the current browser window which is having focus at that time. The remaining browser windows will not be closed. The following code can be used to close the current browser window:

driver.close() //where, ‘driver’ is the Webdriver object.

• Driver.quit() : It is used to shut down the webdriver and close all instances associated to webdriver.

You should use driver.quit whenever you want to end the program. It will close all opened browser window and terminates the WebDriver session. If you do not use driver.quit at the end of program, WebDriver session will not close properly and files would not be cleared off memory. This may result in memory leak errors.

**15. How to handle dropdown elements with select tag and without select tag.**

Select select=new Select(WebElement);

**16. How to handle frames and windows**

To move back to the parent frame, you can either use **switchTo ().parentFrame () or**

use **switchTo ().defaultContent() to switch to main frame.**

**17. How to switch back to parent window.**

Create one parentWindowHandle String variable before for loop and store windowhandle there. Once you are done with your operation with child window, go back to that parentWindowHandle.

String parentWindowHandle = driver.getWindowHandle();

/\* You code to move to child window\*/

//After you done with child window

driver.switchTo().window(parentWindowHandle);

**18. How to perform page scroll in selenium**

Using javascript executor

JavascriptExecutor js=(JavascriptExecutor)driver;

js.executeScript("window.scrollBy(0,400)"); // to scroll by y axis.

//to scroll to a particular element

WebElement ele=driver.findElement(By.*xpath*("//\*[@id=\"app\"]/div/div/div[2]/div[1]/div/div/div[6]/span/div"));

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

**19. Use of Robot keys**

**Robot Class** in Selenium is used to enable automated testing for implementations of Java platform. It generates input events in native systems for test automation, self-running demos and other applications where users need control over mouse and keyboard. Robot class is easy to implement and it can be easily integrated with an automated framework.

## **Why Robot Class?**

**Robot Class** is used in Selenium because, in certain Selenium automation tests, users need control over keyboard or mouse to interact with OS windows like download pop-ups, print pop-ups, etc. and native applications like notepad, calculator, etc. Selenium Webdriver cannot handle these pop-ups/applications, so in[Java](https://www.guru99.com/java-tutorial.html)version 1.3, robot class was introduced which can handle OS pop-ups/applications.

## **Benefits of Robot Class**

1. Robot Class can simulate Keyboard and Mouse Event
2. Robot Class can help in upload/download of files when using selenium web driver
3. Robot Class can easily be integrated with current automation framework (keyword, data-driven or hybrid)

**Some commonly and popular used methods of Robot Class during web automation:**

* keyPress(): **Example:** robot.keyPress(KeyEvent.VK\_DOWN) : This method with press down arrow key of Keyboard
* mousePress() : **Example** : robot.mousePress(InputEvent.BUTTON3\_DOWN\_MASK) : This method will press the right click of your mouse.
* mouseMove() : **Example**: robot.mouseMove(point.getX(), point.getY()) : This will move mouse pointer to the specified X and Y coordinates.
* keyRelease() : **Example:** robot.keyRelease(KeyEvent.VK\_DOWN) : This method with release down arrow key of Keyboard
* mouseRelease() : **Example:**robot.mouseRelease(InputEvent.BUTTON3\_DOWN\_MASK) : This method will release the right click of your mouse

**20. Use of AutoIt**

***AutoIt*** v3 is a freeware BASIC-like scripting language designed for automating the Windows GUI and general scripting. It uses a combination of simulated ***keystrokes, mouse movement and window/control*** manipulation in order to automate tasks in a way not possible or reliable with other languages (e.g. VBScript and SendKeys). AutoIt is also very small, self-contained and will run on all versions of Windows out-of-the-box with no annoying “***runtimes***” required!

In layman's term AutoIt is just another automation tool like Selenium but unlike Selenium it is used for Desktop Automation rather Web Automation.  It is a powerful tool and it just not automate desktop windows, button & form, it automates mouse movements & keystrokes too. Just like Selenium IDE, it also gives you the recording capability which generates the scripts for you to use the same script in you test.

## AutoIt Features

1. ***Easy to learn***: It is just another scripting language and very easy to use. Under help menu of AutoIt it gives you all the functions and methods you can use with detailed explanation and examples.
2. ***Simulates keystrokes***: Where ever it is required to use keystrokes in your test, you can use this for example pressing enter on any dialog box and typing username and password on the popup which you cannot simulate with Selenium.
3. ***Simulate mouse movements***: Like keystrokes there can be situations when you are required to simulate the mouse movements and it is the easiest way out for those situations.

4) ***Scripts can be compiled into standalone executable***: It means that you do not require any IDE to run your scripts, you can easily convert your automation scripts into .exe files which can be run on their own.

1. ***Windows Management***: You can expect to move, hide, show, resize, activate, close and pretty much do what you want with windows. Windows can be referenced by title, text on the window, size, position, class and even internal Win32 API handles.
2. ***Windows Controls***: Directly get information on and interact with edit boxes, check boxes, list boxes, combos, buttons, status bars without the risk of keystrokes getting lost.  Even work with controls in windows that aren’t active!
3. ***Detailed help file and large community-based support forums***: You think of any action on windows, you will get it on help file. You face any issue or get stuck anywhere, the large group of users are there to help you.

In short, any windows,  mouse & keystrokes simulation which we cannot handle with Selenium that can be handled with AutoIt. All we need to do is to use the script in Selenium which is generated with the help of AutoIt tool.

**21. Use of Sikuli**

**22. How to perform File Upload.**

// identify element

WebElement m=*driver*.findElement(By.*xpath*("//input[@name='photo']"));

// file selection field with file path

m.sendKeys("C:\\Users\\sahil.juneja01\\Desktop\\error.png");

Using send keys(path of file)

**23. How to click or send value to an element without using element.click() and element.sendKeys()**

Using javascript executor

**24. Relative and absolute xpath**

**25. Explain few Xpath functions**

1. contains() [By text]
2. starts-with() [By text]
3. text()

The **XPath text () function is a built-in function of selenium webdriver which is used to locate elements based on text of a web element**

* //h1[contains(text(),’ Log in to’)]
  + //h1[starts-with(text(),’Log in’)]
  + //h1[text()=’Log Ino Twitter’]

**26. How to login to site showing authentication popup in selenium**

**Approach 1: Handling Authentication/Login Popup Window using Selenium WebDriver**

By passing user credentials in URL. Its simple, append your username and password with the URL.

e.g., http://Username:Password@SiteURL

http://rajkumar:myPassword@www.softwaretestingmaterial.com

here, Username is rajkumar  
Password is myPassword  
SiteURL is www.softwaretestingmaterial.com

Sample code:

String URL = "http://" + rajkumar + ":" + myPassword + "@" + www.softwaretestingmaterial.com;

driver.get(URL);

Alert alert = driver.switchTo().alert();

alert.accept();



### ****Approach 2: Handling Authentication/Login Popup Window using Selenium WebDriver****

By using AutoIt, we could handle authentication pop up.

### ****Approach 3: Handling Authentication/Login Popup Window using Selenium WebDriver****

By using Alerts in Selenium, we could handle authentication pop up.

|  |  |
| --- | --- |
| 3  4  5  6 | driver.switchTo().alert();  //Selenium-WebDriver Java Code for entering Username & Password as below:  driver.findElement(By.id("userID")).sendKeys("userName");  driver.findElement(By.id("password")).sendKeys("myPassword");  driver.switchTo().alert().accept();  driver.switchTo().defaultContent(); |

**27. Mention few selenium exceptions**

* ***TimeoutException****: Thrown when there is not enough time for a command to complete*.
* ***UnableToCreateProfileException****: You can open a browser with specific options using profiles, but sometimes a new version of Selenium Driverserver or browser may not support the profiles*.
* ***UnableToSetCookieException:****This occurs if a driver is unable to set a cookie*.
* ***UnexpectedAlertPresentException:****This Selenium exception happens when there is the appearance of an unexpected alert*.
* ***UnexpectedTagNameException:****Happens if a support class did not get a web element as expected*.
* ***InvalidCookieDomainException****: This happens when you try to add a cookie under a different domain rather than the current URL*.
* ***InvalidCoordinatesException:****This happens if the coordinates offered to an interacting operation are not valid*.
* ***InvalidElementStateException****: This Selenium exception occurs if a command cannot finish as the element is invalid*.
* ***InvalidSessionIdException****: Takes place when the given session ID does not include in the list of active sessions, which means the session does not exist or is inactive either.*
* ***InvalidSwitchToTargetException****: Happens if a frame or window target to switch does not exist*.
* ***ElementClickInterceptedException****: The command could not complete as the element receiving the events is concealing the element which was requested clicked*.
* ***ElementNotInteractableException****: This Selenium exception gets thrown when an element is present in the DOM, but it is impossible to interact with such an element*.
* ***NoAlertPresentException:****Happens when you switch to no presented alert*.
* ***NoSuchAttributeException****: This occurs when we can’t find the attribute of the element*.
* ***NoSuchContextException****: Happens in mobile device testing and is thrown by ContextAware*.
* ***NoSuchCookieException****: This exception gets thrown if there is no cookie matching with the given pathname found amongst the associated cookies of the current browsing context’s active document*.
* ***NoSuchElementException****: Happens if we can't find an element*.
* ***NoSuchFrameException****: Takes place if frame target to be switch does not exist*.
* ***NoSuchWindowException****: This occurs if the window target to be switch does not exist.*
* ***NotFoundException:****This exception is a subclass of WebDriverException. It happens when an element on the DOM doesn't exist*.

**28. Difference between single slash and double slash.**

* Single slash represents immediate child nodes while double slash represents any nodes. For example: /div/input represents immediate input nodes inside/of div node. ...
* Absolute xpath uses single slash while relative xpath can use both single slash or double slash.

**29. What are the scenarios that cannot be automated using selenium.**

Captcha testing, image verification, window based app can’t be tested using selenium

**30. How to perform right click on selenium**

Using contextClick method of action class

**31. Action class and it's methods**

Action Class in Selenium is a built-in feature provided by the selenium for handling keyboard and mouse events. It includes various operations such as **multiple events clicking by control key, drag and drop events** and many more. These operations from the action class are performed using the advanced user interaction API in Selenium Webdriver

|  |  |
| --- | --- |
| **clickAndHold()** | Clicks (without releasing) at the current mouse location. |
| **contextClick()** | Performs a context-click at the current mouse location. (Right Click Mouse Action) |
| **doubleClick()** | Performs a double-click at the current mouse location. |
| **dragAndDrop(source, target)** | Performs click-and-hold at the location of the source element, moves to the location of the target element, then releases the mouse.  **Parameters:**  source- element to emulate button down at.  target- element to move to and release the mouse at. |
| **dragAndDropBy(source, x-offset, y-offset)** | Performs click-and-hold at the location of the source element, moves by a given offset, then releases the mouse.  **Parameters**:  source- element to emulate button down at.  xOffset- horizontal move offset.  yOffset- vertical move offset. |

**32. Use of HtmlUnit driver**

**It is basically used to run selenium testcases in headless mode**

HTML UnitDriver is the most light weight and fastest implementation headless browser for of WebDriver. It is based on HtmlUnit. It is known as **Headless Browser Driver**. It is same as Chrome, IE, or FireFox driver, but it does not have GUI so one cannot see the test execution on screen.

## Benefits of Html Unit Driver:

* Since it is not using any GUI to test, your tests will run in background without any visual interruption
* Compared to all other instances execution is faster
* To run your tests through HtmlUnit driver you can also select other browser versions
* It is platform independent and easier to run several tests concurrently.

**33. Can we create an object of the WebDriver() in selenium?**

Selenium WebDriver **does not** offer an in-built object repository by default. However, object repositories can be built using the key-value pair approach wherein the key refers to the name given to the object and value refers to the properties used to uniquely identify an object within the web page.

**34. What is stale element exception and how to handle it.**

The Stale Element Reference Exception is one of the infamous exceptions in selenium. This exception is caused when: the element is deleted completely from the DOM. element is no longer associated to DOM.

**35. Xpath and CSS selector**

**Both are the locators used to uniquely identify the static and dynamic webelements on webpage.**

The format of xpath is //tagname[@attribute='value'] while the format of css selector is tagname[attribute='value']. We can traverse both forward and backward in DOM, i.e we can move from parent to child element and also from child to the parent element with xpath.

However for css, we can only traverse from parent to child and not vice-versa.

**36. What is the return type of window handle method?**

Set of Strings

**37. When will you get a null point exception error?**

NullPointerException is thrown when program attempts to use an object reference that has the null value.

**Manually Created**

**1. Different types of data source in Selenium.**

(excel, csv, text file, json)

**2. What are listeners in selenium?**

Webdriver listener and TestNg Listener .

1. **How to Take SS in selenium?**

Interface name: TakeScreenshot and function name: getScreenshotAs

Ans: File sourceFile=((TakeScreenshot)driver).getScreenshotAs(outputType.FILE);

**4. Different types of exceptions face in selenium webdriver.?**

Ans: Exceptions in Selenium are similar to exceptions in other programming languages.

The most common exceptions in Selenium are:

Timeout, NOsuchelementnotfound, nosuchAlertPresent ,nosuchwindow exception

TimeoutException: This exception is thrown when a command performing an operation

does not complete in the stipulated time.

NoSuchElementException: This exception is thrown when an element with given attributes is not found on the web page.

ElementNotVisibleException: This exception is thrown when the element is present in DOM (Document Object Model), but not visible on the web page.

StaleElementException: This exception is thrown when the element is either deleted or no longer attached to the DOM.

We have StaleElementReferenceException in Selenium webdriver. As the name suggests, the word stale refers to something which is not new and perished. There may be a scenario in which an element which was present in DOM previously is now no longer available due to modification in DOM.

In such a condition, if we try to access that element then StaleElementReferenceException is thrown. This type of exception is encountered due to the below reasons −

The element is not present in the DOM any more.

The element has been removed totally.

1. **How do we check if the page is successfully loaded.?**

using element.isvisible().

1. **How many types of webdriver API are available in selenium?**

chrome, firefox, IE, and edge, Iphone driver, remotewebdriver, android driver, htmlunit driver.

**7. How can we launch a batch file in selenium webdriver project?**

**8. How to run selenium test case via Command line?**

**9. POM? in Detail**

Ans:

**10. Three steps in POM?** Declaration, initialization (in constructor using page factory) and utilization

initialize element using page factory

stale reference exception in POM

**11.What are the different types of WAIT statements in Selenium WebDriver? Or the question can be framed like this: How do you achieve synchronization in WebDriver?**

Ans: speed of selenium is faster than browser, so we use implicit and explicit wait in selenium to achieve synchronization.

**12. What is Page Factory?**

Page Factory is a **class provided by Selenium WebDriver to implement the Page Object Model**. The Page Object Repository is separated from the Test Methods using the Page Factory concept. Using it, you can initialize the Page Objects or directly instantiate them.

***Page Factory in Selenium projects***:

1. ***Using the @FindBy annotation****- Unlike the regular approach of initializing web page elements using****[FindElement](https://www.toolsqa.com/selenium-webdriver/find-element-selenium/)****or FindElements, the Page Factory uses the @FindBy annotation. The annotations used in Page Factory are descriptive. Moreover, they help improve code readability, which we will discuss in the next section. It provides the following syntax to locate the web elements:*

@FindBy(id="userName")

WebElement username;

1. ***Initializing the elements using initElements()****- This is a static method used to initialize the web elements that we locate using the****@FindBy****or other annotation(s), thereby instantiating the page class*.

PageFactory.initElements(WebDriver driver, java.lang.Class.pageObjectClass);

POM Implementation without page factory: uses BY(), no imports needed, no cache storage.

POM Implementation with page factory: uses @FindBy(), Import package: Page factory, cache lookup is faster.

**Robot Class**: It is similar to Action class in selenium, but it provides the control over mouse and keyboard devices.

**13.How can you handle network latency in selenium:**

Ans: using driver.manage.pageLoadingTime();

**14.How to capture server-side logs in selenium?**

Ans: go to cmd and type java-jar.jar-log selenium.log

**15. Write Ablsolute xpath and relative xpath**

**16. How do click on an element which is hidden using selenium web driver?**

Ans: ElementNotVisibleException is thrown by Selenium while handling hidden elements.

So JavaScript Executors can be utilized to access and handle these elements. The executeScript method is used in Selenium to

execute the JavaScript commands. The commands to be executed are passed as parameters to that method.

// Javascript executor class

JavascriptExecutor js = (JavascriptExecutor) driver;

// identify element and set value

js.executeScript

("document.getElementById('displayed−text').value='QA';");

String str = (String) js.executeScript("return document.getElementById('displayed−text').value");

System.out.print("Value entered in hidden element: " +str);

**17. We can submit a form in Selenium webdriver even if the submit button cannot be identified.**

This can be achieved by locating any element within the form tag and the applying submit method on it.

**18. How to click Allow on Show Notifications popup using Selenium Webdriver?**

This is done with the help of the ChromeOptions class. We shall create an object of it and apply the

addArguments method on it. Then pass --disable-notifications as a parameter to the method.

ChromeOptions p = new ChromeOptions();

p.addArguments("--disable-notifications");

**19.How to perform double click on an element in Selenium?**

We can perform double click on elements in Selenium with the help of Actions class. In order to perform

the double click action we will use moveToElement() method, then use doubleClick() method. Finally use build().perform()

to execute all the steps.

// doubleClick() method for double click to an element after moving to

//element to with the moveToElement()

Actions a = new Actions(driver);

a.moveToElement(driver.findElement(By.xpath(“input[@type=’text’]))).

doubleClick().

build().perform();

**20. Right click in Selenium**

We can perform right click using Selenium ChromeDriver. On right clicking on a webelement, the context menu gets displayed.

For example, if we right click on a text area, an additional menu with several options come up.

Actions class in Selenium is responsible for simulating this mouse operation. The Actions class provides the contextClick() method to carry out this action and select any option from the context menu.

To perform this entire operation, we shall first move the mouse to the middle of the element with moveToElement() method, then apply contextClick() method. Furthermore, we have to execute the build() method to execute the two actions and then perform() method to actually perform the actions.

// Actions class with moveToElement() and contextClick()

Actions a = new Actions(driver);

a.moveToElement(l).contextClick().build().perform();

**21. The scrolling down to a specific pixel.**

JavascriptExecutor j = (JavascriptExecutor) driver;

// scroll down by 1500 pixel with coordinates 0 and 1500 in x, y axes

j.executeScript("window.scrollBy(0,1500)");

The scrolling down to the bottom of the page.

JavascriptExecutor j = (JavascriptExecutor) driver;

// scroll down the bottom of page

js.executeScript("window.scrollTo(0, document.body.scrollHeight)")

// scroll to particular element

WebElement terms = driver.findElement(By.linkText("Terms of use”));

// scroll down the web element for viewing

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

driver.close();