KASPER
ANALYTICS

kasperanalytics.com

9123820085

Selenium

hr@kasperanalytics.com

kasper-analytics

Selenium

What is Selenium?

Selenium is a widely used open-source automation testing tool primarily used for automating web applications. It provides a suite of tools for automating web browsers across various platforms.

What are the different components of Selenium?

Selenium WebDriver – Selenium WebDriver is used to automate web applications by directly calling the browser's native methods.

The Selenium IDE Plugin – Selenium IDE is an open-source test automation tool that works on record and playback principles.

Selenium Grid – Allows Selenium tests to run in parallel across multiple machines.

What is Selenium WebDriver?

Selenium WebDriver is a popular open-source library and a key component of the Selenium automation framework used to automate testing for web applications. It is a collection of APIs which leverages a programming interface for developers and testers to write scripts in various programming languages such as Java, JavaScript, C#, Python, etc. to automate web browser's action and retrieve information from web pages.

Through test scripts, WebDriver simulates user actions, navigates through web pages, interacts with elements (such as button, text, dropdown menu, forms, links, etc), submit forms, perform validations,

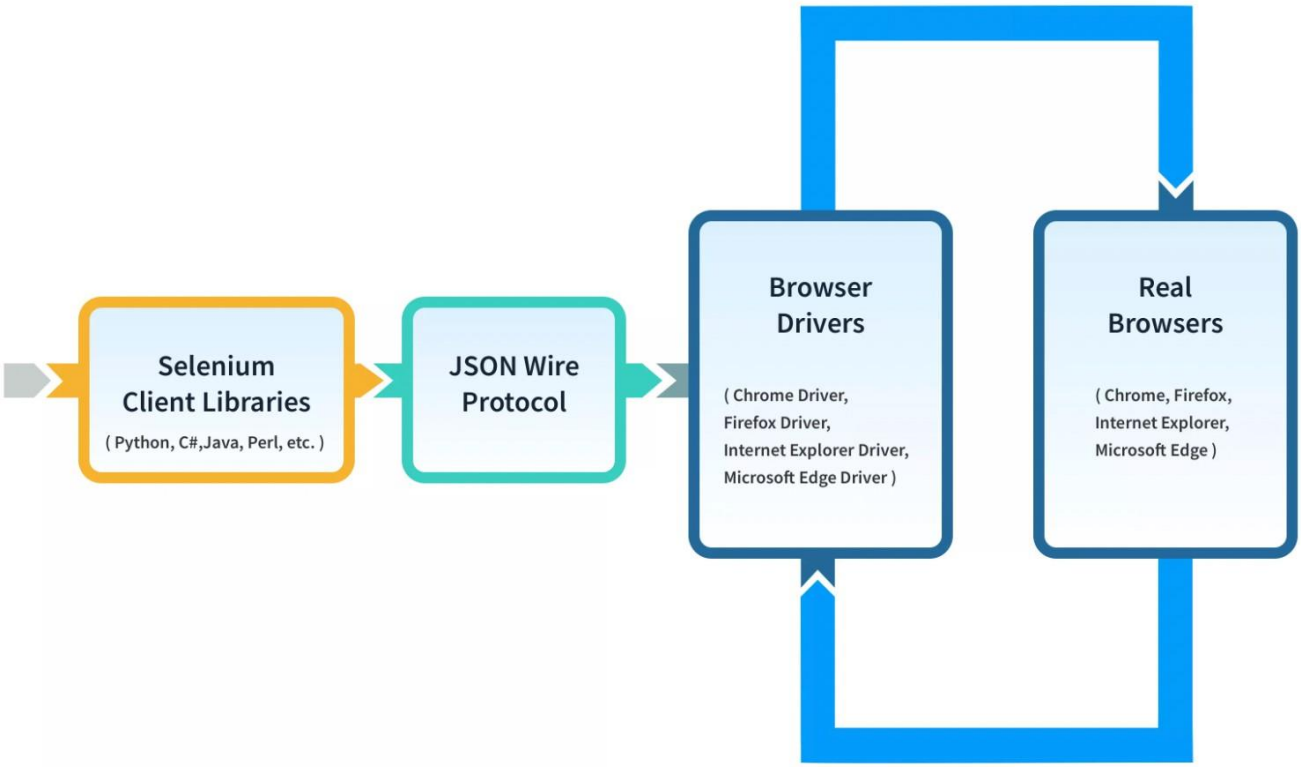
What is Architecture of Selenium WebDriver?

Architecture of Selenium WebDriver (Selenium 3)

- Selenium WebDriver Architecture is made up of four major components:

1. **Selenium Client library**: Selenium provides support to multiple libraries such as Ruby, Python, Java, etc as language bindings
2. **JSON wire protocol over HTTP**: JSON is an acronym for JavaScript Object Notation. It is an open standard that provides a transport mechanism for transferring data between client and server on the web.
3. **Browser Drivers**: Selenium browser drivers are native to each browser, interacting with the browser by establishing a secure connection. Selenium supports different browser drivers such as Chrome Driver, Gecko Driver, Microsoft Edge WebDriver, Safari Driver, and Internet Explorer Driver.
4. **Browsers**: Selenium provides support for multiple browsers like Chrome, Firefox, Safari, Internet Explorer etc.

Below diagram depicts Selenium 3 WebDriver Architecture:

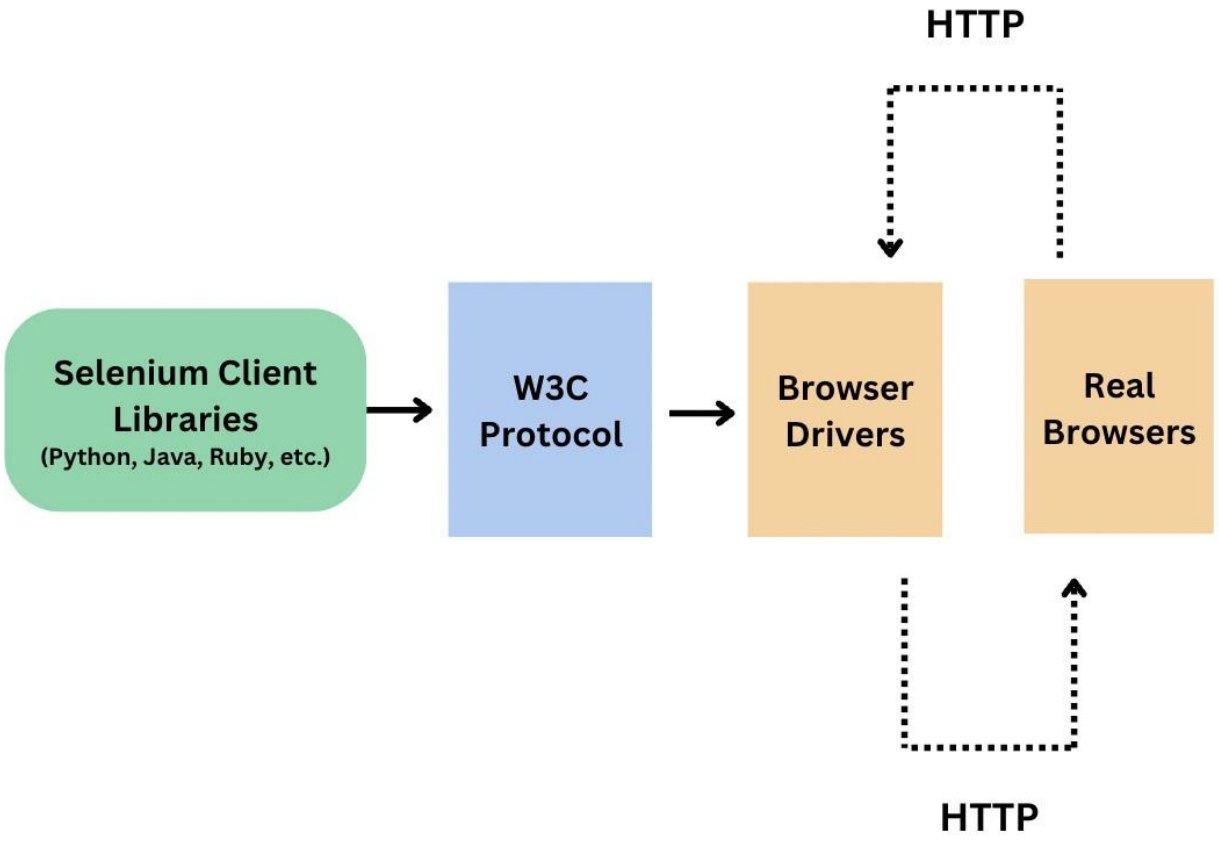


Architecture of Selenium 4 WebDriver

- The architecture of Selenium 4 is similar to Selenium 3, however it uses W3C protocol instead of JSON wire protocol for communication between Client Libraries and Browser Drivers.
- W3C stands for the World Wide Web Consortium, an international community that develops and maintains standards and guidelines for the World Wide Web. The main aim of the W3C is to ensure the long-term growth and interoperability of the Web.

- It creates open standards and specifications that promote compatibility and consistency across various web technologies and platforms. And when we say Selenium 4 is W3C compliant it states that Selenium adheres to the standards and specifications laid by the W3C for web automation.
- All the browsers and the browser drivers in Selenium architecture follow W3C, except Selenium 3 WebDriver. And hence, JSON Wire Protocol is used to encode and decode the requests and responses. Selenium 4 WebDriver was made W3C compliant to make the communication easy and direct between the client libraries and the browser drivers. Improved communication led to more stability.
- This has also enhanced browser compatibility, performance and efficiency as there is no overhead of HTTP requests and responses for communication between the WebDriver client and the browser driver. Instead, WebDriver now utilises native browser communication channels and protocols

Below diagram depicts Selenium 4 WebDriver architecture:



What are some advantages of Selenium?

- Following are the advantages of Selenium-

1. Selenium is open source and free to use without any licensing cost.
2. It supports multiple languages like Java, Ruby, Python, etc.
3. Selenium supports multi-browser testing.
4. It has vast resources and helping-community over the internet.
5. Using the Selenium IDE component, non-programmers can also write

automation scripts.

6. Using the Selenium Grid component, distributed testing can be carried out on remote machines.

What are some limitations of Selenium?

- Following are the limitations of Selenium–

1. We cannot test desktop applications using Selenium.
2. We cannot test web services using Selenium.
3. Programming language knowledge is required for creating robust scripts in Selenium Webdriver.
4. Also, we have to rely on external libraries and tools for performing tasks like – logging(log4J), testing framework-(TestNG, JUnit), reading from external files (POI for excels), etc.

Which browsers/drivers are supported by Selenium Webdriver?

> Some commonly used browsers supported by Selenium are-

- Google Chrome ChromeDriver
- Firefox – FireFoxDriver
- Internet Explorer InternetExplorerDriver
- Safari – SafariDriver

- HtmlUnit (Headless browser) – HtmlUnitDriver
- Android – Selendroid/Appium
- IOS – ios-driver/Appium

What is Selenium 4 and how is it different from other Selenium versions?

Selenium 4 is the latest version of Selenium that is W3C (World Wide Web Consortium) compliant. In simple words, this makes the Selenium test suites more stable and reduces compatibility issues across different Web browsers.

Those who have used Selenium 3 and lower version would know that a Selenium test uses JSON wire protocol to communicate with web browsers. In the case of Selenium 4 there is no need for encoding and decoding the API requests using the JSON wire protocol for communication between browsers and test scripts. This allows the WebDriver to interact directly with the target browser.

What are some features of Selenium 4?

- The different features of Selenium 4 are-

It is W3C compliant. This makes the cross-browser tests more stable.

A new Selenium 4 IDE for both Chrome and Firefox is introduced.

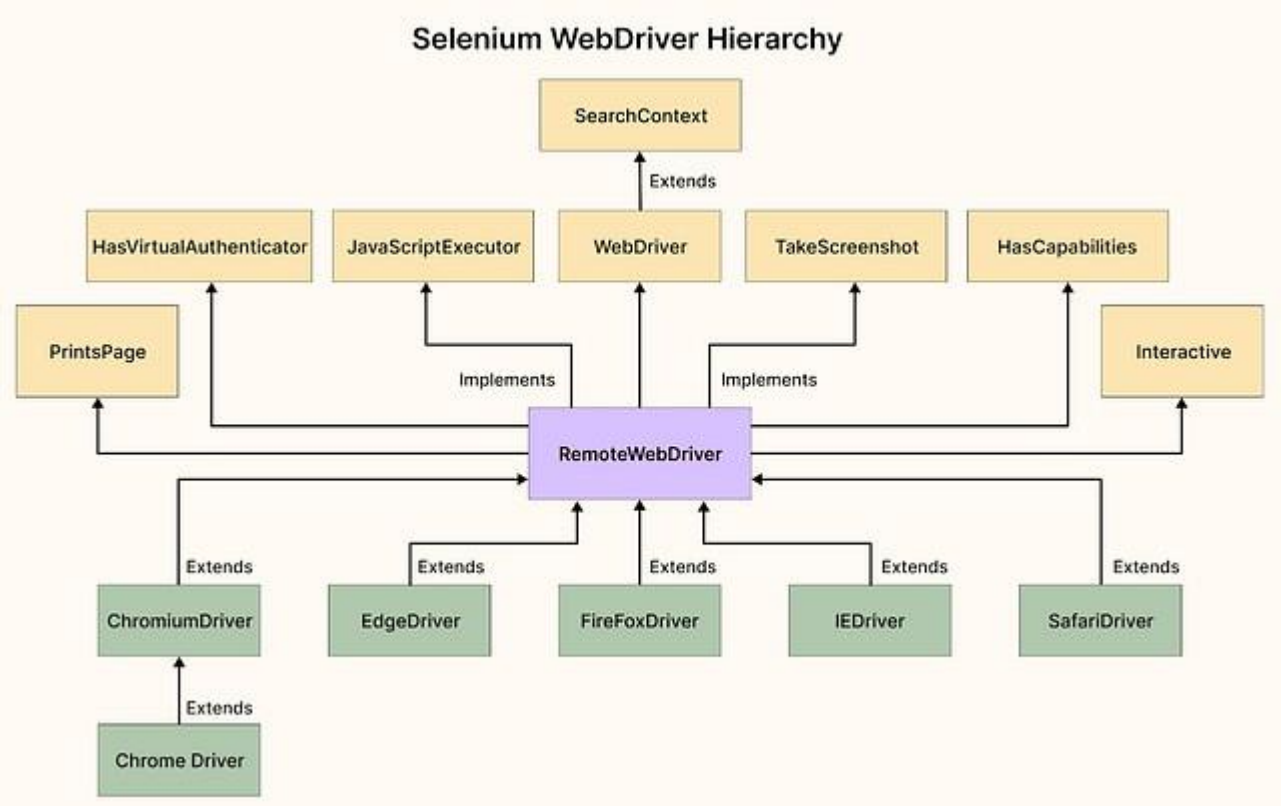
The Selenium 4 IDE tests can be exported to desired programming languages – C#, Java, Javascript, etc.

The Selenium Grid feature is more user-friendly and comes with docker support (a set of platform-as-a-service products that use OS-level virtualization to deliver software in packages called containers).

> Documentation is more detailed and improved in Selenium 4.

WebDriver Hierarchy

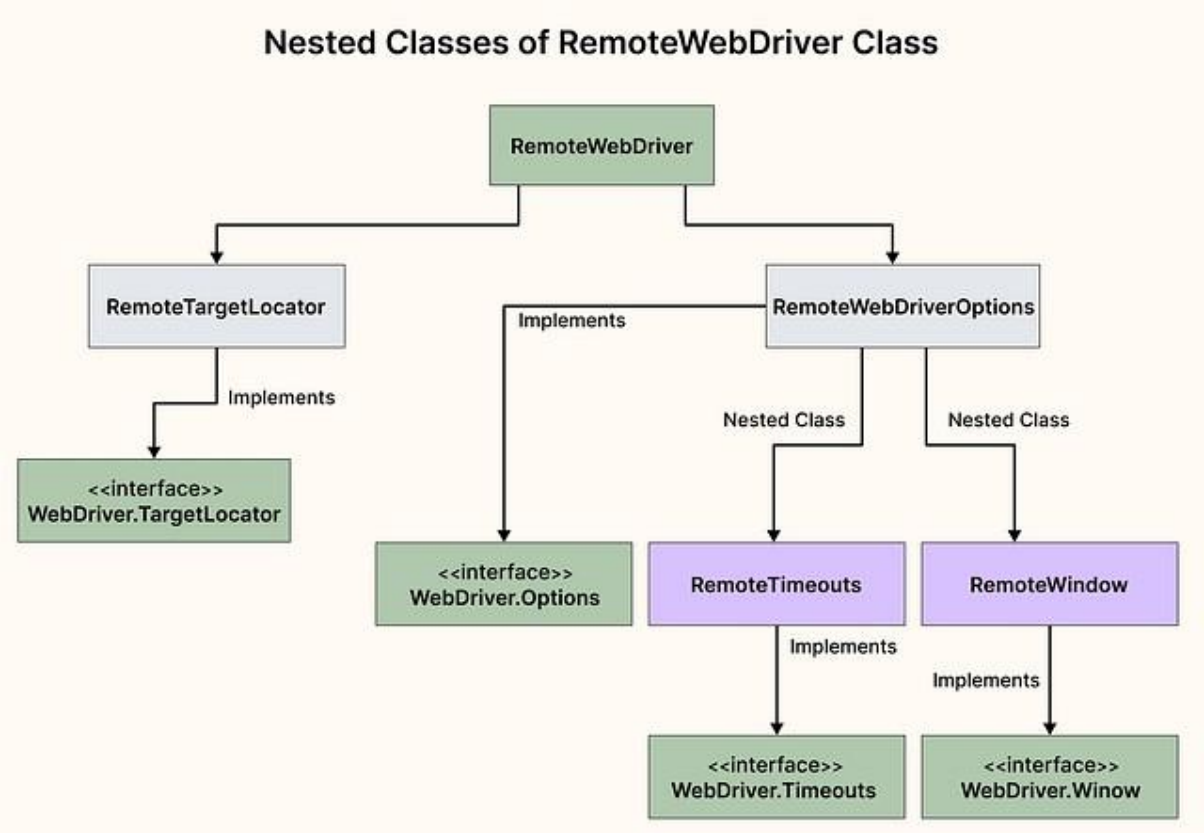
- Here is the pictorial representation of the **Selenium WebDriver hierarchy.**



Remote Web Driver Class

```
6 inheritors
@Augmentable
public class RemoteWebDriver implements WebDriver,
  JavascriptExecutor,
  HasCapabilities,
  HasVirtualAuthenticator,
  Interactive,
  PrintsPage,
  TakesScreenshot {
```

Let's start with the RemoteWebDriver class because it is a fully implemented WebDriver Interface class extended by every BrowserDriver class within the Selenium framework.



RemoteWebDriver class has the following nested classes:

- **RemoteTargetLocator** — This is a fully implemented class of WebDriver.TargetLocator interface.

- **RemoteWebDriverOptions** — This is a fully implemented class of WebDriver.Options interface. This class has the following nested classes:

- **RemoteTimeouts** — This class implements WebDriver.Timeouts interface and provides the full implementation of all its abstract methods.

- **RemoteWindow** — This class implements WebDriver.Window interface and provides the full implementation of all its abstract methods.

Selenium WebDriver Interface Methods

SearchContext is the topmost interface in Selenium API which has two methods - **findElement() and**

Selenium WebDriver interface has many abstract methods like **get(String url), quit(), close(), getWindowHandle(), getWindowHandles(), getTitle() etc**.

WebDriver has nested interfaces like **Window, Navigation, Timeouts etc**. These nested interfaces are used to perform operations like **back(), forward() etc**.

Modifier and Type	Method	Description
void	close()	Close the current window, quitting the browser if it's the last window currently open.
WebElement	findElement(By by)	Find the first WebElement using the given method.
List<WebElement>	findElements(By by)	Find all elements within the current page using the given mechanism.
void	get(String url)	Load a new web page in the current browser window.
String	getCurrentUrl()	Get a string representing the current URL that the browser is looking at.
String	getPageSource()	Get the source of the last loaded page.
String	getTitle()	Get the title of the current page.
String	getWindowHandle()	Return an opaque handle to this window that uniquely identifies it within this driver instance.
Set<String>	getWindowHandles()	Return a set of window handles which can be used to iterate over all open windows of this WebDriver instance by passing them to switchTo().WebDriver.Options.window()
WebDriver.Options	manage()	Gets the Option interface
WebDriver.Navigation	navigate()	An abstraction allowing the driver to access the browser's history and to navigate to a given URL.
void	quit()	Quits this driver, closing every associated window.
WebDriver.TargetLocator	switchTo()	Send future commands to a different frame or window.

Selenium WebElement Interface methods

Selenium WebElement represents an HTML element. We can get an instance of WebElement using

findElement() method and then perform specific actions such as click, submit

etc. Some of the commonly used WebElement methods are:

Modifier and Type	Method and Description
void	`clear()` If this element is a text entry element, this will clear the value.
void	`click()` Click this element.
WebElement	`findElement(By by)` Find the first `WebElement` using the given method.
List<WebElement>	`findElements(By by)` Find all elements within the current context using the given mechanism.
String	`getAttribute(String name)` Get the value of a the given attribute of the element.
String	`getCssValue(String propertyName)` Get the value of a given CSS property.
Point	`getLocation()` Where on the page is the top left-hand corner of the rendered element?
Rectangle	`getRect()`
Dimension	`getSize()` What is the width and height of the rendered element?
String	`getTagName()` Get the tag name of this element.

Selenium Browser Options

The browser options refer to the features and capabilities shared by all browsers. It helps to modify the settings and capabilities of the browser while running an automated test on any browser. Selenium Webdriver begins with a fresh browser profile without any predefined settings on cookies, history, and so on by default.

```
ChromeOptions options = new ChromeOptions();
options.addArguments("start-maximized");        //start with full screen
options.addArguments("--incognito");            //open in incognito mode
options.setBinary("122");                                   //Select binary for 122
options.setBrowserVersion("122");     //Select browser version of 122 options.addArguments("--disable-notifications");
WebDriver driver = new ChromeDriver(options);
```

Selenium Browser DesiredCapabilities

DesiredCapabilities are a set of key-value pairs encoded as a JSON object. It helps QAs define basic test requirements such as operating systems, browser combinations, browser versions, etc. within Selenium test scripts.

```
DesiredCapabilities capabilities = DesiredCapabilities.chrome();
capabilities.setCapability(CapabilityType.ACCEPT_INSECURE_CERTS, true); WebDriver driver = new ChromeDriver(capabilities);
OR
Map<String, Object> prefs = new HashMap<>();//allow mircophone permission
prefs.put("profile.default_content_setting_values.media_stream_mic", 2);
//value "1" is used for allowing the option, "2" -- for blocking.
//prefs.put("profile.default_content_setting_values.media_stream_camera", 1);
options.setExperimentalOption("prefs", prefs);
WebDriver driver = new ChromeDriver(options);
```

How to Launch Browser in WebDriver?

```
WebDriver driver = new ChromeDriver();  // For Chrome WebDriver
driver = new FirefoxDriver(); // For Firefox WebDriver driver = new
EdgeDriver();                                          // For Edge

WebDriver driver = new SafariDriver();   // For Safari
```

Remote WebDriver

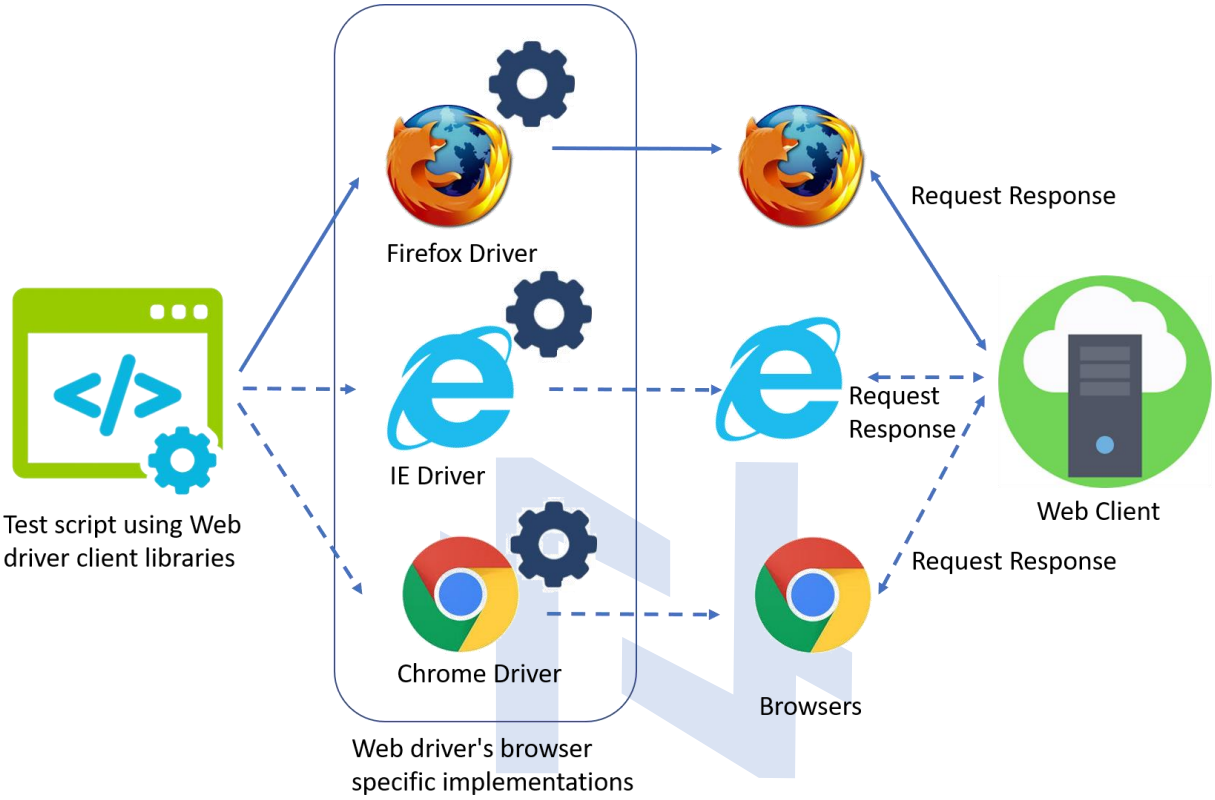
The RemoteWebDriver class implements the WebDriver interface to execute test scripts through the RemoteWebDriver server on a remote machine.

Remote WebDriver Architecture

> Remote WebDriver consists of a server and a client.

> The server is a component that listens on a port for various requests from a Remote WebDriver client. Once the request is received, it forwards the request to the browser driver: FirefoxDriver, IEDriver, or

> The client libraries serve as a Remote WebDriver client. The client translates test script requests to JSON payload and sends it across to the Remote WebDriver server using the JSON wire protocol. The



When test cases are executed, the WebDriver client libraries link with the browser drivers directly. On the other hand, if one tries to execute tests remotely, the WebDriver client libraries communicate with the Remote WebDriver server. Then, the server links to either of the browser drivers that the WebDriver client requests for.

```
ChromeOptions chromeOptions = new ChromeOptions();
chromeOptions.setCapability("browserVersion", "122");
chromeOptions.setCapability("platformName", "Windows 10");
WebDriver driver = new RemoteWebDriver(new URL("ServerURL"), chromeOptions);
driver.get("http://www.google.com");
driver.quit();
```

Different Ways to Define WebElements

1. By highlightedOption = By.xpath("//li[contains(@class,'highlighted')]"); WebElement element = driver.findElement(By.xpath(highlightedOption));

2. @FindBy(xpath ="//div[@class='filterIcon']") WebElement filterIcon;

3. By mySelector = By.xpath("//div[@class='filterIcon']"); WebElement myElement = driver.findElement(mySelector);

4. @FindAll({ @FindBy(xpath = "//select[@id='GridID']//option") }) List<WebElement> gridsIDs;

5. By mySelector = By.xpath("//select[@id='GridID']//option"); List<WebElement> myElements = driver.findElements(mySelector);

6. @FindBy(how = How.XPATH, using = "//span[.='Assign to User']") @CacheLookup WebElement reassignAction;

7. String headerXpath="//div[contains(@class,'row head')]//div"; String rowXpath="//div[contains(@class,'tr row')]//div";

   WebElement headerElement = driver.findElement(By.xpath(headerXpath));

   WebElement rowElement = driver.findElement(By.xpath("//div[contains(@class,'tr row')]//div"));

Different ways to Initialize/Use WebElment

```
1. public boolean isElementDisplayed(WebElement element) { boolean
   isElementDisplayed = element.isDisplayed();
   return isElementDisplayed;
 }

2. public boolean isElementDisplayed(WebDriver driver, By locator) {
   boolean isElementDisplayed = driver.findElement(locator).isDisplayed(); return isElementDisplayed;
 }

3. public boolean isElementDisplayed((WebDriver driver, String locatorStr) {
   boolean isElementDisplayed =
   driver.findElement(By.xpath(locatorStr)).isDisplayed(); return
   isElementDisplayed;
   }
```

Types of Locators used for locating Web Elements

1. **Id**: Locates elements whose ID attribute matches the search value

```
driver.findElement(By.id("idValue"));
```

2. **Name**: Locates elements whose NAME attribute matches the search value

```
driver.findElement(By.name("nameValue"));
```

3. **Link Text**: Locates anchor elements whose visible text matches the search values

```
driver.findElement(By.linkText ("Text in the Link"));
```

4. **Partial Link Text**: Locates anchor elements whose visible text contains the search values. If multiple elements are matching, only the first one will be selected.

```
driver.findElement(By.partialLinkText ("Partial Text in the Link"));
```

5. **Class Name**: Locates elements whose class name contains the search value.

```
driver.findElement(By.className ("classValue"));
```

6. **Tag Name**: Locates elements whose Tag Name matches the search values

```
driver.findElement(By.tagName ("html tagName"));
```

7. **CSS Selector**: Locates elements matching a CSS selector

```
driver.findElement(By.cssSelector("input[type='submit']"));
```

8. **Xpath** : Locates elements matching an Xpath expression

```
driver.findElement(By.xpath("//input[@type='submit']"));
```

What is Xpath?

> Xapth stands for **XML Path Language**
>
> Xpath is used to find the location of any elements on a webpage using HTML
>
> DOM structure. It was defined by the World Wide Web Consortium (W3C)
>
> Xpath is used to navigate nodes in any XML document
>
> Xpath uses "path like" syntax

Basic Format of Xpath

Syntax :

```
Xpath = // tagname [ @Attribute = 'Value']
```

Types of Xpath

Absolute Xpath

- It contains the complete path from the root element of page to the desired element.
- Absolute Xpath starts with root node - Single Forward Slash (/)
- Drawback of using absolute Xpath - Any slight change in HTML DOM makes the Xpath invalid.

Ex:

```
html/body/div[5]/div[1]/div[2]/div/div/form/div/div/input
```

Relative Xpath

- With relative Xpath, the Xpath starts from the mid of the HTML DOM structure.
- It begins with the Double Forward Slash (//)
- It is less brittle

Ex:

```
//input[@id='First_Name']
```

Dynamic XPaths

```
//*[@type='submit']                                  → any tag with type submit
//h2[contains(@id, 'ageCont')]                       → selects id that contains ageCont value (//h2[starts-
with(@id, 'u_')])[1]                                 → the first input whose id starts with u_
//input[ends-with(@id, 'P7')]                        → selects id that ends with p7
//h2[@id='male' or @id='female']                     → one or the other statement
//h2[@id='male' and @class='radio-btn'] → both statements
//*[.= 'Sign in']                                   → any tag & attribute just give me the text
//*[(text() = 'Welcome')]                            → selects only text
//*[contains(text(), 'Welcome to')] → selects only text that contains
(//div[@id='someBtn'])[1]                          → Use index when there are multiple matches
```

CSS Selector:

.classValue → By.cssSelector(".form-control") #idValue → By.cssSelector("#ageCont")

Selenium Operations

Launch a Webpage

```
driver.get("https://www.google.com");  // OR
driver.navigate().to("https://www.google.com");  //
```

Click a button

```
driver.findElement(By.name("btnK")).click(); OR
WebElement searchBtn = driver.findElement(By.name("btnK")); searchBtn.click();
```

Accept an alert pop-up

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

Print the page title

```
String title = driver.getTitle(); System.out.println(title);
```

Clear the input field text

```
WebElement searchInput = driver.findElement(By.name("q")); searchInput.sendKeys("selenium");
searchInput.clear();
```

Disable a field (set the 'disabled' attribute)

```
JavascriptExecutor javascript = (JavascriptExecutor) driver; String toDisable =
"document.getElementsByName('fname')
[0].setAttribute('disabled',  '');";
javascript.executeScript(toDisable);
```

Enable a field (remove the 'disabled' attribute)

```
JavascriptExecutor javascript = (JavascriptExecutor) driver;
String toEnable = "document.getElementsByName('fname')[0].setAttribute(enabled, '');";
javascript.executeScript(toEnable);
```

Wait Operations

Selenium Dynamic Wait

1. Implicit wait → global wait

```
driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
```

2. Explicit wait → local wait

a. Create WebDriverWait object

```
WebDriverWait wait = new WebDriverWait(driver,Duration.ofSeconds(10));
```

b. Use the object to add expected conditions

```
WebElement classABC =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector(".classloc  ator")));
```

→ better than implicit wait when element is not visible / clickable / displayed

3. FluentWait → local wait. Is like Explicit wait with more options

```
Wait<WebDriver> fluentWait = new FluentWait<WebDriver>(driver)
.withTimeout(Duration.ofSeconds(30))
.pollingEvery(Duration.ofSeconds(5))//will check every 5 sec
```

Same as Explicit Wait:

```
WebElement classABC = wait.until(ExpectedConditions
.visibilityOfElementLocated(By.cssSelector(".classlocator")));
```

ScriptTimeout & PageLoad Timeout:

```
driver.manage().timeouts().scriptTimeout(Duration.ofMinutes(2));
driver.manage().timeouts().pageLoadTimeout(Duration.ofSeconds(10));
```

4. Java hard wait →

```
Thread.sleep(Time in MilliSeconds);//Thread.Sleep throws InterruptedException
```

Alerts

Accept an alert: Same as clicking OK of an alert

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

Dissmiss an alert: Same as clicking Cancel of an alert

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

Enter text in an alert box

```
driver.switchTo().alert().sendKeys("Selenium")
```

Retrieve alert text: To get the alert message of the alert

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

Selenium Navigators

Navigate to a URL

```
driver.get("URL") OR
driver.navigate().to("URL");
```

Refresh the page

```
driver.navigate().refresh();
```

Navigate forward in browser

```
driver.navigate().forward();
```

Navigate back in browser

```
driver.navigate().back();
```

iFrame

A page within a page → we must first switch() to the iframe. 3 ways:

1. by index: → index start from 0

```
driver.switchTo().frame(0) will switch the first iframe
```

2. id/name

```
driver.switchTo().frame("id or name of the iframe");
```

3. web element (locators)

```
WebElement middleFrame =
driver.findElement(By.xpath("//frame[@name='left']")); driver.switchTo().frame(middleFrame);
```

→ **Switching back to parent / default frame**

To parent frame goes only 1 level up:

```
driver.switchTo().parentFrame();
```

To get back to the main fraim:

```
driver.switchTo().defaultContent();
```


Returns the total number of iframe on a page:

```
driver.findElements(By.tagName("iframe"));
```

Working with Windows/Tabs

1. Get the current window handle

```
String window1Handle = driver.getWindowHandle();
```

2. Get all window handles

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

3. Switch to a specific window

```
for (String eachHandle : allWindowHandles){ if
    (!eachHandle.equals(window1Handle)){
        driver.switchTo().window(eachHandle);
    }
}
OR
String windowHandle = driver.getWindowHandle(); driver.switchTo().window(windowHandle);
```

Switch to newly created window

```
driver.switchTo().newWindow(WindowType.TAB);
driver.switchTo().newWindow(WindowType.WINDOW);
```

Close the current window

```
driver.close();
```

Set window position

```
driver.manage().window().setPosition(new Point(0, 0));
```

Maximize window

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

Minimize window

```
driver.manage().window().minimize();
```

Fullscreen window

```
driver.manage().window().fullscreen();
```

Actions

Step 1: Create the action object

```
Actions actions=new Actions(driver);
```

Step 2: Locate the WebElement you want to work on

```
WebElement element = driver.findElement(By.id("ID"));
```

Step 3: Perform the action on the WebElement

Right click

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

Hover over

```
actions.moveToElement(element).perform();
```

Actions

```
element.sendKeys(Keys.ARROW_DOWN) element.sendKeys(Keys.ARROW_UP)

element.sendKeys(Keys.PAGE_DOWN) element.sendKeys(Keys.PAGE_UP)

element.build() //OPTIONAL : recommended with method chains element.perform();
//MANDATORY
```

```
//Perform Click on LOGIN button using JavascriptExecutor Actions action =
new Actions(driver);

WebElement element =driver.findElement(By.name("btnLogin"));


action.moveToElement(element).click().perform();//Click

action.doubleClick(element).build().perform();//Double  Click

action.moveToElement(elementToHover).click(elementToClick).build().perform();//Hov  er & Click

action.contextClick(element).build().perform();//Right  Click

action.dragAndDrop(dragElement,  dropElement).build().perform();//Drag  &  Drop


//Set Text using Action class

action.click().keyDown(Keys.CONTROL).sendKeys("a").keyUp(Keys.CONTROL).sendKeys(Ke

ys.BACK_SPACE).build().perform();

action.sendKeys(element,  valueToBeEntered).sendKeys(Keys.ENTER).build().perform();


//Clear Text using Action class

action.click(element).keyDown(Keys.CONTROL).sendKeys("a").keyUp(Keys.CONTROL).send
Keys(Keys.DELETE).build().perform();


actions.keyDown(Keys.CONTROL).sendKeys(Keys.HOME).perform();//Scroll to Top
actions.keyDown(Keys.CONTROL).sendKeys(Keys.END).perform();//Scroll to Bottom
```

keysDown(); → to press and hold a key. Keys mean Shift,Ctrl, Alt keys.

keysUp(); → to release a pressed key after keysDown(), otherwise we may get **IllegalArgumentException**.

sendKeys(**element**,"**text**"); → to type into text box / text area

Javascript Executor

1. Creating a reference

```
JavascriptExecutor jse = (JavascriptExecutor) driver;
```

2. Calling the method

```
js.exectueScript(Script, Arguments); js.executeScript(return something);
```

```
//Perform Click on LOGIN button using JavascriptExecutor JavascriptExecutor
jse = (JavascriptExecutor) driver;

WebElement element =driver.findElement(By.name("btnLogin"));


//arguments[0] -> the first argument in executeScript method jse.executeScript("arguments[0].click();",
button);//Click

jse.executeScript("arguments[0].dispatchEvent(new MouseEvent('dblclick', { bubbles: true }));",
element);//Double Click

jse.executeScript("arguments[0].value='" + valueToBeEntered + "' ;", element);//Set Text

jse.executeScript("arguments[0].value = '';", element);//Clear Text jse.executeScript("window.scrollTo(0,
0)");//Scroll to Top

jse.executeScript("window.scrollTo(0, document.body.scrollHeight)");//Scroll to Bottom

jse.executeScript("window.scrollBy(0,"+pixel+")");//Scroll By Vertical Pixel

jse.executeScript("arguments[0].scrollIntoView(true);", element);//Scroll Till Element

((JavascriptExecutor) driver).executeScript("window.open()");//Opens new window/tab

jse.executeScript("arguments[0].click();", uploadFileBtn);//Uploads file


String result = (String) js.executeScript("return someFunction();");//Get String String theTextIWant = (String)
jse.executeScript("return

arguments[0].innerHTML;",element);//Get Text
```

Drop Down List

Step 1: Locate the dropdown element

```
WebElement month=driver.findElement(By.id("dropdown"));
```

Step 2: Create Select object and pass the variable to that object

```
Select selectMonth=new Select(month);
```

Step 3: Select from a dropdown using select object with 3 different ways

```
selectMonth.selectByIndex(0); selectMonth.selectByValue("1");
selectMonth.selectByVisibleText("Jan");
```


We can put all dropdown elements in a List using getOptions()

```
Select selectOptions = new Select(states);
List<WebElement> options = selectOptions.getOptions();
```

TestNG Annotations

@Test → the main part of the automation script where we write the business logic we want to automate

@BeforeSuite → runs before executing all test methods in the suite

@BeforeTest → executes before executing all test methods of available classes belonging to that folder

@BeforeClass → executes before the first method of the current class is invoked

@BeforeMethod → executes before each test method runs

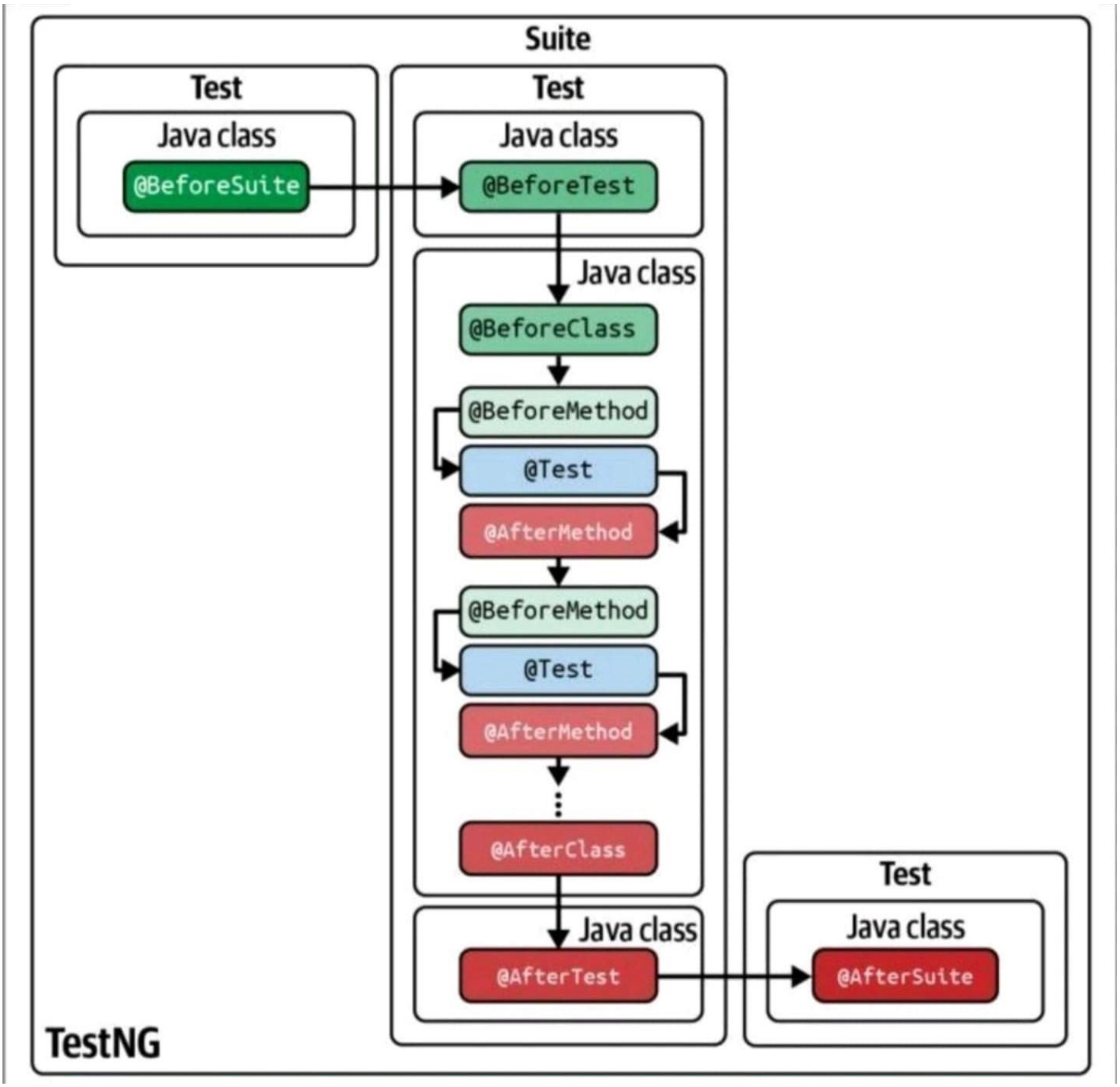
@AfterSuite → executes after executing all test methods in the suite

@AfterMethod → executes after executing each test method

@AfterTest → executes after executing all test methods of available classes belonging to that folder

@AfterClass → executes after executing all test methods of the current class

Sequence of execution for TestNg



Some important attributes of @Test annotations

1. **description** – The 'description' attribute is used to provide a description to the test method. It generally contains a one-liner test summary.

```
@Test(description = "Test summary")
```

2. **dataProvider** – This attribute helps in creating a data driven tests. It is used to specify the name of the data provider for the test.

```
@Test(dataProvider = "name of dataProvider")
```

3. **priority** – This attribute helps in prioritizing the test methods. The default priority starts with 0 and tests execute in ascending order. Prioty can be Negative e.x '-1'.

```
@Test(priority = 2)
```

4. **enabled** – This attribute is used to specify whether the given test method will run with the suite or class or not.

```
@Test(enabled = false)
```

5. **groups** – Used to specify the groups, the test method belongs to.

```
@Test(groups = { "sanity", "regression" })
```

7. **dependsOnMethods** – Used to specify the methods on which the test method depends. The test method only runs after successful execution of the dependent tests.

```
@Test(dependsOnMethods = { "dependentTestMethodName" })
```

8. **dependsOnGroups** – Used to specify the groups on which the test method depends.

```
@Test(dependsOnGroups = { "dependentGroup" })
```

9. **alwaysRun** – When set as True, the test method runs even if the dependent methods fail.

```
@Test(alwaysRun=True)
```

10. **timeOut** – This is used to specify a timeout value for the test(in milli seconds). If test takes more than the timeout value specified, the test terminates and is marked as failure.

```
@Test (timeOut = 500)
```

JUnit Annotations

@Test → Represents the method or class as a test block,

also accepts parameters. **@Before** → The method with

this annotation gets executed before all other tests.

@BeforeClass → The method with this annotation gets

executed once before class.

@After → The method with this annotation gets executed after all other tests are executed.

@AfterClass → The method with this annotation gets executed once after class.

@Ignore → It is used to ignore certain test statements during execution.

@Disabled → Used to disable the tests from execution, but the corresponding reports of the tests are still generated.

Get Screenshots

Take a Screenshot as File

```
import org.apache.commons.io.FileUtils;
import org.openqa.selenium.TakesScreenshot;

File screenshotFile = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
```

Take a Screenshot as Base64 Image

```
String base64String =
((TakesScreenshot)driver).getScreenshotAs(OutputType.BASE64);
```

Working with Files

Upload a file

```
driver.findElement(By.id("upload")).sendKeys("path/to/the/file.txt");
driver.findElement(By.id("file-submit")).submit();
```

Read data from an Excel file

<import Apache POI dependancy>

→ workbook > worksheet > row > cell

→ Index starts with 0 → e.g. row 1 cell 1 has the index of row 0 cell 0

1. Store file path in a string

   String path = "resources/testData.xlsx";

   OR

   File file = new File("resources/testData.xlsx");

2. Open the file

   FileInputStream fileInputStream = new FileInputStream(path);

3. Open the workbook using fileinputstream

   Workbook workbook = WorkbookFactory.create(fileInputStream);

4. Open the first worksheet

   Sheet sheet1 = workbook.getSheet("Sheet1");

   OR

   workbook.getSheetAt(0); //Use index of sheet

5. Go to first row

   Row row1 = sheet1.getRow(0);

6. Go to first cell on that first row and print Cell cell1 =

   row1.getCell(0);

```
FileInputStream file = new FileInputStream(new File("testData.xlsx")); XSSFWorkbook workbook = new XSSFWorkbook(file);

XSSFSheet sheet = workbook.getSheetAt(0); Row row = sheet.getRow(0);

Cell cell = row.getCell(0);

String testData = cell.getStringCellValue(); file.close();
```

Read data from a text file using BufferedReader

```
FileReader reader = new FileReader("MyFile.txt");
BufferedReader bufferedReader = new BufferedReader(reader); String line;
while ((line = bufferedReader.readLine()) != null){ System.out.println(line);
  }
reader.close();
```

Read data from a text file Using InputStream

```
FileInputStream inputStream = new FileInputStream("MyFile.txt");
InputStreamReader reader = new InputStreamReader(inputStream,"UTF-16"); int character;
while ((character = reader.read()) != -1){ System.out.print((char) character);
  }
reader.close();
```

Read data from a text file Using FileReader

```
FileReader reader = new FileReader("MyFile.txt"); int character;
while ((character = reader.read()) != -1) { System.out.print((char) character);
    }
reader.close();
```

Read data from a CSV file

```
import com.opencsv.CSVParser; import
com.opencsv.CSVReader;


String path = "C:\\Users\\user\\csvData.csv"; Reader reader = new
FileReader(path);

CSVReader csvreader = new CSVReader(reader);

List<String[]> listOFRecords = csvreader.readAll(); for (String[] row :
listOFRecords) {

  for (String cell : row) { System.out.println(cell);
```

Selenium Grid

//pull required images

docker pull selenium/hub:latest

docker pull selenium/standalone-chrome:latest docker pull selenium/standalone-firefox:latest docker pull selenium/standalone-edge:latest

docker pull selenium/node-chrome:latest docker pull selenium/node-firefox:latest docker pull selenium/node-edge:latest

docker pull portainer/portainer-ce:latest

//spin up selenium standalone browsers

docker container run -d -p 4444:4444 -p 7900:7900 --shm-size="2g" --name selenium_grid selenium/standalone-chrome

docker container run -d -p 4444:4444 -p 7900:7900 --shm-size="2g" --name selenium_grid selenium/standalone-firefox

docker container run -d -p 4444:4444 -p 7900:7900 --shm-size="2g" --name selenium_grid selenium/standalone-edge

//spin up selenium grid browsers http://ipaddress:4444/ui#