Selenium

Day-21

Selenium WebDriver

What is selenium webdriver?
---
1) WebDriver is one of the component in selenium.
2) WebDriver is a java interface.
3) WebDriver is an API(Application Programming interface)

WebDriver (I)-- RemoteWebDriver (C)---->ChromeDriver , FirefoxDriver, EdgeDriver etc..

Environment setup
---
1) Downloading jars files and attaching them to Java project.(Manually) - Not recomended
---
-> created new java project
-> downloaded webdriver jars(.zip) and extracted.
-> attach jars to java project.

2) Creating Maven project (Recomended)

-->Created a new Maven project in eclipse
--> addedd webdriver dependecy in pom.xml--> updated

Stable: 4.18.1 (February 19, 2024)

pom.xml ---> dependencies

https://mvnrepository.com/
web driver dependecy link:
https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java

Assignment
---
1) Launch browser (chrome)
2) Open URL https://demo.nopcommerce.com/
3) Validate title should be "nopCommerce demo store"
4) close page

**Note:
Opera browser supported till selenium 4.1.0.
Above 4.1.0 , Opera do not support.

1. What is Selenium?

- Selenium is an open-source automation testing tool
- It helps to automate web browsers
- It helps to validate web applications across different browsers like Chrome, Firefox, Internet Explorer and Safari
- It helps to validate the application in different platforms/Operating Systems (OS) like Windows, iOS and Linux
- It supports multiple programming languages like Java, C#, Python etc.

2. Who developed Selenium?

- Selenium was created in 2004 by Thought Works engineer **Jason Huggins**
- He created a JavaScript program that helps to test repetitious manual test of their application
- He named this program as the "JavaScriptTestRunner."
- He did some changes in “JavaScriptRunner” and made it as open source which was later re-named as “Selenium Core”

3. What is Same Origin Policy Issue?

- Selenium Core (i.e. Java script) can access the GUI element from where it is launched (i.e. same domain)
- It is not possible to access from different domain. This is called same origin policy.
- Example: HTML code in google can access only in google domain and it can’t access in different domain yahoo
- If we want to use Selenium Core (i.e. Java script), tester needs to install Selenium Core (i.e. Java script) + Application under test + web server on their own local computer before start automating the application

4. What is Selenium Remote Control (Selenium RC)/ Selenium 1?

- To overcome the same origin policy issue, another ThoughtWork's engineer, Paul Hammant created server
- The server is call Selenium RC
- Selenium RC act as HTTP proxy and helps identify Selenium Core + Application under test in the same domain
- This system known as Selenium RC (or) Selenium 1

5. What is WebDriver in Selenium?

- WebDriver is an interface
- It directly communicating with browser
- It helps to execute your script into multiple browsers and multiple OS
- It helps to create script in multiple languages. You can choose your own language to create Selenium script
- It provide feature to create HTMLUnit browsers (HTMLUnit browser are headless browsers which means these are invisible to the user, in simple words they have no GUI)

6. What is Selenium Grid?

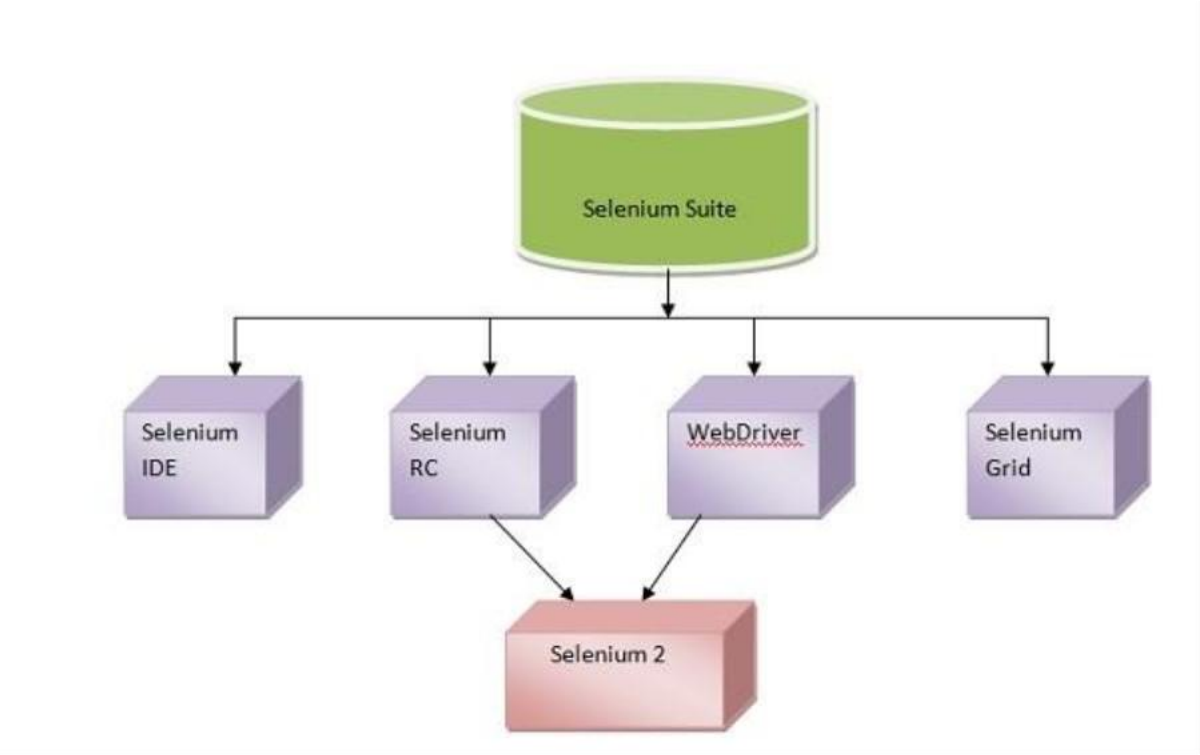
- Selenium Grid is the next level of Selenium
- It helps to run multiple test across different machines at the same time in different browser at the same time

7. What is Selenium Integrated Development Environment (IDE)?

- It is FireFox extension/plugin that can automate the browser through a record-and-playback feature
- It is available only in FireFox
- Can’t create Condition statements (If…else) or Loop statement (For…Next)
- Test execution is slow when compare to Selenium RC and WebDriver

8. What is Selenium Suite?

- Selenium is an open source automation testing tool
- It is not single tool.
- It is a combination of multiple tools.
- These multiple combinations (IDE + RC + WebDriver + Grid) is called Selenium Suite



9. What is Selenium2?

Selenium2 is nothing but integration of Selenium 1 (Selenium RC + WebDriver)

10. How to install Selenium in any machine?

- Install programming language (Example: Java)
- Install IDE (Example: Eclipse)
- Download Selenium Standalone server
- Create Project (Example: Java Project in Eclipse)
- Configure/Build Selenium Standalone server into Java

- Download the Browser Drivers (Example: IE driver, Chrome driver, Geko driver etc)
- Set up the browser driver to start the scripting

11. Why it is name as Selenium?

- Scientifically Selenium helps to remove Mercury
- Before Selenium tool, Quick Test Profession (QTP) tool was very famous in the market
- Now QTP is called as Unified Functional Testing (UFT)
- QTP was developed by the company called Mercury Interactive
- When they introduce Selenium, they predicted it will remove Mercury's QTP from market. Hence, they named it as Selenium

12. What is Selenium Standalone server?

- Selenium Standalone server is combination of multiple java jar file
- It helps to start the Selenium server
- It helps to communicate the Selenium script/test to browser driver instance

13. What is the Selenium Standalone server you have used in your project? (or) What is the Selenium version used in your project?

- Selenium Standalone 2.53 is used in my project

14. What is the stable Selenium Standalone server?

- Selenium Standalone 2.50+ is consider as most stable version

15. What is the current version of Selenium Standalone server?

- Current stable version as on Feb-2018 is 3.9

- Latest alpha version as on 01-July-2019 is 4.0

16. What is the difference between Selenium 2 and 3?

Selenium 2	Selenium 3
Combination of SeleniumRC (Selenium1) + WebDriver	SeleniumRC is removed from Selenium3. But supports Selenium RC indirectly through back-end Webdriver.
Firefox is the default browser	Firefox is not default browser. GeckoDriver has been introduced for Firefox (v47 and above)
WebDriver API supports Selenium RC API and Selenium RC	No Web Driver API
Supports Mozilla, Chrome, IE and Safari drivers	Browser drivers extends Edge
AndroidDriver and iPhone are used for mobile automation	Appium is used for Mobile Automation

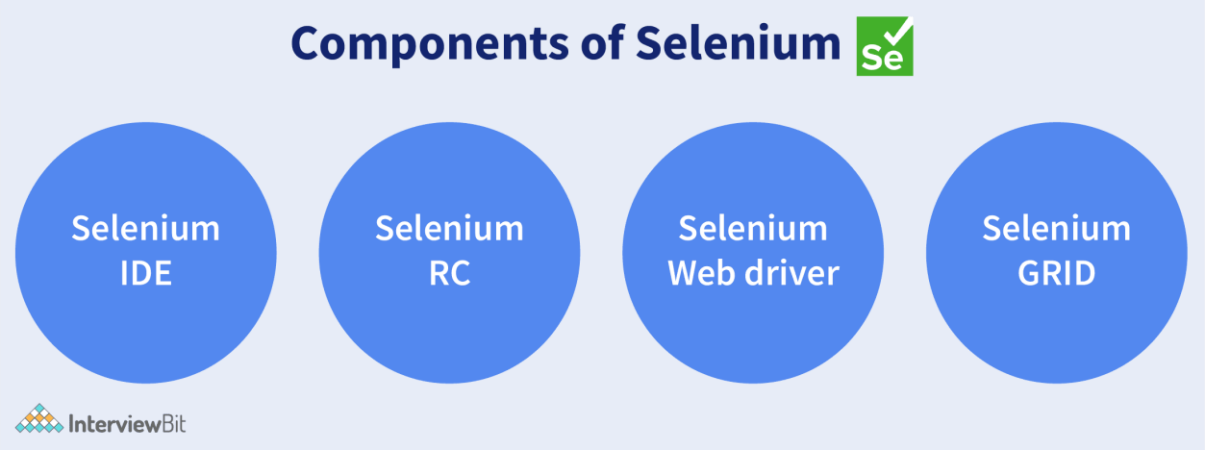
1. What is Selenium?

– Selenium is an **open-source automation testing framework** for web applications, supporting multiple browsers, platforms, and programming languages like Java, Python, and C#.

2. What are the Selenium suite components?

Selenium is a package of several testing tools. It is therefore often referred to as a Selenium Suite with each of these tools designed to cater to a different testing requirement.

The following are the different **components of Selenium Suite**:



- **Selenium Integrated Development Environment (IDE)**: It is a Firefox/Chrome plug-in that is developed to speed up the creation of automation scripts by recording the user actions on the web browser and exporting them as a reusable script.
- **Selenium Remote Control (RC)**: It is a server that enables users to generate test scripts in their preferred programming language. It accepts commands from the test scripts and sends

them to the browser as Selenium core JavaScript commands, for the browser to behave accordingly.

- **Selenium WebDriver:** It is a programming interface that helps create and run test cases by directly communicating with the web browser and using its native compatibility to automate. Unlike RC, it doesn't require an additional server to create and run test cases.
- **Selenium Grid:** It allows parallel execution of tests on different browsers and operating systems by distributing commands to other machines simultaneously

3. What are the advantages of using Selenium as an automation tool?

- Open-source & free.
- Supports multiple languages & frameworks.
- Cross-browser & cross-platform support.
- Integrates with CI/CD tools like Jenkins, Maven, TestNG.
- Supports parallel execution with Selenium Grid.

4. What are the disadvantages of Selenium?

- Works only for **web apps**, not desktop apps.
- **No built-in reporting** (needs TestNG/JUnit).
- Requires **programming knowledge**.
- No dedicated **vendor support** (community-driven).

5. What is automation testing, and what are its advantages?

Automation Testing or Test Automation is a process of automating the manual testing process of an application or a system by using testing tools that allow you to create scripts that can be executed repeatedly, generating detailed test reports of the application or system under test.

Advantages:

- Faster execution & repeatability.
- Reduces human error.
- Supports parallel execution.
- Saves time & cost.
- Useful for large test suites.

6. Why should Selenium be selected as a testing tool?

– Because it is **free, open-source, cross-browser, cross-platform, supports multiple languages, and enables distributed/parallel testing**.

7. What is Selenese? Explain different types of Selenium commands.

The language used for writing test scripts in Selenium IDE is called Selenese. It is a set of commands used to test your web application or system. Selenium commands could be divided into 3 major categories:

- **Actions:** These are the commands interacting directly with web applications.
- **Accessors:** These are the commands which allow users to store values in a user-defined variable.

- **Assertions:** They enable a comparison of the current state of the application with its expected state.

1. What is Selenium WebDriver?

Answer:
Selenium WebDriver is a component of Selenium that provides an API for automating browsers. It is a **Java Interface** that defines methods like get(), findElement(), click(), etc. Each browser (Chrome, Firefox, Edge) has its own implementation class (ChromeDriver, FirefoxDriver, etc.) extending from RemoteWebDriver.

---

2. Explain WebDriver class hierarchy.

Answer:

- WebDriver → Interface.
- RemoteWebDriver → Concrete class implementing WebDriver.
- Browser drivers (ChromeDriver, FirefoxDriver, EdgeDriver) extend RemoteWebDriver.

Hierarchy:
WebDriver (I) → RemoteWebDriver (C) → ChromeDriver / FirefoxDriver / EdgeDriver

---

3. Difference between attaching JAR files manually and using Maven dependency?

Answer:

- **Manual JAR:** You need to download Selenium JARs, extract them, and attach them manually to the project. Hard to maintain.
- **Maven:** Just add a dependency in pom.xml. Maven downloads and manages compatible versions automatically. Easier to update and share projects.

---

4. What is the latest stable version of Selenium WebDriver?

Answer:
As of **19 February 2024**, the stable version is 4.35.0

---

5. Why is Maven recommended for Selenium projects?

Answer:

- Automatic dependency management.

- Easy project sharing.
- Avoids manual JAR handling.
- Supports build tools (Maven, Gradle).

---

6. What happens if you try to automate Opera browser with Selenium 4.18.1?

Answer:
OperaDriver is **deprecated after Selenium 4.1.0**. In newer versions (4.2+), Opera is not supported, so automation fails.

---

7. How to launch a browser using Selenium WebDriver?

Answer (Code):

```
WebDriver driver = new ChromeDriver();

driver.get("https://example.com");
```

---

8. Difference between driver.close() and driver.quit()?

Answer:

- driver.close() → Closes only the current browser window.
- driver.quit() → Closes all windows and ends WebDriver session.

---

9. What is the role of getTitle() method in WebDriver?

Answer:
getTitle() returns the title of the current page as a String.
Example:

```
String title = driver.getTitle();

System.out.println("Page Title: " + title);
```

```
Day-22
------
id
name
linkText
partialLinkText

TagName
classname
```

CSS selector
XPath

<input type="text" name="search" value="" placeholder="Search" class="form-control form-control-lg">

input - tagname
type, name, value, placeholder ---> attributes/properties

name="search"
name--> attribute
search ---> value

findElement(By.name("search")) ----> WebElement

WebElement searchBox=driver.findElement(By.name("search"));

Ctrl +Shift+O --> importing packages.

String s="welcome";
s.lenght()

"welcome".lenght();

Employee emp=new Employee();
emp.display();

new Employee().display();

Click

booklets

By.partialLinkText("lets")

findElement(loc) Vs findElements(loc)
- - - - - - - - - - - - - - - - - -

Scenario1: Locator is smatching with single web element

findElement(loc) ----> single web element ---> WebElement
findElements(loc) ---> single web element---- > List<WebElement>

Sceanrio2 : Locator is matching with multiple web elements

findElement(loc) ---> single web element --> WebElement
findElements(loc) ---> multiple web elements ---> List<WebElement>

Scenario3: Locator is not matching with any element.

findElement(loc)---- > NosuchElementException
findElements(loc)----->will not throw any exception. Returns 0

Assignment

Open application "https://www.demoblaze.com/index.html"

1) Total number of links
2) Total number of images
3) Click on Any product link using linkText /partialLinkText

1. What are different locator types in Selenium?

- Selenium provides 8 locators:
  - **id, name, className, tagName, linkText, partialLinkText, cssSelector, xpath.**

---

2. Which locator is the fastest in Selenium?

- **id** is the fastest and most preferred locator, since it is unique.

---

3. What is the difference between linkText and partialLinkText?

- **linkText** → Matches the exact text of a link.
- **partialLinkText** → Matches a portion of the link text.
  Example:

By.linkText("Click here");

By.partialLinkText("Click");

---

4. What is the difference between findElement() and findElements()?

- **findElement()** → Returns the first matching WebElement, throws **NoSuchElementException** if not found.
- **findElements()** → Returns a List of WebElements. If not found, returns **empty list (size 0)**, no exception.

---

5. What happens if a locator matches multiple elements?

- **findElement()** → Returns the first matching element.
- **findElements()** → Returns all matching elements in a list.

---

6. What's the difference between absolute XPath and relative XPath?

- **Absolute XPath** → Starts from root node (/html/body/...). Long & fragile.
- **Relative XPath** → Starts from any node (//input[@name='search']). More reliable and preferred.

---

7. Which locator is more reliable between CSS Selector and XPath?

- **CSS Selector** → Faster, clean, widely supported.
- **XPath** → More powerful, supports both forward and backward traversing.

```
Day-23
-------

CSS Selector
--


CSS - Cascading Style Sheets


tag id                      tag#id
tag class                   tag.classname
tag attribute     tag[attribute="value"]
tag class attribute         tag.classname[attribute="value"]



Assingment
```

---
https://demo.opencart.com/

try to write css locators to identify elements.

1. What is a CSS Selector in Selenium?

– CSS Selector is a locator strategy in Selenium used to identify web elements based on **CSS attributes** like id, class, name, or custom attributes.

---

2. Why is CSS Selector preferred over XPath sometimes?

- CSS Selectors are **faster** than XPath.
- CSS is **cleaner and shorter**.
- CSS works well in most browsers (except IE has some XPath issues).

---

3. What are different ways to write CSS Selectors?

- **By id** → tag#id → input#search
- **By class** → tag.className → input.form-control
- **By attribute** → tag[attribute='value'] → input[name='search']
- **By class + attribute** → tag.className[attribute='value']
- **Multiple attributes** → tag[att1='val1'][att2='val2']

---

4. Difference between CSS Selector and XPath?

- **CSS** → Faster, simpler syntax, but can't move backwards (no parent axis).
- **XPath** → More powerful (supports both forward and backward traversal).

---

5. Which is faster: CSS or XPath?

– **CSS Selectors are faster** because they are directly supported by browsers' rendering engines.

---

6. How do you find an element by id in CSS?

— tag#id
Example: input#search

---

7. How do you find an element by class in CSS?

— tag.className
Example: input.form-control

---

8. How do you find an element by attribute in CSS?

— tag[attribute='value']
Example: input[name='search']

---

9. How do you combine class and attribute in CSS?

— tag.className[attribute='value']
Example: input.form-control[name='search']

---

10. Can CSS Selectors be used to locate dynamic elements?

- Yes , by using **partial matching** in attributes:
  - tag[attribute^='startText'] → starts with
  - tag[attribute$='endText'] → ends with
  - tag[attribute*='middleText'] → contains

Day-24

XPath

XPath is an address of the element.

DOM - Document Object Model

2 types of xpath
---
1) Absolute xpath(full xpath)
Ex: /html/body/header/div/div/div[2]/div/input

2) Relative xpath(partial xpath)
Ex: //*[@name="search"]

Which xpath will be prefered?
Relative xpath.

Difference between Absolute & Relative xpaths?
---
1) Absolute xpath starts with / ---> represents root node
 Relative xpath starts with //

2) Abosulate xpath do not use attributes
 Relative xpath works with attribute

3) Absolute xpath traverse through each node till it finds element
  Relative xpath directly jump and find the element by using attribute

Relative xpath
---
1) Automatically - Devtools, selectorshub
2) Manually(own xpath)

//img[@title='Your Store']
/html[1]/body[1]/header[1]/div[1]/div[1]/div[1]/div[1]/a[1]/img[1]

syntax:

//tagname[@attribute='value']
//*[@attribute='value']

//img[@title='MacBook']

Xpath with single attribute
---
driver.findElement(By.xpath("//input[@placeholder='Search']")).sendKeys("TShirts");

xpath with multiple attributes

//input[@name='search'][@placeholder='Search']

xpath with 'and' 'or' operators

//input[@name='search' and @placeholder='Search']
//input[@name='search' or @placeholder='Search']

xpath with inner text - text()

```
//a[text()='Desktops']
//a[text()='MacBook']


<a href="https://xyz.com"> Click Me </a>

linktext = yes
inner text = yes

<div>welcome</div>
linktext= no
innertext= yes


xpath with contains()
--------------
//input[contains(@placeholder,'Sea')]


handling dynamic attributes
---------

//*[@id='start' or @id='stop']
//*[contains(@id,'st')]
//*[starts-with(@id,'st')]


name=xyz001  xyz002   xyz 003 xyz004 xyz001 xyz 002

//*[contains(@name,'xyz')]
//*[contains(@name,'00')]
//*[starts-with(@name,'xyz')]

[last()]
[position=2]


name= 1xyz 2xyz 3xyz 4xyz 1xyz
          //*[contains(@name,'xyz')]


name=101xyz  201xyz  301xyz  401xyz
          //*[contains(@name,'  xyz')]
          //*[contains(@name,' 01')]

chained xpath
----------
//div[@id='logo']/a/img



<div></div>

//div[contains(text(),'')]
```

```
//[contains(.,' ')]
```

Q1. What is XPath in Selenium?

– XPath is a language used to navigate through elements and attributes in an XML/HTML document. In Selenium, it is used to locate elements in the DOM when other locators (id, name, className) are not sufficient.

Q2. What is the difference between Absolute XPath and Relative XPath?

- **Absolute XPath:** Starts from the root (/html/body/...) and traverses down to the element.
  - Example: /html/body/header/div/div/div[2]/div/input
  - \+ Fragile: Breaks if the DOM structure changes.
- **Relative XPath:** Starts with // and directly finds elements using attributes.
  - Example: //input[@name='search']
  - 🟩 Preferred: Shorter and more robust.

Q3. Which XPath do you use in real-time projects? Why?

– **Relative XPath** is always preferred because it is shorter, more readable, and less likely to break when the DOM structure changes.

Q4. How do you handle dynamic elements using XPath?

– By using **functions like contains() or starts-with()**.

- Example:
  - //*[contains(@id,'st')] → Matches any element with id containing st.
  - //*[starts-with(@name,'xyz')] → Matches elements where name starts with xyz.

Q5. Difference between text() and contains(text(), ...) in XPath?

- //a[text()='MacBook'] → Matches exact text.
- //a[contains(text(),'Mac')] → Matches partial text (useful when text changes dynamically).

Q6. What is Chained XPath? Give an example.

– Chained XPath means locating an element inside another element.

- Example: //div[@id='logo']/a/img
  Here we first locate the with id='logo', then move inside to <a>, then <img>.

Q7. What are some common XPath functions used in Selenium?

- text() → Match exact text
- contains() → Match partial attribute/text
- starts-with() → Match prefix of attribute/text
- last() → Select the last element
- position() → Select element by position

Q8. Can we write XPath without using tagname?

—Yes, we can use * as a wildcard.

- Example: //*[@id='search'] → Matches any element with id="search".

Q9. How do you locate elements using multiple conditions in XPath?

— By using and / or operators.

- Example:
  - //input[@name='search' and @placeholder='Search']
  - //input[@name='search' or @id='txtSearch']

Q10. Between CSS Selector and XPath, which one is faster?

— **CSS Selector is faster** because browsers are optimized for CSS rendering. But **XPath is more powerful** since it allows traversal in both directions (parent-to-child and child-to-parent), while CSS only works forward (child nodes).

Day-25

XPath axis
--
self
parent
child
ancestor
descendent
following
preceding
following-sibling
preceding-sibling

Xpath Axis: https://youtu.be/BRzlyGXx13Q?list=PLUDwpEzHYYLut2OnS4GlY7fiSAabRmsz3

Selectorhub tools: https://www.youtube.com/playlist?list=PLUDwpEzHYYLv8wElK5Pmtuowv-j4AKxQF

Relative locators: https://youtu.be/prXw_lypLTs

16. Is Selenium WebDriver a library?

Selenium WebDriver is a prominent free open-source library for automating browsers and testing web applications.

17. Is Selenium WebDriver an interface or a class?

Selenium WebDriver is usually a set of methods defined by an interface. The browser-specific classes, on the other hand, provide an implementation of it by extending a class. AndroidDriver, ChromeDriver, FirefoxDriver, InternetExplorerDriver, SafariDriver, and others are some of the implementation classes.

18. What are the different types of WebDriver Application Programming Interfaces in Selenium?

The Various **Types of WebDriver APIs** in Selenium are as follows:

- Opera Driver
- InternetExplorer Driver
- Chrome Driver
- Safari Driver
- Android Driver
- Firefox Driver
- Gecko Driver
- iPhone Driver
- EventFiringWebDriver
- HTMLUnit Driver.

19. What programming languages does Selenium WebDiver support?

The various programming languages that Selenium WebDriver supports are as follows:

- Java
- C#
- Python
- Ruby
- Perl

- PHP

20. What open-source frameworks does Selenium WebDriver support?

The following are the open-source frameworks supported by the Selenium WebDriver:

- **TestNG:**
    - Cédric Beust designed TestNG, a testing framework for the Java programming language that was influenced by JUnit and NUnit.
    - TestNG was created with the purpose of covering a wider range of test categories, including unit, functional, end-to-end, integration, and so on, with more robust and user-friendly functions.
- **JUnit:**
    - It is used for Unit Testing of various types of applications.

21. What is WebDriver's super interface?

SearchContext is the Super Interface of the WebDriver.

22. Explain the following line of code.

Webdriver driver = new FirefoxDriver();

'WebDriver' is an interface, and we are generating a WebDriver object by instantiating a FirefoxDriver object (This object uses Firefox Driver to link the test cases with the Firefox browser).

23. Is it necessary to use Selenium Server to run Selenium WebDriver scripts?

Selenium Server is required when distributing Selenium WebDriver scripts for execution with Selenium Grid. Selenium Grid is a Selenium functionality that allows you to execute test cases on multiple machines on various platforms. You wish to execute your test cases on a remote machine because your local machine is running numerous applications. You will need to set up the remote server so that the test cases can run on it.

```
Day-26
------
1) get methods
2) conditional methods
3) browser methods
4) navigational methods
5) wait methods


get methods - we can access these methods through webdriver instance
-------
get(url) - opens the url on the browser
getTitle() - returns title of the page
getCurrentUrl() - retuns URL of the page
getPageSource()- returns source code of the page
getWindowHandle() - returns ID of the single Browser window
getWindowHandles() - retuns ID's of the multiple browser windows
```

conditional methods - access these commands thorugh WebElement
---
retuns boolean value(true/false)

isDisplayed() - we can check display status of the element
isEnabled() - we can check enable/disable status of the element/operational element
isSelected() - we can use to check the element is selected or not

browser methods
---
close() - close single browser
quit() - close multiple browsers

Assignment
---
https://testautomationpractice.blogspot.com/

Q1. What is the difference between close() and quit()?

- close() → Closes the currently focused browser window/tab.
- quit() → Closes all browser windows/tabs opened by the WebDriver session.

Q2. Difference between getWindowHandle() and getWindowHandles()?

- getWindowHandle() → Returns the **unique ID of the current browser window**.
- getWindowHandles() → Returns a **Set of IDs** of all opened browser windows (useful for switching between tabs).

Q3. What are conditional methods in Selenium? Why are they used?
Conditional methods return **boolean values (true/false)**. They are used for element validation before performing actions.

- isDisplayed() → Checks if element is visible on the page.
- isEnabled() → Checks if element is enabled for interaction.
- isSelected() → Checks if checkbox/radio button/dropdown option is selected.

Q4. Difference between getTitle() and getCurrentUrl()?

- getTitle() → Returns the title of the current web page.
- getCurrentUrl() → Returns the URL of the current web page.

Q5. What is getPageSource() used for?

– It returns the **HTML source code** of the current page. Useful for debugging but not for normal automation (because parsing is costly).

Q6. How do you switch between multiple windows in Selenium?

–Using getWindowHandles() + switchTo().window(windowID).

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

for(String win : windows){

  driver.switchTo().window(win);

}
```

Q7. What are the different wait types in Selenium?

- **Implicit Wait** → Sets a global wait for all elements.
- **Explicit Wait** → Waits for a specific condition (e.g., element clickable).
- **Fluent Wait** → Similar to explicit wait but checks at regular polling intervals.

24. What will happen if I execute this command? driver.get("www.interviewbit.com") ;

An exception is triggered if the URL does not begin with http or https. As a result, the HTTP protocol must be sent to the driver.get() method.

25. What is an alternative option to driver.get() method to open an URL in Selenium Web Driver?

driver.navigate() can be used instead. It is used for navigating forwards and backwards in a browser.

A String URL OR
A java.net.URL object

java
URL url = new URL("
https://example.com");
driver.navigate().to
(url);

26. What is the difference between driver.get() and driver.navigate.to("url")?

The difference between the two is as follows:

- driver.get(): To open a URL and have it wait for the entire page to load.
- driver.navigate.to(): To navigate to a URL without having to wait for the entire page to load.

27. What are the differences between the methods driver.close() and driver.quit()?

The functions of these two methods (driver.close and driver.quit) are nearly identical. Although both allow us to close a browser, there is a distinction.

- To close the current WebDriver instance, use driver.close().
- To close all open WebDriver instances, use driver.quit().

28. What are some cases that Selenium WebDriver cannot automate?

Some of the scenarios which we cannot automate are as follows:

- Selenium WebDriver does not support bitmap comparison.
- Using Selenium WebDriver to automate Captcha is not possible.
- Using Selenium WebDriver, we are unable to read bar codes.
- Video streaming scenarios: Selenium will almost never be able to recognise video controllers. To some extent, JavaScript Executor and flex UI selenium will work, although they are not completely dependable.
- Performance testing can be automated, however, it's preferable to avoid using Selenium for performance testing.

29. In Selenium WebDriver, what is an Object Repository?

Instead of hard-coding element locator data in the scripts, the Object Repository is used to store the element locator data in a centralized location. To store all of the element locators, we create a property file (.properties), which acts as an object repository in Selenium WebDriver.

Day-27
---
wait methods

Synchronization

NoSuchElementException - Element is not present on the page. Synchronization.
ElementNotFoundException - Locator is in-correct

Thread.sleep(time in ms)

Adv:
---
1) easy to use

DisAdv
---
1) if the time is not suffitient then you will get exception
2) it will wait for maximum time out. this will reduce the performace script.
3) multiple times

implicit wait
---
driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));

Adv:
1) single time/one statement
2) it will not wait till maximum time if the element is availble
3) Applicable for all the elements
4) easy to use

Disadvantge:
1) if the time is not suffitient then you will get exception

explicit wait
---
works based on the time and condition

declaration
use

points
---
1) Conditional based, it will work more effectively.
2) finding element is inclusive
3) it will wait for condition to be true, then consider the time
4) we need to write multiple statements for multiple elements

conditions
---
alertIsPresent()
elementSelectionStateToBe()
elementToBeClickable()
elementToBeSelected()
frameToBeAvaliableAndSwitchToIt()
presenceOfAllElementsLocatedBy()
presenceOfElementLocated()
textToBePresentInElement()
textToBePresentInElementLocated()
textToBePresentInElementValue()
titleIs()
titleContains()
visibilityOf()
visibilityOfAllElements()
visibilityOfAllElementsLocatedBy()
visibilityOfElementLocated()

Fluent wait

```
 Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
      .withTimeout(Duration.ofSeconds(time))
      .pollingEvery(Duration.ofSeconds(time))
      .ignoring(NoSuchElementException.class);
```

```
  WebElement foo = wait.until(new Function<WebDriver, WebElement>() {
   public WebElement apply(WebDriver driver) {
    return driver.findElement(By.id("foo"));
   }
  });
```

```
driver.manage().timeouts().pageLoadTimeout(Duration.ofSeconds(5));
```

Why waits matter

Web pages are dynamic: elements may load asynchronously (AJAX), animations/spinners appear, or JavaScript updates the DOM after the page loads. If your script tries to interact with an element that isn't yet present/visible/clickable, Selenium throws exceptions (NoSuchElementException, ElementNotInteractableException, StaleElementReferenceException, TimeoutException). Waits synchronize the test script with the web page so actions happen when the page is ready.

Overview of wait types

1) Thread.sleep(...) — hard wait (not recommended)

- Pauses execution for an exact time (ms).
- Advantages: simple.
- Disadvantages: always waits full time (wastes time), brittle, not condition-aware.

```
Thread.sleep(3000); // waits exactly 3 seconds
```

2) Implicit Wait — a global polling wait for findElement calls

- Set once per WebDriver instance: applies to all findElement/findElements.
- If element is not found, Selenium will poll until timeout then throw NoSuchElementException.

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

Notes

- Polling interval is internal (implementation-defined).
- Implicit waits are *not* condition-based (only waits for presence in DOM).

- **Do not mix with explicit waits** — mixing can cause unpredictable behavior.

3) Explicit Wait (WebDriverWait) — wait for a specific condition

- Wait until a specific condition is met or timeout occurs.
- More precise and preferred for most cases.

```
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(15));

WebElement el = wait.until(ExpectedConditions.elementToBeClickable(By.id("submit")));

el.click();
```

Common ExpectedConditions:
visibilityOfElementLocated, presenceOfElementLocated, elementToBeClickable, textToBePresentInElement, alertIsPresent, frameToBeAvailableAndSwitchToIt, invisibilityOfElementLocated, titleContains, etc.

4) Fluent Wait — explicit wait with custom polling & ignored exceptions

- More flexible: define timeout, polling interval, and exceptions to ignore.

```
Wait<WebDriver> wait = new FluentWait<>(driver)
    .withTimeout(Duration.ofSeconds(30))
    .pollingEvery(Duration.ofMillis(500))
    .ignoring(NoSuchElementException.class)
    .ignoring(StaleElementReferenceException.class);


WebElement element = wait.until(d -> d.findElement(By.id("dynamicElement")));
```

5) Page Load Timeout and Script Timeout

- pageLoadTimeout — how long to wait for driver.get() page loading to finish.
- scriptTimeout — maximum time allowed for asynchronous scripts executed via executeAsyncScript.

```
driver.manage().timeouts().pageLoadTimeout(Duration.ofSeconds(30));

driver.manage().timeouts().scriptTimeout(Duration.ofSeconds(20));
```

Day-28

Navigational commands
---
navigate().to(url)
navigate().back()

navigate().forward()
navigate().refresh()

driver.get() ---> accepts URL in string format only
driver.navigate().to() ---> accepts URL in the string format & URL object format.

getWindowHandle()
getWindowHandles()

driver.switchTo().window(windowID)

Assignment
---
https://testautomationpractice.blogspot.com/

1) provide some string search for it
2) count number of links
3) click on each link using for loop
4) get window ID's for every browser window
5) close specific browser window

Navigational Commands

1. **navigate().to(url)** → Opens a new URL (similar to get(), but more flexible since it accepts both String and URL objects).
2. **navigate().back()** → Simulates the browser **Back** button.
3. **navigate().forward()** → Simulates the browser **Forward** button.
4. **navigate().refresh()** → Refreshes the current page.

Difference:

- driver.get(url) → Only accepts String.
- driver.navigate().to(url) → Accepts both String and URL objects.

Q1: Difference between driver.get() and driver.navigate().to()?

– get() accepts only String URL, while navigate().to() accepts both String and URL object.

Q2: What is the use of getWindowHandle()?

– It returns a unique ID (String) of the current browser window.

Q3: What is the use of getWindowHandles()?

– It returns a Set<String> containing IDs of all open browser windows/tabs.

Q4: How do you switch to another window?

– driver.switchTo().window(windowID);

Q5: How do you close a specific window without quitting all?

– Loop through getWindowHandles(), check driver.getTitle() or driver.getCurrentUrl(), then call driver.close() on the matched one.

Day-24

Check boxes
Dropdowns
---
1) Drop down having select tag in DOM.
2) DropDown not having select tag in DOM (input/div) - Bootstrap dropdown
3) Auto Suggest drop down (Dynamic)

Select class

Assignment
---
1) Handle dropdown without using Select Class

Select Country & State
https://phppot.com/demo/jquery-dependent-dropdown-list-countries-and-states/

Checkboxes

In Selenium, checkboxes can be handled using click() method.

Example:

```
WebElement checkbox = driver.findElement(By.id("rememberMe"));

if(!checkbox.isSelected()) {

   checkbox.click();   // select if not already selected

}
```

Dropdowns

There are **3 types of dropdowns** we commonly deal with:

1. Dropdown with <select> tag

Handled using **Select class**.

```
WebElement country = driver.findElement(By.id("country"));
Select select = new Select(country);
select.selectByVisibleText("India");
select.selectByValue("IN");
select.selectByIndex(2);
```

---

2. Dropdown without <select> tag (Bootstrap / custom dropdown)

Here, Selenium **cannot use Select class**, so we handle it by clicking elements.

```
// Click dropdown to expand
driver.findElement(By.id("dropdownMenuButton")).click();

// Select option from list
List<WebElement> options = driver.findElements(By.xpath("//ul[@class='dropdown-menu']//a"));
for(WebElement option : options){
   if(option.getText().equals("Option 2")){
      option.click();
      break;
   }
}
```

---

3. Auto Suggest Dropdown (Dynamic Search Dropdown)

Used in Google search, IRCTC, etc. Selenium waits for dynamic suggestions.

```
driver.findElement(By.name("q")).sendKeys("Selenium");

List<WebElement> suggestions = driver.findElements(By.xpath("//ul[@role='listbox']//li"));

for(WebElement suggestion : suggestions){
   if(suggestion.getText().contains("selenium  webdriver")){
```

```
        suggestion.click();
        break;
    }
}
```

🟩 **Assignment (Handle dropdown without Select Class)**

—URL: Country & State Dropdown

Here the **country dropdown has <select> tag**, but we must handle without Select class.

Example Solution:

```
driver.get("https://phppot.com/demo/jquery-dependent-dropdown-list-countries-and-states/");

// Country dropdown
List<WebElement> countries = driver.findElements(By.xpath("//select[@id='country-list']/option"));

for(WebElement country : countries) {
    if(country.getText().equals("India")) {
        country.click();
        break;
    }
}

// State dropdown (after country selection)
Thread.sleep(2000); // wait for states to load
List<WebElement> states = driver.findElements(By.xpath("//select[@id='state-list']/option"));

for(WebElement state : states) {
    if(state.getText().equals("Maharashtra")) {
        state.click();
        break;
    }
}
```

🟠 This way, we are handling dropdown **without Select class** (using findElements + loop).

Would you like me to also give you **Interview Questions & Answers** related to Dropdowns & Checkboxes?

You said:

yes

ChatGPT said:

Perfect ' Here are some **important interview Q&A related to Checkboxes & Dropdowns** (Day-24 topic):

🟩 **Interview Questions & Answers**

1. How do you select a checkbox in Selenium?

Answer:

- Locate the checkbox using findElement().
- Use click() to select it.
- Always check isSelected() before clicking to avoid unwanted toggle.

```
WebElement checkbox = driver.findElement(By.id("accept"));

if(!checkbox.isSelected()){

   checkbox.click();

}
```

---

2. How do you verify if a checkbox is selected or not?

Answer:
Using isSelected() method.

```
boolean status = driver.findElement(By.id("accept")).isSelected();

System.out.println("Checkbox selected? " + status);
```

---

3. What are the different ways to handle a dropdown in Selenium?

Answer:
There are **3 ways**:

1. Dropdown with <select> → use Select class.
2. Dropdown without <select> → use findElements and loop through options.

3. Auto-suggest dropdown → enter text, capture dynamic options, and select.

---

4. How do you select an option from a dropdown without using the Select class?

Answer:
Use findElements() to fetch all <option> tags and loop until you find the desired value.

```
List<WebElement> options = driver.findElements(By.xpath("//select[@id='country']/option"));

for(WebElement option : options){
   if(option.getText().equals("India")){
      option.click();
      break;
   }
}
```

---

5. What is the difference between selectByValue, selectByVisibleText, and selectByIndex?

Answer:

- selectByValue("IN") → selects based on value attribute in <option>
- selectByVisibleText("India") → selects based on displayed text
- selectByIndex(2) → selects based on position (0-based index)

---

6. How do you handle a dynamic auto-suggest dropdown?

Answer:

- Type text in input field.
- Capture suggestions with findElements.
- Loop through and click matching value.

```
driver.findElement(By.name("q")).sendKeys("Selenium");

List<WebElement> suggestions = driver.findElements(By.xpath("//ul[@role='listbox']//li"));

for(WebElement s : suggestions){
   if(s.getText().contains("selenium webdriver")){
      s.click();
      break;
   }
```

}

---

7. Difference between Select dropdown and Bootstrap dropdown in Selenium?

Answer:

- **Select dropdown** → has <select> tag, handled by Select class.
- **Bootstrap dropdown** → no <select> tag, usually or , handled by click() and looping elements.

---

8. How to deselect values in multi-select dropdown?

Answer:
Only possible when dropdown has multiple="true". Use deselectByIndex, deselectByValue, or deselectAll.

Select multi = new Select(driver.findElement(By.id("multiselect")));

multi.deselectAll();

---

9. Can you select multiple values from dropdown in Selenium?

Answer:

- Yes, if <select> tag has multiple attribute.
- Otherwise, you can only select one.

---

10. What are the challenges in handling dropdowns in Selenium?

Answer:

- Dropdown without <select> tag.
- Dynamic values (auto-suggest).
- Performance issues with large dropdowns.
- Dependency dropdowns (like Country → State).

What getOptions() does

- getOptions() **returns a List<WebElement>** of all <option> elements in a <select> dropdown.
- It **does not select or click** an option by itself.

How to perform click or selection using getOptions()

You can **iterate through the list** returned by getOptions() and then **click/select** the desired option.

Example

```
WebElement countryDropdown = driver.findElement(By.id("country-list"));
Select select = new Select(countryDropdown);

// Get all options
List<WebElement> options = select.getOptions();

// Loop through and click/select desired option
for(WebElement option : options) {
   if(option.getText().equals("India")) {
      option.click();  // perform click
      break;
   }
}
```

Day-30

Alerts
---

accept()

```
driver.switchTo().alert().accept()   // close alert box using ok button
driver.switchTo().alert().dismiss()  // close alert box using cancel button
driver.switchTo().alert().getText() // capture text value from alert
driver.switchTo().alert().sendKeys()  // pass the text into inputbox inside the alert
```

how to capture alert box without using switcho().alert() ?
Ans: using explicit wait

frames

```
driver.switchTo().frame(name)
driver.switchTo().frame(id)
driver.switchTo().frame(WebElement)
driver.switchTo().frame(index)

driver.switchTo().defaultContent();
```
inner frame/nested frame

Assingments
---
1) click on login then handle alert : https://mypage.rediff.com/login/dologin

2) https://ui.vision/demo/webtest/frames/
1) switch to 5th frame
2) click on link - opens new iframe
3) switch to inner frame
4) check logo presence in the inner frame.

) **Capture alert without using switchTo().alert()**

We can use **Explicit Wait** (ExpectedConditions.alertIsPresent()).

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

Alert alert = wait.until(ExpectedConditions.alertIsPresent());

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

alert.accept();
```

Frames in Selenium

A **frame** is an HTML document embedded inside another HTML document.
To interact with elements inside a frame, you must switch into it first.

Switch to frame (different ways):

```
driver.switchTo().frame("frameName");   // by name

driver.switchTo().frame("frameId");       // by id

driver.switchTo().frame(0);               // by index

driver.switchTo().frame(WebElement);      // by locating iframe element
```

Exit from frame:

```
driver.switchTo().defaultContent();  // go to main page

driver.switchTo().parentFrame();      // go to parent frame (useful in nested)
```

Q1. What is the difference between alert, confirmation, and prompt in Selenium?

- **Alert** → only "OK"
- **Confirmation** → "OK" & "Cancel"

- **Prompt** → input field + buttons

Q2. Can we handle alerts without using switchTo().alert()?

- Yes, using **Explicit Wait** (alertIsPresent()).

Q3. How do you switch back from frame to main page?

- Using driver.switchTo().defaultContent();

Q4. How do you handle nested frames in Selenium?

- Switch step by step:

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

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

Q5. What happens if you try to access an element without switching to the frame?

- Selenium throws **NoSuchElementException** because driver is still in default page context.

An **iframe** (short for **inline frame**) is an HTML element that allows you to embed another HTML document **inside the current web page**.

Think of it as a "page within a page." The embedded page can contain its own HTML, CSS, JavaScript, and even more iframes (nested iframes).

HTML Example of an iframe

<iframe src="https://www.example.com" width="600" height="400"></iframe>

- src → URL of the page to embed

Day-32

Web Tables
---
1) static web table
2) Dynamic web table
3) table with pagination

r=2
c=3

//table[@name='BookTable']//tr["+r+"]//td["+c+"]

Assignment
---
1) https://blazedemo.com/

Day-33

Dynamic table with pagination

String s="Showing 1 to 10 of 19081 (13232 Pages)"

s.substring(s.indexOf("(")+1,s.indexOf("Pages")-1) --> 1909

Assignment
---
tables
---

1) Orangehrm table:
    https://opensource-demo.orangehrmlive.com/web/index.php/admin/viewSystemUsers

2) Dynamic Table
    https://practice.expandtesting.com/dynamic-table

3) Paginatin table
    https://testautomationpractice.blogspot.com/

4) Pagination & Dynamic table (Web Scrapping)
    https://money.rediff.com/gainers/bse/daily/groupall

    Test case 1:
    ---
    1) Read All data on Daily, Weekly and Monthly wise (From Gainers)
    2) Print Company, Group and %change.

    Test case 2:
    ---
    1) Read data from all the groups on Daily, Weekly and Monthly wise (From Gainers)
    2) In each group print company name which is having lowest %Change

1. Basics of Web Tables

Q1. What is a static web table?
A: A static web table is a table where data is fixed and does not change dynamically. You can access any cell using row and column indices.

//table[@name='BookTable']//tr[2]//td[3] // Access row 2, column 3

Q2. What is a dynamic web table?
A: A dynamic web table is a table where the data changes frequently or loads via AJAX. Rows and columns may not have fixed indices. You need to loop through <tr> and <td> elements to access the data dynamically.

Q3. How do you handle pagination in web tables?
A:

- Identify total number of pages.
- Loop through pages, click next or page number.
- Read data from each page.
- Example:

```
for(int p=1; p<=totalPages; p++){
   driver.findElement(By.xpath("//a[text()='"+p+"']")).click();
   // read table rows
}
```

```
Day-35
------
Mouse actions
---------

Mouse hover --- moveToElement(element)
right click --   contextClick(element)
double click -- doubleClick(element)
drag and drop  -- dragAndDrop(source, target)




Actions - pre defined class provided in selenium


build() - create an action
perform() - complete an action

getText()  Vs getAttribute(attribute)
```

<input id="xyz"> welcome </input>

getText() --> returns the inner text --> welcome
getAttribute("id") -- returns the value of attribute ---> xyz

<input value="welcome"></input>
getAttribute("value") ---> welcome

getText() ----> returns inner text of the web element
getAttribute(attribute) -- >> returns value of attribute

Actions Vs Action

Actions -- class, will be used to perform mouse actions,
Action --- interface, will be used to store created actions.

Assignments
---
1) double click & Drag and drop
https://testautomationpractice.blogspot.com/
2) drag and drop
http://demo.guru99.com/test/drag_drop.html

Q1. What are mouse actions in Selenium?
A: Mouse actions are user interactions like hover, right-click, double-click, and drag-and-drop. They are performed using the **Actions class** in Selenium.

Q2. What is the difference between Actions class and Action interface?
A:

Term	Description
Actions	A class in Selenium used to perform complex mouse/keyboard actions.
Action	An interface used to store a single, built action that can be executed using perform().

Q3. How do you perform a mouse hover in Selenium?
A:

Actions actions = new Actions(driver);

actions.moveToElement(element).perform(); // hover on element

---

Q4. How do you perform a right-click (context click)?
A:

Actions actions = new Actions(driver);

actions.contextClick(element).perform(); // right click on element

---

Q5. How do you perform a double-click?
A:

Actions actions = new Actions(driver);

actions.doubleClick(element).perform(); // double click on element

---

Q6. How do you perform drag and drop?
A:

Actions actions = new Actions(driver);

actions.dragAndDrop(sourceElement, targetElement).perform(); // drag and drop

- Alternative method using clickAndHold, moveToElement, release:

actions.clickAndHold(sourceElement).moveToElement(targetElement).release().perform();

Q7. What are build() and perform() methods in Actions class?
A:

- build() → used to compile all the actions into a single action.
- perform() → used to execute the compiled action.

Actions actions = new Actions(driver);

Action action = actions.moveToElement(element).build();

action.perform();

---

2. getText() vs getAttribute()

Q8. What is the difference between getText() and getAttribute() in Selenium?

Method	Description	Example
getText()	Returns the inner text of a web element.	Hello → element.getText() returns "Hello"
getAttribute()	Returns the value of a specified attribute of an element.	<input id="name" value="John"> → element.getAttribute("value") returns "John"

Example:

<input id="xyz">Welcome</input>

element.getText(); // Returns "Welcome"

element.getAttribute("id"); // Returns "xyz"

<input value="Hello"></input>

element.getAttribute("value"); // Returns "Hello"

---

3. Scenario-based Questions

Q9. How do you perform drag and drop on the Guru99 demo page?
A:

WebElement source = driver.findElement(By.id("credit2")); // BANK

WebElement target = driver.findElement(By.id("bank")); // Debit Side

Actions actions = new Actions(driver);

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

Q10. How do you double-click on an element using Selenium?
A:

WebElement button = driver.findElement(By.id("dblClickBtn"));

Actions actions = new Actions(driver);

actions.doubleClick(button).perform();

Q11. How do you perform multiple mouse actions together?
A:

- You can chain multiple actions before calling build() and perform().

Actions actions = new Actions(driver);

actions.moveToElement(menu).click(subMenu).doubleClick(button).build().perform();

Q12. Difference between click() and doubleClick()?

- click() → Single click on an element.
- doubleClick() → Performs two consecutive clicks, used for special events (like opening an item).

Here's a clear explanation of the **difference between Action and Actions in Selenium**:

Feature	**Actions (Class)**	**Action (Interface)**
Type	Class	Interface
Purpose	Used to **create complex user interactions** (mouse & keyboard actions)	Used to **store a single compiled action** that can be executed
Usage	Provides methods like moveToElement(), click(), doubleClick(), dragAndDrop(), etc.	Represents a **single action** which is returned by build() of Actions
Execution	Cannot be executed directly, needs to call perform() on an Action object	Executed using perform() method
Example	Actions actions = new Actions(driver); actions.moveToElement(element);	Action action = actions.moveToElement(element).build(); action.perform();

Day-37

JavascriptExecutor

JavascriptExecutor - interface

executeScript() - we can execute javascript statements

sendKeys()
click()

Scrolling bar

Upload files

Q1. What is JavascriptExecutor in Selenium?

Answer:
JavascriptExecutor is a **Selenium interface** that allows you to execute **JavaScript code** directly in the browser. It is useful when Selenium WebDriver methods like click() or sendKeys() do not work properly due to hidden elements or dynamic content.

Q2. How do you use JavascriptExecutor in Selenium?

Answer:

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript("alert('Hello World')");

- executeScript() is used to run JavaScript code.
- You can pass **arguments**, like web elements, to the script.

Q3. How can JavascriptExecutor be used to click on a hidden element?

Answer:

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

JavascriptExecutor js = (JavascriptExecutor) driver;

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

- arguments[0] refers to the web element passed from Java.

Q4. How can you use JavascriptExecutor to send keys?

Answer:

WebElement input = driver.findElement(By.id("username"));

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript("arguments[0].value='admin';", input);

- This sets the value of the input field via JavaScript.

Q5. How do you scroll using JavascriptExecutor?

Answer:

1. Scroll down by pixels:

js.executeScript("window.scrollBy(0,500)");

2. Scroll to the bottom of the page:

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

3. Scroll to a specific element:

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

Q6. How can JavascriptExecutor be used to upload files?

Answer:

```
WebElement upload = driver.findElement(By.id("fileUpload"));

js.executeScript("arguments[0].style.display='block';", upload);

upload.sendKeys("C:\\path\\to\\file.txt");
```

- Sometimes file input fields are hidden, so JS is used to make them visible before uploading.

Q7. Why and when do you use JavascriptExecutor over Selenium methods?

Answer:

- When click() or sendKeys() fails due to **hidden, disabled, or overlapped elements**.
- For scrolling pages or specific elements.
- For **highlighting elements**, manipulating attributes, or extracting values from dynamic web pages.

```
Day-38
------
How to capture screenshots
----
1) full page
2) specific area of the page
3) web element


ChromeOptions
EdgeOptions
FirefoxOptions
etc...

ChromeOptions
-------
Headless testing
-----------------------------------
ChromeOptions options=new ChromeOptions();
options.addArguments("--headless=new");

advantages
```

---
1) we can do multiple tasks(since execution happend backend)
2) faster execution

disadvantage
---
1) user cannot see the actions on the page. so he cannot understand flow/functionality of the test.

SSL Handling

```
ChromeOptions options=new ChromeOptions();
options.setAcceptInsecureCerts(true); // accepts SSL certificates
```

to remove "Chrome is being controlled by automated test software"

```
ChromeOptions options=new ChromeOptions();
options.setExperimentalOption("excludeSwitches", new String[] {"enable-automation"});
```

To run the tests in Incognito mode

```
ChromeOptions options=new ChromeOptions();
options.addArguments("--incognito");
```

Enable browser extension in run time
---
step 1: Add CRX Extractor/Downloader to chrome Browser (manually)
Step2 : Add SelectorsHub plugin to chrome browser (manually)
step3 : Capture crx file for selectorshub extension
Step4: pass crx file path in automation script in Chrome Options

```
ChromeOptions options=new ChromeOptions();
options.addExtensions(new File("C:\\Drivers\\SelectorsHub.crx"));
```

How to block ads on the page
---
uBlock-Origin
AdBlocker

```
public class CaptureScreenshots {

    public static void main(String[] args) {
```

```
        WebDriver driver=new ChromeDriver();
        driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));

        driver.get("https://demo.nopcommerce.com/");
        driver.manage().window().maximize();

        //1) full page screenshot
        /*TakesScreenshot ts=(TakesScreenshot)driver;
        File sourcefile=ts.getScreenshotAs(OutputType.FILE);
        File targetfile=new File(System.getProperty("user.dir")+"\\screenshots\\fullpage.png");
        sourcefile.renameTo(targetfile); // copy sourcefile to target file
        */

        //2) capture the screenshot of specific section
        /*WebElement featuredProducts=driver.findElement(By.xpath("//div[@class='product-grid
home-page-product-grid']"));

        File sourcefile=featuredProducts.getScreenshotAs(OutputType.FILE);
        File targetfile=new
File(System.getProperty("user.dir")+"\\screenshots\\featredproducts.png");
        sourcefile.renameTo(targetfile); // copy sourcefile to target file
        */

        //3) capture the screenshot of webelement
        WebElement logo=driver.findElement(By.xpath("//img[@alt='nopCommerce demo store']"));
        File sourcefile=logo.getScreenshotAs(OutputType.FILE);
        File targetfile=new File(System.getProperty("user.dir")+"\\screenshots\\logo.png");
        sourcefile.renameTo(targetfile); // copy sourcefile to target file

        //driver.quit();
    //full page screenshot
        File screenshot = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);
        FileUtils.copyFile(screenshot, new File("screenshot.png"));
    }
        //element screenshot
        WebElement e=driver.findElement(By.xpath("/html/body/div[2]/div[4]"));
}       File elementscreenshot = e.getScreenshotAs(OutputType.FILE);
        FileUtils.copyFile(elementscreenshot, new File("elementscreenshot.png"));

        driver.quit();
```

Q2. What is ChromeOptions in Selenium?

Answer:

ChromeOptions is a **class** used to customize Chrome browser behavior during automation, such as enabling headless mode, incognito mode, SSL handling, adding extensions, and removing automation banners.

So ChromeOptions extends MutableCapabilities, which implements the Capabilities interface.

Q3. How can you run tests in headless mode?

Answer:

```
ChromeOptions options = new ChromeOptions();
options.addArguments("--headless=new");  // for Chrome 109+
WebDriver driver = new ChromeDriver(options);
```

Advantages:

- Faster execution.
- Can run multiple tasks in the background.

Disadvantages:

- User cannot see browser actions; cannot verify flow manually.

Q4. How can you handle SSL certificate errors?

Answer:

```
ChromeOptions options = new ChromeOptions();
options.setAcceptInsecureCerts(true);  // Accepts insecure SSL certificates
```

Q5. How do you remove "Chrome is being controlled by automated test software"?

Answer:

```
ChromeOptions options = new ChromeOptions();
options.setExperimentalOption("excludeSwitches",  new  String[]{"enable-automation"});
```

Q6. How to run Chrome in Incognito mode using Selenium?

Answer:

```
ChromeOptions options = new ChromeOptions();
options.addArguments("--incognito");
```

Q7. How to enable Chrome extensions at runtime in Selenium?

Answer:

```
ChromeOptions options = new ChromeOptions();
options.addExtensions(new  File("C:\\Drivers\\SelectorsHub.crx"));
```

Steps:

1. Install extension manually and get CRX file.
2. Add CRX path to ChromeOptions using addExtensions().

Q8. How to block ads while running Selenium tests?

Answer:

- Use ad-blocker extensions like **uBlock-Origin** or **AdBlocker** via ChromeOptions.
- Example:

```
options.addExtensions(new  File("C:\\Drivers\\uBlock-Origin.crx"));
```

Q9. Difference between TakesScreenshot of full page vs web element

Answer:

Type	Captures	Method
Full page	Entire webpage	driver.getScreenshotAs(OutputType.FILE)
Web element	Specific element	element.getScreenshotAs(OutputType.FILE)

```
Day-39
------
Broken links
------


1) Link    href="https://xyz.com"

2) https://xyz.com ---> server ---> status code

3)  status code>=400   broken link
 status code <400   not a broken link

Shandow DOM
-------------
XPath cannot handle shadow dom elements.
Only CSS can handle shadow dom elements.


//1) This Element is inside single shadow DOM.

SearchContext shadow = driver.findElement(By.cssSelector("#shadow-root")).getShadowRoot();
Thread.sleep(1000);
shadow.findElement(By.cssSelector("#shadow-element"));
```

```
//2) This Element is inside 2 nested shadow DOM.

SearchContext shadow0 = driver.findElement(By.cssSelector("#shadow-root")).getShadowRoot();
Thread.sleep(1000);
SearchContext shadow1 = shadow0.findElement(By.cssSelector("#inner-shadow-
dom")).getShadowRoot();
Thread.sleep(1000);
shadow1.findElement(By.cssSelector("#nested-shadow-element"));



//3) This Element is inside 3 nested shadow DOM.

SearchContext shadow0 = driver.findElement(By.cssSelector("#shadow-root")).getShadowRoot();
Thread.sleep(1000);

SearchContext shadow1 = shadow0.findElement(By.cssSelector("#inner-shadow-
dom")).getShadowRoot();
Thread.sleep(1000);

SearchContext shadow2 = shadow1.findElement(By.cssSelector("#nested-shadow-
dom")).getShadowRoot();
Thread.sleep(1000);

shadow2.findElement(By.cssSelector("#multi-nested-shadow-element"));


Handle svg elements
----------------
//*[name()='path' and contains(@d,'M256,302c8')]
//a[normalize-space()='']//*[name()='svg']
//*[name()='path' and contains(@d,'M256.264,0')]



More videos
------
1)Shadow dom
https://youtu.be/bpzyjNZ0jaw


2) SVG Elements, iFrames and Shadow DOM Elements
https://youtu.be/v-6JerEgyMM
```

Q1. What is a broken link and how can you verify it in Selenium?

Answer:

- A **broken link** is a hyperlink that points to a URL that does not work or returns an error.
- In Selenium, you can verify broken links by checking the **HTTP response status code** of the URL.

Logic:

status code >= 400 --> broken link

status code < 400 --> valid link

Example:

```
String url = link.getAttribute("href");

HttpURLConnection conn = (HttpURLConnection) new URL(url).openConnection();

conn.setRequestMethod("HEAD");

conn.connect();

int responseCode = conn.getResponseCode();


if(responseCode >= 400) {

   System.out.println(url + " is a broken link");

} else {

   System.out.println(url + " is a valid link");

}
```

Q2. What is Shadow DOM?

Answer:

- **Shadow DOM** is a web standard that allows encapsulation of HTML elements in a **separate DOM tree**, hiding them from the main document DOM.
- **XPath cannot access Shadow DOM elements**, only **CSS selectors** can be used.

Q3. How do you access a single shadow DOM element in Selenium?

Answer:

```
SearchContext shadow = driver.findElement(By.cssSelector("#shadow-root")).getShadowRoot();

Thread.sleep(1000);

shadow.findElement(By.cssSelector("#shadow-element"));
```

Q4. How do you access nested shadow DOM elements?

Two nested shadow DOM:

```
SearchContext shadow0 = driver.findElement(By.cssSelector("#shadow-root")).getShadowRoot();

Thread.sleep(1000);

SearchContext shadow1 = shadow0.findElement(By.cssSelector("#inner-shadow-dom")).getShadowRoot();

Thread.sleep(1000);

shadow1.findElement(By.cssSelector("#nested-shadow-element"));
```

Three nested shadow DOM:

```
SearchContext shadow0 = driver.findElement(By.cssSelector("#shadow-root")).getShadowRoot();

SearchContext shadow1 = shadow0.findElement(By.cssSelector("#inner-shadow-dom")).getShadowRoot();

SearchContext shadow2 = shadow1.findElement(By.cssSelector("#nested-shadow-dom")).getShadowRoot();

shadow2.findElement(By.cssSelector("#multi-nested-shadow-element"));
```

Q5. How do you handle SVG elements in Selenium?

Answer:

- Use *[name()='tagName'] in **XPath** because SVG uses XML namespaces.

Examples:

```
//*[name()='path' and contains(@d,'M256,302c8')]

a[normalize-space()='']//*[name()='svg']

//*[name()='path' and contains(@d,'M256.264,0')]
```

Q6. Can XPath access Shadow DOM elements?

Answer:

- No, **XPath cannot access Shadow DOM elements**. Only **CSS selectors** along with getShadowRoot() can be used

Q: What is WebElement in Selenium?

Answer:

- WebElement is an **interface in Selenium** that represents an **HTML element on a web page**.
- Every element on a webpage, such as a button, textbox, link, checkbox, dropdown, or image, can be represented as a WebElement.

- Using the WebElement interface, we can **perform actions on the element** like click, type text, get text, get attributes, check if displayed/enabled/selected, etc.

Apache POI

```
Day-40
------
Apache poi  library

dependencies in pom.xml
--------------------
        <!-- https://mvnrepository.com/artifact/org.apache.poi/poi -->
        <dependency>
            <groupId>org.apache.poi</groupId>
            <artifactId>poi</artifactId>
            <version>5.2.5</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/org.apache.poi/poi-ooxml -->
        <dependency>
            <groupId>org.apache.poi</groupId>
            <artifactId>poi-ooxml</artifactId>
            <version>5.2.5</version>
        </dependency>

Excel  File--->Workbook--->Sheets--->Rows --- Cells

FileInputStream - reading
FileOutputStream   - writing

XSSFWorkbook ----- workbook
XSSFSheet  -- sheet
XSSFRow  - row
XSSFCell  -- cell

1) reading data from excel
2) writing data in to excel.


package day40;

import java.io.FileInputStream;
import java.io.IOException;

import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFRow;
```

```
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

//Excel  File--->Workbook--->Sheets--->Rows --- Cells

public class ReadingDataFromExcel {

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

      FileInputStream file=new
FileInputStream(System.getProperty("user.dir")+"\\testdata\\data.xlsx");

      XSSFWorkbook workbook=new XSSFWorkbook(file);

      XSSFSheet  sheet=workbook.getSheet("Sheet1");  //  XSSFSheet  sheet=workbook.getSheetAt
(0);

      int totalRows=sheet.getLastRowNum();

      int totalCells=sheet.getRow(0).getLastCellNum();

      System.out.println("number of rows:"+ totalRows); //5 row-1
      System.out.println("number of cells:"+ totalCells);  //4

      for(int r=0;r<=totalRows;r++)
      {
         XSSFRow currentRow=sheet.getRow(r);

         for(int c=0;c<totalCells;c++)
         {
            XSSFCell cell=currentRow.getCell(c);
            System.out.print(cell.toString()+"\t");

         }
         System.out.println();
      }

      workbook.close();
      file.close();

   }

}

//Writing
package day40;

import java.io.FileOutputStream;
import java.io.IOException;

import org.apache.poi.xssf.usermodel.XSSFCell;
```

```
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

//Excel  File--->Workbook--->Sheets--->Rows --- Cells

public class WritingDatainSpecifRowAndCell {

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

      FileOutputStream file=new
FileOutputStream(System.getProperty("user.dir")+"\\testdata\\myfileRandom.xlsx");

      XSSFWorkbook workbook=new XSSFWorkbook();

      XSSFSheet sheet=workbook.createSheet("Data");

      XSSFRow row=sheet.createRow(3);
      XSSFCell cell=row.createCell(4);

      cell.setCellValue("WELCOME");



      workbook.write(file);
      workbook.close();
      file.close();

      System.out.println("File is createaed.....");

   }

}

package day40;

import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Scanner;

import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

//Excel  File--->Workbook--->Sheets--->Rows --- Cells

public class WritingDynamicDataIntoExcel {

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

```
        FileOutputStream file=new
FileOutputStream(System.getProperty("user.dir")+"\\testdata\\myfile_dynamic.xlsx");

        XSSFWorkbook workbook=new XSSFWorkbook();

        XSSFSheet sheet=workbook.createSheet("DynamicData");

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter how many rows?");
        int noOfrows=sc.nextInt();

        System.out.println("Enter how many cells?");
        int noOfcells=sc.nextInt();


        for(int r=0;r<=noOfrows;r++)
        {
           XSSFRow currentRow=sheet.createRow(r);

           for(int c=0;c<noOfcells;c++)
           {
              XSSFCell cell=currentRow.createCell(c);
              cell.setCellValue(sc.next());
           }
        }


        workbook.write(file);  //attach workbook to the file
        workbook.close();
        file.close();

        System.out.println("File is createaed..... ");

    }

}
```

Q1. What is Apache POI?

Ans:
Apache POI (Poor Obfuscation Implementation) is an open-source Java library provided by Apache to **read, write, and modify Microsoft Office files**, including Excel (.xls and .xlsx). In Selenium

automation, we often use POI for **data-driven testing** (reading test data from Excel and writing results back).

Q2. What dependencies are required for Apache POI in Maven?

Ans:
We need:

- poi (for older Excel formats, .xls)
- poi-ooxml (for newer Excel formats, .xlsx)

Example:

```
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi</artifactId>
  <version>5.2.5</version>
</dependency>
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi-ooxml</artifactId>
  <version>5.2.5</version>
</dependency>
```

Q3. What are the main classes in Apache POI for Excel handling?

Ans:

- **XSSFWorkbook** → Represents the workbook (.xlsx file).
- **XSSFSheet** → Represents a sheet inside the workbook.
- **XSSFRow** → Represents a row inside the sheet.
- **XSSFCell** → Represents a cell inside a row.

Q4. How do you read data from an Excel file using Apache POI?

Ans:

```
FileInputStream fis = new FileInputStream("data.xlsx");
XSSFWorkbook workbook = new XSSFWorkbook(fis);
XSSFSheet sheet = workbook.getSheet("Sheet1");
```

```
XSSFRow row = sheet.getRow(0);
XSSFCell cell = row.getCell(0);

String value = cell.getStringCellValue();
System.out.println("Cell Value: " + value);

workbook.close();
fis.close();
```

Q5. How do you write data into an Excel file using Apache POI?

Ans:

```
FileInputStream fis = new FileInputStream("data.xlsx");
XSSFWorkbook workbook = new XSSFWorkbook(fis);
XSSFSheet sheet = workbook.getSheet("Sheet1");

XSSFRow row = sheet.createRow(1);
XSSFCell cell = row.createCell(0);
cell.setCellValue("Test Passed");

FileOutputStream fos = new FileOutputStream("data.xlsx");
workbook.write(fos);

workbook.close();
fis.close();
fos.close();
```

Q6. What is the difference between HSSF and XSSF in Apache POI?

Ans:

- **HSSF (Horrible Spreadsheet Format)** → Handles older .xls (Excel 97–2003) files.
- **XSSF (XML Spreadsheet Format)** → Handles newer .xlsx (Excel 2007+) files.

Q7. Why do we use Apache POI in Selenium automation?

Ans:
For **Data-Driven Testing (DDT)**:

- Store test cases and input data in Excel.
- Read input data for test execution.
- Write results (Pass/Fail) back to Excel.

This makes tests flexible and reusable without hardcoding values.

Q8. Can we handle both .xls and .xlsx files with Apache POI?

Ans:
Yes.

- Use **HSSFWorkbook, HSSFSheet, HSSFRow, HSSFCell** for .xls.
- Use **XSSFWorkbook, XSSFSheet, XSSFRow, XSSFCell** for .xlsx.

```
Day-41
------


Data Driven testing
-------------

1) Functionality ( Test case)
2) Prepare test data in excel
3) develope automation script

Reading properties file
---


Assignment
--------
Data driven testing

https://www.cit.com/cit-bank/resources/calculators/certificate-of-deposit-calculator

package day41;

import java.io.FileInputStream;
import java.io.IOException;
import  java.util.Collection;
import java.util.Properties;
import java.util.Set;
```

```
public class ReadingPropertiesFile {

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

      //location of properties file
      FileInputStream file=new
FileInputStream(System.getProperty("user.dir")+"\\testdata\\config.properties");


      //Loading properties file
      Properties propertiesobj=new Properties();
      propertiesobj.load(file);

      //Reading data from properties file

      String url=propertiesobj.getProperty("appurl");
      String email=propertiesobj.getProperty("email");
      String pwd=propertiesobj.getProperty("password");
      String orid=propertiesobj.getProperty("orderid");
      String custid=propertiesobj.getProperty("customerid");

      System.out.println(url+" "+email+"  "+pwd+"  "+orid+" "+custid);

      //Reading all the keys from properties file

      //Set<String> keys=propertiesobj.stringPropertyNames();
      //System.out.println(keys); //[password, orderid, customerid, appurl, email]

      Set<Object> keys =propertiesobj.keySet();
      System.out.println(keys); //[password, orderid, customerid, appurl, email]

      //Reading all teh values from properties file
      Collection<Object> values=propertiesobj.values();
      System.out.println(values); //[abcxyz, 123, 234, https://demo.opencart.com/,
abc@gmail.com]

      file.close();

   }

}
```

Q1. What is Data-Driven Testing (DDT) in Selenium?

Ans:
Data-Driven Testing is a testing approach where test data is stored outside the test scripts (e.g.,

Excel, CSV, database, properties file), and the same test case is executed multiple times with different sets of input data.

- Helps avoid hardcoding values inside scripts and supports large-scale testing.

Q2. What tools/libraries do we use for Data-Driven Testing in Selenium?

Ans:

- **Apache POI** → To read/write data from Excel.
- **Properties File** → To store configuration data (like URL, username, password).
- **TestNG DataProvider** → To supply test data programmatically.

Q3. What is the use of a Properties File in Selenium?

Ans:
A **properties file** stores configuration in key-value pairs (e.g., URL, browser, credentials). This helps in:

- Keeping scripts clean and maintainable.
- Updating configurations without modifying code.

Example config.properties:

```
url=https://demo.testsite.com

browser=chrome

username=admin

password=admin123
```

Reading in Selenium:

```
FileInputStream fis = new FileInputStream("config.properties");

Properties prop = new Properties();

prop.load(fis);


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

System.out.println("App URL: " + url);
```

Q4. What is the difference between Data-Driven Testing and Keyword-Driven Testing?

Ans:

- **Data-Driven Testing** → Focuses on separating test data from test scripts.

- **Keyword-Driven Testing** → Focuses on separating actions (keywords) from test scripts, usually via Excel (e.g., login, click, verify).

Q5. How would you implement Data-Driven Testing in Selenium?

Ans:

1. Prepare test data in Excel (using Apache POI).
2. Create a utility to read Excel data.
3. Pass the data into the test case using loops/TestNG DataProvider.
4. Execute the same test case with different data sets.

Invoke Application:

What is WebDriverManager?

WebDriverManager is an open-source Java library that carries out download, setup, and maintenance of different browser drivers automatically in different OS.

Why is WebDriverManager needed?

- Download, setup, maintenance the different browser drivers automatically
- Download the latest version of browser drivers automatically
- Eliminates the needs to store browser drivers in local

How to instantiate a browser using WebDriverManager in Selenium?

```
WebDriverManager.chromedriver().setup();
WebDriverManager.firefoxdriver().setup();
WebDriverManager.iedriver().setup();
WebDriverManager.edgedriver().setup();
```

How to instantiate a specific driver version in WebDriverManager?

```
WebDriverManager.chromedriver().driverVersion("85.0.4183.38").setup();
```

How to instantiate a specific browser version in WebDriverManager?

```
WebDriverManager.chromedriver().browserVersion("83.0.4103").setup();
```

How to instantiate a platform version (x32 or x64) using in WebDriverManager?

```
WebDriverManager.chromedriver().arch32().setup();

WebDriverManager.chromedriver().arch64().setup();
```

How to set a proxy username and password?

```
WebDriverManager.chromedriver()
        .version("83.0.0")
        .arch32()
        .proxy("proxyhostname:80")
        .proxyUser("username")
        .proxyPass("password")
.setup();
```

==========

```
//Invoke Chrome driver
System.setProperty("webdriver.chrome.driver", "D:\\chromedriver.exe");        //Set Browser Driver Path
WebDriver driver = new ChromeDriver();                                        //Set Chrome driver object
driver.manage().deleteAllCookies();                                           //Delete all the cookies before start the application
driver.manage().window().maximize();                                          //Maximize the application

driver.get("https://www.guru99.com/");                                        //Launch Application
```

==========

```
//Invoke IE driver
System.setProperty("webdriver.ie.driver", "D:\\IEDriverServer.exe");      //Set Web Driver Path
WebDriver driver = new InternetExplorerDriver();                          //Set IE driver object

driver.manage().deleteAllCookies();                                       //Delete all the cookies before start the application
```

```
driver.manage().window().maximize();                    //Maximize the application

driver.get("https://www.guru99.com/");                  //Launch Application
```

==========

```
//Invoke FireFox driver if the Firefox version less than or equal to 47.0
WebDriver driver = new FirefoxDriver();                 //Set FireFox driver object

driver.manage().deleteAllCookies();                     //Delete all the cookies before start
the application
driver.manage().window().maximize();                    //Maximize the application

driver.get("https://www.guru99.com/");                  //Launch Application
```

==========

What is Chrome Options Class?

It helps to invoke the customized version of ChromeDriver

It helps to perform the following options

- **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

```
ChromeOptions options = new ChromeOptions();

options.addArguments("--incognito");    //Open chrome in Incognito mode
options.addArguments("--headless");     //Open chrome in Headless mode
options.addArgument("start-maximized");         //Open chrome in Maximize mode
```

```
options.addExtensions(new File("Path to CRX File"));       // activate ad blocker extension
```

==========

Text Box Controls:

```
//Clear value from text box
driver.findElement(By.id("Email")).clear();

//Enter value into text box
driver.findElement(By.id("Email")).sendKeys("senthil.kumar@training.test");

//How to enter the text in caps lock?
driver.findElement(By.id("")).sendKeys(Keys.SHIFT,"test");
```

==========

Button Controls:

```
//Click button
driver.findElement(By.xpath("//a[@class='ico-login']")).click();

//How to hit enter from web driver command?
driver.findElement(By.id("")).sendKeys(Keys.ENTER);
```

==========

Drop down Controls:

```
//Select values using different methods
Select drpCountry = new Select(driver.findElement(By.name("country")));
drpCountry.selectByVisibleText("ANTARCTICA");//Select value by using visible text
drpCountry.selectByIndex(1);     //Select value by using index
drpCountry.selectByValue("234");          //Select value by using  of its "value" attribute
```

```
//Select multiple selections at a time

Select drpCountry = new Select(driver.findElement(By.name("country")));

drpCountry.selectByVisibleText("ANTARCTICA");//Select value by using visible text

drpCountry.selectByIndex(2);     //Select value by using index


//Other methods in select class

selectByVisibleText()/ deselectByVisibleText()     //selects/deselects an option by its displayed text

selectByValue()/ deselectByValue()                                    //selects/deselects an option by the value of its "value" attribute

selectByIndex()/ deselectByIndex()                                    //selects/deselects an option by its index

isMultiple()                                                                          //returns TRUE if the drop-down element allows multiple selection at a time; FALSE if otherwise

deselectAll()                                                                         //        deselects all previously selected options
```

==========

Radio Button Controls:

```
//Select radio button

driver.findElement(By.id("radioButton1")).click();
```

==========

Check Box Controls:

```
//Select check box

driver.findElement(By.id("checkBox")).click();
```

==========

Image Controls:

//Get back ground color of image

driver.findElement(By.id("image")).getCssValue(“background-color”);

Note: getCssValue mostly applicable only for Images

Size:

//Get size of the button

WebElement button = driver.findElement(By.id("image"));

Dimension d = button.getSize();

d.height; //This will help to get the height of the button

d.width; //This will help to get the height of the button

Location:

//Get location of the button

WebElement button = driver.findElement(By.id("image"));

Position p = button.getLocation()

p.x; //This will help to get the x

p.y; //This will help to get the y

==========

Take Screenshot:

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

FileUtils.copyFile(screenShotTempPath, new File("E:\\workspace4\\Selenium_SBA\\src\\results\\"+strStepName+".png"));

==========

Web Table Controls:

Web Table Rows are dynamic and Columns are constant:

Sample Table

Structure	Country	City	Height	Built	Rank	...
Burj Khalifa	UAE	Dubai	829m	2010	1	details
Clock Tower Hotel	Saudi Arabia	Mecca	601m	2012	2	details
Taipei 101	Taiwan	Taipei	509m	2004	3	details
Financial Center	China	Shanghai	492m	2008	4	details
Total	4 buildings					

//Sample Table

List<WebElement> rows = driver.findElements(By.xpath(".//table[@summary='Sample Table']//tbody/tr")); //Get row count from table

for (int i=0;i<rows.size();i++) {

WebElement objColumn = driver.findElement(By.xpath(".//table[@summary='Sample Table']//tbody/tr["+(i+1)+"]/td[2]")); //Get the 'City' column value

if (objColumn.getText().trim().equals("Shanghai")) //Verify the run time 'City' column value with user input

{

WebElement linkDetail = driver.findElement(By.xpath(".//table[@summary='Sample Table']//tbody/tr["+(i+1)+"]/td[6]/a"));

linkDetail.click();//Click the detail linke from a particular row

break;

}

}

Web Table Rows and Columns are dynamic:

1	2	3
4	5	6
5	6	
6	7	8
7		

//To locate table.

WebElement mytable = driver.findElement(By.xpath(".//table[@summary='Sample

```
Table']//tbody"));

//To locate rows of table.

List < WebElement > rows_table = mytable.findElements(By.tagName("tr"));

//To calculate no of rows In table.

int rows_count = rows_table.size();

//Loop will execute till the last row of table.

for (int row = 0; row < rows_count; row++) {

//To locate columns(cells) of that specific row.

List < WebElement > Columns_row = rows_table.get(row).findElements(By.tagName("td"));

//To calculate no of columns (cells). In that specific row.

int columns_count = Columns_row.size();

System.out.println("Number of cells In Row " + row + " are " + columns_count);

//Loop will execute till the last cell of that specific row.

for (int column = 0; column < columns_count; column++) {

// To retrieve text from that specific cell.

String celtext = Columns_row.get(column).getText();

System.out.println("Cell Value of row number " + row + " and column number " + column + " Is " +
celtext);

}

}
```

Get Maximum of all the Values in a Column of Dynamic Table

Company	Group	Prev Close (Rs)	Current Price (Rs)	% Change
Wockhardt Ltd.	A	839.50	909.00	+ 8.28
Thermax	A	822.90	889.75	+ 8.12
Prestige Estates Pro	A	183.80	198.40	+ 7.94
Den Networks Ltd.	A	72.55	78.20	+ 7.79
GE T&D India	A	324.80	345.30	+ 6.31
Repco Home Finance L	A	831.90	881.50	+ 5.96
Firstsource Solution	A	41.65	43.95	+ 5.52
Jammu & Kashmir Bank	A	78.20	82.35	+ 5.31
TVS Motor Co. Ltd.	A	321.10	338.10	+ 5.29
Arvind Ltd.	A	315.55	331.25	+ 4.98
Balrampur Chini	A	102.40	107.05	+ 4.54
Sintex Industrie	A	78.65	82.15	+ 4.45
Alembic Pharmaceutic	A	628.80	655.00	+ 4.17
Edelweiss Fin. Ser	A	106.90	111.20	+ 4.02
Mangalore Refine	A	82.80	85.85	+ 3.68

```
String max;

double m=0,r=0;


//No. of Columns

List  col = driver.findElements(By.xpath(".//*[@id='leftcontainer']/table/thead/tr/th"));

System.out.println("Total No of columns are : " +col.size());


//No.of rows

List  rows = driver.findElements(By.xpath (".//*[@id='leftcontainer']/table/tbody/tr/td[1]"));

System.out.println("Total No of rows are : " + rows.size());


for (int i =1;i<rows.size();i++)

{

max= driver.findElement(By.xpath("html/body/div[1]/div[5]/table/tbody/tr[" + (i+1)+
"]/td[4]")).getText();

NumberFormat f =NumberFormat.getNumberInstance();

Number num = f.parse(max);

max = num.toString();
```

```
m = Double.parseDouble(max);
if(m>r)
{
r=m;
}
}
System.out.println("Maximum current price is : "+ r);
```

Alert & Popup:

```
driver.switchTo().alert().accept();                    // Accept the pop up to proceed
driver.switchTo().alert().dismiss();                   // Reject the pop up to proceed
driver.switchTo().alert().getText();                   // To capture the alert message.
driver.switchTo().alert().sendKeys("Text");            // To send some data to alert box.
```

==========

Mouse Over Controls:

```
//Mouse Over any element
WebElement link_Home = driver.findElement(By.linkText("Home")); //Create object instance

Actions builder = new Actions(driver);    //Instantiate a new Actions object
Action mouseOverHome = builder
                .moveToElement(link_Home)
                .build(); //Instantiate an Action using the Actions object
mouseOverHome.perform();      //Use the perform() method when executing the Action object
```

Mouse Click & Keyboard Event – Series of actions:

```
//Mouse Over any element and do the multiple actions
```

```
WebElement txtUsername = driver.findElement(By.id("email")); //Create object instance

Actions builder = new Actions(driver);    //Instantiate a new Actions object

Action seriesOfActions = builder

.moveToElement(txtUsername)

.click()

.keyDown(txtUsername, Keys.SHIFT)

.sendKeys(txtUsername, "hello")

.keyUp(txtUsername, Keys.SHIFT)

.doubleClick(txtUsername)

.contextClick()

.build(); //Instantiate Series of Action using the Actions object


seriesOfActions.perform();        //Use the perform() method when executing the Action object
```

==========

Uploading Files:

```
//Uploading files

WebElement uploadElement = driver.findElement(By.id("uploadfile_0"));          //Find the part of upload file field name

uploadElement.sendKeys("C:\\newhtml.html");  // enter the file path onto the file-selection input field
```

==========

Handling multiple windows (or) Child window:

```
String MainWindow = driver.getWindowHandle();          //To handle all opened windows by web driver


// To handle all new opened window.

Set < String > s1 = driver.getWindowHandles();

Iterator < String > i1 = s1.iterator();

while (i1.hasNext()) {

String ChildWindow = i1.next();
```

```
    if (!MainWindow.equalsIgnoreCase(ChildWindow)) {
    // Switching to Child window
    driver.switchTo().window(ChildWindow);
    driver.findElement(By.name("emailid")).sendKeys("gaurav.3n@gmail.com");
    driver.findElement(By.name("btnLogin")).click();
    // Closing the Child Window.
    driver.close();
    }
}

// Switching to Parent window i.e Main Window.
driver.switchTo().window(MainWindow);
```

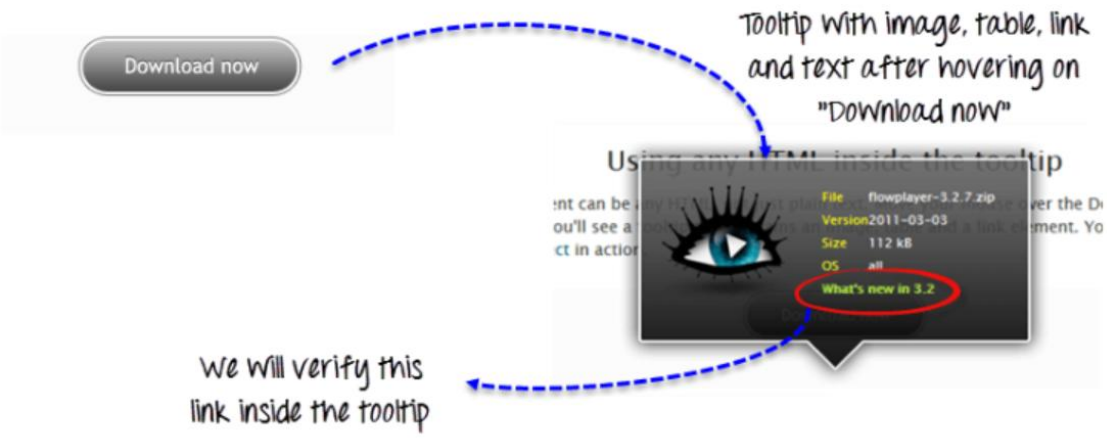
==========

Tool Tip:

Get tool tip value from “title” attribute

```
String strToolTip = driver.findElement(By.xpath(".//*[@class='soc-ico show-round']/a[4]")).getAttribute("title");
```

Get tool tip value from JQuery Plugin



WebElement download = driver.findElement(By.xpath(".//*[@id='download_now']"));

Actions builder = new Actions (driver);

builder.clickAndHold().moveToElement(download);

builder.moveToElement(download).build().perform();

WebElement toolTipElement = driver.findElement(By.xpath(".//*[@class='box']/div/a"));

String actualTooltip = toolTipElement.getText();

System.out.println("Actual Title of Tool Tip "+actualTooltip);

==========

Vertical scroll down:

End of vertical scroll down:

JavascriptExecutor js = (JavascriptExecutor) driver;

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

```
// click
JavascriptExecutor js = (JavascriptExecutor) driver;

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

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

// scroll
// js.executeScript("scrollBy(0,300)");

// draw border
js.executeScript("arguments[0].style.border='3px solid red'", element);

// reload
js.executeScript("history.go(0)");

// alert
// js.executeScript("alert('Test Alerts!!!!!!!!')");
```

```
// alert
// js.executeScript("alert('Test Alerts!!!!!!!!')");

// Title
System.out.println("Title ----" +js.executeScript("return document.title").toString());

// Get Inner Text
System.out.println("Inner Text ----> " +js.executeScript("return document.documentElement.innerText;").

// navigate to new page
js.executeScript("window.location='https://www.selenium.dev/selenium/web/inputs.html'");

// Async
```

```
// Async

 driver.manage().window().maximize();
 long start_time = System.currentTimeMillis();

// Call the execute AsynScript to wait for 5 seconds
 js.executeAsyncScript("window.setTimeout(arguments[arguments.length-1],5000);");

// get difference in time
 System.out.println("Difference---->"+(System.currentTimeMillis() - start_time));
```

General Methods:

==========

// waitForPageLoad

public static void waitForPageLoad(WebDriver driver, long timeOutInSeconds) {

ExpectedCondition < Boolean > pageReadyStateComplete =

new ExpectedCondition < Boolean > () {

public Boolean apply(WebDriver driver) {

return ((JavascriptExecutor) driver).executeScript("return document.readyState").equals("complete");

```
        }
      };

   (new WebDriverWait(driver, timeOutInSeconds)).until(pageReadyStateComplete);
}
```

==========

```
// Implicit Wait
driver.manage().timeouts().implicitlyWait(10,TimeUnit.SECONDS);
```

==========

```
// Explicit Wait
WebDriverWait wait = new WebDriverWait(driver, 50);
WebElement element =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath("//a[@class='ico-login']")));
if (element.getText().trim().equals("Log in")){
System.out.println("Demo Web Shop page is displayed");
}else{
System.out.println("Demo Web Shop page is not displayed");
}
```

Difference between waits

Feature	Implicit Wait	Explicit Wait	Fluent Wait	Thread.sleep()
Defined globally	Yes	No	No	No
Waits for	Every element	Specific element or condition	Specific element or condition	No specific element or condition
Applied by	WebDriver	WebDriverWait	FluentWait	Thread.sleep()
Timeout	Fixed timeout	Configurable timeout	Configurable timeout	Fixed sleep duration
Condition	Presence of element	Custom conditions like element visibility, element clickability, etc.	Custom conditions like element visibility, element clickability, etc.	No specific condition
Polling	yes (500 milliseconds) Default time : 0	Yes	Yes (Default : 500 milliseconds) Custom +Default	No
Expected	Implicitly applied to all elements	Explicitly applied to specific elements	Explicitly applied to specific elements	No specific expectation
Flexibility	Limited control	Greater control over wait conditions and timeout	Greater control over wait conditions and timeout	No specific control
Usage	Suitable for simple scenarios	Suitable for complex scenarios with specific wait conditions	Suitable for complex scenarios with specific wait conditions	Suitable for pausing execution without any specific condition

==========

Steps to Read and Write excel files in Selenium:

Step 1- Download apache poi jar files

Step 2- Add all jar files into Project

Read data from excel using Selenium:

```
// Specify the path of file
File src=new File("filepath/excelsheetname.xlsx");
// load file
FileInputStream fis=new FileInputStream(src);
// Load workbook
XSSFWorkbook wb=new XSSFWorkbook(fis);
// Load sheet- Here we are loading first sheetonly
XSSFSheet sh1= wb.getSheetAt(0);

String strCellValue = sh1.getRow(0).getCell(0).getStringCellValue();
```

Write data from excel using Selenium:

```
// Specify the file path which you want to create or write
File src=new File("./testdata/test.xlsx");
// Load the file
FileInputStream fis=new FileInputStream(src);
// load the workbook
XSSFWorkbook wb=new XSSFWorkbook(fis);
// get the sheet which you want to modify or create
XSSFSheet sh1= wb.getSheetAt(0);
```

```
// here createCell will create column
// and setCellvalue will set the value
sh1.getRow(0).createCell(2).setCellValue("2.41.0");
sh1.getRow(1).createCell(2).setCellValue("2.5");

// here we need to specify where you want to save file
FileOutputStream fout=new FileOutputStream(new File("location of file/filename.xlsx"));

// finally write content
 wb.write(fout);

// close the file
wb.close();
```

==========

Read data from Database:

```
//Load the driver
Class.forName("om.mysql.jdbc.Driver");

// Create a connection
String strDB_URL = "jdbc:mysql://localhost/Testdata";
String strDB_USER = "your_user";
String strDB_PASSWORD = "your_password";

Connection con=DriverManager.getConnection(strDB_URL, strDB_USER, strDB_PASSWORD);

// Create a statement
Statement smt=con.createStatement();

// Execute the query and store th data in Result set
ResultSet rs=  smt.executeQuery("Select * from TABLE");

// Iterate the resultset and get the data
while(rs.next()){
```

```
String uname=    rs.getString("username");
String pass=     rs.getString("password");
String email=    rs.getString("email");

System.out.println("Uname is "+uname+" password is "+pass+" email id is "+email);

}
```

==========

Finding Broken Links with Selenium WebDriver:

```
public static void verifyLinks(String linkUrl) {
        try {
        URL url = new URL(linkUrl);
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setConnectTimeout(5000);
        connection.connect();
        if (connection.getResponseCode() != 200) {
                System.out.println(linkUrl + " - " + "is a broken link");
                }
        } catch (Exception e) {
        }
}
```

Code snippet to find Broken Link

*BrokenLinks.java

```
import org.openqa.selenium.By;

public class BrokenLinks {
    public static void main(String[] args) {;
        WebDriver driver = new ChromeDriver();
        driver.manage().window().maximize();
        driver.get("https://demoqa.com/broken");
        List<WebElement> links = driver.findElements(By.tagName("a"));
        System.out.println("No of links are "+ links.size());
        for(int i=0;i<links.size();i++)
        {
            WebElement E1= links.get(i);
            String url= E1.getAttribute("href");
            verifyLinks(url);
        }
        driver.quit();
    }
```

```
public static void verifyLinks(String linkUrl) {
    try {
        URL url = new URL(linkUrl);
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setConnectTimeout(10000);
        connection.connect();
        if (connection.getResponseCode()>=400 || (connection.getResponseCode()>=301 &&connection.getHeaderF
            System.out.println(linkUrl + " - " + "is a broken link");
        }
    } catch (Exception e) {
    }
}
```

```
29
30⊖itring linkUrl) {
31
32 :l);
33 :tion = (HttpURLConnection) url.openConnection();
34 ieout(10000);
35
36 iseCode()>=400 || (connection.getResponseCode()>=301 &&connection.getHeaderField("Location").contains("404")))
37 .inkUrl + " - " + "is a broken link");
38
39
40
41
42
43
44
```

Reference Site:

https://www.guru99.com/selenium-tutorial.html

http://learn-automation.com/maven-no-compiler-is-provided-in-this-environment-selenium/

TestNG

Day-42

TestNG -Test New Generation

java based unit testing tool.

Advantages:
---
1) Test cases & test suites
2) Grouping of test cases.
3) Prioritize
4) Parameterization
5) parallel testing
6) Reports

TestNG configuration
---
1) Install testng in eclipse
2) add testng library to build path / add testng dependency in pom.xml

@Test - annotation

1) TestNG execute test methods based on alphabetical order.
2) @Test(priority=num) controls the order of execution.
3) Once you provide priorty to the test methods, then order of methods is not considered.
4) priorities can be random numbers(no need to have consecutive numbers)
5) If you dont provide priority then default value is Zero (0).
6) If the priorities are same then again execute methods in alphabetical order.
7) Negitive values are allowed in priority.
8) TestNG execute test methods only if they are having @Test annotation.

execute test cases using testng xml file

test xml file
- - - -
1) generate automatically
2) manually

Test suite--->test cases ---- >test steps
xml file---->classes ---> Test methods

2 things achived through xml
- - - - -
1) executed group of test cases as a 1 suite
2) we can generate testng reports (default)

```
package day42;

import org.testng.annotations.Test;

/*
 1) Open app
 2) Login
 3) Logout

 */

//       -4 -5 -3 -2 -1  0 1 2 3 4 5

public class FirstTestCase
{
   @Test(priority=1)
   void logout()
   {
      System.out.println("Logout from application .....");
   }


   @Test(priority=0)
   void login()
```

```
    {
        System.out.println("Login to application .....");
    }

    @Test(priority=-1)
    void openapp()
    {
        System.out.println("opening application ...");
    }

}
```

Q1. What is TestNG and why do we use it in Selenium?

Ans:
TestNG (Test Next Generation) is a **testing framework** inspired by JUnit and NUnit. It is widely used with Selenium because:

- It allows grouping and prioritizing test cases.
- Supports parallel execution.
- Provides parameterization.
- Generates detailed HTML/XML reports.

Q2. How does TestNG differ from JUnit?

Ans:

Feature	JUnit	TestNG
Annotations	Limited	Rich set of annotations (@Test, @BeforeSuite, @AfterTest, etc.)
Parameterization	Not flexible	Supported with @Parameters & DataProvider
Parallel Execution	No	Yes
Test Reports	Basic	Detailed HTML reports

Q3. How does TestNG decide the order of execution for test methods?

Ans:

- By default → Alphabetical order of method names.
- With @Test(priority=n) → Executes in order of priority.
- If multiple methods have the same priority → Executes in alphabetical order.
- Negative priorities are allowed.
- If no priority is set → default is **0**.

Q4. What are the main annotations in TestNG?

Ans:

- @BeforeSuite → Runs before all test cases in the suite.
- @AfterSuite → Runs after all test cases in the suite.
- @BeforeTest / @AfterTest → Runs before/after <test> tag in XML.
- @BeforeClass / @AfterClass → Runs before/after first test method of the class.
- @BeforeMethod / @AfterMethod → Runs before/after every test method.
- @Test → Marks a test case.

Q5. How do you execute tests using TestNG XML file?

Ans:

- Create testng.xml file.
- Define suite, test, classes, and methods.
- Run the XML file.

Example:

```
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="MySuite">
  <test name="LoginTests">
    <classes>
      <class name="com.project.tests.LoginTest"/>
    </classes>
  </test>
  <test name="HomeTests">
    <classes>
```

```
        <class name="com.project.tests.HomePageTest"/>
      </classes>
   </test>
</suite>
```

```
Day-43
------
Annotations
---

@Test

@BeforeMethod
@AfterMethod

@BeforeClass
@AfterClass

@BeforeTest
@AfterTest

@BeforeSuite
@AfterSuite

TC1
---
1) Login -- @BeforeMethod
2) Search --->  @Test
3) Logout  ---@AfterMethod
4) Login
5) Adv search  ---@Test
6) Logout


TC2
---
1) Login --- @BeforeClass
2) Search --- @Test
3) Adv search   --- @Test
4) Logout  -- -AfterClass

xml
----

<suite name="mysuite>
        <test name="mytest1>
                <classes>
                        <class name="xyz"/>
                <classes>
        </test>
```

```
        <test name="mytest2>
                <classes>
                        <class name="abc"/>
                <classes>
        </test>

<suite>
```

Assertion - validation point

2 kinds of assertions
---
1) Hard assertions
2) Soft assertions

Hard assertions
---
we can access from "Assert" class
methods are static
* if hard assertion failed then rest of the statements will not be executed.

software assertion
---
we can access though "SoftAssert" object

SoftAssert sa=new SoftAssert();
sa.assertTrue()

if soft assertion got failed then rest of the statemetns still execute.

```
package day43;

import org.testng.Assert;
import org.testng.annotations.Test;
import org.testng.asserts.SoftAssert;

public class HardVsSoftAssertions {

   SoftAssert sa;

   //@Test
   void test_hardassertions()
   {
      System.out.println("testign .....");
      System.out.println("testign .....");
```

```
public class CustomAssertions {

    public static void assertTextEquals(String actual, String
expected, String message) {
        if (!actual.equals(expected)) {
            throw new AssertionError(message +
                "\nExpected: " + expected +
                "\nActual: " + actual);
        }
    }
}
```

```
    Assert.assertEquals(1, 2); //hard assertion     Assert.fail() Assert.assertTrue

    System.out.println("testing .... ");
    System.out.println("testing .... ");
  }


  @Test
  void test_softassertion()
  {
    System.out.println("testign ..... ");
    System.out.println("testign ..... ");

    SoftAssert sa=new SoftAssert();

    sa.assertEquals(1, 2); //soft assertion

    System.out.println("testing .... ");
    System.out.println("testing .... ");

    sa.assertAll(); // mandatory
  }



}



SoftAssert sa = new SoftAssert();

sa.assertEquals(actual, expected, "Equality check failed");
sa.assertNotEquals(actual, expected, "Inequality check failed");
sa.assertTrue(condition, "Condition not true");
sa.assertFalse(condition, "Condition not false");
sa.assertNull(object, "Object should be null");
sa.assertNotNull(object, "Object should not be null");
sa.assertSame(actualObject, expectedObject, "Objects not same");
sa.assertNotSame(actualObject, expectedObject, "Objects are same");

sa.assertAll(); // Important - reports all collected failures
```

Q1. What are annotations in TestNG?

Ans:
Annotations in TestNG are **special instructions** that tell the framework when to run a particular method in relation to the test execution cycle (before test, after test, before class, etc.).

Q2. Can you explain TestNG execution flow with annotations?

Ans:
Order of execution:

1. @BeforeSuite → runs once before all tests in the suite.
2. @BeforeTest → runs before <test> block in XML.
3. @BeforeClass → runs before the first method in the class.
4. @BeforeMethod → runs before **each test method**.
5. @Test → the test case itself.
6. @AfterMethod → runs after **each test method**.
7. @AfterClass → runs after all methods in the class.
8. @AfterTest → runs after <test> block in XML.
9. @AfterSuite → runs once after all tests in the suite.

Q3. Difference between @BeforeMethod and @BeforeClass?

Ans:

- @BeforeMethod: Runs **before each test method** inside the class. (login for every test).
- @BeforeClass: Runs **once before the first test method** in the class. (login only once).

Q4. What are Hard Assertions in TestNG?

Ans:

- Provided by the **Assert class**.
- Methods are **static** (e.g., Assert.assertTrue(condition)).
- If a hard assertion fails → execution of that **test method stops immediately**.

Example:

```
@Test
public void hardAssertExample() {
   System.out.println("Step 1");
   Assert.assertTrue(false); // fails here
   System.out.println("Step 2"); // will NOT execute
}
```

Q5. What are Soft Assertions in TestNG?

Ans:

- Provided by the **SoftAssert class**.
- You must create an object of SoftAssert.
- Even if a soft assertion fails, the next statements will still execute.
- At the end, call assertAll() to collect failures.

Example:

```
@Test
public void softAssertExample() {
   SoftAssert sa = new SoftAssert();
   System.out.println("Step 1");
   sa.assertTrue(false); // fails but continues
   System.out.println("Step 2"); // will execute
   sa.assertAll(); // reports failures here
}
```

Q6. When would you prefer SoftAssert over HardAssert?

Ans:

- Use **HardAssert** when one failure should stop the test immediately (e.g., login must pass).
- Use **SoftAssert** when you want to validate multiple things in the same test case without stopping (e.g., verifying all labels on a page).

Q7. Example XML suite for running multiple classes in TestNG?

Ans:

```
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="MySuite">
   <test name="MyTest1">
      <classes>
         <class name="com.project.tests.LoginTest"/>
      </classes>
```

```
  </test>
  <test name="MyTest2">
    <classes>
      <class name="com.project.tests.SearchTest"/>
    </classes>
  </test>
</suite>
```

Q8. Can you mix Hard and Soft assertions in the same test?

Ans:
Yes, but it's not recommended. Mixing them can cause confusion in reporting and execution flow.
Best practice is to **stick to one strategy per test**.

```
Day-44
------
dependency methods


openapp
login
search
advsearch
logout


dependsOnMethods= {"method name1","method name2",.... }


grouping
---------

class1 - m1 m2 m3...
class2 - m4, m5 ,m6...
class3 -- m7,m8,m9....
---

sanity
regression
functional


loginByfacebook - sanity
loginby emial  -sanity
login by twitter  -sanity
```

signupbyfacebook -regression
signupbytwitter -regression
signupbyemail - regression

paymentinrupees - sanity, regression (functional)
paymentindollars - sanity, regression (functional)

1) all sanity tests
2) all regression tests
3) all sanity but not regression
4) all regression but not sanity
5) all methods which are belongs to both sanity & regression

@Test
---
priority
dependsOnMethods
groups
etc....

Dependency Methods in TestNG

Sometimes, execution of one test depends on the success of another test.
This is done using **dependsOnMethods**.

Example

```
@Test
public void openApp() {
   System.out.println("App opened");
}

@Test(dependsOnMethods = {"openApp"})
public void login() {
   System.out.println("Login successful");
}

@Test(dependsOnMethods = {"login"})
public void search() {
   System.out.println("Search done");
}

@Test(dependsOnMethods = {"search"})
public void advSearch() {
```

```
    System.out.println("Advanced search done");
}

@Test(dependsOnMethods = {"advSearch"})
public void logout() {
    System.out.println("Logout successful");
}
```

– Execution order will be:

openApp → login → search → advSearch → logout

If one fails, all dependent ones are **skipped**.

🔹 Grouping in TestNG

You can group test cases logically (like **Sanity, Regression, Functional**).

Example

```
public class LoginTests {
    @Test(groups = {"sanity"})
    public void loginByFacebook() { System.out.println("Login by Facebook"); }

    @Test(groups = {"sanity"})
    public void loginByEmail() { System.out.println("Login by Email"); }

    @Test(groups = {"sanity"})
    public void loginByTwitter() { System.out.println("Login by Twitter"); }
}

public class SignupTests {
    @Test(groups = {"regression"})
    public void signupByFacebook() { System.out.println("Signup by Facebook"); }

    @Test(groups = {"regression"})
    public void signupByTwitter() { System.out.println("Signup by Twitter"); }

    @Test(groups = {"regression"})
    public void signupByEmail() { System.out.println("Signup by Email"); }
}

public class PaymentTests {
    @Test(groups = {"sanity","regression","functional"})
    public void paymentInRupees() { System.out.println("Payment in Rupees"); }

    @Test(groups = {"sanity","regression","functional"})
    public void paymentInDollars() { System.out.println("Payment in Dollars"); }
}
```

🔹 TestNG XML Examples for Group Execution

1 Run all sanity tests

```
<test name="SanityTests">
    <groups>
        <run>
            <include name="sanity"/>
        </run>
```

```
  </groups>
  <classes>
    <class name="com.project.tests.LoginTests"/>
    <class name="com.project.tests.PaymentTests"/>
  </classes>
</test>
Run all regression tests
<include name="regression"/>
Run sanity but not regression
<run>
  <include name="sanity"/>
  <exclude name="regression"/>
</run>
Run regression but not sanity
<run>
  <include name="regression"/>
  <exclude name="sanity"/>
</run>
Run only those belonging to both sanity & regression
– Here, only paymentInRupees & paymentInDollars will run.
<run>
  <include name="sanity"/>
  <include name="regression"/>
</run>
```

Q1. What is dependsOnMethods in TestNG?

– It defines dependency between test methods. If the dependent method fails, the next one will be skipped.

Q2. What happens if a method mentioned in dependsOnMethods is skipped or fails?

– All dependent methods will be **skipped**.

Q3. What is the use of grouping in TestNG?

– Grouping allows you to categorize test cases (like **Sanity, Regression, Functional**) and run them selectively via XML.

Q4. Can a test method belong to multiple groups?

– Yes. Example:

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

Q5. Difference between priority and dependsOnMethods?

- **priority** → controls execution order (numeric).

- **dependsOnMethods** → ensures execution happens **only if the dependent test passes**.

Day-45

Parameterization
---
1) @DataProvider -- data driven testing

2) using xml file -- parallel testing

parallel testing using xml file
---

step1) created test case
step 2) create xml file then run test case through xml
step3) passed browser name, url as parameters from xml file to setup() method
step4) execute test case on chrome,firefox & Edge (serial execution)
step5) execute test case on chrome,firefox & edge (parallel execution)

🔷 Parameterization in TestNG

There are **two main ways**:

Using @DataProvider (Data Driven Testing)

- Used to provide **multiple sets of data** to a test method.
- Useful when you want to test with different inputs (like multiple usernames/passwords).

```
import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;


public class DataProviderExample {


   @DataProvider(name="loginData")

   public Object[][] getData() {

      return new Object[][] {

         {"admin", "admin123"},

         {"user1", "pass1"},
```

```
            {"user2", "pass2"}
        };
    }

    @Test(dataProvider="loginData")
    public void loginTest(String username, String password) {
        System.out.println("Username: " + username + ", Password: " + password);
        // Here you would add Selenium login logic
    }
}
```

- This will run loginTest **3 times** with different data.

Using XML File (Parameters)

- Pass values like **browser name, URL, credentials** from testng.xml.

Example: TestNG.xml

```
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >
<suite name="ParameterSuite">
    <test name="ParameterTest">
        <parameter name="browser" value="chrome"/>
        <parameter name="url" value="https://example.com"/>
        <classes>
            <class name="com.project.tests.ParameterTest"/>
        </classes>
    </test>
</suite>
```

Example: Java Test

```
import org.testng.annotations.Parameters;
import org.testng.annotations.Test;

public class ParameterTest {
```

```
    @Test
    @Parameters({"browser", "url"})
    public void setup(String browser, String url) {
        System.out.println("Browser: " + browser);
        System.out.println("URL: " + url);
        // Add browser launching logic here
    }
}
```

- Values will be picked from XML at runtime.

🔷 Parallel Testing in TestNG

Parallel testing = running tests **at the same time** on multiple browsers/machines.
This helps **reduce execution time**.

❯ Steps for Parallel Testing

Step 1: Create XML with Multiple <test> tags

```
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >
<suite name="ParallelSuite" parallel="tests" thread-count="3">

    <test name="ChromeTest">
        <parameter name="browser" value="chrome"/>
        <classes>
            <class name="com.project.tests.CrossBrowserTest"/>
        </classes>
    </test>

    <test name="FirefoxTest">
        <parameter name="browser" value="firefox"/>
        <classes>
            <class name="com.project.tests.CrossBrowserTest"/>
        </classes>
```

```
    </test>

    <test name="EdgeTest">
        <parameter name="browser" value="edge"/>
        <classes>
            <class name="com.project.tests.CrossBrowserTest"/>
        </classes>
    </test>

</suite>
```

Step 2: Java Class

```
import org.testng.annotations.Parameters;
import org.testng.annotations.Test;

public class CrossBrowserTest {

    @Test
    @Parameters("browser")
    public void setup(String browser) {
        if(browser.equalsIgnoreCase("chrome")) {
            System.out.println("Launching Chrome...");
        } else if(browser.equalsIgnoreCase("firefox")) {
            System.out.println("Launching Firefox...");
        } else if(browser.equalsIgnoreCase("edge")) {
            System.out.println("Launching Edge...");
        }
    }
}
```

🟩 **Execution Modes:**

1. **Serial Execution** (default) → Chrome → Firefox → Edge (one after another).

2. **Parallel Execution** → All browsers open **at the same time** using parallel="tests" and thread-count.

🔷 **Interview Q&A**

Q1. What is the difference between DataProvider and @Parameters in TestNG?

- @DataProvider → Used for **data-driven testing**, provides multiple sets of test data.
- @Parameters → Used to **pass values from XML**, usually for configuration (like browser, URL).

Q2. How do you run tests in parallel in TestNG?

– Use parallel="tests" or parallel="classes" in XML and define thread-count.

Q3. Can we use both @DataProvider and @Parameters in the same project?

— Yes, but usually:

- Use @Parameters for environment setup (browser, URL).
- Use @DataProvider for test data (user credentials, inputs).

Q4. What happens if thread-count is less than number of tests?

– Only that many threads will run simultaneously, others will wait in the queue.

Q5. How do you pass multiple parameters using XML?

– Define multiple <parameter> tags inside <test> and annotate method with @Parameters({"param1","param2"}).

```
Day-46
------
TestNG Listners
---

1) create test case
2) create listener class
3) create xml file and include both test case & listener class


2 ways to implement listener class
--------
Method1
----
class myListner implements ITestListener
{
```

```
}

Method2
---------
class myListener extends TestListenerAdapter
{

}


Extent report
-----
ExtentSparkReporter
ExtentReports
ExtentTest


maven dependency
-------------

<!-- https://mvnrepository.com/artifact/com.aventstack/extentreports -->
        <dependency>
            <groupId>com.aventstack</groupId>
            <artifactId>extentreports</artifactId>
            <version>5.1.1</version>
        </dependency>




package day46;

import org.testng.ITestContext;
import org.testng.ITestListener;
import org.testng.ITestResult;

public class MyListener implements ITestListener
{

    public void onStart(ITestContext context) {

        System.out.println("Test execution is started..... ");

        }


    public void onTestStart(ITestResult result) {
        System.out.println("test started.. ");
        }
```

```
    public void onTestSuccess(ITestResult result) {
       System.out.println("test passed...");
       }


    public void onTestFailure(ITestResult result) {
       System.out.println("test failed...");
       }

    public void onTestSkipped(ITestResult result) {
       System.out.println("test skipped...");
       }

    public void onFinish(ITestContext context) {
       System.out.println("test execution is completed...");
       }
}



package day46;

import java.time.Duration;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Listeners;
import org.testng.annotations.Test;

//@Listeners(day46.MyListener.class)
public class OrangeHRM {

   WebDriver driver;

   @BeforeClass
   void setup() throws InterruptedException
   {
      driver=new ChromeDriver();

      driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
      driver.get("https://opensource-demo.orangehrmlive.com/web/index.php/auth/login");
      driver.manage().window().maximize();
      Thread.sleep(3000);

   }
```

```
    @Test(priority=1)
    void testLogo()
    {
        boolean status=driver.findElement(By.xpath("//img[@alt='company-
branding']")).isDisplayed();
        Assert.assertEquals(status,  true);
    }



    @Test(priority=2)
    void testAppUrl()
    {
        Assert.assertEquals(driver.getCurrentUrl(),"https://opensource-demo.orangehrmlive.com/");
    }


    @Test(priority=3, dependsOnMethods= {"testAppUrl"})
    void testHomePageTitle()
    {
        Assert.assertEquals(driver.getTitle(),"OrangeHRM");

    }

    @AfterClass
    void tearDown()
    {
        driver.quit();
    }
}


<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">

<suite name="Suite">

  <listeners>
    <listener class-name="day46.MyListener" />
  </listeners>

  <test thread-count="5" name="Test">
    <classes>
      <class name="day46.OrangeHRM"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

```
package day46;

import org.testng.ITestListener;

import  org.testng.ITestContext;
import org.testng.ITestListener;
import org.testng.ITestResult;

import com.aventstack.extentreports.ExtentReports;
import com.aventstack.extentreports.ExtentTest;
import com.aventstack.extentreports.Status;
import com.aventstack.extentreports.reporter.ExtentSparkReporter;
import com.aventstack.extentreports.reporter.configuration.Theme;

public class ExtentReportManager implements ITestListener
{
   public ExtentSparkReporter sparkReporter;  // UI of the report
   public ExtentReports extent;  //populate common info on the report
   public ExtentTest test; // creating test case entries in the report and update status of the test
methods

   public void onStart(ITestContext context) {

      sparkReporter=new ExtentSparkReporter(System.getProperty("user.dir")+
"/reports/myReport.html");//specify location of the report

      sparkReporter.config().setDocumentTitle("Automation Report"); // TiTle of report
      sparkReporter.config().setReportName("Functional Testing"); // name of the report
      sparkReporter.config().setTheme(Theme.STANDARD);

      extent=new ExtentReports();
      extent.attachReporter(sparkReporter);

      extent.setSystemInfo("Computer Name","localhost");
      extent.setSystemInfo("Environment","QA");
      extent.setSystemInfo("Tester Name","Pavan");
      extent.setSystemInfo("os","Windows10");
      extent.setSystemInfo("Browser  name","Chrome");

   }

   public void onTestSuccess(ITestResult result) {

      test = extent.createTest(result.getName()); // create a new enty in the report
      test.log(Status.PASS, "Test case PASSED is:" + result.getName()); // update status p/f/s

   }

   public void onTestFailure(ITestResult result) {
```

```
        test = extent.createTest(result.getName());
        test.log(Status.FAIL, "Test case FAILED is:" + result.getName());
        test.log(Status.FAIL, "Test Case FAILED cause is: " + result.getThrowable());

    }

    public void onTestSkipped(ITestResult result) {

        test = extent.createTest(result.getName());
        test.log(Status.SKIP, "Test case SKIPPED is:" + result.getName());

    }


    public void onFinish(ITestContext context) {

        extent.flush();
    }
}
```

🟩 TestNG Listeners – Q&A

Q1. What are TestNG listeners?

– Listeners are interfaces in TestNG that allow you to perform actions based on events (like test start, pass, fail, skip).

Q2. Which interfaces are used for listeners in TestNG?

— Common ones:

- ITestListener – listens to test case events (start, success, failure, skip).
- ISuiteListener – listens to suite-level events.
- IAnnotationTransformer – modifies test annotations at runtime.
- IInvokedMethodListener – listens to before/after method invocations.

Q3. What is the difference between ITestListener and TestListenerAdapter?

– ITestListener → You need to implement **all methods**.

– TestListenerAdapter → You can **override only required methods**, no need to implement all.

Q4. How do you register a listener?

— Two ways:

1. @Listeners(MyListener.class) annotation at class level.
2. In testng.xml inside <listeners> tag.

Q5. Which listener method is triggered when a test case fails?

– onTestFailure(ITestResult result)

Q6. Can listeners be used for logging or screenshots?

– Yes ✅ When a test fails, you can capture a screenshot in onTestFailure() and attach it to reports.

✅ **Extent Reports – Q&A**

Q7. What is Extent Report?

– A reporting library used in automation frameworks to generate **interactive HTML reports** with logs, screenshots, and test execution details.

Q8. What are the main classes in Extent Reports?

—

- ExtentReports – main class to manage reports.
- ExtentSparkReporter – generates HTML reports.
- ExtentTest – represents individual test cases.

Q9. How do you generate an Extent Report?

— Steps:

1. Create ExtentSparkReporter object (define file path).
2. Create ExtentReports object and attach reporter.
3. Create ExtentTest for each test.
4. Use test.pass(), test.fail() etc.
5. Call extent.flush() at end to write results.

Q10. How do you attach screenshots to Extent Reports?

— Example:

test.fail("Test Failed", MediaEntityBuilder

.createScreenCaptureFromPath("screenshot.png").build());

Q11. What is the latest version of Extent Reports?

- Currently 5.x (e.g., 5.1.1).

Q12. How do you integrate Listeners with Extent Reports?

- In onTestFailure() of Listener, log the failure in Extent Report and attach a screenshot.

```
Day-47
----
Page Object model
-----------

Test case
---
locators
test methods ----code---validations+Actions

2 appraoches to create page object classes
-----------------
1) without using PageFactory
2) using PageFactory
-----------


WebElement usertxt=driver.findElement(By.xpath("//input[@placeholder='Username']");


By loc=By.xpath("//input[@placeholder='Username'];
driver.findElement(loc).sendKeys("xyz")
```

🟩 Page Object Model (POM) in Selenium

— Definition:

Page Object Model is a design pattern in Selenium that helps create **object repository** for web elements.
Each web page is represented as a **class**, and the elements of that page are defined as **variables** with their actions as **methods**.

🔷 Benefits of POM

1. Code reusability (same methods can be reused across tests).
2. Easy to maintain (if a locator changes, update only in one place).
3. Improves readability of tests.
4. Reduces code duplication.

🟩 **Two Approaches**

1) Without PageFactory (Normal way)

We define **By locators** and then call them inside methods.

```
public class LoginPage {
   WebDriver driver;

   // Locators
   By username = By.xpath("//input[@placeholder='Username']");
   By password = By.xpath("//input[@placeholder='Password']");
   By loginBtn = By.xpath("//button[@id='loginBtn']");

   // Constructor
   public LoginPage(WebDriver driver) {
      this.driver = driver;
   }

   // Actions
   public void setUsername(String uname) {
      driver.findElement(username).sendKeys(uname);
   }

   public void setPassword(String pwd) {
      driver.findElement(password).sendKeys(pwd);
   }

   public void clickLogin() {
```

```
        driver.findElement(loginBtn).click();
    }
}
```

---

2) Using PageFactory

Here we use @FindBy annotations with **PageFactory.initElements()**.

```
import org.openqa.selenium.support.PageFactory;
import org.openqa.selenium.support.FindBy;

public class LoginPage {
    WebDriver driver;

    // Locators with @FindBy
    @FindBy(xpath="//input[@placeholder='Username']")
    WebElement username;

    @FindBy(xpath="//input[@placeholder='Password']")
    WebElement password;

    @FindBy(id="loginBtn")
    WebElement loginBtn;

    // Constructor
    public LoginPage(WebDriver driver) {
        this.driver = driver;
        PageFactory.initElements(driver, this); // Initialize WebElements
    }

    // Actions
    public void setUsername(String uname) {
        username.sendKeys(uname);
```

```
    }

    public void setPassword(String pwd) {
        password.sendKeys(pwd);
    }

    public void clickLogin() {
        loginBtn.click();
    }
}
```

🟩 Test Class Example

```
public class LoginTest {
    WebDriver driver;

    @BeforeMethod
    public void setup() {
        driver = new ChromeDriver();
        driver.get("https://example.com/login");
    }

    @Test
    public void testLogin() {
        LoginPage lp = new LoginPage(driver);
        lp.setUsername("admin");
        lp.setPassword("admin123");
        lp.clickLogin();
    }

    @AfterMethod
    public void tearDown() {
```

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

🟩 Interview Q&A

Q1. What is Page Object Model (POM)?

– It's a design pattern that creates an object repository for storing web elements, making test code clean, reusable, and maintainable.

Q2. Difference between POM with and without PageFactory?

- **Without PageFactory** → Use By locators and driver.findElement().
- **With PageFactory** → Use @FindBy with PageFactory.initElements() for cleaner code.

Q3. Why use POM in Selenium?

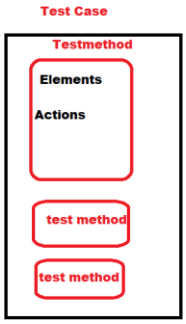
– To reduce code duplication, improve maintainability, and make test automation frameworks more structured.

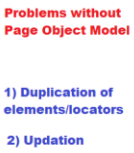
Q4. Can we use POM without PageFactory?

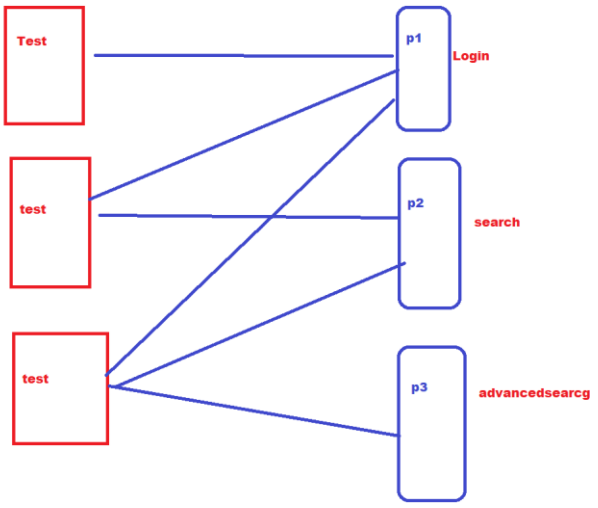
– Yes 🟩 Many frameworks do so. PageFactory is just a helper utility.

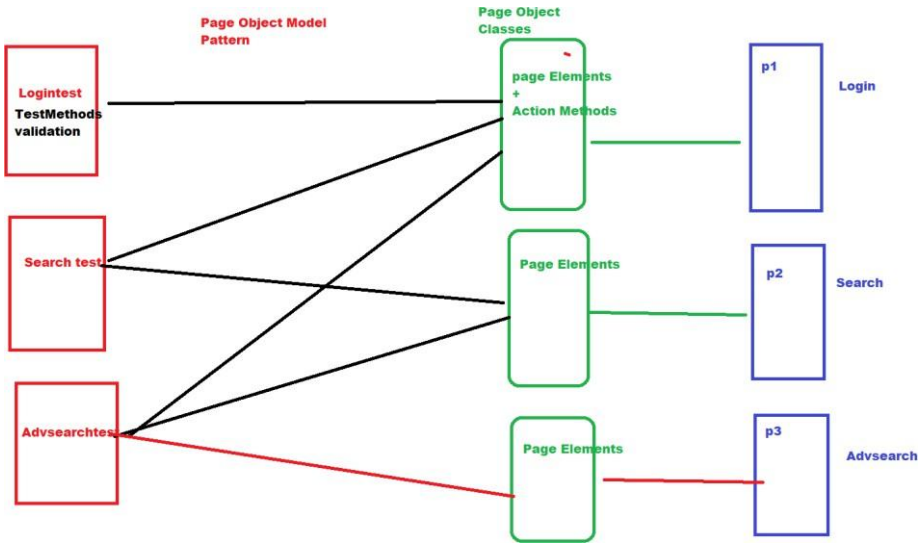
Q5. What happens if a locator changes?

– You just update it in the Page class, and all tests using that class remain unaffecte









1. How to right click in Selenium?

```
Actions actions = new Actions(driver);
WebElement objElement = driver.findElement((By.id("test")));
actions.contextClick(objElement).perform();
```

2. How to double click in Selenium?

```
Actions actions = new Actions(driver);
WebElement objElement = driver.findElement(By.id("test"));
actions.doubleClick(objElement).click();
```

3. How to drag and drop in Selenium?

```
WebElement objFrom = driver.findElement(By.id(""));
WebElement objTo = driver.findElement(By.id(""));

actions = new Actions(driver);
actions.dragAndDrop(objFrom, objTo).build().perform();
```

4. Different methods to refresh a web page?

```
driver.navigate().refresh();
driver.navigate().to(driver.getCurrentUrl());
driver.get(driver.getCurrentUrl());
driver.findElement(By.id("")).sendKeys(Keys.F5);
JavascriptExecutor js = (JavascriptExecutor) driver;
js.executeScript("location.reload()");
```

5. Navigate back?

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

6. Navigate forward?

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

7. What is the difference between get and navigate methods in Selenium?

Get	**Navigate**
Helps only to invoke URL	1) Helps to invoke the URL 2) Helps to navigate back, forward and refresh
Slower than navigate() method as it wait till the page to load fully or completely	Faster than get than get as it does not wait till the page to load fully or completely

8. What is the difference close and quit in Selenium?

Close	**Quit**

It closes the current open window on which driver has focus on.	It closes all Browser Windows associated with that driver. This method is used to destroy the instance of WebDriver.

9. What is the difference implicit and explicit wait in Selenium?

Implicit wait	**Explicit wait**
1. It helps to wait the entire duration of WebDriver object for certain duration of time before proceed to next step. 2. An implicit wait is to tell WebDriver to poll the DOM for a certain amount of time when trying to find an element or elements if they are not immediately available.	1. Explicit waits are intelligent wait that instruct WebDriver object to wait for a certain condition before proceed to next step

10. How many way we can handle frames in WebDriver?

- By Index
- By Name or Id
- By Web Element

By Index:

```
driver.switchTo().frame(0);          //Index 0
driver.switchTo().frame(1);          //Index 1
```

By Name or Id

```
driver.switchTo().frame("iframe1");          //Frame name
driver.switchTo().frame("id of the element");          //Frame id
```

By Web Element

```
WebElement iframeElement = driver.findElement(By.id("IF1"));          //First find the element using any of locator stratedgy
driver.switchTo().frame(iframeElement);
```

11. How to find total number of iFrames on a webpage?

```
//By finding all the web elements using iframe tag
 List<WebElement> iframeElements = driver.findElements(By.tagName("iframe"));
 System.out.println("The total number of iframes are " + iframeElements.size());

//By executing a java script
JavascriptExecutor exe = (JavascriptExecutor) driver;
Integer numberOfFrames = Integer.parseInt(exe.executeScript("return window.length").toString());
System.out.println("Number of iframes on the page are " + numberOfFrames);
```

12. Code to handle 3rd window?

```
Set<String> strWindows = driver.getWindowHandles();
Iterator<String> it = strWindows.iterator();

it.next(); //1st child window
it.next();  //2nd child window
it.next(); //3rd child window
String strThirdWindow = it.next();
driver.switchTo().window(strThirdWindow);
```

13. Code to navigate back to parent window from 3rd child window?

```
String parentWindow = driver.getWindowHandle();    //Get the parent window

Set<String> strWindows = driver.getWindowHandles();
Iterator<String> it = strWindows.iterator();

it.next(); //1st child window
it.next();  //2nd child window
it.next(); //3rd child window
String strThirdWindow = it.next();
driver.switchTo().window(strThirdWindow);
```

driver.switchTo().window(parentWindow);

14. How to handle https certifications?

```
DesiredCapabilities ds = DesiredCapabilities.chrome();
ds.setCapability(CapabilityType.ACCEPT_SSL_CERTS, true);
```

15. Different type of locators available in Selenium?

- ID
- Name
- Class Name
- Xpath
- CSS Selector
- Tag Name
- Link Text & Partial Link Text

16. Write syntax for Xpath and CSS if id and tag are given?

Xpath : //tagname[@id='value']
CSS : tagname[id=value]

17. How to use contains regular expression to Xpath?

Example if id value is dynamic : u0123, uo234, uo345
//tagname[contains(@id='uo')]

18. How to use regular expression to CSS?

tagname[id*='uo']

19. What is the class available in Selenium to handle drop down?

Select

20. What is the method to check if checkbox is selected?

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

21. How to validate if element is visible or hidden in webpages?

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

22. How to get the similar objects list in the web page?

```
driver.findElements(By.className("classname")).size();
```

23. Importance of desired capabilities mechanism?

Selenium Grid: It helps to distribute tests across multiple machines.

24. How to enter the text in caps lock?

```
driver.findElement(By.id("")).sendKeys(Keys.SHIFT,"test");
```

25. How to mouse over on the web element on page?

```
WebElement objLink = driver.findElement(By.id(""));
Actions actions = new Actions(driver);
actions.moveToElement(objLink).build().perform();
```

26. Methods to handle Java Alert?

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

27. How to get links count in the page?

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

28. How to validate if we navigate to child window successfully?

driver.getTitle();

29. What is the difference between absolute and relative xpath?

Absolute	**Relative**
It starts identify the Web Element from the beginning of the HTML DOM structure	It starts identify the Web Element anywhere from the HTML DOM structure
Its start with forward slash (/)	Its start with forward double slash (//)
Failure percentage is high	Failure percentage is low
Faster than Relative path to identify the object	Faster than Absolute path to identify the object
Difficult to handle the complex and dynamic elements	Easily to handle the complex and dynamic elements

30. Write down the sample xpath syntax to handle parent from child object?

//parent/child

31. What driver is must to run tests in Firefox browser?

Gecko driver

32. What driver is must to run tests in Chrome browser?

Chrome driver

33. How to hit enter from web driver command?

driver.findElement(By.id("")).sendKeys(Keys.ENTER);

34. What is the difference between FindElement and FindElements?

Absolute	**Relative**
On One Match - return web element object	On One Match - returns list of one WebElement only
On One + Match - returns the first appearance in DOM	On One + Match - returns list with all matching instance
On zero Match - it throws NoSuchElementException	On zero Match - return an empty list

35. Give the example for method overload in WebDriver?

frame(string), frame(int), frame(WebElement).

36. What is the use of xpath?

- It is used to find the WebElement in web page.
- It is very useful to identify the dynamic web elements.

37. What is the difference between Assert and Verify?

Assert:

- It is used to verify the result.

- If the test case fail then it will stop the execution of the test case there itself and move the control to other step.

Verify:

- It is used to verify the result.
- If the test case fail then it will not stop the execution.

38. What is the alternate way to click on login button?

Use Submit() method but it can be used only when attribute type=submit.

39. How do you click on a menu item in a drop down menu?

If that menu has been created by using select tag then we can use the methods selectByValue() or selectByIndex() or selectByVisibleText(). These are the methods of the Select class.

If the menu has not been created by using the select tag then we can simply find the xpath of that element and click on that to select.

40. What is the difference between '/' and '//' ?

//- it is used to search in the entire structure.
/- it is used to identify the immediate child.

41. Which is the package which is to be imported while working with WebDriver ?

org.openqa.selenium

42. How to check if a text is highlighted on the page ?

```
String color = driver.findElement(By.xpath("//a[text()='Shop']")).getCssValue("color");
String backcolor = driver.findElement(By.xpath("//a[text()='Shop']")).getCssValue("background-color");
System.out.println(color);
System.out.println(backcolor);
```

Here if both color and back color different then that means that element is in different color.

43. How do u get the width of the textbox ?

Identify by getCssValue("border-bottom") or sometime getCssValue("text-decoration") method if the cssValue is 'underline' for that WebElement or not. ex- This is for when moving cursor over element that is going to be underlined.

44. How to change the URL on a webpage using selenium web driver ?

```
driver.get("url1");
driver.get("url2");
```

45. What is the use of getOptions() method ?

getOptions() is used to get the selected option from the dropdown list.

46. Is WebDriver an interface or a class ?

WebDriver is an Interface.

47. FirefoxDriver is class or an interface and from where is it inherited ?

FirefoxDriver is a class. It implements all the methods of WebDriver interface.

48. Which is the super interface of webdriver ?

SearchContext

49. What is Xpath?

Xpath is the XML path of an element which we need to identify

50. What is Locator?

Locator is a command that helps to identify GUI element

https://www.edureka.co/community/49858/what-the-primary-difference-between-xpath-and-css-selector

51. Print some value from 1 to 11 and 11 to 1?

```
//For Loop
String strA = "*";
for (int i = 1;i<=11,i++)
{
 System.out.println(strA);
}

String strA = "*";
for (int i = 11;i>0,i--)
{
 System.out.println(strA);
}
//Do while
int i = 1;
do {
System.out.println(strA);
i++;
}
while(i<=11);

int i = 11;
do {
System.out.println(strA);
i--;
}while(i>0);
```

52. “Elephent” find number of “e” in the string?
53. What are the software required to connect the device?
54. Difference between Emulator and Smiulator?
55. Appium instructor?
56. Default mode to connect Appium?
57. Is it possible to perform parallel execution in Appium?
58. Do we need Appium server machine?
59. How to scroll up and down?
60. Limitations of Appium?
61. Appium inspector?
62. How to launch the native app?
63. Software required to connect real device?
64. How to check device id?
65. Most difficult area in Appium testing?
66. Date picker?