Web Automation Selenium 4.x
Notes - TheTestingAcademy (Pramod Sir)

📗 Mastering Web Automation with Selenium

IDE

1. IntelliJ
2. Eclipse

About Selenium

- Selenium Automates Web Browsers.

Run your First Selenium Script

1. Install Java and Set path to the Home in Windows / Mac
2. Install IDE (IntelliJ) and you can avoid the first step.
3. Create a New Quick Start Maven Project, Add dependencies.
4. Download Browser drivers (can skip if version > Selenium 4.6.0).
5. Add the code and run the program.

How to Set JAVA_HOME path in MAC - Click here

1. Download the JDK latest -

```
echo export "JAVA_HOME=\$(/usr/libexec/java_home)" >> ~/.bash_profile
```

2. If you're using zsh (which probably means you're running macOS Catalina or newer), then it should instead be:

```
echo export "JAVA_HOME=\$(/usr/libexec/java_home)" >> ~/.zshrc
```

3. In either case, restart your shell.

How to Set JAVA_HOME path in Widnows

Set the JAVA_HOME Variable

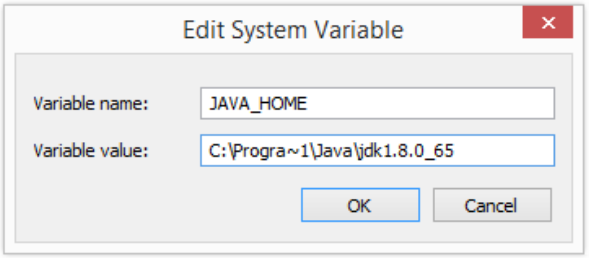
To set the JRE_HOME or JAVA_HOME variable:

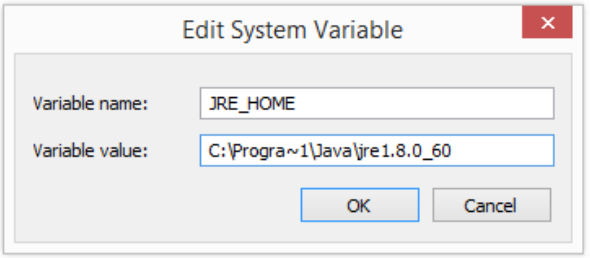
1. Locate your Java installation directory
   If you didn't change the path during installation, it'll be something like

```
C:\Program Files\Java\jdk1.8.0_65
```

You can also type `where java` at the command prompt.

2. Do one of the following:
   Windows 7 – Right click **My Computer** and select **Properties** > **Advanced**
   Windows 8 – Go to **Control Panel** > **System** > **Advanced System Settings**
   Windows 10 – Search for **Environment Variables** then select **Edit the system environment variables**
3. Click the **Environment Variables** button.
4. Under **System Variables**, click **New**.
5. In the **Variable Name** field, enter either:
   - `JAVA_HOME` if you installed the JDK (Java Development Kit)
     or
   - `JRE_HOME` if you installed the JRE (Java Runtime Environment)
6. In the **Variable Value** field, enter your JDK or JRE installation path .
   If the path contains spaces, use the shortened path name. For example,
   `C:\Progra~1\Java\jdk1.8.0_65`





   Note for Windows users on 64-bit systems
   Progra~1 = 'Program Files'
   Progra~2 = 'Program Files(x86)'
7. Click **OK** and **Apply Changes** as prompted

What is Selenium?

Selenium is an open-source suite.

Birth of WebDriver

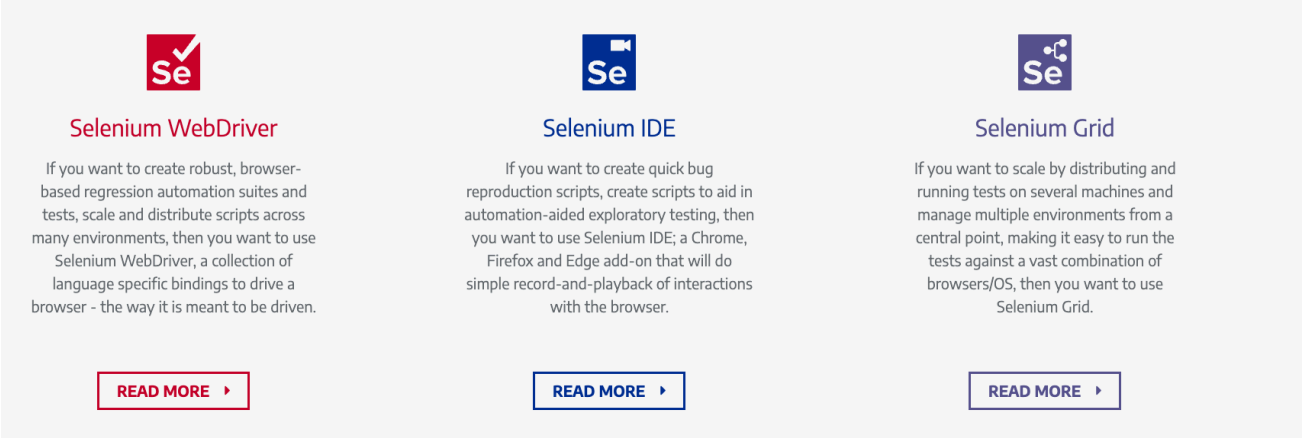

Simon Stewart

Simon Stewart created WebDriver circa 2006 when browsers and web applications were becoming more powerful and more restrictive with JavaScript programs like Selenium Core. It was the first cross-platform testing framework that could control the browser from the OS level.

which can automate browsers

Selenium automates browsers. That's it!

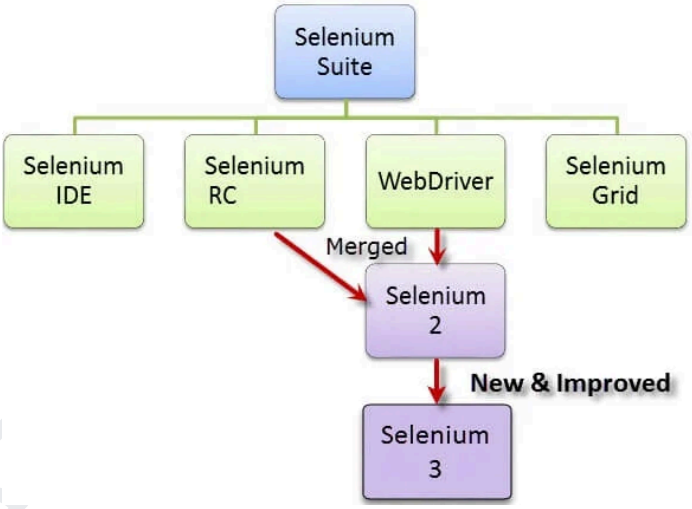
What you do with that power is entirely up to you.

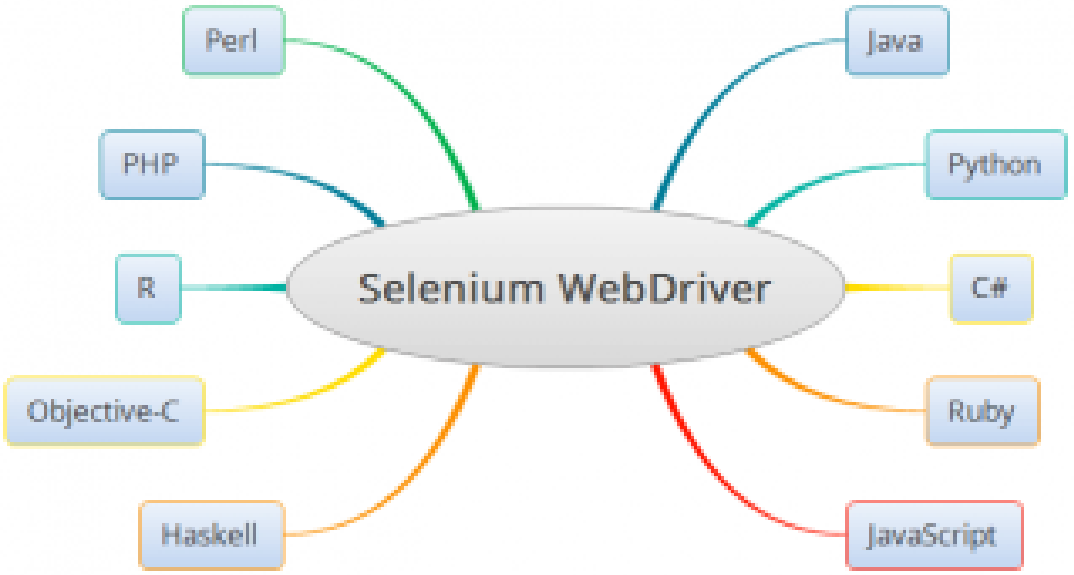


History -

Selenium RC executed tests by injecting JavaScript code into the web browser being automated. RC deprecated,

Webdriver - Find the elements,





Selenium vs Playwright vs Cypress

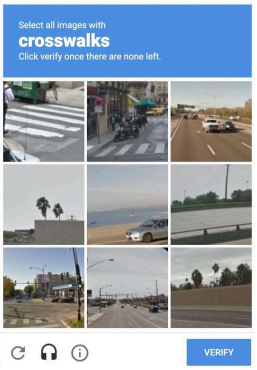
Compare Results - https://blog.checklyhq.com/cypress-vs-selenium-vs-playwright-vs-puppeteer-speed-comparison/

Requirement/Tool	Selenium	Cypress	Playwright
Established solution	Yes	Yes	Yes
Has own runner/reporting	No	Yes	Yes
Compatible with other runners (Junit, Jest, ...)	Yes	No	Yes
Cross Domain Support	Yes	No	Yes
Multi tab support	Yes	No	Yes
Performance	Good	Good	Best
Drag and Drop support	Yes	Yes	Yes
Dynamic waits	No	Yes	Yes
Static waits	Yes	Yes	No
Element selector support	css, xpath, text	css, xpath, text	css, xpath, text
Parallel test execution	Yes	Yes (pro version or Sorry Cypress project)	Yes
Video recording, screenshots	Yes (videos only with 3rd party tools)	Yes	Yes
Possibility to use as AWS Lambda	Yes	Yes	Yes
Support for Java, JavaScript and Python	Yes	No	Yes

Don’t use Selenium here

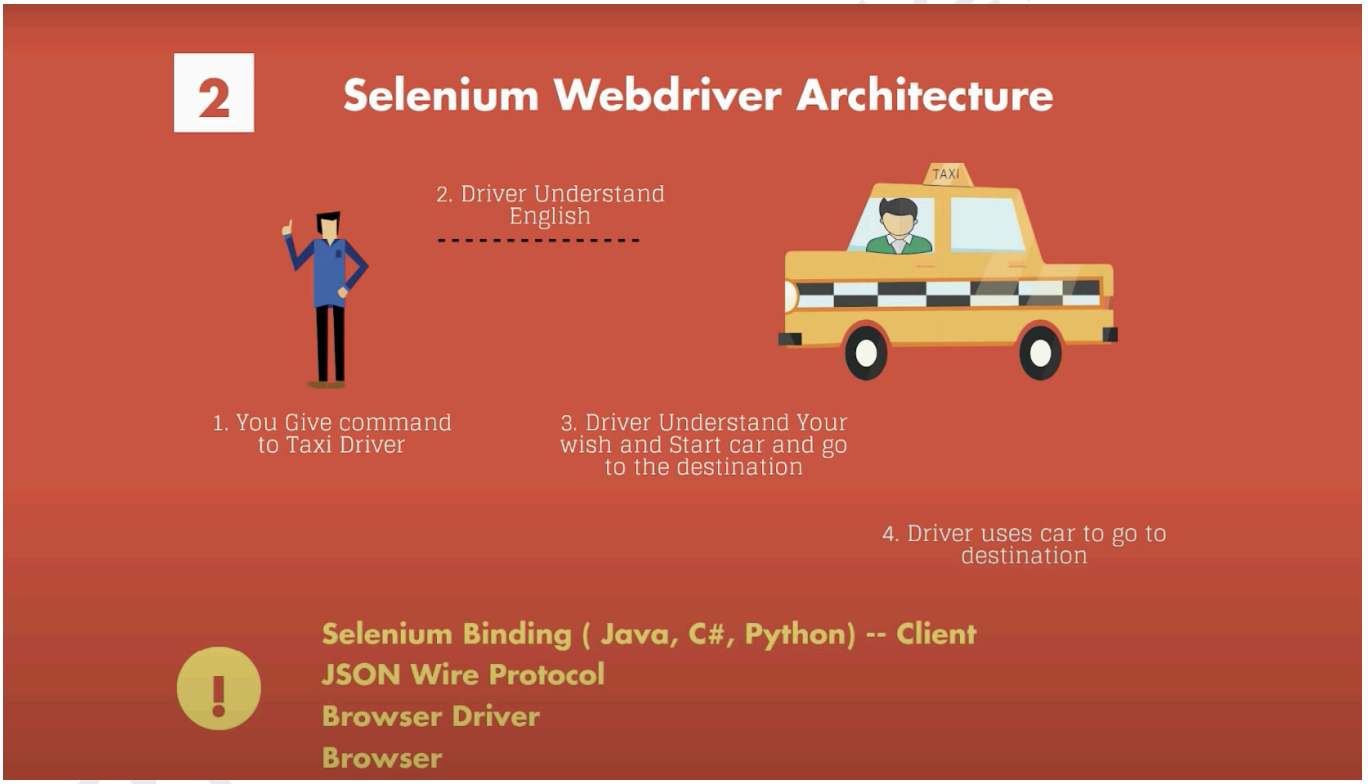
Few situations where you might not want to use Selenium for testing.

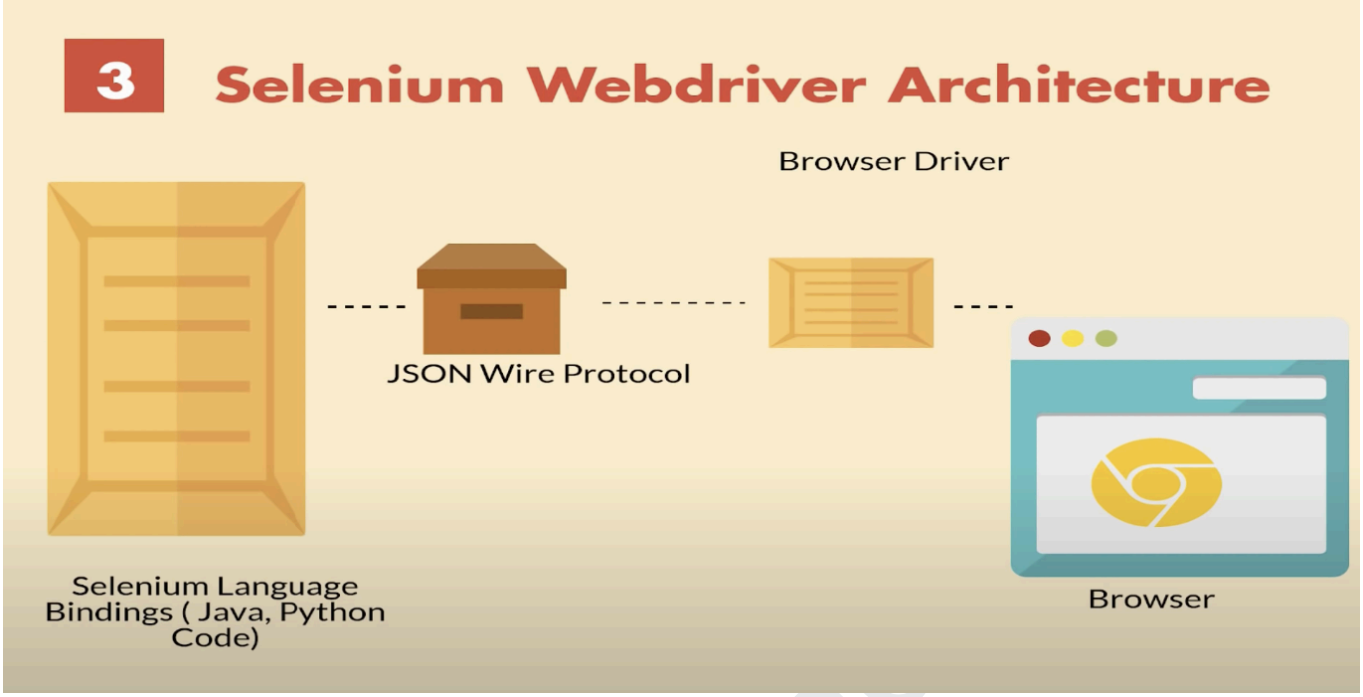
- Selenium is not well-suited for performance or load testing because it is resource-intensive and can slow down the system under test.
- When you need to test native mobile apps.
- Selenium may have difficulty interacting with custom controls or non-standard UI elements.
- Captcha / TWO-FACTOR AUTHENTICATION (2FA)
- FILE DOWNLOADS & VERIFICATION.
- AUDIO OR VIDEO STREAMING
- Security Testing
- API TESTING, mobile Appium is recommended.



WebDriver Architecture

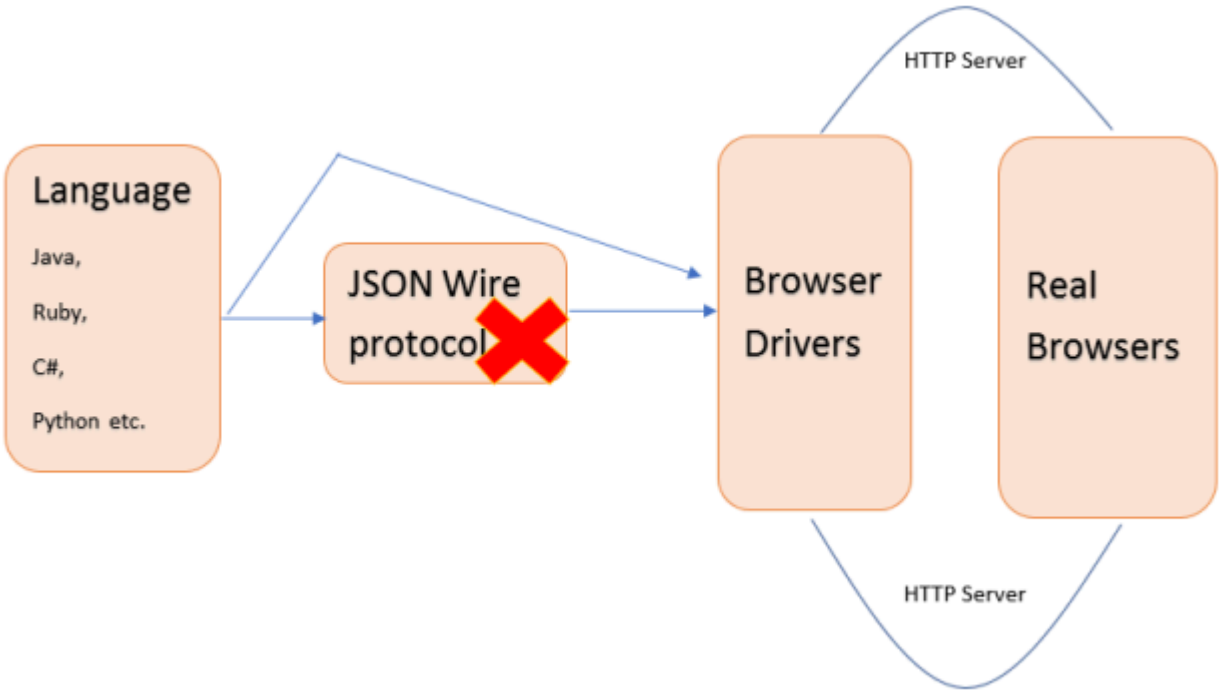
Before Selenium 4





After Selenium 4.x (w3c)

They remove the JSON wire protocol

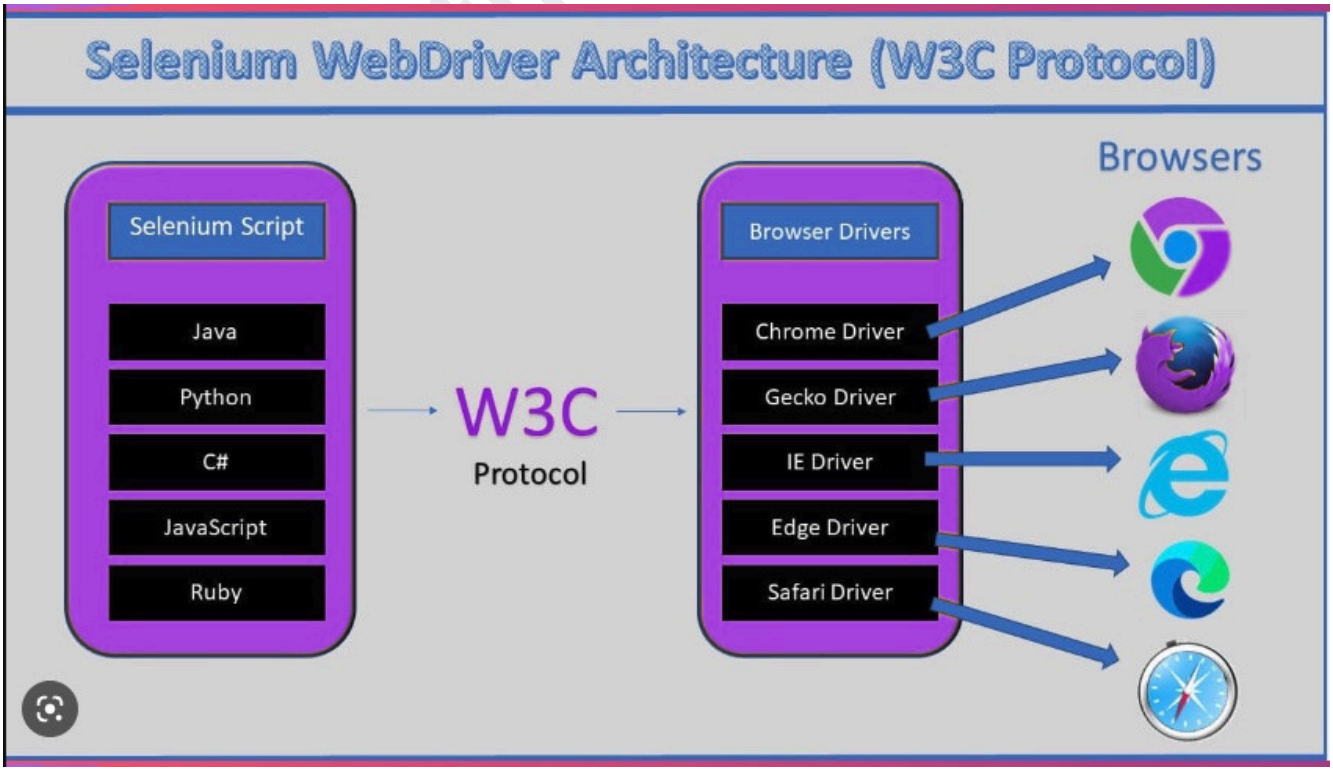


Now communicate directly to Browser via Browser Drivers.

Install Browser Drivers

Quick Reference

Browser	Supported OS	Maintained by
Chromium/Chrome	Windows/macOS/Linux	Google
Firefox	Windows/macOS/Linux	Mozilla
Edge	Windows/macOS/Linux	Microsoft
Internet Explorer	Windows	Selenium Project
Safari	macOS High Sierra and newer	Apple



HTML elements

<input type="email" class="text-input W(100%)" name="username" id="login-username" data-qa="hocewoqisi" pramod="dutta">

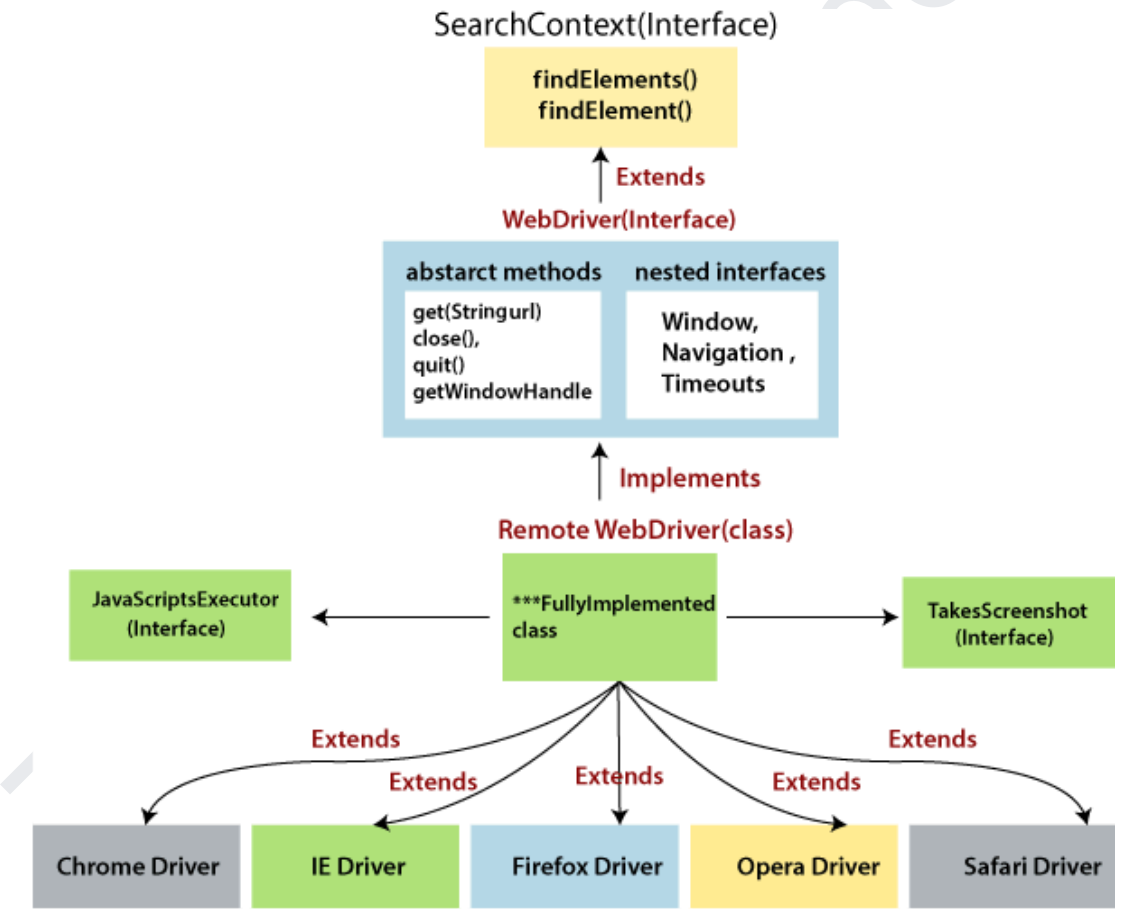
HTML Tag - input
Attribute = value

https://www.w3schools.com/html/html_elements.asp

```
package atb5x;

import org.openqa.selenium.SearchContext;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.edge.EdgeDriver;
import org.openqa.selenium.remote.RemoteWebDriver;

public class Selenium03 {
public static void main(String[] args) {

// SearchContext - Interface - findElement, and findElements - GGF
// WebDriver - Interface - some incomplete functions - GF
// RemoteWebDriver - Class- It also has some functions - F
// ChromeDriver, FirefoxDriver, EdgeDriver, SafariDriver, InternetExplorerDriver
Class - S


// Selenium - Arch - API
// Create Session, Commands or Functions -> API Request to Browser Driver (
// Pass the commands as API


// SearchContext driver = new ChromeDriver(); - GGF - this is useless - 2 functions
// WebDriver driver = new ChromeDriver(); - this contains all the functions
// RemoteWebDriver driver = new ChromeDriver(); - Fix to Remote

// ChromeDriver driver = new ChromeDriver(); - We have all Chrome func but no the
other functions
//EdgeDriver driver = new EdgeDriver();


// Scenarios
// 1. Do want to run on Chrome or Edge?
//ChromeDriver driver = new ChromeDriver();


// 2 Do you want to run on Chrome then change to Edge ?
// WebDriver driver = new ChromeDriver();
// driver = new EdgeDriver();
```

```
// 3. do you want to run on multiple browsers?
// RemoteWebDriver driver (with GRID) - Advance (Last 2 Sessions)




}
}
```

Hierarchy Of Classes & Interfaces Of WebDriver Interface



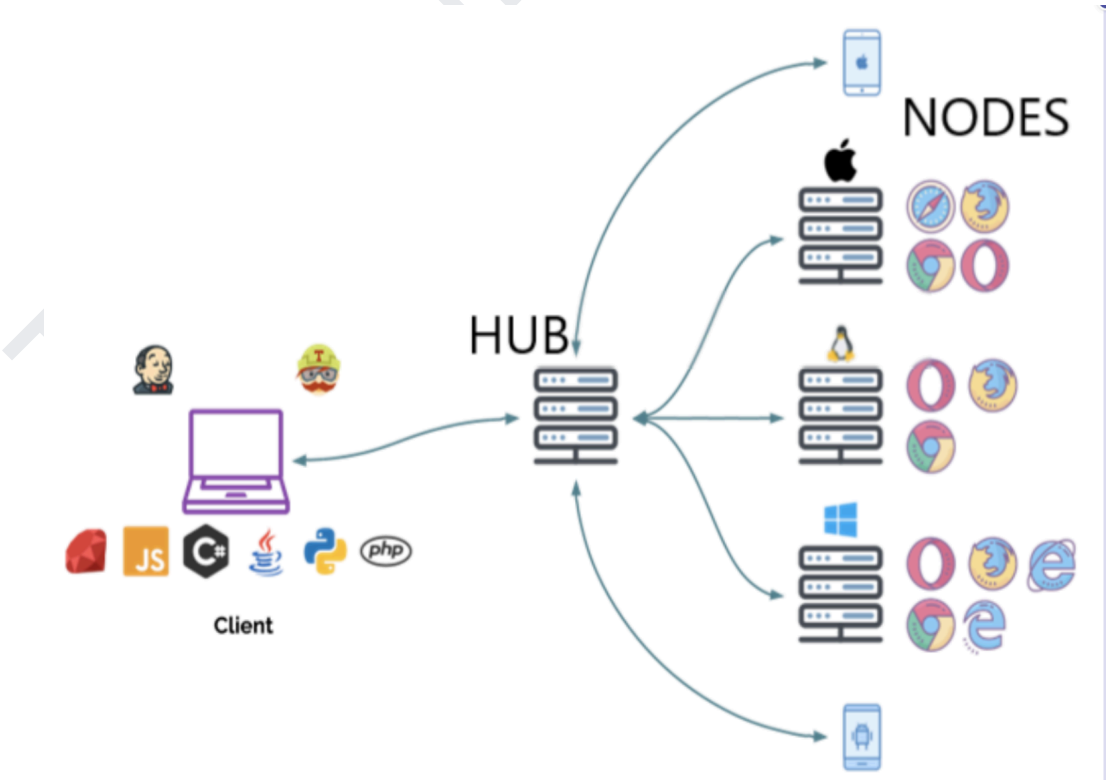
Webdriver Casting Option

Selenium IDE

- Open source record and playback test automation for the web
- Selenide Language
- Installation
- Launch the IDE
- Recording test
    - Suite
    - test
- Command-line Runner
- npm install -g selenium-side-runner
- npm install -g chromedriver
- npm install -g geckodriver
- Control Flow
- Code Export

Selenium Grid (3,4.x)

- Smart proxy server that makes it easy to run tests in parallel on multiple machines.
- Major components of Selenium Grid.
- Hub is a server that accepts the access requests from the WebDriver client, routing the JSON test commands to the remote drives on nodes. It takes instructions from the client and executes them remotely on the various nodes in parallel
- Note device that consists of a native OS and a remote WebDriver. It receives requests from the hub in the form of JSON test commands and executes them using WebDriver



When to use Selenium Grid

- Multiple browsers and their versions.
- Reduce the time that a test suite takes to complete a test.
- Cross Browser Testing.

How to Start Selenium Grid

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

java -jar selenium-server-standalone-<version>.jar -role node -hub
https://localhost:4444/grid/register

DesiredCapabilities capability = DesiredCapabilities.firefox();
WebDriver driver = new RemoteWebDriver(new
URL("https://localhost:4444/wd/hub"), capability);
```

capability.setBrowserName();
capability.setPlatform();
capability.setVersion()
capability.setCapability(,);

```
public class GridExample {

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

        DesiredCapabilities capability = DesiredCapabilities.chrome();
        capability.setBrowserName("chrome");
        WebDriver driver = new RemoteWebDriver(new URL( spec: "http://localhost:4444/wd/hub"), capability)
        driver.get("https://scrolltest.com");


    }
}
```

Selenium Grid 4

- Router - Takes care of forwarding the request to the correct component.
- Distributor - Its main role is to receive a new session request and find a suitable Node where the session can be created.
- Node - Each Node takes care of managing the slots for the available browsers of the machine where it is running.
- Session Map - Keeps the information of the session id and the Node where the session is running
- Event Bus - Event Bus serves as a communication path between the Nodes

- The Grid does most of its internal communication through messages, avoiding expensive HTTP calls

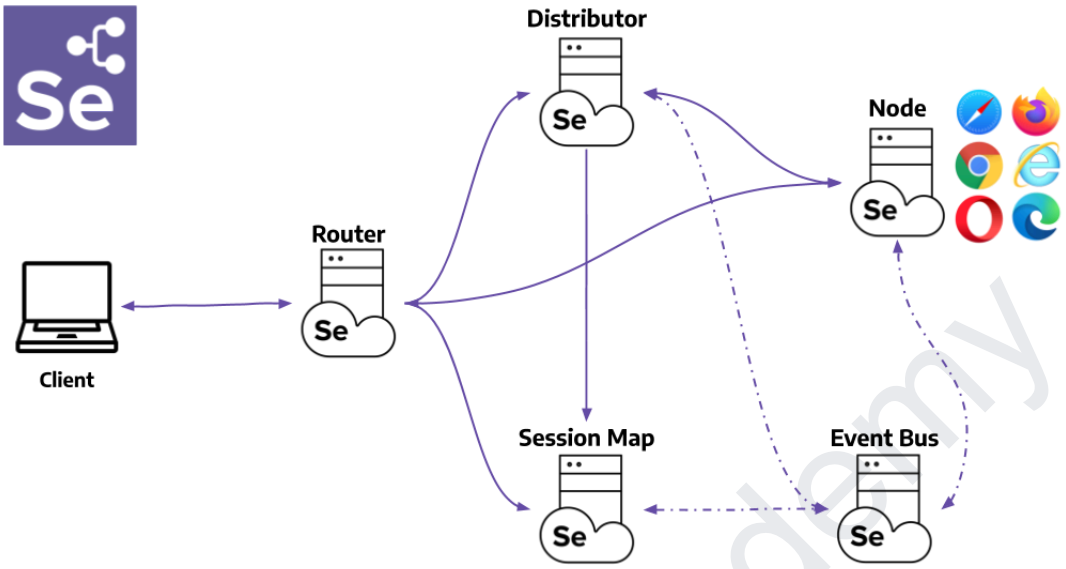
Different Grid Types

1. Standalone Mode
2. Classical Grid (Hub and Node like earlier versions)
3. Fully Distributed (Router, Distributor, Session, and Node)

Run Grid 4

```
Running in Standalone Mode  ( vs Distributed Mode)
java -jar selenium-server-4.0.0-alpha-6.jar standalone

Start the Hub:
java -jar selenium-server-4.0.0-alpha-6.jar hub


Register a Node:
java -jar selenium-server-4.0.0-alpha-6.jar node --detect-drivers
```

https://www.selenium.dev/documentation/en/grid/grid_4/setting_up_your_own_grid/



Run Selenium on Docker

```
https://github.com/SeleniumHQ/docker-selenium


docker run -d -p 4444:4444 -v /dev/shm:/dev/shm
selenium/standalone-chrome:4.0.0-alpha-7-prerelease-20201009
```

Run on Cloud Service Providers - BrowserStack

https://www.browserstack.com/

Browser driver classes and Webdriver Interface

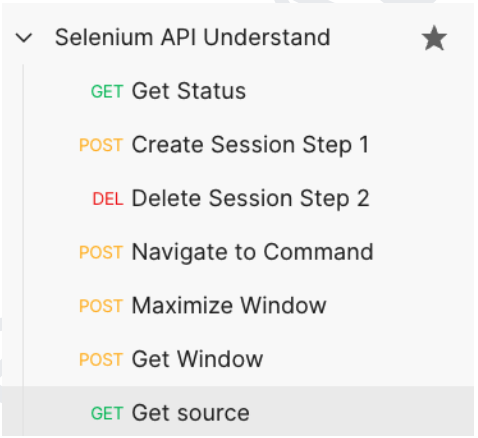
```
// Import the necessary modules
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class TestingAcademy {

    public static void main(String[] args) {
        // Set the path to the Chrome driver
```

```
        System.setProperty("webdriver.chrome.driver",
"/path/to/chromedriver");

        // Create a new Chrome browser instance
        WebDriver driver = new ChromeDriver();

        // Navigate to the Testing Academy website
        driver.get("http://thetestingacademy.com/");
    }
}
```

Understanding Selenium API

https://www.w3.org/TR/webdriver/

https://drive.google.com/drive/folders/1tYeAaBbvE6WptutWSz_pU6DOpuaf22pp?usp=sharing



🚗 WebDriver IS Class OR interface.

WebDriver is an interface that defines a set of methods for interacting with web pages. It is not a class, but rather a collection of abstract methods that must be implemented by classes that implement the WebDriver interface.

Classes that implement the WebDriver interface, such as the ChromeDriver class, must provide implementations for these methods in order to use the WebDriver interface.

ChromeDriver

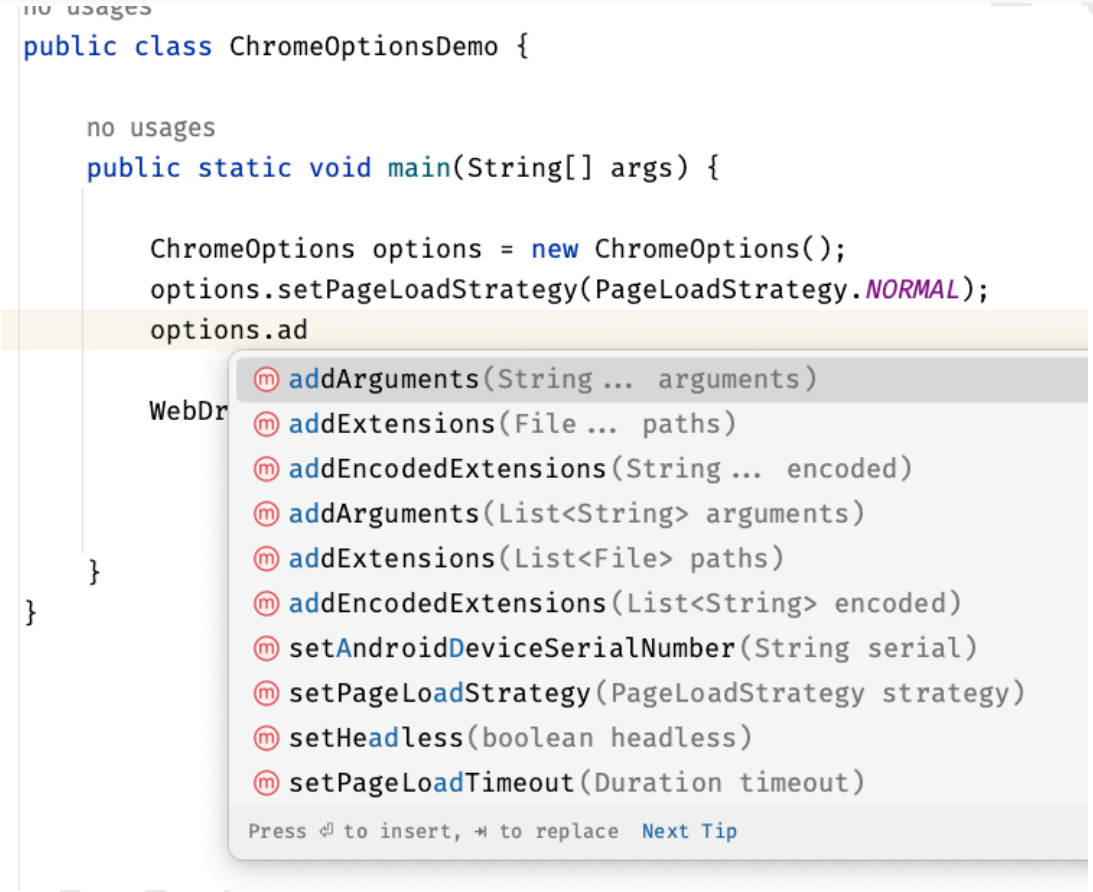
The **ChromeDriver** class provides a number of methods for interacting with the Chrome browser, such as get() for navigating to a specific URL, findElement() for locating elements

on a page, and click() for simulating a mouse click on an element. You can use these methods to automate a variety of actions on the Chrome browser.

ChromeOptions

import the ChromeOptions class from the `org.openqa.selenium.chrome` **package**

```
ChromeOptions options = new ChromeOptions();
options.setBinary("/path/to/chrome");
options.addArguments("--headless");
```




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

pageLoadStrategy

Strategy	Ready State	Notes
normal	complete	Used by default, waits for all resources to download
eager	interactive	DOM access is ready, but other resources like images may still be loading
none	Any	Does not block WebDriver at all

The `document.readyState` property of a document describes the loading state of the current document.

Proxy

A proxy server acts as an intermediary for requests between a client and a server. In simple, the traffic flows through the proxy server on its way to the address you requested and back.

```
public class proxyTest {
public static void main(String[] args) {
Proxy proxy = new Proxy();
proxy.setHttpProxy("<HOST:PORT>");
ChromeOptions options = new ChromeOptions();
options.setCapability("proxy", proxy);
WebDriver driver = new ChromeDriver(options);
driver.get("https://www.google.com/");
driver.manage().window().maximize();
driver.quit();
}
}
```

Remote WebDriver

Remote WebDriver consists of a server and a client. The server is a component that listens on a port for various requests from a Remote WebDriver client.

```
ChromeOptions chromeOptions = new ChromeOptions();
chromeOptions.setCapability("browserVersion", "67");
chromeOptions.setCapability("platformName", "Windows XP");
WebDriver driver = new RemoteWebDriver(new
URL("http://www.selenium-grid.com"), chromeOptions);
driver.get("http://www.google.com");
driver.quit();
```

Remote WebDriver is commonly used in **conjunction with a cloud-based testing service**, which allows for distributed testing across multiple machines and environments.

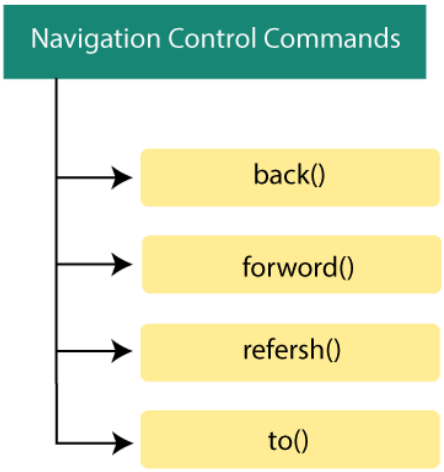
Difference Between Quit and Close in Selenium

	Quit	Close
Definition	Closes all browser windows and ends the WebDriver session	Closes the current browser window
Effect	Ends the WebDriver session completely	Leaves the WebDriver session open
Usage	Typically used at the end of a test suite or test case	Typically used at the end of a single test case
Example	`driver.quit()`	`driver.close()`

driver.close(); // Closed the window, Session id != null, Error - Invalid session Id
driver.quit(); // Closed All the window and Session = null, Error - Session ID is null

🧭Navigation in Selenium

- navigate.to, forward, back
- driver.get() vs navigate.to



Navigation commands in Selenium

get(String url) - This command is used to open a specific URL in the browser.

driver.get("https://www.example.com");
navigate().to(String url) - This command is used to navigate to a specific URL in the browser. It is similar to the get() method.

To reiterate: **`navigate().to()` and `get()` do exactly the same thing**.

https://stackoverflow.com/questions/5664808/difference-between-webdriver-get-and-webdriver-navigate

```
// Navigation
    //driver.get("app.vwo.com"); // invalid argument
    driver.get("https://app.vwo.com");

    // Navigation #2
    //driver.navigate().to("https://thetestingacademy.com");
    //driver.navigate().to(new URL("https://thetestingacademy.com"));
    driver.navigate().back();
    driver.navigate().forward();
    System.out.println(driver.getTitle());
```

driver.navigate().to("https://www.example.com");

navigate().back() - This command is used to navigate back to the previous page in the browser history.

driver.navigate().back();
navigate().forward() - This command is used to navigate forward to the next page in the browser history.

driver.navigate().forward();
navigate().refresh() - This command is used to refresh the current page in the browser.

driver.navigate().refresh();
These are some of the common navigation commands in Selenium Java. You can use these commands to navigate through web pages and perform various actions on them.

🔎 Locators in Selenium

A mechanism used to locate and interact with web elements on a web page

HTML

- HTML is the standard markup language for Web pages.
- <tagname>Content goes here...</tagname>

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>This is a Heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

List of HTML Tags

- <!-...->
- <!DOCTYPE>
- <a>
- <abbr>
- <acronym>
- <address>
- <applet>
- <area>
- <article>
- <aside>
- <audio>
- <b>
- <base>
- <basefont>
- <bdi>
- <bdo>
- <big>
- <blockquote>
- <body>
-
- <button>
- <canvas>
- <caption>
-
- <cite>
- <code>
- <col>
- <colgroup>
- <data>
- <datalist>
- <dd>
- <del>
- <details>
- <dfn>
- <dialog>
- <dir>
-
- <dl>
- <dt>
-
- <embed>
- <fieldset>
- <figcaption>
- <figure>
- <font>
- <footer>
- <form>
- <frame>
- <frameset>
- <h1>to<h6>
- <head>
- <header>
- <hr>
- <html>
- <i>
- <iframe>
- <img>
- <input>
- <ins>
- <kbd>
- <label>
- <legend>
-
- <link>
- <main>
- <map>
- <mark>
- <meta>
- <meter>
- <nav>
- <noframes>
- <noscript>
- <object>
-
- <optgroup>
- <option>
- <output>
- <p>
- <param>
- <picture>
- <pre>
- <progress>
- <q>
- <rp>
- <rt>
- <ruby>
- <s>
- <samp>
- <script>
- <section>
- <select>
- <small>
- <source>
-
- <strike>
-
- <style>
- <sub>
- <summary>
- <sup>
- <svg>
- <table>
- <tbody>
- <td>
- <template>
- <textarea>
- <tfoot>
- <th>
- <thead>
- <time>
- <title>
- <tr>
- <track>
- <tt>
- <u>
-
- <var>
- <video>
- <wbr>

A locator is a way of identifying an element on a web page so that it can be interacted with.

There are several different types of locators that can be used, including:

- ID: This locator type uses the unique ID attribute of an element to locate it on the page.
- Name: This locator type uses the name attribute of an element to locate it on the page.
- Class name: This locator type uses the class attribute of an element to locate it on the page.
- Tag name: This locator type uses the HTML tag name of an element to locate it on the page.
- Link text: This locator type uses the text of a link to locate it on the page.
- Partial link text: This locator type uses part of the text of a link to locate it on the page.

- **CSS selector**: This locator type uses a CSS selector to locate an element on the

page.

- **Xpath:** This locator type uses an XPath expression to locate an element on the page.
- When writing test scripts with Selenium, you can use a combination of these locator types to accurately and reliably locate elements on the page.

It uses "locators" to identify and manipulate elements on a web page. There are several types of locators that can be used in Selenium, including:

Make Appointment

<input type="email" class="text-input W(100%)" name="username" id="login-username" data-qa="hocewoqisi" data-gtm-form-interact-field-id="0">

1. **ID**: This locator uses the unique id attribute of an element to locate it. For example, if the HTML for an element on the page looks like this: ... , you can use the ID locator "#some-id" to find this element.
2. Name: This locator uses the name attribute of an element to locate it. For example, if the HTML for an element on the page looks like this: <input name="username">, you can use the Name locator "username" to find this element.
3. Class Name: This locator uses the class attribute of an element to locate it. For example, if the HTML for an element on the page looks like this: <dclass="some-class">./div>, you can use the Class Name locator ".some-class" to find this element.
4. Link Text: This locator uses the visible text of a link element to locate it. For example, if the HTML for a link on the page looks like this: VWO, you can use the Link Text locator "VWO" to find this element.
5. Partial Link Text: This locator is similar to the Link Text locator, but it only matches a portion of the link text. For example, using the Partial Link Text locator "VWO" would match a link with the text "Welcome to VWO".
6. CSS Selector: This locator uses a CSS selector to locate an element. CSS selectors are strings that specify how to find an element on a page based on its HTML structure. For example, if the HTML for an element on the page looks like this: <div

class="some-class" id="some-id">... , you can use the CSS selector "div.some-class#some-id" to find this element.

7. XPath: This locator uses an XPath expression to locate an element. XPath is a language for navigating and selecting elements in an XML document (including HTML documents). It allows you to specify complex, hierarchical patterns for locating elements on a page. For example, if you want to find all the <p> elements that are descendants of the element with the ID "some-id", you could use the XPath expression "//div[@id='some-id']/p" to find these elements.

These are the main types of locators that are used in Selenium. Which one you use will depend on the specific elements you are trying to locate on the page, and the HTML structure of the page itself.

[Assignment] - Automating the Login Page of VWO.com

1. Fetch the locators - https://app.vwo.com/
2. Create a Maven project and add TestNG.
3. Add the Allure Report (Allure TestNG)
4. Automate the two Test cases of VWO.com
    a. Valid Username and Valid Password
    b. Verify name on dashboard page.
5. Run them and share results.
6. Push the code to github.com
7. Git repo - ReadMe.md a Screen shot of allure.

```
package com.tta.selenium4demo.assignment;

import io.qameta.allure.Description;
import org.openqa.selenium.By;
import org.openqa.selenium.PageLoadStrategy;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import org.testng.Assert;
import org.testng.annotations.AfterSuite;
import org.testng.annotations.BeforeSuite;
import org.testng.annotations.Test;

public class LoginPageAutomation {
    ChromeOptions options;
    WebDriver driver;
```

```
    @BeforeSuite
    public void setUp() {
        options = new ChromeOptions();
        options.setPageLoadStrategy(PageLoadStrategy.EAGER);
        driver = new ChromeDriver(options);
    }

    @Test
    @Description("Verify that with Valid username and Valid password,
Login is successfull !!")
    public void testValidLogin() throws InterruptedException {

        driver.get("https://app.vwo.com/#/login");

driver.findElement(By.id("login-username")).sendKeys("93npu2yyb0@esiix.c
om");

driver.findElement(By.id("login-password")).sendKeys("Wingify@123");
        driver.findElement(By.id("js-login-btn")).click();
        Thread.sleep(5000);

Assert.assertTrue(driver.findElement(By.cssSelector(".page-heading")).is
Displayed());

    }

    @AfterSuite
    public void tearDown() {
        driver.quit();
    }
}
```


Understanding Locators and HTML Forms

Tag
Attribute = Value

<input data-qa="hocewoqisi" type="email" class="text-input W(100%)" name="username" id="login-username" >

data-qa="hocewoqisi"
type="email"
class="text-input W(100%)"
name="username"
id="login-username"

---

findElement vs findElements

findElement() is a method used to locate a single element on a web page. It takes a locator as an argument, and returns the first matching element that it finds. For example:

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

In this example, findElement() is used to locate the element with the ID "username". If it is found, the element is returned and stored in the usernameField variable.

findElements() is similar to findElement(), but it returns a list of all matching elements instead of just the first one. For example:

```
List<WebElement> allLinks = driver.findElements(By.tagName("a"));
```

In this example, findElements() is used to locate all <a> elements on the page. These elements are returned in a list and stored in the allLinks variable.

It's important to note that if findElement() is used and no matching element is found, it will throw a NoSuchElementException. On the other hand, if findElements() is used and no matching elements are found, it will return an empty list.

What is an HTML Form?

A HTML form is a section of a web page that contains form elements, such as text fields, checkboxes, and buttons. Forms allow users to enter data and interact with a website.

Forms are created using the <form> HTML tag. This tag defines the start and end of a form, and it can have several attributes that determine how the form behaves.

For example, the action attribute specifies the URL of the server-side script that will process the form data, and the method attribute specifies whether the form data will be sent to the server using the GET or POST method.

```
<form action="http://www.example.com/form-handler.php" method="POST">
  <label for="username">Username:</label>
  <input type="text" id="username" name="username">
  <br>
  <label for="password">Password:</label>
  <input type="password" id="password" name="password">
  <br>
  <input type="submit" value="Log In">
</form>
```

When the user enters their username and password and clicks the "Log In" button, the form data will be sent to the server-side script at the URL specified in the action attribute (`http://www.example.com/form-handler.php`) using the POST method.

The server-side script can then process the form data and perform the desired action, such as checking the user's credentials against a database and logging them in.

getText() Method

the getText() method is used to retrieve the text of an element on a web page. This method can be called on an element, and it will return the text of the element, including any child elements.

```
WebElement element = driver.findElement(By.id("some-id"));
String elementText = element.getText();
```

getAttribute() Method

the getAttribute() method is used to retrieve the value of an attribute of an element on a web page.

```
WebElement element = driver.findElement(By.id("some-id"));
String attributeValue = element.getAttribute("class");
```

sendKeys

the sendKeys() method is used to enter text into a text field or text area on a web page

click()

the click() method is used to simulate a user clicking on an element on a web page

⚒️ SelectorsHub for the Locators

SelectorsHub is a tool that can be used to help identify and generate locators for elements on a web page in Selenium

https://selectorshub.com/

[Assignment] - Invalid error message Capture for the Login Page of VWO.com

8. Fetch the locators - https://app.vwo.com/
9. Create a Maven project and add TestNG.
10. Add the Allure Report (Allure TestNG)
11. Automate the two Test cases of VWO.com
    a. Invalid Username and Valid Password.
12. Capture the error and pass the test.
13. Run them and share results.

Link Text locator.

the findElement() method is used with the By.linkText() locator to locate a link on the page with the text "VWO". The element is then stored in the vwoLink variable.

The click() method is called on this element, which simulates the user clicking on the link with their mouse. This will navigate the user to the URL specified in the href attribute of the <a> element.

```
WebElement vwoLink = driver.findElement(By.linkText("VWO"));
vwoLink.click();
```

🛣️ Mastering XPath

What is XPath?

XPath is a query language for selecting nodes from an XML document.
XPath was defined by the World Wide Web Consortium

Core Logic - **//tagName[@attribute='value']**

TAG - h1, p, input, a, form, img, video, audio,button, table, ul, li, tr, div, select, span, -> Html Tags

Attrbute - id, class, name, alt, href, src, data-qa, ….srcset ..

- **Relative XPath**
- Absolute XPath

- XPath Functions

Absolute XPath
/html/body/div[2]/div[1]/div[2]/div/div[1]/div/div/div[3]/form[1]/ul/li[1]/div/input

Why do we need to MASTER Locators?

Probably the first question asked by the interviewer.

You should always find small and efficient Locators.
UI Automation is all about finding locators.
Don't use tools at first.

Expression	Description
nodename	Selects all nodes with the name "*nodename*"
/	Selects from the root node
//	Selects nodes in the document from the current node that mat they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

Absolute XPath

Complete path from the Root Element.
If any element is added or deleted, Xpath fails.
/html/body/div[2]/div[1]/div[2]/div/div[1]/div/div/div[3]/form[1]/ul/li[1]/div/input

Relative Xpath

You can simply start by referencing the element you want and go from there
Based on searching an element in DOM. //*[@id="login-username"]

//input[@id="login-username"]

Xpath -> //input[@id="txt-username"]

Css - > #txt-username

XPath Functions

Contains()
//tag_name[contains(@attribute,'value_of_attribute')]

Starts-with()
//tag_name[starts-with(@attribute,'Part_of_Attribute_value')]
Text()
//tag_name[text()='Text of the element']

String functions

concat(string, ...): XPath concat function concatenated number of arguments and return to a concatenated string.

starts-with(string, string): XPath start-with function return True/False. Return True if second argument string is start with first argument.

contains(string, string) - XPath contains function return True/False. Return True if second argument string is a contain of first argument.
string-length(string): XPath string-length function return the length of string.

substring-after(string, string): XPath substring-after function return the substring of the first argument string base on first occurrence of the second argument string after all character.

substring-before(string, string): XPath substring-before function return the substring of the first argument string base on first occurrence of the second argument string before all character.

normalize-space(string): XPath normalize-space function sequence of whitespace combine into single normalize space and removing leading and trailing whitespace.

Operators - AND & OR

And Example
//tag_name[@name = 'Name value' and @id = ‘ID value’]

https://katalon-demo-cura.herokuapp.com/

//a[text()="Make Appointment" and contains(@id,"btn-make-appointment")]

OR Example
//input[@placeholder ='Full Name' or @type = 'text']

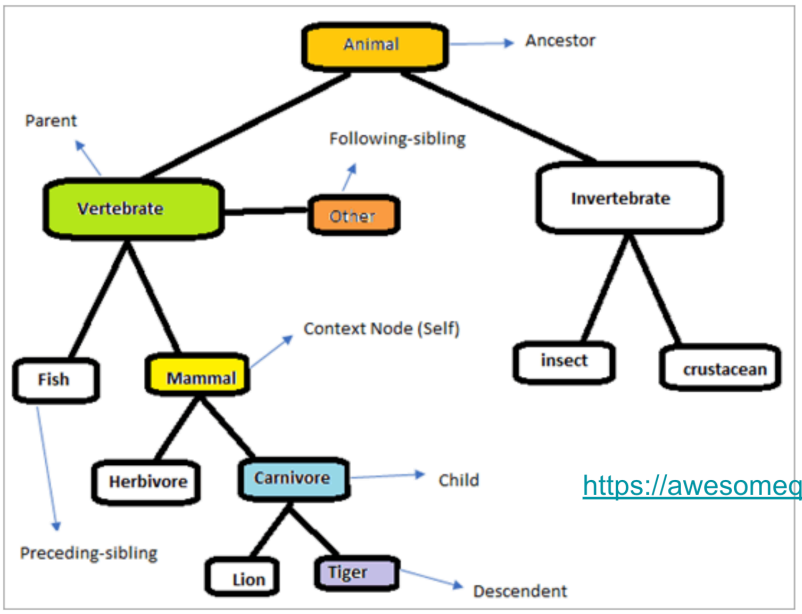
Operator	Description
|	Computes two node-sets
+	Addition
-	Subtraction
*	Multiplication
div	Division
=	Equal
!=	Not equal
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
or	or
and	and
mod	Modulus (division remainder)

XPath Axes

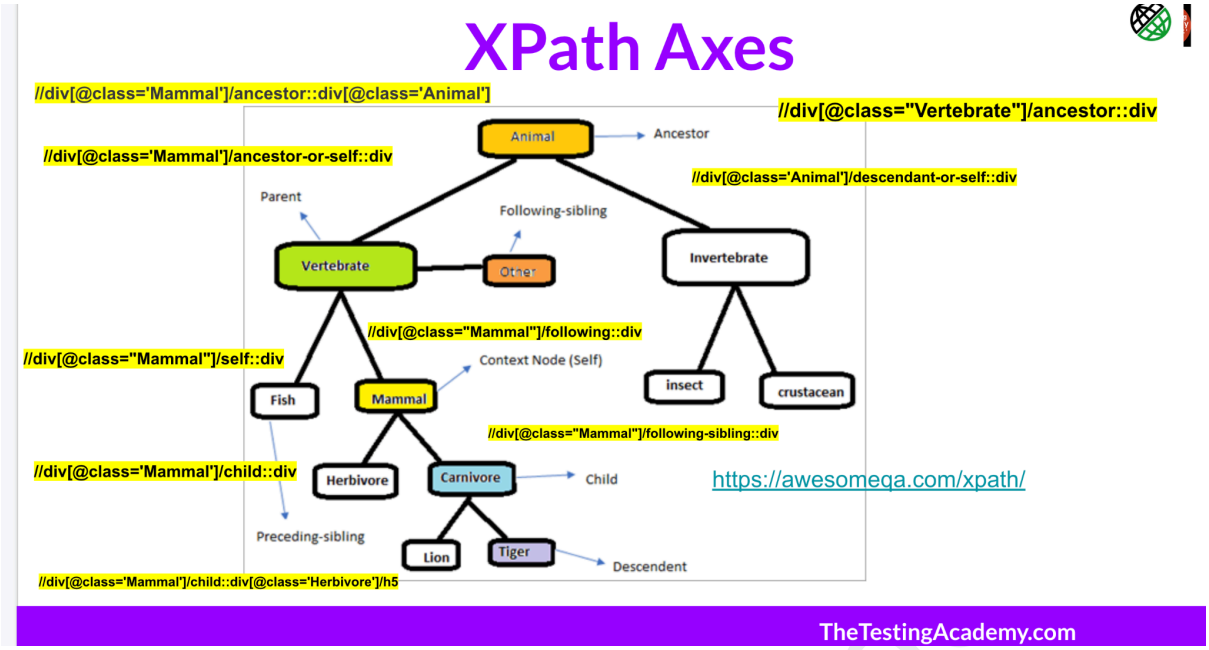
In the XML documents, we have relationships between various nodes to locate those nodes in the DOM structure.

- Ancestor
- Child, parent
- Descendant
- Following, following-sibling
- Self.

XPath Axes



https://awesomeqa.com/xpath/



AxisName	Result
ancestor	Selects all ancestors (parent, grandparent, etc.) of the current node
ancestor-or-self	Selects all ancestors (parent, grandparent, etc.) of the current node and the current node itself
attribute	Selects all attributes of the current node
child	Selects all children of the current node
descendant	Selects all descendants (children, grandchildren, etc.) of the current node
descendant-or-self	Selects all descendants (children, grandchildren, etc.) of the current node and the current node itself
following	Selects everything in the document after the closing tag of the current node
following-sibling	Selects all siblings after the current node
namespace	Selects all namespace nodes of the current node
parent	Selects the parent of the current node
preceding	Selects all nodes that appear before the current node in the document, except ancestors, attribute nodes and namespace nodes
preceding-sibling	Selects all siblings before the current node
self	Selects the current node

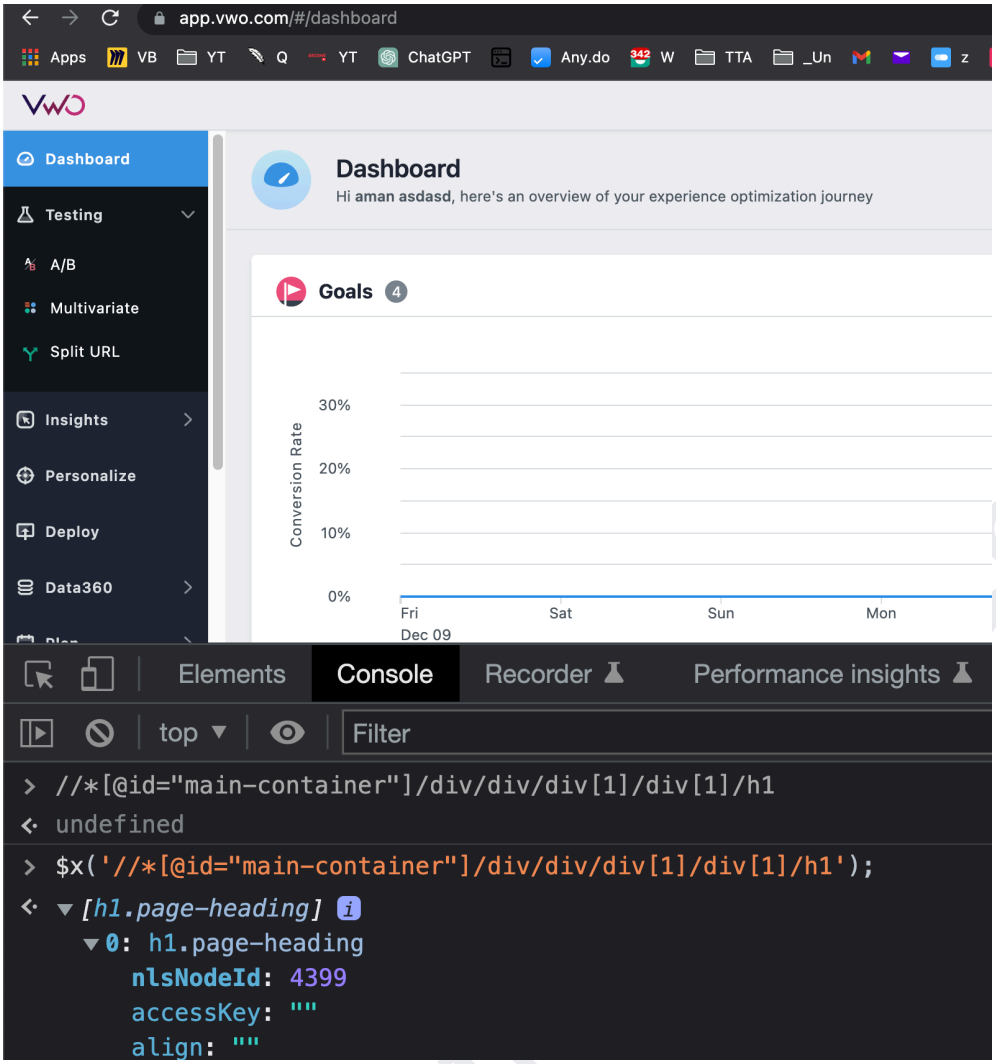
https://www.softwaretestinghelp.com/xpath-axes-tutorial/

//span[text()='Invalid Email']/ancestor::div

//*[@id="main-page"]/div[1]/child::div

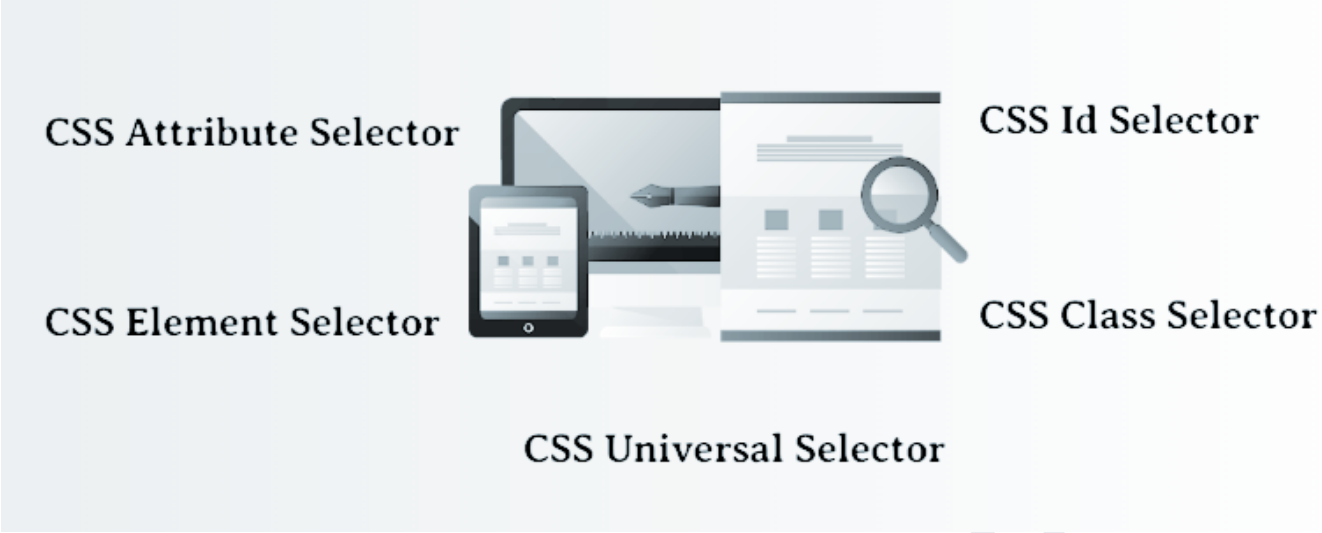
//*[@id="js-main-container-wrap"]/child::div
//*[@id="js-main-container-wrap"]/following::div

https://devhints.io/xpath



⚠️ Master CSS Selectors

CSS selectors are used to select elements in an HTML or XML document in order to apply styles or other manipulations to those elements.



CSS Selectors?

#id
.class
div.first > span

Demo https://awesomeqa.com/css/

div.first > span:nth-child(3)

li:nth-of-type(even)

div.first > span:nth-child(2n+1)

Direct Child Selector > p > span

div.first > span:nth-of-type(2n+1)

Wildcard Selectors (*, ^ and $) in CSS

div.first > span:first-child

[attribute*=”str”] Selector:

- * contains.
- ^ begins with
- $ ends with

div.first > span:last-child

CSS selectors allow you to select elements based on their tag name, id, class, attribute, and other characteristics.

- To select all elements with the tag "p" (paragraph), you could use the following selector: p
- To select an element with the ID "main-heading", you could use the following selector: #main-heading
- To select all elements with the class "error", you could use the following selector: .error
- To select all elements with the attribute "disabled", you could use the following selector: [disabled]
- To select all "a" elements that are descendants of a "nav" element, you could use the following selector: nav a

form#login-form input[type="radio"]

CSS [attribute*=value] Selector

The [attribute*=”str”] selector is used to select those elements whose attribute value contains the specified substring str.

CSS [attribute=value] Selector
The [attribute=value] selector in CSS is used to select those elements whose attribute value is equal to “value”.

CSS [attribute$=value] Selector The [attribute$=”value”] selector is used to select those elements whose attribute value ends with a specified value “value”.
CSS [attribute|=value] Selector This is used to select those elements whose attribute value is equal to “value” or whose attribute value started with “value” immediately followed by hyphen (-).

CSS [attribute~=value] Selector The [attribute~=”value”] selector is used to select those elements whose attribute value contains a specified word.

CSS [attribute^=value] Selector The [attribute^=value] selector is used to select those elements whose attribute value begins with given attribute.

CSS :first-child Selector The :first-child selector is used to select those elements which are the first-child elements.

CSS :last-child Selector The :last-child Selector is used to target the last child element of it’s parent for styling.

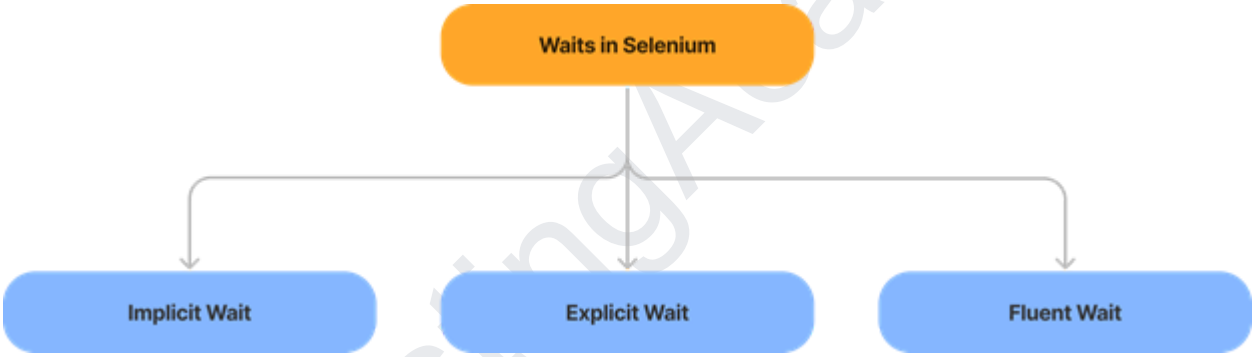
CSS :nth-child() Selector The :nth-child() CSS pseudo-class selector is used to match the elements based on their position in a group of siblings.

CSS :nth-of-type() Selector The :nth-of-type() in css Selector is used to style only those elements which are the nth number of child of its parent element.

⌛ Selenium Waits

Why Do We Need Waits In Selenium?

- Web applications are developed using Ajax and Javascript.
- New JS frameworks are more advanced and use AJax, react, and angular.
- elements which we want to interact with may load at different time intervals.



Implicit Wait

- Selenium Web Driver has borrowed the idea of implicit waits from Watir.
- If the element is not located on the web page within that time frame, it will throw an **exception**.
- WebDriver polls the DOM for a certain duration when trying to find any element.
- Global settings applicable to all elements
- It tells the web driver to wait for the x time before moving to the next command.
- Gives No Such Element Exception.
- Once it is set it is applicable to full automation script.
- Implicit wait is maximum time between the two commands.
- Different from Thread.sleep - Thread.sleep() - It will sleep time for script.
- Not good way to use it in script as it's sleep without condition.
- Do not mix implicit and explicit waits. Doing so can cause unpredictable wait times.

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

Explicit Wait

Explicit Wait in Selenium is used to tell the Web Driver to wait for certain conditions (Expected Conditions) or maximum time exceeded before throwing "ElementNotVisibleException" exception

- Little intelligent wait, wait for certain conditions.
- They **allow your code to halt program execution, or freeze the thread, until the condition you pass it resolves.**
- The condition is called with a certain frequency until the timeout of the wait is Elapsed.
- This means that for as long as the condition returns a falsy value, it will keep trying and waiting.
- It provides better way to handle the dynamic Ajax elements
- Element not visible exception if element not found.
- Good fit for synchronizing the state between the browser and its DOM, and your.
- Replace Thread.sleep with explicit wait always

```
WebDriver script.
WebDriver driver = new ChromeDriver");
driver.get("https://google.com/ncr");
driver.findElement(By.name ("g")).sendKeys("Scroll"test + Keys.ENTER);
/l Initialize and wait till element(link) became clickable - timeout in
10 seconds
WebElement firstResult = new WebDriverWait(driver,
Duration.ofSeconds(10))
.until(ExpectedConditions.elementToBeClickable(By.xpath("//a/h3")));
// Print the first result

System.out.printIn(firstResult.getText));
```

The following are the Expected Conditions that can be used in Selenium Explicit Wait

- alertIsPresent()
- elementSelectionStateToBe()
- **elementToBeClickable()**
- elementToBeSelected()
- frameToBeAvaliableAndSwitchToIt()
- invisibilityOfTheElementLocated()
- invisibilityOfElementWithText()
- presenceOfAllElementsLocatedBy()
- presenceOfElementLocated()
- textToBePresentInElement()
- textToBePresentInElementLocated()
- textToBePresentInElementValue()

- titleIs()
- titleContains()
- visibilityOf()
- visibilityOfAllElements()
- visibilityOfAllElementsLocatedBy()
- visibilityOfElementLocated()

[Assignment] Fix the VWO login page with the heading page visibility, Use Expected Condition

```
package com.tta.selenium4demo.assignment;

import io.qameta.allure.Description;
import org.openqa.selenium.By;
import org.openqa.selenium.PageLoadStrategy;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
import org.testng.Assert;
import org.testng.annotations.AfterSuite;
import org.testng.annotations.BeforeSuite;
import org.testng.annotations.Test;

import java.time.Duration;

public class LoginPageAutomationWaits {
    ChromeOptions options;
    WebDriver driver;

    @BeforeSuite
    public void setUp() {
        options = new ChromeOptions();
        options.setPageLoadStrategy(PageLoadStrategy.EAGER);
        driver = new ChromeDriver(options);
    }

    @Test
    @Description("Verify that with Valid username and Valid password,
Login is successfull !!")
    public void testValidLogin() throws InterruptedException {

        driver.get("https://app.vwo.com/#/login");
```

```
driver.findElement(By.id("login-username")).sendKeys("93npu2yyb0@esiix.com");

driver.findElement(By.id("login-password")).sendKeys("Wingify@123");
        driver.findElement(By.id("js-login-btn")).click();
        WebElement element = new WebDriverWait(driver, Duration.ofSeconds(5))

.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector(".page-heading")));


Assert.assertTrue(driver.findElement(By.cssSelector(".page-heading")).isDisplayed());

    }

    @AfterSuite
    public void tearDown() {
        driver.quit();
    }
}
```

Fluent Wait

Fluent Wait instance defines the maximum amount of time to wait for a condition as well as the frequency with which to check the condition

- Exception - NoSuchElementException
- Waiting 30 seconds for an element to be present on the page, checking for its presence once every 5 seconds.

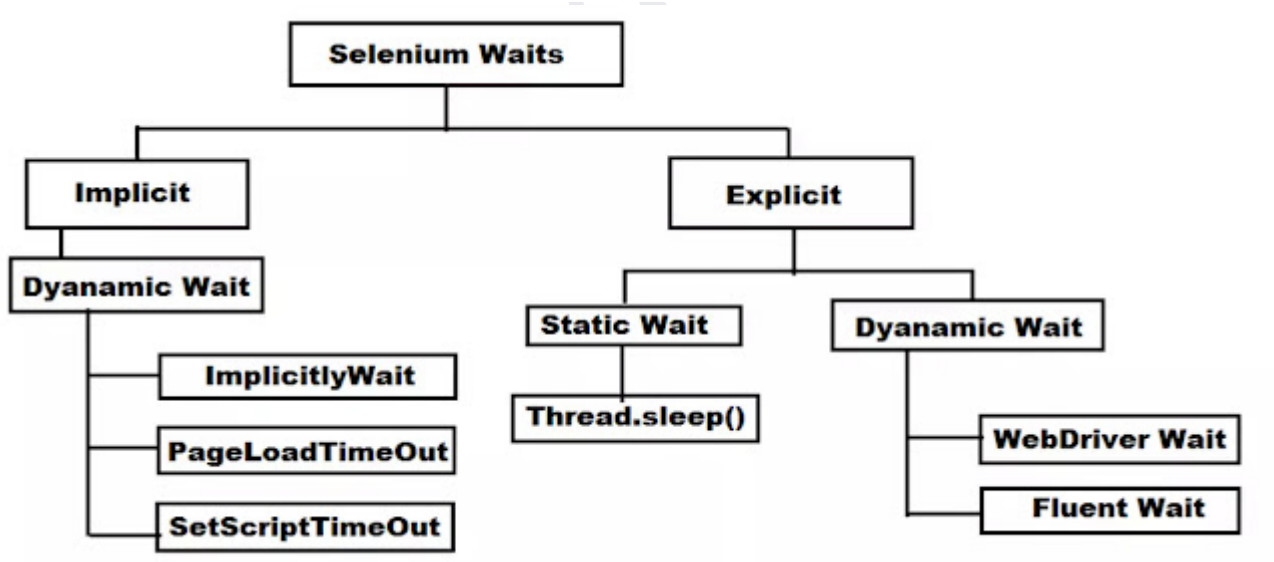
```
Wait<WebDriver wait = new FluentWait<WebDriver(driver)
.withTimeout(Duration.ofSeconds(30))
.pollingEvery(Duration.ofSeconds(5))
•ignoring(NoSuchElementException.class);

WebElement foo = wait.until(new Function<WebDriver, WebElement0 f
public WebElement apply (WebDriver driver) {
return driver.findElement(By.id("foo"));
```

https://www.selenium.dev/documentation/en/webdriver/

waits/

Ref - https://www.guru99.com/implicit-explicit-waits-selenium.html

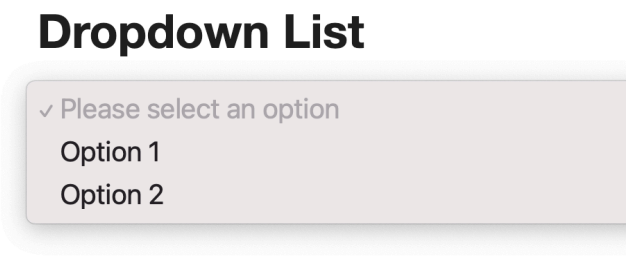
Implicit Wait	Explicit Wait
• Implicit Wait time is applied to all the elements in the script	• Explicit Wait time is applied only to those element by us
• In Implicit Wait, we need **not** specify "ExpectedConditions" on the element to be located	• In Explicit Wait, we need to specify "ExpectedCon element to be located
• It is recommended to use when the elements are located with the time frame specified in Selenium implicit wait	• It is recommended to use when the elements are load and also for verifying the property of the elen like(visibilityOfElementLocated, elementToBeClickable,elementToBeSelected)



Select Demo, Static and Dynamic Dropdowns

Handling Static Dropdowns

https://the-internet.herokuapp.com/dropdown



Handling Dynamic Dropdowns

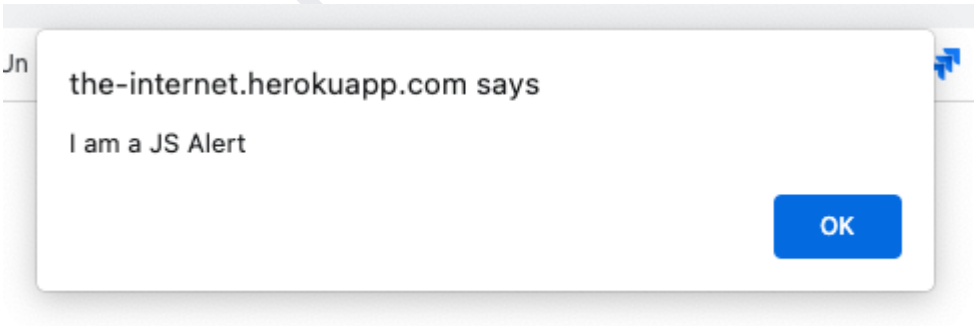
1. We will use the XPath axes.
2. We will use the advanced css selectors for the same.
3. Traditional select classes won't work.
4. Action class and JS Executor

Alert in Selenium

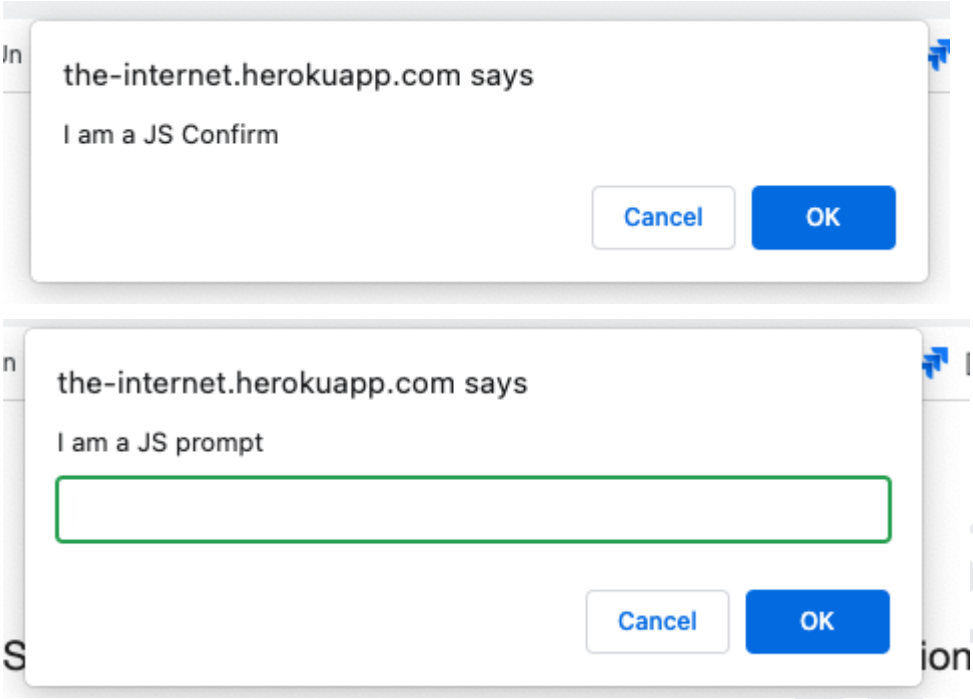
An alert is a small window that appears on top of a web page and displays a message to the user. Most of the time, alerts are used to show important information or ask the user for something.

https://the-internet.herokuapp.com/javascript_alerts

Prompt Alert



Confirmation Alert



Handle Alert in Selenium WebDriver

```
WebDriver driver = new ChromeDriver();

// Navigate to a web page that displays an alert
driver.get("http://example.com/page-with-alert");

// Wait for the alert to appear
WebDriverWait wait = new WebDriverWait(driver, 10);
wait.until(ExpectedConditions.alertIsPresent());

// Get the alert
Alert alert = driver.switchTo().alert();

// Get the text of the alert message
String alertText = alert.getText();

// Accept the alert (click the "OK" button)
alert.accept();
```

1**) void dismiss() // To click on the 'Cancel' button of the alert.**

driver.switchTo().alert().dismiss();
2) void accept()// To click on the 'OK' button of the alert.

driver.switchTo().alert().accept();
3) String getText() // To capture the alert message.

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

4) void sendKeys(String stringToSend) // To send some data to alert box.

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

Full Demo Alert

```
package com.tta.selenium4demo.alerts;

import org.openqa.selenium.Alert;
import org.openqa.selenium.By;
import org.openqa.selenium.PageLoadStrategy;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import org.openqa.selenium.interactions.Actions;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
import org.testng.Assert;

import java.time.Duration;

public class AlertDemo {

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

        //https://the-internet.herokuapp.com/javascript_alerts

        ChromeOptions options = new ChromeOptions();
        options.setPageLoadStrategy(PageLoadStrategy.NORMAL);
        WebDriver driver = new ChromeDriver();

driver.get("https://the-internet.herokuapp.com/javascript_alerts");


        //
driver.findElement(By.xpath("//button[@onclick=\"jsAlert()\"]")).click()
;
//
driver.findElement(By.xpath("//button[@onclick=\"jsConfirm()\"]")).click
();

driver.findElement(By.xpath("//button[@onclick=\"jsPrompt()\"]")).click(
);
        WebDriverWait wait = new WebDriverWait(driver,
Duration.ofSeconds(10));
```

```
        wait.until(ExpectedConditions.alertIsPresent());


// Get the alert
        Alert alert = driver.switchTo().alert();
//        String alertText = alert.getText();
        alert.sendKeys("Pramod");
        alert.accept();

        String result = driver.findElement(By.id("result")).getText();
//        Assert.assertEquals(result,"You successfully clicked an
alert");
//        Assert.assertEquals(result,"You clicked: Ok");
//        Assert.assertEquals(result,"You entered: Pramod");

        Thread.sleep(20000000);



    }
}
```

Handling Checkboxes and Handling Radio Buttons

Checkboxes and radio buttons are types of form elements that allow users to make multiple selections or choose a single option from a group of options.

https://the-internet.herokuapp.com/checkboxes

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

    // Iterate through the checkbox elements
    for (WebElement checkbox : checkboxes) {
        // Check the checkbox if it is not already checked
        if (!checkbox.isSelected()) {
            checkbox.click();
        }
    }
```

Web Table in Selenium

What is a Web Table?

A web table is a way of representing data in rows and columns

"<table>" - It defines a table. You can also say that it's the starting point of a table.

<thead>
<tbody>

"<th>" - It defines a header cell, which means you should define your headings inside th tag.
"<tr>" - It defines a row in a table.
"<td>" - It defines a cell in a table. "td" always lie inside the tr tag.

```
//table[@id="customers"]
```

//table[contains(@id,"cust")]

Static Table - Data will not change.
Dynamic Table - No of Col may change.


```
package com.tta.selenium4learning.webtabledemo;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.Select;

import java.util.List;

public class WebTableDemo01 {
   public static void main(String[] args) {
       WebDriver driver = new FirefoxDriver();
       String URL = "https://awesomeqa.com/webtable.html";
       driver.get(URL);
       driver.manage().window().maximize();

       /*
       * //table[@id="customers"] -
       * //table[contains(@id,"cust")]
       * Row - //table[@id="customers"]/tbody/tr
       * //table[@id="customers"]/tbody/tr[2]
       *
       * */

       // Number of Rows and Column in table
```

```
        // Divide the path

        int row =
driver.findElements(By.xpath("//table[@id=\"customers\"]/tbody/tr")).si
ze();
        int col =
driver.findElements(By.xpath("//table[@id=\"customers\"]/tbody/tr[2]/td
")).size();

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

        String first_part = "//table[@id=\"customers\"]/tbody/tr[";
        String second_part = "]/td[";
        String third_part = "]";



        for (int i = 2; i <= row; i++) { //
//table[@id="customers"]/tbody/tr[1]/th[1] - First is Header
            for (int j = 1; j <= col ; j++) {
                String dynamic_xpath =
first_part+i+second_part+j+third_part;
                String data =
driver.findElement(By.xpath(dynamic_xpath)).getText();
                System.out.print(data +" ");

            }
        }


        // Who is in Google -
        //
//table[@id="customers"]/tbody/tr[2]/td[1]/following-sibling::td[1]
        // Google In Which Country -
        // //table[@id="customers"]/tbody/tr[2]/td[1]/following::td[2]




        // Find Helen Bennett In Which Country
        for (int i = 2; i <= row; i++) { //
//table[@id="customers"]/tbody/tr[1]/th[1] - First is Header
            for (int j = 1; j <= col ; j++) {
                String dynamic_xpath =
first_part+i+second_part+j+third_part;
                String data =
driver.findElement(By.xpath(dynamic_xpath)).getText();
                if(data.contains("Helen Bennett")){
                    String country_path =
dynamic_xpath+"/following-sibling::td";
```

```
                    String country_text =
driver.findElement(By.xpath(country_path)).getText();
                    System.out.println("------");
                    System.out.println("Helen Bennett is In - " +
country_text);
                }

            }
        }

        System.out.println(" |||||||||||||||||||||| \n");

        driver.get("https://awesomeqa.com/webtable1.html");

        // For Dyamic Col use the tagName
        // Get Table
        WebElement table =
driver.findElement(By.xpath("//table[@summary='Sample Table']/tbody"));
        List<WebElement> rows_table =
table.findElements(By.tagName("tr"));
        for (int i = 0;i < rows_table.size(); i++) {
            List<WebElement> columns_table =
rows_table.get(i).findElements(By.tagName("td"));
            for (WebElement element : columns_table) {
                System.out.println(element.getText());
            }

        }



    }
}
```

Actions, Windows and iframe.

Actions class is an ability provided by Selenium for handling keyboard and mouse events.

- Keyboard Events
- Mouse Events

```
Actions action = new Actions(driver);
action.moveToElement(element).click().perform();
```

Methods of Action Class

Action class is useful mainly for mouse and keyboard actions. In order to perform such actions, Selenium provides various methods.

Mouse Actions in Selenium:

1. **Perform Mouse Hover Action on the Web Element**
2. **moveToElement(live).build().perform();**
3. **doubleClick():** Performs double click on the element
4. **clickAndHold()**: Performs long click on the mouse without releasing it
5. **dragAndDrop():** Drags the element from one point and drops to another
6. **moveToElement():** Shifts the mouse pointer to the center of the element
7. **contextClick():** Performs right-click on the mouse Keyboard Actions in Selenium
8. **sendKeys()**: Sends a series of keys to the element
9. **keyUp():** Performs key release
10. **keyDown():** Performs keypress without release.

Keyboard Events

i. keyDown(KeyCode) - Performs key press without releasing it.
Parameters - Key_Code For e.g., Keys.ALT, Keys.SHIFT or Keys.CONTROL

ii. keyUp(KeyCode) - Performs a key release. It has to be used after keyDown to release the key.
Parameters - Key_Code For e.g., Keys.ALT, Keys.SHIFT or Keys.CONTROL

Drag and Drop

With Action or with Function

```
builder.clickAndHold(from)
          .moveToElement(to)
          .release(to)
          .build();
```

```
String URL = "https://the-internet.herokuapp.com/drag_and_drop";
        driver.get(URL);
        driver.manage().window().maximize();
//Actions class method to drag and drop
        Actions builder = new Actions(driver);
        WebElement from = driver.findElement(By.id("column-a"));
        WebElement to = driver.findElement(By.id("column-b"));
//Perform drag and drop
        builder.dragAndDrop(from,to).perform();
```

```
File Upload

public class UploadFile {

    public static void main(String[] args) {

        //https://awesomeqa.com/selenium/upload.html

        ChromeOptions options = new ChromeOptions();
        options.setPageLoadStrategy(PageLoadStrategy.NORMAL);
        WebDriver driver = new ChromeDriver();
        String URL = "https://awesomeqa.com/selenium/upload.html";
        driver.get(URL);
        driver.manage().window().maximize();
        WebElement upload_file =
driver.findElement(By.xpath("//input[@id='fileToUpload']"));

upload_file.sendKeys("/Users/pramod/Documents/Course/apitesting.jpeg");
        driver.findElement(By.name("submit")).click();


    }
}
```

Window:
In any browser, a window is the main webpage to which the user is directed after clicking on a link or URL. Such a window in Selenium is referred to as the "parent window also known as the main window which opens when the Selenium WebDriver session is created and has all the focus of the WebDriver.

```
// Import the necessary modules
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class Main {
    public static void main(String[] args) {


        // Create a new ChromeDriver instance
        WebDriver driver = new ChromeDriver();
```

```
        // Open the page
        driver.get("https://the-internet.herokuapp.com/windows");

        // Store the handle of the current window
        String mainWindowHandle = driver.getWindowHandle();

        // Find the "Click Here" link
        WebElement link = driver.findElement(By.linkText("Click Here"));

        // Click the link to open a new window
        link.click();

        // Store the handles of all open windows in a list
        Set<String> windowHandles = driver.getWindowHandles();

        // Iterate through the list of window handles
        for (String handle : windowHandles) {
            // Switch the focus to each window in turn
            driver.switchTo().window(handle);

            // Check if the text "New Window" is present in the window
            if (driver.getPageSource().contains("New Window")) {
                System.out.println("The text 'New Window' was found in
the new window.");
                break;
            }
        }

        // Switch the focus back to the main window
        driver.switchTo().window(mainWindowHandle);

        // Close the web driver
        driver.quit();
    }
}
```

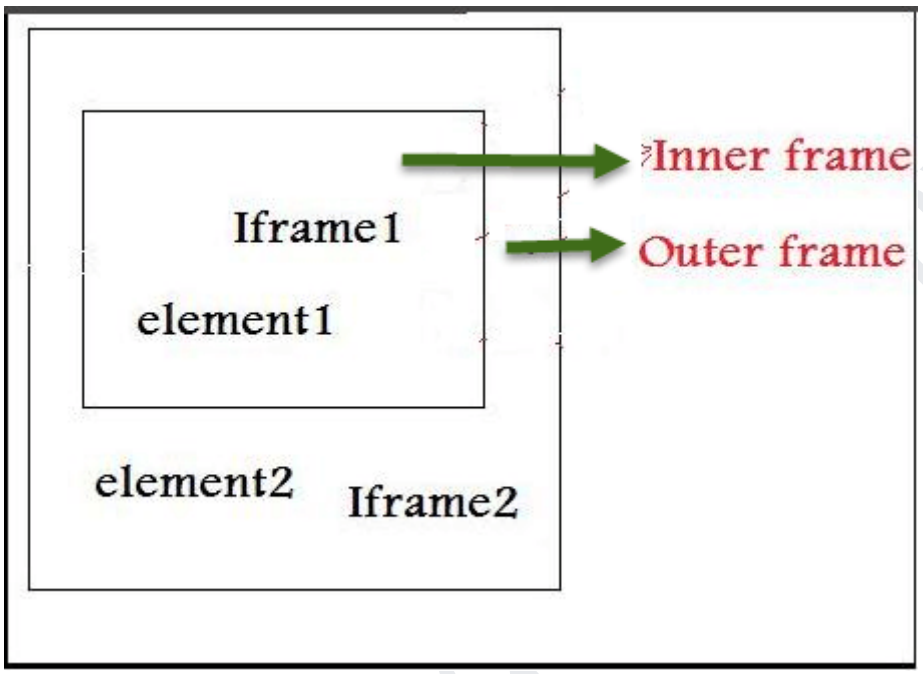
IFRAME

An iframe (short for inline frame) is an HTML element that allows you to embed another HTML document within the current document. Iframes are often used to embed videos, advertisements, or other external content on a webpage.

1.

2. By Index
3. By Name or Id
4. By Web Element

driver.switchTo().defaultContent(); //switching back from the iframe



[Assignment] Open HEATMAP of vwo.com and Click on iframe Click map

1. Open this link with webdriver
2. https://app.vwo.com/#/analyze/osa/13/heatmaps/1?token=eyJhY2NvdW50X2lkIjo2NjY0MDAsImV4cGVyaW1lbnRfaWQiOjEzLCJjcmVhdGVkX29uIjoxNjcxMjA1MDUwLCJ0eXBlIjoiY2FtcGFpZ24iLCJ2ZXJzaW9uIjoxLCJoYXNoIjoiY2IwNzBiYTc5MDM1MDI2N2QxNTM5MTBhZDE1MGU1YTUiLCJzY29wZSI6IiIsImZybiI6ZmFsc2V9&isHttpsOnly=1Use Action to MOVE the mouse to View Heatmap and Click on it.
3. Switch the Window and Switch to iframe
4. Click on button Click Map in the iframe of heatmap.

Solution

```
package com.tta.selenium4demo.WindowsHandles;

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

import java.time.Duration;
```

```
import java.util.Iterator;
import java.util.Set;


public class WindowHandlesAdvance {
    public static void main(String[] args) throws InterruptedException {
        // Create a new ChromeDriver instance
        WebDriver driver = new ChromeDriver();

driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
        driver.manage().window().maximize();
        // Open the page

driver.get("https://app.vwo.com/#/analyze/osa/13/heatmaps/1?token=eyJhY2
NvdW50X2lkIjo2NjY0MDAsImV4cGVyaW1lbnRfaWQiOjEzLCJjcmVhdGVkX29uIjoxNjcxMj
A1MDUwLCJ0eXBlIjoiY2FtcGFpZ24iLCJ2ZXJzaW9uIjoxLCJoYXNoIjoiY2IwNzBiYTc5MD
M1MDI2N2QxNTM5MTBhZDE1MGU1YTUiLCJzY29wZSI6IiIsImZybiI6ZmFsc2V9&isHttpsOn
ly=1");

        // Store the handle of the current window
        String mainWindowHandle = driver.getWindowHandle();

        Actions ac = new Actions(driver);

ac.moveToElement(driver.findElement(By.cssSelector("[data-qa=\"yedexafob
i\"]"))).click().build().perform();


        // Store the handles of all open windows in a list
        Set<String> windowHandles = driver.getWindowHandles();
        System.out.println(windowHandles);

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

        // Here we will check if child window has other child windows
and will fetch the heading of the child window
        while (iterator.hasNext()) {
            String ChildWindow = iterator.next();
            if (!mainWindowHandle.equalsIgnoreCase(ChildWindow)) {
                driver.switchTo().window(ChildWindow);
                driver.switchTo().frame("heatmap-iframe");

driver.findElement(By.cssSelector("[data-qa=\"liqokuxuba\"]")).click();
            }
        }
```

```
        // Close the web driver

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

- String mainwindow = driver.getWindowHandle(): It stores parent window value in a unique identifier of string type.
- Set<String> s = driver.getWindowHandles(): All child windows are stored in a set of strings.
- Iterator<String> i = s.iterator() : Here we will iterate through all child windows.
- if (!MainWindow.equalsIgnoreCase(ChildWindow)) : Now check them by comparing the main window with the child windows.
- driver.switchTo().window (ChildWindow): Switch to the child window and read the heading.

JavaScript executor -

The JavaScript Executor is a feature of the Selenium WebDriver that allows you to execute JavaScript code within the context of the current page.
This can be useful for interacting with elements on the page that are not directly accessible through the Selenium API, or for bypassing certain limitations of the Selenium API.

Here are some common functions that you can use with the JavaScript Executor in Selenium:

- **arguments[0].click()**: This function clicks on the element specified as the first argument.
- **arguments[0].scrollIntoView():** This function scrolls the element specified as the first argument into view.
- **arguments[0].setAttribute(arguments[1], arguments[2]):** This function sets the attribute specified by the second argument to the value specified by the third argument for the element specified as the first argument.

- **arguments[0].innerHTML** = arguments[1]: This function sets the inner HTML of the element specified as the first argument to the value specified by the second argument.
- **return arguments[0].value:** This function returns the value of the element specified as the first argument.
- **return arguments[0].style.display:** This function returns the display style of the element specified as the first argument.

What can you do?

- JavaScriptExecutor provides two methods “executescript” & “executeAsyncScript” to handle.
- Executed the JavaScript using Selenium Webdriver.
- Illustrated how to click on an element through JavaScriptExecutor, if selenium fails to click on element due to some issue.
- Generated the ‘Alert’ window using JavaScriptExecutor.
- Navigated to the different page using JavaScriptExecutor.
- Scrolled down the window using JavaScriptExecutor.
- Fetched URL, title, and domain name using JavaScriptExecutor.

Dynamic Elements

let’s say ‘id’ of a username field is ‘uid_123’

Class=”abc-kkj3k2jk3j2”
id=”web-2323sdsdsd”

[contains(@id,’uid’)]”
/[starts-with(@id,’uid’)]*

Project CRM

1. Login with the Credential
2. Add user https://opensource-demo.orangehrmlive.com/web/index.php/admin/saveSystemUser
3. Search User

Selenium Exception

NoSuchElementException: This exception is thrown when the web driver is unable to locate an element on the page using the specified search criteria.

NoSuchFrameException: This exception is thrown when the web driver is unable to switch to a specified frame.

NoAlertPresentException: This exception is thrown when the web driver is unable to find an alert box on the page.

ElementNotVisibleException: This exception is thrown when the web driver is unable to interact with an element that is not visible on the page. Display : none

ElementNotInteractableException: This exception is thrown when the web driver is unable to interact with an element that is not enabled or not displayed.

StaleElementReferenceException: This exception is thrown when the web driver is unable to interact with an element that has been modified or removed from the DOM after it was located.

TimeoutException: This exception is thrown when the web driver times out while waiting for an element to be located or an action to be performed.

WebDriverException: This is a general exception that is thrown when an error occurs while interacting with the web driver.

- Waits

- Try and Catch
- Throws

Apache POI and Data Driven Testing with Selenium

What is Apache POI?

- Apache POI is a Java library for working with Microsoft Office documents. (after Office Open XML (OOXML)).
- It allows you to read and write Excel, Word, PowerPoint, and other Microsoft Office documents from your Java applications.
- You can use it to create, modify, and save documents, or to extract data from documents for use in your applications.
- XLS and XLSX (can convert to the CSV also).

Commonly used components of Apache POI

- HSSF (Horrible Spreadsheet Format): It is used to read and write xls format of MS-Excel files.
- XSSF (XML Spreadsheet Format): It is used for xlsx file format of MS-Excel.
- POIFS (Poor Obfuscation Implementation File System): This component is the basic factor of all other POI elements. It is used to read different files explicitly.
- HWPF (Horrible Word Processor Format): It is used to read and write doc extension files of MS-Word.
- HSLF (Horrible Slide Layout Format): It is used for read, create, and edit PowerPoint presentations

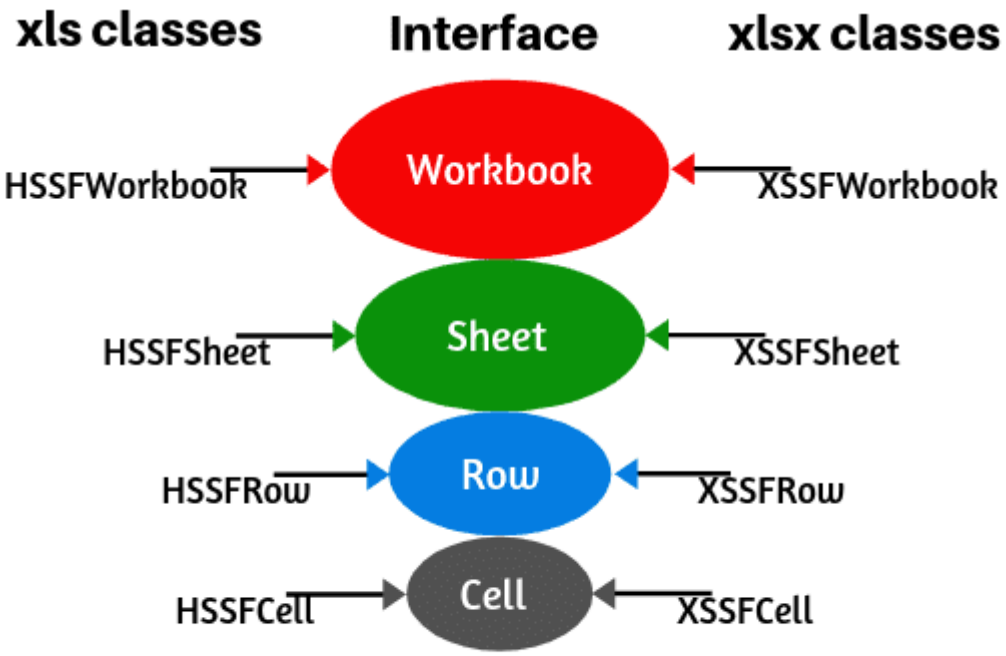
How to add this to the Project?

```
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi</artifactId>
    <version>5.2.3</version>
</dependency>

<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi-ooxml</artifactId>
    <version>5.2.3</version>
</dependency>
```

Reading and Writing Data to Excel File in Java using Apache POI

Flow of the Sheet working.

1. Open Stream
2. Understand Workbook
3. Sheet
4. Row, Column
5. Cells
6. Close Steam



Interfaces and Classes in Apache POI

Task and Assignments

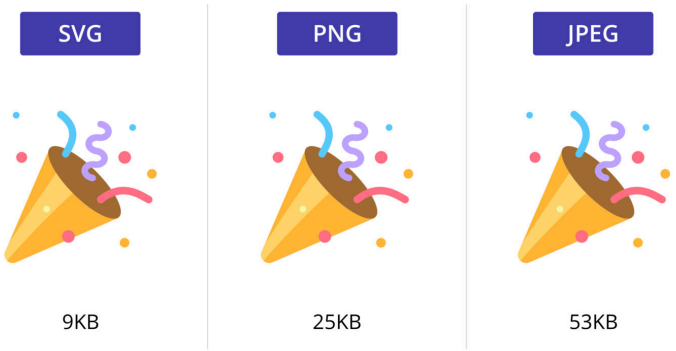
1. Create an Excel File and Add Data.
2. Read the Excel File.
3. Create a cell at a specific position.
4. Formula Excel and Excel Utils

Misc Scenarios in Selenium

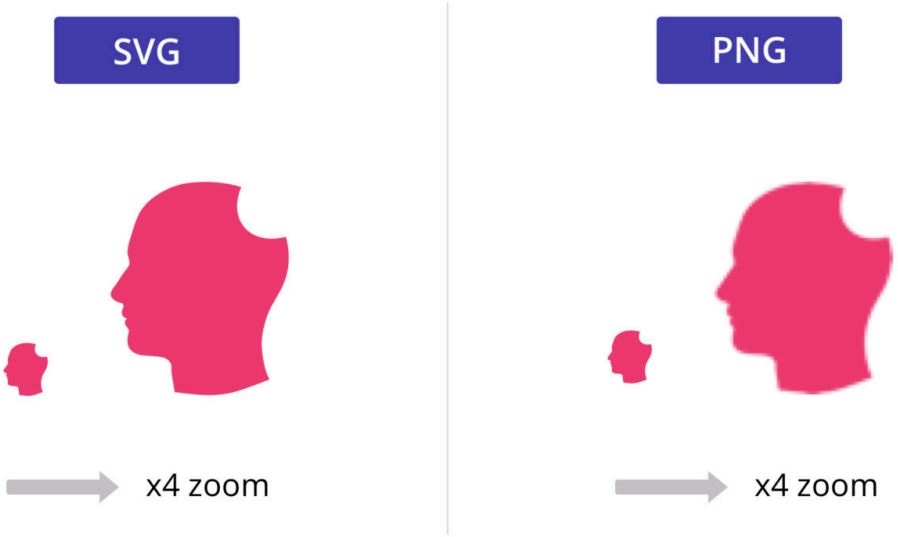
1. Search and Find the text in Web table with pagination.
2. Navigate here - https://codepen.io/templatesio/full/MWqxxog
3. Search “Calvin Golden” on page x.

Handling SVG & Shadow DOM

1. What is SVG - **Scalable Vector Graphics** to define graphics for web.
    a. XML based language to create 2-D graphics/images with animation and interactivity.
    b. Uses geometrical figures to draw an image.



    c. <svg> tag is used as a container for SVG graphics.



2. How to handle SVG Elements in Selenium?
3. How to create XPATH for SVG Elements in HTML DOM?

<svg>
<g>
<circle>
<polygon>

Your First SVG

https://www.w3schools.com/graphics/tryit.asp?filename=trysvg_myfirst

https://www.amcharts.com/svg-maps/?map=india" - Map is SVG

https://flipkart.com - Search button

SVG Automation Problem - Find the Tripura and Click on It

https://www.amcharts.com/svg-maps/?map=india" - Map is SVG

```
driver.get("https://www.amcharts.com/svg-maps/?map=india");

      List<WebElement> states =
driver.findElements(By.xpath("//*[name()='svg']/*[name()='g'][7]/*[name()='g']/*[name()='g']/*[n
ame()='path']"));
      for(WebElement s : states){
         System.out.println(s.getAttribute("aria-label"));
         if(s.getAttribute("aria-label").equals("Tripura  ")){
            actions.moveToElement(s).click().perform();
            break;
         }
      }
```

Shadow DOM

- Shadow DOM is a web standard that allows web components to encapsulate their functionality and styling within a boundary, preventing the styles and behavior of the component from affecting the rest of the page.
- It's a way to create isolated DOM subtrees within a larger DOM.
- When automating web applications with Selenium, you might encounter elements inside Shadow DOMs that are not directly accessible using standard WebDriver locators.
- To interact with elements inside a Shadow DOM, you need to navigate through the Shadow DOM boundary using JavaScript.

```
WebElement link = (WebElement) js.executeScript("return
document.querySelector(\"div.jackPart\").shadowRoot.querySelec
tor(\"div#app2\").shadowRoot.querySelector(\"input#pizza\");")
;
System.out.println(link.getText());
link.sendKeys("Farmhouse");
```

```
  WebElement hostElement = driver.findElement(By.id("shadowHost"));
```

```
        // Execute JavaScript to access the Shadow DOM
        JavascriptExecutor jsExecutor = (JavascriptExecutor) driver;
        WebElement shadowRoot = (WebElement)
jsExecutor.executeScript("return arguments[0].shadowRoot", hostElement);

        // Now, find an element within the Shadow DOM
        WebElement shadowElement =
shadowRoot.findElement(By.cssSelector("#shadowElement"));

        // Perform actions on the shadow element
        shadowElement.click();
```

Relative Locators

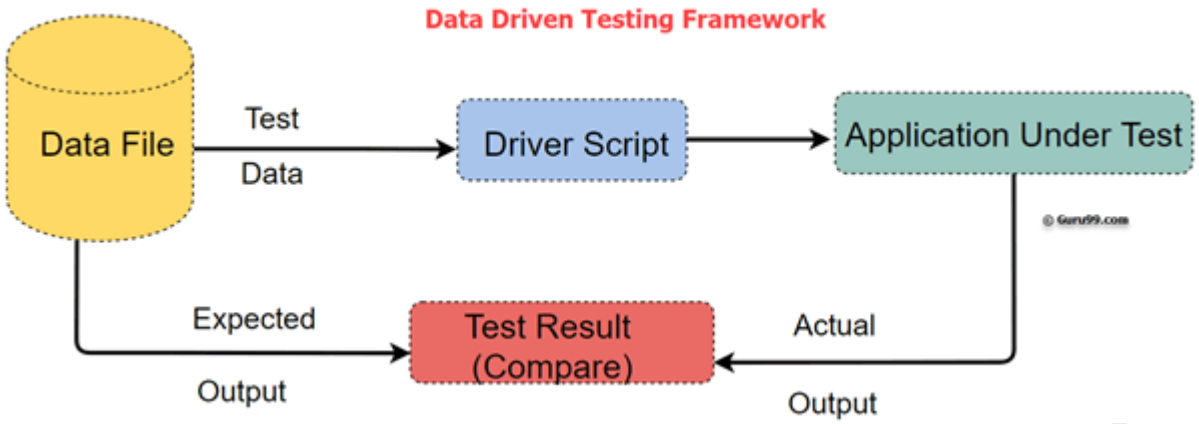
- Relative Locators is a feature introduced in Selenium 4.x that allows you to locate elements relative to other elements on a web page.

- This can be extremely useful when the page layout is dynamic and traditional locators like IDs or XPath are not sufficient.

- Relative Locators help you find elements based on their relationship to other elements, such as being above, below, to the left, or to the right of a reference element.
- In this example, we use the RelativeLocator.with method to specify the base locator (in this case, a tag name "div") and then use the relative methods like .below() and .toRightOf() to find elements relative to the reference element.

Data Driven Testing (Apache POI)

- Test data is stored in table or spreadsheet format.
- In data-driven testing, the input data and expected results are created in a table or spreadsheet.
- Data generation can be done by - https://www.mockaroo.com/
- Run your Test cases based on Data.

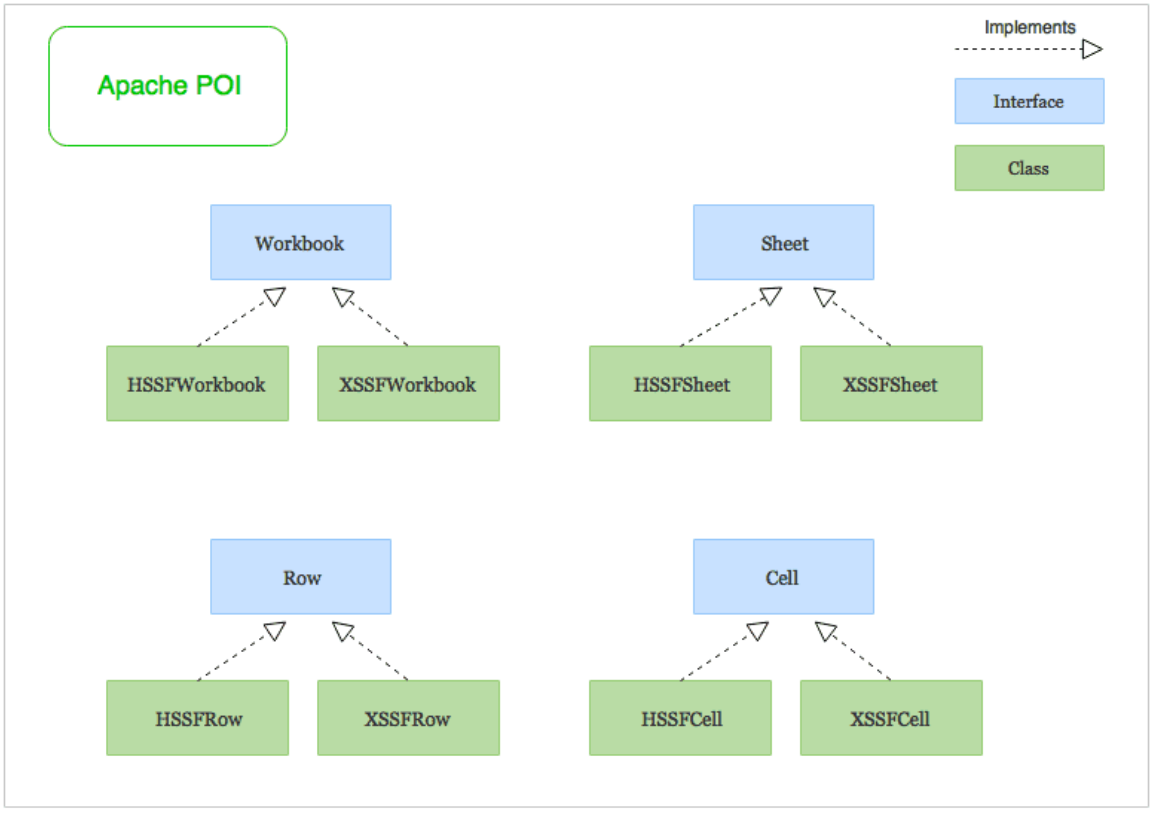


Valid Email	Valid Password	Valid
Invalid Email	Valid Password	Invalid

```
@DataProvider(name = "loginData")
public Object[][] testData() {
   return new Object[][]{
           {"TD1", "93npu2yyb0@esiix.comi",
"Wingify@123", "InValid"},
           {"TD2", "93npu2yyb0@esiix.com",
"Wingify@123", "Valid"},

   };
}
```

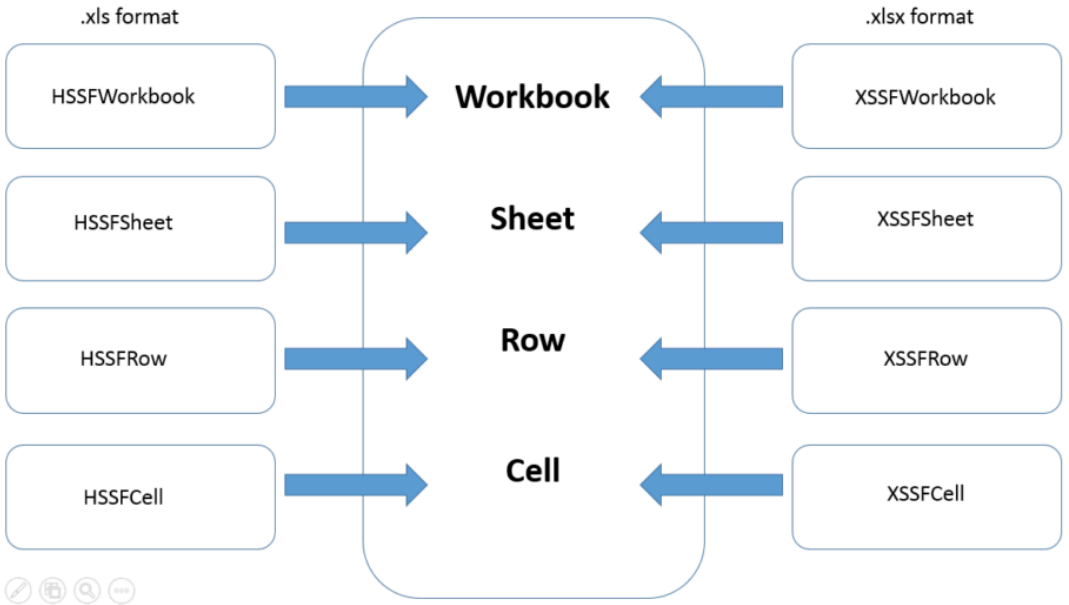
Using Apache POI

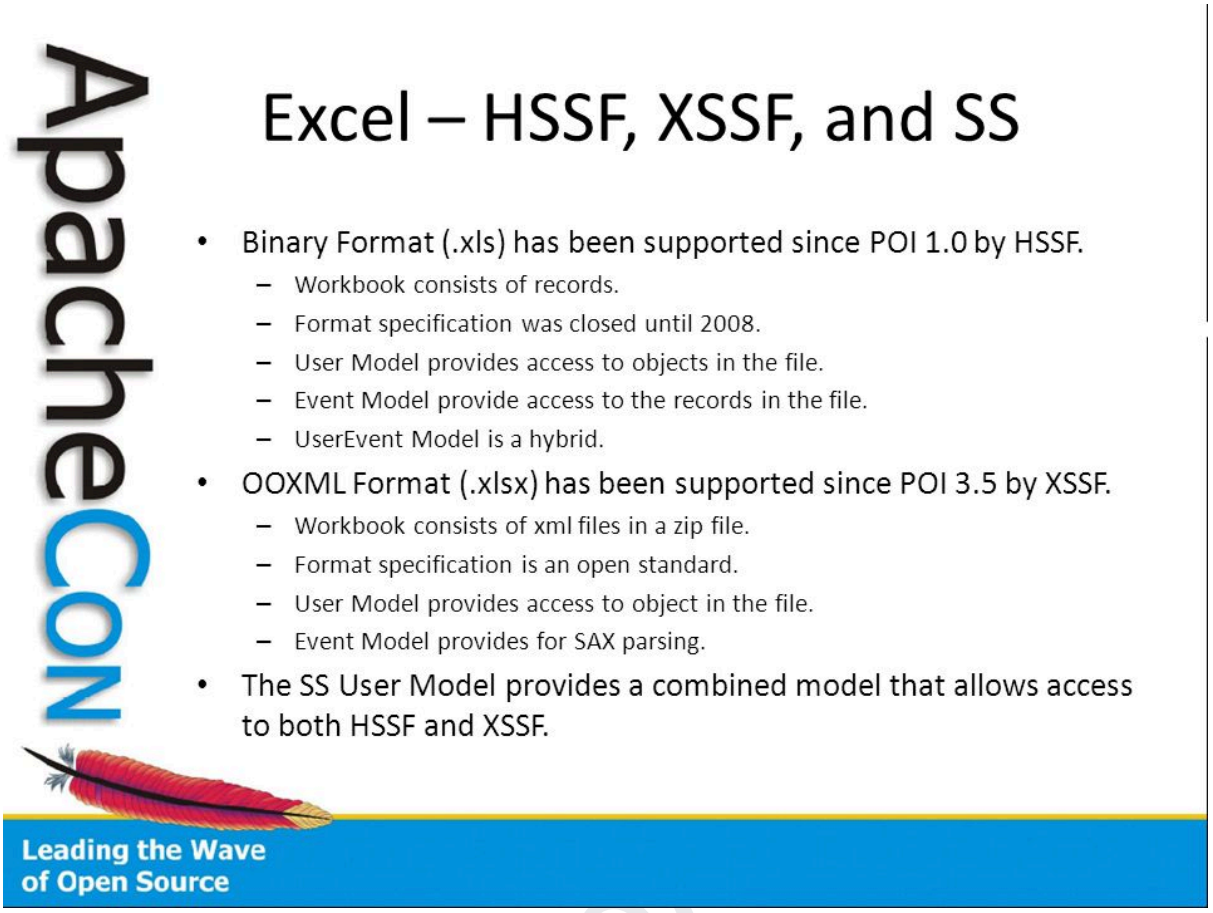

Rows
Columns


Apache POI
Implements
Interface
Class
Workbook
HSSFWorkbook
XSSFWorkbook
Sheet
HSSFSheet
XSSFSheet
Row
HSSFRow
XSSFRow
Cell
HSSFCell
XSSFCell

	HSSF (.xls)		XSSF (.xlsx)		
	eventmodel	usermodel	eventmodel	usermodel	SXSSF
API Type	**Streaming (ala SAX)**	**In memory tree (ala DOM)**	**Streaming (ala SAX)**	**In memory tree (ala DOM)**	**Buffered Streaming**
CPU and Memory Efficiency	Good	Varies	Good	Varies	Good
Forward Only	Yes	No	Yes	No	Yes
Read Files	Yes	Yes	Yes	Yes	No
Write Files	No	Yes	No	Yes	Yes
Feature:					
create sheets / rows / cells	No	Yes	No	Yes	Yes
styling cells	No	Yes	No	Yes	Yes
delete sheets / rows/cells	No	Yes	No	Yes	No
shift rows	No	Yes	No	Yes	No
cloning sheets	No	Yes	No	Yes	No
formula evaluation	No	Yes	No	Yes	No
cell comments	No	Yes	No	Yes	No
pictures	No	Yes	No	Yes	Yes

Important Classes from Apache POI






```
@DataProvider(name = "loginDataExcel")
public String[][] testDataExcel() throws IOException {
   String testDataFile =
"src/test/resources/TataData.xlsx";
   ExcelReader excelReader = new
ExcelReader(testDataFile);
   String [][] data;
   data =
excelReader.getDataFromSheet(testDataFile,"LoginData");
   return data;
}
```

Property Reader

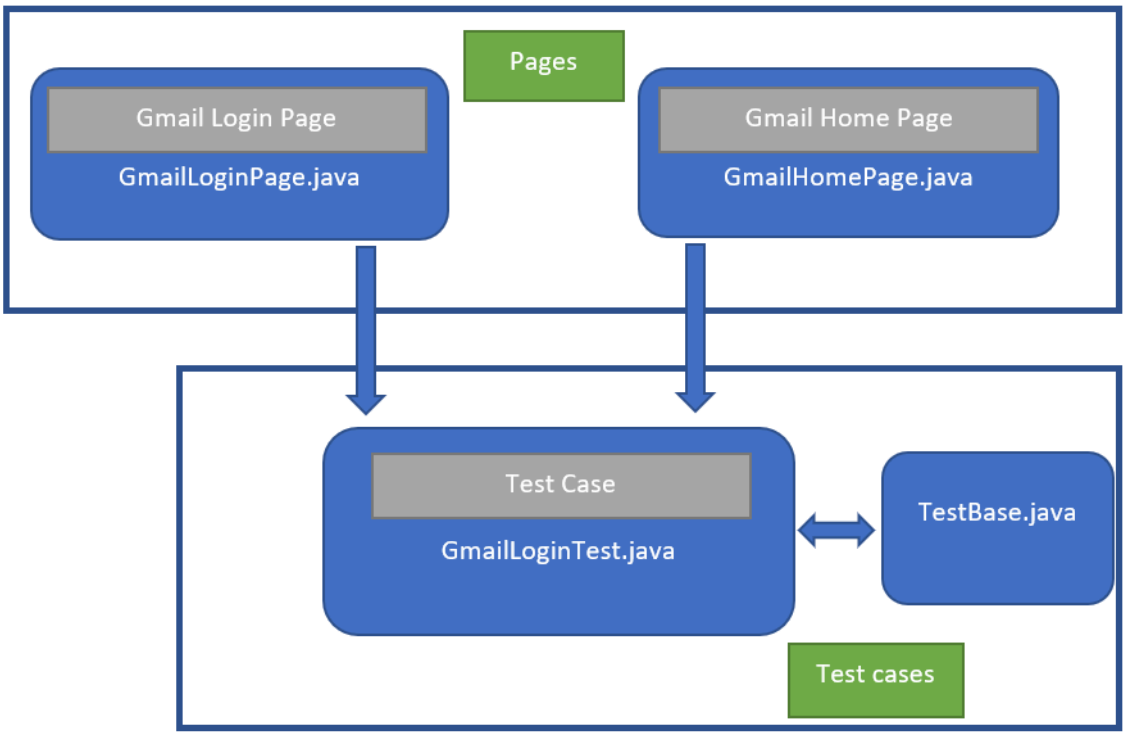
The property file is a file we use in the Java Programming language to keep the configuration parameters.

Create a Propertires file like this

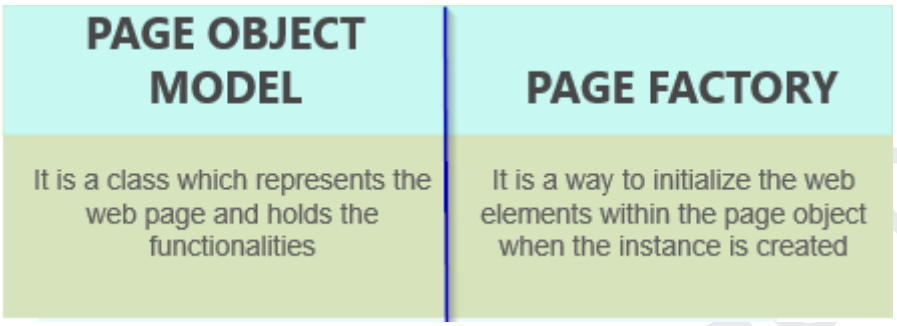
```
url=jdbc:mysql://localhost:3306/DBName
driver=com.mysql.cj.jdbc.Driver //Driver class of mysql
userName=root //Db username
password=root //Db password
```

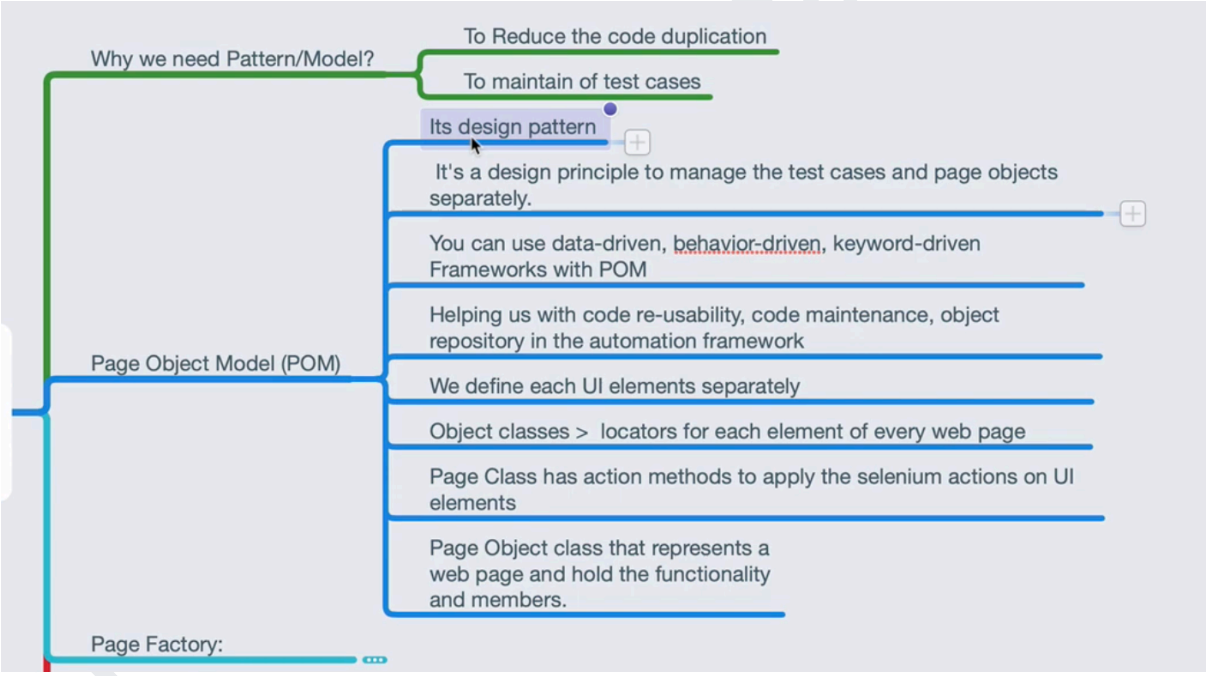
Page Object Model

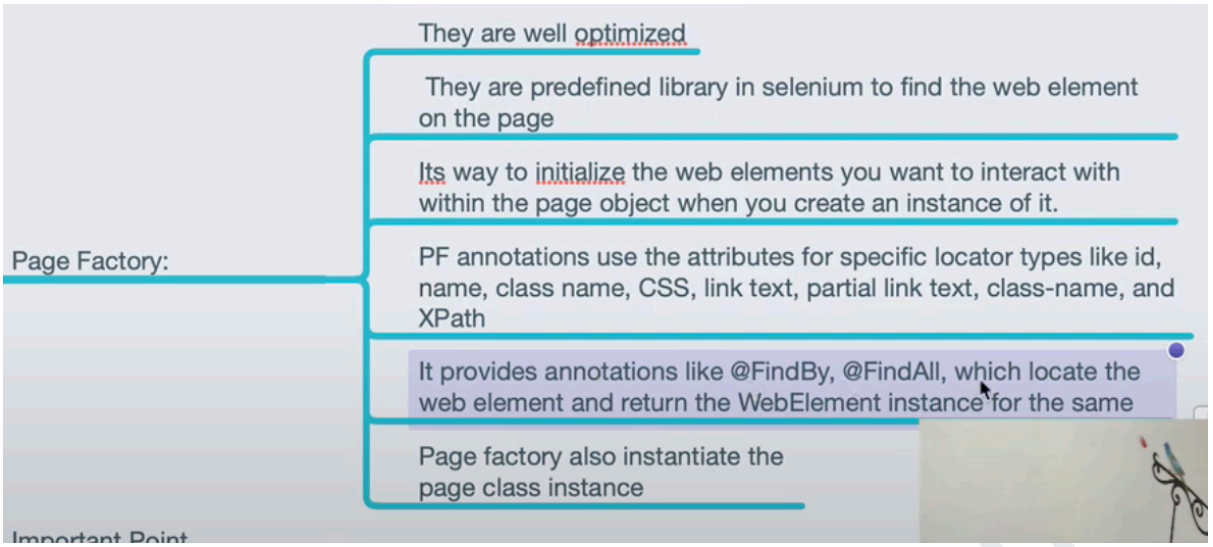
What is a Page Object Model in Selenium?
is a design pattern in Selenium that creates an object repository for storing all web elements.
It helps reduce code duplication and improves test case maintenance.



- Page Object Model in Selenium WebDriver is an Object Repository design pattern.
- Selenium page object model creates our testing code maintainable, reusable.
- Page Factory is an optimized way to create an object repository in the Page Object Model framework concept.
- AjaxElementLocatorFactory is a lazy load concept in Page Factory – page object design pattern to identify WebElements only when they are used in any operation.







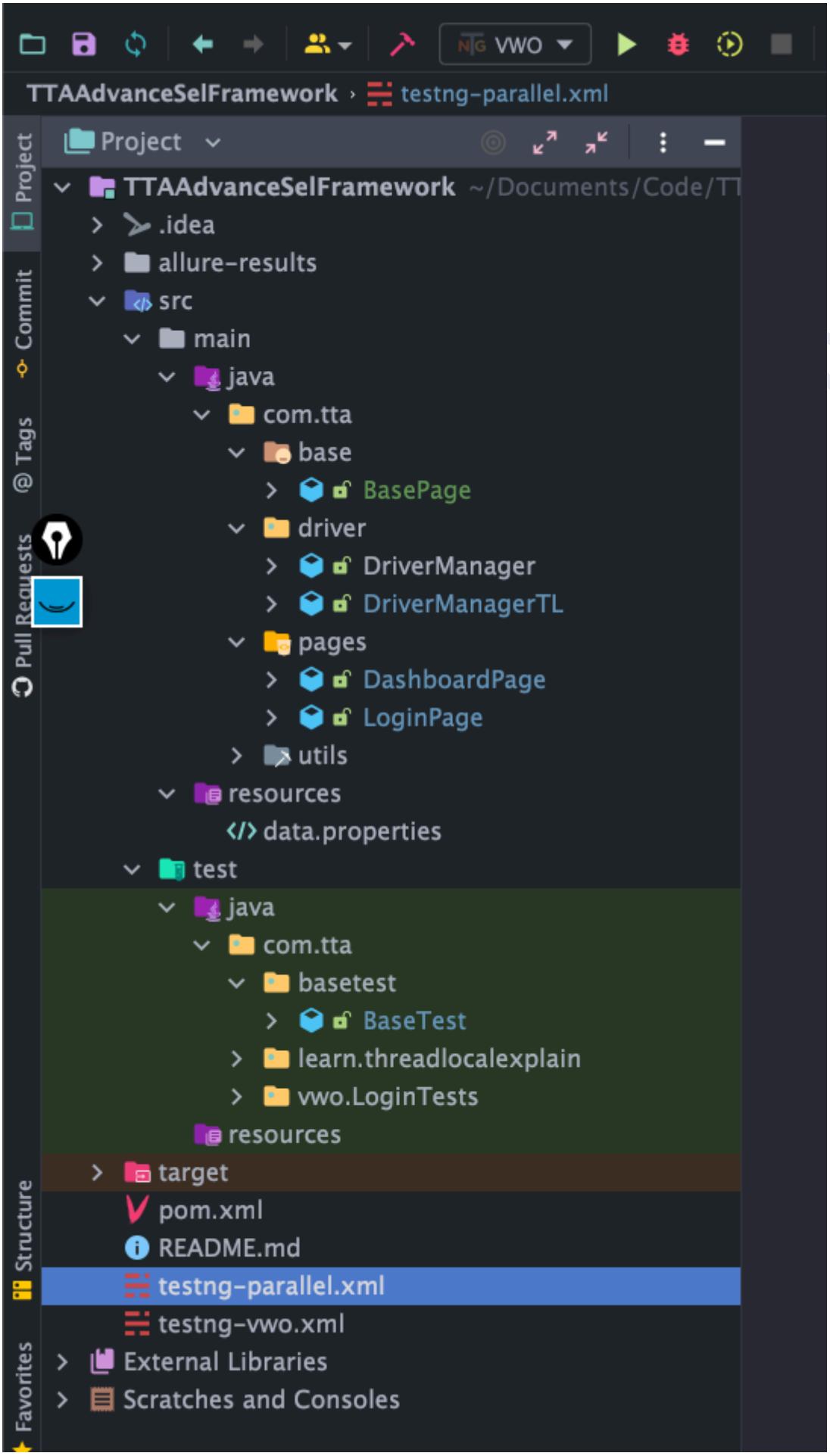
POM vs Page Factory

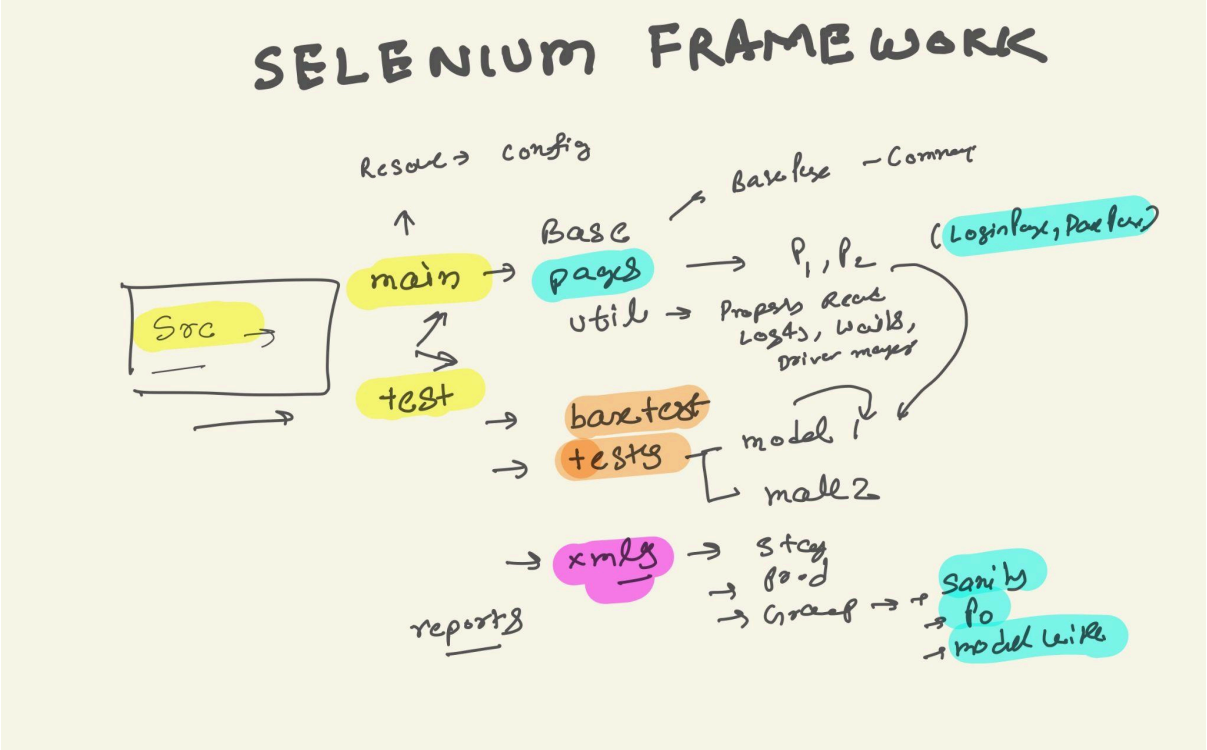
What is the Main difference between Page Object Model and Page Factory in Seleniu…

Selenium Framework

Things in this Framework

- POM Based
- Java, Maven, Selenium, Test NG
- Thread Local Support → Thread Safety
- Allure Report
- Property. / Config Support
- Test NG Listeners
- Cloud Grids
- Parrallel Cross browser testing
- Rest Assured
- Apache POI (Data Driven Testing)
- Sonar Lint, Java > 8

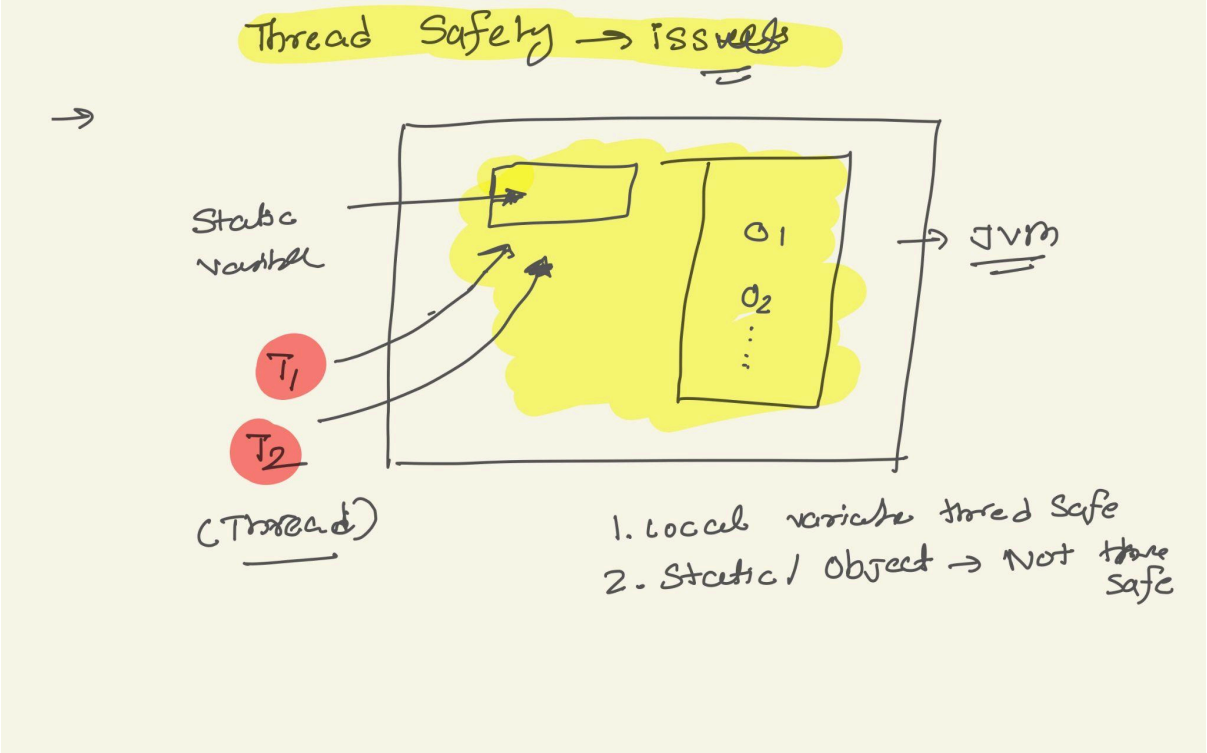

VWO
TTAAdvanceSelFramework › testng-parallel.xml
Project
TTAAdvanceSelFramework ~/Documents/Code/TT
.idea
allure-results
src
main
java
com.tta
base
BasePage
driver
DriverManager
DriverManagerTL
pages
DashboardPage
LoginPage
utils
resources
data.properties
test
java
com.tta
basetest
BaseTest
learn.threadlocalexplain
vwo.LoginTests
resources
target
pom.xml
README.md
testng-parallel.xml
testng-vwo.xml
External Libraries
Scratches and Consoles
Project
Commit
Tags
Pull Requests
Structure
Favorites



Parallel Test

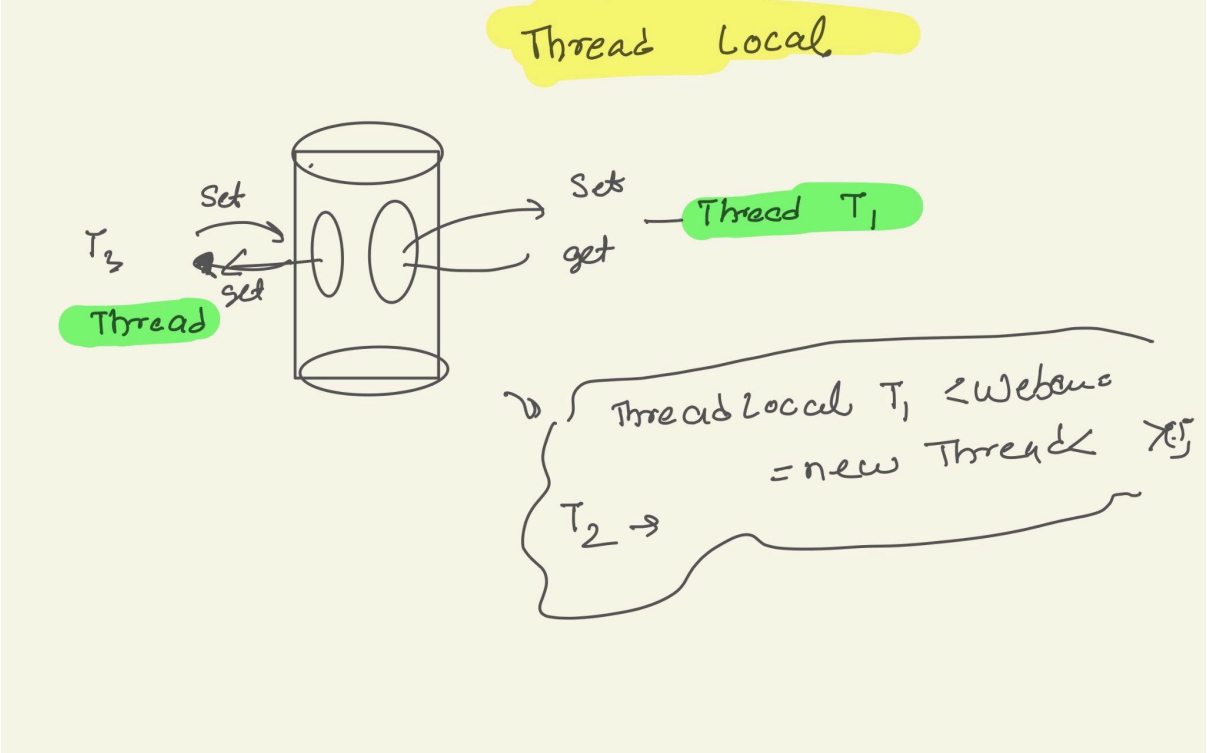
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite thread-count="2" parallel="tests" name="VWO">
   <test name="Firefox">
       <parameter name="BrowserType" value="Firefox"></parameter>
       <classes>
           <class name="com.vwo.tests.tests.loginTest.TestLogin"/>
       </classes>
   </test>
   <test name="Chrome">
       <parameter name="BrowserType" value="Chrome"></parameter>
       <classes>
           <class name="com.vwo.tests.tests.loginTest.TestLogin"/>
       </classes>
   </test>
</suite>
```

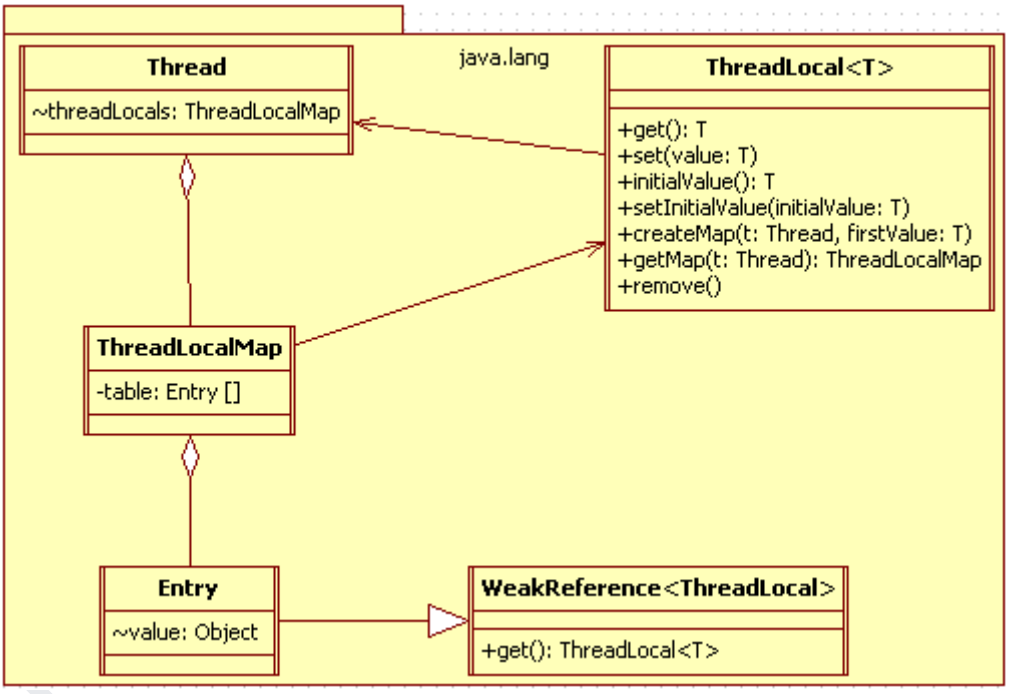
Let's Make our Framework more Thread Safe for Future.



Thread local

- ThreadLocal class in Java allows programmers to create variables that are accessible only to the thread that created them.



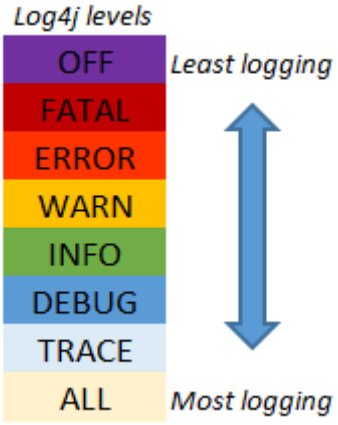


Reference

1. https://www.geeksforgeeks.org/css-selectors-complete-reference/?ref=lbp
2. https://google.com
3.

Log4j

Log4j is a popular logging framework for Java applications that provides a flexible and configurable way to log messages from an application



TRACE: The most detailed level of logging, used to trace the execution flow of an application. This level is typically used for debugging purposes.

DEBUG: A level used to output debugging information about an application. This level is used to log information that can help developers diagnose issues and bugs in an application.

INFO: A level used to output informational messages about an application. This level is used to log important events or milestones that occur during an application's execution.

WARN: A level used to output warning messages about an application. This level is used to log messages that indicate potential issues or errors that may cause problems in the future.

ERROR: A level used to output error messages about an application. This level is used to log messages that indicate errors that have occurred during an application's execution.

FATAL: The most severe level of logging, used to log messages that indicate a critical error that has occurred in an application. This level is typically used when an application is unable to continue executing due to an unrecoverable error.

How to add to Project?

1. Create a log4j2.properties file in resources folder.
2. Add this https://gist.github.com/PramodDutta/e2e085cffdadc68ef7223b5139ed6ff7
3. Use it via

```
static final Logger LOGGER =
LogManager.getLogger(DataProviderDemo.class);
```

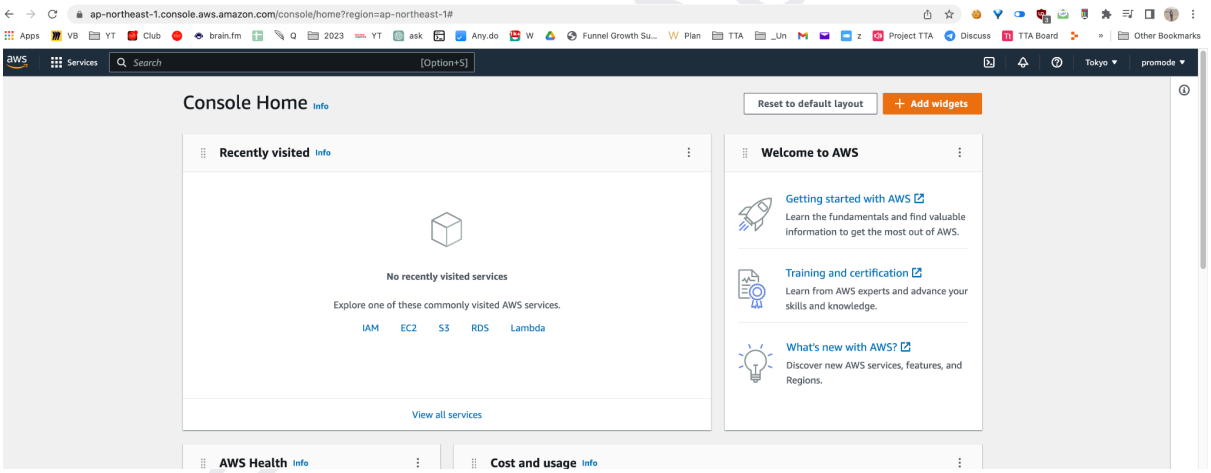
```
// Use it
LOGGER.error("Log4j is Added");
```

Jenkins Run as Free Style Job

AWS Basics

- We will be using AWS for the Jenkins or Selenium Grid Install.
-

1. Create a Free tier account https://aws.amazon.com/free/
2. You will be logged in to the AWS Account



Install Jenkins in AWS

Step - 1 Install Java

Update your system

sudo apt update
Install java
sudo apt install openjdk-11-jre -y
Validate Installation

java -version
It should look something like this

openjdk version "11.0.12" 2021-07-20 OpenJDK Runtime Environment (build 11.0.12+7-post-Debian-2) OpenJDK 64-Bit Server VM (build 11.0.12+7-post-Debian-2, mixed mode, sharing)
Step - 2 Install Jenkins

Just copy these commands and paste them onto your terminal.

**curl -fsSL https://pkg.jenkins.io/debian/jenkins.io.key | sudo tee **
/usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \ https://pkg.jenkins.io/debian binary/ | sudo tee \ /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins

Step -3 Start jenkins

sudo systemctl enable jenkins
sudo systemctl start jenkins
sudo systemctl status jenkins
Step - 4 Open port 8080 from AWS Console:

Edit for Port - /etc/default/jenkins

How to find Java Location?

readlink -f $(which java)

Running Selenium Test cases on Selenium Grid

1. Install Selenoid
2. Install Docker

Sudo su // FOR SU user
Apt-get update
apt install docker.io -y
sudo systemctl status docker
sudo systemctl start docker

sudo wget "https://github.com/aerokube/cm/releases/download/1.8.1/cm_linux_amd64"

sudo chmod +x cm_linux_amd64
sudo ./cm_linux_amd64 selenoid start –vnc

./cm_linux_amd64 selenoid-ui start
./cm_linux_amd64 selenoid-ui stop