INTRODUCTION TO AUTOMATION TESTING

What is Automation Testing?

Automation testing or Test Automation is the process of running tests automatically, managing test data, and utilizing results to improve software quality.

It is the process of converting manual test cases into automation test scripts using automation testing tools and with the help of programming or scripting language.

Advantages of Automation Testing:

1. Faster feedback cycle
2. Reduce business expenses
3. Higher test coverage
4. Reusability of test suite
5. Improved accuracy
6. Quickly determine the stability of the build
7. Eliminates human error
8. Earlier detection of error
9. Data driven testing
10. Maximize ROI (Return On Investment)

Note: ROI is a metric that provides a numerical representation of the return you derive by incorporating automation testing into your QA process.

Disadvantages of Automation Testing:

1. Complexity: Automated tests can take longer to develop than manual tests
2. High initial costs
3. It needs to be rewritten for every new environment
4. Generates false positives and negatives
5. Cannot be used on GUI elements (e.g., graphics, sound files)
6. Difficult to design tests that are both reliable and maintainable

When to start with Automation Testing?

- When the build reaches certain level of functional stability
- When the resources (Manual test cases, automation tools, etc) are ready
- When the testing has more repetitive tasks
- When the testing has more regression cycles
- When you need to run multiple tests at once
- When we have long term and complex project

Types of Automation Testing Tools:

1. Functional Automation testing Tools
2. Non-Functional Automation Testing Tools

Functional Automation Testing Tools: The tools that are used to automate functional, integration, system and regression test cases are called Functional Automation Testing Tools.

Ex: Selenium, UFT(Unit Functional Test), SoapUI, QTP(Quick Test Professional), RFT(Rational Functional Tester), Winium, Appium, Test Complete etc

Non-Functional Automation Testing Tools: The tools that are used to automate non-functional test cases such as performance, security, usability, compatibility testing are called as Non-Functional Automation Testing Tools.

Ex: Apache JMeter, Neo Load, App Load, Web Load, Load Runner, Webserver Stress Tool, vPerformer, Forecast etc.

INTRODUCTION TO SELENIUM

Selenium is an open-source functional automation testing tool that enables and supports the automation of web browsers.

Selenium is not just a tool or an API, it is a suite comprising of several components.

Advantages of Selenium:

- It is an open-source tool, it is available for free.
- It is an open-source tool, so integrating selenium with any third-party tool is easy.
- Selenium is developed using java, like java it is platform independent.
- Selenium supports multiple programming languages like java, python, C-Sharp, Ruby, Perl etc.
- Selenium supports multiple browsers like chrome, firefox, edge, opera, safari etc. Browser compatibility testing is easy using selenium.
- Selenium supports multiple platforms such as Microsoft Windows, macOS and Linux. System compatibility testing is easy using selenium.

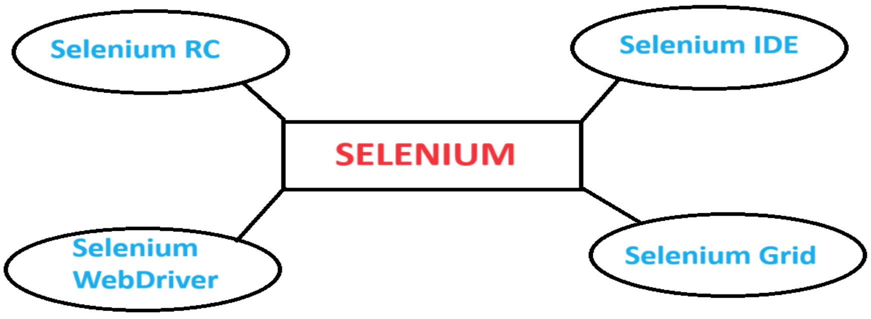
Drawbacks of Selenium:

- Selenium tests web applications only.
- Requires knowledge of programming language.
- No built-in reporting and test management facility, it has to be integrated with tools like TestNG to facilitate test management and reporting.

Flavours of Selenium/ Selenium Suite Components:

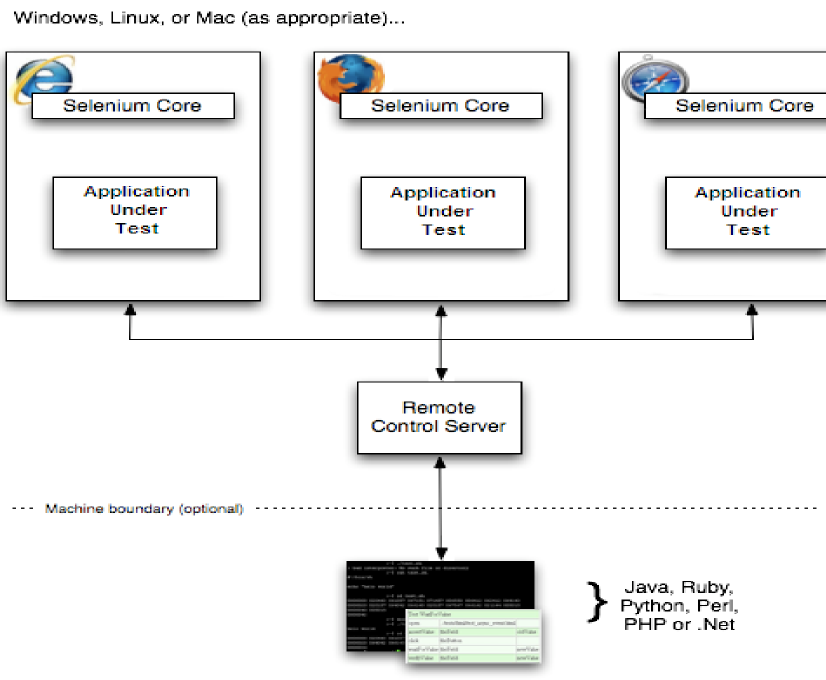
Selenium suite has four major components.

1. Selenium RC
2. Selenium IDE
3. Selenium WebDriver
4. Selenium Grid



Selenium RC (Remote Control):

- It is a testing framework that enables test engineer to write the test scripts in any programming language in order to automate UI tests for web applications against any HTTP website
- Selenium RC comprises of two parts:
  1. Client libraries
  2. Remote Control Server



- RC Server acts as a mediator between the browser and the client.
- When a test script is executed, RC Server injects Javascript commands known as Selenium Core into the browser.
- Once the Selenium Core program is injected, it receives instructions from RC Server and executes these instructions as Javascript commands.
- The web browser executes all the commands given by Selenium Core and returns the test summary to the server.

Limitations of Selenium RC:

- Complicated Architecture
- Execution of test scripts is time consuming as Selenium RC uses Javascript commands
- No support for HTTPS protocol

Selenium IDE (Integrated Development environment):

- It is an open-source web automation testing tool under the Selenium Suite.
- It is basically a plugin for Firefox, Chrome and Edge browsers.
- It does not require any programming logic to write test scripts rather you can simply record your interactions with the browser to create test cases.
- Using the playback option we can re-run the test cases.

- It uses Selenese commands to execute the test scripts.
- The recorded test scripts can also be exported to programming languages like C#, Java, Ruby or Python.
- It is generally used in debugging the test scripts and for quick bug reproductive tasks.
- It is also used in automation aided exploratory testing.

Limitations of Selenium IDE:

- Not suitable for testing extensive data
- Cannot handle the dynamic part of web-based applications
- Does not support capturing of screenshots on test failures
- No feature available for generating result reports

Selenium WebDriver:

- Selenium WebDriver is an enhanced version of Selenium RC.
- It is an API, commonly referred to as WebDriver which drives a browser natively either locally or on a remote machine using the Selenium server.
- It is a collection of language specific bindings to drive a browser.
- It is a web framework which is used for testing web-based applications.
- It is used to create robust, browser-based regression automation suites and tests.
- It also allows you to perform cross browser compatibility testing.
- It supports multiple programming languages, browsers and platforms.
- Selenium WebDriver completes the execution of test scripts faster when compared to other tools.
- More Concise API (Application Programming interface) than Selenium RC.

Limitations of Selenium WebDriver:

- Support for new browsers is not readily available when compared to Selenium RC.
- For the automatic generation of test results, it doesn’t have a built-in command.

Selenium Grid:

- It is the collection of Selenium WebDriver, Selenium RC and Grid Server libraries.
- It allows the execution of WebDriver scripts on remote machines by routing commands sent by the client to remote browser instances.
- It is used for Remote Executions (Execution of test scripts on remote devices like devices on same network or cloud servers).
- It is used for concurrent execution of test cases on different browsers, machines and operating systems simultaneously.
- This tool makes cross-browser and cross-platform compatibility testing very easy.

- It reduces the time needed to execute a test suite.

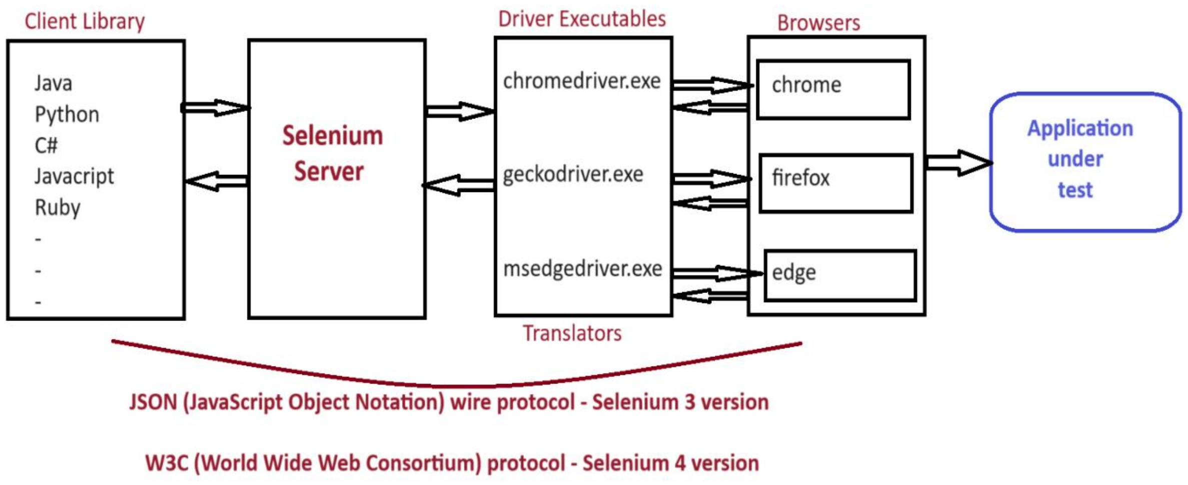
Besides the above major components, selenium community has:

- **For mobile based applications:**
    1. **Selendroid:** used to test mobile applications for Android.
    2. **Appium:** used to test mobile applications for both Android and iOS.
- **Winium:** for testing and automating desktop applications on Windows.

SELENIUM ARCHITECTURE

Selenium architecture contains four major components:

- **Client Library:** This component provides language-specific bindings or APIs (Java, Python, Ruby, etc.) that allow users to write test scripts and interact with the WebDriver.
- **Selenium Server:** This component receives request from the client (user written test script) and communicates with specified driver executables.
- **Driver Executables:** This component acts as a translator between Selenium server and actual browser. They establish a communication channel between the WebDriver and the actual web browsers.
- **Browser:** It is the place where actual test execution takes place. The WebDriver interacts with these browsers through their respective browser drivers to perform actions.



- ✓ The communication between Client, Selenium server, Driver executables and Browsers happens via **JSON wire protocol in Selenium 3 version** and **W3C protocol in Selenium 4 version**.
- ✓ JSON (JavaScript Object Notation) wire protocol provides a transport mechanism for transferring data between client, server and browsers. It defines a set of commands and responses in JSON format exchanged over HTTP requests.
- ✓ W3C (World Wide Web Consortium) protocol provides a standard way to make the communication easy and direct between the client libraries and the browser drivers. Improved communication led to more stability in Selenium 4 version.

Selenium WebDriver Installation Steps

Selenium Server Download:

- Open the browser and type selenium.dev as URL
- Go to Downloads
- Scroll till “Selenium Grid” and click on latest stable version of selenium (currently 4.21.0)

Chromedriver Executable Download:

- First check the browser version available in your computer
  - Open chrome browser
  - Click on the ⋮ on top right corner of the browser
  - Go to “Help” and click on “About Google Chrome”
  - Note the version
- Open the browser and search for chromedriver download
- Click on Downloads | ChromeDriver - Chrome for Developers link
- Click on the Chrome for Testing availability dashboard link
- Click on “Stable” and copy the respective chromedriver link based on the OS of your computer
- Paste the copied link in the new tab and hit on Enter button in keyboard

Java Project Creation:

- Open the Eclipse IDE
- Click on “File” and then click on “New” and then click on “Java Project”
- Name the project and deselect “create module-info.java” checkbox
- Click on “Finish”
- Create “softwares” folder and “drivers” folder in the project
  (Right click on the project => New => Folder => name it => click “Ok”)

Adding Selenium Server to the Java Project:

- Go to Downloads folder in the File explorer
- Copy the downloaded Selenium Server jar file
- Paste it on “softwares” folder in the Java project in Eclipse IDE
- Right click on the pasted jar file, go to “Build path” and click on “Add to build path”
  (selenium server jar file will be added to “Referenced Libraries” in the project)

Adding chromedriver.exe file to the Java Project:

- Go to Downloads folder in the File Explorer
- Extract chromedriver.zip file
- Copy chromedriver.exe file from the extracted folder
- Paste it on “drivers” folder in the Java project in Eclipse IDE
  (**Note:** If you can’t paste it directly on the “drivers” folder, right click on the “drivers” folder => go to “Properties” => click on ⇨ => open “drivers” folder and paste it and refresh the project in Eclipse IDE)

Test Scripts are written in the packages created in the “src” folder in the project.

Script to Launch the Browser in Selenium WebDriver

- **Launching chrome browser**

```
import org.openqa.selenium.chrome.ChromeDriver;
```

```
public class LaunchChromeBrowser {

          public static void main(String[] args) {

                    ChromeDriver driver = new ChromeDriver();
          }
}
```

- **Launching firefox browser**

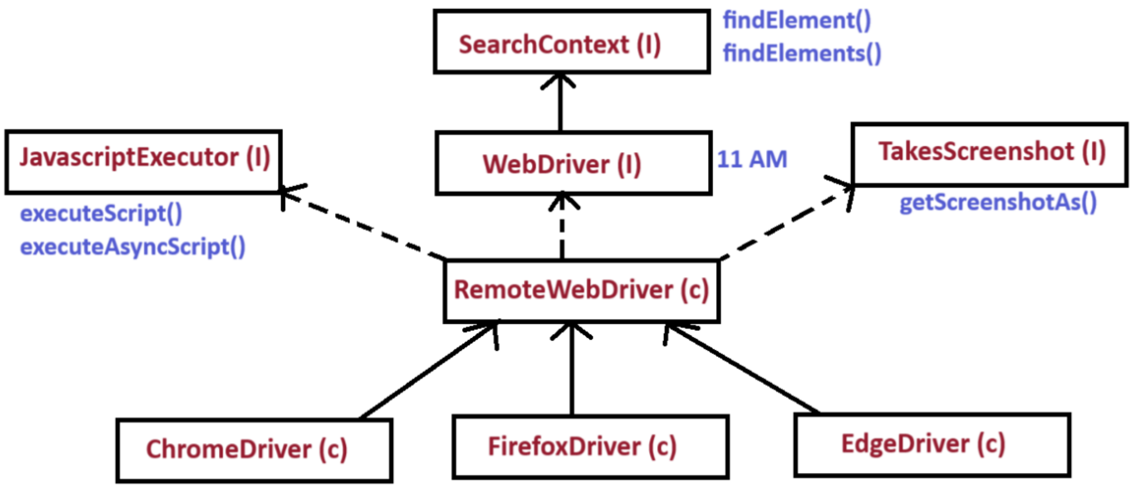
```
import org.openqa.selenium.chrome.FirefoxDriver;

public class LaunchChromeBrowser {

          public static void main(String[] args) {

                    FirefoxDriver driver = new FirefoxDriver ();
          }
}
```

- **Launching edge browser**

```
import org.openqa.selenium.chrome.EdgeDriver;

public class LaunchChromeBrowser {

          public static void main(String[] args) {

                    EdgeDriver driver = new EdgeDriver ();
          }
}
```

To launch a web browser, one should just create an instance of respective browser specific class.

SELENIUM WEBDRIVER CLASS DIAGRAM / JAVA SELENIUM WEBDRIVER ARCHITECTURE



Selenium WebDriver class diagram is a hierarchy of interfaces and classes of Selenium WebDriver API.

SearchContext: It is the super most interface in Selenium WebDriver class diagram. It works like a search engine to find and fetch the elements from the web page. It has two abstract methods –

- findElement()
- findElements()

WebDriver: It is sub-interface of SearchContext interface. It provides driver related input actions to the web browser like launching and closing the browser, navigate to web application, maximize or minimize browser etc. It has eleven abstract methods and two abstract methods are inherited from SearchContext interface. Altogether it has thirteen abstract methods.

- get()
- getTitle()
- getCurrentUrl()
- getPageSource()
- getWindowHandle()
- getWindowHandles()
- manage()
- navigate()
- switchTo()
- close()
- quit()

RemoteWebDriver: It is an implementing class of WebDriver interface. It also implements two other interfaces –

- JavascriptExecutor interface
- TakesScreenshot interface

JavascriptExecutor: It is an interface in Selenium WebDriver which executes Javascript code. It has two abstract methods –

- executeScript()
- executeAsyncScript()

TakesScreenshot: It is an interface in Selenium WebDriver which captures the screenshot of a web page. It has one abstract method –

- getScreenshotAs()

All the browser specific classes like ChromeDriver, FirefoxDriver, EdgeDriver etc., extend to RemoteWebDriver class.

We can launch the browser using the below statement:

WebDriver driver = new ChromeDriver();

- With respect to selenium, the above statement launches empty chrome browser.
- With respect to java, the above statement is an upcasting statement and an example for Run Time Polymorphism.

- We create an instance of ChromeDriver class and upcast it to WebDriver interface so that we can launch different browsers using same driver reference.

```
public void launchBrowser(String browser) {
    switch (browser) {
            case "chrome":
                    driver = new ChromeDriver();
                    break;
            case "firefox":
                    driver = new FirefoxDriver();
                    break;
            case "edge":
                    driver = new EdgeDriver();
                    break;
            default:
                    System.out.println("Invalid browser info");
            }
    }
```

- Here based on the object created respective browser is launched during run time. Hence it is an example for Run Time Polymorphism.

WebDriver Methods

WebDriver is an interface having eleven abstract methods

1. **get():**
   - It is used to navigate to a web application and waits until the web page is loaded.
   - It accepts URL in **String** datatype as argument.
   - Return type is **void**.
   - Usage –

     driver.get("<url>");

2. **getTitle():**
   - This method fetches the title of the current web page.
   - It does not accept arguments.
   - Return type is **String**.
   - Usage –

     String title = driver.getTitle();

3. **getCurrentUrl():**
   - This method fetches the URL of the current web page.
   - It does not accept any arguments.
   - Return type is **String**.
   - Usage –

     String url = driver.getCurrentUrl();

4. **getPageSource():**
   - This method fetches the source code of the current web page.
   - It does not accept any arguments.
   - Return type is **String**.
   - Usage –

String pageSource = driver.getPageSource();

5. **close():**
   - This method closes the current tab or window and exits webdriver process.
   - It does not stop the server.
   - It does not accept any arguments.
   - Return type is **void**.
   - Usage –

     driver.close();

6. **quit():**
   - This method closes all the tabs and windows and exits webdriver process.
   - It stops the server.
   - It does not accept any arguments.
   - Return type is **void**.
   - Usage –

     driver.quit();

7. **manage():**
   - This method manages all browser settings like
     - Window related operations
     - Timeouts related operations
     - Cookies related operations
   - It does not accept any arguments.
   - Return type is **Options** (static interface in WebDriver).
   - **Window related operations:**
     - To maximize the window:

       driver.manage().window().maximize();
     - To minimize the window:

       driver.manage().window().minimize();
     - To view the web page in full screen:

       driver.manage().window().fullscreen();
     - To set the size of the window:

       Dimension dimension = new Dimension(<width>, <height>);
       driver.manage().window().setSize(dimension);
     - To set the position of the window:

       Point point = new Point(<x>, <y>);
       driver.manage().window().setPosition(point);
     - To get the size of the window:

       Dimension dimension = driver.manage().window().getSize();
       int height = dimension.getHeight();
       int width = dimension.getWidth();
     - To get the position of the window:

       Point point = driver.manage().window().getPosition();
       int x = point.getX();
       int y = point.getY();
   - **Cookies related operations:**

     Cookies are small pieces of text sent to your browser by a website you visit. They help that website remember information about your visit.
     - To add a cookie:

       driver.manage().addCookie(Cookie cookie);

- ➢ To delete a cookie:
  driver.manage().deleteCookie(Cookie cookie);
- ➢ To delete all cookies:
  driver.manage().deleteAllCookies();
- ➢ To get all cookies:
  Set<cookie> cookies = driver.manage().getCookies();

8. **navigate():**
   - This method is used to perform all browser navigations.
   - It does not accept any arguments.
   - Return type is **Navigation** (static interface).
   - **Different browser navigations:**
     - ➢ To navigate to before page (←)
       driver.navigate().back();
     - ➢ To navigate to next page (→)
       driver.navigate().forward();
     - ➢ To refresh a web page (↻)
       driver.navigate().refresh();
     - ➢ To navigate to an application:
       driver.navigate().to("<url>");

9. **findElement():**
   - This method is used to fetch the first matching element from the web page.
   - It accepts locator as an argument.
   - Return type is WebElement (I).
   - If the element is not found, it throws NoSuchElementException.
   - **Usage:**

     WebElement element = driver.findElement(<locator>);

10. **findElements():**
    - This method is used to fetch all the matching elements from the web page.
    - It accepts locator as an argument.
    - Return type is List<WebElement>.
    - If the elements are not found, it returns empty array list.
    - **Usage:**

      List<WebElement> element = driver.findElements(<locator>);

Differences between get() and navigate():

get()	navigate()
1. It navigates to an application and waits until web page is loaded. 2. Return type is void 3. It accepts url in String format as argument 4. It does not store browsing history	1. It just navigates to an application. 2. Return type is Navigation (I) 3. It does not accept any arguments 4. It stores browsing history which is why we can perform all browser navigations

Differences between close() and quit():

close()	quit()
1. This method is used to close the current tab or window 2. It does not terminate the browser instance 3. The WebDriver session remains active	1. This method is used to close all tabs and windows 2. It terminates the browser instance 3. It ends the WebDriver session

Differences between findElement() and findElements():

findElement()	findElements()
1. It fetches the first matching element from the web page. 2. It accepts locator as an argument. 3. Return type is WebElement (I). 4. If the element is not found, it throws NoSuchElementException	1. It fetches all the matching elements from the web page. 2. It accepts locator as an argument. 3. Return type is List<WebElement>. 4. If the elements are not found, it returns empty array list.

HTML (Hyper Text Markup Language)

HTML is the standard markup language for creating Web pages. It describes the structure of a web page. It contains three elements:

1. Tags
2. Attributes
3. Text

HTML elements tell the browser how to display the content.

1. **Tags:** The first word enclosed within angular brackets (<>) is a tag.
   Ex: <html> → first tag to start with web page design
     <a> → for links
     <h1>, <h2>, <h3>, <h4>, <h5>, <h6> → for page headers
     <select> → for drop downs
     <input> → for text fields
     <button> → for buttons
     <table> → for tables
     <img> → for images

2. **Attributes:** The name-value pairs enclosed within angular brackets (<>) are called as attributes.
   Ex:
     <input **id="username" name="email"** /> → id and name are the attributes

     <img **src="google.jpg" alt="Google" width="104" height="142"**>
     Here src, alt, width and height are the attributes

3. **Text:** Anything which is not enclosed within angular brackets (<>) is called text.
   Ex:
     <a href="https://mail.google.com/mail/&amp;ogbl" target="_top">**Gmail**</a>

LOCATORS

- Locators are the way to identify an *HTML* element on a web page.
- Locators are static methods of **By** class.
- **By** class is an abstract class in org.openqa.selenium package.

Why Locators?

To act on any element on the web page we should first locate the it. Locators help us to find the element on the web page.

Types of Locators:

1) id
2) name
3) className
4) tagName
5) linkText
6) partialLinkText
7) cssSelector

8) xpath

1. **id:** An ‘id’ locator uses id attribute to identify an element in the web page.
   Usage:

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

2. **name:** A ‘name’ locator uses name attribute to identify an element in the web page.
   Usage:

   WebElement element = driver.findElement(By.name(“<name_attribute_value>”));

3. **className:** A ‘className’ locator uses class attribute to identify an element in the web page.
   Usage:

   WebElement element = driver.findElement(By.className(“<class_attribute_value>”));

4. **tagName:** We can also locate an element using the tag name of the respective element node. Often there can be multiple elements having same tag name.
   Usage:

   WebElement element = driver.findElement(By.tagName(“<tag_name>”));
   Or
   List<WebElement> elements = driver.findElements(By.tagName(“<tag_name>”));
5. **linkText:** A ‘linkText’ locator uses the text on the hyperlinks to identify an element on the web page.
   Usage:

   WebElement element = driver.findElement(By.linkText(“<text_on _the_hyperlink>”));

6. **partialLinkText:** A ‘partialLinkText’ locator uses partial/part of the text present on the hyperlinks to identify an element on the web page.
   Usage:

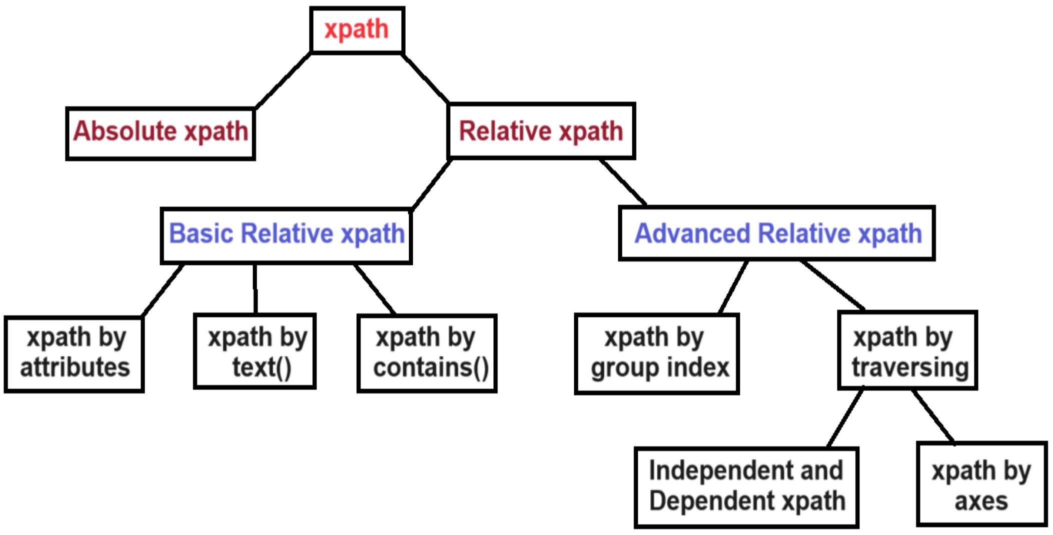
   WebElement element = driver.findElement(By.partialLinkText(“<partial_text>”));

7. **cssSelector:** CSS (Cascading Style Sheet) is used to style the web elements.
   CSS Selector is used to find or locate an element on the web page. It uses attributes to locate an element on the web page.

   Usage:
   WebElement element = driver.findElement(By.cssSelector(“<css_expression>”));

   Syntax:
   - For id attribute:
     tagname#id_attribute_value
   - For class attribute:
     tagname.class_attribute_value
   - Generic syntax:
     tagname[attribute_name = ‘attribute_value’]

8. **xpath:** xpath stands for xml (Extensible Markup Language) path. It uses xml formats to locate an element on the web page.



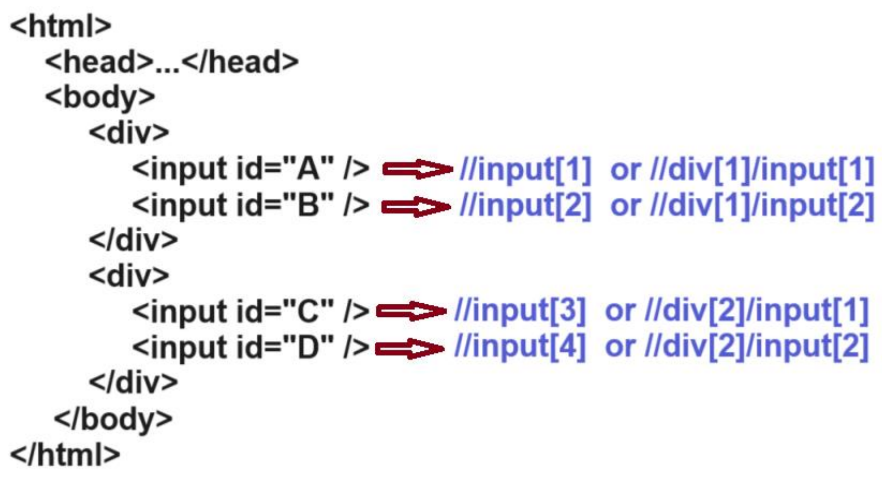
Xpath uses either attributes or text or both to locate element. It helps in traversing through the HTML page. There are two types of xpath:

1) **Absolute xpath:** It is the complete path from the root of HTML tree till the element node. Traversing is done using '/'.
Ex:

```
<html>
    <head>...</head>
    <body>
        <div>
            <input id="A" /> ⇨ /html/body/div[1]/input[1]
            <input id="B" /> ⇨ /html/body/div[1]/input[2]
        </div>
        <div>
            <input id="C" /> ⇨ /html/body/div[2]/input[1]
            <input id="D" /> ⇨ /html/body/div[2]/input[2]
        </div>
     </body>
</html>
```

Drawbacks of Absolute xpath:
- ➢ Xpath is lengthy
- ➢ Time consuming
- ➢ Only forward traversing is possible
- ➢ In real time due to frequent requirement changes, GUI might change which might affect the absolute xpath.

2) **Relative xpath:** Xpath written directly to an element using '//' is called Relative xpath. Using Relative xpath we can traverse from any point in the HTML tree to the required element. In Relative xpath backward traversing is also possible.
Ex:



```
<html>
  <head>...</head>
  <body>
    <div>
      <input id="A" />  => //input[1]  or //div[1]/input[1]
      <input id="B" />  => //input[2]  or //div[1]/input[2]
    </div>
    <div>
      <input id="C" />  => //input[3]  or //div[2]/input[1]
      <input id="D" />  => //input[4]  or //div[2]/input[2]
    </div>
   </body>
</html>
```

RELATIVE XPATH

Basic Relative Xpath:

1. **Xpath by attributes:** Xpath written directly to locate an element using only attributes of the element.
   Syntax:
   //tagname[@attribute_name = 'attribute_value']

2. **Xpath by text():** Xpath written directly to locate an element using text on the element.
   Syntax:
   //tagname[text() = 'text_value']
   Or
   //tagname[. = 'text_value']

   . → used for the text present on the same node or in the child nodes

3. **Xpath by contains():** Xpath written directly to locate an element using attributes or text on the element.
   Syntax:
   Using attribute → **//tagname[contains(@attribute_name, 'attribute_value')]**
   Using text → **//tagname[contains(text(), 'text_value')]**

   Xpath by contains() handles:

   - Lengthy attribute values or text values
   - Partially dynamic elements
     Elements whose values partially change such as software versions, build versions etc
     Ex: Selenium 4.22.0 , Java JDK 22 etc
   - Spaces in the text including non-breakable spaces.
     Sometimes web designers or developers include spaces in the text of an element using a command '&nbsp'. These spaces are called as non-breakable spaces. The text having these spaces are not identified by xpath by text(). We should use xpath by contains() to handle these spaces.

Advanced Relative Xpath:

1. **Xpath by group index:** Whenever we have multiple matching elements, to fetch specific element we use position of that particular element. This is called as xpath by group index.
   Syntax:
   (relative_xpath_expression)[position_value]

2. **Xpath by traversing:**
   Traversing means moving from one element node to another in HTML.
   Xpath by traversing is generally used to locate dynamic elements in a web page.

   Steps to locate dynamic elements:
   - Identify static element and write xpath expression to it
   - Identify the common parent of static and dynamic elements and traverse to it
   - Traverse to dynamic element and write tag name or tag name with attributes

   We have two types:

   I. **Independent and Dependent Xpath:**
   Independent xpath → xpath of a static element
   Dependent xpath → xpath of a dynamic element
   Using independent and dependent xpaths we can locate a dynamic element.
      i. **Forward Traversing:** Traversing from a node to its immediate child node is called forward traversing. It can be done using '/'.
      ii. **Backward Traversing:** Traversing from a node to its immediate parent node is called backward traversing. It can be done using '/..'

   II. **Xpath by axes:**
   Xpath by axes is used to optimize xpath expressions. We have different xpath axes.
   - **parent:** Used to traverse from a node to its immediate parent node. It is similar to backward traversing.
     Syntax:
     /parent::tagname

   - **child:** Used to traverse from a node to its immediate child node. It is similar to forward traversing.
     Syntax:
     /child::tagname

   - **ancestor:** Used to traverse from a node to any parent node including immediate parent.
     Syntax:
     /ancestor::tagname
   - **descendant:** Used to traverse from a node to any child node including immediate child.
     Syntax:
     /descendant::tagname

   - **sibling functions:** Nodes which are at the same level having common parent are called sibling functions.

- **preceding-sibling**: Used to traverse from a node to the sibling nodes that are lying above the current node.
  Syntax:
  /preceding-sibling::tagname

- **following-sibling:** Used to traverse from a node to the sibling nodes that are lying above the current node.
  Syntax:
  /following-sibling::tagname

- **preceding:** Used to traverse from a node to the nodes that are at the same level, lying above the current node having uncommon parent.
  Syntax:
  /preceding::tagname

- **following:** Used to traverse from a node to the nodes that are lying below the current node having uncommon parent.
  Syntax:
  /following::tagname

Xpath functions:

1. **last():** When there are **n** number of child tags under a parent tag, then to locate the last child tag, we can use last() XPath Function. It is basically an index of the last element in the group of similar elements.
   Syntax: **(xpath_expression)[last()]**

2. **normalize-space():** This function strips leading and trailing white-space including non-breakable spaces from a string, replaces sequences of whitespace characters by a single space, and returns the resulting string.
   Syntax: **//tagname[normalize-space(@attribute_name) = ‘attribute_value’]**
   //tagname[normalize-space(text()) = ‘text_value’]

3. **name():** This function handles “svg” tags. “svg” stands for scalable vector graphics. These tags are used for graphic designing of logos and widgets of the webpages. These tags are not identified directly. Hence we use name() to identify these tags.
   Syntax: **//*[name() = ‘svg’][@attribute_name = ‘attribute_value’]**
   //*[name() = ‘svg’ and @attribute_name = ‘attribute_value’]

4. **starts-with():** This function can be used to locate an element/ elements with the beginning portion of the text or beginning portion of the attribute value.
   Syntax: **//tagname[starts-with(@attribute_name, ‘attribute_value)]**
   //tagname[starts_with(text(), ‘text_value’)]

Xpath keywords:

1. **and:** This expression uses two conditions. Both conditions should be true for finding the element. It fails to find the element if any one condition is false.

2. **Or:** This expression uses two conditions, whether the first condition OR second condition should be true. It is also applicable if any one of the conditions is true, or maybe both. This means that any one condition should be true to find the element.

SYNCHRONIZATION

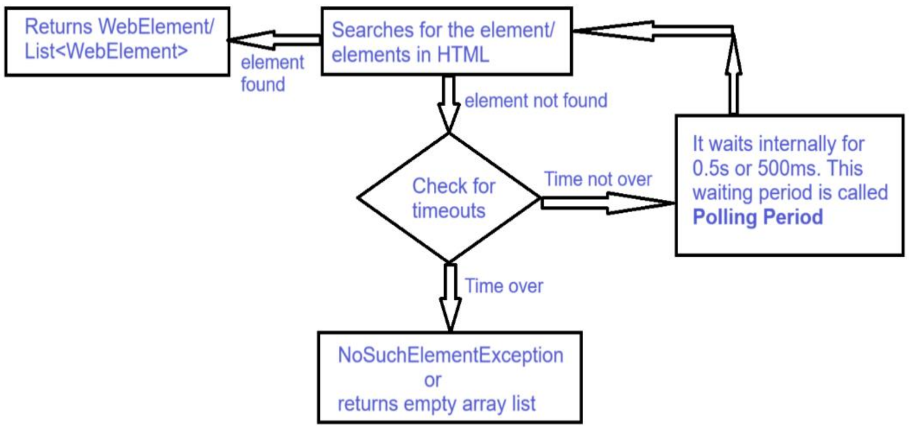
- Synchronization is the process of syncing the selenium script with application when interacting with the web elements on the web page.
- When we perform an action on the web page, it is expected that all components involved work together seamlessly. This collaborative process among components is referred to as synchronization.
- Synchronization ensures that the code and applications execute in more efficiently to carry out the desired operation.

Types of wait statements:

1. **Thread.sleep():**
   - It is java wait statement which waits completely for specified amount of time.
   - It accepts time in milliseconds in long datatype as an argument.
   - Return type is void.
   - It throws InterruptedException.
   - Usage:

     Thread.sleep(long ms);

2. **Implicitly wait:**
   - It is selenium wait statement which synchronizes findElement() and findElements() methods only.
   - It is a global wait setting **that applies to all elements in a Selenium script**.
   - It waits a specified time before throwing an exception if the element is not found.The default implicitly wait value is 0.
   - T**he implicit wait should be set right after initializing the** *WebDriver* **instance.**
   - **WorkFlow:**

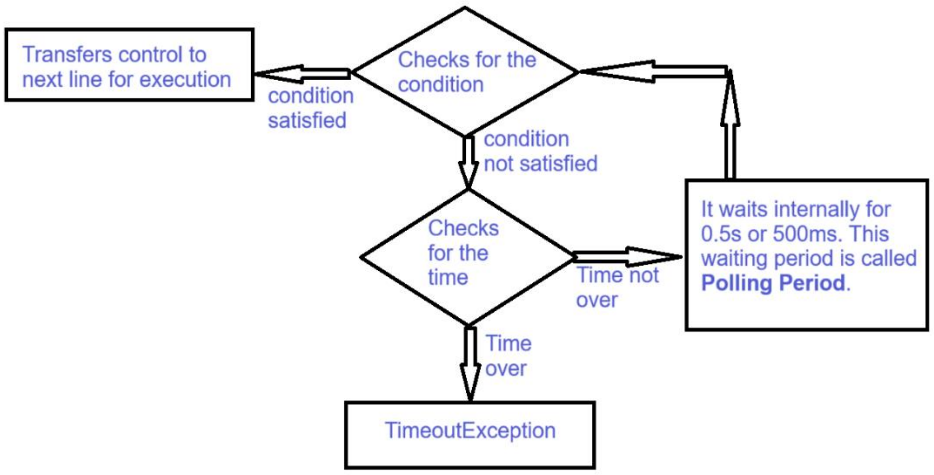

Returns WebElement/ List<WebElement>
element found
Searches for the element/ elements in HTML
element not found
Check for timeouts
Time not over
It waits internally for 0.5s or 500ms. This waiting period is called Polling Period
Time over
NoSuchElementException or returns empty array list

   - Whenever implicitly wait is applied to the selenium script, it searches for the element / list of elements on the web page.
   - If element / elements are found, it returns WebElement / List<WebElement> respectively based on the method used.
   - If the element / elements are not found, it checks if the specified time is over or not.
   - If the time is over, it throws NoSuchElementException or it returns empty array list respectively.

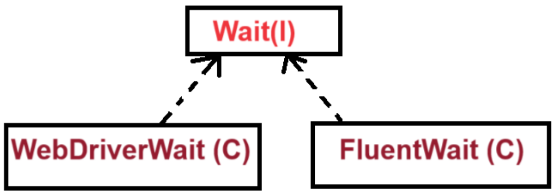
- If the time is not over, it waits internally for 0.5s or 500ms. This waiting period is called as Polling period.
- Once the polling period is done, it searches for the element / elements again.
- This process repeats until the element / elements are found or time is over.
- **Syntax:**
  For Selenium 3 version →
  driver.manage().timeouts().implicitlyWait(<time_in_sec>, TimeUnit.SECONDS);

  For Selenium 4 version →
  driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(<time_in_sec>);

3. **Explicitly wait:**
   - It is selenium wait statement which synchronizes all the methods including findElement() and findElements() methods.
   - It **is a more flexible wait that allows us to wait for a specific condition to be met before continuing test execution.**
   - Unlike implicitly wait, explicitly wait should be given every time whenever synchronization is needed.
   - **WorkFlow:**


Transfers control to next line for execution
condition satisfied
Checks for the condition
condition not satisfied
Checks for the time
Time not over
It waits internally for 0.5s or 500ms. This waiting period is called **Polling Period**.
Time over
TimeoutException

   - Whenever explicitly wait is applied to the selenium script, it checks for the condition.
   - If the condition is satisfied, it transfers control to the next line for execution.
   - If the condition is not satisfied, it checks if the specified time is over or not.
   - If the time is over, it throws TimeoutException.
   - If the time is not over, it waits internally for 0.5s or 500ms. This waiting period is called as Polling period.
   - Once the polling period is done, it checks for the condition again.
   - This process repeats until the condition is satisfied or time is over.
   - Explicitly wait can be set using two classes:
     1. **WebDriverWait**
     2. **FluentWait**
   - The above classes are the implementing classes of Wait interface.



- **WebDriverWait Syntax:**
  For Selenium 3 version →
  WebDriverWait wait = new WebDriverWait(driver, <time_in_sec>);
  wait.until(ExpectedConditions.visibilityOf(<element>));
  or
  wait.until(ExpectedConditions.elementToBeClickable(<element>));
  or
  wait.until(ExpectedConditions.titleContains(<page_title>));

  For Selenium 4 version →

WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(<time_in_sec>);

  wait.until(ExpectedConditions.visibilityOf(<element>));
  or
  wait.until(ExpectedConditions.elementToBeClickable(<element>));
  or
  wait.until(ExpectedConditions.titleContains(<page_title>));

- **FluentWait:**
  - It is another type of explicitly wait.
  - It is used to customize polling period.
  - Unlike WebDriverWait, FluentWait does not have predefined methods to provide condition. User should define a method to provide condition.
  - It also ignores any exception that occurs before timeouts.

4. **Custom wait:**
   - It is user defined wait statement.
   - Wait statement can be defined using loops and exception handling
   - Sample custom wait statement:

```
WebElement header = null;
int count = 0;
while(count < 20) {
        try {
                header = driver.findElement(<locator>));
                break;
        }catch(Exception e) {
                Thread.sleep(1000);
                count++;
        }
}
```

Differences between implicitly wait and explicitly wait:

implicitly wait	explicitly wait
1. It synchronizes only findElement() and findElements() methods	1. It synchronizes all the methods including findElement() and findElements()
2. It does not need condition.	2. Condition is mandatory to specify the wait.
3. It is global wait which is set only once in the beginning of the test script.	3. It should be specified every time when the synchronization is needed.
4. It throws NoSuchElementException or it returns empty array list when elements are not found.	4. It throws TimeoutException when the condition is not satisfied.

WEBELEMENT METHODS

The elements which appear on the web page like text fields, drop downs, links, images etc., are called as web elements.

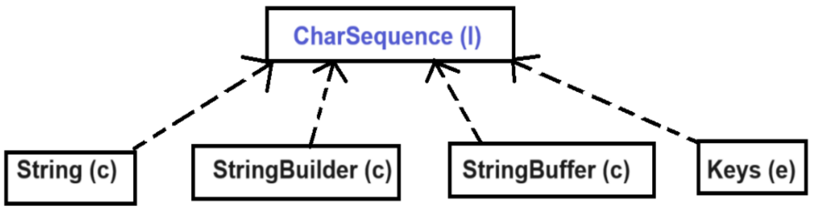
WebElement is an interface in org.openqa.selenium package which is used to perform different operations on web elements.

Abstract methods declared within WebElement interface are categorised into three sections.
1. Action methods
2. Getter methods
3. Validation methods

1. Action methods:

1) sendKeys():

- This method will simulate typing into the WebElement.
- It accepts CharSequence (I) type arguments.



- CharSequence is an interface in java. It is readable sequences of char values. The implementing classes of CharSequence are String, StringBuilder and StringBuffer, and Keys enum.
- Keys enum is present in org.openqa.selenium package, which is the representation of pressable keys which does not have text on it.
- Return type is void.
- If the data passed is null, this method will throw **IllegalArgumentException.**

2) click():

- This method clicks on any element on the web page.
- Return type is void.

3) clear():

- This method will reset the content on the web element.

- Return type is void.

4) **submit():**
   - This method clicks on an element which is present within <form></form> and it should have type="submit" attribute.
   - Return type is void.

2. Getter methods:

1) **getText():**
   - This method fetches the text on the web element, including sub elements.
   - Return type is String.
   - Usage:
     String text = element.getText();

2) **getTagName():**
   - This method fetches the tag name of the web element from the element node.
   - Return type is String.
   - Usage:
     String tagName = element.getTagName();

3) **getAttribute():**
   - This method fetches the attribute value of the specified attribute name from the element node.
   - It accepts attribute name in the String data type as an argument.
   - Return type is String.
   - Usage:
     String attributeValue = element.getAttribute("<attributeName>");

4) **getCssValue():**
   - This method fetches the computed style or CSS (Cascading Style Sheet) properties such as color, font, background-color etc., of an element.
   - It accepts CSS property key in String data type as an argument.
   - Return type is String.
   - Usage:
     String cssValue = element.getCssValue("<css_property_key>");

5) **getLocation():**
   - This method fetches the location of an element on the web page.
   - Return type is Point class.
   - Point class is present in org.openqa.selenium package. It has two non-static methods –
     I. getX() – It fetches the x-coordinate of the element. It returns int.
     II. getY() - It fetches the y-coordinate of the element. It returns int.
   - Usage:
     Point point = element.getLocation();
     int x = point.getX();
     int y = point.getY();

6) **getSize():**
   - This method fetches the size of an element on the web page.
   - Return type is Dimension class.

- Dimension class is present in org.openqa.selenium package. It has two non-static methods –
  I. getHeight() – It fetches the height of the element. It returns int.
  II. getWidth() - It fetches the width of the element. It returns int.
- Usage:
  Dimension dimension = element.getSize();
  int height = dimension.getHeight();
  int width = dimension.getWidth();

7) getRect():

- This method fetches both the size and location of an element on the web page.
- Return type is Rectangle class.
- Rectangle class is present in org.openqa.selenium package. It has four non-static methods –
  I. getX() – It fetches the x-coordinate of the element. It returns int.
  II. getY() - It fetches the y-coordinate of the element. It returns int.
  III. getHeight() – It fetches the height of the element. It returns int.
  IV. getWidth() - It fetches the width of the element. It returns int.
- Usage:
  Rectangle rect = element.getRect();
  int height = rect.getHeight();
  int width = rect.getWidth();
  int x = rect.getX();
  int y = rect.getY();

8) getScreenshotAs():

- This method fetches the screenshot of an element from the web page.
- Steps to capture element screenshot:
  I. File temp = element.getScreenshotAs(OutputType.FILE);
  II. File target = new File("<filePath>");
  III. FileUtils.copyFile(temp, target); → IOException
  To copy the file to permanent memory we use external libraries called apache commons io. These libraries are used for sharing or copying files or directories.

3. Validation methods:

1) isEnabled():

- This method checks if an element is enabled to perform any operation on it.
- Return type is Boolean.

2) isDisplayed():

- This method checks if an element is displayed on the web page.
- Return type is Boolean.

3) isSelected():

- This method checks if an element is selected on the web page. Generally we use this method for checkboxes or radio buttons.
- Return type is Boolean.