How to configure Selenium WebDriver:

1. Install Firefox

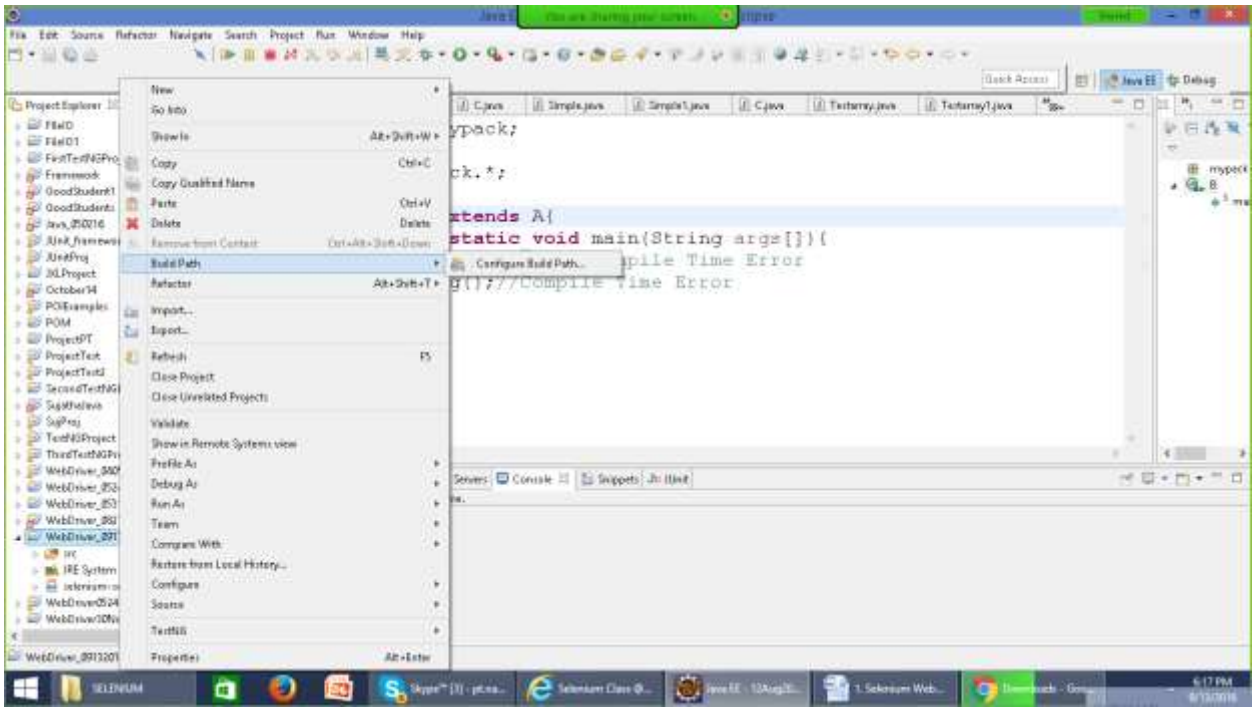
2. Configure Selenium Webdriver:

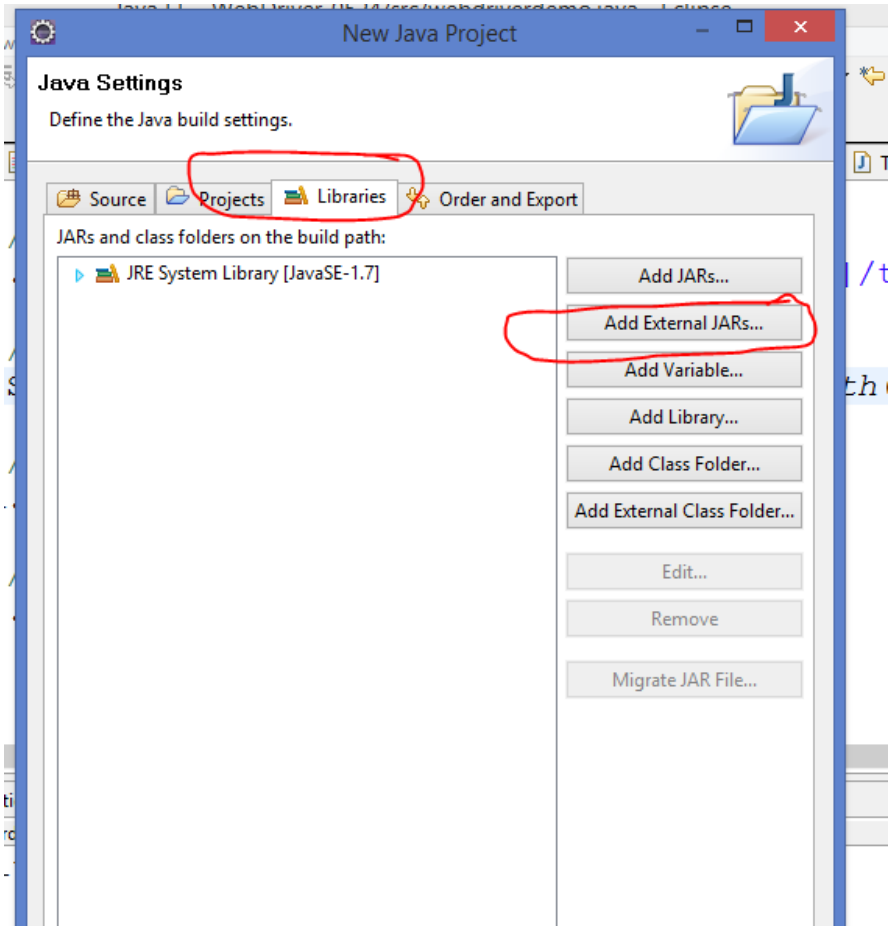
1. Go to

https://www.selenium.dev/downloads/

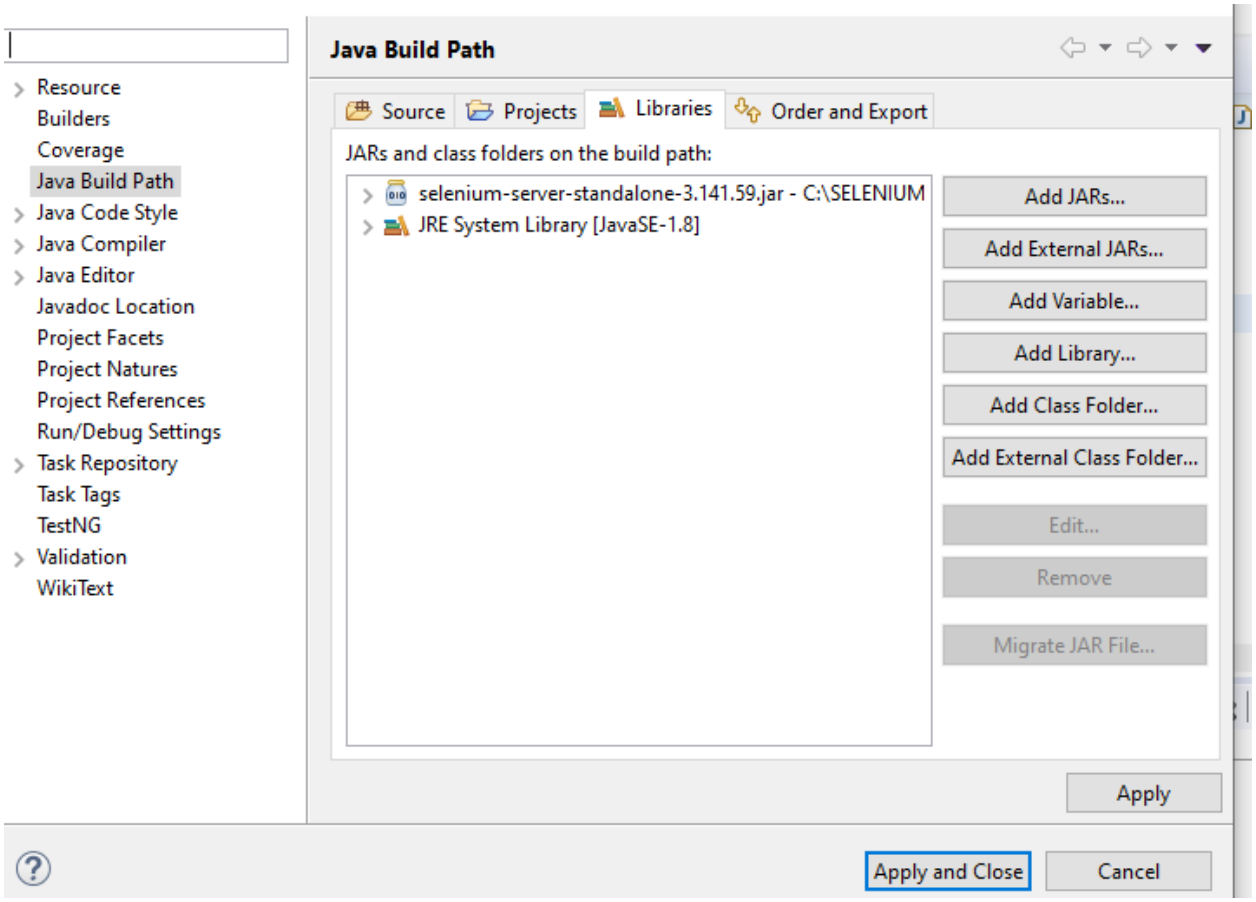
2. Download Selenium Standalone Server from,

3. Save in your local path
4. Create a new project in eclipse → Configure web driver jar file



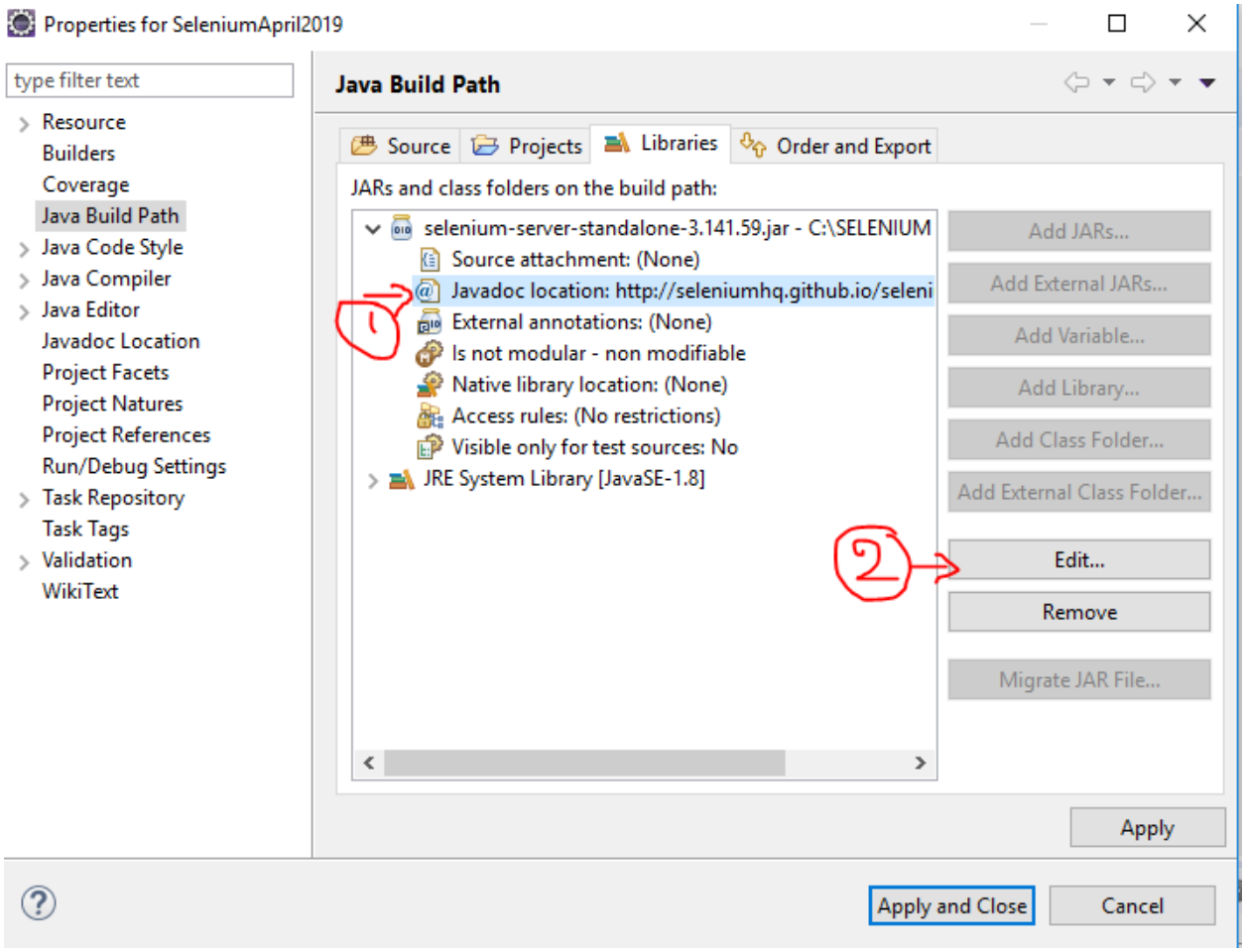


Add your library “selenium-server-standalone-3.141.59.jar”



3. Attaching help doc to the Java project:

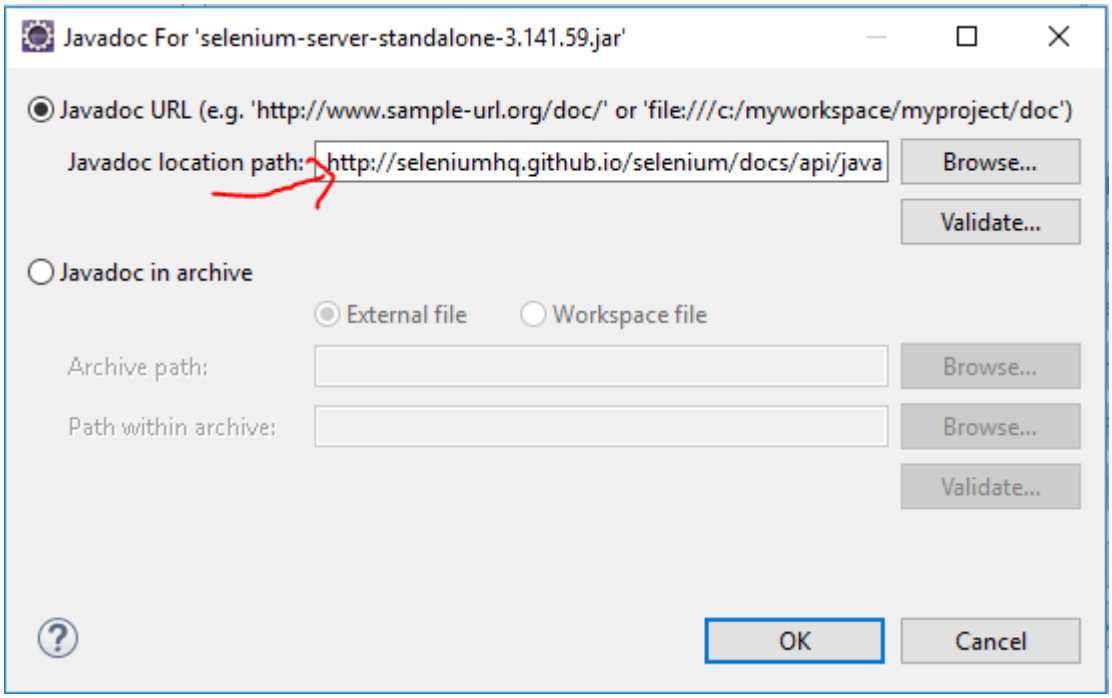
Go to configure Build path,



Copy below path

http://seleniumhq.github.io/selenium/docs/api/java/

and paste as shown in below screen



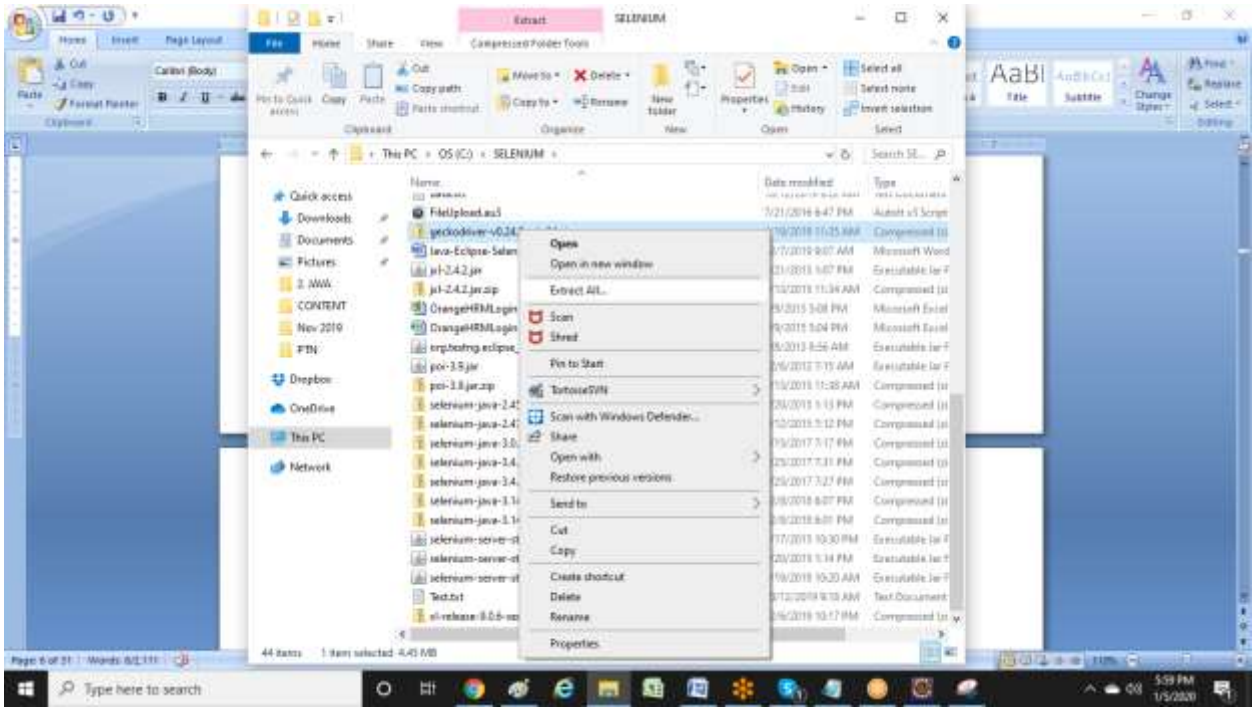
4. Download and save GECKODRIVER:

Go to,

https://github.com/mozilla/geckodriver/releases

and download geckodriver-v0.24.0-win64.zip from,

Save in your local system and unzip it.



About Web Driver:

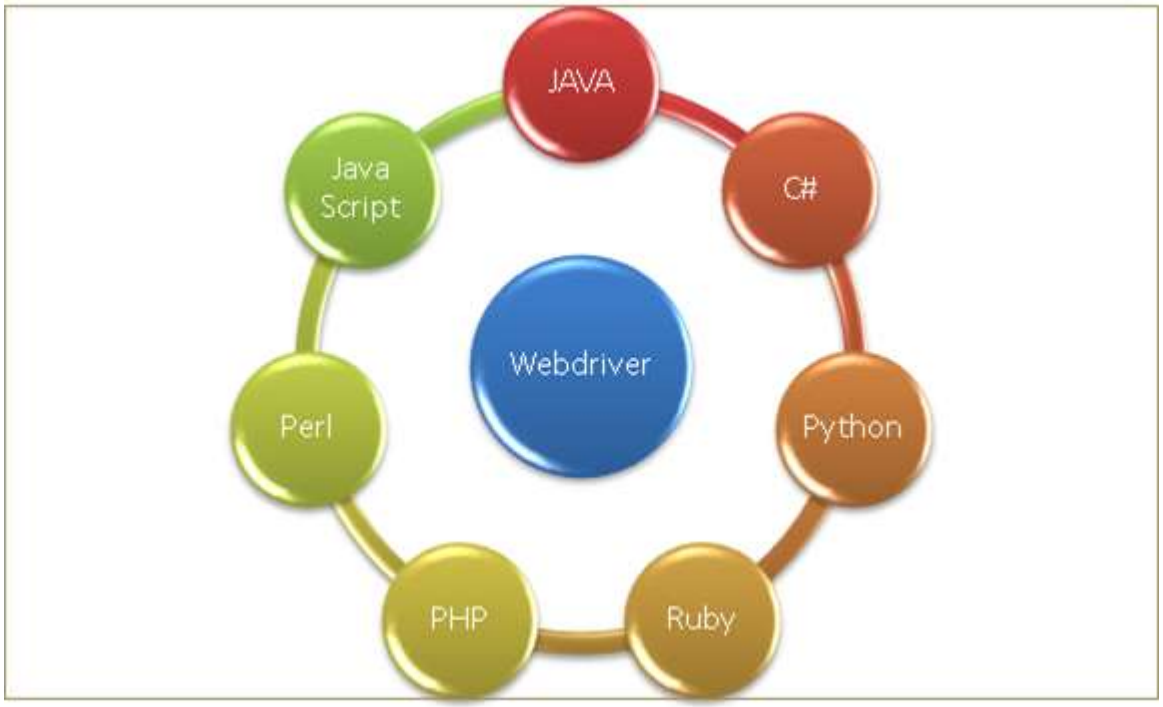
WebDriver is a tool for testing web applications **across different browsers** using different programming languages.

"WebDriver" is an interface, contains below categories of methods.

- Control to browser
- WebElements selection
- Debugging

WebDriver supports "***Multiple Languages***", "***Multiple Support Browsers***" and "***Multiple Platforms***".

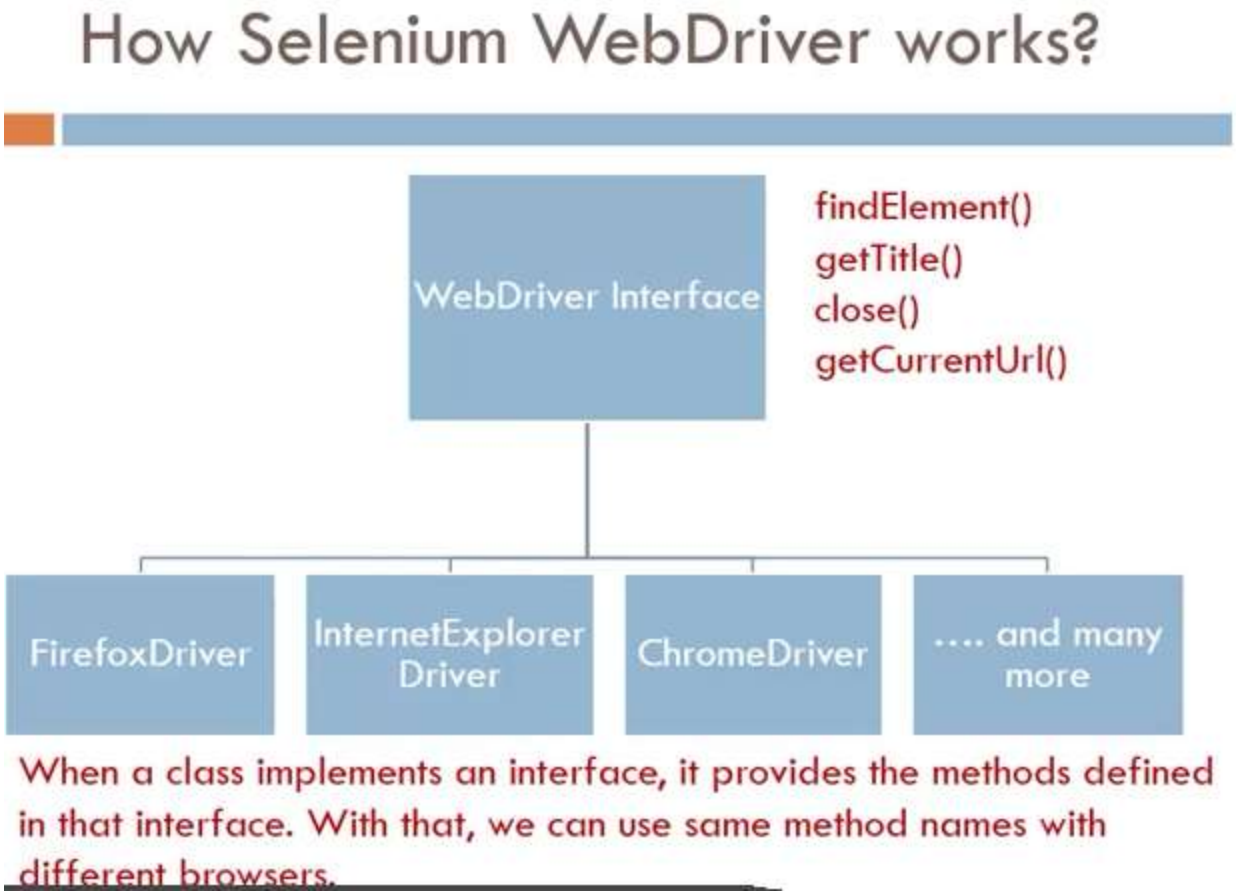
Multiple Languages Binding SUPPORT



Multiple Browsers SUPPORT

Multiple Platforms SUPPORT


iOS
Mac
Linux
Android
Windows



WebDriver Existence in package *org.openqa.selenium*

org.openqa.selenium: The global package of Selenium which we import in our program every time when it is needed to use *WebDriver*.

https://selenium.dev/selenium/docs/api/java/index.html

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class WebDriverExample1  {
    public static void main(String[] args) {

      System.setProperty("webdriver.gecko.driver","C:\\SELENIUM\\geckodriver\\geckodriver.exe");

      FirefoxDriver driver = new FirefoxDriver(); //Opens FF browser

       driver.get("http://www.facebook.com"); // Enter facebook.com
```

```
        String tagName = driver.findElement(By.id("email")).getTagName(); // F E get
tag

        System.out.println(tagName);

        driver.findElement(By.id("email")).sendKeys("Hello"); // Type hello in email
text box

       // driver.findElement(By.xpath("//*[@id=\"email\"]")).sendKeys("Hello");

        //driver.close();
        //System.exit(0);
}
}
```

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;

public class WebDriverExample2  {
    public static void main(String[] args) {

      System.setProperty("webdriver.gecko.driver","C:\\SELENIUM\\geckod
      river\\geckodriver.exe");

      FirefoxDriver driver = new FirefoxDriver();

         // And now use this to visit Google
         driver.get("http://www.google.com");

         // Find the text input element by its name
         WebElement drvr1 = driver.findElement(By.name("q"));

         // Enter something to search for
         drvr1.sendKeys("Selenium");

         // Now submit the form. WebDriver will find the form for
us from the element
         drvr1.submit();

         // Check the title of the page
         System.out.println("Page title is: " +
driver.getTitle());

         driver.quit();
    }
}
```

Some of the Webdriver methods

get("url") – navigate to the url specified.
getTitle() - Get the title of the current browser.
getPageSource() - Get the source code of the current browser /Page .
close() - Closes the current browser.
quit() - Close all associated browsers.
getCurrentUrl() - Get the URL of the current webpage.
getWindowHandle()-Return an handle of the window that uniquely identifies it within this driver instance.
navigate("url")--allowing the driver to access the browser's history and to navigate to a given URL
navigate().back()--Move back to previous page in the browser's history.
navigate().forward()--Move to forward page in the browser's history
navigate().refresh()--Refresh the current page
switchTo()--Send future commands to a different frame or window / change focus to specified window / frame

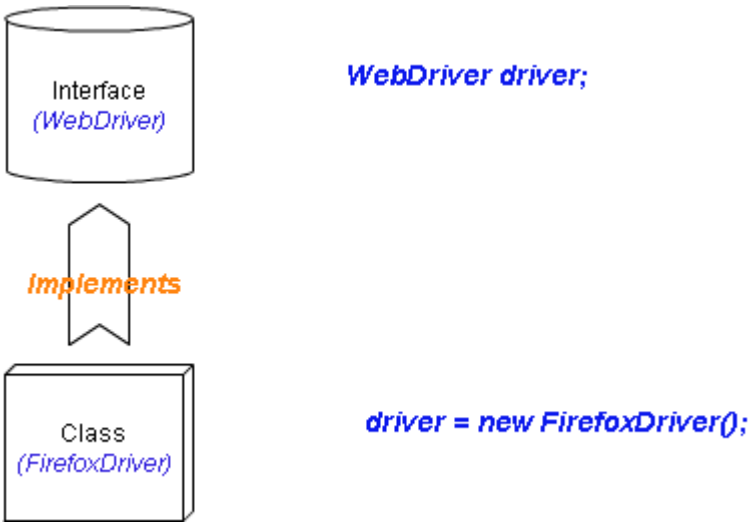
When we first start with selenium automation our very first line of code comes as :-

WebDriver driver = new FireFoxDriver();

What is an Interface?

Interface is like a blueprint of Class. It contains variables and body less methods(Abstract methods), where we just declare methods but we implement them inside the class which inherit Interface.

In our example that Class is FireFoxDriver and Interface is WebDriver, so we can say *WebDriver driver = New FirefoxDriver();*

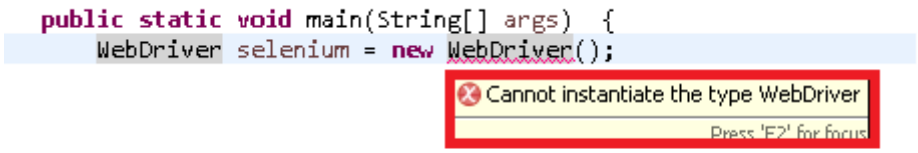


Note:- We need to use the Implements Keyword to consume Interface inside a Class.

More about an Interface:

Interface instance:-
We can create a reference variable of an interface but we can't instantiate any interface since it is just a contract to be implemented in a Class.

WebDriver driver = New WebDriver ...Not allowed,



but below code is allowed....
Example:-

WebDriver driver = New FireFoxDriver();
driver.get(testUrl);

1. Fetch Mercury Tours' homepage

2. Verify its title

3. Print out the result of the comparison

4. Close it before ending the entire program.

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class WebDriverExample3{
    public static void main(String[] args) {
        System.setProperty("webdriver.gecko.driver","C:\\SELENIUM\\geckodriver\\geckodriver.exe");

      // declaration and instantiation of objects/variables
      WebDriver driver = new FirefoxDriver();
      String baseUrl = "http://newtours.demoaut.com";
      String expectedTitle = "Welcome: Mercury Tours";
      String actualTitle = "";

      // launch Firefox and direct it to the Base URL
      driver.get(baseUrl);

      // get the actual value of the title
      actualTitle = driver.getTitle();

      /*
```

```
        * compare the actual title of the page with the expected one and print
        * the result as "Passed" or "Failed"
        */
       if (actualTitle.contentEquals(expectedTitle)){
          System.out.println("Test Passed!");
       } else {
          System.out.println("Test Failed");
       }

       //close Firefox
       driver.close();

       // exit the program explicitly
       System.exit(0);
    }
 }
```

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
public class WebDriverExample4{

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

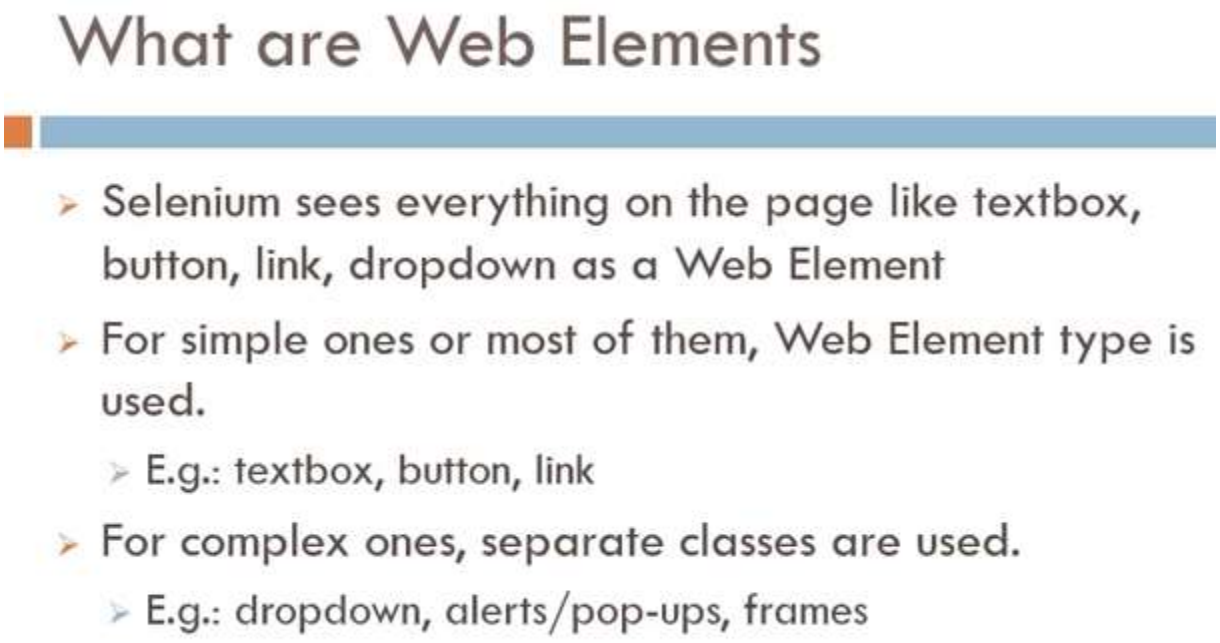
```
System.setProperty("webdriver.gecko.driver","C:\\SELENIUM\\geckodriver\\geckodriver.exe");

    WebDriver driver = new FirefoxDriver();
    String baseUrl = "http://www.facebook.com";
    String tagName = "";

    driver.get(baseUrl);
    tagName = driver.findElement(By.id("email")).getTagName();
    System.out.println(tagName);
    driver.close();
    System.exit(0);
  }
}
```

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
public class WebDriverExample5{

    public static void main(String[] args) {
System.setProperty("webdriver.gecko.driver","C:\\SELENIUM\\geckodriver\\geckodriver.exe");
        WebDriver driver = new FirefoxDriver();
        String baseUrl = "http://www.facebook.com";
        String tagName = "";

        driver.get(baseUrl);
        tagName =
driver.findElement(By.id("email")).getAttribute("name");
        System.out.println(tagName);
        driver.close();
        System.exit(0);
    }
```

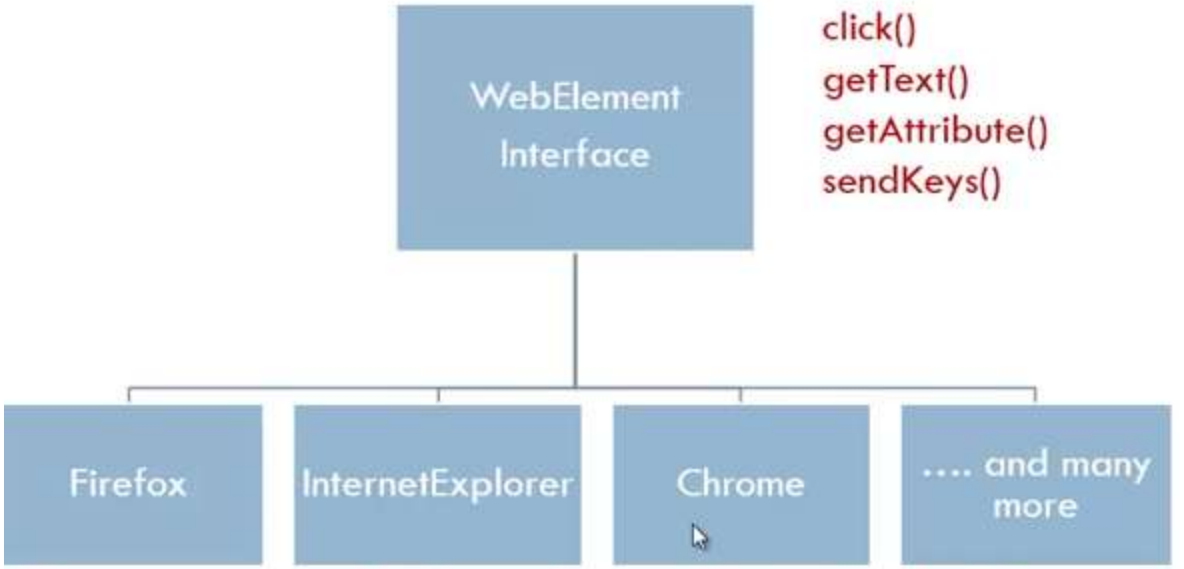
}

Locating GUI Elements:

Locating elements in WebDriver is done by using the "findElement(By.locator())" method.



WebElement Interface



We can use same method names for web elements with different browsers.

Identifying Web Elements in Webpage

- For many Selenium commands, a target is required
- Target identifies a web element uniquely in the content of web application
- Target consists of the locating strategy and format like By.LocatorStrategyType("value")
- findElement() always finds the first WebElement using the given locating strategy
- findElements() returns all web elements matching the given locating strategy

- Locating By Id
  - Locates the first element with matching id attribute.
  - driver.findElement(By.id("UserID")).sendKeys("UserName");
- Locating By Name
  - Locates the first element with matching name attribute.
  - driver.findElement(By.name("pwd")).sendKeys("passsword");
- Locating By LinkText
  - Locates the first hyperlink with matching link text.
  - This is a simple method of locating a hyperlink in your web page by using the text of the link.
  - driver.findElement(By.linkText("Sign in")).click();

- Locating By PartialLinkText
  - Locates the first hyperlink which contains specified link text.
  - driver.findElement(
    By.partialLinkText("Can't access")).click();

- Locating By Xpath
  - Locates the first element as specified by Xpath strategy
  - driver.findElement(
    By.xpath("//input[@placeholder='Email']")).sendKeys("User Name");

- Locating By CssSelector
  - Locates the first element as specified by CssSelector strategy
  - driver.findElement(
    By.cssSelector("css=input.input_submit")).click();

- Locating By TagName
  - Useful to get all elements with a given tag
  - driver.findElements(By.tagName("a")).size();

- Locating By ClassName
  - Useful to get all elements with a given class or display style
  - driver.findElements(
    By.className("input_text")).size();

How to use Locators in Selenium:

Identification of correct GUI elements is a prerequisite to create an automation script.

The different types of locator are:

- ID
- Name
- Link Text
- CSS Selector
    - Tag and ID
    - Tag and class
    - Tag and attribute
    - Tag, class, and attribute
    - Inner text
- XPath

The choice of locator depends largely on your Application Under Test.

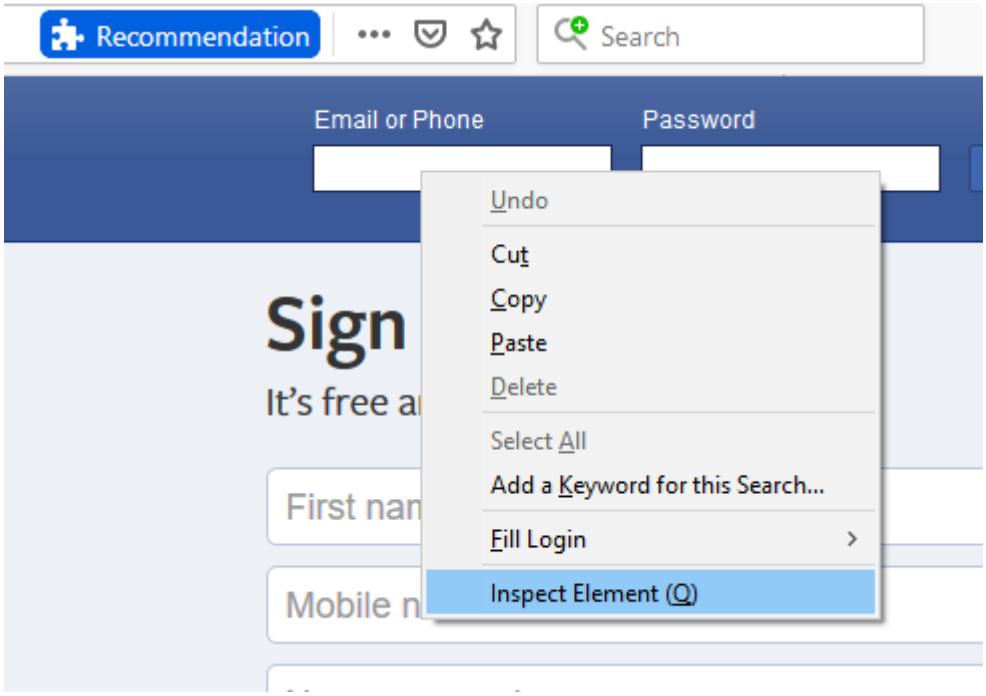
Locating by ID

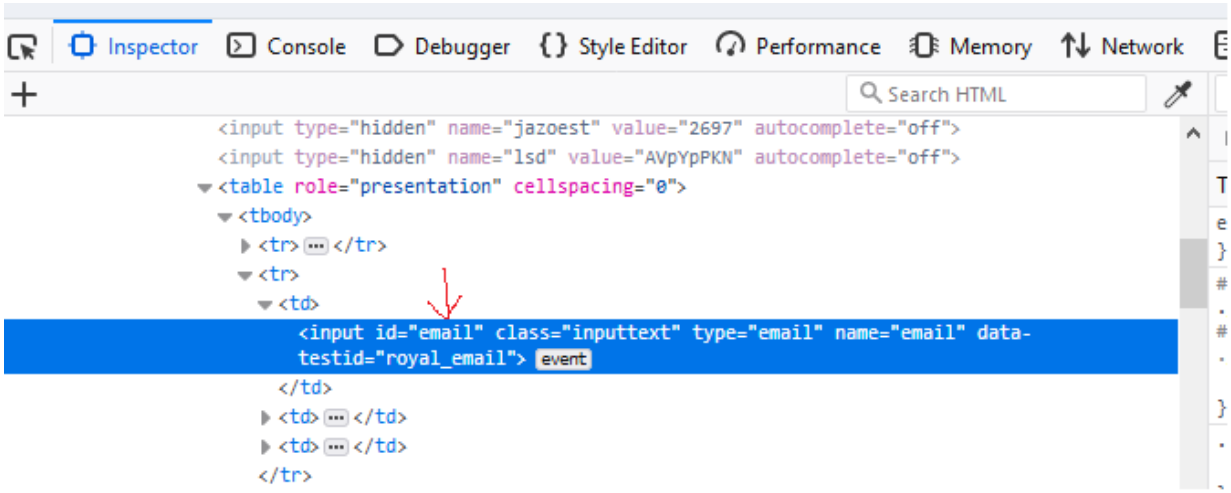
This is the most common way of locating elements since ID's are supposed to be unique for each element.

Target Format: id=*id of the element*

For this example, we will use Facebook as our test app because Mercury Tours does not use ID attributes.

Step 1. Navigate to http://www.facebook.com. Inspect the "Email or Phone" text box using Inspect element as shown in below and take note of its ID. In this case, the ID is "email".






```
By.id("email")
```

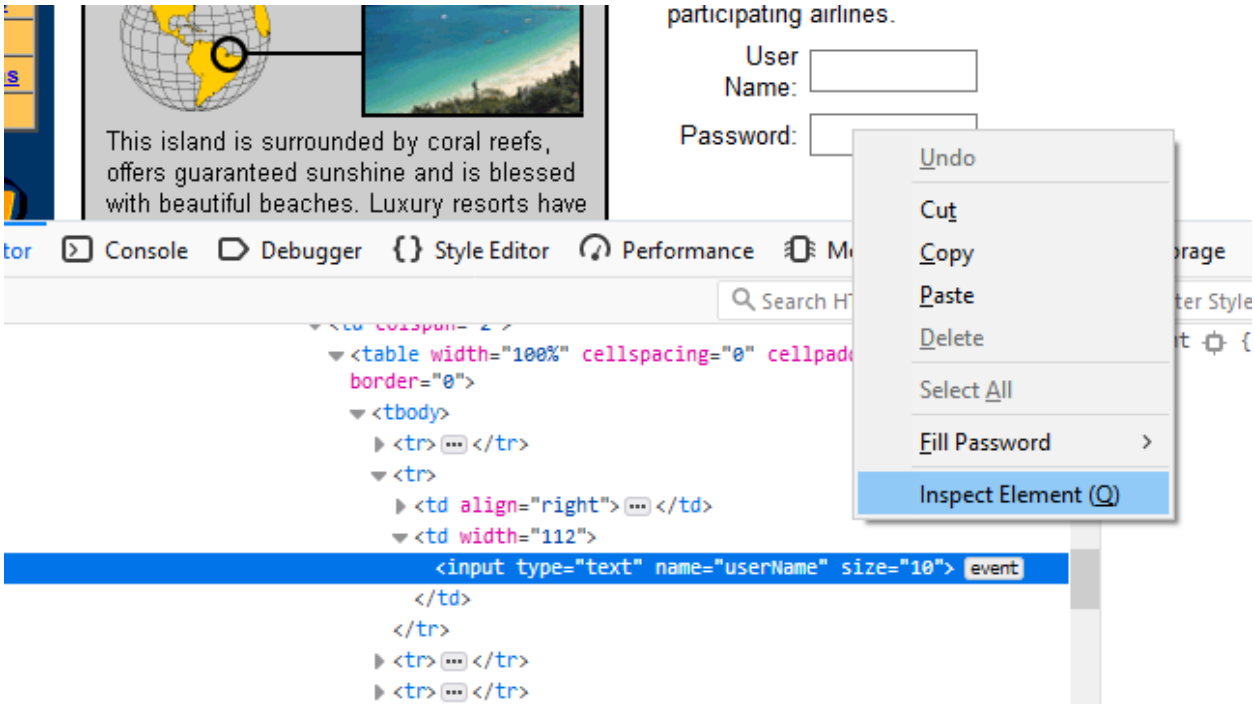
Locating by Name

Locating elements by name are very similar to locating by ID, except that we use the **"name="** prefix instead.

Target Format: name=*name of the element*

In the following demonstration, we will now use Mercury Tours because all significant elements have names.

Step 1. Navigate to http://newtours.demoaut.com/ and use Inspect element as shown in below to inspect the "User Name" text box. Take note of its name attribute.



Here, we see that the element's name is "username".

```
By.name("userName")
```

Locating by Link Text:

This type of locator applies only to hyperlink texts. We access the link by prefixing our target with "link=" and then followed by the hyperlink text.

Target Format: link=*link_text*

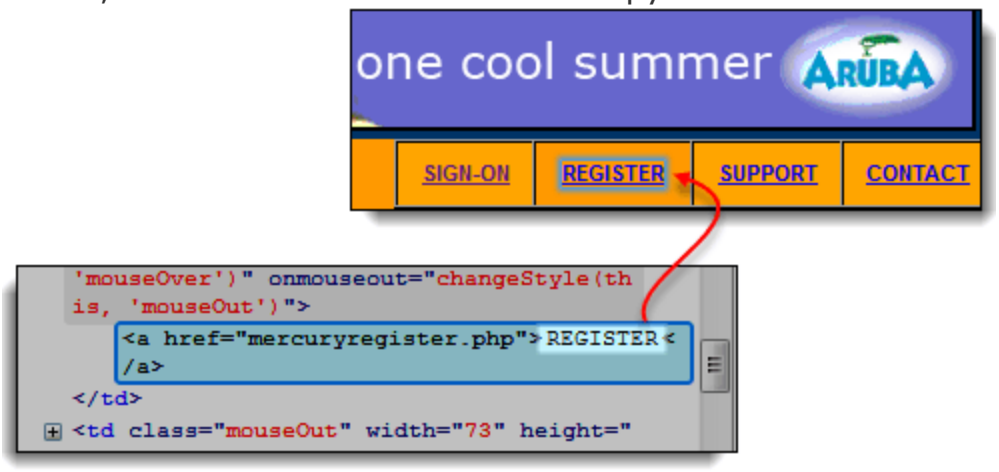
In this example, we shall access the "REGISTER" link found in the Mercury Tours homepage.

Step 1.

- First, make sure that you are logged off from Mercury Tours.
- Go to Mercury Tours homepage.

Step 2.

- Using Inspect element as shown in below, inspect the "REGISTER" link. The link text is found between and tags.
- In this case, our link text is "REGISTER". Copy the link text.




```
By.linkText("REGISTER");
```

featured vacation destinations.

```
By.linkText("featured vacation destinations");
```

```
    By.partialLinkText("featured");
```

```
By.partialLinkText("featured vacation");
```

featured xxxx.

featured vacation destinations.

featured vacation yyyy.

```
By.partialLinkText(“featured”);
```

Locating by CSS Selector:

CSS Selectors are string patterns used to identify an element based on a combination of HTML tag, id, class, and attributes.Locating by CSS Selector is more complicated than the previous methods, but it is the most common locating strategy of advanced Selenium users because it can access even those elements that have no ID or name.

CSS Selectors have many formats, but we will only focus on the most common ones.

- Tag and ID
- Tag and class
- Tag and attribute
- Tag, class, and attribute
- Inner text

When using this strategy, we always prefix the Target box with "css=" as will be shown on the following examples.

Locating by CSS Selector - Tag and ID

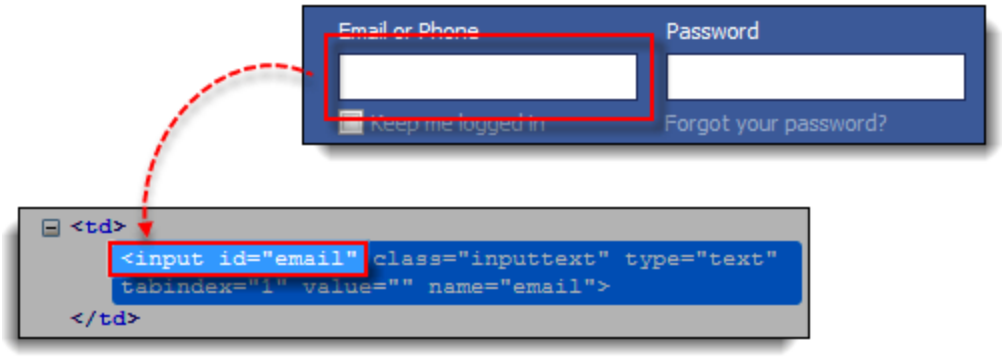
Again, we will use Facebook's Email text box in this example. As you can remember, it has an ID of "email" and we have already accessed it in the "Locating by ID" section. This time, we will use a CSS Selector with ID in accessing that very same element.

Syntax	Description
css=*tag#id*	tag = the HTML tag of the element being accessed # = the hash sign. This should always be present when using a CSS Selector with ID id = the ID of the element being accessed

Keep in mind that the ID is always preceded by a hash sign (#).

Step 1. Navigate to www.facebook.com. Using Inspect element as shown in below, examine the "Email or Phone" text box.

At this point, take note that the HTML tag is "input" and its ID is "email". So our syntax will be "css=input#email".



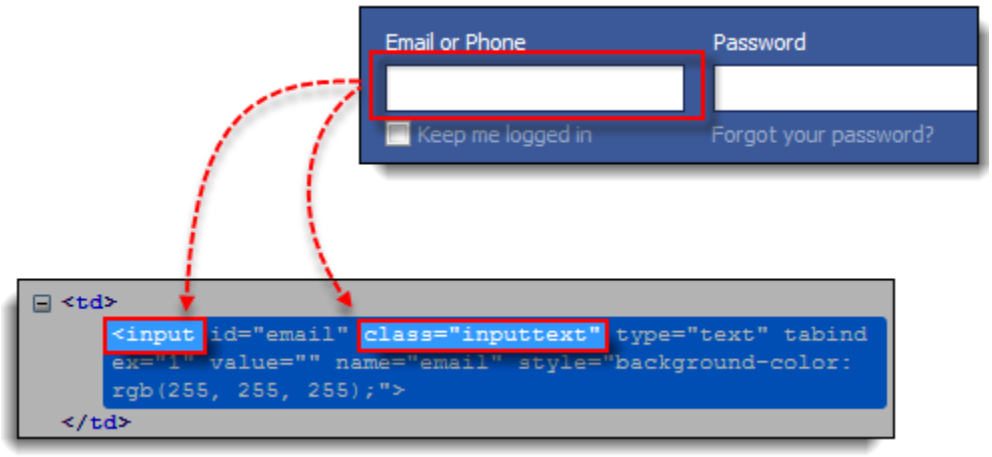

```
By.cssSelector("input#email");
```

Locating by CSS Selector - Tag and Class

Locating by CSS Selector using an HTML tag and a class name is similar to using a tag and ID, but in this case, a dot (.) is used instead of a hash sign.

Syntax	Description
css=*tag*.*class*	tag = the HTML tag of the element being accessed . = the dot sign. This should always be present when using a CSS Selector with class class = the class of the element being accessed

Step 1. Navigate to www.facebook.com and use Inspect element as shown in below to inspect the "Email or Phone" text box. Notice that its HTML tag is "input" and its class is "inputtext".



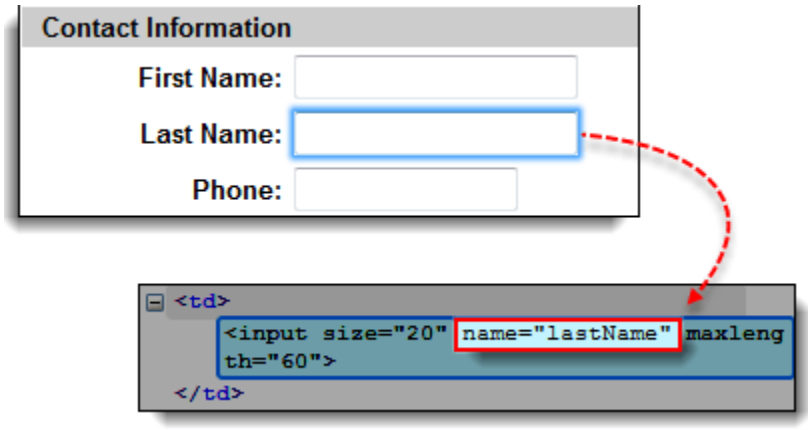

```
By.cssSelector("input.inputtext login_form_input_box");
```

Locating by CSS Selector - Tag and Attribute

This strategy uses the HTML tag and a specific attribute of the element to be accessed.

Syntax	Description
css=*tag*[*attribute*=*value*]	tag = the HTML tag of the element being accessed [and] = square brackets within which a specific attribute and its corresponding value will be placed attribute = the attribute to be used. It is advisable to use an attribute that is unique to the element such as a name or ID. value = the corresponding value of the chosen attribute.

Step 1. Navigate to Mercury Tours' Registration page (http://newtours.demoaut.com/mercuryregister.php) and inspect the "Last Name" text box. Take note of its HTML tag ("input" in this case) and its name ("lastName").



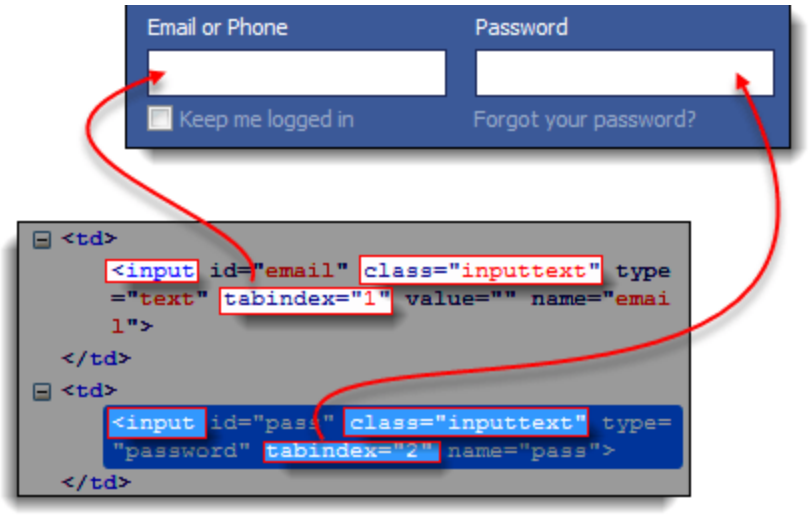

```
By.cssSelector("input[id=\"email\"]");
```

When multiple elements have the same HTML tag and attribute, only the first one will be recognized. This behavior is similar to locating elements using CSS selectors with the same tag and class.

Locating by CSS Selector - tag, class, and attribute

Syntax	Description
css=*tag.class*[*attribute=value*]	tag = the HTML tag of the element being accessed . = the dot sign. This should always be present when using a CSS Selector with class class = the class of the element being accessed [and] = square brackets within which a specific attribute and its corresponding value will be placed attribute = the attribute to be used. It is advisable to use an attribute that is unique to the element such as a name or ID. value = the corresponding value of the chosen attribute.

Step 1. Navigate to www.facebook.com and use Inspect element as shown in below to inspect the 'Email or Phone' and 'Password' input boxes. Take note of their HTML tag, class, and attributes. For this example, we will select their 'tabindex' attributes.



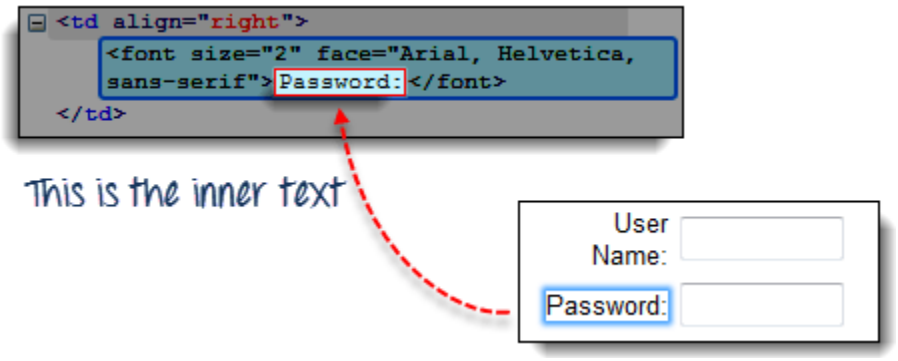

```
By.cssSelector("input.inputtext login_form_input_box[id=\"email\"]");
```

Locating by CSS Selector - inner text

As you may have noticed, HTML labels are seldom given id, name, or class attributes. So, how do we access them? The answer is through the use of their inner texts. **Inner texts are the actual string patterns that the HTML label shows on the page.**

Syntax	Description
css=*tag*:contains("*inner text*")	tag = the HTML tag of the element being accessed inner text = the inner text of the element

Step 1. Navigate to Mercury Tours' homepage (http://newtours.demoaut.com/) and use Inspect element as shown in below to investigate the "Password" label. Take note of its HTML tag (which is "font" in this case) and notice that it has no class, id, or name attributes.



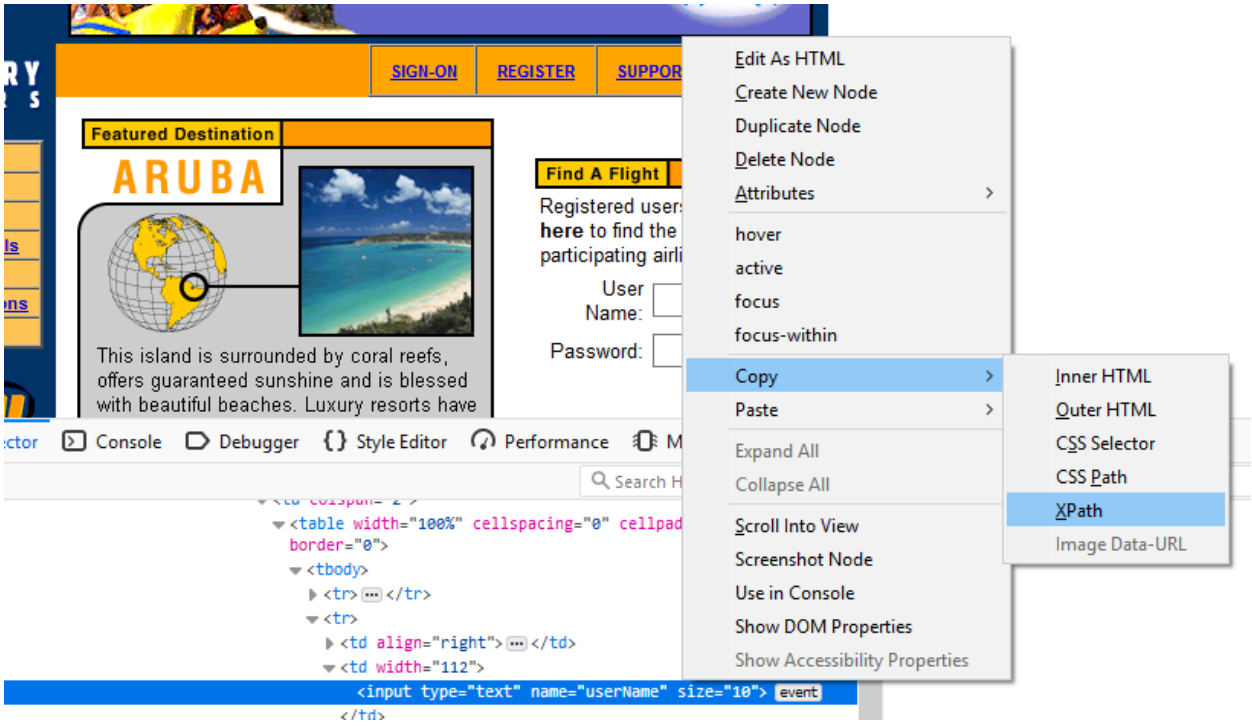

```
By.cssSelector("font:contains(\"Password:\")");
```

Locating by XPath

XPath is the language used when locating XML (Extensible Markup Language) nodes. Since HTML can be thought of as an implementation of XML, we can also use XPath in locating HTML elements.

Advantage: It can access almost any element, even those without class, name, or id attributes.

Step 1. Navigate to Mercury Tours Homepage and use Inspect element as shown in below to inspect the orange rectangle to the right of the yellow "Links" box. Refer to the image below.



Step 2. Right click on the element's HTML code and then select the "Copy XPath" option.

```
By.xpath("/html/body/table/tbody/tr[1]/td[1]");
By.xpath("/html/body/table/tbody/tr[1]/td[2]");
By.xpath("/html/body/table/tbody/tr[2]/td[1]");
By.xpath("/html/body/table/tbody/tr[2]/td[2]");
```

Summary

Syntax for Locator Usage

Method	Target Syntax	Example
By ID	id=*id_of_the_element*	id=email
By Name	name=*name_of_the_element*	name=username
By Link Text	link=*link_text*	link=REGISTER
Tag and ID	css=*tag#id*	css=input#email
Tag and Class	css=*tag.class*	css=input.inputtext
Tag and Attribute	css=*tag*[*attribute=value*]	css=input[name=lastName]
Tag, Class, and Attribute	css=*tag.class*[*attribute=value*]	css=input.inputtext[tabindex=1]
Xpath	By.xpath(“ xpath”)	`By.xpath("/html/body/table/tbody/tr[2]/td[2]");`