**BROWSER LAUNCHING**

1. Open the browser

Firefox browser:

**public class** FacebookAccount {

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

// to configure driver System.*setProperty*("webdriver.gecko.driver",

"C:/Users/siva/workspace/Selenium/driver/geckodriver.exe");

// create the firefox driver

WebDriver driver = **new** FirefoxDriver();

// url mention driver.get("https:/[/www.facebook.com/](http://www.facebook.com/)"); driver.close();

}}

**public** **class** basicSeleniumOperation {

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

// **TODO** Auto-generated method stub

System.setProperty("webdriver.chrome.driver", "./Drivers/chromedriver.exe"); // ./ --> current directory

WebDriver w = new ChromeDriver(); // creating object of chrome driver

w.get("https://opensource-demo.orangehrmlive.com/web/index.php/auth/login");

String title = w.getTitle(); // to get title of page

System.out.println(title);

String url =w.getCurrentUrl(); // to get current page url

System.out.println(url);

String page =w.getPageSource(); // to get page source code

System.out.println(page);

}

}

**Write a Script to open and close the browser based on user input , also use the following:**

* Open the browser
* Open in incognito mode
* Delete all cookies
* Maximize the window
* Thread.sleep
* Implicit wait

**public** **class** basicSeleniumOperation {

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

// **TODO** Auto-generated method stub

// write a program to open browser which is specified by user

// 1. get user input

// 2 create condition for browser (if else)

System.***out***.println("which browser you want?");

Scanner sc = **new** Scanner(System.***in***);

String browsername = sc.next(); // to get user input and store in string

WebDriver driver = **null**; // here driver is instance variable

// if else

**if**(browsername.equalsIgnoreCase("chrome"))

{ // here op is local variable to if function

ChromeOptions op = **new** ChromeOptions();// creating object of chrome option

op.addArguments("--incognito");

driver = **new** ChromeDriver(op); // passing property at the time of creation

}

**else** **if**(browsername.equalsIgnoreCase("edge"))

{

EdgeOptions op = **new** EdgeOptions();

op.addArguments("-inprivate");

driver = **new** EdgeDriver(op);

}

**else** **if**(browsername.equalsIgnoreCase("firefox"))

{

FirefoxOptions op = **new** FirefoxOptions();

op.addArguments("-private");

driver = **new** FirefoxDriver(op);

}

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

driver.manage().deleteAllCookies(); // to delete all cookies

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

// An implicit wait in Selenium WebDriver tells the driver to wait for a certain amount of time before throwing //an exception when trying to find an element. It's a global setting applied to the entire WebDriver instance, //meaning it affects all element location calls throughout the script.

* When the control comes to findElement() or findElements(), it will check whether the element is present or not
* If the element is present, it will return address of the specified element.
* If the element is not present, it will check whether implicitlyWait() is specified or not.
* If implicitlyWait() is not specified then it will throw NoSuchElementException or Empty list.
* If implicitlyWait() is specified then it will check whether the specified time is over or not.
* If the specified time is over then it will throw NoSuchElementException or Empty list.
* If the specified time is not over then for every 500ms it will check whether the element is present or not.

**Thread.*sleep*(3000);** // it will halt/stop the execution for specific time

driver.get("https://opensource-demo.orangehrmlive.com/web/index.php/auth/login");

Thread.*sleep*(3000);

String titlenew =driver.getTitle();

System.***out***.println(titlenew);

driver.close();

driver.quit();

}

}

* **Note**
  + Here driver.quit will give warning because session is closed by driver.close()

**WARNING: Connection reset**

**java.net.SocketException: Connection reset**

## Close();

* It is used to close the application.
* It will close the current browser.

## Quit():

* Destroy the driver object.
* It will close all browser windows opened by selenium webdriver

**Note: The above script is an example for Run Time Ploymorphism**.

To run same script on multiple browsers, we are converting sub class object into interface type(upcasting).

WebDriver driver = new ChromeDriver();

WebDriver driver = new FirefoxDriver();

## Methods of WebDriver Interface:

|  |  |  |
| --- | --- | --- |
| 1 | get() | To enter the url |
| 2 | getTitle() | To get the title of current web page |
| 3 | getCurrentUrl() | To get the url of current web page |
| 4 | getPageSource() | To get the page source of current web page |
| 5 | findElement() | To get single webElements |
| 6 | findElements() | To get multiple webElements |
| 7 | getWindowHandle() | To get the id of parent window |
| 8 | getWindowHandles() | To get the id of All windows |
| 9 | switchTo() | Used to switch one window to other window |
| 10 | manage() | 1. Window 2. Cookies |
| 11 | navigate() | 1. Enter the URL 2. Navigate to previous page 3. Navigate to next page 4. Refresh current web page |
| 12 | close() | To close the current/parent browser |
| 13 | quit() | To close all the browsers opened by selenium |

### NAVIGATION COMMANDS

1. Navigate().to()
2. Refresh()
3. Back()
4. Forword()

|  |  |
| --- | --- |
| **get** | **navigate** |
| It will just enter the URL | 1. It will enter the URL 2. It will navigate to previous page 3. It will navigate to next page 4. It will refresh the current web page |
| After entering the URL it will not allow any statements to execute untill the page loads  completely | After entering the URL it will not wait untill the page loads completely |

**public** **class** navigate\_find\_windows {

**public** **static** WebDriver *w*; // instance variable scope--> entire class

**IV.--> We cannot connect static method to non static object.**

**public** **void** initialize() {

// local variable --> scope limited to method/function

ChromeOptions op = **new** ChromeOptions();

op.addArguments("--incognito");

*w* = **new** ChromeDriver(op);

*w*.manage().window().maximize();

*w*.manage().deleteAllCookies();

*w*.manage().timeouts().implicitlyWait(Duration.*ofSeconds*(10));

}

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

// IV --> can we attach non static variable to static method --> NO

navigate\_find\_windows a = **new** navigate\_find\_windows();

a.initialize();

*w*.get("https://opensource-demo.orangehrmlive.com/web/index.php/auth/login");

*w*.findElement(By.*name*("username")).sendKeys("Admin");

*w*.findElement(By.*name*("password")).sendKeys("admin123");

*w*.findElement(By.*xpath*("//button[@type='submit']")).click(); // IMP

Thread.*sleep*(2000);

String title = *w*.getTitle();

System.***out***.println(title);

w.findElement(By.xpath("//li[6]/a")).click();

w.navigate().back(); // go to back page

w.navigate().forward(); // go to next page

w.navigate().refresh(); // refresh the page

w.navigate().to("https://rahulshettyacademy.com/seleniumPractise/#/"); // takes to

new site

w.quit();

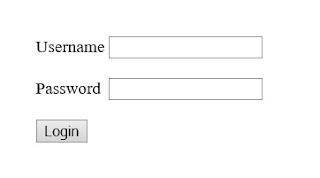
}

}

### Types of Locators:

* **ID:** Locates elements based on their unique "id" attribute.
* **Name:** Locates elements using the "name" attribute.
* **Class Name:** Locates elements based on the "class" attribute.
* **Tag Name:** Locates elements based on their HTML tag (e.g., <div>, <p>, <a>).
* **Link Text:** Locates anchor elements (<a>) based on the visible text they contain.
* **Partial Link Text:** Locates anchor elements based on a portion of their visible text.
* **CSS Selector:** Uses CSS selectors to locate elements, offering a powerful and flexible way to target elements.
* **XPath:** Uses XPath expressions to navigate and locate elements within the DOM. Difference Between get() and navigate():

**1) By ID**  
This is the most common way of locating elements since ID's are supposed to be unique for each element. This mechanism returns the location of web element who's ID is matching with the specified ID in your script. If no element has a matching id attribute, NoSuchElementException will be raised.

Example: If you have a webpage like following  
[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEhN0yhdRXJ1x_JNF6LT693WDAH8rhpshlEb1cHHZ7ZHVIpmwWuqP5kRlhOehGP-fJtT8S0o1WQxjruMQw6FVjP-ip9rZ2Eq4QdkX_VOXy2O_Uu08ogw_Wspg3yliDwuAtYde-47oSKQ3QY3/s1600/Login.jpg)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <html>  <body>  <form id="login">  <label>Username</label>  <input id="username" type="text" name="login" />  <br /><br />  <label>Password</label>  <input id="password" type="password" name="password" />  <br /> <br />  <input type="submit" name="signin" value="Login" />  </form>  </body>  </html> |

You can easily select an element with the help of ID locator from the above example:  
  
id = “username”  
id = “password”  
  
Use the above ID in your selenium script :  
  
driver.findElement(By.id("username"));  
driver.findElement(By.id("password"));  
  
Please note: Avoid using this technique when the ID's are not unique and randomly generated.  
  
**2) By ClassName**  
Use By ClassName locator when you want to locate an element by class attribute name. In this strategy, the location of the web element is returned  who's className attribute is matching with the specified class Name in your script. If no element has a matching class attribute name, a NoSuchElementException will be raised.  
  
Example:   
For instance, consider this page source:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <html>  <body>  <input id="FirstName" type="text" class="fname" />  </body>  <html> |

The input tag element can be located like this:  
driver.findElement(By.className("fname"));  
  
**3) By Name**  
You can use this locator when you know the name attribute of an element. This mechanism returns the location of web element who's name attribute is matching with the specified name attribute in your script. If no element has a matching name attribute, a NoSuchElementException will be raised.  
  
Example:  
For instance, consider this page source:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <html>  <body>  <form id="loginForm">  <input name="username" type="text" />  <input name="password" type="password" />  <input name="continue" type="submit" value="Login" />  </form>  </body>  <html> |

The username & password elements can be located like this:  
driver.findElement(By.name('username'));  
driver.findElement(By.name('password'));  
  
**4) By TagName**  
Use this locator when you want to locate an element by tag name. In this mechanism, the location of the web element can be identified by matching the tag name. If no element has a matching tag name, a NoSuchElementException will be raised.  
  
Example:  
For instance, consider this page source:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <html>  <body>  <h1>Welcome</h1>  <p>Start here..</p>  </body>  <html> |

The heading (h1) element can be located like this:  
driver.findElements(By.tagName(“h1”));  
  
**5) By LinkText**  
Use this locator when you know link text used within an anchor tag. In this mechanism, the location of web element is returned who's link text value is matching with the link text specified in your script. If no element has a matching link text attribute, a NoSuchElementException will be raised.  
  
Example:  
For instance, consider this page source:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <html>  <body>  <p>Please confirm ..</p>  <a href="continue.html">Continue</a>  <a href="cancel.html">Cancel</a>  </body>  <html> |

The continue.html link can be located like this:  
driver.findElement(By.LinkText('Continue'));  
  
**6) By PartialLinkText**  
Use this locator when you want to select link element which contains text matching the specified partial link text.   
  
Example:  
For instance, consider this page source:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <html>  <body>  <p>Please confirm ..</p>  <a href="continue.html">Continue to website</a>  <a href="cancel.html">Cancel</a>  </body>  <html> |

The continue.html link can be located like this:  
driver.findElement(By.partialLinkText("to website"));  
  
**7) By cssSelector**  
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.  
  
Example:  
For instance, consider this page source2222:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <html>  <body>  <input type="email" class="inputtext" name="email" id="email" value="" tabindex="1">  </body>  <html> |

Locating by CSS Selector - Tag and ID  
driver.findElement(By.cssSelector("input[id='email']"));  
  
**8) By Xpath**

1. It is most popular and best way to locate element in selenium webdriver.
2. Xpath is used to locate a web element based on its XML path.
3. XML stands for Extensible Markup Language and is used to store, organize and transport arbitrary data.
4. It stores data in a key-value pair which is very much similar to HTML tags. Both being mark up languages and since they fall under the same umbrella, xpath can be used to locate HTML elements.
5. The fundamental behind locating elements using Xpath is the traversing between various elements across the entire page and thus enabling a user to find an element with the reference of another element.

Xpath can be created in two ways:

Relative xpath --> R -> start with //

Absolute spath --> A -> start with /

**a) Relative Xpath**:  
Relative Xpath begins from the current location and is prefixed with “//”.

//tagname[@attribute=’value’]  
For example: //span[@class=’Name’]  
  
**b) Absolute Xpath:**  
Absolute Xpath begins with a root path and is prefixed with a “/”.   
For example: /html/body/div/div[@id=’Address’]  
  
Absolute Xpath can break if the HTML is changed slightly. So it is less recommended.  
  
Pros of using Xpath: It can access almost any element, even those without class, name, or id attributes.  
  
Cons of using Xpath: It is the most complicated method of identifying elements because of too many different rules and considerations.  
  
Example:  
For instance, consider this page source:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <html>  <body>  <form id="loginForm">  <input name="username" type="text" />  <input name="password" type="password" />  <input name="continue" type="submit" value="Login" />  <input name="continue" type="button" value="Clear" />  </form>  </body>  <html> |

The form element can be located as below  
driver.findElement(By.xpath("/html/body/form[1]"));  
driver.findElement(By.xpath("//form[1]")); //  
driver.findElement(By.xpath("//form[@id='loginForm']"));  
  
The username element can be located as below  
driver.findElement("//form[input/@name='username']"));  
driver.findElement("//form[@id='loginForm']/input[1]"));  
driver.findElement("//input[@name='username']"));  
  
The “Clear” button element can be located as below  
driver.findElement("//input[@name='continue'][@type='button']"));  
driver.findElement("//form[@id='loginForm']/input[4]"));  
  
Following are the ways to write dynamic Xpath:1) start-with method:  
1) startwith method: You can use the start-with in xpath to locate an attribute value that starts with a certain text.  
  
For example, assume you have the following link on the page:  
<a href="mylink\_somerandomstuff">link text</a>  
  
Then you can use the following xpath to find links that have an href that starts with 'mylink'  
driver.findElement("//a[starts-with(@href, "mylink")]"));  
  
2) contains method: Similarly you can use contains method to locate an attribute. Contains() is a method used in XPath expression. It is used when the value of any attribute changes dynamically. driver.findElement("//a[contains(@href, "somerandomstuff")]"));

driver.findElement("//a[contains(text(), "somerandomstuff")]"));

**Selenium: Alert, Actions, and Select - Class vs Interface**

## 1. Alert

## Used to handle JavaScript pop-ups: Alert, Confirm, or Prompt dialogs.

## ➤ Interface: org.openqa.selenium.Alert

## ➤ Common Methods:

## accept() – clicks OK

## dismiss() – clicks Cancel

## getText() – gets the text on alert

## sendKeys(String text) – inputs text into prompt alerts

## Usage: Selenium internally provides the implementation of the Alert interface.

## 2. Actions

## Used for complex user interactions like mouse hover, drag-and-drop, double click, right click, etc.

## ➤ Class: org.openqa.selenium.interactions.Actions

## ➤ Common Methods:

## moveToElement(WebElement) – mouse hover

## click() – click

## doubleClick() – double click

## contextClick() – right-click

## dragAndDrop(source, target) – drag and drop

## sendKeys(Keys.ENTER) – keyboard actions

## perform() – executes the action

## Usage: The Actions class is instantiated using the WebDriver instance.

## 3. Select

## Used to handle drop-down lists (<select> tag in HTML).

## ➤ Class: org.openqa.selenium.support.ui.Select

## ➤ Common Methods:

## selectByVisibleText(String text)

## selectByIndex(int index)

## selectByValue(String value)

## getOptions() – returns all options in dropdown

## getFirstSelectedOption() – gets the currently selected option

## Usage: The Select class works with <select> tags and is instantiated using a WebElement representing the dropdown.

## Comparison Table

|  |  |  |
| --- | --- | --- |
| Feature | Type | Purpose |
| Alert | Interface | To interact with JavaScript alerts, confirms, and prompts. |
| Actions | Class | To perform advanced user interactions (mouse/keyboard actions). |
| Select | Class | To work with <select> dropdown elements. |

File upload

Drag and drop

frame

New tab / window

Multiple Window handle

Table

Screenshot