**What is Selenium?**

**Selenium** is a free (open-source) automated testing framework used to validate web applications across different browsers and platforms. You can use multiple programming languages like Java, C#, Python etc to create Selenium Test Scripts. Testing done using the Selenium testing tool is usually referred to as Selenium Testing.

Selenium Software is not just a single tool but a suite of software, each piece catering to different Selenium QA testing needs of an organization. Here is the list of tools

* Selenium Integrated Development Environment (IDE)
* Selenium Remote Control (RC)
* WebDriver
* Selenium Grid



At the moment, Selenium RC and WebDriver are merged into a single framework to form **Selenium 2**. Selenium 1, by the way, refers to Selenium RC.

**What is Selenium WebDriver?**

**Selenium Webdriver** is an open-source collection of APIs which is used for testing web applications. The Selenium Webdriver tool is used for automating web application testing to verify that it works as expected or not. It mainly supports browsers like Firefox, Chrome, Safari and Internet Explorer. It also permits you to execute cross-browser testing.

package SeleniumBasic;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

public class TestAutomation {

public static void main(String[] args) {

// For Firefox

System.*setProperty*("webdriver.gecko.driver","C:\\Users\\akkip\\Downloads\\geckodriver-v0.29.1-win64\\geckodriver.exe");

WebDriver driver = new FirefoxDriver();

driver.get("http://www.google.com");

// For Chrome

System.*setProperty*("webdriver.chrome.driver", "C:\\Users\\akkip\\Downloads\\chromedriver\_win32\\chromedriver.exe");

WebDriver driver2 = new ChromeDriver();

driver2.get("https://www.cowin.gov.in/home");

}

}

1. **org.openqa.selenium.WebDriver**- contains the WebDriver class needed to instantiate a new browser loaded with a specific driver
2. **org.openqa.selenium.firefox.FirefoxDriver**- contains the FirefoxDriver class needed to instantiate a Firefox-specific driver onto the browser instantiated by the WebDriver class
3. **org.openqa.selenium.chrome.ChromeDriver** - contains the ChromeDriver class needed to instantiate a Chrome -specific driver onto the browser instantiated by the WebDriver class

Normally, this is how a driver object is instantiated.

[First Selenium Webdriver Script: JAVA Code Example](https://www.guru99.com/images/image004(2).png)

A FirefoxDriver class with no parameters means that the default Firefox profile will be launched by our Java program. The default Firefox profile is similar to launching Firefox in safe mode (no extensions are loaded)

The WebDriver class has the **getTitle()** method that is always used to obtain the page title of the currently loaded page.

[First Selenium Webdriver Script: JAVA Code Example](https://www.guru99.com/images/image006(2).png)

The "**close()**" method is used to close the browser window.

[First Selenium Webdriver Script: JAVA Code Example](https://www.guru99.com/images/image008(3).png)

Here, FirefoxDriver is class and WebDriver is an interface which is available in selenium Library.

System.*setProperty*("webdriver.chrome.driver", "C:\\Users\\akkip\\Downloads\\chromedriver\_win32\\chromedriver.exe");

System is a class. And setProperty is method which takes two parameter String Key and String value as shown above program.

String value is nothing but the path of the ChromeDriver. This Drivers are different for each browser. Like

For chrome there is chromedriver available

For Firefox there is geckodriver available

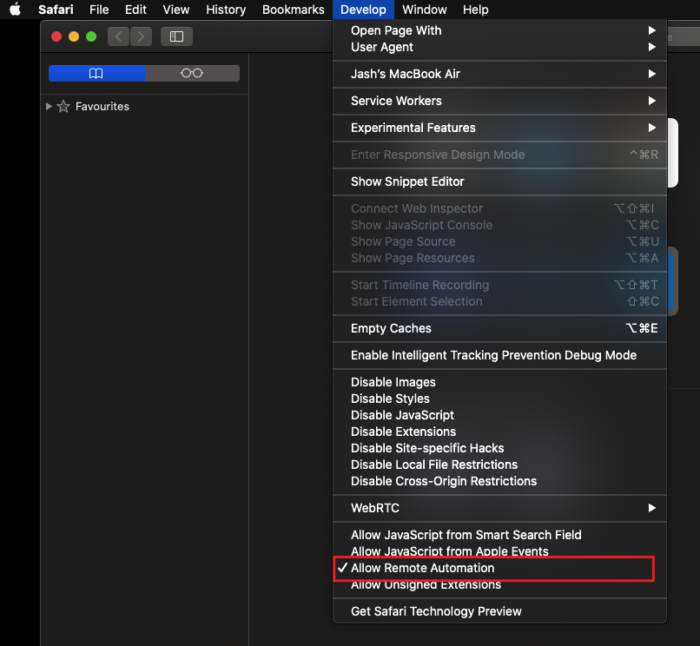
For safari there is safaridriver available which needs to added in the safari browser as an extension we do not need to download it. Safari browser does not require setProperty method.

Directly by instantiating the safariDriver from the WebDriver interface the safari will be lunch.

Prior to Safari automation, enable the Remote Automation feature from the developer menu. To do so, enable the Safari Developer menu first with the steps below:

1. Go to Safari -> Preferences-> Advanced
2. Tick mark the Checkbox with the label – Show Develop menu in menu bar

Once done, go to the Develop menu and click on the Allow Remote Automation option to enable it.



import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.safari.SafariDriver;

public class SafariDemo {

public static void main(String[] args) {

// Instantiate a SafariDriver class.

WebDriver driver = new SafariDriver();

// Launch Website

driver.navigate().to("http://www.google.com/");

// Click on the search text box and send value

driver.findElement(By.id("lst-ib")).sendKeys("BrowserStack");

// Click on the search button

driver.findElement(By.name("btnK")).click();

// Close the Browser

driver.close();

}

Testing is nothing but expected vs actual thing.

**Different locators in WebDriver**

In selenium, everything on the webpage is called Element.

By using HTML tags and other properties of a particular Web elements we can automate the web.

Here are some imp methods:

get() method is used to lunch the URL such as: driver.get("http://www.google.com");

findElement() method is used to find the element on the webpage

**Xpath**

Xpath is always starts with two forward slash //.

Always remember two slash represent the child node.

In below line, we are find the element of the web using Xpath locator

There are 8 different locators in selenium which will be covered in next:

This is the Xpath locator. SendKey is the method to input.

driver.findElement(By.Xpath(//\*[@id=’firstname’])).Sendkey(“Tom”);

there is two types of Xpath which are:

Absolute Xpath: this Xpath is hierarchy based xpath. It start from Parent node to child node. We cannot use this Xpath. It minimizes the performance. As well as it could be change as per the requirements.

Ex. //\*[@id="root"]/div/div/div[2]/div/div[2]/div/div[4]/div/div/div/div/div/div[1]/div/div/div/div[2]

Relative Xpath: this Xpath is recommended. As it cannot be changed and there is no Parent node attached to the path like absolute Xpath.

**(\*)** => it’s a wild card symbol Denotes "any" (node with unrestricted name).

Ex. //\*[@id="firstname"] and .//input[@value='Google Search'] it is customized Xpath.

Both are same.

**Id**

Secondly, we have id which is one the locator for web elements

Driver.findElement(By.id(“firstname”)).sendkey(“Tom”);

**Name**

Name is also a locator. If we have name for a particular element then use the name of the element to locate it.

Driver.findElement(By.name(“firstname”)).sendkey(“Tom”);

**linkText**

lintext locator is used for links only such as:

driver.findElement(By.linkText(“Digital thermometers | Starting ₹299”)).click();

linktext is the name of that link not URL.

**Partial linkText**

This linktext is used whenever there is long linktext. We don’t need to write complete linktext. Such as: we have a link- How to pick up a great userid

Driver.findElement(By.partialLinkText(“How to pick”)).click();

**CssSelector**

We can create a css selector using a property of the Element starts with #.

Suppose we have id=”firstname”;

CSSSelector starts with # followed by id, class, etc.

Here, the CssSelector this Element is “#firstname”

Driver.findElement(By.CssSelector(“#firstname”)).Sendkey(“Tom”);

**Class**

Using class name of the element we can locate the elements on the web.

But there is drawback of this class locator is that more than one element can have class name hence it is not recommended to use.

Driver.findElement(By.classname(“Abc”)).Click();

**Handling Drop box**

To handle the drop box we have to create an object of the Select class and this Select class is already found in Selenium. Such as

Select select = new Select();

We have to pass the argument in select constructor.

Select select = new Select(driver2.findElement(By.*xpath*("//select[@id='searchDropdownBox']")));

select.selectByVisibleText("Computers & Accessories");

driver2.findElement(By.*xpath*("//input[@id='twotabsearchtextbox']")).sendKeys("dell");

driver2.findElement(By.*cssSelector*("#nav-search-submit-button")).click();

**How to handle JavaScript alerts and Popup.?**

While handling with alert we cannot inspect the alert or Popup. Hence we have to use switchTo() method and then we have to create the object of an alert class.

Alert alert = driver.SwitchTo().alert();

While handling with alert popups we need To stop the automation script for some seconds we can use Thread.sleep(5000); method. It will stop the script for 5 sec(5000ms=5sec).

public class AlertPopupHandle {

public static void main(String[] args) {

System.*setProperty*("webdriver.chrome.driver","C:\\Users\\akkip\\Downloads\\chromedriver\_win32\\\\chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.get("https://www.rediff.com/");

driver.findElement(By.*xpath*("//a[@class='signin']")).click();

driver.findElement(By.*xpath*("//input[@class='signinbtn']")).click();

try {

Thread.*sleep*(5000);

} catch (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

Alert alert = driver.switchTo().alert();

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

// Validation Part

String text = alert.getText();

if (text.equals("Please enter a valid user name")) {

System.***out***.println("Correct statement");

} else {

System.***out***.println("Wrong statement");

}

alert.accept(); //To click on Ok

// alert.dismiss(); //To click on cancel

}

}

**How to handle upload windows.?**

When we automate a webpage in which some file needs to uploaded in this case we cannot inspect the window. Hence there are some method as below:

Use the Xpath of that particular element and use SendKey method to pass the path of that file which needs to be upload. It will work if only there is attribute available which is type=”File”

Ex.

driver.findElement(By.*id*("fileupload")).sendKeys("C:\\Users\\akkip\\Desktop\\JAVA\\Java Selenium.docx");

public class HandleUploadWindows {

public static void main(String[] args) {

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

System.*setProperty*("webdriver.chrome.driver","C:\\Users\\akkip\\Downloads\\chromedriver\_win32\\chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.get("https://html.com/input-type-file/");

driver.findElement(By.*id*("fileupload")).sendKeys("C:\\Users\\akkip\\Desktop\\JAVA\\Java Selenium.docx");

driver.findElement(By.*xpath*("//input[@type='submit']")).click();

driver.quit();

}

}

**How to handle frame elements.?**

When there is frame on the webpage. i.e. a webpage on the webpage or it could be anywhere on the parentframe. Hence the selenium locator won’t work until and unless if we do not use the SwitchTo() method to switch from parentFrame to childFrame.

Ex. Driver.switchTo().frame(frameName);

Driver.switchTo().frame(frameIndex);

**Mouse movement**

To perform mouse movement. There is a class named Actions by using this class we can access the invisible element on the webpage and perform mouse movement. Hence firstly, we have to create an object of the Actions class and then pass the driver reference to that class constructor. Such as below

Actions action = new Actions(driver\_reference);

Now, using action object we can access some methods of the Actions class such as: now it will just move mouse to that particular element. & will click on it

Action.moveToElement(driver.findElement(By.linkText(“link\_text”))).build.perform();

public class MouseMovements {

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

System.*setProperty*("webdriver.chrome.driver","C:\\Users\\akkip\\Downloads\\chromedriver\_win32\\chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.get("https://www.spicejet.com/");

driver.findElement(By.*id*("highlight-addons")).click();

Actions action = new Actions(driver);

Thread.*sleep*(3000);

action.moveToElement(driver.findElement(By.*linkText*("Hot Meals"))).build().perform();

driver.findElement(By.*linkText*("Hot Meals")).click();

}

}

**Drag and Drop**

Again using Actions class the drag and drop can be perform. Same as above create an object of Actions class pass the driver. There is a method clickAndHold() and then use moveToElement() and release() for release the element.

public class DragDrop {

public static void main(String[] args) {

System.*setProperty*("webdriver.chrome.driver","C:\\Users\\akkip\\Download s\\chromedriver\_win32\\chromedriver.exe");

WebDriver driver = new ChromeDriver();

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

driver.manage().deleteAllCookies();

driver.get("https://jqueryui.com/droppable/");

driver.switchTo().frame(0); // Only if the frame is found.

Actions action = new Actions(driver);

action.clickAndHold(driver.findElement(By.*xpath*("//\*[@id=\"draggable\"]")))

.moveToElement(driver.findElement(By.*xpath*("//\*[@id=\"droppable\"]")))

.release().build().perform();

}

}

**Implicitly Wait**

Sometimes webpages took long time to load and to show the element of the page hence in this case we have to manage the wait as per the requirements. Hence we have a method called ImplicitlyWait(seconds, Timeout.seconds) & we have to pass to values such as seconds, hours, days and timeout format i.e. seconds, hours, days respectively. This is global wait

driver.manage().timeouts().pageLoadTimeout(40, *TimeUnit*.***SECONDS***);

driver.manage().timeouts().implicitlyWait(30, *TimeUnit*.***SECONDS***);

it is also known as dynamic wait. As we are giving suppose 40 sec of wait to load the page however, if the page is load earlier than given time it will not stop and the next remaining seconds will be skip.

**Explicitly wait**

Explicit wait is also dynamic if the element is clickable in 5sec then it will ignore the remaining wait which we have given to that particular element. This wait is applicable for single element.

By creating a method, we call this method whenever you want to given explicit wait.

public static void clickon(WebDriver driver, WebElement locator, int timeout) {

new WebDriverWait(driver, timeout).ignoring(StaleElementReferenceException.class).until(ExpectedConditions.*elementToBeClickable*(locator));

locator.click();

}

In the above example, we are passing three parameters as driver, locator and time.

**Dynamic Xpath (Custom Xpath)**

Sometimes if there is no attributes available in HTML source like id, name etc. then we have to create our own xpath using few steps.

Xpath should always starts with // slash. After this use HTML tag of that element then in [] we have to like this [@name\_of\_attribute=’value’] hence this will become customize xpath. Value should always written in single quotes ‘value’ like this.

Another method is using contains function.

Xpath – using contains function

//HTML\_tag[contains(text(),’value’)]

//HTML\_Tag[contains(@attribute,’value’)]

Sometimes webpage changes the id attributes when the page is getting refresh. Such as: id=’Test\_123’ then it changes again and again like this id=’Test\_456’ and id=’Test\_789’ so on.

In this case, we can use contains function for dynamic id

Xpath- //HTML\_Tag[contains(@id,’Test\_’)].

Hence the above xpath will get the initial value of that particular attribute even if the remaining part is changes. In this cases, we have understand the pattern of that particular id/ attribute.

There is another method to handle this situation which is **starts-with function**

Xpath- //HTML\_tag[starts-with(@id,’Test\_’)]

Sometimes there might another type of id like id=’test\_test\_123\_test’. Here the numbers are getting change here. So, we can use starts-with.

//HTML\_tag[starts-with(@id,’test\_test\_’)]

It will ignore the remaining parts of the id attribute. And always remember we have to use qama to after the id. [starts-with(@id,’test\_test\_’)] like this.

Sometimes we get another type of id such as id=’123\_test\_t’ and when you refresh the page the id will be change like this id=’456\_test\_t’. so it’s a dynamic id always changes whenever the is refreshed. To overcome this we have another function, ends-with.

//HTML\_tag[ends-with(@id,’\_test\_t’)]

Here, it will ignore the initial dynamic numbers of the id value. So this all function should used for dynamic id and the ids are some kind of pattern.

Selenium provides one more xpath which mostly use only for links

So, we know that all the links represented by <a> HTML tag.

Xpath - //a[contains(text(),’My Account’)]

Here, text() is a function provided by selenium webdriver and ‘My Account’ is a linktext. Always remember we cannot use = sign in this kind of xpath. While using contains function we have to use qama , after text() function

If we do not want to use contains() function then the xpath will as below:

We have to use equal sign:

//a[text()=’My Account’]

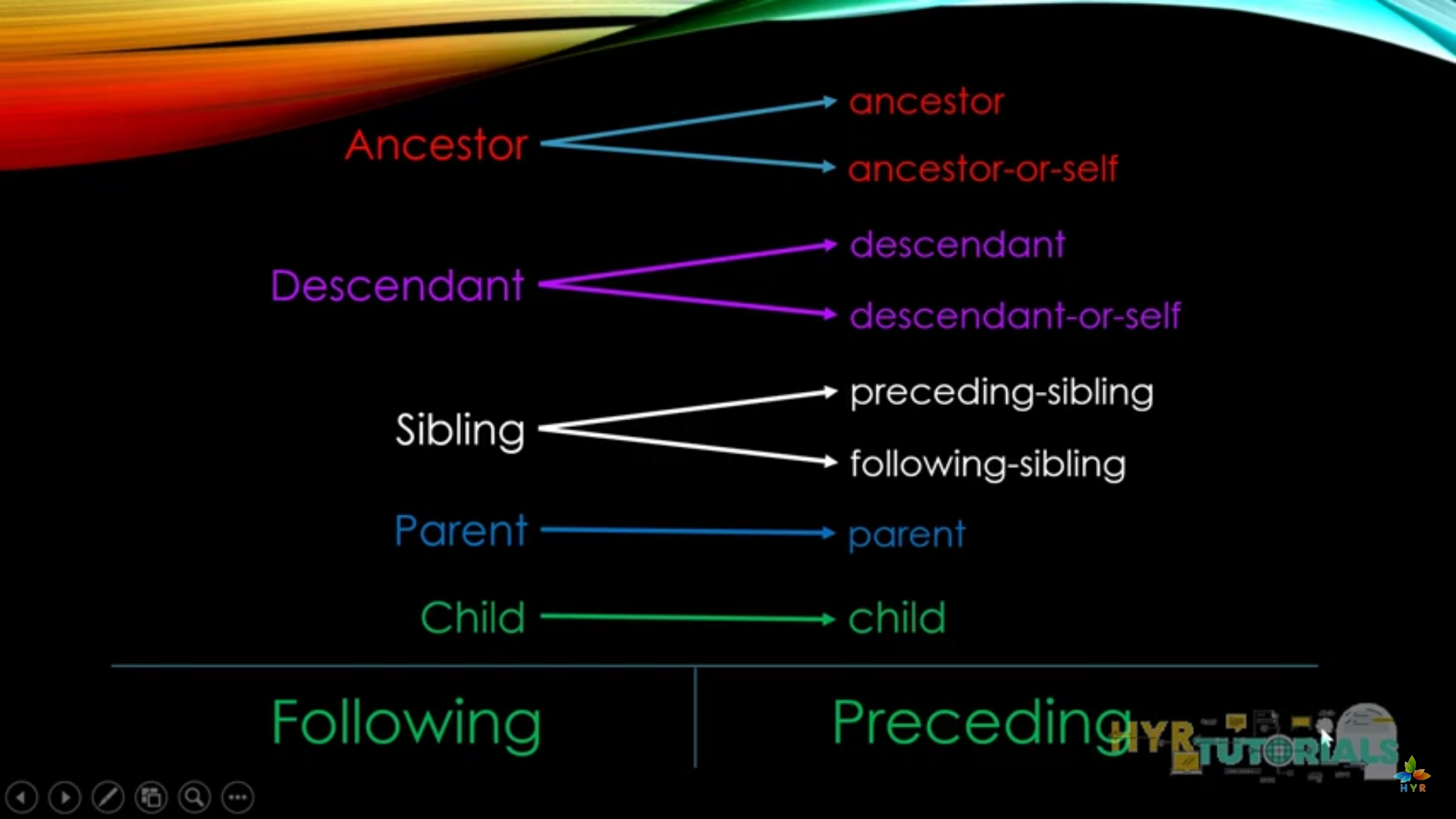
**Customize Xpath’**

In the below xpath we have used and operator to make the xpath vary robust.

//a[@class='link' and @id='ctl00\_HyperLinkLogin']

**Xpath Axes methods**

There are total five method of axes which as Ancestor, Descendant, Sibling, Parent, Child.



**What is findElements() method.?**

This method is used to get the counts of links, buttons, inputs etc from the webpage. To perform this action we have to use list class.

List<WebElement>

Create an object of list class and use findElements() method to get all the elements on the current webpage by using Tagname method.

To get the count of elements print the object followed by size() method.

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

public class FindElements {

public static void main(String[] args) {

System.*setProperty*("webdriver.chrome.driver", "C:\\Users\\akkip\\Downloads\\chromedriver\_win32\\chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.get("https://www.amazon.in/");

driver.manage().deleteAllCookies();

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

driver.manage().timeouts().pageLoadTimeout(40, *TimeUnit*.***SECONDS***);

driver.manage().timeouts().implicitlyWait(30, *TimeUnit*.***SECONDS***);

List<WebElement> linkList = driver.findElements(By.*tagName*("a"));

System.***out***.println(linkList.size());

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

String lintextString = linkList.get(i).getText();

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

}

}

}

**How to handle window popups.?**

To handle browser windows there is a method getWindowHandles() hence we will get the set object from java set collection classes. And window id will be stored in the set object.

Again there is one method iterator() . window id does not have any index in which we have to use iterator object. This will point to the top of that particular set object such as below

This is a set object =>

So, above diagram have three window and their address. Now the iterator is pointing to 1st window and will get the window id of that window. Then it will go to 2nd window by using object\_name.next() method. Using this method we can get the id of each of the windows.

public class WindowHandler {

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

System.*setProperty*("webdriver.chrome.driver", "C:\\Users\\akkip\\Downloads\\chromedriver\_win32\\chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.get("https://www.seleniumeasy.com/test/window-popup-modal-demo.html");

driver.findElement(By.*xpath*("//a[contains(text(),'Follow On Twitter')]")).click();

Thread.*sleep*(2000);

driver.findElement(By.*linkText*("Like us On Facebook")).click();

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

Iterator<String> it = handler.iterator();

String parentID = it.next(); // To get parent window ID

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

String childID = it.next(); // To get child1 window ID

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

String childID2 = it.next(); //To get child 2 window ID

System.***out***.println(childID2);

driver.switchTo().window(childID);

driver.close();

driver.switchTo().window(parentID);

driver.switchTo().window(childID2);

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

}

}

**Properties File**

These files are very useful during framework design. And it stores the data on the basis of keys and values pair such as key = value

# is used to comment in properties file. It is case sensitive.

We can create a new properties file using file\_name.properties

So, there are few methods which we have to use to read the properties file. We have Properties class which help to read the files and load the files.

InputStreamClass is used to get the files in from the particular file path..

public class ReadPropFile {

static WebDriver *driver*;

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

Properties prop = new Properties();

FileInputStream ip = new FileInputStream("C:\\Users\\akkip\\eclipse-workspace\\Test\\src\\SeleniumBasic\\ObjectRepositery.properties");

prop.load(ip); //To read the property from the input byte stream

System.***out***.println(prop.getProperty("firstname"));

System.***out***.println(prop.getProperty("age"));

String firstname = prop.getProperty("lastname");

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

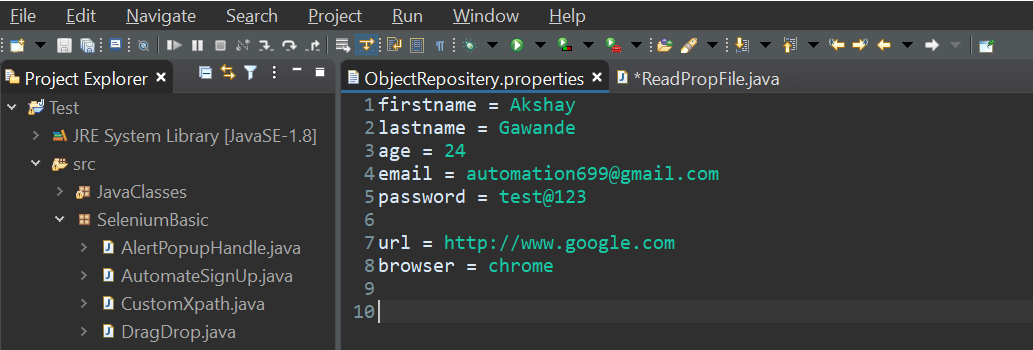
String url = prop.getProperty("url");

String browserName = prop.getProperty("browser");

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

}}

We can create properties file to store id, password, locator’s attributes, etc.



**Headless browser automation**

It means without using any browser the testing script will run and output will show on the console as per the script instructions. To use this we have to download the HTMLUnitdriver for java. And create an object using WebDriver interface.

WebDriver driver = new HtmlUnitDriver();

public class HTMLUnitDriver {

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

System.*setProperty*("webdriver.chrome.driver", "C:\\Users\\akkip\\Downloads\\chromedriver\_win32\\chromedriver.exe");

//WebDriver driver = new ChromeDriver();

WebDriver driver = new HtmlUnitDriver(); // It is known as HTML headless browser

// HTMLUnit driver is not available in version 3.x. to use this concept we have to donwload HTMLunitdriver JAR file

// also known as ghost driver as it's all working is behind the scean --no browser will lunch.

//very fast execution of test cases and high performance

//there are multiple drivers for different languages

driver.manage().timeouts().implicitlyWait(30, *TimeUnit*.***SECONDS***);

driver.get("http://www.amazon.com/");

System.***out***.println("main page "+driver.getTitle());

driver.findElement(By.*id*("nav-link-accountList-nav-line-1")).click();

System.***out***.println("login page "+driver.getTitle());

System.***out***.println(driver.getCurrentUrl());

}

}

**What is JavaScript executer.?**

To highlight the specific thing or element we can use JavaScript executer. To use this we will create a maethod named flash(). JavaScriptExecuter is a class available in selenium to handle javaScript code. Then we have to caste the driver into JavaScript executer and using the object of that javaScript executer we will use executeScript() method to execute the JavaScript code.

public static void flash(WebElement element, WebDriver driver) {

JavaScriptExecuter js = ((JavaScriptExecuter) driver); //Casting JS driver in JS executer

String bgcolor = element.getCssValue("backgroundColor");//To get color

for(int i=0; i<10;i++) {

*changcolor*("rgb(0,200,0)", element, driver);

*changcolor*(bgcolor, element, driver);

}}

public static void changcolor(String color, WebElement element, WebDriver driver) {

JavaScriptExecuter js = ((JavaScriptExecuter) driver);

//js.executeScript("argument[0].style.backgroundColor= '"+color+"'", element);

}

**Difference between:**

**IsDisplayed() V/S isEnabled() V/S isSelected()**

* isDisplayed() is the method used to verify the presence of a web element within the web page. The method returns a “true” value if the specified web element is present on the web page and a “false” value if the web element is not present on the webpage.
* isDisplayed() is capable to check for the presence of all kinds of web elements available.
* isEnabled() is the method used to verify if the web element is enabled or disabled within the web page.
* isEnabled() is primarily used with buttons.
* isSelected() is the method used to verify if the web element is selected or not.
* isSelected() method is predominantly used with radio buttons, dropdowns and checkboxes.

**How to handle a Dynamic Web table in Selenium.?**

There are two ways of handling Web Table:

Method – 1:

* Iterate row and column and get the cell value.
* Using for loop
* Get total rows and iterate table
* Put if (string matches) then select the respective checkbox
* Lengthy method

Method – 2:

* Using custom XPath
* Using parent and preceding-sibling tags
* No need to write for loop
* No full iteration of table
* Single line statement
* More dynamic
* Efficient and fast

We can handle calendars using same method.

**What is Page Object Model.? (POM)**

is a design pattern, popularly used in test automation that creates Object Repository for web UI elements. The advantage of the model is that it reduces code duplication and improves test maintenance.

Under this model, for each web page in the application, there should be a corresponding Page Class. This Page class will identify the WebElements of that web page and also contains Page methods which perform operations on those WebElements. Name of these methods should be given as per the task they are performing, i.e., if a loader is waiting for the payment gateway to appear, POM method name can be waitForPaymentScreenDisplay().

**Advantages of POM**

1. Page Object Design Pattern says operations and flows in the UI should be separated from verification. This concept makes our code cleaner and easy to understand.
2. The Second benefit is the object repository is independent of test cases, so we can use the same object repository for a different purpose with different tools. For example, we can integrate Page Object Model in Selenium with TestNG/JUnit for functional[Testing](https://www.guru99.com/software-testing.html)and at the same time with JBehave/Cucumber for acceptance testing.
3. Code becomes less and optimized because of the reusable page methods in the POM classes.
4. Methods get more realistic names which can be easily mapped with the operation happening in UI. i.e. if after clicking on the button we land on the home page, the method name will be like 'gotoHomePage()'.

**What is Page factory in Selenium.?**

**Page Factory in Selenium** is an inbuilt Page Object Model framework concept for Selenium WebDriver but it is very optimized. It is used for initialization of Page objects or to instantiate the Page object itself. It is also used to initialize Page class elements without using "FindElement/s."

**AjaxElementLocatorFactory**

**AjaxElementLocatorFactory** is a lazy loading concept of PageFactory in Selenium. It is used to find the web elements only when the elements are used in any operation. It assigns a timeout for WebElements to the object page class. One of the key advantages of using the pattern PageFactory in Selenium is AjaxElementLocatorFactory Class.

Here, when an operation is performed on an element the wait for its visibility starts from that moment only. If the element is not found in the given time interval,[Test Case](https://www.guru99.com/test-case.html)execution will throw 'NoSuchElementException' exception.

**How to find broken links and images.?**

There are four steps to get the broken links and images on the web pages as follow:

* Get the list of all the links and images in list
* Iterate the list and exclude all the links and images which does not have href attributes
* Get the size of active links and images
* Check the href url using HTTPConnection API

public class BorkenLinksAndImages {

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

System.*setProperty*("webdriver.chrome.driver","C:\\Users\\akkip\\Downloads\\chromedriver\_win32\\chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.get("https://www.amazon.com/");

//Get the list of all links and images

List<WebElement> linkList = driver.findElements(By.*tagName*("a"));

linkList.addAll(driver.findElements(By.*tagName*("img")));

System.***out***.println("all the links and images--> "+linkList.size());

List<WebElement> activList = new ArrayList<WebElement>();

// iterate linklist - exlude all the links and images which not have href attribute

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

if(linkList.get(i).getAttribute("href")!=null && (! linkList.get(i).getAttribute("href").contains("javascript"))) {

activList.add(linkList.get(i));

}

}

//Get the size of active links and images

System.***out***.println("List of Acitve links and images are --> "+ activList.size());

//check the href url, with HTTPconnection API

for(int j=0;j<activList.size();j++) {

HttpsURLConnection connection = (HttpsURLConnection)new URL(activList.get(j).getAttribute("href")).openConnection();

connection.connect();

String response = connection.getResponseMessage();

connection.disconnect();

System.***out***.println(activList.get(j).getAttribute("href")+"----> "+ response);

}

}

}

**PageFavtory Class**

**Page Factory in Selenium** is an inbuilt Page Object Model framework concept for Selenium WebDriver but it is very optimized. It is used for initialization of Page objects or to instantiate the Page object itself. It is also used to initialize Page class elements without using "FindElement/s."

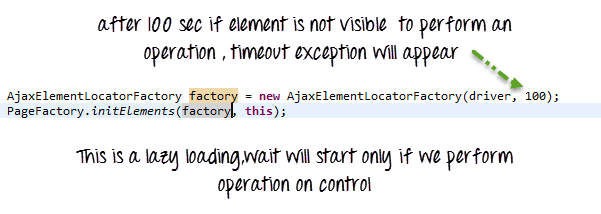
Here as well, we follow the concept of separation of Page Object Repository and Test Methods. Additionally, with the help of class PageFactory in Selenium, we use annotations **@FindBy** to find WebElement. We use initElements method to initialize web elements



**AjaxElementLocatorFactory**

**AjaxElementLocatorFactory** is a lazy loading concept of PageFactory in Selenium. It is used to find the web elements only when the elements are used in any operation. It assigns a timeout for WebElements to the object page class. One of the key advantages of using the pattern PageFactory in Selenium is AjaxElementLocatorFactory Class.

Here, when an operation is performed on an element the wait for its visibility starts from that moment only. If the element is not found in the given time interval, Test case execution will throw 'NoSuchElementException' exception.



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

We have two annotations in selenium mostly useful during page object model with PageFactory class

@FindBy annotation, it is basically used in POM to locate the webElements using initElement() method from PageFactory class.

@casheLookup annotation, is used to find the webElement from the webpage, que is how hence this annotation will create a cache memoery and it will store this element location in that memory then instead of going into html dom the locator will see the element immediately and get it.

But there are some problems while using this annotation: In case the page got refreshed then the cache memory will be corrupted or staled in this case we will get StaleElementException

**What is Web driver fire event.?**

It is basically used for testing logs and there is interface in selenium WebDriverEventListener using this interface we can create very good logs for the testing framework.

EventFiringWebDriver is a class in selenium which help to create logs. Then in TestBase class we have to create an object of WebdriverEventListener to register it with EventFiringWebDriver.