

SELENIUM

- History
- Locators
- Web elements
- Handling dropdown
- Handling popup
- Synchronization
- Java Script Executor
- Take screenshot
- Data Driven Testing
- Test NG
- POM
- Framework
- Github
- Maven
- Selenium Grids
- Jenkins





⇒ History of Selenium

Selenium was invented by **Jason** in the year 2004. It is a free and open source web based automation tool.

Selenium can be downloaded from the following website

- **URL** – www.seleniumhq.org/download/
- **File** – Selenium stand alone server (click on download link inline with 'Java' under 'Selenium client and WebDriver language bindings' section)

⇒ Flavors of Selenium

1. Selenium Core core is deprecated, earlier version before RC and WebDriver
2. Selenium IDE
3. Selenium RC (Remote control) RC is deprecated it inject js code into browser to interact and no drivers are needed eg, chromedriver,...
 - Supports non-secured websites (**http://**)
4. Selenium WebDriver there is no js code injection it directly interact with the browser with the help of browserDriver (code{java} -> webdriver interface -> chromedriver -> chromebrowser)
 - Supports both secured and non-secured websites (**http and https**)

For mobile based application

1. Selandroid – For android applications
2. Appium – For IOS, Windows and android applications

⇒ Advantages

- ✓ Free and open source
- ✓ Supports 13 languages
- ✓ Supports for multiple browsers

⇒ Disadvantages

- ✓ Supports only for web based application
- ✓ Does not support for client server and stand alone applications



⇒ Difference between Selenium and QTP

Selenium	QTP
1. Free version	1. Paid version
2. Supports 13 languages	2. Supports only 1 language i.e. VB scripting
3. Open source tool	3. Not an open source tool
4. Supports multiple browsers	4. Supports limited browsers

⇒ Required Software's

- ✓ JDK 1.8
- ✓ Eclipse (Any versions : Neon, Mars or Oxygen)
- ✓ Driver executable files
- ✓ Selenium stand alone source file

Downloads:

- **Selenium stand alone source file**

1. <https://www.seleniumhq.org/download/>
2. Click on **download** link inline with 'Java' under 'Selenium client and WebDriver language bindings' section

Note: jar file will be downloaded

- It is a compiled and compressed file consist of set of methods to perform action on the browsers

- **Driver Executable files**

1. <https://www.seleniumhq.org/download/>
2. Navigate to 'Third Party Drivers, Binding and Plugins' section
3. Click on **version no (0.20.0)** inline with "Mozilla Gecko Driver"
4. Select which file to be download (**win32 or win64**)

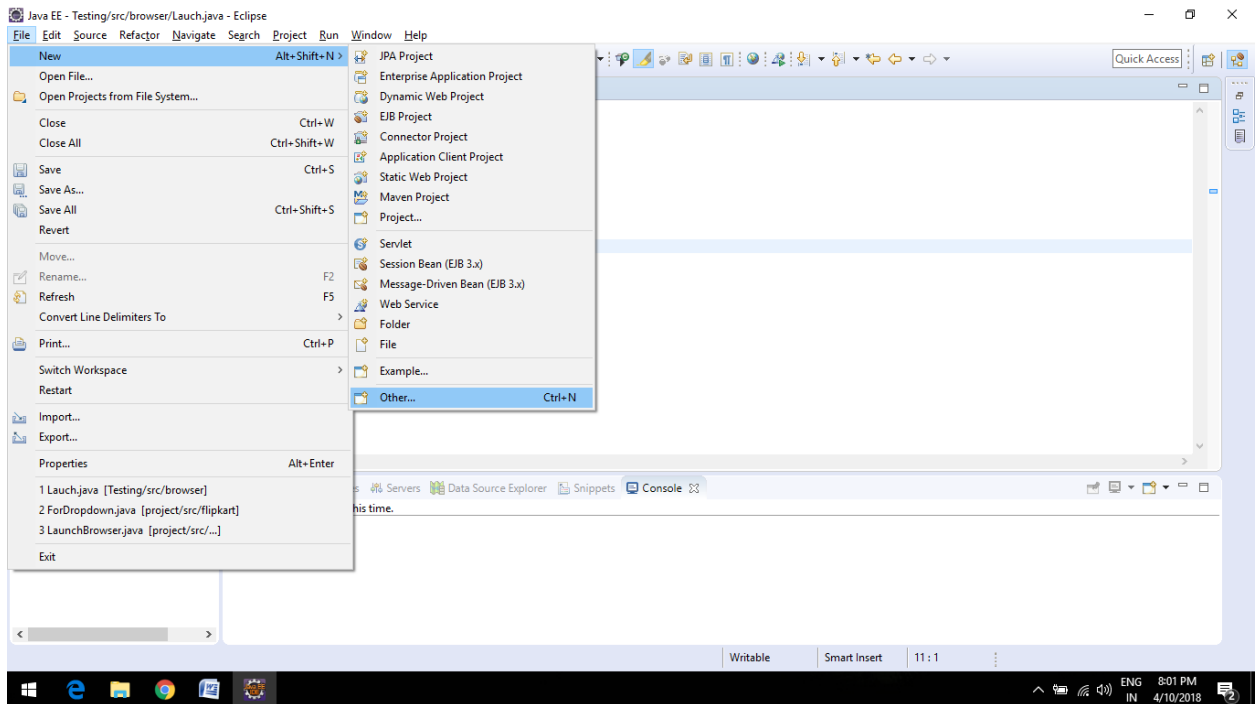
Note: zip file will be downloaded

1. Zip file consist of .exe file for **Mozilla Firefox Browser**
2. To download for **Chrome Browser** click on the version number inline with "Google Chrome Driver"

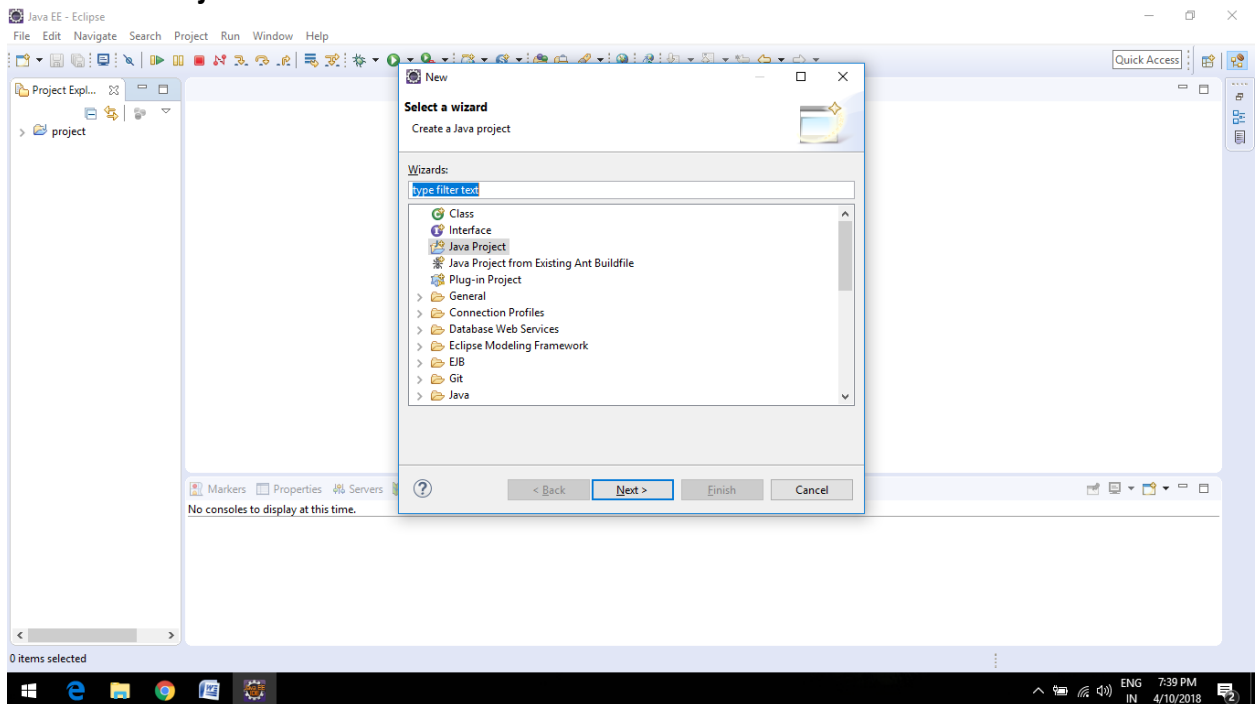


⇒ Steps to configure the downloaded files to Eclipse

1. Open Eclipse
2. Click on File→New→Other (OR ctrl + N)

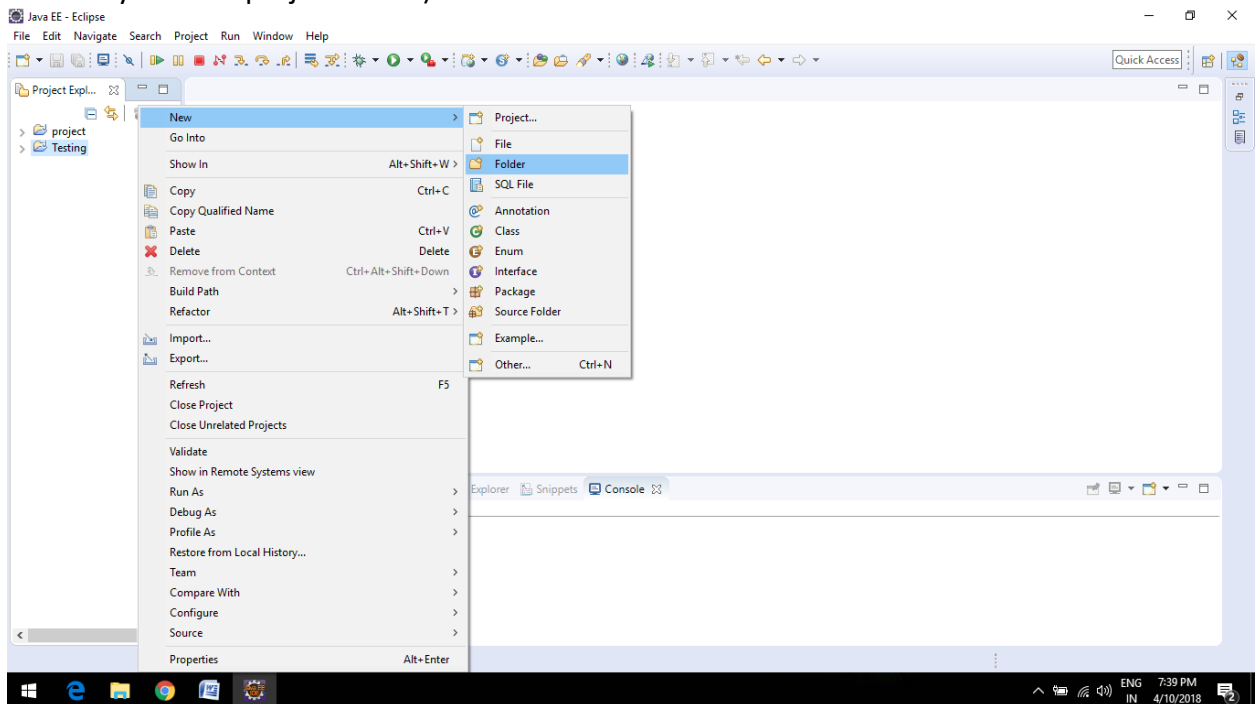


3. Select Java Project and click on 'Next' button

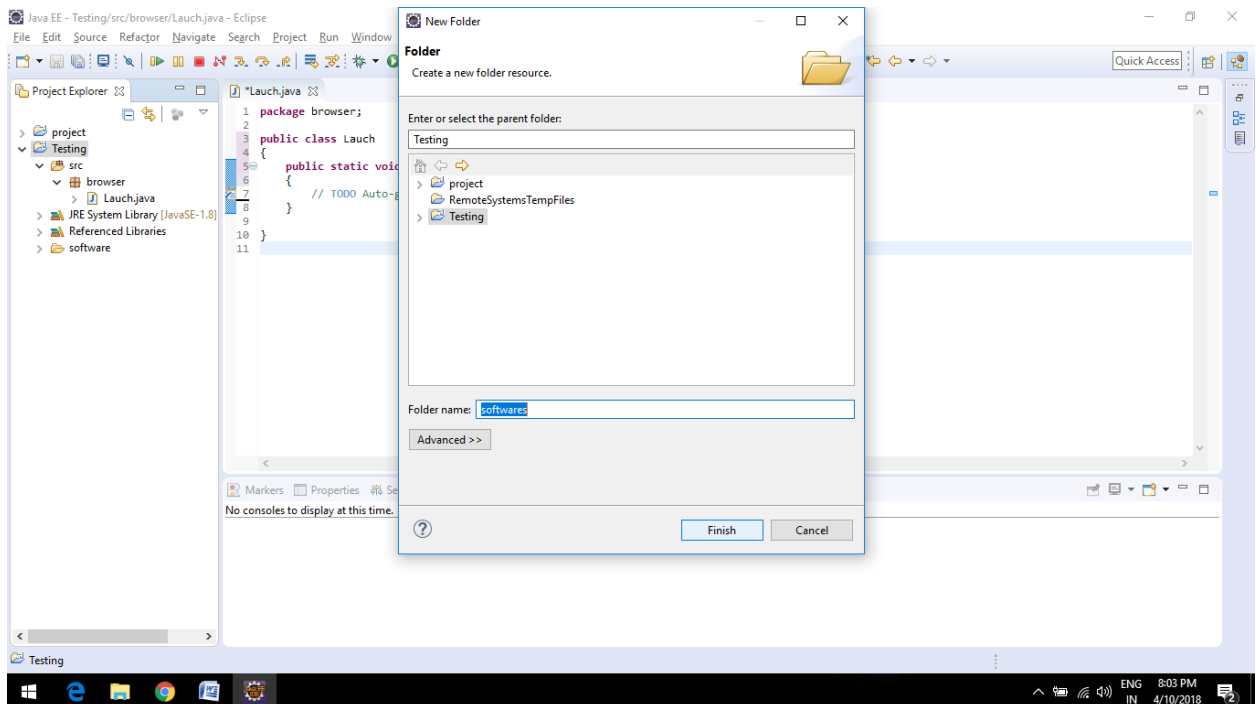




4. Enter the **Project name** click on 'Finish' button
5. Right click on the newly created project folder → New → Folder (To create one folder in the newly created project folder)

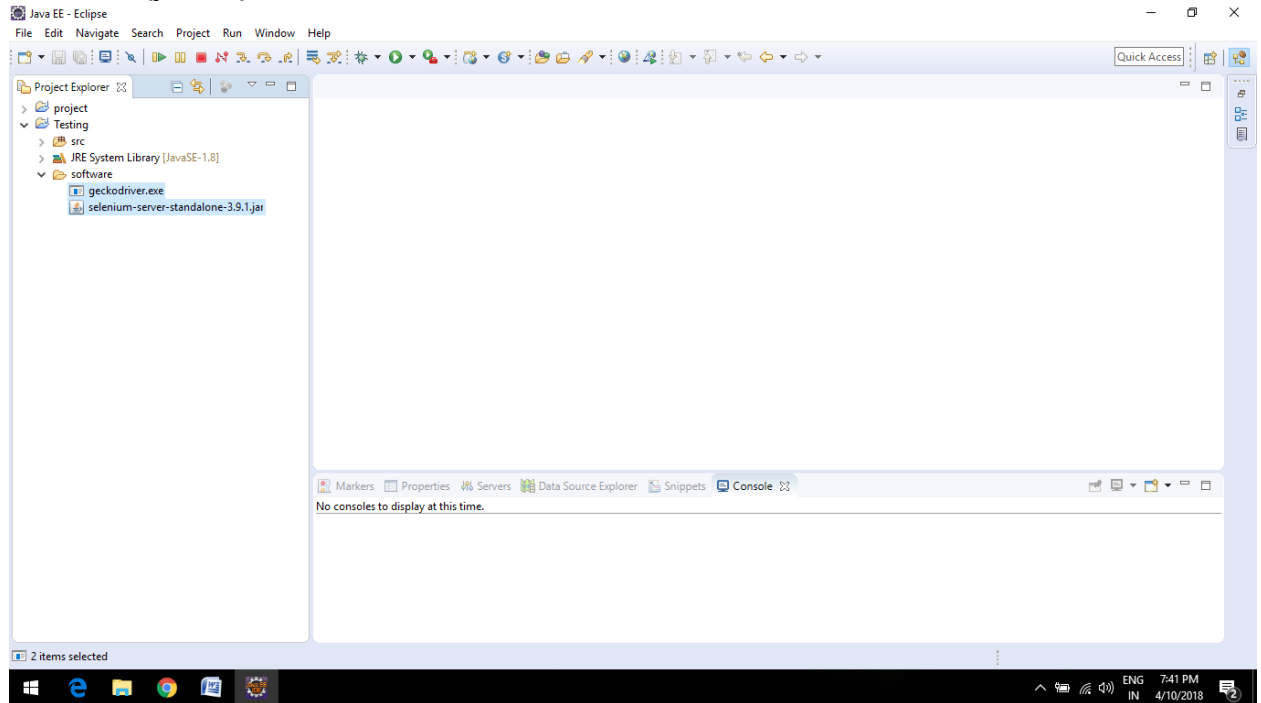


6. Enter the folder name as **Software** and click on 'Finish' button

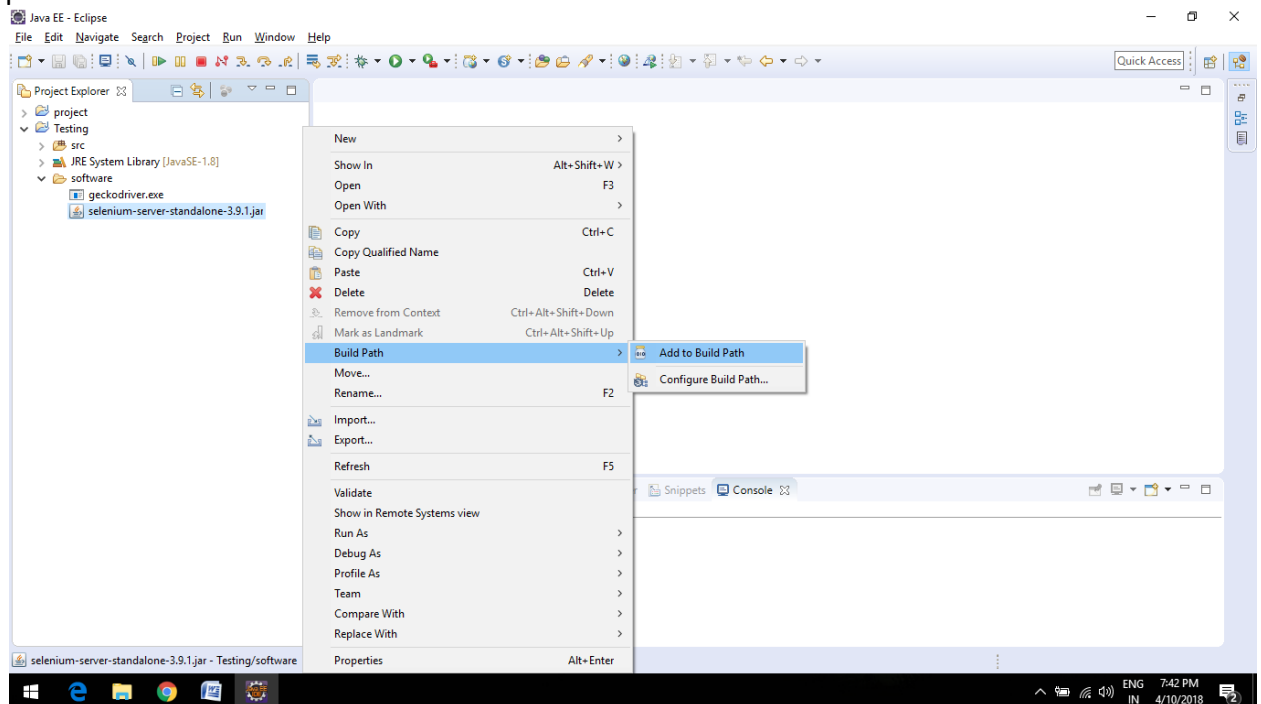




- Copy paste the downloaded files i.e. **Driver executable file (.exe file)** and **stand alone server file (jar file)**

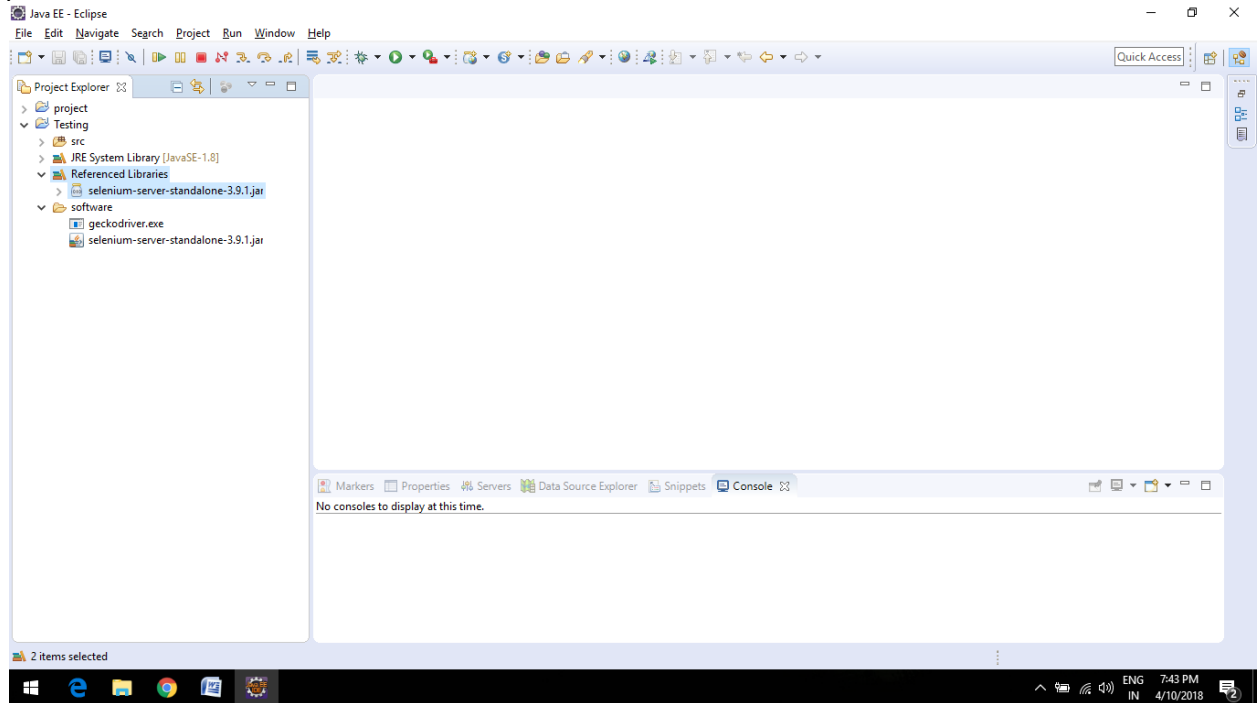


- Right click on 'Selenium-server-standalone' jar file → Build Path → click on 'Add to Build path'

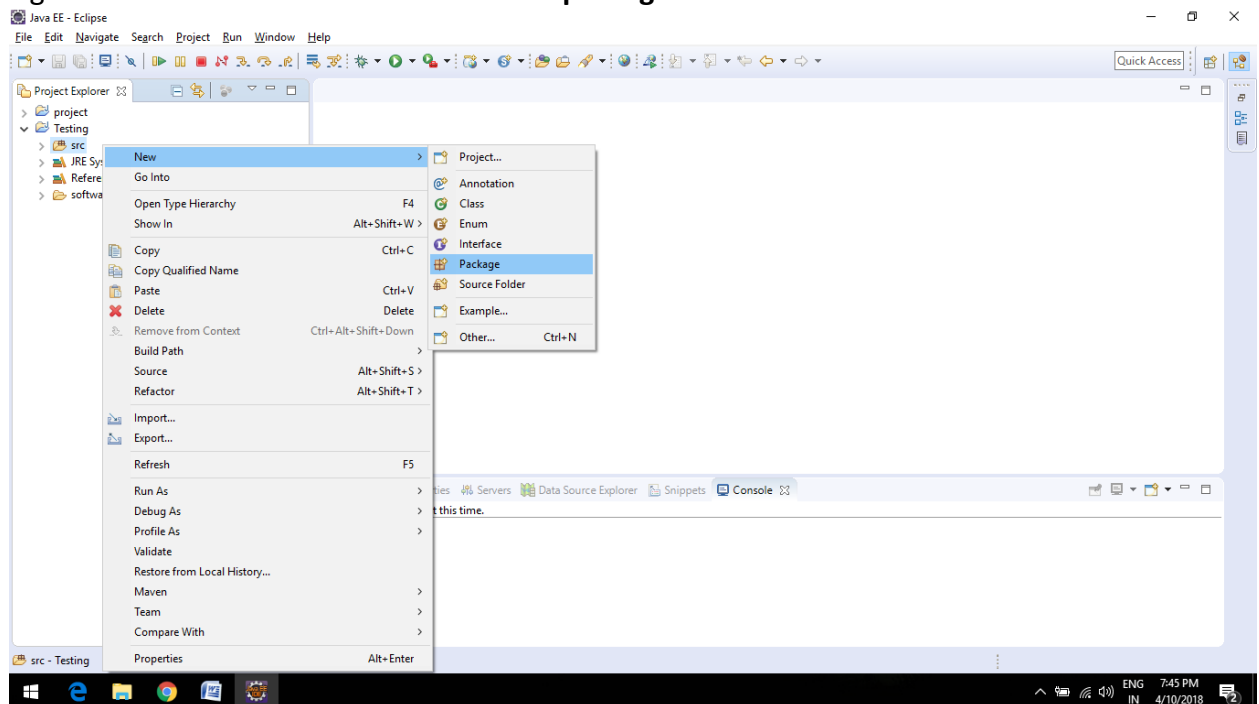




9. 'Referenced Libraries' will be created as below which consist of set of methods to perform action on browsers

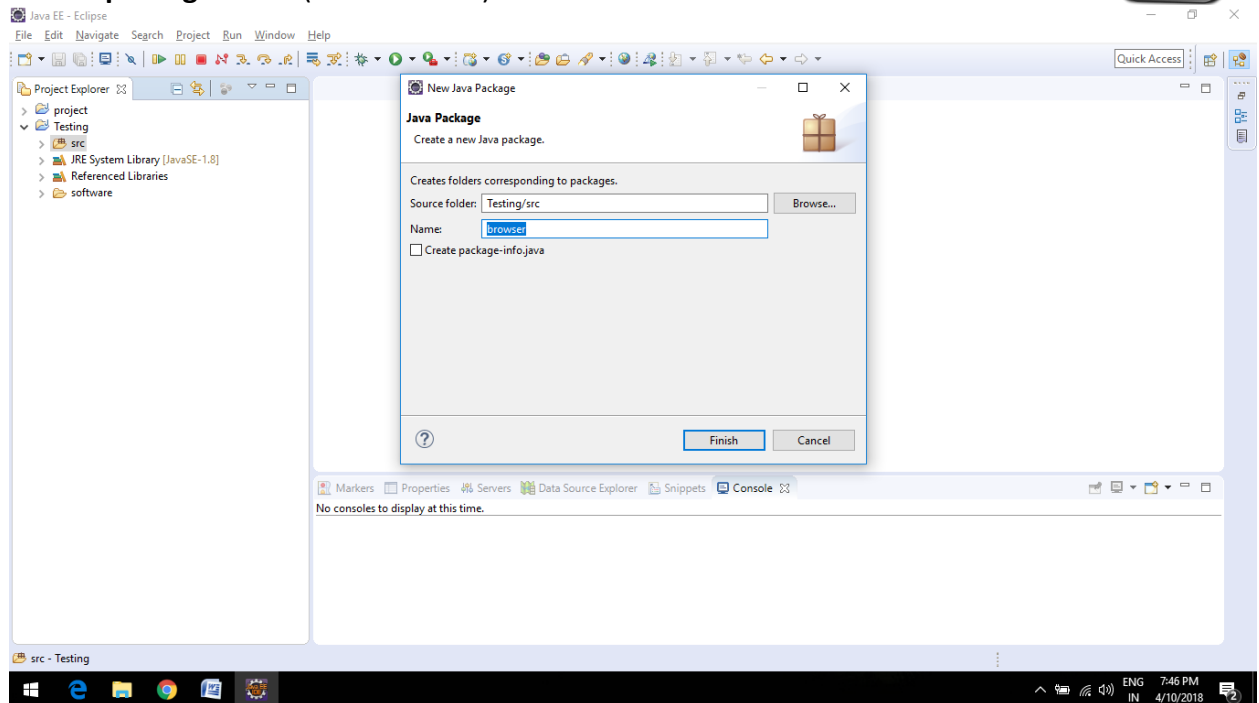


10. Right click on **src** folder → New → click on **package**

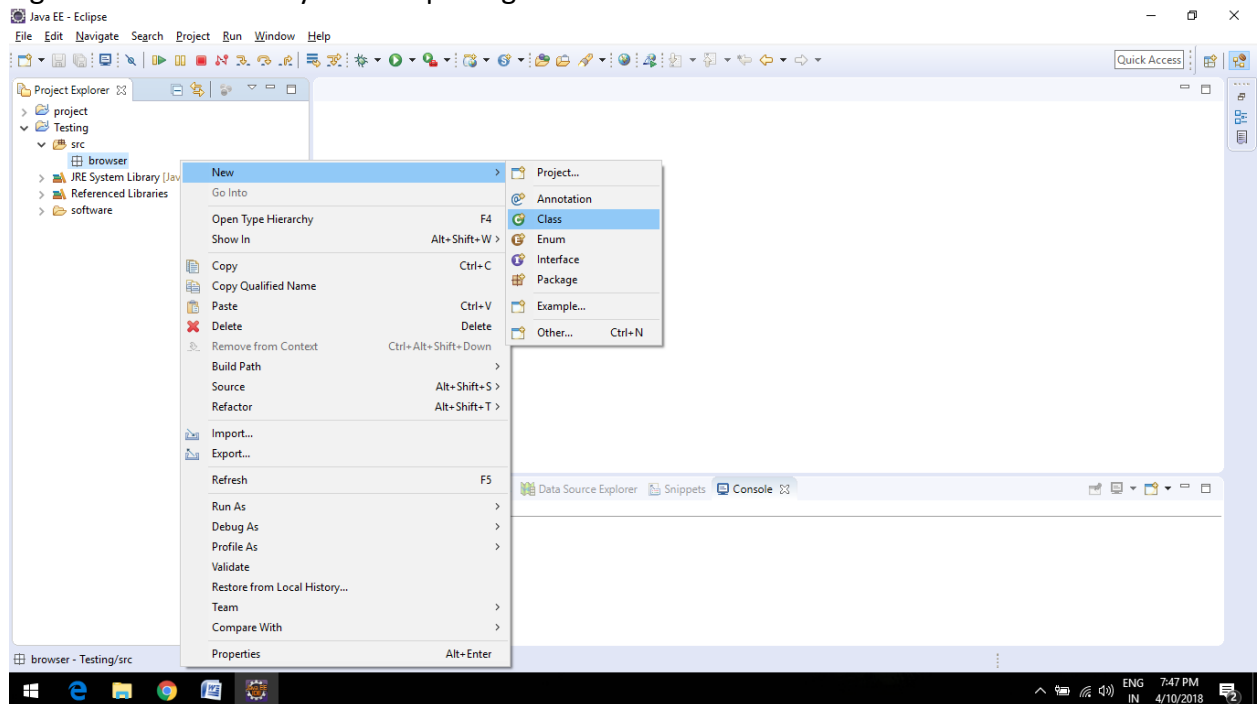




11. Enter a **package** name (in lower case) and click on 'Finish' button

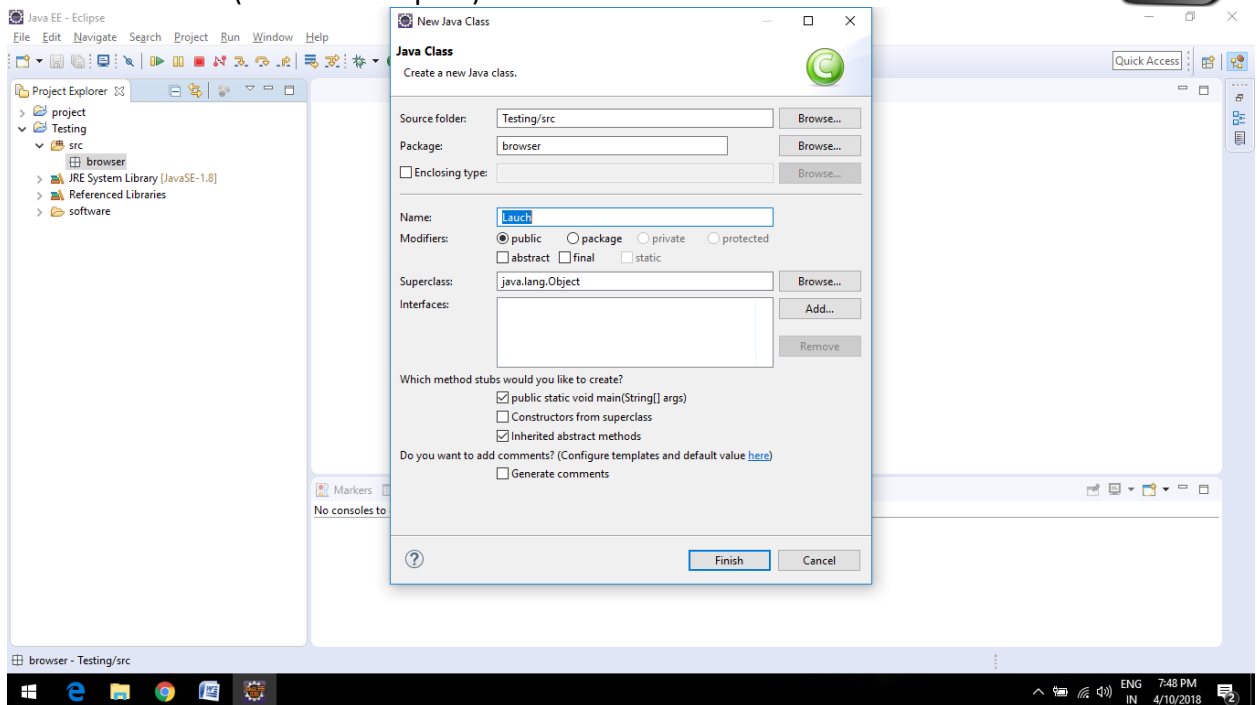


12. Right click on the newly created package → click on 'Class'

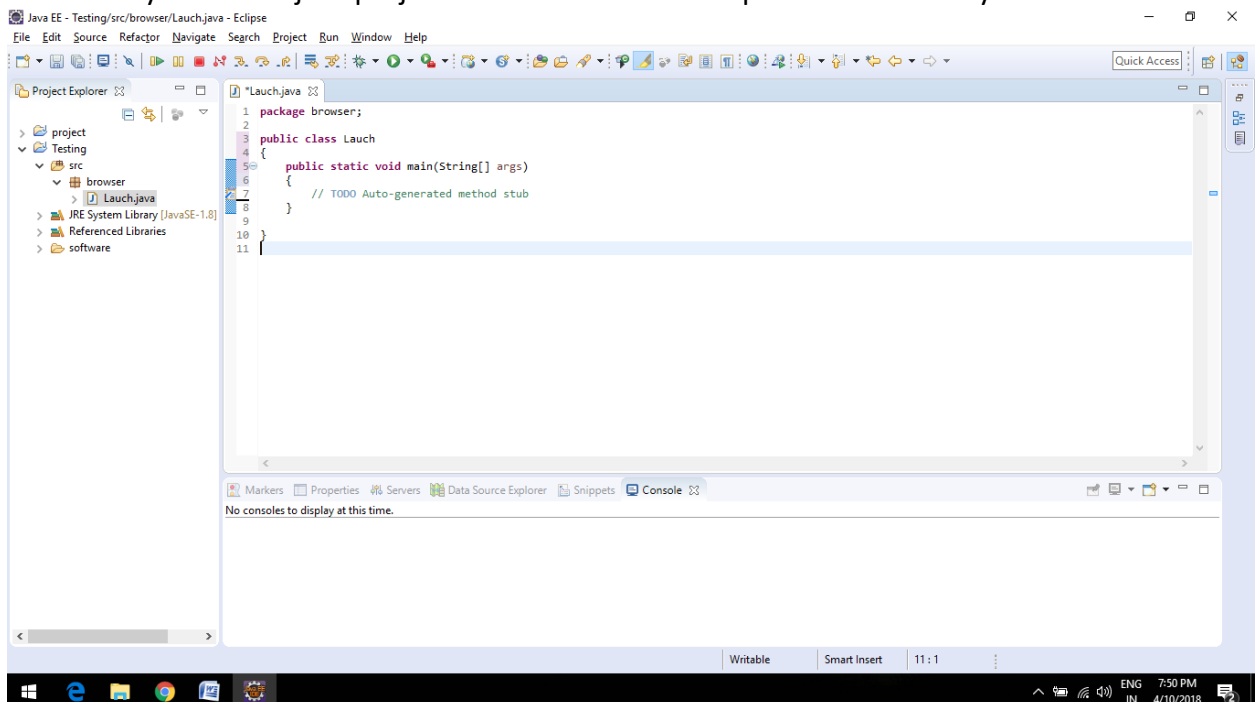




13. Enter class name (First letter capital) and click on 'Finish' button

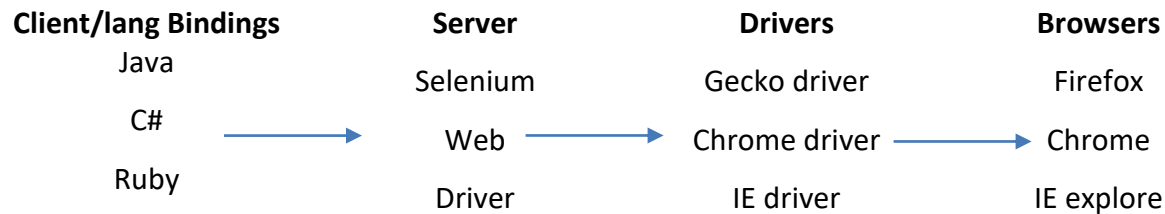


14. Successfully created a java project. You can write the script in the class body





⇒ Selenium Architecture



1. Selenium supports for multiple languages i.e. Java, C#, Ruby etc... these languages are called as Client/language bindings
2. These client bindings will communicate to server and it will perform action on browsers with the help of Drivers i.e. Gecko driver, chrome driver etc...
3. Drivers acts as a translator between Server and the Browser
4. While communicating between server and browser uses protocol called as **JSON (Java Script Object Notation)** wire protocol
5. To specify the path of driver we use **System.setProperty();**

Where, System → Concrete class and setProperty → Static method

⇒ WAP to launch empty Firefox Browser

```
class Launch
{
    public static void main(String[] args)
    {
        String key="webdriver.gecko.driver";
        String value="./software/geckodriver.exe";
        System.setProperty(key,value);
        FirefoxDriver driver = new FirefoxDriver();
    }
}
```



⇒ **WAP to launch empty Google Chrome Browser**

```
class Launch
{
    public static void main(String[] args)
    {
        String key="webdriver.chrome.driver";
        String value="./software/chromedriver.exe";
        System.setProperty(key,value);
        ChromeDriver driver = new ChromeDriver();
    }
}
```

Note:

1. To specify path we use **System.setProperty()**
2. **System** is a concrete class
3. **setProperty()** is static method of System class
4. setProperty() takes 2 arguments
 - a. key → specific type of browser (whether Firefox or chrome browsers)
 - b. value → path of driver executable file (path .exe file the browsers)
5. **Dot (.)** → represents current java project
6. **/** → traverse from parent to child folder
7. If **key or value** are invalid → throws → **IllegalStateException** (java run time exception)

⇒ **WAP to launch empty Firefox Browser and close the browser**

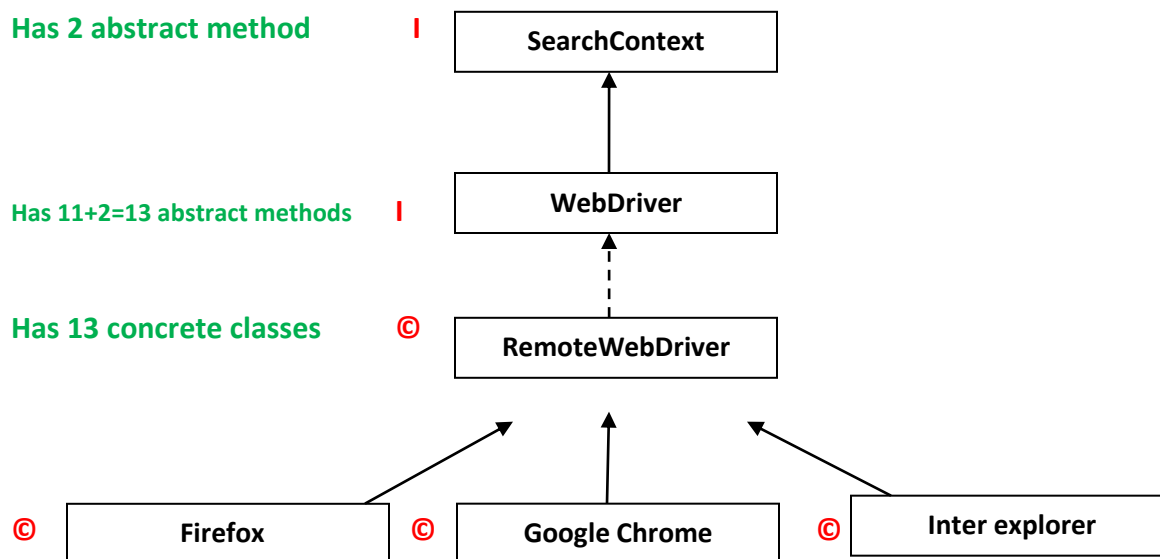
```
class Launch
{
    public static void main(String[] args)
    {
        String key="webdriver.gecko.driver";
        String value="./software/geckodriver.exe";
        System.setProperty(key,value);
    }
}
```



```
FirefoxDriver driver = new FirefoxDriver(); //To launch the browser
driver.close(); //To close specific tab
driver.quit(); //To close browser
```

quit is the dispose method of webDriver, it close all the window opened by the webdriver and terminate webdriver session

⇒ Selenium Java Architecture



1. Search context is the super most **interface** which has 2 abstract methods
2. WebDriver is an **interface** which consist of 13 abstract methods (11 of webdriver and 2 of search context)
3. Remote web driver is an **implantation class** which consist of 13 concrete methods
4. As per industry standards, we always up-cast browser to **WebDriver** interface
5. All the browsers will extends from **RemoteWebDriver** class



Example for Run time polymorphism:

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


Where,

WebDriver	→	Is an interface
driver	→	It is a reference variable
=	→	Assignment operator
new	→	Keyword & operator which creates random memory space in heap memory block
FirefoxDriver	→	It is a constructor used to launch empty firefox driver and initialize all the non static members

⇒ **WAP to launch empty Firefox Browser and enter the URL**

```
class Launch
{
    public static void main(String[] args)
    {
        String key="webdriver.gecko.driver";
        String value="./software/geckodriver.exe";
        System.setProperty(key,value);
        WebDriver driver = new FirefoxDriver(); //To launch the browser
        driver.get("https://www.google.com"); //To enter the URL
    }
}
```

Advantage of get() method

- It is used to enter URL
- It waits until the page loads completely
- In get() method, protocol (https://) is compulsory 



⇒ WAP to print title of the application

```
class Launch
{
    public static void main(String[] args)
    {
        String key="webdriver.gecko.driver";
        String value="./software/geckodriver.exe";
        System.setProperty(key,value);

        WebDriver driver = new FirefoxDriver(); //To launch the browser
        driver.get("https://www.google.com"); //To enter the URL
        String title = driver.getTitle(); // this method used to get the title of the tab
        System.out.println(title);
    }
}
```

⇒ Methods of 'Search Context' interface

In this interface we have 2 abstract methods

1. findElement(By.args())

Return type of findElement is → **WebElement** – WebDriver

2. findElements(By.args())

Return type of findElements is → **List<WebElement>** – WebDriver

⇒ Methods of 'WebDriver' interface

In this interface we have 13 abstract methods

1. close()
2. findElement(By.args())
3. findElements(By.args())
4. get(String args())
5. getCurrentUrl()
6. getPageSource ()
7. getTitle()
8. getWindowHandle()
9. getWindowHandles()



10. manage()
11. navigate()
12. quit()
13. swtichTo()

LOCATORS

- In selenium before performing any action first we need to inspect the element.
- To inspect elements we use Locators.
- Locators are static method of '**By**' class. 'By' class is a abstract class

Locators are classified into 8 types

1. tagName()
2. id()
3. name()
4. className()
5. linkText()
6. partialLinkText()
7. cssSelector()
8. xpath()

Ex: Refer the below source code to develop hyperlink

Type the below code in notepad and save the page with **filename.html** extension

```
<html>
```

```
<body>
```

```
<a id= "a1" name = "n1" class= "c1" href= https://www.google.com>Google</a>
```

```
</body>
```

```
</html>
```

→ **Write a script to click on link by using tagName() locator**

```
class Sample
```

```
{
```

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

```
    {
```

```
        String key="webdriver.gecko.driver";
```

```
        String value="./software/geckodriver.exe";
```



```
System.setProperty(key,value);

WebDriver driver = new FirefoxDriver(); //To launch the browser
driver.get("https://www.google.com"); //To enter the URL

WebElement ele = driver.findElement(By.tagName("a")); //find element by tagname
ele.click(); //To click the element/link
}
}
```

Note:

1. Return type of **findElement** is **WebElement**
2. WebElement is a **interface**
3. findElement will point to the first matching, in case of multiple webelements
4. if the specified locator is not matching with any elements, then it will throw **'NoSuchElementException'** (selenium run time exception)
5. **How to explain the below statement?**

```
driver.findElement(By.tagName("a")).click();
```

In the browser find the element with tagName as 'a' and perform click action

→ **Write a script to click on link by using id() locator**

```
class Sample
{
    public static void main(String[] args)
    {
        System.setProperty("webdriver.gecko.driver","./software/geckodriver.exe");
        WebDriver driver = new FirefoxDriver();
        driver.get("https://www.google.com");

        driver.findElement(By.id("a1")).click();
        //find element by id and perform click action
    }
}
```




→ **Write a script to click on link by using name() locator**

```
class Sample
{
    public static void main(String[] args)
    {
        System.setProperty("webdriver.gecko.driver","./software/geckodriver.exe");
        WebDriver driver = new FirefoxDriver();
        driver.get("https://www.google.com");
        driver.findElement(By.name("n1")).click();
                                //find element by name and perform click action
    }
}
```

→ **Write a script to click on link by using className() locator**

```
class Sample
{
    public static void main(String[] args)
    {
        System.setProperty("webdriver.gecko.driver","./software/geckodriver.exe");
        WebDriver driver = new FirefoxDriver();
        driver.get("https://www.google.com");

        driver.findElement(By.className("c1")).click();
                                //find element by className and perform click action
    }
}
```

→ **Write a script to click on link by using linkText() locator**

```
class Sample
{
    public static void main(String[] args)
    {
        System.setProperty("webdriver.gecko.driver","./software/geckodriver.exe");
        WebDriver driver = new FirefoxDriver();
        driver.get("https://www.google.com");

        driver.findElement(By.linkText("Google")).click();
                                //find element by linkText and perform click action
    }
}
```



→ **Write a script to click on link by using partialLinkText() locator**

```
class Sample
{
    public static void main(String[] args)
    {
        System.setProperty("webdriver.gecko.driver","./software/geckodriver.exe");
        WebDriver driver = new FirefoxDriver();
        driver.get("https://www.google.com");

        driver.findElement(By.partialLinkText("Goo")).click();
        //find element by partialLinkText and perform click action
    }
}
```

Note:

- linkText and partialLinkText are case sensitive
- These 2 locators are used only in case of hyperlinks

→ **CSS Selector**

It is a locator and it is an expression

Syntax:

tag[Attribute_Name= 'Attribute_Value']

Example:

a[id='a1']

where,

a → tag name

id → attribute name

a1 → attribute value

- To verify css expression in Firefox browser we use **Firepath**
- To verify css expression in Chrome browser we use **Developer tool bar**

Steps to download Firebug and Firepath

1. Open Firefox browser
2. Click Tools → click 'Add-ons'
3. In the 'Search all add-ons' text box enter **Firebug**
4. Install the Firebug



5. In the 'Search all add-ons' text box enter **Firepath**
6. Install the Firepath

Steps to verify css expressions in Fire fox

1. Open page in fire fox browser
2. Click on Firebug OR click F12
3. Click on 'Firepath' → select 'css' in dropdown
4. Type css expression (a[id='a1']) and click enter
5. Notice that it will display source code and matching element will be highlighted

Steps to verify css expressions in chrome browser

1. Open page in chrome browser
2. Click on F12
3. Press Ctrl+F and type css expression (a[id='a1']) and click enter
4. Notice that it will display source code only
5. Mouseover the source code it will highlight the element

Note:

In css expression # represents id and **Dot (.)** represents class

Example: a#a1, a.c1

→ Write a script to click on link by using css selector() locator

```
class Sample
{
    public static void main(String[] args)
    {
        System.setProperty("webdriver.gecko.driver", "./software/geckodriver.exe");
        WebDriver driver = new FirefoxDriver();
        driver.get("https://www.google.com");

        driver.findElement(By.cssSelector("a[id='a1']")).click();

        //find element by css selector and perform click action
    }
}
```

```
}
}
```

X-path

X-path is a path of the element in HTML tree structure. In x-path we have 2 types,

1. Absolute x-path / x-path by position
2. Relative x-path / x-path by attributes

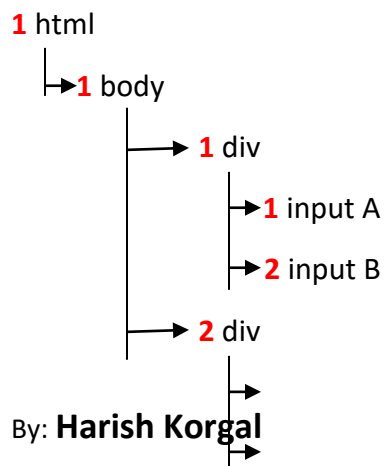
⇒ Absolute x-path

In this we use single slash '/' to traverse from parent to immediate child element

Example: consider the below source code

```
<html>
  <body>
    <div>
      <input type= "text" value= "A">
      <input type= "text" value= "B">
    </div>
    <div>
      <input type= "text" value= "C">
      <input type= "text" value= "D">
    </div>
  </body>
</html>
```

In x-path index starts from 1





1 input C

2 input D

Traverse to the element	X-Path
A	html/body/div[1]/input[1]
B	html/body/div[1]/input[2]
C	html/body/div[2]/input[1]
D	html/body/div[2]/input[2]
AB	html/body/div[1]/input
CD	html/body/div[2]/input
AC	html/body/div/input[1]
BD	html/body/div/input[2]
ABCD	html/body/div/input
AD	html/body/div[1]/input[1] html/body/div[2]/input[2]
BC	html/body/div[1]/input[2] html/body/div[2]/input[1]

Note:

1. To combine multiple x-path we use pipeline (|) operator
2. For n number of x-path we use n-1 pipeline operator

⇒ **Relative X-path** starts from the node of your choice - it doesn't need to start from the root node

In order to reduce the length of x-path expression we use relative x-path.

In relative x-path we use double slash “//” to traverse from parent to any child element. Where “//” indicates traverse from parent to any child or immediate child.



Traverse to the element	X-Path
A	<code>//div[1]/input[1]</code>
D	<code>//div[2]/input[2]</code>
ABCD	<code>//div/input</code>
AD	<code>//div[1]/input[1] //div[2]/input[2]</code>

Note:

1. To find number of elements in any web page we use the below tag for x-path

For images → `//img`

For dropdowns → `//select`

For tables → `//table`

For links and images → `//a | //img`

2. `//a` → all the matching links and `//a[1]` → all the links which has index as 1

`//a[1]`: Selects the first `<a>` tag in the entire document. if `<a>` tag present in nested it target the first `<a>` tag as well.

→ **X-path by attributes**

In X-path we can use attribute in both absolute and relative x-path.

Syntax:

`tag[@Attribute_Name= 'Attribute_Value']`

Example:

`//a[@id= 'a1']`

@ → it will inspect the element in all the direction within the same time

→ **Using multiple attributes in X-path**

To combine multiple attributes we use **and** operator

Syntax:

`tag[@Attribute_Name= 'Attribute_Value' and @Attribute_Name= 'Attribute_Value']`

Example:

`//a[@id= 'a1' and @value= 'text']`

→ **X-path by text() function**



When we don't have attributes we use text() function.

Syntax:

tag[text() = 'Text_Value']

Example:

Source code - <div>Login</div>

X-path - //div[text()='Login']

Source code - <td>Java</td>

X-path - //td[text()='Java']

Note: text() function can be replaced with . (Dot) in X-path

Ex: //td[. = 'Java']

When do we go for text() function

- When there is no attributes
- If attribute is matching with multiple web elements

→ Handling partially dynamic elements

To handle partially dynamic element we use **contains()** function

Syntax:

tag[contains(text(), 'text_value')]

Example: Inbox in Gmail application (Inbox 4,934)

X-path = //a[contains(. , 'Inbox')]

<input name = 'abcdtxtUserID'>

Xpath=//input [contains (@name, 'txtUserID')]

'name' is an attribute in html, we use the contains() method in this way also

contains method use to check partial value is equal to the actual value or not

Xpath=//input[starts-with(@class, 'button')]

this method to check the beginning value is equal to the attribute 'class'

→ Handling non breakable space

Space can be of 2 ways

- a. Space bar / tab
- b. Using ' ' (non breakable space)

X-path doesn't supports for ** **. To handle we use contains() function

Example: source code to develop space between text

India **

Asia **



//span[.='india **'] --> ele found
//span[.='Asia **'] --> ele not found

X-path

//span[contains(. , 'India')] OR //span[. = 'India']

//span[contains(. , 'Asia')]

How to verify whether text contains or not

1. Verify by using text() function
2. If it return 'No matching nodes' then text contains

When do we go for contains() function

1. To handle partially dynamic elements
2. To handle non breakable space

→ Handling completely dynamic elements

To handle **completely dynamic element** we use **Independent-Dependent X-path OR X-path by traversing**

Source code:

```
<html>
<body>
  <table border= "1">
    <tbody>
      <td>1</td> <td>Pizza</td> <td>550</td>
      <td>2</td> <td>Burger</td> <td>350</td>
      <td>3</td> <td>Cake</td> <td>250</td>
    </tbody>
  </table>
</body>
</html>
```

Output of the source code

1	Pizza	550
2	Burger	350
3	Cake	250

→ X-path by traversing

Navigating from one element to another element is called as traversing. It is classified into 2 types

1. Forward traversing
2. Backward traversing



Forward traversing – Navigating from parent to child element by using / or //

Backward traversing – Navigating from child element to parent element by using /..

Steps to inspect completely dynamic elements

1. Inspect the static element
2. Traverse from static element to common parent element **i.e.** it should highlight both static and dynamic element

In the example Pizza, Burger and Coke are the **static element** and 550, 350 and 250 are the completely **dynamic elements**

1	Pizza	550
2	Burger	350
3	Cake	250

Example 1: Inspect the price of pizza

Pizza → static and independent element

550 → dynamic and dependent element

X-path - //td[.='Pizza'] /.. /td[3]

First, we navigate pizza then step back to parent (tr) after that traverse forward td[3] that is price

Example 2: Inspect the price of Burger

Burger → static and independent element

350 → dynamic and dependent element

X-path - //td[.='Burger'] /.. /td[3]

Example 3: Inspect the price of Cake

Cake → static and independent element

250 → dynamic and dependent element

X-path - //td[.='Cake'] /.. /td[3]

⇒ Siblings function

Navigating from one child element to another child element is called as siblings functions. It is classified into 2 types,

1. Following sibling
2. Preceding sibling

→ **Following sibling** – It will highlight element below/after the current element

Syntax: /following-sibling :: tag

Example: Box office collection of Bahubali movie

A	1	Bahubali	999	B
2	1		1	2



← **Preceding Sibling** **Following Sibling** →

X-path - `//td[.='Bahubali']/following-sibling :: td[1]`

→ **Preceding sibling** – It will highlight element above/before the current element

Syntax: `/preceding-sibling :: tag`

Example: To inspect element A

A	1	Bahubali	999	B
---	---	----------	-----	---

2 ← **1**
Preceding Sibling

1 → **2**
Following Sibling

X-path - `//td[.='Bahubali']/preceding-sibling :: td[2]`

To traverse from child to any parent element we use **ancestor** tag

Syntax: `/ancestor :: tag`

Example: `/ancestor :: a`

Note:

/	→	Traverse from parent to immediate child
//	→	Traverse from parent to any child
/..	→	Traverse from child to immediate parent
Sibling functions	→	Traverse from one child to another child
/ancestor	→	Traverse from child to any parent element

→ **X-path by group index**

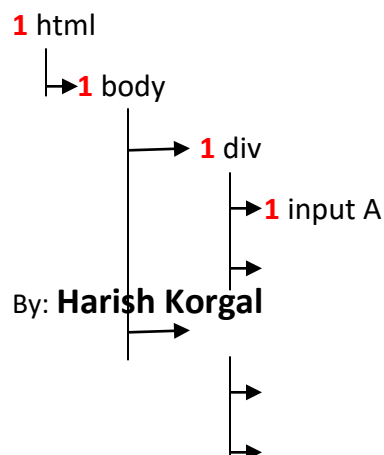
If we cannot inspect the element by using attributes, `text()` function, `contains()` function and Independent-Dependent x-path then we go for **X-path by group index**

In this we specify x-path within parenthesis and we specify index outside parenthesis

Syntax:

`(X-path)[Index]`

Example:





2 input B

2 div

1 input C

2 input D

//input	ABCD	4 matching
//input[1]	AC	2 matching
(//input)[3]	C	1 matching
(//input[1])[2]	C	1 matching
(//input)[4]	D	1 matching

(//input)[4] → It will execute parenthesis part first **i.e.** it will create one x-path array and it will store all matching element inside an array and it will specify index for individual component or element and later it will execute the index part

- **To develop 10 checkbox and below are the x-path**

```
<input type="checkbox"><br>
<input type="checkbox"><br>
<input type="checkbox"><br>
<input type="checkbox"><br>
<input type="checkbox"><br>
<input type="checkbox"><br>
<input type="checkbox"><br>
<input type="checkbox"><br>
<input type="checkbox"><br>
<input type="checkbox"><br>
```

X-path	Matching element
//input	10 (1 to 10)
(//input)[1]	1 (1)
(//input)[7]	1 (7)
(//input)[last()]	1 (10)
(//input)[last()-1]	1 (9)
(//input)[last()+1]	No matching element



<code>(//input)[position()>3]</code>	7 (4 to 10)
<code>(//input)[position()>3 and position()<8]</code>	4 (4 to 7)
<code>(//input)[position() > last()-1]</code>	1 (10)
<code>(//input)[position() mod 2=0]</code>	5 (even checkbox)
<code>(//input)[position() mod 2=1]</code>	5 (odd checkbox)

To inspect 1st and last checkbox

`(//input)[1] | (//input)[last()]`

To inspect 2nd and 8th checkbox

`(//input)[2] | (//input)[last()-2]`

Note:

Out of 8 Locators majorly we use,

1. id
2. name
3. linkText
4. Xpath
 - a. By attributes
 - b. By text() function
 - c. contains() function
 - d. Independent-Dependent xpath or siblings
 - e. Group by index

WEBELEMENT

Webelement is an **Interface** and it consists of 17 abstract methods

1. clear()
2. click()
3. findElement(By arg)
4. findElements(By arg)
5. getAttribute(String arg)
6. getCssValue(String arg)
7. getLocation()
8. getRect()
9. getScreenshotAs(OutputType<X> arg)
10. getSize()



- 11. getTagName()
- 12. getText()
- 13. isDisplayed()
- 14. isSelected()
- 15. sendKeys(CharSequence..arg)
- 16. submit()
- 17. isEnabled()

Script1 – To verify text box is displayed in page or not

Source code → `<input type= "text" value= "abc">`

```
public class Launch
{
    public static void main(String[] args)
    {
        String key="webdriver.gecko.driver";
        String value="./software/geckodriver.exe";
        System.setProperty(key,value);

        WebDriver driver = new FirefoxDriver();
        driver.get("file:///C:/Users/Harish%20Korgal/Desktop/google.html");

        WebElement ele = driver.findElement(By.xpath("//input[@value='abc']"));
        if(ele.isDisplayed())
        {
            System.out.println("Element is displayed");
        }else{
            System.out.println("Element is not displayed");
        }
    }
}
```

Script2 – To verify whether element is enabled or disabled

Source code → `<input type= "text" value= "abc" disabled>`

```
public class Launch
{
    public static void main(String[] args)
    {
        String key="webdriver.gecko.driver";
        String value="./software/geckodriver.exe";
        System.setProperty(key,value);
        WebDriver driver = new FirefoxDriver();
```



```
driver.get("file:///C:/Users/Harish%20Korgal/Desktop/google.html");
WebElement ele = driver.findElement(By.xpath("//input[@value='abc']"));
if(ele.isEnabled())
{
    System.out.println("Element is enable");
}else{
    System.out.println("Element is disable");
}
}
```

Script3 – To verify whether checkbox is selected or not

Source code → `<input type= “checkbox” checked>`

```
boolean b=ele.isSelected()
System.out.println(b);

if(ele. isSelected ())
{
    System.out.println("checkbox is checked");
}else{
    System.out.println("checkbox is unchecked ");
}
```

Script4 – To clear the default value in text box

```
ele.clear()
```

Script5 – To clear the data in text box without using clear() method

```
ele.sendKeys(Keys.CONTROL+ “a”);
ele.sendKeys(Keys.DELETE);
```

Script6 – To click link without using click() method

```
ele.sendKeys(Keys.ENTER);
```

Script7 – To copy data from 1 text box to another text box

```
WebElement ele1 = driver.findElement(By.id("t1"));
ele1.sendKeys(Keys.CONTROL+"a");
ele1.sendKeys(Keys.CONTROL+"c");

WebElement ele2 = driver.findElement(By.id("t2"));
ele2.sendKeys(Keys.CONTROL+"v");
```



Note: we can write as `"ele.sendKeys(Keys.CONTROL+"ac");"`

Script8 – To print text of the element

```
WebElement ele = driver.findElement(By.xpath("//a[.='Actitime']"));
String text = ele.getText();
System.out.println(text);
```

Script9 – To print location of the element in webpage

```
WebElement ele = driver.findElement(By.xpath("//a[.='Actitime']"));
Point p = ele.getLocation();
System.out.println(p);
```

Output: (352, 261) → x and y axis of the element

HANDLING MULTIPLE ELEMENTS

- To handle multiple elements we use **findElements()** method
- Return type of findElements() is **List<WebElement>** (collection of web elements)
- List is an **interface** and it is one of the collection type
- List should be imported from the package 'import.java.util.List';
- If the specified locator is matching with multiple web elements then findElements() will return address of all the matching elements
- If the specified locator is not matching with any of the web elements then findElements() will throw **empty list or empty array**

→ To find number of links in facebook application

```
List<WebElement> links = driver.findElements(By.tagName("a"));
int count = links.size(); //No of links in page
System.out.println(count);
```

→ To print the text of all the links in webpage

```
List<WebElement> links = driver.findElements(By.tagName("a"));
int count = links.size(); //No of links in page
```



```
System.out.println(count);

for (int i=0; i<count; i++)
{
    WebElement we=links.get(i);
    String text = we.getText();    //get the title of links
    System.out.println(text);
}
```

Using Enhanced for loop

```
for(WebElement we : links)
{
    String text = we.getText();
    System.out.println(text);
}
```

⇒ Handling auto suggestions

Scenario:

1. Open browser
2. Enter url (<https://www.google.com>)
3. Type 'Java' in search box
4. Count number of auto suggestions
5. Print text of all auto suggestions
6. Select the last auto suggestion

Script:

```
public class Launch
{
    public static void main(String[] args) throws InterruptedException
    {
        String key="webdriver.gecko.driver";
        String value="./software/geckodriver.exe";
        System.setProperty(key,value);

        WebDriver driver = new FirefoxDriver();    //step 1
        driver.get("https://www.google.com");    //step 2

        driver.findElement(By.id("//input[@id='lst-ib']")).sendKeys("Java");    //step 3
        Thread.sleep(3000);

        List<WebElement> auto =
            driver.findElements(By.className("//div[@class='sbqs_a']"));
        int count=auto.size();    //step 4
    }
}
```




```
System.out.println(count);

for(int i=0; i<count; i++)
{
    WebElement we = auto.get(i);
    String text=we.getText();
    System.out.println(text);    //step5
}

auto.get(count-1).click();    //step6
}
}
```

→ **Difference between findElement() and findElements()**

findElement()	findElements()
1. Return type is – WebElement	1. Return type is – List <WebElement>
2. If specified locator matching with multiple element, it will return the 1 st matching element	2. If specified locator matching with multiple element, it will return all the matching elements
3. If specified locator is not matching any element it will return ‘NoSuchElementException’	3. If specified locator is not matching any element it will return Empty list or Empty array

→ **Enter URL without using get() method**

To enter URL we use **navigate()** method

```
driver.navigate().to(https://www.google.com)
```

where, **navigate()** method internally calls **get()** method

Note:

1. driver.navigate().back() → to move previous page
2. driver.navigate().forward() → to move next page
3. driver.navigate().refresh() → to refresh the page

HANDLING FRAMES

Developing page inside another webpage is called as nested webpage or embedded webpage. To develop frame, developer use **<iframe>** tag



To perform action on the element which is present inside a frame, first we need to switch control from page to frame by using below statement,

driver.switchTo().frame();

Frame method is overloaded i.e.

1. frame(int arg); → using index
2. frame(String arg) → using id
3. frame(WebElement arg) → using address of the frame

If we try to use multiple frame methods for a single frame it will throw '**NoSuchElementException**' (Selenium run time exception)

To switch back the control from frame to page, we use below statement,

driver.switchTo().defaultContent();

Note: To verify web element is in frame or not

1. Right click on element
2. If it display '**This frame**' option the it is present in frame
 - a. For Chrome browser → it display as 'Frame source / Frame page source'
 - b. For IE browser → it display as 'Frame element'

Source code:

For page3.html (frame)

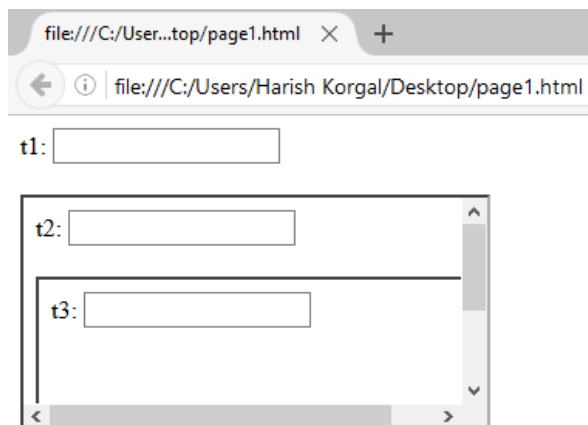
t2 : <input id= "t3" type= "text">

For page2.html (frame)

t2 : <input id= "t2" type= "text">
<iframe id= "f2" src = "page3.html">

For page1.html

T1 : <input id= "t1" type= "text">
<iframe id= "f1" src = "page2.html">



Script

```
public class Launch
{
    public static void main(String[] args) throws InterruptedException
    {
        String key="webdriver.gecko.driver";
        String value="./software/geckodriver.exe";
```



```
System.setProperty(key,value);

WebDriver driver = new FirefoxDriver();
driver.get("file:///C:/Users/Harish%20Korgal/Desktop/page1.html");

driver.findElement(By.id("t1")).sendKeys("acb"); //text box 1

driver.switchTo().frame(0); //by index
driver.switchTo().frame("f1"); //by id

WebElement text = driver.findElement(By.xpath(".*[@id='f1']"));
driver.switchTo().frame(text); //by address of the element

driver.findElement(By.id("t2")).sendKeys("abc"); //text box 2

driver.switchTo().defaultContent(); //switch back to parent page
driver.findElement(By.id("t1")).sendKeys("mno"); //text box 1

    }
}
```

Note:

1. driver.switchTo().frame() → page to frame
2. driver.switchTo().defaultContent() → default page
3. driver.switchTo().parentFrame(); → switch to immediate parent frame

HANDLING DROPDOWNS

Dropdwons can be classified into 2 types

1. Single select dropdown
2. Multiple select dropdown
 - To handle dropdown we don't have specific method in 'WebElement' or 'WebDriver'
 - To handle dropdown we go for '**Select**' class
 - 'Select' class is a **concrete** class and should be imported from

```
import org.openqa.selenium.support.ui.Select;

class Select
{
    Select(WebElement ele) //ele is a address of the dropdown
    {
```



```
}  
}
```

Select s = new Select(**ele**); //ele is a address of the dropdown

Source code to develop dropdown

```
<html>  
<body>  
    <select id="MTR">  
        <option value="i">Idly</option>  
        <option value="d">Dosa</option>  
        <option value="u">Upma</option>  
        <option value="p">Pongal</option>  
        <option value="d">Dosa</option>  
    </select><br><br><br><br>  
  
    <select id="Iyengar" multiple>  
        <option value="i">Idly</option>  
        <option value="d">Dosa</option>  
        <option value="u">Upma</option>  
        <option value="p">Pongal</option>  
        <option value="d">Dosa</option>  
    </select>  
</body>  
</html>
```

To select an option we have list of methods in 'Select' class

- a. selectByIndex(int index);
- b. selectByValue(String value);
- c. selectByVisibleText(String text);
 - In dropdown index starts from 0
 - Value and visible text are case sensitive
 - If the specified option is invalid the it will show '**NoSuchElementException**'
 - If the specified option is duplicate in single select dropdown, then it will select 1st matching value
 - If the specified option is duplicate in multiple select dropdown, then it will select all duplicate values

Note:



Select class consist of parameterized constructor which takes argument of WebElement (i.e. for the address of the dropdown as argument inside Select class constructor)

→ **WAS to select specified option using select class**

```
public class Explot
{
    public static void main(String[] args) throws InterruptedException
    {
        String key="webdriver.gecko.driver";
        String value="./software/geckodriver.exe";
        System.setProperty(key,value);

        WebDriver driver = new FirefoxDriver();
        driver.get("file:///C:/Users/Harish%20Korgal/Desktop/dropdown.html");

        WebElement ele = driver.findElement(By.id("Iyengar"));
        Select s = new Select(ele);
        s.selectByIndex(2);
        s.selectByValue("d");
        s.selectByVisibleText("Dosa");
    }
}
```

To deselect an option in 'Select' class we have below list of methods,

- a. `deselectAll();`
- b. `deselectByIndex(int index);`
- c. `deselectByValue(String value);`
- d. `deselectByVisibleText(String text);`

Note:

- If we try to use deselect methods on a single select dropdown it will throw **'UnsupportedOperationException'**



- To verify whether dropdown is single/multiple select, we use '**isMultiple()**' method

→ **WAS to select and deselect the values**

```
public class Explicit
{
    public static void main(String[] args) throws InterruptedException
    {
        String key="webdriver.gecko.driver";
        String value="./software/geckodriver.exe";
        System.setProperty(key,value);

        WebDriver driver = new FirefoxDriver();
        driver.get("file:///C:/Users/Harish%20Korgal/Desktop/dropdown.html");

        WebElement ele = driver.findElement(By.id("Iyengar"));
        Select s = new Select(ele);
        s.selectByIndex(2);
        s.selectByValue("d");
        s.selectByVisibleText("Dosa");

        if(s.isMultiple())
        {
            s.deselectByIndex(2);
            s.deselectByValue("d");
            s.deselectByVisibleText("Dosa");
        }else{
            System.out.println("Dropdown is a single select dropdown");
        }
    }
}
```

→ **WAS to count number of values in dropdown**

```
{
    WebElement ele = driver.findElement(By.id("Iyengar"));
    Select s = new Select(ele);
```



```
List<WebElement> option = s.getOptions();  
int count = option.size();  
System.out.println(count);  
}
```

→ **WAS to count number of values in dropdown, select all and deselect all**

```
{  
    WebElement ele = driver.findElement(By.id("Iyengar"));  
    Select s = new Select(ele);  
    List<WebElement> option = s.getOptions();  
    int count = option.size(); //get size  
    for(int i=0;i<count;i++)  
    {  
        s.selectByIndex(i); //select all  
    }  
    s.deselectAll(); //deselect all  
}
```

→ **WAS to print all the values in dropdown**

```
{  
    WebElement ele = driver.findElement(By.id("Iyengar"));  
    Select s = new Select(ele);  
    List<WebElement> option = s.getOptions();  
    int count = option.size(); //get size  
    for(WebElement we : option)  
    {  
        String text = we.getText();  
        System.out.println(text);  
    }  
}
```

→ **WAS to print all the values in dropdown in ascending/descending order**

```
{  
    WebElement ele = driver.findElement(By.id("Iyengar"));  
    Select s = new Select(ele);  
    List<WebElement> option = s.getOptions();  
    ArrayList<String> l = new ArrayList<String>();  
    for(WebElement we : option)
```



```
{
    String text = we.getText();
    l.add(text);
}
Collections.sort(l); //sorting
for(String t : l)
{
    System.out.println(t); //printing
}
}
```

→ **WAS to verify whether specified option is present in dropdown or not**

```
{
    WebElement ele = driver.findElement(By.id("Iyengar"));
    Select s = new Select(ele);
    List<WebElement> option = s.getOptions();
    ArrayList<String> l = new ArrayList<String>();
    for(WebElement we : option)
    {
        String text = we.getText();
        l.add(text);
    }

    if(l.contains("Idly"))
    {
        System.out.println("Option is present");
    }else{
        System.out.println("Option is not present");
    }
}
```

Note:

1. 'Select' class methods will work only if the **tag is SELECT**
2. If we try to use 'Select' class method on different tag it will throw 'NoSuchElementException' or 'UnexpectedTagName'

HANDLING MOUSE OVER ACTIONS

- To handle mouse over action we go for '**Actions**' class
- 'Action' class is a concrete class
- Action class should be imported from the package

```
import org.openqa.selenium.support.ui.Actions;
```




- Action class consist of parameterized constructor and it takes argument of address of application

class **Actions**

```
{
    Actions(WebDriver driver) //driver is a application control (or) address of the app
    {

    }
}
Actions act = new Actions(driver);
```

Methods of 'Actions' class:

1. moveToElement()
2. dragAndDrop()
3. rightClick() / contextClick()
4. doubleClick()

→ **WAS to mouse over of men and select option in flipkart**

```
public class Sample
{
    public static void main(String[] args) throws InterruptedException
    {
        String key="webdriver.gecko.driver";
        String value="./software/geckodriver.exe";
        System.setProperty(key,value);

        WebDriver driver = new FirefoxDriver();
        driver.get("https://www.flipkart.com");

        driver.findElement(By.xpath("//button[.='X']")).click();
        Thread.sleep(3000);
        WebElement ele = driver.findElement(By.xpath("//span[.='Men']"));

        Actions act = new Actions(driver); //address of the application
        act.moveToElement(ele).perform();
        Thread.sleep(3000);
    }
}
```



```
        driver.findElement(By.xpath("//span[.='T-Shirts']")).click();  
    }  
}
```

Note:

- To perform mouse over action in Actions class we use 'moveToElement()' method
- In Action class by default we should call '**perform()**' method

HANDLING DRAG AND DROP ACTION

- To handle drag and drop we use **Actions** class
- In Actions class we use '**dragAndDrop()**' method
- It takes 2 arguments (source and target)
- In Actions class by default we should use '**perform()**' method

Script:

```
{  
  
    WebElement b1 = driver.findElement(By.xpath("h1[.='Book1']"));  
    WebElement b2 = driver.findElement(By.xpath("h1[.='Book3']"));  
  
    Actions act = new Actions(driver);  
    act.dragAndDrop(b1,b2).perform();  
  
}
```

HANDLING RIGHT CLICK OR CONTEXT CLICK ACTION

- To handle right click action we go for **Actions** class
- In Actions class we use '**contextClick()**' method
- In Actions class by default we should use '**perform()**' method

Script:

```
public class Sample  
{  
    public static void main(String[] args) throws InterruptedException, AWTException
```



```
{
    String key="webdriver.gecko.driver";
    String value="./software/geckodriver.exe";
    System.setProperty(key,value);

    WebDriver driver = new FirefoxDriver();
    driver.get("https://www.seleniumhq.org/");

    WebElement link = driver.findElement(By.xpath("//li[@id='menu_download']"));

    Actions act = new Actions(driver);
    act.contextClick(link).perform();

    Robot r = new Robot();
    r.keyPress(KeyEvent.VK_W);
    r.keyRelease(KeyEvent.VK_W);
}
```

Note: VK → virtual key and W → shortcut to click

1. To handle keyboard functionality we use '**Robot**' class
2. Robot class is a concrete class
3. All the keyboard functionality are static final variables

HANDLING DOUBLE CLICK ACTION

1. To handle double click action, we use '**Actions**' class
2. In Actions class we use '**doubleClick()**' method
3. In Actions class by default we should call perform() method

Script:

```
{
    WebElement link = driver.findElement(By.xpath("//li[@id='menu_download']"));
    Actions act = new Actions(driver);
    Thread.sleep(3000);
    act.doubleClick(link).perform();
}
```

HANDLING POP-UPS

In selenium pop-ups are classified into 6 types,



1. Alert and confirmation popup
2. Hidden division popup
3. File upload popup
4. File download popup
5. Child browser popup
6. Window popup

⇒ **Alert and confirmation popup (Java script popup)**

To handle this popup we use '**Alert**' interface

Characteristics:

1. We cannot inspect the popup
2. We cannot move the popup
3. It consist of 'Ok' and 'Cancel' buttons

Note:

- To click 'Ok' button we use → **accept()** method
- To click 'Cancel' button we use → **dismiss()** method
- To switch control from page to alert popup we use,
driver.switchTo().alert();
- To return the text inside the popup we use, **.getText()** (present in Alert interface)
- If we try to use both accept() and dismiss() method for single popup, it will throw '**NoAlertPresentException**'

Script:

```
{  
  
    WebElement block = dirver.findElement(By.xpath("//input[.= 'Men']"));  
    block.click();  
  
    Alert a = driver.switchTo().alert();    //switch control to alert popup
```



```
String text = a.getText();  
System.out.println(text);
```

```
    a.accept();    OR  
    a.dismiss()  
}
```

⇒ **Hidden division popup**

Initially the popup is hidden and if we perform action on the element then it will display the popup and basically this popup was developed by using '**div**' tag.

To handle this popup we use **findElement()** method

Characteristics:

1. We can inspect the popup
2. We cannot move the popup

Script to click on the date in the calendar

```
{  
    driver.findElement(By.xpath("//i[.='Cal']")).click();  
    driver.findElement(By.xpath("//i[.='27']")).click();  
}
```

⇒ **File upload popup**

To handle this popup we use '**sendKeys()**' method. In this method we specify the path of file which has to be selected.

Characteristics:

1. We cannot inspect element
2. We can name the popup



3. It consist of 2 buttons 'Open' and 'Cancel'

Note: To copy file path we use (shift + right click on file → copy as path)

Source code to develop browser button

```
<html>
  <body>
    <form>
      Pick any file to upload: <input type= "file" name= "uploadfile" id= "upload">
    </form>
  </body>
</html>
```

Script to upload the file

```
{
    driver.findElement(By.id("upload")).sendKeys ("C:\\Users\\Harish
    Korgal\\Desktop\\dropdown.html");
}
```

⇒ File download popup

- In case of firefox browser, it will display the file download popup
- In case of chrome browser and IE browser it will not display popup
- To hand file download popup we use '**Robot**' class

Characteristics:

1. We cannot inspect the popup
2. We can move the popup
3. It consist of 2 radio buttons
 - a. Open with
 - b. Save file
4. It consist of 2 buttons
 - a. Ok
 - b. Cancel
5. To click 'Ok' button we use '**ENTER**' command
6. To click 'Cancel' button we use '**ESCAPE**' command



7. To shift control from open with → to save file we use '**Alt+S**'
8. To shift control from save file → open with we use '**Alt+O**'

Script to download a file in browser

```
public class FileDownload
{
    public static void main(String[] args) throws InterruptedException, AWTException
    {
        String key="webdriver.gecko.driver";
        String value="./software/geckodriver.exe";
        System.setProperty(key,value);

        WebDriver driver = new FirefoxDriver();
        driver.get("https://www.seleniumhq.org/download/");

        WebElement ele = driver.findElement(By.xpath("//td[.='C#']/following-sibling::td[3]"));

        ele.sendKeys(Keys.ENTER);
        Thread.sleep(3000);

        Robot r = new Robot();
        r.keyPress(KeyEvent.VK_ALT);
        r.keyPress(KeyEvent.VK_S);

        r.keyRelease(KeyEvent.VK_ALT);
        r.keyRelease(KeyEvent.VK_S);

        r.keyPress(KeyEvent.VK_ENTER);
        r.keyRelease(KeyEvent.VK_ENTER);
    }
}
```

⇒ Child browser popup

The browser re-sized to look like a popup is called as child browser popup



Characteristics:

1. We cannot inspect the popup
2. We can move the popup
3. We can minimize, maximize and close the popup

To handle child browser popup we use, **getWindowHandle()** and **getWindowHandles()** method

getWindowHandle() → retrieves only the id of the parent browser

getWindowHandles() → retrieves only the id of the parent browser and child browsers (all browsers) **it return Set<WebElement>**

windowHandle – it is a unique alpha-numeric String of the browser

Script to print the id of the parent browser

```
public class Sample
{
    public static void main(String[] args)
    {
        String key="webdriver.gecko.driver";
        String value="./software/geckodriver.exe";
        System.setProperty(key,value);

        WebDriver driver = new FirefoxDriver();
        String parent = driver.getWindowHandle();
        System.out.println(parent);
    }
}
```

Script to print id of all browsers and count number of browsers

⇒ Window popup

1. To handle window popup we use '**AutoIT**' software
2. 'AutoIT' is a free automation tool which is mainly used to handle window popup/ stand alone application
3. AutoIT can be downloaded from - <https://www.autoitscript.com/site/autoit/downloads/>

Steps to fetch the title of window popup



1. Start → All program → AutoIT V3 → AutoIT window Infor(x64)
2. Drag and drop finder tool on standalone application or window popup
3. Finder tool will display the options like 'Title', 'Class', 'Position'...

Steps to write VB script in AutoIT

1. Launch (Start → AutoIT → **SciTE script executor**)
2. Specify path of stand alone application (.exe) i.e. ("C:\Windows\System32\calc.exe")
3. Open application
4. Perform action on application
5. Close the application

Write the below script in 'SciTE script executor' application to perform action on calculator

```
Run("C:\Windows\System32\calc.exe")
```

```
WinWaitActive("Calculator")
```

```
Sleep(3000)
```

```
Send("6")
```

```
Sleep(3000)
```

```
Send("-")
```

```
Sleep(3000)
```

```
Send("4")
```

```
Sleep(3000)
```

```
Send("=")
```

```
Sleep(3000)
```

```
WinClose("Calculator")
```

1. Click on File → Save → file_name.au3
2. Click Tools → compile (It generates .exe file)

Script:

```
import java.io.IOException;
```



```
public class Sample
{
    public static void main(String[] args) throws IOException
    {
        Runtime.getRuntime().exec("C:\\Users\\Harish
                                   Korgal\\Desktop\\cal1.exe");
    }
}
```

Note:

Sl.No	Pop-ups	Solution
1.	Alert and confirmation	driver.switchTo().alert();
2.	Hidded division	Ok – accept(); and cancel – dismiss();
3.	File upload	sendKeys(path of file);
4.	File download	Robot class or AutoIT
5.	Child browser	getWindowHandle() or getWindowHandles() driver.switchTo().window(id of browser)
6.	Window popup	AutoIT

Interview Questions:

1. Minimize browser

```
Robot r = new Robot();

r.keyPress(KeyEvent.VK_ALT);
r.keyPress(KeyEvent.VK_SPACE);
r.keyPress(KeyEvent.VK_N);

r.keyRelease(KeyEvent.VK_ALT);
r.keyRelease(KeyEvent.VK_SPACE);
r.keyRelease(KeyEvent.VK_N);
```

2. Close browser (without using quit and close method)

```
Robot r = new Robot();

r.keyPress(KeyEvent.VK_ALT);
r.keyPress(KeyEvent.VK_SPACE);
r.keyPress(KeyEvent.VK_C);

r.keyRelease(KeyEvent.VK_ALT);
```



```
r.keyRelease(KeyEvent.VK_SPACE);  
r.keyRelease(KeyEvent.VK_C);
```

3. Resize the browser and maximize browser

```
Dimension d = new Dimension(300,200);  
driver.manage().window().setSize(d);
```

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

4. To drag browser to particular x and y axis

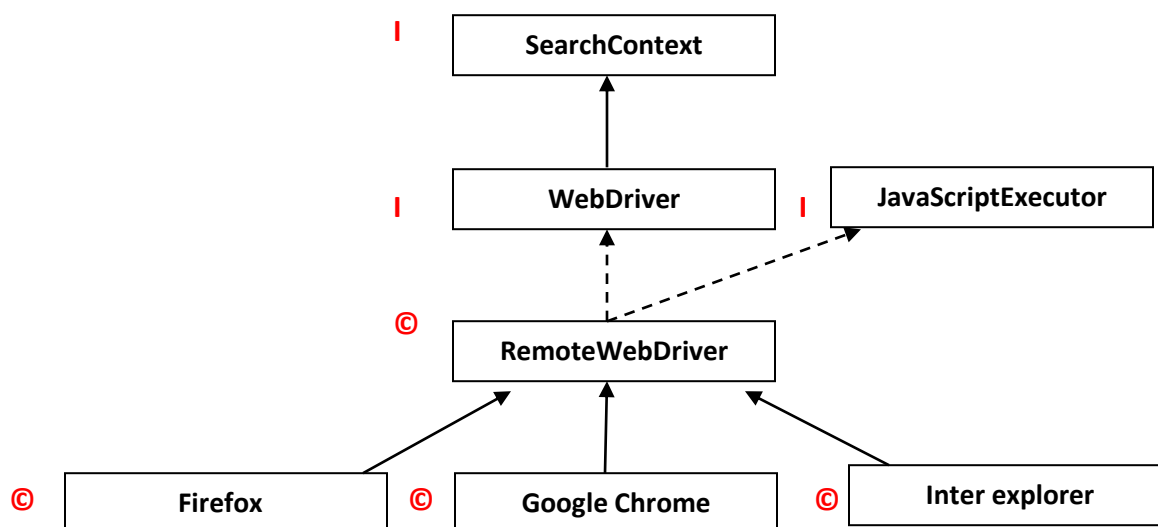
```
Point p = new Point(300,200);  
driver.manage().window().setPosition(p);
```

⇒ Java Script Executor

- To handle disabled element we use Java Script executor
- It is a **Interface**

To access Java Script Executor methods we **Type cast** from **WebDriver** to **Java script interface** (explicit type casting) i.e.

```
WebDriver driver = new FirefoxDriver();  
JavascriptExecutor js = (JavascriptExecutor) driver;
```





- Java script executor should be imported from the following package

```
import org.openqa.selenium.JavascriptExecutor;
```

Java script executor consist of **2 abstract methods**,

1. `js.executeAsyncScript(String...arg0, Object..args.arg1);`
2. `js.executeScript(String...arg0, Object..args.arg1);`

Script: To pass the data inside disabled text box without using sendKeys method

Source code: `<input id="t1" type="text" value="abc" disabled>`

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

        System.setProperty("webdriver.gecko.driver","./softwares/geckodriver.exe");

        WebDriver driver = new FirefoxDriver();
        driver.get("file:///C:/Users/Harish%20Korgal/Desktop/disabled.html");

        JavascriptExecutor js = (JavascriptExecutor) driver;
        js.executeScript("document.getElementById('t1').value('XYZ')");
    }
}
```

Note:

- document → partially implemented class
- getElementById → static method of document class
- To pass data inside text box we use **value** comment. **It will override the value**



Script to scroll down webpage for 500 pixels and scroll up for 500 pixels

```
public class Sample
{
    public static void main(String[] args) throws InterruptedException
    {
        System.setProperty("webdriver.gecko.driver", "./softwares/geckodriver.exe");

        WebDriver driver = new FirefoxDriver();
        driver.get("https://www.amazon.in/");

        JavascriptExecutor js = (JavascriptExecutor) driver;

        js.executeScript("window.scrollTo(0,500)");    //scroll down
        Thread.sleep(4000);
        js.executeScript("window.scrollTo(0,-500)");    //scroll up
    }
}
```

Script to scroll down webpage for specified element

```
public class Sample
{
    public static void main(String[] args) throws InterruptedException
    {
        System.setProperty("webdriver.gecko.driver", "./softwares/geckodriver.exe");

        WebDriver driver = new FirefoxDriver();
        driver.get("https://docs.seleniumhq.org/");

        WebElement ele = driver.findElement(By.xpath("//a[.='Selenium  
Conf 2016']"));

        Point p = ele.getLocation();
        int x = p.getX();
        int y = p.getY();

        JavascriptExecutor js = (JavascriptExecutor) driver;

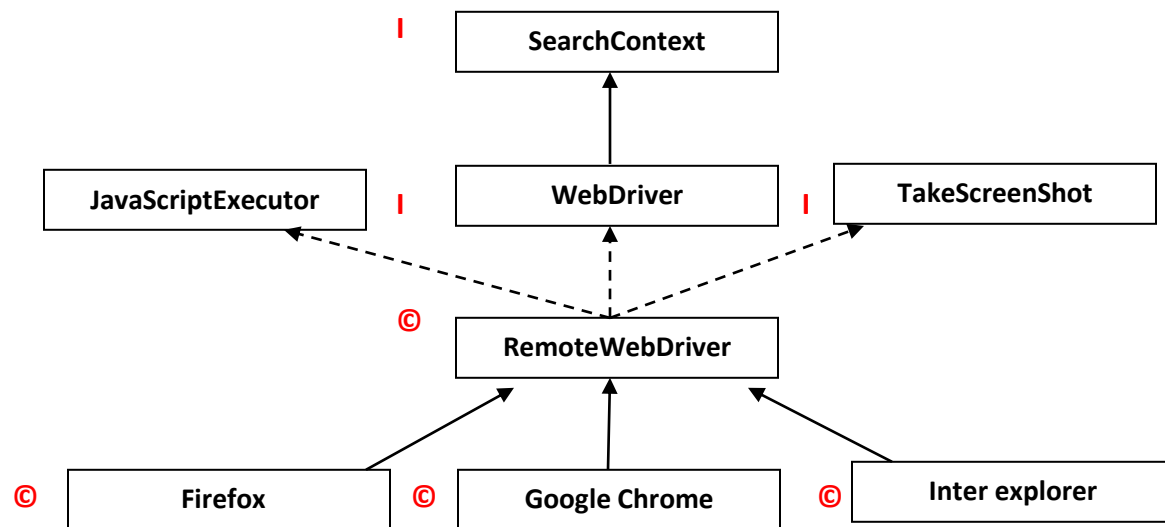
        js.executeScript("window.scrollTo("+x+","+y+")");
    }
}
```



Advantages of Java Script Executor:

1. To handle disabled element
2. To pass the data inside text box without sendKeys()
3. To clear the data inside text box without clear()
4. To scroll down and scroll up web page

⇒ Take Screen Shot:



1. Take screen shot is **Interface** consist of **1 Abstract method** (getScreenshotAs())
2. Take screen shot should be imported from the following package,

Steps to take screenshot of application:

1. Typecast from **WebDriver** to **TakeScreenShot** interface
2. Capture the screenshot by using **getScreenshotAs()** method and it will store screenshot in temp folder
3. Create a file in required location (Hard disk)
4. Copy from temp folder to → hard disk



Script:

```
public class Sample
{
    public static void main(String[] args) throws InterruptedException,
                                           IOException
    {
        String key="webdriver.gecko.driver";
        String value="./softwares/geckodriver.exe";
        System.setProperty(key,value);

        WebDriver driver = new FirefoxDriver();
        driver.get("https://www.amazon.in/");

        //step1
        TakesScreenshot ts = (TakesScreenshot) driver;

        //step2
        File src = ts.getScreenshotAs(OutputType.FILE);

        //step3
        File dst = new File("D:\\login.png");

        //step4
        FileUtils.copyFile(src,dst);  Files.copy(src, des); this is working
    }
}
```

⇒ Synchronization:

The process of matching selenium speed with application speed is called as synchronization. It is classified into 2 types,

1. Implicit wait
2. Explicit wait

Implicit wait:

By using this we can synchronize only 2 methods

1. findElement()
2. findElements()

Statement:

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

TimeUnit is now deprecated use `implicitlyWait(Duration.ofSeconds(10));`

By: **Harish Korgal**



In the browser manage the time and implicitly wait for 10 seconds

This method takes 2 arguments

1. Specified time
2. Time unit (Ex for abstract enum)

Time unit can be DAYS, HOURS, MINUTES, SECONDS...

Script:

```
public class Sample
{
    public static void main(String[] args) throws InterruptedException,
                                           IOException
    {
        String key="webdriver.gecko.driver";
        String value="./softwares/geckodriver.exe";
        System.setProperty(key,value);

        WebDriver driver = new FirefoxDriver();
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        driver.get("https://www.amazon.in/");

        driver.findElement(By.xpath("//a[.='SeleniumConf 2016']"));
    }
}
```

1. When the control comes to findElement(), it will verify whether element present or not
2. If element is present findElement() will return the address of first element
3. If element is not present then it will verify whether implicit wait is specified or not
4. If implicit wait is not specified then it will throw '**NoSuchElementException**' or '**Empty list/array**'
5. If implicit wait is specified then it will verify the specific time is over or not
6. If the specific time is over then it will throw '**NoSuchElementException**' or '**Empty list/array**'
7. If implicit wait time is not over then it waits for **500ms (1/2) seconds**
8. Waiting for 500ms is called as **Pooling period** which is present in **Fluent wait(Concrete class)**



9. And this process continues until,
 - a. Finds the element **OR**
 - b. Until specific time is over

Explicit wait:

- In order to synchronize all the methods including findElement() and findElements() method we go for explicit wait statement
 - In selenium explicit wait is represented by **WebDriverWait** class
 - In explicit wait we specify individual condition for individual methods
 - In explicit wait if condition is not satisfied it will throw '**TimeOutException**'
 - In explicit wait by default time unit is **SECONDS**
1. In explicit wait it will verify whether condition is satisfied or not
 2. If condition is satisfied then it will go to next statement
 3. If condition is not satisfied then it will verify whether the specific time is over or not
 4. If time is over it will throw '**TimeOutException**'
 5. If time is not over then it will wait for 500ms (1/2) second
 6. Waiting for 500ms is called as **Pooling period** which is present in **Fluent wait(Concrete class)**
 7. And this process continues until,
 - a. Finds the element **OR**
 - b. Until specific time is over

Implicit wait	Explicit wait
1. Supports for findElement() and findElements()	1. Supports for all the methods including findElement() and findElements()
2. Time unit can be set to min, sec, days	2. Time unit always seconds
3. We don't specify the conditions	3. We specify condition for individual method
4. If time is over it will throw	4. It will throw 'TimeOutException'



'NoSuchElementException' or empty list/ array	
---	--

Explicit wait	Thread.sleep()
1. We specify the condition	1. We don't specify the condition
2. It will throw 'TimeoutException'	2. It will throw 'InterruptedException'
3. Selenium exception	3. Java exception
4. Run time exception	4. Compile time exception
5. Time unit is seconds	5. Time unit is milli seconds
6. If page found it will go for next page	6. It waits for specified duration then it will go for next page

⇒ Test NG:

- Test NG is a unit testing framework
- Test NG is basically used by developers in order to perform white box testing or unit testing
- In selenium we use Test NG because,
 1. To run multiple scripts
 2. To generate reports/result
 3. To perform verification
 4. To achieve parallel execution (executing in multiple browsers)

Steps to download Test NG:

1. In Eclipse → Help → Eclipse market place
2. Search for Test NG
3. Click on Install



Steps to attach Test NG to the project

1. Right click on Project folder → Build path → Add libraries
2. Select Test NG → Next → Finish

Rules for developing Test NG class:

1. Create user defined package, never go for default package
2. Never use **main()** method for execution. Use test method annotation **@Test**
3. To print the message we use **Reporter.log()**

Note:

1. JVM will search for either main() method or @Test
2. In a single class we can have multiple @Test, but we cannot have multiple main() method
3. By using Reporter.log() we can print message in both report and console

Test NG class:

- It is a Java class which consist of Test method annotation **@Test**
- Test method – any method developed by using **@Test** is called test method annotation

```
import org.testng.Reporter;  
import org.testng.annotations.Test;
```

```
public class Sample  
{
```

```
    @Test
```

```
    public void test1()  
    {
```

```
        Reporter.Log("1", true);
```

```
    }
```

```
}
```

the true indicate the string also print in console if true is not provided default value is false it print only in html report.

//create in another file

```
import org.testng.Reporter;  
import org.testng.annotations.Test;
```

```
public class Sample2  
{
```

```
    @Test
```

```
    public void test2()  
    {
```

```
        Reporter.Log("2", true);
```



```
        Reporter.Log("2", true);  
    }  
}
```

- To **run multiple scripts** and to generate reports we use Test NG
- Test NG we generate **testng.xml** file
- testing.xml file consist of multiple classes

Steps to generate testng.xml file

1. Right click on project → click on Test NG
2. Convert to Test NG → Finish

Contents of testng.xml file

```
<suite name="Suite">  
  <test thread-count="5" name="Test">  
    <classes>  
      <class name="popup.Sample"/>  
      <class name="popup.Sample2"/>  
    </classes>  
  </test>  
</suite>
```

Steps to run testng.xml file

1. Right click on **testng.xml** file
2. Run as → click on **Test NG suite**

Steps to generate reports

1. Once after execution **Refresh** the project, it will generate folder called **test output**
2. Expand folder → Right click on '**emailable-report.html**' → open with → web browser

Steps to export report to excel sheet

Right click on report → click on '**Export to Excel**'



Interview questions

- **Priority for Test methods**

```
import org.testng.Reporter;
import org.testng.annotations.Test;

class Sample
{
    @Test (priority=1)
    public void test1()
    {
        Reporter.log("1", true);
    }
}

class Sample2
{
    @Test //default priority=0
    public void test2()
    {
        Reporter.log("2", true);
    }
}
```

Output: 2 1

By default priority=0, if the priority is same it will execute based on alphabetical order of method name

- **How to execute same test cases multiple times?**

By using parameter called **invocation count**

```
import org.testng.Reporter;
import org.testng.annotations.Test;

class Sample
{
    @Test (invocationCount=2, priority= -4)
    public void test1()
    {
        Reporter.log("1", true);
    }
}

class Sample2
{

```



```
@Test(priority= -8)
public void test2()
{
    Reporter.log("2", true);
}
}
```

Output:

2
1
1

- By default **invocationCount=1**
- If we specify invocation count ≤ 0 then it will skip the test case
- **How to skip the Test cases without using invocation count**

We using `@Test (enabled=false)` //by default enabled = true

→ **Important annotation of Test NG**

1. `@BeforeSuite`
2. `@BeforeTest`
3. `@BeforeClass`
4. `@BeforeMethod`
5. **`@Test`**
6. `@AfterMethod`
7. `@AfterClass`
8. `@AfterTest`
9. `@AfterSuite`
10. `@Parameters` (Used for parallel execution)
11. `@FindBy` (used in POM)
12. `@DataProvider`

1. @BeforeSuite

In order to verify the server issue before executing the frame work (pre condition)

2. @BeforeTest

In order to connect JDBC/to retrieve data from data base (Test data)



3. @BeforeClass

In order to open the application in each and every script

4. @BeforeMethod

We can use for opening application / login into application

5. @Test

To perform action on the application

6. @AfterMethod

We can use for closing application / logout to application

7. @AfterClass

To close the application

8. @AfterSuite

To verify whether reports are generated or not

Script:

```
package inbox;

public class Generic
{
    @BeforeSuite
    public void beforeSuite()
    {
        Reporter.Log("BeforeSuite",true);
    }

    @BeforeTest
    public void beforeTest()
    {
        Reporter.Log("BeforeTest",true);
    }

    @BeforeClass
    public void beforeClass()
    {
        Reporter.Log("BeforeClass",true);
    }

    @BeforeMethod
    public void beforeMethod()
    {
        Reporter.Log("BeforeMethod",true);
    }
}
```



```
@AfterMethod
public void afterMethod()
{
    Reporter.Log("AfterMethod", true);
}

@AfterClass
public void afterClass()
{
    Reporter.Log("AfterClass", true);
}

@AfterTest
public void afterTest()
{
    Reporter.Log("AfterTest", true);
}

@AfterSuite
public void afterSuite()
{
    Reporter.Log("AfterSuite", true);
}
}
```

```
package inbox;

public class Demo2 extends Generic
{
    @Test
    public void test2()
    {
        Reporter.Log("2", true);
    }
}
```

Output:

```
BeforeSuite
BeforeTest
BeforeClass
BeforeMethod
2
AfterMethod
AfterClass
AfterTest
PASSED: test2
```




```
=====
Default test
Tests run: 1, Failures: 0, Skips: 0
=====
```

AfterSuite

```
=====
Default suite
Total tests run: 1, Failures: 0, Skips: 0
=====
```

```
package inbox;

public class Demo1 extends Generic
{
    @Test
    public void test1()
    {
        Reporter.Log("1",true);
    }

    @Test
    public void test2()
    {
        Reporter.Log("2",true);
    }
}
```

Output:

```
BeforeSuite
BeforeTest
BeforeClass
BeforeMethod
1
AfterMethod
BeforeMethod
2
AfterMethod
AfterClass
AfterTest
PASSED: test1
PASSED: test2
```



```
=====
Default test
Tests run: 2, Failures: 0, Skips: 0
=====

AfterSuite

=====
Default suite
Total tests run: 2, Failures: 0, Skips: 0
=====
```

⇒ Verification

- In order to perform verification we use Test NG
- In TestNG we use **Assert** class
- In Assert class we use **assertEquals()** method
- Assert class methods are **static** methods
- assertEquals() methods take 2 arguments (**Actual, Expected**)

If Actual and Expected results are same, then it will perform the following actions

1. Pass the test case
2. Continues execution
3. Not throw exception
4. Display in Green color (Results)

If Actual and Expected results are not same, then it will perform the following actions

1. Fail the test case
2. Stops execution
3. Throw exception (Assertion error)
4. Display in Red color (Reports)



```
public class Generic
{
    public WebDriver driver;        //global declaration

    @BeforeMethod
    public void openApp()
    {
        System.setProperty("webdriver.gecko.driver","./softwares/geckodriver.exe");

        driver = new FirefoxDriver();
        driver.get("http://localhost/login.do");
    }

    @AfterMethod
    public void closeApp()
    {
        driver.quit();
    }
}
```

```
public class Login extends Generic
{
    @Test
    public void testLogin() throws InterruptedException
    {
        driver.findElement(By.id("username")).sendKeys("admin");
        driver.findElement(By.name("pwd")).sendKeys("manager");
        driver.findElement(By.xpath("//div[.='Login ']")).click();

        Thread.sleep(3000);

        String title = driver.getTitle();
        System.out.println(title);

        Assert.assertEquals(title,"actiTIME - Enter Time-Track");
        System.out.println("1");
    }
}
```

Output:

```
actiTIME - Enter Time-Track
1
PASSED: testLogin
```

```
=====
    Default test
    Tests run: 1, Failures: 0, Skips: 0
=====
```



```
=====
Default suite
Total tests run: 1, Failures: 0, Skips: 0
=====
```

→ List of methods in Assert class

1. **Assert.assertEquals(actual, expected);**
2. **Assert.assertNotEquals(actual, expected);**
3. **Assert.assertTrue(condition);**
4. **Assert.assertFalse(condition);**
5. **Assert.fail();**

⇒ Soft Assert

Even though if comparison gets failed, in order to continue the execution we use **soft assert**

In soft assert comparison gets fail, it will perform below

1. It will fail the test case
2. It will throw exception
3. It will continue exception
4. Reports displayed in Red color

In soft assert, by default we should call `assertAll()` method at the last

```
public class SoftAssert extends Generic
{
    @Test
    public void softAssert() throws InterruptedException
    {
        driver.findElement(By.id("username")).sendKeys("admin");
        driver.findElement(By.name("pwd")).sendKeys("manager");
        driver.findElement(By.xpath("//div[.='Login ']")).click();

        Thread.sleep(3000);

        String title = driver.getTitle();
        System.out.println(title);

        SoftAssert sa = new SoftAssert();
    }
}
```



```
sa.assertEquals(title, "actiTIME - Enter Time-Track"); //(a)
System.out.println("1");
sa.assertAll();
System.out.println("2");
}
}
```

Output:

1

2

- If **(a)** is **pass** then S.o.p("1") and S.o.p("2") will display
- If **(a)** is **fail** then S.o.p("1")
- **assertAll()** should always be at last [This will report all failures at once if we not call, failures will not be reported and the test is pass even the assertion fail](#)
- If **(a)** is **fail**, all the status after assertAll() will not be executed

Assert	SoftAssert
1. All are static method	1. All are non-static methods
2. If comparision gets failed it will stop exection	2. If comparision gets fail it will continue exection
3. We don't call assetAll()	3. We call assertAll() at last by default

TestNG	Junit
1. We can generate report	1. We cannot generate report
2. We can achieve parallel exection	2. We cannot achieve parallel exection

⇒ Data Driven Testing

1. Testing the application for multiple test data is called as data driven testing
2. In order to store the data we use Excel files
3. Excel file is also called as **work book**
4. Excel file is a **stand alone** application
5. In order to retrieve data from excel file, we use **Apache POI** (third party/supporting file)



6. Apache POI is used to **read the data from excel**
7. Apache POI consist of **13 jar files**

→ **Apache POI can be downloaded from below website**

1. Navigate to → <https://poi.apache.org/download.html>
2. Binary Distrubution section →download [poi-bin-3.17-20170915.zip](#) (consists of 13 jar files)
3. Create a folder '**Apache POI**' in project and copy paste 13 jar files and add to '**Build path**'
4. Create a folder '**Excel**' and create **data.xlsx** (excel) file in the respective path of that particular folder (in 'D' or 'E' driver)

→ **Steps to read the data from excel file**

1. Specify the path of the file
2. Open the excel file
3. Go to sheet by specifying sheet name
4. Go to row (starts from 0)
5. Go to column/cell (starts from 0)
6. Retrieve the data from the cell

```
public class Excel
{
    public static void main(String[] args) throws
    EncryptedDocumentException, InvalidFormatException, IOException
    {
        FileInputStream fs = new FileInputStream("./Excel/data.xlsx");
        //step 1
        Workbook wb = WorkbookFactory.create(fs); //step 2
        Sheet sh = wb.getSheet("Sheet1"); //step 3
        int row = sh.getLastRowNum(); //step 4
        System.out.println(row); //step 5
        Row r = sh.getRow(0); //step 6
    }
}
```



```
Cell c = r.getCell(1);
String v = c.getStringCellValue();
System.out.println(v);
}
}
```

Note:

1. Row position starts from 0
2. Cell position starts from 0
3. Row count starts from 0
4. Cell count starts from 1
5. If we try to read the data from empty cell it throw 'NullPointerException'
6. If we try to retrieve numeric value from the cell it throw 'IllegalStateException'
7. To convert numeric to string we use **single quote (')** at left side of number

WAS to count number of rows in a sheet

	0	1	2
0	123		
1	456		
2	789		
3	101		
4			
5			

Total data = 4

Row count = 4-1 = **3**

- Consider only data cell not empty cell

```
{
    Sheet sh = wb.getSheet("Sheet1");
    int count = sh.getLastRowNum();
    System.out.println(count);
}
```

Output: 3



10 rows	15 cells	100 rows	100 cells
5 rows(data)	9 cells(data)	100 cells(data)	100 cells (data)
O/p: 4 (5-1)	O/p: 9 (9-0)	O/p: 99 (100-1)	O/p: 100 (100-0)

⇒ 'StaleElementReference' exception

```
public class Demo
{
    public static void main(String[] args) throws InterruptedException
    {
        System.setProperty("webdriver.gecko.driver", "./softwares/geckodriver.exe");
        WebDriver driver = new FirefoxDriver();
        driver.get("http://localhost/login.do");

        driver.findElement(By.id("username")).sendKeys("admin");
        driver.findElement(By.name("pwd")).sendKeys("manager");
        WebElement ele = driver.findElement(By.xpath("[.='Login ']"));

        ele.click();           //pass

        Thread.sleep(3000);
        driver.navigate().refresh(); //pass

        Thread.sleep(3000);
        ele.click();           //throws 'StaleElementReferenceException'
    }
}
```

- In the above script after refreshing page, address of the element will change
- If you perform click operation it will throw '**StaleElementReferenceException**'
 - Stale – Old (or) expired
 - Reference – address (i.e. address is expired)
- To handle '**StaleElementReferenceException**' we go for **POM (Page Object Model)**



⇒ POM

- POM – Page Object Model
- POM is a java design pattern which is mainly used to handle 'StaleElementReference Exception'
- POM consists of 3 stages
 1. Declaration
 2. Initialization
 3. Utilization

→ Declaration:

In POM we declare element by using **@findBy** annotation

Syntax: `@findBy(LocatorName = "LocatorValue")`
 `private WebElement/List of webelements VariableName/ElementName`

Example: ` Login `

```
@findBy (id = "a1")
private WebElement loginBtn;
```

→ Initialization:

1. In POM we use **constructor** for initializing elements
2. In constructor we use '**PageFactory**' class
3. In 'PageFactory' class we use '**initElements()**' method
4. 'initElements()' takes 2 arguments
 - a. driver – Application control (OR) object of WebDriver
 - b. this – current object (OR) current page

```
public PomLogin (WebDriver driver)
{
    PageFactory.initElements(driver, this);
}
```



→ Utilization:

In POM we utilize the elements by developing methods

```
public void clickLogin()
{
    loginBtn.click();
}
```

Login script for ActiTime application

POM class for login page

```
public class PomLogin
{
    //Declaration
    @FindBy(id="username")
    private WebElement unBox;

    @FindBy(name="pwd")
    private WebElement pwdBox;

    @FindBy(xpath="//div[.='Login ']")
    private WebElement loginBtn;

    //Initialization
    public PomLogin (WebDriver driver)
    {
        PageFactory.initElements(driver, this);
    }

    //utilization
    public void setUserName(String un)
    {
        unBox.sendKeys(un);
    }

    public void setPassword(String pwd)
    {
        pwdBox.sendKeys(pwd);
    }

    public void clickLogin()
    {
        loginBtn.click();
    }
}
```

Test case for Login page

```
public class Demo
```



```
{
    public static void main(String[] args)
    {
        System.setProperty("webdriver.gecko.driver", "./softwares/geckodriver.exe");

        WebDriver driver = new FirefoxDriver();
        driver.get("http://localhost/login.do");

        PomLogin pl = new PomLogin(driver);
        pl.setUsername("admin");
        pl.setPassword("manager");
        pl.clickLogin();
    }
}
```

Interview Question

1. What is POM

It is a java design pattern which is mainly used to handle 'StaleElementReference Exception'

2. Advantages of POM

- To handle 'StaleElementReference Exception'
- To achieve encapsulation / to encapsulate by gets() and sets()
- To store elements
- POM is also called as **Object Repository**

3. How to handle elements in POM

- @FindBy annotation

4. How to initialize element in POM

- in constructor by using **PageFactory.initElements(driver, this);**

5. What happens if we do not initialize elements

- NullPointerException

6. How to handle multiple elements

@FindBy (//a)

private <list of WebElements> links;



FRAMEWORK

Framework is a set of procedure / guideliness / rules / protocol followed while automating the application.

In order to have consistency we use framework.

Framework consist of 3 stages

1. Design (Automation lead / manager)
2. Implimentation (Sn/Jn Automation engineer)
3. Exection (Jn Automation engineer/ fresher)

→ Framework Design:

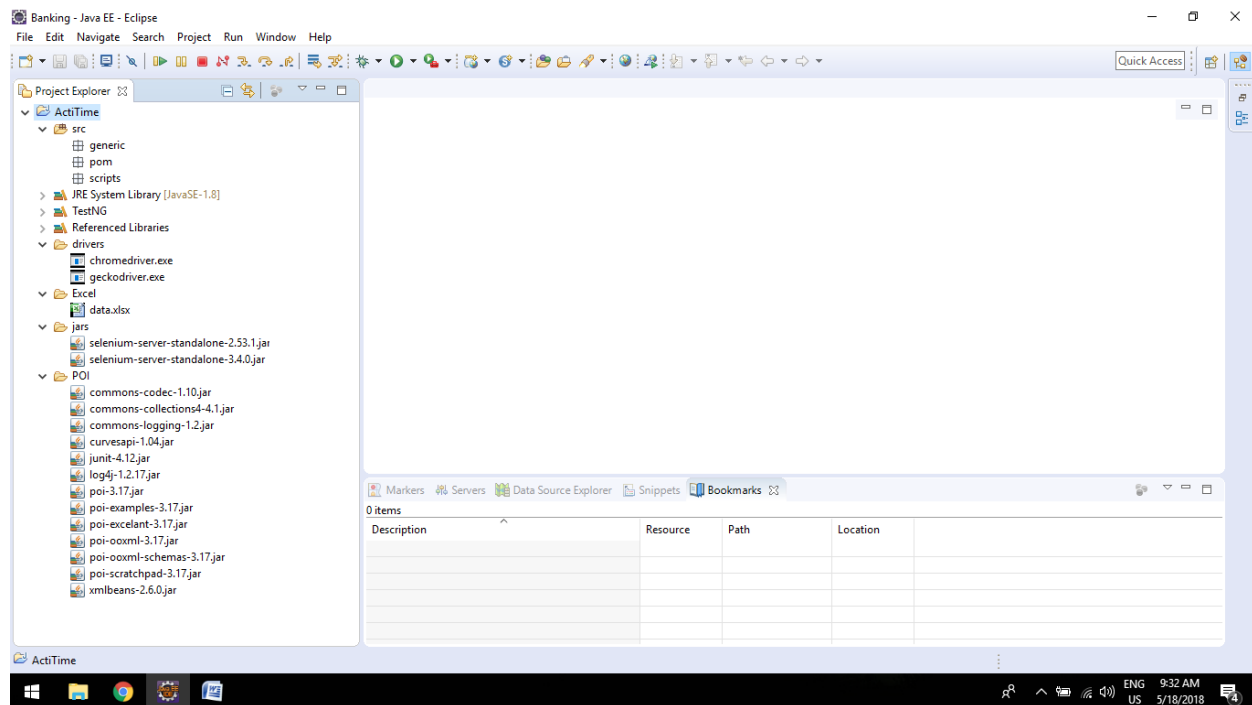
- In this stage either Automation lead/ manager is involved
- They will develop standard files and folder structure and also generic libraries (methods) required for framework

Steps to configure the design

1. Create a **folder** in respective drive and specify the folder name as Domain name (Banking, helathcare, transport)
2. Create a **project** inside the workspace and specify project name as application name (Actitime, gmail, facebook)
3. Create a folder '**jars**' and copy paste 'Selenium Stand alone jar' files (3.4.4 and 2.5.3.1) jar file and attach jar files to build path
4. Create a folder **drivers** and copy paste driver executable files (geckdriver.exe and chromedriver.exe)
5. Create a folder **Apache POI** and copy paste 13 jar files and attach all to build path
6. Create a folder **Excel** and create one excel sheet inside the folder
7. In order to run multiple scripts and to genrate reports attach **TestNG** to the project
8. Create **3 packages** inside the project
 - a. Generic (Design)
 - b. POM (Implementation)
 - c. Scripts (Execution)



Initial structure of folder structure of framework



Using interface in the framework

- In our framework/ project we have set of constants such as key, value and path of excel file etc...
- **In our framework we store constants inside interface**
- In interface if we declare any variables by default it is **public, static and final** variable
- Create a interface with name '**Automation_CONST**' inside generic package

package generic;

```
public interface Automation_CONST
{
    String key = "webdriver.gecko.driver";
    String value = "./drivers/geckodriver.exe";

    String PATH = "./Excel/data.xlsx";
}
```

Using inheritance in the framework

1. For each and every test case we develop Test class



2. In each and every test class we perform common actions such as (open application and close application)
3. In order to avoid the repetition of code we develop generic method for (open application and close application)
4. In order to access the generic method we use **Inheritance**
5. Instead of creating an object we use **annotation**
6. Create a class with the name '**Base_TEST**' inside generic package

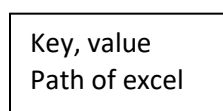
```
package generic;
```

```
public class Base_TEST implements Automation_CONST
{
    public WebDriver driver;

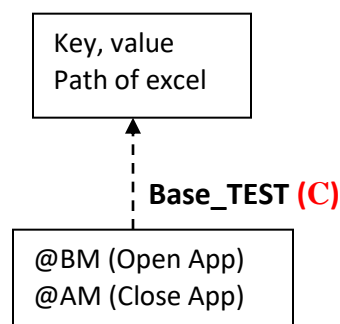
    @BeforeMethod
    public void openApplication()
    {
        System.setProperty(key, value);
        driver=new FirefoxDriver();
        driver.get("http://localhost/login.do");
    }

    @AfterMethod
    public void closeApplication()
    {
        driver.quit();
    }
}
```

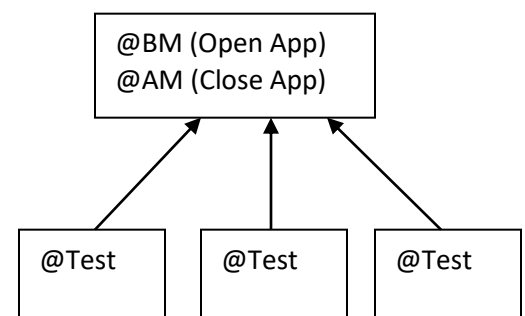
1. Auto_CONST (I)



2. Auto_CONST (I)



3. Base_TEST (C)



(for all 3 @Test method below is the input)

Output: @BeforeMethod
@Test
@AfterMethod



Using Excel file in the framework

1. In order to test the application with the multiple data is called as data driven testing
2. In order to store the data we use excel file
3. In order to read the data from excel we use **Apache POI** files
4. We develop generic method to read the data from excel file
5. Create a class '**Excel**' in generic package

```
package generic;

public class Excel
{
    public static String getCellValue(String PATH, String sheet, int row,
int cell) throws EncryptedDocumentException, InvalidFormatException,
IOException
    {
        String v="";
        try
        {
            FileInputStream fis = new FileInputStream(PATH);
            Workbook wb = WorkbookFactory.create(fis);
            Cell c = wb.getSheet(sheet).getRow(row).getCell(cell);
            v=c.getStringCellValue();
        }
        catch(Exception e)
        {
        }
        return v;
    }
}
```

Using abstract class in the framework:

1. For each and every page we develop POM class
2. In each and every POM class we perform common action such as (verification of title and verification of element etc...)
3. In order to avoid repetition of code we develop generic methods in the generic class
4. In order to access the generic methods we use **inheritance**
5. Since this class only for inheritance purpose, not for execution we declare class as **abstract**



6. Create a class 'Base_PAGE' inside generic package

```
package generic;

public abstract class Base_PAGE
{
    public WebDriver driver;
    public Base_PAGE(WebDriver driver)
    {
        this.driver=driver;
    }

    public void verifyTitle(String title)
    {
        WebDriverWait wait = new WebDriverWait(driver,10);
        try
        {
            wait.until(ExpectedConditions.titleContains(title));
            Reporter.Log("title is matching",true);
        }
        catch (Exception e)
        {
            Reporter.Log("title is not matching",true);
            Assert.fail();
        }
    }

    public void verifyElement(WebElement element)
    {
        WebDriverWait wait = new WebDriverWait(driver,10);
        try
        {
            wait.until(ExpectedConditions.visibilityOf(element));
            Reporter.Log("element is present",true);
        }
        catch (Exception e)
        {
            Reporter.Log("element is not present",true);
            Assert.fail();
        }
    }
}
```

→ Framework Implementation:

- Converting manual test cases to Automation scripts with the help of framework design (generic methods/libraries)
- In this stage either Senior /Junior Automation engineer will be involved



- In this stage they will develop **POM** classes for **individual** webpage.

Test cases for Automation

Script 1: Valid login and logout

- Step1: Enter valid user name
- Step2: Enter valid password
- Step3: Click on Login button
- Step4: Verify home page is displayed or not
- Step5: Click on Logout
- Step6: Verify login page is displayed or not

Script2 : Invalid login

- Step1: Enter invalid user name
- Step2: Enter invalid password
- Step3: Click on Login button
- Step4: Verify Error message is displayed

Script3 : Verify ActiTime version

- Step1: Enter valid user name
- Step2: Enter valid password
- Step3: Click on Login button
- Step4: Verify home page is displayed or not
- Step5: click on help button
- Step6: click on 'About ActiTime'
- Step7: Verify ActiTime version
- Step8: Click on close button
- Step9: Click on Logout
- Step10: Verify login page is displayed or not



Elements required for POM class

Login page	Home page
1. User name	1. Logout
2. Password	2. Help
3. Login	3. About actitime
4. Error message	4. Actitime version
	5. Close popup

Rules for developing POM class

1. POM class depends on **number of web pages**
2. For each and every webpage we develop POM class
3. **POM class name** should be same as **title of the web page**
4. POM class name should ends with keyword called '**page**'
5. Each and every POM class shold extends from '**Base_PAGE**' class
6. POM class should be develop inside '**POM**' package

POM class for login page

```
package pom;

public class LoginPage extends Base_PAGE
{
    //declarations

    @FindBy(id="username")
    private WebElement untbox;

    @FindBy(name="pwd")
    private WebElement pwdtbox;

    @FindBy(xpath="//div[.='Login ']")
    private WebElement loginbtn;

    @FindBy(xpath="//span[contains(.,'invalid')]")
    private WebElement errormsg;

    //initilization

    public LoginPage(WebDriver driver)
    {
        super(driver);           //to achieve constructor chaining
    }
}
```



```
        PageFactory.initElements(driver, this);
    }

    //utilization

    public void setUsername(String un)
    {
        untbox.sendKeys(un);
    }

    public void setPassword(String pwd)
    {
        pwdtbox.sendKeys(pwd);
    }

    public void clickLogin()
    {
        loginbtn.click();
    }

    public void verifyErrorMsg()
    {
        verifyElement(errormsg);
    }

    public void verifyLoginPage(String lp_title)
    {
        verifyTitle(lp_title);
    }
}
```

POM class for home page

```
package pom;

public class EnterTimeTrackPage extends Base_PAGE
{
    //declarations

    @FindBy (id="logoutLink")
    private WebElement logout;

    @FindBy (xpath="(((div[@class='popup_menu_arrow']))[3])")
    private WebElement help;

    @FindBy (xpath="//a[.='About actiTIME'"]")
    private WebElement aboutActiTime;

    @FindBy (xpath="//span[.='actiTIME 2014 Pro'"]")
    private WebElement version;
}
```



```
@FindBy (xpath="//img[@title='Close']")
private WebElement closePopup;

//initilization

public EnterTimeTrackPage(WebDriver driver)
{
    super(driver);    //constructor chaining
    PageFactory.initElements(driver,this);
}

//utilization

public void logout()
{
    logout.click();
}

public void clickHelp()
{
    help.click();
}

public void clickAboutActiTime()
{
    aboutActiTime.click();
}

public void clickClose()
{
    closePopup.click();
}

public void verifyHomePage(String hp_title)
{
    verifyTitle(hp_title);
}

public void verifyActiTimeVersion(String eversion)
{
    String aversion = version.getText();
    Assert.assertEquals(aversion,eversion);
}
}
```

→ **Framework Execution:**

1. In this stage Jn. Automation engineer / fresher will be involved
2. In this stage we will develop test classes



3. Test cases depends on number of test cases
4. For each and every test case we develop test classes

Rules for developing test class

1. Test class depends on number of test cases
2. For each and every test case we develop test class
3. **Test class name should be same as test case name or test script name**
4. Test class should be developed inside '**scripts**' package
5. Each and every test class extends from '**Base_TEST**' class
6. In test class we call POM class method to perform action on the browser

Script 1: Valid login and logout

```
package scripts;

public class ValidLoginLogout extends Base_TEST
{
    @Test
    public void testValidLoginLogout() throws EncryptedDocumentException,
InvalidFormatException, IOException
    {
        String un = Excel.getCellValue(PATH,"ValidLoginLogout", 1, 0);
        String pwd = Excel.getCellValue(PATH,"ValidLoginLogout", 1, 1);
        String lp_title = Excel.getCellValue(PATH,"ValidLoginLogout", 1, 2);
        String hp_title = Excel.getCellValue(PATH,"ValidLoginLogout", 1, 3);

        LoginPage lp = new LoginPage(driver);
        lp.setUsername(un);
        lp.setPassword(pwd);
        lp.clickLogin();

        EnterTimeTrackPage ep = new EnterTimeTrackPage(driver);
        ep.verifyHomePage(hp_title);
        ep.logout();
        lp.verifyLoginPage(lp_title);
    }
}
```

Script 2: Invalid login

```
package scripts;

public class InvalidLogin extends Base_TEST
{
}
```



```
@Test
public void testInvalidLogin() throws EncryptedDocumentException,
    InvalidFormatException, IOException
{
    String un = Excel.getCellValue(PATH,"InvalidLogin", 1, 0);
    String pwd = Excel.getCellValue(PATH,"InvalidLogin", 1, 1);

    LoginPage lp = new LoginPage(driver);
    lp.setUserName(un);
    lp.setPassword(pwd);
    lp.clickLogin();
    lp.verifyErrorMsg();
}
}
```

Script 3: Verify ActiTime version

```
package scripts;

public class VerifyVersion extends Base_TEST
{
    @Test
    public void testValidLoginLogout() throws EncryptedDocumentException,
        InvalidFormatException, IOException
    {
        String un = Excel.getCellValue(PATH,"VerifyVersion", 1, 0);
        String pwd = Excel.getCellValue(PATH,"VerifyVersion", 1, 1);
        String lp_title = Excel.getCellValue(PATH,"VerifyVersion", 1, 2);
        String hp_title = Excel.getCellValue(PATH,"VerifyVersion", 1, 3);

        LoginPage lp = new LoginPage(driver);
        lp.setUserName(un);
        lp.setPassword(pwd);
        lp.clickLogin();

        EnterTimeTrackPage ep = new EnterTimeTrackPage(driver);
        ep.verifyHomePage(hp_title);
        ep.clickHelp();
        ep.clickAboutActiTime();
        ep.verifyActiTimeVersion("actiTIME 2014 Pro");
        ep.clickClose();
        ep.logout();
        lp.verifyLoginPage(lp_title);
    }
}
```

}



- In order to run multiple scripts we use TestNG
- In order to generate reports and perform verification we use TestNG
- In the framework we develop **TestNG.xml** file
- **To convert project to TestNG.xml** (Right click on project → TestNG → Convert to TestNG → Finish

- Contents of TestNG.xml file

```
<suite name="Suite">
  <test thread-count="5" name="Test">
    <classes>
      <class name="scripts.VerifyVersion"/>
      <class name="scripts.InvalidLogin"/>
      <class name="scripts.ValidLoginLogout"/>
    </classes>
  </test>
</suite>
```

- **To Run TestNG.xml file** (Right click on TestNG.xml → Run As → TestNG suite
- **To generate reports** after project (Refresh the project → it will generate **test-output** folder
- Expand the folder → right click on **Emailable report.html** → open with → web browser
- Right click on report → Export to excel

FRAMEWORK ARCHITECTURE

1. In our project we use **Hybrid Framework**
2. Hybrid framework is a combination of multiple frameworks such as POM, Data driven, Method driven and TestNG

TestNG

1. In order to run multiple scripts, generate report and to perform verification we use TestNG



2. TestNG is a unit testing framework

POM

1. In order to handle '**StaleElementReferenceException**' and to store the elements we go for POM
2. POM is also called as **Object framework**

Data Driven Framework

In order to test the application with multiple test data from the excel and retrieve data with the help of **Apache POI** libraries is called as Data driven framework

Method Driven Framework

In order to avoid repetition of the code we develop individual methods for individual components in POM class is called as Method Driven Framework

- In our framework we store the **constants** in **interface** called as **Auto_CONST**
- **Base_TEST** class consists of 2 methods i.e. **@BeforeMethod** → to open application
@AfterMethod → to close application
- In **Base_PAGE** class we develop generic methods such as verification of title and verification of elements etc...
- Each and every POM class should be **extend** from **Base_PAGE** class
- During the run time, first it will execute **@BeforeMethod** and it perform open browser and enter the URL
- After executing **@BeforeMethod**, it will execute **@Test** method i.e. it will perform action on the browser with the help of POM class methods (i.e. generic methods)
- Once after executing **@Test** control will come to **@AfterMethod** (it will close browser)
- Once after executing all the scripts it will generate report in HTML format using TestNG