**What is Selenium?**

Selenium is a free (open source) automated testing suite for web applications across different browsers and platforms. Testing done using Selenium tool is usually referred as Selenium Testing.

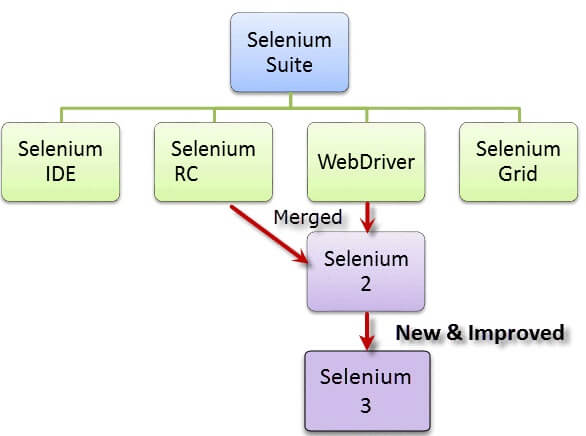
Selenium is not just a single tool but a **suite of software's,** each catering to different testing needs of an organization. **It has four components**.

1. Selenium Integrated Development Environment (IDE)

2. Selenium Remote Control (RC)

3. WebDriver

4. Selenium Grid



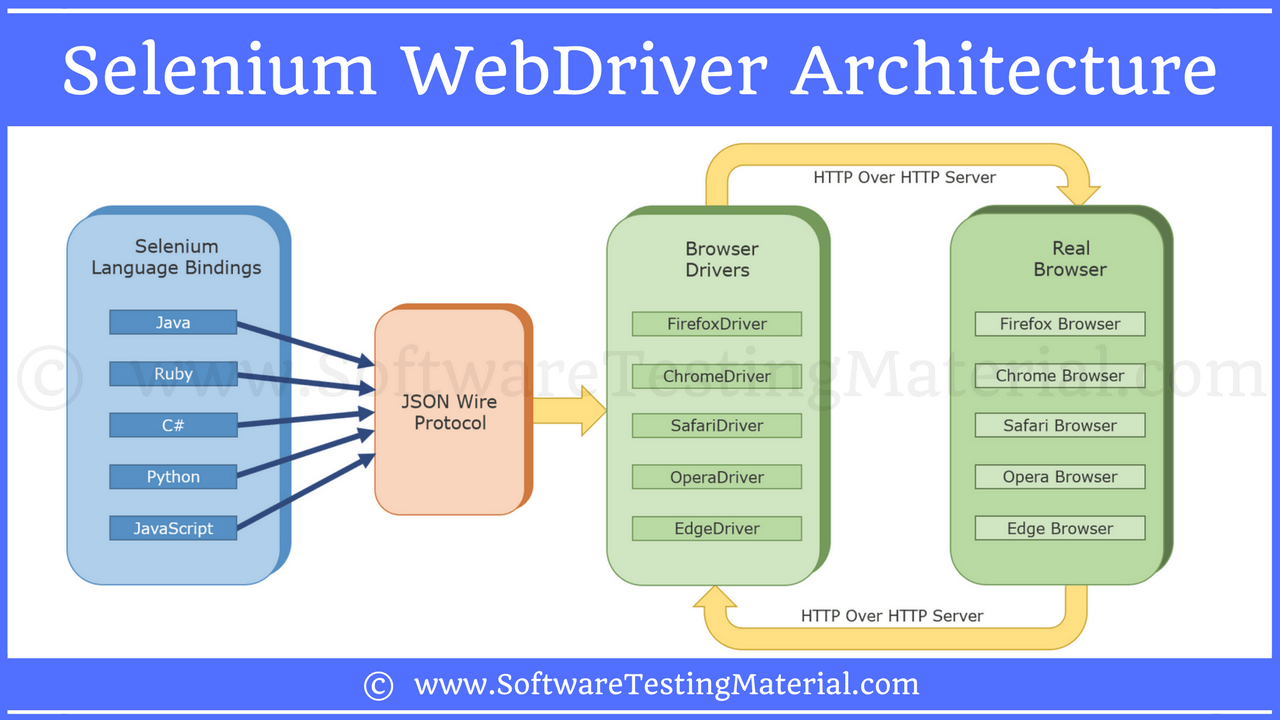
At the moment, Selenium RC and WebDriver are merged into a single framework to form Selenium 2. Selenium 1, by the way, refers to Selenium RC.

1. **Selenium Integrated Development Environment (IDE)** is the simplest framework in the Selenium suite and is the easiest one to learn. It is a Firefox plugin that you can install as easily as you can with other plugins. However, because of its simplicity, Selenium IDE should only be used as a prototyping tool. If you want to create more advanced test cases, you will need to use either Selenium RC or WebDriver.
2. **Selenium RC** was the flagship testing framework of the whole Selenium project for a long time. This is the first automated web testing tool that allowed users to use a programming language they prefer. As of version 2.25.0, RC can support the following programming languages: Java, C#, PHP, Python, Perl, Ruby.
3. **WebDriver** proves itself to be better than both Selenium IDE and Selenium RC in many aspects. It implements a more modern and stable approach in automating the browser's actions. WebDriver, unlike Selenium RC, does not rely on JavaScript for Automation. It controls the browser by directly communicating with it. The supported languages are the same as those in Selenium RC. Java, C#, PHP, Python, Perl, Ruby.
4. **Selenium Grid** is a part of the Selenium Suite that specializes in running multiple tests across different browsers, operating systems, and machines in parallel.

**Selenium WebDriver :** is a browser automation framework that accepts commands and sends them to a browser. It is implemented through a browser-specific driver. It controls the browser by directly communicating with it. Selenium WebDriver supports Java, C#, PHP, Python, Perl, Ruby.

Operation System Support – Windows, Mac OS, Linux, Solaris

Browser Support – Mozilla Firefox, Internet Explorer, Google Chrome 12.0.712.0 and above, Safari, Opera 11.5 and above, Android, iOS, HtmlUnit 2.9 and above



the url will be: [http://localhoat:8080/{“url](http://localhoat:8080/%7b)”: “ <http://www.google.com>”}

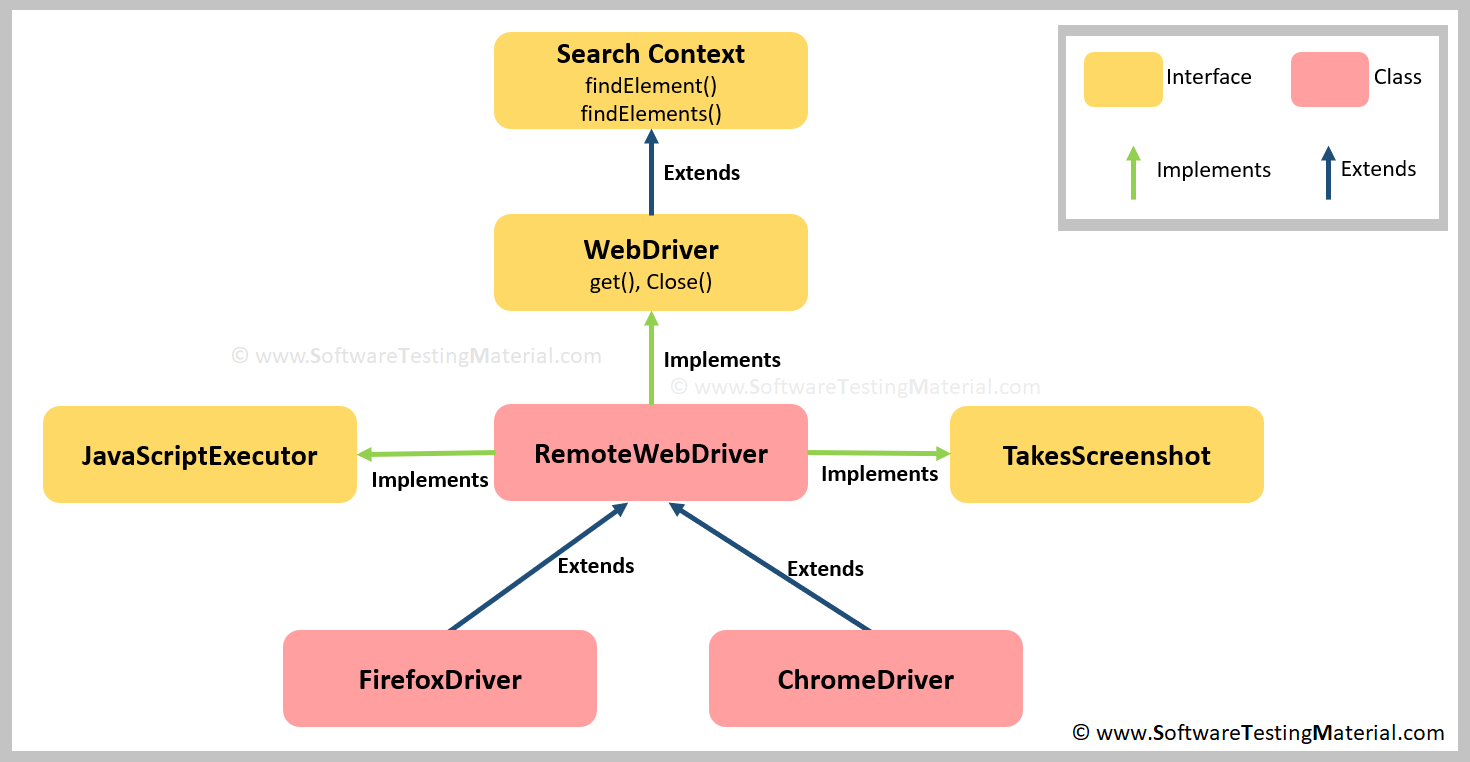
if the request is Get: the corresponding response is generated at the browser end and will send back over HTTP to browser

if the request is Post: then action is performed on the browser.

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

Here : driver is a object of FirefoxDriver and reference variable.

1. We cannot create object of the Interface.
2. We are creating an instance of WebDriver and casting it to FirefoxDriver class.



WebDriver contains all the Selenium methods signatures, whereas definition of the methods are in RemoteWebDriver class.WebDriver will start up a web browser on the computer where the code instantiates it. RemoteWebDriver require the Selenium standalone Server to be running. If we are working with Selenium Grid we need RemoteWebDriver.

**What is meant by Selenium WebDriver?**

**Selenium WebDriver** is a collection of open source APIs which are used to automate the testing of a web application. It supports many programming languages Java, C#, PHP, Python, Perl, Ruby. It supports many browsers such as Firefox, Chrome, IE, and Safari.

**WebDriver** is a public interface, we just define a reference variable(driver) whose type is interface. Now any object we assign to it must be a instance of a class (FireFoxDriver) that implement the interface.

**Advantages of Selenium:**

* 1. Open source software.
* 2. Supports various programming languages.
* 3. Supports various operating systems.
* 4. Selenium supports various browsers.
* 5. Selenium supports Parallel testing.
* 6. Selenium uses less Hardware resources.

**Limitations of Selenium:**

* 1. No reliable Technical support.
* 2. It supports Web based applications only.
* 3. Do not support automation tests on Web Services like SOAP OR REST.
* 4. Selenium WebDriver has programming interface, no IDE.
* 5. Cannot automate captcha.
* 6. Limited support for Image properly.
* 7. No built-in Reporting facilities. Need plugins like Junit or TestNG test reports.

**Selenium Driver object Syntax:**

private static WebDriver driver;  
 private String browserOptions=System.getProperty ("browser");

private String url=System.getProperty ("url");

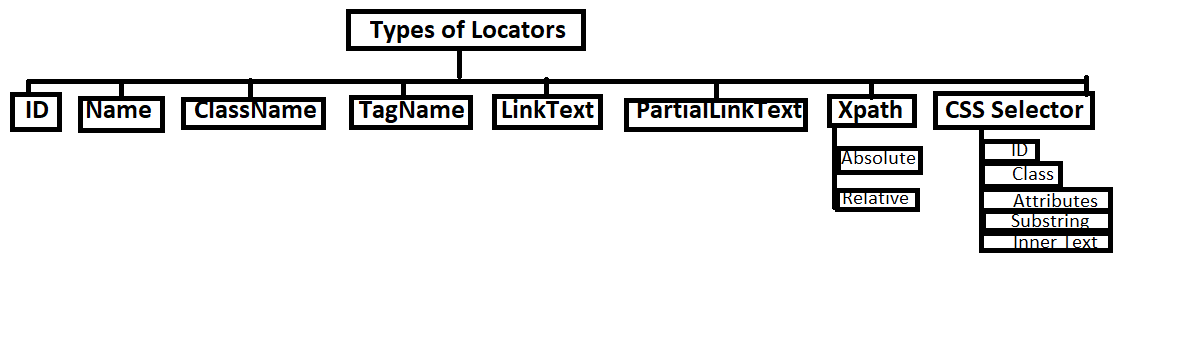
// to pass the variable browser – > go to Run > Edit configuration> in VM options add Dbrowser="chrome" -Durl=”web -address “ and then apply

public void chooseBrowser ( ) {  
 switch (browserOptions) {  
 case "firefox":  
 System.setProperty ("webdriver.gecko.driver","C:\\Users\\sball\\Downloads\\geckodriver-v0.22.0-win64\\geckodriver.exe");  
FirefoxOptions options = new FirefoxOptions().setProfile(new FirefoxProfile ());  
WebDriver driver = new FirefoxDriver(options);

break;  
 case "chrome":  
 System.setProperty ("webdriver.chrome.driver","C:\\Users\\sball\\Downloads\\chromedriver\_win32\\chromedriver.exe");  
 driver=new ChromeDriver ();  
 System.out.println (url);  
 break;  
 case "safari" :  
 driver=new SafariDriver ();  
 break;  
 case "ie" :  
 System.setProperty ("webdriver.ie.driver","C:\\Users\\sball\\Downloads\\MicrosoftWebDriver.exe");  
 driver=new InternetExplorerDriver ();  
 break;  
 }  
 }  
**Web Element:** is a reference to the elements of the web page or DOM. [ DOM – it defines the logical structure of elements and they the way JavaScript sees its containing pages data].

**Locators:** is termed as an address of that identifies a web element uniquely within web page.

Selenium WebDriver uses 8 locators to find the elements on a web page.



-**By ID:**

<input id="email" class="required" type="text"/>

WebElement Ele = driver.findElement(By.id("toolbar"));

**By Name:**

<input name="register" class="required" type="text"/>

WebElement register= driver.findElement(By.name("register"));

**By ClassName:**

<input name="register" class="required" type="text"/>

WebElement classtest =driver.findElement(By.className(“required”));

**By TagName:**

Select select = new Select(driver.findElement(By.tagName("select")));

select.selectByVisibleText("Nov");

or

select.selectByValue("11");

**By LinkText:**

<a href="http://www.seleniumhq.org">Downloads</a>

WebElement download = driver.findElement(By.linkText("Downloads"));

**By PartialLinkText:**

<a href="seleniumhq.org">Download selenium server</a> WebElement download = driver.findElement(By.PartialLinkText("Download"));

**Xpath:** [Inspect > for windows ctrl + f ]

Syntax:

//htmltag[@atribute1='value1’ and @attribure2-‘value2’]

//input[@class='octext']

//input[@name='quesry'] and @value=’text’]

//input[contains(@class,'octext')]

//input[starts-with(@class,'oct')]

//input[ends-with(@class,'ext')]

**Single (forward) Slash “/”:** It represents the absolute path. In this case, the XPath engine navigates the DOM right from the first node.

**Double (forward) Slash “//”:** It represents the relative path. In this case, the XPath engine searches for the matching element anywhere in the DOM.

**CSS Selector:**

WebElement CheckElements =driver.findElements(By.cssSelector("input[id=email']"));

**Most common interaction commands in Selenium**

1) get()

2) getCurrentUrl()

3) findElement(By, by).click()

4) isEnabled()

5) findElement(By, by).sendKeys()

6) findElement(By, by).getText()

7) Submit()

8) findElements(By, by)

9) findElements(By, by).size()

10) driver.manager().pageLoadTimeout(time,unit

11) driver.manager().timouts().implicitlyWait()

12) getTitle()

13) driver.navigate().to ("url") and driver.navigate().back() and driver.naviagte.forward()

14) getScreenshotAs()

15) moveToElement() - of Actions class

16) dragAndDrop() - of Action class

17) 20) switchTo() and accept(), dismiss() and sendKeys() - of Alert class

18) getWindowHandle() and getWindowHandles() - to handle multiple windows

19) assertEquals(),assertNotEquals(), assertTrue() and assertFalse()

20) close() and quit() - quit is used to quit driver instance.

**Assertions in selenium can be used in 3 modes which are explained below:**

**1. assert:** If you use assert in your tests then the test will be aborted if the assert fails. Next test case will start executing in case you are running a test suite.

**2. verify:** If verify is used then the test case will not abort even if the verify fails. It will continue executing and log the failure for the failed conditions.

**3. waitFor:** waitFor command waits for the condition to become true. If the condition is true already the test case continues else it waits for the conditions to become true. If the condition doesn’t becomes true within specified time-out period test will fail and halt.

**Selenium waits or how will you handle synchronization or timeout.**

**Selenium waits:**

By default, Selenium WebDriver do not wait for an element once the page load completes. It checks for an element on the page then it performs some operation based upon the script, but if the element is not present then it throws NoSuchElementException.

* **By default, timeout in Selenium is Zero.**

**Implicit wait:**

* wait for certain specified amount of time before throwing Exception.
* Applicable throughout the script and will works on all elements in the script once specified implicit wait.

**driver.manage().timeout().implicitlywait(3000,TimeUnits);**

**Explicit wait: [ is in WebDriverWait class ]**

* Wait for specific condition to be met.
* It is a dynamic wait which wait dynamically for a specific condition.
* It is implemented by WebDriver class.

**WebDriverWait wait=new WebDriverWait(driver,2000);**

**WebElement ele=wait.until(ExpectedConditions.visibilityOfElementLocated(By.id” uname ”);**

**Fluent wait:** **[ is a plain class ]**

* It is a combination of implicit and explicit wait

FluentWait <WebDriver> wait=new FluentWait(driver);

wait.pollingEvery().wait Timeout();

**DesiredCapabiities in Selenium:** is a class helps us to tell the WebDriver, which environment we are going to use in our test scripts. Every test scenario should be executed on some specific testing environment. Testing environment can be a web browser, mobile device, mobile emulator, mobile simulator, etc.

**DesiredCapabilities cap=new DisiredCapabiities();**

setCapability method can be used in selenium Grid.

getCapabiity() , getBrowserName(), setBrowserName(), getVersion(), setVersion, getPlatform(),setPlatform().

**Exceptions in Selenium WebDriver:**

* **Java.lang.illegalStateException**

Occurs -if the path to driver is not correct or specified

Solution -set the path to driver.

* **ElementNotVisibleException**

Occurs – duplicate xpath or element currently visible.

Solution-use explicit wait to visibilityOfElement()

* **ElementNotClickableException**

Occurs – mostly in chrome because chrome does not calculate element location

Solution -use JavaScriptExecutor method scrollIntoView(webElement);

* **InvalidElementStateException**

Occurs- when element is disabled

Solution- perform operation based upon element state.

* **NoSuchElementException**

Occurs –when unable to identify the element during runtime.

* **ElementNotVisibleException**

Occurs – when element is present in DOM but not visible.

* **NoSuchFrameException**

Occurs – when driver switching to invalid frame.

* **NoSuchWindowException**

Occurs- when driver switching to invalid window.

* **NoAlertPresentException**

Occurs- when driver is switching to invalid alert.

* **WebDriverException**

Occurs – when performing action after immediately closing the browser.

* **SessionNotFoundException**

Occurs-after quitting the browser.

* **StaleElementReferedException**

Occurs=during which element belong to different frame.

**WebDriverManager class**: allows to automate the management of binary driver( eg. Chromedriver,geckodriver,etc) required by Selenium WebDriver.

In order to work with browser, we need the respective driver to work.Or to manage the binaries.

To use it first we need to add the dependency.

<dependency>

<groupId>io.github.bonigarcia</groupId>

<artifactId>webdrivermanager</artifactId>

<version>3.4.0</version>

<scope>test</scope>

</dependency>

WebDriver driver;

WebdriverManager.chromedriver().setup();

driver=new ChromeDriver();

**Actions Class:**

**[ Note build() and perform() methods are mandatory to follow ]**

Actions action = new Actions(driver);

moveToElement(Webelement).build().perform(); -> mouse hover.

clickAndHold(Webelement).build.perform(); -> click and hold.

dragAndDrop(Webelement src, Webelement tgt).build().perform(); -> drag and drop.

dragAndDropBy( Webelement src, xoffset,yoffset).build().perform(); -> drag and drop to x ,y.

moveByOffset(aoffset, yoffset).build().perform(); -> move mouse from current location to x ,y.

Click().build().perform(); -> click on the web page.

click(Webelement).build.perform().perform(); -> click on web element.

doubleClick(Webelement).build().perform(); -> double click on web element

release().build().perform(); -> release the element

**KeyBoard Activities Event: .sendKeys(keys.ARROW\_DOWN);,. sendKeys(Keys.ARROW\_UP);**

**Alert Interface:**

Since alerts are separate windows so before performing using alert methods, we need to switch to alert window. Web based alerts and JavaScript alerts are same.

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

driver.switchTo().alert().accept();

driver.switchTo().alert().dismiss();

code to check if alerts are present.

public boolean isAlertPresent(ldriver)

{

try

{

ldriver.switchTo.alert();

return true;

}catch(NoAlertException e)

{

return false;

}

}

If(isAlertPresent(ldriver))

}

Alert alert=ldriver.switchTo.alert();

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

Alert.accept();

}

**Handling Frames:**

if the element exists in the frame then we need to switch to frame before performing operation.

driver.switchTo().frame(index); -> to switch frame with index.

driver.switchTo().frame(framename); -> to switch frame with name.

driver.switchTo().defaultcontent(); -> to go back to parent frame.

**Handling Windows:**

if we want to switch to tabs.

In selenium we can get window name of the current window using

**String win=driver,getWindowHandle(); - > current window name**

To get all window names opened by the WebDriver

**Set<String> wList=driver.getWindowHandles(); -> all windows opened by WebDriver.**

**Select Class:**

is used to perform multiple operations on dropdown object and Multiple select Object.

Select dropdown=new Select(WebElement);

dropdown.selectByVisibleText(”string”);

dropdown.selectByIndex(index”);

dropdown.selectByValue(”string”);

List<WebElement el= dropdown.getOptions();

dropdown.deselecAll(); , deselectByIndex(index), deselctByValue(“string”),

dropdown.isMultiple() -> return boolean. Then we can do multiple select.

**TakeScreenShot class:**

TakeScreenShot ts=(TakeScreenShot)driver;

File shot=ts.getScreenShotAs(OutpuType.File);

FileHandler.copy(shot,new File(“path”);

**Selenium: How to print current date and time using Java/Selenium:**

public class DateTime {  
 public static void main (String[] args) {  
 DateFormat df=new SimpleDateFormat ("dd/MM/yyyy:hh:mm:ss");  
 Date dt=new Date ()

String str=de.format(dt);

System.out.println (str));  
 }  
}

**TakeScreenshot:**

public void takeScrShot() throws IOException {  
 TakesScreenshot ts=(TakesScreenshot)*driver*;  
 File scrShotFile=ts.getScreenshotAs (OutputType.*FILE*);  
 FileHandler.*copy* (scrShotFile, new File ("C:\\Users\\sball\\IdeaProjects\\cathbdd\\src\\test\\java\\com\\cath\\kidston\\bbd\\ScreenShot\\screenShot.png"));  
  
}

**Read data from Excel Sheet:**

**public class** GetDataFromExl {  
  
 *// to obtain data from column1* **public** ArrayList<String> getcol1() **throws** IOException {  
  
 FileInputStream instm=**new** FileInputStream (**"C:\\Users\\sball\\Desktop\\testdatalogin.xls"**);  
 HSSFWorkbook bk=**new** HSSFWorkbook (instm );  
 HSSFSheet sh=bk.getSheetAt (0);  
  
 Iterator <Row> rw= sh.iterator ();  
  
 ArrayList<String> uid=**new** ArrayList <String> ();  
  
 **while**(rw.hasNext ())  
 {  
 uid.add (rw.next ().getCell (0).getStringCellValue ());  
  
 }  
 **return** uid;  
 }  
  
 *// to obtain data from column2* **public** ArrayList<String> getcol2() **throws** IOException {  
  
 FileInputStream ipst=**new** FileInputStream (**"C:\\Users\\sball\\Desktop\\testdatalogin.xls"**);  
 HSSFWorkbook book=**new** HSSFWorkbook ((ipst));  
 HSSFSheet sheet=book.getSheetAt (0);  
 Iterator<Row> rw2=sheet.iterator ();  
  
 ArrayList<String> pwd=**new** ArrayList <String> ();  
 **while**(rw2.hasNext ())  
 {  
 pwd.add (rw2.next ().getCell (1).getStringCellValue ());  
 }  
 **return** pwd;  
 }

*// to obtain all excel data passing column number as arguments* **public** ArrayList<String> addExcelData(**int** colNumber ) **throws** IOException {  
 FileInputStream inpstm=**new** FileInputStream (**"C:\\Users\\sball\\Desktop\\testdatalogin.xls"**);  
 HSSFWorkbook book=**new** HSSFWorkbook (inpstm);  
 HSSFSheet sheet=book.getSheetAt (0);  
 ArrayList<String> coldata=**new** ArrayList <String> ();  
 Iterator<Row> rows=sheet.iterator ();  
 **while** (rows.hasNext ())  
 {  
 coldata.add (rows.next ().getCell (colNumber).getStringCellValue ());  
 }  
 **return** coldata;  
 }  
}

**public class** ReadExIterator {  
GetDataFromExl **getData**=**new** GetDataFromExl ();  
**private static** WebDriver *exldr*;  
 **@Before  
 public void** openBrows()  
 {  
 System.*setProperty* (**"webdriver.chrome.driver"**,**"C:\\Users\\sball\\Downloads\\chromedriver\_win32\\chromedriver.exe"**);  
 *exldr*=**new** ChromeDriver ();  
 *exldr*.manage ().window ().maximize ();  
 }  
 **@Test  
 public void** login() **throws** IOException, InterruptedException {  
 *// declare ArrayList to get column 1 data* ArrayList <String> getCol1 = **new** ArrayList <String> ();  
 *// declare ArrayList to get column 2 data* ArrayList <String> getCol2 = **new** ArrayList <String> ();  
 *exldr*.get (**"https://www.everything5pounds.com/en/login"**);  
 *// get data from column1* getCol1 = **getData**.getcol1 ();  
 *// get data from column2* getCol2 = **getData**.getcol2 ();  
 WebElement uidtxt = *exldr*.findElement (By.*id* (**"j\_username"**));  
 WebElement pwdtxt = *exldr*.findElement (By.*id* (**"j\_password"**));  
 *// use for to till size of array and sent data from index 0* **for** (**int** i = 0; i <= getCol1.size (); i++) {  
 uidtxt.sendKeys (getCol1.get (i));  
 *exldr*.manage ().timeouts ().implicitlyWait (5, TimeUnit.***SECONDS***);  
 uidtxt.clear ();  
  
 pwdtxt.sendKeys (getCol2.get (i));  
 *exldr*.manage ().timeouts ().implicitlyWait (5, TimeUnit.***SECONDS***);  
 uidtxt.clear ();  
  
 } }  
 **@Test  
public void** loginParameters() **throws** IOException {  
  
 *// similar to above method but this parameterized and code reusability.* ArrayList <String> collist1 = **new** ArrayList <String> ();  
 ArrayList <String> collist2 = **new** ArrayList <String> ();  
  
 collist1 = **getData**.addExcelData (0);  
 collist2 = **getData**.addExcelData (1);  
  
 *exldr*.get (**"https://www.everything5pounds.com/en/login"**);  
  
 WebElement uidtxt = *exldr*.findElement (By.*id* (**"j\_username"**));  
 WebElement pwdtxt = *exldr*.findElement (By.*id* (**"j\_password"**));  
  
 **for** (**int** i = 0; i <= collist1.size ();i++)  
 {  
 uidtxt.sendKeys (collist1.get (i));  
 *exldr*.manage ().timeouts ().implicitlyWait (10, TimeUnit.***SECONDS***);  
 uidtxt.clear ();  
  
 pwdtxt.sendKeys (collist2.get (i));  
 *exldr*.manage ().timeouts ().implicitlyWait (10,TimeUnit.***SECONDS***);  
 pwdtxt.clear ();  
  
 }  
 *exldr*.quit ();  
  
}  
// code for slider bar

public void usingDragAndDrop() {  
 WebElement barLength = wDriver.findElement (By.xpath ("//div[@id='slider-spb']"));  
 int priceRange=barLength.getSize ().getWidth ();  
 WebElement slideBar = wDriver.findElement (By.xpath ("//\*[@id='slider-pb']/div[1]/a[1]"));  
 Actions moveSlider = new Actions (wDriver);  
 moveSlider.dragAndDropBy (slideBar, 262, 0).build ().perform ();  
}

public void usingClickAndHold () {  
 WebElement slider = wDriver.findElement (By.xpath ("//[@id='slider-spb']"));  
 Actions action = new Actions (wDriver);  
 action.clickAndHold (slider).moveByOffset (150, 0).release ().perform ();  
  
}

**JavaScriptExecutor Interface:**

Java cannot be used to handle some of the features, so we use JavaScript in our WebDriver script. We do not have to write separate code, we can have predefined interface. JavaScriptExecutor is in org.openqa.Selenium.JavaScriptExecutor.

Method executeScript() we need to pass as String, it will executed by JavaScriptExecutor.

* Enter text without using sendKeys(“ hello”) method.
* JavascriptExecutor js=(JavascriptExecutor)*driver*;  
  js.executeScript(**"arguments[0].value=’hello’;”,element"**);
* ScrollIntoView(true)
* JavascriptExecutor js=(JavascriptExecutor)*driver*;  
  js.executeScript(**"arguments[0].scrollIntoView(true);"**,element);
* To click()
* JavascriptExecutor js=(JavascriptExecutor)*driver*;  
  js.executeScript(**"arguments[0].click();"**,element);
* Scroll up and down

javaScriotExecutor js=(JavaScriptExecutor)diver;

js.executeScript(“window.scrollBy(0,350);”,webelementName); [ + is down , - is up ]