## What is Selenium

Selenium is one of the most widely used open source Web UI (User Interface) automation testing suite. Selenium supports automation across different browsers, platforms and programming languages.

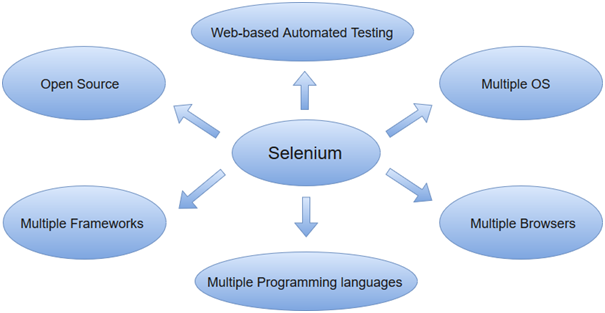
Selenium can be easily deployed on platforms such as Windows, Linux, Solaris and Macintosh. Moreover, it supports OS (Operating System) for mobile applications like iOS, windows mobile and android.

Selenium supports a variety of programming languages through the use of “drivers” specific to each language.Languages supported by Selenium include C#, Java, Perl, PHP, Python and Ruby. Selenium test scripts can be coded in any of the supported programming languages and can be run directly in most modern web browsers.

Browsers supported by Selenium include Internet Explorer, Mozilla Firefox, Google Chrome and Safari.

Selenium can be used to automate functional tests and can be integrated with automation test tools such as **Maven**, **Jenkins**, **& Docker** to achieve continuous testing.

It can also be integrated with tools such as **TestNG**, & **JUnit** for managing test cases and generating reports.



.

Selenium is a set of different software tools. Each tool has a different approach in supporting web based automation testing.

It has four components namely,

i. Selenium IDE (Integrated Development Environment)

ii. Selenium RC (Remote Control)

iii. Selenium WebDriver

iv. Selenium Grid

## Selenium Integrated Development Environment (IDE)

Selenium IDE is a record and playback automation tool using which we can automate the web applications. It comes in the form of a Chrome and Firefox browser extension.

It is the simplest framework in the Selenium Suite.

It allows us to record and playback the scripts. Even though we can create scripts using Selenium IDE, we need to use Selenium RC or Selenium WebDriver to write more advanced and robust test cases.

## Selenium Remote Control(RC)

Selenium Remote Control(RC) is officially deprecated by Selenium. It used to inject the javascript code in the browser for automation and required an additional server for running the automation scripts.

Apart from this, it had many limitations like – it was slow, it didn’t have a headless browser like HtmlUnitDriver and before executing the test scripts the server was required to be started.

## Selenium Grid

It helps in the distributed running of Selenium tests in parallel across multiple remote machines.

It has a hub and multiple nodes. The nodes run the Selenium instances on which the test cases are executed. These nodes are connected to a central hub which acts as a server and controls the whole test execution.

## Selenium WebDriver

Selenium WebDriver AKA Selenium 2 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.

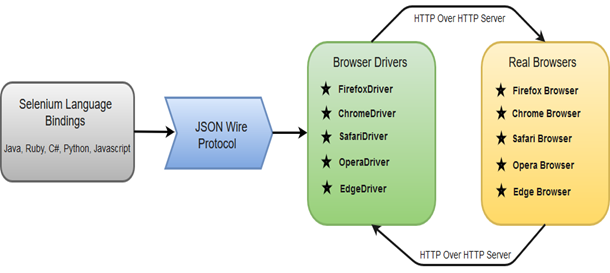
## Limitations of Selenium

* Selenium does not provide desktop application automation support.
* Web Services – REST or SOAP cannot be automated using Selenium.
* Selenium WebDriver requires programming language requirements for script creation.
* For performing common tasks required in automation like logging, reading-writing to external files, we have to rely on external libraries.

## Selenium WebDriver- Architecture

Selenium WebDriver API provides communication facility between languages and browsers.

The following image shows the architectural representation of Selenium WebDriver.



here are four basic components of WebDriver Architecture:

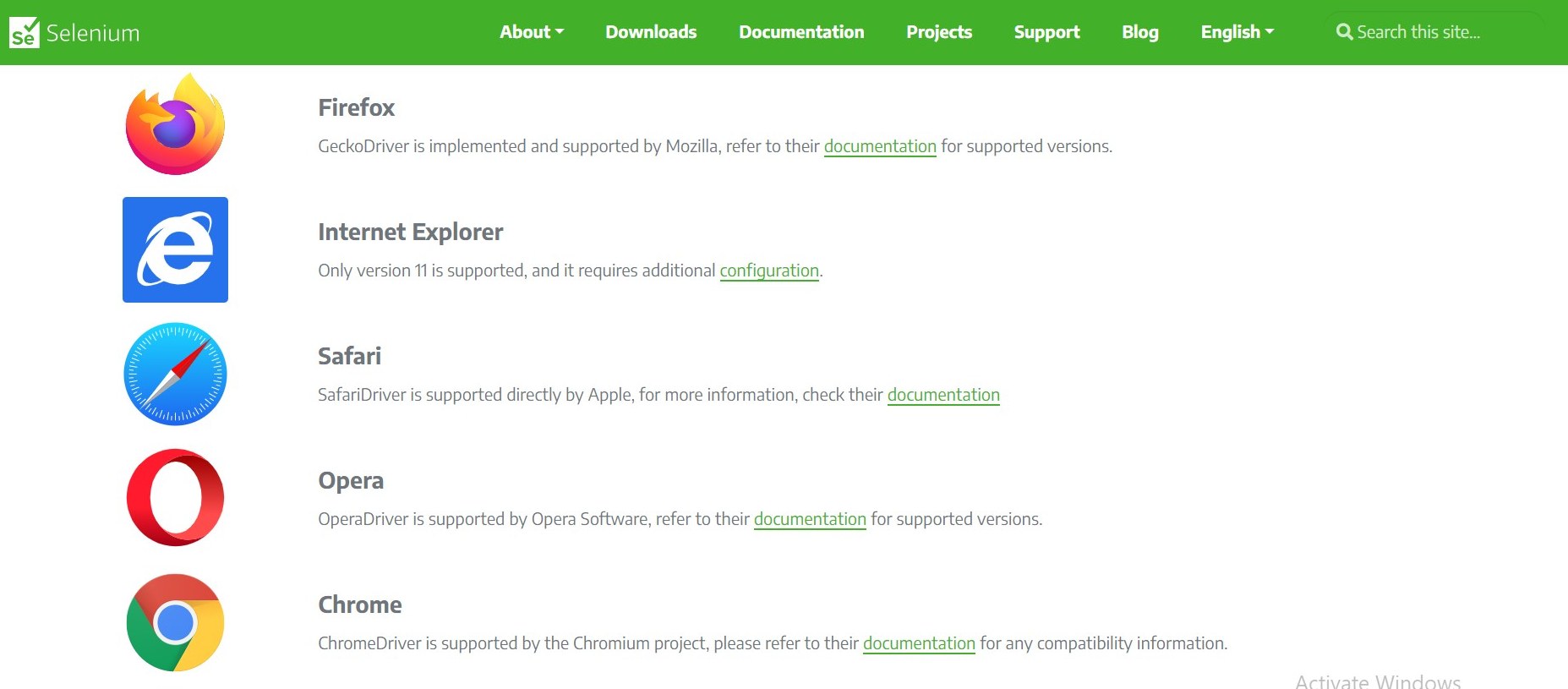
* Selenium Language Bindings: Selenium developers have built language bindings/Selenium Client Libraries in order to support multiple languages. For instance, if you want to use the browser driver in java, use the java bindings.
* JSON Wire Protocol: JSON (JavaScript Object Notation) is an open standard for exchanging data on web. It supports data structures like object and array. So, it is easy to write and read data from JSON. JSON Wire Protocol provides a transport mechanism to transfer data between a server and a client. JSON Wire Protocol serves as an industry standard for various REST web services.
* Browser Drivers: Selenium uses drivers specific to each browser in order to establish a secure connection with the browser without revealing the internal logic of the browser's functionality.
* Real Browsers.

## Selenium WebDriver SETUP

* Configuring Java on a machine.
* Download eclipse or any Java IDE of your choice.
* Create java project in eclipse
* Download Selenium server jar file from “<https://www.selenium.dev/downloads/>”
* Include Selenium server jar file in your project. Follow the below given step

Open eclipse > right click on project > Build path > configure build path > libraries > Add library > select the “selenium-server.jar” file

* Download the webdriver of the browser you want to execute test on from “<https://www.selenium.dev/downloads/>”



* To execute your webdriver, extract the downloaded zip file and give the path of the folder in which the webdriver is existing with the webdriver name e.g. “chromedriver.exe” in the “System.setProperty()” method.

e.g.

System.setProperty("webdriver.chrome.driver", "E:\\BrowserDriver\\chromedriver.exe");

Selenium WebDriver SETUP program

## 

## Launching Browsers in Selenium

Selenium WebDriver calls the native methods of the different browsers to automate them. Hence, in Selenium we have different WebDrivers for different browsers like – FirefoxDriver for Firefox browser, ChromeDriver for Google Chrome, InternetExplorerDriver for Internet Explorer, etc.

Now let’s take an example of launching a Firefox browser and understand the command in detail-

WebDriver driver = new FirefoxDriver();

This is the java implementation of launching a browser in Selenium. Here, ‘WebDriver’ is an interface and we are creating a reference variable ‘driver’ of type WebDriver, instantiated using ‘FireFoxDriver’ class.

An interface contains a set of variables and methods without anybody(no implementation, only method name, and signature). We cannot instantiate objects from interfaces. Hence, the below line of code is incorrect and throws a compile-time error saying “Cannot instantiate the type WebDriver”.

WebDriver driver = new WebDriver(); //Cannot instantiate the type WebDriver error will thrown

For instantiation of driver objects, we need classes like FirefoxDriver or ChromeDriver which have implemented the WebDriver interface. In other words, these driver classes have followed the contract of WebDriver by implementing all the methods of the WebDriver interface. Thus making all the different types of driver classes uniform, following the same protocol.

Please note that we can also create a reference variable of type FirefoxDriver like this-

FirefoxDriver driver = new FirefoxDriver();

But having a WebDriver reference object helps in multi-browser testing as the same driver object can be used to assign to any of the desired browser-specific drivers.

## Launching Firefox Browser

Firefox is one of the most widely used browsers in automation. The following steps are required to launch the firefox browser.

1. Download “geckodriver.exe” from [GeckoDriver Github Release Page](https://github.com/mozilla/geckodriver/releases). Make sure to download the right driver file based on your platform and OS version.
2. Set the System Property for “webdriver.gecko.driver” with the geckodriver.exe path –

System.setProperty(“webdriver.gecko.driver”,”geckodriver.exe path”);

Code snippet to launch the firefox browser-



## Launching Chrome Browser

For running the Chrome browser in Selenium, we need to set the webdriver.chrome.driver system property to point to a chrome driver executable file-

1. Download the latest ChromeDriver binary from [Chromium.org download page](https://sites.google.com/a/chromium.org/chromedriver/downloads) and place the executable on your local machine.
2. Set the webdriver.chrome.driver property to the chromeDriver.exe’s location as-  
   System.setProperty(“webdriver.chrome.driver”, “chromeDriver.exe path”);

Code snippet to launch the Chrome browser-



## Launching Internet Explorer Browser

Like ChromeDriver, InternetExplorer driver also requires setting up the “webdriver.ie.driver” property with the location of IEDriverServer.exe. The IEDriverServer.exe can be downloaded from [here](https://selenium-release.storage.googleapis.com/index.html). Following code, snippet can be used to launch IE browser-



## Launching Safari Browser

The Safari browser doesn’t require any additional configuration and can be directly launched by instantiating with SafariDriver.

The following code snippet can be used to launch the Safari browser-



| Finding Web Elements |
| --- |

A simple automation process in Selenium can be presented as-

* Launching browser
* Opening the desired website to be automated
* Locating web elements like a textbox
* Performing operations on the located web elements like writing in the textbox
* Performing assertion like checking ‘Success’ message

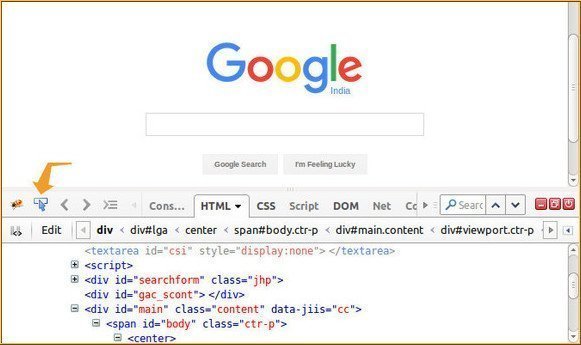
Now, let’s see a sample code snippet implementing the above process.



## Using Developer Tool

Locating web elements requires knowledge of their HTML attributes. For the HTML source code of specific elements, we can use the inbuilt developer tool (launched by pressing f12 in a browser).

Steps for finding element’s HTML attributes-

* Launch the website to be automated e.g. – https://www.google.com
* Press F12 to launch the developer tool.
* Click on the inspect-element icon as displayed in the image below.
* After clicking on the inspect-element icon, click on the web element to be located e.g. Google Search box. Once we click on the element, its HTML will get displayed in the firebug UI. 
* Here, we can see the different attributes of the web elements like – id, class, name, along with its tag like input, div, etc. Now, we will be using these tags, attributes, and values to locate elements using different locators.

## Locators in Selenium

There are a total of 8 locators in Selenium WebDriver-

1. **By Id** – Locates element using id attribute of the web element.

WebElement element = driver.findElement(By.id("elementId"));

2. **By className** – Locates the web element using className attribute.

WebElement element = driver.findElement(By.className("objectClass"));

3. **By tagName** – Locates the web element using its HTML tag like div, a, input etc.

WebElement element = driver.findElement(By.tagName("a"));

4. **By name** – Locates the web element using name attribute.

WebElement element = driver.findElement(By.name("male"));

5. **By linkText** – Locates the web element of link type using their text.

WebElement element = driver.findElement(By.linkText("Click Here"));

6. **By partialLinkText** – Locates the web element of link type with partial matching of text.

WebElement element = driver.findElement(By.partialLinkText("Click"));

7. **By cssSelector** – Locates the web element using css its CSS Selector patterns(explained in detailed here – [CSS Locators](https://artoftesting.com/css-selector-in-selenium-webdriver)).

WebElement element = driver.findElement(By.cssSelector("div#id"));

8. **By xpath** – Locates the web element using its XPaths(explained in detailed here [XPath Locators](https://artoftesting.com/xpath-in-selenium-tutorial)).

WebElement element = driver.findElement(By.xpath("//div[@id='id']"));

## Selenium WebDriver Commands

### Opening a URL

#### Using Get method-

The driver.get() method is used to navigate to a web page by passing the string URL as parameter. Syntax-

driver.get("<https://artoftesting.com>");

#### Using Navigate method-

The driver.navigate().to() method does the task of opening a web page like driver.get() method. Syntax-

driver.navigate().to("https://artoftesting.com");

### Clicking on WebElements

The click() method in Selenium is used to perform the click operation on web elements. In our previous tutorial, [Locators in Selenium WebDriver](https://artoftesting.com/locators-in-selenium-webdriver), we studied about locating the webElements in Selenium. The click() method is applied on the webElements identified, to perform the click operation.

//Clicking an element directly

driver.findElement(By.id("button1")).click();

//Or by first creating a WebElement and then applying click() operation

WebElement submitButton = driver.findElement(By.id("button2"));

submitButton.click();

### Writing in a Textbox

The sendKeys() method can be used for writing in a textbox or any element of text input type.

//Creating a textbox webElement

WebElement element = driver.findElement(By.name("q"));

//Using sendKeys to write in the textbox

element.sendKeys("ArtOfTesting!");

### Clearing text in a Textbox

The clear() method can be used to clear the text written in a textbox or any web element of text input type.

//Clearing the text written in text fields

driver.findElement(By.name("q")).clear();

### Fetching text written over any web element

In automation, many times we need to fetch the text written over a web element for performing some assertions or debugging. For this, we have getText() method in selenium webDriver.

//Fetching the text written over web elements

driver.findElement(By.id("element123")).getText();

## Selenium Waits

In UI automation, waits are required because certain elements get loaded on the page asynchronously, so after triggering an event a page may get loaded successfully but some of its elements may still not get loaded.

This causes an “elementNotFound” exception while locating the element. In such cases, we are left with using Thread.sleep() i.e. a static wait that will halt the test execution for some specified time and then perform the next step.

As Thread.sleep() will wait for the specified time no matter if the elements get visible before that time. So, using Thread.sleep() is never advisable in UI automation.

To avoid this Selenium provides different types of waits, out of which Implicit and Explicit waits are most commonly used.

### implicit Waits

The main function of implicit Wait is to tell the web driver to wait for some time before throwing a **"No Such Element Exception".** Its default setting is knocked at zero. Once the time is set, the driver automatically will wait for the amount of time defined by you before throwing the above-given exception.

**Syntax:**

1. driver.manage().timeouts().implicitlyWait(TimeOut,
2. TimeUnit.SECONDS);

### Explicit Waits

Explicit Waits also known as Dynamic Waits because it is highly specific conditioned. It is implemented by WebDriverWait class.

Unlike implicit waits, the explicit waits are applied to each and every web element. In explicit wait, certain conditions are defined for which the WebDriver instance waits before locating web elements or performing actions on them. Some of the most common conditions specified in explicit waits are-

elementToBeClickable, presenceOfElementLocated etc.

WebDriverWait wait = new WebDriverWait(driver, 15);

wait.until(ExpectedConditions.presenceOfElementLocated(ElementLocator));

Here the WebDriver instance will wait until the condition specified is met i.e. the presence Of Element located by the ElementLocator with the maximum wait time of 15 seconds after which if the condition is still not met then it will throw an exception.

## Handling drop-downs in Selenium

### Select in Selenium WebDriver

The ‘Select’ class in Selenium WebDriver is used for selecting and deselecting the option in a dropdown. The objects of Select type can be initialized by passing the dropdown webElement as parameter to its constructor.

WebElement testDropDown = driver.findElement(By.id("testingDropdown"));

Select dropdown = new Select(testDropDown);

### Selecting options from dropdown

There are three ways of selecting options from dropdown-

1. **selectByIndex** – To select an option based on its index, beginning with 0.

dropdown.selectByIndex(3);

2. **selectByValue** – To select an option based on its ‘value’ attribute.

dropdown.selectByValue("Database");

3. **selectByVisibleText** – To select an option based on the text over the option.

dropdown.selectByVisibleText("Database Testing");

### Different utility methods in the Select class

* **deselectAll()** – To deselect all the selected options.
* **deselectByIndex(int index)** – To deselect the option based on its index.
* **deselectByValue(String valueAttribute)** – To deselect the option its ‘value’ attribute.
* **deselectByVisibleText(String text)** – To deselect the option based on the text over the option.
* **getOptions()** – To return list of all the options(List<WebElement>).
* **getAllSelectedOptions()** – To return the list of all the selected options(List<WebElement>).
* **getFirstSelectedOption()** – To return the selected option or the first selected option in case of dropdowns allowing multi-select.
* **isMultiple()** – To return a boolean value, checking if the dropdown allows multiple option select or not.

### Here is the complete code



# 

## Handling Alerts

Alerts can be handled in Selenium WebDriver using the Alert interface. Selenium WebDriver provides three methods to accept and reject the Alert depending on the Alert types.

### 1. void dismiss()

This method is used to click on the 'Cancel' button of the alert.

**Syntax:**

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

### 2. void accept()

This method is used to click on the 'Ok' button of the alert.

**Syntax:**

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

### 3. String getText()

This method is used to capture the alert message.

**Syntax:**

1. driver.switchTo().alert().getText();

### 4. void sendKeys(String stringToSend)

This method is used to send some data to the alert box.

**Syntax:**

1. driver.switchTo().alert().sendKeys("Text");



## Press Keys in Selenium – ENTER, TAB, SPACE, CONTROL, ARROW, FUNCTION Keys

### 

### Press Enter/Return Key in Selenium

For pressing Enter key over a textbox we can pass “Keys.ENTER” or “Keys.RETURN” to the sendKeys method for that textbox.

WebElement textbox = driver.findElement(By.id("idOfElement"));

textbox.sendKeys(Keys.ENTER);

Or

WebElement textbox = driver.findElement(By.id("idOfElement"));

textbox.sendKeys(Keys.RETURN);

Similarly, we can use Keys enum for different non-text keys and pass them to the sendKeys method. The following table has an entry for each of the non-text key present in a keyboard.

**Keyboard’s Key Keys enum’s value**

Arrow Key – Down Keys.ARROW\_DOWN

Arrow Key – Up Keys.ARROW\_UP

Arrow Key – Left Keys.ARROW\_LEFT

Arrow Key – Right Keys.ARROW\_RIGHT

Backspace Keys.BACK\_SPACE

Ctrl Key Keys.CONTROL

Altkey Keys.ALT

DELETE Keys.DELETE

Enter Key Keys.ENTER

Shift Key Keys.SHIFT

Spacebar Keys.SPACE

Tab Key Keys.TAB

Equals Key Keys.EQUALS

Esc Key Keys.ESCAPE

Home Key Keys.HOME

Insert Key Keys.INSERT

PgUp Key Keys.PAGE\_UP

PgDn Key Keys.PAGE\_DOWN

Function Key F1 Keys.F1

Function Key F2 Keys.F2

Function Key F3 Keys.F3

Function Key F4 Keys.F4

Function Key F5 Keys.F5

Function Key F6 Keys.F6

Function Key F7 Keys.F7

Function Key F8 Keys.F8

Function Key F9 Keys.F9

Function Key F10 Keys.F10

Function Key F11 Keys.F11

Function Key F12 Keys.F12

## Refresh a webpage in selenium

### Multiple other ways to refresh a page

#### 1) Get method:

Get method can be used in a recursive way to refresh a page. In order to achieve this, we need to pass another method as an argument to the get method.

### **Example:**

driver.get("https://www.guru99.com");

driver.get(driver.getCurrentUrl());

## **2) Navigate method:**

This method uses the same concept of recursion as mentioned above. getCurrentURL() method is passed as an argument to driver.navigate.to method.

### **Example:**

driver.get("https://www.guru99.com");

driver.navigate().to(driver.getCurrentUrl());

## **3) Send Keys method using F5 Key:**

This is the second most commonly used method to refresh a page in Selenium. It takes the refresh key (F5 Key) as an argument to the send keys method. Since send keys work only on web elements rather than the browser, we must initially identify a valid web element on the web page and then use the send keys method. This can be accomplished as shown below.

### **Example:**

driver.get("https://www.guru99.com");

driver. findElement(By.id("username")).sendKeys(Keys.F5);

## **4) Send Keys method using ASCII Code:**

This method uses the same concept as above, but instead of passing the F5 key as an argument, we send the ASCII Code of refresh key as an argument. This can be accomplished as shown below.

driver.get("https://www.guru99.com");

driver. findElement(By.id("username")).sendKeys(“\uE035”);

## **Actions Class Method for Keyboard Interaction**

1. **keyDown(Keys modifierKey)-**The keyDown(Keys modifierKey) method takes the modifier Keys as parameter (Shift, Alt and Control Keys – that modifies the purpose of other keys, hence the name). It is used to simulate the action of pressing a modifier key, without releasing. The expected values for the keyDown() method are – Keys.SHIFT, Keys.ALT and Keys.CONTROL only, passing key other than these results in IllegalArgumentException.
2. **keyDown(WebElement element, Keys modifierKey)-**This another implementation of keyDown() method in which the modifier key press action is performed on a WebElement.
3. **keyUp(Keys modifierKey)-**The keyUp() method is used to simulate the modifier key-up or key-release action. This method follows a preceeding key press action.
4. **keyUp(WebElement element, Keys modifierKey)-**This implementation of keyUp() method performs the key-release action on a web element.
5. **sendKeys(CharSequence KeysToSend)-**The sendKeys(CharSequence KeysToSend) method is used to send a sequence of keys to a currently focussed web element. Here, we need to note that it is different from the webElement.sendKeys() method. The Actions sendKeys(CharSequence KeysToSend) is particularly helpful when dealing with modifier keys as it doesn’t release those keys when passed(resulting in correct behaviour) unlike the webElement.sendKeys() method.
6. **sendKeys(WebElement element, CharSequence KeysToSend)-**This implementation of sendKeys() method is used to send a sequence of keys to a web element.