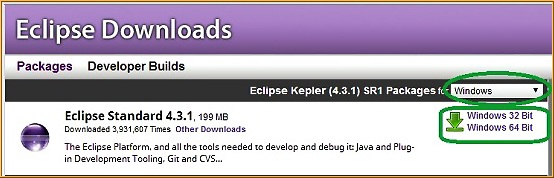
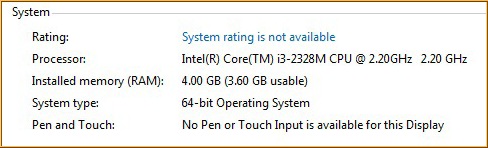
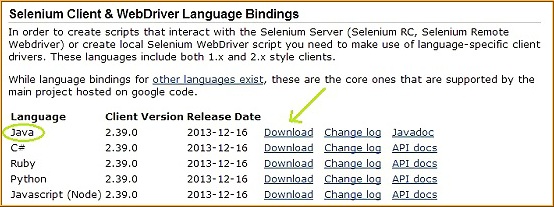
**Step.1. Configuring Java on machine**  
Download latest version of Java Development Kit (JDK) from [here](http://www.oracle.com/technetwork/java/javase/downloads/index.html#_blank). Follow the steps mentioned to install JDK. We are installing JDK as it will be required for developing and running our automation scripts which are nothing but java programs.  
  
**Step.2. Download eclipse or any Java IDE of your choice**  
Download the latest version of the Java IDE you would like to use, for Eclipse the download link is [http://www.eclipse.org/downloads/](http://www.eclipse.org/downloads/#_blank)

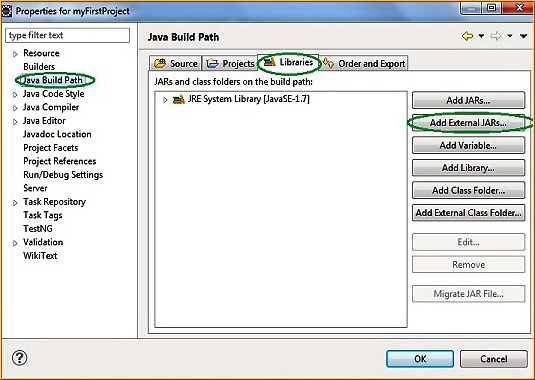
Select the appropriate version of Eclipse depending on your system type- 34 bit or 64 bit.  
  
In order to check your system type- go to my computer and right click and select 'properties', under system section you will see a 'System type' parameter. This parameter will define your system type.  


**Step.3. Download Selenium Webdriver jar from SeleniumHQ website**  
Go to SeleniumHQ website- <http://docs.seleniumhq.org/download/>and under "Selenium Client & WebDriver Language Bindings" download the webdriver for java.

Unzip the package and place it on any directory as a library folder.  
  
**Step.4. Creating project and configuring selenium jars-**

* Launch eclipse.exe
* Set your workspace to any location preferably other than C:(a workspace is a physical location where we store our project or group of related projects).
* Now create a new project- File->New->Project...->Java->Java Project
* Name you project and click Finish
* Now you will see ansrc folder under your project. Under this we need to create a package-Right Click src->New->Package (Basically these packages are used to group together related classes). Name your package e.g. 'myTestPackage'
* Inside this package create a new class and name it e.g. Test, your Test.java class will get created

**Step.5. Adding selenium jars**  
Right Click your project on the left and click on properties. A "Properties for {project name}" dialog box will appear. Click on "Java Build Path" on the left and then click on Libraries tab on the right. In this tab click on "Add External Jars.." button.



Now browse to the location where selenium libraries are placed(library folder Step#3). Make sure to add both the libraries-selenium-java-2.39.0.jar and selenium-java-2.39.0-srcs.jar along with the libraries present in libs folder(selenium-2.39.0\libs). The selected libraries will appear, click OK to add these libraries to your project. You can verify the same in the "Referenced Libraries" section under your project in the "Package Explorer" section on the left.  
  
**Step.6. Creating first selenium webdriver project**  
Time to test the steup. Now, we will create our first selenium project, in which we will just open Firefox browser and launch a website. 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 Chrome browser-

**publicclassFirefoxBrowserLaunchDemo**{

**publicstaticvoid**main(String[]args){

*//Creating a driver object referencing WebDriver interface*

WebDriver driver;

*//Setting webdriver.gecko.driver property*

System.setProperty("webdriver.gecko.driver", "{path to geckodriver}\\geckodriver.exe");

*//Instantiating driver object and launching browser*

driver=**new**FirefoxDriver();

*//Using get() method to open a webpage*

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

*//Closing the browser*

driver.quit();

}

}

To run the test, right click on Test.java file on the Package Explorer section, hover over "Run As" and select "Java Application". Firefox broswer will launch and open [artoftesting.com](http://artoftesting.com/).

**Launching Browsers in Selenium**

In this post, we will study the Selenium WebDriver commands used to launch browsers in detail. We will also learn the different additional customization required for launching certain browsers like - Chrome and InternetExplorer.

**Understanding the browser launching command**

As we have studied in previous tutorials that 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 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.  
For those who are not very proficient in Java, an interface is like a contract that classes implementing it must follow. An interface contains a set of variables and methods without any body(no implementaion, only method name and signature). We cannot instantiate objects from interfaces. Hence, the below line of code is incorrect and throws compile time error saying "Cannot instantiate the type WebDriver".

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

For instantiation of driver object, 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 driver.

**Launching Firefox Browser**

Firefox is one of the most widely used browsers in automation. 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 Chrome browser-

**publicclassFirefoxBrowserLaunchDemo**{

**publicstaticvoid**main(String[]args){

*//Creating a driver object referencing WebDriver interface*

WebDriver driver;

*//Setting webdriver.gecko.driver property*

System.setProperty("webdriver.gecko.driver", pathToGeckoDriver + "\\geckodriver.exe");

*//Instantiating driver object and launching browser*

driver=**new**FirefoxDriver();

*//Using get() method to open a webpage*

driver.get("[http://artoftesting.com](http://artoftesting.com/)");

*//Closing the browser*

driver.quit();

}

}

## Launching Chrome Browser

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

**publicclassChromeBrowserLaunchDemo**{

**publicstaticvoid**main(String[]args){

*//Creating a driver object referencing WebDriver interface*

WebDriver driver;

*//Setting the webdriver.chrome.driver property to its executable's location*

System.setProperty("webdriver.chrome.driverC:\Users\manis\Desktop\Selenium Webdriver Setup\\chromedriver.exe");

*//Instantiating driver object*

driver=**new**ChromeDriver();

*//Using get() method to open a webpage*

driver.get("[http://artoftesting.com](http://artoftesting.com/)");

*//Closing the browser*

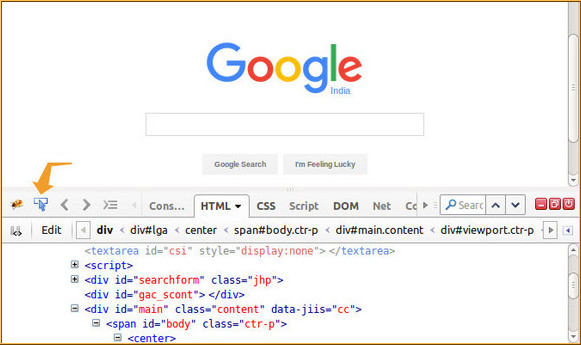
driver.quit();

}

}

## Using Firebug or Developer Tool

Locating web elements requires knowledge of their HTML attributes. For the HTML source code of specific elements, we can use a Mozilla Firefox plugin - firebug or use the inbuilt developer tools. Throughout the course of this tutorial we will use Firebug to locate elements. You can download firebug from here -[Firebug addon for Mozilla](https://addons.mozilla.org/en-US/firefox/addon/firebug/#_blank).  
Steps for finding element's HTML attributes-

* Launch the website to be automated e.g. -[https://www.google.com](https://www.google.com/)
* Press F12 to launch firebug or 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 their 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.

driver.findElement(By.id("login-button"));

1. **By className**- Locates the web element using className attribute.

driver.findElement(By.className("elementsClass"));

1. **By tagName**- Locates the web element using its html tag like div, a, input etc.

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

1. **By name**- Locates the web element using name attribute.

driver.findElement(By.name("male"));

1. **By linkText**- Locates the web element of link type using their text.

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

1. **By partialLinkText**- Locates the web element of link type with partial matching of text.

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

1. **By cssSelector**- Locates the web element using css its CSS Selector patterns(explained in detailed here -[CSS Locators](http://artoftesting.com/automationTesting/css-selector-in-selenium-webdriver.html#_blank)).

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

1. **By xpath**- Locates the web element using its XPaths(explained in detailed here[XPath Locators](http://artoftesting.com/automationTesting/xpath-in-selenium-tutorial.html#_blank)).

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

# Selenium Web Driver Commands

In this post, we will learn some of the basic selenium commands for performing operations like opening a URL, clicking on buttons, writing in textbox, closing the browser etc. For your practice, a [dummy webpage](http://www.artoftesting.com/sampleSiteForSelenium.html#_blank)with different types of web elements is available. Now, let's see the basic commands of Selenium WebDriver.

## 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("[http://artoftesting.com](http://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("[http://artoftesting.com](http://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](http://artoftesting.com/automationTesting/locatorsInSelenium.html), 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*

driver.findElement(By.id("login")).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();

## Closing browser

Selenium provides two commands to close browsers close() and quite(). The driver.close() command is used to close the browser having focus. Whereas, the driver.quite command is used to close all the browser instances open.

*//To close the current browser instance*

driver.close();

*//To close all the open browser instances*

driver.quit();