**DAY 3 – SELENIUM BASICS ASSIGNMENT CONTINUED**

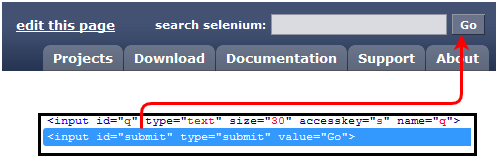
1. **Locate Your Element by ID:**

Format : *id=submit (Where “Submit” is the value of ID locator.)*

You can record the location of web element through Selenium IDE. In below example we will see how we can manually add locator with the use of Firebug.

Example :

– Hit the URL : http://docs.seleniumhq.org/ in Firefox browser.  
– Look for “search selenium” on Top right corner of the screen.  
– Open Firebug and click on Inspect button.  
– Select “Go” button. You will see something like this.



As you can see, id=”submit” has been associated with input button, which we can use as the locator in Selenium IDE.

– Open Selenium IDE and in Target box enter id=submit. Now Click on “Find” button, you’ll see the “Go” button gets highlighted.

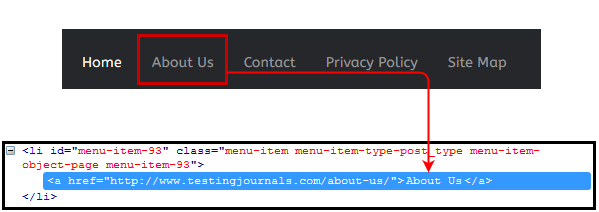
1. **Using Link Text Locator :**

This locator allows you to find the “Link” elements having hyperlink texts. The element is located by providing “link=” into IDE target field followed by hyperlink text.

Format : *link=About Us (Where “About Us” is the hyperlink text.)*

Example :

– Hit the URL : http://www.testingjournals.com in Firefox browser.  
– Right click on “About Us” from the header bar. As it is an hyperlink, you can get the locator value as shown below :



Using above mentioned format you can easily execute the command in IDE. Provide “link=About Us” in Target field and use command as “clickAndWait” and execute the command. You will be redirected to About Us page.

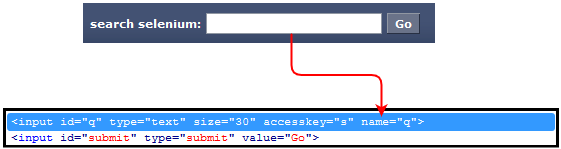
1. **Name Locator :**

Format : *name=q (Where “q” is the Name value assigned to the element.)*

Example :

– Hit the URL : http://docs.seleniumhq.org/ in Firefox browser.  
– Look for “search selenium” on Top right corner of the screen.  
– Open Firebug and click on Inspect button.  
– Select “Search” text-box. You will see something like this.

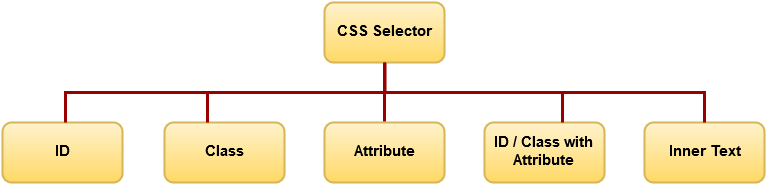
As shown in above image, name=”q” has been attached with search input box.



Same as ID locator, provide “name=” in the IDE target field followed by the value of Name attribute.

1. **CSS Selector :**

CSS locator can be formed in different ways. Most used and common are as shown below :



**CSS selector with Element ID :**

First step would be same as what we did in locating ID earlier in this tutorial. We will use the same example here.

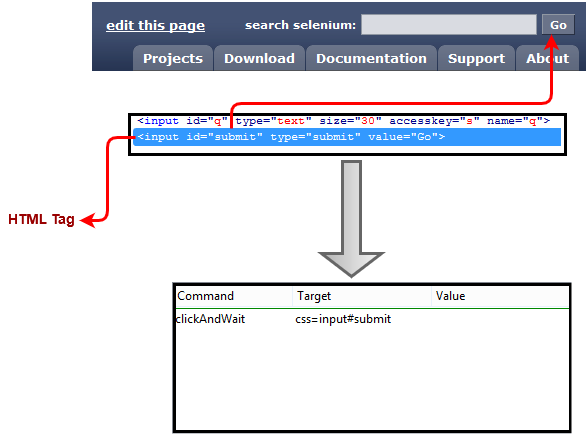
Format : *css=[HTML Tag]#[Element ID]*where,

HTML Tag = the HTML Tag of the interacted Element  
# = This Hash Tag should be present for each CSS locator using ID  
ID = ID locator of the interacted Element

Example :

– Hit the URL: http://docs.seleniumhq.org/ in Firefox browser.  
– Look for “search selenium” on Top right corner of the screen.  
– Open Firebug and click on Inspect button.  
– Select “Go” button.

Mentioned below is the generated CSS Locator  of “Go” button using Element’s ID. Don’t forget to use Find button to verify you have located proper element.



**CSS selector with Class :**

This CSS selector works with the class attribute of the web element. To showcase the example we will use Testing Journal’s global search button.

Format : *css=[HTML Tag].[Element Class Name]* where,

HTML Tag = the HTML Tag of the interacted Element  
. = This Dot should be present for each CSS locator using Class Name  
Class Name = Class name of the interacted Element which you can extract through Firebug

Example :

– Hit the URL : http://www.testingjournals.com  
– Open Firebug and locate the Search field from the Right side bar  
– Now generate the CSS locator for Search input field as mentioned in below image.



Hence, as mentioned in the above image we can generate the CSS locator using Class Name.

Note : Many times multiple elements have been allocated the same HTML Tag and Name. In such cases, the first element from the source code will be located. So, it better to avoid using Name in such cases.

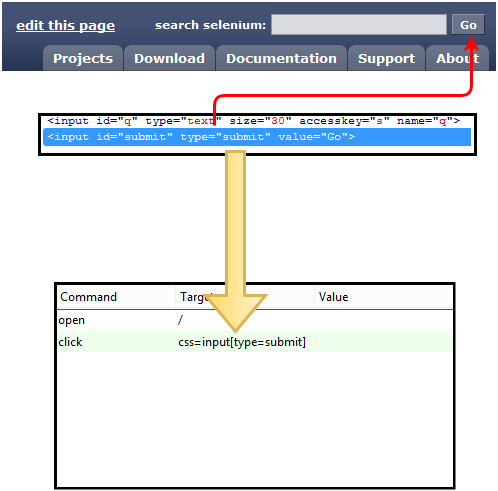
**CSS Selector with Attribute :**

Format : *css=HTML Tag[Attribute Value]* where,

HTML Tag = the HTML Tag of the interacted Element  
[] = Attribute value should be given in this bracket  
Attribute Value = Attribute value of the interacted Element which you can extract through Firebug

Example :

– Hit the URL : http://docs.seleniumhq.org/ in firefox browser.  
– Look for “search selenium” on Top right corner of the screen.  
– Open Firebug and click on Inspect button.  
– Select “Go” button. Now refer below image to generate CSS selector using element attribute value.



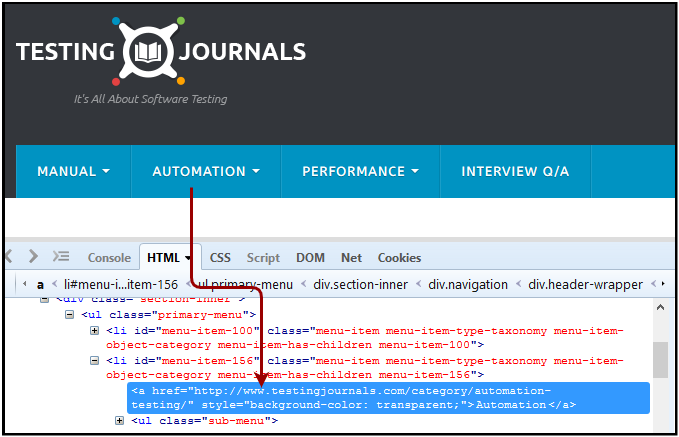
**CSS Selector with Inner Text :**

Format : *css=HTML Tag:contains(“Inner Text”)* where,

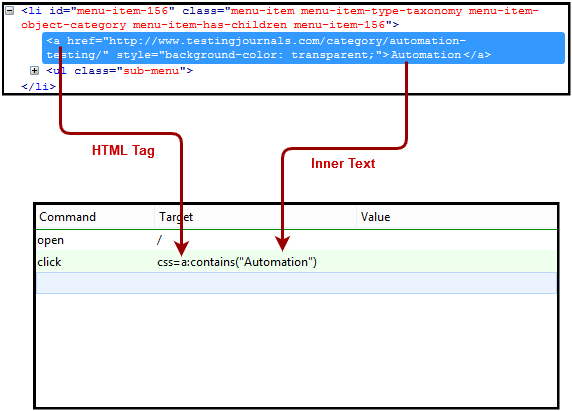
HTML Tag = the HTML Tag of the interacted Element  
:contains = Inner text should be provided in IDE command in this format  
Inner Text = Inner Text of the interacted Element which you can extract through Firebug

Example :

– Hit the URL : http://www.testingjournals.com  
– Open Firebug and locate the “Automation” as the Menu option in the header bar.



As you can see in above image, Inner text for the identified element is “Automation”. Now let’s locate this element through Selenium IDE using CSS with Inner text selector.



**LINK TEXT SELECTOR CODE -**

**import org.openqa.selenium.By;**

**import org.openqa.selenium.WebDriver;**

**import org.openqa.selenium.chrome.ChromeDriver;**

**public class MyClass {**

**public static void main(String[] args) {**

**String baseUrl = "http://demo.guru99.com/test/link.html";**

**System.setProperty("webdriver.chrome.driver","G:\\chromedriver.exe");**

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

**driver.get(baseUrl);**

**driver.findElement(By.linkText("click here")).click();**

**System.out.println("title of page is: " + driver.getTitle());**

**driver.quit();**

**}**

**}**

**PARTIAL LINK TEXT SELECTOR CODE -**

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

public class P1 {

public static void main(String[] args) {

String baseUrl = "http://demo.guru99.com/test/accessing-link.html";

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

WebDriver driver = new ChromeDriver();

driver.get(baseUrl);

driver.findElement(By.partialLinkText("here")).click();

System.out.println("Title of page is: " + driver.getTitle());

driver.quit();

}

}

**USER INTERACTIONS CODE 1- SELENIUM FORMS CODE**

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.\*;

public class Form {

public static void main(String[] args) {

// declaration and instantiation of objects/variables

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

WebDriver driver = new ChromeDriver();

String baseUrl = "http://demo.guru99.com/test/login.html";

driver.get(baseUrl);

// Get the WebElement corresponding to the Email Address(TextField)

WebElement email = driver.findElement(By.id("email"));

// Get the WebElement corresponding to the Password Field

WebElement password = driver.findElement(By.name("passwd"));

email.sendKeys("[abcd@gmail.com](mailto:abcd@gmail.com)

");

password.sendKeys("abcdefghlkjl");

System.out.println("Text Field Set");

// Deleting values in the text box

email.clear();

password.clear();

System.out.println("Text Field Cleared");

// Find the submit button

WebElement login = driver.findElement(By.id("SubmitLogin"));

// Using click method to submit form

email.sendKeys("[abcd@gmail.com](mailto:abcd@gmail.com)");

password.sendKeys("abcdefghlkjl");

login.click();

System.out.println("Login Done with Click");

//using submit method to submit the form. Submit used on password field

driver.get(baseUrl);

driver.findElement(By.id("email")).sendKeys("[abcd@gmail.com](mailto:abcd@gmail.com)");

driver.findElement(By.name("passwd")).sendKeys("abcdefghlkjl");

driver.findElement(By.id("SubmitLogin")).submit();

System.out.println("Login Done with Submit");

//driver.close();

}

}

**USER INTERACTIONS CODE 2 – RADIO BUTTONS**

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.\*;

public class Form {

public static void main(String[] args) {

// declaration and instantiation of objects/variables

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

WebDriver driver = new ChromeDriver();

driver.get("http://demo.guru99.com/test/radio.html");

WebElement radio1 = driver.findElement(By.id("vfb-7-1"));

WebElement radio2 = driver.findElement(By.id("vfb-7-2"));

//Radio Button1 is selected

radio1.click();

System.out.println("Radio Button Option 1 Selected");

//Radio Button1 is de-selected and Radio Button2 is selected

radio2.click();

System.out.println("Radio Button Option 2 Selected");

// Selecting CheckBox

WebElement option1 = driver.findElement(By.id("vfb-6-0"));

// This will Toggle the Check box

option1.click();

// Check whether the Check box is toggled on

if (option1.isSelected()) {

System.out.println("Checkbox is Toggled On");

} else {

System.out.println("Checkbox is Toggled Off");

}

//Selecting Checkbox and using isSelected Method

driver.get("http://demo.guru99.com/test/facebook.html");

WebElement chkFBPersist = driver.findElement(By.id("persist\_box"));

for (int i=0; i<2; i++) {

chkFBPersist.click ();

System.out.println("Facebook Persists Checkbox Status is - "+chkFBPersist.isSelected());

}

//driver.close();

}

}

**USER INTERACTIONS CODE 3 – DROP DOWN BUTTONS**

package newpackage;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

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

import org.openqa.selenium.By;

public class accessDropDown {

public static void main(String[] args) {

System.setProperty("webdriver.firefox.marionette","C:\\geckodriver.exe");

String baseURL = "http://demo.guru99.com/test/newtours/register.php";

WebDriver driver = new FirefoxDriver();

driver.get(baseURL);

Select drpCountry = new Select(driver.findElement(By.name("country")));

drpCountry.selectByVisibleText("ANTARCTICA");

//Selecting Items in a Multiple SELECT elements

driver.get("http://jsbin.com/osebed/2");

Select fruits = new Select(driver.findElement(By.id("fruits")));

fruits.selectByVisibleText("Banana");

fruits.selectByIndex(1);

}

}

**USER INTERACTIONS CODE 4 - MOUSE OVER CODE**

package newproject;

import org.openqa.selenium.\*;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.openqa.selenium.interactions.Action;

import org.openqa.selenium.interactions.Actions;

public class PG7 {

public static void main(String[] args) {

String baseUrl = "http://demo.guru99.com/test/newtours/";

System.setProperty("webdriver.firefox.marionette","C:\\geckodriver.exe");

WebDriver driver = new FirefoxDriver();

driver.get(baseUrl);

WebElement link\_Home = driver.findElement(By.linkText("Home"));

WebElement td\_Home = driver

.findElement(By

.xpath("//html/body/div"

+ "/table/tbody/tr/td"

+ "/table/tbody/tr/td"

+ "/table/tbody/tr/td"

+ "/table/tbody/tr"));

Actions builder = new Actions(driver);

Action mouseOverHome = builder

.moveToElement(link\_Home)

.build();

String bgColor = td\_Home.getCs sValue("background-color");

System.out.println("Before hover: " + bgColor);

mouseOverHome.perform();

bgColor = td\_Home.getCssValue("background-color");

System.out.println("After hover: " + bgColor);

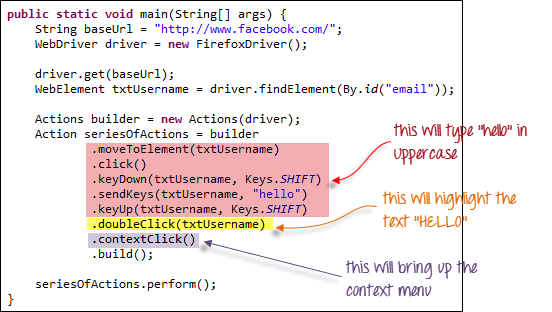
driver.close();

}

}

## **USER INTERACTIONS CODE 5 - MOUSE OVER AND KEY BOARD MULTIPLE ACTIONS**

**You can build a series of actions using the Action and Actions classes**. Just remember to close the series with the build() method. Consider the sample code below.

[](https://www.guru99.com/images/image053.png)

public static void main(String[] args) {

String baseUrl = "http://www.facebook.com/";

WebDriver driver = new FirefoxDriver();

driver.get(baseUrl);

WebElement txtUsername = driver.findElement(By.id("email"));

Actions builder = new Actions(driver);

Action seriesOfActions = builder

.moveToElement(txtUsername)

.click()

.keyDown(txtUsername, Keys.SHIFT)

.sendKeys(txtUsername, "hello")

.keyUp(txtUsername, Keys.SHIFT)

.doubleClick(txtUsername)

.contextClick()

.build();

seriesOfActions.perform() ;

}

**USER INTERACTIONS CODE 6 - WINDOW HANDLE CODE**

import java.util.Iterator;

import java.util.Set;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

public class WindowHandle\_Demo {

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

WebDriver driver=new FirefoxDriver();

//Launching the site.

driver.get("http://demo.guru99.com/popup.php");

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

driver.findElement(By.xpath("//\*[contains(@href,'popup.php')]")).click();

String MainWindow=driver.getWindowHandle();

// To handle all new opened window.

Set<String> s1=driver.getWindowHandles();

Iterator<String> i1=s1.iterator();

while(i1.hasNext())

{

String ChildWindow=i1.next();

if(!MainWindow.equalsIgnoreCase(ChildWindow))

{

// Switching to Child window

driver.switchTo().window(ChildWindow);

driver.findElement(By.name("emailid"))

.sendKeys("[gaurav.3n@gmail.com](mailto:gaurav.3n@gmail.com)

");

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

// Closing the Child Window.

driver.close();

}

}

// Switching to Parent window i.e Main Window.

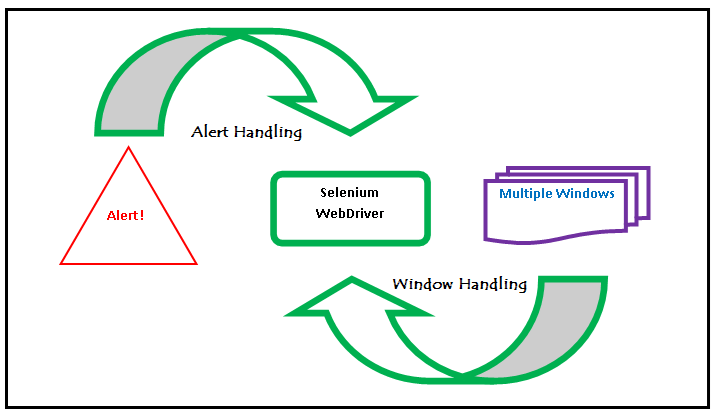
driver.switchTo().window(MainWindow);

}

}

**Output:**

When you execute the above code, it launches the site and on clicking the link "Click here," it opens up a child window in a new tab. You can close the child window, and switch to the parent window once the operation is completely done. Hence handling more than one window in the application.

[](https://www.guru99.com/images/3-2016/032216_1314_AlertPopuph18.png)

**USER INTERACTIONS CODE 7 - UPLOAD FILE CODE**

package newproject;

import org.openqa.selenium.\*;

import org.openqa.selenium.firefox.FirefoxDriver;

public class PG9 {

public static void main(String[] args) {

System.setProperty("webdriver.firefox.marionette","C:\\geckodriver.exe");

String baseUrl = "http://demo.guru99.com/test/upload/";

WebDriver driver = new FirefoxDriver();

driver.get(baseUrl);

WebElement uploadElement = driver.findElement(By.id("uploadfile\_0"));

// enter the file path onto the file-selection input field

uploadElement.sendKeys("C:\\newhtml.html");

// check the "I accept the terms of service" check box

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

// click the "UploadFile" button

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

}

}

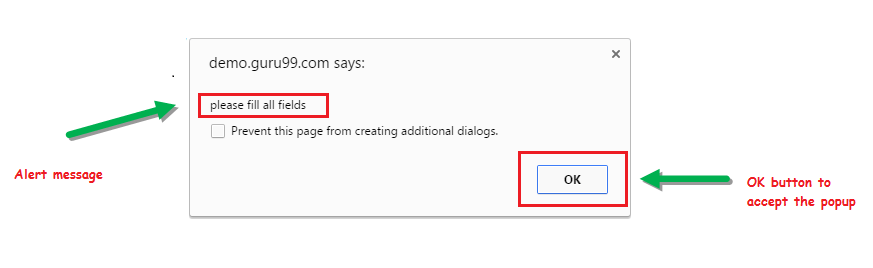
**USER INTERACTIONS CODE 8 – ALERTS**

Alert is a small message box which displays on-screen notification to give the user some kind of information or ask for permission to perform certain kind of operation. It may be also used for warning purpose.

Here are few alert types:

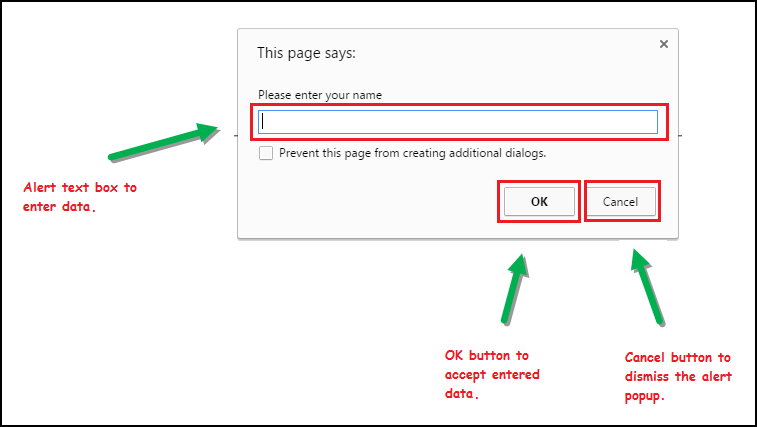
**1) Simple Alert**

This simple alert displays some information or warning on the screen.

[](https://www.guru99.com/images/3-2016/032216_1314_AlertPopuph1.png)

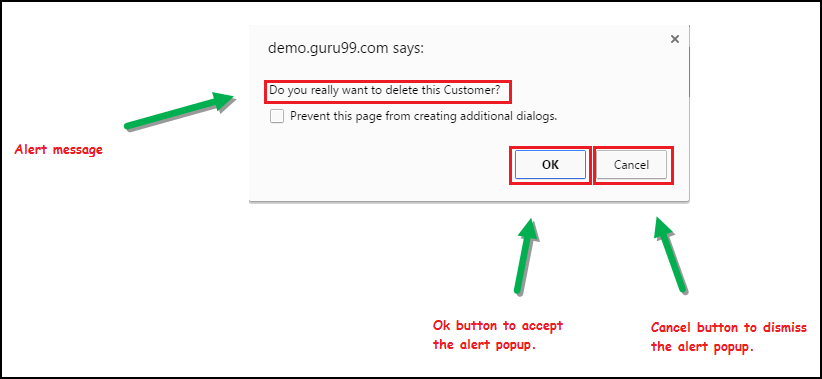
**2) Prompt Alert.**

This Prompt Alert asks some input from the user and selenium webdriver can enter the text using sendkeys(" input…. ").

[](https://www.guru99.com/images/3-2016/032216_1314_AlertPopuph2.png)

**3) Confirmation Alert.**

This confirmation alert asks permission to do some type of operation.

[](https://www.guru99.com/images/3-2016/032216_1314_AlertPopuph3.png)

## How to handle Alert in Selenium WebDriver

Alert interface provides the below few methods which are widely used in Selenium Webdriver.

1) void dismiss() **// To click on the 'Cancel' button of the alert.**

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

2) void accept() **// To click on the 'OK' button of the alert.**

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

3) String getText**() // To capture the alert message.**

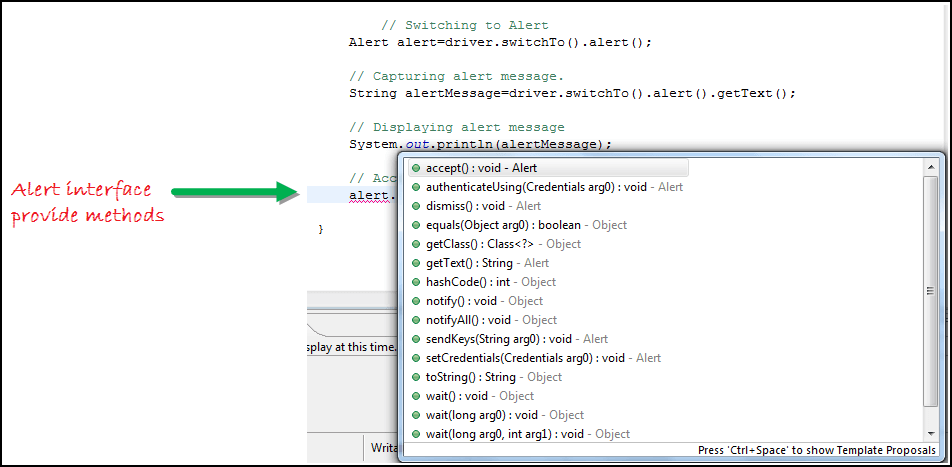
driver.switchTo().alert().getText();

4) void sendKeys(String stringToSend) **// To send some data to alert box.**

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

You can see a number of Alert methods are displayed as shown in below screen suggested by Eclipse.

We can easily switch to alert from the main window by using Selenium's **.switchTo()** method.

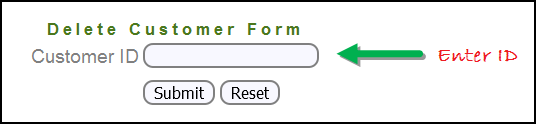
[](https://www.guru99.com/images/3-2016/032216_1314_AlertPopuph4.png)

**Now we automate the given below scenario.**

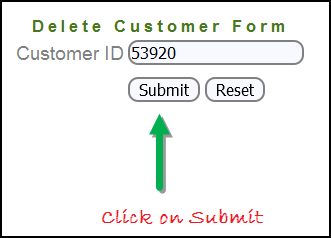
In this scenario, we will use Guru99 demo site to illustrate Selenium Alert handling.

**Step 1)** Launch the web browser and open the site "<http://demo.guru99.com/test/delete_customer.php> "

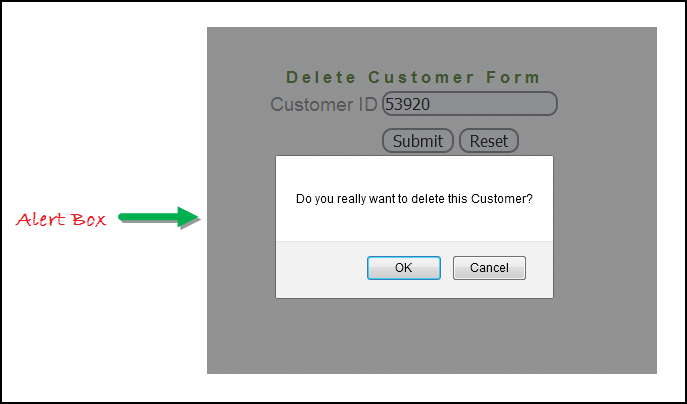
**Step 2)** Enter Any Customer id.

[](https://www.guru99.com/images/3-2016/032216_1314_AlertPopuph8.png)

**Step 3)** After entering the customer ID, Click on the "Submit" button.

[](https://www.guru99.com/images/3-2016/032216_1314_AlertPopuph9.png)

**Step 4)** Reject/accept the alert.

[](https://www.guru99.com/images/3-2016/032216_1314_AlertPopuph10.png)

**Handling Alert in Selenium Webdriver using above scenario**

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.NoAlertPresentException;

import org.openqa.selenium.Alert;

public class AlertDemo {

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

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

WebDriver driver = new ChromeDriver();

// Alert Message handling

driver.get("http://demo.guru99.com/test/delete\_customer.php");

driver.findElement(By.name("cusid")).sendKeys("53920");

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

// Switching to Alert

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

// Capturing alert message.

String alertMessage= driver.switchTo().alert().getText();

// Displaying alert message

System.out.println(alertMessage);

Thread.sleep(5000);

// Accepting alert

alert.accept();

}

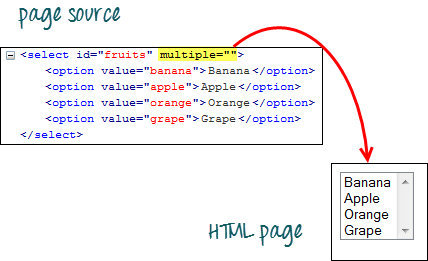
}

**Output:**

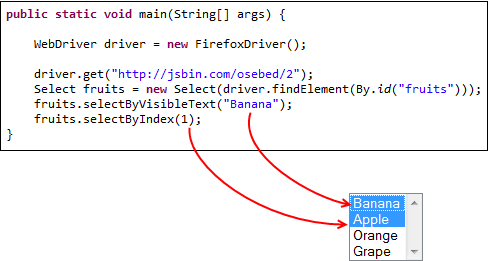
When you execute the above code, it launches the site. Try to delete Customer ID by handling confirmation alert that displays on the screen, and thereby deleting customer id from the application.

**USER INTERACTIONS CODE 9 – MULTI SELECT**

We can also use the **selectByVisibleText()** method in selecting multiple options in a multi SELECT element. As an example, we will take <http://jsbin.com/osebed/2> as the base URL. It contains a drop-down box that allows multiple selections at a time.

[](https://www.guru99.com/images/image015(3).png)

The code below will select the first two options using the selectByVisibleText() method.

[](https://www.guru99.com/images/image016(3).png)

## Select Methods

The following are the most common methods used on drop down list.

|  |  |
| --- | --- |
| **Method** | **Description** |
| **selectByVisibleText()**and **deselectByVisibleText()** Example: [How to Select Option from DropDown using Selenium Webdriver](https://www.guru99.com/images/image017(3).png) | * Selects/deselects the option that displays the text matching the parameter. * **Parameter**: The exactly displayed text of a particular option |
| **selectByValue()** and **deselectByValue()** Example: [How to Select Option from DropDown using Selenium Webdriver](https://www.guru99.com/images/image018(3).png) | * Selects/deselects the option whose "value" attribute matches the specified parameter. * **Parameter**: value of the "value" attribute * Remember that not all drop-down options have the same text and "value", like in the example below.   [How to Select Option from DropDown using Selenium Webdriver](https://www.guru99.com/images/image019(2).png) |
| **selectByIndex()** and **deselectByIndex()** Example: *[How to Select Option from DropDown using Selenium Webdriver](https://www.guru99.com/images/image020(2).png)* | * Selects/deselects the option at the given index. * **Parameter**: the index of the option to be selected. |
| **isMultiple()** Example: *[How to Select Option from DropDown using Selenium Webdriver](https://www.guru99.com/images/image021(2).png)* | * Returns TRUE if the drop-down element allows multiple selections at a time; FALSE if otherwise. * **No parameters needed** |
| **deselectAll()** Example: *[How to Select Option from DropDown using Selenium Webdriver](https://www.guru99.com/images/image022(2).png)* | * Clears all selected entries. This is only valid when the drop-down element supports multiple selections. * **No parameters needed** |

## Here is the complete code

package newpackage;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

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

import org.openqa.selenium.By;

public class accessDropDown {

public static void main(String[] args) {

System.setProperty("webdriver.gecko.driver","C:\\geckodriver.exe");

String baseURL = "http://demo.guru99.com/test/newtours/register.php";

WebDriver driver = new FirefoxDriver();

driver.get(baseURL);

Select drpCountry = new Select(driver.findElement(By.name("country")));

drpCountry.selectByVisibleText("ANTARCTICA");

//Selecting Items in a Multiple SELECT elements

driver.get("http://jsbin.com/osebed/2");

Select fruits = new Select(driver.findElement(By.id("fruits")));

fruits.selectByVisibleText("Banana");

fruits.selectByIndex(1);

}

}

# Implicit, Explicit, & Fluent Wait in Selenium WebDriver

In selenium "Waits" play an important role in executing tests. In this tutorial, you will learn various aspects of both "Implicit" and "Explicit" waits in Selenium.

In this tutorial, you will learn-

* [Why Do We Need Waits In Selenium?](https://www.guru99.com/implicit-explicit-waits-selenium.html#1)
* [Implicit Wait](https://www.guru99.com/implicit-explicit-waits-selenium.html#2)
* [Explicit Wait](https://www.guru99.com/implicit-explicit-waits-selenium.html#3)
* [Fluent Wait](https://www.guru99.com/implicit-explicit-waits-selenium.html#4)

## Why Do We Need Waits In Selenium?

Most of the web applications are developed using Ajax and Javascript. When a page is loaded by the browser the elements which we want to interact with may load at different time intervals.

Not only it makes this difficult to identify the element but also if the element is not located it will throw an "**ElementNotVisibleException**" exception. Using Waits, we can resolve this problem.

Let's consider a scenario where we have to use both implicit and explicit waits in our test. Assume that implicit wait time is set to 20 seconds and explicit wait time is set to 10 seconds.

Suppose we are trying to find an element which has some **"ExpectedConditions** "(Explicit Wait), If the element is not located within the time frame defined by the Explicit wait(10 Seconds), It will use the time frame defined by implicit wait(20 seconds) before throwing an "**ElementNotVisibleException**".

**Selenium Web Driver Waits**

1. Implicit Wait
2. Explicit Wait

## Implicit Wait

Selenium Web Driver has borrowed the idea of implicit waits from Watir.

The implicit wait will tell to the web driver to wait for certain amount of time before it throws a "No Such Element Exception". The default setting is 0. Once we set the time, web driver will wait for that time before throwing an exception.

In the below example we have declared an implicit wait with the time frame of 10 seconds. It means that if the element is not located on the web page within that time frame, it will throw an exception.

To declare implicit wait:

**Syntax**:

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

package guru.test99;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.testng.annotations.Test;

public class AppTest {

protected WebDriver driver;

@Test

public void guru99tutorials() throws InterruptedException

{

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

driver = new ChromeDriver();

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

String eTitle = "Demo Guru99 Page";

String aTitle = "" ;

// launch Chrome and redirect it to the Base URL

driver.get("http://demo.guru99.com/test/guru99home/" );

//Maximizes the browser window

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

//get the actual value of the title

aTitle = driver.getTitle();

//compare the actual title with the expected title

if (aTitle.equals(eTitle))

{

System.out.println( "Test Passed") ;

}

else {

System.out.println( "Test Failed" );

}

//close browser

driver.close();

}

}

**Explanation of Code**

In the above example,

**Consider Following Code:**

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

Implicit wait will accept 2 parameters, the first parameter will accept the time as an integer value and the second parameter will accept the time measurement in terms of SECONDS, MINUTES, MILISECOND, MICROSECONDS, NANOSECONDS, DAYS, HOURS, etc.

## Explicit Wait

The explicit wait is used to tell the Web Driver to wait for certain conditions (**Expected Conditions**) or the maximum time exceeded before throwing an "**ElementNotVisibleException**" exception.

The explicit wait is an intelligent kind of wait, but it can be applied only for specified elements. Explicit wait gives better options than that of an implicit wait as it will wait for dynamically loaded Ajax elements.

Once we declare explicit wait we have to use "**ExpectedCondtions**" or we can configure how frequently we want to check the condition using **Fluent Wait**. These days while implementing we are using **Thread.Sleep()**generally it is not recommended to use

In the below example, we are creating reference wait for "**WebDriverWait**" class and instantiating using "**WebDriver**" reference, and we are giving a maximum time frame of 20 seconds.

**Syntax:**

WebDriverWait wait = new WebDriverWait(WebDriverRefrence,TimeOut);

package guru.test99;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.support.ui.ExpectedConditions;

import org.openqa.selenium.support.ui.WebDriverWait;

import org.testng.annotations.Test;

public class AppTest2 {

protected WebDriver driver;

@Test

public void guru99tutorials() throws InterruptedException

{

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

driver = new ChromeDriver();

WebDriverWait wait=new WebDriverWait(driver, 20);

String eTitle = "Demo Guru99 Page";

String aTitle = "" ;

// launch Chrome and redirect it to the Base URL

driver.get("http://demo.guru99.com/test/guru99home/" );

//Maximizes the browser window

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

//get the actual value of the title

aTitle = driver.getTitle();

//compare the actual title with the expected title

if (aTitle.contentEquals(eTitle))

{

System.out.println( "Test Passed") ;

}

else {

System.out.println( "Test Failed" );

}

WebElement guru99seleniumlink;

guru99seleniumlink= wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath( "/html/body/div[1]/section/div[2]/div/div[1]/div/div[1]/div/div/div/div[2]/div[2]/div/div/div/div/div[1]/div/div/a/i")));

guru99seleniumlink.click();

}

}

**Explanation of Code**

**Consider Following Code:**

WebElement guru99seleniumlink;

guru99seleniumlink = wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath("/html/body/div[1]/section/div[2]/div/div[1]/div/div[1]/div/div/div/div[2]/div[2]/div/div/div/div/div[1]/div/div/a/i")));

guru99seleniumlink.click();

In the above example, wait for the amount of time defined in the "**WebDriverWait**" class or the "**ExpectedConditions**" to occur whichever occurs first.

The above[Java](https://www.guru99.com/java-tutorial.html)code states that we are waiting for an element for the time frame of 20 seconds as defined in the "**WebDriverWait**" class on the webpage until the "**ExpectedConditions**" are met and the condition is "**visibilityofElementLocated**".

The following are the Expected Conditions that can be used in Explicit Wait

1. alertIsPresent()
2. elementSelectionStateToBe()
3. elementToBeClickable()
4. elementToBeSelected()
5. frameToBeAvaliableAndSwitchToIt()
6. invisibilityOfTheElementLocated()
7. invisibilityOfElementWithText()
8. presenceOfAllElementsLocatedBy()
9. presenceOfElementLocated()
10. textToBePresentInElement()
11. textToBePresentInElementLocated()
12. textToBePresentInElementValue()
13. titleIs()
14. titleContains()
15. visibilityOf()
16. visibilityOfAllElements()
17. visibilityOfAllElementsLocatedBy()
18. visibilityOfElementLocated()

## Fluent Wait

The fluent wait is used to tell the web driver to wait for a condition, as well as the **frequency** with which we want to check the condition before throwing an "ElementNotVisibleException" exception.

**Frequency:**Setting up a repeat cycle with the time frame to verify/check the condition at the regular interval of time

Let's consider a scenario where an element is loaded at different intervals of time. The element might load within 10 seconds, 20 seconds or even more then that if we declare an explicit wait of 20 seconds. It will wait till the specified time before throwing an exception. In such scenarios, the fluent wait is the ideal wait to use as this will try to find the element at different frequency until it finds it or the final timer runs out.

**Syntax:**

Wait wait = new FluentWait(WebDriver reference)

.withTimeout(timeout, SECONDS)

.pollingEvery(timeout, SECONDS)

.ignoring(Exception.class);

Above code is deprecated in Selenium v3.11 and above. You need to use

Wait wait = new FluentWait(WebDriver reference)

.withTimeout(Duration.ofSeconds(SECONDS))

.pollingEvery(Duration.ofSeconds(SECONDS))

.ignoring(Exception.class);

package guru.test99;

import org.testng.annotations.Test;

import java.util.NoSuchElementException;

import java.util.concurrent.TimeUnit;

import java.util.function.Function;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.support.ui.ExpectedConditions;

import org.openqa.selenium.support.ui.FluentWait;

import org.openqa.selenium.support.ui.Wait;

import org.openqa.selenium.support.ui.WebDriverWait;

import org.testng.annotations.Test;

public class AppTest3 {

protected WebDriver driver;

@Test

public void guru99tutorials() throws InterruptedException

{

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

String eTitle = "Demo Guru99 Page";

String aTitle = "" ;

driver = new ChromeDriver();

// launch Chrome and redirect it to the Base URL

driver.get("http://demo.guru99.com/test/guru99home/" );

//Maximizes the browser window

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

//get the actual value of the title

aTitle = driver.getTitle();

//compare the actual title with the expected title

if (aTitle.contentEquals(eTitle))

{

System.out.println( "Test Passed") ;

}

else {

System.out.println( "Test Failed" );

}

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

.withTimeout(30, TimeUnit.SECONDS)

.pollingEvery(5, TimeUnit.SECONDS)

.ignoring(NoSuchElementException.class);

WebElement clickseleniumlink = wait.until(new Function<WebDriver, WebElement>(){

public WebElement apply(WebDriver driver ) {

return driver.findElement(By.xpath("/html/body/div[1]/section/div[2]/div/div[1]/div/div[1]/div/div/div/div[2]/div[2]/div/div/div/div/div[1]/div/div/a/i"));

}

});

//click on the selenium link

clickseleniumlink.click();

//close~ browser

driver.close() ;

}

}

**Explanation of Code**

**Consider Following Code:**

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

.withTimeout(30, TimeUnit.SECONDS)

.pollingEvery(5, TimeUnit.SECONDS)

.ignoring(NoSuchElementException.class);

In the above example, we are declaring a fluent wait with the timeout of 30 seconds and the frequency is set to 5 seconds by ignoring "**NoSuchElementException**"

**Consider Following Code:**

public WebElement apply(WebDriver driver) {

return driver.findElement(By.xpath("/html/body/div[1]/section/div[2]/div/div[1]/div/div[1]/div/div/div/div[2]/div[2]/div/div/div/div/div[1]/div/div/a/i"));

We have created a new function to identify the Web Element on the page. (Ex: Here Web Element is nothing but the selenium link on the webpage).

Frequency is set to 5 seconds and the maximum time is set to 30 seconds. Thus this means that it will check for the element on the web page at every 5 seconds for the maximum time of 30 seconds. If the element is located within this time frame it will perform the operations else it will throw an"**ElementNotVisibleException**"

## Difference between Implicit Wait Vs Explicit Wait

|  |  |
| --- | --- |
| **Implicit Wait** | **Explicit Wait** |
| * Implicit Wait time is applied to all the elements in the script | * Explicit Wait time is applied only to those elements which are intended by us |
| * In Implicit Wait, we need **not** specify "ExpectedConditions" on the element to be located | * In Explicit Wait, we need to specify "ExpectedConditions" on the element to be located |
| * It is recommended to use when the elements are located with the time frame specified in implicit wait | * It is recommended to use when the elements are taking long time to load and also for verifying the property of the element like(visibilityOfElementLocated, elementToBeClickable,elementToBeSelected) |

**Conclusion:**

**Implicit, Explicit** and **Fluent Wait** are the different waits used in selenium. Usage of these waits are totally based on the elements which are loaded at different intervals of time. It is always **not recommended** to use **Thread.Sleep()**while[Testing](https://www.guru99.com/software-testing.html)our application or building our framework.

**References –**

1. Selenium Locators Examples - <http://www.testingjournals.com/understanding-selenium-locators/>
2. Selenium User Interactions - <http://guru99.com/checkbox-and-radio-button-webdriver.html>
3. Selenium User Interactions -<https://www.tutorialspoint.com/selenium/selenium_user_interactions.htm>
4. Advanced User Interactions -

<https://github.com/SeleniumHQ/selenium/wiki/Advanced-User-Interactions>

1. [Selenium Advanced User Interactions](http://www.theautomatedtester.co.uk/blog/2011/selenium-advanced-user-interactions.html) -http://www.theautomatedtester.co.uk/blog/2011/selenium-advanced-user-interactions.html
2. Know More About Selenium User Interactions -

<https://itelearn.com/blog/know-more-about-selenium-user-interactions/>

1. Actions Class in Selenium - <https://www.toolsqa.com/selenium-webdriver/actions-class-in-selenium/>

# How to handle Actions class in Selenium WebDriver?

<https://www.edureka.co/blog/keyboard-mouse-events-actions-class>

# Java-Selenium | Advanced User Interactions | Performing drag and drop operations

<https://cosmocode.io/java-selenium-advanced-user-interactions-performing-drag-and-drop-operations/>

1. Best selenium user Interactions of ‘2019’ **-** <https://onlineitguru.com/blog/best-selenium-user-interactions-of-2019>

# Java Code Examples for org.openqa.selenium.interactions.Actions -

<https://www.programcreek.com/java-api-examples/?api=org.openqa.selenium.interactions.Actions>