<https://www.guru99.com/>

<https://www.seleniumhq.org/>

ngWebDriver

<https://github.com/paul-hammant/ngWebDriver/blob/master/src/test/java/com/paulhammant/ngwebdriver/AngularAndWebDriverTest.java>

<https://www.guru99.com/first-webdriver-script.html>

Using the[Java](https://www.guru99.com/java-tutorial.html)class "myclass"  that we created in the previous tutorial, let us try to create a WebDriver script that would:

1. fetch Mercury Tours' homepage
2. verify its title
3. print out the result of the comparison
4. close it before ending the entire program.

## WebDriver Code

Below is the actual WebDriver code for the logic presented by the scenario above

Class PG1

**Explaining the code**

**Importing Packages**

To get started, you need to import following two packages:

1. **org.openqa.selenium.\***- contains the WebDriver class needed to instantiate a new browser loaded with a specific driver
2. **org.openqa.selenium.firefox.FirefoxDriver**- contains the FirefoxDriver class needed to instantiate a Firefox-specific driver onto the browser instantiated by the WebDriver class

If your test needs more complicated actions such as accessing another class, taking browser screenshots, or manipulating external files, definitely you will need to import more packages.

**Instantiating objects and variables**

Normally, this is how a driver object is instantiated.

[First Selenium Webdriver Script: JAVA Code Example](https://cdn.guru99.com/images/image004(2).png)

A FirefoxDriver class with no parameters means that the default Firefox profile will be launched by our Java program. The default Firefox profile is similar to launching Firefox in safe mode (no extensions are loaded).

For convenience, we saved the Base URL and the expected title as variables.

**Launching a Browser Session**

WebDriver's **get()** method is used to launch a new browser session and directs it to the URL that you specify as its parameter.

[First Selenium Webdriver Script: JAVA Code Example](https://cdn.guru99.com/images/image005(2).png)

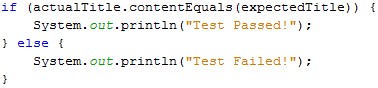
**Get the Actual Page Title**

The WebDriver class has the **getTitle()** method that is always used to obtain the page title of the currently loaded page.

[First Selenium Webdriver Script: JAVA Code Example](https://cdn.guru99.com/images/image006(2).png)

**Compare the Expected and Actual Values**

This portion of the code simply uses a basic Java if-else structure to compare the actual title with the expected one.

[](https://cdn.guru99.com/images/image007(2).png)

**Terminating a Browser Session**

The "**close()**" method is used to close the browser window.

[First Selenium Webdriver Script: JAVA Code Example](https://cdn.guru99.com/images/image008(3).png)

**Terminating the Entire Program**

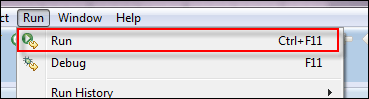
If you use this command without closing all browser windows first, your whole Java program will end while leaving the browser window open.

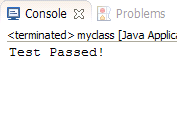
[First Selenium Webdriver Script: JAVA Code Example](https://cdn.guru99.com/images/image009(3).png)

**Running the Test**

There are two ways to execute code in Eclipse IDE.

1. On Eclipse's menu bar, click **Run > Run.**
2. Press **Ctrl+F11** to run the entire code.

[](https://cdn.guru99.com/images/image010(2).png)

 If you did everything correctly, Eclipse would output "Test Passed](https://cdn.guru99.com/images/image011(2).png)

**Locating GUI Elements**

Locating elements in WebDriver is done by using the "**findElement(By.*locator*())**" method. The "locator" part of the code is same as any of the locators previously discussed in the Selenium IDE chapters of these tutorials. Infact, it is recommended that you locate GUI elements using IDE and once successfully identified export the code to webdriver.

Here is a sample code that locates an element by its id. Facebook is used as the Base URL.

package newproject;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

public class PG2 {

public static void main(String[] args) {

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

WebDriver driver = new FirefoxDriver();

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

String tagName = "";

driver.get(baseUrl);

tagName = driver.findElement(By.id("email")).getTagName();

System.out.println(tagName);

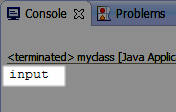
driver.close();

System.exit(0);

}

}

We used the **getTagName()** method to extract the tag name of that particular element whose id is "email". When run, this code should be able to correctly identify the tag name "input" and will print it out on Eclipse's Console window.

[](https://cdn.guru99.com/images/image013(2).png)

Summary for locating elements

|  |  |  |
| --- | --- | --- |
| **Variation** | **Description** | **Sample** |
| By.**className** | finds elements based on the value of the "class" attribute | findElement(By.className("someClassName")) |
| By.**cssSelector** | finds elements based on the driver's underlying CSS Selector engine | findElement(By.cssSelector("input#email")) |
| By.**id** | locates elements by the value of their "id" attribute | findElement(By.id("someId")) |
| By.**linkText** | finds a link element by the exact text it displays | findElement(By.linkText("REGISTRATION")) |
| By.**name** | locates elements by the value of the "name" attribute | findElement(By.name("someName")) |
| By.**partialLinkText** | locates elements that contain the given link text | findElement(By.partialLinkText("REG")) |
| By.**tagName** | locates elements by their tag name | findElement(By.tagName("div")) |
| By.**xpath** | locates elements via XPath | findElement(By.xpath("//html/body/div/table/tbody/tr/td[2]/table/ tbody/tr[4]/td/table/tbody/tr/td[2]/table/tbody/tr[2]/td[3]/ form/table/tbody/tr[5]")) |

**Common Commands**

**Instantiating Web Elements**

Instead of using the long "driver.findElement(By.locator())" syntax every time you will access a particular element, we can instantiate a WebElement object for it. The WebElement class is contained in the "org.openqa.selenium.\*" package.

[First Selenium Webdriver Script: JAVA Code Example](https://cdn.guru99.com/images/image016(2).png)

**Clicking on an Element**

Clicking is perhaps the most common way of interacting with web elements**. The click() method is used to simulate the clicking of any element.** The following example shows how click() was used to click on Mercury Tours'  "Sign-In" button.

[First Selenium Webdriver Script: JAVA Code Example](https://cdn.guru99.com/images/image017(2).png)

Following things must be noted when using the click() method.

* **It does not take any parameter/argument.**
* The method **automatically waits for a new page to load** if applicable.
* The element to be clicked-on, **must be visible** (height and width must not be equal to zero).

**Get Commands**

Get commands fetch various important information about the page/element. Here are some important "get" commands you must be familiar with.

|  |  |
| --- | --- |
| **get()** *Sample usage:* | * It automatically opens a new browser window and fetches the page that you specify inside its parentheses. * It is the counterpart of Selenium IDE's "open" command. * The parameter must be a **String** object. |
| **getTitle()** *Sample usage:* | * Needs no parameters * Fetches the title of the current page * Leading and trailing white spaces are trimmed * Returns a null string if the page has no title |
| **getPageSource()** *Sample usage:* | * Needs no parameters * Returns the **source code of the page** as a String value |
| **getCurrentUrl()** *Sample usage:* | * Needs no parameters * Fetches the string representing the **current URL** that the browser is looking at |
| **getText()** *Sample usage:* | * Fetches the **inner text** of the element that you specify |

**Navigate commands**

These commands allow you to  refresh,go-into and switch back and forth between different web pages.

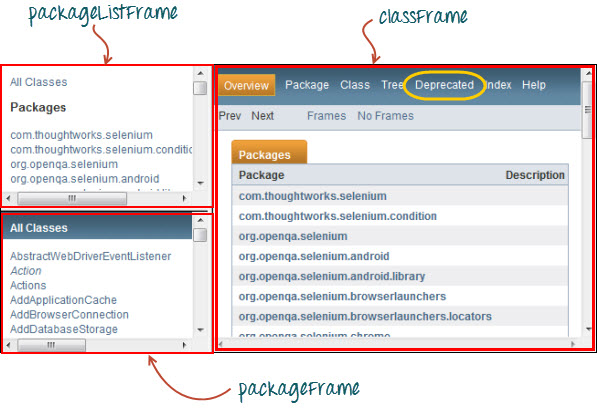
|  |  |
| --- | --- |
| **navigate().to()** *Sample usage:* | * It automatically **opens a new browser window and fetches the page** that you specify inside its parentheses. * **It does exactly the same thing as the get() method.** |
| **navigate().refresh()***Sample usage:* | * Needs no parameters. * It **refreshes** the current page. |
| **navigate().back()***Sample usage:* | * Needs no parameters * Takes you **back by one page** on the browser's history. |
| **navigate().forward()***Sample usage:* | * Needs no parameters * Takes you **forward by one page** on the browser's history. |

**Closing and Quitting Browser Windows**

|  |  |
| --- | --- |
| **close()** *Sample usage:* | * Needs no parameters * **It closes only the browser window that WebDriver is currently controlling**. |
| **quit()** *Sample usage:* | * Needs no parameters * **It closes all windows that WebDriver has opened.** |

## Switching Between Frames

To access GUI elements in a Frame, we should first direct WebDriver to focus on the frame or pop-up window first before we can access elements within them. Let us take, for example, the web page <http://demo.guru99.com/selenium/deprecated.html>

[](https://cdn.guru99.com/images/image033.jpg)

This page has 3 frames whose "name" attributes are indicated above. We wish to access the "Deprecated" link encircled above in yellow. In order to do that, we must first instruct WebDriver to switch to the "classFrame" frame using the **"switchTo().frame()"** method. We will use the name attribute of the frame as the parameter for the "frame()" part.

package newproject;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

public class PG4 {

public static void main(String[] args) {

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

WebDriver driver = new FirefoxDriver();

driver.get("http://demo.guru99.com/selenium/deprecated.html");

driver.switchTo().frame("classFrame");

driver.findElement(By.linkText("Deprecated")).click();

driver.close();

}

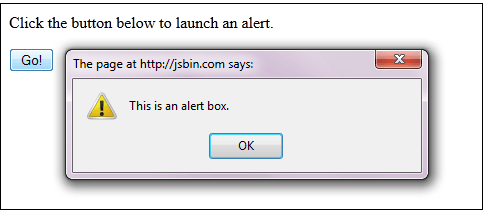
}

After executing this code, you will see that the "classFrame" frame is taken to the "Deprecated API" page, meaning that our code was successfully able to access the "Deprecated" link.

**Switching Between Pop-up Windows**

WebDriver allows pop-up windows like alerts to be displayed, unlike in Selenium IDE. To access the elements within the alert (such as the message it contains), we must use the **"switchTo().alert()"**method. In the code below, we will use this method to access the alert box and then retrieve its message using the **"getText()"** method, and then automatically close the alert box using the **"switchTo().alert().accept()"** method.

First,  head over to <http://jsbin.com/usidix/1> and manually click the "Go!" button there and see for yourself the message text.

[](https://cdn.guru99.com/images/image037(1).png)

Lets see the WebDriver code to do this-

package mypackage;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

public class myclass {

public static void main(String[] args) {

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

WebDriver driver = new FirefoxDriver();

String alertMessage = "";

driver.get("http://jsbin.com/usidix/1");

driver.findElement(By.cssSelector("input[value=\"Go!\"]")).click();

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

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

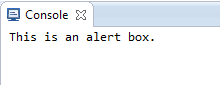
System.out.println(alertMessage);

driver.quit();

}

}

On the Eclipse console, notice that the printed alert message is:

[](https://cdn.guru99.com/images/image036(1).png)

**Waits**

There are two kinds of waits.

1. Implicit wait - used to set the default waiting time throughout the program
2. Explicit wait - used to set the waiting time for a particular instance only

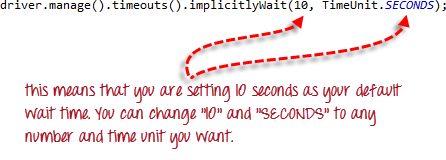
**Implicit Wait**

* It is simpler to code than Explicit Waits.
* It is usually declared in the instantiation part of the code.
* You will only need one additional package to import.

To start using an implicit wait, you would have to import this package into your code.

[First Selenium Webdriver Script: JAVA Code Example](https://cdn.guru99.com/images/image038(1).png)

Then on the instantiation part of your code, add this.

[](https://cdn.guru99.com/images/image039(1).png)

**Explicit Wait**

**Explicit waits are done using the WebDriverWait and ExpectedCondition classes**. For the following example, we shall wait up to 10 seconds for an element whose id is "username" to become visible before proceeding to the next command. Here are the steps.

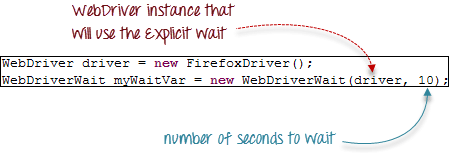
**Step 1**

Import these two packages:

**[First Selenium Webdriver Script: JAVA Code Example](https://cdn.guru99.com/images/image040(1).png)**

**Step 2**

Declare a WebDriverWait variable. In this example, we will use "myWaitVar" as the name of the variable.

[](https://cdn.guru99.com/images/image041(1).png)

**Step 3**

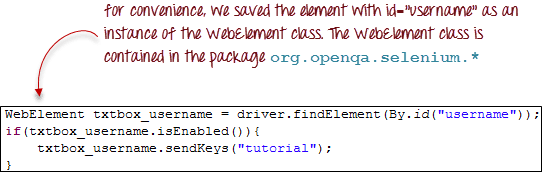
Use myWaitVar with ExpectedConditions on portions where you need the explicit wait to occur. In this case, we will use explicit wait on the "username" (Mercury Tours HomePage) input before we type the text "tutorial" onto it.

[First Selenium Webdriver Script: JAVA Code Example](https://cdn.guru99.com/images/image042(1).png)

**Conditions**

Following  methods are used  in conditional and looping operations --

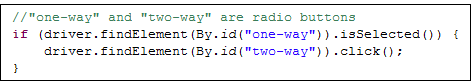
* **isEnabled()** is used when you want to verify whether a certain element is enabled or not before executing a command.

[](https://cdn.guru99.com/images/image043(1).png)

* **isDisplayed()** is used when you want to verify whether a certain element is displayed or not before executing a command.

[First Selenium Webdriver Script: JAVA Code Example](https://cdn.guru99.com/images/image044(1).png)

* **isSelected()** is used when you want to verify whether a certain **check box, radio button, or option in a drop-down box** is selected. It does not work on other elements.

[](https://cdn.guru99.com/images/image045.png)

**Using ExpectedConditions**

The ExpectedConditions class offers a wider set of conditions that you can use in conjunction with WebDriverWait's until() method.

Below are some of the most common ExpectedConditions methods.

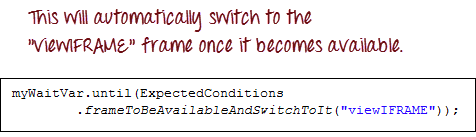
* **alertIsPresent()**- waits until an alert box is displayed.

[First Selenium Webdriver Script: JAVA Code Example](https://cdn.guru99.com/images/image046(1).png)

* **elementToBeClickable()** - Waits until an element is visible and, at the same time, enabled. The sample code below will wait until the element with id="username" to become visible and enabled first before assigning that element as a WebElement variable named "txtUserName".

[First Selenium Webdriver Script: JAVA Code Example](https://cdn.guru99.com/images/image047(1).png)

* **frameToBeAvailableAndSwitchToIt()**- Waits until the given frame is already available, and then automatically switches to it.

[](https://cdn.guru99.com/images/image048(1).png)

**Catching Exceptions**

When using isEnabled(), isDisplayed(), and isSelected(), WebDriver assumes that the element already exists on the page. Otherwise, it will throw a **NoSuchElementException**. To avoid this, we should use a try-catch block so that the program will not be interrupted.

WebElement txtbox\_username = driver.findElement(By.id("username"));

try{

if(txtbox\_username.isEnabled()){

txtbox\_username.sendKeys("tutorial");

}

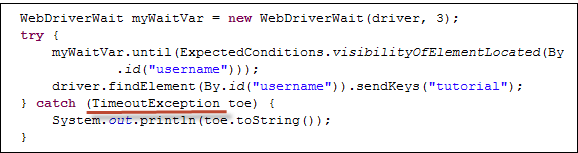
}

catch(NoSuchElementException nsee){

System.out.println(nsee.toString());

}

If you use explicit waits, the type of exception that you should catch is the "TimeoutException".

[](https://cdn.guru99.com/images/image050(2).png)

**Summary**

* To start using the WebDriver API, you must import at least these two packages.
* **org.openqa.selenium.\***
* **org.openqa.selenium.firefox.FirefoxDriver**
* The **get()** method is the equivalent of Selenium IDE's "open" command.
* Locating elements in WebDriver is done by using the **findElement()** method.
* The following are the available options for locating elements in WebDriver:
* By.**className**
* By.**cssSelector**
* By.**id**
* By.**linkText**
* By.**name**
* By.**partialLinkText**
* By.**tagName**
* By.**xpath**
* The By.cssSelector() **does not** support the **"contains"** feature.
* You can instantiate an element using the **WebElement** class.
* Clicking on an element is done by using the **click()** method.
* WebDriver provides these useful **get commands**:
* get()
* getTitle()
* getPageSource()
* getCurrentUrl()
* getText()
* WebDriver provides these useful **navigation commands**
* navigate().forward()
* navigate().back()
* navigate().to()
* navigate().refresh()
* The close() and quit() methods are used to close browser windows. **Close()** is used to close a single window; while **quit()** is used to close all windows associated to the parent window that the WebDriver object was controlling.
* The **switchTo().frame()** and **switchTo().alert()** methods are used to direct WebDriver's focus onto a frame or alert, respectively.
* **Implicit waits** are used to set the waiting time throughout the program, while **explicit waits** are used only on specific portions.
* You can use the **isEnabled(), isDisplayed(),isSelected(),** and a combination of **WebDriverWait**and **ExpectedConditions** methods when verifying the state of an element. However, they do not verify if the element exists.
* When isEnabled(), isDisplayed(),or isSelected() was called while the element was not existing, WebDriver will throw a **NoSuchElementException**.
* When WebDriverWait and ExpectedConditions methods were called while the element was not existing, WebDriver would throw a **TimeoutException**.