## How to handle Selenium Pop-up window using Webdriver

In automation, when we have multiple windows in any web application, the activity may need to switch control among several windows from one to other in order to complete the operation. After completion of the operation, it has to return to the main window i.e. parent window in Selenium. We will see this further in the article with an example.

In Selenium web driver there are methods through which we can handle multiple windows.

**Driver.getWindowHandles();**

To handle all opened windows by web driver, we can use “Driver.getWindowHandles()” and then we can switch window from one window to another in a web application. Its return type is Iterator<String>.

**Driver.getWindowHandle();**

When the site opens, we need to handle the main window by **driver.getWindowHandle()**. This will handle the current window that uniquely identifies it within this driver instance. Its return type is String.

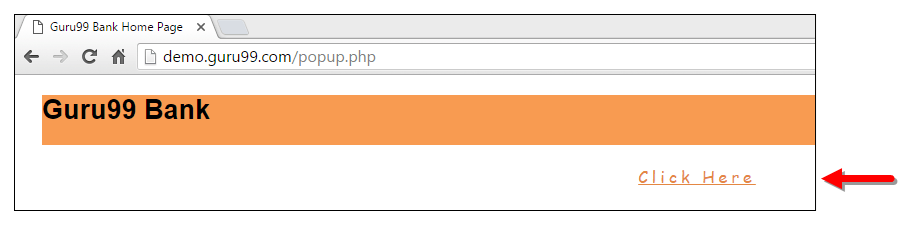
**Step 1)** Launch the site.

Launch the browser and open the site http://demo.guru99.com/popup.php



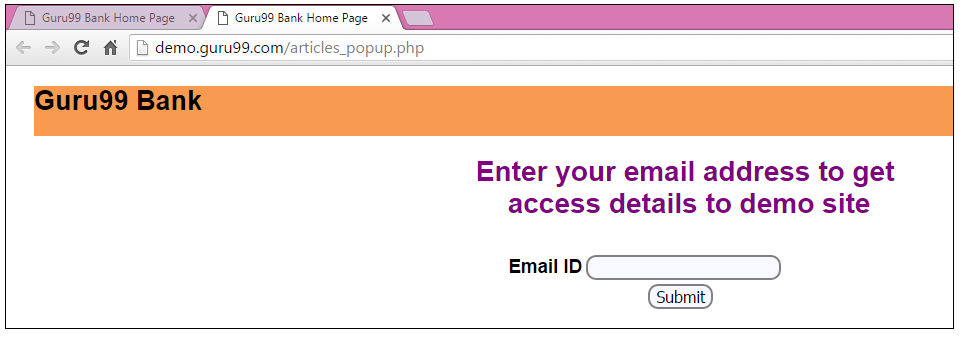
**Step 2)** Click on link “Click Here “.

When the user clicks on the “Click Here ” link, new child window opens.

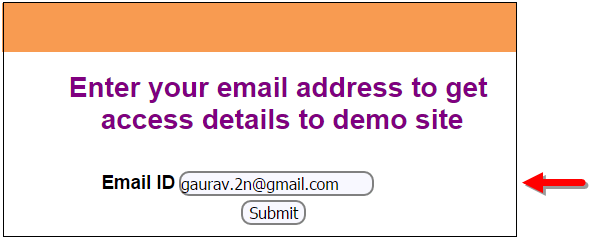


**Step 3)** New Child Window opens.

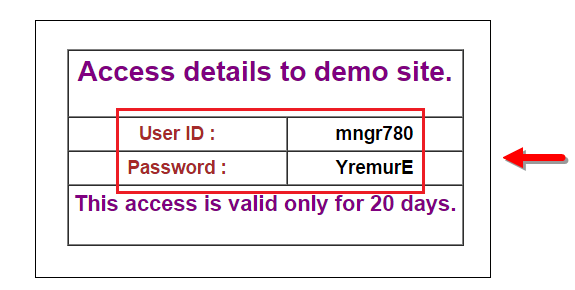
A new window opens, ask the user to enter email id and submit the page.



**Step 4)** Enter your email ID and submit.

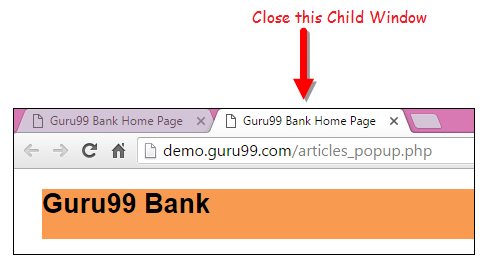


**Step 5)** Display the Access Credentials on submitting the page.

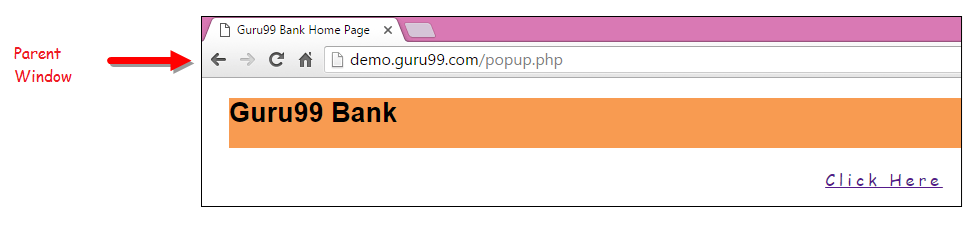


When you execute the code, you will see the child window is open in new tab.

1. Close the Child window on which credentials are displayed.



1. Switch to the parent window.



**Handling multiple windows in Selenium webdriver using above scenario.**

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");

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

// Closing the Child Window.

driver.close();

}

}

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

driver.switchTo().window(MainWindow);

}

}

## Tooltip in Selenium

A **Tooltip in Selenium** is a text that appears when a mouse hovers over an object on a web page. The object can be a link, an image, a button, a text area, etc. The tooltip text often gives more information about the object on which the user hovers over the mouse cursor.

Tooltips were traditionally implemented as a ‘title’ attribute to an element. The value of this attribute was shown as a tooltip on mouse-hover. This is a static text giving information of the element with no styling.

## A Brief of the Advanced User Interactions API:

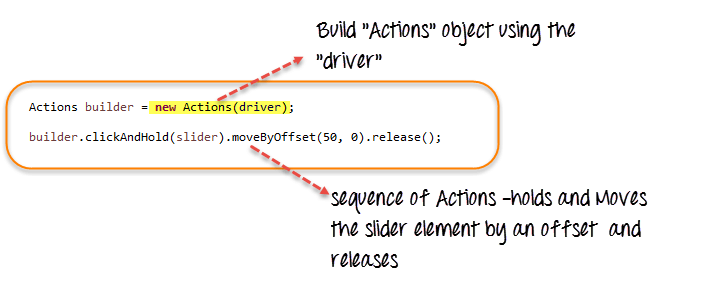
Advanced User Interactions API provides the API for user actions like drag and drop, hovering, multi selecting, key press and release and other actions using keyboard or mouse on a webpage.

**Step 1)**In order to use the API, the following packages/classes needs to be imported:

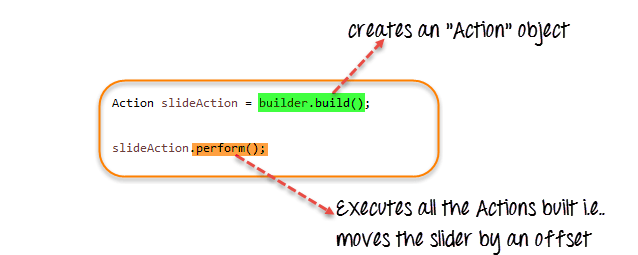
https://www.guru99.com/images/2-2017/072717_0606_VerifyToolt1.png

**Step 2)**Create an object of “Actions” class and build the Sequence of user actions. Actions class is used to build the sequence of user actions like moveToElement(), dragAndDrop() etc. Various methods related to user actions are provided by API.

The driver object is provided as a parameter to its constructor.



**Step 3)** Create an Action Object using the build() method of “Actions” class. Call the perform() method to execute all the actions built by the Actions object(builder here).



We have seen how to use some of the user Actions methods provided by the API – clickAndHold(element), moveByOffset(10,0), release(). The API provides many such methods.

Refer to the [link](https://seleniumhq.github.io/selenium/docs/api/java/index.html?org/openqa/selenium/interactions/Actions.html) for more details.

## How to get Tooltip Text in Selenium Webdriver

Let’s see the demonstration of accessing and verifying the tool tips in the simple scenario

* Scenario 1: Tooltip is implemented using the “title” attribute

**Scenario 1: HTML ‘title’ Attribute**

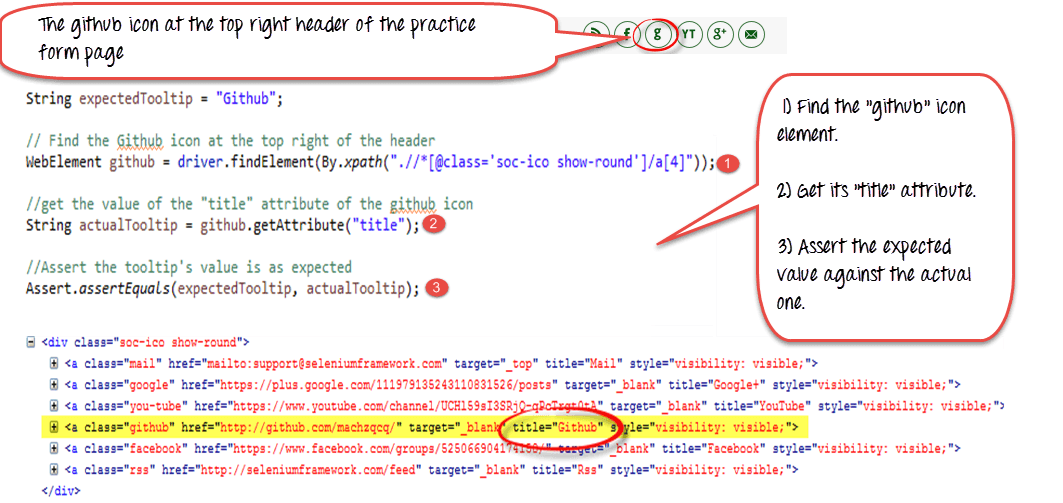
For this case, let’s take the example site – http://demo.guru99.com/test/social-icon.html.

We will try to verify the tooltip of the “github” icon at the top right of the page.

https://www.guru99.com/images/2-2017/072717_0606_VerifyToolt4.png

In order to do it, we will first find the element and get its ‘title’ attribute and verify with the expected tool tip text.

Since, we are assuming the tool tip is in the “title” attribute, we are not even automating the mouse hover effect but simply retrieving the attribute’s value using the “getAttribute()” method.



**Here is the code**

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.\*;

public class ToolTip {

public static void main(String[] args) {

String baseUrl = "http://demo.guru99.com/test/social-icon.html";

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

WebDriver driver = new ChromeDriver();

driver.get(baseUrl);

String expectedTooltip = "Github";

// Find the Github icon at the top right of the header

WebElement github = driver.findElement(By.xpath(".//\*[@class='soc-ico show-round']/a[4]"));

//get the value of the "title" attribute of the github icon

String actualTooltip = github.getAttribute("title");

//Assert the tooltip's value is as expected

System.out.println("Actual Title of Tool Tip"+actualTooltip);

if(actualTooltip.equals(expectedTooltip)) {

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

}

driver.close();

}

}

**Explanation of code**

1. Find the WebElement representing the “github” icon.
2. Get its “title” attribute using the getAttribute() method.
3. Assert the value against the expected tooltip value.

Selenium - Exception Handling

When we are developing tests, we should ensure that the scripts can continue their execution even if the test fails. An unexpected exception would be thrown if the worst case scenarios are not handled properly.

If an exception occurs due to an element not found or if the expected result doesn't match with actuals, we should catch that exception and end the test in a logical way rather than terminating the script abruptly.

Syntax

The actual code should be placed in the try block and the action after exception should be placed in the catch block. Note that the 'finally' block executes regardless of whether the script had thrown an exception or NOT.

try {

//Perform Action

} catch(ExceptionType1 exp1) {

//Catch block 1

} catch(ExceptionType2 exp2) {

//Catch block 2

} catch(ExceptionType3 exp3) {

//Catch block 3

} finally {

//The finally block always executes.

}

Example

If an element is not found (due to some reason), we should step out of the function smoothly. So we always need to have a try-catch block if we want to exit smoothly from a function.

public static WebElement lnk\_percent\_calc(WebDriver driver)throws Exception {

try {

element = driver.findElement(By.xpath(".//\*[@id='menu']/div[4]/div[3]/a"));

return element;

} catch (Exception e1) {

// Add a message to your Log File to capture the error

Logger.error("Link is not found.");

// Take a screenshot which will be helpful for analysis.

File screenshot = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);

FileUtils.copyFile(screenshot, new File("D:\\framework\\screenshots.jpg"));

throw(e1);

}

}

# Selenium - Log4j Logging

Log4j is an audit logging framework that gives information about what has happened during execution. It offers the following advantages −

* Enables us to understand the application run.
* Log output can be saved that can be analyzed later.
* Helps in debugging, in case of test automation failures.
* Can also be used for auditing purposes to look at the application's health.

## Components

1. Instance of Logger class.

2. Log level methods used for logging the messages as one of the following −

* error
* warn
* info
* debug
* log

**Import Required:**

**import** org.apache.log4j.Logger;

**import** org.apache.log4j.PropertyConfigurator;

**Dependency:**

<dependencies>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-api</artifactId>

<version>2.13.3</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-api</artifactId>

<version>2.13.3</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-core</artifactId>

<version>2.13.3</version>

</dependency>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

</dependencies>

**Initialize:**

**static** Logger *logger* = Logger.*getLogger*("test1");

Where “test1” is the class name.

**Add this in main methord:**

PropertyConfigurator.*configure*("Log4j.properties");

**Place Log4j.properties file inside project from shared path.**

# Selenium - Capture Screenshots

This functionality helps to grab screenshots at run time when required, in particularly when a failure happens. With the help of screenshots and log messages, we will be able to analyze the results better.

Screenshots are configured differently for local executions and Selenium Grid(remote) executions. Let us take a look at each one them with an example.

## Localhost Execution

In the following example, we will take a screenshot after calculating the percentage. Ensure that you give a valid path to save the screenshot.

import java.io.File;

import java.io.IOException;

import java.util.concurrent.TimeUnit;

import org.apache.commons.io.FileUtils;

import org.openqa.selenium.\*;

import org.openqa.selenium.firefox.FirefoxDriver;

public class WebdriverDemo {

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

WebDriver driver = new FirefoxDriver();

// Puts an Implicit wait, Will wait for 10 seconds before throwing exception

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

// Launch website

driver.navigate().to("http://www.calculator.net/");

// Maximize the browser

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

// Click on Math Calculators

driver.findElement(By.xpath(".//\*[@id = 'menu']/div[3]/a")).click();

// Click on Percent Calculators

driver.findElement(By.xpath(".//\*[@id = 'menu']/div[4]/div[3]/a")).click();

// Enter value 10 in the first number of the percent Calculator

driver.findElement(By.id("cpar1")).sendKeys("10");

// Enter value 50 in the second number of the percent Calculator

driver.findElement(By.id("cpar2")).sendKeys("50");

// Click Calculate Button

driver.findElement(By.xpath(".//\*[@id = 'content']/table/tbody/tr/td[2]/input")).click();

// Get the Result Text based on its xpath

String result =

driver.findElement(By.xpath(".//\*[@id = 'content']/p[2]/span/font/b")).getText();

File screenshot = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);

FileUtils.copyFile(screenshot, new File("D:\\screenshots\\screenshots1.jpg"));

// Print a Log In message to the screen

System.out.println(" The Result is " + result);

//Close the Browser.

driver.close();

}

}

### **Output**

Upon executing the script, the screenshot is saved in the 'D:\screenshots' folder with the name 'screenshots1.jpg' as shown below.

