**Working with Alerts –**

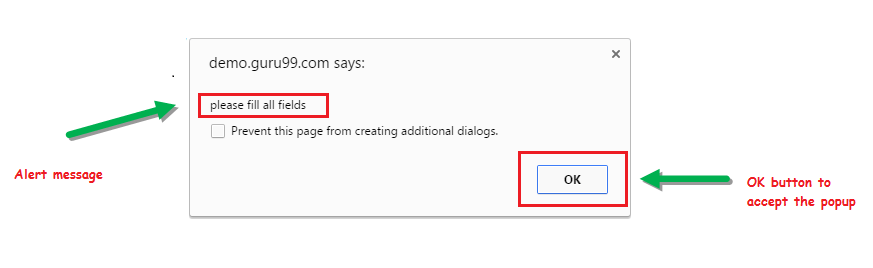
What is Alert in Selenium?

An Alert in Selenium is a small message box which appears on screen to give the user some information or notification. It notifies the user with some specific information or error, asks for permission to perform certain tasks and it also provides warning messages as well.

**Types of Alerts in Selenium**

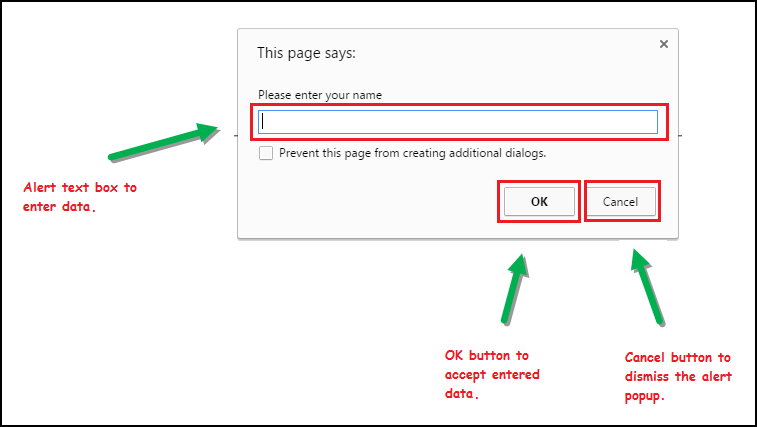
**1) Simple Alert**

The simple alert class in Selenium displays some information or warning on the screen.



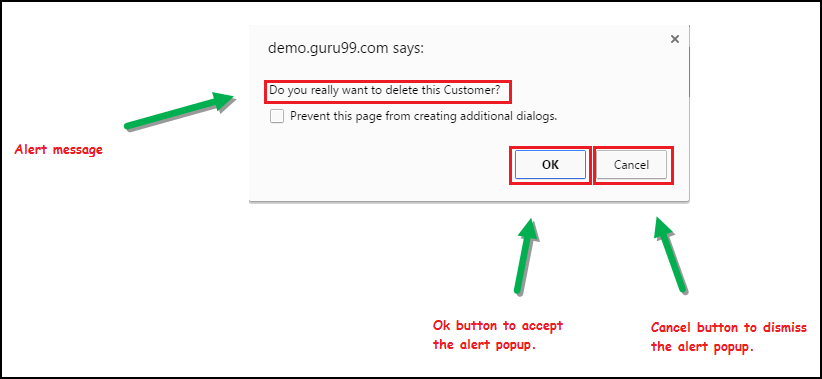
**2) Prompt Alert**

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



**3) Confirmation Alert**

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



**How to handle Alert in Selenium WebDriver**

Alert interface provides the below few methods which are widely used in [Selenium Webdriver](https://www.guru99.com/introduction-webdriver-comparison-selenium-rc.html).

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();

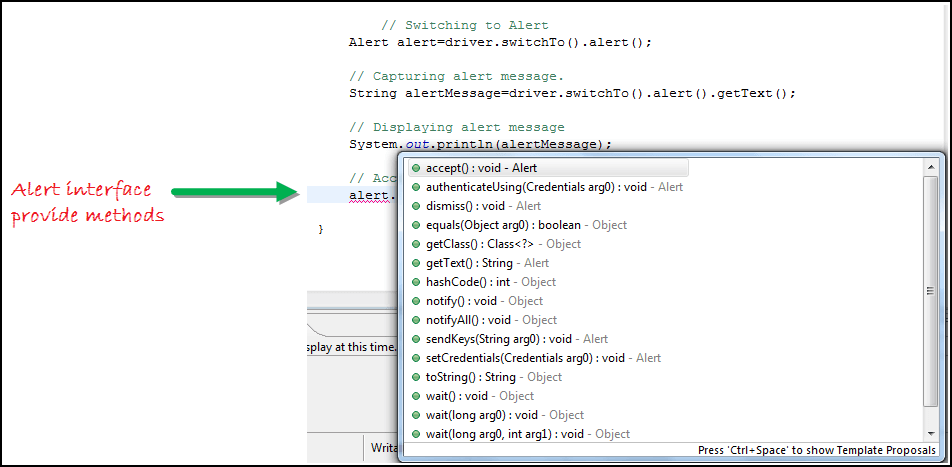
3) **String getText()** // To capture the alert message.

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

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

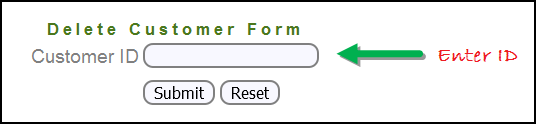
driver.switchTo().alert().sendKeys("Text");

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

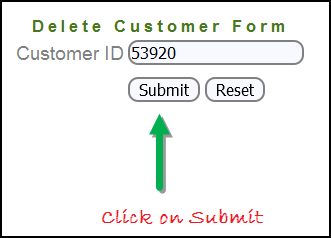


**Step 1)** Launch the web browser and open the site <https://demo.guru99.com/test/delete_customer.php>

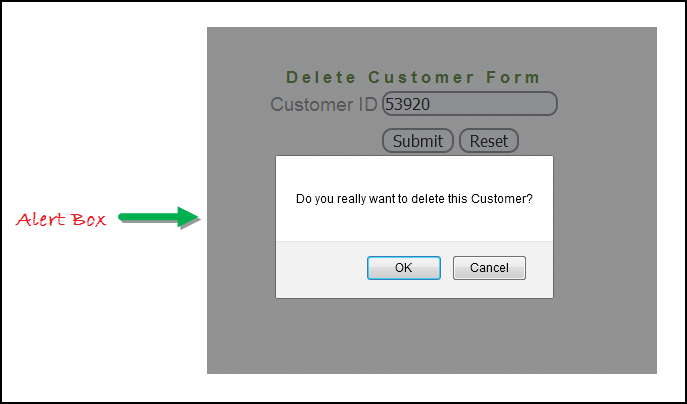
**Step 2)** Enter Any Customer id.



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



**Step 4)** Reject/accept the alert.



#### Handling Alert in Selenium Webdriver

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("https://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.

Execute and run

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

For Window handling in Selenium, we will follow the below steps:

Now, we will automate the given below scenario to see how to handle multiple windows using Selenium Webdriver.

In this scenario, we will use “Guru99” demo site to illustrate window handling.

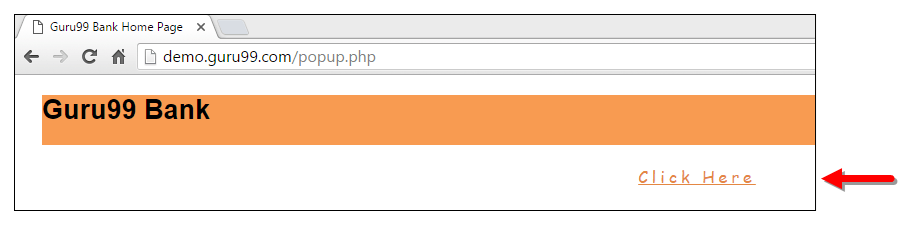
**Step 1)** Launch the site.

Launch the browser and open the site <https://demo.guru99.com/popup.php>



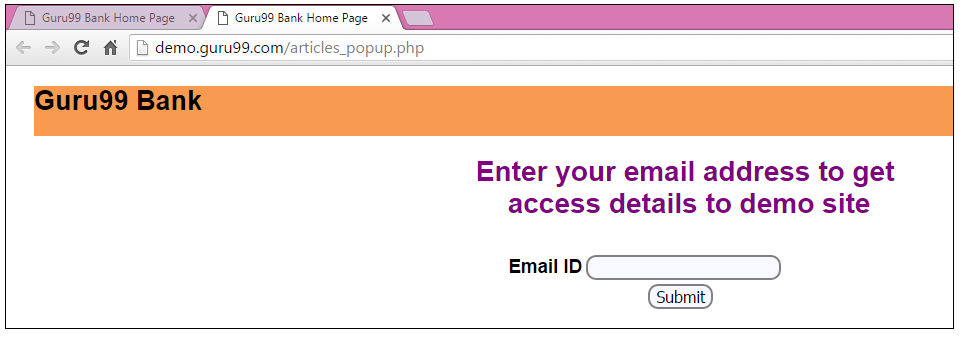
**Step 2)** Click on link “Click Here”.

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

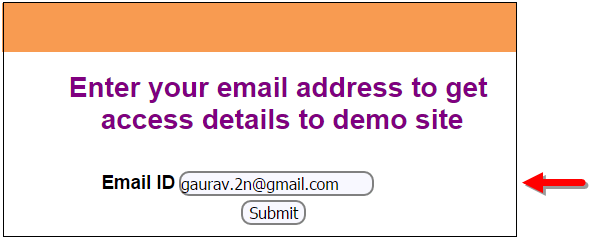


**Step 3)** New Child Window opens.

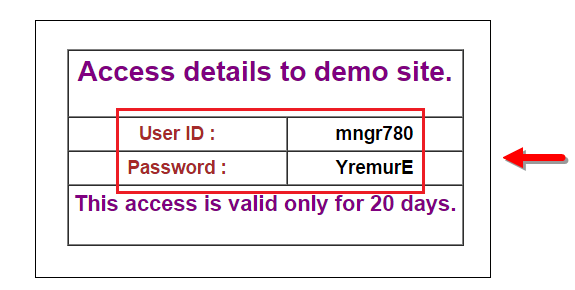
A new window opens, ask the user to enter email id and submit the page.



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

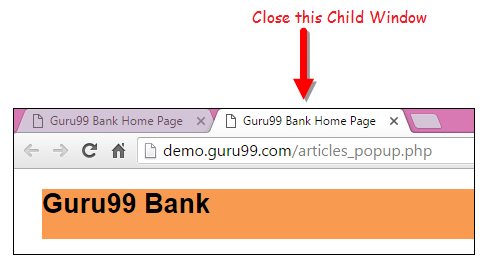


**Step 5)** Display the Access Credentials on submitting the page.

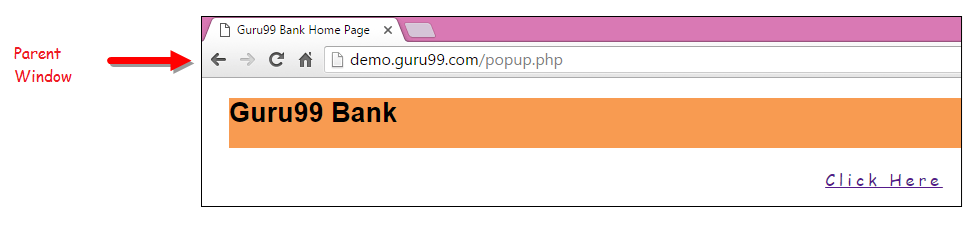


When you execute the code, you will see the child window is open in new tab.

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



1. Switch to the parent window.



## How to Handle Multiple Windows in Selenium

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("https://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);

}

}

**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.

## What is JavaScriptExecutor?

JavaScriptExecutor is an Interface that helps to execute JavaScript through Selenium Webdriver. JavaScriptExecutor provides two methods “executescript” & “executeAsyncScript” to run javascript on the selected window or current page.

**Why do we need JavaScriptExecutor?**

In Selenium Webdriver, locators like XPath, CSS, etc. are used to identify and perform operations on a web page.

In case, these locators do not work you can use JavaScriptExecutor. You can use JavaScriptExecutor to perform an desired operation on a web element.

Selenium supports javaScriptExecutor. There is no need for an extra plugin or add-on. You just need to import (org.openqa.selenium.JavascriptExecutor) in the script as to use JavaScriptExecutor.

**JavaScriptExecutor Methods in Selenium**

**executeScript -**

This method executes JavaScript in the context of the currently selected frame or window in Selenium. The script used in this method runs in the body of an anonymous function (a function without a name). We can also pass complicated arguments to it.

The script can return values. Data types returned are

Boolean

Long

String

List

WebElement.

**JavascriptExecutor syntax:**

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript(Script,Arguments);

**Script –** This is the JavaScript that needs to execute.

**Arguments** – It is the arguments to the script. It’s optional.

**executeAsyncScript-**

With Asynchronous script, your page renders more quickly. Instead of forcing users to wait for a script to download before the page renders. This function will execute an asynchronous piece of JavaScript in the context of the currently selected frame or window in Selenium. The JS so executed is single-threaded with a various callback function which runs synchronously.

**How to use JavaScriptExecutor in Selenium**

Here is a step-by-step process on how to use JavaScriptExecutor in Selenium:

**Step 1)** Import the package.

import org.openqa.selenium.JavascriptExecutor;

**Step 2)** Create a Reference.

JavascriptExecutor js = (JavascriptExecutor) driver;

**Step 3)** Call the JavascriptExecutor method.

js.executeScript(script, args);

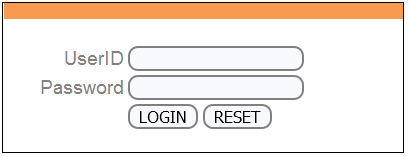
**Example of Click an Element using JavaScripExecutor in Selenium**

For executeScript, we will see three different example one by one.

**1) Example: Click a button to login and generate Alert window using JavaScriptExecutor.**

In this scenario, we will use “Guru99” demo site to illustrate JavaScriptExecutor. In this example,

* Launch the web browser
* open the site <https://demo.guru99.com/V4/> and
* login with credentials



* Display alert window on successful login.

import org.openqa.selenium.By;

import org.openqa.selenium.JavascriptExecutor;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.annotations.Test;

public class JavaSE\_Test {

public void Login()

{

WebDriver driver= new FirefoxDriver();

//Creating the JavascriptExecutor interface object by Type casting

JavascriptExecutor js = (JavascriptExecutor)driver;

//Launching the Site.

driver.get("https://demo.guru99.com/V4/");

WebElement button =driver.findElement(By.name("btnLogin"));

//Login to Guru99

driver.findElement(By.name("uid")).sendKeys("mngr34926");

driver.findElement(By.name("password")).sendKeys("amUpenu");

//Perform Click on LOGIN button using JavascriptExecutor

js.executeScript("arguments[0].click();", button);

//To generate Alert window using JavascriptExecutor. Display the alert message

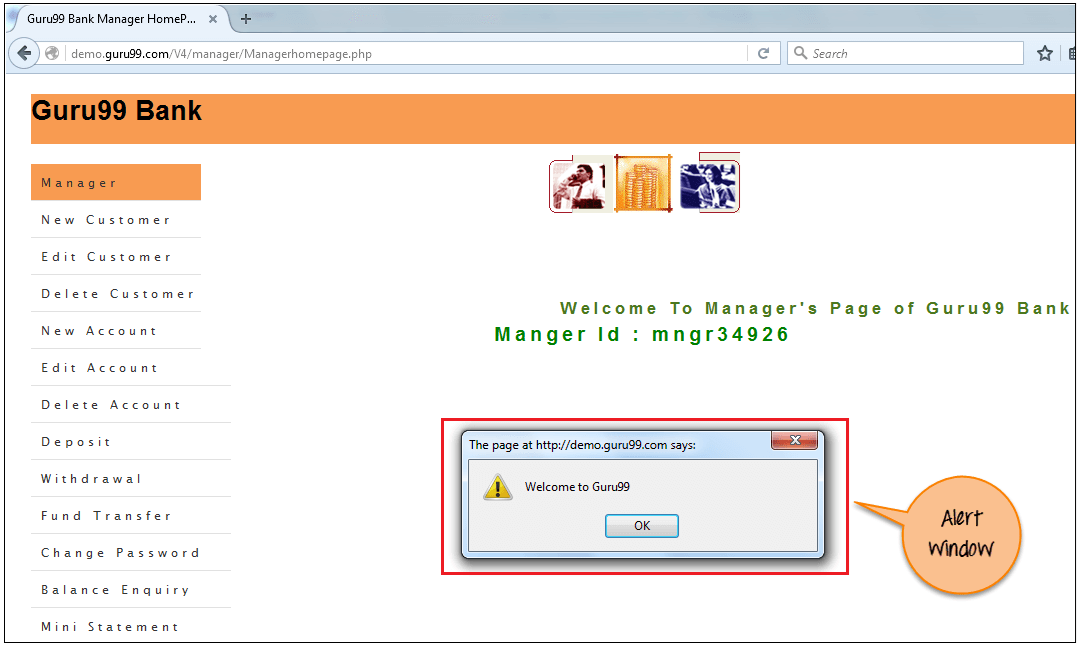
js.executeScript("alert('Welcome to Guru99');");

}

}

**Output:** When the code is executed successfully. You will observe

* Successful click on login button and the
* Alert window will be displayed (see image below).



# **How to Drag and Drop in Selenium?**

**Introduction to Drag and Drop in Selenium WebDriver**

Few web applications have the ability to automate the functionality of drag and drop, i.e. drag web elements from one end and drop them on a defined area or element. In order to automate the drag and drop action of such elements, one can use Selenium Webdriver.

**What is Drag and Drop in Selenium?**

This action is performed using a **mouse** when a user **moves (drags)** a web element from one location and then **places (drops)** it at another point.

This is a common action used in Windows Explorer when moving a file from one folder to another. Here, the user selects a file in the folder, drags it to the desired folder, and drops it.

**Syntax:**

Actions action = new Actions(driver);

action.dragAndDrop(Sourcelocator, Destinationlocator).build().perform();

In the Selenium**dragAndDrop** method, we pass two parameters:

* The first parameter is the *Sourcelocator* element which is being dragged
* The second parameter is the *Destinationlocator* on which the previous element needs to be *droppedMethods* for performing drag and drop on a web element are as follows:
  + **clickAndHold(WebElement element)** – Clicks a web element at the middle (without releasing)
  + **moveToElement(WebElement element)** – Moves the mouse pointer to the middle of the web element without clicking
  + **release(WebElement element)** – Releases the left click (which is in the pressed state)
  + **build()** – Generates a composite action

### ****Code Snippet****

The code snippet below demonstrates the automation of the Drag and Drop action program.

//WebElement on which drag and drop operation needs to be performed

WebElement fromElement = driver.findElement(By Locator of fromElement);

//WebElement to which the above object is dropped

WebElement toElement = driver.findElement(By Locator of toElement);

//Creating object of Actions class to build composite actions

Actions builder = new Actions(driver);

//Building a drag and drop action

Action dragAndDrop = builder.clickAndHold(fromElement).moveToElement(toElement).release(toElement).build();

//Performing the drag and drop action

dragAndDrop.perform();

**Difference between Drag and Drop vs Action Class Build**

|  |  |
| --- | --- |
| **Drag and Drop** | **Action Class Build** |
| Selenium provides action.dragandDrop class to move elements from one end to another | Used to move an element from one end to another by working on the element coordinates |
| The user does not have to use build() and perform() actions here to move the elements which is an added advantage | **build**() **method in Actions class** is used to create the chain of **action** or operations to be performed. **perform**() method **in**[**Actions Class**](https://www.browserstack.com/guide/action-class-in-selenium) is used to **execute** the chain of **action** created using**build() method** |

**Code to Execute-**

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.openqa.selenium.interactions.Actions;

public class DragAndDropExample {

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

//Set system properties for geckodriver System.setProperty("webdriver.gecko.driver", "Path\_of\_the\_driver");

WebDriver driver = new FirefoxDriver();

String URL = "https://the-internet.herokuapp.com/drag\_and\_drop";

driver.get(URL);

// It is always advisable to Maximize the window before performing DragNDrop action driver.manage().window().maximize();

driver.manage().timeouts().implicitlyWait(10000, TimeUnit.MILLISECONDS);

//Actions class method to drag and drop

Actions builder = new Actions(driver);

WebElement from = driver.findElement(By.id("column-a"));

WebElement to = driver.findElement(By.id("column-b"));

//Perform drag and drop

builder.dragAndDrop(from, to).perform();

//verify text changed in to 'Drop here' box

String textTo = to.getText();

if(textTo.equals("Dropped!")) {

System.out.println("PASS: File is dropped to target as expected");

}else {

System.out.println("FAIL: File couldn't be dropped to target as expected");

}

driver.close();

}

}