1. Demonstrate the features of selenium

Features of Selenium

* **Open-Source:**

Selenium is a freeware and a portable tool. It has no upfront direct costs involved. The tool can be freely downloaded and the support for it is freely available, as it is community-based.

* **Supports languages:**

Selenium supports a range of languages, including Java, Perl, Python, C#, Ruby, Groovy, Java Script, etc. It has its own script, but it doesn’t limit it to that language. It can work with various languages and whatever the developers/testers are comfortable with.

* **Supports Operating Systems:**

Selenium operates across and supports multiple Operating Systems, (OS) like Windows, Mac, Linux, UNIX, etc. With Selenium Suite of solutions, a tailored testing suite can be created over any platform and then executed on another one. For instance, you can create test cases using Windows OS and run it with ease on a Linux-based system.

* **Supports multiple browsers:**

Selenium provides support across multiple browsers, namely, Internet Explorer, Chrome, Firefox, Opera, Safari, etc. This becomes highly resourceful while executing tests and testing it across various browsers simultaneously.

The browsers supported by the Selenium packages are:

* Selenium IDE can be used with Firefox as a plug-in.
* Selenium RC and Webdriver supports diverse browsers, such as Internet Explorer.
* Supports programming languages and frameworks

Selenium integrates with programming languages and various frameworks. For instance, it can integrate with ANT or Maven type of framework for source code compilation. Furthermore, it can integrate with the TestNG testing framework for testing applications and reporting purposes. It can integrate with Jenkins or Hudson for Continuous Integration (CI) and can even integrate with other Open-Source tools to support other features.

* **Tests across devices**

Selenium Test Automation can be implemented for mobile web application automation on Android, IPhone, and Blackberry. This can help in generating necessary results and addresses issues on a continuous basis.

* **Constant updates**

Selenium support is community-based and active community support enables constant updates and upgrades. These upgrades are readily available and do not require specific training. This makes Selenium resourceful and cost-effective as well.

* **Loaded Selenium Suites**

Selenium is not just a singular tool or utility, it is a loaded package of various testing tools and so is referred to as a Suite. Each tool is designed to cater to different testing needs and requirements of test environments.

Additionally, Selenium comes with capabilities to support Selenium IDE, Selenium Grid, and Selenium Remote Control (RC).

* **Ease of implementation**

Selenium offers a user-friendly interface that helps create and execute tests easily and effectively. Its open-source features help users to script their own extensions which makes it easy to develop customized actions and even manipulate at an advanced level. Tests run directly across browsers and users can watch while the tests are being executed. Additionally, Selenium’s reporting capabilities are one of the reasons for being chosen, as it allows testers to extract results and take follow-up actions.

* **Reusability and Add-ons**

Selenium Test Automation Framework uses scripts that can be tested directly across multiple browsers. Concurrently, it is possible to execute multiple tests with Selenium, as it covers almost all aspects of functional testing by implementing add-on tools that broaden the scope of testing.

1. Demonstrate how Selenium web driver is installed and integrated in Eclipse.

**Step 1:** Downloading Selenium Standalone Server jar

* Selenium is already installed in your practice lab. (Refer QA to QE: Lab Guide - Phase 1)

**Step 2:** Launching Eclipse and creating a Java project

* Launch the Eclipse and create a Workspace.
* Create Project:

Click on File -> New -> Java Project.

**Step 3:** Configuring WebDriver with Eclipse

* Add selenium standalone server jars.
* Right-click on Project -> select Properties -> Select Java Build Path.
* Navigate to the Libraries tab and click on the Add External Jars button.
* Add selenium standalone server Jar files.
* Click on the Apply and Close button.
* In Eclipse, it looks like the screenshot below:



1. Demonstrate how elements are located using Selenium WebDriver.

**Step 1:** Using ID as a Locator

* Open Eclipse
* Find a web element using Locator **ID**

1. Syntax: id = id of the element
2. Example: driver.findElement(By.id(“Email”));

**Step 2** Using class name as a Locator

* Find a web element using Locator **ClassName**
  1. Syntax: class = Class Name of the element
  2. Example: driver.findElement(By.class(“classname”));

**Step 3** Using Name as a Locator

* Find a web element using Locator **Name**
  1. Syntax: name = Name of the element
  2. Example: driver.findElement(By.name(“name”));

**Step 4** Using LinkText as a Locator

* Find a web element using Locator **Link Text**
  1. Syntax: link = partialLink of the element
  2. Example: driver.findElement(By.partialLinkText(“plink”));

**Step 5** Using Xpath as a Locator

* Find a web element using Locator **Xpath**
* Xpath can be created in two ways
  1. **Relative Xpath**
* Syntax: relativeXpath : //\*[@class=’relativexapath’]
* Example: driver.findElement(By.xpath(“//\*[@class=’relativexapath’]”));
  1. **Absolute Xpath**
* Syntax: absoluteXpath : html/body/div[1]/div[1]/div/h4[1]/b
* Example: driver.findElement(By.xpath(“html/body/div[1]/div[1]/div/h4[1]/b”));

**Step 6** Using Xpath as a **CSS Selector**

* CSS Selector have many formats, namely

1. **Tag and ID**
   * Syntax:”css = tag#id”
   * Example: driver.findElement(By.cssSelector(“input#email”));
2. **Tag and Class**
   * Syntax: “css = tag.class”
   * Example: driver.findElement(By.cssSelector(”input.inputtext”));
3. **Tag and Attribute**
   * Syntax: “css = tag[attribute=value]”
   * Example: driver.findElement(By.cssSelector(“input[name=lastName]”));
4. **Tag, Class, and Attribute**
   * Syntax: “tag.class[attribute=value]”
   * Example: driver.findElement(By.cssSelector(“input.inputtext[tabindex=1]”));
5. **Inner text**
   * Syntax: ”css = tag.contains(“innertext”)”
   * Example: driver.findElement(By.cssSelector(font:contains(“Boston”)));

**Step 7** Using Xpath for handling complex and dynamic elements

* Dynamic Xpath has many formats, namely

1. **Contains();**
   * Syntax: “xpath = //\*[contains(text(),’text’)]
   * Example: driver.findElement(By.xpath(”//\*[contains(text(),’sub’]”));
2. **Using OR & AND**
   * Syntax: xpath=//\*[@type=’submit’ or @name=’btnReset’]
   * Example:

driver.findElement (By.xpath(”=//\*[@type=’submit’ or @name=’btnReset’]”));

1. **Start-with function**
   * Syntax: xpath= //label[starts-with(@id,’message’)]
   * Example:

driver.findElement (By.xpath(”//label[starts-with(@id,’message’)]”));

1. **Text();**
   * Syntax: xpath=//td[text()=’UserID’]
   * Example: : driver.findElement (By.xpath(”=//td[text()=’UserID’]”));
2. **Following**
   * Syntax: xpath=//\*[@type=’text’]//following::input
   * Example: driver.findElement(By.xpath(”=//\*[@type=’text’]//following::input”));
3. **Preceding**
   * Syntax: xpath=//\*[@type=’text’]//preceding::input
   * Example: driver.findElement(By.xpath(”//\*[@type=’text’]//preceding::input”));
4. **Following - sibling**
   * Syntax: xpath=//\*[@type=’submit’]//preceding::input
   * Example:

driver.findElement (By.xpath (”//\*[@type=’text’]//following-sibling::input”));

The code for the above steps is as follows:

**package** com.seleniumtest

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

public class LocatorExample {

public static void main(String[] args) {

// Set the path to the ChromeDriver executable

System.setProperty("webdriver.chrome.driver", " drivers/chromedriver-win64/chromedriver.exe");

// Create a new instance of the ChromeDriver

WebDriver driver = new ChromeDriver();

// Launch the website

driver.get("https://www.seleniumhq.org");

// Step 1: Using ID as a Locator

WebElement downloadTab = driver.findElement(By.id("menu\_download"));

downloadTab.click();

// Step 2: Using class name as a Locator

WebElement versionsTab = driver.findElement(By.className("version"));

versionsTab.click();

// Step 3: Using Name as a Locator

WebElement searchField = driver.findElement(By.name("q"));

searchField.sendKeys("Selenium WebDriver");

// Step 4: Using LinkText as a Locator

WebElement aboutTab = driver.findElement(By.linkText("About"));

aboutTab.click();

// Step 5: Using Xpath as a Locator

// Relative Xpath

WebElement relativeElement = driver.findElement(By.xpath("//a[@class='navbar-brand']"));

relativeElement.click();

// Absolute Xpath

WebElement absoluteElement = driver.findElement(By.xpath("/html/body/div[1]/div[1]/a"));

absoluteElement.click();

// Step 6: Using CSS Selector as a Locator

// Tag and ID

WebElement downloadButton = driver.findElement(By.cssSelector("a#downloadSeleniumBtn"));

downloadButton.click();

// Tag and Class

WebElement projectName = driver.findElement(By.cssSelector("h1.project-name"));

System.out.println("Project Name: " + projectName.getText());

// Step 7: Using Xpath for handling complex and dynamic elements

// Contains()

WebElement projectsLink = driver.findElement(By.xpath("//\*[contains(text(),'Projects')]"));

projectsLink.click();

// Close the browser

driver.quit();

}

}

4. Demonstrate how elements are located through CSS and XPath.

**Step 4.1:** Finding the element present on the page using CSS Selector.

* Using CSS Selectors in Selenium. As we all know, CSS stands for Cascading Style Sheets. By using CSS selectors, we can find or select HTML elements on the basis of their id, class, or other attributes. CSS is faster and simpler than XPath, particularly in case of IE browser where Path works very slowly.
* Open Eclipse
* Using Path as a CSS Selector
* CSS Selector has many formats, namely:

1. **Tag and ID**
   * Syntax: “css = tag#id”
   * Example: driver.findElement(By.cssSelector(“input#email”));
2. **Tag and Class**
   * Syntax: “css = tag.class”
   * Example: driver.findElement(By.cssSelector(”input.inputtext”));
3. **Tag and Attribute**
   * Syntax: “css = tag[attribute=value]”
   * Example: driver.findElement(By.cssSelector(“input[name=lastName]”));
4. **Tag, Class, and Attribute**
   * Syntax: “tag.class[attribute=value]”
   * Example:

driver. findElement(By.cssSelector(“input.inputtext[tabindex=1]”));

1. **Inner text**
   * Syntax: “css = tag.contains(“innertext”)”
   * Example: driver.findElement(By.cssSelector(font:contains(“Boston”)));

**Step 4.2:** Finding the element present on the page using Path.

* In Selenium automation, if the elements are not found by the general locators like id, class, name, etc., then XPath is used to find an element on the web page.
* XPath contains the path of the element situated at the web page. Standard syntax for creating XPath is:

XPath=//tagname[@attribute='value']

* **//:** Select current node.
* **Tagname:**Tagname of the particular node.
* **@:** Select attribute.
* **Attribute:** Attribute name of the node.
* **Value:** Value of the attribute.
* Types of XPath:

There are two types of XPath:

1. **Absolute XPath**

* It is a direct way to find the element, but the disadvantage of the absolute XPath is that if there are any changes made in the path of the element, then that XPath fails.
* The key characteristic of XPath is that it begins with the single forward slash (/), which means you can select the element from the root node.
* Syntax for absolute Path: html/body/div[1]/div[1]/div/h4[1]/b
* Example: driver.findElement(By.xpath(“html/body/div[1]/div[1]/div/h4[1]/b”));
* Writing absolute XPath on the elements which are present in the web page will be very lengthy. To reduce the length, we use relative XPath.

1. **Relative XPath**

* For relative XPath, the path starts from the middle of the HTML DOM structure. It starts with the double forward-slash (//), which means it can search the element anywhere on the web page.
* You can start from the middle of the HTML DOM structure and you don’t need to write long XPath.
* Syntax for relativeXPath: //\*[@class=’relativexapath’]
* Example: driver.findElement(By.xpath(“//\*[@class=’relativexapath’]”))

The code for the above is as follows:

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

public class ElementLocatorExample {

public static void main(String[] args) {

// Set the path to the ChromeDriver executable

System.setProperty("webdriver.chrome.driver", " drivers/chromedriver-win64/chromedriver.exe ");

// Launch the Chrome browser

WebDriver driver = new ChromeDriver();

// Navigate to the W3Schools HTML Examples page

driver.get("https://www.w3schools.com/html/html\_examples.asp");

// Locating elements using CSS selectors

// Example 1: Tag and ID

WebElement element1 = driver.findElement(By.cssSelector("a#topnavbtn\_references"));

// Example 2: Tag and Class

WebElement element2 = driver.findElement(By.cssSelector("h2.w3-container.w3-red"));

// Example 3: Tag and Attribute

WebElement element3 = driver.findElement(By.cssSelector("img[alt='W3Schools.com']"));

// Example 4: Tag, Class, and Attribute

WebElement element4 = driver.findElement(By.cssSelector("div.w3-panel.w3-leftbar.w3-sand.w3-padding"));

// Example 5: Inner text

WebElement element5 = driver.findElement(By.cssSelector("a:contains('W3Schools')"));

// Locating elements using XPath

// Example 1: Absolute XPath

WebElement element6 = driver.findElement(By.xpath("/html/body/div[5]/div[1]/div[1]/div[4]/h2"));

// Example 2: Relative XPath

WebElement element7 = driver.findElement(By.xpath("//\*[@class='w3-sidebar w3-bar-block w3-light-grey w3-card']//a[contains(text(),'Try it Yourself')]"));

// Perform actions on the located elements

// ...

// Close the browser

driver.quit();

}

}

5. Demonstrate how web elements are handled in Selenium

**Step 5.1:** Handling External pop-ups.

* WebDriver has the ability to interact with multiple windows, which includes alerts using the method switchTo. This method allows to switch the control to pop-up while keeping the browser open in the background.
* Syntax for handling the different types of pop ups:
* To click on the “OK” button in pop-up

Syntax: WebDrive driver = new chromeDriver();

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

* To click on the “Cancel” button in pop-up

Syntax: WebDrive driver = new chromeDriver();

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

* To Capture the alert message

Syntax: WebDrive driver = new chromeDriver()

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

* To enter the information

Syntax: WebDrive driver = new chromeDriver()

driver.switchTo().alert().sendKeys(“text”);

* To exit from the pop-up

Syntax: WebDrive driver = new chromeDriver();

driver.switchTo().alert().close();

**Step 5.2:** Handling new tabs and new windows.

* Opening a new tab  
  Syntax: WebDrive driver = new chromeDriver();  
   driver.findElement(By.id(“xyz”)).sendKeys(Keys.CONTROL + “t”);
* Opening a new window

Syntax: WebDriver driver = new chromeDriver();

driver.findElements(By.id(“xyz”).sendKeys(Keys.CONTROL + “w”);

**6. Demonstrate how to automate calendars on the web page**

**Step 1:** Create a Selenium project

* Open Eclipse and create a new Java project.
* Add selenium jar files to the build path.
* Add browser executable files in the **resource** folder.

**Step 2:** Write code for calendar automation

* Create a Java file with the name **calendar.java** and write the code given below:

import java.util.List;

import org.openqa.selenium.By;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

public class DatePicker

{

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

{

String dot="19/October/2020";

String date,month,year;

String caldt,calmonth,calyear;

/\*

\* Split the String into String Array

\*/

String dateArray[]= dot.split("/");

date=dateArray[0];

month=dateArray[1];

year=dateArray[2];

ChromeDriver driver=new ChromeDriver();

driver.get("http://cleartrip.com");

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

WebElement cal;

cal=driver.findElement(By.className("calendar"));

calyear=driver.findElement(By.className("ui-datepicker-year")).getText();

/\*\*

\* Select the year

\*/

while (!calyear.equals(year))

{

driver.findElement(By.className("nextMonth")).click();

calyear=driver.findElement(By.className("ui-datepicker-year")).getText();

System.out.println("Displayed Year::" + calyear);

}

calmonth=driver.findElement(By.className("ui-datepicker-month")).getText();

/\*\*

\* Select the Month

\*/

while (!calmonth.equalsIgnoreCase(month))

{

driver.findElement(By.className("nextMonth ")).click();

calmonth=driver.findElement(By.className("ui-datepicker-month")).getText();

}

cal=driver.findElement(By.className("calendar"));

/\*\*Select the Date

\*/

List<WebElement> rows,cols;

rows=cal.findElements(By.tagName("tr"));

for (int i = 1; i < rows.size(); i++)

{

cols=rows.get(i).findElements(By.tagName("td"));

for (int j = 0; j < cols.size(); j++)

{

caldt=cols.get(j).getText();

if (caldt.equals(date))

{

cols.get(j).click();

break;

}

}

}

}

}

* Run the project as a Java application.
  1. **Using Selenium WebDriver, write a program to handle alerts.**

**Step 1:** Handling External pop-ups.

* WebDriver has the ability to interact with multiple windows, which includes alerts using the method switchTo. This method allows to switch the control to pop-up while keeping the browser open in the background.
* Syntax for handling the different types of pop ups:
* To click on the “OK” button in pop-up

Syntax: WebDrive driver = new chromeDriver();

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

* To click on the “Cancel” button in pop-up

Syntax: WebDrive driver = new chromeDriver();

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

* To Capture the alert message

Syntax: WebDrive driver = new chromeDriver()

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

* To enter the information

Syntax: WebDrive driver = new chromeDriver()

driver.switchTo().alert().sendKeys(“text”);

* To exit from the pop-up

Syntax: WebDrive driver = new chromeDriver();

driver.switchTo().alert().close();

**Step 7.2:** Handling new tabs and new windows.

* Opening a new tab  
  Syntax: WebDrive driver = new chromeDriver();  
   driver.findElement(By.id(“xyz”)).sendKeys(Keys.CONTROL + “t”);
* Opening a new window

Syntax: WebDriver driver = new chromeDriver();

driver.findElements(By.id(“xyz”).sendKeys(Keys.CONTROL + “w”);

* 1. **Demonstrate how screenshots are captured and browser profiles are changed in Selenium**.

**Step 1:** Screenshots

* Open Eclipse
* Convert web driver object to **TakeScreenshot**
* Call getScreenshotAs method to create image file
* Copy file to desired location

**Step 1.1** Convert web driver object to TakeScreenshot

Syntax: TakesScreenshot scrShot = (TakesScreenshot)driver;

**Step 1.2** Call getScreenshotAs method to create image file

Syntax: File srcFile = scrShot.getScreenshotAs(OutType.FILE);

**Step 1.3** Copy file to desire location

Syntax: FileUtils.copyFile(source, filePath);

The script looks like this:

A screen shot of a computer program

Description automatically generated

**Step 2:** Browser profiles**:**

* First , close the Firefox if it is open.
* Open Run (Windows+R) and type firefox.exe -p and click OK.
* A dialogue box will open named “Firefox -choose user profile.”
* Select the option “Create Profile” from the window, and a Wizard will open. Click on Next.
* Provide your profile name which you want to create, and click on the Finish button.
  1. **Demonstrate installation and configuration of AutoIT**.

**Step 1:** Installing and Configuring Auto IT

* Download Auto IT from <https://www.autoitscript.com/site/autoit/downloads/> link.
* Save it in one folder.
* Double click on autoit-v3-setup.exe file and click on **Install**.
* After successful installation, open up AutoIT Editor.

C:\Program Files(x86)\AutoIt3\SciTE

10. Handling File Uploads

**Step 10.1:** Handling file upload by SendKeys

* Launch Eclipse and create a Java project.
* Create project: Click on file->New->Java project.
* Enter the project name as **UploadFile** and click on Finish.
* In the project explorer, expand **UploadFile.**
* Right-click on **src** and choose **New->Class.**
* In Package Name, enter **com.ecommerce** and in **Name** enter **Upload** and click on **Finish.**
* Locate the browse button using chropath/firebug.
* Set the path using SendKeys. And the code looks like below:

//Locating 'browse' button

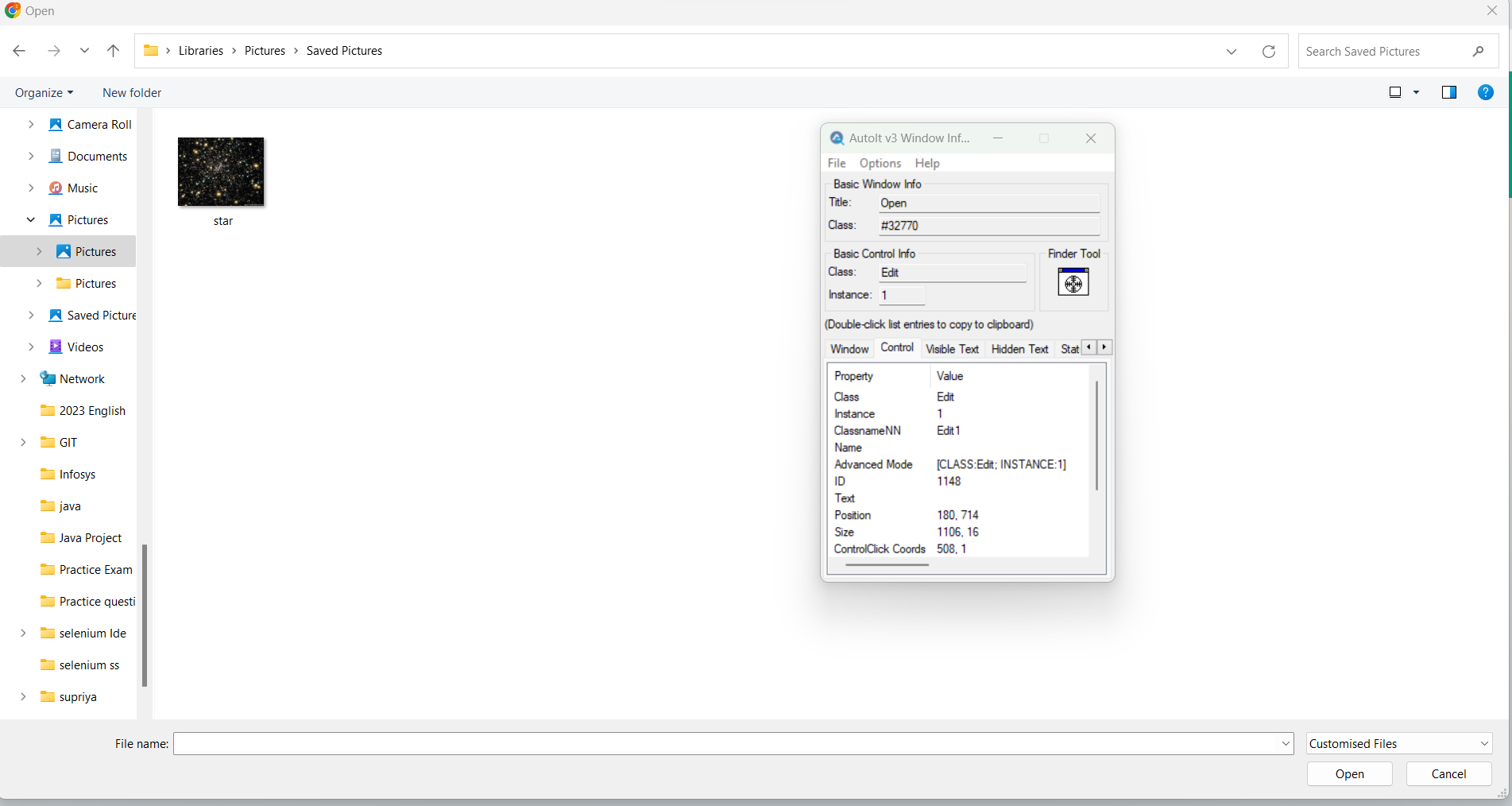
WebElement browse =driver.findElement(By.id("uploadfile"));

//pass the path of the file to be uploaded using Sendkeys method

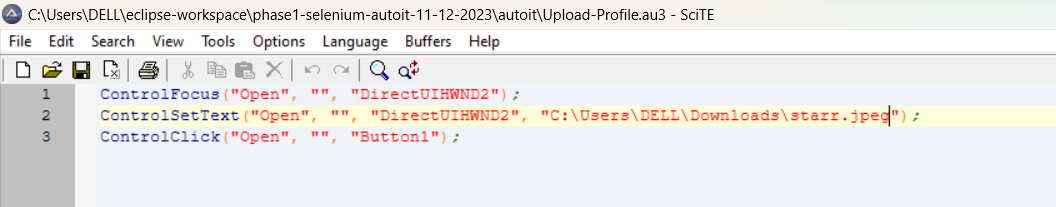
browse.sendKeys("D:\\SoftwareTestingMaterial\\UploadFile.txt");

**Step 10.2:** Handling file upload by AutoIT script

* Go to **Start->Autoit v3->Autoit window info.**
* Now drag the Finder toolbox to the object in which you are interested.



* Build an AutoIT script using **SciTE editor** and write the script using **ControlFocus, ControlsetText, and ControlClick** commands.
* And the script looks like below:



* Save the Script with **.au3** extension.
* Compile the **.au3** script which converts into **.exe** file.
* Pass the **.exe** path into selenium test script using method

**Runtime.getRuntime().exec(“C:\AutoIt\Autoitscript.exe”)**

* The complete script looks like this:

import java.io.IOException;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

public class AutoIt {

private static WebDriver driver = null;

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

driver = new FirefoxDriver();

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

driver.get("http://toolsqa.com/automation-practice-form");

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

Runtime.getRuntime().exec("D:\AutoIt\AutoItTest.exe");

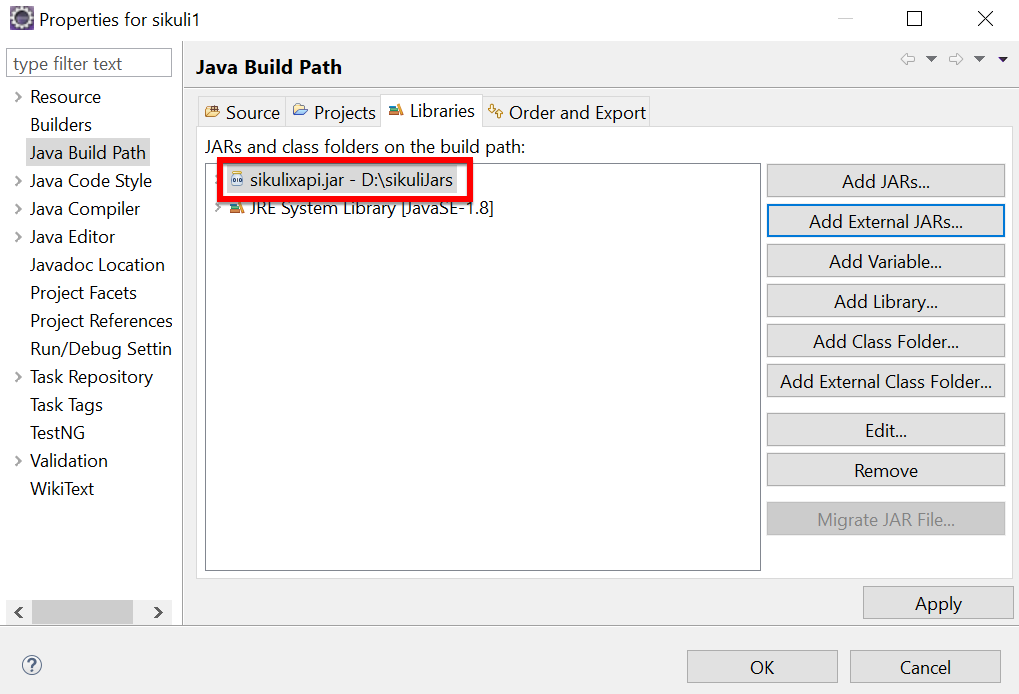
Thread.sleep(5000);

driver.close();

11. Demonstrate how Sikuli is used for UI testing in Selenium.

**Steps 11.1:** Integrating Sikuli with Selenium WebDriver

* Sikuli Jar files are already present in your practice labs. To learn about its directory path details, you can refer to the lab guide for Phase 1.
* Open Eclipse and create a new Java project
* Right-click on the project. Navigate through the given path: Build path->Configure build path->Add external Jars.
* Click on **Apply and OK.**



**Steps 11.2:** Screen class in Sikuli

* Screen class is a base class which contains some predefined methods to perform operations, such as click, double click, providing input to the text box and hover, etc.
* Below are the commonly used methods:
* Click

Syntax: Screen s = new Screen();

s.click()("imag.png”);

* doubleClick

Syntax: Screen s = new Screen();

s.doubleClick()("imag.png”);

* Type

Syntax: s.type(“imag.png”, “Text”);

* Hover

Syntax: s.hover(“imag.png”);

* Find

Syntax: s.find(“imag.png”);

**Steps 11.3:** Pattern class in Sikuli

* Pattern class is used to associate the image file to identify the element
* Pattern class takes the path of the image as a parameter
* Below are the commonly used methods:
* getFileName

Syntax: Pattern p = new Pattern(“D:\Test\imag.png”)

* Similar

Syntax: Pattern p1 = p.similar Pattern(“0.7f”);

* Exact

Syntax: Pattern p1 = p.exact();

The script looks like this:

package com.simplilearn.seleniumtest;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.sikuli.script.FindFailed;

import org.sikuli.script.Pattern;

import org.sikuli.script.Screen;

/\*\*

\* This class demonstrate test of selenium with sikuli tools

\*/

public class AmazonSikuliTest {

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

// step1: formulate a test domain url & driver path

String siteUrl = "https://www.amazon.in";

String driverPath = "drivers/windows/geckodriver.exe";

// step2: set system properties for selenium dirver

System.setProperty("webdriver.geckodriver.driver", driverPath);

// step3: instantiate selenium webdriver

WebDriver driver = new FirefoxDriver();

Screen screen = new Screen();

// make browser window full screen

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

// step4: launch browser

driver.get(siteUrl);

Thread.sleep(3000);

Pattern pattern = new Pattern("C:\\DELL\\Eclipse workshop\\phase1-selenium-sikuli-12-12-2023\\sikuli-inputs\\amazon-tv.png");

Thread.sleep(3000);

screen.click(pattern);

// step6: close driver

// driver.close();

}

}

* Run the script and notice the action performed on the image (The path, which we have mentioned in the script).

12. Demonstrate the usage of JDBC in Selenium.

**Step 12.1:** Creating a table in Database

* Create a table and enter the data in the table in the Database.

**Step 12.2:** Writing the JDBC connection integrating with selenium

* Load the driver class

Syntax: class.forName(“Connection URL”);

com.mysql.cj.jdbc.Driver

URL -

* Create a Connection

Connection con = DriverManager.getConnection(“URL”, “UserName”, “Password”);

* Create a statement

Syntax: Statement stmt = con.createStatement();

* Execute SQL query

Syntax: ResultSet rs= stmt.executeQuery(“sql query”);

* Close the connection

Syntax: Con.close();

The code in Eclipse will look like this:

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.testng.annotations.AfterTest;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Test;

public class TestDatabaseWithSelenium {

private WebDriver driver;

@BeforeTest

public void setup() {

// Set the path to the ChromeDriver executable

System.setProperty("webdriver.chrome.driver", " drivers/chromedriver-win64/chromedriver.exe");

// Create a new instance of the ChromeDriver

driver = new ChromeDriver();

}

@Test

public void testVerifyDB() throws ClassNotFoundException, SQLException {

// Step 1: Load the driver class

Class.forName("oracle.jdbc.driver.OracleDriver");

// Step 2: Create the connection object

Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "your\_username", "your\_password");

// Step 3: Create the statement object

Statement stmt = con.createStatement();

// Step 4: Execute the SQL query

ResultSet rs = stmt.executeQuery("SELECT \* FROM Products");

// Step 5: Iterate through the result set and perform web testing

while (rs.next()) {

int productId = rs.getInt(1);

String productName = rs.getString(2);

String productDescription = rs.getString(3);

// Perform web testing using Selenium

driver.get("https://www.seleniumhq.org");

WebElement searchInput = driver.findElement(By.id("q"));

searchInput.sendKeys(productName);

searchInput.submit();

// Print the database record and web page title

System.out.println("Product ID: " + productId);

System.out.println("Product Name: " + productName);

System.out.println("Product Description: " + productDescription);

System.out.println("Web Page Title: " + driver.getTitle());

System.out.println("--------------------------------------------");

}

// Step 6: Close the connection object

con.close();

}

@AfterTest

public void teardown() {

// Close the browser

driver.quit();

}

}