1 .Demonstrate the features of the selenium

Selenium is a powerful tool for automating web browsers and is widely used for testing web applications. It supports multiple programming languages, including Python, Java, C#, and more. Here, I'll demonstrate some basic features of Selenium using Python.

1.Launching Browser-

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class BasicSeleniumExample {
    public static void main(String[] args) {
        // Set the path to the ChromeDriver executable
        System.setProperty("webdriver.chrome.driver", "path/to/chromedriver.exe");

        // Create a new instance of the Chrome driver
        WebDriver driver = new ChromeDriver();

        // Open a website
        driver.get("https://www.example.com");

        // Close the browser
        driver.quit();
    }
}
```

_-Locating the Elements:-

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class ElementLocatingExample {
```

```
    public static void main(String[] args) {
        System.setProperty("webdriver.chrome.driver", "path/to/chromedriver.exe");
        WebDriver driver = new ChromeDriver();

        // Find element by ID
        WebElement elementById = driver.findElement(By.id("element_id"));

        // Find element by name
        WebElement elementByName = driver.findElement(By.name("element_name"));

        // Find element by class name
        WebElement elementByClass = driver.findElement(By.className("element_class"));

        // Find element by CSS selector
        WebElement elementByCss = driver.findElement(By.cssSelector("css_selector"));

        // Find element by XPath
        WebElement elementByXPath = driver.findElement(By.xpath("//xpath_expression"));

        // Close the browser
        driver.quit();
    }
}
```

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

This guide has mainly three subsections, namely:

4.2.1 Downloading Selenium Standalone Server jar

4.2.2 Launching Eclipse and creating a Java project

4.2.3 Configuring WebDriver with Eclipse

Step 4.2.1: Downloading Selenium Standalone Server jar

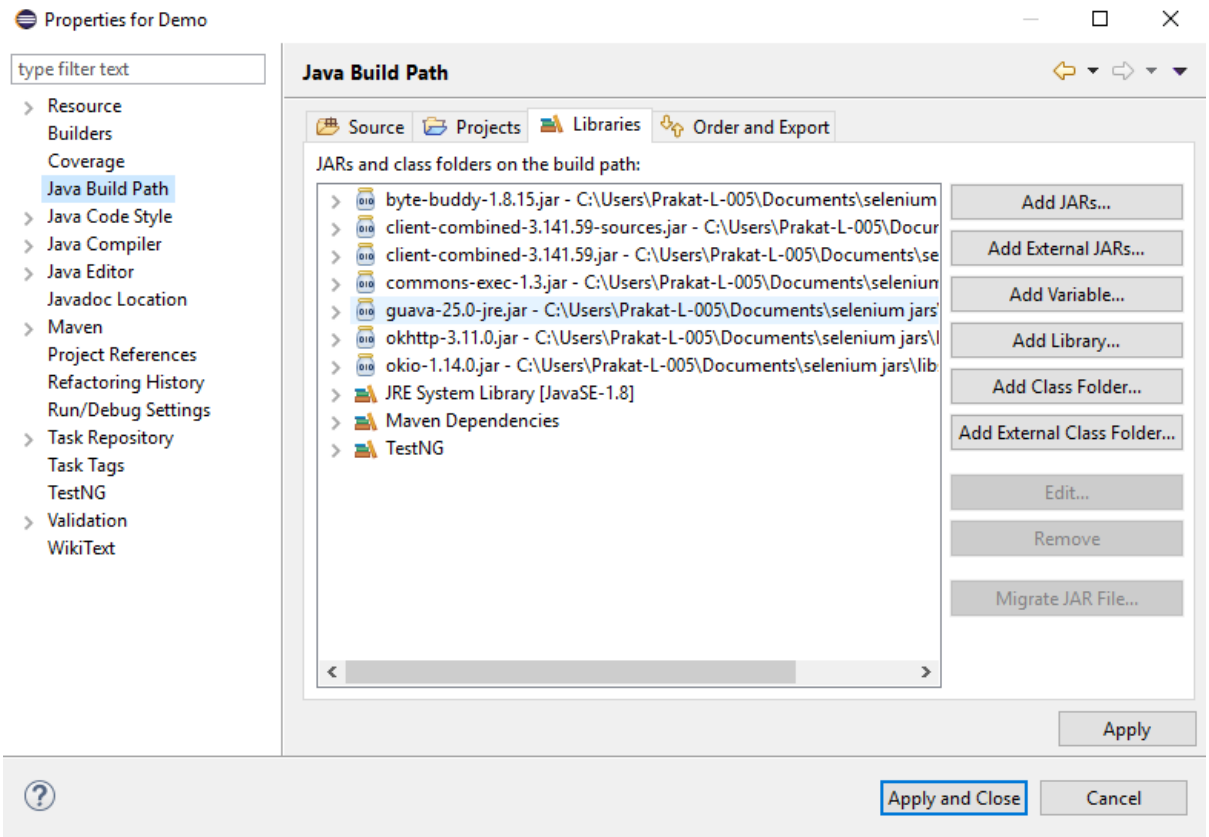
- Selenium is already installed in your practice lab. (Refer QA to QE: Lab Guide - Phase 1)

Step 4.2.2: Launching Eclipse and creating a Java project

- Launch the Eclipse and create a Workspace.
- Create Project:
  Click on File -> New -> Java Project.

Step 4.2.3: 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:



3. Demonstrate how elements are located using Selenium WebDriver.

Locating elements is a crucial aspect of using Selenium WebDriver for automated testing. Selenium provides various methods to locate HTML elements on a web page. Below are examples of different element locating strategies using Selenium WebDriver in Java.

<!DOCTYPE html>

<html>

<head>

<title>Element Locating Example</title>

</head>

<body>

<h1 id="heading">Welcome to Selenium WebDriver</h1>

<input type="text" id="username" name="username" class="input-field">

```
<button id="loginButton" onclick="login()">Login</button>
<select id="dropdown">
<option value="option1">Option 1</option>
<option value="option2">Option 2</option>
<option value="option3">Option 3</option>
</select>

<script>
        function login() {
            alert("Login button clicked!");
        }
</script>
</body>
</html>
```

-Locating Elements By ID-

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class ElementLocatingExample {
    public static void main(String[] args) {
        System.setProperty("webdriver.chrome.driver", "path/to/chromedriver.exe");
        WebDriver driver = new ChromeDriver();
        driver.get("file:///path/to/your/html/file.html");

        // Locate element by ID
        WebElement heading = driver.findElement(By.id("heading"));
```

```
        System.out.println("Heading Text: " + heading.getText());


        // Close the browser
        driver.quit();
    }
}
```

Locating elements by class name-

```
// Locate element by class name
WebElement inputField = driver.findElement(By.className("input-field"));
inputField.sendKeys("Text to input");
```

Locating the elements by tag name-

```
// Locate element by tag name
List<WebElement> buttons = driver.findElements(By.tagName("button"));
for (WebElement button : buttons) {
    System.out.println("Button Text: " + button.getText());
}
```

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

Step 4.4.1: 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:
    a. **Tag and ID**
        - Syntax: “css = tag#id”
        - Example: driver.findElement(By.cssSelector(“input#email”));

b. Tag and Class
   - Syntax: “css = tag.class”
   - Example: driver.findElement(By.cssSelector(”input.inputtext”));

C. Tag and Attribute
   - Syntax: “css = tag[attribute=value]”
   - Example: driver.findElement(By.cssSelector(“input[name=lastName]”));

c. Tag, Class, and Attribute
   - Syntax: “tag.class[attribute=value]”
   - Example: driver. findElement(By.cssSelector(“input.inputtext[tabindex=1]”));

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

Step 4.4.2: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:

a. 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.

b. **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:

```
importorg.openqa.selenium.By;
importorg.openqa.selenium.WebDriver;
importorg.openqa.selenium.WebElement;
importorg.openqa.selenium.chrome.ChromeDriver;

publicclassElementLocatorExample {
   publicstaticvoidmain(String[] args) {
      // Set the path to the ChromeDriver executable
      System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");

      // Launch the Chrome browser
      WebDriverdriver=newChromeDriver();

      // 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
      WebElementelement1=driver.findElement(By.cssSelector("a#topnavbtn_references"));

      // Example 2: Tag and Class
      WebElementelement2=driver.findElement(By.cssSelector("h2.w3-container.w3-red"));

      // Example 3: Tag and Attribute
```

```
        WebElementelement3=driver.findElement(By.cssSelector("img[alt='W3Schools.com']"));

        // Example 4: Tag, Class, and Attribute
        WebElementelement4=driver.findElement(By.cssSelector("div.w3-panel.w3-leftbar.w3-
sand.w3-padding"));

        // Example 5: Inner text
        WebElementelement5=driver.findElement(By.cssSelector("a:contains('W3Schools')"));

        // Locating elements using XPath
        // Example 1: Absolute XPath

WebElementelement6=driver.findElement(By.xpath("/html/body/div[5]/div[1]/div[1]/div[4]/h2
"));

        // Example 2: Relative XPath
        WebElementelement7=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();
    }
}
```

Note: Replace the "**path/to/chromedriver**" with the actual path to the chromedriver executable on your system.

Step 4.4.3: Pushing the code to GitHub repositories

Open your command prompt and navigate to the folder where you have created your files cd <folder path>

Initialize your repository using the following command:

git init

Add all the files to your git repository using the following command:

git add .

Commit the changes using the following command:

git commit . -m “Changes have been committed.”

Push the files to the folder you initially created using the following command:

git push -u origin master

5 .Demonstrate how web elements are handled in Selenium.

Handling web elements is a fundamental aspect of Selenium WebDriver automation. Once you've located an element on a web page, you can interact with it by performing various actions such as clicking, sending keys, retrieving text, and more. Here's a demonstration using Java with Selenium WebDriver:

Assume you have the following HTML page:

```
<!DOCTYPE html>
<html>
<head>
<title>Element Handling Example</title>
</head>
<body>
<input type="text" id="username" name="username">
<input type="password" id="password" name="password">
<button id="loginButton" onclick="login()">Login</button>
<div id="message">Welcome!</div>

<script>
	function login() {
	    var username = document.getElementById("username").value;
	    var password = document.getElementById("password").value;

	    if (username === "user" && password === "pass") {
	        document.getElementById("message").innerText = "Login successful!";
```

```
        } else {
            document.getElementById("message").innerText = "Invalid credentials.";
        }
    }
</script>
</body>
</html>
// Locate login button and click it
WebElement loginButton = driver.findElement(By.id("loginButton"));
loginButton.click();
---// Locate message element and retrieve text
WebElement messageElement = driver.findElement(By.id("message"));
String messageText = messageElement.getText();
System.out.println("Message Text: " + messageText);
---// Check if an element is present
if (driver.findElement(By.id("message")).isDisplayed()) {
   System.out.println("Message element is present on the page.");
} else {
   System.out.println("Message element is not present on the page.");
}
```

6 .Demonstrate how to automate calendars on the web page

Automating calendars on a web page can be achieved using Selenium WebDriver. The challenge with calendars is that they often involve selecting specific dates or ranges. Here, I'll demonstrate how to automate a simple scenario where we pick a date from a calendar using Selenium WebDriver in Java.

```
<!DOCTYPE html>
<html>
```

```
<head>
<title>Calendar Automation Example</title>
<link rel="stylesheet" href="https://code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css">
<script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>
<script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>
<script>
    $(function () {
        $("#datepicker").datepicker();
    });
</script>
</head>
<body>
<p>Select a date: <input type="text" id="datepicker"></p>
</body>
</html>
```

Automating Calendar Selection:

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

import java.util.concurrent.TimeUnit;

public class CalendarAutomationExample {
    public static void main(String[] args) {
        System.setProperty("webdriver.chrome.driver", "path/to/chromedriver.exe");
```

```
        WebDriver driver = new ChromeDriver();
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        driver.get("file:///path/to/your/html/file.html");

        // Locate the datepicker input field
        WebElement datepicker = driver.findElement(By.id("datepicker"));

        // Click on the datepicker input field to open the calendar
        datepicker.click();

        // Select a specific date (e.g., 10th day of the month)
        WebElement dateToSelect = driver.findElement(By.xpath("//a[text()='10']"));
        dateToSelect.click();

        // Alternatively, you can use the calendar's built-in functions to select a date
        // WebElement datepickerIcon = driver.findElement(By.cssSelector(".ui-
datepicker-trigger"));
        // datepickerIcon.click();
        // WebElement dateToSelect = driver.findElement(By.xpath("//a[text()='10']"));
        // dateToSelect.click();

        // Close the browser
        driver.quit();
    }
}
```

7 .Using Selenium WebDriver, write a program to handle alerts.

Handling alerts is a common scenario in web automation, and Selenium provides methods

to interact with JavaScript alerts, confirms, and prompts. Here's a simple Java program using Selenium WebDriver to demonstrate how to handle alerts:

```
import org.openqa.selenium.Alert;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class AlertHandlingExample {
    public static void main(String[] args) {
        // Set the path to the ChromeDriver executable
        System.setProperty("webdriver.chrome.driver", "path/to/chromedriver.exe");

        // Create a new instance of the Chrome driver
        WebDriver driver = new ChromeDriver();

        // Navigate to a web page with an alert
        driver.get("https://www.example.com");

        // Locate the button that triggers the alert
        WebElement alertButton = driver.findElement(By.id("alertButton"));

        // Click the button to trigger the alert
        alertButton.click();

        // Switch to the alert
        Alert alert = driver.switchTo().alert();

        // Get the text of the alert
```

```
        String alertText = alert.getText();
        System.out.println("Alert Text: " + alertText);

        // Accept the alert (click OK)
        alert.accept();

        // Alternatively, you can dismiss the alert (click Cancel)
        // alert.dismiss();

        // Close the browser
        driver.quit();
    }
}
```

In this example, we assume there's a button with the ID "alertButton" on the web page, and clicking it triggers a simple alert. The script navigates to the web page, clicks the button to trigger the alert, switches to the alert, retrieves the alert text, and finally accepts it.

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

Step 4.7.1: Screenshots

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

Step 4.7.1.1 Convert web driver object to TakeScreenshot

- Syntax: TakesScreenshot scrShot = (TakesScreenshot)driver;

Step 4.7.1.2 Call getScreenshotAs method to create image file

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

Step 4.7.1.3 Copy file to desire location

- Syntax: FileUtils.copyFile(source, filePath);

The script looks like this:

```
package screenshots.screenshot;
import java.io.File;
import java.io.IOException;
import org.openqa.selenium.By;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import com.sun.jna.platform.FileUtils;

publicclass Screenshots {
		publicstaticvoid main(String[] args )throwsIOException
		{
				System.setProperty("webdriver.chrome.driver","C:\\Users\\Testing-L-
064\\chromedriver_win32\\chromedriver.exe");
			WebDriver driver =new ChromeDriver();
			driver.get("https://www.flipkart.com/");
			WebElement upload =	driver.findElement(By.xpath("//*[@type='text']"));
			upload.click();
			TakesScreenshot ts =(TakesScreenshot)driver;
		File scr = ts.getScreenshotAs(OutputType.FILE);
			FileUtils.copyFile(scr,newFile("/Screenshot/test.png");


		}
}
```

Step 4.7.2: Running the code

- Run the code through eclipse.

Step 4.7.3: Pushing the code to your GitHub repositories

Open your command prompt and navigate to the folder where you have created your files

cd <folder path>

Initialize your repository using the following command:

git init

Add all the files to your git repository using the following command:

git add .

Commit the changes using the following command:

git commit . -m “Changes have been committed.”

Push the files to the folder you initially created using the following command:

git push -u origin master

Step 1.6.4: 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.

9-Demonstrate installation and configuration of AutoIT.

4.10.2 Pushing the code to GitHub repositories

Step 4.10.1: 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

Step 4.10.2: Pushing the code to GitHub repositories

- Open your command prompt and navigate to the folder where you have created your files:

  Cd <folder path>

- Initialize your repository using the following command:

  Git init

- Add all the files to Git repository using following command:

  Git add.

- Commit the changes using the following command:

  Git commit -m “add the comment”

- Push the files to the folder you initially created using the following command:

Git push -u origin master

10-Demonstrate how file uploads are handled in AutoIT.

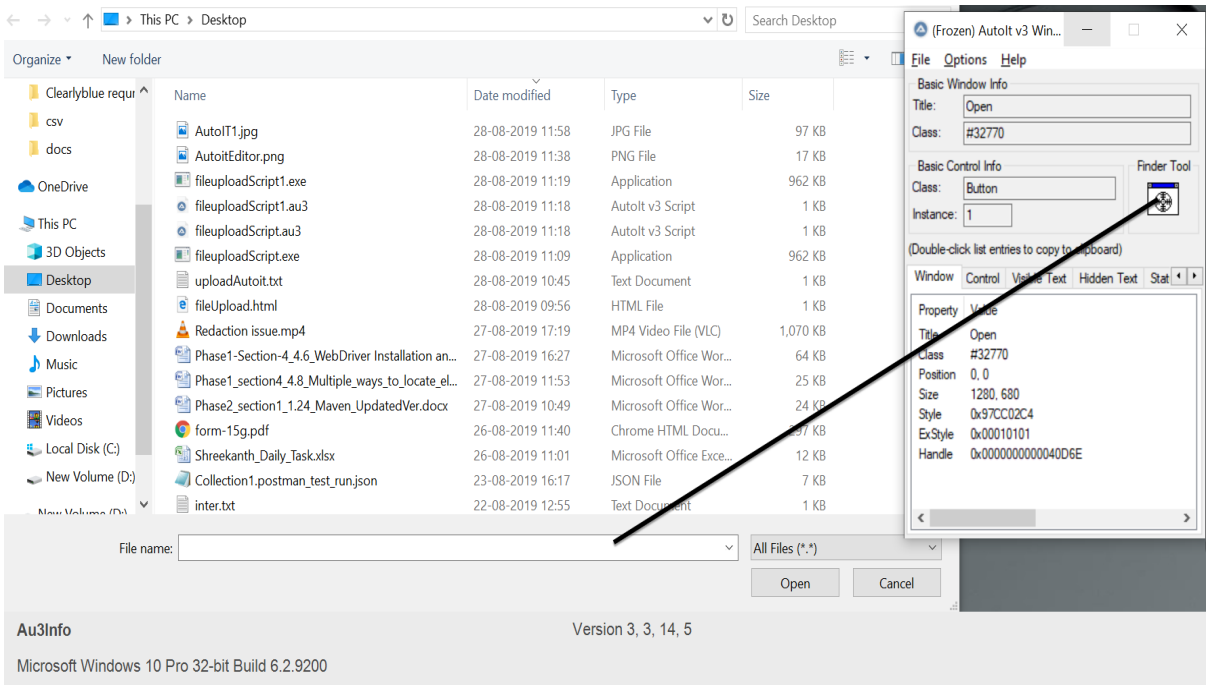
Step 4.9.1: Handling file upload by SendKeys

- 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:

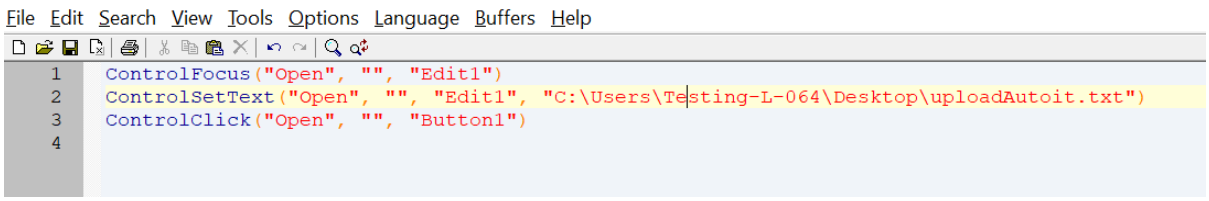
```
//Locating 'browse' button
 WebElement browse =driver.findElement(By.id("uploadfile"));
//pass the path of the file to be uploaded using Sendkeys method
 browse.sendKeys("D:\\SoftwareTestingMaterial\\UploadFile.txt");
```

Step 4.9.2: 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:

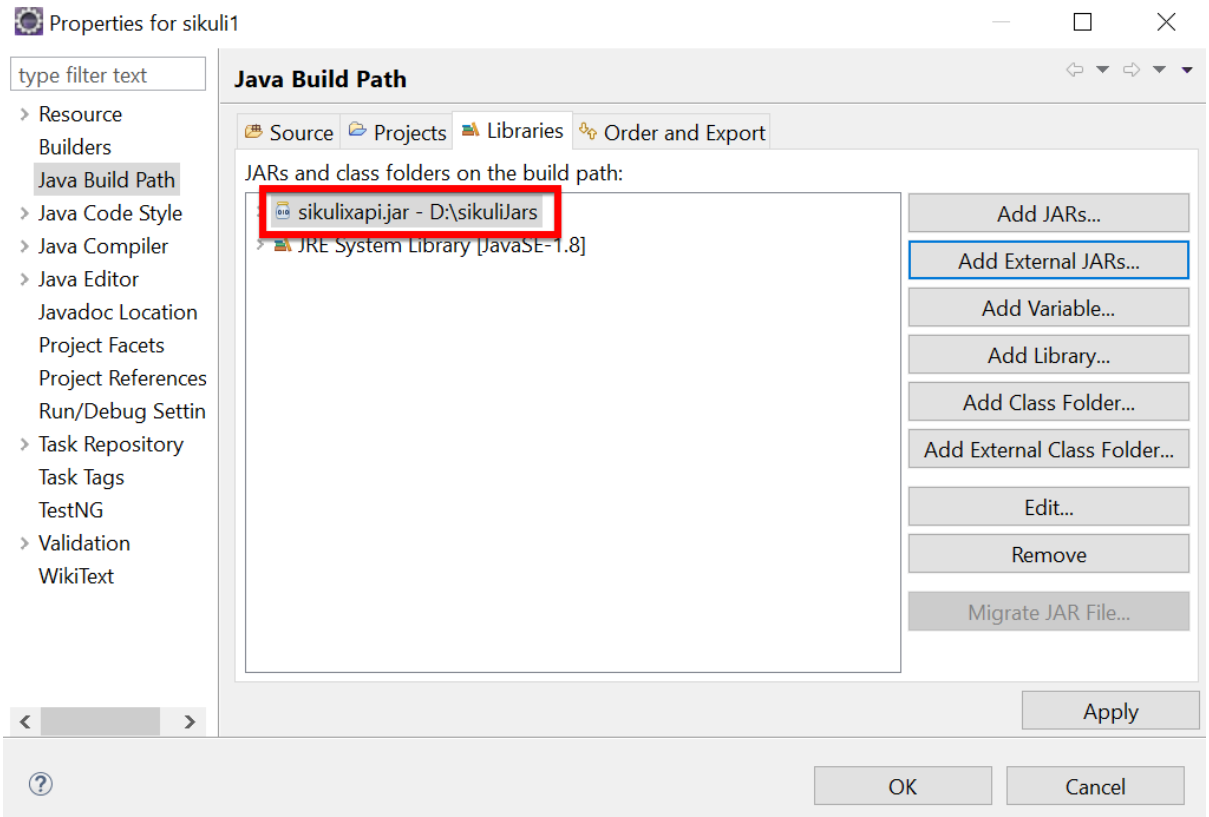
```
import java.io.IOException;
```

```
import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

publicclass AutoIt {

privatestatic WebDriver driver =null;

publicstaticvoid main(String[] args)throwsIOException,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 4.11.1: 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 4.11.2: 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 4.11.3: 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 sikuli1;
import org.sikuli.script.FindFailed;
import org.sikuli.script.Pattern;
import org.sikuli.script.Screen;

publicclass SikuliClass {
	publicstaticvoid main(String[] args )throws FindFailed {
		Screen s =new Screen();
		Pattern p =newPattern("C:\\Users\\Testing\\Desktop\\siluli\\Capture.PNG");
		s.doubleClick(p);
	}
}
```

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

Steps 4.11.4: Pushing the code to your GitHub repositories

- Open your command prompt and navigate to the folder where you have created your files.

cd <folder path>

- Initialize your repository using the following command:

  git init

- Add all the files to your git repository using the following command:

  git add .

- Commit the changes using the following command:

  git commit . -m “Changes have been committed.”

- Push the files to the folder you initially created using the following command:

  git push -u origin master

12 .Demonstrate the usage of JDBC in Selenium.

Step 4.12.1: Creating a table in Database

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

Step 4.12.2: 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:

```
importjava.sql.Connection;
importjava.sql.DriverManager;
importjava.sql.ResultSet;
importjava.sql.SQLException;
importjava.sql.Statement;

importorg.openqa.selenium.By;
importorg.openqa.selenium.WebDriver;
importorg.openqa.selenium.WebElement;
importorg.openqa.selenium.chrome.ChromeDriver;
importorg.testng.annotations.AfterTest;
importorg.testng.annotations.BeforeTest;
importorg.testng.annotations.Test;

publicclassTestDatabaseWithSelenium {
   privateWebDriverdriver;

   @BeforeTest
   publicvoidsetup() {
      // Set the path to the ChromeDriver executable
      System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");

      // Create a new instance of the ChromeDriver
      driver =newChromeDriver();
   }

   @Test
   publicvoidtestVerifyDB() throwsClassNotFoundException, SQLException {
      // Step 1: Load the driver class
      Class.forName("oracle.jdbc.driver.OracleDriver");

      // Step 2: Create the connection object
      Connectioncon=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",
"your_username", "your_password");

      // Step 3: Create the statement object
      Statementstmt=con.createStatement();

      // Step 4: Execute the SQL query
      ResultSetrs=stmt.executeQuery("SELECT * FROM Products");

      // Step 5: Iterate through the result set and perform web testing
      while (rs.next()) {
         intproductId=rs.getInt(1);
         StringproductName=rs.getString(2);
         StringproductDescription=rs.getString(3);
```

```
        // Perform web testing using Selenium
        driver.get("https://www.seleniumhq.org");
        WebElementsearchInput=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
    publicvoidteardown() {
      // Close the browser
      driver.quit();
    }
}
```

Note: Replace the "**path/to/chromedriver**" with the actual path to the **chromedriver** executable on your system.

Step 4.12.3: Pushing the code to GitHub repositories

- Open your command prompt and navigate to the folder where you have created your files.

  Cd <folder path>

- Initialize your repository using the following command:

  Git init

- Add all the files to Git repository using following command:

  Git add.

- Commit the changes using the following command:

  Git commit -m “add the comment”

- Push the files to the folder you initially created using the following

command:

Git push -u origin master