1. **Features of Selenium.**

Description

This section will guide you to understand:

* The features of Selenium

This guide has mainly one subsection, namely:

4.1.1  Features of Selenium

****Steps 4.1.1:****  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.

2 WebDriver Installation and Integration in Eclipse

This section will guide you to:

Install and integrate WebDriver in Eclipse

**Development Environment**

* Eclipse IDE for Enterprise Java Developers Version Oxygen.3a Release (4.7.3a)
* JavaDevelopment Kit Version 8

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 Multiple Ways to Locate Elements

This section will guide you to:

* Locate elements in Multiple ways using selenium web driver

This guide has mainly seven subsections, namely :

4.3.1 Using ID as a Locator

4.3.2Using class name as a Locator

4.3.3 Using name as a Locator

4.3.4 Using Link Text as a Locator

4.3.5 Using Xpath as a Locator

4.3.6 Using CSS Selector as a Locator

4.3.7 Using XPath for handling complex and dynamic elements

**Step 4.3.1:** 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 4.3.2** 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 4.3.3** 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 4.3.4** 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 4.3.5** 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 4.3.6** 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 4.3.7** 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:

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", "path/to/chromedriver");

        // Create a new instance of the ChromeDriver

        WebDriver driver = new ChromeDriver();

        // Launch the website

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

        // Step 4.3.1: Using ID as a Locator

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

        downloadTab.click();

        // Step 4.3.2: Using class name as a Locator

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

        versionsTab.click();

        // Step 4.3.3: Using Name as a Locator

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

        searchField.sendKeys("Selenium WebDriver");

        // Step 4.3.4: Using LinkText as a Locator

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

        aboutTab.click();

        // Step 4.3.5: 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 4.3.6: 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 4.3.7: 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();

    }

}

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

4 Locating Elements through CSS and XPath

This section will guide you to:

* Locate elements on the web page.

This guide has mainly three subsections, namely:

4.4.1 Finding the element present on the page by using CSS Selector.

4.4.2 Finding the element present on the page by using XPath.

4.4.3 Pushing the code to your GitHub repositories

**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:

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 4.4.2:** 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", "path/to/chromedriver");

        // 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();

    }

}

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 Handling Various Web Elements

This section will guide you to:

* Handle various web elements present on the page.

This section is divided into eleven subsections, namely:

4.5.1 Edit box

4.5.2 Link

4.5.3 Button

4.5.4 Image, image link, and image button

4.5.5 Text area

4.5.6 Checkbox

4.57 Radio button

4.5.8 Dropdown list

4.5.9 Web table/HTML table

4.5.10 Frame

### 4.5.11 Switching between tabs in the same browser window

**Step 4.5.1:** Edit box

* Open Eclipse
* It is a basic text control that enables a user to type a small amount of text.
* Operations on Edit box
  + Enter a Value,
  + Clear the Value,
  + Check enabled status,
  + Check edit box existence,
  + Get the value

**Step 4.5.2:** Link

* Link is more appropriately referred to as a hyperlink and connects one web page to another. It allows the user to click their way from page to page.
* Operations on Link
  + Click the link,
  + Check the link existence,
  + Check the link enabled status,
  + Return the link name

**Step 4.5.3:** Button

* This represents a clickable button, which can be used in forms and places in the document that needs a simple, standard button functionality.
* Operations on Button
  + Click
  + Check Enabled status
  + Display status

**Step 4.5.4:** Image, image link, and image button

* It helps in performing actions on images like clicking on the image link or the image button, etc.
* Operations Image
  + Three types of Image elements in Web Environment
  + General Image (No functionality)
  + Image Button (Submits)
  + Image Link (Redirects to another page/location)

**Step 4.5.5:** Text area

* It is an inline element used to designate a plain-text editing control containing multiple lines.
* Return / Capture Text Area or Error message from a web page

**Step 4.5.6:** Checkbox

* This is a selection box or a tick box which is a small interactive box that can be toggled by the user to indicate an affirmative or a negative choice.
* Operations on checkbox
  + Check if the checkbox is displayed or not
  + Check if the checkbox is enabled or not
  + Check if the checkbox is selected or not
  + Select the checkbox
  + Unselect the checkbox

**Step 4.5.7:** Radio button

* It is an option button which is a graphical control element that allows the user to choose only one predefined set of mutually exclusive options.
* Operations on Radio Button
  + Select Radio Button
  + Verify if the Radio Button is displayed or not
  + Verify if the Radio Button is enabled or not
  + Verify if the Radio Button is selected or not
* Example:

oRadioButton.get(1).click();

**Step 4.5.8:** Drop-down list

* It is a graphical control element, similar to the list box, which allows the user to choose one value from the list. When this drop-down list is inactive, it displays only a single value.
* Operations on drop-down list
  + Check the drop-down box’s existence
  + Check if the drop-down is enabled or not
  + Select an item
  + Get Items Count
* Example:

Select fruits = new Select(driver.findElement(By.id("fruits")));

fruits.selectByVisibleText("Banana");

fruits.selectByIndex(1);

**Step 4.5.9:** Web table/HTML table

* Operations on Web Table/HTML Table
  + Get cell value
  + Get Rows Count
  + Get Cells Count

**Step 4.5.10:** Frame

* Operations on Frame
  + Switch from Top window to a frame
  + Switch from a frame to Top window
* Examples:
  + driver.switchTo().frame("iframe1");
  + driver.switchTo().frame("id of the element");

**Step 4.5.11:** Switching between tabs in the same browser window

* Open a new tab using Ctrl + t.
* Driver control automatically switches to the newly opened tab.
* Perform the required operations here.
* Next, switch back to the old tab using Ctrl + Tab. You need to keep pressing this unless you reach the desired tab.
* Once the desired tab is reached, then perform the operations in that tab.
* Example:

driver.switchTo().window(tabs2.get(1));

driver.switchTo().window(tabs2.get(0));

Here is the code for the above steps:

import org.openqa.selenium.By;

import org.openqa.selenium.Keys;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.support.ui.Select;

import java.util.ArrayList;

public class WebElementsDemo {

    public static void main(String[] args) {

        // Set the path to the chromedriver executable

        System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");

        // Launch the Chrome browser

        WebDriver driver = new ChromeDriver();

        // Navigate to Google homepage

        driver.get("https://www.google.com");

        // Step 4.5.1: Edit box

        // Enter a value in the search box

        driver.findElement(By.name("q")).sendKeys("Hello World");

        // Clear the value

        driver.findElement(By.name("q")).clear();

        // Check enabled status of the search box

        boolean isSearchBoxEnabled = driver.findElement(By.name("q")).isEnabled();

        // Check search box existence

        boolean doesSearchBoxExist = driver.findElements(By.name("q")).size() > 0;

        // Get the value from the search box

        String searchBoxValue = driver.findElement(By.name("q")).getAttribute("value");

        // Step 4.5.2: Link

        // Click the "Images" link

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

        // Check the "Images" link existence

        boolean doesImagesLinkExist = driver.findElements(By.linkText("Images")).size() > 0;

        // Check the "Images" link enabled status

        boolean isImagesLinkEnabled = driver.findElement(By.linkText("Images")).isEnabled();

        // Return the "Images" link name

        String imagesLinkName = driver.findElement(By.linkText("Images")).getText();

        // Step 4.5.3: Button

        // Click the "Google Search" button

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

        // Check enabled status of the "Google Search" button

        boolean isSearchButtonEnabled = driver.findElement(By.name("btnK")).isEnabled();

        // Display status of the "Google Search" button

        boolean isSearchButtonDisplayed = driver.findElement(By.name("btnK")).isDisplayed();

        // Step 4.5.4: Image, image link, and image button

        // Operations on Image

        // Step 4.5.5: Text area

        // Return / Capture Text Area or Error message from a web page

        String textAreaValue = driver.findElement(By.tagName("textarea")).getAttribute("value");

        // Step 4.5.6: Checkbox

        // Check if the "I'm Feeling Lucky" checkbox is displayed or not

        boolean isFeelingLuckyCheckboxDisplayed = driver.findElement(By.name("btnI")).isDisplayed();

        // Check if the "I'm Feeling Lucky" checkbox is enabled or not

        boolean isFeelingLuckyCheckboxEnabled = driver.findElement(By.name("btnI")).isEnabled();

        // Check if the "I'm Feeling Lucky" checkbox is selected or not

        boolean isFeelingLuckyCheckboxSelected = driver.findElement(By.name("btnI")).isSelected();

        // Select the "I'm Feeling Lucky" checkbox

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

        // Unselect the "I'm Feeling Lucky" checkbox

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

        // Step 4.5.7: Radio button

        // Select the "Search" radio button

        driver.findElement(By.cssSelector("input[name='tbm'][value='s']")).click();

        // Verify if the "Search" radio button is displayed or not

        boolean isSearchRadioButtonDisplayed = driver.findElement(By.cssSelector("input[name='tbm'][value='s']")).isDisplayed();

        // Verify if the "Search" radio button is enabled or not

        boolean isSearchRadioButtonEnabled = driver.findElement(By.cssSelector("input[name='tbm'][value='s']")).isEnabled();

        // Verify if the "Search" radio button is selected or not

        boolean isSearchRadioButtonSelected = driver.findElement(By.cssSelector("input[name='tbm'][value='s']")).isSelected();

        // Step 4.5.8: Drop-down list

        // Check the drop-down box existence

        boolean doesDropDownExist = driver.findElements(By.name("lang")).size() > 0;

        // Check if the drop-down box is enabled or not

        boolean isDropDownEnabled = driver.findElement(By.name("lang")).isEnabled();

        // Select an item from the drop-down list

        Select dropdown = new Select(driver.findElement(By.name("lang")));

        dropdown.selectByVisibleText("English");

        dropdown.selectByValue("fr");

        // Get the count of items in the drop-down list

        int itemCount = dropdown.getOptions().size();

        // Step 4.5.9: Web table/HTML table

        // Get cell value from a table

        String cellValue = driver.findElement(By.xpath("//table/tbody/tr[1]/td[1]")).getText();

        // Get the count of rows in a table

        int rowCount = driver.findElements(By.xpath("//table/tbody/tr")).size();

        // Get the count of cells in a table row

        int cellCount = driver.findElements(By.xpath("//table/tbody/tr[1]/td")).size();

        // Step 4.5.10: Frame

        // Switch from the default content to a frame

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

        // Switch back from a frame to the default content

        driver.switchTo().defaultContent();

        // Step 4.5.11: Switching between tabs in the same browser window

        // Open a new tab

        driver.findElement(By.cssSelector("body")).sendKeys(Keys.CONTROL + "t");

        // Switch to the newly opened tab

        ArrayList<String> tabs = new ArrayList<>(driver.getWindowHandles());

        driver.switchTo().window(tabs.get(1));

        // Perform operations in the new tab

        // Switch back to the old tab

        driver.findElement(By.cssSelector("body")).sendKeys(Keys.CONTROL + "\t");

        driver.switchTo().defaultContent();

        // Perform operations in the old tab

        // Close the browser

        driver.quit();

    }

}

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

6 Date Picker

This section will guide you to:

* Work with the date picker library and automate calendars on web pages.

This guide has three subsections, namely:

4.6.1 Create a Selenium project

4.6.2 Write code for calendar automation

4.6.3 Push the code to your GitHub repository

**Step 4.6.1:** 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 4.6.2:** 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="9/October/2018";**

**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.

**Step 4.6.3:** Push the code to your GitHub repository

* 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**

8 Working with External Elements

This section will guide you to:

* How to handle External elements using Selenium.

**Development Environment**

* Eclipse IDE for Enterprise Java Developers Version Oxygen.3a Release (4.8.3a)
* JavaDevelopment Kit Version 8
* Selenium standalone server Version 3.141.59

This guide has mainly three subsections, namely:

4.8.1 Handling external popups.

4.8.2 Handling new tabs and new windows.

4.8.3 Pushing the code to your GitHub Repository.

**Step 4.8.1:** 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 4.8.2:** 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”);

**Step 4.8.3:** 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

7 Screenshots and Browser Profiles

This section will guide you to:

* How to take screenshots using selenium web driver and how to set the browser profile.

**Development Environment**

* Eclipse IDE for Enterprise Java Developers Version Oxygen.3a Release (4.7.3a)
* JavaDevelopment Kit Version 8
* Selenium standalone server Version 3.141.59

This guide has mainly four subsections, namely:

4.7.1 Screenshots

4.7.2 Running the code

4.7.3 Pushing the code to your GitHub repositories

4.7.4 Browser profiles

**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;

**public** **class** Screenshots {

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

{

**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, **new** **File**("/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.

10 Auto IT Installation and Configuration

This section will guide you to understand:

* How to install and configure Auto IT

**Development Environment:**

* Eclipse IDE for Enterprise Java Developers Version Oxygen.3a Release (4.7.3a)
* JavaDevelopment Kit Version 8

This guide has mainly two subsection, namely:

4.10.1 Installing and configuring Auto IT

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

9 Handling File Uploads

This section will guide you to:

* Upload the Desktop file in Selenium WeDriver

**Development Environment**

* Eclipse IDE for Enterprise Java Developers Version Oxygen.3a Release (4.7.3a)
* JavaDevelopment Kit Version 8
* Selenium standalone server
* Autoit

This guide has mainly two subsections, namely:

4.9.1 Handing file upload by **SendKeys**

4.9.2 Handling file upload by **AutoIT script**

4.9.3 Pushing the code to the GitHub repositories

**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

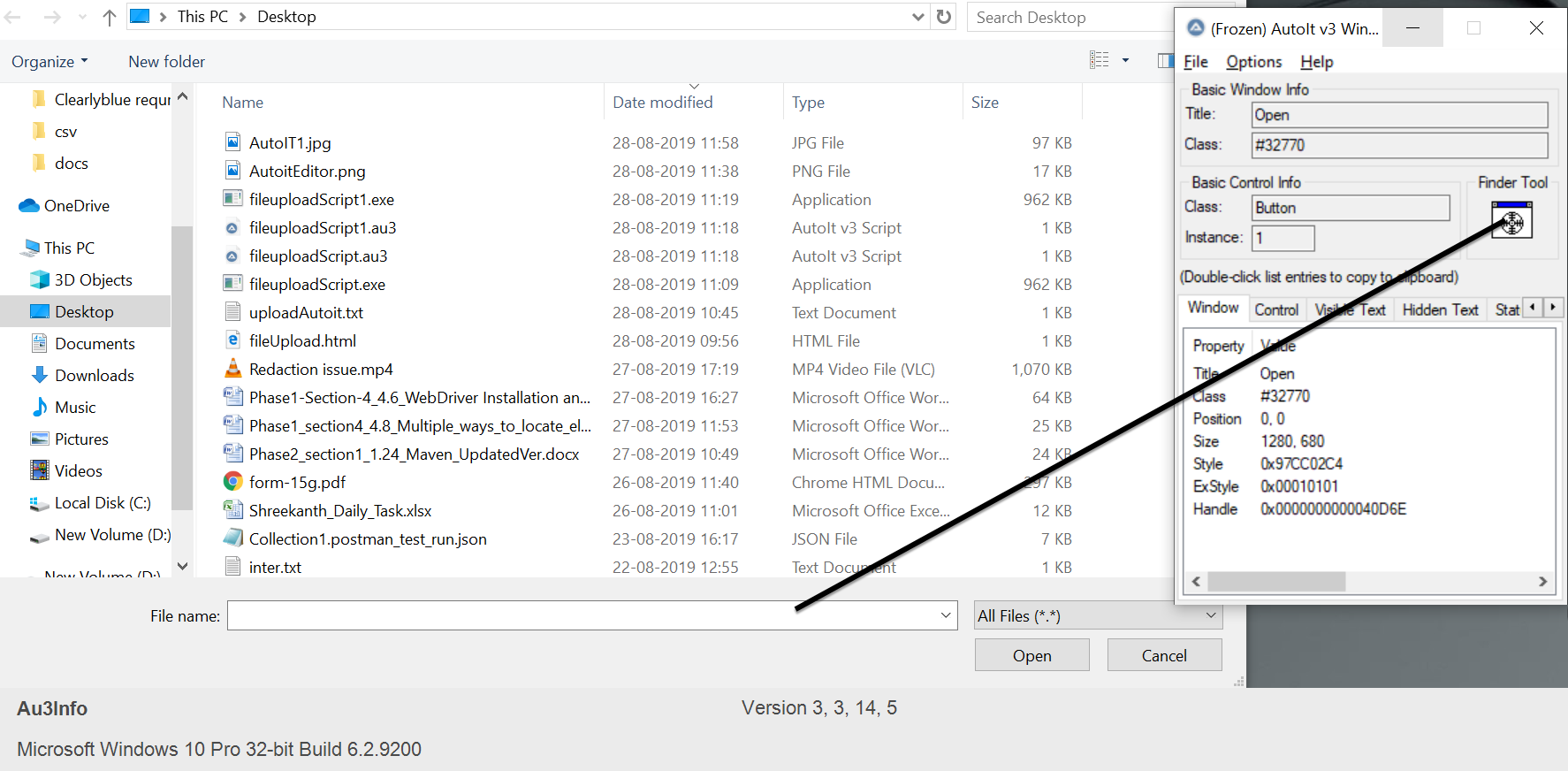
WebElement browse =driver.findElement(By.id("uploadfile"));

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

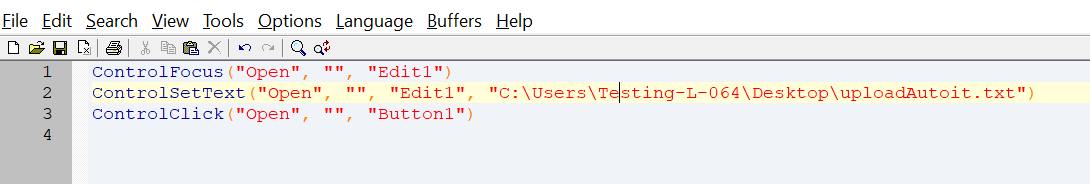
browse.sendKeys("D:\\SoftwareTestingMaterial\\UploadFile.txt");

**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;

**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();

**Step 4.9.3:** 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

11 Sikuli for UI Testing

This section will guide you to:

* Integrate Sikuli with Selenium WebDriver and interact with web elements

**Development Environment:**

* Eclipse IDE for Enterprise Java Developers Version Oxygen.3a Release (4.7.3a)
* JavaDevelopment Kit Version 8

This guide has mainly four subsections, namely:

4.11.1 Integrating Sikuli with Selenium WebDriver

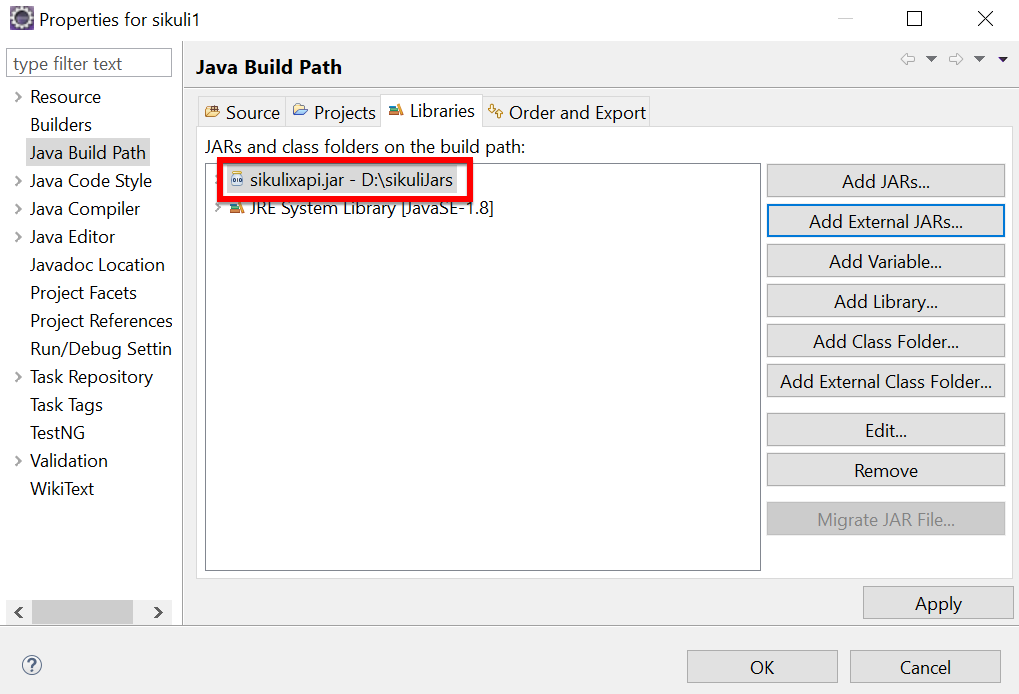
4.11.2 Screen class in Sikuli

4.11.3 Pattern class in Sikuli

4.11.4 Pushing the code to your GitHub repositories

**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;

**public** **class** SikuliClass {

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

Screen s = **new** Screen();

**Pattern** p = **new** **Pattern**("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 Selenium and JDBC

This section will guide you to:

* Integrate JDBC with selenium

**Development Environment:**

* Eclipse IDE for Enterprise Java Developers Version Oxygen.3a Release (4.7.3a)
* JavaDevelopment Kit Version 8
* Selenium Standalone Server Version 3.141.59
* Structured query language server Version SQL Server 2016 SP2

This guide has mainly three subsections, namely:

4.12.1 Creating a Table in Database

4.12.2 Writing the JDBC connection integrating with selenium

4.12.3 Pushing the code to GitHub repositories

**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:

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", "path/to/chromedriver");

        // 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();

    }

}

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