1- Handling alerts :

1. Switching to the Alert

To handle an alert, you first need to switch the driver context to the alert.

```
Alert alert = driver.switchTo().alert();
```

2. Accepting the Alert

If the alert is informational or requires a simple acknowledgment (e.g., clicking "OK"), you can accept it:

```
alert.accept();
```

3. Dismissing the Alert

If the alert offers options, like "OK" and "Cancel," and you need to click "Cancel," use `dismiss()`:

```
alert.dismiss();
```

4. Retrieving Alert Text

To verify or log the message displayed in the alert, you can use `getText()`:

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

5. Sending Text to a Prompt Alert

For alerts that accept text input (prompt alerts), use `sendKeys()` to input the text.

```
alert.sendKeys("Sample Text");
alert.accept();  // Accept after entering text
```

2- JS Executor

1. Scrolling Actions

- **Scroll to a Specific Element**: Useful for bringing elements into view, especially if they are far down the page.

```
js.executeScript("arguments[0].scrollIntoView(true);", element);
```

- **Scroll to Page Bottom**: Often used in infinite scrolling pages to load additional content.

```
js.executeScript("window.scrollTo(0, document.body.scrollHeight);");
```

2. Clicking Hidden Elements

- **Click an Element Not Directly Clickable**: Ideal for bypassing issues with overlays or elements that are not easily accessible.

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

3. Filling Form Fields

- **Set Value of Hidden or Dynamic Input Fields**: This bypasses `sendKeys()` and directly sets a value, useful for complex forms or hidden fields.

```
js.executeScript("arguments[0].value='test@example.com';",
emailInputElement);
```

4. Retrieve Browser and Document Properties

- **Get Page Title or URL**:

```
String pageTitle = (String) js.executeScript("return document.title;");
String pageURL = (String) js.executeScript("return document.URL;");
```

- **Get Element's Inner Text or HTML**: Retrieve the text or HTML content of an element.

```
String innerText = (String) js.executeScript("return
arguments[0].innerText;", element);
```

5. Simulating Mouse Hover

- **Hover Over an Element**: Useful for dropdown menus or tooltips that appear only on hover.

```
js.executeScript("var evt = document.createEvent('MouseEvents');
evt.initMouseEvent('mouseover', true, true);
arguments[0].dispatchEvent(evt);", element);
```

6. Highlight Element for Debugging

- **Add a Temporary Highlight**: Useful for visually identifying elements during test execution.

```
js.executeScript("arguments[0].style.border='3px solid red'", element);
```

7. Managing Alerts

- **Triggering JavaScript Alerts**: Creates a custom alert box that can be used to pause the test or display information.

```
java
js.executeScript("alert('Test Alert');");
```

8. Calculations or Element Dimensions

- **Get Element Position or Size**: Retrieve coordinates, height, or width of an element, which can be useful in responsive design tests.

```
Long width = (Long) js.executeScript("return
arguments[0].offsetWidth;", element);
Long height = (Long) js.executeScript("return
arguments[0].offsetHeight;", element);
```

Another usages:

- **JavascriptExecutor in Selenium to click a button**

```
[java]

js.executeScript(“document.getElementByID(‘element id ’).click();”);

[/java]
```

- **JavascriptExecutor in Selenium to send text**

```
[java]

js.executeScript(“document.getElementByID(‘element id ’).value = ‘xyz’;”);

[/java]
```

- **JavascriptExecutor in Selenium to interact with checkbox**

```
[java]

js.executeScript(“document.getElementByID(‘element id ’).checked=false;”);

[/java]
```

- **JavascriptExecutor in Selenium to refresh the browser window**

```
[java]

js.executeScript(“location.reload()”);

[/java]
```

Use Case Scenarios

To explain the concept of JavascriptExecutor, let’s look at two simple use cases:

Example 1

Problem Statement: Generate an alert window using JavascriptExecutor.

Objective: Create a login automation script using Selenium that generates an alert window using JavascriptExecutor methods.

Code:

```
[java]

package newpackage;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.testng.annotations.Test;

import org.openqa.selenium.JavascriptExecutor;

public class LoginAutomation {

@Test



public void login() {

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

WebDriver driver = new ChromeDriver();

JavascriptExecutor js = (JavascriptExecutor)driver;

driver.manage().window().maximize();
driver.get("https://www.browserstack.com/users/sign_in");

js.executeScript("document.getElementById('user_email_login').value='rbc@xyz.com';");

js.executeScript("document.getElementById('user_password').value='password';");

js.executeScript("document.getElementById('user_submit').click();");

js.executeScript("alert('enter correct login credentials to continue');");
```

3. Logs : Using Log4j with Selenium

Log4j is a popular logging framework with extensive configuration options. To use Log4j, add it to your project dependencies and set up a `log4j.properties` file for configuration.

- **Add Log4j Dependency**:
  - **For Maven**:

    ```
    xml
    Copy code
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
    </dependency>
    ```

- **Using Log4j in Your Test**:

```
java
Copy code
import org.apache.log4j.Logger;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class SeleniumWithLog4j {
    private static final Logger logger =
Logger.getLogger(SeleniumWithLog4j.class);

    public static void main(String[] args) {
         BasicConfigurator.configure();


        System.setProperty("webdriver.chrome.driver",
"path/to/chromedriver");
        WebDriver driver = new ChromeDriver();

        logger.info("Browser opened");
        driver.get("https://example.com");
        logger.info("Navigated to example.com");

        // Log each action similarly to debug any issues
```

```
     driver.quit();
     logger.info("Browser closed");
 }
```

4. Frames

In Selenium, handling frames involves switching the WebDriver's focus to interact with elements inside `<iframe>` or `<frame>` tags. Frames act as a separate browsing context, so you must switch to them explicitly to access the elements they contain. Here's how to work with frames in Java Selenium:

1. Switching to a Frame by Index

Each frame on the page has an index, with the first frame being `0`. This method is helpful when frames don't have unique identifiers.

```
java
Copy code
// Switch to the first frame (index 0)
driver.switchTo().frame(0);
```

2. Switching to a Frame by Name or ID

If the frame has a unique `name` or `id` attribute, you can use it to switch to the frame directly. This is often a more stable option than using the index.

```
java
Copy code
// Switch to the frame with name or ID "frameName"
driver.switchTo().frame("frameName");
```

3. Switching to a Frame by WebElement

If neither index nor name/ID is available, locate the frame as a `WebElement` and switch to it.

```
java
Copy code
// Locate the frame and switch
WebElement frameElement =
driver.findElement(By.cssSelector("iframe.someClass"));
driver.switchTo().frame(frameElement);
```

4. Switching Back to the Main Document

After finishing actions within a frame, switch back to the main document to interact with elements outside the frame.

```
java
Copy code
// Switch back to the main document
driver.switchTo().defaultContent();
```

5. Switching Between Nested Frames

If frames are nested (a frame inside another frame), you'll need to switch step-by-step, first to the outer frame, then to the inner frame.

```
java
Copy code
// Switch to the outer frame
driver.switchTo().frame("outerFrame");

// Then switch to the inner frame within the outer frame
driver.switchTo().frame("innerFrame");

// Afterward, switch back to the main document
driver.switchTo().defaultContent();
```

Example Scenario

Let's say there's a frame with an ID of `outerFrame` containing another frame with the ID `innerFrame`. Inside `innerFrame`, there's a button you want to click:

```
java
Copy code
driver.switchTo().frame("outerFrame"); // Switch to outer frame
driver.switchTo().frame("innerFrame"); // Switch to inner frame within
outer frame
driver.findElement(By.id("buttonID")).click(); // Perform action inside
the frame
driver.switchTo().defaultContent(); // Return to the main page
```