Selenium Day 3 - Classroom Session – Trainer's Handbook

Topics : Actions, Alerts, Frames

Session Agenda

1. **Introduction to Advanced User Interactions in Selenium**
2. **Handling Mouse & Keyboard Actions (Actions Class)**
3. **Handling JavaScript Alerts & Popups**
4. **Working with Frames & Nested Frames**
5. **Hands-on Exercises (During the Session)**
6. **Best Practices & Interview Questions**
7. **Post-Classroom Assignment (2 Hours)**
8. **Expected Outcomes by End of Day 3**

Session details

1. Introduction to Advanced User Interactions in Selenium

- Selenium WebDriver provides various ways to handle advanced user interactions like:
    - **Mouse Hover, Drag & Drop, Right Click, Double Click** (using Actions class).
    - **Handling JavaScript Alerts & Popups** (using Alert class).
    - **Working with Frames & Nested Frames** (using SwitchTo()).

These actions are **crucial for automating real-world web applications**, making test scripts more robust and interactive.

---

2. Handling Mouse & Keyboard Actions (Actions Class)

What is the Actions Class?

- The **Actions** class in Selenium is used to handle complex user interactions such as:
    - Mouse Hover
    - Drag & Drop
    - Right Click (Context Click)
    - Double Click
    - Keyboard Interactions (KeyPress, KeyRelease)

Basic Syntax of Actions Class:

```
Actions action = new Actions(driver);
```

A. Mouse Hover

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

public class MouseHoverExample {
    public static void main(String[] args) {
        WebDriver driver = new ChromeDriver();
        driver.get("https://example.com");

        WebElement menu = driver.findElement(By.id("menu-item"));
        Actions action = new Actions(driver);
        action.moveToElement(menu).perform();

        driver.quit();
    }
}
```

B. Right Click (Context Click)

```
Actions action = new Actions(driver);
action.contextClick(element).perform();
```

C. Drag and Drop

```
Actions action = new Actions(driver);
action.dragAndDrop(sourceElement, targetElement).perform();
```

D. Keyboard Actions

```
import org.openqa.selenium.Keys;
action.sendKeys(Keys.ENTER).perform();
```

---

3. Handling JavaScript Alerts & Popups

Types of JavaScript Alerts in Selenium:

1. **Simple Alert** - Just shows an "OK" button.
2. **Confirmation Alert** - Provides "OK" and "Cancel" buttons.
3. **Prompt Alert** - Allows user input with a text box.

A. Handling Simple Alert

```
import org.openqa.selenium.Alert;

Alert alert = driver.switchTo().alert();
alert.accept();  // Clicks OK
```

B. Handling Confirmation Alert

```
Alert alert = driver.switchTo().alert();
alert.dismiss();  // Clicks Cancel
```

C. Handling Prompt Alert (With Text Input)

```
Alert alert = driver.switchTo().alert();
alert.sendKeys("Hello");
alert.accept();
```

---

4. Working with Frames & Nested Frames

What are Frames?

Frames are used to divide a web page into multiple sections, and Selenium needs to **switch between frames** to interact with elements inside them.

A. Switching to a Frame using Index

```
driver.switchTo().frame(0);
```

B. Switching to a Frame using Name or ID

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

C. Switching to a Frame using WebElement

```
WebElement frameElement = driver.findElement(By.tagName("iframe"));
driver.switchTo().frame(frameElement);
```

D. Switching Back to Default Page

```
driver.switchTo().defaultContent();
```

E. Handling Nested Frames

```
driver.switchTo().frame("outerFrame");
driver.switchTo().frame("innerFrame");
```

Hands-on Exercises (During the Session)

- ✅ **Exercise 1:** Automate mouse hover action and validate submenu visibility. (Solution)
- ✅ **Exercise 2:** Implement drag and drop functionality. (Solution)
- ✅ **Exercise 3:** Handle alert popups dynamically using Selenium. (Solution)
- ✅ **Exercise 4:** Switch between multiple frames and interact with elements. (Solution)

Best Practices & Interview Questions

Best Practices:

- ✔ Always use perform() to execute Actions class methods.
- ✔ Always switch to an alert before interacting with it.
- ✔ Use **explicit wait** when handling alerts or frames.
- ✔ For frames, switch back to the default content after operations.
- ✔ Use try-catch to handle NoSuchFrameException gracefully.

Interview Questions:

- ❓ How do you perform **mouse hover actions** in Selenium?
- ❓ What are **different types of alerts** in Selenium? How do you handle them?
- ❓ What happens if you try to interact with an element inside an **iframe** without switching?
- ❓ How do you switch back to the **main document** after working inside a frame?
- ❓ How do you handle **nested frames** in Selenium?

Post-Classroom Assignment (2 Hours)

Task 1: Write a Selenium script to handle a **multi-level dropdown menu** using Actions class. (Solution)
Task 2: Automate a **right-click (context menu) operation** on a web element. (Solution)
Task 3: Handle an **alert with a dynamic message** and validate the response. (Solution)
Task 4: Switch between **multiple nested frames** and extract text from the last frame. (Solution)

Expected Outcomes by End of Day 3

- ✔ Ability to handle **mouse and keyboard interactions** using Actions class.
- ✔ Understanding of **handling alerts & popups** dynamically.
- ✔ Expertise in **switching between frames & nested frames**.
- ✔ Hands-on practice with **real-world automation scenarios**.
- ✔ Confidence to answer **client interview questions** on these topics.

Solutions

Exercise 1: Automate mouse hover action and validate submenu visibility

To automate mouse hover actions and validate submenu visibility, you can use the `Actions` class in Selenium.

Here's an example:

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

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

        // Initialize WebDriver
        WebDriver driver = new ChromeDriver();

        // Open the webpage
        driver.get("https://example.com");

        // Locate the main menu element
        WebElement mainMenu = driver.findElement(By.id("mainMenu"));

        // Initialize Actions class
        Actions actions = new Actions(driver);

        // Perform mouse hover action
        actions.moveToElement(mainMenu).perform();

        // Locate the submenu element
        WebElement subMenu = driver.findElement(By.id("subMenu"));

        // Validate submenu visibility
        if (subMenu.isDisplayed()) {
            System.out.println("Submenu is visible.");
        } else {
            System.out.println("Submenu is not visible.");
        }

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

```
    }
}
```

Exercise 2: Implement drag and drop functionality

To implement drag and drop functionality, you can use the `Actions` class in Selenium.

Here's an example:

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

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

        // Initialize WebDriver
        WebDriver driver = new ChromeDriver();

        // Open the webpage
        driver.get("https://example.com");

        // Locate the source and target elements
        WebElement sourceElement = driver.findElement(By.id("source"));
        WebElement targetElement = driver.findElement(By.id("target"));

        // Initialize Actions class
        Actions actions = new Actions(driver);

        // Perform drag and drop action
        actions.dragAndDrop(sourceElement, targetElement).perform();

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

Exercise 3: Handle alert popups dynamically using Selenium

To handle alert popups dynamically, you can use the `Alert` interface in Selenium.

Here's an example:

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

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

        // Initialize WebDriver
        WebDriver driver = new ChromeDriver();

        // Open the webpage
        driver.get("https://example.com");

        // Trigger the alert popup
        WebElement alertButton = driver.findElement(By.id("alertButton"));
        alertButton.click();

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

        // Handle the alert
        System.out.println("Alert text: " + alert.getText());
        alert.accept(); // or alert.dismiss();

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

Exercise 4: Switch between multiple frames and interact with elements

To switch between multiple frames and interact with elements, you can use the `switchTo().frame()` method in Selenium.

Here's an example:

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

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

        // Initialize WebDriver
        WebDriver driver = new ChromeDriver();

        // Open the webpage
        driver.get("https://example.com");

        // Switch to the first frame
        driver.switchTo().frame("frame1");

        // Interact with an element in the first frame
```

```
        WebElement elementInFrame1 =
driver.findElement(By.id("elementInFrame1"));
        elementInFrame1.click();

        // Switch back to the main content
        driver.switchTo().defaultContent();

        // Switch to the second frame
        driver.switchTo().frame("frame2");

        // Interact with an element in the second frame
        WebElement elementInFrame2 =
driver.findElement(By.id("elementInFrame2"));
        elementInFrame2.click();

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

Task 1: Write a Selenium script to handle a multi-level dropdown menu using Actions class

To handle a multi-level dropdown menu, you can use the `Actions` class to perform mouse hover actions.

Here's an example:

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

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

        // Initialize WebDriver
        WebDriver driver = new ChromeDriver();

        // Open the webpage
        driver.get("https://example.com");

        // Initialize Actions class
        Actions actions = new Actions(driver);

        // Locate the main menu element
        WebElement mainMenu = driver.findElement(By.id("mainMenu"));

        // Perform mouse hover action on the main menu
        actions.moveToElement(mainMenu).perform();

        // Locate the submenu element
```

```
        WebElement subMenu = driver.findElement(By.id("subMenu"));

        // Perform mouse hover action on the submenu
        actions.moveToElement(subMenu).perform();

        // Locate the sub-submenu element and click it
        WebElement subSubMenu = driver.findElement(By.id("subSubMenu"));
        subSubMenu.click();

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

Task 2: Automate a right-click (context menu) operation on a web element

To automate a right-click operation, you can use the `Actions` class to perform a context click.

Here's an example:

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

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

        // Initialize WebDriver
        WebDriver driver = new ChromeDriver();

        // Open the webpage
        driver.get("https://example.com");

        // Locate the element to right-click
        WebElement elementToRightClick =
driver.findElement(By.id("rightClickElement"));

        // Initialize Actions class
        Actions actions = new Actions(driver);

        // Perform right-click (context click) action
        actions.contextClick(elementToRightClick).perform();

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

Task 3: Handle an alert with a dynamic message and validate the response

To handle an alert with a dynamic message, you can use the `Alert` interface and validate the alert text.

Here's an example:

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

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

        // Initialize WebDriver
        WebDriver driver = new ChromeDriver();

        // Open the webpage
        driver.get("https://example.com");

        // Trigger the alert popup
        WebElement alertButton = driver.findElement(By.id("alertButton"));
        alertButton.click();

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

        // Validate the alert message
        String alertMessage = alert.getText();
        System.out.println("Alert message: " + alertMessage);

        // Accept the alert
        alert.accept();

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

Task 4: Switch between multiple nested frames and extract text from the last frame

To switch between multiple nested frames and extract text from the last frame, you can use the `switchTo().frame()` method.

Here's an example:

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

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

        // Initialize WebDriver
        WebDriver driver = new ChromeDriver();

        // Open the webpage
        driver.get("https://example.com");

        // Switch to the first frame
        driver.switchTo().frame("frame1");

        // Switch to the second nested frame
        driver.switchTo().frame("frame2");

        // Switch to the third nested frame
        driver.switchTo().frame("frame3");

        // Extract text from an element in the last frame
        WebElement elementInLastFrame =
driver.findElement(By.id("elementInLastFrame"));
        String extractedText = elementInLastFrame.getText();
        System.out.println("Extracted text: " + extractedText);

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