Selenium Day 1 - Classroom Session – Trainer's handbook

Selenium Basics (WebDriver, Locators, Handling WebElements)

Session Agenda

1. **Introduction to Selenium** – What is Selenium? Why Selenium? Key components.
2. **Selenium WebDriver** – Architecture, setup, and first automation script.
3. **Locators in Selenium** – Understanding different locators (ID, Name, XPath, CSS Selector, etc.).
4. **Handling WebElements** – Clicking buttons, entering text, selecting dropdowns, handling checkboxes & radio buttons.
5. **Practical Hands-on Exercises** – Real-world scenarios for WebDriver and WebElements.
6. **Best Practices & Common Interview Questions** – What to focus on for client interviews.
7. **Post-Classroom Assignment** – Reinforcing learning through practical tasks.

Introduction to Selenium

Selenium is an open-source automation framework used to automate web applications across multiple browsers.

1. Why Selenium?

✓ **Cross-Browser Testing** – Supports Chrome, Firefox, Edge, Safari, etc.
✓ **Supports Multiple Languages** – Java, Python, C#, etc.
✓ **Open-Source & Free** – No licensing costs.
✓ **Rich Community Support** – Large developer base with continuous updates.
✓ **Integrates with TestNG, JUnit, and CI/CD tools** like Jenkins.

Key Components of Selenium

1. **Selenium WebDriver** – The core of Selenium, which allows interaction with web applications.
2. **Selenium Grid** – Enables execution on multiple machines & browsers.
3. **Selenium IDE** – A record & playback tool (not used in enterprise testing).

---

2. Selenium WebDriver - Architecture & First Automation Script

Selenium WebDriver interacts directly with the browser without requiring a separate server like Selenium RC. It follows this architecture:

1. **Test Script (Java/Python etc.)** → 2. **Selenium WebDriver API** → 3. **Browser-Specific Driver** → 4. **Web Browser**

Setting Up Selenium WebDriver (Java)

1. **Install Java & Eclipse/IntelliJ**
2. **Add Selenium JARs & WebDriver Dependencies** (Maven)

```
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>4.6.0</version>
</dependency>
```

3. **Download Browser Drivers** (ChromeDriver, GeckoDriver, etc.)

First Automation Script - Open Browser & Navigate

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

public class FirstSeleniumScript {
    public static void main(String[] args) {
        System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
        WebDriver driver = new ChromeDriver();
        driver.get("https://www.google.com");
        System.out.println("Title of the page: " + driver.getTitle());
        driver.quit();
    }
}
```

---

3. Locators in Selenium

Locators help identify web elements on a page. The key Selenium locators are:

Locator Type	Usage
id	driver.findElement(By.id("username"))
name	driver.findElement(By.name("email"))
className	driver.findElement(By.className("login-btn"))
tagName	driver.findElement(By.tagName("input"))
linkText	driver.findElement(By.linkText("Sign Up"))
partialLinkText	driver.findElement(By.partialLinkText("Sign"))
CSS Selector	driver.findElement(By.cssSelector("#login-form input"))
XPath	driver.findElement(By.xpath("//input[@name='password']"))

Example - Using Different Locators

```java
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) {
        WebDriver driver = new ChromeDriver();
        driver.get("https://example.com");

        WebElement emailField = driver.findElement(By.id("email"));
        WebElement submitButton = driver.findElement(By.cssSelector("button.submit"));

        emailField.sendKeys("test@example.com");
        submitButton.click();

        driver.quit();
    }
}
```

---

4. Handling WebElements (Click, Type, Select, Checkboxes, Radio Buttons)

Once locators are identified, we perform actions like click, enter text, and retrieve values.

Clicking a Button

```java
driver.findElement(By.id("submit")).click();
```

Typing in a Text Field

```
driver.findElement(By.name("username")).sendKeys("testUser");
```

Handling Dropdowns (Select Class)

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

Select dropdown = new Select(driver.findElement(By.id("country")));
dropdown.selectByVisibleText("India");
```

Handling Checkboxes & Radio Buttons

```
driver.findElement(By.cssSelector("input[type='checkbox']")).click();
```

Handling Alerts & Popups

```
driver.switchTo().alert().accept(); // Accept alert
```

Hands-on Exercises (During the Session)

✅ **Exercise 1:** Open https://www.google.com, search for a keyword, and print the results. (Solution)
✅ **Exercise 2:** Automate login for a test site using different locators. (Solution)
✅ **Exercise 3:** Handle a dropdown and verify the selected option. (Solution)
✅ **Exercise 4:** Click on a checkbox and validate if it's selected. (Solution)

Best Practices & Interview Questions

Best Practices:

✔ Always use explicit waits instead of Thread.sleep().
✔ Prefer id and name locators over XPath for better performance.
✔ Use **Page Object Model (POM)** for maintainability.
✔ Use quit() instead of close() to ensure proper browser cleanup.

Interview Questions:

❓ What is the difference between findElement() and findElements()?
❓ What are the advantages of using CSS Selector over XPath?
❓ How do you handle dynamic elements in Selenium?
❓ What is the difference between driver.close() and driver.quit()?
❓ How do you handle alerts and popups in Selenium?

Post-Classroom Assignment (2 Hours)

Task 1: Automate the login functionality of any e-commerce website. (Solution)
Task 2: Automate the dropdown selection process on a travel booking site. (Solution)
Task 3: Write a script to validate the presence of all links on a webpage. (Solution)
Task 4: Capture a screenshot of the webpage after performing an action. (Solution)

Expected Outcomes by End of Day 1

✓ Strong understanding of Selenium WebDriver & its architecture.
✓ Hands-on experience with different **locators** in Selenium.
✓ Ability to interact with **WebElements** dynamically.
✓ Clear approach to solving **real-world automation problems**.
✓ Confidence to answer **basic Selenium interview questions**.

Solutions –

✅ **Exercise 1: Open https://www.google.com, search for a keyword, and print the results.**

Here's a Selenium script to open Google, search for a keyword, and print the results:

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

public class GoogleSearch {
    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 Google
        driver.get("https://www.google.com");

        // Search for a keyword
        WebElement searchBox = driver.findElement(By.name("q"));
        searchBox.sendKeys("Selenium WebDriver");
        searchBox.submit();

        // Wait for the results to load and display
        try {
            Thread.sleep(2000); // Adjust the sleep time as needed
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // Print the results
        List<WebElement> results =
driver.findElements(By.cssSelector("h3"));
        for (WebElement result : results) {
            System.out.println(result.getText());
        }

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

✅ **Exercise 2: Automate login for a test site using different locators.**

Here's a Selenium script to automate login using different locators:

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

public class LoginAutomation {
    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 test site
        driver.get("https://example.com/login");

        // Locate elements and perform login
        WebElement usernameField = driver.findElement(By.id("username"));
        WebElement passwordField = driver.findElement(By.name("password"));
        WebElement loginButton = driver.findElement(By.className("login-
button"));

        usernameField.sendKeys("testuser");
        passwordField.sendKeys("password");
        loginButton.click();

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

✅ **Exercise 3: Handle a dropdown and verify the selected option.**

Here's a Selenium script to handle a dropdown and verify the selected option:

```
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.support.ui.Select;

public class DropdownHandling {
    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 test site
        driver.get("https://example.com/dropdown");

        // Locate the dropdown element
        WebElement dropdownElement = driver.findElement(By.id("dropdown"));

        // Create a Select object
        Select dropdown = new Select(dropdownElement);

        // Select an option by visible text
        dropdown.selectByVisibleText("Option 1");

        // Verify the selected option
        WebElement selectedOption = dropdown.getFirstSelectedOption();
        System.out.println("Selected option: " + selectedOption.getText());

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

✅ Exercise 4: Click on a checkbox and validate if it's selected.

Here's a Selenium script to click on a checkbox and validate if it's selected:

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

public class CheckboxValidation {
    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 test site
        driver.get("https://example.com/checkbox");

        // Locate the checkbox element
        WebElement checkbox = driver.findElement(By.id("checkbox"));
        // Click the checkbox
        checkbox.click();

        // Validate if the checkbox is selected
        if (checkbox.isSelected()) {
            System.out.println("Checkbox is selected.");
        } else {
            System.out.println("Checkbox is not selected.");
        }

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

Task 1: Automate the login functionality of any e-commerce website.

Here's an example of how you can automate the login functionality of an e-commerce website using Selenium WebDriver. For this example, I'll use a hypothetical e-commerce website with the following login page elements:

- Username field with `id="username"`
- Password field with `id="password"`
- Login button with `id="loginButton"`

Here's the code:

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

public class ECommerceLoginAutomation {
    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 e-commerce website login page
        driver.get("https://example-ecommerce.com/login");

        // Locate the username field and enter the username
        WebElement usernameField = driver.findElement(By.id("username"));
        usernameField.sendKeys("testuser");

        // Locate the password field and enter the password
        WebElement passwordField = driver.findElement(By.id("password"));
        passwordField.sendKeys("password");

        // Locate the login button and click it
        WebElement loginButton = driver.findElement(By.id("loginButton"));
        loginButton.click();
        // Add a wait to ensure the login process completes (optional)
        try {
            Thread.sleep(2000); // Adjust the sleep time as needed
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // Verify login by checking the presence of a logout button or user
profile element
        WebElement logoutButton =
driver.findElement(By.id("logoutButton"));
        if (logoutButton.isDisplayed()) {
            System.out.println("Login successful!");
        } else {
            System.out.println("Login failed.");
        }

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

Steps to Follow:

1. **Set up Selenium WebDriver**: Ensure you have the Selenium WebDriver and the ChromeDriver executable set up in your project.
2. **Open the e-commerce website**: Navigate to the login page of the e-commerce website.
3. **Locate the login elements**: Use the appropriate locators (e.g., `By.id`, `By.name`, `By.className`) to find the username, password fields, and the login button.
4. **Perform the login action**: Enter the username and password, then click the login button.
5. **Verify the login**: Check for the presence of a logout button or user profile element to confirm the login was successful.

Task 2: Automate the dropdown selection process on a travel booking site.

Here's an example of how you can automate the dropdown selection process on a travel booking site using Selenium WebDriver. For this example, I'll use a hypothetical travel booking site with the following dropdown elements:

- Dropdown for selecting the departure city with `id="departureCity"`
- Dropdown for selecting the destination city with `id="destinationCity"`

Here's the code:

```
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.support.ui.Select;
```

```
public class TravelBookingDropdownAutomation {
    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 travel booking site
        driver.get("https://example-travelbooking.com");

        // Locate the departure city dropdown element
        WebElement departureCityDropdown =
driver.findElement(By.id("departureCity"));

        // Create a Select object for the departure city dropdown
        Select departureCitySelect = new Select(departureCityDropdown);

        // Select an option by visible text
        departureCitySelect.selectByVisibleText("New York");

        // Locate the destination city dropdown element
        WebElement destinationCityDropdown =
driver.findElement(By.id("destinationCity"));

        // Create a Select object for the destination city dropdown
        Select destinationCitySelect = new Select(destinationCityDropdown);

        // Select an option by visible text
        destinationCitySelect.selectByVisibleText("Los Angeles");

        // Verify the selected options
        WebElement selectedDepartureCity =
departureCitySelect.getFirstSelectedOption();
        WebElement selectedDestinationCity =
destinationCitySelect.getFirstSelectedOption();

        System.out.println("Selected departure city: " +
selectedDepartureCity.getText());
        System.out.println("Selected destination city: " +
selectedDestinationCity.getText());

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

Steps to Follow:

1. **Set up Selenium WebDriver**: Ensure you have the Selenium WebDriver and the ChromeDriver executable set up in your project.
2. **Open the travel booking site**: Navigate to the travel booking site.
3. **Locate the dropdown elements**: Use the appropriate locators (e.g., `By.id`, `By.name`, `By.className`) to find the dropdown elements for departure and destination cities.
4. **Create `Select` objects**: Use the `Select` class to interact with the dropdown elements.

5. **Select options**: Use methods like `selectByVisibleText`, `selectByValue`, or `selectByIndex` to select options from the dropdowns.
6. **Verify the selected options**: Retrieve and print the selected options to verify the selections.

Task 3: Write a script to validate the presence of all links on a webpage.

Here's a Selenium script to validate the presence of all links on a webpage. This script will find all the links on the page and print their text and URLs to verify their presence.

Steps to Follow:

1. **Set up Selenium WebDriver**: Ensure you have the Selenium WebDriver and the ChromeDriver executable set up in your project.
2. **Open the webpage**: Navigate to the webpage you want to validate.
3. **Find all links**: Use the appropriate locators to find all the links on the page.
4. **Print the link text and URLs**: Iterate through the links and print their text and URLs to verify their presence.

Here's the code:

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

public class LinkValidation {
    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");

        // Find all links on the page
        List<WebElement> links = driver.findElements(By.tagName("a"));

        // Validate the presence of each link
        for (WebElement link : links) {
            if (link.isDisplayed()) {
                System.out.println("Link text: " + link.getText() + " |
URL: " + link.getAttribute("href"));
            } else {
                System.out.println("Link text: " + link.getText() + " is
not displayed.");
            }
        }

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

Explanation:

- **Set up Selenium WebDriver**: Ensure you have the Selenium WebDriver and the ChromeDriver executable set up in your project.
- **Open the webpage**: Navigate to the webpage you want to validate.
- **Find all links**: Use `driver.findElements(By.tagName("a"))` to find all the links on the page.
- **Print the link text and URLs**: Iterate through the links and print their text and URLs to verify their presence.

Task 4: Capture a screenshot of the webpage after performing an action.

Here's how you can capture a screenshot of a webpage after performing an action using Selenium WebDriver. For this example, I'll demonstrate how to click a button on a webpage and then capture a screenshot.

Steps to Follow:

1. **Set up Selenium WebDriver**: Ensure you have the Selenium WebDriver and the ChromeDriver executable set up in your project.
2. **Open the webpage**: Navigate to the webpage you want to interact with.
3. **Perform an action**: Click a button or perform any other action on the webpage.
4. **Capture a screenshot**: Use the `TakesScreenshot` interface to capture the screenshot and save it to a file.

Here's the code:

```
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.OutputType;
import org.openqa.selenium.TakesScreenshot;
import java.io.File;
import java.io.IOException;
import org.apache.commons.io.FileUtils;

public class CaptureScreenshot {
    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");

        // Perform an action (e.g., click a button)
        WebElement button = driver.findElement(By.id("button"));
        button.click();

        // Capture a screenshot
        File screenshot = ((TakesScreenshot)
driver).getScreenshotAs(OutputType.FILE);
        try {
            FileUtils.copyFile(screenshot, new File("screenshot.png"));
            System.out.println("Screenshot captured.");
        } catch (IOException e) {
            e.printStackTrace();
        }

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

Explanation:

- **Set up Selenium WebDriver**: Ensure you have the Selenium WebDriver and the ChromeDriver executable set up in your project.
- **Open the webpage**: Navigate to the webpage you want to interact with.
- **Perform an action**: Locate the button element using `By.id("button")` and click it.
- **Capture a screenshot**: Use the `TakesScreenshot` interface to capture the screenshot and save it to a file using `FileUtils.copyFile`.