HANDLING WEB ELEMENTS

Auto Suggestions:

Auto suggestions or dynamic drop downs are the web elements which try to predict the values of typed text based on the input value provided.

Auto suggestions can be handled by collecting all the suggestions and traversing through this list of the suggestions and performing necessary action.

We use **findElements()** method of WebDriver interface to store the list of auto suggestions.

Example Script:

```
public class Practice1 {
	public static void main(String[] args) {
		WebDriver driver = new ChromeDriver();
		driver.manage().window().maximize();
		driver.get("https://www.google.com/");
		driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));

		driver.findElement(By.name("q")).sendKeys("selenium");
List<WebElement>  elements =  driver.findElements(By.xpath("//li[contains(.,'selenium')]"));
		for (WebElement e : elements) {
			System.out.println(e.getText());
		}
		driver.quit();
	}
}
```

Actions class:

The Action class in Selenium WebDriver is a utility class that enables advanced user interactions such as mouse and keyboard events.

By using the Actions class, you can interact with a website in the same way a user would, allowing you to test more complex interactions that can't be accomplished with just simple method calls.

Actions class methods are broadly divided into two categories:

1) Mouse actions

2) Keyboard actions

1) Mouse actions:

- **click():** performs a single mouse click on the specified element.
- **clickAndHold():** holds down the left mouse button on the specified element.
- **contextClick():** performs a right-click on the specified element.
- **doubleClick():** performs a double-click on the specified element.
- **dragAndDrop():** performs a drag and drop operation between two elements.
- **release():** releases the left mouse button on the specified element.
- **moveToElement():** moves the mouse cursor to the middle of the specified element.

2) Keyboard actions:

- **sendKeys(CharSequence… keysToSend):** sends a series of key presses to the specified element.
- **keyDown(Keys theKey):** holds down the specified key.
- **keyUp(Keys theKey):** releases the specified key.

Implementation of Action class methods:

Step 1: Create an instance of Actions class

Actions actions = new Actions(driver);

Step 2: Use the methods of Actions class to perform actions on the web page

actions.moveToElement(element).perform();

actions.doubleClick(element).perform();

actions.contextClick(element).perform();

actions.dragAndDrop(element, target).perform(); etc…

Note: perform() method is mandatory method which actually performs specified actions on the web page.

build() method generates a composite action containing all actions so far, ready to be performed.

Drop downs:

Dropdowns are the web elements which when clicked on provides list of options on the website.

Dropdowns can be **static** and **dynamic**.

- **Static** dropdowns have fixed options. The static dropdowns can be **select** and **non-select** dropdowns.
  - **Select** dropdowns have <select> tag in the element node. These dropdowns are of two types:
    1. **Single select dropdown:**
       Select dropdowns which allows user to select only one option at a time are called Single select dropdowns. Deselection is not possible in single select dropdown.
    2. **Multi select dropdown:**
       Select dropdowns which allows user to select more than one option at a time are called Multi select dropdowns. Deselection is possible in multi select dropdown.

  - **Non-select** dropdowns do not have <select> tag in the element node.

- **Dynamic** dropdowns have options which are not fixed.

Any kind of dropdown can be handled using *findElements()* method by storing all the options in a list and iterating through the list.

Select Dropdowns can be handled using *Select* class.

Select class:

Select class is present in org.openqa.selenium.support.ui package. It is used to select or deselect an option in a dropdown. It has several non-static methods to handle select dropdowns.

Implementation of Select class methods:

I. **WebElement dropdown = driver.findElement(<locator>);**
II. **Select select = new Select(dropdown);**
III. **Different operations that can be performed are:**
   - **To select an option from dropdown:**
     - **select.selectByIndex(int index);**
     - **select.selectByValue(String value);**
     - **select.selectByVisibleText(String text);**
       (Return type of the above three methods is void)

- **To verify if the dropdown is multi select**
  - **select.isMultiple(); (return type is Boolean)**
- **To get all the options from the dropdown**
  - **List<WebElement> options = select.getOptions();**
- **To get first selected option**
  - **WebElement firstOption = select.getFirstSelectedOption();**
- **To get all selected options**
  - **List<WebElement> selectedOptions = select.getAllSelectedOptions();**
- **To deselect an option from the dropdown**
  - **select.deselectByIndex(int index);**
  - **select.deselectByValue(String value);**
  - **select.deselectByVisibleText(String text);**
    (Return type of the above three methods is void)
- **To deselect all the options at once**
  - **select.deselectAll(); (return type is void)**

iFrames:

iFrame is a HTML document embedded inside an HTML document. iFrame is defined by an *<iframe></iframe>* tag in HTML.

Identifying frames on the web page:

- ✓ When you right click on a frame we can view an option having the text – "view frame source"
- ✓ When you inspect a frame you can see the following HTML code:

```
<iframe id="" name="" . . . >
    #document
    <html>
        <head>…</head>
        <body>…</body>
    </html>
</iframe>
```

Handling the frames on the web page:

- ✓ To handle the frames the control should be shifted to the frame from main web page.
- ✓ To switch to the frame using index:
  driver.switchTo().frame(int index);
- ✓ To switch to the frame using name or id attribute value of the iframe tag
  driver.switchTo().frame(String idOrName);
- ✓ To switch to the frame using frame element reference
  driver.switchTo().frame(WebElement frameElement);

- ✓ To switch the control back from frame
  driver.switchTo().defaultContent();

switchTo():

- ✓ It is one of the WebDriver interfaces' abstract methods.
- ✓ It is used to switch the control to the target location like frames or windows or pop ups.
- ✓ Return type is TargetLocator (static interface in WebDriver).
- ✓ TargetLocator interface is used to select a frame or window.

frame():

- ✓ It is an abstract method in TargetLocator interface.
- ✓ It selects a frame based on index or id or name or frame element reference.
- ✓ It is an overloaded method. It accepts one of the three arguments:
  - frame(int index)
  - frame(String idOrName)
  - frame(WebElement frameElement)
- ✓ Return type is WebDriver.
- ✓ It throws NoSuchFrameException if frame is not found.

defaultContent():

- ✓ It is an abstract method in TargetLocator interface.
- ✓ It selects either a first frame or the main document on the web page.
- ✓ Return type is WebDriver.

Screenshot:

- ✓ Screenshot of a web page can be captured with the help of TakesScreenshot interface and apache commons io libraries.
- ✓ Apache commons io libraries contains utility classes which are used to copy or share files or directories.

Steps to capture screenshot:

I. Typecast WebDriver reference to TakesScreenshot interface reference
TakesScreenshot ts = (TakesScreenshot) driver;

II. Call getScreenshotAs() method to create image file
File temp = ts.getScreenshotAs(OutputType.FILE);

III. Create a permanent location to store the captured image
File target = new File("<permanent_file_path>");

IV. Copy the image from temporary memory to the permanent file
FileUtils.copyFile(temp, target);
FileUtils class is present in org.apache.commons.io package. copyFile() is a static method of FileUtils class. The above statement throws IOException.

JavascriptExecutor (I):

- ✓ It is an interface in Selenium WebDriver which executes Javascript code on selenium.
- ✓ It has two abstract methods –
  - **executeScript():** Executes JavaScript in the context of the currently selected frame or window. Return type is Object class.
  - **executeAsyncScript():** Executes JavaScript in the context of the currently selected frame or window. Return type is Object class.

Implementation of JavascriptExecutor:

I. **JavascriptExecutor js = (JavascriptExecutor) driver;**
II. **js.executeScript("<Javascript_code>", args);**

Actions that can be performed using JavascriptExecutor:

- Handles scroll bar
  - **js.executeScript("window.scrollTo(<x>, <y>)");**
    The scrollTo() method scrolls the document to specified coordinates.
  - **js.executeScript("window.scrollBy(<x>, <y>)");**
    The scrollBy() method scrolls the document by the specified number of pixels.
  - **js.executeScript("arguments[0].scrollIntoView(true)", element);**
    The scrollIntoView() method scrolls an element into the visible area of the browser window.

- Navigates to an application
  js.executeScript("window.location=arguments[0]", <url>);

- Fetches title and URL of a web page
  - To fetch title
    js.executeScript("return document.title");
  - To fetch URL
    js.executeScript("return document.URL");

- Refresh a web page
  js.executeScript("history.go(0)");

- Performing type action
  js.executeScript(“arguments[0].value=arguments[1]”, element, “<data>”);

- Performing click action
  js.executeScript(“arguments[0].click()”, element);

- Handles disabled elements
  js.executeScript("arguments[0].removeAttribute('disabled','disabled')", element);

Pop ups:

Popup is a window that displays or pops up on the screen due to some activity.

There are different kinds of pop ups.

1. Javascript alerts, prompts and confirmations pop up:

- **Javascript alerts** shows a custom message and a single button which dismisses the alert, labelled as OK. It can also be dismissed in most browsers by pressing the close button.

- **Javascript confirmation pop up** is similar to an alert, except the user can also choose to cancel the message.
- **Javascript prompt pop** up is similar to confirm pop up, except they also include a text input.
- Javascript pop ups are neither inspectable nor movable.
- These pop ups can be handled using **Alert interface**.
- Alert interface is present in org.openqa.selenium package.

Steps to handle Javascript pop up:

I. **Alert a = driver.switchTo().alert();**

II. To click OK → **a.accept();**
To click on Cancel → **a.dismiss();**
To fetch the text from pop up → **a.getText(); (returns String)**
To pass text to pop up → **a.sendKeys(“<text>”);**

2. Hidden Division or Calendar pop up:

- These pop ups are inspectable but not movable.
- Since these pop ups are inspectable they can be handled using findElement() method.

3. Authentication pop up:

- Authentication popups often appear when accessing secure areas of a web application, requiring valid credentials to proceed.
- These popups can disrupt the flow of automated tests if not properly managed, leading to incomplete or failed test executions.
- These pop ups are neither inspectable nor movable.
- The simplest way to handle this pop up is by directly passing username and password in the URL separated by colon.
- Syntax:
  https://<username>:<password>@<domain>

  Ex: https://admin:admin@the-internet.herokuapp.com/basic_auth

4. Window or Child Browser pop up:

- Browser window popup is a window popup that suddenly pops up on after clicking on some link or on selecting an option.
- It just likes another web page.
- This is having its own title and url.
- These pop ups are inspectable and movable.
- These pop ups can be handled using **getWindowHandle()** and **getWindowHandles()** methods.
- **getWindowHandle():**
  - ✓ It will get the handle of the page the webDriver is currently controlling.
  - ✓ This handle is a unique identifier for the web page.
  - ✓ This is different every time you open a page even if it is the same URL.
  - ✓ Return type is String.
- **getWindowHandles():**
  - ✓ It will get all the handles for all the pages that the web driver understands are open.
  - ✓ They are listed in the order that they have been opened.
  - ✓ Return type is Set<String>
- **Handling Child Windows in Selenium:**
  - ✓ Capture the handles of both the parent and child windows.
  - ✓ Use getWindowHandles() to retrieve all handles.
  - ✓ Iterate through the handles to switch to the desired window.

5. File Upload pop up:

- File upload dialog appears when you wanted to upload a file by clicking on upload button.
- This pop up is a desktop application. Hence it is not inspectable but movable.
- These pop ups can be handled using:
  I. sendKeys():
  It is used to upload the file even without opening file dialog provided the element is an **input** element with **type = ”file”** attribute.

II. **Robot class:** The Robot class in Selenium is a Java-based utility class that provides a way to simulate keyboard and mouse events on the screen.

The Robot class in Selenium works by creating a virtual robot on the screen that can perform keyboard and mouse actions. The robot can simulate key presses, mouse clicks, mouse movement, and other user interactions.

Steps to upload a file using Robot class:

1. Select the file path
   StringSelection str = new StringSelection("<file_path>");
2. Set the file path to system clipboard
   Toolkit.getDefaultToolkit().getSystemClipboard()
   .setContents(str, null);
3. Create Robot instance
   Robot r = new Robot();
4. Paste the path to the file dialog box
   r.keyPress(KeyEvent.VK_CONTROL);
   r.keyPress(KeyEvent.VK_V);
   r.keyRelease(KeyEvent.VK_CONTROL);
   r.keyRelease(KeyEvent.VK_V);
5. Press ENTER button

   r.keyPress(KeyEvent.VK_ENTER);
   r.keyRelease(KeyEvent.VK_ ENTER);

For macOS:

1. Select the file path
   StringSelection str = new StringSelection("<file_path>");
2. Set the file path to system clipboard
   Toolkit.getDefaultToolkit().getSystemClipboard()
   .setContents(str, null);
3. Create Robot instance
   Robot r = new Robot();
4. Cmd + Tab is needed since it launches a Java app and the browser looses focus

   robot.keyPress(KeyEvent.VK_META);

   robot.keyPress(KeyEvent.VK_TAB);

   robot.keyRelease(KeyEvent.VK_META);

   robot.keyRelease(KeyEvent.VK_TAB);
5. Open Goto window CMD+SHIFT+G

robot.keyPress(KeyEvent.VK_META);

robot.keyPress(KeyEvent.VK_SHIFT);

robot.keyPress(KeyEvent.VK_G);

robot.keyRelease(KeyEvent.VK_META);

robot.keyRelease(KeyEvent.VK_SHIFT);

robot.keyRelease(KeyEvent.VK_G);

6. Paste the clipboard value CMD+V

robot.keyPress(KeyEvent.VK_META);

robot.keyPress(KeyEvent.VK_V);

robot.keyRelease(KeyEvent.VK_META);

robot.keyRelease(KeyEvent.VK_V);

7. Press Enter key to close the Goto window and Upload window

robot.keyPress(KeyEvent.VK_ENTER);

robot.keyRelease(KeyEvent.VK_ENTER);

robot.delay(500);

robot.keyPress(KeyEvent.VK_ENTER);

robot.keyRelease(KeyEvent.VK_ENTER);

6. Notifications pop up:

- These pop ups are browser specific pop ups.
- They are neither inspectable nor movable.
- These pop ups are handled using browser specific classes like
  ChromeOptions for chrome
  FirefoxOptions for firefox
  EdgeOptions for edge
- Steps to handle these pop ups:
  1. Create an instance for ChromeOptions class
     ChromeOptions options = new ChromeOptions();
  2. Disable notifications or geolocation
     options.addArguments("—disable-notifications");
     options.addArguments("—disable-geolocation");
  3. Configure options parameter to ChromeDriver
     WebDriver driver = new ChromeDriver(options);