LOCATING WEBELEMENTS USING WEBDRIVER

The web elements can be located by following method,

- findElement() method,
- By.name() method,
- WebElement interface.

The findElement() and By() methods instruct WebDriver to locate a WebElement on a web page, and once found, the findElement() method returns the WebElement instance of that element.

i. The findElement method

In UI automation, locating an element is the first step before executing any user actions on it. WebDriver's findElement() method is a convenient way to locate an element on the web page. The method declaration is as follows:

WebElement findElement(By by)

So, the input parameter for the findElement() method is the By instance. The By instance is a WebElement-locating mechanism. There are eight different ways to locate a WebElement on a web page.

The return type of the findElement() method is the WebElement instance that represents the actual HTML element or component of the web page. The method returns the first WebElement that the driver comes across that satisfies the locating-mechanism condition. This WebElement instance will act as a handle to that component from then on. Appropriate actions can be taken on that component by the test-script developer using this returned WebElement instance. If WebDriver doesn't find the element, it throws a runtime exception named NoSuchElementException, which the invoking class or method should handle.

The findElements method

For finding multiple elements matching the same locator criteria on a web page, the findElements() method can be used. It returns a list of WebElements found for a given locating mechanism. The method declaration of the findElements() method is as follows:

java.util.List findElements(By by)

The input parameter is the same as the findElement() method, which is an instance of the By class. The difference lies in the return type. Here, if no element is found, an empty list is returned and if there are multiple WebElements present that satisfy the locating mechanism, all of them are returned to the caller in a list.

ii. Using the By locating mechanism

By is the locating mechanism passed to the findElement() method or the findElements() method to

fetch the respective WebElement(s) on a web page. There are eight different locating mechanisms; that is, eight different ways to identify an HTML element on a web page. They are located by,

- ID,
- Name,
- ClassName,
- TagName,
- CSS Selector

The By.id() method

On a web page, each element is uniquely identified by an ID attribute, which is optionally provided. An ID can be assigned manually by the developer of the web application or left to be dynamically generated by the application. Dynamically-generated IDs can be changed on every page refresh or over a period of time. Now, consider the HTML code of the Search box:

<**input** id="search" **type="search" name="q" value="" class="input-text required-entry" maxlength="128" placeholder="Search entire store here..." autocomplete="off"**>

In the preceding code, the id attribute value of the search box is search.

Let's see how to use the ID attribute as a locating mechanism to find the Search box:

```
@Test
public void byIdLocatorExample() {
   WebElement searchBox = driver.findElement(By.id("q"));
searchBox.sendKeys("Bags");
   searchBox.submit();
   assertThat(driver.getTitle())
         .isEqualTo("Search results for: 'Bags'");
}
```

In preceding code, we used the By.id() method and the search box's id attribute value to find the element.

The By.name() method

As seen earlier, every element on a web page has many attributes. Name is one of them. For instance, the HTML code for the Search box is:

<input id="search" type="search" name="q" value="" class="input-text required-entry" maxlength="128" placeholder="Search entire store here..." autocomplete="off">

Here, name is one of the many attributes of the search box, and its value is q. If we want to identify this search box and set a value in it in your test script, the code will look as follows:

```
@Test
public void searchProduct() {
   // find search box and enter search string
```

```
    WebElement searchBox = driver.findElement(By.name("q"));
    searchBox.sendKeys("Phones");
    searchBox.submit();
    assertThat(driver.getTitle())
          .isEqualTo("Search results for: 'Phones'");
}
```

If you observe line four, the locating mechanism used here is By.name and the name is q.

The By.className() method

So, in order to apply styles to an element, they can be declared directly in the element tag, or placed in a separate file called the CSS file and can be referenced in the element using the class attribute. For instance, a style attribute for a button can be declared in a CSS file as follows:

```
.buttonStyle{
    width: 50px;
    height: 50px;
    border-radius: 50%;
    margin: 0% 2%;
}
```

Now, this style can be applied to the button element in a web page as follows:

<button name="sampleBtnName" id="sampleBtnId" **class="buttonStyle"**>I'm Button</button>
So, buttonStyle is used as the value for the class attribute of the button element, and it inherits all the styles declared in the CSS file. Now, let's try this on our Homepage. We will try to make WebDriver identify the search button using its class name and click on it.

First, in order to get the class name of the search button, as we know, we will use Developers tools to fetch it. After getting it, change the location mechanism to By.className and specify the class attribute value in it. The code for that is as follows:

```
@Test
public void byClassNameLocatorExample() {
   WebElement searchBox = driver.findElement(By.id("search"));
   searchBox.sendKeys("Electronics");
   WebElement searchButton =
         driver.findElement(By.className("search
         -button"));

   searchButton.click();
   assertThat(driver.get
   Title())
         .isEqualTo("Search results for: 'Electronics'");
}
```

In the preceding code, we have used the By.className locating mechanism by passing the class attribute value to it.

Sometimes, an element might have multiple values given for the class attribute. For example, the Search button has *button* and *search-button* values specified in the class attribute in the following HTML snippet:

```
<button type="submit" title="Search" class="button search
  button"><span><span>Search</span></span></button>
```

We have to use one of the values of the class attribute with the By.className method. In this case, we can either use *button* or *search-button*, whichever uniquely identifies the element.

The By.cssSelector() method

The By.cssSelector() method is similar to the By.xpath() method in its usage, but the difference is that it is slightly faster than the By.xpath locating mechanism. The following are the commonly used syntaxes to identify elements:

- To identify an element using the div element with the #flrs ID, we use the #flrs syntax
- To identify the child anchor element, we use the #flrs > a syntax, which will return the link element
- To identify the anchor element with its attribute, we use the #flrs > a[a[href="/intl/en/about.html"]] syntax

Let's try to modify the previous code, which uses the XPath locating mechanism to use the cssSelector mechanism:

```
@Test
public void
   byCssSelectorLocatorExample
   () { WebElement searchBox =
         driver.findElement(By.cssSelector("#search"));

   searchBox.sendKeys("Bags");
   searchBox.submit();
   assertThat(driver.getTitle())
         .isEqualTo("Search results for: 'Bags'");
}
```

The preceding code uses the By.cssSelector locating mechanism, which uses the css selector ID of the Search box.