# JUnit 5 tutorial - Learn how to write unit tests

**Prerequisite:**

- Eclipse
- Junit5
- Selenium

## 1. Test using Selenium

Create a JUNIT class under the package of src/test/java folder. This time check BeforeAll, Before Each, AfterEach method

**Step 1**: Create an object of WebDriver class

```
WebDriver driver;
```

**Step 2**: Under @BeforeAll, setup the web driver manager. Let's assume, we want to automate Chrome Browser.

```
WebDriverManager.chromedriver().setup();
```

**Step 3**: The session will be started. This step will be implemented under @BeforeEach

```
driver=new ChromeDriver();
```

**Step 4:** Take actions on Browser. In this example, we are navigating to a web page. This step also can be implemented under @BeforeEach

```
driver.get("https://www.selenium.dev/selenium/web/web-form.html");
```

**Step 5**: In this step, we will also do one more task that is to maximize the window.

```
driver.manage().window().maximize();
```

*We can also automate full screen and custom size window of the page*

**Step 6**: Request browser information. There are a bunch of types of information about the browser you can request, including window handles, browser size / position, cookies, alerts, etc. This step will be implemented under @Test.

```
String title = driver.getTitle();
assertEquals("Web form", title);
```

**Here, we will test the title of the web page*

**Step 7:** Establish Waiting Strategy. Read more about Waiting strategies. This step can also be implemented under @Test.

```
driver.manage().timeouts().implicitlyWait(Duration.ofMillis(500));
```

**The Implicit Wait in Selenium is used to tell the web driver to wait for a certain amount of time before it throws a "No Such Element Exception". The default setting is 0. Once we set the time, the web driver will wait for the element for that time before throwing an exception.*

**Step 8:** Find an Element. The majority of commands in most Selenium sessions are element related, and you can't interact with one without first finding an element. This step can also be implemented under @Test.

```
WebElement textBox = driver.findElement(By.name("my-text"));
WebElement submitButton = driver.findElement(By.cssSelector("button"));
```

**In this example, we will find one textbox named as my-text and submit buttion.*

**Step 9**: Take action on element. There are only a handful of actions to take on an element, but you will use them frequently. This step can also be implemented under @Test.

```
textBox.sendKeys("Selenium");
submitButton.click();
```

**In this example, "Selenium" is written inside the text box and submit button will be clicked.*

**Step 9**: Request element information. Elements store a lot of information that can be requested.

```
WebElement message = driver.findElement(By.id("message"));
String value = message.getText();
assertEquals("Received!", value);
```

**In this example, "Selenium" is written inside the text box and submit button will be clicked.*

**Step 10**: Request element information. Elements store a lot of information that can be requested.

```
WebElement message = driver.findElement(By.id("message"));
String value = message.getText();
assertEquals("Received!", value);
```

**In this example, a web element is found out by id("message"), then information of that element is requested using getText(). Next the value of the text is tested.*

**Step 11**: End the session. This ends the driver process, which by default closes the browser as well. No more commands can be sent to this driver instance. This step can be implemented under @AfterEach.

```
driver.quit();
```

**Step 12**: Run the class as a junit. You can see chrome browser will be started automatically and do all the tasks that you have specified in the program

## 2. Locators

Locators can be termed as an address that identifies an web element uniquely on a web page. These are the HTML properties of a web element


There are different ranges of web elements like radio button, text box, id etc. Identifying these elements is a tricky approach. Selenium uses locators to interact with the web elements on a web page. The locator is a "property-value" pair – For example "id" is the property and "email" is the value.

These are some Elements that can be located using Lacator:

| LOCATORS | DESCRIPTION | SYNTAX (IN JAVA) |
|---|---|---|
| id | Identify the WebElement using the ID attribute. | driver.findElement(By .id(“IdValue”)); |
| name | Identify the WebElement using the Name attribute. | driver.findElement(By .name(“nameValue”)); |
| className | Use the Class attribute for identifying the object. | driver.findElement(By .className(“classValue”)); |
| linkText | Use the text in hyperlinks to locate the WebElement. | driver.findElement(By .linkText(“textofLink”)); |

| | | |
|---|---|---|
| partialLinkText | Use a part of the text in hyperlinks to locate the WebElement. | driver.findElement(By .partialLinkText("PartialTextofLink")); |
| tagName | Use the tagName to locate the desired WebElement. | driver.findElement(By .tagName("htmlTag")); |
| cssSelector | CSS used to create style rules in the web page is leveraged to locate the desired WebElement. | driver.findElement(By .cssSelector("cssValue")); |
| xpath | Use XPath to locate the WebElement. | driver.findElement(By .xpath("xpathValue")); |

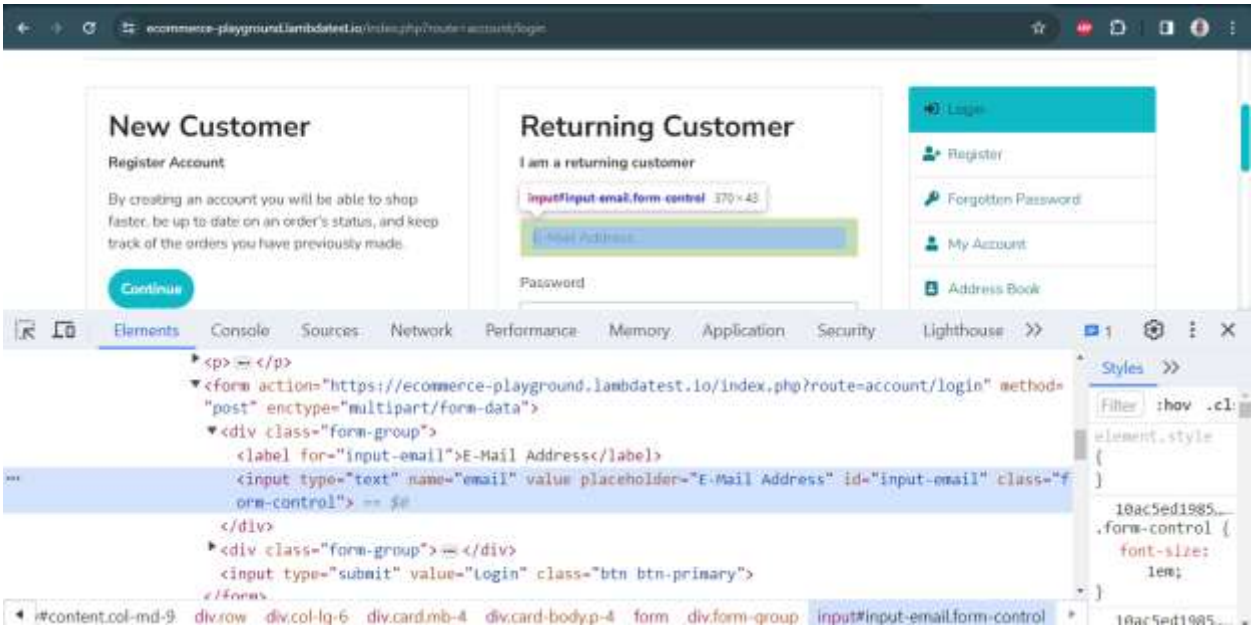
## 2.1. ID Locator

Ids are the most preferred way to locate elements on a web page, as each id is supposed to be unique which makes ids faster and more reliable way to locate elements on a page. With ID Locator strategy, the first element with the id attribute value matching the location will be returned.

*A NoSuchElementException will be raised, if no element has a matching id attribute.*

Below is an example of the LambdaTest eCommerce Playground showcasing how the E-Mail Address field can be located using the id locator:

**https://ecommerce-playground.lambdatest.io/index.php?route=account/login**




The below method is used for locating the desired element using the id locator:

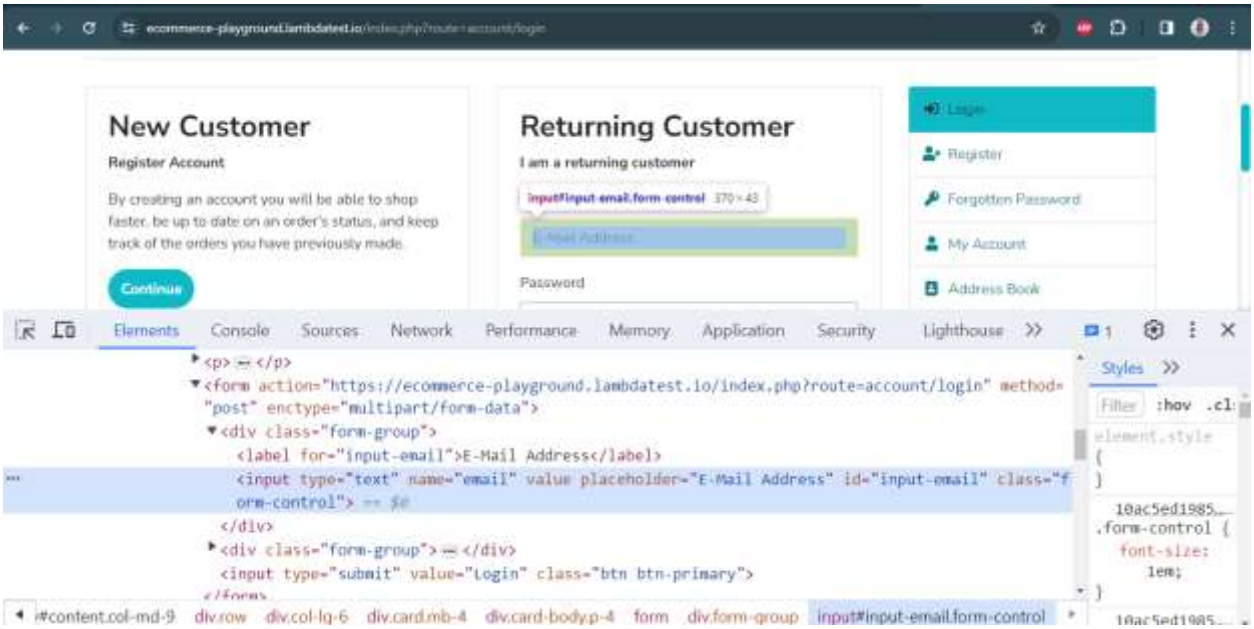
driver.findElement(By.id("input-email")).sendKeys("seleniumtesting@yahoo.com"); //id locator for text box

## 2.2 Name Locator

This is also the most efficient way to locate an element with a name attribute. With this strategy, the first element with the value name attribute will be returned.

*A NoSuchElementException will be raised, if no element has a matching name attribute.*

Now, let's understand the working of the name locator with an example. In the below image you can see, the name locator with a value called username. The difference is that you should use a name instead of id.

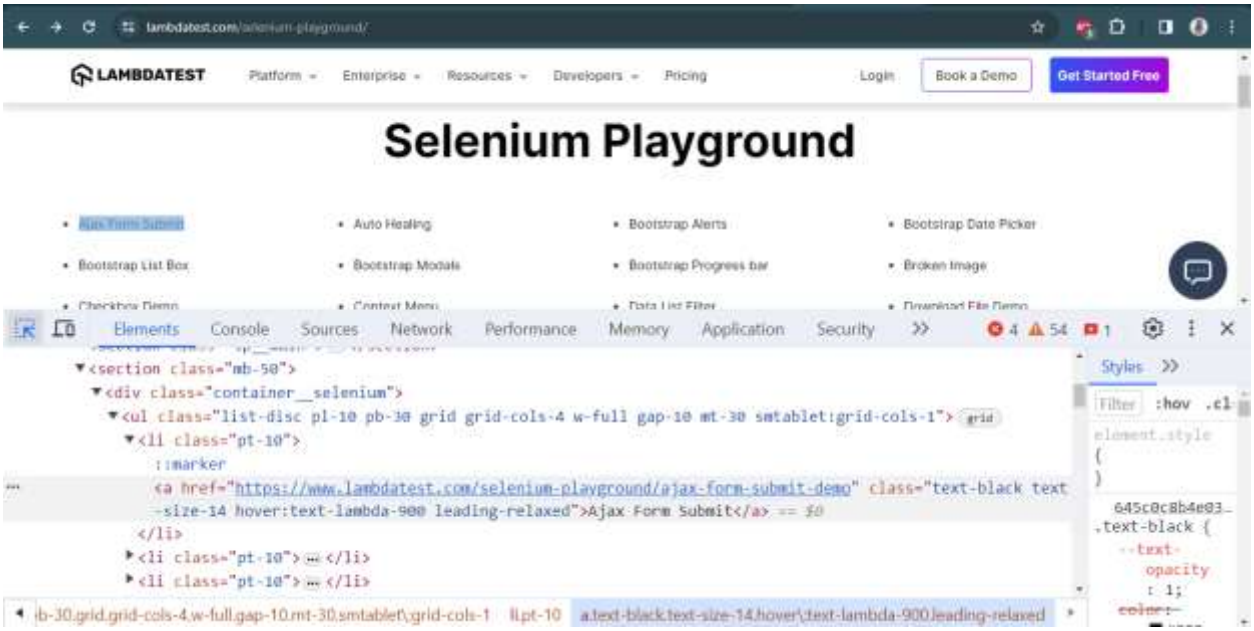



Here is how the desired WebElement can be located using the *name* locator in Selenium:

```
driver.findElement(By.name("email")).sendKeys("seleniumtesting@yahoo.com");
```

## 2.3 linkText Locator

Selenium linkText locator is used for identifying the hyperlinks on a web page. It can be identified with the help of an anchor tag “a”. In order to create the hyperlinks on a web page, you can use anchor tags.

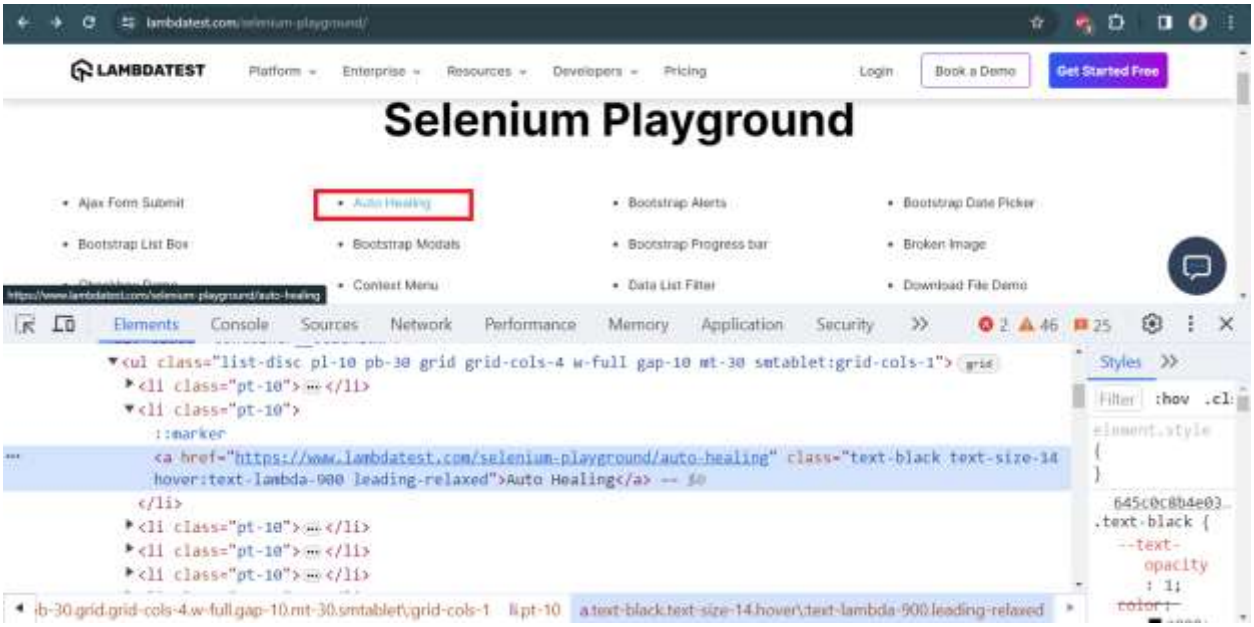



Below is an example of the LambdaTest Selenium Playground website showcasing the selection of the Ajax Form Submit link that is available on the home page. The DOM below shows the highlighted element:

driver.findElement(By.linkText("Continue")).click();

## 2.4 partialLinkText Locator

In some situations, we may need to find links by a portion of the text in a linkText element. In those situations, we use Partial Link Text to locate elements.




The syntax for locating elements by *partialLinkText* is:

driver.findElement(By.partialLinkText ("Con")).click();

## 2.6 TagName Locator

As the name specifies, this CSS locator in Selenium WebDriver is used to identify elements using tag names like *div*, *table*, *h1*, etc.

Here is the syntax for locating the fisrt links on the LambdaTest Selenium Playground homepage. Notice here that the *findElements* method is used. It will return a WebElements with the *tagName* ”*a*” that normally holds the links:

driver.findElement(By.*tagName*("a")).click();

*One important thing to note is that using a tagName() locator may lead to multiple elements being identified due to similar tags. In such cases, Selenium will select the first tag that matches the provided locator.*

## 2.7 CSS Selectors

CSS (Cascading Style Sheets) is used to style web pages. At the same time, CSS is also one of the widely-used ways to locate WebElements in the DOM.

The CSS Selector in Selenium should be chosen if you cannot locate an element using ID or Name locators. It can be chosen over the XPath locator.

Since multiple debates go around the corner for both of them, their usage may depend on the complexity of the scenario. However, most people prefer using CSS Selectors since those are faster than XPath.

**Tag and ID in CSS Selector**

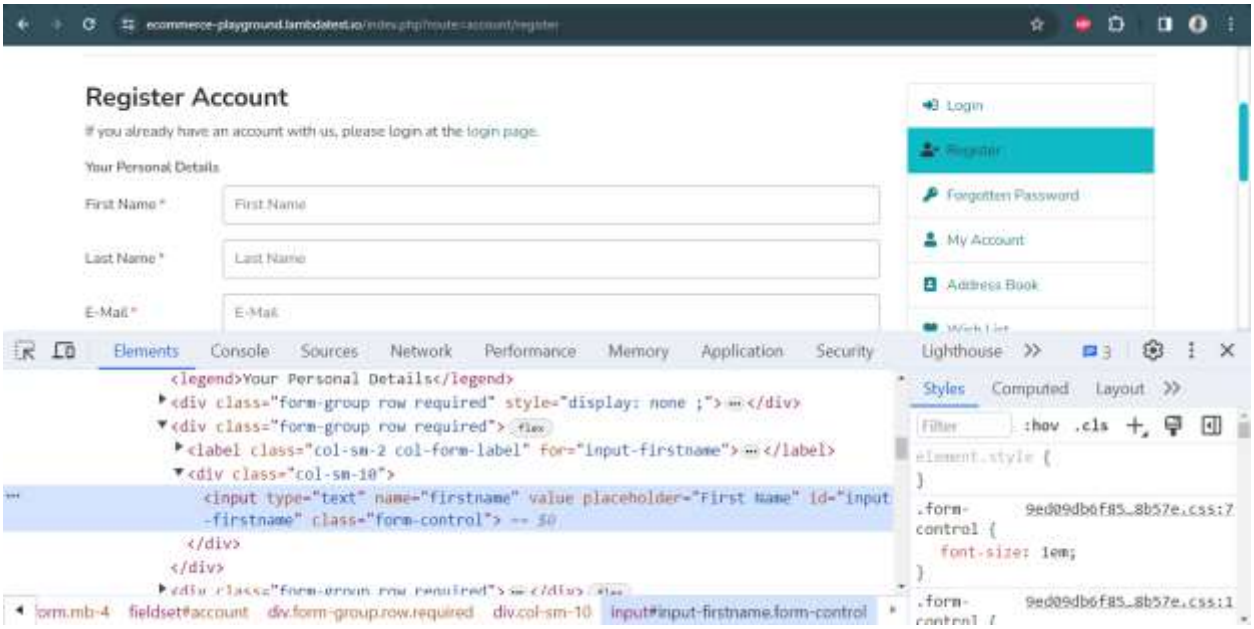
To locate elements by Tag and ID, you have to use the following components:

- **HTML tag**: It provides the tag we wish to locate (e.g., input tag).
- #: It is used to represent the ID attribute. Remember that when you wish to locate an element via ID through a CSS Selector, it must have a hash sign on the same. For other attributes, we need not use the hash sign.
- **Value of the ID attribute**: This represents the ID value we use to locate the element.

**Syntax:**

```
css=(Html tag )(#) (value of the ID attribute)
```

**Example:** Below is the DOM part indicating the First Name field on the Register Account page of the LambdaTest eCommerce Playground website.




driver.findElement(By.cssSelector("input#input-firstname")).sendKeys("seleniumtesting@yahoo.com");

### Tag and Class in CSS Selector

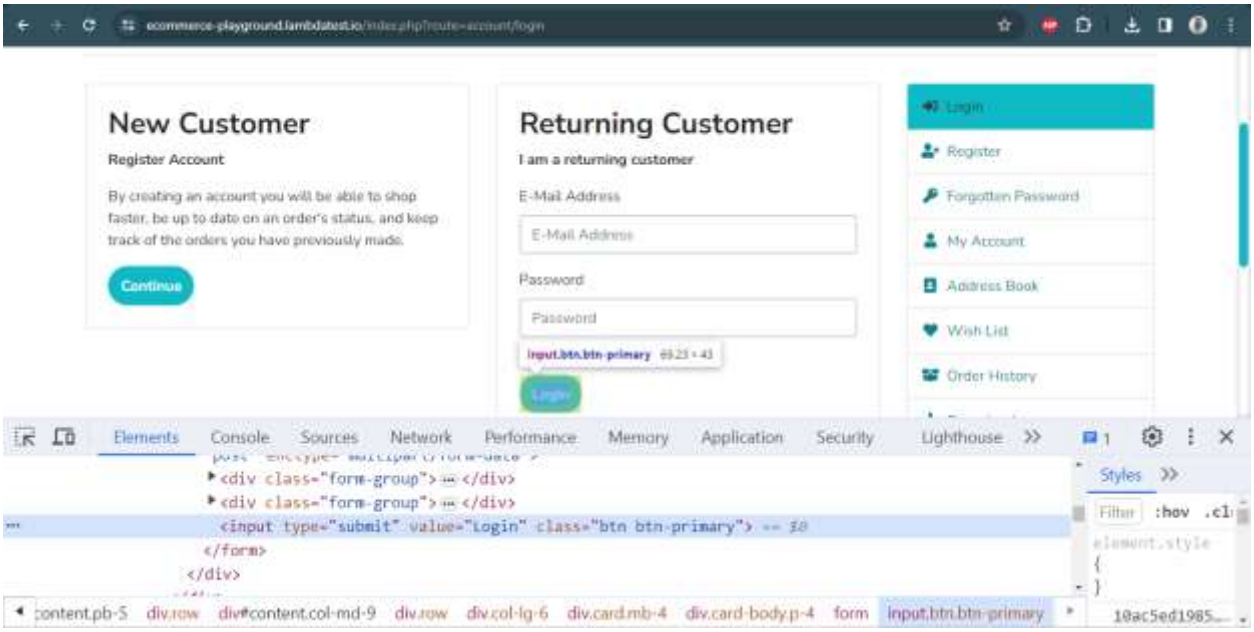
Apart from the syntax (or format) difference, the tag and class locator are identical to the ID locator. A dot (.) is used when denoting the class attribute value rather than hash (#) in the case of class.

### Syntax:

1css=(HTML tag)(.)(Value of Class attribute)

### Example:

Here is the DOM snapshot and the corresponding command to access the required WebElement using CSS Selector in Selenium for the Login button on the Account Login page of the LambdaTest eCommerce Playground website.




driver.findElement(By.*cssSelector*("input.form-control")).
sendKeys("seleniumtesting@yahoo.com");

**Tag and Attribute**

The element can be located via tag name, and the corresponding attribute is defined using its value. The first one will be selected if multiple elements have the same tag and attribute.

**Syntax:**

css=(HTML Page)[Attribute=Value]

**Example:**

Here is how the WebElement – 'phone' can be located using the *cssSelector* in Selenium.

driver.findElement(By.cssSelector("input[name=\"password\"]")).
sendKeys("password");

## 2.8 Relative Locators

https://www.selenium.dev/documentation/webdriver/elements/locators/

These locators are helpful when it is not easy to construct a locator for the desired element, but easy to describe spatially where the element is in relation to an element that does have an easily constructed locator.



Above

If the email text field element is not easily identifiable for some reason, but the password text field element is, we can locate the text field element using the fact that it is an “input” element “above” the password element.

```
By emailLocator = RelativeLocator.with(By.tagName("input")).above(By.id("password"));
```

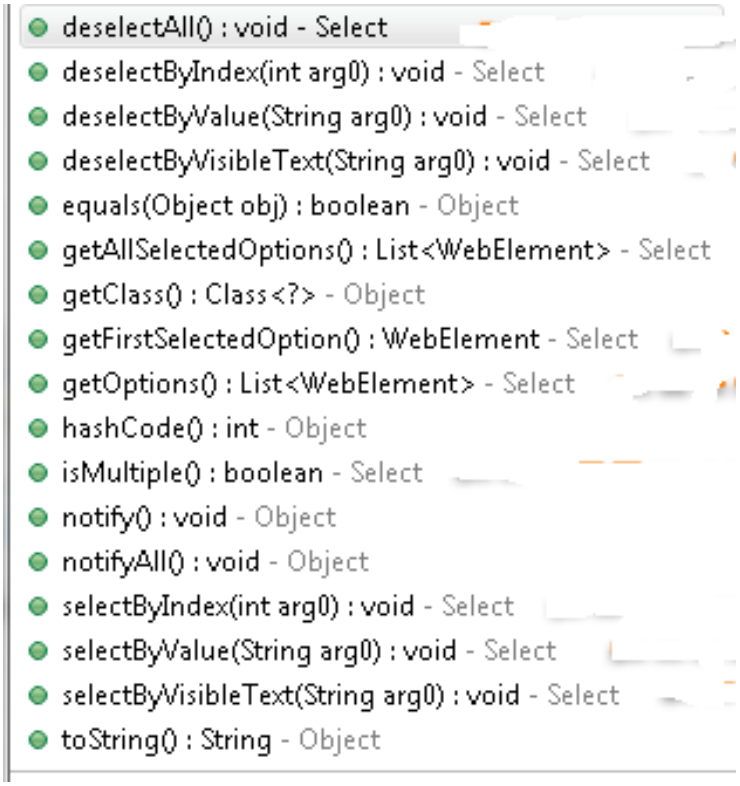
## 3. Handle

### 3.1 handle Dropdown

Select class is a WebDriver class provides the implementation of the HTML Select tag

Syntax:

Select oselect=new Select(findElementBy.id(“id”));




The select class has several methods to select or de-select dropdown's values

selectByVisibleText("parameter")

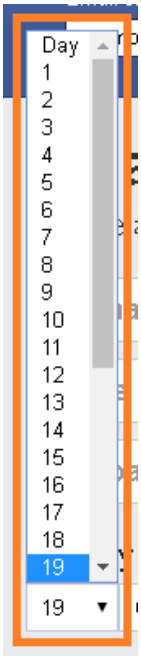
deselectByVisibleText("parameter")

selectByIndex(index)

deselectByIndex(index)

selectByValue("value")

delectByValue("value")




1. parameter - represents the value of dropdown.
2. index - represents the index of the dropdown's value. It starts with 0, i.e., the index of the first value of dropdown is zero.
3. value - represents the value of "value" property.

### 3.1.1 **selectByVisibleText**

**WebElement daytext = driver.findElement(By.id("day"));**//identifing the 'day' dropdown available on facebook login page.

**Select select = new Select(daytext);**//instantiating the dropdown elements as Select class object.

**select.selectByVisibleText("10");**//selecting value from dropdown.

*Explanation:*

- *"day" is the drop-down element having day values from (1-30/31).*
- *"select" is the object of the Select class that refers to "day" dropdown's values.*
- *We are selecting "10" as the text parameter.*

### 3.1.2 selectByIndex

**WebElement dayIndex = driver.findElement(By.id("day"));**//identifing the 'day' dropdown available on facebook login page.

**Select select = new Select(dayIndex);**//instantiating the dropdown elements as **Select class object. select.selectByIndex(11);**//selecting value from dropdown.

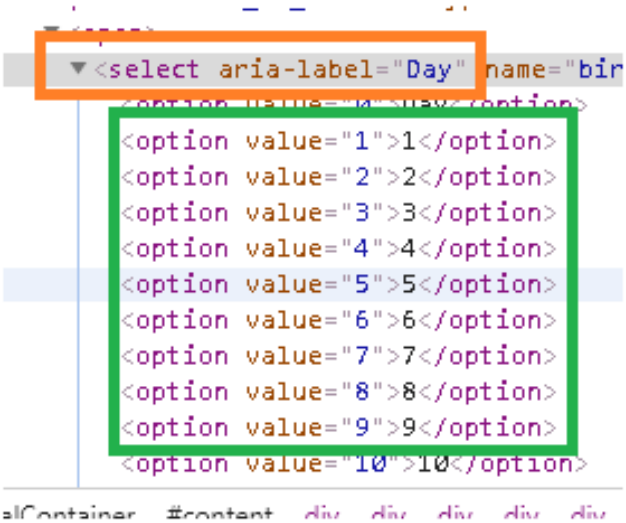
**Explanation –**

- **"day"** is the drop-down element having day values from (**1-30/31**).
- **"select"** is the object of **the Select** class that refers to "**day**" dropdown's values.
- We are selecting "11" as the index value. It will select "11" as the date in the drop-down.

### 3.1.3 selectByValue

**WebElement dayValue = driver.findElement(By.id("day"));** //identifing the 'day' dropdown available on facebook login page.

**Select select = new Select(dayValue);** //instantiating the dropdown elements as Select class object. **select.selectByValue("11");** //selecting value from dropdown




Here, I inspected "**day**" drop-down using developer tool. In the image you can see **options** available for the dropdown. You can see "**value**" property and its **value** beside <**option**> tag. In the below code, we will use this **value**.

### 3.1.4 getOptions

This method helps to get all the options belonging to a select tag. It takes no parameters and returns a list of web elements

**Select oselect= driver.findElement(By.id("day"));** //identifing the 'day' dropdown available on facebook **List<WebElement> elementcount= oselect.getOptions();**

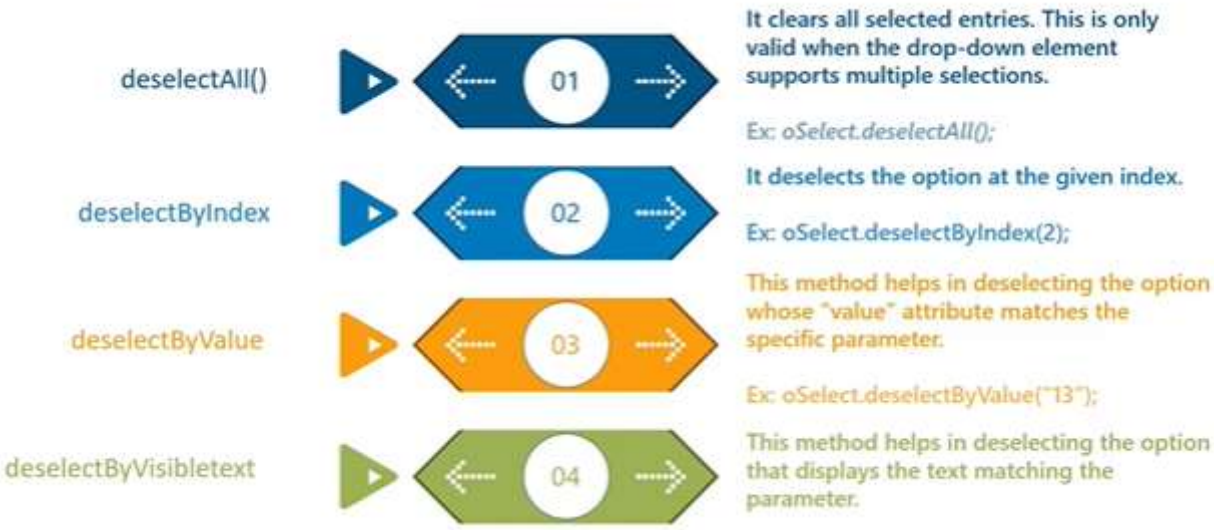
**System.out.println(elementcount.size());**

### 3.1.5 **isMultiple**

Multiple Select attribute is a Boolean expression. When it is present, it specifies that multiple options can be selected at once

**Select.isMultiple()**

This method tells whether the select element support multiple select options at the same time or not. It takes no parameters and returns Boolean value




## 3.2 dynamic elements

Advanced topics

# 4 Actions API

Advanced topics