SYNTAX TECHNOLOGIES

Selenium

Class 2

Agenda

WebDriver commands

What is an element?

Locators in Selenium

WebDriver Browser Commands

➔ **get(String url);**

- Command launches a new browser and opens the specified URL in the browser instance
- The command takes a single string type parameter that is usually a URL of application under test

```
driver.get("https://google.com");
//Or can be written as
String URL = "https://google.com";
driver.get(URL);
```

➔ **getTitle();**

- Command is used to retrieve the title of the webpage the user is currently working on.
- A null string is returned if the webpage has no title
- Command doesn’t require any parameter and returns a trimmed string value

```
String title = driver.getTitle();
```

WebDriver Browser Commands

- ➔ **close();**
  - This method **Close** only the current window the WebDriver is currently controlling.
  - Accepts nothing as a parameter and returns nothing.
  - Quit the browser if it's the last window currently open.

```
driver.close();
```

- ➔ **quit();**
  - This method **Closes** all windows opened by the WebDriver.
  - Accepts nothing as a parameter and returns nothing.
  - Close every associated window.

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

WebDriver Navigation Commands

➔ **navigate().to(String url);**

- This method **Loads** a new web page in the current browser window. It accepts a String parameter and returns nothing.
- It does exactly the same thing as the **driver.get(String url)** method.
- Able to redirect from the current web page to the expected web page.

```
driver.navigate().to("https://www.google.com");
```

➔ **navigate().refresh();**

- This method ***Refresh*** the current page. It neither accepted nor returns anything.
- Perform the same function as pressing F5 in the browser.

```
driver.navigate().refresh();
```

WebDriver Navigation Commands

➔ **navigate().forward();**

- This method does the same operation as clicking on the **Forward Button** of any browser.
- It neither accept nor returns anything.
- Takes you forward by one page on the browser’s history.

driver.navigate().forward();

➔ **driver.navigate().back();**

- This method does the same operation as clicking on the **Back Button** of any browser.
- It neither accepted nor returns anything.
- Takes you back by one page on the browser’s history.

driver.navigate().back();

Difference Between get() and navigate()

driver.get("URL")	**driver.navigate().to("URL")**
driver.get("http://www.google.com");	driver.navigate().to("http://www.google.com");
It's used to go to the particular website , But it doesn't maintain the browser History and cookies, we can't use forward and backward button , if we click on that , page will not get schedule	It's used to go to the particular website , but it maintains the browser history and cookies, we can use forward and backward button to navigate between the pages during the coding of Testcase
is used to navigate particular URL(website) and wait till page load.	is used to navigate to particular URL and does not wait to page load

Test Case

TC 1: Amazon Page Title Verification:

1. Open chrome browser
2. Go to "https://www.amazon.com/"
3. Verify Title "Amazon.com: Online Shopping for Electronics, Apparel, Computers, Books, DVDs & more" is displayed

TC 2: Syntax Page URL Verification:

1. Open chrome browser
2. Navigate to "https://www.syntaxtechs.com/"
3. Navigate to "https://www.google.com/"
4. Navigate back to Syntax Technologies Page
5. Refresh current page
6. Verify url contains "Syntax"

What are objects or webelements in a webpage?

1. A Web Page consists of buttons, links, inputs boxes, drop-down lists, check-boxes, radio buttons, plain texts and other items. These are called Web Elements or Objects in Test Automation.

2. In order to perform actions (like click, type, select etc) on these elements, Selenium needs to locate them uniquely in page. And to help Selenium locate them we need to identify their properties in page to pass them to Selenium.

What are Locators ?

In Selenium Automation, Locators are used to locate the UI (User Interface) elements of a page like Text Box field, Button etc

Locator can be termed as an address that identifies a web element uniquely within the webpage

Locators are the HTML properties of a web element which tells the Selenium about the web element it need to perform action on.

Locators are: Identifying an UI element on the application screen as an important feature for any automation tool. Locators allows the Selenium Automation tool to find UI elements on a page that can be used in our automation tests.

What are Locators ?

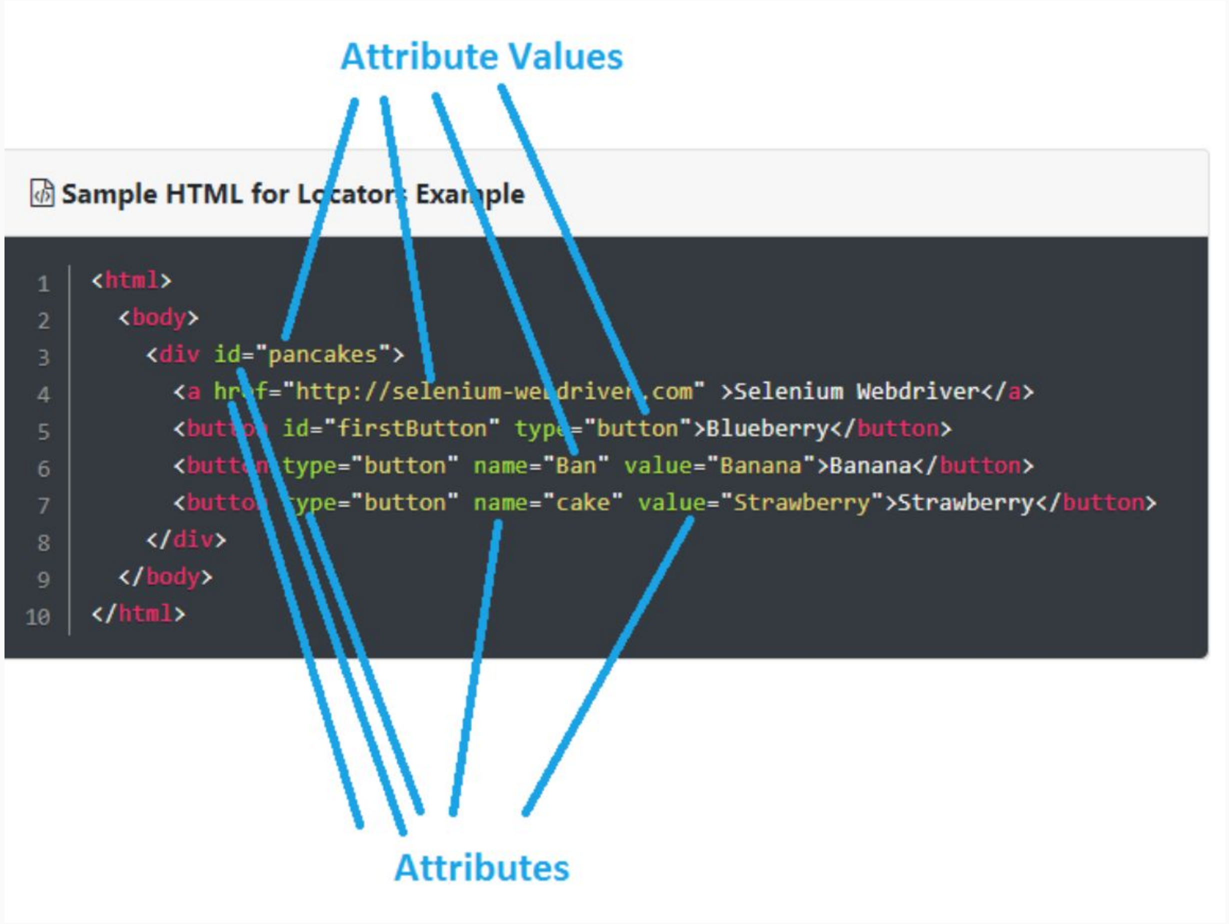
The Selenium Tool uses Locators to find and match the elements on your page that it needs to interact with while running the automation scripts.

Locators are the lifeblood of the tests. Using the right locator ensures the tests are faster, more reliable or has lower maintenance over releases.

If you're fortunate enough to be working with unique IDs and Classes, then you're usually all set. But there will be times when choosing a right locator will become a nightmare.

It can be a real challenge to verify that you have the right locators to accomplish what you want.

Sample of HTML Locators



Sample HTML for Locators Example

```
<html>
  <body>
    <div id="pancakes">
      <a href="http://selenium-webdriver.com" >Selenium Webdriver</a>
      <button id="firstButton" type="button">Blueberry</button>
      <button type="button" name="Ban" value="Banana">Banana</button>
      <button type="button" name="cake" value="Strawberry">Strawberry</button>
    </div>
  </body>
</html>
```

Locating Web Elements in Browser

Now days, it is very easy to find elements on the page. All the modern browsers have the web inspector, the built-in tool for to easily examine the structure of web pages.

To identify to which element this refers to just do the following:

- Open the URL / web page in the Chrome browser
- Open the Web developer tools by pressing:
- Cmd + Alt + I (on Mac)
- or by clicking View -> Developer -> Developer tool
- or by Right-Click and Inspect Element

Locating Web Elements in Google Chrome

Google Chrome has its own developer tool that can be used to identify and locate web elements on the web page.

Follow the below steps to locate web elements using Chrome's Developer tool:

Step #1: The primary step is to launch the Google Chrome's Developer tool. Press F12 to launch the tool.
You can also launch developer tool by right-clicking anywhere within the web page and by selecting "Inspect element"

Step #2: The next step is to locate the desired object within the web page. One way to do the same is to right click on the desired web element and inspect. The HTML property belonging to that web element would be highlighted in the developer tool.
Another way is to hover through the HTML properties and the matching web element would be highlighted. Thus, in this way user can locate ids, class, links etc.

Locating Web Elements in Internet Explorer

Internet Explorer also has its own Developer Tool that can be used to identify web elements based on their properties within the web page.

Follow the below steps to locate web elements using IE Developer tool:

Step #1: The primary step is to launch the IE Developer tool.

Press F12 to launch the tool. The user would be able to see something like the below screen.

Step #2: The next step is to locate the desired object within the web page. One way to this is to select the HTML element and the matching web element would be highlighted.

Thus, in this way user can locate ids, class, links etc. Check out in the below screenshot in which Email Textbox would be highlighted as soon as we select the corresponding HTML property.

Locating GUI Elements

Locating elements in WebDriver is done by using the "findElement(By.locator())" method.

Selenium webdriver uses 8 locators to find the elements on web page:

- ID
- Name
- LinkText
- partialLinkText
- className
- tagName
- XPath
- CSS Selector

Locators

- Id - Select element with the specified @id attribute.
- Name - Select first element with the specified @name attribute.
- Linktext - Select link (anchor tag) element which contains text matching the specified link text
- Partial Linktext - Select link (anchor tag) element which contains text matching the specified partial link text
- Tag Name - Locate Element using a Tag Name
- Class name - Locate Element using a Class Name
- Css - Select the element using css selectors
- Xpath - Locate an element using an XPath expression.

UI elements using the following example:

Example#1

Suppose we've to automate a test, where the User have to log in to Facebook

Steps to Log In to Facebook application:

1. Browse www.facebook.com in any browser
2. On 'Facebook' login page, enter your email id into the **Username text box.**
3. Enter your Facebook password into the **Password text box.**
4. Click on **'Log In' button**.

In the above example, Username text box, Password text box and Log In text box are the UI elements to be identified by the automation tool.

Locating an Element By ID

- **Ids** are the **most preferred way** to locate elements on a page, as each id is supposed to be unique which makes ids a very faster and reliable way to locate elements.

- IDs are the safest and fastest locator option and should always be the first choice even when there are multiple choices, It is like an Employee Number or Account which will be unique.

- With this strategy, the first element with the id attribute value matching the location will be returned. If no element has a matching id attribute, a **NoSuchElementException** will be raised.

Locating an Element By ID:

```
<div id="toolbar">....</div>

<input id="email" class="required" type="text"/>
```

We can write the scripts as

```
WebElement Ele = driver.findElement(By.id("toolbar"));
                                  or
WebElement emailEle = driver.findElement(By.id("email"));
```

Even though this is a great locator, obviously it is not realistic for all objects on a page to have ids. In some cases developers make it having non-unique ids on a page or auto-generate the ids, in both cases it should be avoided.

Locating an Element By Name

- When there is no Id to use, the next worth seeing if the desired element has a name attribute. But make sure there the name cannot be unique all the times. If there are multiple names, Selenium will always perform action on the first matching element

- If no element has a matching name attribute, a **NoSuchElementException** will be raised.

```
<input name="register" class="required" type="text"/>
```

```
WebElement register= driver.findElement(By.name("register"));
```

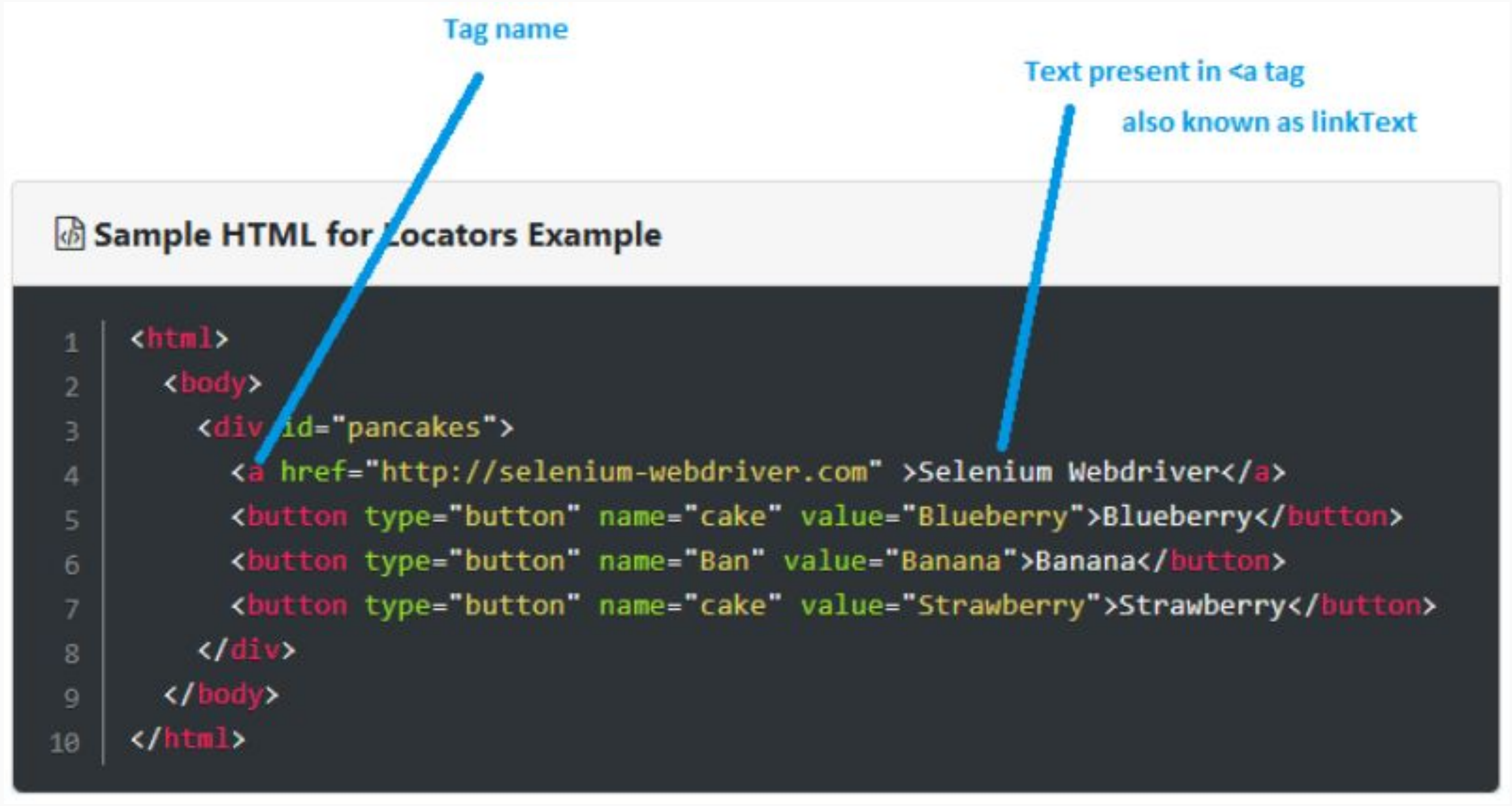
Locating an Element By LinkText

- Finding an element with link text is very simple. But make sure, there is only one unique link on the web page.
- If there are multiple links with the same link text (such as repeated header and footer menu links), in such cases Selenium will perform action on the first matching element with link
- With this you can find elements of "**a**" tags(**Link**) with the link names. Use this when you know link text used within an anchor tag.

```
<a href="http://www.seleniumhq.org">Downloads</a>

WebElement download = driver.findElement(By.linkText("Downloads"));
```

Locating an Element By LinkText



```
<html>
  <body>
    <div id="pancakes">
      <a href="http://selenium-webdriver.com" >Selenium Webdriver</a>
      <button type="button" name="cake" value="Blueberry">Blueberry</button>
      <button type="button" name="Ban" value="Banana">Banana</button>
      <button type="button" name="cake" value="Strawberry">Strawberry</button>
    </div>
  </body>
</html>
```

Locating an Element By Partial LinkText

- In the same way as LinkText, PartialLinkText also works in the same pattern.
- User can provide partial link text to locate the element.

```
<a href="seleniumhq.org">Download selenium server</a>

WebElement download = driver.findElement(By.PartialLinkText("Download"));
```

Locating an Element By TagName

- TagName can be used with Group elements like , Select and checkboxes / dropdowns.

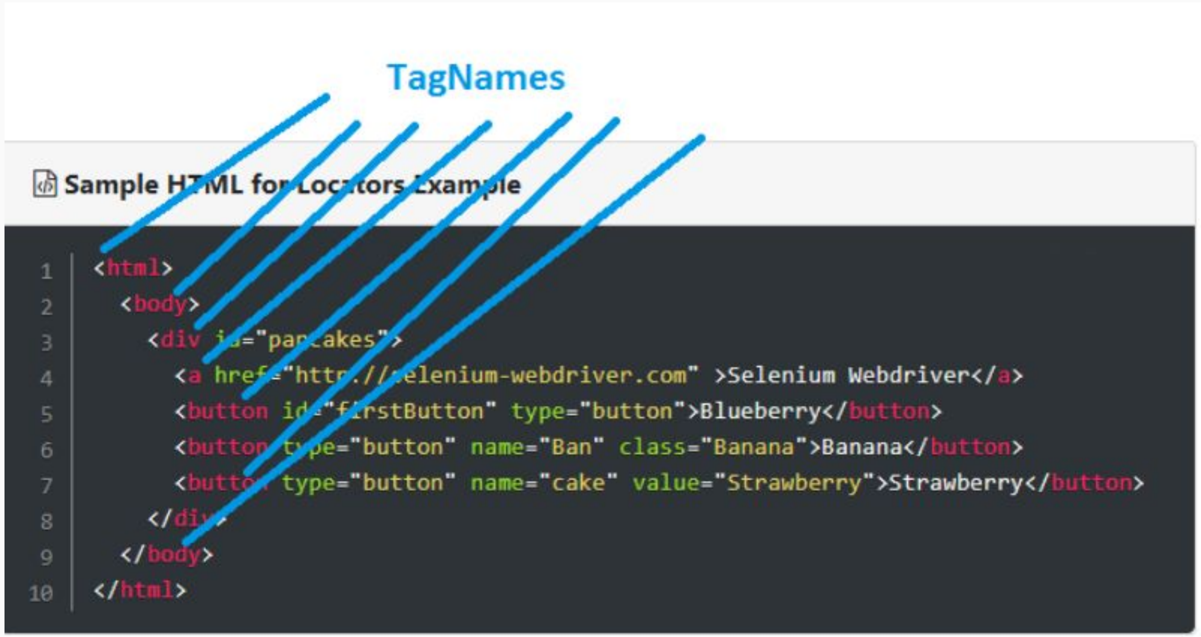
```
<h1>Welcome</h1>
driver.findElement(By. tagName (“h1"));
```

```
<select>
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="mercedes">Mercedes</option>
  <option value="audi">Audi</option>
</select>
```

```
Select select = new Select(driver.findElement(By.tagName("select")));
select.selectByVisibleText(“Audi");
```

Locating an Element By TagName

- The tagname locator looks for an element in the page having an tagname, like <a>, <button>, <p>, <label>, <div> etc



```html
<html>
  <body>
    <div id="pancakes">
      <a href="http://selenium-webdriver.com" >Selenium Webdriver</a>
      <button id="firstButton" type="button">Blueberry</button>
      <button type="button" name="Ban" class="Banana">Banana</button>
      <button type="button" name="cake" value="Strawberry">Strawberry</button>
    </div>
  </body>
</html>
```

Locating an Element By Class Name

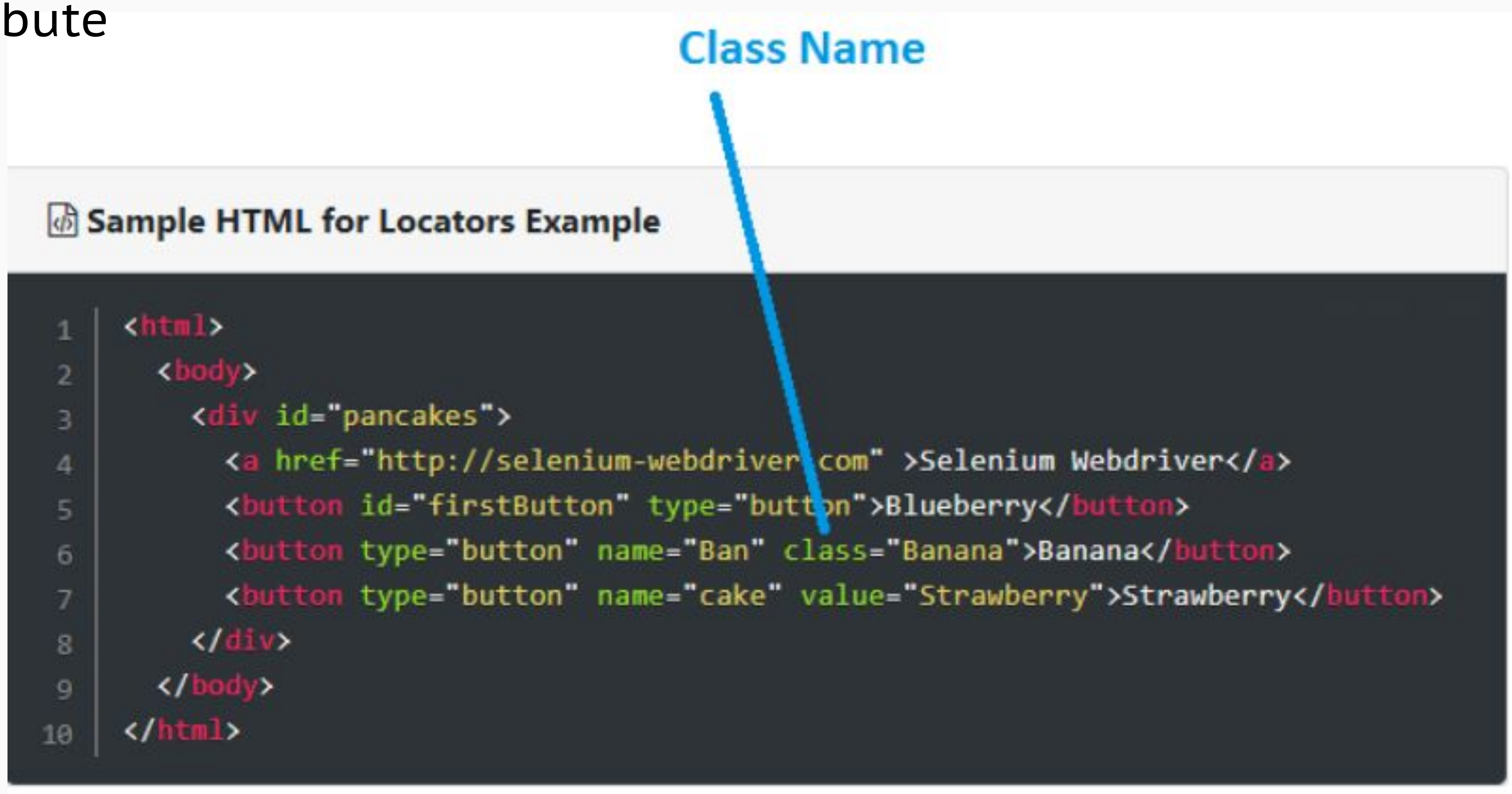
- There may be multiple elements with the same name, if we just use findElementByClassName , make sure it is only one.
- If not then you need to extend using the className and its sub elements.

```
<input id="gbqfq" class="Password" type="text" value=""
autocomplete="off" name="q" style="border: medium none; padding: 0px;
margin: 0px; height: auto...; width: 100%; background: url('" dir="ltr"
spellcheck="false"></input>
```

```
WebElement login= driver.findElement(By.className("Password"));
```

Locating an Element By Class Name

The Class name locator looks for an element in the page having an class attribute



Sample HTML for Locators Example

```
<html>
  <body>
    <div id="pancakes">
      <a href="http://selenium-webdriver.com" >Selenium Webdriver</a>
      <button id="firstButton" type="button">Blueberry</button>
      <button type="button" name="Ban" class="Banana">Banana</button>
      <button type="button" name="cake" value="Strawberry">Strawberry</button>
    </div>
  </body>
</html>
```