# Selenium-WebDriver-Introduction

<Video-transcript>: Exposing webdriver syntax for the first time.

Now that you have seen glimpse of what an actual framework looks like and played around with the framework, you would remember that when test case was executed through framework, opening the browser & application, populating the fields in the application with appropriate values, button clicks, checking checkboxes, managing popups were handled through code. Let us understand how it happens programmatically.

Let us visit our Test Application used in framework, Cyclos.

On Day-1, if you would have wondered how the code opens browser and opens URL etc. Let us visit the code which does that in the framework.

Go to code, src🡪Utils🡪BaseClass.java



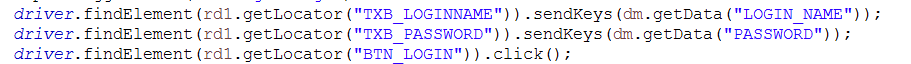
You can possibly guess first line opens Chrome Browser and second line opens Cyclos URL. Okay that’s enough to observe here.

Let us open Cyclos Test application manually. A url is given below this video so that you can open it. So it’s a login page with Login Name, Password and Submit button and other stuff.

Suppose you want to login, you need to enter Login Name, Password and Click on Submit button.

For our application, Admin credentials are Login Name:admin, Password:1234. Let’s do it manually and verify that these are working.

Ok. So they are working. Now, let’s go to our Framework code, src🡪PageObjects🡪Login.java .



If you observe, there is a method named performLogin. There are other lines of code which does bunch of other things, but if we just focus on *(focus in on those 3 lines in video blurring other things*), can you see some similarities in the fields we have in Cyclos Login page and lines of code we have highlighted in the code. At least, some English words are matching. That means these 3 lines does login operation in our code.

Well, these lines are coded using something known as WebDriver API. Let us understand what web driver API is, in next video.

Open src🡪PageObjects🡪EnrollMember.java

This class has steps to register a new member to the application

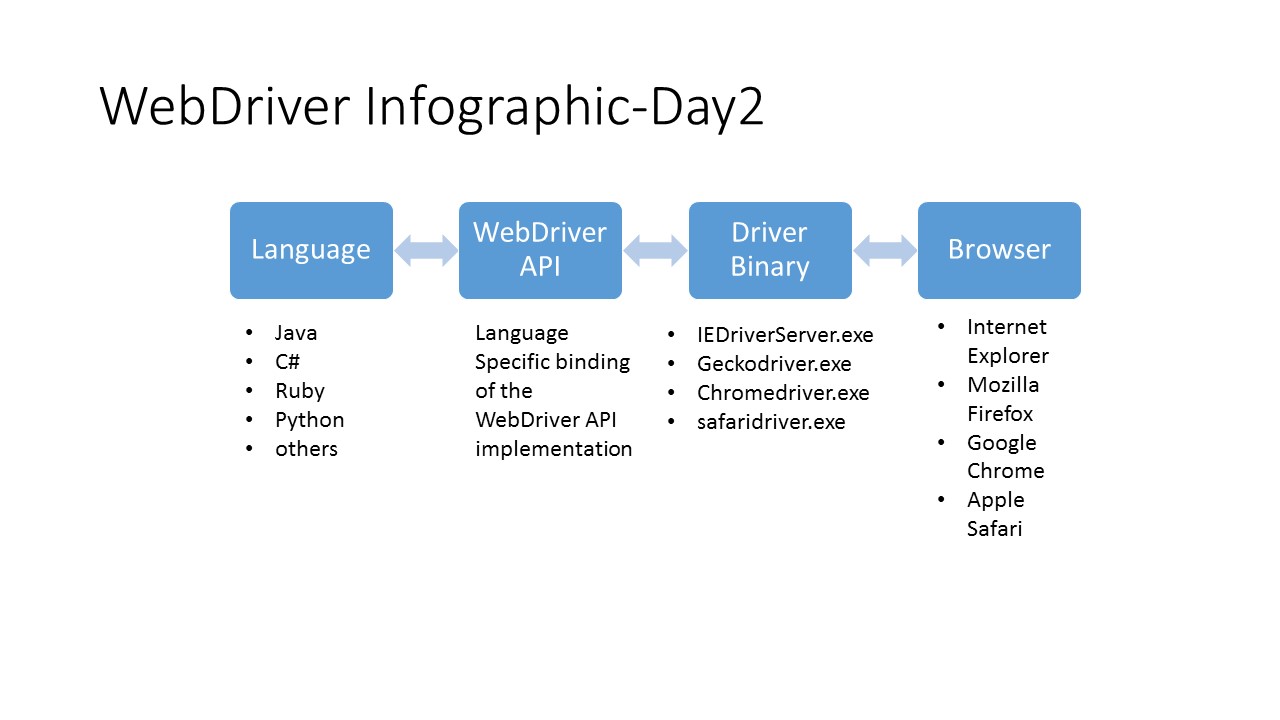
<Video-transcript>

# WebDriver Origin & Usage

<Video-transcript>:

In this video, we will discuss how webdriver does all the magic of talking to the browser and instructing it to do as written in your java code.

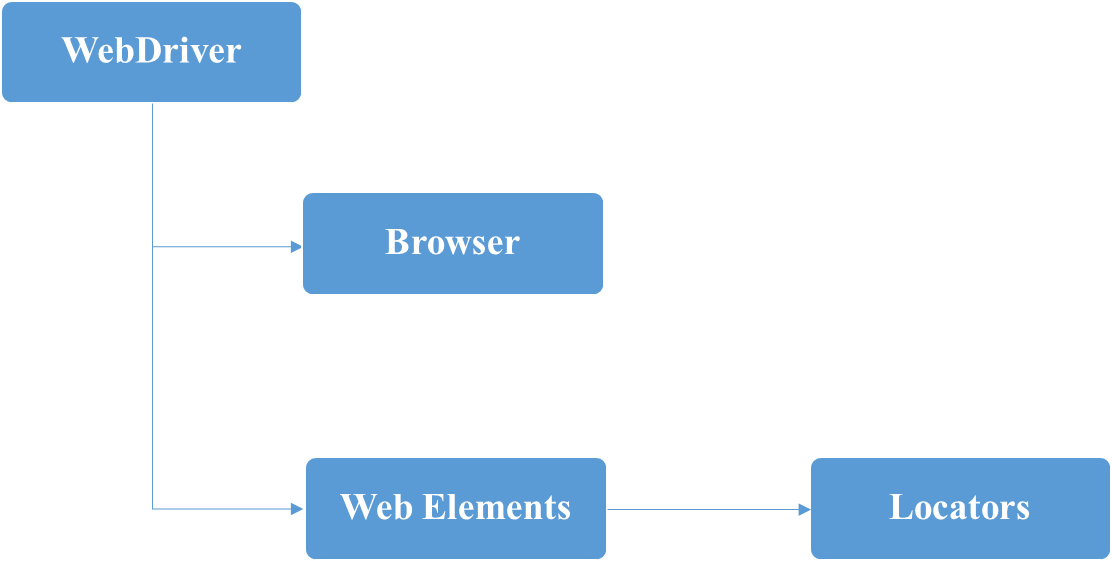
Let’s understand it using this infographic:



* Highlighting 1: Browser tests can be written in any language of choice. Languages like Java, C#, Ruby, Python, etc. are officially supported by SeleniumHQ folks and some like Perl, PHP, and Haskell are some 3rd party bindings.
* Highlighting 2: WebDriver is a set of interfaces (similar to Interface in Java, remember? Classes, methods and Interfaces) to remotely control behavior of web browsers. Recently W3C standard implementation of WebDriver is initiated so that all the vendors have to follow this while maintaining their driver binary.
* Highlighting 3: Since each browser has different technologies underlying them and it is impossible to have common commands to automate them all. Besides it is not possible to expose proprietary APIs to control browsers remotely because of security issues. Hence each vendor (Google for Chrome, Microsoft for IE, Mozilla for FireFox, etc.) maintain a Driver binary as we call it here, which translates WebDriver commands into commands which Browser can understand.

<Video-transcript>

# How are we going to learn WebDriver

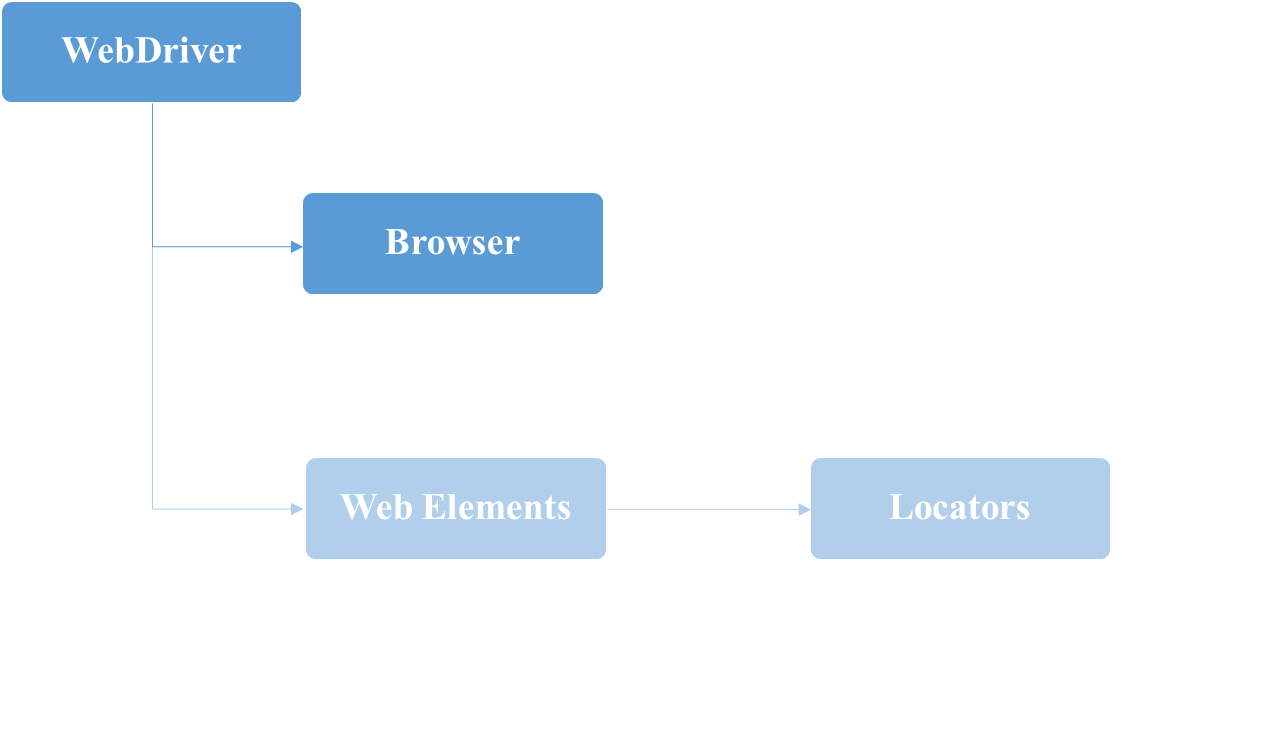


To create a framework like the one we have seen before, you will need above things inside webdriver to learn and understand the syntax.

1. We have used Browser interactions from webdriver API in framework. Normal things like opening a URL, navigating across history of pages, closing the browser, etc. will be covered here.
2. Inside a browser, we would require to interact with text box, drop down, radios, check boxes, web tables, etc. They can all be called Web Elements in general.
3. To identify any web element as discussed above, we need unique property. These are known as locators. You know them from Selenium IDE from Day-1.

# WebDriver: Browser Interactions

Let us understand Browser Interactions with the help of Demo1.java in Demos package.



1. System.setProperty: This is not a WebDriver command, it’s a java command to set system variable path for Driver Binary for respective browser you wish to test your application with. Something similar has to be done in other languages in whatever way environment variables can be created.

Ex:



In this snippet, webdriver.chrome.driver is name of an environment variable and second argument is value of that variable. WebDriver automatically picks up driver executable from above mentioned location for interacting with Chrome browser.

* webdriver.ie.driver for Internet Explorer
* webdriver.chrome.driver for Chrome driver
* webdriver.gecko.driver for Firefox driver

1. Creating object of browser driver:



To simplify things for now, this line opens Chrome Browser automatically. If you want to know more, interesting explanation on Core Java concepts used in this line is provided at the end of the course in Appendix. You can go through it at your leisure.

1. Loads the web page of the corresponding URL passed as String, in the current browser window.

Ex:



1. Returns the title of the current page

Ex:



1. Maximizes the current browser window, if it’s not already maximized.

Ex:



1. Navigation in WebDriver:

* Move back a single “item” in the browser’s history.



* Move a single “item” forward in the browser’s history. Does nothing if we are on the latest page viewed.



* Refresh the current web page.



* Loads the web page of the corresponding URL, in the current browser window, using HTTP protocol



1. Closes the browser tab or page currently which is having the focus, if only one tab is open it closes the browser.



1. Closes all the tabs and hence the browser.



# WebDriver: Web Elements & Locators

Hope you got some idea about controlling browser with code in previous section, in this one let us understand how to interact with content inside a web page.

For demo, we will refer to Demo2.java given in your Demos package. You can execute it. It is a rough implementation of the Enroll Member test case.

1. TextBox:



*Note:* Since this is the first syntax with any element, let us dig in deep a little for this one, for later syntaxes, we will assume these things implicitly.

So driver points to current page in Browser. In Cyclos Login Page, there are many web elements like [Login Name] textbox, [Password] textbox, [Submit] button. We want to point exactly to [Login Name] textbox.

So just like each person can be uniquely identified by PAN number, Social security number, each element can be identified by id or xpath or linktext or cssselector. Now just like every person might not have PAN, in which case they can be identified by their Voter ID number or Adhaar number, similarly the way, whatever properties a front-end developer would have coded, one has to make a choice for choosing appropriate locator out of available ones.

Remembering Day-1 IDE locators, in Java, it will be mentioned as

* By.id(“id of that element”)
* By.cssSelector(“cssSelector of that element”)
* By.XPath(“xpath of that element”)
* By.tagName(“tagName of that element”)
* By.linkText(“linkText of that element”)
* By.partialLinkText(“partialLinktext of that element”)
* By.className(“className of that element”)
* By.name(“name of that element”)

We will see more on Xpath later.

1. Button:



1. HyperLink: Let us take example of clicking on Web Apps link in Sparsh. (Refer demo1.java)



Another way, to point to a link is to use a partial link text. For eg. Above line can also be written as



Or



1. CheckBox: (Demo2.java)

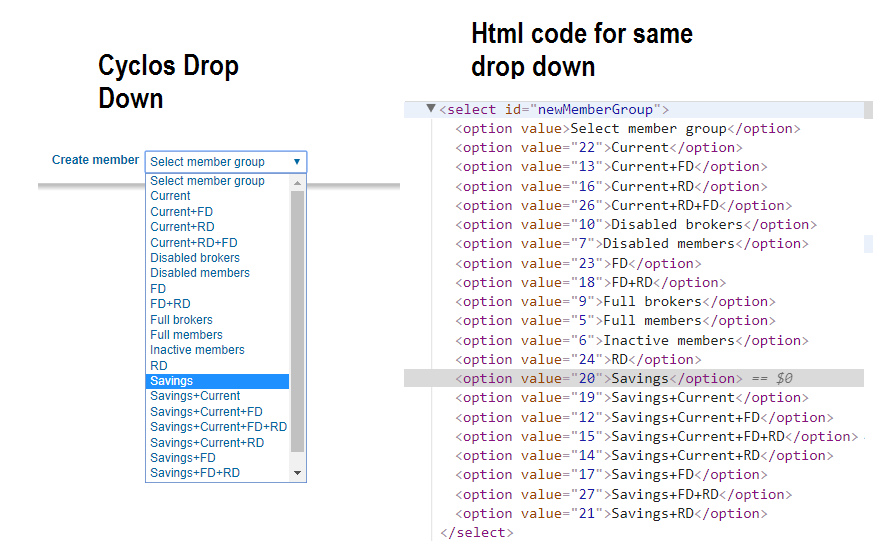


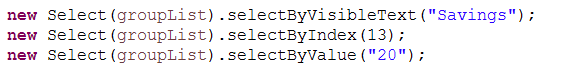
1. RadioButton:



1. DropDownList

This would be used to select drop down value. However, there are other syntaxes which help in selecting drop down values by index and by value html property of that option in drop down.

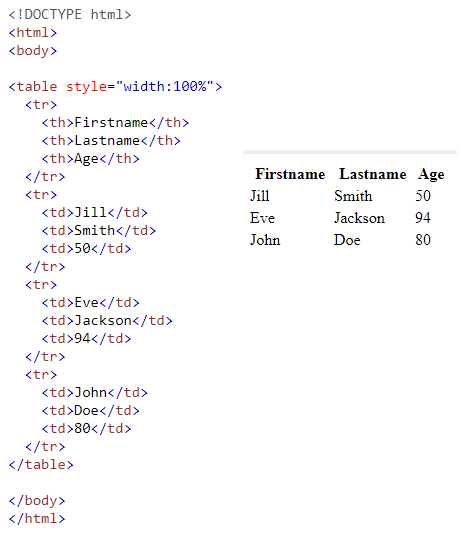




Above 3 syntaxes do the same thing, select Savings from that drop down.

1. WebTable:

Reading data from a web table is little tricky. But not difficult. Let us understand it.



Conclusions:

1. Every web table starts with <table> tag.
2. It has as many rows, as many <tr> tags.
3. First <tr> tag would have <th> tags and later all the <tr> tags would have <td> tags for column.

Observe Demo3.java in Demos package. <Possible video here>

# Methods in WebElement Interface

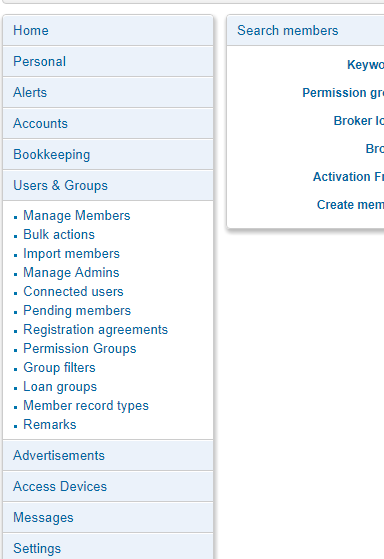
As you know, each user element in a web page is known as Web Element in terms of WebDriver.

Now there are lot of ways, one can interact with any web element. Let’s see few of them in brief.

1. 

At times there will be elements on the page that will be hidden under a menu as a submenu.

Just like this ones



To check whether any element is displayed on the page or hidden, we use isDisplayed() method in WebElement interface. It returns Boolean value.

1. isEnabled():

This method returns TRUE or FALSE based on whether that element is enabled or disabled.

1. Click(): you already know. It clicks on that element.
2. isSelected(): Returns Boolean value based on whether a checkbox or a radio button is selected already or not.
3. Clear(): clears the text field. Usually used when one wants to enter text into field but wants to clear it to delete previously filled data.
4. getTagName(): to get tag name of that element as a String.
5. getText(): this method retuirns the content of that webelement, in string format. Used to read any text value or webtable cell value or link text value.

# Some other Use Cases: Handling Popup

As you may have observed, Cyclos is very generous in throwing popups now and then. So handling popups, verifying messages before accepting or dismissing one is very important. More important than we can emphuftasize because based on the text thrown in the popup, the testcase is perceived as **PASS** or **FAIL**, at least for our test application here.

Let us revisit Framework, src🡪Utils🡪BaseClass.java method validateAndAcceptPopup,

1. 

To simplify, popups are actually different window on top of browser. So if we consider browser as one window, any popup over that can be perceived as another window. So if one wants to switch focus between two windows, *switchTo()* is the method to use here. It switches between windows. Since we want to switch the focus from current browser window to the alert window as and when it appears, we will use above syntax.

This captures the reference to alert window into a reference of Class Alert which is present in org.qa.selenium package in selenium jar that we have associated in the build path.

1. 

For verifying the string message, whether it is same as expected message or not, we have to capture the alert text into a string variable, which is what .gettext() does, for almost all the type of elements in web driver.

1. 

Verifying whether popup message string is equal to expected message string.

1. 

For accepting the popup. (Clicking on OK button in the popup.

1. 

For dismissing the popup. (Clicking on cancel button in the popup.)

We will understand rest of the code in the method later or either you know it already.

# Some other Use Cases: Handling multiple windows

This is not a part of framework as cyclos doesn’t have any such situations. However this is quite common scenario in practice whenever you have more than one windows and you want to switch between them back and forth.

Go through demo4.java to understand this.

<Video-transscript>



1. So sparsh is opened and title is printed on the console.
2. 

Web apps is clicked and title Web Apps is printed on the console.

1. 

Directory system is clicked. New tab is opened and when title is printed, it is still Web Apps.

Which means, driver is still pointing to Web Apps even though Infosys Directory is opened in another tab.

How do we point to Infosys Directory then?

1. 

This method switches the driver focus from Web Apps to Directory System.

How, let us see that in this video.

1. 

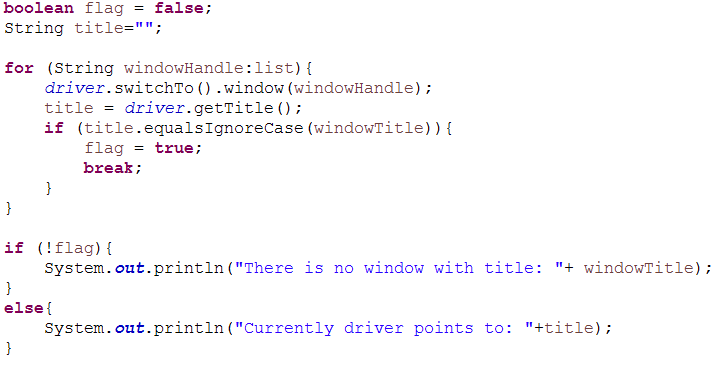
Getwindowhandles() method will return reference pointers to all the window handles to all the tabs or browser instances opened in current run through web driver.

1. 

We are defining an ArrayList object of string.

1. 

We are adding all the strings in that set into this arraylist so that we can iterate through the set. This can also be done using Java Iterator. But here we have used arraylist to do this.

1. 

Flag is for checking whether whatever window title we have passed to the method whether it exists or not within all the open windows.

For each pointer in that arraylist, we are switching window to each browser or tab one by one and if the title matches the title that we want to switch, we set the flag true and break the loop.

If none of the window titles are matching the title we have passed to the method, it will print an error log in the console.

<video-transcript>

# Some other Use Cases: Checking whether page is navigated

1. Suppose after login, you want to check whether login is successful or not. This objective can be achieved by checking whether an element on the page is present or not. Or whether a link which is supposed to be displayed only after successful login (logout link), is present on the page or not.
2. In short, let’s see how to print all the links on the page or check whether a link is present on the page or not.
3. Explain Demo5.java in a video.

# Special Use Cases: Doctored Xpath

Go to src🡪Utils🡪LeftNavigationPane.java file and observe it.

There is a NavigateTo(String, String) method with two arguments, which menu is to be clicked and which submenu is to be clicked.

Let’s understand, how it works.

1. 

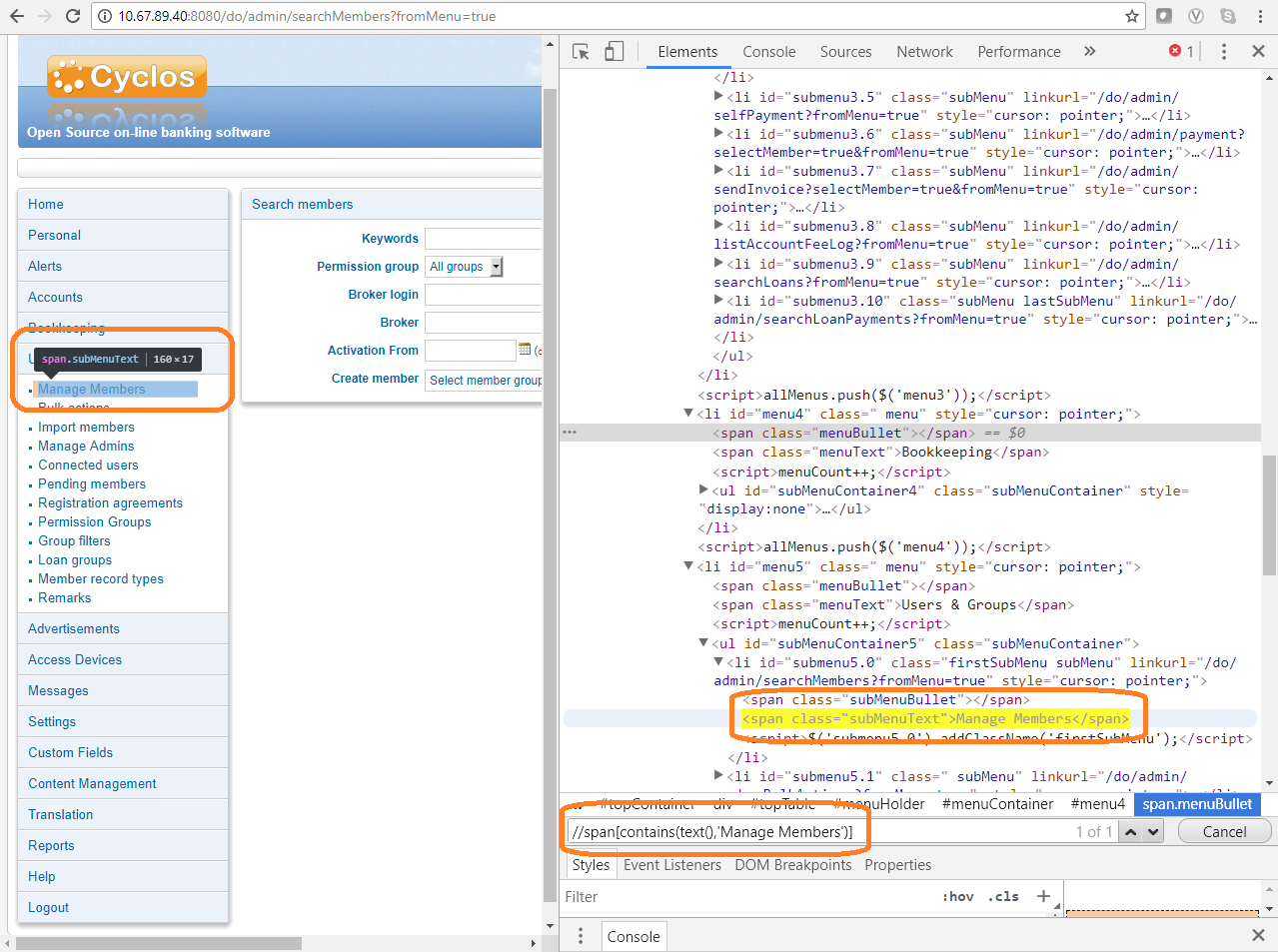
Inspect the submenus in LeftNavigationPane in Cyclos using Chrome Developer tools and you will understand the pattern. This generates a xpath for submenu.

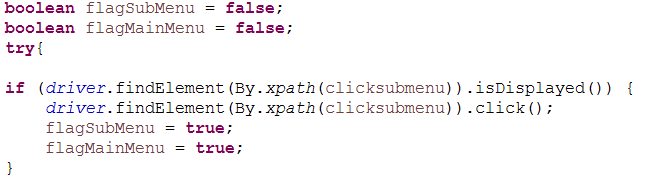
It’s rather advanced method to find something with xpath. It basically means return that element which has span tag and inside the tag, it contains some text with submenu string value which we want to click.

Say for eg: we want to click Manage Members under Users & Groups, above string would become

“//span[contains(text(),’Manage Members’)]”

Is identified correctly in chrome Elements extension under Developer Options.



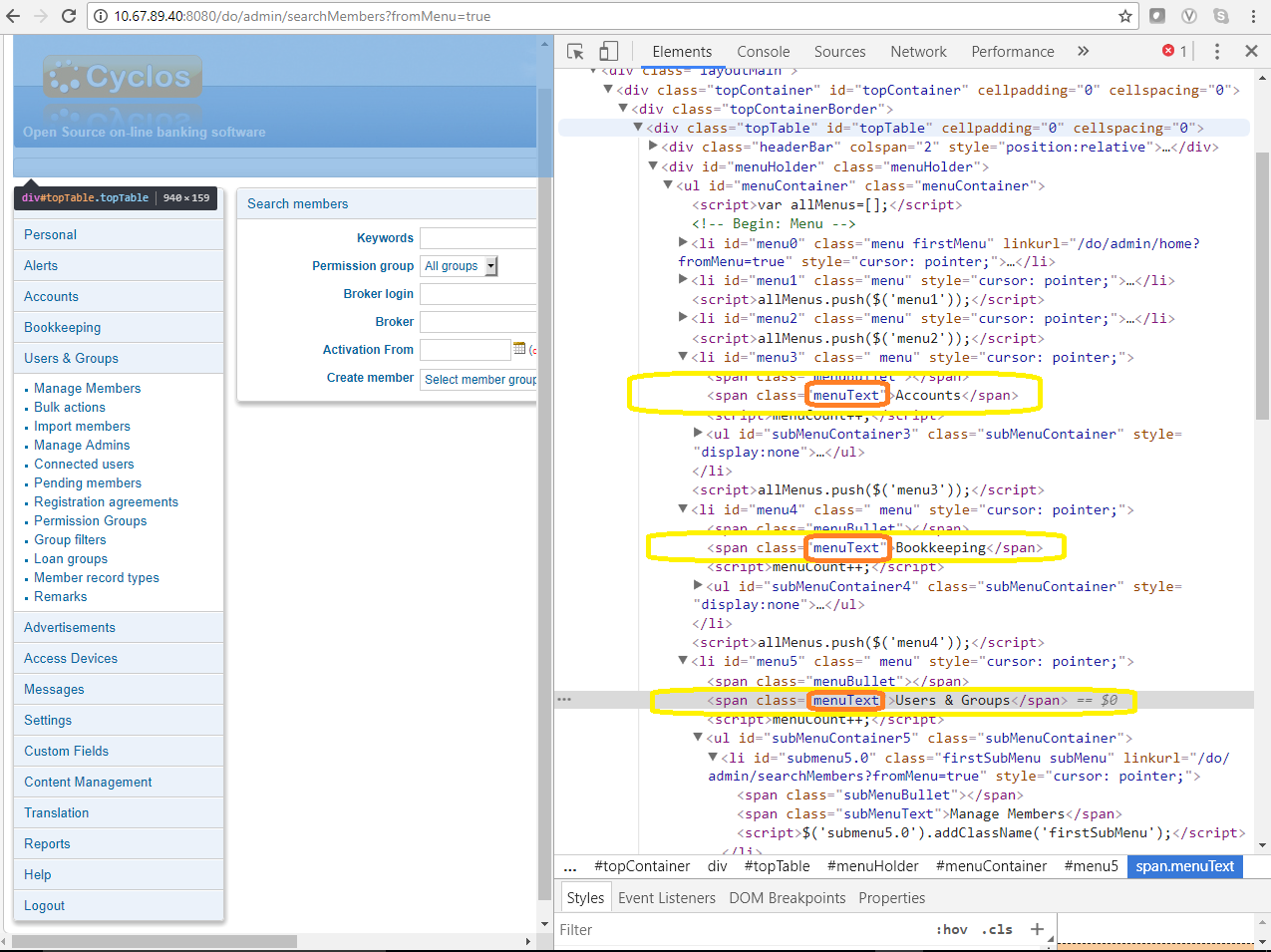
1. 

The purpose of clicking on mainMenu and then on submenu is only when submenu is hidden. But if some sub items in that submenu are already clicked in previous step in the test case, one does not need to click the main menu again as that collapses the menu instead of allowing us to click on submenu.

Hence, we are checking here whether the submenu is already displayed. If yes, then let’s click on it without doing anything else and set both the flags as true as we are checking the flags later in the code for logging and error handling.

1. 

Here, list of all the MainMenu webelements is captured by using xpath. One can understand xpath pattern by observing below image.



1. 

Get number of menu items in an integer variable.

1. 

For each element in the main menu except Home as home doesn’t have any submenus inside it, check if length of the text of mainMenu element is greater or equal to menu text that we have passed to the method.

That is because contains() method of xpath takes partial texts as matches. So suppose we want to click on Users & Groups, text that we pass is Users & Grou, there is no point to check whether Alerts(another main menu item) string is equal to Users & Grou.