# CLASS NOTES: DAY 13-2

Today's schedule:

    - Review of Synchronization  
        - Fluentwait

    - Explicit wait practices  
    - FULL REVIEW OF SELENIUM SECTION INTERVIEW PERSPECTIVE

------------------------------------------------------------------------------------------------------

- SYNCHRONIZATION:

- What is synchronization? Why do we need it?

- Definition: Multiple things working at the same time.  
- We need synchronization because we need to make sure our driver and our browser are on the same page at all times.  
      
#1- Thread.sleep():  
    - This is not coming from Selenium library.  
    - This method comes from JAVA library  
    - It does not wait for ANY CONDITION TO HAPPEN.  
    - It will wait for the given duration no matter what.  
    - Therefore this is not a good practice to use.

#2- implicitlyWait(10);  
    - This is coming from Selenium library.  
      
- Does it wait for any condition to happen?  
- Yes, it is waiting for ONE condition to happen when it is triggered.  
- ImplicitlyWait is ONLY triggered when findElement (@FindBy) method can NOT find a web element with given locator.  
- It will wait until the web element is found or timer runs out.

- By default, timer value is 0. We need to change it to custom number we want to change.  
- We use it once, and it will be applied to the rest of our driver session life.

- What happens when the timer for implicitlyWait runs out?  
- NoSuchElementException is thrown by findElement method.

.  
.  
.  
findElement(LOCATOR); ----> TIMER STARTS  
                                10  
                                9  
                                8  
                                .  
                                .  
                                1  
                                0 ---> NOSUCHELEMENTEXCEPTION IS THROWN IF WEB ELEMENT IS NOT FOUND

- If web element is found, the execution will continue with no problem.  
- Even if the timer has some time left in it, code execution will continue.

- What is POLLING?  
- POLLING is how many times the driver checks the DOM (HTML page) to see if the web element is there (or condition happened) or not.  
- By default, polling happens every 500 ms (or twice in 1 second)

- FluentWait:  
    - Very similar to ExplicitWaits (WebDriverWait).  
    - We can change the polling time.

- ExplicitWait: (WebDriverWait)  
    - Comes from Selenium library.  
    - It can wait for different conditions to happen on the page.  
    - By default it does POLLING every 500ms.  
    - If given timer runs out, TimeOutException.

- How to use explicit waits?

#1- Create object of WebDriverWait class.

    WebDriverWait wait = new WebDriverWait(Driver.getDriver(), 10);

#2- Use the object we just created to create our condition.

    wait.until(ExpectedConditions.visibilityOf());  
    wait.until(ExpectedConditions.invisibilityOf());  
    wait.until(ExpectedConditions.titleIs());  
    wait.until(ExpectedConditions.titleContains());  
    wait.until(ExpectedConditions...);

alertIsPresent()  
elementSelectionStateToBe()  
elementToBeClickable()  
elementToBeSelected()  
frameToBeAvaliableAndSwitchToIt()  
invisibilityOfTheElementLocated()  
invisibilityOfElementWithText()  
presenceOfAllElementsLocatedBy()  
presenceOfElementLocated()  
textToBePresentInElement()  
textToBePresentInElementLocated()  
textToBePresentInElementValue()  
titleIs()  
titleContains()  
visibilityOf()  
visibilityOfAllElements()  
visibilityOfAllElementsLocatedBy()  
visibilityOfElementLocated()

- What is the difference in between ExplicitWait and ImplicitlyWait?  
    - ImplicitlyWait is waiting for ONLY one condition to happen, which is if web element is located or not.   
    - NoSuchElementException will be thrown by findElement() method if web element is not found during the polling of ImplicitlyWait

    - ExplicitWait can wait for many different conditions to happen.  
    - TimeOutException will be thrown if condition does not happen.

====================================================================================================

------------------------------------------------------------------------------------

GIVE ALL ANSWERS AS IF YOU ARE ANSWERING IN THE INTERVIEW:

- WHAT IS SELENIUM?  
    - Selenium is an open source project (library) to automate browsers

-------------------------

- WHY ARE WE USING SELENIUM? WHAT ARE THE ADVANTAGES OF SELENIUM?  
    - Open source --> Free  
    - Supports multiple programming languages  
    - Supports multiple OS (Windows, Mac, Linux)  
    - Supports multiple browsers  
    - It has a major community support behind it

-------------------------

- WHAT ARE SOME OF THE DISADVANTAGES OF SELENIUM?  
    - We cannot automate desktop applications  
    - Requires advance programming language experience/knowledge  
    - No costumer service  
    - There is no built-in report coming from Selenium library

-------------------------

- WHAT IS A WEB ELEMENT?  
    - Everything we see on a web page from links, to images, to input boxes, to checkboxes all of them are web elements.

-------------------------

- WHAT IS THE DIFFERENCE BETWEEN GETTEXT AND GETATTRIBUTE METHOD?  
    - .getText():  
        - It doesn't accept any argument.  
        - It will return the text of the provided WebElement as a String  
        - Return type: String  
    - Where does the .getText() method, gets the text from?  
        - .getText() method can only read in between the <openingTag> and </closingTag>

    - .getAttribute("attributeName"):  
        - It accepts a String argument as an "attributeName".  
        - It will find the matching "attributeName" and return its value as a String.  
        - Return type is String.  
    - Where does the .getAttribute() method, gets the text from?  
        - Gets the text from the opening tag.  
      
-------------------------

- WHAT ARE LOCATORS?  
    - Locators are methods coming from Selenium library that help us locate WebElements.  
    - There are 8 locators  
    - id, name, linkText, partialLinkText, cssSelector, xpath, tagname, className

-------------------------

- WHAT IS YOUR LOCATOR APPROACH? HOW DO YOU DECIDE WHICH LOCATOR TO USE?  
    - If there is id, I make sure it is not dynamic and I use it.  
    - If there is class, I can check if it is unique or not by using "." from cssSelector  
    - If the web element is a LINK, I use "linkText" locator  
    - If none of the above is applicable, I am comfortable creating custom locators using XPATH.

-------------------------

- XPATH

- HOW MANY TYPES OF XPATH ARE THERE?  
- 2 TYPES OF XPATH  
    #1- ABSOLUTE XPATH  
    #2- RELATIVE XPATH

#1- ABSOLUTE XPATH:  
    - Starts with "/" single slash  
    - "/" means start from the root element "html" and go 1 by 1 to the desired web element.  
    - This locator is not stable and will break very easily if there is any minimal change in the html page.  
    - Therefore it is not recommended to use.

#2- RELATIVE XPATH:  
    - Starts with "//" DOUBLE slash  
    - "//" means jump to the web element provided  
    - Relative xpath is more reliable because we are being very specific compared to "absolute xpath"  
   
     - //input[@id='something']

- commonly used xpath locators:  
    //tagName[@attribute='value']  
    //tagName[.='text']  
    //tagName[text()='text']  
    /following-sibling::  
    /preceding-sibling::

-------------------------

- HOW DO YOU HANDLE DYNAMIC WEB ELEMENTS?

- We can use the xpath locator methods such as : contains, starts-with, and ends-with to locate web elements that has dynamic attribute value.

//tagName[contains(@attribute, 'value')]   
//tagName[starts-with(@attribute, 'value')]   
//tagName[ends-with(@attribute, 'value')]

- We can also locate a static (not-changing/unique) parent or child and move from there to desired web element.

-------------------------

- How do we go from child to parent using XPATH?   
- "/.." will take our locator from child to parent

- How do we go from parent to child using XPATH?   
- "/" will take our locator from parent to child

-------------------------

- What is Maven?  
- Maven is a "build automation tool"

- What is a "build"?  
- Repeating steps when we are creating and managing our maven project, such as: creating the folder structure, adding, compiling our code, testing, deploying

- What is the most important file in a Maven project?  
- pom.xml file

- What is pom.xml file and why do you use it?  
- pom -> project object model  
- xml -> extensible mark up language

- Why do we use it?  
- We manage (add, remove, and change versions) of dependencies and plugins etc.

- Where does maven store all of the dependencies?  
- .m2  
- by default it is hidden folder.  
- if you are in a situation where your maven project is not working after adding/removing/changing version of a dependency, and you tried to re-load project, you can go into .m2 folder, and delete everything and reload project.  
- maven will auto-download everything from scratch

-------------------------

- What are the differences in between findElement() and findElements() methods

    - findElement() method:  
    - Return type: WebElement type  
    - It returns a single WebElement  
- What happens if it cannot find a WebElement?  
    - NoSuchElementException will be thrown.

    - findElements() method:  
    - Return type: List<WebElement>  
    - It returns multiple WebElements in a List of WebElement.

- What happens if it cannot find a WebElement?  
    - It will not throw exception.   
    - It will return empty list.

-------------------------

- How do we handle checkboxes and radio buttons?  
- First we locate, then we can click.

- How do we verify if checkbox is selected or not?  
- isSelected() method:  
    - if checkbox/radiobutton is selected, isSelected method will return "true"  
    - if checkbox/radiobutton is NOT selected, isSelected method will return "false"

- isEnabled() method:  
    - if web element is enabled, isEnabled method will return "true"  
    - if web element is NOT enabled, isEnabled method will return "false"

-------------------------

WHAT IS TESTNG?  
    - UNIT TESTING FRAMEWORK.  
    - Originally it was created by a developer for developers.  
    - As testers we are using some of the annotations and methods to create certain structure for our tests.

Why do we use annotations?  
    - Annotations allows us to change the behaviors of regular java methods and allows us to create certain executable flow.

-------------------------

- How do we handle DROPDOWNS?  
- How many types of dropdowns do we have?  
    - 2 types

#1- Non-select dropdowns (HTML):  
    - Just locate with any locator and click.

#2- Select dropdowns:  
    - If a dropdown is created using <select> tag, we can use SELECT class' object and methods coming from it.

Syntax:   
    Select yearDropdown = new Select(dropdown\_as\_WebElement);  
    Select monthDropdown = new Select(dropdown\_as\_WebElement);

- How do we get currently selected option using select object?  
- yearDropdown.getFirstSelectedOption() --> currently selected option as a WebElement  
- What is the return type: WebElement

- How many options do we have for selecting <option> from a dropdown?  
#1- selectByIndex(int) -> accepts index as int, and indexes start from 0  
#2- selectByValue(String) -> accepts the value of attribute "value"  
#3- selectByVisibleText(String) -> accept the text of the option as a String as it is displayed on the page

- How do we get all of the options?  
- yearDropdown.getOption();  
- Return type: List<WebElement>

-------------------------

ALERTS

- HOW MANY TYPES OF ALERTS WE HAVE?  
- We have 2 types of Alert

#1- Non-JavaScript (HTML) Alerts:  
    - How do we handle HTML alerts?  
    - If it is not blocking the page, if you are able to right click and inspect, it is an HTML Alert.  
    - We inspect, locate and click just as any other WebElement.

#2- JavaScript Alerts  
    - How many types we have?  
    - We have 3 types.

    #1- Information  Alert:   
        - User can only .accept();

    #2- Confirmation Alert:   
        - User can .accept(), and decline()

    #3- Prompt Alert:   
        - User can .accept(), decline(), sendKeys();

    - How do you handle Alerts?  
    - Handle JS Alerts using Alert from Selenium.

    Alert alert = driver.switchTo().alert();  
    Alert alert = Driver.getDriver().switchTo().alert();

    alert.accept();  
    alert.dismiss();  
    alert.sendKeys("I can send keys here");

IFRAMES:

- What is iframe?  
- HTML inside of another HTML.

- How do we handle it?  
- We need to locate the iframe and switch to it.

- Why do we have to handle it?  
- Because selenium can focus one thing at a time.   
- By default it will be looking in the main <html> code.   
- If there is any inner <html> we have to explicitly switch to it to be able to do any action in it.  
- Otherwise Selenium will not be able to see any web element from the inner html <iframe>

- How many ways do we have to switch to an iframe?  
- 3

#1- index: indexes start from 0.

    driver.switchTo().frame(int index);

#2- id, name: if there is id or name attribute we can use to locate and switch to iframe.  
      
    driver.switchTo().frame(String id/name);

#3- WebElement: we can locate the iframe as a web element and switch to it.

    driver.switchTo().frame(WebElement);  
    driver.switchTo().frame(driver.findElement(By.locator);

- After switching to inner frame, how do we go back to parent frame?  
    - parentFrame() -> this will switch back to direct parent  
    - defaultContent() -> will switch back to the default <html> of the page

WINDOWS/TABS:

- What is the difference between a window and a tab for selenium?  
- Both are same for selenium.   
- Both TABS and WINDOWS will be treated as WINDOWS.

- How do we handle WINDOWS?  
- We use window handles to handle windows?

- What is a window handle?  
- Randomly generated alphanumeric unique id for each window or tab

driver.switchTo().window(windowHandle);

-------------------------

CONFIGURATION\_READER  
    - prevent hard coding  
    - centralize our important test data  
        - enables us to do cross browser testing  
        - data driven testing

DRIVER  
    1- we were typing too many repeated lines for just instiating our driver instance and do setups.  
    2- we were having hard time passing the same instance around in our project.

    - we created a private constructor, and closed access to the object.  
    - we created a private static WebDriver.  
    - we created a public method which delivers the WebDriver instance in the way we want to deliver.

    - the way we want to deliver:  
        - if session does not exist, it will create a new session/instance  
        - if session already exists, it will return existing session.

    - We achieved this by implementing Singleton Design Pattern.  
        if (driver == null){  
            create new  
        }

        return driver;

ACTIONS:  
    - Actions class handle the "advanced" mouse and keyboard actions, such as doubleclick, drag and drop, right click, click and hold

    - How do we use Actions?  
    #1- Create object of Actions class, we pass the Driver instance into constructor  
    #2- we use the object  
    #3- we use .perform() method

    Actions actions = new Actions();  
    actions.doubleClick(link).perform();

JSEXECUTOR:  
    - What is JavascriptExecutor?  
    - A simple interface with 2 methods coming from Selenium library.

    - How do we use JavascriptExecutor?  
    - We downcast our driver type to JavascriptExecutor, to be able to reach methods in it.

    JavascriptExecutor js = (JavascriptExecutor) Driver.getDriver();

    js.executeScript("scrollIntoView")  
    js.executeScript("scrollBy")  
    js.executeScript("open new tab")  
    js.executeScript("sendkeys")

----------------------------------------------------------------------------

- POM DESIGN PATTERN:  
- WHAT IS POM DESIGN PATTERN?

- Creating .java class for each page of our web application.  
- And store the relevant web elements and methods into their related classes.

- How do we implement POM Design Pattern?

#1- Create a constructor and initialize the object and driver instance using PageFactory.initElements() method.

    public LoginPage(){  
        PageFactory.initElements(Driver.getDriver(), this);  
    }  
      
    Driver.getDriver() --> provides the current instance of our driver.  
    this --> provides the current class' object.

- We can think of this as if we are loading our driver instance INTO our class object, so that our class object is able to call Selenium methods.

#2- Use @FindBy annotation to locate web elements, instead of findElement();

--> StaleElementReferenceException is solved by POM Design pattern. Because every time we try to use the WebELement the reference of the Web Element will be refreshed. Therefore, no more StaleElementReferenceException

- WHY DO WE USE PAGE OBJECT MODEL DESIGN PATTERN?  
- We create centralized repository for our WebElements.  
- REUSABILITY  
- EASY TO MAINTAIN  
- LESS CODE  
- CLEANER CODE  
- EASY TO COLLABORATE IN BETWEEN TEAM MEMBERS

How do you handle web tables?

- We create custom locators with xpath or cssSelector to select any content from the webtable.