CLASS NOTES: DAY 11-2

Today's schedule:

- Review

- Actions practice

- JSExecutor practice

- closeDriver

- Driver utility:

- Why did we create driver utility?

1- We were typing too many lines just to instantiate our browser driver, and also to do setup (maximize windows, implicitly wait etc)

2- We were having hard time passing the same exact instance of our driver around in our project

  - from one test to another

  - from one method to another

  - from one class to another

  - from one package to another

- How did we solve our driver passing issue?

- We implemented Singleton Design Pattern.

- What is a design pattern?

- A repeatable solution to a repeating issue/problem.

- What is Singleton Design Pattern?

- Singleton Design Pattern makes sure we are returning the same instance of our driver every time we call it regardless of where we call it from.

- It can be different package, different class, different method it will always return the same instance.

- How do we implement Singleton Design Pattern?

#1- Create private constructor to close access to the object of the class.

#2- Create a public static getter method to deliver the object in the way we want to deliver.

- Actions:

- Why do we need Actions class?

- To handle "advanced" mouse and keyboard actions.

- Such as:

  - moveToElement

  - clickAndHold

  - dragAndDrop

  - contextClick (right click)

  - doubleClick

  - pause

  - perform

  - keyDown : imitates as if user presses a key from keyboard and holds it down

  - keyUp : imitates as if user lets go (release) a key that is already used by keyDown method.

How do we create and use Actions object?

syntax:

#1- We create Actions class object

#2- We pass the driver instance into Actions class' constructor

#3- Now we can use the object for the methods coming from Actions class.

#4- We MUST use .perform() method at the end to perform our actions.

JavascriptExecutor:

What is it?

- It is a simple interface coming from Selenium library that allows us to inject(pass) JavaScript methods(functions) into our Java-Selenium code.

Why do we need it?

- Because JavaScript is a very strong web-development programming language.

- Therefore it is useful to be able to pass JavaScript code in our Java-Selenium code in certain situations.

How do we use JavascriptExecutor?

1- We need to downcast our driver type to JavascriptExecutor interface

2- Now we can use the methods coming from JavascriptExecutor

3- We pass our Javascript methods into .executeScript method which will apply it in our driver session.

---

session_id for driver : "driver" --> driver_1209381203987askdf34098

driver.get

driver.findELement

driver.maximize

driver.method

driver.quit(); --> driver_1209381203987askdf34098 --> session_id will be deleted, terminated, erased

driver_session_id: driver_123098123019328adsf123

How to handle Closing or Quitting driver with Driver utility class?

--> When we created a new .getDriver() method in Driver utility class and implemeted Singleton design pattern.

--> This design pattern requires my driver to be "driver == null (true)" to be able to generate a new driver.

--> When we use default driver.quit() method that is coming from Selenium library, we terminate the existing driver session completely.

--> This creates issue for our existing structure. Since driver is not null or session is completely deleted/terminated, we cannot continue with our execution of following tests.

--> To solve this issue, we created Driver.closeDriver() method.

--> In this method 2 thing are happening:

#1- We use driver.quit() to terminate the session and close browsers.

#2- Set the driver session value back to "null", so rest of our tests can be executed.

```
/*
    This method will make sure our driver value is always null after using quit() method
*/
    public static void closeDriver(){
        if (driver != null){
            driver.quit(); // this line will terminate the existing session. value will not even be null
            driver = null;
        }
    }
```

---

js.executeScript("arguments[0].scrollIntoView(true)", cydeoLink, homeLink, someOtherLink);

- We are trying to use a JavaScript function (method) which selects the web elements by index number, and scrolls until they are in the view.

js.executeScript(" --> this method accepts and applies the javascript method

arguments[0] --> here we are passing the index number of the argument

.scrollIntoView(true)" --> scroll until the argument[0] is in visible on the screen

, cydeoLink); --> this is where we pass our arguments