CLASS NOTES : DAY 10-2

Today's schedule:

- Review

- Task

- Upload & Download

- Actions

- JavascriptExecutor

- Why do we read data from configuration.properties?

#1- To eliminate hard coding important test data.

#2- We can easily do cross browser testing.

- What is cross browser testing?

- Running the same test against different browsers.

- Just by changing the value of "browser" key from "chrome" to "firefox", our tests will be running in different browser.

saucelabs

#3- We can easily do DATA DRIVEN TESTING?

- What is DATA DRIVEN TESTING?

- Running the same set of tests against different set of data.

- Since we are entering our test data from configuration.properties, we can change the value from there and we will be able to run same tests against different test data.

How do we read from configuration.properties?

#1- Create Properties class object

```
Properties properties = new Properties();
```

#2- We need to open the file in the Java memory

```
FileInputStream fileStream = new FileInputStream("pathOfTheFile");
```

#3- We need to load the "properties" object with the fileStream

```
properties.load(fileStream);
```

#4- Use properties object and getProperty method to pass "key" and read "value"

properties.getProperty("key") --> value

browser -> chrome

username -> helpdesk1@cydeo.com

- Why did we create ConfigurationReader?

- We created the re-usable .getProperty() utility method in ConfigurationReader class

- We can call this method in any class and any package when we need to read from configuration.properties file

---

Driver Utility Review:

- Why did we create Driver utility class and method?

#1- We were having hard time passing the SAME(current) driver instance in different classes and different packages.

#2- We had to write too many lines just to instantiate our "driver"

#3- Now we are not only instantiating our driver in just one line, we are also optimizing the setups.

- determine the type of browser by reading from "configuration.properties" file

- window.maximize is implemented in our Driver util class

- timeOut.implicitlyWait is implemented in our Driver util class

- How are we able to return the same instance of our driver?

- We implemented Singleton Design Pattern.

- What is a design pattern?

- Repetable solution to a repeating issue.

- Singleton Design Pattern:

- Singleton Design Pattern makes sure we are returning the same instance every time we call our method.

- In our case we are trying to return the same instance of DRIVER

How do we implement the Singleton Design Pattern?

#1- Create private constructor.

- Once we create private constructor, we are closing access to the object the class (Driver util)

- We create private variable for our WebDriver to close access from outside.

#2- Create getter method to deliver the object (WebDriver) in the way we want to deliver it.

- The way we deliver it with using Singleton Design Pattern.

  - If first time calling of the method/object, it will create and do setups.

  - Next time we call the same method/object, it will keep returning the same object until the sesion ends or we terminate the session ourselves.

- How do we handle downloads using Selenium?

- We do NOT. We can NOT handle(verify) downloads using only Selenium library.

- We can click to a link on a browser page.

- But once the file is downloaded, it will be out of scope for our selenium.

- How do we handle uploads using Selenium?

- We can upload a file by simply passing the path of the file we want to upload.

- We use sendKeys, and pass the "path" as a String into the WebElement that can upload.

- We just need to find the path of the file and pass it as a String.

Windows:

#1- Right click on the file

#2- Select "properties"

#3- Go to "Security" tab

#4- Get the path from "Object name"

For windows, either double backward "\\" or single forward "/" works.

String path ="C:\\Users\\hayat\\Desktop\\note.txt

Mac:

#1- Right click on the file

#2- Press and hold "option" button from keyboard

#3- Select "copy file as path name" option

#4- Paste into IntelliJ

Actions:

- Actions class is used to handle "advanced" mouse and keyboard actions.
- such as:
  - moveToElement
  - doubleClick
  - dragAndDrop
  - clickAndHold
  - contextClick

JavascriptExecutor:

- This interface allows us to pass JavaScript functions(methods) into our JAVA-Selenium code.
- It has two methods in it.
- executeAsyncScript
- executeScript: this method accepts JavaScript methods as a String and applies them into our code.

P.I.Q:

- How many ways do you know how to scroll?

#1- I can use window.scroll by method with JSExecutor to scroll certain number of pixels.

syntax :JavascriptExecutor.executeScript("window.scrollBy(0,750)")

#2- arguments[0].scrollIntoView --> JavaScript method

#3- PageUp, and PageDown keys using keyboard actions.

#4- We can use the moveToElement() method from Actions class to scroll to certain web element