DAY 9 : CLASS NOTES

Today's schedule:

- Properties review
    - finish the task
- Javafaker
- TestBase
- Driver utility
- Singleton Design Pattern
- Guest speaker

- What is properties file?
- It is just another type of file just like .txt, and .pdf whatever.
- But this file has .properties extention.
- Why do we use properties file? What makes it different then other type of files?
- It stores value in "key=value" format
- We are trying to avoid hard coding some of the important test data in our project.
- What is hard coding?
- Writing data directly inside of the source code is called hard coding.
- If I have to go inside of my .java class to change the data, it means I hard coded that data.
- How do we read from properties type of file?

#1- Create Properties class' object.

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

#2- Create FileInputStream object to open file as a stream in Java memory.

```
FileInputStream file = new FileInputStream("pathOfTheFileWeAreTryingToOpen");
```

#3- Load "properties" object with the "file" we opened using FileInputStream

```
properties.load(file);
```

#4- We can use properties.getProperty method to read from the file we loaded.

(configuration.properties)

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

browser ----> chrome

env ----> qa1.vytrack.com

username ----> tester5@cydeo.com

- Which part of this is hard coded in our code : key=value
- "key" is what we write in our .java class.
- Therefore "key" will not change, and is hard coded.
- "value" is inside of our configuration.properties file.
- We can change the value from outside of our code

- What is ConfigurationReader? Why did we create this?
- To create utility method and be able to read from configuration.properties file by just calling our ConfigurationReader.getProperty("key"); method

----

JAVAFAKER

- JavaFaker is a library that allows us to generate random data in organized manner
    - name
    - address
    - finance
    - numerify: random digits in the order we want to get
    - bothify : random alphanumeric in the order we want to get

```
Faker faker = new Faker();

String name = faker.name().fullName(); // Miss Samanta Schmidt

String firstName = faker.name().firstName(); // Emory
```

String lastName = faker.name().lastName(); // Barton

String streetAddress = faker.address().streetAddress(); // 60018 Sawayn Brooks Suite 449

TestBase - BaseTest

- What is TestBase?

- TestBase is an abstract class where we create and store some re-usable methods/annotations, objects, and also variables if needed.

- TestBase is not a utility class/method.

- TestBase will store some commonly used steps in our tests.

- Ideally you want to make TestBase abstract, because an object cannot be created from an abstract class.

- Is it mandatory to make it abstract? No.

- When we want to add any logic to TestBase, we need to make sure it is applicable to all of the tests.

- If we add a line that is not applicable to all of the Tests, it might create challenges rather than solving them.

  - What is the topic? What are we doing?

  - Why are we learning?

    - What kind of issue we are solving with this?

    - What are we making easy by using this?

  - How are we using it?

  - How are we implementing it into our code (framework)?

DRIVER UTILITY CLASS

- What, Why, How

- What is the topic?

- We are going to be creating a new utility class: Driver

- Why are we creating this class?

1- We are writing too many lines just to be able to instantiate our WebDriver

2- We are having hard time to pass the SAME 'driver' instance around in our project.

    - When we are using any utilty method, currently we have to pass "WebDriver driver" as argument in each utility method.

    session_id: driver_asd9a8sdf79a8s7df

    driver.quit

    session_id: driver_asd9a8sdf79a8s7df

SOLUTION:

- Driver utils class and new .getDriver() method we will be creating.

- We will create a new logic which will guarantee the same exact instance every time we call the method.

- It will also handle the lines where we maximize the page, and implict wait etc.

- HOW?

    - We will use a "design pattern".

- What is a design pattern?

- A design pattern is a general repeatable solution to a commonly occurring problem in software design.

- We will use "Singleton Design pattern"

- What is Singleton Design Pattern?

- Singleton Design Pattern guarantuees to return same object everytime we want to use the object.

- How do we apply Singleton Design Pattern?

#1- We create private constructor

#2- We create getter method to deliver the object in the way we want to deliver.

    - in the utility method, we will create the logic below.

        - if object == null, create new object and return it.

        - if object is not null, just return existing object.