CLASS NOTES : DAY 19-2

Today's schedule:

- Review
    - rerun
    - plugin vs dependency
    - maven cucumber reporting
    - parallel testing
- Run your project with maven command line/s
- Interview questions
- Tell me about your framework

- Cucumber Rerun:

- What is Cucumber rerun used for?
    - To run the failed tests only

- Why would we want to run only the failed tests?
    - Essentially just to save some time to see if there is something wrong with the failed tests specifically.

- How do we implement Cucumber rerun?

#1- Add rerun plugin into our main TestRunner class

syntax: "rerun:target/rerun.txt"

rerun: plugin name

target : where we want this .txt file to be generated

/rerun.txt : name the file "rerun" and generate a ".txt" file

#2- Create a new Runner class that will be only focusing to run FAILED TESTS.

#1- Create a new Runner class, ideally named : FailedTestRunner

#2- We add glue path for step_definitions same as CukesRunner

#3- BUT, as feature file path; we only provide the path the rerun.txt that was generated by our "rerun plugin"

This class will only be able to run the "failed tests" that are stored in this file, if any.

- MAVEN CUCUMBER REPORTING

- Is this report mandatory to have?

- No, this is optional.

- This is basically just another type of reporting that allows us to generate "pretty" reports.

- Currently we are generating this using a dependency, but we can also generate exactly the same report using a plugin.

- What is the difference in between plugin and dependency?

- BASICALLY THEY ARE BOTH SOME JAR FILES.

- BUT, the difference is the DEPENDENCIES are not involved in the MAVEN LIFECYCLES.

- PLUGINS are involved in maven life cycles.

- What does this mean?

- It means we can pass additional configurations under our plugins, and when we run our different maven lifecycles, the configurations will be executed.

- How did you implement PARALLEL TESTING in your project?

- How did you implement PARALLEL TESTING while you still have SINGLETON DESIGN PATTERN?

#1- WE ADD "MAVEN SUREFIRE PLUGIN" WITH ADDITIONAL CONFIGURATIONS THAT ENABLES PARALLEL TESTING.

```
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>          --> THE COMPANY NAME
            <artifactId>maven-surefire-plugin</artifactId>       --> PROJECT NAME
            <version>3.0.0-M5</version>                          --> VERSION OF THE PLUGIN
            <configuration>                                      --> We provide addition configurations
               <parallel>methods</parallel>                      --> We run .feature files in parallel
               <threadCount>4</threadCount>                      --> How many threads we want to generate
               <testFailureIgnore>true</testFailureIgnore> --> Will not stop if a test fails
               <includes>
                  <include>**/CukesRunner*.java</include> --> when we use "mvn test" lifecycle, it will find the .java class that has the name contains "CukesRunner"
               </includes>
            </configuration>
        </plugin>
```

#2- We adjust our Driver Singleton logic to be able to handle multi-threads.

How do we change our Driver to handle multi-threads?

#1- Driver.getDriver() method name stays same.

#2- We wrap our "driver" instance with InheritableThreadLocal

```
    private static InheritableThreadLocal<WebDriver> driverPool = new InheritableThreadLocal<>();
```

#3- Now that we are using "InheritableThreadLocal", we adjust our code in .getDriver() method to use methods coming from "InheritableThreadLocal"

Instead of using : "driver" we use "driverPool.get()"

Instead of using : driver = new ChromeDriver(); we use --> driverPool.set(new ChromeDriver());

Instead of using : driver == null --> driverPool.remove();

- We didn't touch the Singleton Design Pattern logic we previously created.

```
if(driver == null){
    create new
}
return driver;
```

- We still have Singleton Design Pattern.

- Even if we run our features in parallel, every thread using WebDriver instance will be Singleton in itself.

WHY DO WE NEED PARALLEL TESTING?

- 100 TESTS, 1 TEST = 1 MIN, 1 MACHINE : TOTAL DURATION OF TEST RUNNING TIME --> 100 MINS

- 100 TESTS, 1 TEST = 1 MIN, 2 MACHINE : TOTAL DURATION OF TEST RUNNING TIME --> 50 MINS

- 100 TESTS, 1 TEST = 1 MIN, 4 MACHINE : TOTAL DURATION OF TEST RUNNING TIME --> 25 MINS

- 100 TESTS, 1 TEST = 1 MIN, 10 MACHINE : TOTAL DURATION OF TEST RUNNING TIME --> 10 MINS

- When we kick off our mvn lifecycle, it will execute the code we in our Runner class.

- If we have specific tag in there, and this tag is in only a few of our feature files, it will only run those feature files in parallel.

- "pretty" plugin:

- basically it just prints out additional information about the scenario that is being executed.

---

publish = true;

--> will enable a functionality from cucumber to generate a public link for the report of our final execution of our code

--> the link will be automatically generated and printed in the console

--> it will be ready to share with anyone

We can run our project using some maven commands from outside of our intelliJ using the line below:

mvn test -Dcucumber.filter.tags="@smoke"

- Why do we need this?

- To be able to run our code from outside of our intelliJ, possibly from Jenkins or from command line.

#1- dryRun :

- to turn and off our step_definition implementations running or not so we can easily generate snippet.

- if true: turned on. and will not execute the actual code (java-selenium-junit)

- if false: turned off. and will execute the actual code (java-selenium-junit)

#2- tags:

- we can create different scenario suites where we can include or exclude different scenarios.

- or, and, and not.

- or : @a or @b --> scenario will be executed if it has either one of the tags.

  --> similar to || in java

- and : @a and @b --> scenario will be executed if it has BOTH of the tags.

  --> similar to && in java

- and not : @a and @b and not @c

  - if a scenario has @c tag, it will NOT be executed.

#3- pluginS :

- inside of the plugin, we determine

  - the type of the reportS

  - where do we want to store our reportS

  - the name of the report file

plugins{

#1- html:target/cucumber-report.html --> this generates default cucumber html report

#2- "rerun:target/rerun.txt", --> this generates rerun.txt for keep tracking of failed tests

#3- "me.jvt.cucumber.report.PrettyReports:target/cucumber",

-> this plugin generates the "MAVEN CUCUMBER REPORTING"

#4- "pretty" --> used to print out additional information about the scenario currently executed

}

#4- Background:

- Background is used to re-use the steps that are common in the same FEATURE FILE.

- The steps we pass under background is only effective for the scenarios in the same feature file.

#5- Hooks:

- Hooks are used to create pre- and post- condition for ALL scenarios in our project.

- Since Hooks effect the whole project, we should be very selective on what to put in the hooks.

#6- Parameterization

- What is Parameterization?

- Parameterization is being able to pass and change data directly from feature file.

- We don't have to go in the step definitions to change the data when we use Parameterization.

- What is the difference between Parameterization and datatables?

#7- dataTables

- We can pass COLLECTIONS TYPE OF DATA under the same STEP.
    - LIST
    - MAP
    - LIST OF MAP
    - MAP OF MAP

- What is the difference between Parameterization and Scenario outlines?

#8- scenario outlines

- Scenario outlines allows us to simplify the DDT (Data Driven Testing) by creating "Example:" tables under our "Scenario Outline:""

- We don't have to come to feature file to change the data everytime we run. We only provide once, and our scenario will be executed against all of this data.

- Whereas, in parameterization, we would have to come back to change all the data just to run the same scenario with another test data.

Tell me about yourself?

Tell me about your project?

Can you explain your project?

Can you explain your meeting ceremony? What do you do as a tester in meetings?

Can you explain your daily activity?

What are your responsibilities?

What is your testing approach?

How do you decide which test will be automated?

- make sure functionality is working

- is it a repeating scenario

- is it high priority scenario (business decision)

- is it complex scenario and also repeating

- do not automate flaky (unstable) tests.

What are the steps while you are writing your test scenario?

Can you give an example how you test in manual?

What is the difference between Test NG and JUnit? Can you give an example about annotations?

Why do you prefer Junit as a company? Not Test NG?

How do you use the OOP Concept in your framework?

Encapsulation: Driver

- make private WebDriver

- create getter

Polymorphism: WebDriver --> ChromeDriver

- Inheritance/Abstract

- TestBase

- BasePage

How do you locate dropdowns?

How do you handle pop ups?

How do you locate your dynamic web elements?

How do you use javascript in your framework?

How do you use Java while you are writing test scenarios?

What is the difference between keyword driven and data driven testing?

Can you explain how you set up parallel testing?

#1

#2

What is the differences between @findby and pageFactory?

What is the main goal of the Singleton Design pattern?

Can you explain the mentality of the POM?

Which version of Selenium do you use?

selenium version 3.141.59

How do you use single slash (/) while you are finding locators?

What is the difference between CSS and Xpath?

How do you prioritise your scenarios?

How do you run your failed runner/tests?

How do you use external data? Can you explain step by step?

- configuration.properties --> ConfigurationReader

- Feature files : Scenario Outlines

- EXCEL : using APACHE POI

Can you explain BDD steps?

---

tell me about your framework

- I used java as my programming language in my framework.

- I used Selenium to automate my browsers.

- I used maven as my build automation tool. which has pom.xml file that allows me to manage my dependencies/versions easily.

- I used Page Object Model to simplify managing and maintaing my framework for myself and my team.

- this design pattern allows me to locate web elements only once, in their respective classes.

- so that if there are any problems with any web elements, I know exactly where to go and how to fix it.

- I created Singleton Design Pattern to allow my framework to pass the same instance of my webdriver in one session.

- (one session: when you click run, selenium creates one session. the session will end when the driver stops.)

- I created a configuration.PROPERTIES type of file where I keep the important test data about my framework. I keep Test data that can change the running flow of the whole framework, such as:

- browser

- username/password

- url: to change and run on different environments

- I created utility class from existing java library to read from properties type of file.

  - ConfigurationReader

- you should be ready to talk about how to read from properties file.

- opening the file and passing the path of the file into FileInputStream

- loading the file to properties class object.

- I implement BDD approach to simplify reading and understanding my framework for the non-technical teammates in my team.

- I write my test cases as if they are scenarios from the end users perspective in Gherkin language in my feature files.

- Gherkin is very similar to English. THerefore it is easy to understand for non-technical teammates.

- I implemented the actual coding logic with JAVA-SELENIUM-JUNIT... inside of my step_definitions package and it their own respective/related classes

- I trigger my framework from my runner class.

- Runner class allows me to run different types of testing suites that I created with my tags, such as smoke, regression, mini-regression.

- I have different types of reports. But mainly I use "maven-cucumber-reporting" which is a very detailed reporting tool that has pie-charts, matrixes on which tests are passing and failing.

- Even has the option to show what percentage of which tags are failing and passing.

- Hooks class, where we implement some cucumber annotations such as Before, After, beforestep, afterstep to create outline for my scenarios.

- I also implemented a logic where my framework is taking a screenshot and attaching it to my report if a scenario is failing.

- DDT - SCENARIO OUTLINE

- APACHE POI

- SQL/JBDC

- API