Project Preparation - Part 1 - Add dependencies

1- Create new maven project (demonopcommerce)

This is the website that you're supposed to perform your automation scenarios using Cucumber Framework

https://demo.nopcommerce.com/

2- check cucumber plugins are already installed: File >> Settings >> Plugins

under marketplace tab: search for **"cucumber for java"** and make sure it's already installed

3- Add essential required dependencies

Search google on: Selenium maven dependency

https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java

Search google on: Cucumber java maven dependency

https://mvnrepository.com/artifact/io.cucumber/cucumber-java

Search google on: Cucumber TestNG maven dependency

https://mvnrepository.com/artifact/io.cucumber/cucumber-testng

Note: No need to use TestNG dependency from this link anymore

https://mvnrepository.com/artifact/org.testng/testng

4- Create your first feature file then run as a test to make sure that everything is working fine before start coding your project

4.1- under **"main/resources"** create directory and give it any name like >> features

(you could choose any other name)

4.2- under **"features"** directory create feature file and give it any name like **"F01_Register.feature"**

Note: the extension should be **".feature"**

Take care, it's case sensitive so it's wrong to write it like .features or .Feature etc

4.3- in **"F01_Register.feature"** you could write the following

@smoke

Feature: F01_Register | users could register with new accounts

Scenario: guest user could register with valid data successfully

Given user go to register page

5- We will do this step now to guarantee that cucumber report will be generated properly at the end of the project

5.1- under **"test/java"** you need to create package **"org.example"**

Where this name come from? open your pom.xml file and observe the name inside this tag <groupId>

It's almost **"org.example"** > that's why we will use this name while creating the package

Note: if <groupId> name is something else so you should use this name to create the package

5.2- under **"test/java/org.example"** you need to create package and give it any name like **"stepDefs"**

(you could choose any other name)

Note: this step is a little confused because the package may be changed to "org.example.stepDefs" but no need to worry about this, just continue

5.3- Right click again on **"org.example.stepDefs"** and create another package

this time, package name will be >>   org.example.pages   (not org.example.stepDefs.pages)

So expected now the structure will be like this

- Java
  - org.example
    - stepDefs
    - pages

You should make sure that your structure is similar like this

If no, so please remove org.example and packages inside and repeat step 4 from the beginning again

5.4- Right click again on **"org.example"** and create another package

this time, package name will be >> org.example.testRunner (not org.example.stepDefs.testRunner)

So expected now the structure will be like this

- java
  - org.example
    - pages
    - stepDefs
    - testRunner

As we mentioned in the beginning of this step, we are doing all of this to guarantee that the cucumber report will be generated correctly at the end

6- Create relative step Definition class for Registration feature

6.1- under **"test/java/org.example/stepDefs"**

you need to create class and give it any name like "D01_registerStepDef"

(you could choose any other name)

6.2- in **"D01_registerStepDef"** class which is under **"stepDefs"** package

you need to create a method for Given step and write simple code inside as a test

Remember: you should import Given from cucumber before creating this method

```
import io.cucumber.java.en.Given;

        @Given("user go to register page")
   public void goRegisterPage()
   {
      System.out.println("This is a test before start coding");
   }
```

Run your code at this point from the feature file "main/resources/features/F01_Register.feature" to make sure that everything is implemented correctly till now.

If you have any problem, please don't proceed and share your problem on discourse

expected to see the line is printed in the console like this

```
This is a test before start coding

1 Scenarios (1 passed)

1 Steps (1 passed)
```

Reviewers will check the following

1- Screenshot from your environment variables

Open Start menu and search on "Edit the system environment variables" then go to "path" and take a screenshot that proves you already added jdk path inside

2- IntelliJ plugins

in this screenshot, reviewers should make sure that the student installed

"Cucumber for Java", "Gherkin" plugins

Project Preparation - Part 2 - Hooks

1- Start coding the Hooks

1.1- under "test/java/org.example/stepDefs"

you need to create your first class which is "Hooks"

what's Hooks?

https://cucumber.io/docs/cucumber/api/#:~:text=fix%20the%20ambiguity.,Hooks,steps%20it%20is%20run%20for.

Note: We don't create feature file for Hooks class

1.2- Define your public static WebDriver driver = null;

and set your @Before@After

```
// your code inside the class will be like this
import io.cucumber.java.After;
import io.cucumber.java.Before;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import java.util.concurrent.TimeUnit;
public class Hooks {

public static WebDriver driver;

@Before
public static void OpenBrowser()
   {
 // 1- Bridge
 System.setProperty("webdriver.chrome.driver","C:\\Program Files\\chromedriver.exe");

 //Note: you could use WebDriverManager.chromedriver().setup() instead of System.setProperty() as explained in the
videos


 // 2- create object from chrome browser
 driver = new ChromeDriver();

 //3- Configurations
driver.manage().window().maximize();
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

// 4- navigate to url
driver.get("https://demo.nopcommerce.com/");
                          }
@After
public static void quitDriver() throws InterruptedException {
Thread.sleep(3000);
driver.quit();
  }
                        }
```

Remember what is the function of implicitlyWait?

driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

- The Implicit Wait in Selenium is used to tell the web driver to wait for a certain amount of time as long as the element still not present in DOM Page

- The mechanism is to check if the element is present in DOM Page every 0.5 second, if the element is present in DOM Page then the code will proceed without having to wait the whole timeout assigned

for example, if the element is present in DOM Page after 2 seconds, the code won't have to wait all the 7 seconds

- If the element still not exist in DOM page after the timeout using (findElement not findElements) then the code will throw a “No Such Element Exception” error and the test case will be failed.

In case of using findElements >> No error will be thrown and output will be empty which means size() = zero

- It's not preferred to use large number like

driver.manage().timeouts().implicitlyWait(100, TimeUnit.SECONDS);

because if any element is not exist in DOM page for any out of hand reason >> that means your test case will wait 100 seconds before it fails

- Beside ImplicitlyWait there's also another thing in Selenium called explicitlyWait which will be discussed later in the project. Each one has its own usage and we are using both in Automation.

Keep in Mind: implicitlyWait has only one condition to check which is waiting until the element is exist in DOM Page

but explicitlyWait has multiple conditions unlike implicitlyWait, For example

wait unitl element is visible on UI Page

wait until element is clickable on UI Page

wait until the url of the browser to be...

wait until number of browser tabs to equal...

1.3- in "**D01_registerStepDef**" class which is under "**stepDefs**" package

you need to Call Hooks.driver and do your first action which is click on register button

```
@Given("user go to register page")
   public void goRegisterPage()
   {
//  System.out.println("This is a test before start coding");
WebElement registerBtn =
Hooks.driver.findElement(By.cssSelector("a[href=\"/register?returnUrl=%2F\"]"));
registerBtn.click();
// Note: you will need to apply pom design pattern later on, but now let's focus first on running
your cucumber scenarios
  }
```

Run your code at this point from the feature file "main/resources/features/F01_Register.feature" to make sure that everything is implemented correctly till now

2- Apply POM Design pattern

under "test/java/org.example/pages" you need to create class for register and give it any name "P01_register"

```
// your code inside the class will be like this

import com.automation.step_definitions.Hooks;

import org.openqa.selenium.By;

import org.openqa.selenium.WebElement;


public class P01_register {

   public WebElement registerlink()

   {

   return  Hooks.driver.findElement(By.className("ico-register"));

   }

}
```

2.1- Go back to "D01_registerStepDef" class which is under "stepDefs" package to apply design pattern.

```
import org.example.pages.P01_register;

import io.cucumber.java.en.Given;

import org.openqa.selenium.By;

import org.openqa.selenium.WebElement;

public class D01_registerStepDef {

P01_register register = new P01_register();

            @Given("user go to register page")

   public void goRegisterPage()

   {

          register.registerlink().click();

   }

          }
```

Run your code at this point from the feature file "main/resources/features/F01_Register.feature" to make sure that everything is implemented correctly till now

If you have any problem, please don't proceed and share your problem on discourse

Reviewers will check the following

1- Student already added cucumber annotations

Note: some students mix it up and use testNG annotations (@BeforeMethod, @AfterMethod)instead of cucumber annotations (@Before, @After)

So please make sure that you're using cucumber annotations

io.cucumber.java.Before;

io.cucumber.java.After;

2- Student setup the bridge in @Before using one of two methods

System.setProperty() or WebDriverManager.chromedriver().setup()

3- Implicit wait is added inside @Before like this

driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

4- pom design pattern is applied correctly

Note: your project will be rejected without applying pom design pattern

Project Preparation - Part 3 - testRunner

1- Create testRunner

1.1- under "**test/java/org.example/testRunner**" package, you need to create class and give it any name like "**runners**" (you could choose any other name)

1.2- what distinguish this class is that you will write your code out of the class like this

```
import io.cucumber.testng.AbstractTestNGCucumberTests;
import io.cucumber.testng.CucumberOptions;


        @CucumberOptions(
    features = "src/main/resources/features",
    glue =    {"org.example.stepDefs"},
    plugin = {         "pretty",
        "html:target/cucumber.html",
        "json:target/cucumber.json",
        "junit:target/cukes.xml",
        "rerun:target/rerun.txt"},
    tags = "@smoke"
        )


        public class runners extends AbstractTestNGCucumberTests {
        }
```

Notes

- as you see, @CucumberOptions is written above the class not inside it

- easy way to get this path "src/main/resources/features" >> Right click on "stepDefs" package > select "Copy Path/Reference" > select "Path From Content Root"

- tags value could be anything else instead of "@smoke", it's not a must to give it this name

2- Before running your project from "runners" you should go to your feature file and make sure @smoke is already added as mentioned before

```
@smoke

Feature: F01_Register | users could register with new accounts

Scenario: user could register with valid account

Given user go to register page
```

Run your code at this point from "runners" class (not from feature file this time) and make sure that everything is going well

If you have any problem, please don't proceed and share your problem on discourse

Reviewers will check the following

1- Student should create runners class in the correct place (under testRunner package)

2- Student adds the required CucumberOptions (features, glue, plugin, tags)

Project Preparation - Part 4 - Prepare Professional Report

1- Go to pom.xml file

1.1- make sure the following tags are exist in pom xml file

```
            <modelVersion>4.0.0</modelVersion>

            <groupId>org.example</groupId>

            <artifactId>untitled</artifactId>     <!-- This is the name of your project -->

            <version>1.0-SNAPSHOT</version>
```

1.2- add the following inside pom.xml file

```
<build>

  <plugins>

  <!-- you will add your plugins here in the following steps -->

  </plugins>

 </build>
```

It's important >> Don't add it inside <dependencies>   </dependencies>

1.3- First plugin should be added inside <plugins> </plugins>

```
<plugin>

          <!--  https://maven.apache.org/surefire/maven-surefire-plugin/plugins.html -->

        <groupId>org.apache.maven.plugins</groupId>

        <artifactId>maven-surefire-plugin</artifactId>

        <version>3.0.0-M4</version>

        <configuration>

          <testFailureIgnore>true</testFailureIgnore>

          <runOrder>Alphabetical</runOrder>

          <includes>

            <include>**/*runners.java</include>       <!-- this should be the name of your runner class -->

          </includes>

        </configuration>

 </plugin>
```

1.4- Second plugin should be added inside <plugins> </plugins>

```
<plugin>
  <!-- https://repo.maven.apache.org/maven2/net/masterthought/cucumber-reporting/ -->
  <groupId>net.masterthought</groupId>
  <artifactId>maven-cucumber-reporting</artifactId>
  <version>5.7.0</version>
   <executions>
    <execution>
        <id>execution</id>
        <phase>verify</phase>
        <goals>
          <goal>generate</goal>
        </goals>
        <configuration>
          <projectName>Cucumber HTML Reports</projectName>
          <outputDirectory>${project.build.directory}</outputDirectory>
          <inputDirectory>${project.build.directory}</inputDirectory>
          <jsonFiles>
            <param>**/cucumber*.json</param>
          </jsonFiles>
        </configuration>
      </execution>
    </executions>
  </plugin>
```

2- How to run your project test cases from maven to generate the report

2.1- click on maven tab on the top right of intelliJ

2.2- expand project name then expand "Lifecycle"

2.3- double click on "clean" to remove "target" folder if it was found before

2.4- double click on "verify" and wait for the execution to be finished

2.5- after execution, new "target" folder will be created including the test execution report

2.6- open "target" folder > "cucumber-html-reports" folder > "overview-features.html" file > open this file in HTML Browser

Reviewers will check the following

1- first and second plugin should be added inside <plugins> </plugins>

2- class runner name under first plugin should be the same class name under "test/java/org.example/testRunner"