

1. Cucumber Setup: Maven, Extent Reports, Cucumber and Eclipse Plugin

Step 1.1: Configuring Cucumber Eclipse

Cucumber jar files are already present in your practice lab. Refer to the lab guide for Phase 2 for more information.

- Launch Eclipse.
- Create Java Project.
- Go to Java Project and create a folder “jars.”
- Add all the downloaded jars to the “Jars” folder.
- Select the Created Project → Right Click → Build path → Configure Build path → Click on Libraries → Add JARs → Select all the jars from “Jars” folder → Click on Apply → Add Library → Select TestNG → Click on Apply → Click on Finish.

Step 1.2: Configuring Cucumber with Maven Project

- Create a Maven Project.
 - ✧ Go to File → New → Others → Maven → Maven Project → Next.
 - ✧ Provide group ID (group ID will identify your project uniquely across all projects).
 - ✧ Provide artifact ID (artifact ID is the name of the jar without version. You can choose any name, which is in lowercase). Click on Finish.
- Open pom.xml.
 - ✧ Go to the package explorer on the left-hand side of Eclipse.

- ✧ Expand the created project.
- ✧ Locate the pom.xml file.
- ✧ Right-click and select the option, open with "Text Editor."
- Adding dependencies to the project: This will indicate Maven about the jar files to be downloaded from the central repository to the local repository.
 - ✧ Open the pom.xml in the edit mode, create dependencies tag (<dependencies></dependencies>), inside the project tag.
 - ✧ Inside the dependencies tag, create dependency tag (<dependency></dependency>).
 - ✧ Copy the dependency tag for the following from Maven Repository.
 - ✧ Selenium Webdriver
 - ✧ Cucumber-Core jar
 - ✧ Cucumber-Java jar
 - ✧ Cucumber-TestNG jar
 - ✧ Provide the information of all copied dependencies within the dependency tag.
- Verify binaries.
 - ✧ Once the pom.xml is edited successfully, save it.
 - ✧ Go to Project → Clean. After a few minutes, you will be able to see a Maven repository.

Step 1.3: Configuring Cucumber Extent Report

- Add below Cucumber Extent Report library to Maven project.


```
<dependency>
<groupId>com.aventstack</groupId>
<artifactId>extentreports</artifactId>
```

```
<version>3.0.6</version>
```

```
</dependency>
```

```
<dependency>
```

```
<groupId>com.vimalselvam</groupId>
```

```
<artifactId>cucumber-extentsreport</artifactId>
```

```
<version>3.0.2</version>
```

```
</dependency>
```

- Add Extent Config to the project.

Extent Config is required by the Cucumber Extent Report plugin to read the report configuration. This gives the capability to set many useful settings to the report from the *XML* configuration file.

- Create a **New File** and name it as **extent-config.xml** by right clicking on the **configs** folder in the project.
- Read the extent-config.xml path.

✧ Make an entry for the path of the config in the **Configuration.properties** file.

reportConfigPath=C:/ToolsQA/CucumberFramework/configs/extent-config.xml

✧ Write a method **getReportConfigPath()** in the **ConfigFileReader** class to return the extent report config file path.

```
public String getReportConfigPath(){  
  
    String reportConfigPath = properties.getProperty("reportConfigPath");  
  
    if(reportConfigPath!= null) return reportConfigPath;  
  
    else throw new RuntimeException("Report Config Path not specified in the  
    Configuration.properties file for the Key:reportConfigPath");  
  
}
```

- Modify TestRunner to Implement Cucumber Extent Reporter

- ✧ Modify the runner class and add `com.cucumber.listener.ExtentCucumberFormatter:output/report.html` as a plugin followed by the report file as input. This should be done within the `@CucumberOptions` annotation.

```
@CucumberOptions( plugin =  
{"com.cucumber.listener.ExtentCucumberFormatter:target/cucumber-  
reports/report.html"})
```

The above setup will generate the report in the output directory with the name report.html.

- Write extent reports

Add a method `writeExtentReport()` in the `TestRunner` class to write the report.

```
@AfterClass
```

```
public static void writeExtentReport() {
```

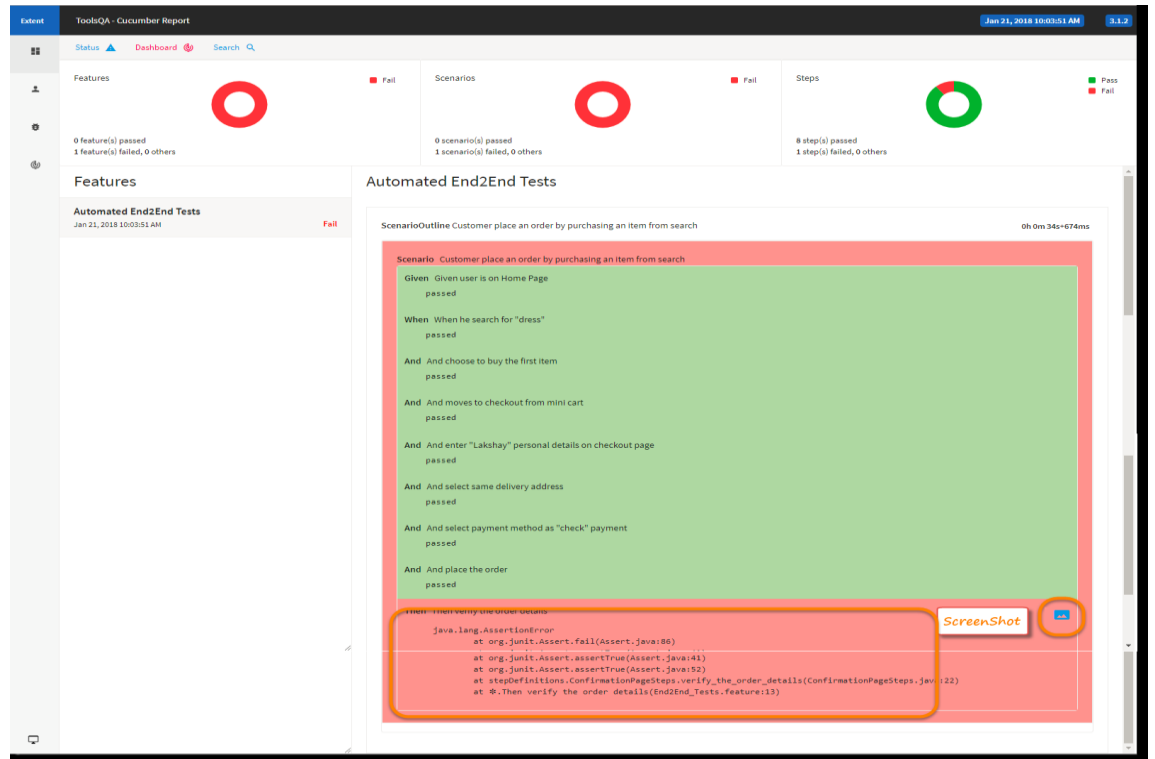
```
Reporter.loadXMLConfig(new
```

```
File(FileReaderManager.getInstance().getConfigReader().getReportConfigPath()))
```

```
;
```

```
}
```

- Sample extent report



2.Gherkin Keywords

Step 2.1: Demonstrating Gherkin keywords with example.

The following are the keywords used in Gherkin:

- **Feature:** This gives information on the high-level business functionality and the purpose of application under test. Everybody should be able to understand the intent of feature file by reading the first feature step.

Feature: Login Action Test

Description: This feature will test a Login and Logout functionality

- **Scenario:** Basically, a scenario represents a particular functionality which is under test. By seeing the scenario, user should be able to understand the intent behind the scenario and test.

Scenario: Successful Login with Valid Credentials

Given User is on Home Page

When User Navigate to Login Page

And User enters UserName and Password

Then Message displayed Login Successfully

3. Gherkin: Given, When, Then, and Background Steps

Step 3.1: Demonstrating Gherkin keyword/scenarios explanation

Each scenario should follow **given**, **when**, and **then** format.

- Given: n Specifies the preconditions. It is basically a known state.
- When: This is used when some action is to be performed. Then: The expected outcome or result should be placed here. For Instance, verify if the login and page navigation is successful.
- Background: Whenever any step is required to be performed in each scenario, then those steps are to be placed in the Background. For Instance, If user needs to clear the database before each scenario, then those steps can be put in the background.
- And: Used to combine two or more actions of the same type.
- But: Signifies logical OR condition between any two statements. OR can be used in conjunction with GIVEN, WHEN ,and THEN statement.

Feature: Add to Cart

This feature will test functionality of adding different products to the User basket from different flow

Background: User is Logged In

Scenario: Search a product and add the first result/product to the User basket

Given User searched for Lenovo Laptop

When Add the first laptop that appears in the search result to the basket

Then User basket should display with 1 item

4 Gherkin Step Argument

Step 4.1: Gherkin step argument

- In some cases, you might want to pass more data to a step that fits on a single line. For this purpose, Gherkin has Docstrings and Data tables.

Step 4.2: Docstrings

- If you need to specify information in a scenario that won't fit on a single line, you can use Docstrings.
- A Docstring follows a step. It starts and ends with three double quotes, like this:

When I ask to reset my password

Then I should receive an email with:

"""

Dear bozo,

Please click this link to reset your password

"""

Step 4.3: Data Tables

- Data Tables are handy for passing a list of values to a step definition

name	email	twitter	
Aslak	aslak@cucumber.io	@aslak_hellesoy	

Julien	julien@cucumber.io	@jbpros	
Matt	matt@cucumber.io	@mattwynne	

- Just like Docstrings, Data Tables will be passed to the step definition as the last argument.

5. Gherkin Comments and Tags

Step 5.1: Gherkin comments with example.

- Comment is basically a piece of code meant for documentation purposes and not for execution.
- Feature file: In case of a feature file, we just need to put # before beginning your comment.

Example:

```
Feature: annotation  
  
#This is how background can be used to eliminate duplicate steps  
  
Background:  
User navigates to Facebook  
Given I am on Facebook login page
```

- Step definition file: If you are using Java as a platform then mark your comments with //.

Example:

```
//scroll to the bottom of the page  
  
((JavascriptExecutor) driver).executeScript("window.scrollTo(0,  
document.body.scrollHeight)");
```

Step 5.2: Gherkin tags with example.

- if we have many scenarios in a feature file, to put them under a single umbrella, we use tags to generate reports for specific scenarios under the same tag.

- Tags are defined in our runner class like this:

```
@RunWith(Cucumber.class)

@CucumberOptions{

    format= {"Pretty", "json:target/output.json", "html:target/html"},

    feature={"src/functional-test/resources"},

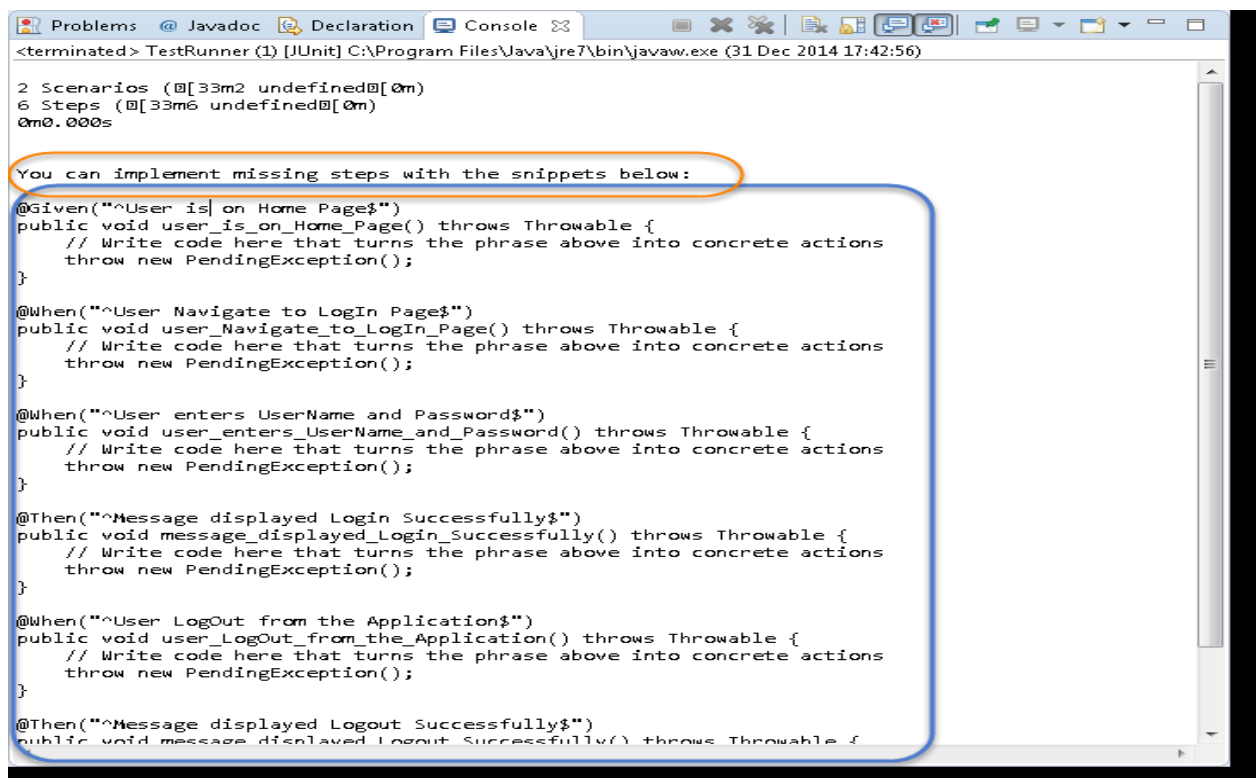
    tags={"@tag", "@tag1" }
```

- When we define multiple tags in runner class in below form, it will be defined with the use of logical operator:
 - tags = {"@tag", "@tag1"}: means AND condition. It says that scenarios matching both these tags need to be executed.
 - tags = {"@tag1, @tag2"}: means OR condition. It says that scenarios matching any of this tag need to be executed.

6. Step Definition

Step 6.1 Adding a step definition file.

- Create a new Class file in the “stepDefinition” package and name it as “Test_Steps”
- When we run the Feature file as Cucumber feature, the following message will be displayed in the console window:



The screenshot shows a Java IDE console window with the following content:

```
<terminated> TestRunner (1) [JUnit] C:\Program Files\Java\jre7\bin\javaw.exe (31 Dec 2014 17:42:56)

2 Scenarios ([33m2 undefined[0m)
6 Steps ([33m6 undefined[0m)
0m0.000s

You can implement missing steps with the snippets below:

@Given("^User is on Home Page$")
public void user_is_on_Home_Page() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@When("^User Navigate to Login Page$")
public void user_Navigate_to_Login_Page() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@When("^User enters UserName and Password$")
public void user_enters_UserName_and_Password() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

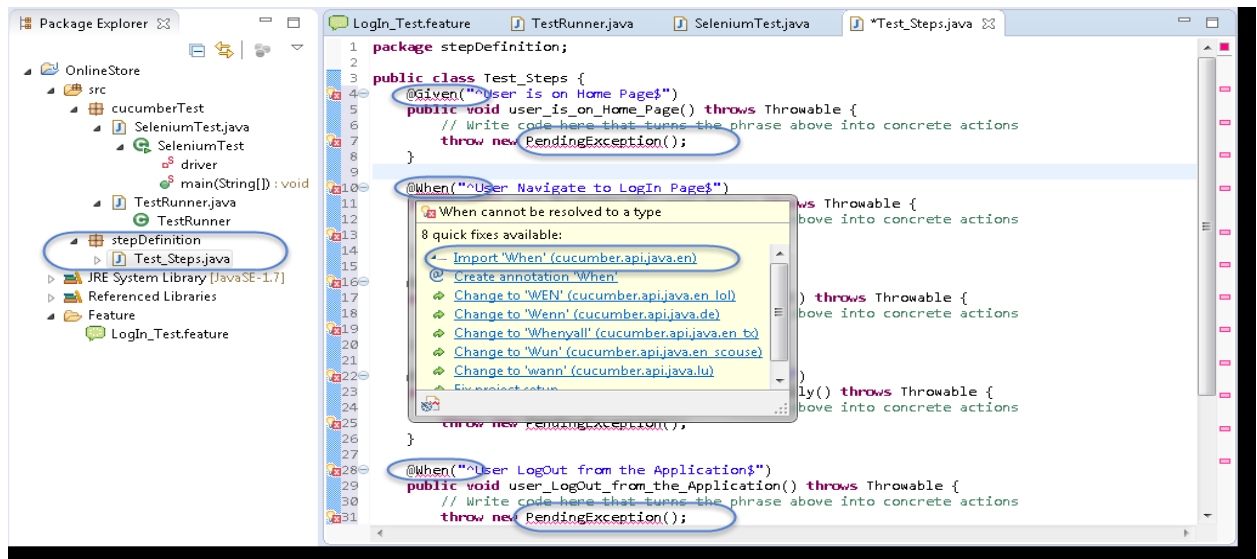
@Then("^Message displayed Login Successfully$")
public void message_displayed_Login_Successfully() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@When("^User Logout from the Application$")
public void user_LogOut_from_the_Application() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@Then("^Message displayed Logout Successfully$")
public void message_displayed_Logout_Successfully() throws Throwable {
```

In the image, the text "You can implement missing steps with the snippets below:" is circled in orange. The subsequent code snippets are enclosed in a blue rounded rectangle.

- Copy the complete text marked in a blue box and paste it into the above created Test_Steps class.
- The test will show many errors on “@” annotations. At the annotations, import the “cucumber.api.java.en” for all the annotations.



Step 6.2: Providing implementations using Selenium Java for the Step Definition methods.

- Using Selenium Java, provide the implementation to the first method “@Given (“^User is on Home Page\$”)”
 - Launch the Browser
 - Navigate to Home Page

Method will look like this now:

```
@Given("^User is on Home Page$")
public void user_is_on_Home_Page() throws Throwable
{
driver = new FirefoxDriver();
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
driver.get("http://www.store.demoqa.com");
}
```

- Using Selenium Java, provide the implementation to the second method “@When (“^User Navigate to Login Page\$”)”
 - Click on the Login link

Method will look like this now:

```
@When("^User Navigate to Login Page$")
public void user_Navigate_to_Login_Page() throws Throwable
{
    driver.findElement(By.xpath("./*[@id='account']/a")).click();
}
```

- Using Selenium Java, provide the implementation to the second method.

“@When (“^User enters Username and Password\$”)

- Enter Username and Password
- Click on Submit button

Method will look like this now:

```
@When("^User enters Username and Password$")
public void user_enters_Username_and_Password() throws Throwable
{
    driver.findElement(By.id("log")).sendKeys("testuser_1");
    driver.findElement(By.id("pwd")).sendKeys("Test@123");
    driver.findElement(By.id("login")).click();
}
```

- Do the same steps for the rest of the methods and the complete Test_Steps class will look like this:

```
package stepDefinition;
```

```
import java.util.concurrent.TimeUnit;
```

```
import org.openqa.selenium.By;
```

```
import org.openqa.selenium.WebDriver;
```

```
import org.openqa.selenium.firefox.FirefoxDriver;
```

```

import cucumber.api.java.en.Given;

import cucumber.api.java.en.Then;

import cucumber.api.java.en.When;


public class Test_Steps

{
    public static WebDriver driver;

    @Given("^User is on Home Page$")

    public void user_is_on_Home_Page() throws Throwable{

        driver = new FirefoxDriver();

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

        driver.get("http://www.store.demoqa.com"); }


    @When("^User Navigate to LogIn Page$")

    public void user_Navigate_to_LogIn_Page() throws Throwable {

        driver.findElement(By.xpath(".*[@id='account']/a")).click(); }


    @When("^User enters UserName and Password$")

    public void user_enters_UserName_and_Password() throws Throwable {

        driver.findElement(By.id("log")).sendKeys("testuser_1");

        driver.findElement(By.id("pwd")).sendKeys("Test@123");

        driver.findElement(By.id("login")).click(); }


    @Then("^Message displayed Login Successfully$")

    public void message_displayed_Login_Successfully() throws Throwable{

        System.out.println("Login Successfully"); }

```

```
@When("^User LogOut from the Application$")  
  
public void user_LogOut_from_the_Application() throws Throwable{  
    driver.findElement (By.xpath("./*[@id='account_logout']/a")).click(); }
```

```
@Then("^Message displayed Logout Successfully$")  
  
public void message_displayed_Logout_Successfully() throws Throwable {  
    System.out.println("LogOut Successfully"); }}
```


7. Hooks

Step 7.1: Before and After Hook

- Hooks are blocks of code that run **before** or **after** each scenario. You can define them anywhere in your project or step definition layers using the methods **@Before** and **@After**.

```
package utilities;
import cucumber.api.java.After;
import cucumber.api.java.Before;

public class Hooks {
    @Before
    public void beforeScenario(){
        System.out.println("This will run before the Scenario");
    }
    @After
    public void afterScenario(){
        System.out.println("This will run after the Scenario");A
    }
}
```

8. Tagged Scenarios

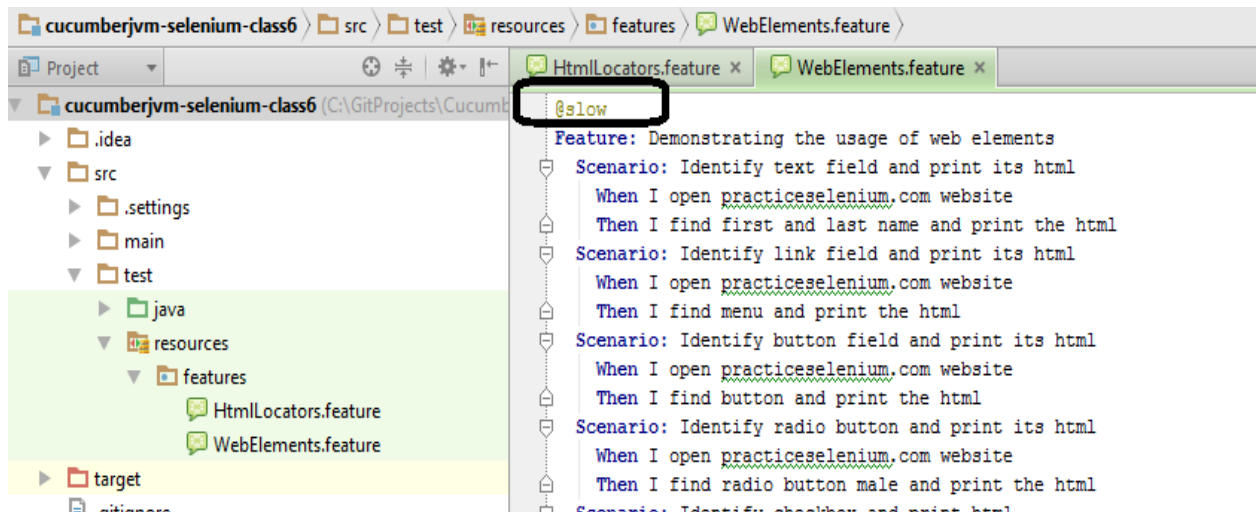
Step 8.1: How to add a tag to a feature file and scenario

- Syntax to add tag for feature file

Type “@<tag_name>” at the top of Feature or Scenario

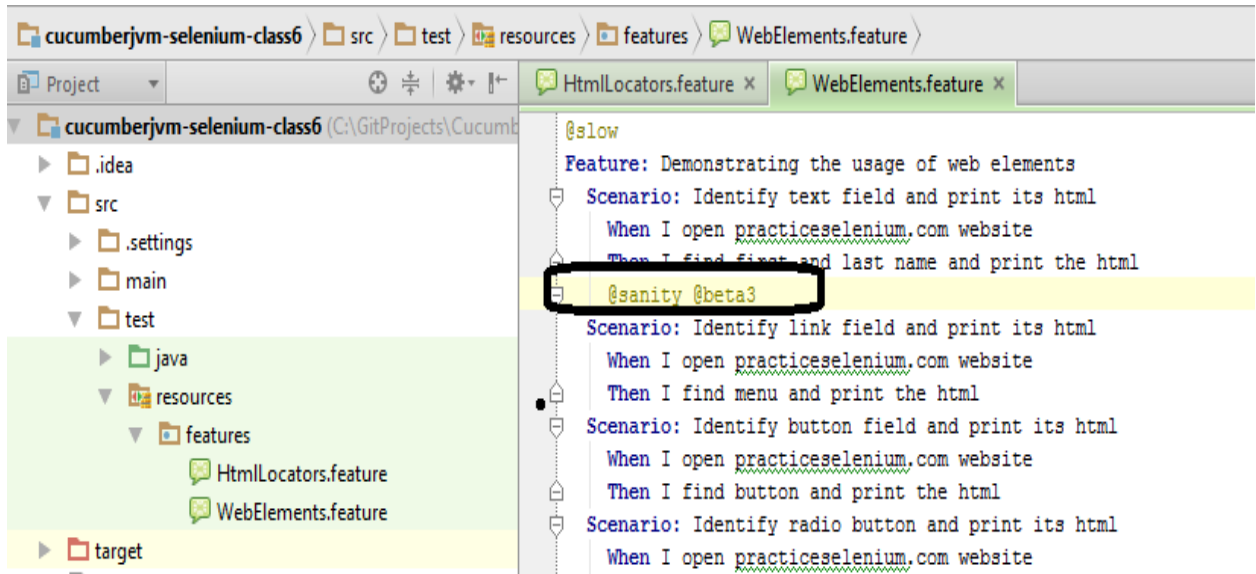
- Tag a feature

The tag applies automatically to all the scenarios on the feature file



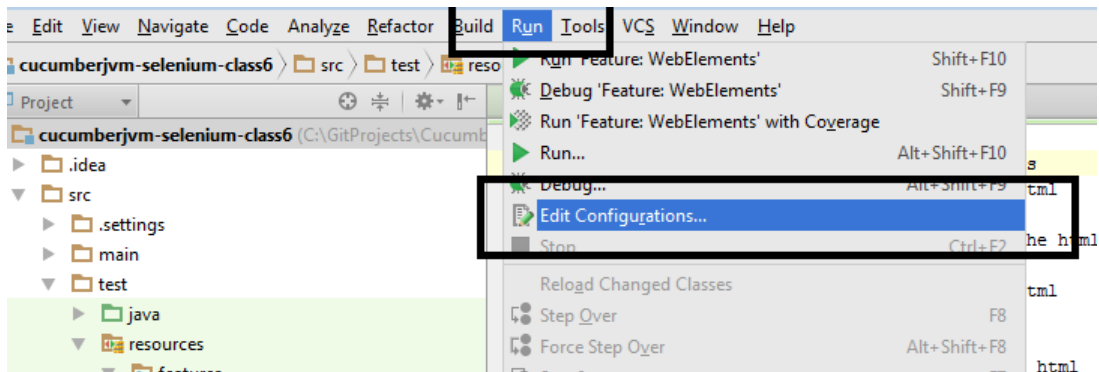
- Tag a scenario

Place the tag at the line exactly above the Scenario.

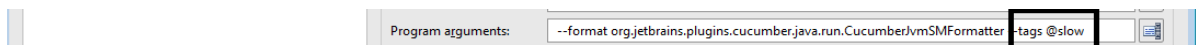


Step 8.2: Methods to run the tagged file

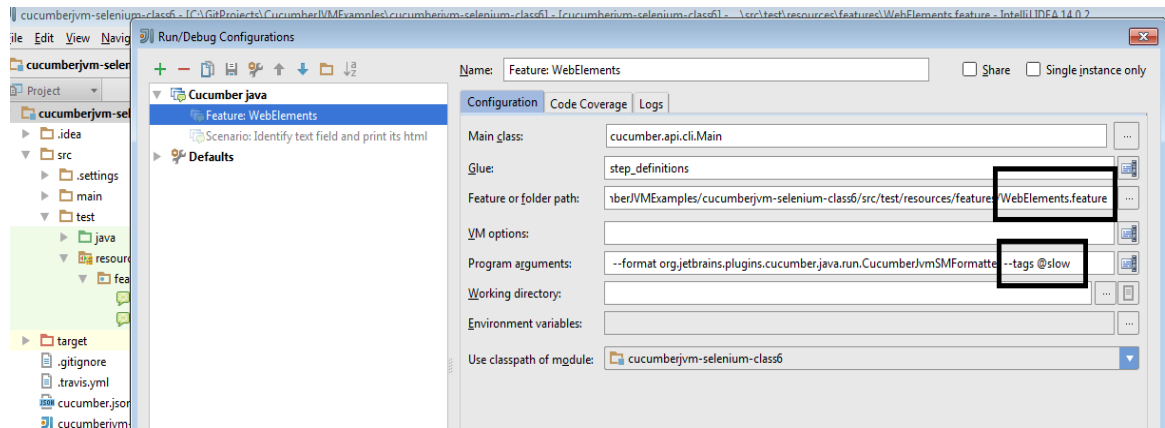
- Method1 (Run all @slow in features directory)



- Paste the path to **feature** folder in your project and write the following in Program arguments:



- Method2 (Run all @slow scenarios in a particular feature)



- Method3 (Run from the command line)

```
mvn test -Dcucumber.options="--tags @slow"
```

9. Execute Multiple Scenarios

Step 9.1: Multiple scenarios with examples

- Feature file can have more than one scenario or scenario outline. You can write your all possible requirement or scenarios for a particular feature in a feature file.

Feature: Registration, Login and MyAccount

Background:

Given I am on the homepage
And I follow "Sign in"

@regression @smoke

Scenario: Verify Login Functionality

And I fill "email address" with "goswami.tarun77@gmail.com"
And I fill "password" with "Test@1234"
And I click "sign in"
Then I should see "MY ACCOUNT" heading

@regression

Scenario: Create New User

When I fill "registration email text box" with "goswami.tarun77+1@gmail.com"
Then I click "create an account button"
And I enter following details
First Name	Tarun
Last Name	Goswami
Password	Test1234
Date	13
Year	1989
And I click "register button"

10. Tagged Hooks

Step 10.1: Tagged Hooks in Cucumber

- Tagged Hooks are used where you need to perform different tasks before and after scenarios.
- The first step is to annotate the required scenarios using **@ + AnyName** at the top of the Scenario. For this example, you just annotate each scenario with the sequence order of it, like **@First**, **@Second** & **@Third**.

FEATURE FILE

Feature: Test Tagged Hooks

@First

Scenario: This is First Scenario

Given this is the first step

When this is the second step

Then this is the third step

@Second

Scenario: This is Second Scenario

Given this is the first step

When this is the second step

Then this is the third step

@Third

Scenario: This is Third Scenario

Given this is the first step

When this is the second step

Then this is the third step

- Create a step definition file and print the execution order of the steps in the console.

STEP DEFINITION

```
package stepDefinition;

import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;

public class Hooks_Steps {

    @Given("^this is the first step$")
    public void This_Is_The_First_Step(){
        System.out.println("This is the first step");
    }

    @When("^this is the second step$")
    public void This_Is_The_Second_Step(){
        System.out.println("This is the second step");
    }

    @Then("^this is the third step$")
    public void This_Is_The_Third_Step(){
        System.out.println("This is the third step");
    }

}

Given this is the first step
When this is the second step
Then this is the third step
```

- Define *tagged hooks* in Hooks class file. Hooks can be used like **@Before("@TagName")**. Create before and after hooks for every scenario.

HOOKS

```
import cucumber.api.java.After;
import cucumber.api.java.Before;

public class Hooks {

    @Before
    public void beforeScenario(){
        System.out.println("This will run before the every Scenario");
    }

    @After
    public void afterScenario(){
        System.out.println("This will run after the every Scenario");
    }

    @Before("@First")
    public void beforeFirst(){
        System.out.println("This will run only before the First Scenario");
    }

    @Before("@Second")
    public void beforeSecond(){
        System.out.println("This will run only before the Second Scenario");
    }

    @Before("@Third")
    public void beforeThird(){
        System.out.println("This will run only before the Third Scenario");
    }

    @After("@First")
    public void afterFirst(){
        System.out.println("This will run only after the First Scenario");
    }

    @After("@Second")
    public void afterSecond(){
        System.out.println("This will run only after the Second Scenario");
    }
}
```



```
}  
  
@After("@Third")  
    public void afterThird(){  
        System.out.println("This will run only after the Third Scenario");  
    }  
}
```

- Run the feature file

11. Logical AND or OR

Step 11.1: Logical ANDing and ORing Tags

- There would be situations when you want to run all the scenarios related to two or more features. Cucumber tagging gives us the capability to choose what we want with the help of *ANDing* and *ORing*.

```
Example of Non-Optional Capture Group
@nonoptional @smoke @basicsearch
Scenario: Keyword Search bu different users
When I fill "search textbox" with "dress"
Then I click "search button"
Then I follow "TOP SELLERS" link
Then She follow "TOP SELLERS" link
Then He follow "TOP SELLERS" link
Then User follow "TOP SELLERS" link
```

```
#String transformation example
@smoke @datetest @advancedsearch
Scenario: Enter date in different format
Given The date is 2012-03-01T06:54:12
```

- **ANDing:** When we want to run the Scenarios with all the required Tags.

For example: when you want to execute scenarios tagged with @basicsearch and @smoke. Use below code in your Cucumber runner class:

- **ORing:** When we want to run the Scenarios with either of the mentioned Tags. For example, when you want to execute the scenario tagged with @basicsearch or @advancedsearch.

```
@RunWith(Cucumber.class)
@CucumberOptions(
    plugin = {"pretty", "html:target/cucumber"},
    features = {"src/test/resources/features"},
    glue={"com.pb.cucumberdemo.stepdefinitions"},
    tags = {"@basicsearch", "@smoke"},
```

)

12. Cucumber Data Tables

Step 12.1: Data Tables

- Data table is a set of inputs to be provided for a single tag. This tag can be GIVEN, WHEN, or THEN.
- Create a package named **dataTable** under **src/test/java**
- Create a Feature file.
 - Create a feature file, named as **dataTable.feature** inside the package **dataTable** (see section scenario outline for more detailed steps).
 - Write the following data table

```
Feature ' Data table
Verify that the new user registration is unsuccessful after passing
incorrect inputs
Scenario:
Given I am on the new user registration page
When I enter invalid data on the page

| Fields          | Values          |
| First Name      | Tom             |
| Last Name       | Kenny           |
| Email Address   | someone@someone.com |
| Re-enter Email Address | someone@someone.com |
| Password        | Password1       |
| Birthdate       | 01              |
```

- Save the file.
- Create a step definition file.
- Create the step definition file named as 'dataTable.java' inside the package **dataTable** (see section scenario outline for more detailed steps).
- Write the following code:

```
package dataTable;
```

```

import java.util.List;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.Select;

import cucumber.annotation.en.Given;
import cucumber.annotation.en.Then;
import cucumber.annotation.en.When;
import cucumber.table.DataTable;

public class stepdefinition {
    WebDriver driver = null;

    @Given("^I am on new user registration page$")
    public void goToFacebook() {
        //Intiate web browser instance. driver = new FirefoxDriver();
        driver.navigate().to("https://www.facebook.com/");
    }

    @When("^I enter invalid data on the page$")
    public void enterData(DataTable table){
        //Initialize data table
        List<list> data = table.raw();
        System.out.println(data.get(1).get(1));

        //Enter data
        driver.findElement(By.name("firstname")).sendKeys(data.get(1).get(1));
        driver.findElement(By.name("lastname")).sendKeys(data.get(2).get(1));
        driver.findElement(By.name("reg_email__")).sendKeys(data.get(3).get(1));
        driver.findElement(By.name("reg_email_confirmation__")).
            sendKeys(data.get(4).get(1));
        driver.findElement(By.name("reg_passwd__")).sendKeys(data.get(5).get(1));

        Select dropdownB = new
        Select(driver.findElement(By.name("birthday_day")));
        dropdownB.selectByValue("15");

        Select dropdownM = new
        Select(driver.findElement(By.name("birthday_month")));
        dropdownM.selectByValue("6");
    }
}

```

```

        Select dropdownY = new
Select(driver.findElement(By.name("birthday_year")));
        dropdownY.selectByValue("1990");

        driver.findElement(By.className("_58mt")).click();
        // Click submit button driver.findElement(By.name("websubmit")).click();
    }

    @Then("^User registration should be unsuccessful$")
    public void User_registration_should_be_unsuccessful() {
        if(driver.getCurrentUrl().equalsIgnoreCase("https://www.facebook.com/")){
            System.out.println("Test Pass");
        } else {
            System.out.println("Test Failed");
        }
        driver.close();
    }
}

```

- Save the file
- Create runner class
 - Create runner class named as runTest.java inside the package.
 - Write the following code.

```

package dataTable;

import org.junit.runner.RunWith;
import cucumber.junit.Cucumber;

@RunWith(Cucumber.class)
@Cucumber.Options(format = {"pretty", "html:target/cucumber"})

public class runTest { }

```

- Save the file.
- Run the test using the option

- Select runTest.java file from the package explorer.
- Right-click and select the option, Run as.
- Select JUnit test.

13. Cucumber Integration with Extent Report

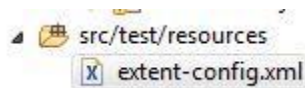
Step 13.1: Integrating Cucumber with Extent Report

- In POM.Xml please add below maven dependencies.

```
<dependency>
<groupId>com.aventstack</groupId>
<artifactId>extentreports</artifactId>
<version>3.0.6</version>
</dependency>

<dependency>
<groupId>com.vimalselvam</groupId>
<artifactId>cucumber-extentsreport</artifactId>
<version>3.0.2</version>
```

- Add **extent-config.xml** file under **test/resources** in your maven project.



- And add below code inside it.

```
<?xml version="1.0" encoding="UTF-8"?>
<extentreports>
```



```
<configuration>
```

```
<!-- report theme -->
```

```
<!-- standard, dark -->
```

```
<theme>standard</theme>
```

```
<!-- document encoding -->
```

```
<!-- defaults to UTF-8 -->
```

```
<encoding>UTF-8</encoding>
```

```
<!-- protocol for script and stylesheets -->
```

```
<!-- defaults to https -->
```

```
<protocol>https</protocol>
```

```
<!-- title of the document -->
```

```
<documentTitle>ExtentReports 2.0</documentTitle>
```

```
<!-- report name - displayed at top-nav -->
```

```
<reportName></reportName>
```

```
<!-- report headline - displayed at top-nav, after reportHeadline -->
```

```
<reportHeadline>Automation Report</reportHeadline>
```

```
<!-- global date format override -->
```

```
<!-- defaults to yyyy-MM-dd -->
```

```
<dateFormat>yyyy-MM-dd</dateFormat>
```

```
<!-- global time format override -->
```

```
<!-- defaults to HH:mm:ss -->
```

```
<timeFormat>HH:mm:ss</timeFormat>
```

```
<!-- custom javascript -->
```

```
<scripts>
```

```
<![CDATA[
```

```
$(document).ready(function() {
```

```
});
```

```
]]>
```

```
</scripts>
```

```
<!-- custom styles -->
```

```
<styles>
```

```
<![CDATA[
```

```
]]>
```

```
</styles>
```

```
</configuration>
```

```
</extentreports>
```

- In the Runner File under plugin add below code.

```
Plugin={“com.cucumber.listener.ExtentCucumberFormatter:target/html/ExtentReport.html “}
```

- Also add below code in After class of runner file.

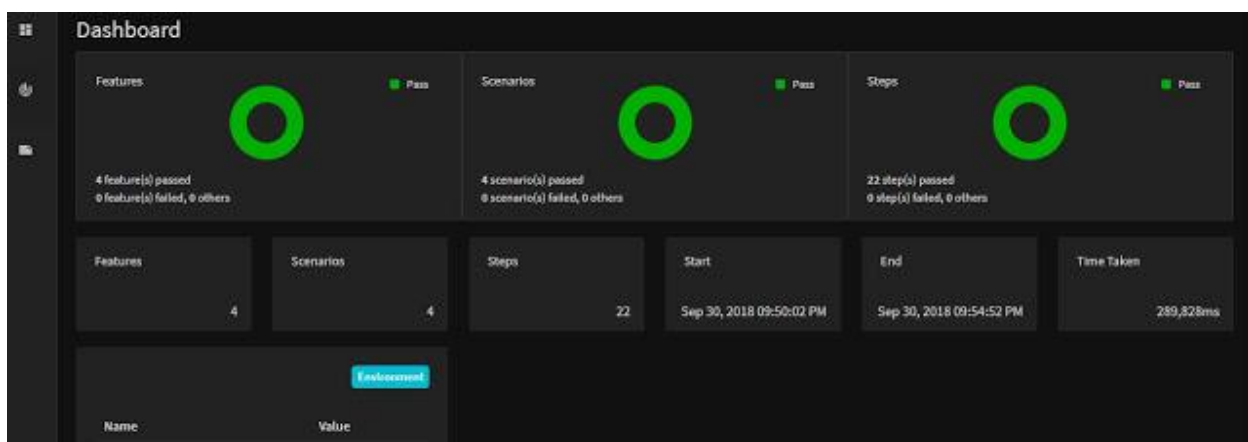
```
public class TestRunner  
{  
    @AfterClass
```

```

public static void setup()
{
    Reporter.loadXMLConfig(new File("src/test/resources/extent-config.xml"));
    Reporter.setSystemInfo("User Name", "AJ");
    Reporter.setSystemInfo("Application Name", "Test App ");
    Reporter.setSystemInfo("Operating System Type",
        System.getProperty("os.name").toString());
    Reporter.setSystemInfo("Environment", "Production");
    Reporter.setTestRunnerOutput("Test Execution Cucumber Report");
}
}

```

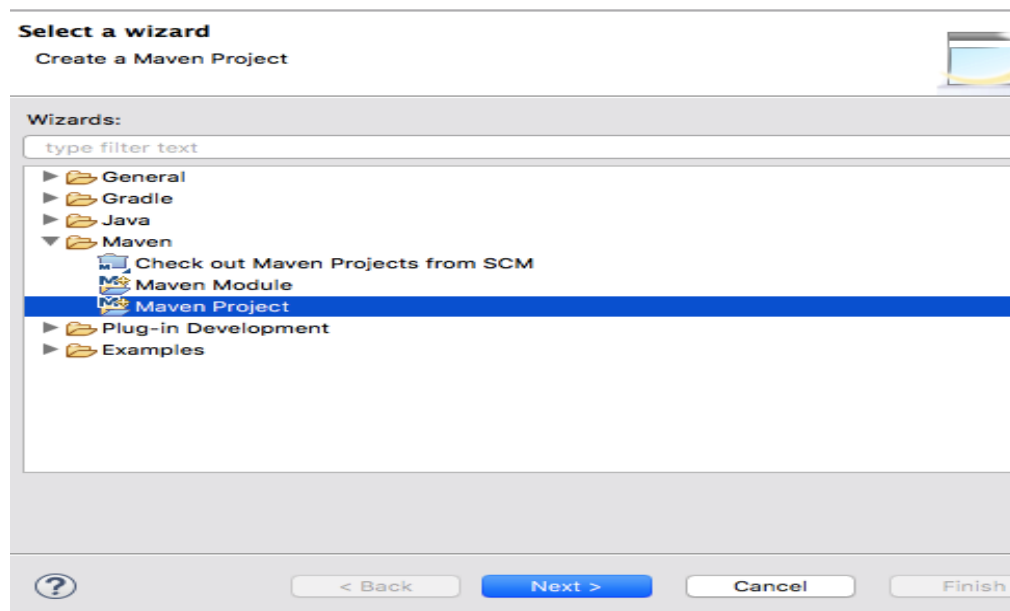
Now, after executing the code extent report would be generated inside /target/html folder and it would be looking something like this



14. Cucumber Execution with Maven

Step 14.1: Executing Cucumber Tests with Maven on Local Machine

- To create a Maven Project in Eclipse, click on **New** → **Project** → **In the wizard**, select **Maven Project**.



- On the new Maven Project pop-up, select the checkbox to create your project at the default location OR you can also browse and set a new location of your choice. Click on Next to proceed.
- In the next screen, you will have to mention a Group ID and Artifact ID of your own choice; this is the name of your Maven project. Once you click the Finish button, a Maven project will be created in Eclipse.

New Maven Project

Specify Archetype parameters

Group Id: Cucumber_Selenium

Artifact Id: Cucumber_Selenium

Version: 0.0.1-SNAPSHOT

Package: Cucumber_Selenium.Cucumber_Selenium

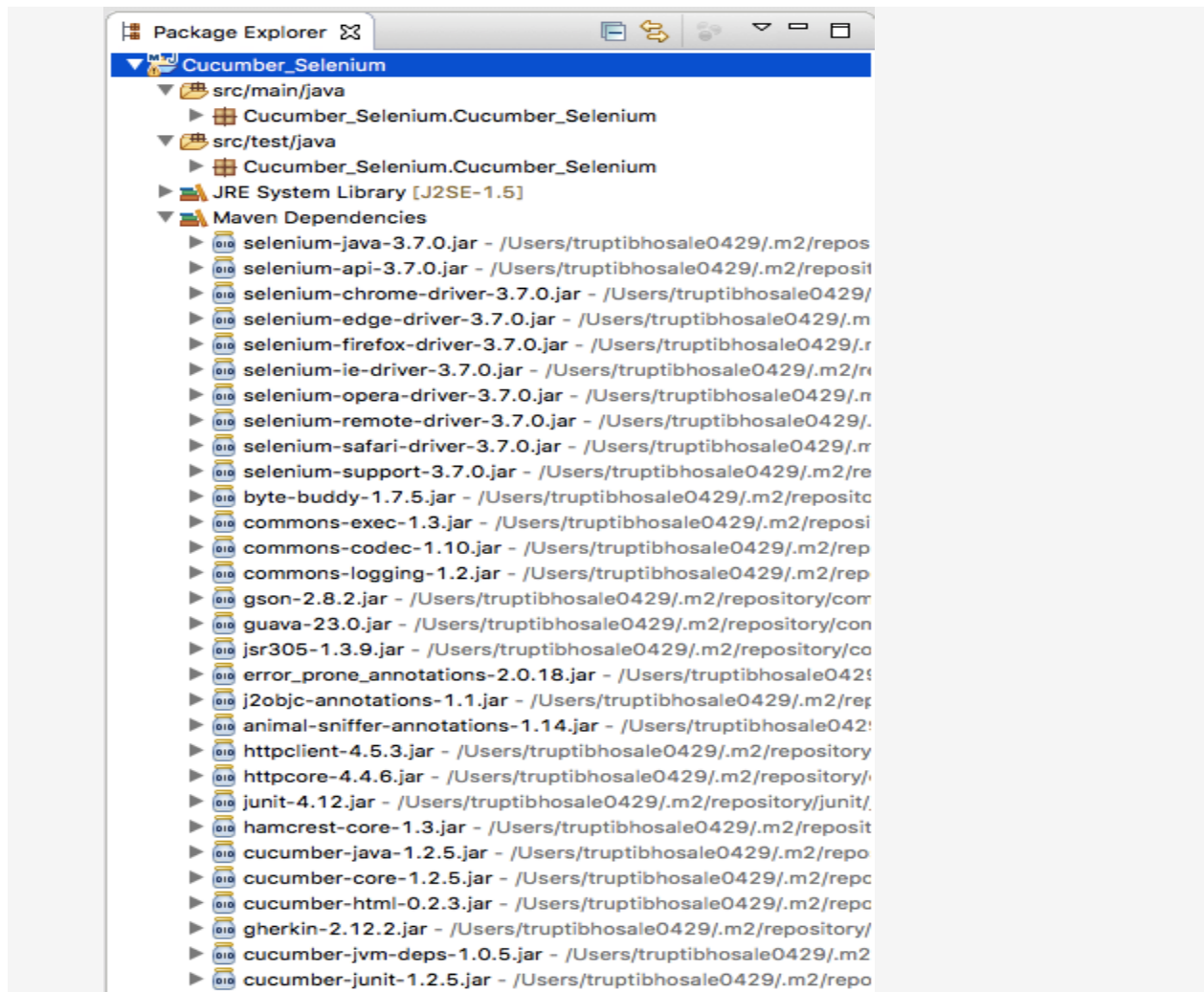
Properties available from archetype:

Name	Value

Advanced

< Back Next > Cancel Finish

- Now, in order to build a Selenium-Cucumber framework, we need to add a dependency for Selenium and Cucumber in pom.xml.
- Copy the dependency tag for the following from the Maven Repository.
 - Selenium Web driver
 - Cucumber-Core jar
 - Cucumber-Java jar
 - Cucumber-TestNG jar
- Make sure to update the project after adding dependencies to pom.xml; you can do that by right clicking *Project* → *Maven* → *Update Project*. Once you update the project, you will see that many JAR files are added to the Maven Dependencies folder in your project.



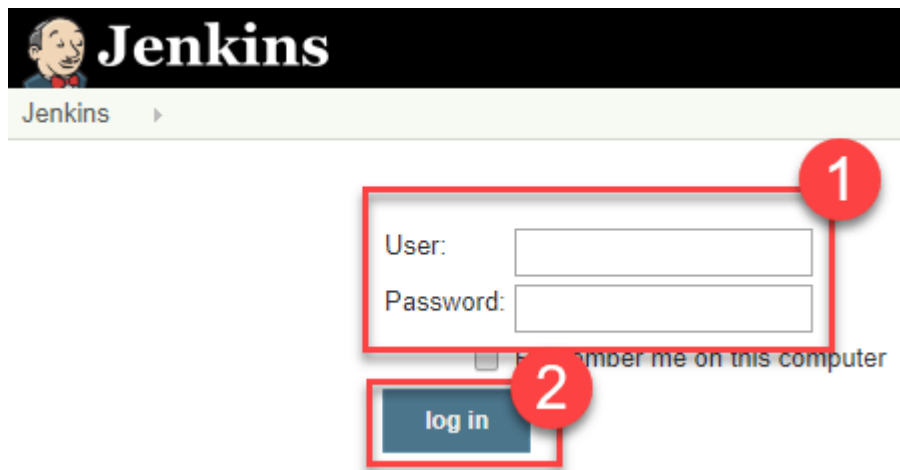
- To proceed with Cucumber implementation, you need to create three packages to store the feature files, step definition code, and test runner code. Let us create three packages: the features, seleniumgluecode, and runner. To create a new package in src/test/java, right Click the folder-> New-> Package.
- Now create the feature file in the Feature package. Right click ->New ->File->Enter name test.feature.
- Create a class test.java to write the gluecode for the features written. Right click seleniumgluecode->New ->Class->enter name as test and save.

- To run the feature files and their respective code, we need to write a TestNG runner class. Right click **runner** -> **New**-> **Class**->enter name as **testrunner**.

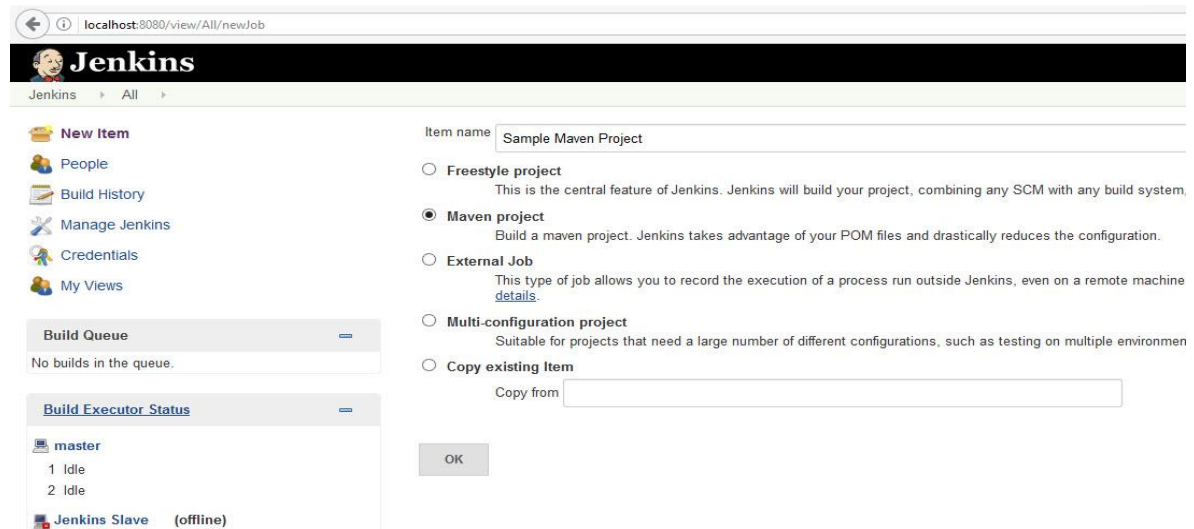
15. Build (Execute) Jenkins Job

Step 15.1: Executing through Jenkins

- Login into Jenkins

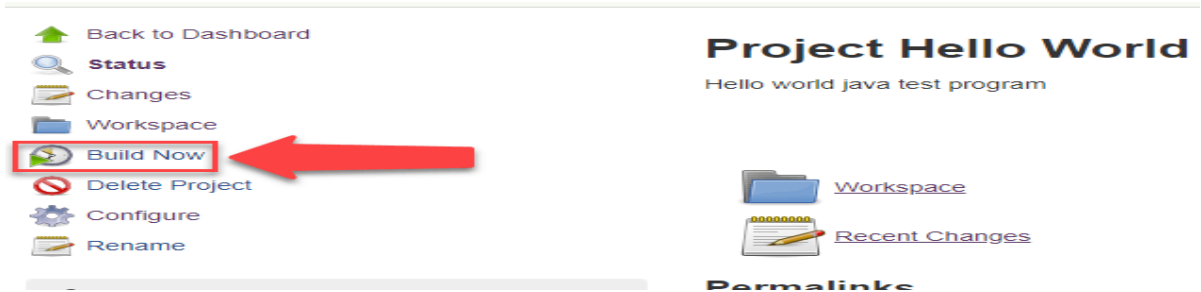


- Create a new job as Maven Project



- After then click on the created projected from the Jenkins Dashboard and click on "Configure".

- In Job Configuration, go to the “Build” section and provide the path of pom.xml which has to be executed.
- And click on “Save” and click on “Build Now” to start the execution on Jenkins.



- After Executing the build, you can review the result as below:

```
Results :

Tests run: 8, Failures: 0, Errors: 0, Skipped: 0

[JENKINS] Recording test results
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 13.848 s
[INFO] Finished at: 2017-01-28T22:26:26+05:30
[INFO] Final Memory: 15M/36M
[INFO] -----
[JENKINS] Archiving F:\Selenium\mavenparameterize\pom.xml to com.easy/mavenparameterize/0.0.1-SNAPSHOT/mavenparameterize-0.0.1-SNAPSHOT.pom
channel stopped
Finished: SUCCESS
```

16. Jenkins Integration with Extent Report

Step 16.1: Jenkins Integration with Extent Report

To setup an extent report in Jenkins Dashboard:

- Login into Jenkins-->Manage Jenkins and Install HTML Publisher Plugin. Please make sure to restart Jenkins after the plugin installation.
- Now go to Jenkins Home Page create New Jenkins Job and in Post-Build Action select publish Selenium HTML report plugin.



- Now Give Path of your Extent Report, Name and Title.

Publish HTML reports

Reports

HTML directory to archive: D:\Test_PP_POM\test-output

Index page[s]: STMExtentReport9.html

Index page title[s] (Optional):

Report title: Extent Report

Publishing options...

Add

- Execute your Jenkins Job and now, Extent Report would be visible in your Jenkins Dashboard.



