**Cucumber-**

Cucumber is a testing framework which supports Behavior Driven Development (BDD). Cucumber itself is written in Ruby, but it can be used to “test” code written in Ruby or other languages including but not limited to Java, C# and Python.

In BDD we only focus on application behavior, all requirements will be in the plain text file and in pure English language. It helps client, developer and tester to understand the requirement easily.

**How to Install cucumber**

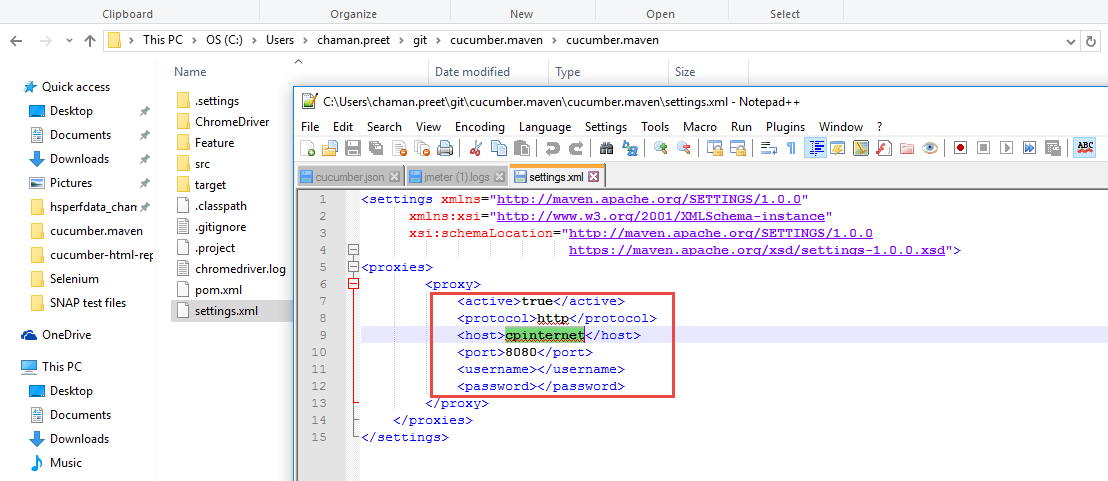
Get ready with Eclipse, jdk and jre path set

**Install maven-**

* Download Maven from the following link − <https://maven.apache.org/download.cgi>
* Create environment variable MAVEN\_HOME giving path of apache maven
* Add maven bin path in Path variable
* Download MAVEN plugin from Eclipse.

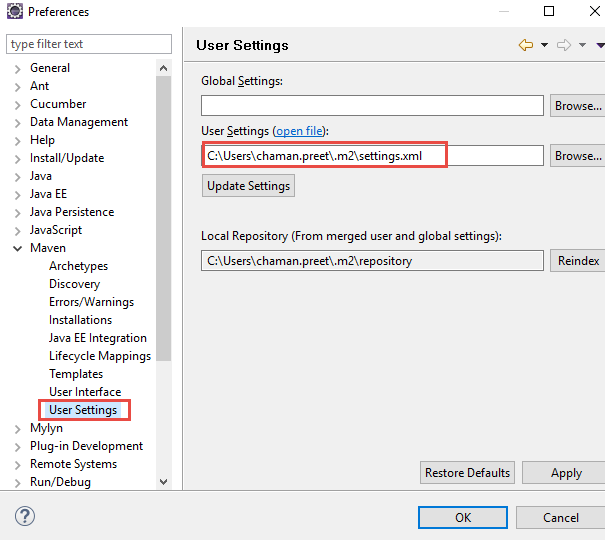
**Configure Cucumber with Maven**

If working behind proxy, Add a file –setting.xml giving proxy setting for your system under project.

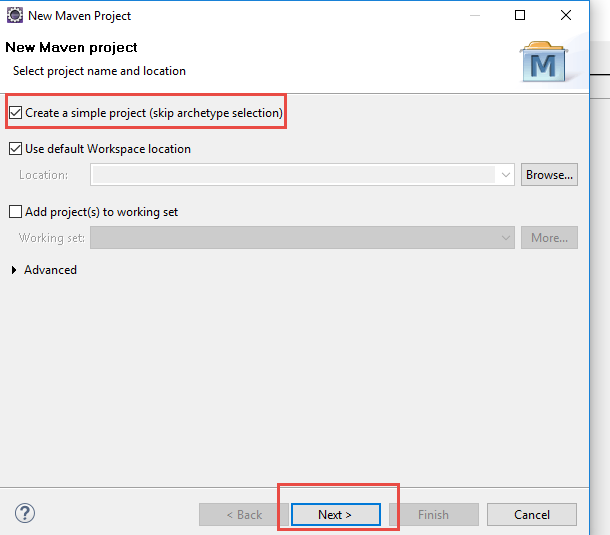


* Check maven setting in windows> Preferences> Maven> user settings

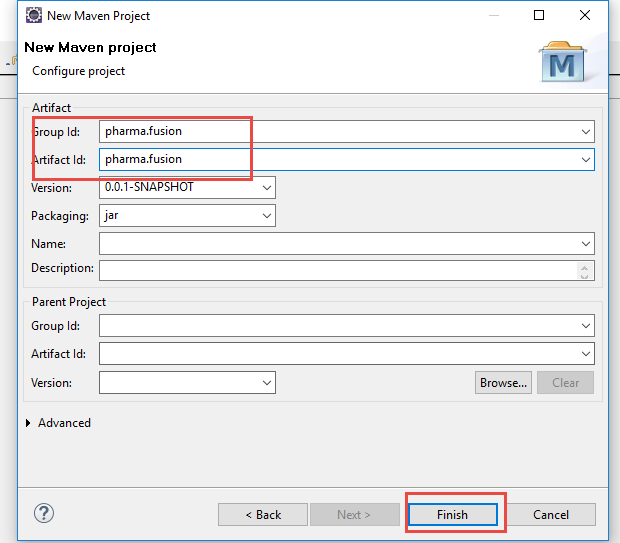
Setting.xml should be routed to correct file with proxy settings.



* Create Maven project



* Give user id and group id (unique for ur project), complete until Finish



* In POM.xml file, Add dependency for selenium-java, Cucumber-Java, Cucumber-JUnit, JUnit.
* For testing, Add dependency for cucumber-testng, testing
* In plugin tag for reporting, add plugin for maven-compiler-plugin, maven-cucumber-reporting, maven-surefire-plugin.

Clean project

**Create feature file**- used to write test automation steps or acceptance tests. All the feature files end with .feature extension. **Add plugin**- cucumber eclipse feature for feature file formatting.

Feature file has various keywords (Feature, Scenario, Background, Given, When, And and Then)

**Feature**: Defines what feature you will be testing in the tests below

**Given**: Tells the pre-condition of the test

**And**: Defines additional conditions of the test

**Then**: States the post condition. You can say that it is expected result of the test.

**Background:**Whenever any step is required to perform in each scenario then those steps need to be placed in Background. For Instance: If a user needs to clear database before each scenario then those steps can be put in a background.

This is gherkin language.

**Example:**

**Feature**: Login Functionality Feature

**Background:**

**Given** user logged in as databases administrator  
**And** all the junk values are cleared

**Scenario**: Login Functionality  
**Given** user navigates to SOFTWARETETINGHELP.COM  
**When** user logs in using Username as “USER”  
**And**password as “password”  
**Then** login should be successful  
**And** Home page should be displayed

**Scenario Outline keyword**- To provide multiple inputs.

Example-

Scenario Outline: Search a keyword

When I Enter a "<Keyword>" and press search

Then Search results appear

Then Application should be closed

Examples:

| Keyword |

| Selenium |

| Cucumber |

To Enter multiple fields and values in When-

When I enter valid data

| Fields | Values |

| Username | test1 |

| Password | xyz |

Use of tags- Can give any tag name before Feature and specify in runner class to run.

Multiple tags can also be given- **@SmokeTest @LoginTest**

**Create a step definition file-**

Steps definition file stores the mapping between each step of the scenario defined in the feature file with a code of function to be executed.

Execute program and copy methods to be written in step definition file

### **Example of Step Definition File**

public void goToFacebook() {

driver = new FirefoxDriver();

driver.navigate().to("https://www.facebook.com/");

}

@When "^user logs in using Username as \"([^\"]\*)\" and Password as \"([^\"]\*)\"$"

public void I\_enter\_Username\_as\_and\_Password\_as(String arg1, String arg2) {

driver.findElement(By.id("email")).sendKeys(arg1);

driver.findElement(By.id("pass")).sendKeys(arg2);

driver.findElement(By.id("u\_0\_v")).click();

}

@Then"^login should be unsuccessful$"

public void validateRelogin() {

if(driver.getCurrentUrl().equalsIgnoreCase(

"https://www.facebook.com/login.php?login\_attempt=1&lwv=110")){

System.out.println("Test Pass");

} else {

System.out.println("Test Failed");

}

driver.close();

}

**Create a JUnit runner to run the test –** To run the specific feature file cucumber uses standard JUnit Runner and specify tags in @Cucumber. Options.package runner;

@RunWith(Cucumber.class)

@CucumberOptions(features="Feature",glue={"step\_definition"},

plugin={"html:target/cucumber-html-report","json:target/cucumber.json"}

,tags={"@web"}

)

public class Run\_feature extends AbstractTestNGCucumberTests{

}

**Cucumber hooks-** To execute code before and after whole program

before hook gets executed well before any other test scenarios, and after hook gets executed after executing all the scenarios.

@Before public void setUp(){

driver = new FirefoxDriver();

}

@After public void cleanUp(){

driver.close();

}

**Reporting-**

Add plugins like maven-cucumber-reporting, maven-surefire-plugin for reporting purpose

* Add configuration details to mention directory and project name in execution tag in maven-cucumber-reporting.
* To continue even after error, add <testfailureignore>true</testfailureignore> in maven-surefire-plugin

**Running projects using maven commands in cmd-**

Go to project directory and open cmd as admin.

Use commands-

Mvn clean

Mvn compile

Mvn verify

Mvn install

Cucumber- it is used to test the system as a whole rather than testing a particular piece of code.

This tool is useful for writing acceptance tests for the web application.