CUCUMBER

BDD TESTING FRAMEWORK

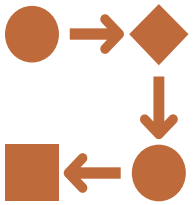
CUCUMBER

Cucumber is one of the most powerful tools. It offers us the real communication layer on top of a robust testing framework.

The tool can help run automation tests on a wide-ranging testing needs from the backend to the frontend.

Moreover, Cucumber creates deep connections among members of the testing team, which we hardly find in other testing framework.

Test Driven Development (TDD)



TDD is an iterative development process. Each iteration starts with a set of tests written for a new piece of functionality.

Benefits of TDD:

~ Unit test proves that the code actually works

~ Can drive the design of the program

~ Refactoring allows improving the design of the code

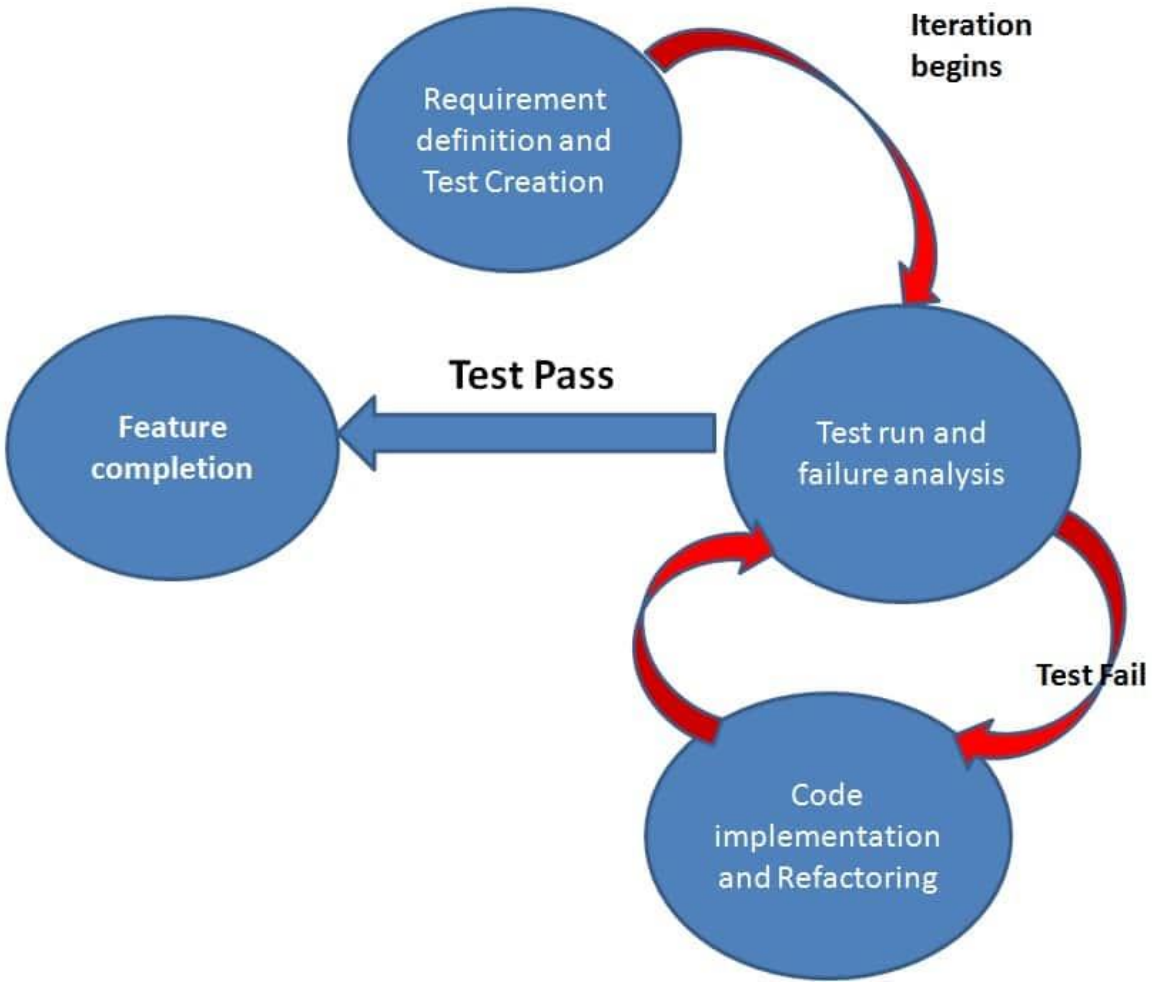
~ Low-Level regression test suite

~ Test first reduce the cost of the bugs

Drawbacks of TDD

- Developer can consider it as a waste of time
- The test can be targeted on verification of classes and methods and not on what the code really should do
- Test become part of the maintenance overhead of a project
- Rewrite the test when requirements change

Summarizing TDD in Figure



Behavior-Driven Development (BDD)

Behavior Driven testing is an extension of TDD. The major difference that we get to see here are

- *Tests are written in plain descriptive English type grammar*
- *Tests are explained as behavior of application and are more user-focused*
- *Using examples to clarify requirements*

This difference brings in the need to have a language that can define, in an understandable format.

Behavior-Driven Development (BDD)

This amazing feature of *Behavior-Driven Development (BDD)* approach with the advantages as below:

- *Writing BDD tests in an omnipresent language, a language whose structure is built around the domain model and widely used by all team members comprising of developers, testers, BAs, and customers.*
- *Connecting technical with non-technical members of a software team.*
- *Allowing direct interaction with the developer's code, but we write BDD tests in a language that can also be made out by business stakeholders.*
- *Last but not least, acceptance tests can execute automatically, while business stakeholders manually perform it.*

Features of BDD

- *Shifting from thinking in "tests" to thinking in "behavior"*
- *Collaboration between Business stakeholders, Business Analysts, QA Team and developers*
- *Ubiquitous language, it is easy to describe*
- *Driven by Business Value*
- *Extends Test-Driven Development (TDD) by utilizing natural language that non-technical stakeholders can understand*
- *Connecting technical with non-technical members of a software team.*

Features of BDD

CONTINUE.....

- *BDD frameworks such as Cucumber or JBehave are an enabler, acting a "bridge" between Business & Technical Language.*
- *Allowing direct interaction with the developer's code, but we write BDD tests in a language that can also be made out by business stakeholders.*
- *Last but not least, acceptance tests can execute automatically, while business stakeholders manually perform it.*

Set Up Cucumber with Eclipse

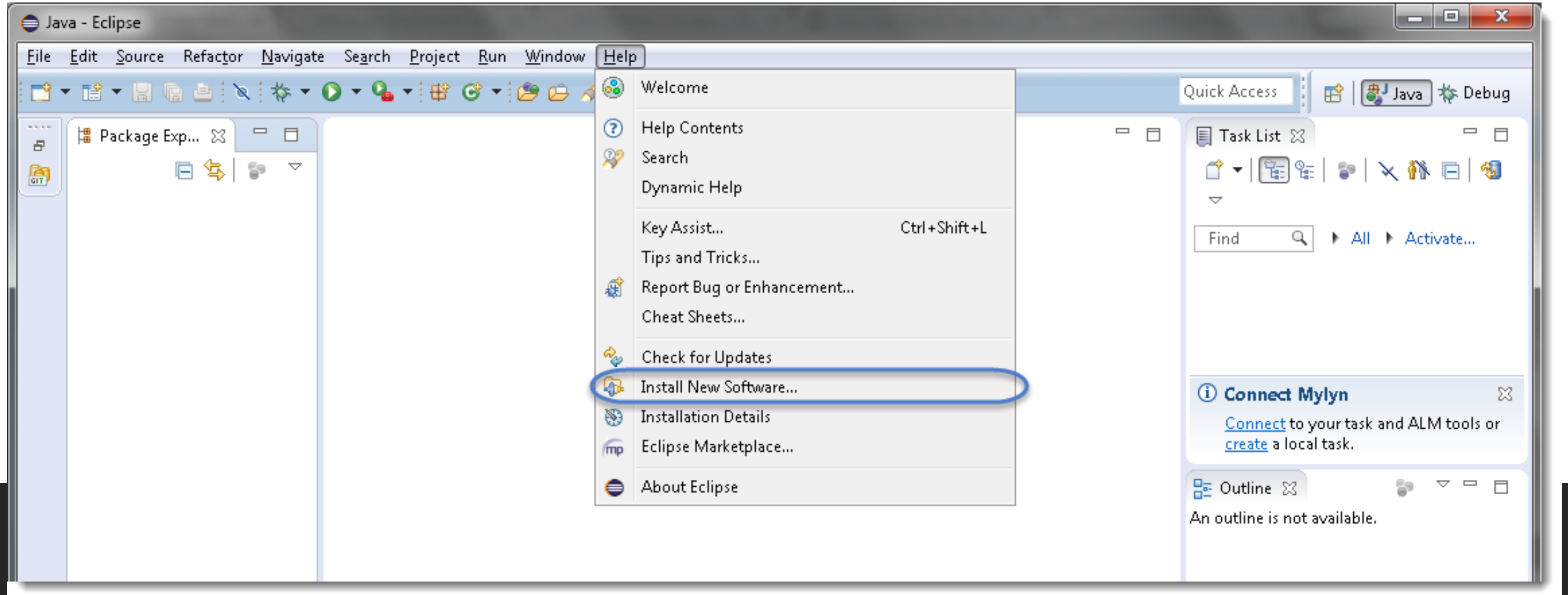
Pre-Requisites

- ***Set Up Java on System***
- ***Set Up Eclipse IDE or any other IDE***
- ***Set Up Maven***
- ***Create a new Maven Project***
- ***Create a 'resources' folder for Cucumber Tests***
- ***Add Selenium to Project***
- ***Add Maven Compiler Plugin***

Install Cucumber Eclipse Plugin

- It is easy to install ***Cucumber Eclipse Plugin***, as it comes as a plugin for ***Eclipse IDE***.
- Prerequisite for installing this plugin is your Internet connection should be up & running during installation of this plugin and Eclipse IDE should be installed in your computer.
- Please see ***Download and Install Eclipse*** to setup Eclipse to your system.

STEPS TO FOLLOW:



1) Launch the *Eclipse IDE* and from Help menu, click "***Install New Software***".

Install

Available Software

Select a site or enter the location of a site.

Work with: type or select a site | Add...

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Version
There is no site selected.	

Select All | Deselect All

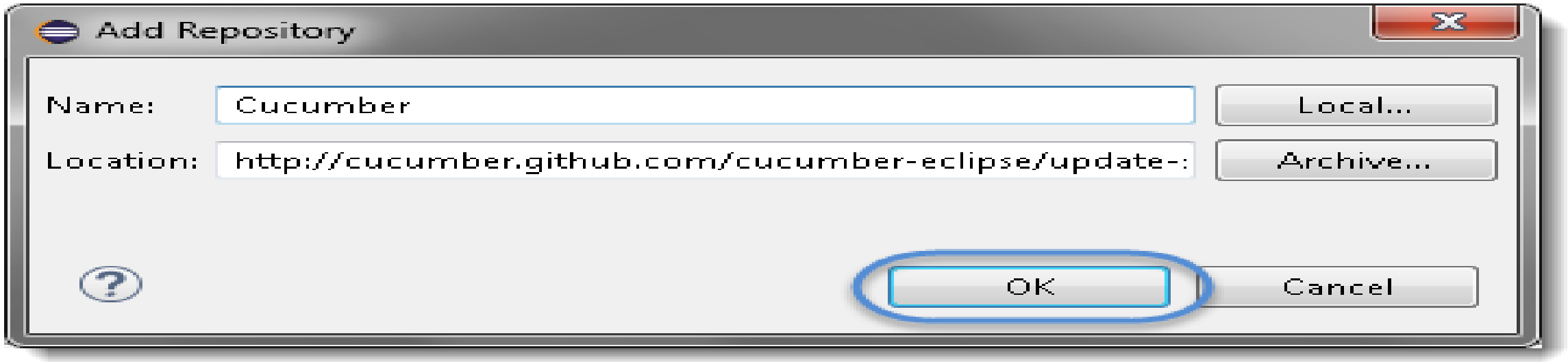
Details

- [x] Show only the latest versions of available software
- [x] Group items by category
- [ ] Show only software applicable to target environment
- [x] Contact all update sites during install to find required software
- [x] Hide items that are already installed

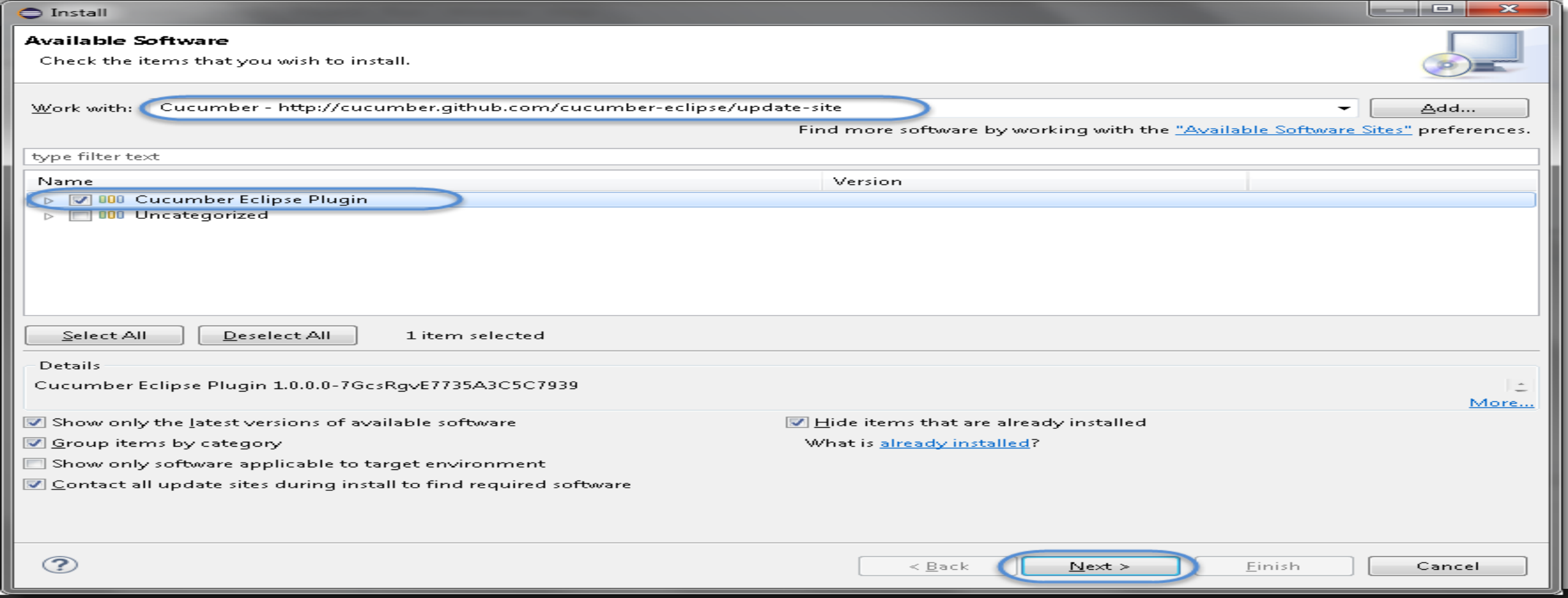
What is already installed?

< Back | Next > | Finish | Cancel

2) You will see a dialog window, click "***Add***" button.

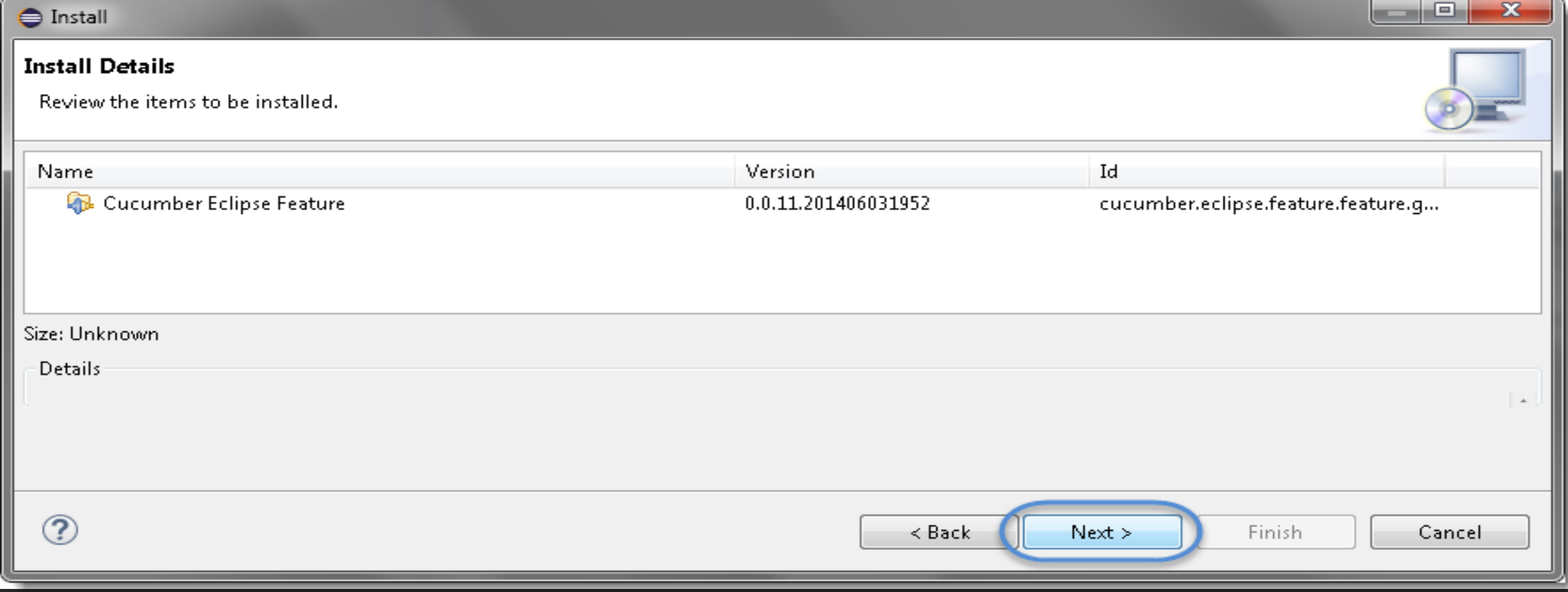


3) Type name as you wish, let's take "***Cucumber***" and type "***http://cucumber.github.com/cucumber-eclipse/update-site***" as location. Click ***OK***.

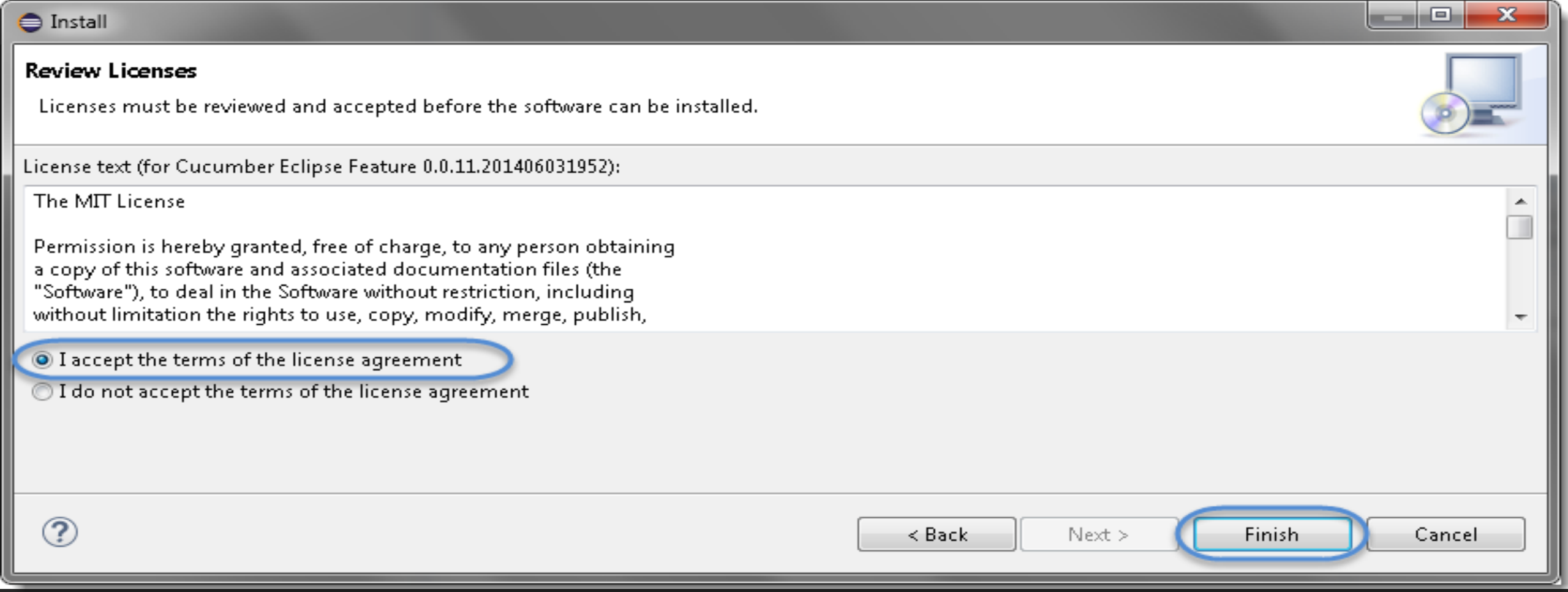


4) You come back to the previous window but this time you must see ***Cucumber Eclipse Plugin*** option in the available software list. Just ***Check*** the box and press "***Next***" button.

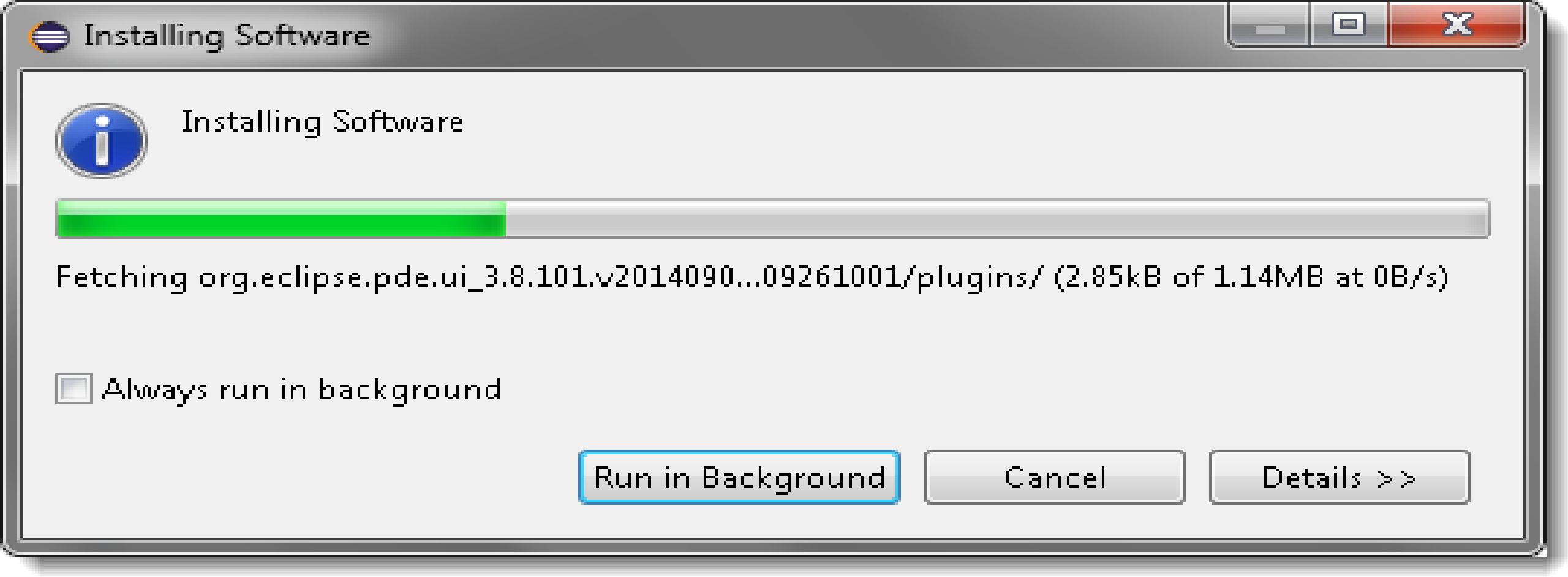
Note: *If running behind a proxy server and you get a 'HTTP Proxy Authentication Required' error you may need to contact a system administrator to set up your proxy server settings.*



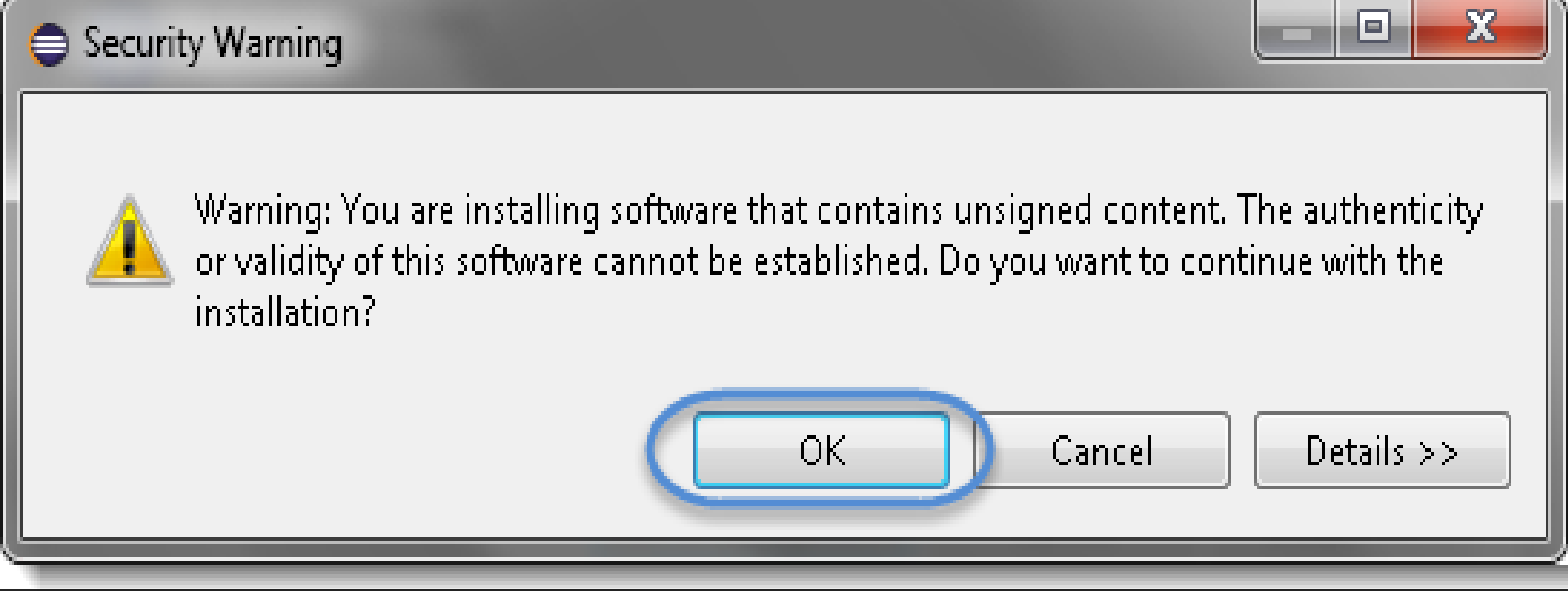
5) Click on ***Next***.



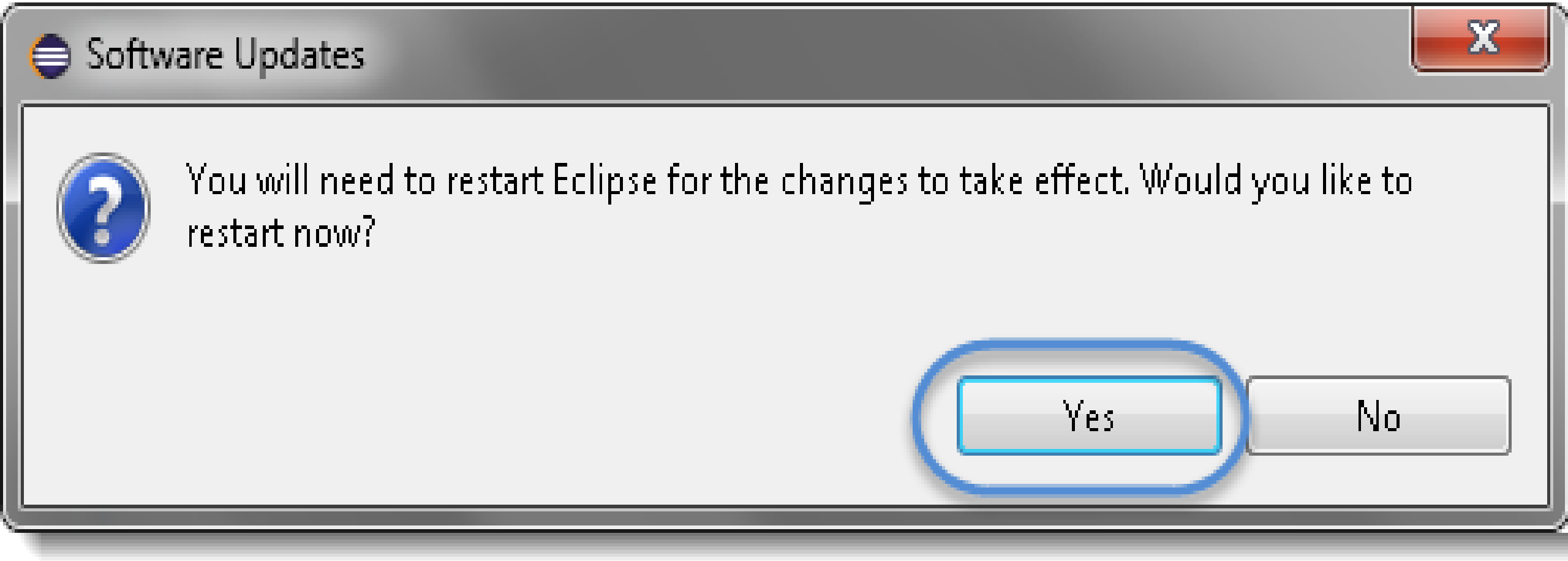
6) Click "**I accept the terms of the license agreement**" then click **Finish**.



7) Let it install, it will take few seconds to complete.



8) You may or may not encounter a Security warning, if in case you do just click ***OK***.



9) You are all done now, just click **Yes**.

Basic Things Of Cucumber

We must know three basic things before we write our first cucumber code.

- FEATURE FILE - **Feature** file is the place where we will write our plain English descriptions.
- RUNNER CLASS - **Runner class** solves this problem, **@CucumberOptions** provide a way to link the feature file with the step definitions. This is called "**Glue**".
- STEP DEFINITION FILE - The **Step definition file** is the one, which is an **actual code implementation** of the steps that we have written in the feature.

Cucumber Feature File

- **Cucumber proposes to write scenario in the** *Given/When/Then* **format.**
- A Feature File is an entry point to the *Cucumber* tests.
- This is a file where you will describe your tests in Descriptive language (Like English) called Gherkin. It is an essential part of Cucumber, as it serves as an automation test script as well as live documents.
- A feature file can contain a scenario or can contain many scenarios in a single feature file, but it usually contains a list of scenarios.

```
Feature: Login Action

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

Scenario: Successful LogOut
 When User LogOut from the Application
 Then Message displayed LogOut Successfully
```

Cucumber Feature File – Gherkin Keyword

- ✓You will quickly notice that there are some colored words. These words are *Gherkin keywords,* and each keyword holds a meaning.
- ✓Here is the list of keywords that *Gherkin* supports:
  - ***Feature***
  - ***Background***
  - ***Scenario***
  - ***Given***
  - ***When***
  - ***Then***
  - ***And***
  - ***But***
  - *******

Gherkin Keywords

Feature Keyword

- Each *Gherkin* file begins with a ***Feature*** keyword. *Feature* defines the logical test functionality you will test in this feature file.

Background Keyword

- ***Background*** keyword is used to define steps that are common to all the tests in the feature file.

Scenario Keyword

- Each Feature will contain a number of tests to test the feature. Each test is called a ***Scenario*** and is described using the *Scenario:* keyword.

Gherkin Keywords

Given Keyword

- ***Given*** defines a precondition to the test.

When Keyword

- ***When*** keyword defines the test action that will be executed.

Then Keyword

- ***Then*** keyword defines the Outcome of previous steps.

And Keyword

- ***And*** keyword is used to add conditions to your steps.

But Keyword

- ***But*** keyword is used to add negative type comments. It is not a hard & fast rule to use but only for negative conditions.
- It makes sense to use *But* when you will try to add a condition which is opposite to the premise your test is trying to set.

Gherkin Keywords

* **Keyword**
  - This keyword is very special. This keyword defies the whole purpose of having Given, When, Then and all the other keywords. Basically, Cucumber doesn't care about what Keyword you use to define test steps, all it cares about what code it needs to execute for each step. That code is called a ***step definition*** . All the keywords can be replaced by the **** keyword*** and your test will just work fine.

Let's see with example:

Feature: *LogIn Action Test*
Description: This feature will test a LogIn and LogOut functionality
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*

Using * Keyword
Feature: *LogIn Action Test*
Description: This feature will test a LogIn and LogOut functionality
Scenario: *Successful Login with Valid Credentials*
  ***** *User is on Home Page*
  ***** *User Navigate to LogIn Page*
  ***** *User enters UserName and Password*
  ***** *Message displayed Login Successfully*

JUnit Test Runner Class

- As *Cucumber* uses *Junit* we need to have a **Test Runner class**.
- It more like a starting point for Junit to start executing your tests.
- *@RunWith annotation* tells *JUnit* that tests should run using **Cucumber class** present in '**Cucumber.api.junit**' package.
- The **@CucumberOptions** present in '***cucumber.api.CucumberOptions***' package tells Cucumber a lot of things like where to look for feature files, what reporting system to use and some other things also.

Test Runner Class

```
2
3  import org.junit.runner.RunWith;
4  import cucumber.api.CucumberOptions;
5  import cucumber.api.junit.Cucumber;
6
7  @RunWith(Cucumber.class)
8  @CucumberOptions(
9   features = "Feature"
10  ,glue={"stepDefinition"}
11  )
12
13 public class TestRunner {
14
15 }
```

JUnit Test Runner Class - @CucumberOptions

- In layman language, **@CucumberOptions** are like property files or settings for your test.
- Basically *@CucumberOptions* enables us to do all the things that we could have done if we have used cucumber command line.
- This is very helpful and of utmost importance, if we are using IDE such eclipse only to execute our project.

WE CAN SAY THAT *@CUCUMBEROPTIONS* ARE USED TO SET SOME SPECIFIC PROPERTIES FOR THE *CUCUMBER* TEST. FOLLOWING MAIN OPTIONS ARE AVAILABLE IN CUCUMBER:

Options Type	Purpose	Default Value
dryRun	true: Checks if all the Steps have the Step Definition	false
features	set: The paths of the feature files	{}
glue	set: The paths of the step definition files	{}
tags	instruct: What tags in the features files should be executed	{}
monochrome	true: Display the console Output in much readable way	false
format	set: What all report formaters to use	false
strict	true: Will fail execution if there are undefined or pending steps	false

THE FIRST CUCUMBER TEST CASE!

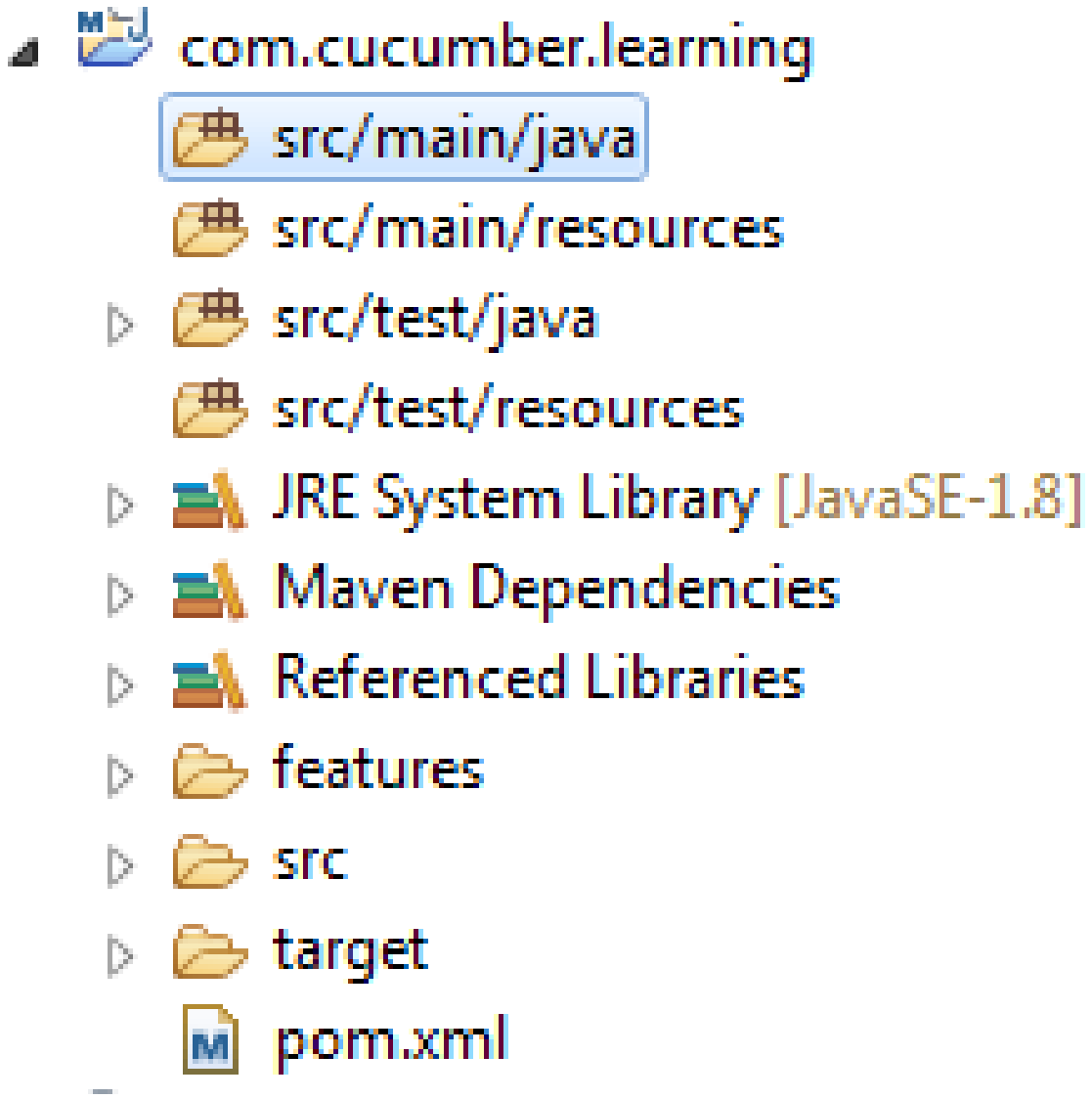
TASK: OPEN GOOGLE.COM USING SELENIUM-CUCUMBER AND PERFORM A SEARCH OPERATION

Steps for writing the first test case for our task using Cucumber:

STEP 1:

- Create a new Maven project in Eclipse and give a suitable name.
- (com.cucumber.learning) is my project name. You can even create a normal Java project but adding and removing the referenced libraries will be some difficult task.
- If we use Maven, managing dependencies will be very easier with pom.xml.

It will take some time and once the project is created you can see the project structure like this.



STEP 2:

- We have **not added the necessary jars to the project**. So, to add those jars, all we have to do is **include the maven dependencies in the pom.xml** file from mvn repository and you can see the jars are getting downloaded and once it is finished you can see the maven dependencies.

OpenGoogle.feature | RunnerClass.java | OpenGoogle.java | cucumberlearning/pom.xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:sc
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.cucumber.learning</groupId>
  <artifactId>cucumberlearning</artifactId>
  <version>0.0.1-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>3.14.0</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.11</version>
        </dependency>
        <dependency>
            <groupId>io.cucumber</groupId>
            <artifactId>cucumber-java</artifactId>
            <version>6.10.1</version>
        </dependency>
        <dependency>
            <groupId>io.cucumber</groupId>
            <artifactId>cucumber-junit</artifactId>
            <version>6.10.1</version>
        </dependency>
    </dependencies>
</project>
```

STEP 3:

- Under the project create a folder named "**features**" this is where we are going to create our feature files.
- Inside the features folder, let us create our first feature file called "**OpenGoogle.feature**".
- In the newly created feature file, we are going to write the steps (behaviours) for our test scenario.

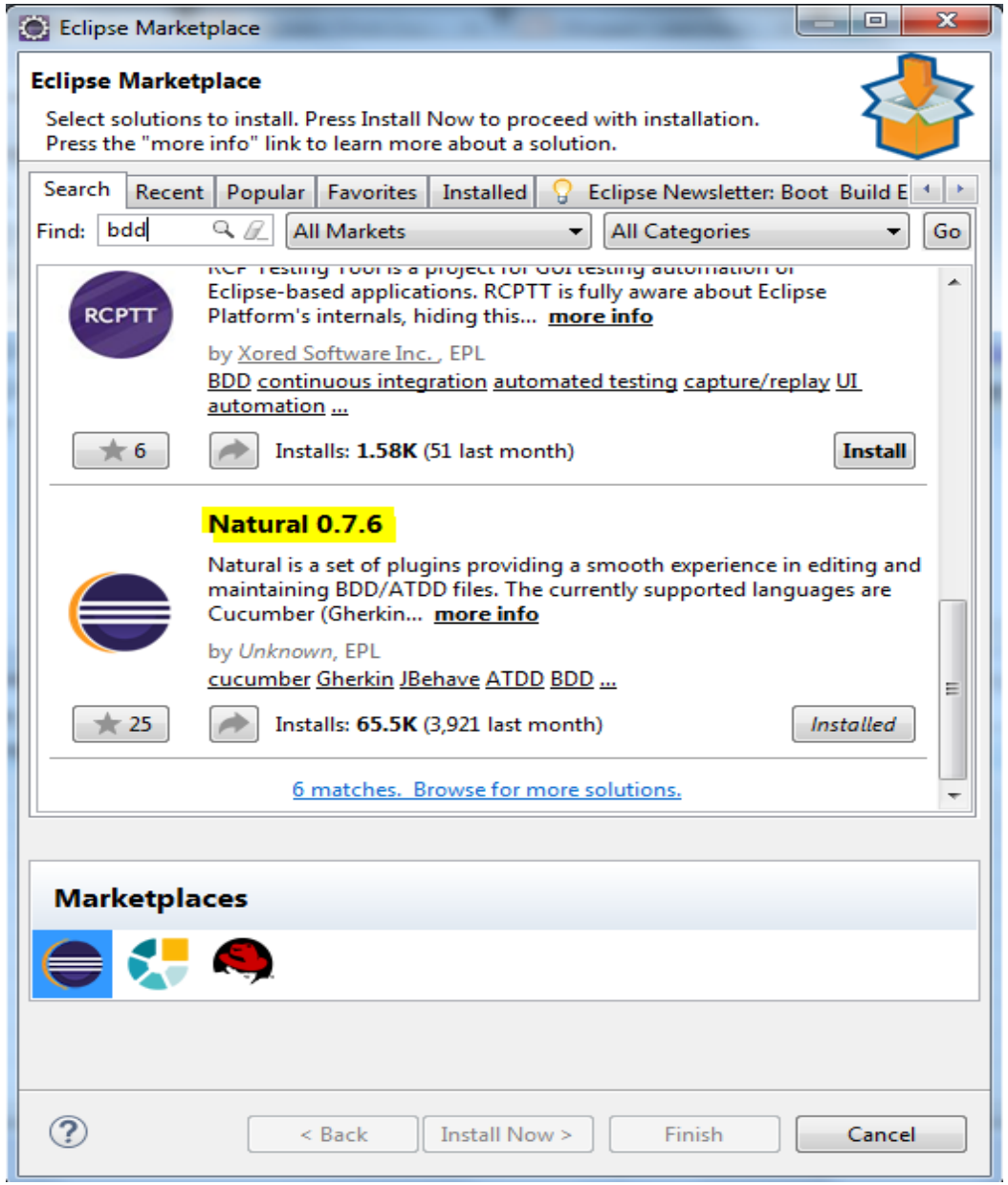
e - Eclipse IDE

OpenGoogle.feature

```
1 Feature: open google.com feature
2 Scenario: open google website and search
3  Given user is entering google.com
4  When user clicks the search box
5  And types the search term as "Cucumber"
6  Then the user should see the search results for Cucumber
```

STEP 4:

- If you are getting only black text and no other color differentiation in feature file , then that is not a big issue.
- I Go to your Eclipse marketplace and search for BDD, you will get some results and install a plugin called **Natural**.



STEP 5:

- After writing the feature file, we have to **write the step definitions** for the feature file. Step definition class is nothing but **the actual code implementation of plain English** that we have given in the feature file.
- Under **src/test/java** create a package named "**stepDefinitons**" and inside that create a class called "**OpenGoogle**".

OpenGoogle.feature | RunnerClass.java | OpenGoogle.java | cucumberlearning/pom.xml

```java
package stepDefinitons;

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

public class OpenGoogle {
 @Given("user is entering google.com")
 public void user_is_entering_google_com() throws Throwable {
     // Write code here that turns the phrase above into concrete actions
    System.out.println("User is on Google home");
 }

 @When("user clicks the search box")
 public void user_clicks_the_search_box() throws Throwable {
     // Write code here that turns the phrase above into concrete actions
   System.out.println("User is clicking the search box");
 }

 @When("types the search term as \"([^\"]*)\"$")
 public void types_the_search_term_as(String arg1) throws Throwable {
     // Write code here that turns the phrase above into concrete actions
   System.out.println("User is typing the search term");
 }

 @Then("the user should see the search results for Cucumber")
 public void the_user_should_see_the_search_results_for_Cucumber() throws Throwable {
     // Write code here that turns the phrase above into concrete actions
   System.out.println("User is seeing the search results");
 }

}
```

STEP 6:

- Now under, **src/test/java** create another package called "**runner**" and inside that create a class named "**RunnerClass**".
- This is where we run all our tests. This will be the entry point of our program.
- Run the RunnerClass as **JUnit Test**. You can see the steps that we have written in our feature file are automatically getting mapped to a java method with the **Given, When, And, Then annotations**

OpenGoogle.feature | RunnerClass.java | OpenGoogle.java | cucumberlearning/pom.xml

```java
package runner;

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


@RunWith(Cucumber.class)
@CucumberOptions(features="features",
glue= {"stepDefinitions"}
)
public class RunnerClass {

}
```

STEP 7:

- ***Run the Cucumber Test***

Even from the IDE, there are a couple of ways to run these feature files.

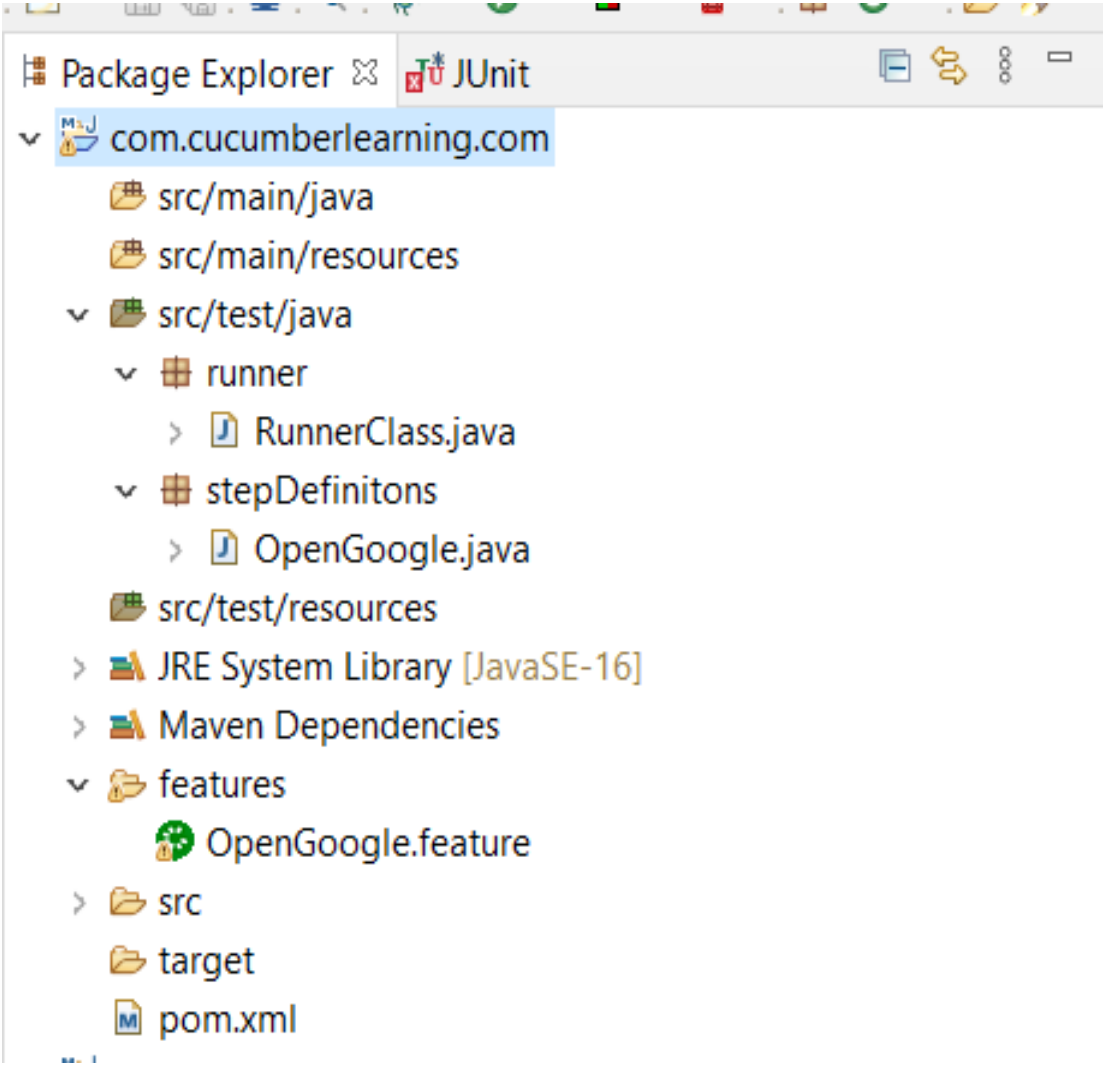
- *Click on the **Run** button on eclipse and you have your test run*
- *Right Click on **Runner** class and Click **Run As** > **JUnit Test Application***

CONSOLE OUTPUT:

```
@ Javadoc  Declaration  Console
<terminated> test.feature [Cucumber Feature] C:\Users\Navii\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.2.v20210721-1149\jre\bin\javaw.exe  (Oct 7, 2021, 11:35:31 AM –
Oct 07, 2021 11:35:31 AM cucumber.api.cli.Main run
WARNING: You are using deprecated Main class. Please use io.cucumber.core.cli.Main

Scenario: open google website and search                # features/OpenGoogle.feature:2
User is on Google home
  Given user is entering google.com                     # stepDefinitons.OpenGoogle.user_is_entering_google_com()
User is clicking the search box
  When user clicks the search box                       # stepDefinitons.OpenGoogle.user_clicks_the_search_box()
User is typing the search term
  And types the search term as "Cucumber"               # stepDefinitons.OpenGoogle.types_the_search_term_as(java.lang.String)
User is seeing the search results
  Then the user should see the search results for Cucumber # stepDefinitons.OpenGoogle.the_user_should_see_the_search_results_for_Cucumber()

1 Scenarios (1 passed)
4 Steps (4 passed)
0m0.421s


????????????????????????????????????????????????????????????????????????????????????????????????
? Share your Cucumber Report with your team at https://reports.cucumber.io         ?
```

CUCUMBER

STRUCTURE



Data Driven Testing in Cucumber

Cucumber inherently supports **Data Driven Testing using Scenario Outline**.

There are different ways to use the data insertion within the *Cucumber* and outside the *Cucumber* with external files.

Data-Driven Testing in Cucumber

- *Parameterization without Example Keyword*

Data-Driven Testing in Cucumber using Scenario Outline

- *Parameterization with Example Keyword*
- *Parameterization using Tables*

Data-Driven Testing in Cucumber using External Files

- *Parameterization using Excel Files*
- *Parameterization using Json*
- *Parameterization using XML*

Scenario Outline – This is used to run the same scenario for 2 or more different sets of test data. **E.g**. In our scenario, if you want to register another user you can data drive the same scenario twice.

LogIn_Test.fetaure

```
2
3  Scenario: Successful Login with Valid Credentials
4   Given User is on Home Page
5   When User Navigate to LogIn Page
6   And User enters UserName and Password
7   Then Message displayed Login Successfully
8
9  Scenario: Successful LogOut
10  When User LogOut from the Application
11  Then Message displayed LogOut Successfully
```

DATA-DRIVEN TESTING - Example

TestRunner.java

```java
package cucumberTest;

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

@RunWith(Cucumber.class)
@CucumberOptions(
 features = "Feature"
 ,glue={"stepDefinition"}
 )

public class TestRunner {

}
```

DATA-DRIVEN TESTING - Example

Test_Steps.java

```java
package stepDefinition;

import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
public class Test_Steps {
        public static WebDriver driver;
    @Given("^User is on Home Page$")
    public void user_is_on_Home_Page() throws Throwable {
        driver = new FirefoxDriver();
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        driver.get("https://www.store.demoqa.com");
    }
    @When("^User Navigate to LogIn Page$")
    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$")
```

Test_Steps.java

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

DATA-DRIVEN TESTING - Example

Parameterizing without Example Keyword

LogIn_Test.feature

```
Feature: Login Action

Scenario: Successful Login with Valid Credentials
 Given User is on Home Page
 When User Navigate to LogIn Page
 And User enters "testuser_1" and "Test@123"
 Then Message displayed Login Successfully

Scenario: Successful LogOut
 When User LogOut from the Application
 Then Message displayed LogOut Successfully
```

1) Go to the ***Feature File*** and change the statement where passing *Username & Password* as per below:

And User enters "testuser_1" and "Test@123"

In the above statement, we have passed *Username & Password* from the *Feature File* which will feed in to *Step Definition* of the above statement automatically.

Parameterizing without Example Keyword

2) Changes in the *Step Definition* file is also required to make it understand the *Parameterization of the feature file*. So, it is required to update the *Test Step* in the *Step Definition* file which is linked with the above-changed *Feature* file statement. Use the below code:

```
@When("^User enters \"(.*)\" and \"(.*)\"$")
```

The same can be achieved by using the below code as well:

```
@When("^User enters \"([^\"]*)\" and \"([^\"]*)\"$")
```

Parameterizing without Example Keyword

3) Same parameters should also go into the associated *Test_Step*.

As the Test step is nothing but a simple Java method, syntax to accept the parameter

in the Java method is like this:

```
public void user_enters_UserName_and_Password(String username, String password) throws Throwable {
}
```

Parameterizing without Example Keyword

4) Now the last step is to feed the parameters in the actual core statements of *Selenium WebDriver*. Use the below code:

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

driver.findElement(By.id("pwd")).sendKeys(password);

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

```
@When("^User enters \"(.*)\" and \"(.*)\"$")
public void user_enters_UserName_and_Password(String username, String password) thr
driver.findElement(By.id("log")).sendKeys(username);
    driver.findElement(By.id("pwd")).sendKeys(password);
    driver.findElement(By.id("login")).click();
}
```

5) Run the test by *Right Click* on **TestRunner class** and Click **Run As > JUnit Test** Application.

You would notice that the *Cucumber* will open the Website in the browser and enter *username & password* which is passed from the *Feature File*.

Data Driven Testing Using Examples Keyword

1) Enter the ***Example Data*** just below the *LogIn* Scenario of the *Feature* File.

Examples:

username	***password***
testuser_1	***Test@153***
testuser_2	***Test@153***

Note*: The table must have a header row corresponding to the variables in the Scenario Outline steps.*

The Examples section is a table where each argument variable represents a column in the table, separated by "**|**". Each line below the header represents an individual run of the test case with the respective data.

Data Driven Testing Using Examples Keyword

2) Need to update the Statement in the *feature* file, which tells *Cucumber* to enter *username & Password*.

And User enters <username> and <password>

The complete code will look like this:

```
Feature: Login Action

Scenario Outline: 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
Examples:
    | username   | password |
    | testuser_1 | Test@153 |
    | testuser_2 | Test@153 |
```

Data Driven Testing Using Examples Keyword

3) There are no changes in **TestRunner** class.

4) There are no changes in **Test_Steps** file.

5) Run the test by *Right Click* on ***TestRunner class*** and Click ***Run As > JUnit Test*** Application.

This takes the *parameterization* one step further: now our scenario has "***variables***" and they get filled in by the values in each row.

Data Tables in Cucumber

Scenario Outline:

- *This uses Example keyword to define the test data for the Scenario*
- *This works for the whole test*
- *Cucumber automatically run the complete test the number of times equal to the number of data in the Test Set*

Test Data:

- *No keyword is used to define the test data*
- *This works only for the single step, below which it is defined*
- *A separate code needs to understand the test data and then it can be run single or multiple times but again just for the single step, not for the complete test*

- *Data Tables* can be used in many ways because it has provided many different methods to use.
- we will pass the test data using the *data table* and handle it using **Raw()** method

Problems @ Javadoc Declaration Console

```
<terminated> TestRunner [JUnit] C:\Program Files\Java\jre1.8.0_65\bin\javaw.exe (15 Dec 2015, 18:28:32)

You can implement missing steps with the snippets below:

@When("^User enters Credentials to LogIn$")
public void user_enters_Credentials_to_LogIn(DataTable arg1) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    // For automatic transformation, change DataTable to one of
    // List<YourType>, List<List<E>>, List<Map<K,V>> or Map<K,V>.
    // E,K,V must be a scalar (String, Integer, Date, enum etc)
    throw new PendingException();
}
```

- The complete scenario is same as what we have done earlier. But the only difference is in this, we are not passing parameters in the step line and even we are not using Examples test data. We declared the data under the step only. So, we are using Tables as arguments to Steps.
- If you run the above scenario without implementing the step, you would get the following error in the Eclipse console window.

Problems @ Javadoc Declaration Console

```
<terminated> TestRunner [JUnit] C:\Program Files\Java\jre1.8.0_65\bin\javaw.exe (15 Dec 2015, 18:28:32)

You can implement missing steps with the snippets below:

@When("^User enters Credentials to LogIn$")
public void user_enters_Credentials_to_LogIn(DataTable arg1) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    // For automatic transformation, change DataTable to one of
    // List<YourType>, List<List<E>>, List<Map<K,V>> or Map<K,V>.
    // E,K,V must be a scalar (String, Integer, Date, enum etc)
    throw new PendingException();
}
```

The implementation of these step will be like this:

```
The implementation of the above step will belike this:
 @When("^User enters Credentials to LogIn$")
 public void user_enters_testuser__and_Test(DataTable usercredentials) throws Throw

 //Write the code to handle Data Table
 List<List<String>> data = usercredentials.raw();

 //This is to get the first data of the set (First Row + First Column)
 driver.findElement(By.id("log")).sendKeys(data.get(0).get(0));

 //This is to get the first data of the set (First Row + Second Column)
     driver.findElement(By.id("pwd")).sendKeys(data.get(0).get(1));

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

Maps in Data Tables

Maps in Data Tables

Maps in Data Tables can be used if different ways. ***Headers*** can also be defined for the *data tables.* A same step can be executed multiple times with different set of test data using ***Maps***.

Maps in Data Tables with Multiple Test Data

In this test we will pass *Username and Password* two times to the test step. So our test should enter *Username & Password* once, click on *LogIn* button and repeat the same steps again.

IMPLEMENTATION OF MAPS IN DATA TABLES

Feature File Scenario

```
Scenario: Successful Login with Valid Credentials
 Given User is on Home Page
 When User Navigate to LogIn Page
 And User enters Credentials to LogIn
 | Username    | Password |
    | testuser_1 | Test@153 |
    | testuser_2 | Test@154 |
 Then Message displayed Login Successfully
```

The implementation of the above step will be like this:

```java
public void user_enters_testuser_and_Test(DataTable usercredentials) throws Throwal

//Write the code to handle Data Table
for (Map<String, String> data : usercredentials.asMaps(String.class, String.class)
driver.findElement(By.id("log")).sendKeys(data.get("Username"));
    driver.findElement(By.id("pwd")).sendKeys(data.get("Password"));
    driver.findElement(By.id("login")).click();
}

}
```

Map Data Tables to Class Objects

- Luckily there are easier ways to access your data than *Data Table*.
- For instance, you can create a *Class-Object* and have Cucumber map the data in a table to a list of these.

Feature File Scenario

```
Scenario: Successful Login with Valid Credentials
 Given User is on Home Page
 When User Navigate to LogIn Page
 And User enters Credentials to LogIn
 | Username    | Password |
    | testuser_1 | Test@153 |
    | testuser_2 | Test@154 |
 Then Message displayed Login Successfully
```

The implementation of the above step will be like this:

```
@When("^User enters Credentials to LogIn$")
public void user_enters_testuser_and_Test(List<Credentials>  usercredentials) throw

//Write the code to handle Data Table
for (Credentials credentials : usercredentials) {
driver.findElement(By.id("log")).sendKeys(credentials.getUsername());
    driver.findElement(By.id("pwd")).sendKeys(credentials.getPassword());
    driver.findElement(By.id("login")).click();
}
}
```

What are Tags in Cucumber?

- Features and Scenarios can be marked with Tags
- Tags use @ symbol with some text e.g. @SmokeTest
- In the test runner we can run specific tags
- A feature or scenario can have multiple tags

Features Of Tags

Runs with single OR multiple Tags

Runs with a combination of tags or using AND, OR conditions

Can skip scenarios having specific Tag

FEATURE File

~ Saved in .feature Extension

```
Feature: Feature to demo tags
@smoke
Scenario:Sample1
 Given
 When
 And

  @regression
Scenario:Sample2
 Given
 When
 And

  @smoke @regression
Scenario:Sample3
 Given
 When
 And
```

Runner File

```
import org.junit.runner.RunWith;
import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;


@RunWith(Cucumber.class)
@CucumberOptions(features="features",
glue= {"stepDefinitions"},
plugin = {"json:target/cucumber.json",
          "html:target/HtmlReports"
},
tags = "@smoke or not @regression"
)
public class Runner {

}
```

Types Of Tags

Single tag

- tags = {"@smoke"}

Multiple tags

Tags with AND OR conditions

- tags = {"@smoke or @regression"}
- tags = {"@smoke and @regression"}
- tags = {"@smoke and not @regression"}

To Skip or Ignore Tags

tags = {"(@smoke or @regression) and not @important"}

Useful tips

Tags

- Tags can be placed below the following gherkin elements:

- Feature
- Scenario
- Scenario Outline

Examples

- It is not possible to place tags above Background or steps (Given, When, Then, And and But

Useful Tips

Tag Inheritance

- Tags are inherited by child elements.
- Tags that are placed above a feature will be inherited by scenario, scenario outline, or examples.
- Tags that are placed above a scenario outline will be inherited by examples

What are Hooks in Cucumber?

Blocks of code that runs before OR after each scenario

Hooks in cucumber are like listeners in testing

Can define hooks by using annotations @before @after

Types Of Hooks

Scenario Hooks

- Runs before and after each scenario

Step Hooks

- Runs before and after each step

Conditional Hooks

- Hooks associated with tags for conditional execution

Why to use HOOKS?

To manage the setup and teardown

To avoid rewriting the common setup or teardown actions

Allow better management of code workflow

How To Use Hooks?

Step 1 - create a new or use an existing feature file

Step 2 - create the steps for the scenario in the feature file

Step 3 - create setup and teardown methods and mark with annotation

@Before

@After

@Beforesteps

@Aftersteps

Step 4 - create new or use an existing testrunner class

Step 5 - run the testrunner class and check execution

Conditional Hooks

Conditional Hooks

- Hooks can be conditionally selected for execution based on the tags of the scenario
- To run a particular hook only for certain scenarios, you can associate a Before or After hook with a tag expression
- Tags can be used with
  - @BeforeSteps
  - @AfterSteps
  - @After(value="@smoke", order=2)

Ordering Hooks

We can use multiple Before and After hooks and also assign order of execution

- @Before(order=0)
- @Before(order=1)

```java
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import io.cucumber.java.After;
import io.cucumber.java.AfterStep;
import io.cucumber.java.Before;
import io.cucumber.java.BeforeStep;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
public class Steps {
    WebDriver driver=null;
    @Before(order=1)
    public void browserSetup() throws Throwable{
        System.out.println("i am inside browser setup");
        System.setProperty("webdriver.chrome.driver","E:\\selenium\\chromedriver.exe");
         driver=new ChromeDriver();
         driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
         driver.manage().timeouts().pageLoadTimeout(20, TimeUnit.SECONDS);
         driver.manage().window().maximize();
    }
    @Before(order=2)
    public void setup2() {
        System.out.println("i am inside browser setup2");
    }
    @After(order=2)
    public void tearDown2() {
        System.out.println("i am inside teardown2");
    }
    @After(order=1)
    public void tearDown() {
        System.out.println("i am inside teardown");
        driver.close();
        driver.quit();
    }
    @BeforeStep
    public void beforeSteps() {
        System.out.println("i am inside beforestep");
    }
```

```java
    @After(order=1)
    public void tearDown() {
        System.out.println("i am inside teardown");
        driver.close();
        driver.quit();
    }
    @BeforeStep
    public void beforeSteps() {
        System.out.println("i am inside beforestep");
    }
    @AfterStep
    public void afterSteps() {
        System.out.println("i am inside afterstep");

    }

    @Given("use is on login page")
    public void use_is_on_login_page() {
        System.out.println("---------use_is_on_login_page-----------");

    }
    @When("user enters valid username and password")
    public void user_enters_valid_username_and_password() {
     System.out.println("----------user_enters_valid_username_and_password----------------");
    }
    @When("clicks on login button")
    public void clicks_on_login_button() {
        System.out.println("-----------clicks_on_login_button-------------");
    }
    @Then("user is navigated to the home page")
    public void user_is_navigated_to_the_home_page() {
        System.out.println("-------------------user_is_navigated_to_the_home_page----------------");
    }
```

Example Program - Hooks

Background in Cucumber

- ***Background in Cucumber*** is used to define a step or series of steps that are common to all the tests in the feature file.
- It allows you to add some context to the scenarios for a feature where it is defined.
- A Background is much like a scenario containing a number of steps.
- But it runs before each and every scenario were for a feature in which it is defined.

Background in Cucumber – Example Scenerio

For example,

To purchase a product on any E-Commerce website, you need to do the following steps:

Navigate to Login Page

Submit UserName and Password

After these steps only you will be able to add a product to your *cart/basket* and able to perform the payment. Now as we are in a feature file where we will be testing only the *Add to Cart* or *Add to Bag* functionality, these tests become common for all tests. So instead of writing them again and again for all tests, we can move it under the *Background* keyword

Feature File

```
Feature: Test Background Feature
Description: The purpose of this feature is to test the Background keyword

Background: User is Logged In
 Given I navigate to the login page
 When I submit username and password
 Then I should be logged in

Scenario: Search a product and add the first product to the User basket
 Given User search for Lenovo Laptop
 When Add the first laptop that appears in the search result to the basket
 Then User basket should display with added item

Scenario: Navigate to a product and add the same to the User basket
 Given User navigate for Lenovo Laptop
 When Add the laptop to the basket
 Then User basket should display with added item
```

Background in Cucumber – Example Scenerio

- In the above example, we have two different scenarios where a user is adding a product from search and directly from the product page.

  But the common step is to logIn to website for both the scenario.

  Therefore, we create another Scenario for Login but named it as Background rather than a Scenario.

  So that it executes for both the Scenarios.

StepDefinition

```java
package stepDefinition;

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

public class BackGround_Steps {

 @Given("^I navigate to the login page$")
 public void i_navigate_to_the_login_page() throws Throwable {
 System.out.println("I am at the LogIn Page");
 }

 @When("^I submit username and password$")
 public void i_submit_username_and_password() throws Throwable {
 System.out.println("I Submit my Username and Password");
 }

 @Then("^I should be logged in$")
 public void i_should_be_logged_in() throws Throwable {
 System.out.println("I am logged on to the website");
 }

 @Given("^User search for Lenovo Laptop$")
 public void user_searched_for_Lenovo_Laptop() throws Throwable {
 System.out.println("User searched for Lenovo Laptop");
 }

 @When("^Add the first laptop that appears in the search result to the basket$")
 public void add_the_first_laptop_that_appears_in_the_search_result_to_the_basket()
 System.out.println("First search result added to bag");
 }

 @Then("^User basket should display with added item$")
 public void user_basket_should_display_with_item() throws Throwable {
 System.out.println("Bag is now contains the added product");
 }
```

Output

```
Feature: Test Background Feature
  Description: The purpose of this feature is to test the Background keyword

I am at the LogIn Page
I Submit my Username and Password
I am logged on to the website
User searched for Lenovo Laptop
First search result added to bag
Bag is now contains the added product

I am at the LogIn Page
I Submit my Username and Password
I am logged on to the website
User navigated for Lenovo Laptop
Laptop added to the basket
Bag is now contains the added product
```

Cucumber Reports

- When ever we do test execution, it is also required to understand the output of the execution. Whether it is Manual execution or an Automated, the output of the same must be in format, which immediately depicts the overall results of the execution.
- Hence, our framework also should have the same capability to create output or generate test execution reports.
- It is essential to know, how better we can generate our Cucumber test reports.
- As we know that Cucumber is a BDD framework, it does not have a fancy reporting mechanism. In order to achieve this, Cucumber itself has provided a nice feature to generate reports.

Complete *TestRunner* would look like this:

TestRunner.java

```
1  package runners;
2
3  import org.junit.runner.RunWith;
4  import cucumber.api.CucumberOptions;
5  import cucumber.api.junit.Cucumber;
6
7  @RunWith(Cucumber.class)
8  @CucumberOptions(
9   features = "src/test/resources/functionalTests",
10  glue= {"stepDefinitions"},
11  plugin = { "pretty" },
12  monochrome = true
13  )
14
15 public class TestRunner {
16
17 }
```

Eclipse Console Output

```
JQuery call is in Progress
JQuery call is in Progress

Scenario Outline: Customer place an order by purchasing an item from search [90m# End2End_Tests.feature:16[0m
  [32mGiven [0m[32muser is on Home Page[0m                                    [90m# HomePageSteps.user_is_on_Home_Page()[
  [32mWhen [0m[32mhe search for "[0m[32m[1mdress[0m[32m"[0m                                  [90m# HomePageSteps.h
  [32mAnd [0m[32mchoose to buy the first item[0m                              [90m# ProductPageSteps.choose_to_buy_the_fi
  [32mAnd [0m[32mmoves to checkout from mini cart[0m                          [90m# CartPageSteps.moves_to_checkout_from_
  [32mAnd [0m[32menter "[0m[32m[1mLakshay[0m[32m" personal details on checkout page[0m          [90m# CheckoutPageSte
  [32mAnd [0m[32mselect same delivery address[0m                              [90m# CheckoutPageSteps.select_same_delivery
  [32mAnd [0m[32mselect payment method as "[0m[32m[1mcheck[0m[32m" payment[0m                  [90m# CheckoutPageSte
  [32mAnd [0m[32mplace the order[0m                                           [90m# CheckoutPageSteps.place_the_order()[
  [32mThen [0m[32mverify the order details[0m                                 [90m# ConfirmationPageSteps.verify_the_orde

1 Scenarios ([32m1 passed[0m)
9 Steps ([32m9 passed[0m)
0m38.094s
```

Pretty Report

The first plugin, we will talk about is ***Pretty.*** This provides more verbose output. To implement this, just specify *plugin = "pretty"* in *CucumberOptions*. This is what the code looks like:

@CucumberOptions(plugin = { "pretty" })

Full *CucumberOption* code will be like this:

TestRunner.java | CucumberFramework/pom.xml

```
1  package runners;
2
3  import org.junit.runner.RunWith;
7
8  @RunWith(Cucumber.class)
9  @CucumberOptions(
10         features = "src/test/resources/functionalTests",
11         glue= {"stepDefinitions"},
12         plugin = { "pretty" },
13         monochrome = true
14         )
15
16
17 public class TestRunner {
18
19 }
```

Console

```
<terminated> TestRunner (4) [JUnit] C:\Program Files\Java\jre1.8.0_144\bin\javaw.exe (Jan 20, 2018, 9:13:35 AM)

Scenario Outline: Customer place an order by purchasing an item from search # End2End_Tests.feature:16
  Given user is on Home Page                                                # HomePageSteps.user_is_on_Home_Page()
  When he search for "dress"                                                # HomePageSteps.he_search_for(String)
  And choose to buy the first item                                          # ProductPageSteps.choose_to_buy_the_first_item()
  And moves to checkout from mini cart                                      # CartPageSteps.moves_to_checkout_from_mini_cart()
  And enter "Lakshay" personal details on checkout page                     # CheckoutPageSteps.enter_personal_details_on_checkout_page(String)
  And select same delivery address                                          # CheckoutPageSteps.select_same_delivery_address()
  And select payment method as "check" payment                              # CheckoutPageSteps.select_payment_method_as_payment(String)
  And place the order                                                       # CheckoutPageSteps.place_the_order()
  Then verify the order details                                             # ConfirmationPageSteps.verify_the_order_details()

1 Scenarios (1 passed)
9 Steps (9 passed)
0m32.936s
```

Monochrome Mode Reporting

If the monochrome option is set to false, then the console output is not as readable as it should be. The output when the monochrome option is set to false is shown in the above example. It is just because, if the *monochrome* is not defined in *Cucumber Options*, it takes it as ***false by default***. How to specify it:

@CucumberOptions(monochrome = true);

Full *CucumberOption* code will be like this:

```
@CucumberOptions(
 features = "src/test/resources/functionalTests",
 glue= {"stepDefinitions"},
 plugin = { "usage" },
 monochrome = true
)
```

This is what the output looks like:

Console

<terminated> TestRunner (4) [JUnit] C:\Program Files\Java\jre1.8.0_144\bin\javaw.exe (Jan 20, 2018, 4:54:20 PM)


```
  },
  {
    "source": "^choose to buy the first item$",
    "steps": [
      {
        "name": "choose to buy the first item",
        "aggregatedDurations": {
          "average": 7.679406091,
          "median": 7.679406091
        },
        "durations": [
          {
            "duration": 7.679406091,
            "location": "End2End_Tests.feature:7"
          }
        ]
      }
    ]
  },
  {
    "source": "^moves to checkout from mini cart$",
    "steps": [
      {
        "name": "moves to checkout from mini cart",
        "aggregatedDurations": {
          "average": 5.153910466,
          "median": 5.153910466
        },
        "durations": [
          {
            "duration": 5.153910466,
            "location": "End2End_Tests.feature:8"
          }
        ]
      }
    ]
  }
```


Usage Report

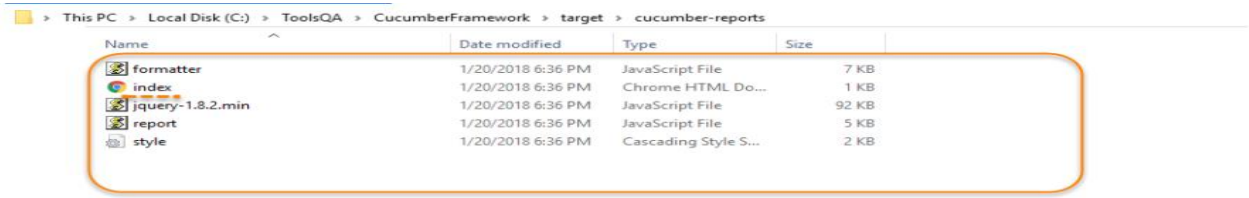
If we are more concerned about the time taken by each ***Step Definition***, then we should use the ***usage plugin***. This is how we specify the same in *@CucumberOptions:*

@CucumberOptions(plugin = { "usage" })

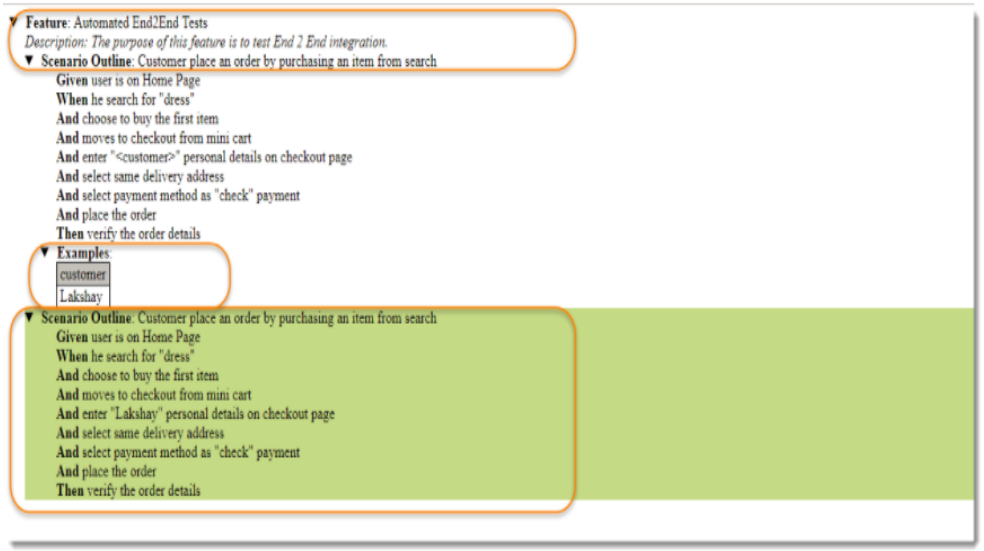
Full *CucumberOption* code will be like this:

```
@CucumberOptions(
 features = "src/test/resources/functionalTests",
 glue= {"stepDefinitions"},
 plugin = { "pretty", "html:target/cucumber-reports" },
 monochrome = true
)
```

Report Output Location



HTML Report Output



Cucumber HTML Reports

For HTML reports, add ***html:target/cucumber-reports*** to the *@CucumberOptions* plugin option

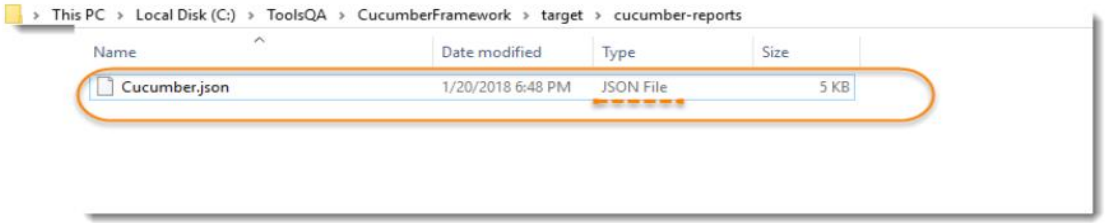
***Note**: We have specified the path of the Cucumber report, which we want it to generate it under the target folder.*

This will generate an HTML report at the location mentioned in the formatter itself.

Full *CucumberOption* code will be like this:

```
@CucumberOptions(
 features = "src/test/resources/functionalTests",
 glue= {"stepDefinitions"},
 plugin = { "pretty", "json:target/cucumber-reports/Cucumber.json" },
 monochrome = true
)
```

Report Output Location



JSON Report Output


```
    ],
    "type": "scenario",
    "keyword": "Scenario Outline",
    "steps": [
      {
        "result": {
          "duration": 5218914964,
          "status": "passed"
        },
        "line": 5,
        "name": "user is on Home Page",
        "match": {
          "location": "HomePageSteps.user_is_on_Home_Page()"
        },
        "keyword": "Given "
      },
      {
        "result": {
          "duration": 3555571608,
          "status": "passed"
        },
        "line": 6,
        "name": "he search for \"dress\"",
        "match": {
          "arguments": [
            {
              "val": "dress",
              "offset": 15
            }
          ],
          "location": "HomePageSteps.he_search_for(String)"
        },
        "keyword": "When "
      },
```


Cucumber JSON Report

For *JSON* reports,
add ***json:target/cucumber-reports/Cucumber.json*** to the *@CucumberOptions* plugin option.

Full *CucumberOption* code will be like this:

```
@CucumberOptions(
 features = "src/test/resources/functionalTests",
 glue= {"stepDefinitions"},
 plugin = { "pretty", "junit:target/cucumber-reports/Cucumber.xml" },
 monochrome = true
)
```

XML Report Output

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<testsuite failures="0" name="cucumber.runtime.formatter.JUnitFormatter" skipped="0" tests="1" time="38.624653">
<testcase classname="Automated End2End Tests" name="Customer place an order by purchasing an item from search" time="38.624653">
<system-out><![CDATA[Given user is on Home Page.....................................passed
When he search for "dress".....................passed
And choose to buy the first item.....................passed
And moves to checkout from mini cart.....................passed
And enter "Lakshay" personal details on checkout page.....................passed
And select same delivery address.....................passed
And select payment method as "check" payment.....................passed
And place the order.....................passed
Then verify the order details.....................passed
]]></system-out>
</testcase>
</testsuite>
```

Cucumber JUNIT XML Report

For *JUNIT* reports,
add ***junit:targe/cucumber-reports/Cucumber.xml*** to the *@CucumberOptions* plugin option.

Full *CucumberOption* code will be like this:

```
@CucumberOptions(
 features = "src/test/resources/functionalTests",
 glue= {"stepDefinitions"},
 plugin = { "pretty", "json:target/cucumber-reports/Cucumber.json",
 "junit:target/cucumber-reports/Cucumber.xml",
 "html:target/cucumber-reports"},
 monochrome = true
)
```

All Reports Together

We can even generate all reports together as well.

Cucumber Extent Report

- This is again an awesome plugin that is built on **Extent Report specially for Cucumber** by **Vimal Selvam**.
- This is why it is named as **Cucumber Extent Reporter.**
- This one is made to ease the implementation of *Extent Report* in *Cucumber Framework*.

Let’s start by implementing the same in our Selenium Cucumber Framework.

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

Add Extent Report library
Dependencies information can be taken from ***Maven Repository – Extent Report***.

```
<dependency>
 <groupId>com.aventstack</groupId>
 <artifactId>extentreports</artifactId>
 <version>3.1.2</version>
</dependency>
```

Step 1: Add Cucumber Extent Reporter library to Maven Project

This is really simple, as we have been using Maven Project, all we need to do is to ***add the dependencies in to the project POM file***. Dependencies information can be taken from ***Maven Repository – Cucumber Extent Reporter***.

1. Create a **New File** and name it as **extent-config.xml** by right click on the **configs** folder in the project. In this config file you can set many elements like :

- **Report Theme** *: <theme> : standard or dark*
- **Document Encoding** *: <encoding> : UFT-8*
- **Title of the Report** *: <documentTitle> : This will display on the Browser Tab*
- **Name of the Report***: <reportName>: This will display at the top of the Report*
- **Global Date Format** *: <dateFormat> : Like this yyyy-MM-dd*
- **Global Time Format** *: <timeFormat> : Like this HH:mm:ss*

Step 2 – Add Extent Config to the Project

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

1. Make an entry for the Path of the config in the **Configuration.properties** file.

reportConfigPath=C:/ToolsQA/CucumberFramework/configs/extent-config.xml
Note: *Make sure to edit the path as per your machine path.*

2. 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 Configurat
}
```

Step 3: Read the extent-config.xml path

1. Modify the runner class and add the *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 of the report.html.

2. Write Extent Reports

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

```
1  @AfterClass
2  public static void writeExtentReport() {
3  Reporter.loadXMLConfig(new File(FileReaderManager.getInstance().getConfigReader().g
4  }
```

Step 4: Modify TestRunner to Implement Cucumber Extent Reporter

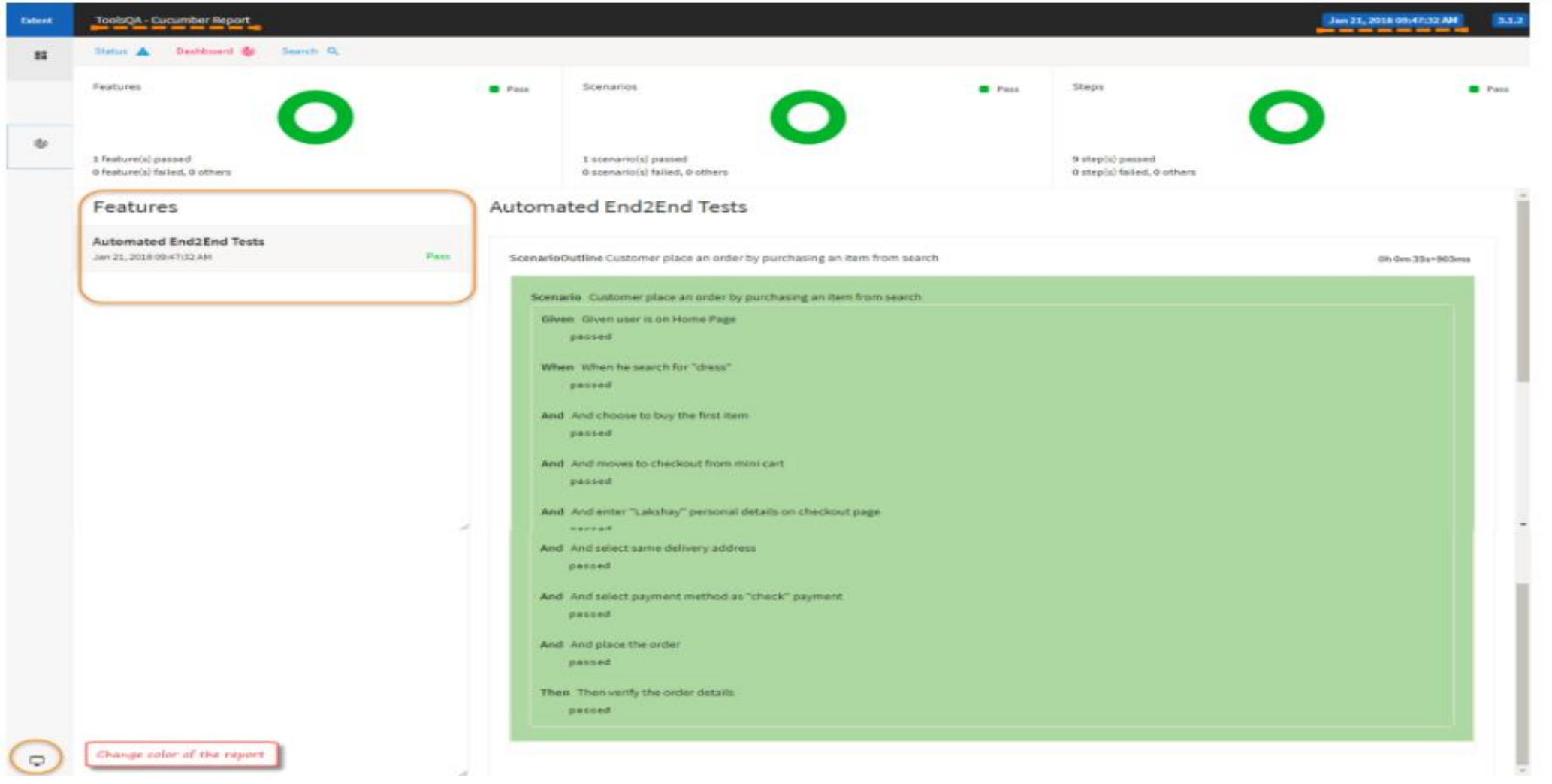
TestRunner,java

```
1  package runners;
2
3  import java.io.*;
4  import org.junit.AfterClass;
5  import org.junit.runner.RunWith;
6  import com.cucumber.listener.Reporter;
7  import cucumber.api.CucumberOptions;
8  import cucumber.api.junit.Cucumber;
9  import managers.FileReaderManager;
10
11 @RunWith(Cucumber.class)
12 @CucumberOptions(
13  features = "src/test/resources/functionalTests",
14  glue= {"stepDefinitions"},
15  plugin = { "com.cucumber.listener.ExtentCucumberFormatter:target/cucumber-reports/
16  monochrome = true
17  )
18
19
20 public class TestRunner {
21  @AfterClass
22  public static void writeExtentReport() {
23  Reporter.loadXMLConfig(new File(FileReaderManager.getInstance().getConfigReader().
24  }
25 }
```

Run as JUnit

Now we are all set to run the Cucumber test. Right Click on **TestRunner** class and Click **Run As >> JUnit Test.** Cucumber will run the script the same way it runs in Selenium WebDriver and the result will be shown in the left-hand side project explorer window in JUnit tab.

You may find the report at C:\ToolsQA\CucumberFramework\target\cucumber-reports folder



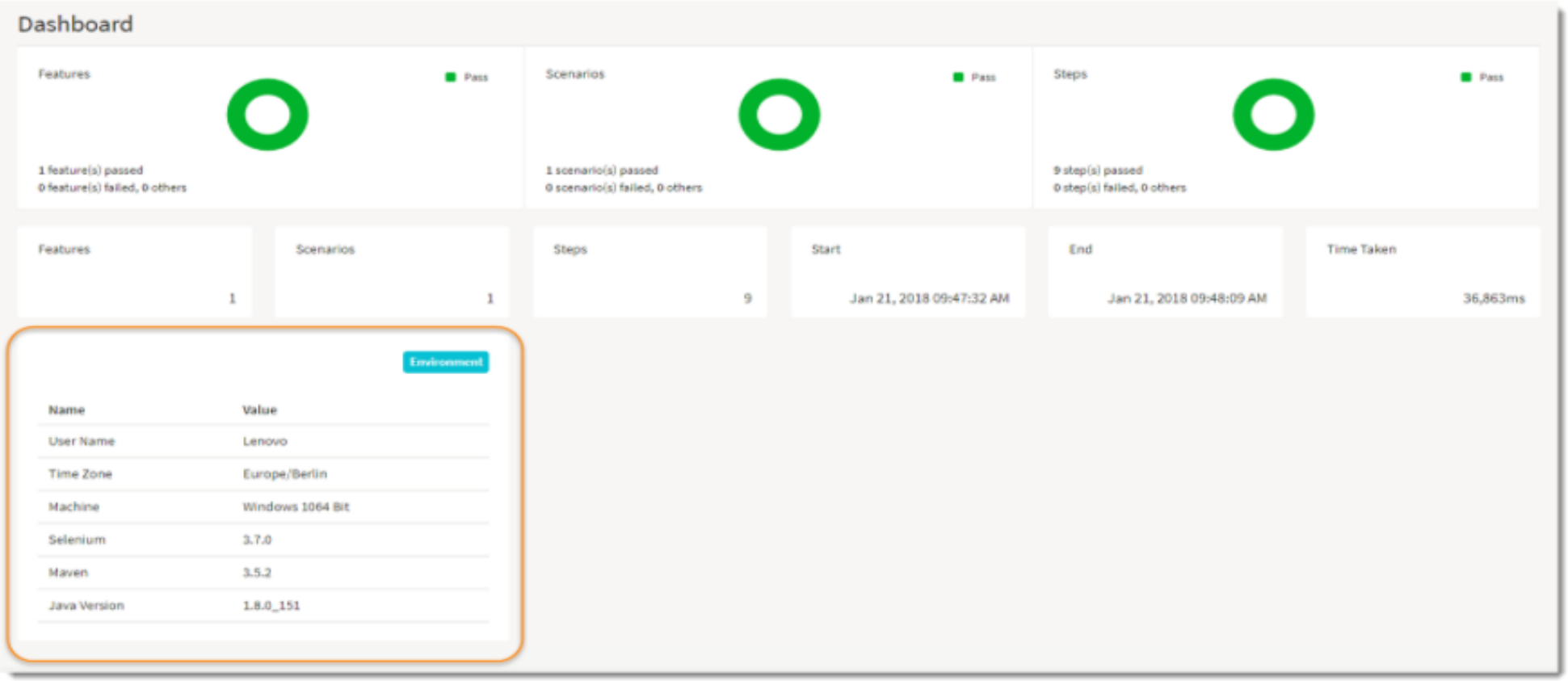
Cucumber Extent Reporter Features

This gives you nice feature to set multiple System properties to the report, so that you know under which system configurations your test suite was executed and when. To set this just make use of ***Reporter*** class and access its static method ***setSystemInfo()*** and pass it your information like below.

```
@AfterClass
public static void writeExtentReport() {
Reporter.loadXMLConfig(new File(FileReaderManager.getInstance().getConfigReader().
    Reporter.setSystemInfo("User Name", System.getProperty("user.name"));
    Reporter.setSystemInfo("Time Zone", System.getProperty("user.timezone"));
    Reporter.setSystemInfo("Machine", "Windows 10" + "64 Bit");
    Reporter.setSystemInfo("Selenium", "3.7.0");
    Reporter.setSystemInfo("Maven", "3.5.2");
    Reporter.setSystemInfo("Java Version", "1.8.0_151");
}
```

Set System Information in Report

Output

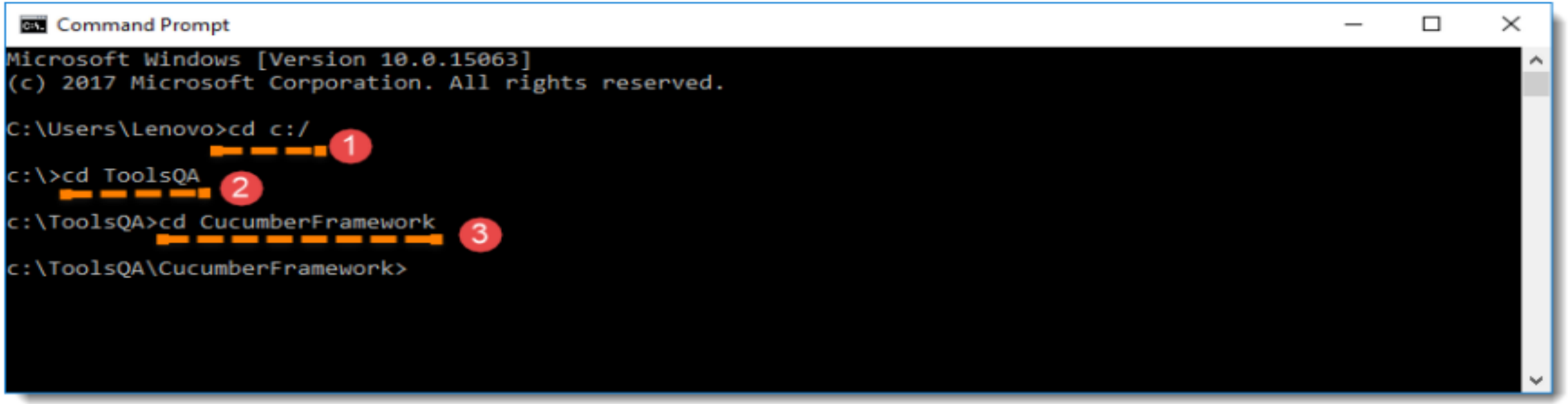


OUTPUT

Run Cucumber Test from Command Line / Terminal

- There are different ways to run Cucumber Test from command line.
- Tests can be run by using JUnit and Maven as well. But maven is the most suggested way and has extra benefits to it.
- This is why we started this Project as Maven project. And remember, Maven has a lot of advantages over other build tools, such as dependency management, lots of plugins and the convenience of running integration tests.
- Maven will allow our test cases to be run in different flavors, such as from the **Terminal**, integrating with **Jenkins**, and **parallel execution**.

To Run Test from Command Line:



1. Open the ***command prompt*** and ***cd*** until the project root directory.

Command Prompt

```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Lenovo>cd c:/

c:\>cd ToolsQA

c:\ToolsQA>cd CucumberFramework

c:\ToolsQA\CucumberFramework>mvn test
```

2. First, let's run all the Cucumber Scenarios from the *command prompt*. Since it's a Maven project and we have added Cucumber in **test scope** dependency and all features are also added in **src/test** packages, run the following command in the command prompt: **mvn test**

```
Command Prompt
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 7 source files to c:\ToolsQA\CucumberFramework\target\test-classes
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ CucumberFramework ---
[INFO] Surefire report directory: c:\ToolsQA\CucumberFramework\target\surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running runners.TestRunner
Starting ChromeDriver 2.33.506120 (e3e53437346286c0bc2d2dc9aa4915ba81d9023f) on port 18413
Only local connections are allowed.
Jan 21, 2018 4:02:19 PM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Detected dialect: OSS
JQuery call is in Progress
```

You would notice below that it actually triggered the **TestRunner** file.

```
Command Prompt

1 Scenarios (1 passed)
9 Steps (9 passed)
0m39.204s

Tests run: 10, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 40.715 sec

Results :

Tests run: 10, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 45.566 s
[INFO] Finished at: 2018-01-21T16:02:56+01:00
[INFO] Final Memory: 19M/194M
[INFO] ------------------------------------------------------------------------

c:\ToolsQA\CucumberFramework>
```

Build Success Output

What is REST ?

- Representational State Transfer in short-form as **REST** defines a set of constraints for creating Web Services.
- Rest API is the most-used web service technology nowadays, and it's an almost meaningless description.
- A REST API is a way to communicate for two computer systems over HTTP, which is similar to web browsers and servers.

REST API Testing

REST API testing is testing API using 4 major methods.

1. POST,
2. GET,
3. PUT and
4. DELETE.

rest-assured Dependency from central repository be like:

```
<!-- https://mvnrepository.com/artifact/io.rest-assured/rest-assured -->
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>4.3.3</version>
    <scope>test</scope>
</dependency>
```

pom.xml file:

```
<!-- https://mvnrepository.com/artifact/io.rest-assured/rest-assured -->
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>4.3.3</version>
    <scope>test</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-junit -->
<dependency>
      <groupId>io.cucumber</groupId>
      <artifactId>cucumber-junit</artifactId>
      <version>6.8.1</version>
      <scope>test</scope>
</dependency>

<dependency>
      <groupId>io.cucumber</groupId>
      <artifactId>cucumber-java</artifactId>
      <version>6.8.1</version>
      <scope>test</scope>
</dependency>

<dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
</dependency>
```

- You should place rest-assured before the JUnit dependency declaration in your pom.xml

  REST Assured includes JsonPath and XmlPath as transitive dependencies.

```
1  Feature: Validation of get method
2
3  @GetUserDetails
4    Scenario Outline: Send a valid Request to get user details
5    Given I send a request to the URL to get user details
6    Then the response will return status 200 and id <id>
7                  and salary <employee_salary> and name "<employee_name>"
8                  and age <employee_age> and message "<message>"
9  Examples:
10     |id  |employee_salary|employee_name |employee_age  |message                                |
11     |1   |320800         |Tiger Nixon   |61            |Successfully! Record has been fetched. |
```

- Feature file

An example of a Test Scenario where we are using the GET method to get the information from the API.

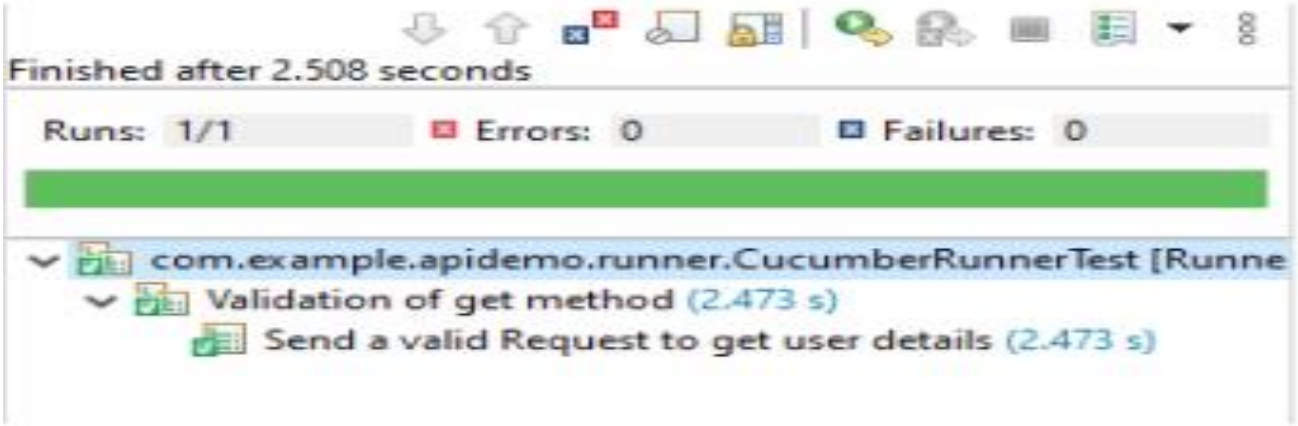
```java
package testRunners;


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


@RunWith(Cucumber.class)
@CucumberOptions(features="features",
glue= {"stepDefinitions"},
plugin = {"json:target/cucumber.json"
}
//tags = "@smoke or not @regression"
)
public class Runner {

}

```

- Runner file

```java
import io.restassured.http.ContentType;

import io.restassured.response.ValidatableResponse;

import static io.restassured.RestAssured.given;
import static org.hamcrest.Matchers.equalTo;

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

public class API_GETDefinitions {


    private ValidatableResponse validatableResponse;

    private String endpoint = "http://dummy.restapiexample.com/api/v1/employee/1";

    @Given("I send a request to the URL to get user details")
    public void sendRequest(){
        validatableResponse = given().contentType(ContentType.JSON)
                .when().get(endpoint).then();

        System.out.println("Response :"+validatableResponse.extract().asPrettyString());
    }


    @Then("the response will return status {int} and id {int} and salary {int} and name {string} and age {int} and message {string}")
    public void verifyStatus(int statusCode, int id, int emp_Salary, String emp_name, int emp_age, String message ){

        validatableResponse.assertThat().statusCode(statusCode);

        validatableResponse.assertThat().body("data.id",equalTo(id));

        validatableResponse.assertThat().body("data.employee_salary",equalTo(emp_Salary));

        validatableResponse.assertThat().body("data.employee_name",equalTo(emp_name));

        validatableResponse.assertThat().body("data.employee_age",equalTo(emp_age));

        validatableResponse.assertThat().body("message",equalTo(message));

    }
}
```

- StepDefinition FIle

```
Response :{
    "status": "success",
    "data": {
        "id": 1,
        "employee_name": "Tiger Nixon",
        "employee_salary": 320800,
        "employee_age": 61,
        "profile_image": ""
    },
    "message": "Successfully! Record has been fetched."
}
```

Output

You can execute the test script by right-clicking on TestRunner class -> Run As JUnit.

```java
import org.testng.annotations.Test;
public class Get_Request {
    @Test
    public void getSingleUserData()
    {
        given()
        .urlEncodingEnabled(false)
        .when()
            .get("http://localhost:8082/api/v1//users/30")
        .then()
            .statusCode(200)
            .body("name",equalTo("naveenkumar"))
            .body("age",equalTo(30))
            .body("gender",equalTo("male"))
            .body("city",equalTo("hyd"))
            .body("contry",equalTo("india"))
            .contentType(ContentType.JSON)
            .log().all();


    }
```

GET METHOD

- If User want to get the data from particular Server we have to use GET Method
- By using .body() tag user can validate expected data from the server
- By using .statuscode() user can verify the expected output.
- By using .contentType() user can change the output format ex. JSON, XML
- By using .log().all() user can check the output in the console

```java
package com.employeeapi.base;

import static io.restassured.RestAssured.given;

public class Post_Request {
public HashMap map=new HashMap();
    @BeforeClass
    public void postData()
    {
        map.put("name","dhivya");
        map.put("age",25);
        map.put("gender", "Female");
        map.put("city", "hyd");
        map.put("contry","india");
        map.put("mobileNumber",995191);
    }
    @Test
    public void postUser()
    {
        given()
        .urlEncodingEnabled(false)
        .contentType("application/json")
        .body(map)

        .when()
            .post("http://localhost:8082/api/v1/customers")

        .then()
            .statusCode(200)
            .body("name",equalTo("dhivya"))
```

```java
        map.put("contry","india");
        map.put("mobileNumber",995191);
    }
    @Test
    public void postUser()
    {
        given()
        .urlEncodingEnabled(false)
        .contentType("application/json")
        .body(map)

        .when()
            .post("http://localhost:8082/api/v1/customers")

        .then()
            .statusCode(200)
            .body("name",equalTo("dhivya"))
            .body("age",equalTo(25))
            .body("gender",equalTo("Female"))
            .body("city",equalTo("hyd"))
            .body("contry",equalTo("india"))
            .body("mobileNumber",equalTo(995191))
            //.contentType("application/json")
            .header("Content-Type", "application/json")
```

POST METHOD

- If User want to post the data to the server, we have to use POST method.
- User can check whether the proper data is added to the server by using this method.
- We can validate the data like name, age etc.

```java
package com.employeeapi.base;
import org.testng.annotations.BeforeClass;
public class Put_Request {
public HashMap map=new HashMap();


    @BeforeClass
    public void putData()
    {
        map.put("name","David");
        map.put("age",25);
        map.put("gender", "male");
        map.put("city", "hyd");
        map.put("contry","india");
        map.put("mobileNumber",995191);


    }


    @Test
    public void putUser()
    {
        given()
        .urlEncodingEnabled(false)
        .contentType("application/json")
        .body(map)

        .when()
            .put("http://localhost:8082/api/v1//users/25")
```

```java
    }


    @Test
    public void putUser()
    {
        given()
        .urlEncodingEnabled(false)
        .contentType("application/json")
        .body(map)

        .when()
            .put("http://localhost:8082/api/v1//users/25")

        .then()
            .statusCode(200)
            .body("name",equalTo("David"))
            .body("age",equalTo(25))
            .body("gender",equalTo("male"))
            .body("city",equalTo("hyd"))
            .body("contry",equalTo("india"))
            .body("mobileNumber",equalTo(995191))
            //.contentType("application/json")
            .header("Content-Type", "application/json")
            .log().body();
```

PUT METHOD

- If User want to update the data from particular server data, we have to use PUT Method.
- By using .body() tag user can validate expected data from the server
- By using .statuscode() user can verify the expected output.
- By using .contentType() user can change the output format ex. JSON, XML
- By using .log().all() user can check the output in the condole

```java
package com.employeeapi.base;

import org.testng.annotations.Test;

public class Delete_Request {
    @Test
    public void getSingleUserData()
    {
        given()
        .urlEncodingEnabled(false)
        .when()
            .delete("http://localhost:8082/api/v1//user/20")
        .then()
            .statusCode(200)
            .log().body();


    }
}
```

DELETE METHOD

- If User want to delete the data from particular server data, we have to use DELETE Method.
- By using .body() tag user can validate expected data from the server
- By using .statuscode() user can verify the expected output.
- By using .contentType() user can change the output format ex. JSON, XML
- By using .log().all() user can check the output in the condole


Thank
You