**TP ATELIER CUCUMBER :**

**Setting Up Development Environment**

**Step 1: Create a Maven Project**

Create a new Maven project from scratch and add the following dependencies and plugins to the pom.xml file.

<!--https://mvnrepository.com/artifact/io.cucumber/cucumber-java -->  
<dependency>  
 <groupId>io.cucumber</groupId>  
 <artifactId>cucumber-java</artifactId>  
 <version>7.3.4</version>  
</dependency>  
<!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-junit -->  
<dependency>  
 <groupId>io.cucumber</groupId>  
 <artifactId>cucumber-junit</artifactId>  
 <version>7.3.4</version>  
 <scope>test</scope>  
</dependency>

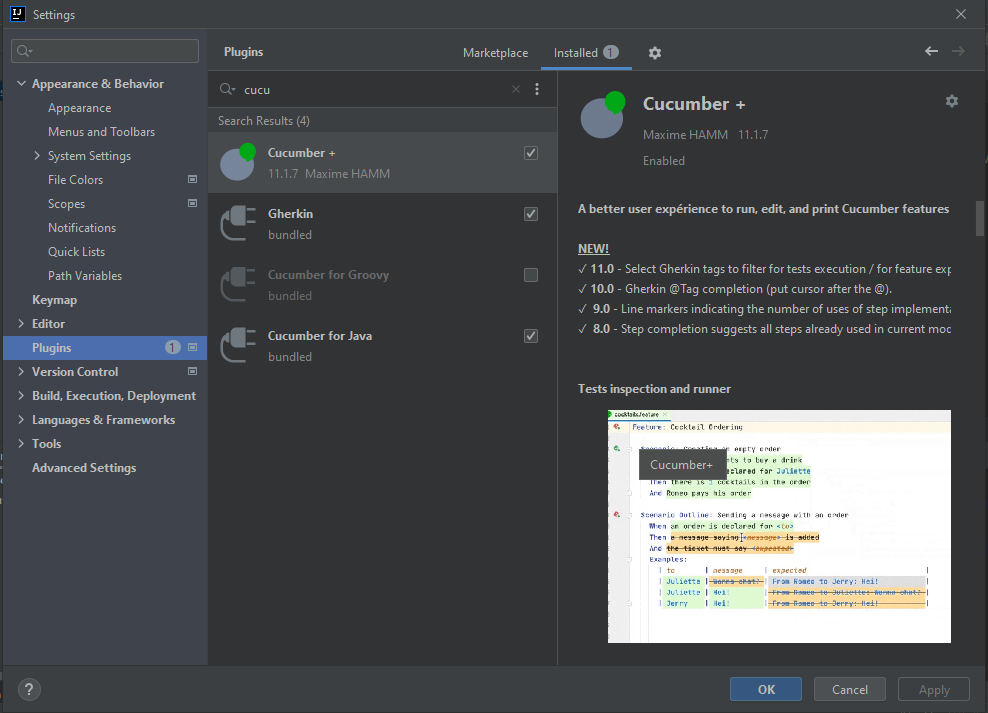
you can take it at :

<https://mvnrepository.com/artifact/io.cucumber/cucumber-java/7.3.4>

https://mvnrepository.com/artifact/io.cucumber/cucumber-junit/7.3.4

**Step 2: Add Cucumber for Java and Gherkin IntelliJ IDEA plugin**

Go to*File > Settings > Plugins* *> search ‘Cucumber for Java’ and ‘Gherkin’ > enable.*



## Step 3: Project Directory Structure

The directory structure of the sample project looks like the following.

+---.idea  
+---main  
| +---java  
| +---resources  
\---test  
| +---java  
| | \---org.example  
| | | |---SearchTest.java

| | |---RunTest.java  
| \---resources  
| \---features  
| |---SearchTest.feature  
|---pom.xml

# Getting Started with Development

## Step 1: Writing Features

Cucumber executes your**.feature files** in test/resources/features directory. These files contain executable specifications written in a domain-specific language (DSL) called **Gherkin**which is a business-readable, plain-text, English-like language with simple grammar. To specify business rules by real-world examples, Gherkin uses main keywords: **Feature, Scenario, Given, When, Then, And, But, Background, Scenario Outline, Examples**and some extra syntax **“””** (Doc strings),**|** (Data tables), **@**(Tags), **#** (Comments).

Feature: Search on Wikipedia  
 Scenario: Search direct on Wiki  
 Given Enter search term **'Cucumber'** When Do search  
 Then Single result is shown for **'Cucumber'**

A .feature file is supposed to describe a single feature of the system, or a particular aspect of a feature. It's just a way to provide a high-level description of a software feature, and to group related scenarios. A feature file gets the following format.

## Step 2: Writing Step Definitions

Cucumber doesn’t know how to execute your scenarios out-of-the-box. It needs Step Definitions to **translate plain text Gherkin steps into actions**that will interact with the system. **When Cucumber executes a Step in a Scenario, it will look for a matching Step Definition to execute.**

When Cucumber matches a Step against a pattern in a Step Definition, it passes the value of all the capture groups to the Step Definition’s arguments.

package org.example;  
  
import io.cucumber.java.en.Given;  
import io.cucumber.java.en.Then;  
import io.cucumber.java.en.When;  
  
public class SearchTest {  
 @Given("Enter search term {string}")  
 public void enterSearchTermCucumber(String name) {  
 System.out.println("test 1");  
 }  
  
 @When("Do search")  
 public void doSearch() {  
 System.out.println("test 2");  
 }  
  
 @Then("Single result is shown for {string}")  
 public void singleResultIsShownForCucumber(String name) {  
 System.out.println("test 3");  
 }  
}

Note that Cucumber does not differentiate between the five-step keywords **Given, When, Then, And** and **But**.

After writing features and step definitions, you are ready to implement the class TestRun.java . Thanks to Cucumber, the annotations and empty methods which map to the steps in feature files can be auto-generated.

## Step 3: Writing Test Runner

After writing the features and the step definitions , the test runner code is implemented. In the following code RunTest.java class, note the **@CucumberOptions**. One can define the location of features, glue files (step definitions), and formatter plugins inside this Cucumber options.

import io.cucumber.junit.Cucumber;  
import io.cucumber.junit.CucumberOptions;  
import org.junit.runner.RunWith;  
@RunWith(Cucumber.class)  
@CucumberOptions(  
 features = "src/test/resources/features/SearchTest.feature",  
 glue = {"org.example"}

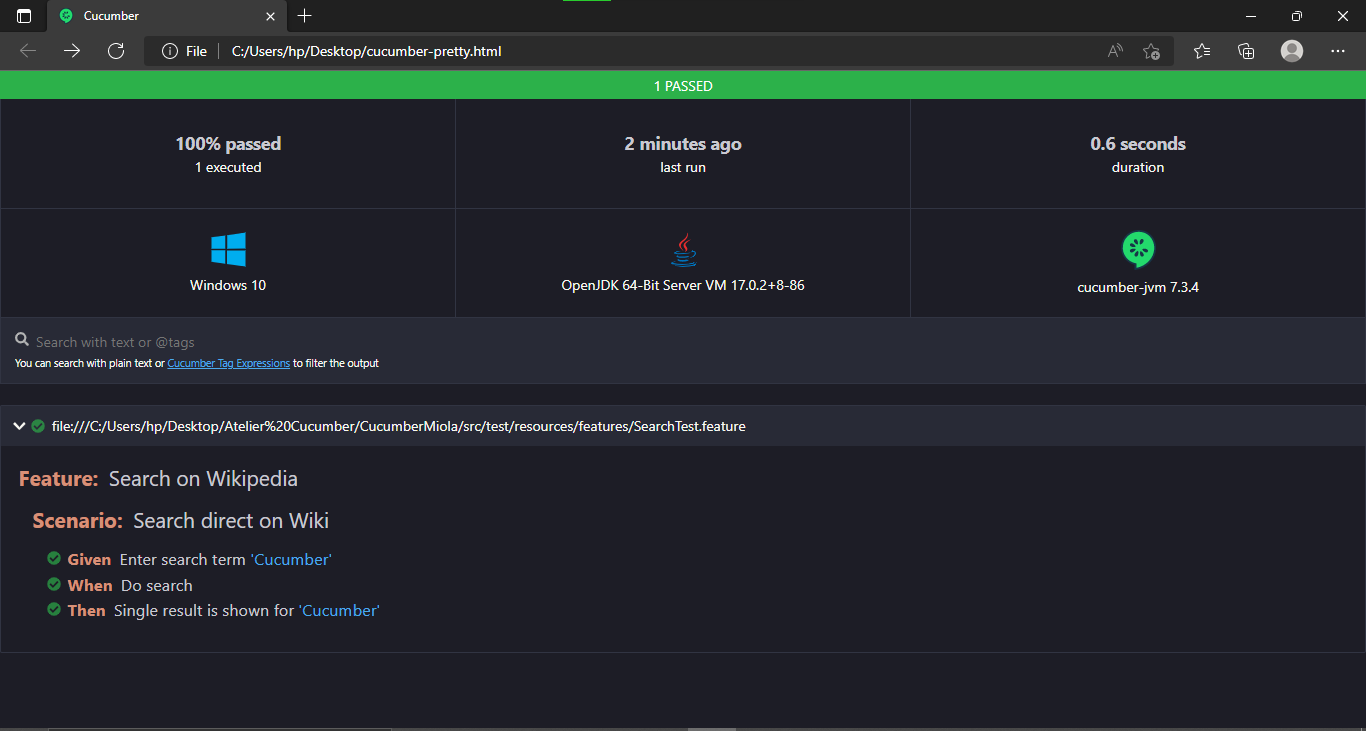
)  
public class RunTest {  
}

## Step 4: Generating Reports

This is one another cool option in Cucumber. If you carefully look at the pom.xml file, you can see maven-project-info-reports-plugin.

Add plugin = {  
 "pretty",  
 "html:target/cucumber-reports/cucumber-pretty",  
 "json:target/cucumber-reports/CucumberTestReport.json",  
 "rerun:target/cucumber-reports/rerun.txt" to the file RunTest.java.

import io.cucumber.junit.Cucumber;  
import io.cucumber.junit.CucumberOptions;  
import org.junit.runner.RunWith;  
@RunWith(Cucumber.class)  
@CucumberOptions(  
 features = "src/test/resources/features/SearchTest.feature",  
 glue = {"org.example"},  
  
 plugin = {  
 "pretty",  
 "html:target/cucumber-reports/cucumber-pretty",  
 "json:target/cucumber-reports/CucumberTestReport.json",  
 "rerun:target/cucumber-reports/rerun.txt"  
}  
 )  
public class RunTest {  
}



**DONE IS DONE**