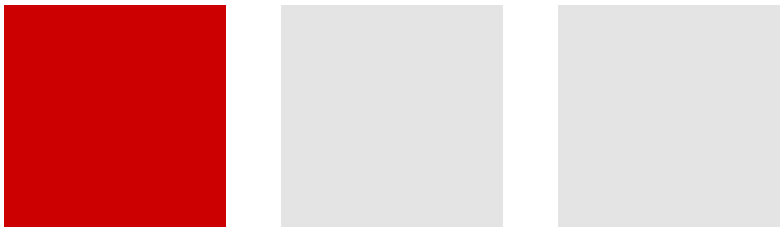


trivadis
makes IT easier.

# Cucumber – Your First Bite

Nick Jagger
Senior Consultant
16.01.2014

## Abstract

**Cucumber is a software testing tool which allows development teams to create software by writing "executable specifications" and create systems from the "outside-in". In doing so, we can ensure that what we deliver, along with the quality of what we deliver, meets the expectations of the business stakeholders.**

**In this short article, we're going to see how we can quickly setup a Java based Cucumber development environment using cucumber-jvm, Eclipse and Maven.**

## Introduction

Behavior-driven development (BDD) was developed by Dan North back in 2003 as a response to the problems and confusion he encountered when trying to following agile practices like test-driven development. He regularly faced questions around how much to test, what to call tests and where to start and finish the process.

He began to realize that a lot of these problems went away when you stopped thinking in terms of "testing" and instead began to think of "behaviors". By describing how different parts of a system should behave, he was able to respond to all his previous questions – what you call your test is a sentence describing the behavior you want to exercise; you are limited in how much you can test by the amount of behavior you can describe in a sentence; and so on.

As a response to this, he developed his own alternative to JUnit called JBehave, which placed a greater emphasis on behavior and removed all references to testing. Written in Java this was a behavior-driven testing framework which embodied the ideas and concepts he'd discovered.

The Ruby community quickly embraced the BDD approach and invented the RSpec framework, to which Dan North co-authored and helped integrate a story based framework called RBehave. This was later rewritten and replaced with what we know now today as Cucumber.

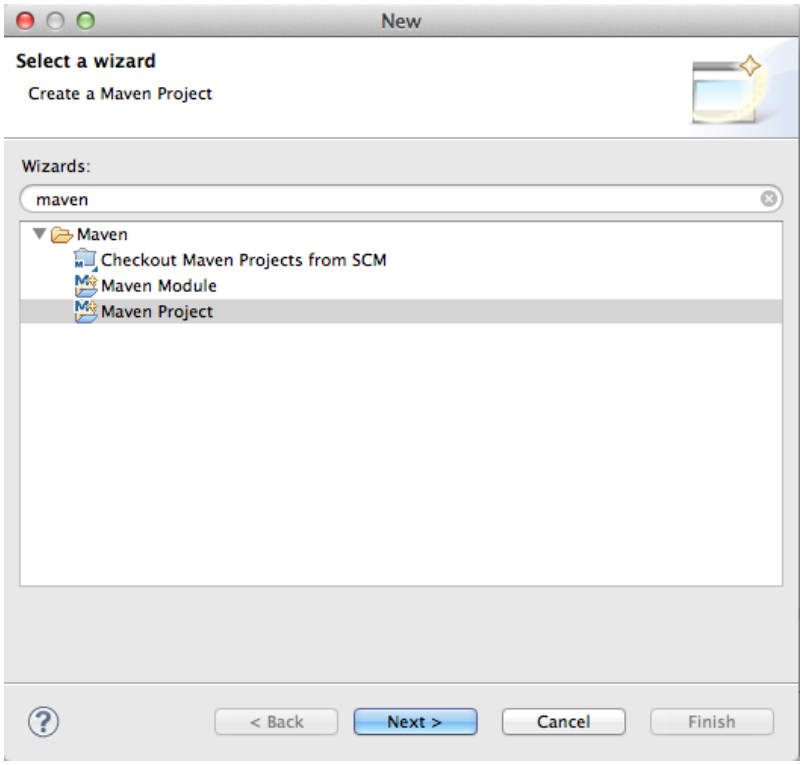
Cucumber introduces the notion of "features" which describe the behavior you wish to test. The feature is then broken down into a number of different "scenarios" which comprise the test you wish to execute which will subsequently validate the feature. Each scenario is further broken down into a number of "steps" which describe the execution path of each scenario. Typically, these follow a strict "given-when-then" format which aids consistency and provides a clear template for writing acceptance tests.

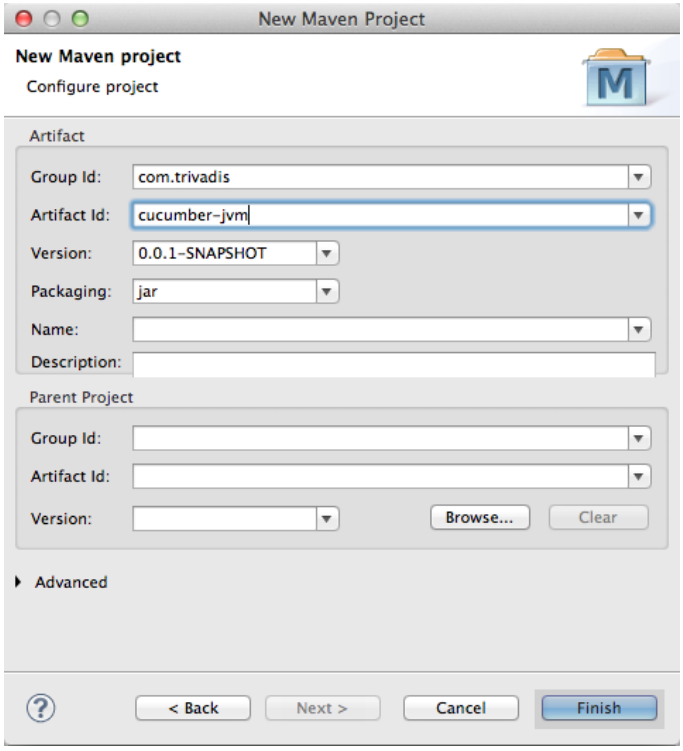
Originally written in Ruby, Cucumber has now been ported to many different languages and has a wide community-based support network. In the following article, we're going to look at how to setup and get started with Cucumber-JVM, a Java based variant of Cucumber.

**Tutorial**

# 1st Getting started

Open Eclipse and create a new Maven project:



## 2nd Add Maven dependencies

In your pom.xml, add the following dependencies:

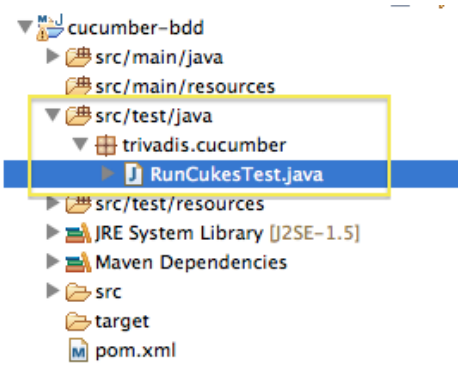
```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>
  <groupId>com.trivadis</groupId>
  <artifactId>cucumber-bdd</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <dependencies>

     <dependency>
        <groupId>info.cukes</groupId>
        <artifactId>cucumber-picocontainer</artifactId>
        <version>1.1.5</version>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>info.cukes</groupId>
        <artifactId>cucumber-junit</artifactId>
        <version>1.1.5</version>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.11</version>
        <scope>test</scope>
    </dependency>

  </dependencies>
</project>
```

## 3rd Create the JUnit Cucumber Test Runner

Next we will create the JUnit test runner which will allow you to run your Cucumber tests with JUnit. Under your *src/test/java* folder, create the package *trivadis.cucumber*. Within this package structure, create the class *RunCukesTest.java* and add the following code:

```
package trivadis.cucumber;

import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;

import org.junit.runner.RunWith;

// Associates Cucumber-JVM with the JUnit runner
@RunWith(Cucumber.class)
@CucumberOptions(format = {"html:target/cucumber-html-report", "json:target/cucumber-json-
```

```
report.json"})
public class RunCukesTest {
}
```

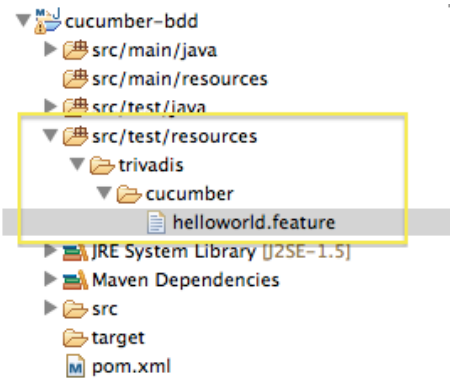



Note that the content of this class is empty – it serves only to associate JUnit with our Cucumber specifications.

## 4th Adding our first feature

Now we are ready to add our first feature. This is the executable specification which will form the basis of the first piece of functionality we want to create. Under the *src/test/resources* folder, create the folder structure *trivadis/cucumber*. Within the cucumber folder, create a file called *helloworld.feature* and add the following code:

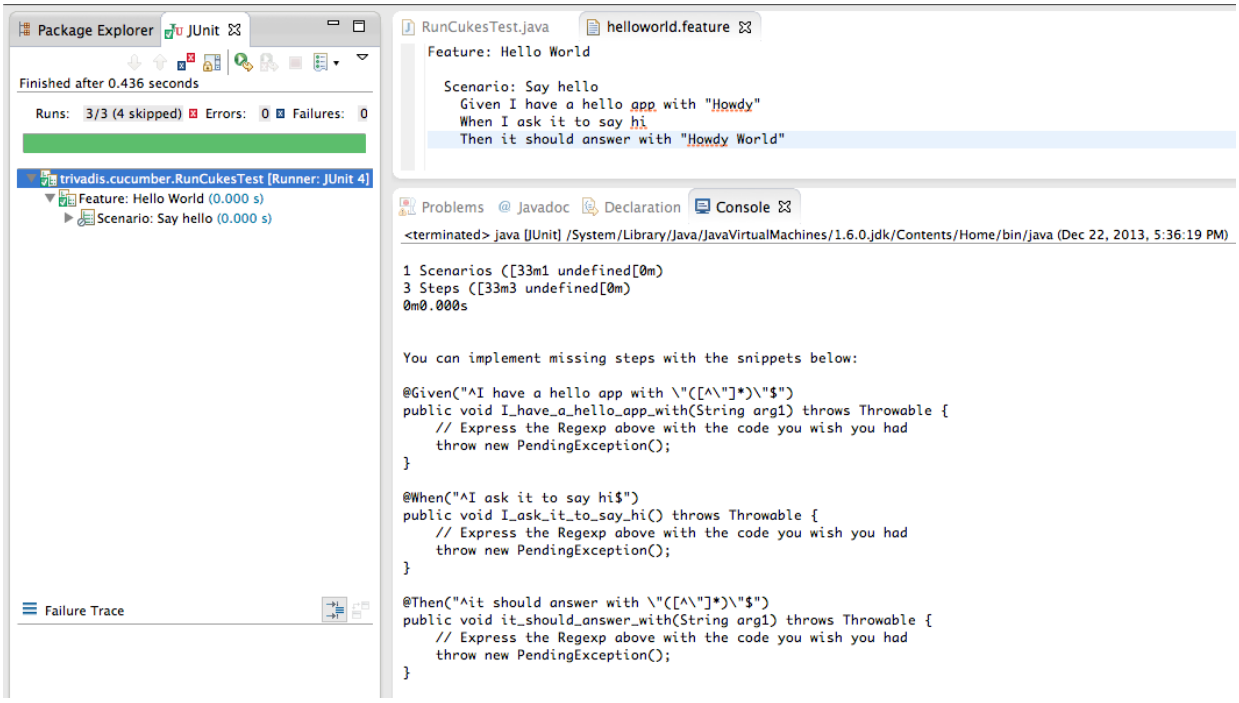
```
Feature: Hello World

  Scenario: Say hello
    Given I have a hello app with "Howdy"
    When I ask it to say hi
    Then it should answer with "Howdy World"
```



## 5th Run the feature

We are now ready to run our first feature. Note that we haven't yet implemented our steps described in the helloworld feature, so we should expect JUnit to complain.

Right click on the *src/test/java* folder and select *Run As > JUnit Test*. We should see the following output, telling us that we have not yet implemented our scenario:




On the left hand pane in the JUnit tab, we can see that the **Scenario: Say hello** test is greyed out, indicating that it has not been run.

In the console tab on the bottom right hand side of the screen, we can see that Cucumber tells us that we have missing step definitions, and even gives us some advice on how to implement them. We'll do this in the next step.

## 6th Create the step definitions

In the *src/testjava* folder under the package *trivadis.cucumber*, create a class called *HelloStepdefs.java* and add the output from the initial test run:

```
@Given("^I have a hello app with \"([^\"]*)\"$")
public void I_have_a_hello_app_with(String arg1) throws Throwable {
    // Express the Regexp above with the code you wish you had
    throw new PendingException();
}

@When("^I ask it to say hi$")
public void I_ask_it_to_say_hi() throws Throwable {
    // Express the Regexp above with the code you wish you had
    throw new PendingException();
}

@Then("^it should answer with \"([^\"]*)\"$")
public void it_should_answer_with(String arg1) throws Throwable {
    // Express the Regexp above with the code you wish you had
    throw new PendingException();
```

```
}
```

Run the test again, as per step 5, and you should receive the following:

```
1 Scenarios (_[33m1 pending_[0m)
3 Steps (_[36m2 skipped_[0m, _[33m1 pending_[0m)
0m0.289s

cucumber.api.PendingException: TODO: implement me
        at trivadis.cucumber.HelloStepdefs.I_have_a_hello_app_with(HelloStepdefs.java:14)
        at ?.Given I have a hello app with "Howdy"(trivadis/cucumber/helloworld.feature:4)
```

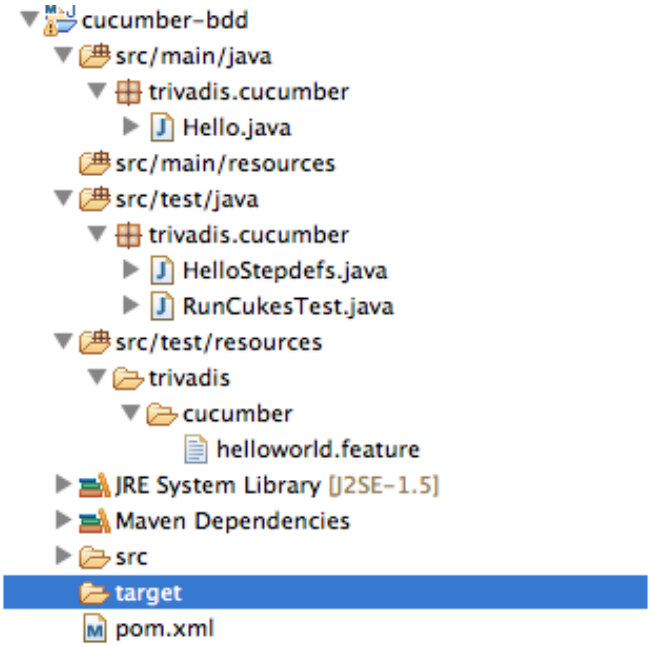
## 7th Implement the code and update the step

In the *src/main/java* folder, create the package *trivads.cucumber,* then add the class *Hello.java* with the code as follows:

```
package trivadis.cucumber;

public class Hello {
    private final String greeting;

    public Hello(String greeting) {
        this.greeting = greeting;
    }

    public String sayHi() {
        return greeting + " World";
    }
}
```

Then modify the class *HelloStepdefs.java* that we created in step 6 to be as follows:

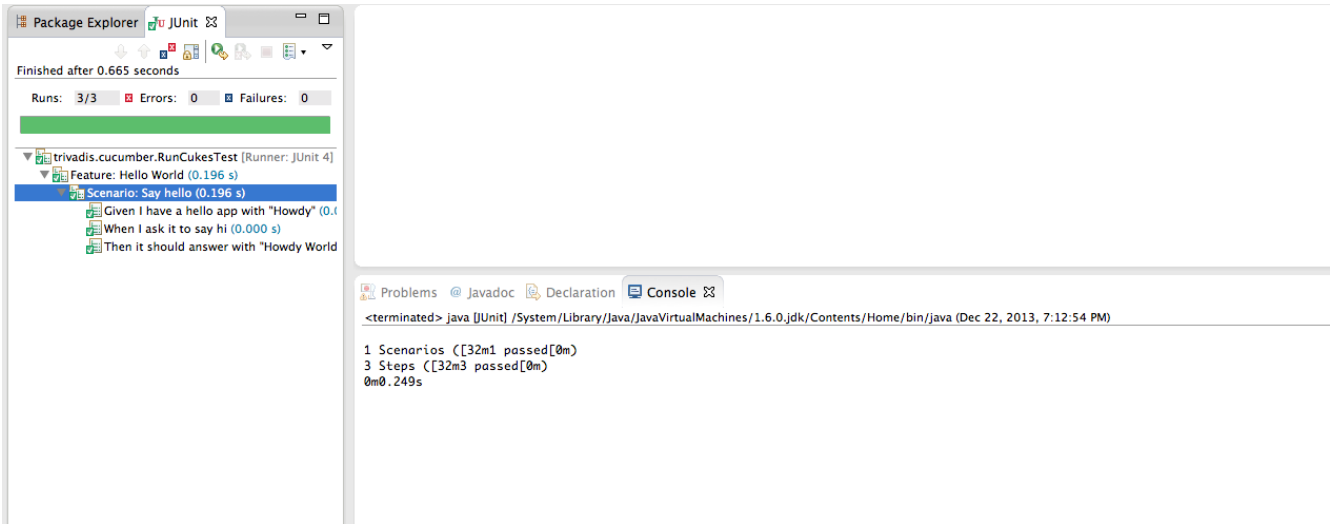
```
package trivadis.cucumber;

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

import static org.junit.Assert.assertEquals;

public class HelloStepdefs {
    private Hello hello;
    private String hi;

    @Given("^I have a hello app with \"([^\"]*)\"$")
    public void I_have_a_hello_app_with(String greeting) {
        hello = new Hello(greeting);
    }

    @When("^I ask it to say hi$")
    public void I_ask_it_to_say_hi() {
        hi = hello.sayHi();
    }

    @Then("^it should answer with \"([^\"]*)\"$")
    public void it_should_answer_with(String expectedHi) {
        assertEquals(expectedHi, hi);
    }
}
```




This should be your finished project structure.

## 8th Implement the code and update the step

With all this in place, right click the *src/test/java* folder and click *Run as > JUnit Test.* With any luck, you should have the following output:




On the left in the JUnit tab, we can see that we have successfully executed the feature *Hello World* containing the scenario *Say Hello* which consists of 3 steps. On the bottom right in the console, we can see that we have 1 passing scenario with 3 passing steps.

Congratulations, this is your first successful Cucumber acceptance test!

## 9th Further reading

We have only just scraped the surface of behavior-driven development and acceptance testing in this short article, and there is a large body of information available online and in the published technical literature. The next logical step would be to consult the Cucumber website http://cukes.info and the tutorials (https://github.com/cucumber/cucumber/wiki/tutorials-and-related-blog-posts) and examples (https://github.com/cucumber/cucumber/tree/master/examples/i18n) available on GitHub.

For published literature, the Cucumber Book (http://pragprog.com/book/hwcuc/the-cucumber-book) is an excellent technical manual which not only introduces you to the technology, but also to the theory and best practices associated with BDD.

Nick Jagger
Trivadis SA
Rue Marterey 5
1005 Lausanne
Internet: www.trivadis.com Mail: info@trivadis.com