Extracted from:

Cucumber Recipes

Automate Anything with BDD Tools and Techniques

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

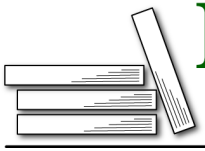
The Pragmatic Programmers

Cucumber Recipes

Automate Anything with BDD Tools and Techniques

Ian Dees,
Matt Wynne,
and Aslak Hellesøy

Edited by Jacquelyn Carter



Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *http://pragprog.com*.

Printed in the United States of America.
ISBN-13: 978-1-937785-01-7
Encoded using the finest acid-free high-entropy binary digits.
Book version: B1.0—August 15, 2012

Recipe 14

Use Cucumber With Java Via Cucumber-JVM

Problem

You need to test Java code using Cucumber syntax. You'd like to write your step definitions in pure Java, without bringing Ruby into the mix. And you need to connect it to your existing set of Java IDE and build tools.

Ingredients

- Cucumber-JVM,[4] a pure-Java (no Ruby) implementation of Cucumber
- IntelliJ IDEA Community Edition,[5] the open source edition of the beloved Java IDE
- JUnit[6] to serve as the test harness
- Maven[7] for dependency management

Solution

There are a few different ways to use Cucumber to test code written for the Java platform. The simplest is to use JRuby, an implementation of Ruby written in Java. But it's not the best fit for every project; it has a long startup time, sparse tool support, and a single choice of step definition language (Ruby). Fortunately, there are alternatives with different tradeoffs.

If your project uses a particular JVM langage, such as Clojure, Scala, or Java, you'd probably prefer to write your Cucumber step definitions in that language, rather than JRuby. It's also nice to be able to plug Cucumber into whatever IDE and build ecosystem you're using.

Cucumber-JVM fills these needs. It's written entirely in Java, so there's no need to bring in Ruby code if you're not already writing Ruby. It's provided as a set of jars, so that you can incorporate it into your workflow. It plugs

4. https://github.com/cucumber/cucumber-jvm
5. http://www.jetbrains.com/idea/download
6. http://www.junit.org
7. http://maven.apache.org

into the JUnit test harness, so you can run your Cucumber tests from your IDE.

For this recipe, we're going to write Cucumber features for a Java-powered soda machine. Remember those? In the 1990s, we were promised a bright future where all you'd have to do is wave a magic ring at a vending machine, and it would dispense your soda.[8] That bright future hasn't quite come to pass yet; we might as well build it ourselves.

We'll start with an empty project in IntelliJ IDEA, add Cucumber support using Maven, and then write and run a few features.

Setup

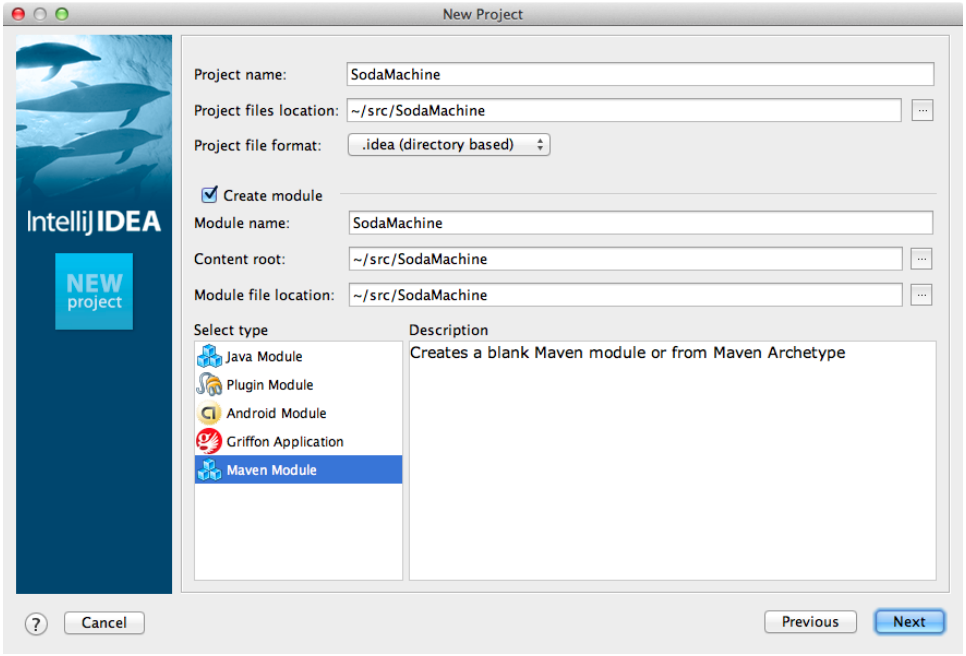


Figure 13—New Cucumber-JVM Project

Download IntelliJ IDEA and install it onto your system. Install Maven, either directly from their download page[9] or by installing a Java implementation that includes it.[10]

8. http://en.wikipedia.org/wiki/Jini
9. http://maven.apache.org/download.html
10. http://support.apple.com/kb/DL1421

Launch the IDE, click Create New Project, and choose Create project from scratch. Type "SodaMachine" for the project name, and select Maven Module as the project type, as in Figure 13, *New Cucumber-JVM Project*, on page 4. When the wizard prompts you for an archetype, leave it set to none.

IntelliJ IDEA will open your project's pom.xml automatically.[11] Fill in the Cucumber-JVM and JUnit dependencies just before the closing </project> tag:

jvm/pom.xml

```
<dependencies>
    <dependency>
        <groupId>info.cukes</groupId>
        <artifactId>cucumber-java</artifactId>
        <version>1.0.11</version>
    </dependency>
    <dependency>
        <groupId>info.cukes</groupId>
        <artifactId>cucumber-junit</artifactId>
        <version>1.0.11</version>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.10</version>
    </dependency>
</dependencies>
```

Select the View → Tool Windows → Maven Projects menu item. You should see your new Soda Machine 1.0 project. Click the Reimport All Maven Projects button—the one with two arrows, as in Figure 14, *Maven Dependencies*, on page 6.

Now, you need to tell JUnit that it will be running Cucumber tests. Expand the directory tree in the Project window on the left to show the src/test/java folder. Right-click that folder, select New → Java Class, and give RunCukesTest for the class name. Replace the file's contents with the following code:

jvm/src/test/java/RunCukesTest.java

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

@RunWith(Cucumber.class)
public class RunCukesTest {
}
```

Now IntelliJ IDEA is ready to run Cucumber. Time to write some features.

11. http://maven.apache.org/guides/introduction/introduction-to-the-pom.html

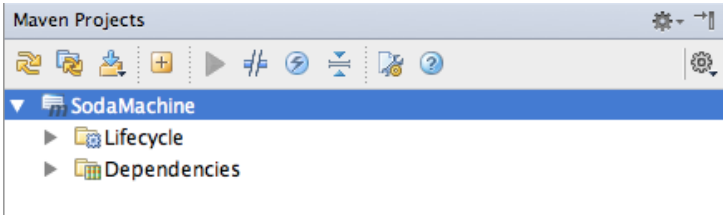


Figure 14—Maven Dependencies

Write Features

Create a directory in your project called src/test/resources, and create a plain text file in it called SodaMachine.feature with the following contents:

jvm/src/test/resources/SodaMachine.feature
```
Feature: Soda machine
  Scenario: Get soda
    Given I have $2 in my account
    When I wave my magic ring at the machine
    Then I should get a soda
```

Now you're ready to run your feature. Open the RunCukesTest file you were working on a minute ago. From the Run menu, choose Run.... In the small Run window that pops up, choose RunCukesTest. You should see the the following text in the output pane:

```
Test '.Scenario: Get soda.Given I have $2 in my account' ignored
Test '.Scenario: Get soda.When I wave my magic ring at the machine' ignored
Test '.Scenario: Get soda.Then I should get a soda' ignored


You can implement missing steps with the snippets below:
...
```

...followed by the usual set of suggested step definitions. Go ahead and copy those, so you can use them in your step definitions.

Implement step definitions

In the src/test/java directory, create a new StepDefinitions class with the following text copied and pasted from the output window:

jvm/src/test/java/StepDefinitions.java
```
Line 1 import cucumber.annotation.en.*;
  -
  - public class StepDefinitions {
  -     @Given("^I have \\$(\\d+) in my account$")
  5     public void I_have_$_in_my_account(int dollars) {
  -         // Express the Regexp above with the code you wish you had
```

```
    }

    @When("^I wave my magic ring at the machine$")
    public void I_wave_my_magic_ring_at_the_machine() {
        // Express the Regexp above with the code you wish you had
    }

    @Then("^I should get a soda$")
    public void I_should_get_a_soda() {
        // Express the Regexp above with the code you wish you had
    }
}
```

If you rerun your Cucumber tests, they should pass, and IntelliJ IDEA should show something like Figure 15, *Completed Test Run*, on page 7 in the Run window. The only thing left is to implement the soda machine. I'll leave that step as an exercise for the reader.

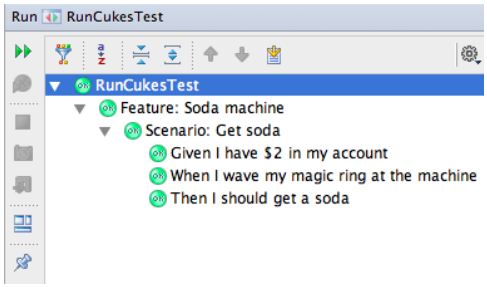


Figure 15—Completed Test Run

Further Exploration

In this recipe, we used IntelliJ IDEA to set up Cucumber-JVM. The approach is similar for other IDEs. To read one developer's experience in using Eclipse with Cucumber-JVM, see Zsolt Fabók's article, *Cucumber-JVM: Preparation*.[12]

12. http://www.zsoltfabok.com/blog/2011/12/cucumber-jvm-preparation/