FREE eBook

LEARNING cucumber

Free unaffiliated eBook created from
Stack Overflow contributors.

#cucumber

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: cucumber

It is an unofficial and free cucumber ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official cucumber.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with cucumber

Remarks

About Cucumber

Cucumber is a tool which runs executable specifications of software. Specifications, called "features", are written in structured natural language. Cucumber executes a feature by mapping each of its steps to a "step definition" written in the programming language supported by that implementation of Cucumber. Cucumber is implemented in many programming languages including Ruby (the original), Java and Javascript. It is also translated into many human languages.

Cucumber was written to support the agile methodology called Behavior-Driven Development (BDD). In BDD, one begins development outside-in by writing acceptance tests which describe the software's functionality from the user's point of view (rather than from a programmer's point of view such as in unit tests). Cucumber features serve as these acceptance tests.

In general, Cucumber features are human-readable documentation which is also an executable test suite, meaning that documentation and tests always agree. Cucumber is useful in communicating with non-programmer stakeholders about documentation and tests. It also allows programmers to write tests at a conceptual level without irrelevant programming-language concerns.

Cucumber is most often used to specify and test web applications, using a browser driver such as Selenium or PhantomJS. However, it can be used with any software that can be executed and whose state or results can be determined from the programming language that a Cucumber implementation supports.

Other documentation

Official documentation is at https://cucumber.io/docs. Documentation generated from the Cucumber features which describe Cucumber implementations is at

- JavaScript: https://relishapp.com/cucumber/cucumber-js/docs
- Ruby: https://relishapp.com/cucumber/cucumber/docs

https://relishapp.com/explore includes some other Cucumber-related tools and examples, although not, unfortunately, Cucumber-JVM.

This topic

This topic should only give a few examples which introduce the reader to Cucumber concepts. Other sections will give complete examples of installation, command-line and IDE usage, features, step definitions, etc.

Examples

A Cucumber feature

Cucumber uses Gherkin syntax to describe your software's behaviors in structured natural language.

As such Cucumber is **not** a test framework (a common misunderstanding), but a *system documentation framework*, not very different from others like Use Case Scenario. The common misunderstanding is due to the fact Cucumber documentation *can be automated in order to ensure it reflects the real system behavior*.

A Cucumber documentation suite is composed of `Features`, each describing a feature of your software, written in Gherkin and hosted in its own file. By organizing those files into a directory structure you can *group* and *organize* features:

- banking/
  - withdrawal.feature
  - atm.feature
  - personal-loan.feature
- trading/
  - portfolio.feature
  - intraday.feature
- mortgage/
  - evaluation.feature
  - accounting.feature

Each `Feature` is a plain text file composed by an optional, unstructured, purely informational introductory section and one or more `Scenarios`, each one representing a usage condition or use case.

Example:

```
Feature: Documentation
As a StackOverflow user or visitor
I want to access the documentation section

    Scenario: search documentation on Stack Overflow
        Given I am on StackOverflow
        And I go to the Documentation section
        When I search for "cucumber"
        And I follow the link to "cucumber"
        Then I should see documentation for "cucumber"
```

Each line beginning with *Given*, *When*, *And*, *But* or *Then* is called a `Step`. Any step can begin with any of those words regardless of order, but it is conventional to use them in the most natural way possible.

Features can also be organized via `Tags`, annotations the editor can put on a `Feature` or a `Scenario`

to further categorize it.

Executability of a Feature is achieved via *glue* code which can be written in many different languages (Java, Ruby, Scala, C/C++): each `Step` is matched against the glue code in order to identify `Step Definitions` (commonly abbreviated to *StepDef*) via regular expressions.

Every `Step` can have only one associated `Step Definition`.

When a `Feature` is executed each composing `Scenario` is executed, meaning each StepDef matching the `Step`s in every `Scenario` gets executed.

Pure Ruby Installation

To install Cucumber for use with Ruby simply use the command

```
gem install cucumber
```

Alternatively, if you are using bundler, you can add the following line to your Gemfile

```
gem 'cucumber'
```

And then run bundler

```
bundle install
```

[I think this belongs in its own topic, Installation. I created that topic and copied this example there. When that topic is approved I'll move this there and delete the copy.]

A Cucumber step definition in Ruby

In features/step_definitions/documentation.rb:

```
When /^I go to the "([^"]+)" documentation$/ do |section|
  path_part =
    case section
      when "Documentation"
        "documentation"
      else
        raise "Unknown documentation section: #{section}"
    end
  visit "/documentation/#{path_part}/topics"
end

Then /^I should see the "([^"]+) documentation"$/ do |section|
  expect(page).to have_css('h2.doctag_title a', text: section)
end
```

These steps exercise a web application. They are about as simple as they can be while still being practical.

Each step begins with a Gherkin keyword, which in a step definition file is a method which

registers a step with Cucumber. The step-defining method takes a regular expression, which matches a line in a scenario, and a block, which is executed when the scenario gets to a matching line. Capture groups in the regular expression are passed to the block as block parameters.

The `When` step has a simple, in-line example of going from a human-readable reference to a page ("Documentation") to a URL. Real Cucumber suites usually put this logic in a separate method. The `visit` method is provided by Capybara. Capybara is not required to use Cucumber, although it is very commonly used with it. `visit` tells the browser controlled by Capybara to visit the given URL.

The `Then` step shows how the content of a page can be tested. `expect`/`to` is provided by RSpec (again, not required by Cucumber but very commonly used with it). `have_css` is provided by Capybara. The expectation is that the given CSS selector matches an element on the page which contains the given text. Note that this expectation would fail if the browser request had failed.

For more examples, see the "Step definition" topic.

Read Getting started with cucumber online: https://riptutorial.com/cucumber/topic/4875/getting-started-with-cucumber

Chapter 2: Features

Introduction

You can use cucumber as a plugin in QTP and Selenium. The steps in the cucumber scenario are global variables. You can implement once and call many times. Hence reduces the code maintenance, and can reuse the same code when required.

Remarks

Cucumber features are written in the Gherkin language and stored in files with the suffix `.feature`. This topic gives examples of each feature of Gherkin.

Examples

A minimal Cucumber feature

In features/documentation.feature:

```
Feature: Documentation

  Scenario: User views documentation
    When I go to the "Cucumber" documentation
    Then I should see the "Cucumber" documentation
```

A minimal feature has a `Feature` line and a `Scenario` with one or more steps beginning with `When`, `Then` or another Gherkin keyword.

A sensible scenario would probably have more than one step.

Scenario Outline

Template as below

```
Scenario Outline: As a homemaker i want to buy and pay for the below product
  Given I purchase <a product>
    And I require a carry bag to take things to home
  When I pay bill using <payment method> to successfully checkout
  Then I should have a receipt

Examples:
| a product     | payment method |
|  Cake         | Visa           |
|  Coke         | Paypal         |
```

Syntax Usage

```
Feature: Some terse yet descriptive text of what is desired
     Textual description of the business value of this feature
     Business rules that govern the scope of the feature
     Any additional information that will make the feature easier to understand

Background:
    Given some precondition needed for all scenarios in this file
        And another precondition

Scenario: Some determinable business situation
  Textual description of the business value of this scenario
  Business rules that govern the scope of the scenario
  Any additional information that will make the scenario easier to understand
    Given some precondition
      And some other precondition
    When some action by the actor
      And some other action
      And yet another action
    Then some testable outcome is achieved
      And something else we can check happens too
      But something else we can check does not happen

Scenario Outline: Some determinable business situation
    Given I am <precondition>
        And some other precondition
    When some action by the actor
    Then I have <outcome> rights

Examples:
    | precondition | outcome  |
    | username1    | customer |
    | username2    | admin    |
```

The following keywords are interchangable, but depending on context, may be better to use:

- `Feature:` | `Ability:` | `Business Need:`
- `Scenario Outline:` | `Scenario Template:`
- `Examples:` | `Scenarios:`
- `Given` | `When` | `Then` | `And` | `But` | `*` |

Read Features online: https://riptutorial.com/cucumber/topic/6023/features

Chapter 3: Gherkin Syntax

Introduction

Gherkin is a business readable language for test automation and test documentation. It is understood by Cucumber and together exists as a Behavior Driven Development tool.

Syntax

- Feature: this keyword signifies that what follows is a basic description or name of the feature being tested or documented.
- Background: this keyword signifies steps that will be ran before every scenario in the feature.
- Scenario: this keyword represents the name or basic description of a particular scenario testing the feature.
- Scenario Outline: This keyword signifies that the scenario will run N times for every argument listed in examples explicitly passed by column name wrapped in angled brackets.
- Examples: this keyword notes the list of static arguments that will be passed into a scenario outline.
- Given: this keyword represents a given step, or precondition that is assumed before continuing. In the Arrange, Act, Assert paradigm, given represents "Arrange".
- When: this keyword represents a when step, or the behavior that is to be asserted against. In the Arrange, Act, Assert paradigm, given represents "Act".
- Then: this keyword represents a then step, or in other words, the step in which a behavior's result is validated. In the Arrange, Act, Assert paradigm, given represents "Assert".
- And: This keyword is used in conjunction with any of the keywords above. If you have two given statements, instead of explicitly calling Given twice, you can say, " Given A And B".

Examples

The Basics

This example will go over the basic structure of a Cucumber feature file in Gherkin. Feature files use several keywords in the basic syntax.

Lets look at the basic keywords:

- **Feature:** this keyword signifies that what follows is a basic description or name of the feature being tested or documented.
- **Scenario:** this keyword represents the name or basic description of a particular scenario testing the feature.
- **Given** this keyword represents a given step, or precondition that is assumed before continuing. In the Arrange, Act, Assert paradigm, given represents "Arrange".
- **When** this keyword represents a when step, or the behavior that is to be asserted against. In the Arrange, Act, Assert paradigm, given represents "Act".

- **Then** this keyword represents a then step, or in other words, the step in which a behavior's result is validated. In the Arrange, Act, Assert paradigm, given represents "Assert".
- **And** This keyword is used in conjunction with any of the keywords above. If you have two given statements, instead of explicitly calling Given twice, you can say, " Given A And B".
- **But** This keyword is used in conjunction **Given**, **When** and **Then** to signify that something should not happen. Then A But not B.

All keywords must be on a new line and must be the first word on a new line in order to be recognized by the Gherkin parser. The Feature and Scenario keywords must have a colon immediately after, as expressed in the example below. Given, When, Then, and And do not require a colon.

In addition to keywords, you can write descriptions and comments. Descriptions occur after the keyword but on the same line, where as comments occur on lines underneath the keywords. When writing Feature comments, it is customary to provided explicit rules outlining edges and conditions that lead to different outcomes of behaviors.

```
Feature: Product Login
    As a user, I would like to be able to use my credentials to successfully
    login.

    Rules:
    - The user must have a valid username
    - The user must have a valid password
    - The user must have an active subscription
    - User is locked out after 3 invalid attempts
    - User will get a generic error message following
      login attempt with invalid credentials

    Scenario: The user successfully logs in with valid credentials
        This scenario tests that a user is able to successfully login
        provided they enter a valid username, valid password, and
        currently have an active subscription on their account.

        Given the user is on the login page
        When the user signs in with valid credentials
        Then the user should be logged in
```

Parameterized Steps

When writing Gherkin, there may be times in which you want to parameterize your steps for reusability by the engineer who is implementing the test plans. Steps receive parameters through regular expression capturing groups. (**Engineering Note:** If you do not have matching parameters for each capturing group in your regular expression you can expect an "CucumberException: Arity mismatch" to be thrown) In the below example, we have decided to wrap arguments in double quotes, as well as accept integers as arguments.

```
 Feature: Product Login
    As a user, I would like to be able to use my credentials to successfully
    login.

    Rules:
```

```
    - The user must have a valid username
    - The user must have a valid password
    - The user must have an active subscription
    - User is locked out after 3 invalid attempts
    - User will get a generic error message following
      login attempt with invalid credentials

    Scenario: The user successfully logs in with valid credentials
        This scenario tests that a user is able to successfully login
        provided they enter a valid username, valid password, and
        currently have an active subscription on their account.

        Given the user is on the login page
        When the user signs in with "valid" credentials
        Then the user should be logged in

    Scenario: The user attempts to log in with invalid credentials
        This scenario tests that a user is not able to log in when
        they enter invalid credentials

        Given the user is on the login page
        When the user signs in with "invalid" credentials
        Then the user should be logged in

    Scenario: The user is locked out after too many failed attempts
        This scenario validates that the user is locked out
        of their account after failing three consecutive
        attempts to log in

        Given the user is on the login page
        When the user fails to log in 3 times
        Then the user should be locked out of their account
```

Feature Background

As you may have noticed in the example above, we are rewriting the same step multiple times:

```
Given the user is on the login page
```

This can be exceptionally tedious, especially if we have more than one given step that is reused. Gherkin provides a solution for this by giving us another keyword to work with: **Background:**.

The background keyword serves to run the steps declared underneath it before every scenario in the Feature. Be sure not to add background step unless you are positive it is necessary for every scenario. Like the other keywords, Background is followed by a description or name and can have comments listed below it. Much like Feature and Scenario, Background must be proceeded by a colon.

```
Feature: Product Login
    As a user, I would like to be able to use my credentials to successfully
    login.

    Rules:
    - The user must have a valid username
    - The user must have a valid password
    - The user must have an active subscription
```

```
    - User is locked out after 3 invalid attempts
    - User will get a generic error message following
      login attempt with invalid credentials

    Background: The user starts out on the login page
        Given the user is on the login page

    Scenario: The user successfully logs in with valid credentials
        This scenario tests that a user is able to successfully login
        provided they enter a valid username, valid password, and
        currently have an active subscription on their account.

        When the user signs in with "valid" credentials
        Then the user should be logged in

    Scenario: The user attempts to log in with invalid credentials
        This scenario tests that a user is not able to log in when
        they enter invalid credentials

        When the user signs in with "invalid" credentials
        Then the user should be logged in

    Scenario: The user is locked out after too many failed attempts
        This scenario validates that the user is locked out
        of their account after failing three consecutive
        attempts to log in

        When the user fails to log in 3 times
        Then the user should be locked out of their account
```

Scenario Outline

In some cases you may want to rerun the same scenario over and over, substituting out the arguments. In this case, Gherkin provides several new keywords to accommodate this situation, **Scenario Outline:** and **Example:**. The Scenario Outline keyword tells Cucumber that the scenario is going to run multiple times substituting out arguments from a list. The Examples keyword is called before the list is explicitly notated. Arguments for Scenario Outlines should be wrapped in angled brackets. In the example below, note that the argument names wrapped in the angled brackets correspond to the column names listed under Examples. Each column is separated by vertical bars, with column names on the first row.

```
Feature: Product Login
    As a user, I would like to be able to use my credentials to successfully
    login.

    Rules:
    - The user must have a valid username
    - The user must have a valid password
    - The user must have an active subscription
    - User is locked out after 3 invalid attempts
    - User will get a generic error message following
      login attempt with invalid credentials

    Background: The user starts out on the login page
        Given the user is on the login page

    Scenario Outline: The user successfully logs in with their account
```

```
        This scenario outlines tests in which various users attempt
        to sign in successfully

        When the user enters their <username>
        And the user enters their <password>
        Then the user should be successfully logged on

        Examples:
        | username | password |
        | frank    | 1234     |
        | jack     | 4321     |
```

Tags

For the purposes of documentation, you may want to filter test plans or scenarios by categories. Developers may want to run tests based on those same categories. Gherkin allows you to categorize Features as well as individual Scenarios via the user of Tags. In the example below, notice the above the Feature keyword is the Tag "@Automation". Gherkin recognizes this as a tag by the user of the "@" symbol. In this example, the engineer wants to make it clear that these tests are used for automation, where not every test is automate-able, some tests must be done by manual QA.

Notice as well that the tag @Production has been added to the scenario testing user lock out. In this example, this is because this scenario is only active in the production environment of the application. The developers don't want their sandbox accounts locked out during development. This tags allows them to enforce that this test will only be ran against the production environment.

Lastly, the Scenario Outline has the tag @Staging. For the purposes of this example, this is because the accounts being used are staging accounts and will not working in the other environments. Like the @Production tag, this ensures that these tests will only be ran in the Staging environment.

These are just a few examples of where, how, and why you might use tags. Ultimately these tags are going to have meaning to you and the developers and can be anything and used to categorize however you see fit.

```
@Automation
Feature: Product Login
    As a user, I would like to be able to use my credentials to successfully
    login.

    Rules:
    - The user must have a valid username
    - The user must have a valid password
    - The user must have an active subscription
    - User is locked out after 3 invalid attempts
    - User will get a generic error message following
      login attempt with invalid credentials

    Background: The user starts out on the login page
        Given the user is on the login page

    Scenario: The user successfully logs in with valid credentials
```

```
        This scenario tests that a user is able to successfully login
        provided they enter a valid username, valid password, and
        currently have an active subscription on their account.

        When the user signs in with "valid" credentials
        Then the user should be logged in

    Scenario: The user attempts to log in with invalid credentials
        This scenario tests that a user is not able to log in when
        they enter invalid credentials

        When the user signs in with "invalid" credentials
        Then the user should be logged in

    @Production
    Scenario: The user is locked out after too many failed attempts
        This scenario validates that the user is locked out
        of their account after failing three consecutive
        attempts to log in

        When the fails to log in 3 times
        Then the user should be locked out of their account

    @Staging
    Scenario Outline: The user successfully logs in with their account
         This scenario outlines tests in which various users attempt
         to sign in successfully

         When the user enters their <username>
         And the user enters their <password>
         Then the user should be successfully logged on

         Examples:
         | username | password |
         | frank    | 1234     |
         | jack     | 4321     |
```

Gherkin Tips

- Each scenario tests one behaviour
- Scenarios are written in a declarative way
- Avoid incidental details inside the scenario
- Omit the obvious
- Avoid conjunctive steps
- Keep your scenarios short
- Don’t have to many scenarios in the same feature
- Use descriptive scenario names
- Have only one When step
- Use the “should” in Then steps

Read Gherkin Syntax online: https://riptutorial.com/cucumber/topic/9296/gherkin-syntax

Chapter 4: Install cucumber plugin in Intellij

Introduction

The Cucumber plugins for IntelliJ IDEA offer convenient IDE features for working with Gherkin feature files in an IntelliJ project using the Cucumber framework. Plugins are available for Java, Scala, or Groovy languages.

Remarks

The Cucumber for Java IntelliJ plugin offers IDE features for conveniently developing with Cucumber, including

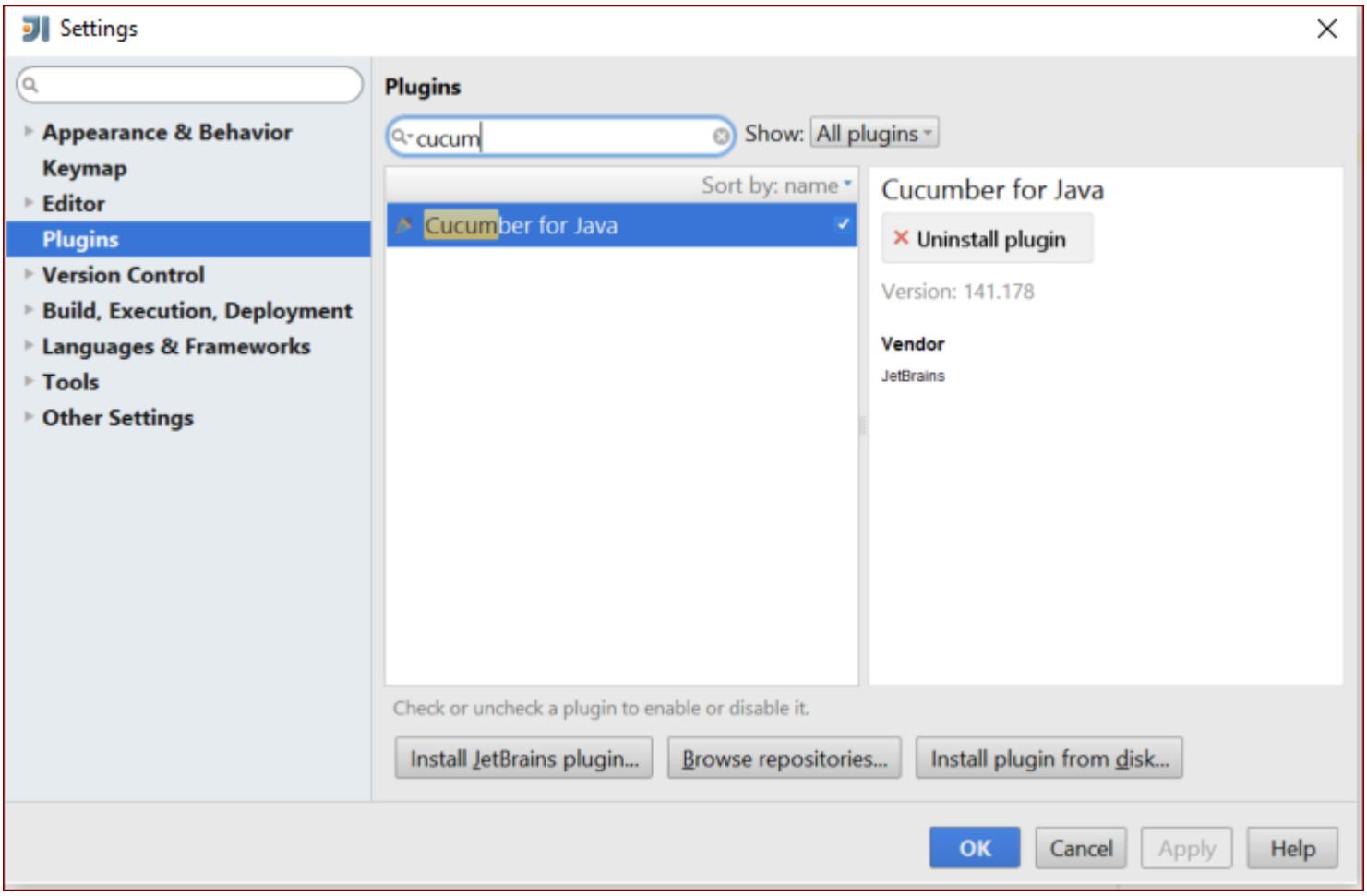
- Gherkin step glue generation for unimplemented steps
- Gherkin step code completion
- Step-to-glue method code jumping
- Gherkin syntax highlighting in ".feature" files matching step regex

and other convenient features.

Examples

Install Cucumber plugin

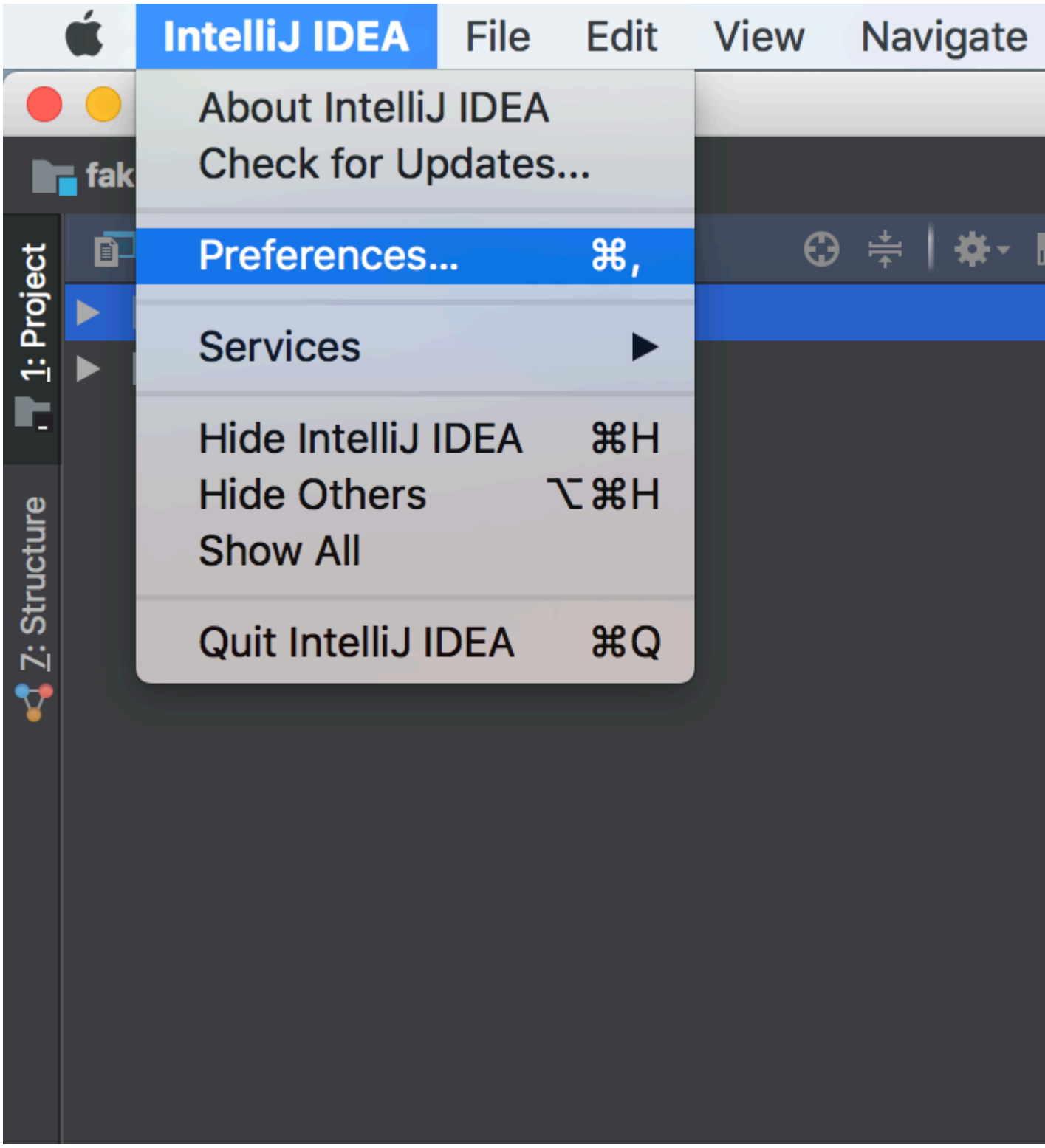
Go to File --> Settings --> click plugins in left hand pane --> Search for cucumber --> Install plugin



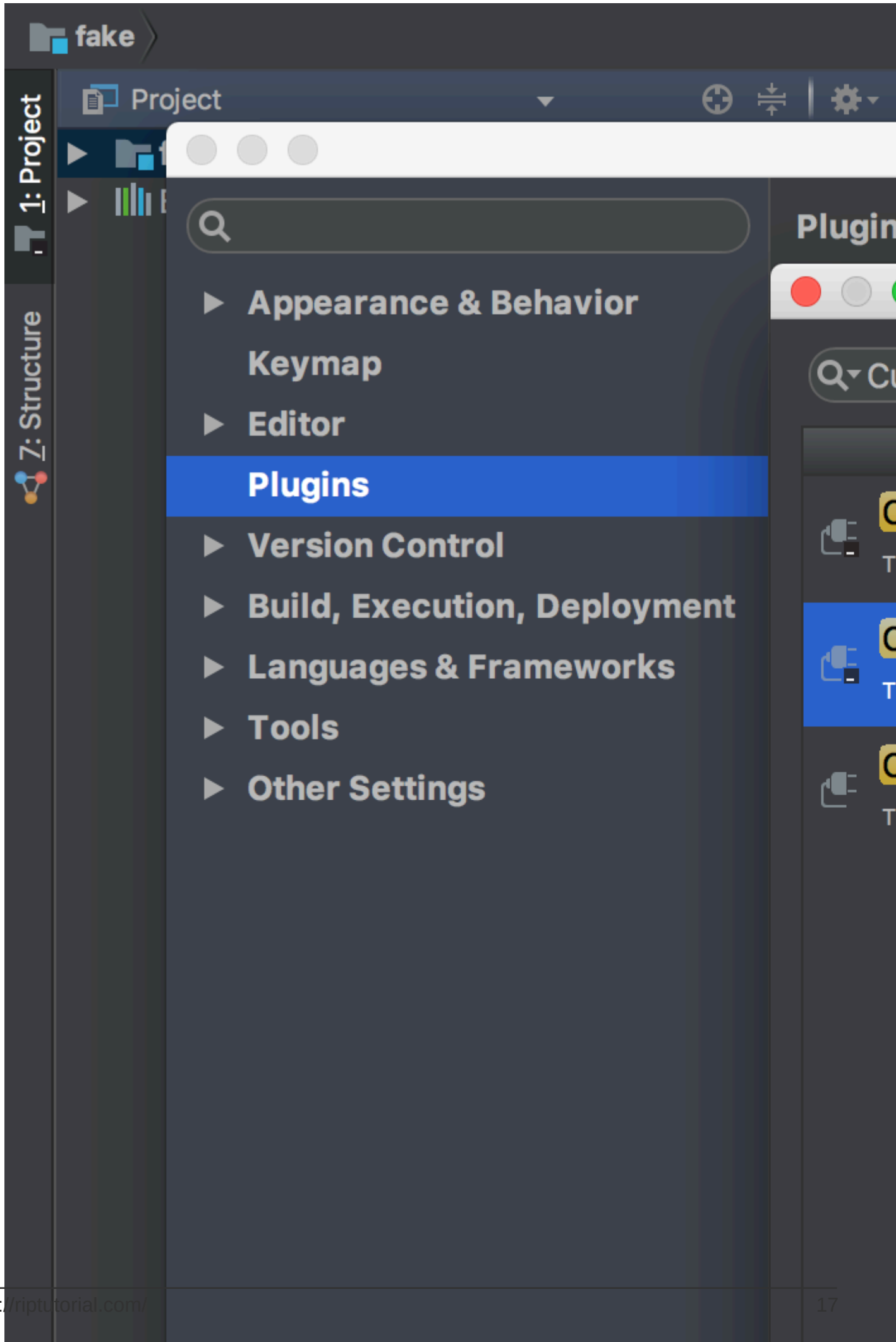
Install IntelliJ Cucumber for Java Plugin (Mac)

To install the Cucumber for Java plugin for IntelliJ on a Mac,

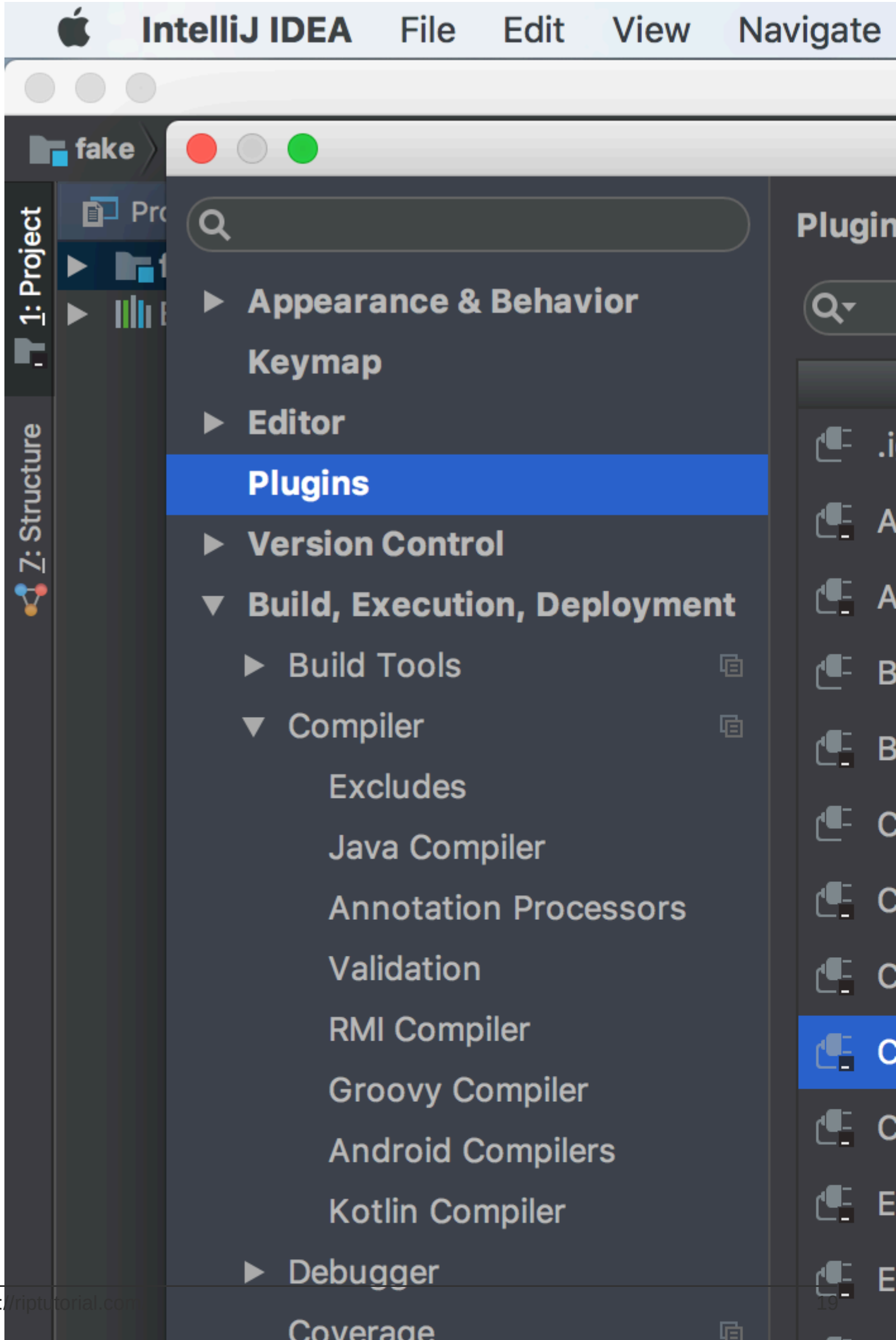
1. Start IntelliJ IDEA.
2. Click on the "IntelliJ IDEA" tab in the top bar.



3. Click on "Preferences".
4. In Preferences/Settings, click "Plugins" in the left-hand pane.
5. Click the "Browse Repositories" button, which brings up a new window.
6. Search for "Cucumber" in the search bar.


fake
Project
1: Project
Z: Structure
Appearance & Behavior
Keymap
Editor
Plugins
Version Control
Build, Execution, Deployment
Languages & Frameworks
Tools
Other Settings

7. Install the "Cucumber for Java" plugin.
8. Restart the IDE for the plugin to take effect. The Cucumber for Java is now installed.


IntelliJ IDEA
File
Edit
View
Navigate
fake
1: Project
7: Structure
Appearance & Behavior
Keymap
Editor
Plugins
Version Control
Build, Execution, Deployment
Build Tools
Compiler
Excludes
Java Compiler
Annotation Processors
Validation
RMI Compiler
Groovy Compiler
Android Compilers
Kotlin Compiler
Debugger
Coverage

https://riptutorial.com/cucumber/topic/8356/install-cucumber-plugin-in-intellij

Chapter 5: pom.xml for Maven_ cucumber project.

Introduction

The below project object model is the template pom.xml. If you want to create a maven with cucumber project, you can use the below example as template

Examples

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

4.0.0

```
<groupId>Project name</groupId>
<artifactId>MulitClients</artifactId>
<version>1.0-SNAPSHOT</version>


<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.11</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>info.cukes</groupId>
        <artifactId>cucumber-core</artifactId>
        <version>1.2.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>info.cukes</groupId>
        <artifactId>cucumber-testng</artifactId>
        <version>1.2.0</version>
    </dependency>
    <dependency>
        <groupId>info.cukes</groupId>
        <artifactId>cucumber-junit</artifactId>
        <version>1.2.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>info.cukes</groupId>
        <artifactId>cucumber-java</artifactId>
        <version>1.2.0</version>
        <scope>test</scope>
    </dependency>
```

```
    <dependency>
        <groupId>info.cukes</groupId>
        <artifactId>gherkin</artifactId>
        <version>2.12.2</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-java</artifactId>
        <version>2.53.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-firefox-driver</artifactId>
        <version>2.53.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-htmlunit-driver</artifactId>
        <version>2.53.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.yaml</groupId>
        <artifactId>snakeyaml</artifactId>
        <version>1.13</version>
    </dependency>
    <dependency>
        <groupId>com.esotericsoftware.yamlbeans</groupId>
        <artifactId>yamlbeans</artifactId>
        <version>1.06</version>
    </dependency>


</dependencies>
```

Read pom.xml for Maven_ cucumber project. online:
https://riptutorial.com/cucumber/topic/8331/pom-xml-for-maven--cucumber-project-

Chapter 6: Step definitions

Remarks

Step definitions are in the programming language supported by a given implementation of Cucumber. This topic gives examples of step definitions in each supported programming language and examples of using Cucumber API calls in step definitions.

Examples

Some simple Ruby step definitions

In features/step_definitions/documentation.rb:

```
When /^I go to the "([^"]+)" documentation$/ do |section|
  path_part =
    case section
      when "Documentation"
        "documentation"
      else
        raise "Unknown documentation section: #{section}"
    end
  visit "/documentation/#{path_part}/topics"
end

Then /^I should see the "([^"]+) documentation"$/ do |section|
  expect(page).to have_css('h2.doctag_title a', text: section)
end
```

These steps exercise a web application. They are about as simple as they can be while still being practical.

Each step begins with a Gherkin keyword, which in a step definition file is a method which registers a step with Cucumber. The step-defining method takes a regular expression, which matches a line in a scenario, and a block, which is executed when the scenario gets to a matching line. Capture groups in the regular expression are passed to the block as block parameters.

The `When` step has a simple, in-line example of going from a human-readable reference to a page ("Documentation") to a URL. Real Cucumber suites usually put this logic in a separate method. The `visit` method is provided by Capybara. Capybara is not required to use Cucumber, although it is very commonly used with it. `visit` tells the browser controlled by Capybara to visit the given URL.

The `Then` step shows how the content of a page can be tested. `expect`/`to` is provided by RSpec (again, not required by Cucumber but very commonly used with it). `have_css` is provided by Capybara. The expectation is that the given CSS selector matches an element on the page which contains the given text. Note that this expectation would fail if the browser request had failed.

Read Step definitions online: https://riptutorial.com/cucumber/topic/5681/step-definitions

Credits

S. No	Chapters	Contributors
1	Getting started with cucumber	Community, Dave Schweisguth, Mo H., Roberto Lo Giacco, SirLenz0rlot, user3554664
2	Features	Dave Schweisguth, Kyle Fairns, Priya
3	Gherkin Syntax	jordiPons, tramstheman, user3554664
4	Install cucumber plugin in Intellij	George Pantazes, Priya
5	pom.xml for Maven_ cucumber project.	user
6	Step definitions	Dave Schweisguth