5 Lab: BDD with Cucumber framework

v2025-03-13

5.0 Introduction

Learning objectives

- Write descriptive test scenarios with Gerking DSL.
- Automate test scenarios using Cucumber framework for Java and JUnit.

Key Points

- The Cucumber framework enables the concept of "executable specifications": with Cucumber we use **examples** to specify what we want the software to do. The **scenarios are written before the production code**.
- Cucumber executes features (test scenarios) written with the Gherkin language (readable by non-programmers too).
- The steps included in the feature description (scenario) must be mapped into Java test code by annotating test methods with matching "expressions". Expressions can be (traditional) regular expressions or the (new) Cucumber expressions.
- BDD plays well with user stories: the user story can be used as a unit of specification, work assignment, acceptance, and delivery. In this sense, we can have better traceability from business requirements to code.

Explore

- You may configure IntelliJ to offer extended support to run Cucumber.
- Cucumber school: guided exercises, video lessons... Especially: Cumber for Java developers.

5.1 Getting started (calculator)

Consider that you are implementing a Reverse Polish Notation (RPN) calculator module and you want to supply some tests[1].

a) Start a (regular) Java project with the appropriate POM settings to run Cucumber (cucumber-java ; cucumber-junit-platform-engine; junit-platform-suite ; junit-jupiter). [Alternative POM with dependencies management, based on official "skeleton" for JVM]

[1] This example is discussed in "Cucumber in a Nutshell" section, in B. Garcia's book; solution code available, if needed.

b) Create a .feature file (a regular text file with .feature extension) to describe the intended use of the calculator (related example).

Note that the *.feature* file must go into the tests resources folder, e.g.:

./src/**test/resources**/*mypackage*/calculator_ops.*feature*

c) Be sure to implement the required test steps.

You will need two test files: one (generic) to activate Cucumber engine and another with the effective test steps implementation (related example).

d) Run the tests. Confirm that you get the

Note: in the example from Boni García book, step matching uses regular expressions. The best practice, however, is to use "cucumber expressions". **Be sure to "upgrade"** the sample code. E.g.:

(**old**) regular expressions style:	(**better**) Cucumber expressions style:
@When("^I add (\\d+) and (\\d+)$")	@When("I add {int} and {int}")

e) Add a few more test scenarios in your feature (e.g.: multiply, invalid operation...)

Now, if you try to build the project, the tests will not pass because they still are incomplete.

Note the "clues" in the output, giving suggestions to implement the missing steps (i.e., test methods).

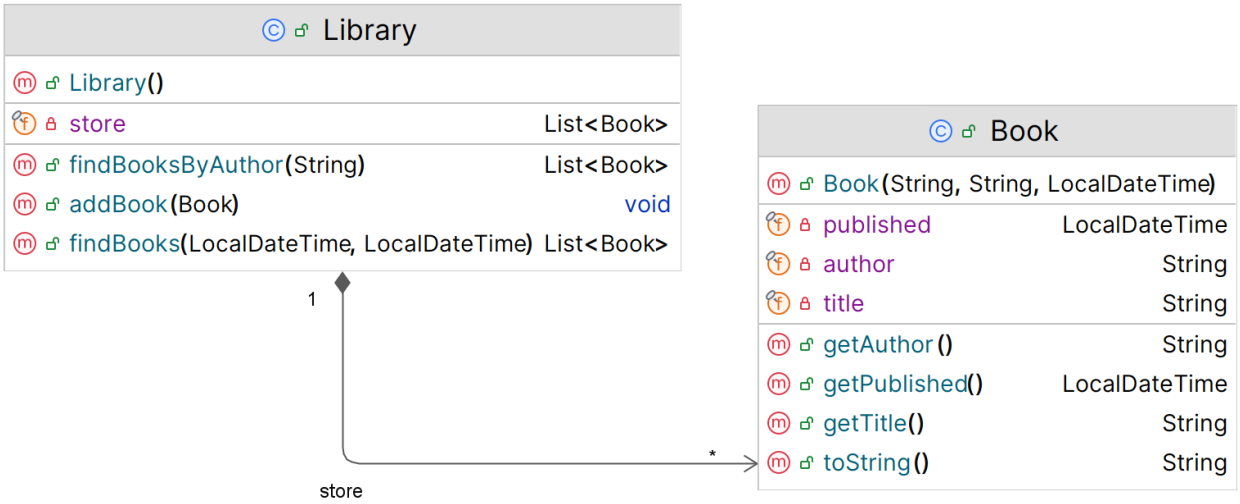
Implement the required test steps to have your feature specification satisfied.

5.2 Passing data to tests

Consider a Library class that manages a collection of Books.

Write a feature to verify representative scenarios related to the **book search** user story. Consider a few search options (by author, by category, etc).

Consider the scenariospresented this example, and write your own tests. Feel free to add different scenarios/features.



Notes:

- Write the features before the test steps. Steps can be partially generated from features.
- Prefer the "cucumber expressions" (instead of regular expressions) to write the steps definitions.
- To handle dates matching, consider using a ParameterType configuration. This defines a new custom parameter type to use in the matching expressions. [adapt from → partial snippet].

  @When("the customer searches for books published between {iso8601Date} and {iso8601Date}")

  The dates in the feature description need also to match the date mask defined in the ParameterType (**aaaa-mm-dd**).
- To handle data tables in the feature description (e.g.: define a "database" of books), consider using a DataTable mapping and access you data as a *list of maps* (use headings in the data).

5.3 Web automation (online library)

Cucumber is often used with Selenium WebDriver to write expressive web automation tests.

a) Consider the example from the previous lab, related to the "fake" online library (section 4.3).

   Develop some scenarios to test key user stories (search). You may consider using suggestions from an AI tool to help with writing the "feature" file.

b) Implement the test automation using Cucumber, Jupiter and Selenium [related example].