COMP5903 Project Outline

Enhanced “Cucumberized” Junit

Charles Chen 101035684

Introduction

Behavior Driven Development (BDD) is a software development process that enhances collaboration between business people and technical people to build the shared understanding of the problem to be solved. Such shared understanding contains concrete, real-world examples called User Stories that demonstrates how the system should behave [1]. Hence, to allow software development driven by User Stories, known as the BDD approach, the Cucumber testing tool was developed so that business people can define their user stories in a clear syntax that are then implemented (as so-called step-definitions) by software developers as tests for the production code.

Despite the benefits of BDD and Cucumber, developers in the world of Java are still not quite sold on BDD, focusing instead on unit testing with JUnit. Not only is the setup of Cucumber intricate (especially because the integration setup of Cucumber with JUnit 4 and JUnit 5 are completely different), but also the learning curve is steeper compared to JUnit. Hence, Professor Jean-Pierre Corriveau came up with an idea called “Cucumberized” JUnit that aims to support BDD using JUnit while simplifying the setup procedure and learning curve. This tool is to bring the Cucumber’s core features to JUnit so that developers who are familiar with JUnit can quickly pick up Cucumber-based BDD and start writing test cases.

This idea was first implemented by Alexei Krumshyn as his Honor Project under the supervision of Professor Corriveau in the Winter term of 2022. The Honor Project was a great success. But due to the short development time of a single school term, Alexei’s “Cucumberized” JUnit contains constraints and lacks some commonly used features from Cucumber.

Main Objectives

Therefore, the main objective of this project is to enhance Alexei’s “Cucumberized” JUnit to a production-ready level, while maintaining the advantage of an easy setup procedure and smoother learning curve. More specifically, the planned improvements for this project are as follows:

1. Tooling and Development
    a. Make the “Cucumberized” JUnit as a Maven package
2. User Experiences
    a. Use dedicated exceptions with error code and human readable error message to help developers determining the issues
3. Refactor and improvement

    a. JScenario code should be separated into several classes for better maintainability and testability. New separated classes include but are not limited to Jfeature, JScenario, JScenarioBuilder, and etc.
    b. Nested collection such as List of Map of List is not a best practice, instead data classes should be created for storing information.
    c. If a step has a number in the JScenaio description, the project always tries to find the method that has an int parameter. Relax this requirement and allow empty parameter step definition method matches this kind of step.
        i. For example, "I have 3 apples" will currently always be mapped to a step definition method with an int parameter, such as "void my_definition(int numberOfApples)”
    d. The name of the step definition method must be written in a fixed pattern that matches what is written in the JScenario description. Relax this requirement.
        i. For example, to match "I have an apple", the method name must be "i_have_an_apple()"
4. New features
    a. Support Tags [2], as well as allow running subset of scenarios
        i. (Optional, time-permitting) Supports tags on example tables which allows a scenario outline to have multiple example table candidate and enable one of them through tags
    b. Support comments in #, and """ doc string
    c. Support keyword “And” and “But”
    d. Support parameter types of double, BigDecimal, BigInteger
    e. Allow users providing their own instance of the step definition class, instead of relying on JScenario to create an instance of step definition through Java Reflection. This allows integration with one’s favourite dependency injection framework like Spring.
    f. Allow users providing multiple step definition class/instances for one scenario, like how Cucumber does it
    g. (Optional, time-permitting) Support Background [3], a “Scenario” that runs before all Scenarios
    h. (Optional, time-permitting) Support selected Hooks [4] such as BeforeAll, AfterAll, etc.
    i. (Optional, time-permitting) Support i18n [5]
    j. (Optional, time-permitting) Support custom data type conversion [6]

Please note it is not my goal to re-implement Cucumber, even though many improvements were inspired by the original Cucumber framework. Cucumber itself is a complicated framework with a deeper learning curve, re-implementation would break the first objective. Not only that, reinventing wheels waste time.

Requirement

The requirement is simple, to develop and run the project, only Java 11 and Maven 3.6.x^ are required.

Implementation

Expected End Product

The project will be developed as a Java project which serves as a maven package, and the way developers use this tool will remain similar to Alexei's "Cucumberized" JUnit. Hence, there won't be many API changes. However, the internal structure will be changed significantly to achieve the improvement list above. First, the procedure of Alexei's "Cucumberized" JUnit is decomposed into four stages:

```
Given inputs: feature file, classes
```

1. `JfeatureDetail <- parseJfeatureFile(feature file)`
2. `JStepDefMethodDetail <- parseStepDefinition(classes)`
3. `executableJFeature <- createExecutable(JfeatureDetail, JStepDefMethodDetail)`
4. `User calls executableJFeature.execute() to run the cucumber test`

These four stages can be illustrated as a flow chart below:

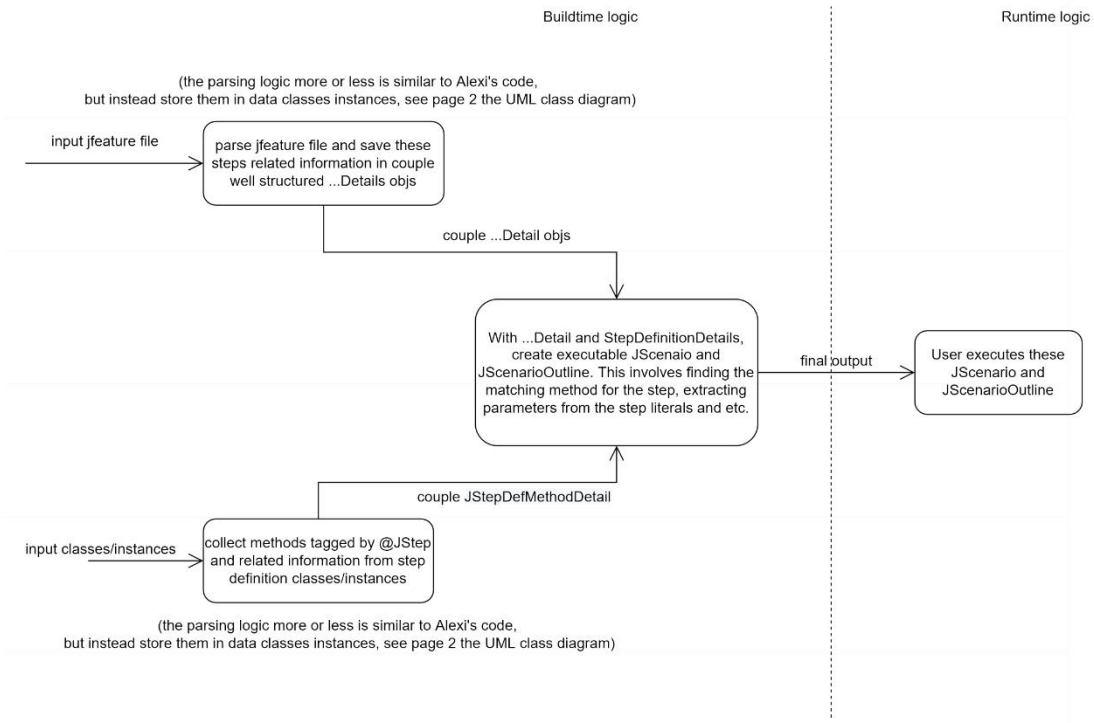

Figure 1 - Flow Chart of the "Cucumberized" JUnit problem

Based on this decomposition, a highly extensible structure with all the improvements in mind is designed as the following UML Class Diagram:

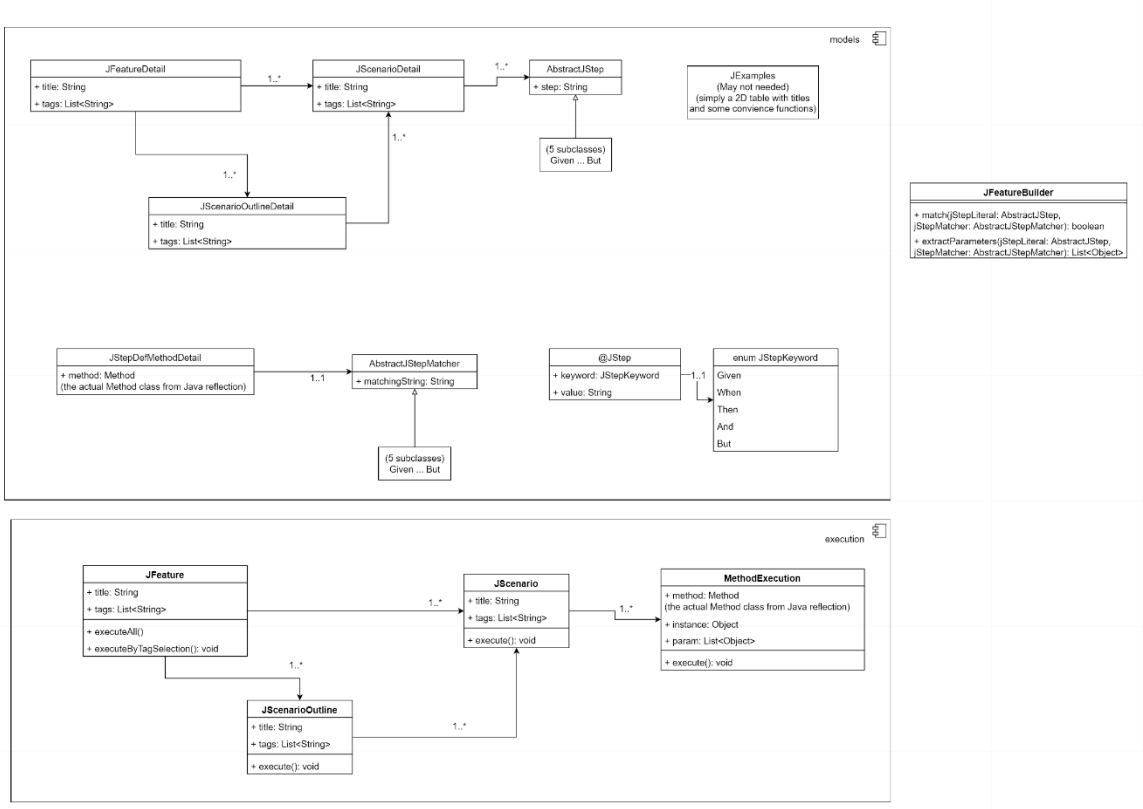

Figure 2 - UML Class Diagram of the new "Cucumberized" JUnit

`JScenarioDetail` and `JStepDefMethodDetail` represent the results from stage 1 and stage 2 from the procedure decomposition. Likewise, `JFeature` is the `executableJFeature` from stage 3 from the procedure decomposition. The procedure in the third stage can be further decomposed as two method calls shown in the `JFeatureBuilder` class.

Analysis and test

Unit tests are essential and will be applied for ensuring the core logic of this project is working properly. Furthermore, A comparison between Cucumber, Alexei's "Cucumberized" JUnit, and this improved "Cucumberized" JUnit will be made by writing BDD tests using these three different frameworks on complex scenarios of a typical Spring MVC web application that can handle race conditions.

If time allows it, the project will be released as a maven package and tested in a real situation. This will be done by picking up one or two students as customers of this project, asking them to use this improved "Cucumberized" JUnit on a real course project. A small questionnaire or interview can be held to receive feedback from chosen customers.

Schedule

At this time, investigation and brainstorming of the possible project have already been done. Hence the following schedule starts from the migration of Alexei's

“Cucumberized” JUnit. In this schedule, the improvement mentioned in the list of improvements will be labelled using their corresponding abbreviation. For example, 1a represents the improvement of making the project as a maven package.

Given 8 months of time, which is around 32 weeks, the schedule will be as following:

Time Period	Milestone (Things to be done by the date)
End of Week 1	• Migration of the original code to the new structure shown in figure 2. This also cover 3a, 3b • 4c
End of Week 2	• 3d
End of Week 3	• 4b
End of Week 4	• 4d
End of Week 5	• 4e
End of Week 6	• 4f
End of Week 7	• 3c • 1a
End of Week 8	• 4a
Start of Week 9	• Start comparison between Cucumber, Alexei’s “Cucumberized” JUnit and this improved “Cucumberized” JUnit
End of Week 9	• If found any defect of this improved “Cucumberized” JUnit, fix it
End of Week 10	• Start testing the project in real situation by finding one or two students to try out this project, given them 2 weeks of time of trying
Start of Week 11	• Selected two or three optional features, and given 4 weeks of time to implement them
End of Week 13	• Using a survey or interview, collect any feedbacks from students. And then analyse and decide any improvement that can be done from students’ feedback
End of Week 15	• Finish the implementation of selected optional features. • If possible, finish any improvement from student feedbacks • If need to, start testing the project in real situation again with all the improvements made so far. Given users 3 weeks of time trying
Start of Week 16 – End of Week 17	• Leave it empty to catch up any unfinished progress • Otherwise, start writing the report. The latest date for starting the report writing is the end of week 17
Start of Week 18	• Start more comparison between Cucumber and this improved "Cucumberized" JUnit. More effort will put on features that were not compared in the first round
End of Week 19	• If the new improvement was tried by anyone, collect feedbacks, analyse and decide improvement again
End of Week 22	• Any development and improvement should be finished

End of Week 25	• Finish the report • Start preparing for presentation
End of Week 29	• End of everything

In this schedule, 2a, which is the better exception handling, is not mentioned as it will be implemented during the development of other features and improvements.

References

[1] Cucumber, "Behaviour-Driven Development - Cucumber Documents," SmartBear Software, [Online]. Available: https://cucumber.io/docs/bdd/.

[2] Cucumber, "Tags," SmartBear Software, [Online]. Available: https://cucumber.io/docs/cucumber/api/#tags.

[3] Cucumber, "Background," SmartBear Software, [Online]. Available: https://cucumber.io/docs/gherkin/reference/#background.

[4] Cucumber, "Hooks," SmartBear Software, [Online]. Available: https://cucumber.io/docs/cucumber/api/#hooks.

[5] Cucumber, "Spoken Languages," SmartBear Software, [Online]. Available: https://cucumber.io/docs/gherkin/reference/#spoken-languages.

[6] Cucumber, "Type Registry," Spoken Languages, [Online]. Available: https://cucumber.io/docs/cucumber/configuration/#type-registry.