Automation with Cucumber Java Tutorial

Workbook for attendees

Gáspár Nagy

coach, trainer and BDD addict,
creator of SpecFlow

gaspar@specsolutions.eu
http://gasparnagy.com
@gasparnagy

A1: AUTOMATE MENU SCENARIO WITH CONTROLLER AUTOMATION

Automate scenario "Only active pizza menu items are listed on the menu page" using controller automation.

- Imagine what you would do to verify the scenario manually: Think it over how you would automate the different steps using controller automation.
- As you can see from the color of the steps, the "Given" step has been automated already. You will need no to automate the "When" and "Then" steps using the following steps.
- Generate step definition skeletons for the "When" and "Then" steps and add them to the existing `GeekPizzaStepDefinitions.java` file (you can find this class inside the folder `test/java/eu.specsolutions.bddcourse.geekpizza.controller_tests`). (To add step definitions to an existing file, you can use "lightbulb" function of the IDE or simply copy out the step definition snippet from the test result and paste it into the right class.)
- Implement step definitions to make the test fail: proves that the business rule illustrated by the scenario is not implemented yet! See Hints below, if you need help.
- Now (and only now) we can try to implement the application logic to satisfy the scenario. Check the `MenuController.getMenuItems()` method, and you will see how this could be done.
- *Bonus:* Take a look at the provided automation for the "Given" step. (You can navigate to the step definition from the feature file with *Go To Declaration* command of your IDE.) It saves the menu items directly to the database. *Is this a good idea? What are the disadvantages of it?*
- *Bonus:* If you have still some time, you can try automating the Given step using controller-automation (by using the `AdminController` class). Feel free to cleanup the code duplications as well in the Given step definition. You can also do this as a homework.

HINTS

- The items on the menu page are retrieved using the "`getMenuItems`" method of the "`MenuController`".
- How to call an API controller method? Check the step definition of the "When the client checks the home page" step!
- The method returns the items in a "`List<MenuItem>`" structure.
- How to work with a result returned by an API controller method? Check the step definition of the "When the client checks the home page" and the "Then the home page main message should be: "Welcome to Geek Pizza!"" steps!
- To compare the result with the expected count, you can use the JUnit "`assertEquals(expected, actual)`" assertion method.

A2: Automate DataTable assertion of the Menu scenario

Automate the missing "Then" step of the scenario "The pizzas are listed in alphabetically on the menu page" using controller automation with a simple FOR loop.

- Imagine what you would do to verify the scenario manually: Think it over how you would automate the different steps using controller automation.
- The "Given" and "When" steps have been defined (automated) already. You only need to work on the "Then" step. Review how the "Given" step is defined, you can see how you can work with DataTables (mapped to `List<Map<String, String>>` in this case) in Cucumber Java!
- Generate step definition skeleton for the "Then" step and add them to the existing `GeekPizzaStepDefinitions.java` file.
- Implement step definitions **using a FOR loop first**! Make the test fail: proves that the business rule illustrated by the scenario is not implemented yet! See Hints below, if you need help.
- Implement the application logic to satisfy the scenario. It will be close to the one you did for the previous exercise.
- Now the test pass, but the logic you wrote into the "Then" step is pretty complex. Are you sure that it is bug-free? Test the test by making the scenario wrong! Try to change pizza names in the scenario or add/remove pizzas.
- *Bonus:* Extend the step definition so that it can also assert on the ingredients. Can you make it in a way that it works for both cases (with and without ingredients)? Use `Map<String, String>.ContainsKey()`!

Hints

- A for loop that iterates through a list looks like this:

```
1. for (int i = 0; i < mylist.size(); i++)
   { … do somwthing with mylist.get(i) … }
```

- A row of the data table can be retrieved by `myTable.get(i)`.
- A cell within the row can be retrieved by `myRow.get("header")` (or `myTable.get(i).get("header")`).

A3: Split Feature files and Step definition classes

Split feature file and step definition class to ensure a logical structure

- This is a refactoring exercise, so make sure the tests are passing before you would make any changes.

- Move Menu-related scenarios to a new feature file: `menu.feature`. *Do they still pass? Did you have to make any changes in the step definitions to make it pass? Why?*
- The two "Given" step definitions are all related to the "admin" interface of the application. Extract them to a separate step definition class: `AdminStepDefinitions.java`.
- Split the remaining step definitions in `GeekPizzaStepDefinitions` to `HomeStepDefinitions` and `MenuStepDefinitions` accordingly.

A4: SHARE DATA BETWEEN STEP DEFINITIONS IN DIFFERENT CLASSES

Practice data sharing between step definitions placed in different classes.

- This is a refactoring exercise, so make sure the tests are passing before you would make any changes.
- The step definitions that are related to the same features of the system should be kept in the same step definition class. We have seen this in `HomeStepDefinitions` (the `HomePageModel` was shared) and also in `MenuStepDefinitions` (the list of pizzas was shared).
- In this exercise, there is a new scenario introduced in `Home.feature`: "The logged-in user name is shown on home page". The related step definitions are currently automated in the `HomeStepDefinitions` class, but to keep a healthy structure, we would like to move the `theClientIsLoggedIn` method to the `AuthStepDefinitions`, because that does not belong to the home page but to the authentication.
- We have learned how to move step definitions to another class, but this one shares data (the authentication token, stored in the `authToken` field) with the `theClientChecksTheHomePage` method, that supposed to remain in `HomeStepDefinitions`. To fix this, we will use **Dependency Injection (DI).** As this project uses Spring Boot, we are going to use the Spring DI mechanism, but the same would work with all major DI frameworks.
- To share the data with DI, you need to do the following step:
    1. Review the context class that has been prepared for you already: `AuthContext`. (Normally you would need to create such a class with properties for the data you want to share.) The class has been annotated with `@Component @ScenarioScope` that enables managing it by DI. The `@ScenarioScope` sets the lifetime of it for the lifetime of the scenario execution (each scenario execution will get a new one).
    2. Inject the context into *both step definition classes*, by declaring a class variable and annotate it with `@Autowired`:

```
public class SomeStepDefinitions
{
    @Autowired
    private AuthContext authContext;
    ...
}
```

3. Wherever the code used the `authToken` field (for reading or writing), use the `getAuthToken()` and `setAuthToken()` methods of the context class instead: `authContext.getAuthToken()`

- The scenarios are passing again! This means that Spring created instances of the AuthContext class whenever it was needed. *How many `AuthContext` instance are created if you run all scenarios. Why?*
- *Bonus:* The `theUserNameOfTheClientShouldBeOnTheHomePage` method of the `HomeStepDefinitions` has the user name "Marvin" hard-coded. Extend the AuthContext class so that it "remembers" the name of the currently logged in user and use that instead.

A5: Practice using Scenario Outlines

> Convert the scenarios in the rule "The password should be valid and verified" of registration.feature to a Scenario Outline.

- There are three scenarios within the rule "The password should be valid and verified" of the registration.feature file that check the password validation of the registration process.
- Replace the three scenarios with a single Scenario Outline with three examples:
    1. Password was not provided
    2. The re-entered password is different
    3. Password is too short
- You can find the Scenario Outline syntax on the Gherkin Cheat Sheet.
- Add an additional first column (e.g. "description") to the Examples table to describe the purpose of the individual examples. Check the name of the generated tests!
- *Bonus:* Add a second "Examples" block and provide further test cases. What is the difference of a "test case" and an "illustrative scenario"?
- *Bonus:* How could you include the scenario "User name was not provided" as well into your scenario outline. What problems you see with the approach you have chosen?

B1: Review scenario automated using Web API automation

> Review the automation of the scenario "The user name should be shown on the home page if logged in" to find potential maintenance and scaling problems.

- We have switched over to a new automation solution that automates the Web API. For the sake of simplicity, the scenarios we have automated using the API controllers have been removed.
- The current codebase contains a single scenario: "The user name should be shown on the home page if logged in" in the `home.feature`. The scenario has been automated already using the step

definitions in the `WebApiStepDefinitions` class. The scenario should be passing (make sure it does).

- The current automation solution is a "simplest possible" approach and while it makes the scenario pass it has a few problems. *Review the step definitions and make a note of the ones you find.*
- The following orientation questions might help:
    1. Is the code clean? Are there any code duplications?
    2. Assuming that the automation solution will contain many other similar GET and POST requests, would you use these automation blocks as samples for those? Why?
    3. Do these step definitions support reusability (reusing the steps in other scenarios, potentially in different order)?
    4. Does the current structure of the code support structure the step definitions into different step definition classes?
- *How could you fix these issues?* (You should not start fixing it, just make notes so that we can discuss together.)
- *Bonus:* What is the difference between sanity and functional checks? Why are they separated?

B2: AUTOMATE SCENARIOS USING WEB API AUTOMATION

Automate "Registration" and "Menu" scenarios using Web API automation.

- The "Given" step has been already automated in `AdminStepDefinitions.java` (it adds the expected pizzas to the database).
- Automate the remaining steps by making POST and GET requests. The exact method, endpoint and input/output data you can find in the feature files as `#TODO` comments.
- The (empty) step definition classes are prepared (`RegistrationStepDefinitions` and `MenuStepDefinitions`) and the `WebApiContext` has been injected already.
- Check `HomeStepDefinitions.theClientChecksTheHomePage` and `AuthStepDefinitions.theClientIsLoggedIn` methods to see how you can use the WebApi Context to make a GET and POST request.
- *Was it easy? Did you enjoy the separation of the infrastructural details from the functional parts of the automation?*