FROM TDD TO BDD AND BACK

HANDS ON LAB

GETTING STARTED GUIDE CUCUMBER

Mieke Kemme and Elke Steegmans

May 17, 2016

TABLE OF CONTENTS

INTRODUCTION

CUCUMBER

Cucumber lets software development teams describe how software should behave in plain text. The text is written in a business-readable domain-specific language and serves as documentation, automated tests and development-aid - all rolled into one format.

Official Website: https://cucumber.io/

THIS LAB

This lab should give you a practical introduction to writing executable specifications with *Cucumber*.

You receive:

- a small demo-application written in Java
- a few user stories
- scenarios for one of the stories

The goal of this lab is to write a few specifications for this application and make them executable, using the tool *Cucumber.*

This lab is organized as follows:

1. *installation* of the demo-application and the software needed to write the tests,
2. follow the *step-by-step* guide to make the scenarios provided executable. This should give you the general idea of the tool,
3. play around and write the *scenarios* for another user story yourself. Look at the *whole picture*: can you make better user stories this way?

At the end we of the lab we reserve 15 minutes to discuss the differences between the tools.

MATERIAL

You can find all the material of this lab on GitHub: https://github.com/thinfir/think-first

CONTEXT

The example application used in this lab monitors the physical state of patients. A patient is examined on a regular base. The examination details (length, weight) are registered. These data can be used to calculate the body mass index (BMI) of the patient.

For the user stories and the design of the application, check the *GitHub* repository in the folder */context.*

In this lab, we will test the method *getPerson()* of the class *PersonServiceJavaApi*.

SETUP

PREREQUISITES

- Eclipse
- Java
- Maven
- Maven-eclipse-plugin: http://download.eclipse.org/technology/m2e/releases

INSTALLATION ECLIPSE PLUGIN

Eclipse installer:

- In the menu choose *Help* | *Install New Software …*
- Click *Add…* to add a new software site
- Enter `Cucumber` as *Name* and http://cucumber.github.com/cucumber-eclipse/update-site as *Location*.
- Choose *OK*
- Select *Cucumber Eclipse Plugin* and choose *Next*
- *Next*
- Accept the license agreement and choose *Finish*
- Ignore the warning and choose *OK*
- Choose *Yes* to restart Eclipse

CREATE ECLIPSE PROJECT

1. Create a new workspace: *File* | *Switch Workspace* | *Other …*
2. Get the bmi-app for *Cucumber* from the GitHub repository. You find it in the folder */cucumber/bmi-app*
3. Import de bmi-application in *Eclipse*:
    - *File* | *Import…* | *Maven* | *Existing Maven Projects*
    - Click *Next*
    - For the field *Root Directory:* browse to the folder bmi-app and choose *Open*
    - The pom.xml should appear in the *Projects* list
    - Click *Finish*

    The project is created. In the root of your project, you will find the *pom.xml*. Click on the file and choose *Run As | Maven install* to check if you can build the application.

DEVELOPMENT

INTRODUCTION: THE GENERAL IDEA

To write executable specifications with *Cucumber*, you always have to write 3 types of files:

1. The **feature**: a plain text file containing the specifications in Given-When-Then style. The filename is in CamelCase and the extension is .feature

   Example: `ShowPatientDetails.feature`

2. A **configuration file**: a special kind of Java test class that is used to run the steps as a JUnit test class. The filename is the same name as the name of the story + Test and with extension .java

   Example: `ShowPatientDetailsTest.java`

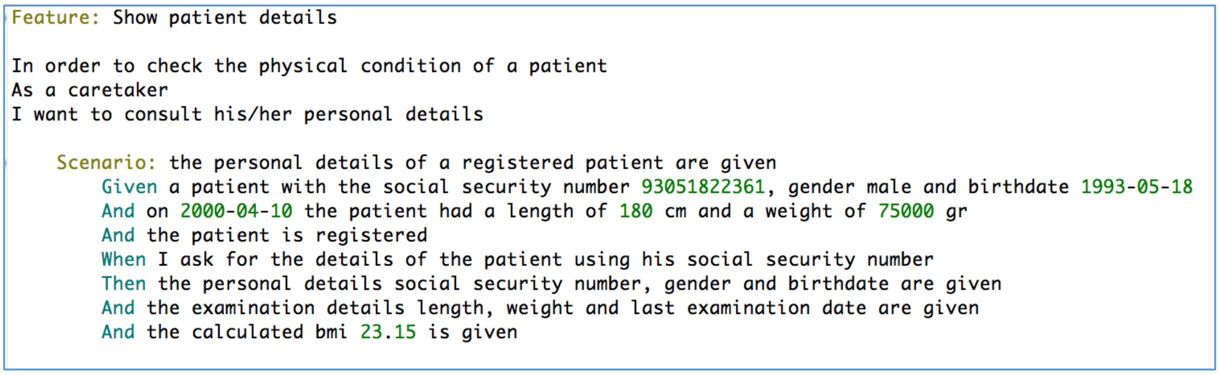
3. The **story steps**: a Java test class containing the java code to make each Given-, When- or Then- step executable. The filename is the same name as the name of the story + Steps and with extension .java

   Example: `ShowPatientDetailsSteps.java`

In the next steps you will be guided through a first example using *Cucumber*. You can perform these steps yourself, starting from scratch, or you can use the example solution and take it from there. The solution of this first example can be found in the *GitHub* repository in the folder */cucumber/gettingstarted*, in the file `ShowPatientDetailsTest.java`.

STEP 1: WRITE SPECIFICATIONS

1. Create a package `org.ucll.demo.service` in the test folder *src/test/resources*.
2. Create a new feature:
   - Choose *New* | *Other…* | *General* | *File*
   - Click *Next*
   - Enter the name of the feature, i.e. `ShowPatientDetails.feature`
     - Make sure that the extension is `.feature`
   - Click *Finish*
   - The feature is created
3. Overwrite the existing tekst of the feature file with the story and the scenario given in the file *specification_1.txt* in the GitHub folder */cucumber/gettingstarted*
4. Choose *Save*.



```
Feature: Show patient details

In order to check the physical condition of a patient
As a caretaker
I want to consult his/her personal details

    Scenario: the personal details of a registered patient are given
        Given a patient with the social security number 93051822361, gender male and birthdate 1993-05-18
        And on 2000-04-10 the patient had a length of 180 cm and a weight of 75000 gr
        And the patient is registered
        When I ask for the details of the patient using his social security number
        Then the personal details social security number, gender and birthdate are given
        And the examination details length, weight and last examination date are given
        And the calculated bmi 23.15 is given
```

Figure 1. Example feature show patient details

STEP 2: CONFIGURATION FILE

1. Create also a package `org.ucll.demo.service` in the test folder *src/test/java* and create here a Java class with the configuration, i.e. `ShowPatientDetailsTest.java`.

2. Add the following before the name of the class:

   ```
   @RunWith(Cucumber.class)
   ```

   and add the import statements:

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

3. Add the following between the name of the class and the @RunWith annotation

   ```
   @CucumberOptions(
   	features={"src/test/resources/org/ucll/demo/service/ShowPatient
   	Details.feature"},
   	plugin={"html:target/cucumber", "json:target/cucumber.json"}
   )
   ```

   and add the import statement:

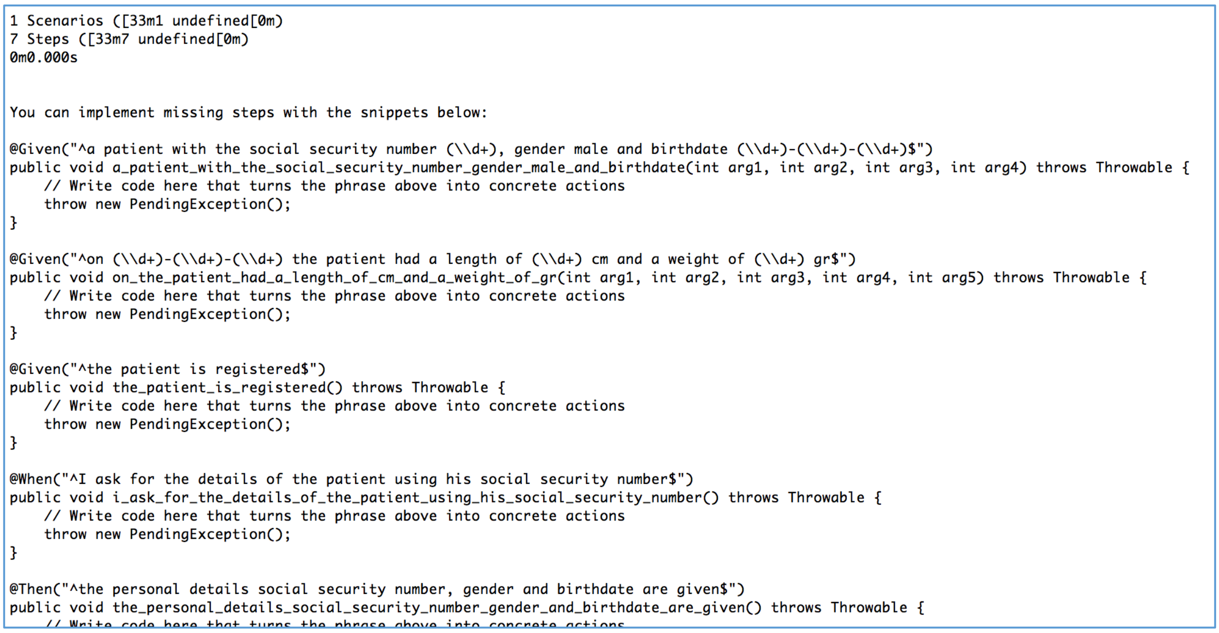
   ```
   import cucumber.api.CucumberOptions;
   ```

EXECUTE SPECIFICATIONS

Select the configuration file, right mouse click and choose *Run as JUnit Test.*

In the **output console** you can see the result:

- The number of scenarios and steps ran
- For each scenario, the steps with the current status
- For each step an empty Java method with annotation is generated. The annotations are used to:
  - map the method to a step in the story
  - say the step is pending (=no useful test is executed)

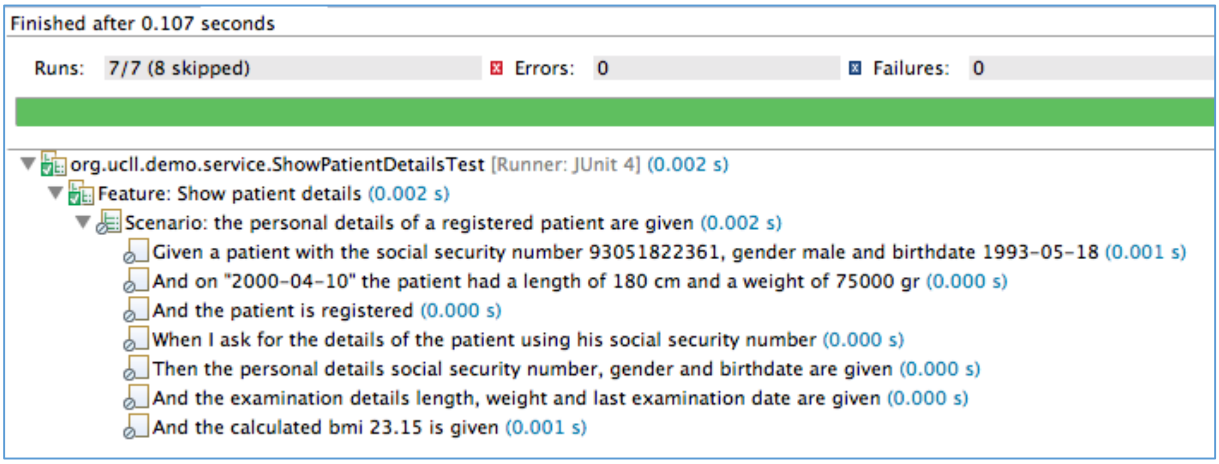


```
1 Scenarios ([33m1 undefined[0m)
7 Steps ([33m7 undefined[0m)
0m0.000s


You can implement missing steps with the snippets below:

@Given("^a patient with the social security number (\\d+), gender male and birthdate (\\d+)-(\\d+)-(\\d+)$")
public void a_patient_with_the_social_security_number_gender_male_and_birthdate(int arg1, int arg2, int arg3, int arg4) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@Given("^on (\\d+)-(\\d+)-(\\d+) the patient had a length of (\\d+) cm and a weight of (\\d+) gr$")
public void on_the_patient_had_a_length_of_cm_and_a_weight_of_gr(int arg1, int arg2, int arg3, int arg4, int arg5) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@Given("^the patient is registered$")
public void the_patient_is_registered() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@When("^I ask for the details of the patient using his social security number$")
public void i_ask_for_the_details_of_the_patient_using_his_social_security_number() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@Then("^the personal details social security number, gender and birthdate are given$")
public void the_personal_details_social_security_number_gender_and_birthdate_are_given() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
```

Figure 2. Example console output

In the **JUnit console** you can see the result:

- The number of scenarios and steps ran
- For each scenario, the steps with the current status



```
Finished after 0.107 seconds

Runs:  7/7 (8 skipped)     ☒ Errors:  0     ☒ Failures:  0

▼ org.ucll.demo.service.ShowPatientDetailsTest [Runner: JUnit 4] (0.002 s)
  ▼ Feature: Show patient details (0.002 s)
    ▼ Scenario: the personal details of a registered patient are given (0.002 s)
        Given a patient with the social security number 93051822361, gender male and birthdate 1993-05-18 (0.001 s)
        And on "2000-04-10" the patient had a length of 180 cm and a weight of 75000 gr (0.000 s)
        And the patient is registered (0.000 s)
        When I ask for the details of the patient using his social security number (0.000 s)
        Then the personal details social security number, gender and birthdate are given (0.000 s)
        And the examination details length, weight and last examination date are given (0.000 s)
        And the calculated bmi 23.15 is given (0.001 s)
```

Figure 3. Example JUnit output

STEP 3: CREATE THE STEPS FILE

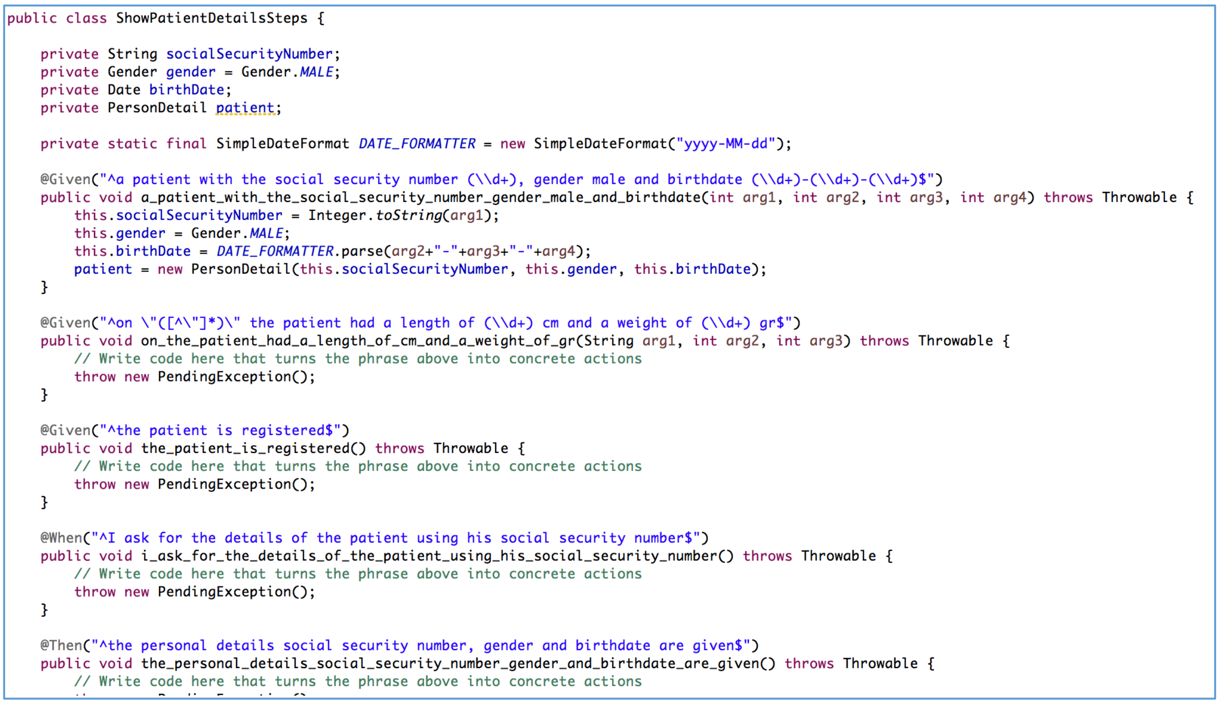
1. Create in the test folder *src/test/java* in the package `org.ucll.demo.service` a Java class for the story steps, i.e. `ShowPatientDetailsSteps.java`

MAKE THE STEPS EXECUTABLE

1. Copy the method headings outputted in the console to the Steps-file `ShowPatientDetailsSteps.java`
2. Add the import statements:

```
import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;
import cucumber.api.PendingException;
```

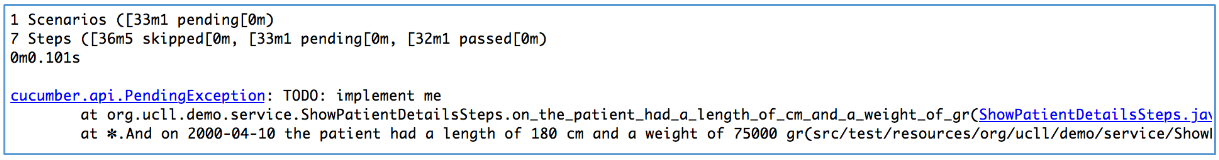
3. Implement the first step-method:
   - remove the *throw new PendingException();*
   - call the necessary code to connect to your actual domain classes…



```
public class ShowPatientDetailsSteps {

    private String socialSecurityNumber;
    private Gender gender = Gender.MALE;
    private Date birthDate;
    private PersonDetail patient;

    private static final SimpleDateFormat DATE_FORMATTER = new SimpleDateFormat("yyyy-MM-dd");

    @Given("^a patient with the social security number (\\d+), gender male and birthdate (\\d+)-(\\d+)-(\\d+)$")
    public void a_patient_with_the_social_security_number_gender_male_and_birthdate(int arg1, int arg2, int arg3, int arg4) throws Throwable {
        this.socialSecurityNumber = Integer.toString(arg1);
        this.gender = Gender.MALE;
        this.birthDate = DATE_FORMATTER.parse(arg2+"-"+arg3+"-"+arg4);
        patient = new PersonDetail(this.socialSecurityNumber, this.gender, this.birthDate);
    }

    @Given("^on \"([^\"]*)\" the patient had a length of (\\d+) cm and a weight of (\\d+) gr$")
    public void on_the_patient_had_a_length_of_cm_and_a_weight_of_gr(String arg1, int arg2, int arg3) throws Throwable {
        // Write code here that turns the phrase above into concrete actions
        throw new PendingException();
    }

    @Given("^the patient is registered$")
    public void the_patient_is_registered() throws Throwable {
        // Write code here that turns the phrase above into concrete actions
        throw new PendingException();
    }

    @When("^I ask for the details of the patient using his social security number$")
    public void i_ask_for_the_details_of_the_patient_using_his_social_security_number() throws Throwable {
        // Write code here that turns the phrase above into concrete actions
        throw new PendingException();
    }

    @Then("^the personal details social security number, gender and birthdate are given$")
    public void the_personal_details_social_security_number_gender_and_birthdate_are_given() throws Throwable {
        // Write code here that turns the phrase above into concrete actions
```

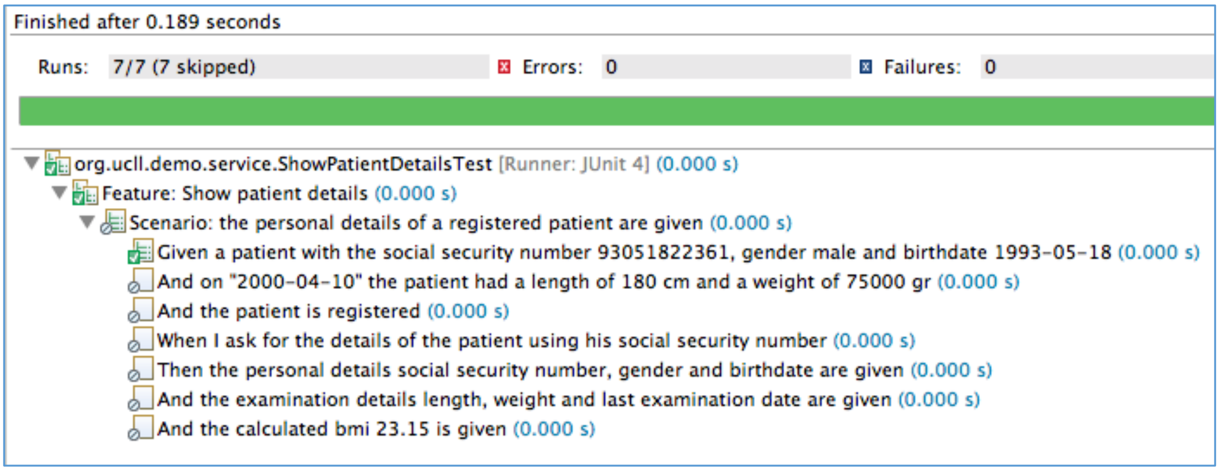
Figure 4. Example implementation of the first step

4. Run the Java story configuration again as a JUnit Test. The console and JUnit output has changed:
   - The Java methods have disappeared
   - The step you implemented is not PENDING anymore



```
1 Scenarios ([33m1 pending[0m)
7 Steps ([36m5 skipped[0m, [33m1 pending[0m, [32m1 passed[0m)
0m0.101s

cucumber.api.PendingException: TODO: implement me
	at org.ucll.demo.service.ShowPatientDetailsSteps.on_the_patient_had_a_length_of_cm_and_a_weight_of_gr(ShowPatientDetailsSteps.jav
	at *.And on 2000-04-10 the patient had a length of 180 cm and a weight of 75000 gr(src/test/resources/org/ucll/demo/service/Show
```

Figure 5.Example console output after implementing a step



```
Finished after 0.189 seconds
  Runs:  7/7 (7 skipped)          Errors:  0          Failures:  0

▼ org.ucll.demo.service.ShowPatientDetailsTest [Runner: JUnit 4] (0.000 s)
  ▼ Feature: Show patient details (0.000 s)
    ▼ Scenario: the personal details of a registered patient are given (0.000 s)
        Given a patient with the social security number 93051822361, gender male and birthdate 1993-05-18 (0.000 s)
        And on "2000-04-10" the patient had a length of 180 cm and a weight of 75000 gr (0.000 s)
        And the patient is registered (0.000 s)
        When I ask for the details of the patient using his social security number (0.000 s)
        Then the personal details social security number, gender and birthdate are given (0.000 s)
        And the examination details length, weight and last examination date are given (0.000 s)
        And the calculated bmi 23.15 is given (0.000 s)
```

Figure 6. Example JUnit output after implementing a step

5. Implement all the steps until you have a successful test

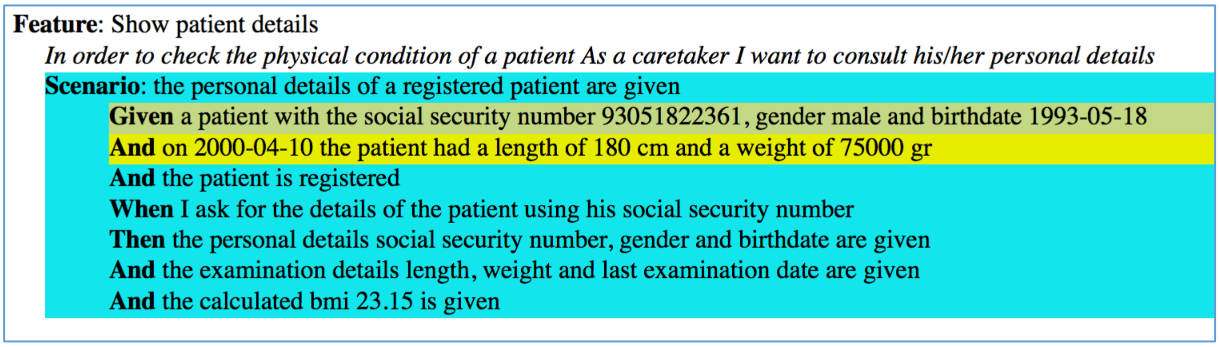
CONSULT THE HTML REPORT

Run the Maven build:

- Go to the pom.xml file and remove the <!-- and --> in this file (otherwise not all reporting is generated)
- Select your project
- Right mouse click
- Choose *Run As | Maven install*
  The build will be executed: the java code will be compiled, tested, … outside Eclipse

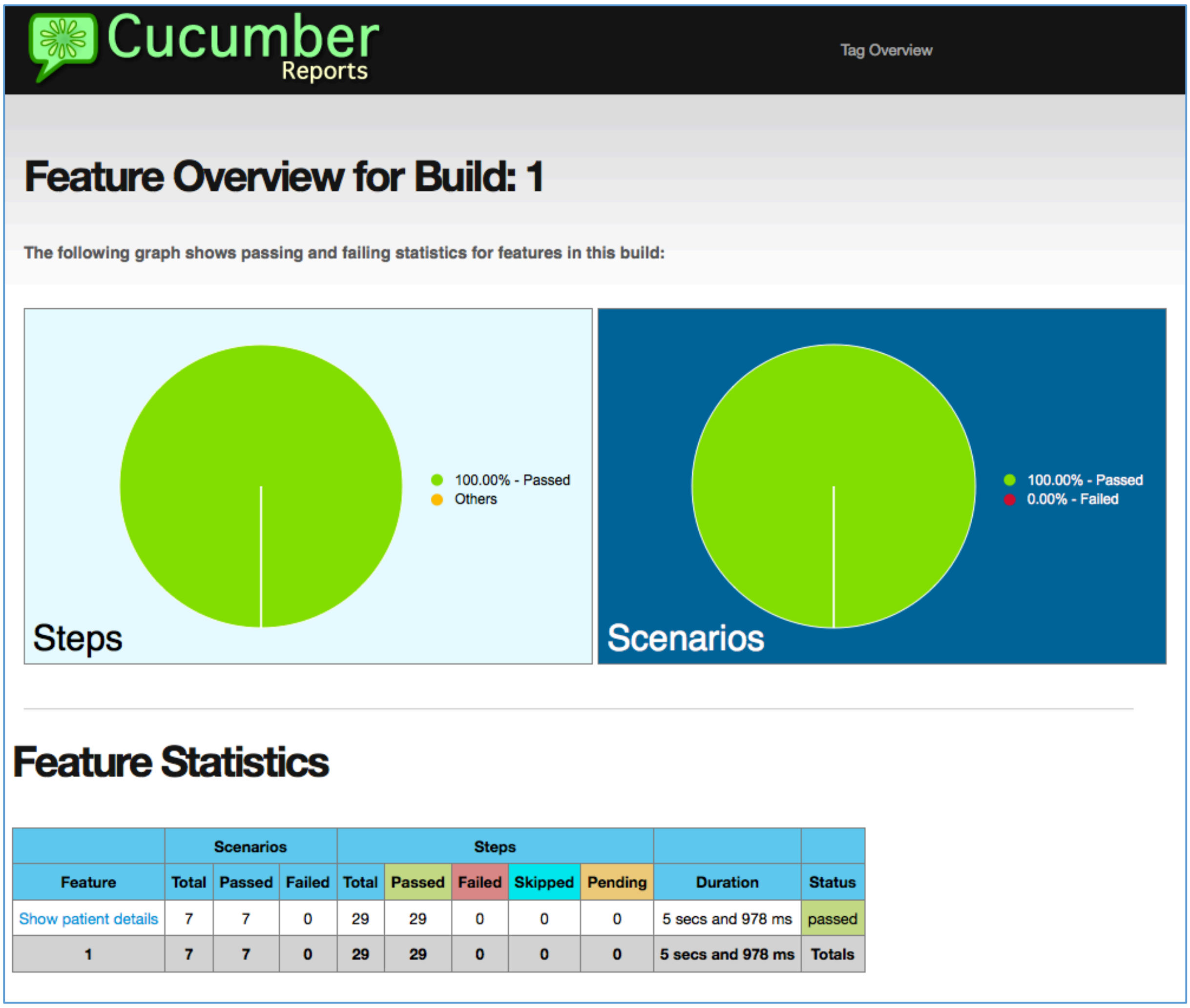
Consult the report:

- Refresh your project. You will find a folder *target*.
- In the subfolder *target/* you can find two subfolder with HTML files:
  - Open in *cucumber/* the `index.html` file, this is a HTML report form the test results point of view
  - Open in *cucumber-html-reports/* the `feature-overview.html` file, this is a HTML report from the feature point of view.



Feature: Show patient details
In order to check the physical condition of a patient As a caretaker I want to consult his/her personal details
Scenario: the personal details of a registered patient are given
Given a patient with the social security number 93051822361, gender male and birthdate 1993-05-18
And on 2000-04-10 the patient had a length of 180 cm and a weight of 75000 gr
And the patient is registered
When I ask for the details of the patient using his social security number
Then the personal details social security number, gender and birthdate are given
And the examination details length, weight and last examination date are given
And the calculated bmi 23.15 is given

Figure 7. HTML report from the feature result point of view



Feature Overview for Build: 1

The following graph shows passing and failing statistics for features in this build:

Feature Statistics

	Scenarios			Steps						
Feature	Total	Passed	Failed	Total	Passed	Failed	Skipped	Pending	Duration	Status
Show patient details	7	7	0	29	29	0	0	0	5 secs and 978 ms	passed
1	**7**	**7**	**0**	**29**	**29**	**0**	**0**	**0**	**5 secs and 978 ms**	**Totals**

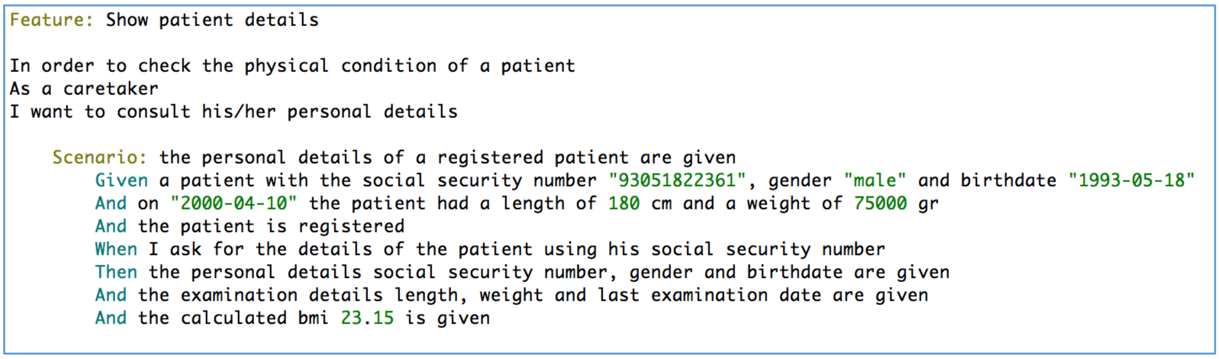
Figure 8. HTML report from the test point of view

EXTRA

FINETUNE PARAMETERS

The method headings in our Steps-file are not very nice, with only int or double types of parameters. We can refactor this to make them more reusable, by redefining the steps in the feature file and by using parameters:

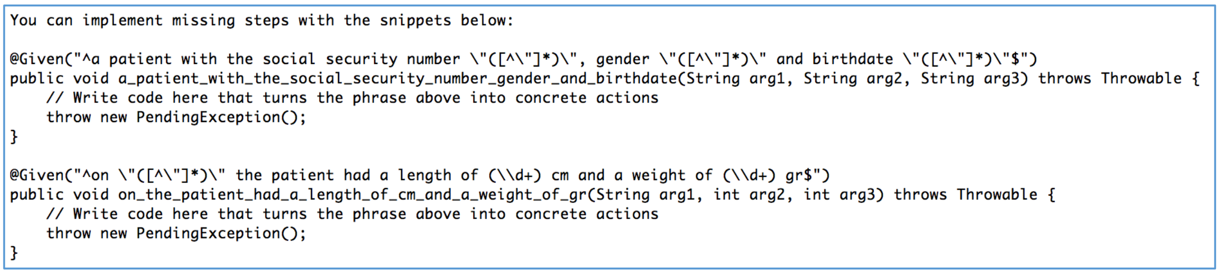
1. In the Given step set the examples that you want to use as String inputs between “”



```
Feature: Show patient details

In order to check the physical condition of a patient
As a caretaker
I want to consult his/her personal details

    Scenario: the personal details of a registered patient are given
        Given a patient with the social security number "93051822361", gender "male" and birthdate "1993-05-18"
        And on "2000-04-10" the patient had a length of 180 cm and a weight of 75000 gr
        And the patient is registered
        When I ask for the details of the patient using his social security number
        Then the personal details social security number, gender and birthdate are given
        And the examination details length, weight and last examination date are given
        And the calculated bmi 23.15 is given
```

Figure 9. The usage of more parameters in the feature

2. Run the test.



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

@Given("^a patient with the social security number \"([^\"]*)\", gender \"([^\"]*)\" and birthdate \"([^\"]*)\"$")
public void a_patient_with_the_social_security_number_gender_and_birthdate(String arg1, String arg2, String arg3) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@Given("^on \"([^\"]*)\" the patient had a length of (\\d+) cm and a weight of (\\d+) gr$")
public void on_the_patient_had_a_length_of_cm_and_a_weight_of_gr(String arg1, int arg2, int arg3) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}
```

Figure 10. Extra parameters are added to the steps now

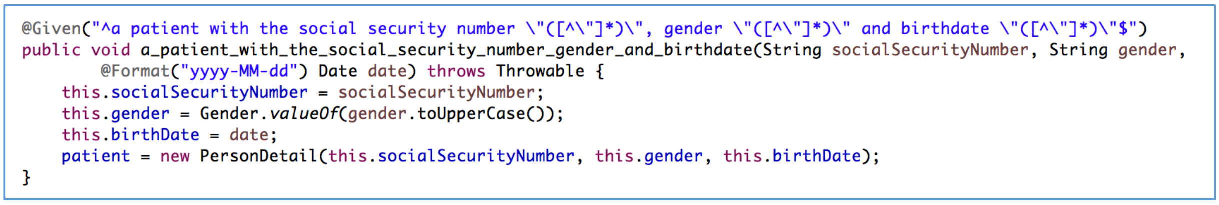
3. Copy the new step to implement in the steps class.
4. Rename the parameters of the method into readable parameter names.

   Example: change `arg1` into `socialSecurityNumber`

5. Change the types of the parameters if you want to make your life easier. Another example: you can parse the date by using the annotation @Format

   Example: change String `arg3` into `@Format(“yyyy-MM-dd”) Date date`

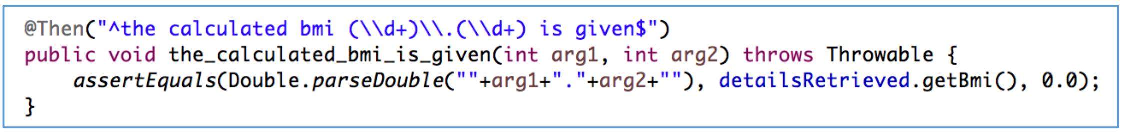
6. Use the method parameters in your method body.



```
@Given("^a patient with the social security number \"([^\"]*)\", gender \"([^\"]*)\" and birthdate \"([^\"]*)\"$")
public void a_patient_with_the_social_security_number_gender_and_birthdate(String socialSecurityNumber, String gender,
        @Format("yyyy-MM-dd") Date date) throws Throwable {
    this.socialSecurityNumber = socialSecurityNumber;
    this.gender = Gender.valueOf(gender.toUpperCase());
    this.birthDate = date;
    patient = new PersonDetail(this.socialSecurityNumber, this.gender, this.birthDate);
}
```

Figure 11. The usage of parameters

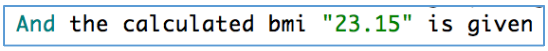
7. Have a look at the implementation of the “the calculated bmi 23.15 is given“ step. The strange thing here is that the double 23.15 from the specification is translated to two integer parameters. As such we need to parse it in our code to a double, which is not the best way to implement it.



```
@Then("^the calculated bmi (\\d+)\\.(\\d+) is given$")
public void the_calculated_bmi_is_given(int arg1, int arg2) throws Throwable {
    assertEquals(Double.parseDouble(""+arg1+"."+arg2+""), detailsRetrieved.getBmi(), 0.0);
}
```

Figure 12. A double from the specification is translated to two integer values.

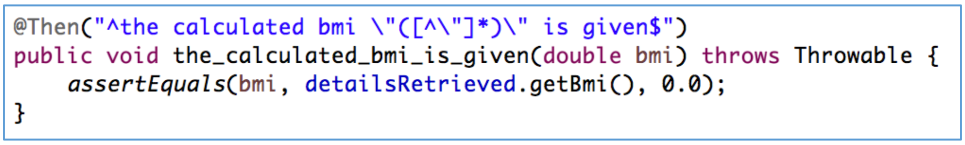
8. You can refactor this in two different ways
   a. by changing the Then step in the specification to



```
And the calculated bmi "23.15" is given
```

Figure 13. Using a string in the specification and then changing the parameter type.

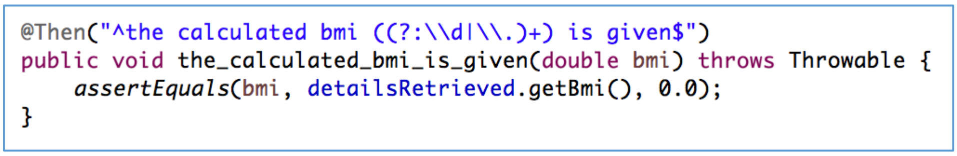
In the generated step, the type of the parameter is a string, we change it to double and then the implementation is easier.



```
@Then("^the calculated bmi \"([^\"]*)\" is given$")
public void the_calculated_bmi_is_given(double bmi) throws Throwable {
    assertEquals(bmi, detailsRetrieved.getBmi(), 0.0);
}
```

Figure 14. Using a string in the specification and then changing the parameter type.

b. by using another and better regex for the parameter in order to make it a double right away.



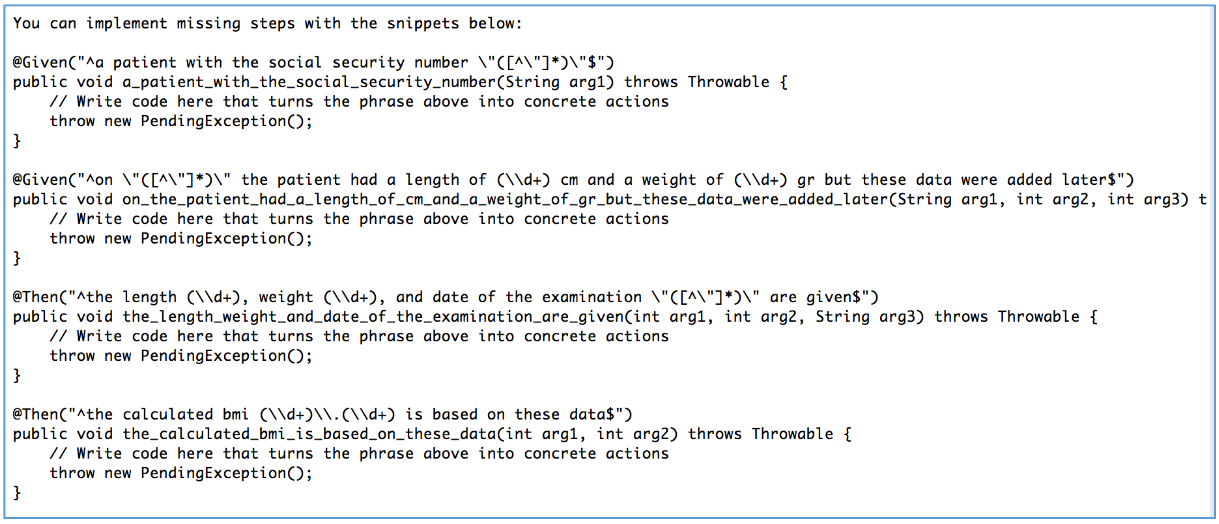
```
@Then("^the calculated bmi ((?:\\d|\\.)+) is given$")
public void the_calculated_bmi_is_given(double bmi) throws Throwable {
    assertEquals(bmi, detailsRetrieved.getBmi(), 0.0);
}
```

Figure 15. Using another regex makes it a double right away.

The solution of this first example can be found in the *GitHub* repository in the folder */cucumber/gettingstarted*, in the file `ShowPatientDetailsTest.java`.

REUSE STEPS

1. Add the scenarios given in the file *specification_2.txt* in the *GitHub* folder */cucumber/gettingstarted/specification_2.txt*
2. Run the test. In the console you can see the *@Given* and *@Then* steps are created, but there is no suggestion for the *@When* steps. That's because Cucumber saw we can reuse the *@When* step from the previous scenario…

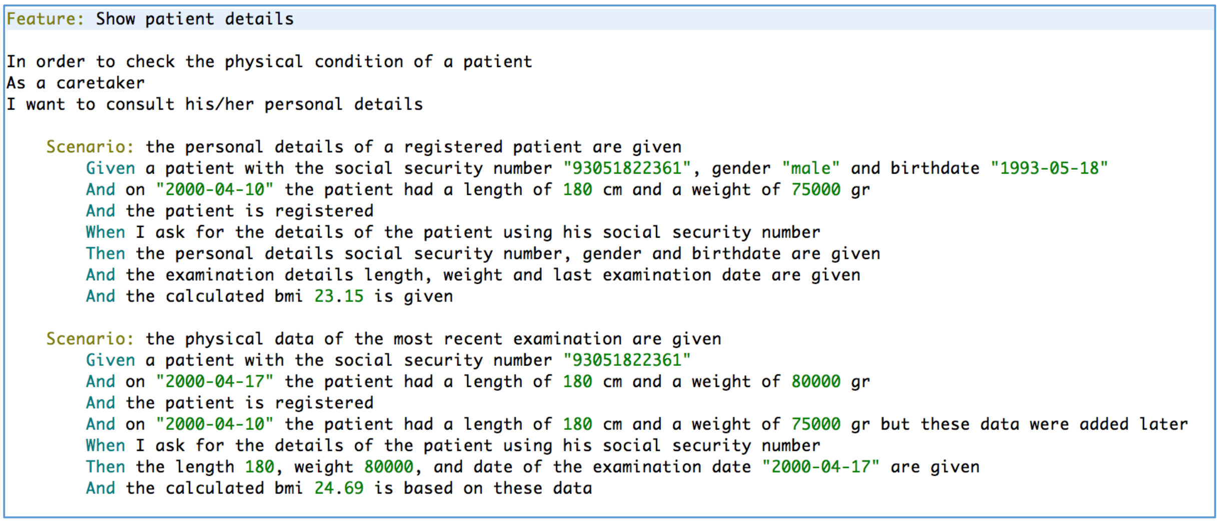


```
You can implement missing steps with the snippets below:

@Given("^a patient with the social security number \"([^\"]*)\"$")
public void a_patient_with_the_social_security_number(String arg1) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@Given("^on \"([^\"]*)\" the patient had a length of (\\d+) cm and a weight of (\\d+) gr but these data were added later$")
public void on_the_patient_had_a_length_of_cm_and_a_weight_of_gr_but_these_data_were_added_later(String arg1, int arg2, int arg3) t
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@Then("^the length (\\d+), weight (\\d+), and date of the examination \"([^\"]*)\" are given$")
public void the_length_weight_and_date_of_the_examination_are_given(int arg1, int arg2, String arg3) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@Then("^the calculated bmi (\\d+)\\.(\\d+) is based on these data$")
public void the_calculated_bmi_is_based_on_these_data(int arg1, int arg2) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}
```

Figure 16. Example console output adding a second story – existing steps are left out

SETUP AND TEARDOWN

When looking at the scenarios given in the picture below, you can see that the *@Given* steps are almost the same.

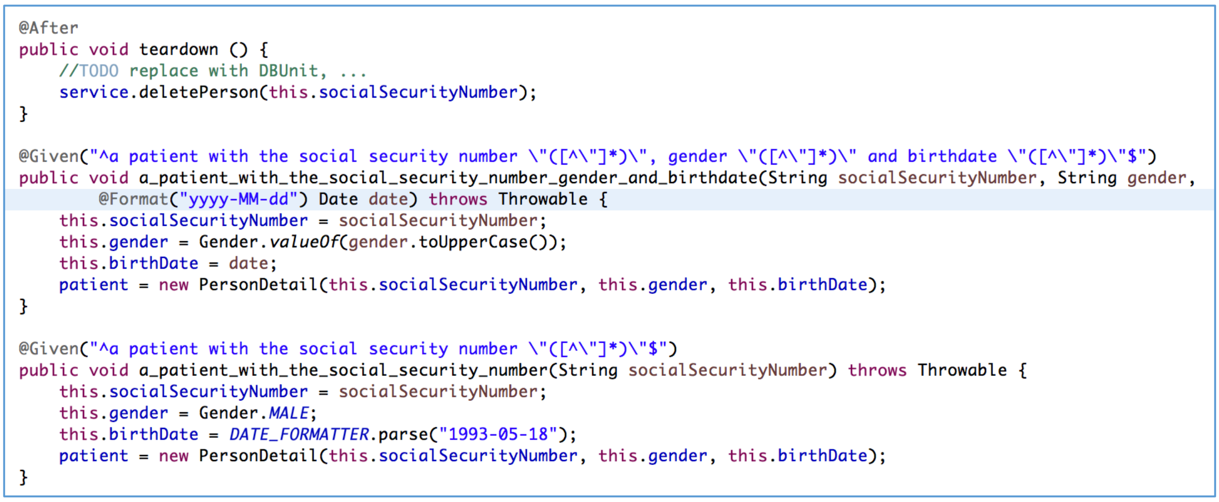


```
Feature: Show patient details

In order to check the physical condition of a patient
As a caretaker
I want to consult his/her personal details

    Scenario: the personal details of a registered patient are given
        Given a patient with the social security number "93051822361", gender "male" and birthdate "1993-05-18"
        And on "2000-04-10" the patient had a length of 180 cm and a weight of 75000 gr
        And the patient is registered
        When I ask for the details of the patient using his social security number
        Then the personal details social security number, gender and birthdate are given
        And the examination details length, weight and last examination date are given
        And the calculated bmi 23.15 is given

    Scenario: the physical data of the most recent examination are given
        Given a patient with the social security number "93051822361"
        And on "2000-04-17" the patient had a length of 180 cm and a weight of 80000 gr
        And the patient is registered
        And on "2000-04-10" the patient had a length of 180 cm and a weight of 75000 gr but these data were added later
        When I ask for the details of the patient using his social security number
        Then the length 180, weight 80000, and date of the examination date "2000-04-17" are given
        And the calculated bmi 24.69 is based on these data
```

Figure 17. **Two almost identical *@Given* steps**

As this is an integration test, the patient created in first scenario was added to the 'database'. We do not want one scenario to interfere with another scenario, so we have to remove the patient after the test is executed.

You can do this by creating a teardown method, using the `@After` annotation from Cucumber. This method is called after the execution of each scenario. You need to import the right package here: `import cucumber.api.java.After`



```
@After
public void teardown () {
    //TODO replace with DBUnit, ...
    service.deletePerson(this.socialSecurityNumber);
}

@Given("^a patient with the social security number \"([^\"]*)\", gender \"([^\"]*)\" and birthdate \"([^\"]*)\"$")
public void a_patient_with_the_social_security_number_gender_and_birthdate(String socialSecurityNumber, String gender,
        @Format("yyyy-MM-dd") Date date) throws Throwable {
    this.socialSecurityNumber = socialSecurityNumber;
    this.gender = Gender.valueOf(gender.toUpperCase());
    this.birthDate = date;
    patient = new PersonDetail(this.socialSecurityNumber, this.gender, this.birthDate);
}

@Given("^a patient with the social security number \"([^\"]*)\"$")
public void a_patient_with_the_social_security_number(String socialSecurityNumber) throws Throwable {
    this.socialSecurityNumber = socialSecurityNumber;
    this.gender = Gender.MALE;
    this.birthDate = DATE_FORMATTER.parse("1993-05-18");
    patient = new PersonDetail(this.socialSecurityNumber, this.gender, this.birthDate);
}
```

Figure 18. **Add *@After* teardown method to clean object used in multiple scenarios**

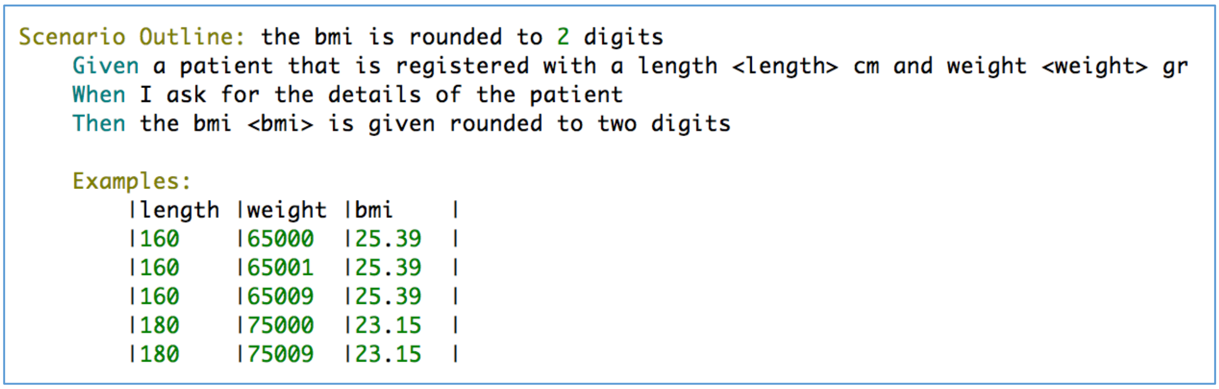
In the same way you can also use a setup method, using the `@Before` annotation from Cucumber. This method is called before the execution of each scenario.

SCENARIOS WITH EXPECTED EXCEPTIONS

1. Add the scenario given in the file *specification_3.txt* in the *GitHub* folder */cucumber/gettingstarted.*
2. Declare an instance variable: `private boolean errorThrown;`
3. In the methods for the *@When* steps, where an exception might be thrown*:*
    - put a try-catch block around the code.
    - In the catch block you can set the instance variable `errorThrown` to `true`;
4. In the *@Then* steps where you want to check if an exception is thrown, you can check the value of the instance variable `errorThrown` …

DATA TABLES

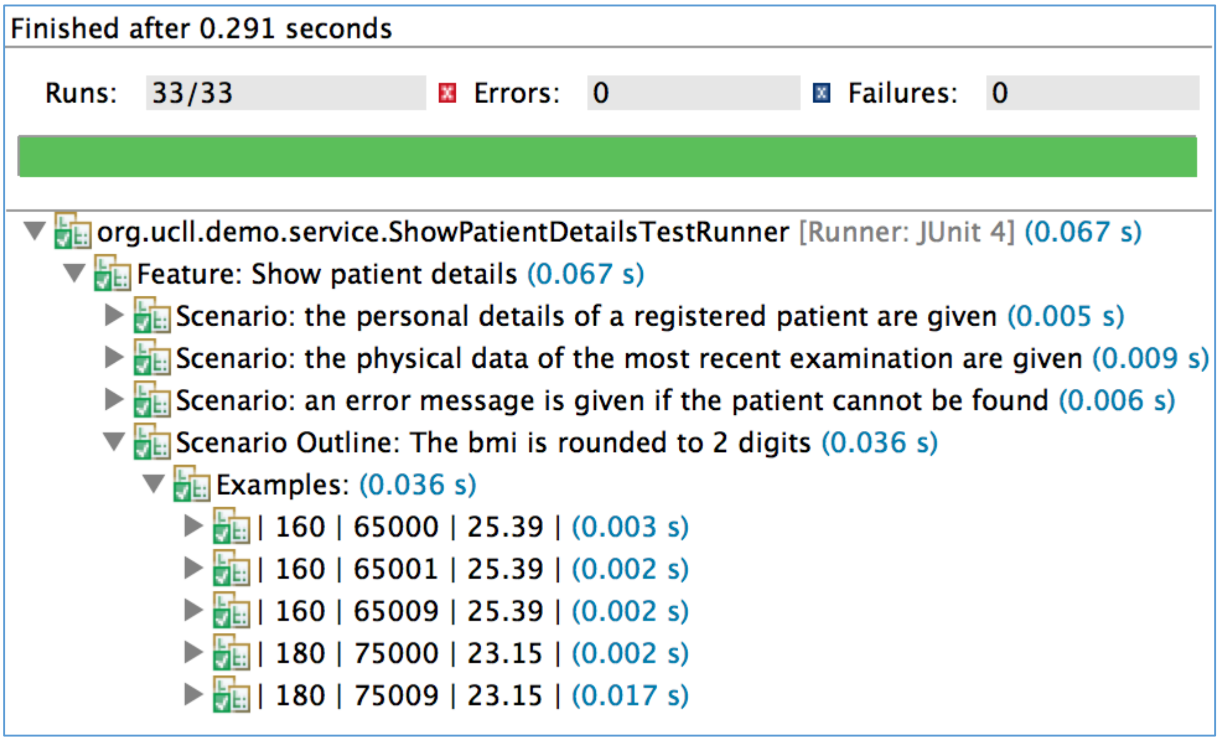
1. Add the scenario given in the file *specification_4.txt* in the *GitHub* folder /cucumber/*gettingstarted*. This scenario is different from the previous ones:
    - It must be a Scenario Outline instead of a Scenario
    - Instead of concrete values, you can see parameter names enclosed with < and >
    - After the scenario, you can see a table of examples. Columns are separated with the '|' character.



```
Scenario Outline: the bmi is rounded to 2 digits
    Given a patient that is registered with a length <length> cm and weight <weight> gr
    When I ask for the details of the patient
    Then the bmi <bmi> is given rounded to two digits

    Examples:
        |length |weight |bmi    |
        |160    |65000  |25.39  |
        |160    |65001  |25.39  |
        |160    |65009  |25.39  |
        |180    |75000  |23.15  |
        |180    |75009  |23.15  |
```

Figure 19. Table with examples

2. Run as JUnit test and copy the new missing steps from the console output to your Steps-file.
3. In these steps, you also use parameters as described in one of the above paragraphs
4. Run again as JUnit test.
5. Now you can see that this Scenario is ran for all the examples defined in the data table.



```
Finished after 0.291 seconds

  Runs:  33/33      ☒ Errors:  0      ☒ Failures:  0

▼ org.ucll.demo.service.ShowPatientDetailsTestRunner [Runner: JUnit 4] (0.067 s)
  ▼ Feature: Show patient details (0.067 s)
    ▶ Scenario: the personal details of a registered patient are given (0.005 s)
    ▶ Scenario: the physical data of the most recent examination are given (0.009 s)
    ▶ Scenario: an error message is given if the patient cannot be found (0.006 s)
    ▼ Scenario Outline: The bmi is rounded to 2 digits (0.036 s)
      ▼ Examples: (0.036 s)
        ▶ | 160 | 65000 | 25.39 | (0.003 s)
        ▶ | 160 | 65001 | 25.39 | (0.002 s)
        ▶ | 160 | 65009 | 25.39 | (0.002 s)
        ▶ | 180 | 75000 | 23.15 | (0.002 s)
        ▶ | 180 | 75009 | 23.15 | (0.017 s)
```

Figure 20. JUnit output for an example table

ANOTHER STORY

Write specifications for the user story *Add physical examination data*.

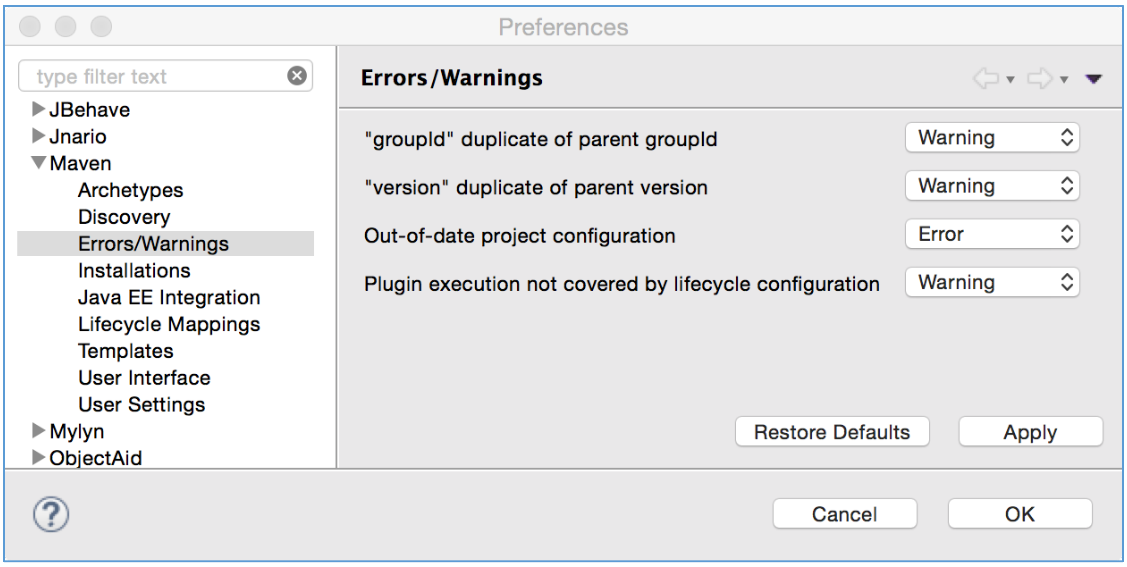
1. How easy is it to focus on the content, without thinking about technical aspects?
2. Investigate other possibilities of *Cucumber*. Look fo features not described in this manual.
   - Can you avoid the naming of the parameters to arg1, arg2, … ?
   - …

In the project *bmi-app-example-solution* can find an example of *Cucumber* used in combination with *Mockito* and *Selenium*.

TROUBLESHOOTING

- If you get an error about missing jars when running your Maven build, this might be caused by a specific configuration in your *Maven settings.xml* file. If this is the case, move your *settings.xml* temporarily to another folder or rename it for the duration of this hands on lab.

- If you get the error message ‘Plugin execution not covered by lifecycle configuration’ in your pom.xml, this might be annoying but it should not cause your build to fail. If you want the message to disappear:
    - Choose *Eclipse* (or *Window*) | *Preferences…*
    - Choose *Maven* > *Errors/Warnings*
    - In the dropdown next to *Plugin execution not covered by lifecycle configuration*, select *Ignore* or *Warning*
    - Choose *OK*
    - Choose *Yes*
    - The error message should have disappeared, and the build should still run as normal.



Errors/Warnings

"groupId" duplicate of parent groupId: Warning

"version" duplicate of parent version: Warning

Out-of-date project configuration: Error

Plugin execution not covered by lifecycle configuration: Warning

ATTACHMENTS

CREATE A PROJECT FOR CUCUMBER WITHOUT USING MAVEN

1. Create a new workspace: in the menu choose *File* | *Switch Workspace* | *Other …*
2. Create a new project: *File* | *New* | *Dynamic Web Project*
   - Choose as name for your project *bmi-cucumber*
   - Click *Finish*
3. Add the dependencies needed for *Cucumber*:
   - Right click on project
   - Choose *Build Path | Configure Build Path…*
   - Choose the tab *Libraries*
   - Click *Add External JARS…* and browse to the folder *noMaven/cucumber/dependencies* on the USB stick
   - Select all the jars and choose *Open*
   - Click *OK*
4. Modify the structure of the source folder:
   - Click to expand the folder *Java Resources*
   - Right click on the `src` folder and choose *Delete*
   - Click on *Java Resources* again
   - Right click *New | Source Folder*, choose your Project name and as Folder name `src/main/java.` In the same way, add the folders:
     - `src/main/resources`
     - `src/test/java`
     - `src/test/resources`
   - Right click on project and choose *Refresh*
5. Add the code for the example application to your project:
   - Copy the files from the folder *noMaven/bmi-app* into the corresponding folders made in the previous step
   - Also copy the *webapp* folder in your `src/main` folder on your file system
   - Right click on project and choose *Refresh*

Now you can start following the *Cucumber GettingStarted* guide from the paragraph *Development* onwards…