FROM TDD TO BDD AND BACK
HANDS ON LAB

GETTING STARTED GUIDE CONCORDION

Mieke Kemme and Elke Steegmans
May 17, 2016

TABLE OF CONTENTS

INTRODUCTION

CONCORDION

Concordion is an open source tool for automating Specification by Example.

Official Website: http://concordion.org/

THIS LAB

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

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 *Concordion.*

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 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 `Concordion` as *Name* and http://concordion-eclipse.googlecode.com/svn/trunk/org.concordion.ide.eclipse.update-site-helios as *Location*.
- Choose *OK*
- Select *Concordion 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 *Concordion* from the GitHub repository. You find it in the folder *concordion/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 *Concordion*, you always have to write 2 types of file:

1. The **feature**: an html file containing the specifications. The filename is in lowercase and the extension is .html.

   Example: `showPatientDetails.html`

2. A file containing the **fixture** and the **story steps**: a Java test class that is used to run as a *JUnit* test class and that contains the java code to make the specifications executable. The filename is the same name as the name of the story + Test, but in CamelCase and with extension .java

   Example: `ShowPatientDetailsTest.java`

In the next steps you will be guided through a first example using *Concordion*. 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 */concordion/gettingstarted*, in the files `ShowPatientDetails.html` and `ShowPatientDetailsTest.java`.

STEP 1: WRITE SPECIFICATIONS

1. Rename the source folder `src/test/resources` to `src/test/specs`.
2. Create a package `org.ucll.demo.service` in this new folder.
3. Create a new html file in this package and in this specs folder:
   - Choose *New* | *Other…* | *Web* | *HTML File*
   - Click *Next*
   - Enter the name of the feature, i.e. `ShowPatientDetails.html`
   - Click *Finish*
   - The HTML file is created.
4. Replace the code with the story and the scenario given in the file *specification_1.tx*t in the GitHub folder */concordion/gettingstarted*
   - Open the *specification_1.txt* file with an application in which you can see HTML code, such as for example Brackets
   - A css file is already made by *Concordion* for you, you don't need to make it yourself, you just add it with the necessary HTML tags.
5. Choose *Save*.

```
<h1>Show patient details</h1>
<h2>Narrative</h2>
<div>
<p>In order to check the physical condition of a patient</p>
<p>As a caretaker</p>
<p>I want to consult his/her personal details</p>
</div>
<h2>Acceptance Criteria</h2>
<h3>Scenario: the personal details of a registered patient are given</h3>
<div>
    <p>
        Given a patient with the social security number 93051822361, gender male and birthdate 1993-05-18
    </p>
    <p>
        And on 2000-04-10 the patient had a length of 180 cm and a weight of 75000 gr
    </p>
    <p>
        And the patient is registered
    </p>
    <p>
        When I ask for the details of the patient using his social security number
    </p>
    <p>
        Then the personal details social security number, gender and birthdate are given
    </p>
    <p>
        And the examination details length, weight and last examination date are given
    </p>
    <p>
        And the calculated bmi 23.15 is given
    </p>
   <hr></hr>
</div>
```

Figure 1. Example HTML show patient details

STEP 2: CREATE JAVA FIXTURE AND STEPS

1. Create in the folder `src/test/java` and in the package `org.ucll.demo.service` a Java class for the fixture, i.e. `ShowPatientDetailsTest.java`.

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

   ```
   @RunWith(ConcordionRunner.class)
   ```

   and add the import statements:

   ```
   import org.concordion.integration.junit4.ConcordionRunner;
   import org.junit.runner.RunWith;
   ```

EXECUTE SPECIFICATIONS

1. Select the Java file `ShowPatientDetailsTest.java`, right mouse click and choose *Run as JUnit Test.*

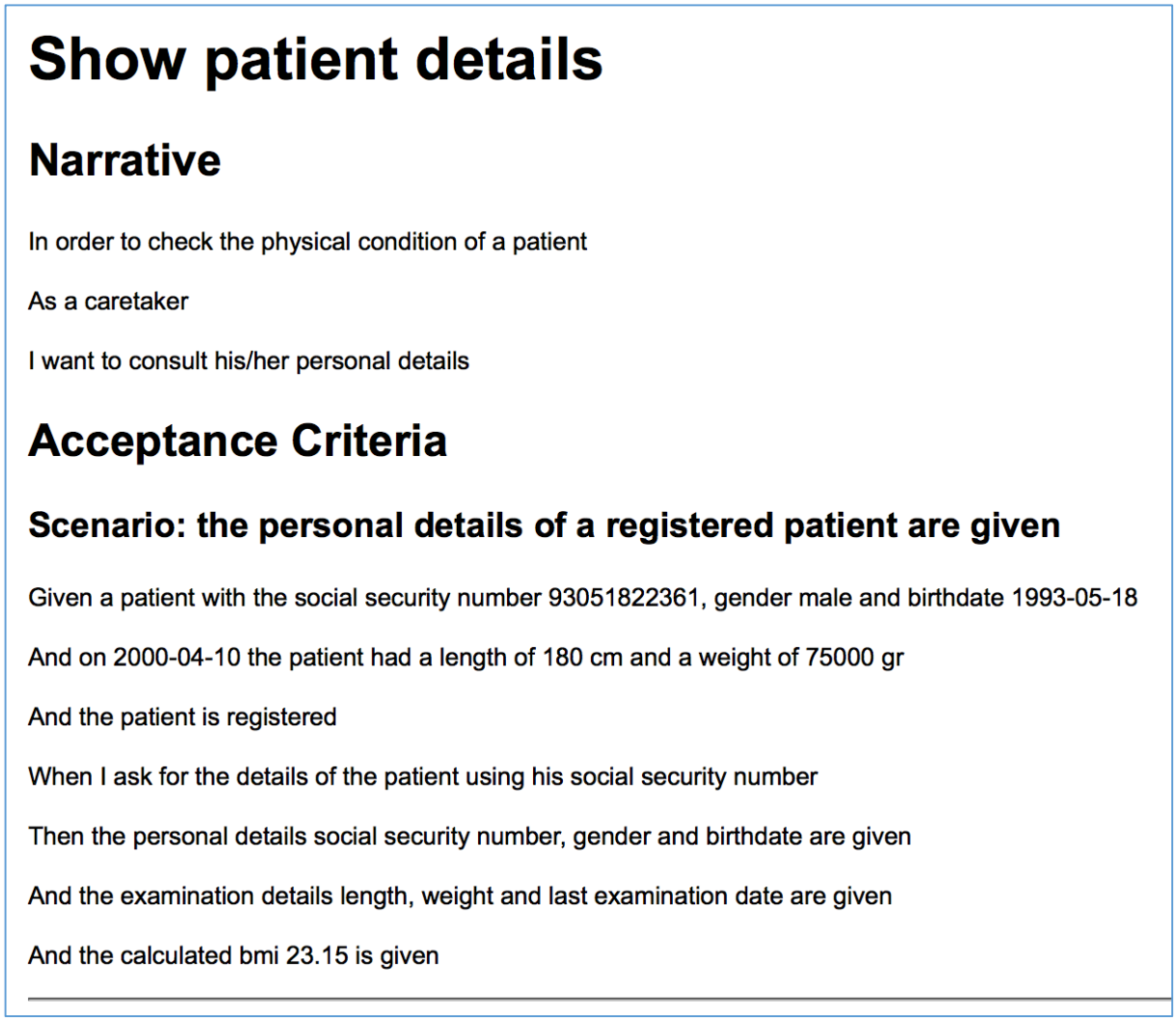
   In the **output console** you can see the result:

   - The path to the html file in which the results are shown
   - The number of successes and the number of failures

```
/var/folders/qb/kyvsxvcj3vd3bzwx9gbz8fp00000gn/T/concordion/org/ucll/demo/service/ShowPatientDetails.html
Successes: 0, Failures: 0
```

Figure 2. Example console output

2. Copy the link to the html file to your browser to open **HTML output**.
   Once the tests are implemented, colors will indicate the results.



Show patient details

Narrative

In order to check the physical condition of a patient

As a caretaker

I want to consult his/her personal details

Acceptance Criteria

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 3. Example HTML output

STEP 4: MAKE THE STEPS EXECUTABLE

INTRODUCTION

In the html file you need to add specific *Concordion* tags to the existing html tags.

There exist different types of Concordion specific tags, they all start with "`concordion:*`" where the asterisk needs to be replaced by a command. You can add the Concordion specific tags to a lot of existing HTML tags, such as for example <label>, <p>, … An example of each Concordion specific tag:

- `<label concordion:set="#number>93051822361</label>`
  This created a temporary variable **number** with the value **93051822361**

- `<p concordion:execute="setSocialSecurityNumber(#number)"></p>`
  This calls and executes the method **setSocialSecurityNumber** with the temporary variable created before as a parameter.
  In this case, the "setSocialSecurityNumber" is a *Java* method on **story steps**, a .java class that represents this *.html* file, such as

  ```
  public void setSocialSecurityNumber(String number){
  …
  }
  ```

  You can also call a method which returns something and put the return value in a variable. For example:

  ```
  <p concordion:execute="#validNumber =
  checkSocialSecurityNumber(#number)"></p>
  ```

- ```
  <label concordion:assertTrue="validSocialSecurityNumber(#number)">
      true
  </label>
  ```
  This calls the **checkSocialSecurityNumber** method with the temporary variable *#number*. It then checks/asserts if the result is **true**
  In this case, the "checkSocialSecurityNumber" is a *Java* method on **story steps**, a .java class that represents this *.html* file, such as

  ```
  public boolean checkSocialSecurityNumber(String number){
  …
  }
  ```

  The label with the assert will display a green or red background depending on the result of the test. Also `assertFalse`, `assertEquals`, … can be used here. For example:
  `<label concordion:assertEquals="#number">93051822361</label>`

Remark: in eclipse you can type `<label con` and then do *ctrl-space* to get an overview of all the existing *Concordion* specific tags.
  ```

CODE

To make the specifications executable:

1. Add *Concordion* specific tags on the right places.

```
<h1>Show patient details</h1>
<h2>Narrative</h2>
<div>
<p>In order to check the physical condition of a patient</p>
<p>As a caretaker</p>
<p>I want to consult his/her personal details</p>
</div>
<h2>Acceptance Criteria</h2>
<h3>Scenario: the personal details of a registered patient are given</h3>
<div>
    <p concordion:execute="initializePersonDetails(#socialSecurityNumber, #gender, #birthDate)">
        Given a patient with the social security number <label concordion:set="#socialSecurityNumber">93051822361</label>,
        gender <label concordion:set="#gender">male</label> and birthdate <label concordion:set="#birthDate">1993-05-18</label>
    </p>
    <p>
        And on 2000-04-10 the patient had a length of 180 cm and a weight of 75000 gr
    </p>
    <p>
        And the patient is registered
    </p>
    <p>
        When I ask for the details of the patient using his social security number
    </p>
    <p>
        Then the personal details social security number, gender and birthdate are given
    </p>
    <p>
        And the examination details length, weight and last examination date are given
    </p>
    <p>
        And the calculated bmi 23.15 is given
    </p>
   <hr></hr>
</div>
```

Figure 4. Example adding Concordion specfic tags

2. Implement each method called in the *Concordion* specific tags in the .java file.
    - call the necessary code to connect to your actual domain classes…

```
package org.ucll.demo.service;

import java.text.SimpleDateFormat;
import java.util.Date;

import org.concordion.integration.junit4.ConcordionRunner;
import org.junit.runner.RunWith;
import org.ucll.demo.domain.Gender;
import org.ucll.demo.service.api.java.to.PersonDetail;

@RunWith(ConcordionRunner.class)
public class ShowPatientDetailsTest {

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

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

    public void initializePersonDetails (String socialSecurityNumber, String gender, String birthDate) throws Exception {
        this.socialSecurityNumber = socialSecurityNumber;
        this.gender = Gender.valueOf(gender.toUpperCase());
        this.birthDate = DATE_FORMATTER.parse(birthDate);
        detailsRetrieved = new PersonDetail(this.socialSecurityNumber, this.gender, this.birthDate);
    }

}
```

Figure 5. Example step implementation

3. Implement all the steps until you have a successful test. As you can see in the screenshot in Figure 6.

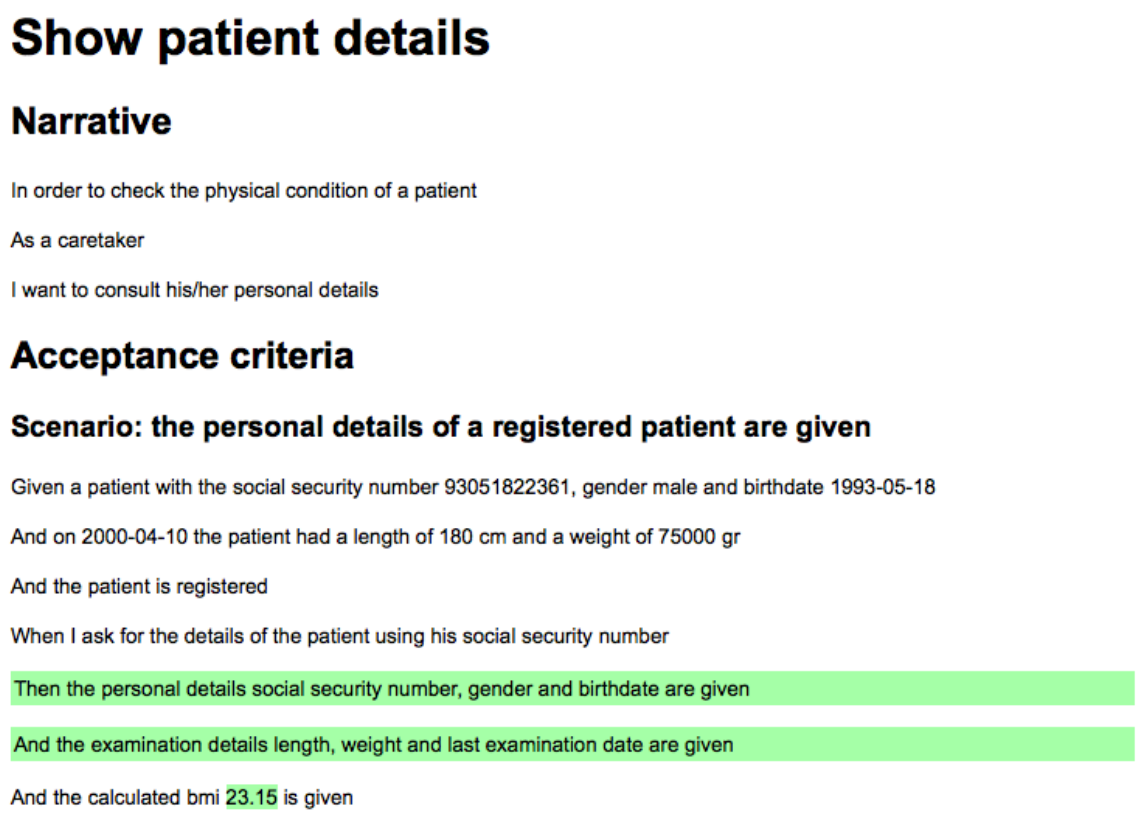


Figure 6. Example HTML output with successfully running of the test.

The solution of this first example can be found in the GitHub repository in the folder */concordion/gettingstarted*, in the files `ShowPatientDetails.html` and `ShowPatientDetailsTest.java`.

EXTRA

REUSE STEPS

Identical steps:

1. Add the scenarios given in the file *specification_2.txt* in the GitHub folder */concordion/specifications/specification2_txt*.
2. Run the test.
3. In *Concordion* you need to add *Concordion* specific tags to each step again, and you can reuse the same method in these different steps, you don't need to make different methods. In the example in Figure 7 the step 'And the patient is registered' is implemented by the same method.

```
        a weight of <label concordion:set="#weight">75000</label> gr
    </p>
    <p concordion:execute="registerPatient()">
        And the patient is registered
    </p>
    <p concordion:execute="askPatientDetails()">
        When I ask for the details of the patient using his social security number
    </p>
    <p concordion:assertTrue="verifyPatientDetails()">
        Then the personal details social security number, gender and birthdate are given
    </p>
    <p concordion:assertTrue="verifyExaminationDetails()">
        And the examination details length, weight and last examination date are given
    </p>
    <p>
        And the calculated bmi <label concordion:assertEquals="verifyBmi()">23.15</label> is given
    </p>
   <hr></hr>
</div>
<h3>Scenario: the physical data of the most recent examination are given</h3>
<div>
    <p concordion:execute="initializePersonDetails(#socialSecurityNumber)">
        Given a patient with the social security number <label concordion:set="#socialSecurityNumber">93051822361</label>
    </p>
    <p concordion:execute="initializeExaminationDetails(#examinationDate, #length, #weight)">
        And on <label concordion:set="#examinationDate">2000-04-10</label> the patient had
        a length of <label concordion:set="#length">180</label> cm and
        a weight of <label concordion:set="#weight">75000</label> gr
    </p>
    <p concordion:execute="registerPatient()">
        And the patient is registered
    </p>
    <p concordion:execute="addOlderExaminationDetails(#examinationDate, #length, #weight)">
        And on <label concordion:set="#examinationDate">2000-04-17</label> the patient had
        a length of <label concordion:set="#length">180</label> cm and
        a weight of <label concordion:set="#weight">80000</label> gr
```

Figure 7. Calling the same method in different scenarios

CLEAN

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.

That is why we need a `clean()` method which we call in the `registerPatient()` method.

```
public void registerPatient () {
    clean();

    service.addPerson(patient);
}

private void clean() {
    //TODO replace with DBUnit, ...
    service.deletePerson(REGISTERED_SOCIAL_SECURITY_NUMBER);
}
```

Figure 8. Calling a `clean()` method in the `registerPatient()` method to clean up the patient created in the previous scenario.

SCENARIOS WITH EXPECTED EXCEPTIONS

1. Add the scenario given in the file *specification_3.txt* in the *GitHub* folder */concordion/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…
5. In the then step in the html page you can use than in the right html tags of the *then* steps *concordion assertTrue="exceptionThrown()"*
6. The result of the execution of this step is colored now: red if it has failed and green if it has succeeded.

Scenario: an error message is given if the patient cannot be found

Given a patient that is not registered

When I ask for the details of the patient using his social security number

Then an error message is given

And no details are given

Figure 9. HTML output of a successful assertTrue

DATA TABLES

1. Add the scenario given in the file *specification_4.txt* in the *GitHub* folder */concordion/gettingstarted.* This scenario is different from the previous ones:
    - Instead of concrete values, you can see a table with parameters and concrete values

```
<h3>Scenario Outline: the bmi is rounded to 2 digits</h3>
<div>
    <p>Given a patient that is registered with a length in cm and weight in gr</p>
    <p>When I ask for details of the patient</p>
    <p>Then the bmi given is rounded to 2 digits</p>
    <div class="example">
        <h3>Examples</h3>
        <table concordion:execute="registerPatient(#length, #weight)">
            <tr>
                <th concordion:set="#length">length</th>
                <th concordion:set="#weight">weight</th>
                <th concordion:assertEquals="askPatientDetailsAndVerifyBmi()">bmi</th>
            </tr>
            <tr>
                <td>160</td>
                <td>65000</td>
                <td>25.39</td>
            </tr>
            <tr>
                <td>160</td>
                <td>65001</td>
                <td>25.39</td>
            </tr>
            <tr>
                <td>160</td>
                <td>65009</td>
                <td>25.39</td>
            </tr>
            <tr>
                <td>180</td>
                <td>75000</td>
                <td>23.15</td>
            </tr>
            <tr>
                <td>180</td>
                <td>75000</td>
                <td>23.15</td>
```

Figure 10. Table with examples

2. Implement all steps and all methods used and run as JUnit test.

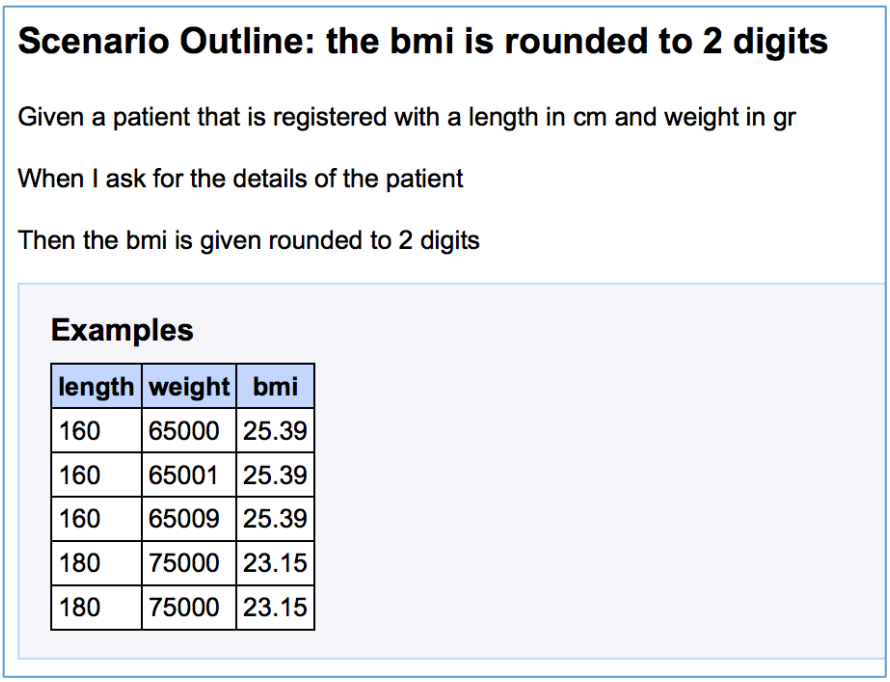


Figure 11. HTML output for an example table

ANOTHER STORY

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

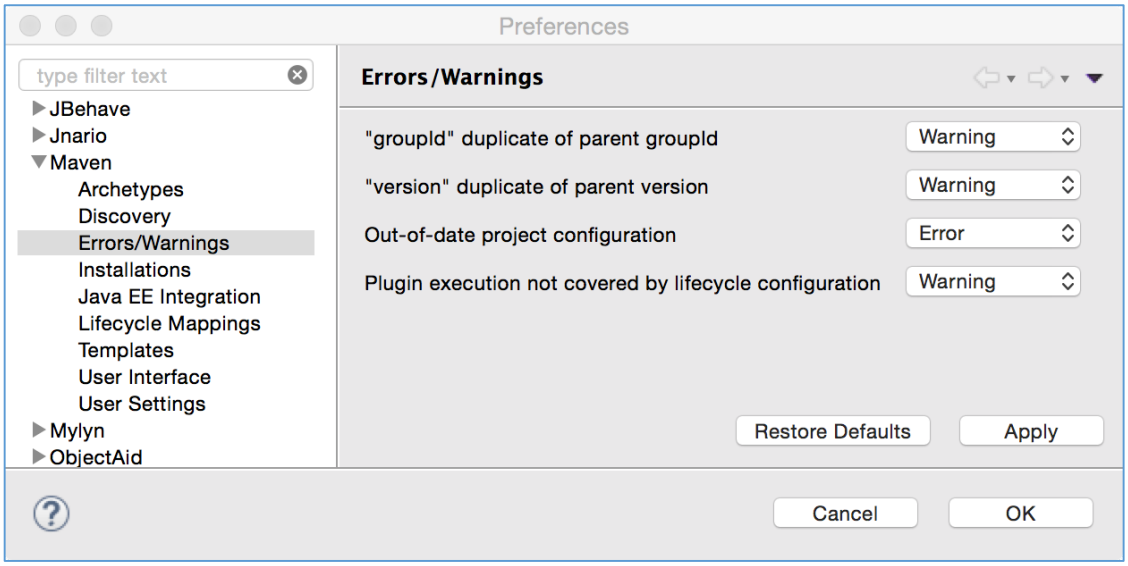
1. How easy is it to focus on the content, without thinking about technical aspects?
2. Investigate other possibilities of *Concordion*. Look at features not described in this manual.
    - How can you include images in your HTML pages?
    - …

In the project *bmi-app-example-solution* can find an example of *Concordion* 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.



ATTACHMENTS

CREATE A PROJECT FOR CONCORDION WITHOUT USING MAVEN

4. Create a new workspace: in the menu choose *File* | *Switch Workspace* | *Other …*
5. Create a new project: *File* | *New* | *Dynamic Web Project*
    - Choose as name for your project *bmi-concordion*
    - Click *Finish*
6. Add the dependencies needed for *Concordion*:
    - Right click on project
    - Choose *Build Path | Configure Build Path…*
    - Choose the tab *Libraries*
    - Click *Add External JARS…* and browse to the folder *noMaven/concordion/dependencies* on the USB stick
    - Select all the jars and choose *Open*
    - Click *OK*
7. 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*
8. 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 *Concordion GettingStarted* guide from the paragraph *Development* onwards…