FROM TDD TO BDD AND BACK HANDS ON LAB

GETTING STARTED GUIDE SCALATEST

Mieke Kemme and Elke Steegmans
May 17, 2016

TABLE OF CONTENTS

TABLE OF CONTENTS **2**

INTRODUCTION **3**

ScalaTest 3

This Lab 3

Material 3

Context 3

SETUP **4**

Prerequisites 4

Installation Eclipse Plugin 4

Create Eclipse project 4

DEVELOPMENT **5**

Introduction: the General idea 5

Step 1: Write specifications 5

Execute specifications 6

Step 2: Make the steps executable 7

Consult the HTML report 8

EXTRA **8**

Before and After 8

Reuse fixture 9

Scenarios with expected Exceptions 11

Data tables 11

ANOTHER STORY **12**

TROUBLESHOOTING **13**

ATTACHMENTS **15**

Create a project for ScalaTest without using Maven 15

INTRODUCTION

SCALATEST

ScalaTest is a free, open-source testing toolkit for Scala and Java programmers.

Official Website: http://www.scalatest.org/

THIS LAB

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

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 *ScalaTest.*

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 `ScalaTest` as *Name* and http://download.scala-ide.org/sdk/lithium/e44/scala211/stable/site as *Location*.
- Choose *OK*
- Select *Scala IDE for Eclipse* and *ScalaTest for Scala IDE* 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 …* Make sure the folder you choose as workspace does not contain any spaces!
2. Get the bmi-app for *ScalaTest* from the *GitHub* repository. You find it in the folder */scalatest/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*

    If you get a window *Setup Maven Plugin Connectors* with error messages, click *Finish* and ignore the warning.

    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.

    You can ignore the error message *Plugin execution not covered by lifecycle configuration*. If it bothers you, see the section *Troubleshooting* in this document.
4. Make it a Scala project:
    - Create a source folder for Scala: *File* | *New* | *Source Folder – Folder Name:* `src/test/scala`
    - Right click on project | *Configure* | *Add Scala nature*

DEVELOPMENT

INTRODUCTION: THE GENERAL IDEA

To write executable specifications with *ScalaTest*, you put everything in one Scala Class per story.

Example: `ShowPatientDetails`

1. **Headers** with the feature, story, steps, …
2. After each step, **Scala code** to make the step executable

In the next steps you will be guided through a first example using *ScalaTest*. 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 */scalatest/gettingstarted*, in the file *ShowPatientDetails.scala*.

STEP 1: WRITE SPECIFICATIONS

1. Create a package `org.ucll.demo.service` in the test folder *src/test/scala*.
2. Create a new Scala Class in this package:
   - Choose *New* | *Other…* | *Scala Wizards* | *Scala Class*
   - Click *Next*
   - Enter the name of the story, i.e. `org.ucll.demo.service.ShowPatientDetails`
   - Click *Finish*

   The story is created.
3. Open the file: right click on the file | *Open with* | *Scala Editor*.
4. Add the following after the name of the class:

   ```
   extends FeatureSpec with GivenWhenThen
   ```

   and add the import statements:

   ```
   import org.scalatest.FeatureSpec
   import org.scalatest.GivenWhenThen
   ```

5. Add the story and the scenario given in the file *specification_1.txt* in the *GitHub* folder */scalatest/gettingstarted*.
   - In the body of the class, you can find the **feature**
   - In the body of the feature, you can find the **story**
   - After the story, you can find the **scenario**
   - In the body of the scenario, you can find the **steps**
   - Add the end, we added **`pending`**, as there is no code to execute yet

```
package org.ucll.demo.service.api.java

import org.scalatest.FeatureSpec
import org.scalatest.GivenWhenThen

class ShowPatientDetails extends FeatureSpec with GivenWhenThen {
  feature("Show patient details") {
    info("In order to check the physical condition of a patient")
    info("As a caretaker")
    info("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")
      pending
    })
  }
}
```

Figure 1. Example pending story

EXECUTE SPECIFICATIONS

Select the Scala class, right mouse click and choose *Run as ScalaTest - Test.*

In the *ScalaTest* view you can see the test result:

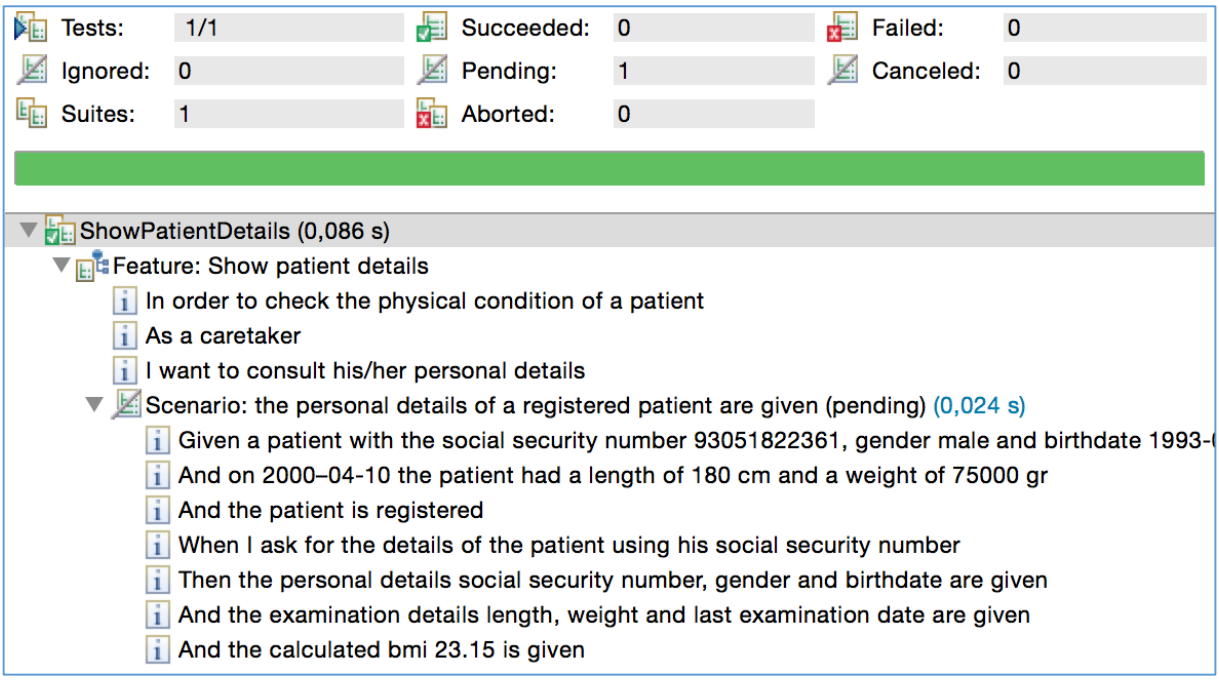


Figure 2. Example test output

In the **output console** you can see the textual representation:

```
<terminated> Feature- Show patient details Scenario- the personal details of a registered patient are given [ScalaTest] /Library/Java/JavaVirtu
Run starting. Expected test count is: 1
ShowPatientDetails:
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 (pending)
    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
Run completed in 247 milliseconds.
Total number of tests run: 0
Suites: completed 1, aborted 0
Tests: succeeded 0, failed 0, canceled 0, ignored 0, pending 1
No tests were executed.
```

Figure 3. Example console output

STEP 2: MAKE THE STEPS EXECUTABLE

After each step, write the Scala code to connect to your actual domain classes…
Remove the `pending` statement.

```
package org.ucll.demo.service.api.java

import org.scalatest.FeatureSpec
import org.scalatest.GivenWhenThen
import java.text.SimpleDateFormat
import org.ucll.demo.domain.Gender
import org.ucll.demo.service.api.java.to.PersonDetail

class ShowPatientDetails extends FeatureSpec with GivenWhenThen {
  val service = new PersonServiceJavaApi
  val dateFormatter= new SimpleDateFormat("yyyy-MM-dd")

  feature("Show patient details") {
    info("In order to check the physical condition of a patient")
    info("As a caretaker")
    info("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")
      val socialSecurityNumber = "93051822361"
      val gender = Gender.MALE
      val birthDate = dateFormatter.parse("1993-04-18")
      val patient = new PersonDetail(socialSecurityNumber, gender, birthDate)

      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 4. Example implementation of the first step

The solution of this first example can be found in the *GitHub* repository in the folder */scalatest/gettingstarted*, in the file *ShowPatientDetails.scala*.

CONSULT THE HTML REPORT

Run the Maven build:

- 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/htmldir*, you can find website-pages created by *ScalaTest*.
- Open the file `index.html` in your browser. Try to navigate to the html report of your story.

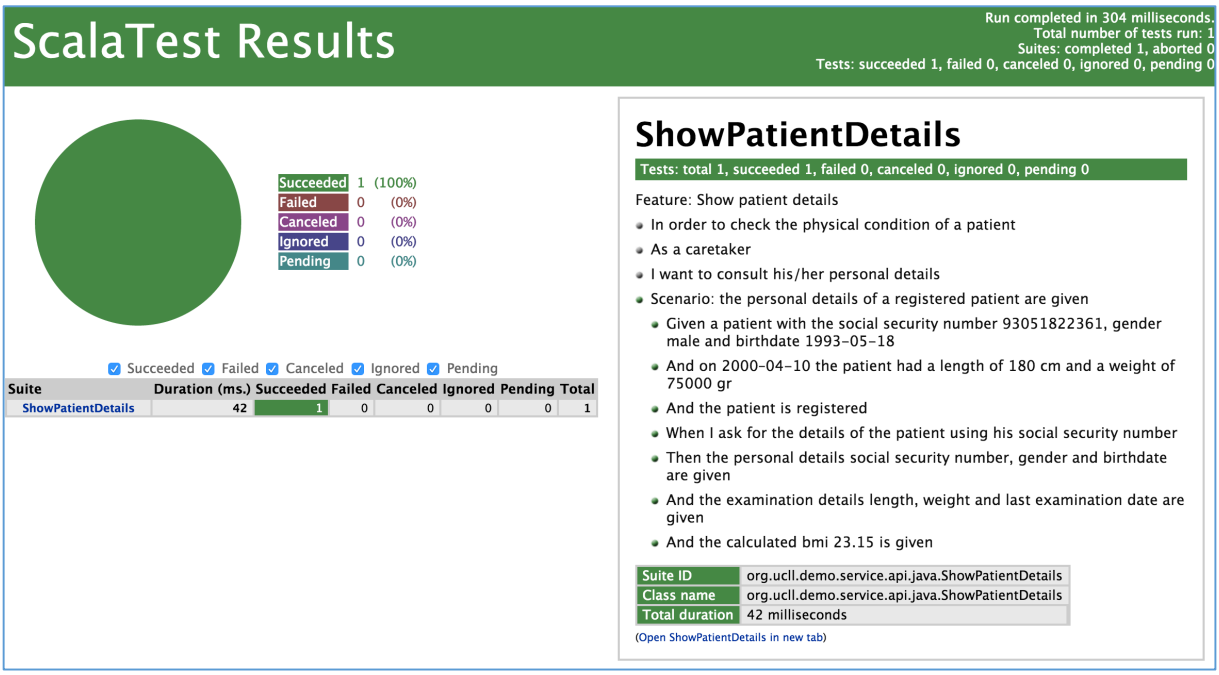


Figure 5. Example HTML report

EXTRA

BEFORE AND AFTER

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

In *ScalaTest*, you can use the trait `BeforeAndAfter` to execute code before or after each scenario:

1. Add the import statement
   `import org.scalatest.BeforeAndAfter`
2. Add the following to your class declaration:
   `… with BeforeAndAfter`

```
class ShowPatientDetails extends FeatureSpec with GivenWhenThen with BeforeAndAfter {
```

Figure 6.Use the BeforeAndAfter trait

3. Use the `after` method to clean delete the patient after the execution of the scenario:

```
class ShowPatientDetails extends FeatureSpec with GivenWhenThen with BeforeAndAfter {
  val service = new PersonServiceJavaApi
  val dateFormatter= new SimpleDateFormat("yyyy-MM-dd")

  after {
    service.deletePerson("93051822361")
  }

  feature("Show patient details") {
    info("In order to check the physical condition of a patient")
    info("As a caretaker")
    info("I want to consult his/her personal details")
```

Figure 7. Use after method to clean after each scenario

In the same way you can use the `before` method to execute any code before each scenario.

REUSE FIXTURE

1. Add the scenarios given in the file *specification_2.txt* in the *GitHub* folder */scalatest/gettingstarted*.
2. You can see that you get code duplication: the *Given* steps of both scenarios, for instance, are identical. If you need the same mutable fixture objects in multiple tests, *ScalaTest* allows you to use *get-fixture* methods. A *get-fixture* method returns a new instance of a needed fixture object each time it is called.
    - Write a *get-fixture* method
    - Move the code to create a patient object to this method
    - In the *Given* step, call the *get-fixture* method and store the result in a local variable
    - Refactor the other steps: get the data you need from the fixture object.

```
class ShowPatientDetails extends FeatureSpec with GivenWhenThen {
  val service = new PersonServiceJavaApi
  val dateFormatter= new SimpleDateFormat("yyyy-MM-dd")

  def fixture() = new {
    val socialSecurityNumber = "93051822361"
    val gender = Gender.MALE
    val birthDate = dateFormatter.parse("1993-05-18")

    val patient = new PersonDetail(socialSecurityNumber, gender, birthDate)
  }

  feature("Show patient details") {
    info("In order to check the physical condition of a patient")
    info("As a caretaker")
    info("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")
      val testData = fixture();

      And("on 2000-04-10 the patient had a length of 180 cm and a weight of 75000 gr")
      val examinationDate = dateFormatter.parse("2000-04-10")
      val examination = new ExaminationDetail(180, 75000, examinationDate)
      testData.patient.setExaminationDetail(examination)

      And("the patient is registered")
      service.addPerson(testData.patient)
```

Figure 8. get-fixture method

3. Run the test to check everything still works.

The second step of both scenarios is almost the same: only the actual data used to create the examination object differ. As you can use parameters in a *get-fixture* method, this should not be a problem.

```
class ShowPatientDetails extends FeatureSpec with GivenWhenThen with BeforeAndAfter {
  val service = new PersonServiceJavaApi;
  val dateFormatter = new SimpleDateFormat("yyyy-MM-dd");

  def fixture(length: Int, weight: Int, date: String) =
    new {
      val socialSecurityNumber = "93051822361";
      val gender = Gender.MALE;
      val birthDate = dateFormatter.parse("1993-04-18");
      val patient = new PersonDetail(socialSecurityNumber, gender, birthDate);

      val examinationDate = dateFormatter.parse(date);
      val examination = new ExaminationDetail(length, weight, examinationDate);
      patient.setExaminationDetail(examination)
    };

  feature("Show patient details") {
    info("In order to check the physical condition of a patient");
    info("As a caretaker");
    info("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");
      val testData = fixture(180, 75000, "2000-04-10");

      And("the patient is registered");
      service.addPerson(testData.patient);

      When("I ask for the details of the patient using his social security number");
      val detailsRetrieved = service.getPerson(testData.socialSecurityNumber);

      Then("the personal details social security number, gender and birthdate are given");
```

Figure 9.get-fixture method with parameters

4. Implement the other scenario, using the fixture.

Of course, you can avoid duplicate methods by writing 'plain' methods...

```
def differNoMoreThanFewSeconds(date: Date, otherDate: Date): Boolean = {
  return date.compareTo(date) <= 0 && date.compareTo(otherDate) >= -2;
}
```

Figure 10. Example method written in Scala

SCENARIOS WITH EXPECTED EXCEPTIONS

1. Add the scenario given in the file *specification_3.txt* in the *GitHub* folder */scalatest/gettingstarted*.
2. To check if an exception is thrown, do not ask for the patient's details in the *When* step. Instead, leave the When step empty, and try to ask the details in the *Then* step. To check if an exception is thrown, surround the code with: `intercept[IllegalArgumentException] { }`

```
scenario("an error message is given if the patient cannot be found"){
  Given(" a patient that is not registered")
  val testData = fixture(180, 75000, "2000-04-10")

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

  Then("an error message is given")
  var detailsRetrieved: PersonDetail = null
  intercept[IllegalArgumentException] {
    detailsRetrieved = service.getPerson(testData.socialSecurityNumber)
  }
  And("no details are given")
  assert(detailsRetrieved == null);
})
```

Figure 11. Intercept expected exception

DATA TABLES

1. Add the scenario in the file *specification_4.txt* in folder the *GitHub* folder */scalatest/gettingstarted*. This scenario is different from the previous ones:
   - It has no concrete values in the *Given When Then* steps
   - After the scenario, you can see a table of examples. We want to run the scenario for all the examples.

```
scenario("the bmi is rounded to 2 digits"){
  val examples = Table(
    ("length", "weight", "date", "bmi"),
    (160, 65000, "2000-04-10", 25.39),
    (160, 65001, "2000-04-11", 25.39),
    (160, 65009, "2000-04-12", 25.39),
    (180, 75000, "2000-04-13", 23.15),
    (180, 75009, "2000-04-14", 23.15))

      Given("the patient has a length " + length + " cm and weight " + weight + " gr on " + date)
      When("I ask for the details of the patient")
      Then("the bmi " + bmi + " is given rounded to 2 digits")
})
```

Figure 12. Table with examples

2. Add the import statement
   `import org.scalatest.prop.TableDrivenPropertyChecks._`
3. Remark: the before and after methods are only executed before and after the execution of the **entire** scenario! As we want to make sure that the patient is only create once, we will register the patient **before** the *Given* step:

```
scenario("the bmi is rounded to 2 digits"){
  val examples = Table(
    ("length", "weight", "date", "bmi"),
    (160, 65000, "2000-04-10", 25.39),
    (160, 65001, "2000-04-11", 25.39),
    (160, 65009, "2000-04-12", 25.39),
    (180, 75000, "2000-04-13", 23.15),
    (180, 75009, "2000-04-14", 23.15))

  val testData = fixture(180, 75000, "2000-04-09")
  service.addPerson(testData.patient)

      Given("the patient has a length " + length + " cm and weight " + weight + " gr on " + date)
      When("I ask for the details of the patient")
      Then("the bmi " + bmi + " is given rounded to 2 digits")
})
```

Figure 13. Create the fixture in the beginning

4. Iterate over the table to test all the examples:

```
scenario("the bmi is rounded to 2 digits"){
  val examples = Table(
    ("length", "weight", "date", "bmi"),
    (160, 65000, "2000-04-10", 25.39),
    (160, 65001, "2000-04-11", 25.39),
    (160, 65009, "2000-04-12", 25.39),
    (180, 75000, "2000-04-13", 23.15),
    (180, 75009, "2000-04-14", 23.15))

  val testData = fixture(180, 75000, "2000-04-09")
  service.addPerson(testData.patient)

  forAll(examples) { (length: Int, weight: Int, date: String, bmi: Double) =>
    whenever(length != 0) {

      Given("the patient has a length " + length + " cm and weight " + weight + " gr on " + date)
      val examinationDate = dateFormatter.parse(date)
      service.addExamination(new ExaminationDetail(length, weight, examinationDate), testData.socialSecurityNumber)

      When("I ask for the details of the patient")
      val detailsRetrieved = service.getPerson(testData.socialSecurityNumber)

      Then("the bmi " + bmi + " is given rounded to 2 digits")
      assert(bmi == detailsRetrieved.getBmi);

    }
  }
})
```

Figure 14. Test data table

See the *GitHub* folder */scalatest/ bmi-app-example-solution* for an example implementation of all scenarios

ANOTHER STORY

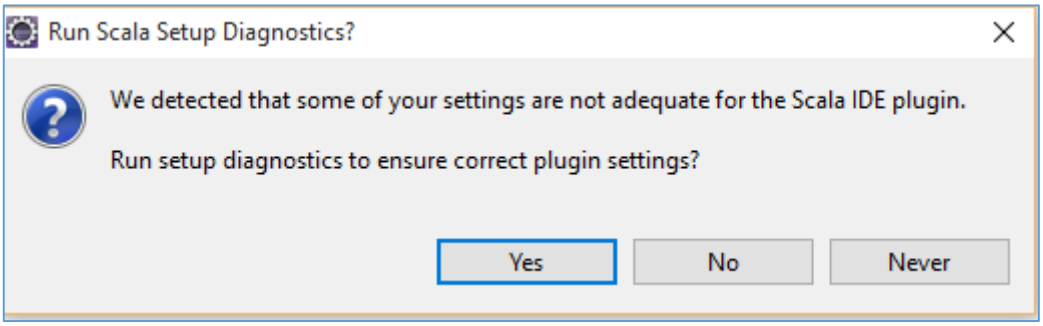
Write specifications for the user story *Add physical examination data*.

1. How easy is it to focus on the content, without thinking about technical aspects?
2. Investigate other possibilities of *ScalaTest*. Look at features not described in this manual.
    - Can you avoid duplicate methods using a setup or teardown method?
    - How does *Property-based testing* work?
    - …

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

TROUBLESHOOTING

- If you get the message about your settings, click *Yes* and choose the recommended settings.



- If the scalatest-maven-plugin complains it cannot find …`/htmldir`, check if you gave any spaces in the path of your project. If so, move your project to a location without spaces in the path.

- If Eclipse is very slow: replace following settings in the *eclipse.ini* file:

```
-Xms40m
-Xmx512m
```

by

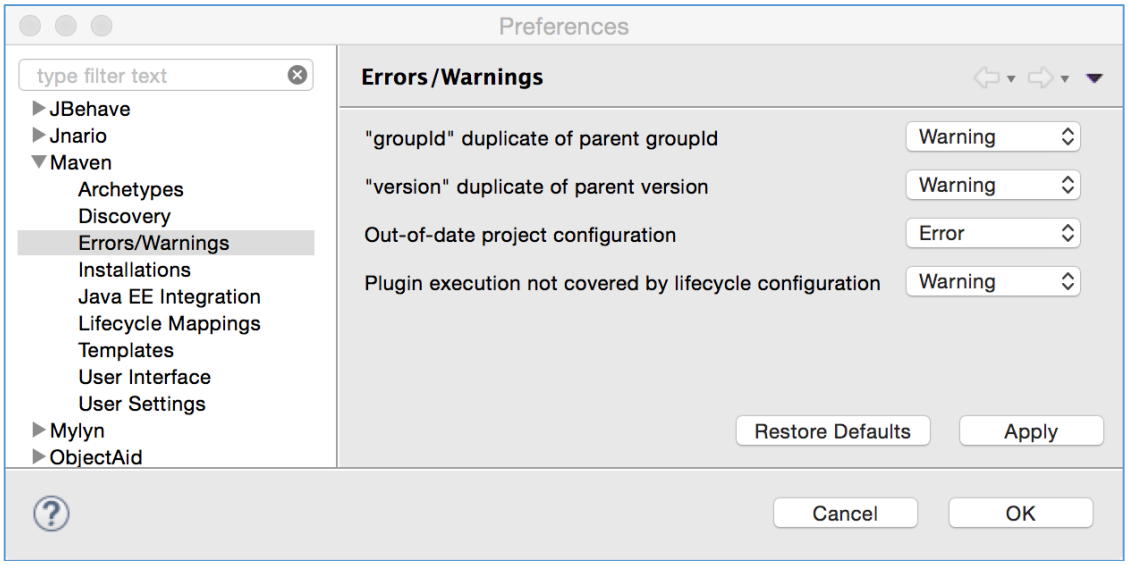
```
-Xms512m
-Xmx1024m
```

On a **Mac OS X** system, you can find `eclipse.ini` by right-clicking (or *Ctrl+click*) on the Eclipse executable in Finder, choose *Show Package Contents*, and then locate *eclipse.ini* in the *Eclipse* folder under *Contents*.

- If you cannot choose *Run As* | *Run as ScalaTest - Test*, check if your package declaration is OK

- 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.


Preferences
type filter text
JBehave
Jnario
Maven
Archetypes
Discovery
Errors/Warnings
Installations
Java EE Integration
Lifecycle Mappings
Templates
User Interface
User Settings
Mylyn
ObjectAid
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
Restore Defaults
Apply
Cancel
OK

ATTACHMENTS

CREATE A PROJECT FOR SCALATEST 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-scalatest*
    - Click *Finish*
3. Add the dependencies needed for *ScalaTest*:
    - Right click on project
    - Choose *Build Path | Configure Build Path…*
    - Choose the tab *Libraries*
    - Click *Add External JARS…* and browse to the folder *noMaven/scalatest/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/scala`
    - Right click on project and choose *Refresh*
    - Make it a *Scala* project: right click on project: *Configure | Add Scala nature*
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 *Scalatest GettingStarted* guide from the paragraph *Development* onwards…