# TCI – Week 1 practical PA1

This document is divided into two parts. The first part relates to configuring your local and school environment for TCI. The second part relates to unit tests.

These exercises should be executed individually. All of these are assignments are needed in the course.

## Assignment – Setup environment

**Guidelines**

The following unit tests can be run using only IntelliJ IDEA and a building tool called Gradle. We recommend to make these assignments under a Linux distribution (e.g.: Ubuntu).

### Install JDK

Download Java JDK and install it on your system.

### Install Gradle

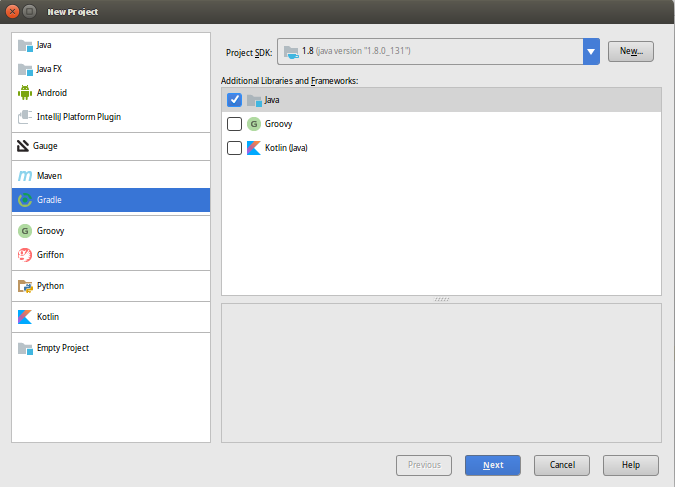
Follow the link: [https://gradle.org/install#with-a-package-manager](https://gradle.org/install)

Install Gradle on your system.

### Install IntelliJ Ultimate

Download the ultimate edition of IntelliJ. Register as a student: you will receive a license for the ultimate edition.

### Assignment – Running IntelliJ with Gradle



### Use Git

Install Git locally. Make a repository for this week assignments. Make sure IntelliJ uses the code from your repository (see: <https://www.jetbrains.com/help/idea/set-up-a-git-repository.html> )

Now create a project making use of Git & Gradle to manage the building of it.

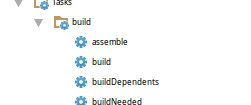
Note: you may have to point IntelliJ to your project SDK installation (/usr/lib/jvm/…).

On pressing Next, you will be prompted to choose a groupId and ArtifactId.

The group id normally looks like: org.<yourOrganizationName>.<yourProductName> ;

The artifact is the actual application name.

By default, IntelliJ adds JUnit to your project. Now either run “gradle build” from your command line (in your IntelliJ project folder) or double click on build task from the Gradle tool window (view, tool windows, Gradle: as you can see below).



|  |  |
| --- | --- |
|  | **Do not worry about exploring around!** By reading and getting your hands dirty you will be learning a lot about Gradle… and problem solving. |

Add the following dependencies to your project (gradle.build):

compile **group**: **'pl.pragmatists'**, **name**: **'JUnitParams'**, **version**: **'1.0.4'**

Intellij will download and add the dependencies for you. That is highly convenient since it allows you to move the project from a computer to another without having to mess around with downloaded packages and libraries (and of course, there is more to it).

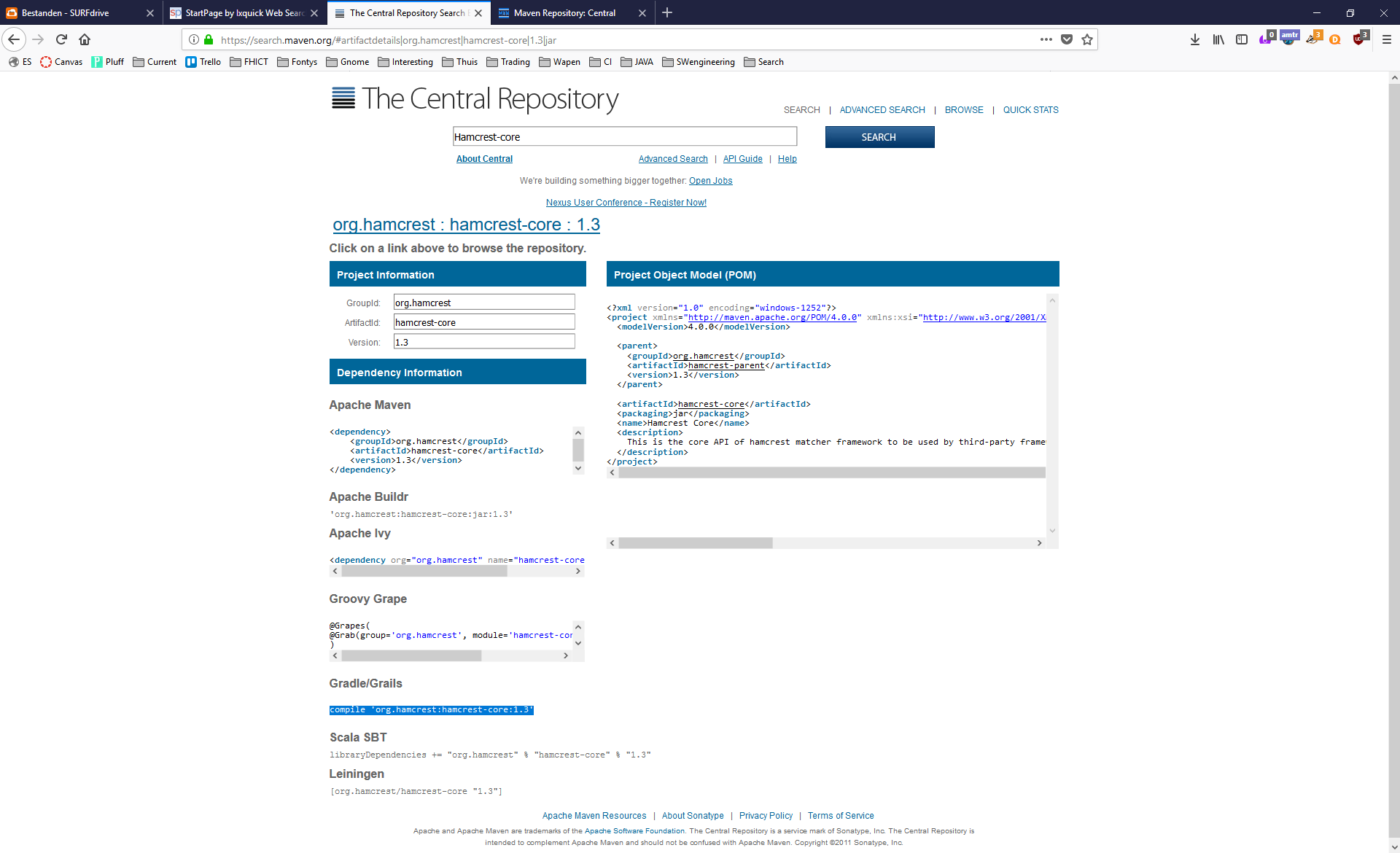
**Resources:**

Gradle home: [https://gradle.org](https://gradle.org/)

Gradle tutorial:<http://tutorials.jenkov.com/gradle/gradle-tutorial.html>

### Assignment – – Create your first JUnit test class

This step-by-step tutorial will show you how to create a simple JUnit test class and make it work by adding JUnit and Hamcrest as dependencies.

1. Use the Gradle Java project from the assignments above.
2. Go to <https://search.maven.org/> and search for the Hamcrest Core library
3. Add the reference found to you build.gradle file.
4. Now create a file called MyFirstClass under the new “main/java” folder. Add some attributes and a constructor and getters. Have IntelliJ create a test file for this class by using Ctrl-Shift-T.
5. Now you can run tests by pressing CTRL+SHIFT+F10.

### Assignment – Testing direct inputs and outputs

To get acquainted with Junit tests, let’s make some tests ourselves using IntelliJ. We use the BasicStatisticInterface where the behaviour of methods is described. To be able to end up with testable code, you can make a very simple implementation of class BasicStatistic, which implements the BasisStatisticInterface. (See week1\_code.zip)

Perform tasks below in the order they are described:

|  |  |
| --- | --- |
| Task | Description |
|  |  |
| Create tests from requirements for constructor and addDoubleToData | Create tests which test the following behaviours:  -the number of data items is 0 when none are present  -after adding a Double, the number of items is != 0 |
| Implement code | For constructor and addDoubleToData |
|  |  |
| Create tests from requirements for addDoubleToData and clearData | Create tests which test the following behaviours:  - after clearing of data, the number of items is 0  - after adding of X Double data items, the number of items is increased by X |
| Implement code | For addDoubleToData and clearData |
|  |  |
| Create tests for method sum() | Define and create tests which test the behaviour of the sum() method  -return sum of all items  -if no items return 0 |
| Implement code | For sum() |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

*Reflection points:*

* Is one test case enough for each method you have created? How many do you think should be created?

One test case is not enough for each method, ideally there should be tests for each method for the happy scenario, tests for border cases and exceptions that will be expected. There is no perfect number for the amount of tests but if these are tested it should mean good coverage for various outcomes.

* What kind of assert statements did you use to verify the test outcome?

assertEquals and assertNotEquals were used rather than assertTrue or asserFalse with equality operators as this would not give the correct result when comparing objects.

* Did your test manage to test all the methods you previously created?

Yes

* If you would be limited in time, how would YOU prioritize the order of the tests?

The test for happy scenario would be done first and then some exception testing

* Are your test names self-explanatory? Do you think another developer, even without knowing almost anything about your application, would be able to understand what you are testing?

Yes, the test names have been written to describe the input and outcome that is wanted for each test.

* Do you think that reading the body of your tests would be enough to understand what the purpose of each of them is? Are they -in other words- self-describing what they are meant to test? If not, what could you do to improve them?

The tests that I have written are self-describing for other developers.

### Assignment – HashMap (optional)

Complete the assignment 3.11.3 “HashMap” at page 47 from the book “Practical JUnit testing”.

========================== End of practical. ===============================