# TCI – week 1 practical

This document is divided into two main 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.

## Part A: 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).

### Assignment 1 – Install JDK

Download Java JDK and install it on your system.

### Assignment 2 – Install Gradle

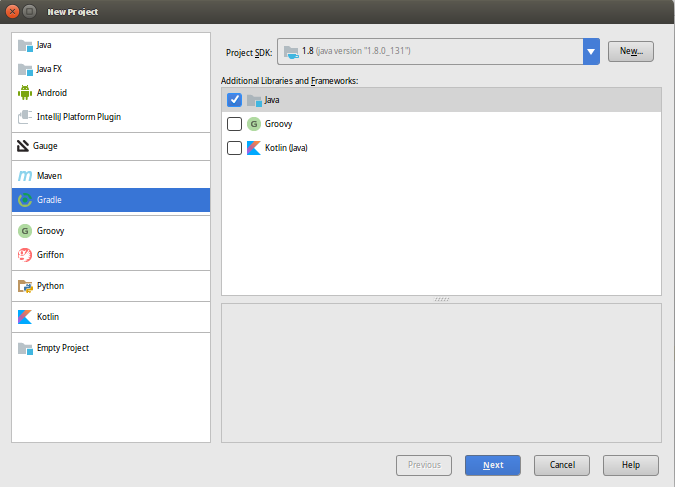
Follow the link: [https://gradle.org/install#with-a-package-manager](https://gradle.org/install#_blank)

Install Gradle on your system.

### Assignment 3 – Install IntelliJ Ultimate

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

### Assignment 4 – Running IntelliJ with Gradle



### Assignment 5 – 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.

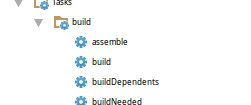
Note: you may have to point IntelliJ to your project SDK installation (/usr/lib/jvm/…).

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).

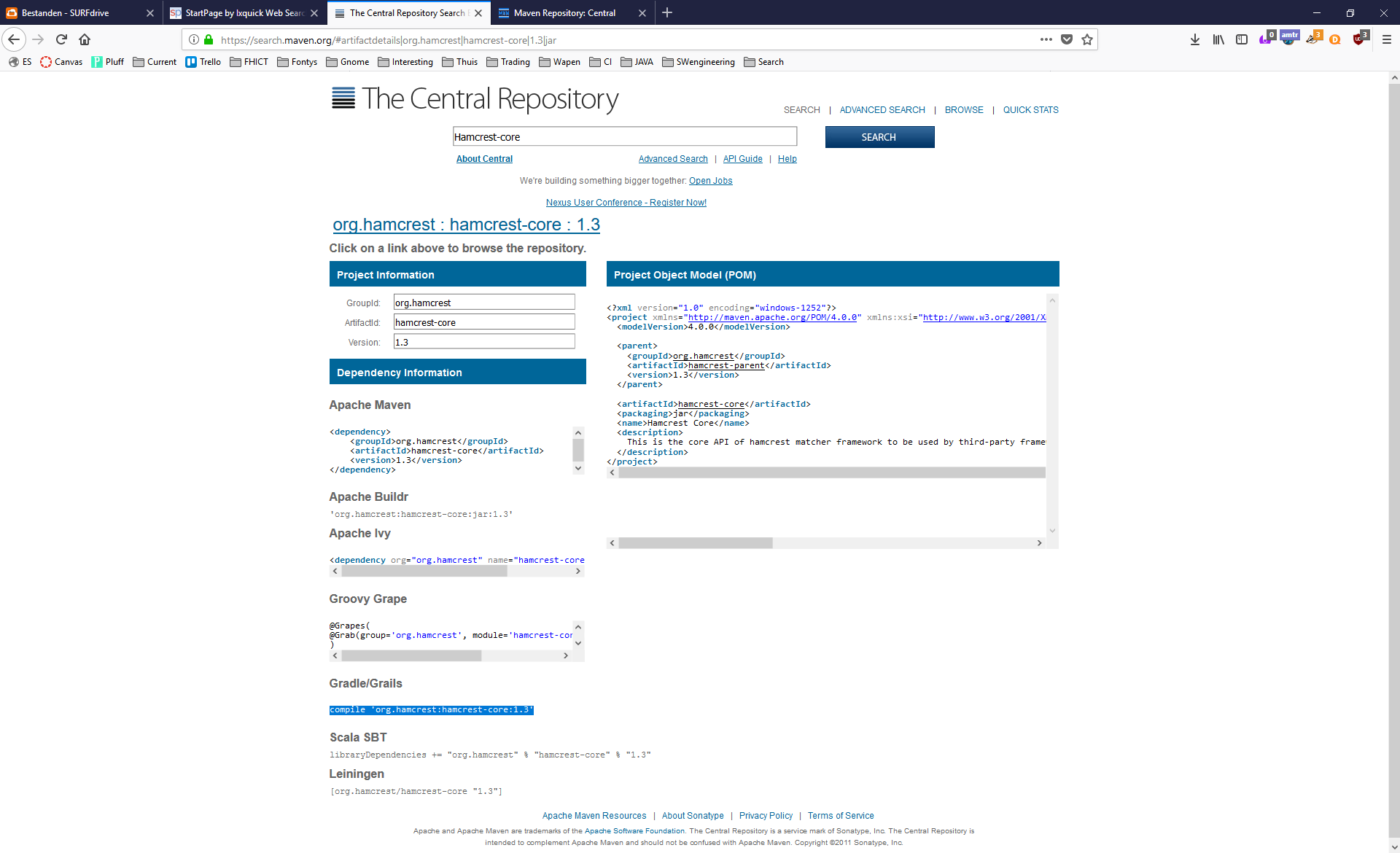
**Resources:**

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

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

### Tutorial 1 – 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 5 – 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 |
| Derive test cases from JavaDoc description of methods | Create tests which test the following behaviours:  -the number of data items is 0 when none are present  -after clearing of data, the number of items is 0  -after adding, the number of items is != 0  -after adding of X data items, the number of items is increased by X |
| Create tests for method sum() | Define and create tests which test the behaviour of the sum() method |
| Implement constructor and other methods | -add necessary attributes and constructor.  -add implementation of addDoubleToData, clearData, numberOfDataItems methods.  -run your rests, they all should be green. |
|  |  |
|  |  |
|  |  |
|  |  |

*Reflection points:*

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

*Based on the complexities of methods, more than one test case can be created for a method. However, in the current case, the tests are easy to be implemented and also do not interfere each other, therefore the tests can be combined (such as checking the data size and diagnosing if the data is empty or not).*

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

*Mostly assertEquals/NotEquals and assertTrue/False.*

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

*All the tests return good results.*

* Do you think it is relevant to know how many items are found in the list, for each test?

*No, since the tests are isolated, which is a behavior of Junit that new instance of BasicStatisticTest will be created for each test method.*

* 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?

*For each method it is clear to see what kind of test will be performed, and what to be expected as the results.*

* Did you have to make use of a lot of new instantiations in your test class?

*Only trivial instantiations, the mutually used object(s) between methods are initially created.*

* 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?

*We believe that the tests that we have written provide a clear and comprehensible structure to the readers.*

### Assignment 6 – Exceptions

* In assignment 5 you have worked on creating tests and later implementing the code for some basic statistic. Now we are going to expand this by making tests for the rest of the methods in BasicStatisticInterface

|  |  |
| --- | --- |
| Task | Description |
| * Describe behaviour | * Specify in JavaDoc which behaviour every method has (expand the existing JavaDoc of the interface. Also take into account maximum Double values (see [Double](https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html) JavaDoc). |
| * Create tests for each method | * Define and create tests which test the behaviour of the getMean and getHighestValue methods, both for ‘nice’ and ‘unique’ behaviour (throwing exceptions) |
| * Implement the methods | * Implement the minimum needed amount of code. * Run your rests, they all should be green. |
|  |  |

* Did you use of a lot of new instantiations in your test class?
* *Yes.*
* Did you decide to catch the exceptions by using the property:

@Expect(AnException.class)? Did you think of any other method to check whether the exception was thrown correctly? If so, what are the advantages/disadvantages of such method?

* *No, I used @Test(expected = AnException.class). I create a condition inside the function (If the list is empty, throw “IndexOutOfBoundsException”)*

### Assignment 7 – HashMap (optional)

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

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