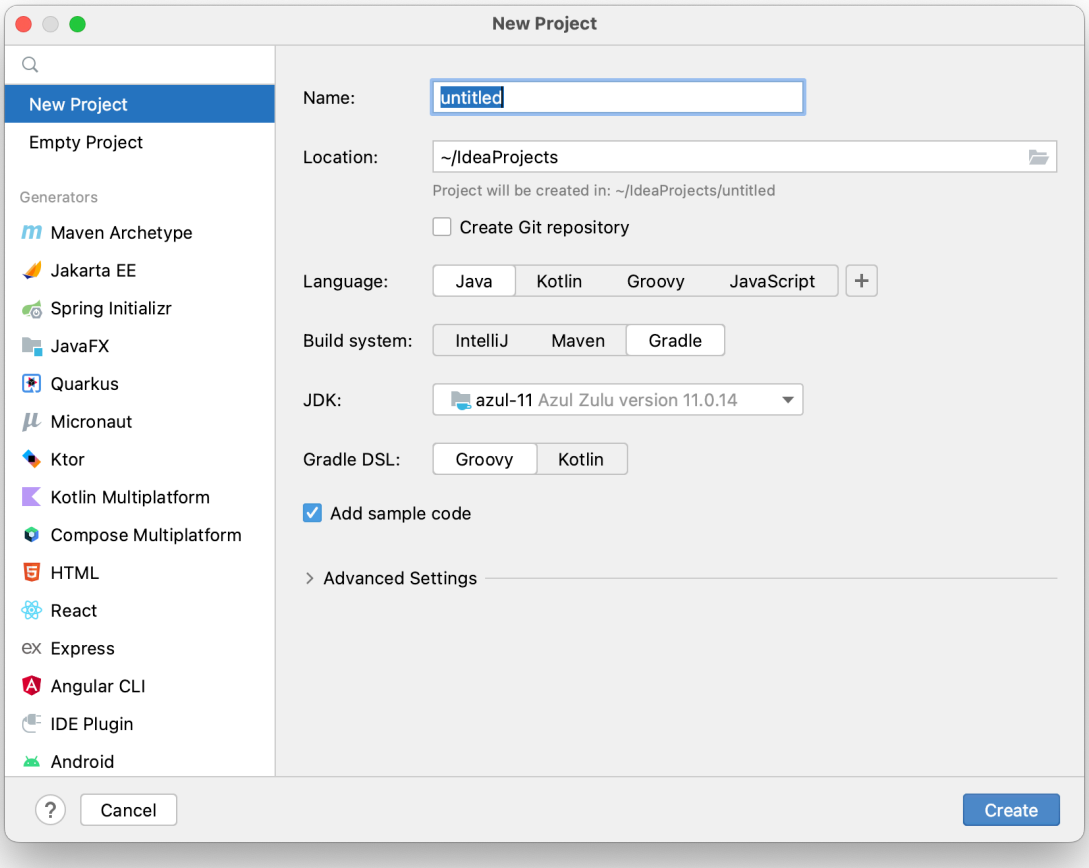
Object Oriented Architectures & Secure Development

© Howest University of Applied Sciences

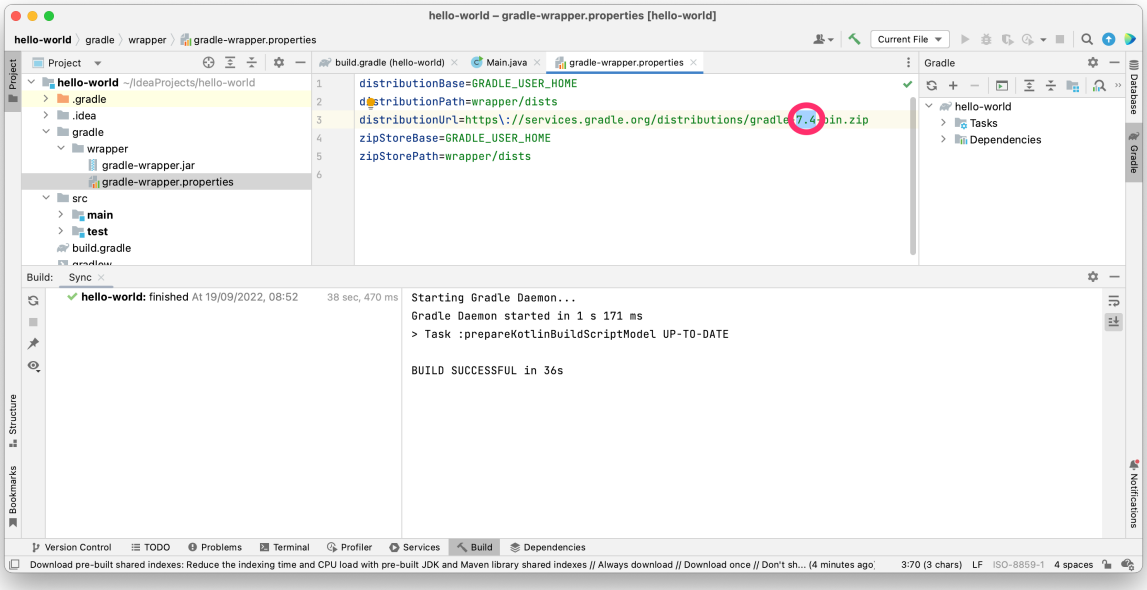
00-01 Basic concepts, a refresher - exercise

First things first

- Make sure you have installed the most recent version of the JetBrains IntelliJ IDEA. If necessary, upgrade your installation.
- Publish all your solutions in your (private) group on gitlab: https://git.ti.howest.be/TI/2022-2023/s3/object-oriented-architectures-and-secure-development/students.
- Create a new project/repository for each exercise with a different theme, reuse the existing project if the exercise has the same theme as a previous exercise.
- Create **a Gradle 7.4 project** with JDK version 11 (Azul Zulu). This is new as opposed to what we did in OOP last year, where we created a *simple* Java project.



Verify if the version of your gradle is 7.4, not lower, not higher. You can do this once IntelliJ is ready building your project (might take 1-2 minutes) and by locating the `grade-wrapper.propertes` -file.The version should be 7.4 (in the red circle). If not, change the number in this file, and click the elephant-with-the arrows icon that appears. This will cause IntelliJ to update (or downgrade) your gradle project to the right version.



This will be the starting point for each exercise (and or demo) for this course.

Lab intro

Today, we will be building a simple version of the student administration, namely the part where `students` enroll in `courses`. It should be feasible to finish the exercise within the duration of the lab session. Please signal any troubles or problems as soon as possible: all future topics will be taught based on this kind of base code for which we will not have time to explain them again in future sessions except today.

Students

Students have a name and an email address but also a uniquely generated number. By default the email is just the name of the student (with dots instead of spaces) and `@student.howest.be`. Only in some cases you will want to provide the email address explicitly during construction. In even more rare cases, you will want to supply the unique student number.

Either way, two students with the same number are expected to be the same student. And none of the properties of a student are expected to change. (These features require *special techniques* to implement).

Courses

A course has a title and a lecturer. Here `String` is a sufficient type for both. Further, a course has a number of ECTS-credits (e.g., 3 or 6) and an academic year (e.g., 2021). When displayed on screen, a course should look like this: `2020-2021: Programming Fundamentals (6 ECTS)`.

Besides the getters and setters (only implement those that you actually need), we want to following functionality:

We can `enroll` (*add*) students in a course. Of course each student can only take the same course once a year. Enrolling a student twice for the same course, is an exceptional situation. We should also be able to `retrieve all students` that take the course.

After the exam, each student can be `given a grade`: a whole number between 0 and 20 (check for valid input). Of course only students that take the course, can be graded and a student can only be graded once.

We should also be able to `retrieve the grade of student for a given course`. Retrieving the grade of a student that does not take the course is an exceptional situation. Retrieving the grade of a student before the exam (students initially have no grade) is also an exceptional situation.

Implement the method `double averageScore()` which, if possible, computes the average score.

Finally, implement the method `toFullString()` which should return a string which looks like this (a summary of all the grades):

```
2021-2022: Programming Fundamentals (6 ECTS)
============================================
Alice Anderson                            18
Bob Bobelina                               9
Cedric C.                                 11
```

(Extra) School

If all the previous features are implemented, add a final class `StudentAdministration` which keeps track of **all courses over all years**. Write the method `toString(Student)` which creates a (huge?) string that contains all the scores of all the courses a given student (argument) has ever taken. Make a second version `toString(Student, boolean)` that expects a boolean which specifies if the courses for which no grade is available is included or not. You can decide upon the format.

Tips

- Apply the various techniques studied during the classes last year.
- There isn't one "single solution". Make sure you can motivate your choices.
- It is your software, take ownership.