Object Oriented Architectures & Secure Development

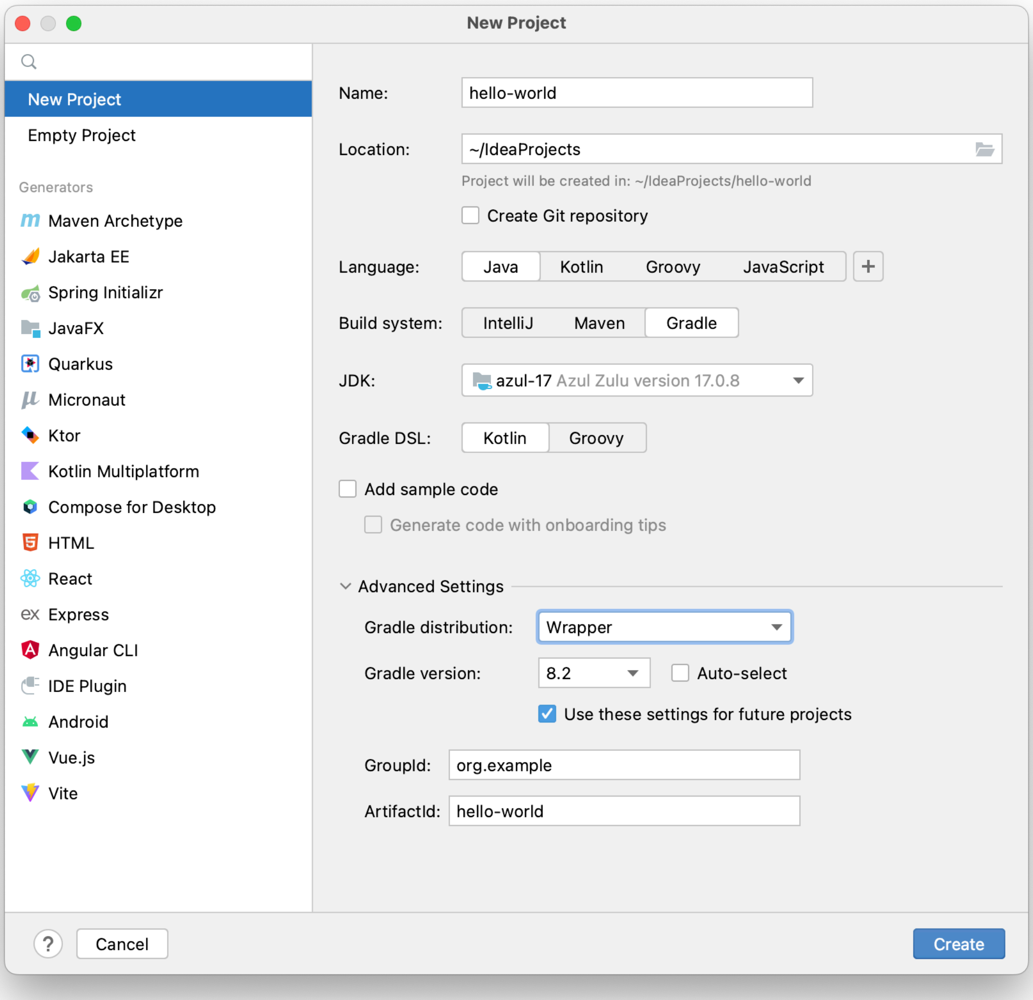
© Howest, University of Applied Sciences

Week 1

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) Gitlab group, which can be found under https://gitlab.ti.howest.be/ti/2023-2024/s3/object-oriented-architectures-and-secure-development/students.

  Attention: this is a new server (**gitlab**.ti.howest.be instead of git.ti.howest.be). So you'll need to reconfigure your SSH keys accordingly. Follow the instructions which can be found here: https://gitlab.ti.howest.be/ti-public/2023-2024/kickoff/-/wikis/software/SSH-Keys.
- 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.
- In this course, we will be using JDK version 17 and Gradle version 8.2. This is different from the OOP course last year, where we created a simple Java project.



Name: hello-world

Location: ~/IdeaProjects

Project will be created in: ~/IdeaProjects/hello-world

☐ Create Git repository

Language: [✓ Java] Kotlin Groovy JavaScript +

Build system: IntelliJ Maven [✓ Gradle]

JDK: azul-17 Azul Zulu version 17.0.8

Gradle DSL: [✓ Kotlin] Groovy

☐ Add sample code

☐ Generate code with onboarding tips

Gradle distribution: Wrapper

Gradle version: 8.2 ☐ Auto-select

☑ Use these settings for future projects

GroupId: org.example

ArtifactId: hello-world

Lab introduction: “jBamaflex”

Today we will be building a simple version of a Student Administration application, which we will call jBamaflex.

So, to begin, create a new Gradle project as specified above with the name `jBamaflex`.

Students

A student has a firstname, lastname and an email address. Also, a uniquely generated number is assigned to each student.

By default the email address is the firstname followed by a dot and a lastname, suffixed by `@student.howest.be`. Any spaces in the name are converted into dots as well. We do not take into account accented characters in this solution.

Attention: in some situations we might want to override the default email address and in even more rare situations, even the unique student number.

Students are considered to be the same when they have the same student number.

Also note that the number, name and email address are not subject to change: they cannot be modified after initialisation.

Courses

Of course, a student administration software packages also needs to manage courses.

Each course has a title and a lecturer (a string suffices) assigned to it, as well as a number of ECTS credits (e.g. 3 or 6) and an academic year (e.g. 2023). When displayed on screen, a course should look like this: `2023-2024: Programming Fundamentals (6 ECTS)`

Besides the classic getters and setters (only implement those actually needed), we want the following functionality.

We can enroll (=add) students in a course. A student can only take the course once per academic year. Trying to enroll a student twice for the same course is an exceptional situation.

It should also be possible to retrieve all students that take a specific course.

For our course, an exam is organised. After the exam, each student enrolled can be given a grade: an whole number between 0 and 20. Make sure to check for valid input, since getting a -1 or a 21 is not a valid situation. Of course, only students that actually take the course can be graded. Also, a student cannot be graded twice. This is an exceptional situation.

We should also be able to retrieve the grade of a specific student for the course. Retrieving the grade of a student that does not take the course is an exceptional situation. Retrieving the grade of a student before this student has been assigned a grade is also an exceptional situation.

For a course, it should also be possible to calculate the average score as a double. Try do this with as few lines of code as possible and avoid classic for loops.

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

```
2023-2024: Programming Fundamentals (6 ECTS)
============================================
Alice Anderson                            18
Bob Bobelina                               9
Cedric C.                                 11
```

Try to align the names and grades as in the example above to generate a nicely formatted table.

Testing

Write at least 3 useful tests to test the logic in your application. For example:

- Test whether sequentially creating two students will assign sequential student numbers to each invidial student.
- Test whether trying to grade a non-enrolled student causes an exceptional situation
- ...

Make sure these tests adhere to the AAA-pattern (see course).

In your main program

Try out all functionality in a running Java program:

- Create some students
- Create a course
- Enroll students
- Grade them
- Write out the full string representation (table) as described above
- Calculate and write out the average
- Per enrolled student, write out their email address

Attention points

- In case of exceptional situations, make sure to use a logger. *Do not* print the stack trace to the standard output (screen). Choose the appopriate log level.
- Avoid looping over collections of items: use the Stream API where possible/appropriate.
- Apply the various techniques studied during the classes last and this year.
- There isn't one "single solution". Make sure you can motivate your choices.
- It is your software, take ownership.

Some useful resources

- https://www.baeldung.com/java-string-of-repeated-characters
- https://www.baeldung.com/java-strings-concatenation
- https://www.baeldung.com/java-string-newline → see §2.2 to make your program platform independent
- https://www.baeldung.com/java-optional