Tutorials — Bootstrap

version **#1.0.0**

Yet Another Kind of Assistants 2026 <assistants@tickets.assistants.epita.fr>

Copyright

This document is for internal use at EPITA (website) only.

Contents

1 IntelliJ IDEA **3**
- 1.1 Introduction . . . 3
- 1.2 Installation . . . 3
- 1.3 Configuration . . . 3

2 Java **5**
- 2.1 Introduction . . . 5

3 Maven **5**
- 3.1 Introduction . . . 5
- 3.2 Installation . . . 5
- 3.3 Useful commands . . . 5

4 Bootstrap a maven project **6**
- 4.1 Setup of a Maven project . . . 6
- 4.2 Structure of a Maven project . . . 6

5 Dependencies **7**
- 5.1 Installing by yourself . . . 7
- 5.2 Using Nix . . . 7

*https://intra.forge.epita.fr

The goal of this document is to provide you with a list of tools that will be used during the *Java* workshop.

A more detailed presentation of each tool will be given in later tutorials so do not worry if you do not understand everything for now.

1 IntelliJ IDEA

1.1 Introduction

IntelliJ IDEA is an IDE (Integrated Development Environment) for projects built around the JVM (Java Virtual Machine). This tool has been chosen mainly for the following reasons:

- It is one of the best Java IDEs available so far.
- You can get a free education license for the Ultimate edition using your EPITA email address.
- It fully supports and integrates Maven, and many other useful tools.

1.2 Installation

If you would like to work on your own computer, you can install IDEA on *Windows*, *MacOS* and *Linux*. A complete tutorial is available here.

Be careful!

On *Windows*, you can use it with *WSL 2* at the cost of a very memory hungry software and virtual environment running (at least 16GB of RAM are recommanded).

1.3 Configuration

When starting the application, you will be asked for a license key or a JetBrains account. You can get an education license when creating a JetBrains account with your EPITA credentials here.

After opening a Maven project, you may be prompted to select the Java JDK that will be used by the IDE[1] to compile this project. On the PIE, you will be using *OpenJDK 21*. If no *SDK* is set up when trying to code on your first project, you can go to *Files -> Project Structure* and then set the *SDK* to the one available on your machine. On your personal setup, you can also download an *SDK* using the built-in feature of IDEA.

If you are at school in a machine room, you should be able to use the EPITA license server. To do this, click on the “Server License” tab.

Going further...

IDEA can be hard to use at first. You can find with this tutorial a cheat sheet with the essentials to get started. Furthermore, we recommend doing the optional (and short) *Triad* exercise to get more hands-on with IDEA.

[1] https://www.jetbrains.com/help/idea/sdk.html

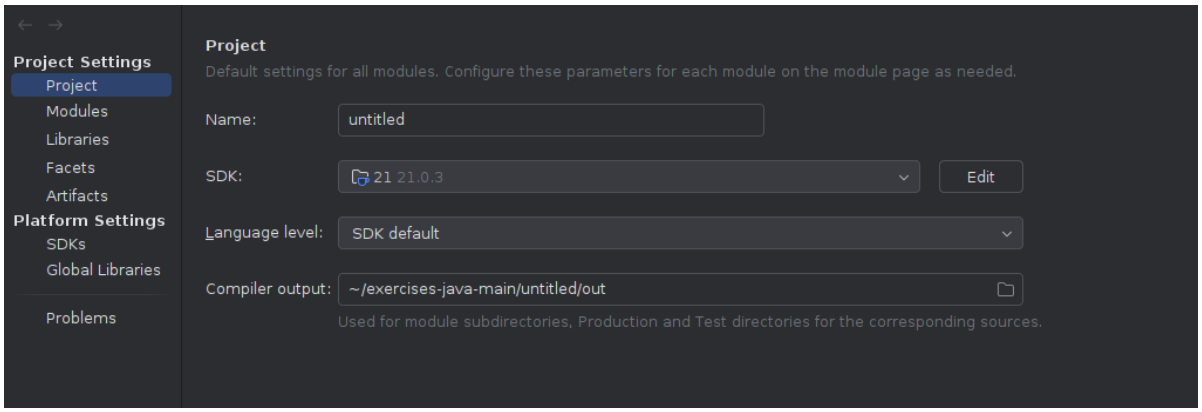

Fig. 1: SDK setup menu

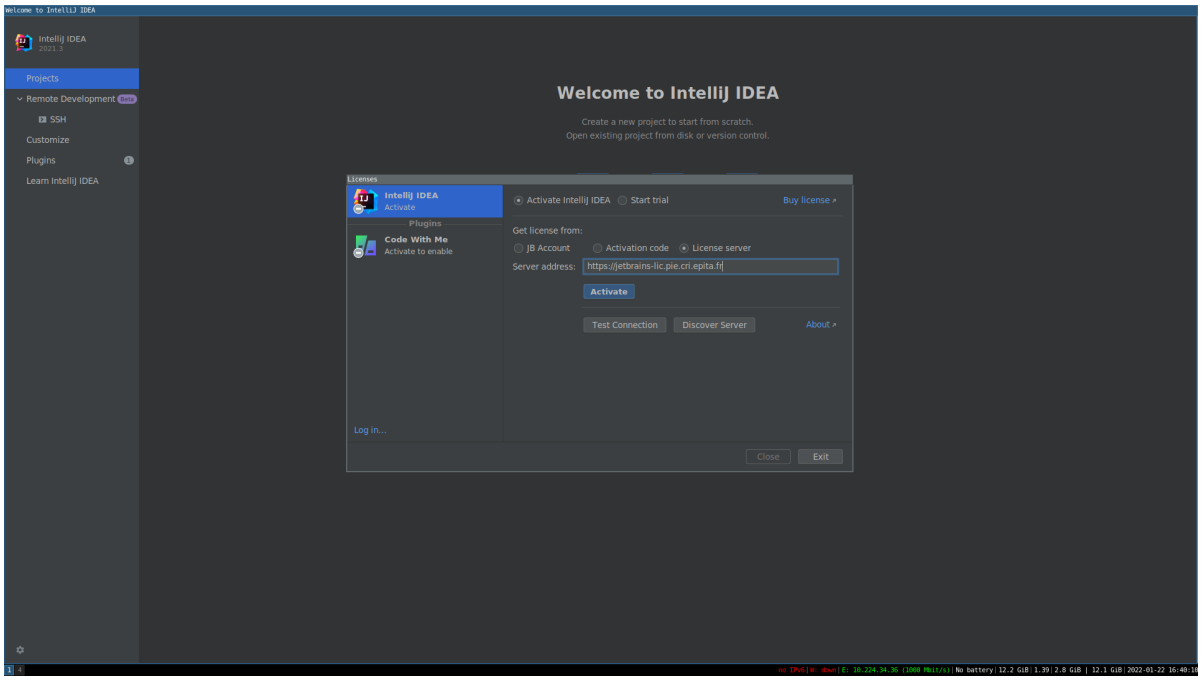

Fig. 2: License server menu

2 Java

2.1 Introduction

The *Java* programming language will be the focus of the coming workshop.

In order to develop and run *Java* applications, you need a Java Development Kit (JDK). We will be using JDK 21 for every Java-based project this year.

If for some reason one of your projects does not have the correct JDK setup, you can access the JDK selection from the `File` > `Project Structure` menu. Make sure `Project Language Level` is set to use the 21 *Java* language level.

3 Maven

3.1 Introduction

Maven is a project management and build tool for *Java*-based projects. During the workshop, you will mainly use Maven as a build system for your projects, similar to Meson with C.

3.2 Installation

We will be using Maven version 3.9.6. You can find helpful guides (such as Maven phases and project bootstraps, which will also be explained later on in this document) on the Maven website[1].

3.3 Useful commands

Here are the Maven commands you will use the most during the workshop:

- `mvn clean`: This command cleans the Maven project by deleting the target directory.
- `mvn compile`: This command compiles the individual source *Java* classes of the project.
- `mvn test`: This command runs the test cases of the project.
- `mvn package`: This command builds the Maven project and packages it into a JAR (compressed archive). It also runs the test cases of the project.

[1] https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

4 Bootstrap a maven project

4.1 Setup of a Maven project

You can generate a Maven project using the following command:

```
mvn archetype:generate -DgroupId=fr.epita.example \
                       -DartifactId=example-app \
                       -DarchetypeArtifactId=maven-archetype-simple \
                       -DarchetypeVersion=1.4 \
                       -DinteractiveMode=false
```

Here are the key parameters to understand:

- `groupId`: The unique identifier of your project. By convention, it takes the following form: `<reversed_domain_name>.<project_name>`.
- `artifactId`: The name of your project.

In order to enforce the use of Java 21 as our target Java version, you need to modify the `pom.xml` file and change the `properties` fields as follows:

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>21</maven.compiler.source>
    <maven.compiler.target>21</maven.compiler.target>
</properties>
```

If you prefer, it is also possible to setup a Maven project directly from the IDEA IDE. You can learn more from the IntelliJ website[3].

4.2 Structure of a Maven project

Here is a quick overview of the project generated by the above command.

- `example-app/`: The root directory of your project.
- `example-app/pom.xml`: The POM (Project Object Model), a file that describes your project, and is used by Maven to build and manage it.
- `example-app/src/main/java/`: The root directory of your project's main source files.
- `example-app/src/test/java/`: The root directory of your project's test source files.
- `[...]/fr/epita/example/App.java`: The *App* class from the *fr.epita.example* package. Classes and Packages are core elements of Java projects. You will learn more about them shortly.
- `example-app/src/site/`: You can ignore and even remove this directory.

You can find more information about the Standard Directory Layout on the Maven website[4].

[3] https://www.jetbrains.com/help/idea/maven-support.html
[4] https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html

5 Dependencies

You are asked to have some packages installed for the workshop. You have three choices:

- Working on the PIE
- Installing all packages by yourself
- Installing Nix and using the given commands

5.1 Installing by yourself

You **must** have *Java21*[1], *postgresql*, *maven*, *Quarkus*, *Nodejs22*[2] and *docker*[3] as root.

5.2 Using Nix

You need to install *docker* as root[3] on your own. You **should** install Nix as non-root using the guide[4]. You **should not** install Nix using your package manager.

You can test your install with this command:

```
42sh$ nix-shell -p lolcat
...
nix-shell$ echo test | lolcat
```

This command should print “test” on your terminal with a random color.

Then you need to write this as superuser in `/etc/nix/nix.conf`:

```
experimental-features = flakes nix-command
```

In the `triad` node, you have been given a `flake.nix` file. You should use the following command in the directory where `flake.nix` is stored.

Be careful!

You **should** add the `flake.nix` file to git.

```
42sh$ nix develop
```

This command creates a shell environment in which you have all required dependencies.

Be careful!

You **should not** add the `flake.lock` file created by the command to git.

[1] https://www.oracle.com/fr/java/technologies/downloads/#java21
[2] https://nodejs.org/en/download
[3] https://docs.docker.com/engine/install
[4] https://nixos.org/download

Be careful!

You **should** open IntelliJ while being in the shell environment.

Being a hero means fighting back even when it seems impossible.