

Java, IntelliJ IDEA and Maven

<https://github.com/heig-vd-dai-course>

[Web](#) • [PDF](#)

L. Delafontaine and H. Louis, with the help of GitHub Copilot.

This work is licensed under the [CC BY-SA 4.0](#) license.

Objectives

- Learn why Java is a popular programming language
- Manage multiple Java versions with SDKMAN!
- Develop Java apps with IntelliJ IDEA and Maven
- Manage dependencies with Maven
- Develop essential skills for professional Java development



Java

More details for this section in the [course material](#). You can find other resources and alternatives as well.

Java

- General-purpose, object-oriented language
- Write once, run anywhere (WORA)
- Created by James Gosling, 1995 at Sun Microsystems
- The documentation seems scary but you will get used to it and it is very useful



Java virtual machine

- Compiles source code to bytecode
- Executes in Java virtual machine (JVM)
- Where a JVM exists, Java can run (most of the time)

The screenshot shows the Oracle Java SE 21 & JDK 21 API Specification page. The browser address bar shows `docs.oracle.com/en/java/javase/21/docs/api/`. The page title is "Java® Platform, Standard Edition & Java Development Kit Version 21 API Specification". Below the title, it states: "This document is divided into two sections: Java SE and JDK." The Java SE section describes the core Java platform APIs, and the JDK section describes APIs specific to the JDK. Below this, there is a table of modules.

Module	Description
<code>java.base</code>	Defines the foundational APIs of the Java SE Platform.
<code>java.compiler</code>	Defines the Language Model, Annotation Processing, and Java Compiler APIs.
<code>java.datatransfer</code>	Defines the API for transferring data between and within applications.
<code>java.desktop</code>	Defines the AWT and Swing user interface toolkits, plus APIs for accessibility, audio, imaging, printing, and JavaBeans.
<code>java.instrument</code>	Defines services that allow agents to instrument programs running on the JVM.
<code>java.logging</code>	Defines the Java Logging API.
<code>java.management</code>	Defines the Java Management Extensions (JMX) API.
<code>java.management.rmi</code>	Defines the RMI connector for the Java Management Extensions (JMX) Remote API.
<code>java.naming</code>	Defines the Java Naming and Directory Interface (JNDI) API.
<code>java.net.http</code>	Defines the HTTP Client and WebSocket APIs.
<code>java.prefs</code>	Defines the Preferences API.
<code>java.rmi</code>	Defines the Remote Method Invocation (RMI) API.
<code>java.scripting</code>	Defines the Scripting API.
<code>java.se</code>	Defines the API of the Java SE Platform.

JVM versions

- Multiple implementations exist
- Can target different platforms and/or specific features
- JDK for development, JRE for running
- Eclipse Temurin is recommended

Which Version of JDK Should I Use?

To build and run Java applications, a Java Compiler, Java Runtime Libraries, and a Virtual Machine are required that implement the Java Platform, Standard Edition ("Java SE") specification.

The [OpenJDK](#) is the open source reference implementation of the Java SE Specification, but it is only the source code. Binary distributions are provided by different vendors for a number of supported platforms. These distributions differ in licenses, commercial support, supported platforms, and update frequency.

This site gives independent, yet opinionated recommendations.

TL;DR

✓ Recommendation: Use [Adoptium Eclipse Temurin 17](#) and ensure that your local version matches the CI and production version. Make sure, you have the latest patch level 17.0.3 or later, due to [CVE-2022-21449](#).

Java versions and version managers

- Java 21 is the latest LTS
- You can use [SDKMAN!](#) to manage multiple versions of Java
- Match versions for project consistency



Compiling and running Java programs

- Compile manually with `javac` command

```
javac HelloWorld.java
```

- Execute with `java` command

```
java HelloWorld
```

- Modern way: package into JAR files with the help of Maven

```
java -Xmx1024M -Xms1024M -jar minecraft_server.1.20.1.jar nogui
```


Summary

- Java is a general-purpose, class-based, object-oriented programming language.
- Java is compiled to bytecode, which is then executed by a Java virtual machine (JVM).
- Java is intended to be portable, thanks to the JVM.
- Java has various versions, each with its own set of features and improvements.
- Versions managers allow you to install and switch between different versions of Java.

IntelliJ IDEA

More details for this section in the [course material](#). You can find other resources and alternatives as well.

IntelliJ IDEA

- IDE for (Java) software development
- Developed by JetBrains
- Works on Windows, macOS, Linux
- Quite a standard in the industry



Community Edition and Ultimate Edition

- Community (free) and Ultimate (paid)
- Free student license available - you can get it!
- The Community Edition is sufficient for this course

The screenshot shows the JetBrains IntelliJ IDEA pricing page. The browser address bar displays 'jetbrains.com'. The page header includes the 'JET BRAINS' logo and 'IntelliJ IDEA'. A 'Pricing' dropdown menu is visible. The main heading is 'Subscription options & Pricing'. Below this, there are three tabs: 'For Organizations', 'For Individual Use' (selected), and 'Special Offers'. Under the 'For Individual Use' tab, there are two billing options: 'Yearly billing save 2 months' (selected) and 'Monthly billing'. The page displays two main product cards. The first card is for 'IntelliJ IDEA Ultimate', described as 'The Leading Java and Kotlin IDE', with a price of \$599.00 per user for the first year, \$479.00 for the second year, and \$359.00 for the third year onwards. The second card is for the 'All Products Pack', described as 'Get 10 IDEs, 3 extensions, 2 profilers, and a collaborative development service – all in one subscription.', with a price of \$779.00 per user for the first year, \$623.00 for the second year, and \$467.00 for the third year onwards. A 'Best offer' badge is present on the 'All Products Pack' card.

Product	Billing Cycle	First Year	Second Year	Third Year Onwards
IntelliJ IDEA Ultimate	Yearly billing (save 2 months)	\$599.00	\$479.00	\$359.00
	Monthly billing	-	-	-
	Special Offers	-	-	-
All Products Pack	Yearly billing (save 2 months)	\$779.00	\$623.00	\$467.00
	Monthly billing	-	-	-
	Special Offers	-	-	-

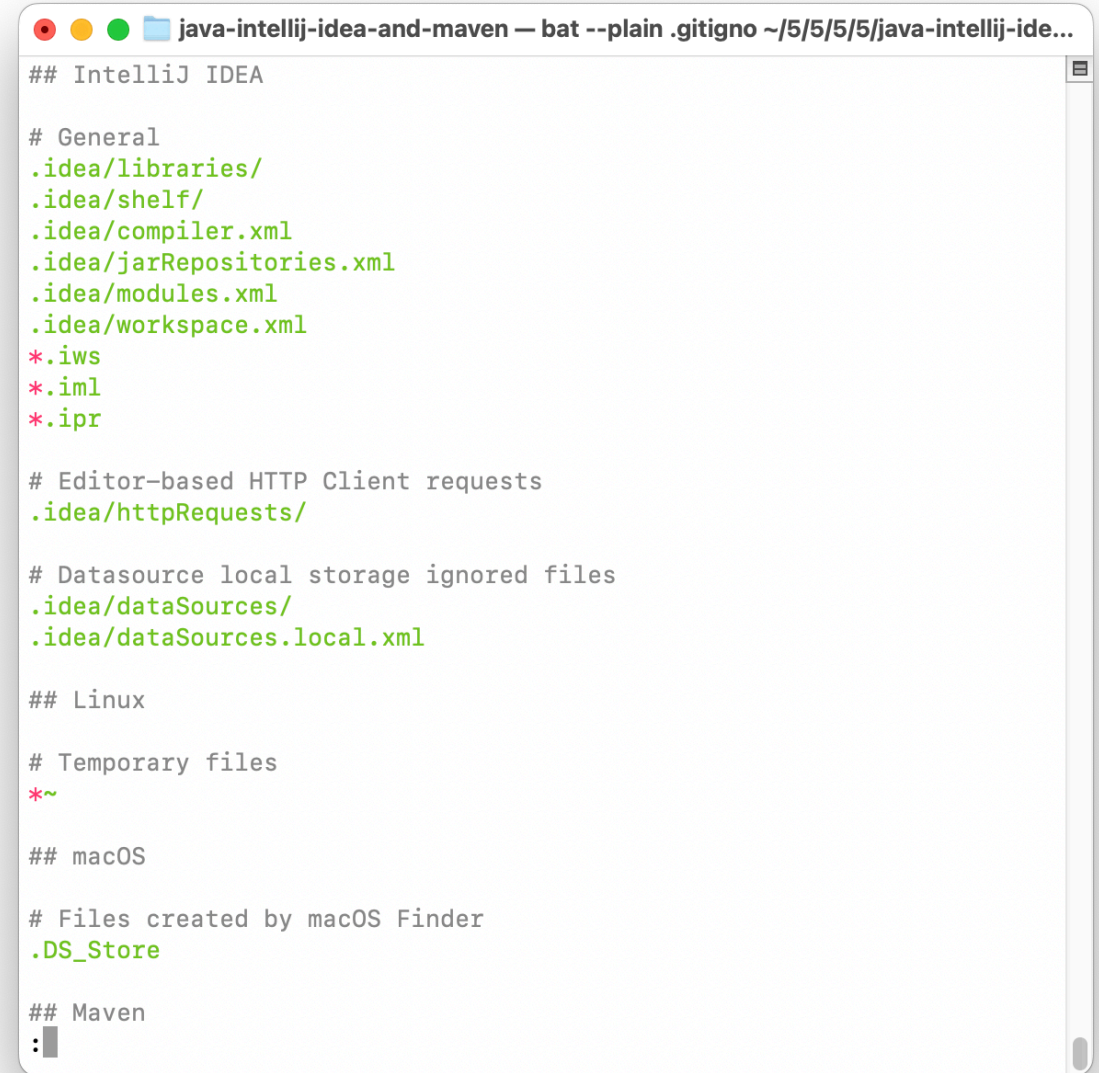
JetBrains Toolbox App

- Manage multiple JetBrains IDEs
- Install and update in one place
- Optional but very useful



Configuration files and Git

- `.idea` directory for project config
- Allow to share project settings between developers
- Some files must be ignored to avoid issue (local config) in Git



```
## IntelliJ IDEA

# General
.idea/libraries/
.idea/shelf/
.idea/compiler.xml
.idea/jarRepositories.xml
.idea/modules.xml
.idea/workspace.xml
*.iws
*.iml
*.ipr

# Editor-based HTTP Client requests
.idea/httpRequests/

# Datasource local storage ignored files
.idea/dataSources/
.idea/dataSources.local.xml

## Linux

# Temporary files
*~

## macOS

# Files created by macOS Finder
.DS_Store

## Maven
:
```

Summary

- IntelliJ IDEA is an integrated development environment (IDE) written in Java for developing computer software.
- IntelliJ IDEA is available in two editions: the Community Edition (free and open-source) and the Ultimate Edition (proprietary).
- You are eligible for a free student license for the Ultimate Edition.
- When creating a new project, IntelliJ IDEA will create a `.idea` directory containing the project configuration files.
- Some of these files must be ignored by Git, as they contain local configuration that is specific to your computer.

Maven

More details for this section in the [course material](#). You can find other resources and alternatives as well.

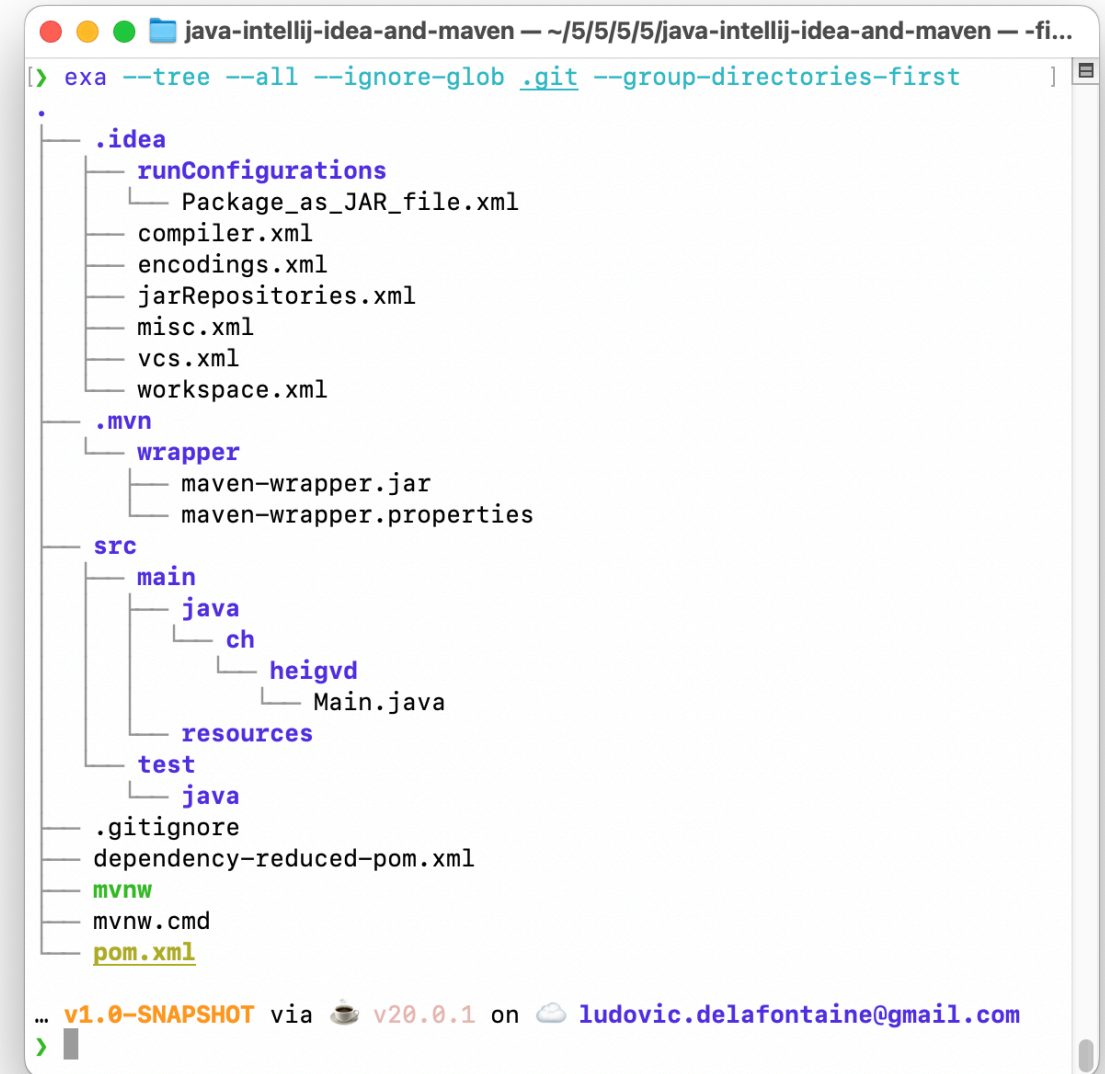
Maven

- Maintained by Apache Software Foundation
- Software project management tool
- Manages dependencies
- Build automation tool
- Allows to define a standard project structure



Maven project structure

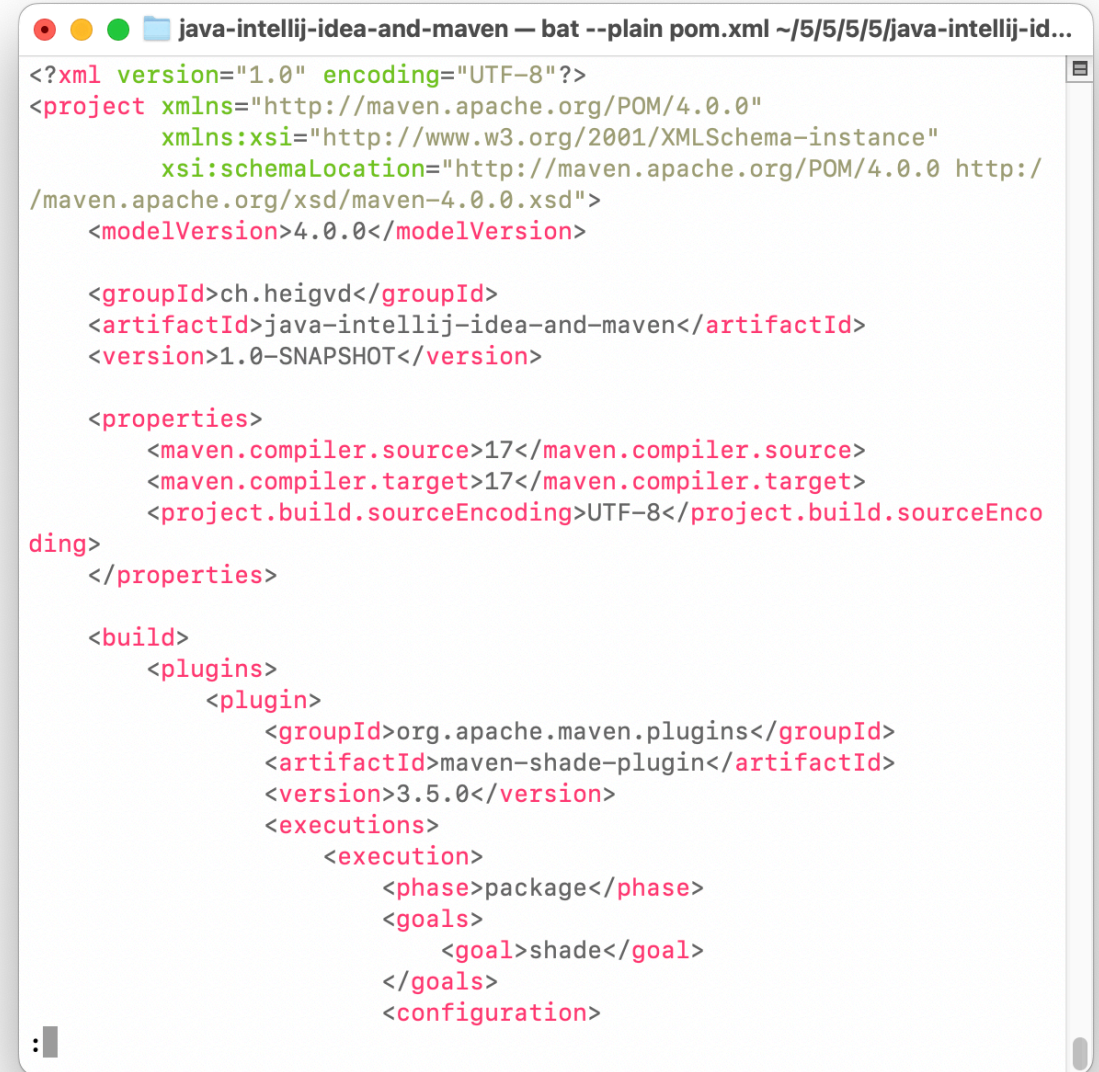
- Standardized directory structure
 - `src/main/java`
 - `src/main/resources`
 - `src/test/java`
- Simplifies build process with conventions



```
java-intellij-idea-and-maven — ~/5/5/5/java-intellij-idea-and-maven — -fi...  
[> exa --tree --all --ignore-glob .git --group-directories-first ]  
.  
├── .idea  
│   ├── runConfigurations  
│   │   └── Package_as_JAR_file.xml  
│   ├── compiler.xml  
│   ├── encodings.xml  
│   ├── jarRepositories.xml  
│   ├── misc.xml  
│   ├── vcs.xml  
│   └── workspace.xml  
├── .mvn  
│   └── wrapper  
│       ├── maven-wrapper.jar  
│       └── maven-wrapper.properties  
├── src  
│   ├── main  
│   │   ├── java  
│   │   │   └── ch  
│   │   │       └── heigvd  
│   │   │           └── Main.java  
│   │   └── resources  
│   └── test  
│       └── java  
├── .gitignore  
├── dependency-reduced-pom.xml  
├── mvnw  
├── mvnw.cmd  
└── pom.xml  
... v1.0-SNAPSHOT via ☕ v20.0.1 on ☁ ludovic.delafontaine@gmail.com  
> █
```

pom.xml file

- Configuration and build settings
- Shared among developers
- Defines dependencies and plugins:
 - Plugins extend Maven functionality
 - Dependencies are external libraries



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
/maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>ch.heigvd</groupId>
  <artifactId>java-intellij-idea-and-maven</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEnco
ding>
  </properties>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-shade-plugin</artifactId>
        <version>3.5.0</version>
        <executions>
          <execution>
            <phase>package</phase>
            <goals>
              <goal>shade</goal>
            </goals>
            <configuration>
```

Maven lifecycle

- Maven defines build process
- Composed of phases and goals
- Phases load plugin goals
- Goals execute tasks on your project (e.g., compile, test, package)



The screenshot shows the Apache Maven Project website. The header includes the Apache Maven Project logo and the URL <http://maven.apache.org/>. The main navigation bar contains links for [Apache](#), [Maven](#), [Introduction to the Build Lifecycle](#), [Download](#), [Get Sources](#), and [Last Published: 2023-09-15](#).

The left sidebar contains a navigation menu with the following sections:

- Welcome
- License
- ABOUT MAVEN
 - What is Maven?
 - Features
 - Download
 - Use
 - Release Notes
- DOCUMENTATION
 - Maven
 - Plugins
 - Maven Extensions
 - Index (category)
 - User Centre
 - Maven in 5 Minutes
 - Getting Started Guide
 - Naming Conventions
 - The

The main content area is titled "Introduction to the Build Lifecycle" and includes a "Table Of Contents" with the following items:

- Build Lifecycle Basics
- Setting Up Your Project to Use the Build Lifecycle
 - Packaging
 - Plugins
- Lifecycle Reference
- Built-in Lifecycle Bindings

The "Build Lifecycle Basics" section explains that Maven is based around the central concept of a build lifecycle, which is clearly defined for building and distributing a particular artifact (project). It states that for the person building a project, it is only necessary to learn a small set of commands to build any Maven project, and the **POM** will ensure they get the results they desired.

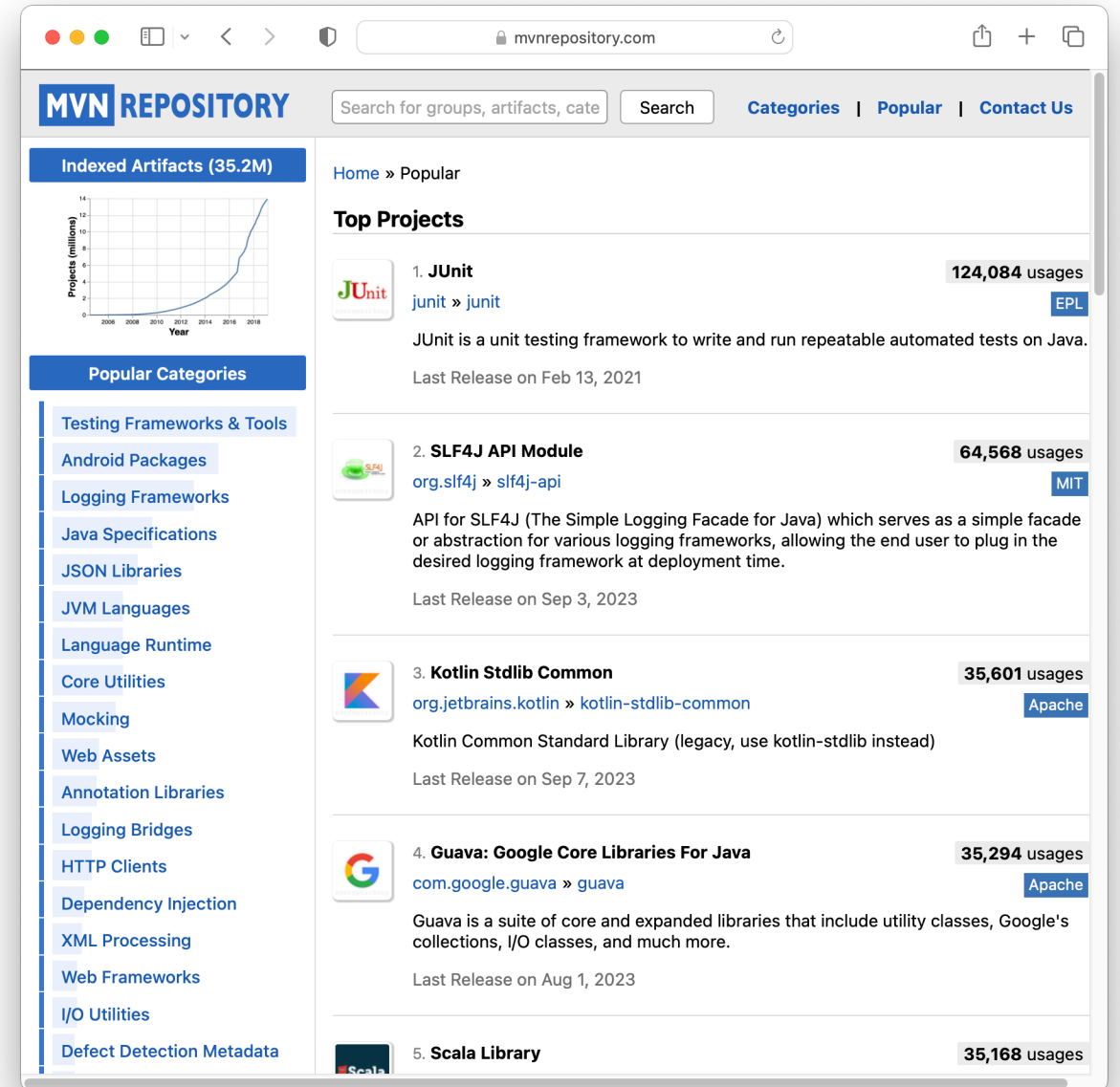
There are three built-in build lifecycles: default, clean and site. The **default** lifecycle handles your project deployment, the **clean** lifecycle handles project cleaning, while the **site** lifecycle handles the creation of your project's web site.

The "A Build Lifecycle is Made Up of Phases" section explains that each of these build lifecycles is defined by a different list of build phases, wherein a build phase represents a stage in the lifecycle. For example, the default lifecycle comprises of the following phases (for a complete list of the lifecycle phases, refer to the [Lifecycle Reference](#)):

- validate** - validate the project is correct and all necessary information is available
- compile** - compile the source code of the project
- test** - test the compiled source code using a suitable unit testing framework. These tests should not

Maven Repository

- Public repository of Java libraries
- Maven can download dependencies automatically
- You can publish your own libraries
- Many libraries available such as picocli, a library for building CLI applications



Maven wrapper

- Allows to use Maven without installing it
- Wrapper script downloads Maven
- Ensures consistent Maven version
- Use `mvnw` or `mvnw.cmd` instead of `mvn`



Summary

- Maven is a software project management and comprehension tool.
- Maven is a dependency manager for Java projects.
- Maven is a build automation tool for Java projects.
- Maven defines a standard directory structure for Java projects.
- Maven defines a standard build process for Java projects.
- The `pom.xml` file contains the configuration of your Maven project.

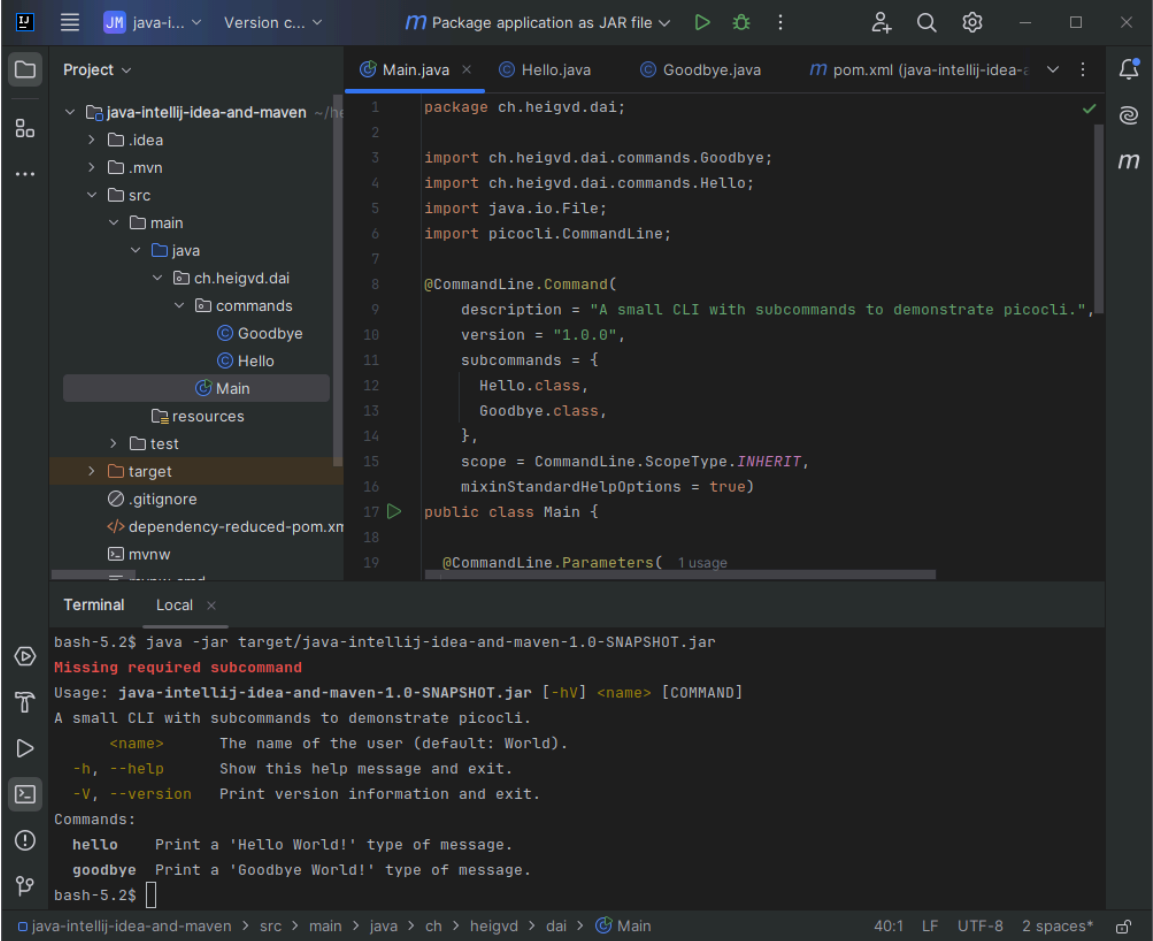
Questions

Do you have any questions?

Practical content

What will you do?

- Install and configure SDKMAN!, Java, Maven and IntelliJ IDEA
- Create and run a small CLI application with picocli, an external Maven dependency
- Publish your project on GitHub



The screenshot shows the IntelliJ IDEA IDE interface. The top toolbar includes a 'Package application as JAR file' button. The 'Project' view on the left shows the project structure: `java-intellij-idea-and-maven` (root) contains `.idea`, `.mvn`, `src` (with `main` and `test` subdirectories), `target`, `.gitignore`, `dependency-reduced-pom.xml`, and `mvnw`. The `src/main/java` directory contains the package `ch.heigvd.dai.commands` with classes `Goodbye`, `Hello`, and `Main`. The `Main.java` file is open in the editor, showing the following code:

```
1 package ch.heigvd.dai;
2
3 import ch.heigvd.dai.commands.Goodbye;
4 import ch.heigvd.dai.commands.Hello;
5 import java.io.File;
6 import picocli.CommandLine;
7
8 @CommandLine.Command(
9     description = "A small CLI with subcommands to demonstrate picocli.",
10    version = "1.0.0",
11    subcommands = {
12        Hello.class,
13        Goodbye.class,
14    },
15    scope = CommandLine.ScopeType.INHERIT,
16    mixinStandardHelpOptions = true)
17 public class Main {
18
19     @CommandLine.Parameters(1 usage
```

The terminal at the bottom shows the command `bash-5.2$ java -jar target/java-intellij-idea-and-maven-1.0-SNAPSHOT.jar` being executed. The output indicates a missing subcommand and shows the usage information:

```
Missing required subcommand
Usage: java-intellij-idea-and-maven-1.0-SNAPSHOT.jar [-hV] <name> [COMMAND]
A small CLI with subcommands to demonstrate picocli.
<name> The name of the user (default: World).
-h, --help Show this help message and exit.
-V, --version Print version information and exit.
Commands:
hello Print a 'Hello World!' type of message.
goodbye Print a 'Goodbye World!' type of message.
bash-5.2$
```

Find the practical content

You can find the practical content for this chapter on [GitHub](#).



Finished? Was it easy? Was it hard?

Can you let us know what was easy and what was difficult for you during this chapter?

This will help us to improve the course and adapt the content to your needs. If we notice some difficulties, we will come back to you to help you.

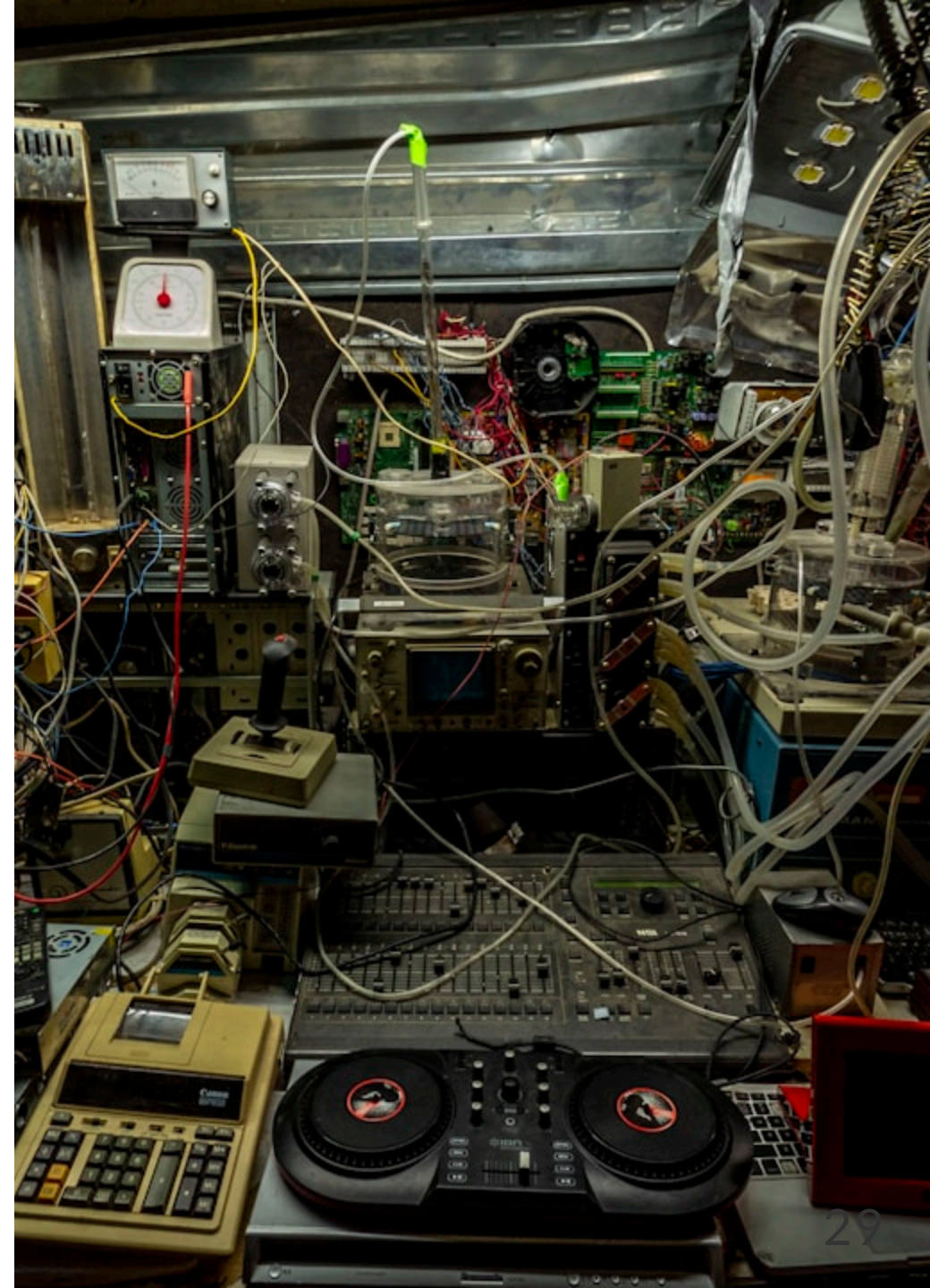
 [GitHub Discussions](#)

You can use reactions to express your opinion on a comment!

What will you do next?

In the next chapter, you will learn the following topics:

- Java IOs: input/output processing
 - How to read and write files?
 - Why is encoding important?
 - How to deal with exceptions?



Sources

- Main illustration by [Nathan Dumlao](#) on [Unsplash](#)
- Illustration by [Aline de Nadai](#) on [Unsplash](#)
- Java logo by [Java](#)
- SDKMAN! logo by [SDKMAN!](#)
- IntelliJ IDEA and IntelliJ Toolbox logos by [JetBrains](#)
- Maven logo by [Apache Software Foundation](#)
- Illustration by [Nathan Dumlao](#) on [Unsplash](#)