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.

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

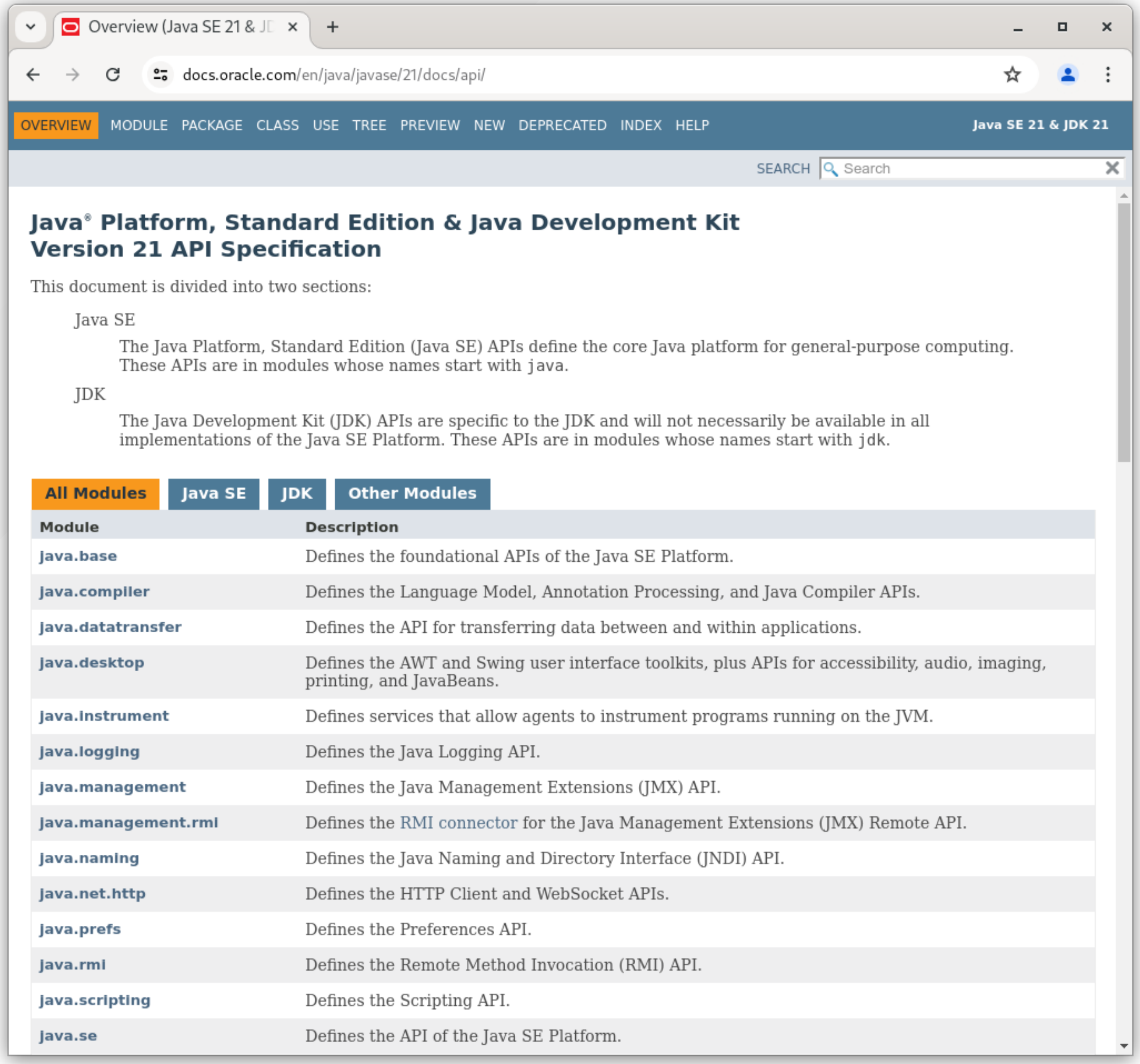
More details for this section in the course material. You can find other resources and alternatives as well.

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)


Overview (Java SE 21 & JD
docs.oracle.com/en/java/javase/21/docs/api/
OVERVIEW MODULE PACKAGE CLASS USE TREE PREVIEW NEW DEPRECATED INDEX HELP
Java SE 21 & JDK 21
SEARCH
Search

Java® Platform, Standard Edition & Java Development Kit Version 21 API Specification

This document is divided into two sections:

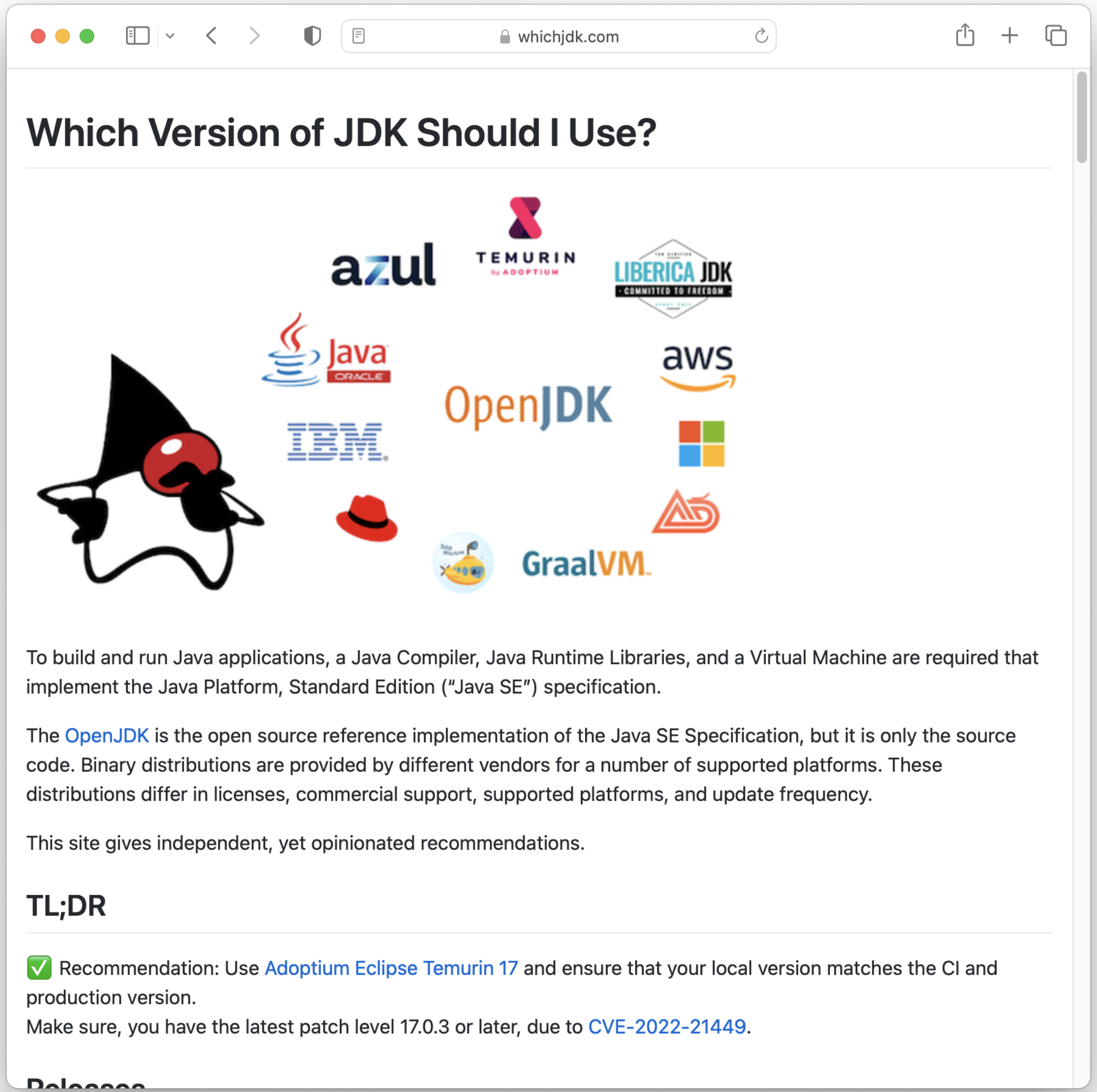
Java SE
: The Java Platform, Standard Edition (Java SE) APIs define the core Java platform for general-purpose computing. These APIs are in modules whose names start with `java`.

JDK
: The Java Development Kit (JDK) APIs are specific to the JDK and will not necessarily be available in all implementations of the Java SE Platform. These APIs are in modules whose names start with `jdk`.

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

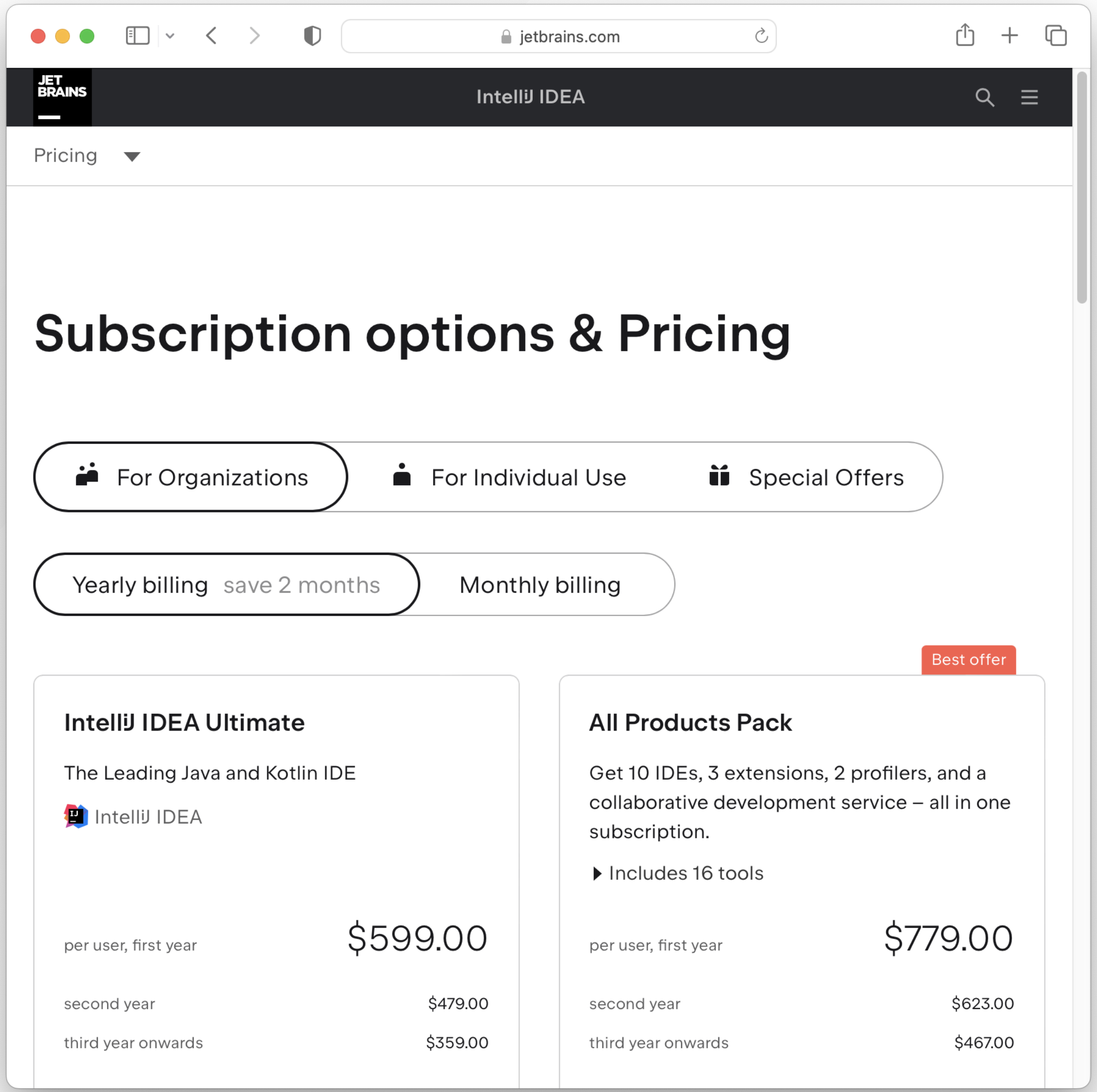
More details for this section in the course material. You can find other resources and alternatives as well.

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



Subscription options & Pricing

For Organizations | For Individual Use | Special Offers

Yearly billing save 2 months | Monthly billing

Best offer

IntelliJ IDEA Ultimate

The Leading Java and Kotlin IDE

IntelliJ IDEA

per user, first year	$599.00
second year	$479.00
third year onwards	$359.00

All Products Pack

Get 10 IDEs, 3 extensions, 2 profilers, and a collaborative development service – all in one subscription.

Includes 16 tools

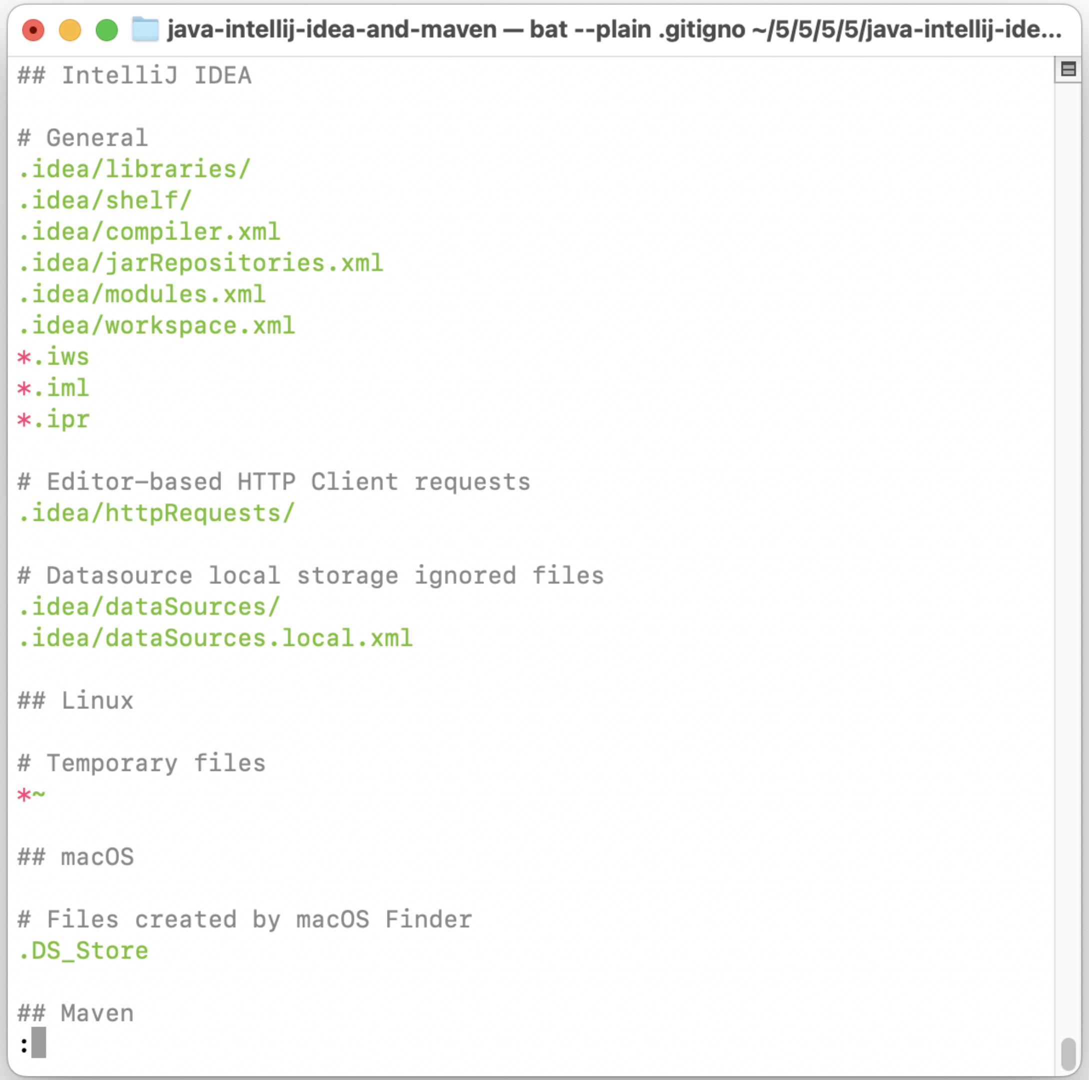
per user, first year	$779.00
second year	$623.00
third year onwards	$467.00

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


java-intellij-idea-and-maven — bat --plain .gitigno ~/5/5/5/5/java-intellij-ide...

```
## 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

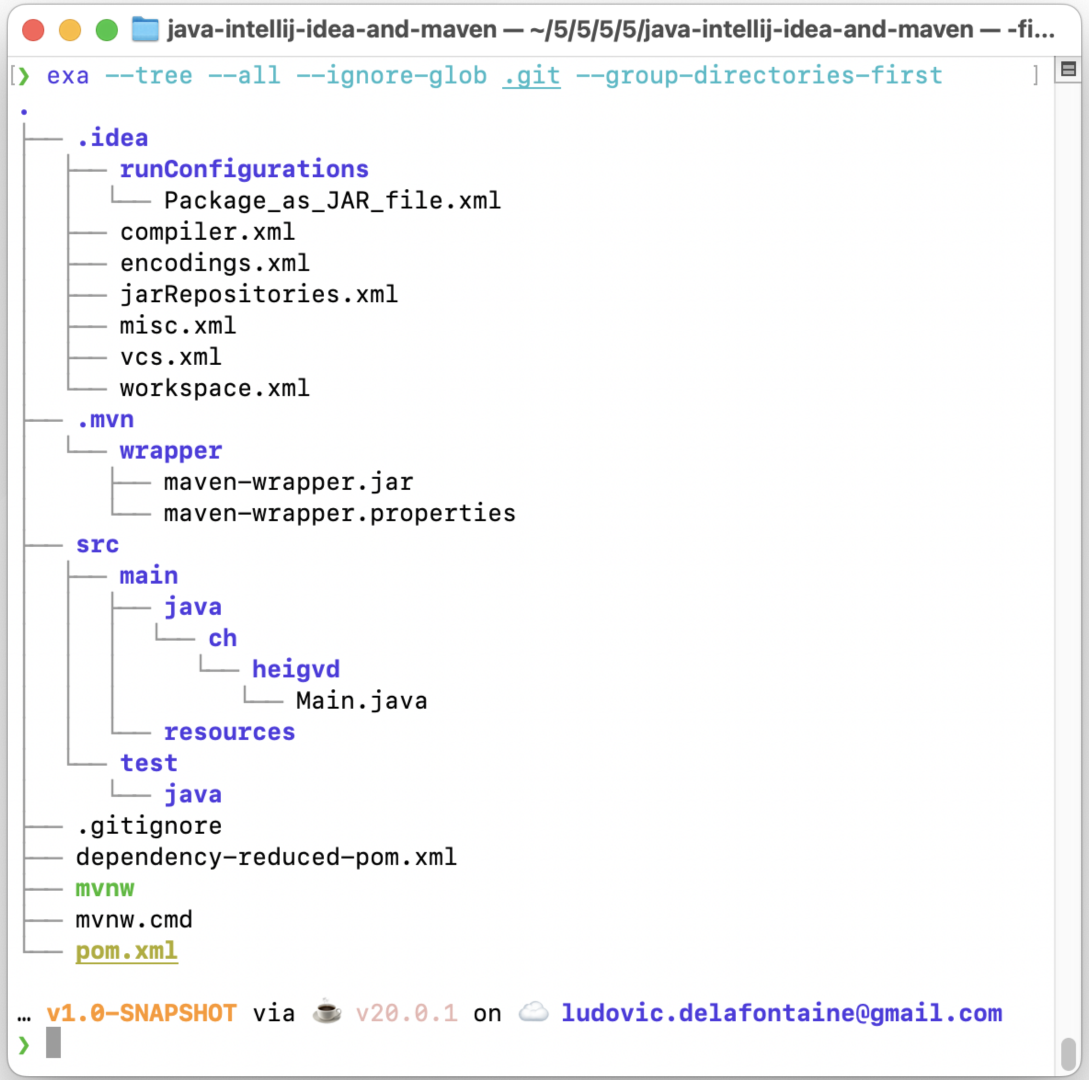
More details for this section in the course material. You can find other resources and alternatives as well.

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

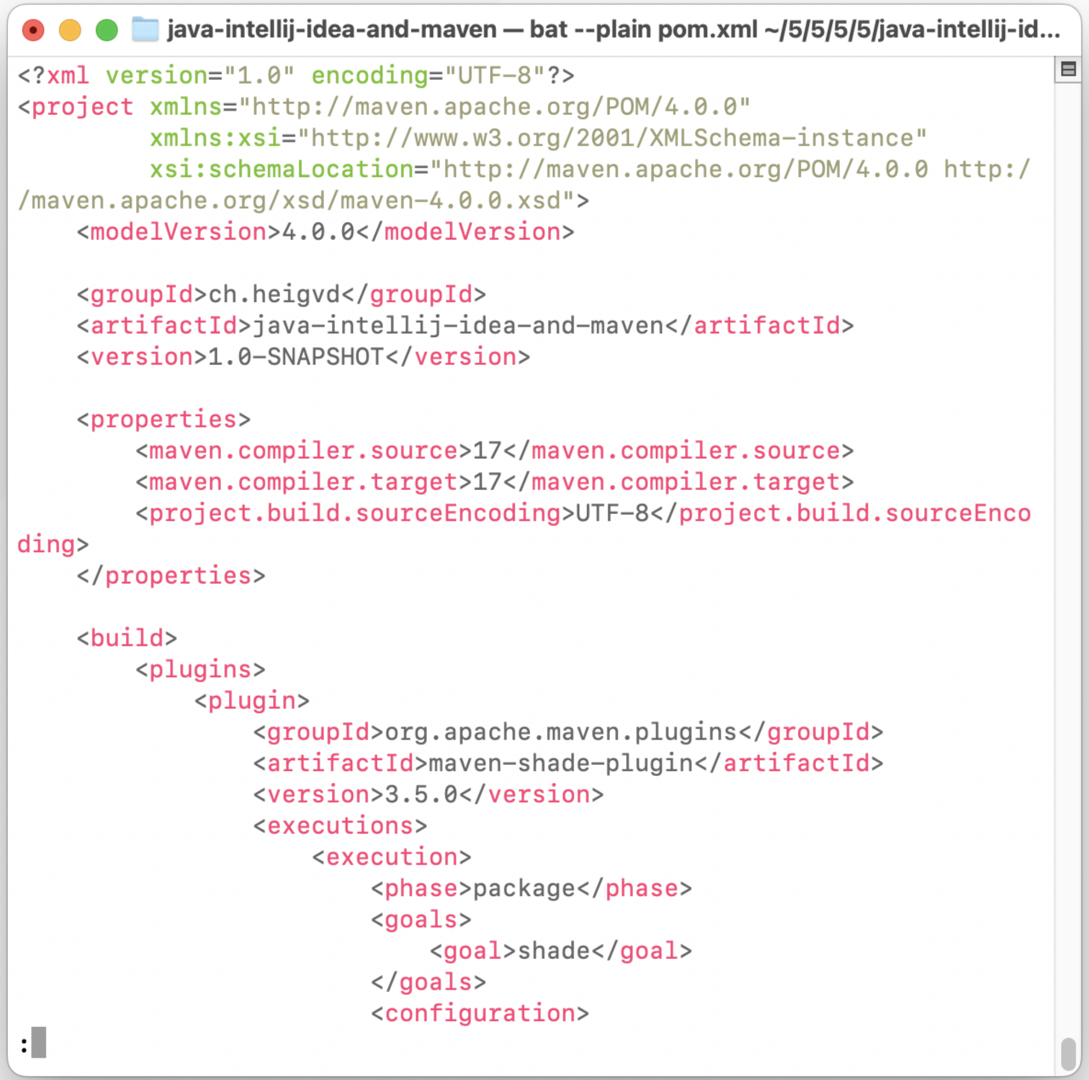


java-intellij-idea-and-maven — ~/5/5/5/5/java-intellij-idea-and-maven — -fi...

```
❯ 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

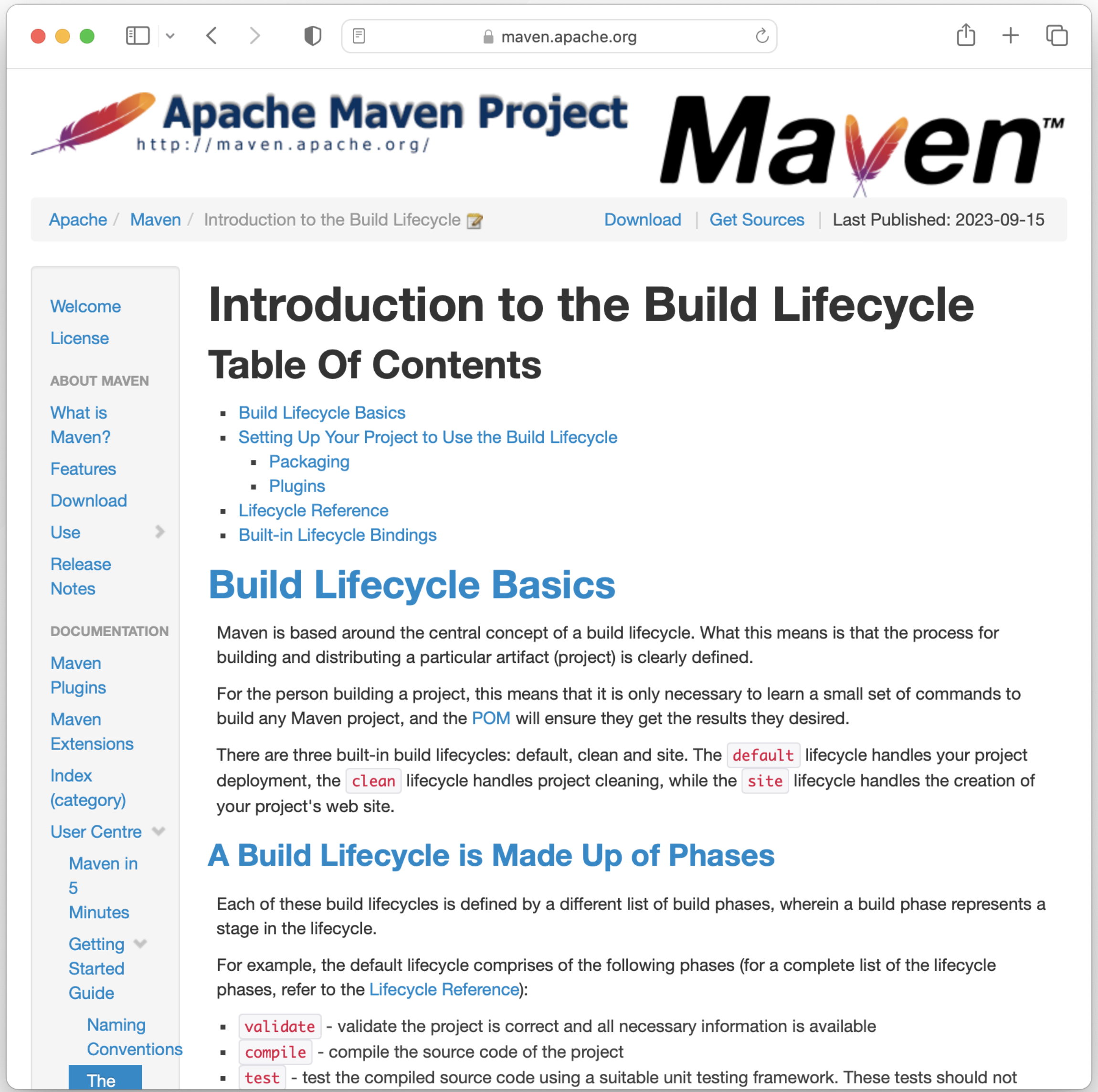
- Configuration and build settings
- Shared among developers
- Defines dependencies and plugins:
  - Plugins extend Maven functionality
  - Dependencies are external libraries


java-intellij-idea-and-maven — bat --plain pom.xml ~/5/5/5/5/java-intellij-id...

```
<?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)



Introduction to the Build Lifecycle

Table Of Contents

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

Build Lifecycle Basics

Maven is based around the central concept of a build lifecycle. What this means is that the process for building and distributing a particular artifact (project) is clearly defined.

For the person building a project, this means that 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.

A Build Lifecycle is Made Up of Phases

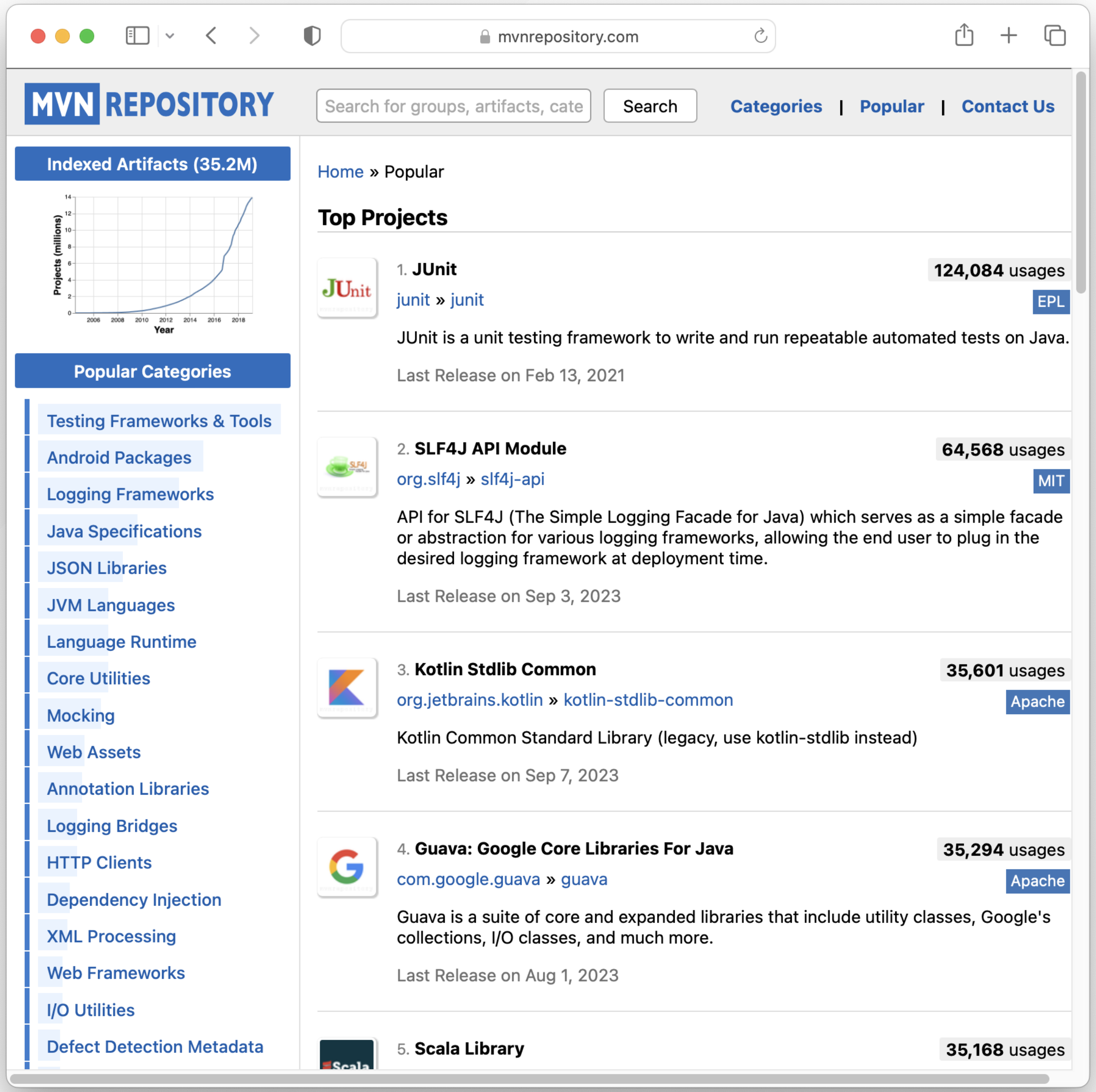
Each of these build lifecycles is defined by a different list of build phases, wherein a build phase represents a stage in the lifecycle.

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


mvnrepository.com
MVN REPOSITORY
Search for groups, artifacts, cate
Search
Categories | Popular | Contact Us
Indexed Artifacts (35.2M)
Popular Categories

Testing Frameworks & Tools
Android Packages
Logging Frameworks
Java Specifications
JSON Libraries
JVM Languages
Language Runtime
Core Utilities
Mocking
Web Assets
Annotation Libraries
Logging Bridges
HTTP Clients
Dependency Injection
XML Processing
Web Frameworks
I/O Utilities
Defect Detection Metadata

Home » Popular

Top Projects

1. JUnit — 124,084 usages
junit » junit — EPL
JUnit is a unit testing framework to write and run repeatable automated tests on Java.
Last Release on Feb 13, 2021

2. SLF4J API Module — 64,568 usages
org.slf4j » slf4j-api — MIT
API for SLF4J (The Simple Logging Facade for Java) which serves as a simple facade or abstraction for various logging frameworks, allowing the end user to plug in the desired logging framework at deployment time.
Last Release on Sep 3, 2023

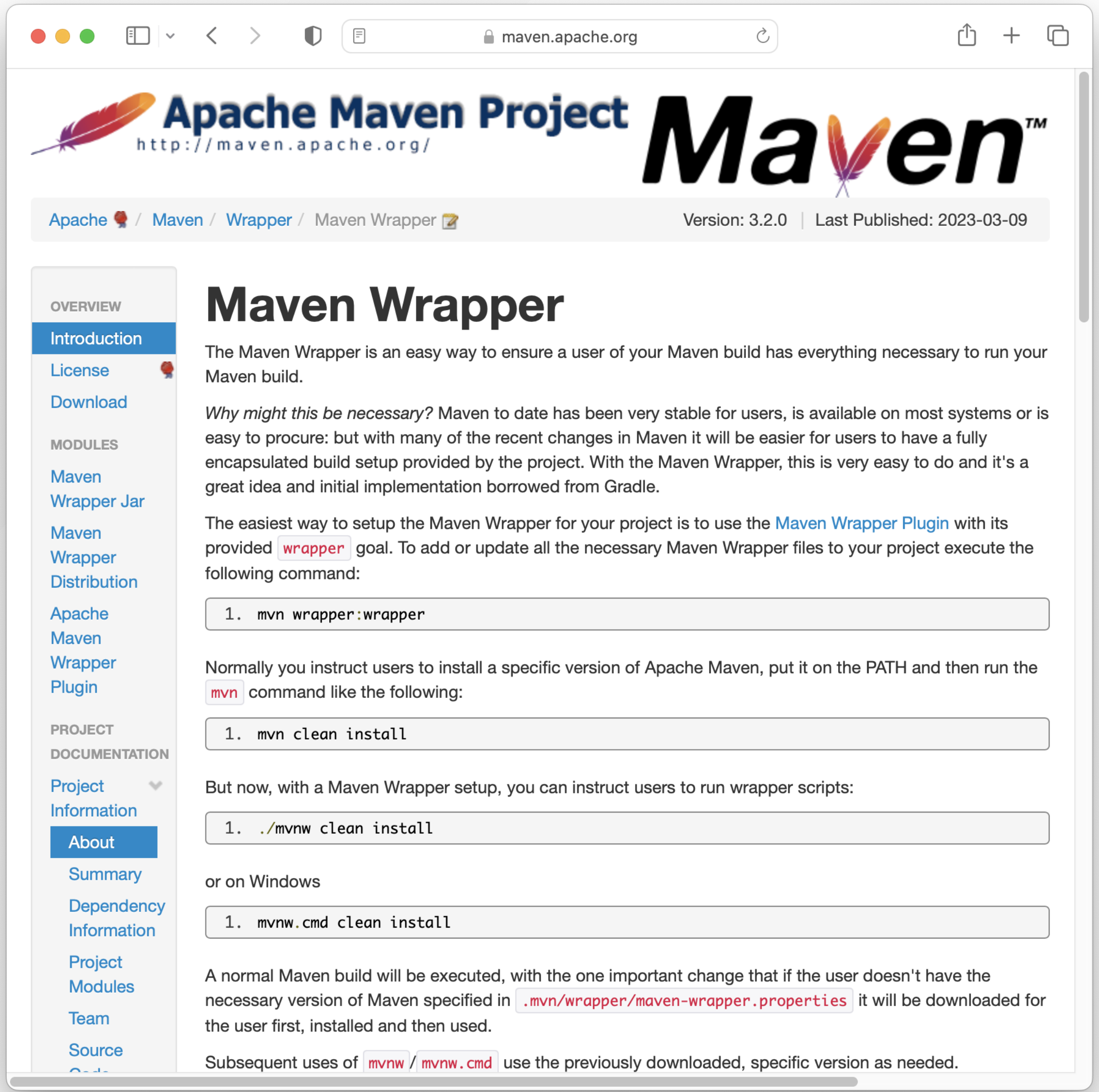
3. Kotlin Stdlib Common — 35,601 usages
org.jetbrains.kotlin » kotlin-stdlib-common — Apache
Kotlin Common Standard Library (legacy, use kotlin-stdlib instead)
Last Release on Sep 7, 2023

4. Guava: Google Core Libraries For Java — 35,294 usages
com.google.guava » guava — Apache
Guava is a suite of core and expanded libraries that include utility classes, Google's collections, I/O classes, and much more.
Last Release on Aug 1, 2023

5. Scala Library — 35,168 usages

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`



Maven Wrapper

The Maven Wrapper is an easy way to ensure a user of your Maven build has everything necessary to run your Maven build.

Why might this be necessary? Maven to date has been very stable for users, is available on most systems or is easy to procure: but with many of the recent changes in Maven it will be easier for users to have a fully encapsulated build setup provided by the project. With the Maven Wrapper, this is very easy to do and it's a great idea and initial implementation borrowed from Gradle.

The easiest way to setup the Maven Wrapper for your project is to use the Maven Wrapper Plugin with its provided `wrapper` goal. To add or update all the necessary Maven Wrapper files to your project execute the following command:

```
mvn wrapper:wrapper
```

Normally you instruct users to install a specific version of Apache Maven, put it on the PATH and then run the `mvn` command like the following:

```
mvn clean install
```

But now, with a Maven Wrapper setup, you can instruct users to run wrapper scripts:

```
./mvnw clean install
```

or on Windows

```
mvnw.cmd clean install
```

A normal Maven build will be executed, with the one important change that if the user doesn't have the necessary version of Maven specified in `.mvn/wrapper/maven-wrapper.properties` it will be downloaded for the user first, installed and then used.

Subsequent uses of `mvnw` / `mvnw.cmd` use the previously downloaded, specific version as needed.

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.

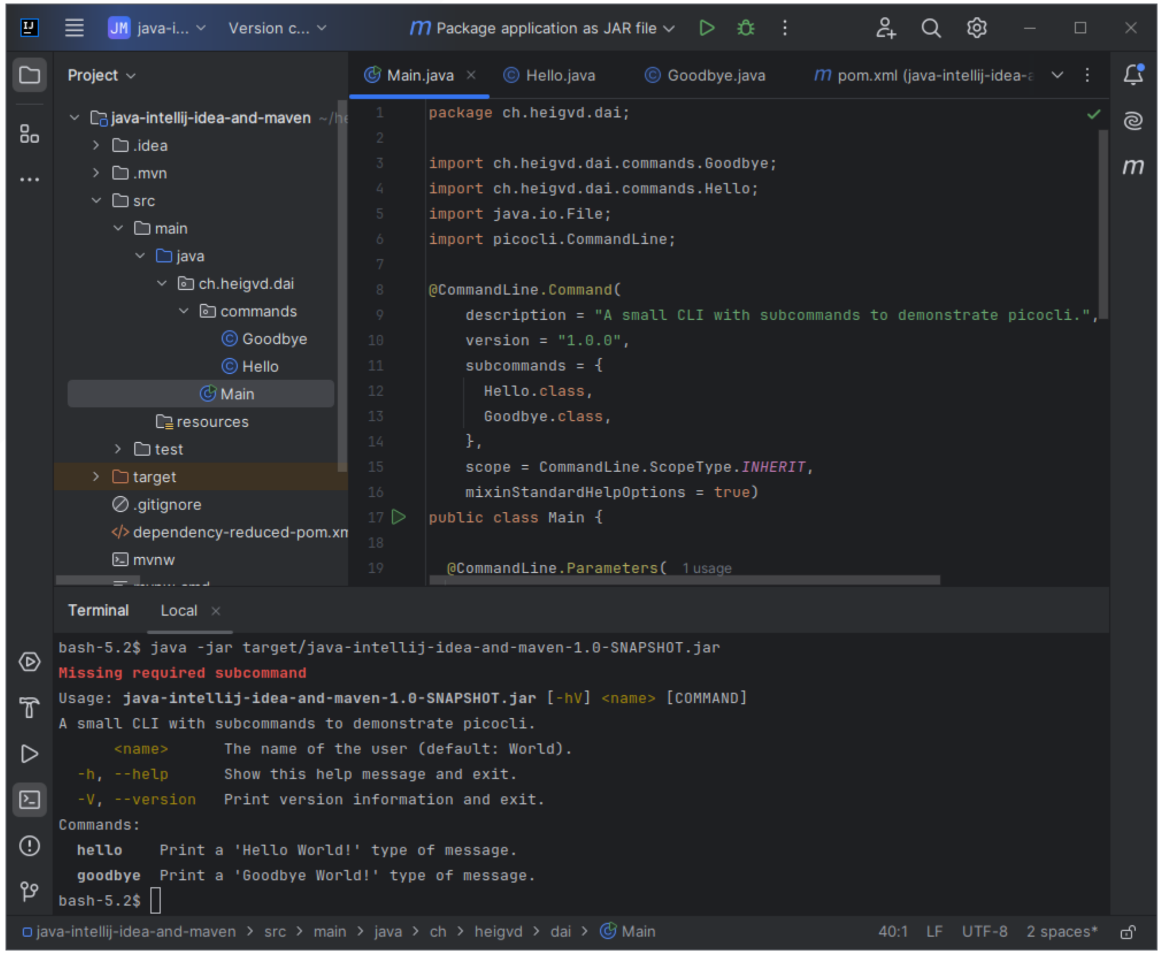
Questions

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


java-i...
Version c...
Package application as JAR file
Project
java-intellij-idea-and-maven
.idea
.mvn
src
main
java
ch.heigvd.dai
commands
Goodbye
Hello
Main
resources
test
target
.gitignore
dependency-reduced-pom.xml
mvnw
Main.java
Hello.java
Goodbye.java
pom.xml (java-intellij-idea-a
Terminal
Local
java-intellij-idea-and-maven > src > main > java > ch > heigvd > dai > Main
40:1 LF UTF-8 2 spaces*

```java
package ch.heigvd.dai;

import ch.heigvd.dai.commands.Goodbye;
import ch.heigvd.dai.commands.Hello;
import java.io.File;
import picocli.CommandLine;

@CommandLine.Command(
    description = "A small CLI with subcommands to demonstrate picocli.",
    version = "1.0.0",
    subcommands = {
      Hello.class,
      Goodbye.class,
    },
    scope = CommandLine.ScopeType.INHERIT,
    mixinStandardHelpOptions = true)
public class Main {

  @CommandLine.Parameters(  1 usage
```

```
bash-5.2$ java -jar target/java-intellij-idea-and-maven-1.0-SNAPSHOT.jar
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.

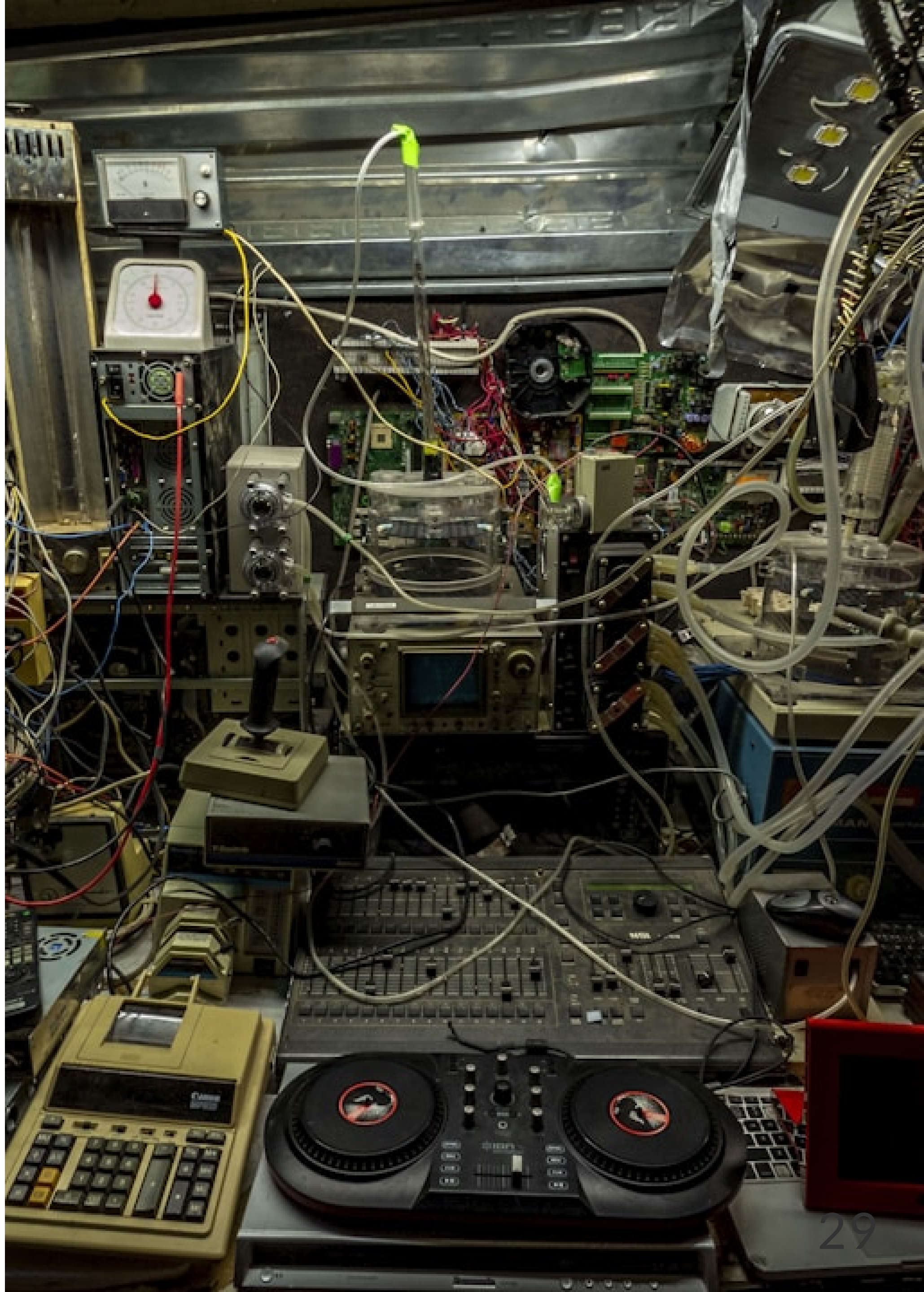
➡️ GitHub Discussions

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