Set-up a Project Using Maven

Prof. Dr. Peter Braun

17. Oktober 2024

1 Titlepage

Set-up a Project Using Maven

Prof. Dr. Peter Braun

thws Technical University of Applied Sciences Würzburg-Schweinfurt

No Text – maybe some music :-)

2 Learning Goals

Learning Goals

- You understand the reason for Maven
- You can set up a project using Maven
- You can configure Maven in your project
- You can build and test your project

Welcome to this unit about the build tool we will use in this course. Maven is a powerful build automation tool primarily used for Java and Kotlin programming languages. In this unit, we will show you how to install Maven on your computer and how to configure Maven for your projects. Some of you have previous knowledge about Maven. This unit will not be useful for you, and you can skip it if you like. However, in the next units, we expect you to know how to use Maven on the command line and not only in your editor. We expect you to understand how to manage dependencies and configure plugins for Maven.

3 Introduction to Maven

Introduction to Maven

- Maven is a build automation tool for Java and Kotlin
- It helps manage project dependencies
- Maven uses conventions to simplify project setup
- It automates tasks like compiling and testing
- Centralized repository (Maven Central) provides easy access to libraries and plugins

Maven is a build automation tool that simplifies build processes like code compilation, packaging code into executable files, testing the code, and managing the underlying dependencies. Having mentioned dependencies, let's spend a moment understanding them in the context of Maven. When developing a project, it's not uncommon to rely on numerous external libraries or modules. Manually managing these dependencies can rapidly turn into a complex pursuit. This is where Maven steps up, effectively managing and organizing these project dependencies, saving you from potential dependency hell. Let's dive a little deeper. Maven prefers conventions over configuration. It adopts a specific standard project structure that makes it easier to understand a project's setup. This convention-over-configuration aspect is often a double-edged sword. The rigid structure may seem restrictive, but on the flip side, it caters to a predictable and uniform setup, a benefit for new project team members. Part of the Maven eco-system is a centralized repository for all dependencies known as Maven Central. When I use the word "repository here, don't mix it up with Git repository. Maven repositories are places from where you can download dependencies. Most dependencies you must use in your projects are published on Maven Central. It is also easy to add new repositories to your project if required. In conclusion, Maven, with its well-defined default configurations, robust automation capabilities, and centralized repositories, makes project setup remarkably efficient and standardized.

4 Installing and Using Maven

Installing and Using Maven

- Download and install Maven
- https://maven.apache.org/

- Set environment variables: `MAVEN_HOME`
- Add `mvn` to `PATH` to run it on the command line

- Verify installation by running `mvn -v` in the terminal

Let's now move on to setting up Maven on your system. You first need to go to the official Maven website and download the latest version of Maven. You can see the URL beside me. Depending on your operating system, installing Maven might be done differently. On Linux, you can use "apt" for this. On macOS, you can use the Homebrew package manager. It doesn't matter how you install the software, but having Maven available on the command line at the end is necessary. For this, it might be required to include the folder where Maven was installed in the PATH environment variable. After you have installed and configured your environment, please check that you can execute "mvn -v" on the console.

5 Maven Project Object Model (POM)

Maven Project Object Model (POM)

- The POM file `pom.xml` is the core of a Maven project

- Project metadata like groupId, artifactId, and version
- Dependencies (libraries) required for the project
- Plugins and goals (e.g., compiling, testing)

Let's dive deeper into the heart of a Maven project: the Project Object Model, or as it's better known, the POM file, titled 'pom.xml'. As the foundation of any Maven project, this file is imperative in managing and successfully executing your projects. A key role of the POM file is to store project metadata. These include the "groupId," "artifactId," and "version" of your project. Here's what each means: the groupId identifies the project's group or organization, the artifactId specifies the project's name, and the version signifies your project's specific version or iteration. These pieces of information are vital as they uniquely identify each project in a repository. Another important content of the POM file is the dependencies, as already mentioned. To include a dependency, you must define "groupId," "artificatId" and "version" of the library you want to use. The Maven Central platform helps you to find this information. Next are the plugins and goals Maven can perform, like compiling your Java code or running unit tests. Plugins are customized tailoring tools that make Maven adaptable to many project needs. Goals refer to specified tasks that a plugin can execute. For instance, a "compile" goal within a Java plugin will instruct Maven to compile your Java source code. These goals must be aligned to the phases of the build process. But we talk about this a little bit later.

6 Create a POM file

Create a POM file

- Use `mvn archetype:generate` to create a `pom.xml`
- Use one of our templates for different project types
- `https://git.fiw.thws.de/backend-systems/templates`

In the pursuit of setting up a Maven project, one of the first steps you must undertake is to create a POM file. Creating a POM file is simpler than you might think, thanks to the Maven command, "mvn archetype:generate." Enter this command in your terminal to initialize the creation of a new POM file. Remember, running this instruction in the directory where you desire the project setup is important. Maven also allows you to choose from a wide array of templates optimized for different project types. These templates can save time by adopting standard configurations for various project forms. Thus, they give you a head start in the project setup process. You can find more information about this in the documentation. In addition, to make it easier for you, we have curated a small list of templates hosted on our GitLab Server in particular for this course. You can access these at the web address shown on the slide. It provides direct access to our templates repository, for example, for a plain Java project or an app using the Jakarta Persistency API. The templates contain all necessary dependencies and the latest version numbers.

7 Directory Structure

Directory Structure

- Maven projects follow a standardized structure
- Source code is in `src/main/java`
- Resources in `src/main/resources`
- Tests are placed in `src/test/java`
- Compiled files are stored in the `target` directory

One of the central characteristics of Maven projects is that they all follow a standardized structure. This means that irrespective of the specific task or code, you'll typically see similar organization of your files and folders. Let's start with the location of the source code. In a Maven project, your source code lives in the directory labeled 'src/main/java'. This is the heart of your project, where the essential code files reside. You'll often spend most of your development time in this particular area. Parallelly, you have a section designated for resources under 'src/main/resources.' What are these resources? Well, they could include your XML files, properties files, and configuration files—any non-code items that your project is expected to use or manage. Moving on, let's see where our test cases are placed. In Maven, your test files are saved in 'src/test/java.' Importantly, keeping your tests separated from your main source code enhances maintainability and presents a much clearer picture when troubleshooting. Lastly, we must touch upon the 'target' directory. After running a build, Maven stores the compiled files in this directory. It's essentially Maven's place for creating the final build artifacts. It's time to look at the first POM file in action.

8 Screencast: Example of a POM File

I would like to give you some first examples of POM, of so-called POM files, the configuration files for Maven projects. And on one of the earlier slides, you saw a link to a template Git repository. And we just want to look into one or two of these templates here. So we start with the plain Java template. And we have a look at the POM file here. This is the smallest POM file you can imagine. The important points here are, first of all, these three lines, the group ID, the artifact ID, and the version number. So whenever you use one of our templates here for your own projects, please be sure to modify these names here. So you can just slightly modify the group ID. Maybe you remove this word templates as the last word here and replace this by some project name or so. And the artifact ID also should be replaced. And of course, the version number. Name and URL, you can just delete this. This is not so important. Then we have here this properties area. As example, we have defined three properties here, which are like system-wide properties that are used by other plugins now. You can also use this area, this section of the POM file to define your own properties. For example, this is very, very common for version numbers. I show you an example later. Then the second important block is the dependencies block. And there one dependency always should be defined like this. So define a group ID, define an artifact ID, and a version number in this scope tag here. This is meant to define, is it for production or for testing code? And the last block defines all the plugins that extend the standard lifecycle. So here we just define the Maven compiler plugin to let the Java code be compiled. So as I said, this was the most simple one. And we go and look into one more. In one of the later units, you learned about gRPC. So let's go to the gRPC template. This is more complex, definitely. And here we start again with the group ID, artifact ID, version numbers. Here you have the properties. And this, for example, now is an example for a property that you define by yourself. And then

later, you can use this, for example, here in line 34, you use this as a variable name, so to speak, for all the other dependencies that should have the same version number here. So the list of dependencies is much longer here. And also the build process here is much longer. First we define an extension. I think I will explain this in the unit about gRPC. Then we have here, that's the same Maven Java plugin. But here we have the protobuf plugin. What protobuf is will be explained in the next unit. And a lot of configuration data here. Here for example, we define in which phase of the lifecycle we are hooked into. So you see this is much longer. We have even more longer templates when it comes to using Docker, for example. This is really a long pomp file.

9 Dependencies

Dependencies

- Dependencies are libraries your project needs
- Defined inside the `<dependencies>` section
- Scope controls when dependencies are included

- Maven automatically downloads them
- Maven keeps a local copy in `$HOME/.m2`

We already mentioned that the core part of the POM file is the dependencies section. In addition to "artificatId," groupId, and version number, a "scope" element is used to inform Maven if the dependency is necessary for production mode or test cases. Maven downloads these dependencies on your computer to save bandwidth. When you increase the version number of a dependency and start the build process, the new version is downloaded, of course. All dependencies are stored locally in a hidden directory named "m2" in your home directory. In rare cases, when Maven has a hick-up, it might help to delete all dependencies locally by deleting the files in this "m2" folder. But this is only an option if nothing else helped. You might sometimes see the word "SNAPSHOT" in capital letters in a version number. This indicates a project that the developers updated without increasing the version number. This happens when a project is actively under development. This should mainly only happen with internal dependencies within one organization. Because Maven does not automatically update dependencies if the version number has not changed, you need to tell Maven using the "-U" option on the command line to update these snapshots.

10 Phases of the Build Process

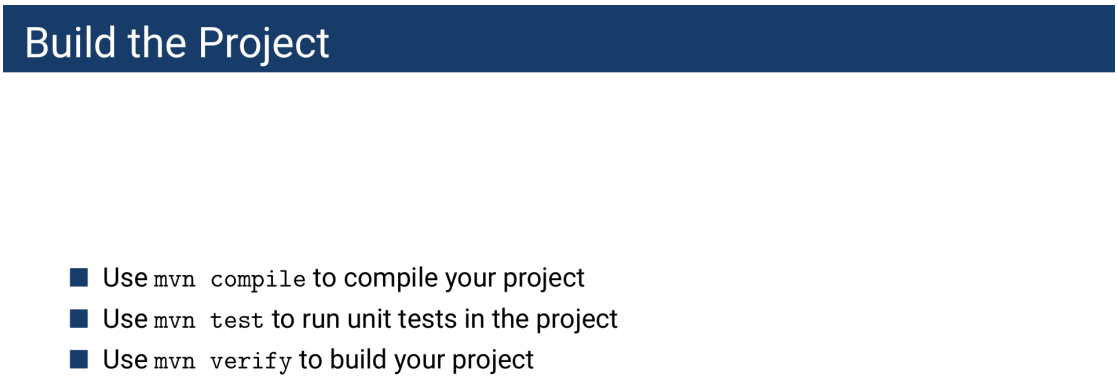
Phases of the Build Process

- The build lifecycle consists of multiple phases
- Validate, compile, test, package, install, deploy

- Each phase performs a specific task
- Phases are executed sequentially by default
- Custom build steps can be added via plugins

Moving on to our next topic, let's delve into the phases of the build process. The build lifecycle is the most important part of the Maven project setup. It involves a sequence of various phases, each with a significant role to play. Starting with "validate," our project is validated to ensure all necessary information is available. An erroneous or missing information can abort the whole process at this stage. Next is the "compile" phase. Here, the project's source code is compiled. If there are any errors in your code, they will be discovered during this phase. This leads us to the "test" phase. All the project's unit tests are run to verify that the application's business logic produces the expected results. The "package" phase comes next, where the compiled code is packed into a distributable format like a JAR or WAR file. Whereas "install" phase installs the package into the local repository, that means in your local "m2" folder, making it available as a dependency in other local projects. The "deploy" phase concludes the general phase sequence. This wraps up our lifecycle by copying the final package to the remote repository, making it accessible to other developers and projects. This list of phases is by far not complete. Another important aspect is that these phases are executed sequentially by default. This means that Maven runs the phases from top to bottom in the order mentioned here. Remember, customization is possible in this process with the use of plugins. Plugins present us with an option to add custom build steps, catering to any project-specific needs that may arise.

11 Build the Project



You can now start Maven by giving the name of the phase as a parameter. For example, on the slide beside me, we call Maven with the "Compile" phase. That means all phases before the compile phase, and the compile phase itself will be executed. When you call Maven without a phase as a parameter, it will terminate with an error unless you have defined some default values in the POM file. The "test" phase compiles the code and runs all unit test cases. The "verify" phase is one of the last in the build process, executing integration tests. How unit and integration tests are distinguished is a matter of configuration in the POM file. It's fair to use "test" or "verify" using your development cycle.

12 Summary

Summary

- Maven is one of the most-used build tools
- Defines the build process and dependencies
- Maven is highly customizable using plugins

That's it for today. The goal of this short video was to introduce you to Maven, one of the most used build tools in the software industry. We will use Maven as a build tool in this course, and you must be able to set up a project from scratch using Maven archetypes or one of our templates. When you continue to develop larger software systems, you might need to add dependencies and plugins for further functionality. It would be best if you also understood the difference between Maven as part of your IDE and Maven as a stand-alone tool that can be used on the command line. I prefer Maven on the command line because I have full control over which Maven version is used, which Java version is used, and so on.