CODEGYM

Mentor-supported training program from CodeGym

Java Developer in 12 months

MODULE 3. JAVA PROFESSIONAL

Lesson #1 Maven: phases, plugins, dependencies, and build

Lesson plan

- Introduction to Maven
- Object model of project description
- Structure of a Maven project
- Dependencies in a Maven project
- Maven project phases
- Maven plugins
- Maven properties
- Building a Maven project

About this module

This module serves as a foundation for the fifth module on Spring and provides a general understanding of how servers work on the internet, the development process, and so on.

In this module, there will be three types of assignments:

1. **Tasks** (similar to the ones in the first two modules).
2. **Quizzes** (theory-based).
3. **Projects** — Projects are the most important part of this module.

In this module, there will be four projects based on the following topics: Maven, testing and logging, frontend development, and JSP and servlets. Additionally, there will be a final project that covers the entire module.

The projects in this module are divided into two types: **assignments and tutorials**. For each project, there will be a recorded video breakdown available, which provides a detailed explanation and guidance on completing the project.

Large-scale programs

As programs grew in size, developers encountered two new circumstances:

- A large number of people work on a single program.
- There is no individual who knows the entire codebase of the program.

To address these challenges, developers adopted a technical approach of breaking programs into smaller parts: **libraries and modules**. Each module served as a small building block that could be combined to create larger projects.

Libraries, on the other hand, are universal components that can be used in different programs. They provide reusable functionality and can be shared across multiple projects.

Introduction to Maven™

Maven is a specialized framework for project management and build automation. It standardizes three main aspects:

- Project description;
- Project build scripts;
- Dependencies between libraries.

Ant was the predecessor of Maven, and **Gradle** is its successor

CODEGYM

Maven download and installation

Maven has an official website at maven.apache.org. There is a wealth of documentation available on the website, so if you encounter any difficulties or have additional questions, feel free to visit it.

Maven is written in Java and requires JRE **version 7** or higher. Additionally, you need to set the environment variables such as **JAVA_HOME** to ensure proper configuration.

CODEGYM

Maven local repository

The special folder where Maven stores the **jar libraries** it uses during project builds is called the local Maven repository.

If this folder is not specified, Maven will create it in the home directory of the current user.

Important! Do not place Maven in system folders as it requires write permissions in those folders, which can raise concerns from antivirus software or the operating system.

CODEGYM

Object model of project description

Maven introduced a universal open standard based on XML, where various tags are used to describe the project, its build process, and its dependencies.

The project description is encapsulated in a single file, typically named **pom.xml**.

CODEGYM

Example of a pom.xml file

```xml
<?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>example.com</groupId>
  <artifactId>example</artifactId>
  <version>1.0-SNAPSHOT</version>

  <dependencies>
     <dependency>
        <groupId>commons-io</groupId>
        <artifactId>commons-io</artifactId>
      <version>2.6</version>
      </dependency>
  </dependencies>

</project>
```

In this example, three things are specified:

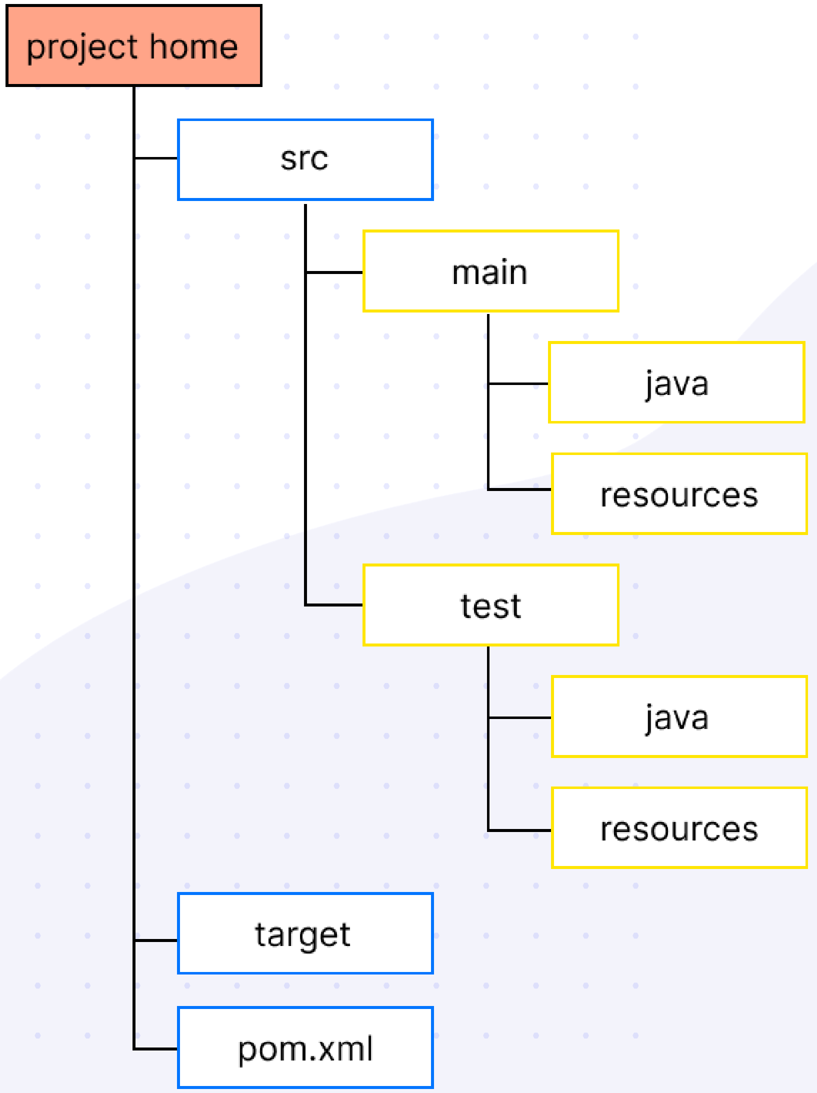
- Maven project version information - in blue.
- Project information itself - in red.
- Information about the libraries being used - in green.

CODEGYM

Structure of a Maven project

Maven standardized the structure of a project.

There are several options for organizing code within a project, and the most common one looks like this:



How IDEA works with Maven

IntelliJ IDEA works seamlessly with Maven.

It can open Maven projects, automatically create and execute various build scenarios, and has excellent support for handling dependencies.

IntelliJ IDEA even has its own built-in Maven, but you still need to know how to install and configure it yourself. This feature was not mentioned earlier as it requires additional setup.

Introduction to archetypes

There is another way to create a Maven project in IntelliJ IDEA - based on an archetype.

Maven has standardized project templates, known as archetypes.

You can obtain a list of available archetypes by executing the following command in the console:
mvn archetype:generate

The most popular archetypes are:

- **maven-archetype-quickstart;**
- **maven-archetype-site;**
- **maven-archetype-webapp;**
- **maven-archetype-j2ee-simple;**
- **jpa-maven-archetype;**
- **spring-mvc-quickstart.**

CODEGYM

Dependencies in a Maven project

If you want to add a library to your **Maven project**, you simply need to add it to the **pom.xml file** under the **dependencies** section.

```
<dependencies>


<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-core</artifactId>
   <version>5.3.18</version>
</dependency>


<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-core</artifactId>
   <version>6.0.0.Final</version>
</dependency>


<dependencies>
```

How to search for libraries in the Maven Repository

On the internet, there is a central public **Maven repository** that stores millions of libraries.

It can be found at the following link: [https://mvnrepository.com/,](https://mvnrepository.com/)
You can search for the desired library directly on this website.

dependency repository

If you decide to add a third-party repository to your project, you can do it just as easily as adding dependencies:

```
<repositories>
  <repository>
    <id>public-javarush-repo</id>
    <name>Public JavaRush Repository</name>
    <url>http://maven.javarush.com</url>
  </repository>

  <repository>
    <id>private-javarush-repo</id>
    <name>Private JavaRush Repository</name>
    <url>http://maven2.javarush.com</url>
  </repository>
 </repositories>
```

Each repository has three components: **Key/ID, Name**, and **URL**. The Name can be any name you choose, primarily for your convenience. The ID is also for internal purposes. In practice, you only need to specify the URL when adding a repository.

Maven project phases

The entire project build process has been divided into phases, and their descriptions are provided in the table below:

#	Phase	
1	**validate**	validates the correctness of the project's metadata
2	**compile**	compile the source code of the project
3	**test**	runs tests for the classes from the previous step
4	**package**	packages the compiled classes into a new artifact, such as a JAR, WAR, ZIP, etc.
5	**verify**	verifies the correctness of the artifact and ensures it meets quality requirements
6	**install**	installs the artifact into the local repository
7	**deploy**	deploys the artifact to the production server or a remote repository

These steps are executed in a clear sequential order.

Project build

If you want to compile the project, you need to execute the compile phase. This can be done using the command line: **mvn compile** or through the IntelliJ IDEA interface by clicking on the "**compile**" option.

After that, Maven will initiate the project build, and you will see a build process log similar to this:

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 0.742 s
[INFO] Finished at: 2016-09-19T22:41:26+04:00
[INFO] Final Memory: 7M/18M
[INFO] ------------------------------------------------------------------------
```

Build lifecycles

All Maven commands are divided into three groups called lifecycles. They are called lifecycles because they define the order of phases that are executed during the build or a specific lifecycle, as not all Maven actions are related to the build process.

There are three lifecycles in total:

- **clean;**
- **default;**
- **site.**

Each of them has its own sequence of phases.

Maven plugins

Standard lifecycles can be extended with additional functionality using Maven plugins. Plugins allow you to insert new steps into the standard lifecycle (e.g., deploying to an application server) or extend existing steps.

Since plugins are artifacts just like dependencies, they are described in a similar way. Instead of the "dependencies" section, we use "plugins"; instead of "dependency," we use "plugin"; instead of "repositories," we use "pluginRepositories"; and instead of "repository," we use "pluginRepository."

```
<plugins>
   <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-checkstyle-plugin</artifactId>
      <version>2.6</version>
   </plugin>
</plugins>
```

Goals in Maven

In Maven, there is another concept called "goal."
A goal represents a specific task to be executed by Maven. The main goals align with the primary phases:

- **validate;**
- **compile;**
- **test;**
- **package;**
- **verify;**
- **install;**
- **deploy.**

During each phase of the project's lifecycle, a specific plugin (a JAR library) is invoked, which includes a set of goals to be executed.

CODEGYM

Maven properties

Frequently encountered parameters in Maven can be extracted and stored in variables. For example, you can use variables to store the Java version, library versions, or paths to specific resources.

To achieve this, there is a dedicated section in the **pom.xml** called **<properties>**, where variables are declared. The general format of a variable is as follows:

```
<variable-name>value</variable-name>
```

To reference variables, a different syntax is used:

```
${variable-name}
```

Wherever this code is written, Maven will substitute the value of the variable.

Predefined variables in Maven

When describing a project in the pom.xml file, you can use predefined variables. These variables can be roughly categorized into several groups:

- Built-in project properties;
- Project properties;
- Settings.

There are only two built-in project properties:

Property	Description
${basedir}	The root directory of the project where the pom.xml file is located

${version}	Artifact version;
	you can use ${project.version} or ${pom.version}

Predefined variables in Maven

Project properties can be referenced using the prefixes «**project**» or «**pom**».

We have four of them:

Property	Description
${project.build.directory}	"target" directory of the project
${project.build.outputDirectory}	"target" directory of the compiler. By default, it is "target/classes"
${project.name}	Project name
${project.version}	Project version

Access to properties in the settings.xml file can be obtained using the prefix "**settings**"

CODEGYM

Building a Maven project

The project structure is described in the pom.xml file, which should be located in the project's root folder.

Not all sections need to be present in the **pom.xml** description. Sections like **properties** and **repositories** are often not used. Parameters describing the current project are mandatory.

The build section is not mandatory - Maven can build the project even without it.

Homework

MODULE 3. JAVA PROFESSIONAL

Level 1 Maven: phases, plugins, dependencies, and build

MODULE 3. JAVA PROFESSIONAL

CODEGYM

Answers to questions

MODULE 3. JAVA PROFESSIONAL