Apache Maven

Sergio Di Meglio
sergio.dimeglio@unina.it

Nov 17,2023

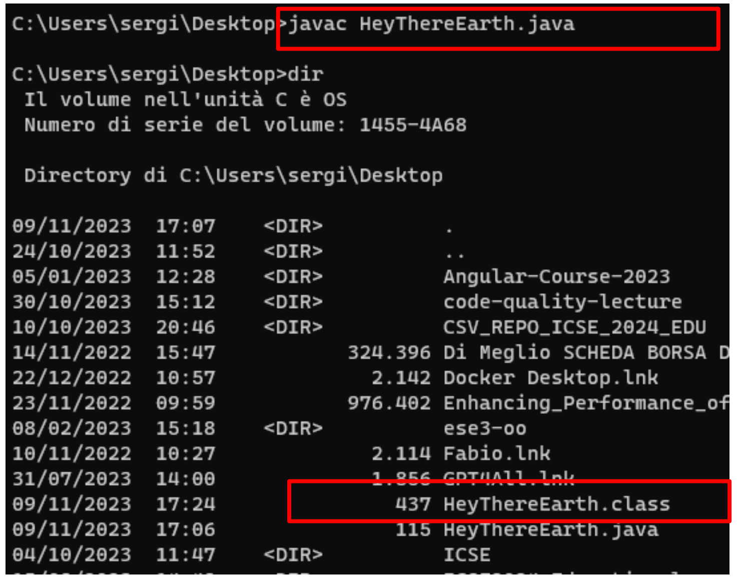
Software Engineering

What is Build?

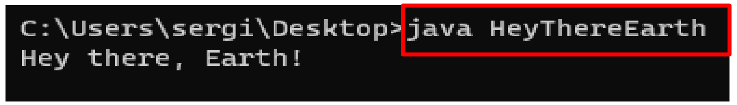
- This process aims to **compile** the source code, **resolves** depedencies, and **generates** the desired output, such as executable files or libraries.

HeyThereEarth.java

```
class HeyThereEarth{
		public static void main(String[] args) {
				System.out.println("Hey there, Earth!");
		}
}
```



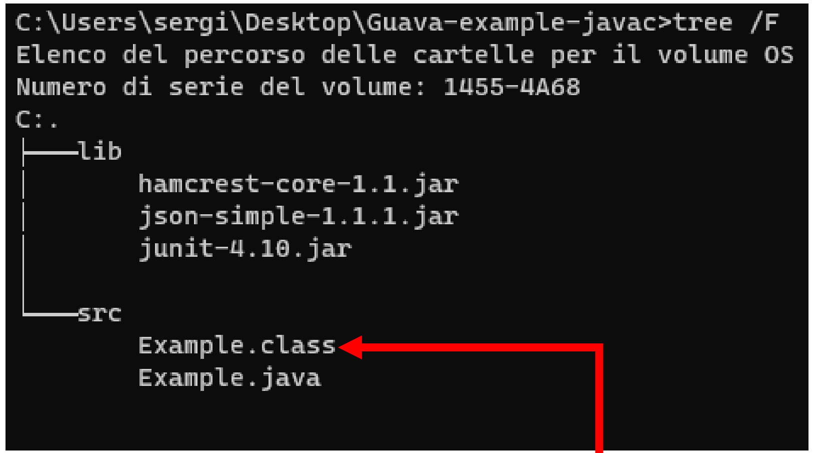
```
C:\Users\sergi\Desktop>javac HeyThereEarth.java

C:\Users\sergi\Desktop>dir
 Il volume nell'unità C è OS
 Numero di serie del volume: 1455-4A68

 Directory di C:\Users\sergi\Desktop

09/11/2023  17:07    <DIR>          .
24/10/2023  11:52    <DIR>          ..
05/01/2023  12:28    <DIR>          Angular-Course-2023
30/10/2023  15:12    <DIR>          code-quality-lecture
10/10/2023  20:46    <DIR>          CSV_REPO_ICSE_2024_EDU
14/11/2022  15:47           324.396 Di Meglio SCHEDA BORSA D
22/12/2022  10:57             2.142 Docker Desktop.lnk
23/11/2022  09:59           976.402 Enhancing_Performance_of
08/02/2023  15:18    <DIR>          ese3-oo
10/11/2022  10:27             2.114 Fabio.lnk
31/07/2023  14:00             1.856 GPT4All.lnk
09/11/2023  17:24               437 HeyThereEarth.class
09/11/2023  17:06               115 HeyThereEarth.java
04/10/2023  11:47    <DIR>          ICSE
```



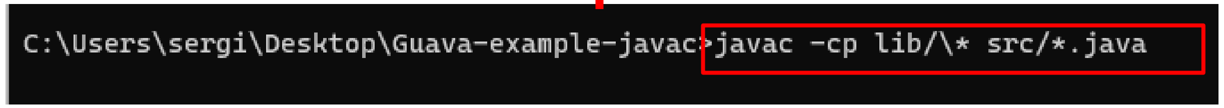
```
C:\Users\sergi\Desktop>java HeyThereEarth
Hey there, Earth!
```

Build Java files with dependencies

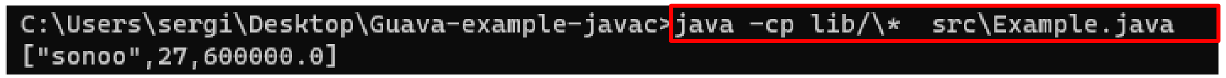
- It's very common that java program depends on **one or more** external libraries (jar files).



```
C:\Users\sergi\Desktop\Guava-example-javac>tree /F
Elenco del percorso delle cartelle per il volume OS
Numero di serie del volume: 1455-4A68
C:.
├───lib
│       hamcrest-core-1.1.jar
│       json-simple-1.1.1.jar
│       junit-4.10.jar
│
└───src
        Example.class
        Example.java
```



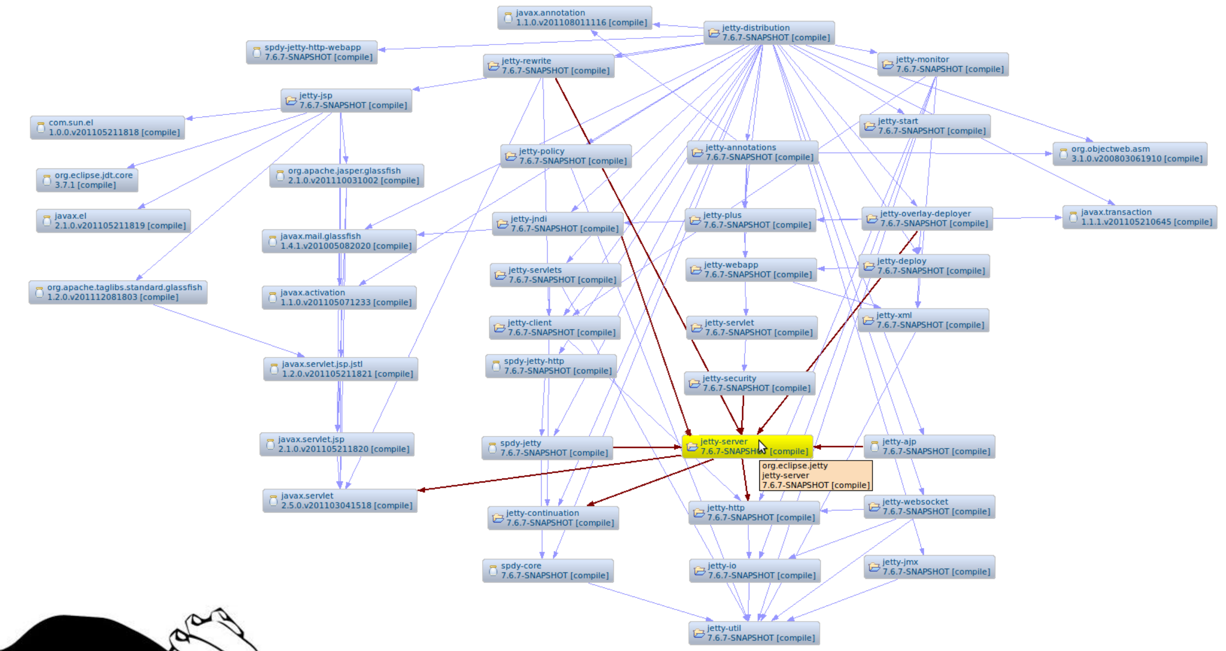
```
C:\Users\sergi\Desktop\Guava-example-javac>javac -cp lib/\* src/*.java
```

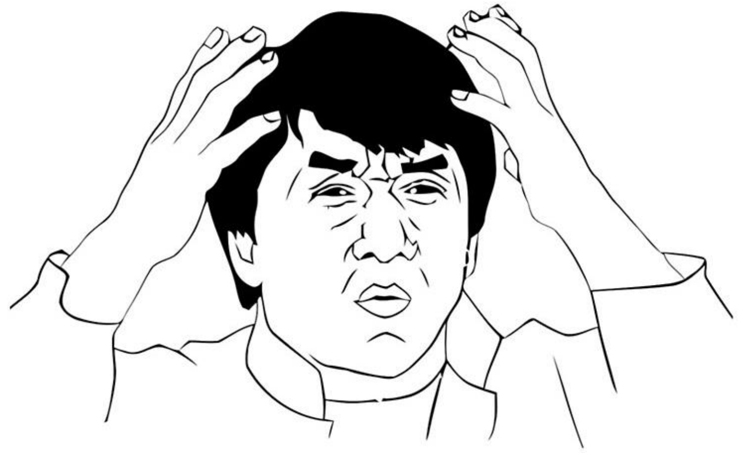


```
C:\Users\sergi\Desktop\Guava-example-javac>java -cp lib/\*  src\Example.java
["sonoo",27,600000.0]
```

How to handle transitivity dependencies?

- Industrial projects depend on a number of libraries.
- The downloaded jar file may depend on other different libraries.
- JARs sharing among teams can be **challenging**.


javax.annotation 1.1.0.v201108011116 [compile]
jetty-distribution 7.6.7-SNAPSHOT [compile]
spdy-jetty-http-webapp 7.6.7-SNAPSHOT [compile]
jetty-rewrite 7.6.7-SNAPSHOT [compile]
jetty-monitor 7.6.7-SNAPSHOT [compile]
jetty-jsp 7.6.7-SNAPSHOT [compile]
com.sun.el 1.0.0.v201105211818 [compile]
jetty-start 7.6.7-SNAPSHOT [compile]
org.eclipse.jdt.core 3.7.1 [compile]
org.apache.jasper.glassfish 2.1.0.v201110031002 [compile]
jetty-policy 7.6.7-SNAPSHOT [compile]
jetty-annotations 7.6.7-SNAPSHOT [compile]
org.objectweb.asm 3.1.0.v200803061910 [compile]
javax.el 2.1.0.v201105211819 [compile]
jetty-jndi 7.6.7-SNAPSHOT [compile]
jetty-plus 7.6.7-SNAPSHOT [compile]
jetty-overlay-deployer 7.6.7-SNAPSHOT [compile]
javax.transaction 1.1.1.v201105210645 [compile]
javax.mail.glassfish 1.4.1.v201005082020 [compile]
jetty-servlets 7.6.7-SNAPSHOT [compile]
jetty-webapp 7.6.7-SNAPSHOT [compile]
jetty-deploy 7.6.7-SNAPSHOT [compile]
org.apache.taglibs.standard.glassfish 1.2.0.v201112081803 [compile]
javax.activation 1.1.0.v201105071233 [compile]
jetty-client 7.6.7-SNAPSHOT [compile]
jetty-servlet 7.6.7-SNAPSHOT [compile]
jetty-xml 7.6.7-SNAPSHOT [compile]
javax.servlet.jsp.jstl 1.2.0.v201105211821 [compile]
spdy-jetty-http 7.6.7-SNAPSHOT [compile]
jetty-security 7.6.7-SNAPSHOT [compile]
javax.servlet.jsp 2.1.0.v201105211820 [compile]
spdy-jetty 7.6.7-SNAPSHOT [compile]
jetty-server 7.6.7-SNAPSHOT [compile]
jetty-ajp 7.6.7-SNAPSHOT [compile]
org.eclipse.jetty jetty-server 7.6.7-SNAPSHOT [compile]
javax.servlet 2.5.0.v201103041518 [compile]
jetty-continuation 7.6.7-SNAPSHOT [compile]
jetty-http 7.6.7-SNAPSHOT [compile]
jetty-websocket 7.6.7-SNAPSHOT [compile]
spdy-core 7.6.7-SNAPSHOT [compile]
jetty-io 7.6.7-SNAPSHOT [compile]
jetty-jmx 7.6.7-SNAPSHOT [compile]
jetty-util 7.6.7-SNAPSHOT [compile]



What is Maven?

- Apache Maven is a **project management** and **comprehension** tool that provides developers a complete build lifecycle framework, and as such provides a way to help with managing:
    - Builds
    - Documentation
    - Reporting
    - Dependencies
    - Releases
    - Distribution

Objectives

- Maven primary goal is to provide developer:
  - A comprehensive model project which is **reusable**, **maintainable**, and **easier** to comprehend.
  - Plugins or tools that interact with this declarative model.
- Maven project structure and contents are declared in an xml file named POM.

Maven ArcheType

- Maven provides users, a very large list of different types of project templates using concept of **Archetype.**
- Maven helps users to quickly start a new project using following command : **mvn archetype:generate**
- **What is Archetype?**
    - Archetype is a Maven plugin whose task is to create a project structure as per its template. We are going to use quickstart archetype plugin to create a simple java application here.

Maven ArcheType Example

- **maven-archetype-quickstart** is an archetype which generates a sample Maven project.

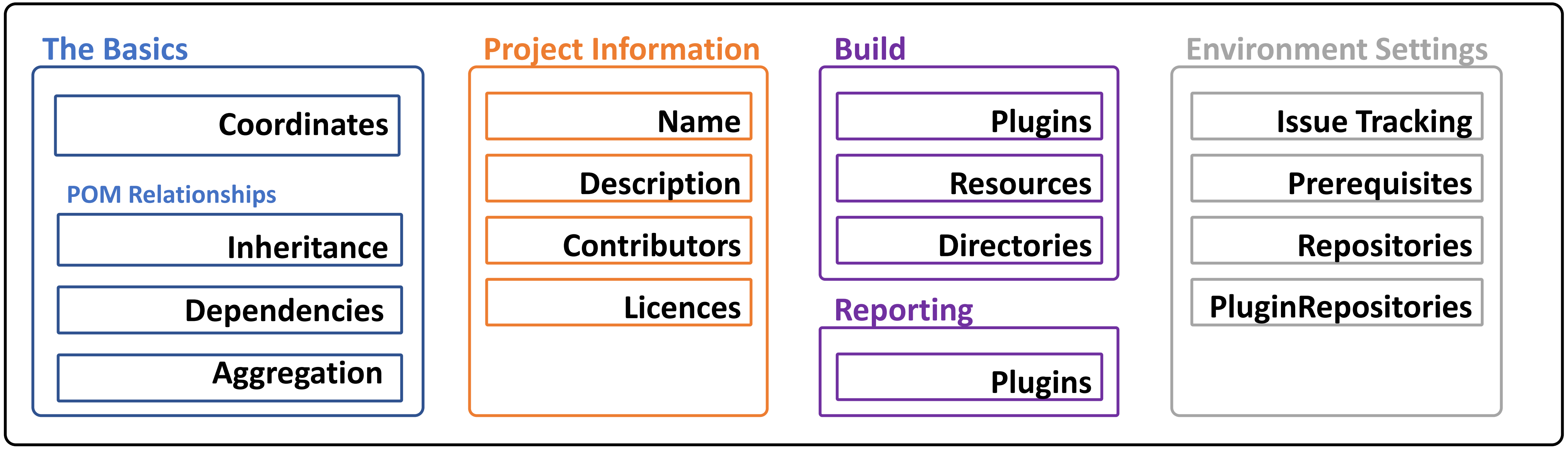
```
project
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- $package
    |           `-- App.java
    `-- test
        `-- java
            `-- $package
                `-- AppTest.java
```

- To generate a new project from this archetype, type:

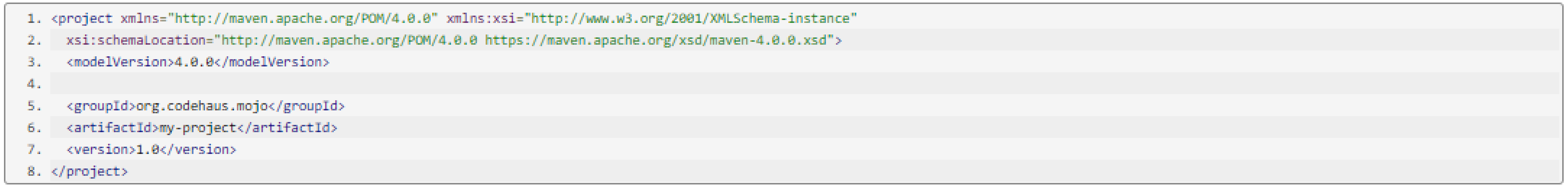
```
mvn archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4
```

The Project Object Model (POM)

- POM is **fundamental unit** of work in Maven.
- It is an XML file that contains information about the project and various configuration detail.



POM – The Basics



```xml
<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.codehaus.mojo</groupId>
  <artifactId>my-project</artifactId>
  <version>1.0</version>
</project>
```

- POM defined above is the **bare minimum** that Maven allows.

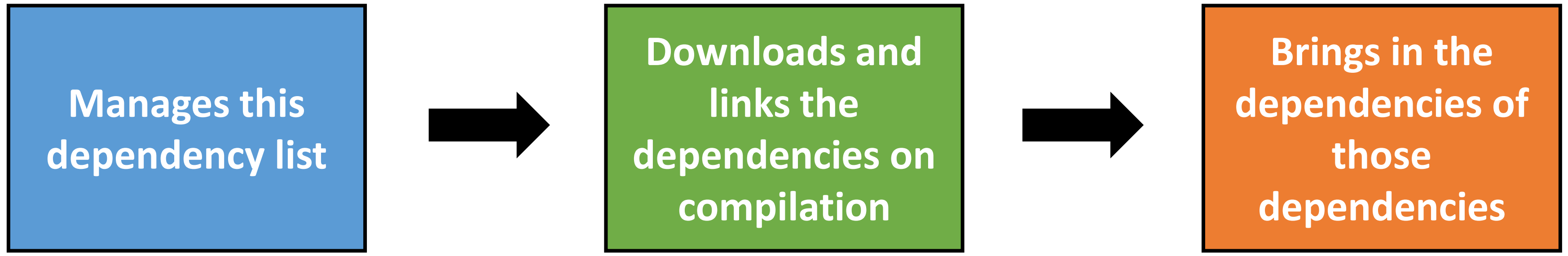
MAVEN COORDINATES

- **groupId:** is generally unique amongst an organization or a project.
- **artifactId:** is generally the name that the project is known by.
- **version:** is the last piece of the naming puzzle; groupId:artifactId denotes a single project but they cannot delineate which incarnation of that project we are talking about.

groupId:artifactId:version → $M2_REPO/org/codehause/mojo/my-project/1.0

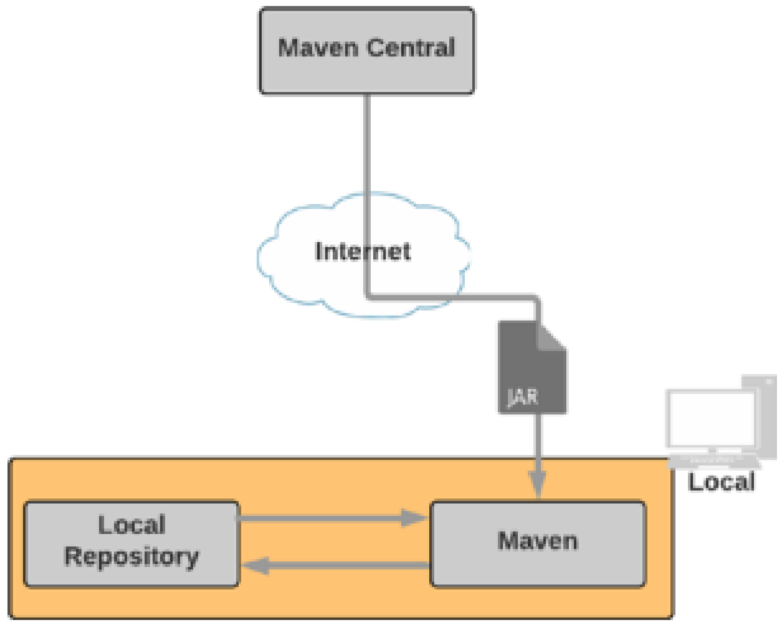
POM – Relationships

- One powerful aspect of Maven is its handling of project relationships: this includes depedencies, inheritance, and aggregation.
- Most projects depend on others to build and run correctly.
- Maven is able to:


Manages this dependency list
Downloads and links the dependencies on compilation
Brings in the dependencies of those dependencies

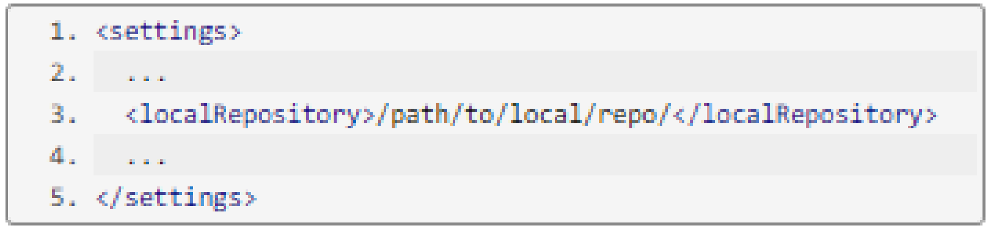
What is Maven Repository?

- In Maven terminology, a repository is a place i.e. directory where all the project jars, library jar, plugins or any other project specific artifacts are stored and can be used by Maven easily.
- Maven repository are of two types:
  - Local
  - Remote



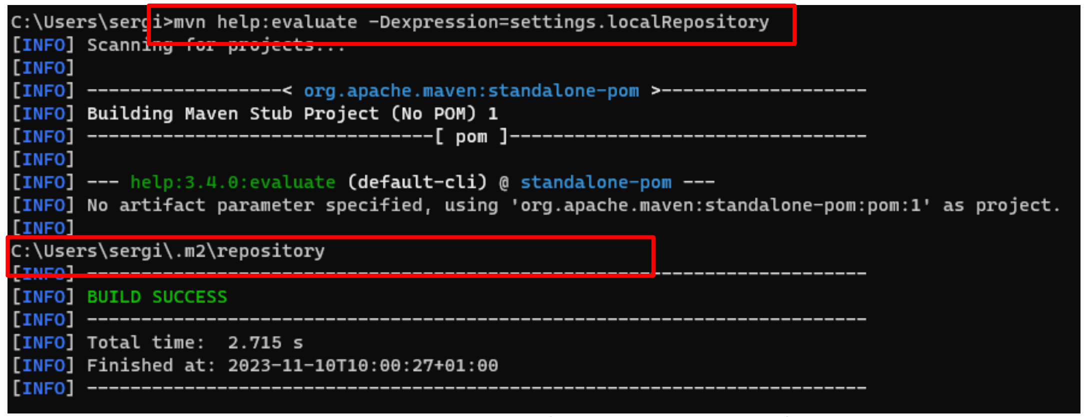
Local Repository

- Maven local repository is a folder location on your machine. It gets created when you run any maven command for the first time.
- The location of your local repository can be changed in your user configuration. The default values is *${user.home}/.m2/repository/*



```
<settings>
  ...
  <localRepository>/path/to/local/repo/</localRepository>
  ...
</settings>
```

Change the location of the local repo within the pom



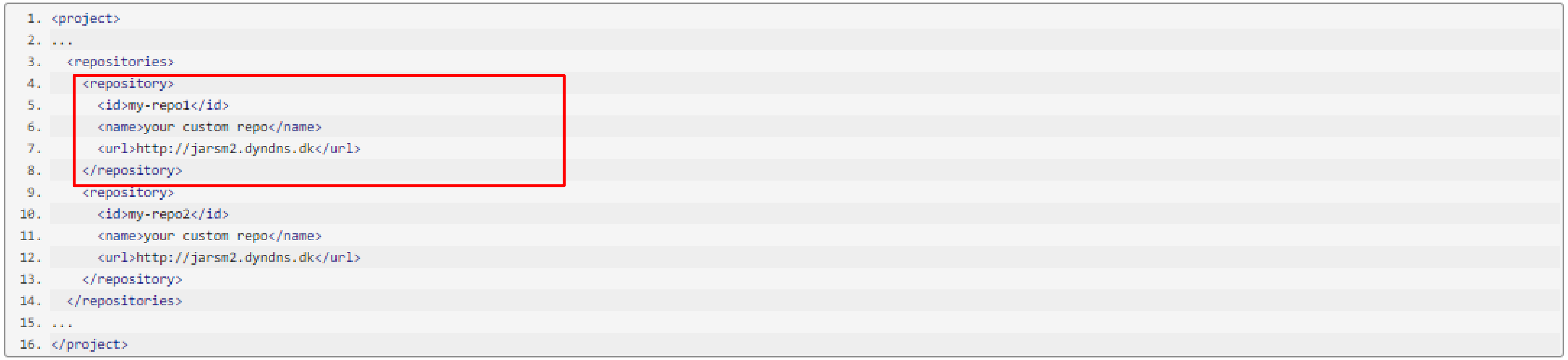
```
C:\Users\sergi>mvn help:evaluate -Dexpression=settings.localRepository
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------< org.apache.maven:standalone-pom >-------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] --------------------------------[ pom ]---------------------------------
[INFO]
[INFO] --- help:3.4.0:evaluate (default-cli) @ standalone-pom ---
[INFO] No artifact parameter specified, using 'org.apache.maven:standalone-pom:pom:1' as project.
[INFO]
C:\Users\sergi\.m2\repository
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.715 s
[INFO] Finished at: 2023-11-10T10:00:27+01:00
[INFO] ------------------------------------------------------------------------
```

Show current location of the local repo from CLI

Central Repository

- Maven central repository is provided by Maven community. It contains a large number of commonly used libraries.
- When Maven does not find any dependency in local repository, it starts searching in central repository using following URL: https://repo.maven.apache.org/maven2/
- Key concepts of Central repository
  - This repository is managed by Maven community
  - It is not required to be configured
  - It requires internet access to be searched

Setting up Multiple Repositories

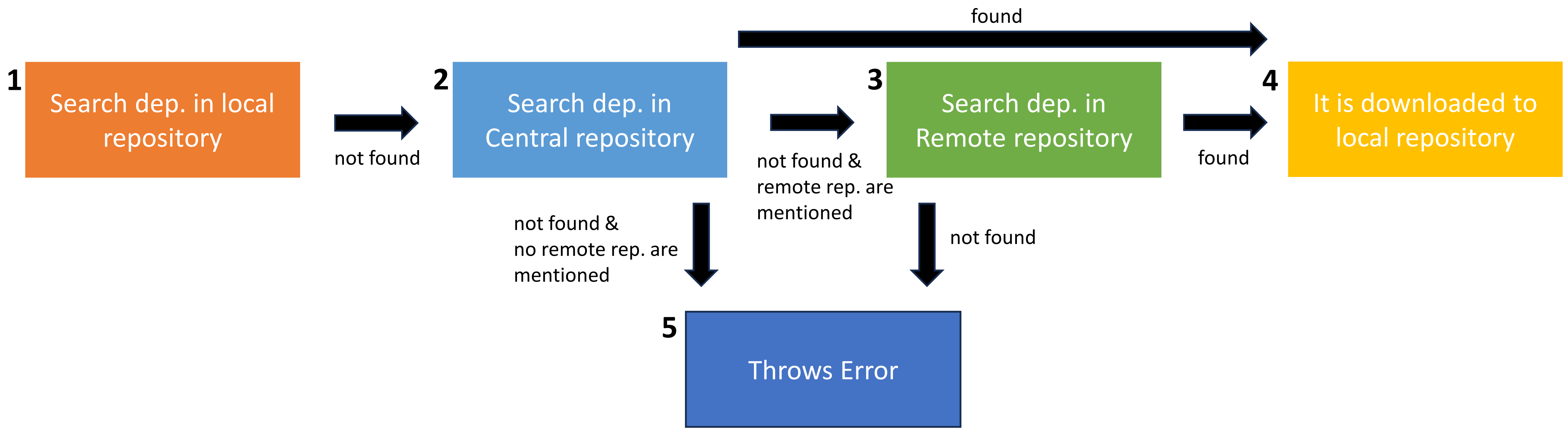


```
<project>
...
  <repositories>
    <repository>
      <id>my-repo1</id>
      <name>your custom repo</name>
      <url>http://jarsm2.dyndns.dk</url>
    </repository>
    <repository>
      <id>my-repo2</id>
      <name>your custom repo</name>
      <url>http://jarsm2.dyndns.dk</url>
    </repository>
  </repositories>
...
</project>
```

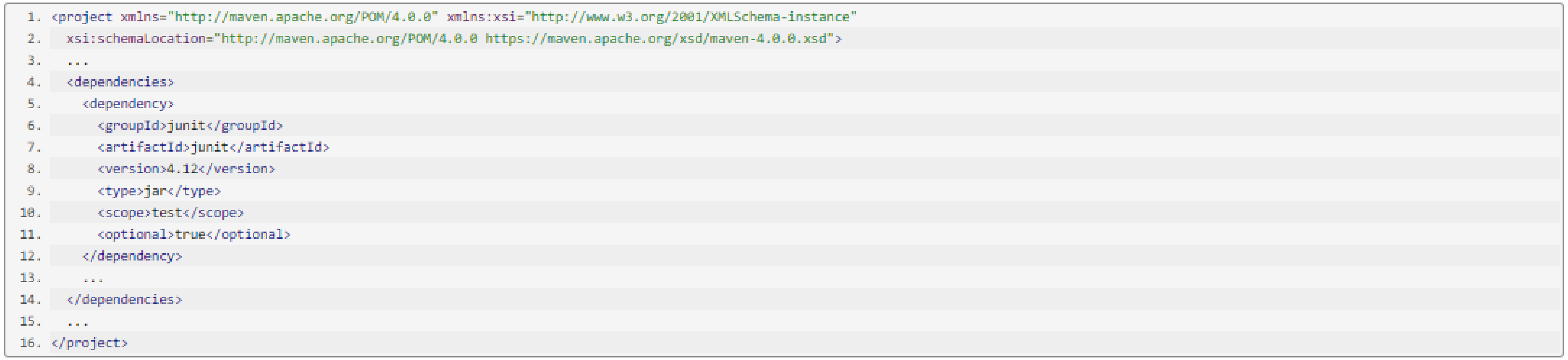
- Additional repositories can be configured in the pom.xml.

Maven Dependency Search Sequence

- When we execute Maven build commands, Maven starts looking for dependency libraries, in the following sequence:


found
1
Search dep. in local repository
not found
2
Search dep. in Central repository
not found & remote rep. are mentioned
3
Search dep. in Remote repository
found
4
It is downloaded to local repository
not found & no remote rep. are mentioned
not found
5
Throws Error

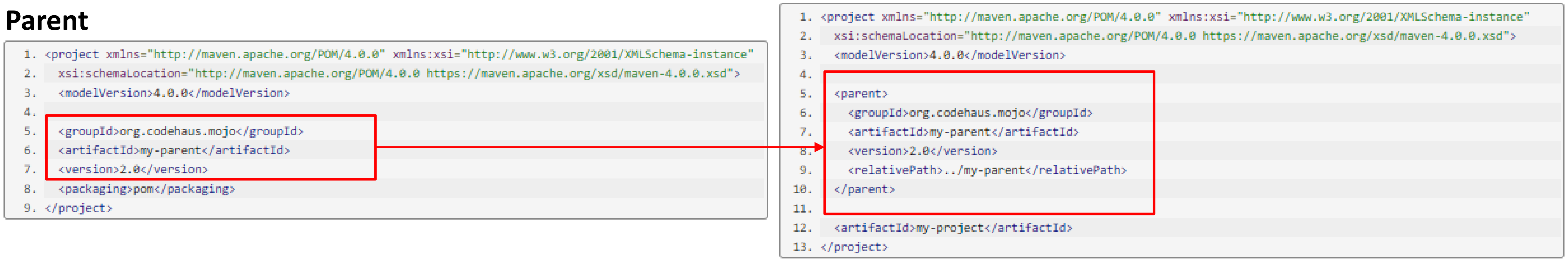
POM – Dependencies



```
<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  ...
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <type>jar</type>
      <scope>test</scope>
      <optional>true</optional>
    </dependency>
    ...
  </dependencies>
  ...
</project>
```

- **groupId:artifactId:version**, used to compute the Maven coordinate of a specific project in time.
- **type**, corresponds to the chosen dependency type (default: jar).
- **scope**: *compile(default)*, *runtime*, *test*, *provided*, *system*.

POM – Inheritance



Parent

```xml
<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.codehaus.mojo</groupId>
  <artifactId>my-parent</artifactId>
  <version>2.0</version>
  <packaging>pom</packaging>
</project>
```

Children

```xml
<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>my-parent</artifactId>
    <version>2.0</version>
    <relativePath>../my-parent</relativePath>
  </parent>

  <artifactId>my-project</artifactId>
</project>
```

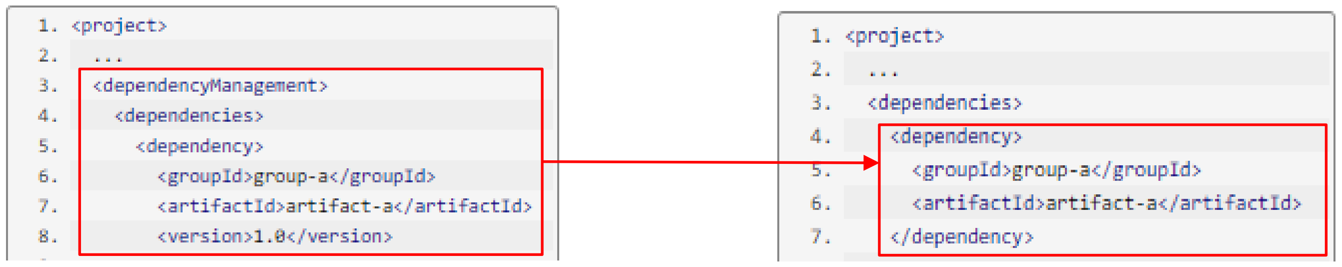
- You can add values to the parent POM, which will be inherited by its children.
- Most of the elements of the parent POM are inherited, including: groupild, version, depedencies, repositories, build etc, for more details https://maven.apache.org/pom.html#POM_Reference

POM – Inheritance

- All POMs inherit from a parent. This base POM is known as the **Super POM**, and contains values inherited by default.
- An easy way to look at the default configurations of the super POM is by the following link https://maven.apache.org/ref/3.0.4/maven-model-builder/super-pom.html
- While you can look your effective POM with the following command > **mvn help:effective-pom**

POM – Dependency Management

- In addition to inheriting some top-level elements, parents have elements to configure the values of child POMs dependencies. One of these elements is **dependencyManagement**.
- It is used by a POM to help manage dependency information in all its children.



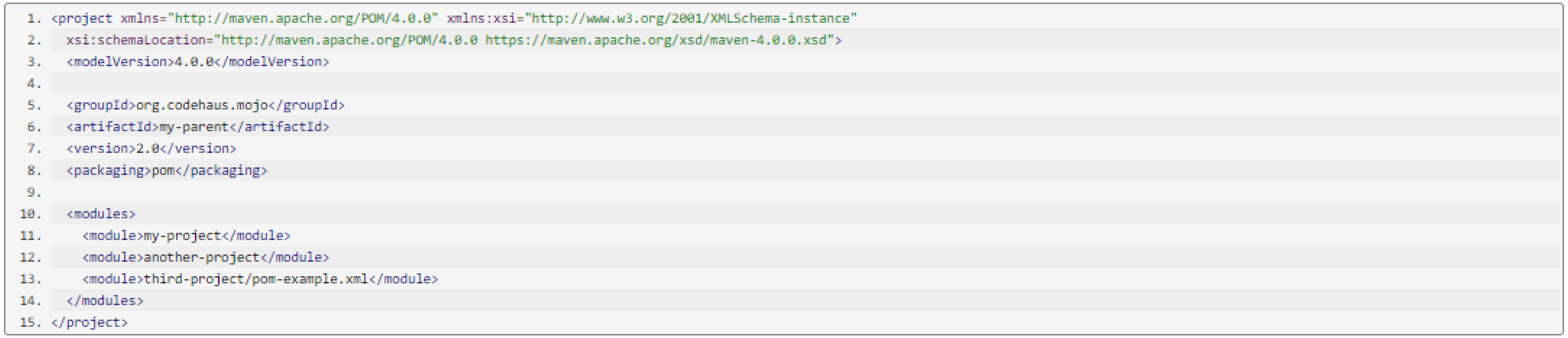
Parent

```
<project>
  ...
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>group-a</groupId>
        <artifactId>artifact-a</artifactId>
        <version>1.0</version>
```

Child

```
<project>
  ...
  <dependencies>
    <dependency>
      <groupId>group-a</groupId>
      <artifactId>artifact-a</artifactId>
    </dependency>
```

POM – Aggregation



```
<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.codehaus.mojo</groupId>
  <artifactId>my-parent</artifactId>
  <version>2.0</version>
  <packaging>pom</packaging>

  <modules>
    <module>my-project</module>
    <module>another-project</module>
    <module>third-project/pom-example.xml</module>
  </modules>
</project>
```

- A project with modules is known as a **multi-module**, or **aggregator project**.
- When we execute a Maven command against the aggregator project the same command then gets **propagated** to the modules below.

Build Lifecycle

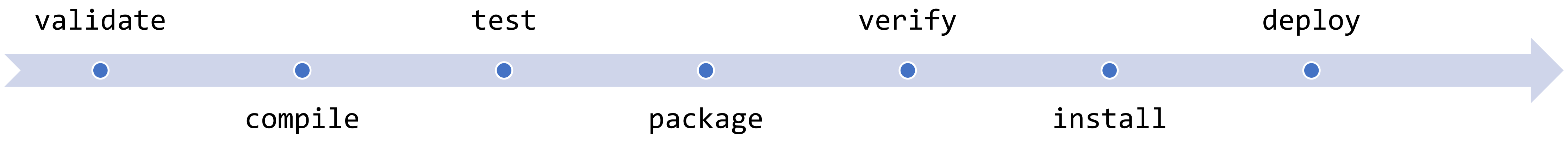
- Maven is based on the concept of **build lifecycles**, i.e., processes for building and distributing a particular artifact.
- Three built-in build lifecycles:

1. **default**: handles the deployment of the entire project.
2. **clean:** handles project cleaning (remove temporary files).
3. **site:** handles the creation of the project site documentation.

- A build lifecycle is defined by a sequence of **build phases**.

Build Phases

- The **default** lifecycle includes the following phases (and some more!)



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.
package	take the compiled code and package it in its distributable format, such as a JAR.
verify	run any checks on results of integration tests to ensure quality criteria are met.
install	Install the package into the local repository.
deploy	done in the build environment, copies the final package to the remote repository.

Usual Command Line Calls

- You should select the phase that matches your outcome:
  - If you want your jar, run

```
mvn package
```

  - if you want to run the unit test, run

```
mvn test
```

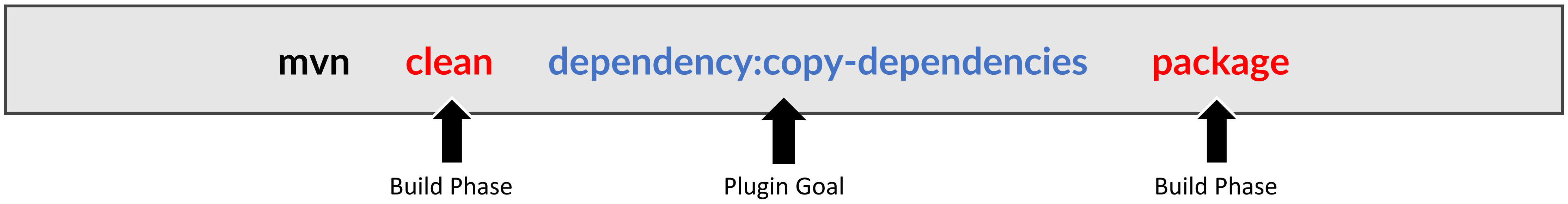
  - If you are uncertain what you want, the preferred phase to call is

```
mvn verify
```

- This command executes each default lifecycle phase in order (validate, compile, etc.), before executing verify.

Plugin Goals

- A **goal** represents a specific task which contributes to the building and managing of a project. It may be bound to zero or more build phases.
- The order of execution depends on the order in which the goal(s) and the build phase(s) are invoked.

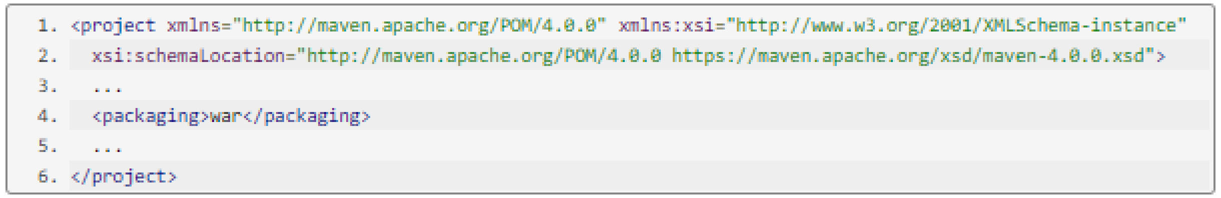


```
mvn    clean    dependency:copy-dependencies    package
```

Setting Up Your Project to Use the Build Lifecycle

- **How do you assign tasks to each of these build phases?**

- The first, and most common way, is to set the packaging for your project via <packaging> POM element.



```
<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  ...
  <packaging>war</packaging>
  ...
</project>
```

- Each packaging contains a list of goals binded to a particular phase.

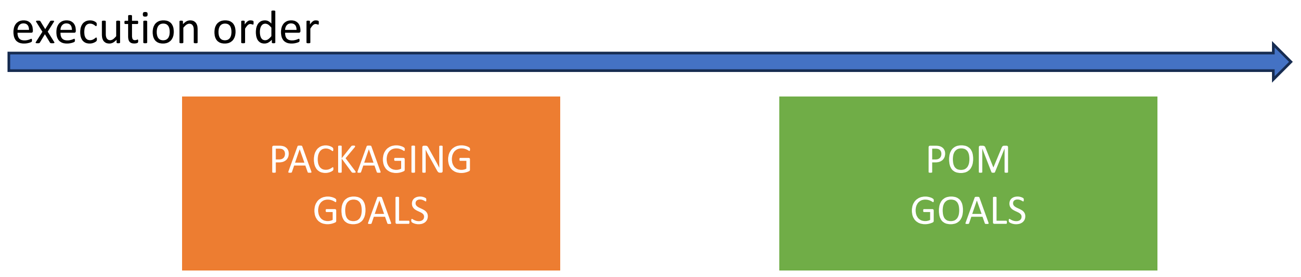
phase	**plugin:goal for the jar packaging**
process-resources	resources:resources
compile	compiler:compile
process-test-resources	resources:testresources
test-compile	compiler:testcompile
test	surefire:test
package	jar:jar
install	install:install
deploy	deploy:deploy

What are Maven Plugins?

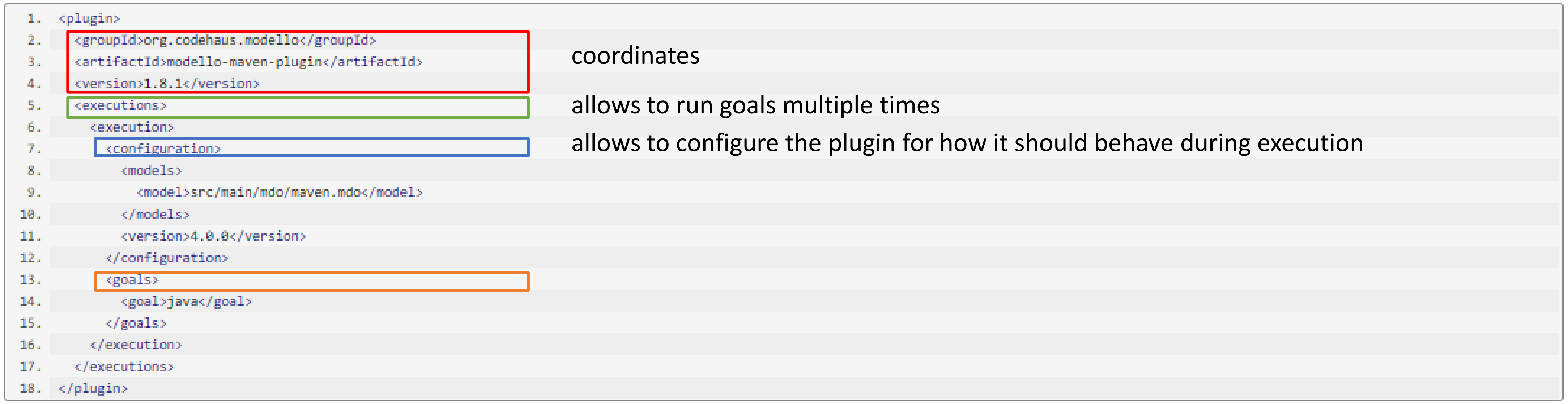
- The second way to add goals to phases is to configure plugins in your project.
- Maven is actually a plugin execution framework where every task is actually done by plugins. Plugins are artifacts that provide goals to Maven.
- Which can be executed using following syntax:
  - mvn [plugin-name]:[goal-name]

Maven Plugins

- Plugins can contain information that indicates which **lifecycle phase to bind a goal to.**
- You must also specify the **goals** you want to run as part of your build.
- The goals that are configured will be added to the goals already bound to the lifecycle from the packaging selected.



Plugin Example (1)



```
<plugin>
  <groupId>org.codehaus.modello</groupId>
  <artifactId>modello-maven-plugin</artifactId>
  <version>1.8.1</version>
  <executions>
    <execution>
      <configuration>
        <models>
          <model>src/main/mdo/maven.mdo</model>
        </models>
        <version>4.0.0</version>
      </configuration>
      <goals>
        <goal>java</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

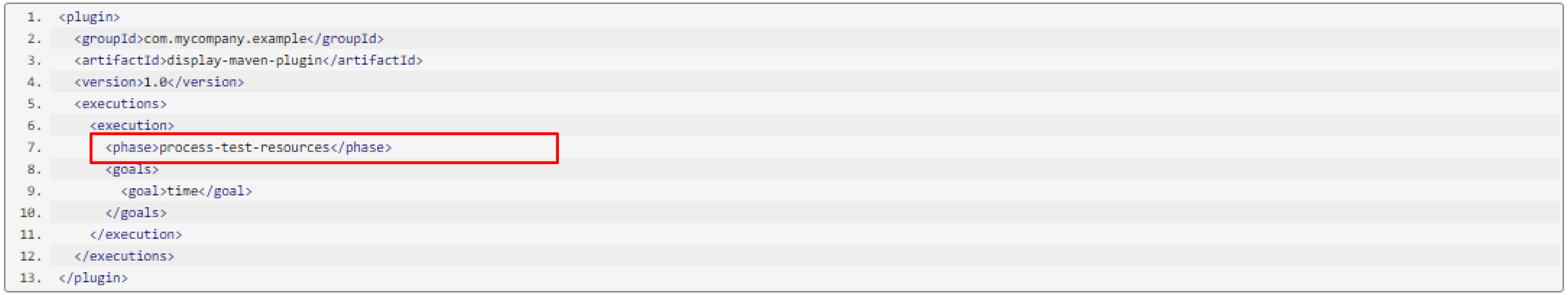
coordinates

allows to run goals multiple times

allows to configure the plugin for how it should behave during execution

- To introduce the **model** plugin to run the **java** goal within our build, we should add the following code to our POM in the *<plugins>* section of *<build>*.

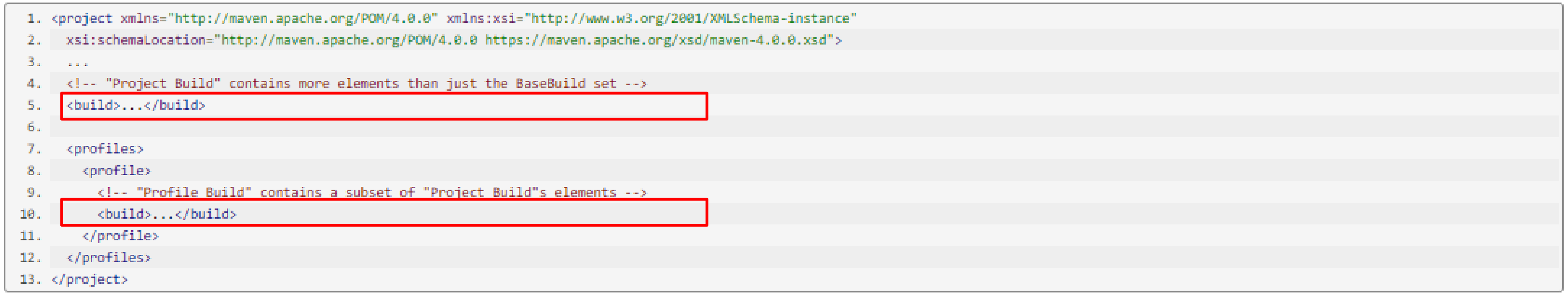
Plugin Example (2)



```xml
<plugin>
  <groupId>com.mycompany.example</groupId>
  <artifactId>display-maven-plugin</artifactId>
  <version>1.0</version>
  <executions>
    <execution>
      <phase>process-test-resources</phase>
      <goals>
        <goal>time</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

- For example, let's say you have a goal **display:time** that echos the current time to the commandline, and you want it to run in the **process-test-resources** phase to indicate when the tests were started.
- For more details: https://maven.apache.org/guides/mini/guide-configuring-plugins.html

POM – Build Settings



```
<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  ...
  <!-- "Project Build" contains more elements than just the BaseBuild set -->
  <build>...</build>

  <profiles>
    <profile>
      <!-- "Profile Build" contains a subset of "Project Build"s elements -->
      <build>...</build>
    </profile>
  </profiles>
</project>
```

- The *<build>* element is conceptually divided into two parts:
  - **BaseBuild** which contains *defaultGoal*, *resources*, *plugins* etc. elements (common to both project and profile build).
  - **BuildElement** which contains directories and extensions elements (just for build project).

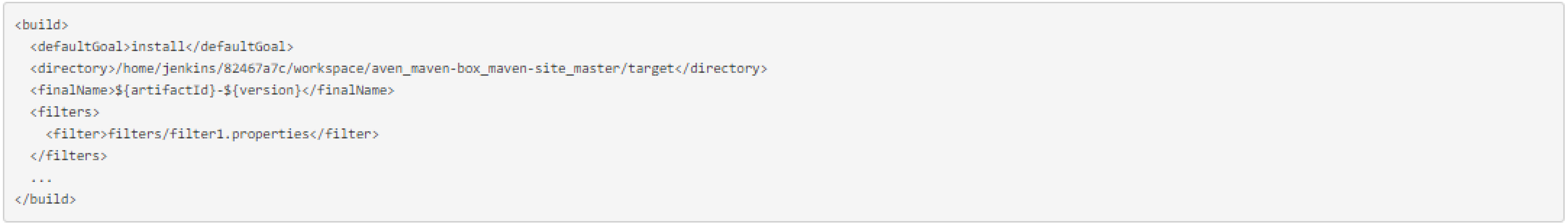
POM – Maven Profile

- Maven profiles can be used to create **customized build** configurations.
- A basic Example

```
<profile>
   <id>no-tests</id>
   <properties>
      <maven.test.skip>true</maven.test.skip>
   </properties>
</profile>
```

- **mvn package –P no-tests**

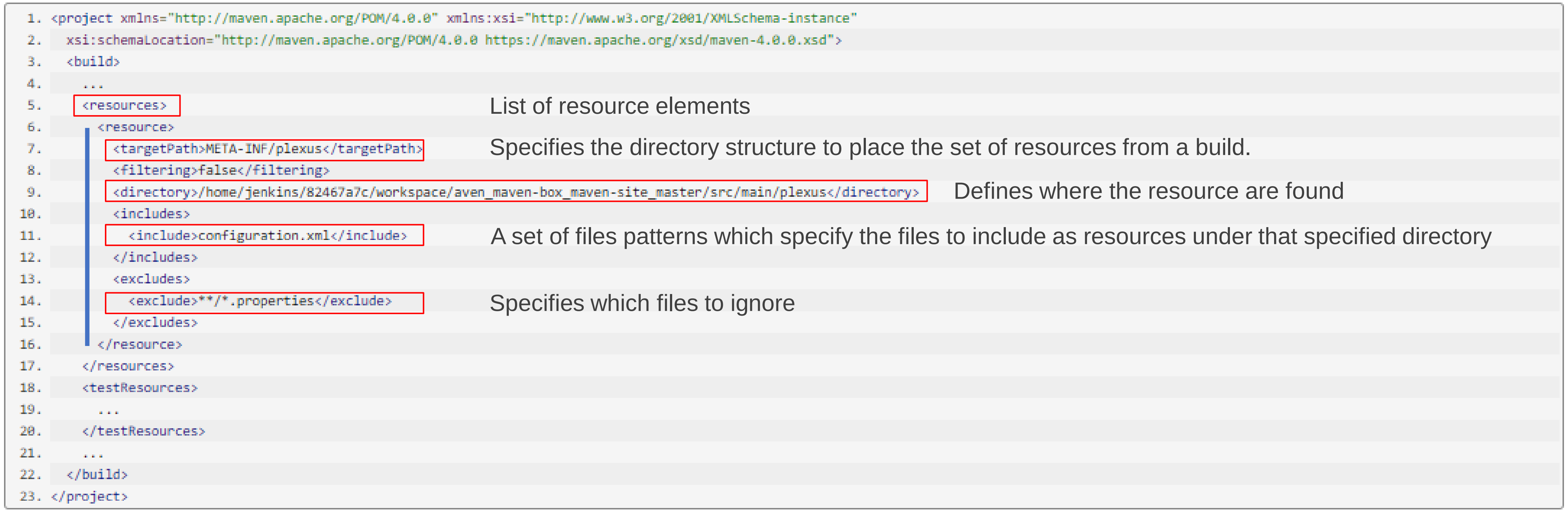
POM – BaseBuild Element



```
<build>
  <defaultGoal>install</defaultGoal>
  <directory>/home/jenkins/82467a7c/workspace/aven_maven-box_maven-site_master/target</directory>
  <finalName>${artifactId}-${version}</finalName>
  <filters>
    <filter>filters/filter1.properties</filter>
  </filters>
  ...
</build>
```

- **defaultGoal:** the default goal or phase to execute.
- **directory:** location where the build dump its files, defaults to ${project.basedir}/target.
- **finalName:** is the name of the bundled project when it is finally built. Defaults ${artifactId}-${version}

POM – Resources



```
<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <build>
    ...
    <resources>
      <resource>
        <targetPath>META-INF/plexus</targetPath>
        <filtering>false</filtering>
        <directory>/home/jenkins/82467a7c/workspace/aven_maven-box_maven-site_master/src/main/plexus</directory>
        <includes>
          <include>configuration.xml</include>
        </includes>
        <excludes>
          <exclude>**/*.properties</exclude>
        </excludes>
      </resource>
    </resources>
    <testResources>
      ...
    </testResources>
    ...
  </build>
</project>
```

List of resource elements

Specifies the directory structure to place the set of resources from a build.

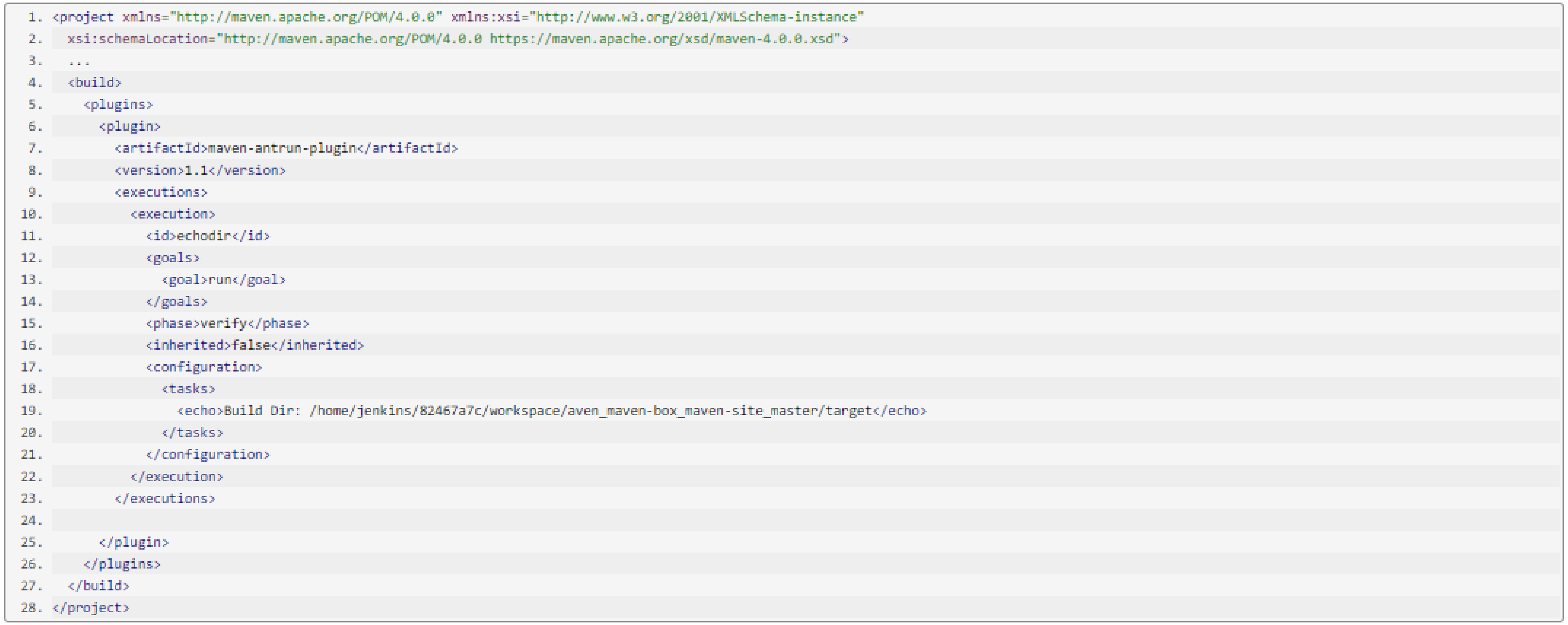
Defines where the resource are found

A set of files patterns which specify the files to include as resources under that specified directory

Specifies which files to ignore

- For example, a Plexus project requires a configuration.xml file (which specifies component configurations to the container) to live within the META-INF/plexus directory.

POM – Plugins



```xml
<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  ...
  <build>
    <plugins>
      <plugin>
        <artifactId>maven-antrun-plugin</artifactId>
        <version>1.1</version>
        <executions>
          <execution>
            <id>echodir</id>
            <goals>
              <goal>run</goal>
            </goals>
            <phase>verify</phase>
            <inherited>false</inherited>
            <configuration>
              <tasks>
                <echo>Build Dir: /home/jenkins/82467a7c/workspace/aven_maven-box_maven-site_master/target</echo>
              </tasks>
            </configuration>
          </execution>
        </executions>

      </plugin>
    </plugins>
  </build>
</project>
```

POM – Build Element

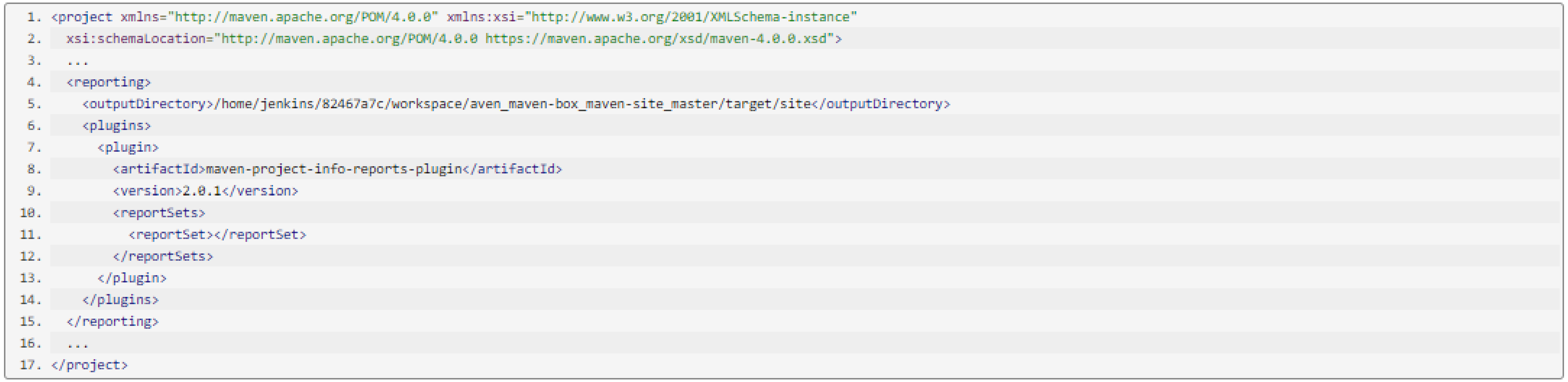
Directories

```xml
<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  ...
  <build>
    <sourceDirectory>/home/jenkins/82467a7c/workspace/aven_maven-box_maven-site_master/src/main/java</sourceDirectory>
    <scriptSourceDirectory>/home/jenkins/82467a7c/workspace/aven_maven-box_maven-site_master/src/main/scripts</scriptSourceDirectory>
    <testSourceDirectory>/home/jenkins/82467a7c/workspace/aven_maven-box_maven-site_master/src/test/java</testSourceDirectory>
    <outputDirectory>/home/jenkins/82467a7c/workspace/aven_maven-box_maven-site_master/target/classes</outputDirectory>
    <testOutputDirectory>/home/jenkins/82467a7c/workspace/aven_maven-box_maven-site_master/target/test-classes</testOutputDirectory>
    ...
  </build>
</project>
```

Extensions

```xml
<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  ...
  <build>
    ...
    <extensions>
      <extension>
        <groupId>org.apache.maven.wagon</groupId>
        <artifactId>wagon-ftp</artifactId>
        <version>1.0-alpha-3</version>
      </extension>
    </extensions>
    ...
  </build>
</project>
```

POM – Reporting



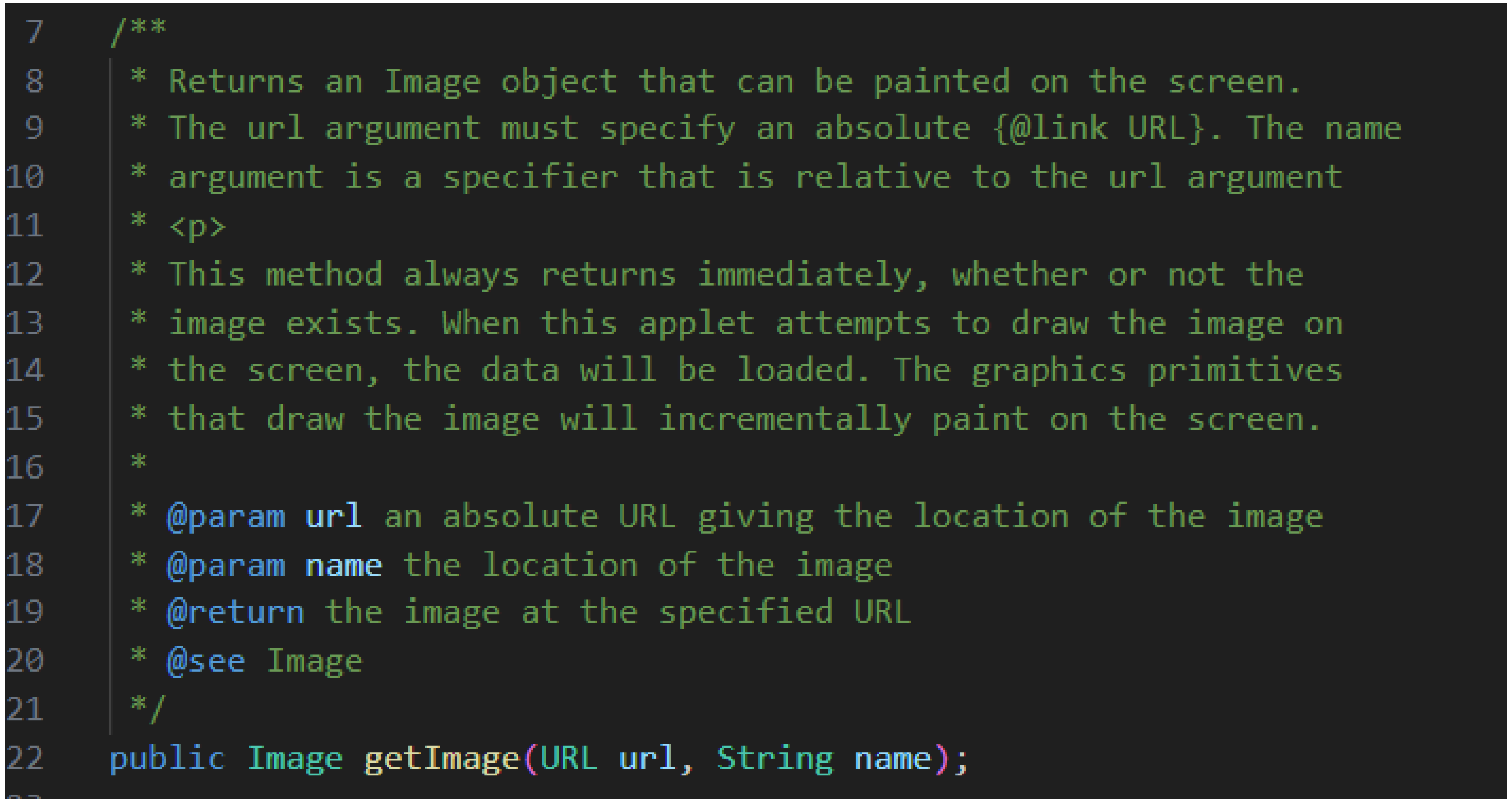
```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  ...
  <reporting>
    <outputDirectory>/home/jenkins/82467a7c/workspace/aven_maven-box_maven-site_master/target/site</outputDirectory>
    <plugins>
      <plugin>
        <artifactId>maven-project-info-reports-plugin</artifactId>
        <version>2.0.1</version>
        <reportSets>
          <reportSet></reportSet>
        </reportSets>
      </plugin>
    </plugins>
  </reporting>
  ...
</project>
```

- Reporting contains the elements that correspond specifically for the **site** generation phase.
- Much like the build element’s ability to configure plugins, reporting commands the same ability.

What is JavaDoc?

- Javadoc is a tool that allows you to **document** the sources of a program within the **sources themselves**.
- The programmer inserts comments in the source code in a particular format.
- These comments are extracted by the javadoc, which converts them into a more user-friendly format (HTML, PDF etc).

JavaDoc Example



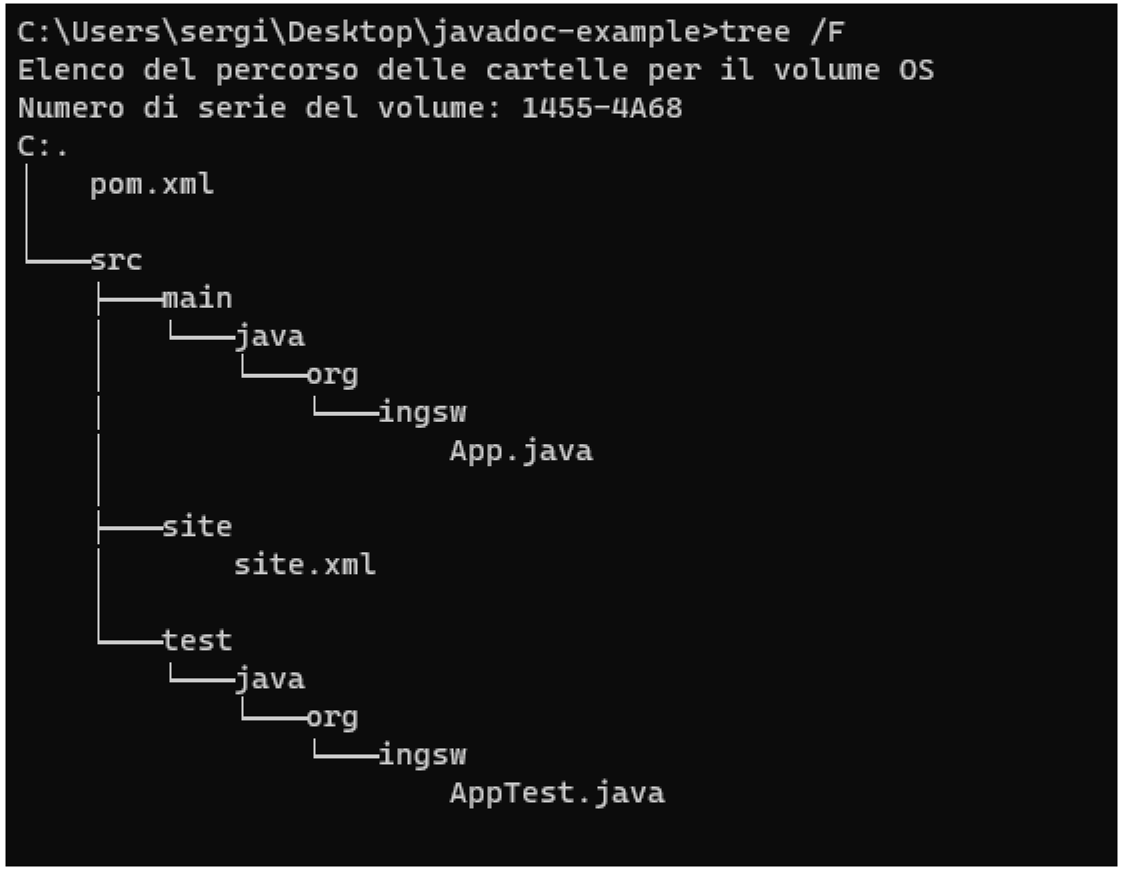
```java
/**
 * Returns an Image object that can be painted on the screen.
 * The url argument must specify an absolute {@link URL}. The name
 * argument is a specifier that is relative to the url argument
 * <p>
 * This method always returns immediately, whether or not the
 * image exists. When this applet attempts to draw the image on
 * the screen, the data will be loaded. The graphics primitives
 * that draw the image will incrementally paint on the screen.
 *
 * @param url an absolute URL giving the location of the image
 * @param name the location of the image
 * @return the image at the specified URL
 * @see Image
 */
public Image getImage(URL url, String name);
```

How to Generate JavaDoc in the Maven Project

1 STEP

- Generate a simple java project using archetype

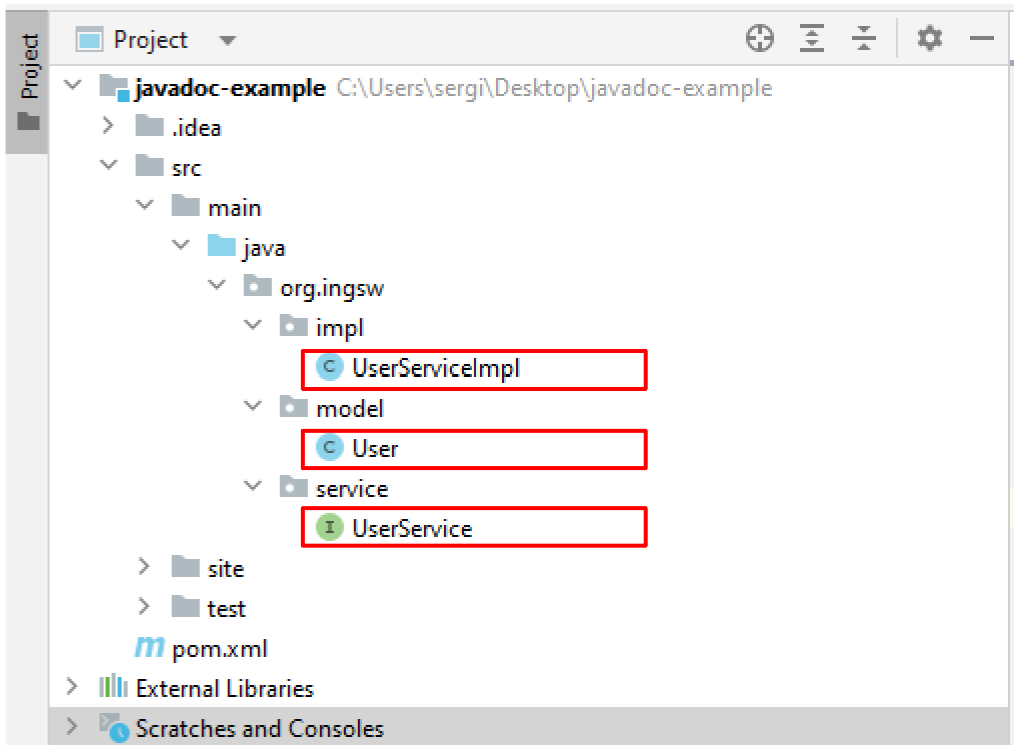
```
> mvn archetype:generate
        -DarchetypeGroupId=org.apache.maven.archetypes
        -DarchetypeArtifactId=maven-archetype-simple
        -DarchetypeVersion=1.4
        -DgroupId=org.ingsw
        -DartifactId=javadoc-example
```



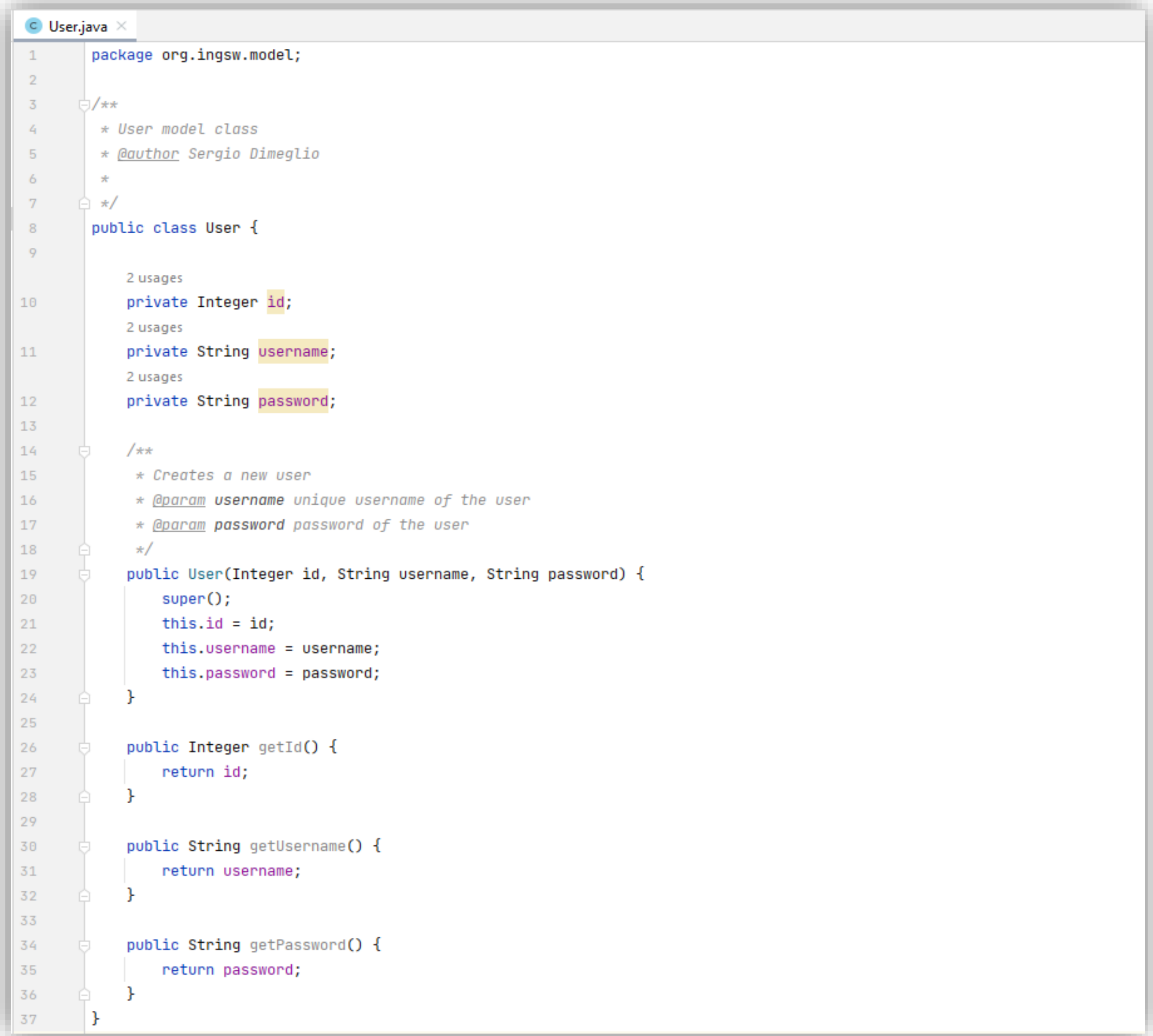
```
C:\Users\sergi\Desktop\javadoc-example>tree /F
Elenco del percorso delle cartelle per il volume OS
Numero di serie del volume: 1455-4A68
C:.
│   pom.xml
│
└───src
    ├───main
    │   └───java
    │       └───org
    │           └───ingsw
    │                   App.java
    │
    ├───site
    │       site.xml
    │
    └───test
        └───java
            └───org
                └───ingsw
                        AppTest.java
```

2 STEP

- Write a simple java code with javadoc comments.

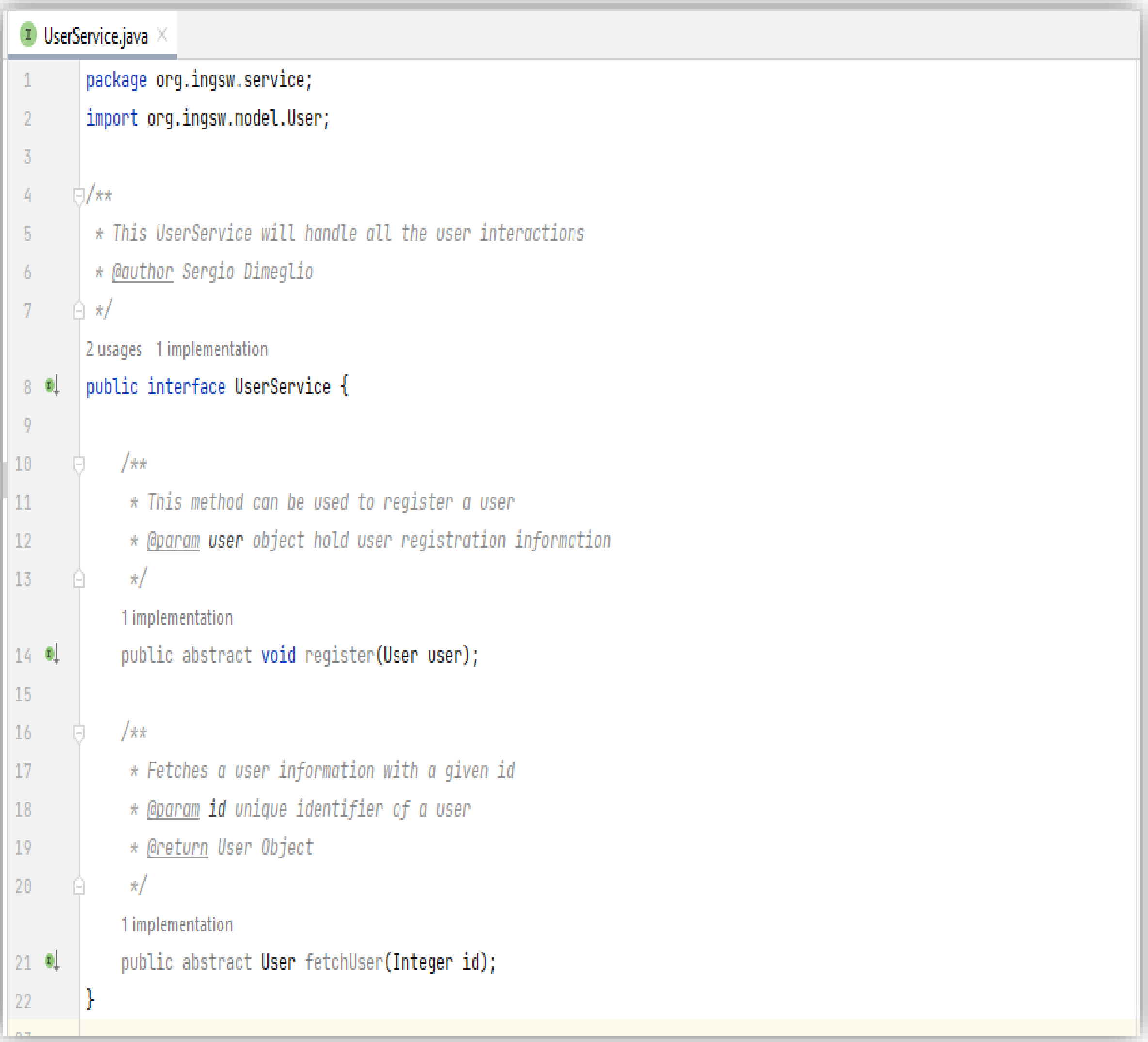


2 STEP


User.java

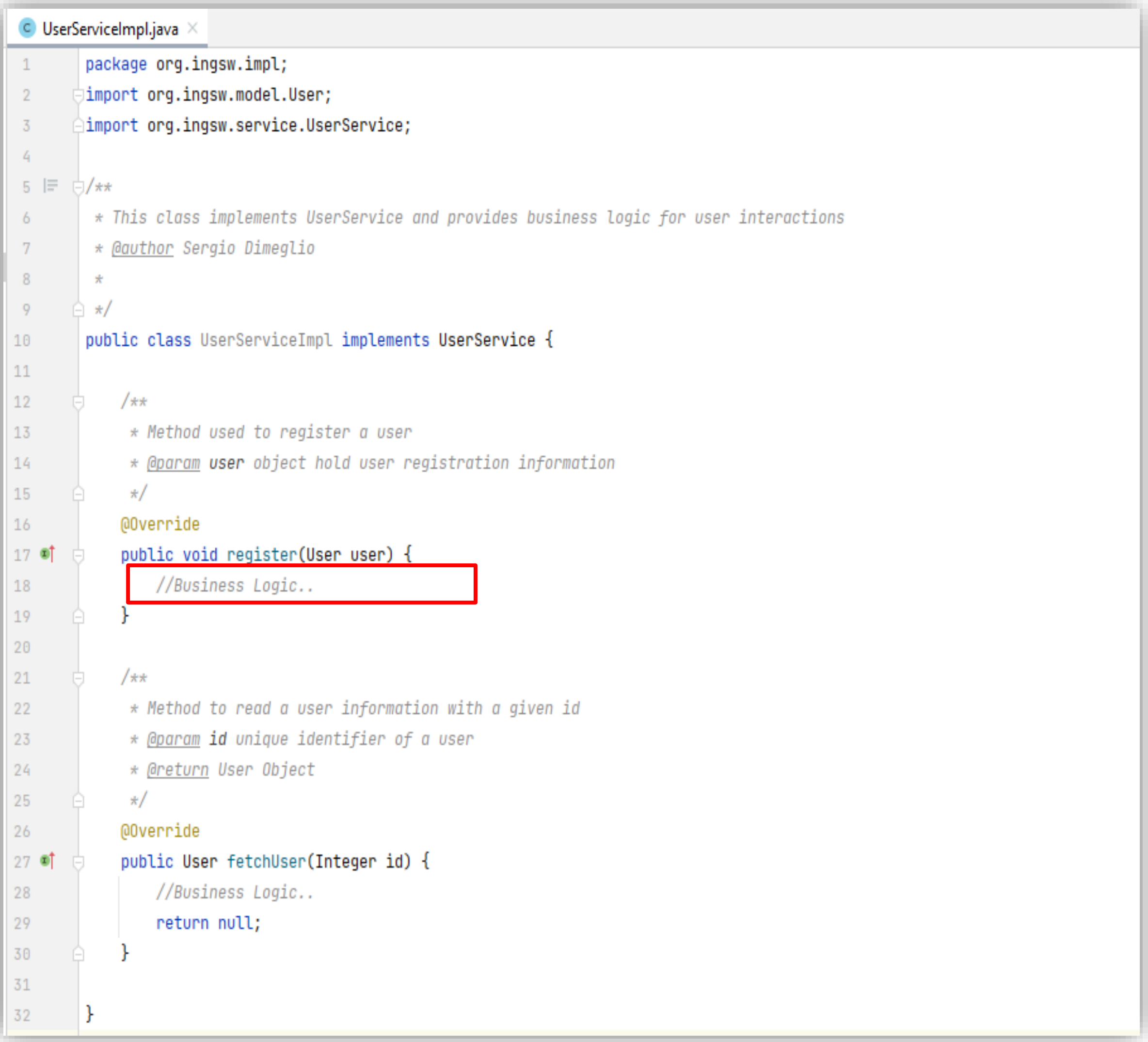
```java
package org.ingsw.model;

/**
 * User model class
 * @author Sergio Dimeglio
 *
 */
public class User {

    2 usages
    private Integer id;
    2 usages
    private String username;
    2 usages
    private String password;

    /**
     * Creates a new user
     * @param username unique username of the user
     * @param password password of the user
     */
    public User(Integer id, String username, String password) {
        super();
        this.id = id;
        this.username = username;
        this.password = password;
    }

    public Integer getId() {
        return id;
    }

    public String getUsername() {
        return username;
    }

    public String getPassword() {
        return password;
    }
}
```

2 STEP


UserService.java

```
package org.ingsw.service;
import org.ingsw.model.User;

/**
 * This UserService will handle all the user interactions
 * @author Sergio Dimeglio
 */
2 usages   1 implementation
public interface UserService {

    /**
     * This method can be used to register a user
     * @param user object hold user registration information
     */
    1 implementation
    public abstract void register(User user);

    /**
     * Fetches a user information with a given id
     * @param id unique identifier of a user
     * @return User Object
     */
    1 implementation
    public abstract User fetchUser(Integer id);
}
```

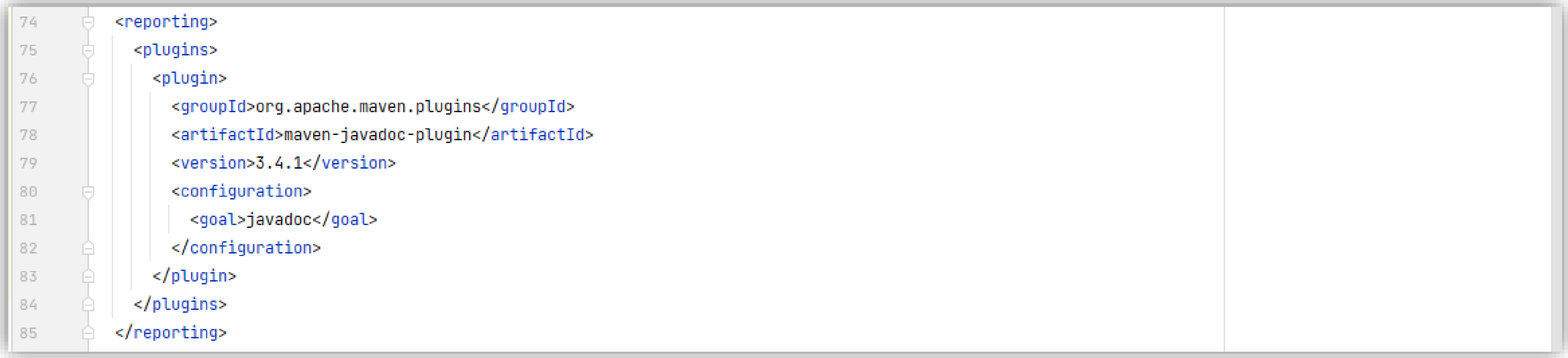
2 STEP


UserServiceImpl.java

```java
package org.ingsw.impl;
import org.ingsw.model.User;
import org.ingsw.service.UserService;

/**
 * This class implements UserService and provides business logic for user interactions
 * @author Sergio Dimeglio
 *
 */
public class UserServiceImpl implements UserService {

    /**
     * Method used to register a user
     * @param user object hold user registration information
     */
    @Override
    public void register(User user) {
        //Business Logic..
    }

    /**
     * Method to read a user information with a given id
     * @param id unique identifier of a user
     * @return User Object
     */
    @Override
    public User fetchUser(Integer id) {
        //Business Logic..
        return null;
    }

}
```

3 STEP

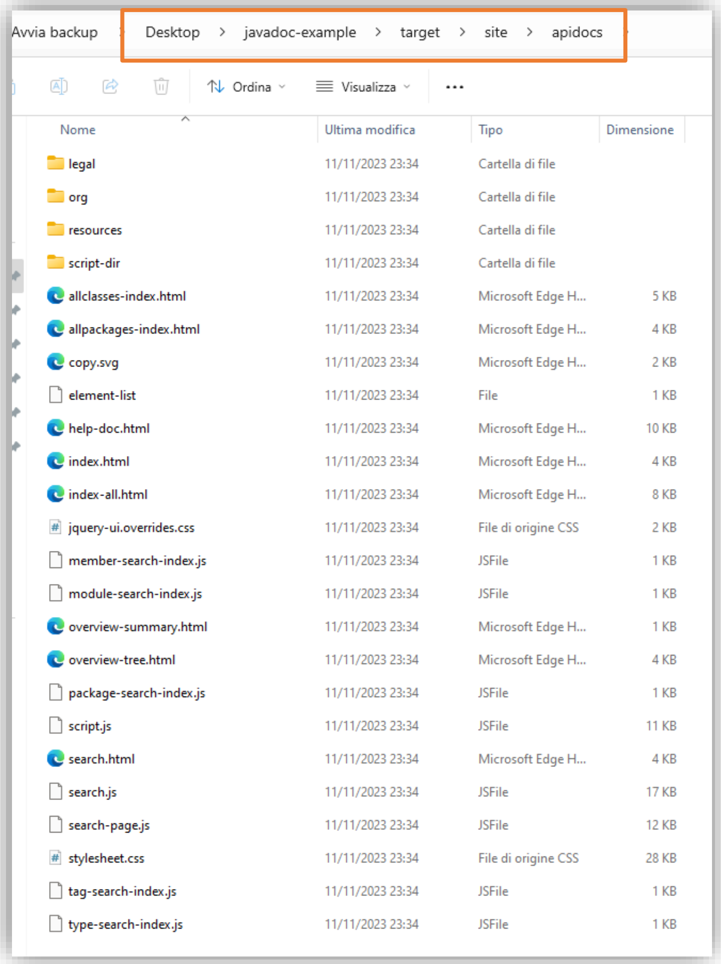
- To generate javadocs as part of the **site** generation, add the javadoc plugin in the *<reporting>* section of your pom.



```xml
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-javadoc-plugin</artifactId>
      <version>3.4.1</version>
      <configuration>
        <goal>javadoc</goal>
      </configuration>
    </plugin>
  </plugins>
</reporting>
```

```
> mvn site
```

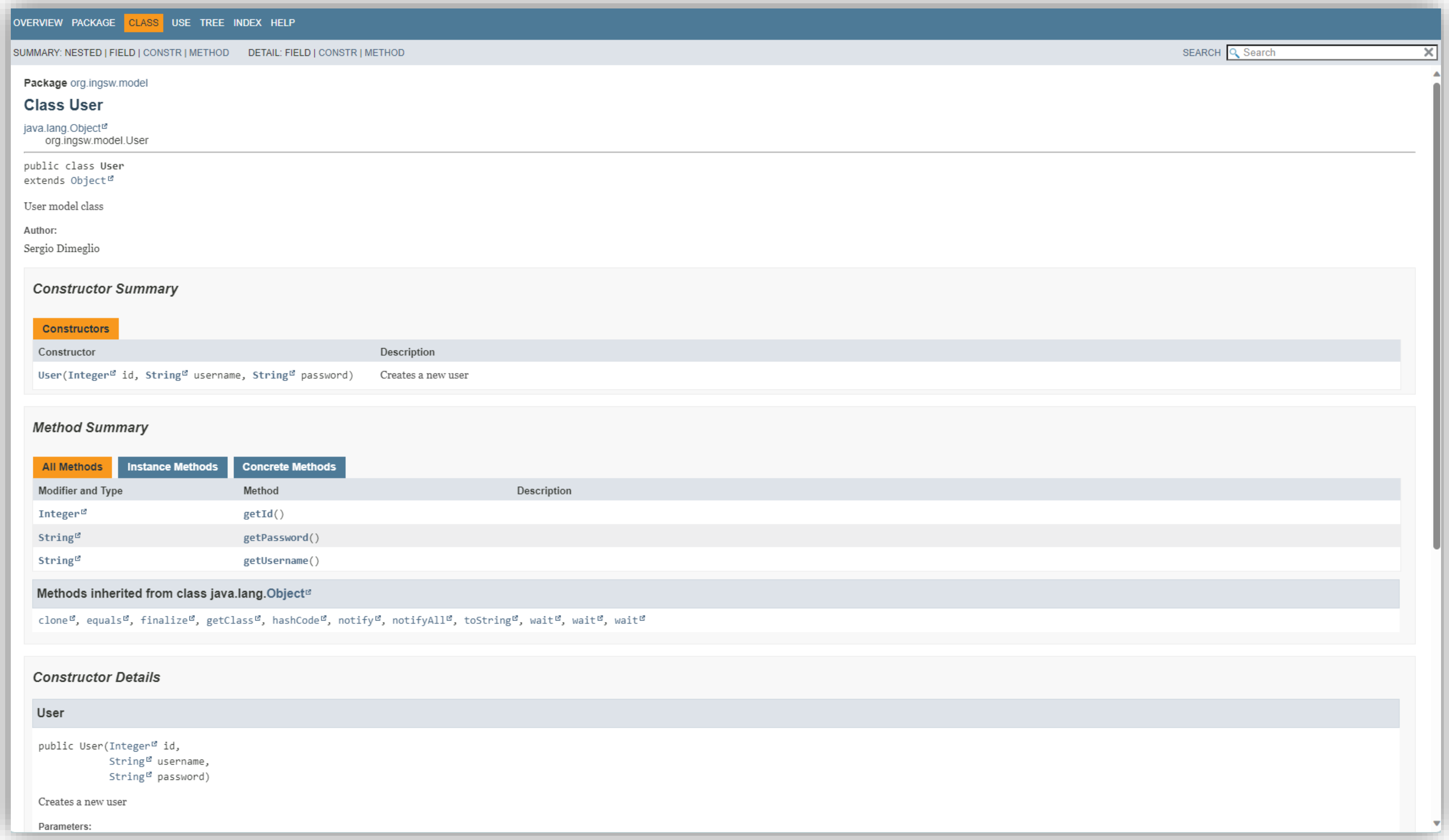
Generated Results



Avvia backup · Desktop › javadoc-example › target › site › apidocs

Ordina · Visualizza · ...

Nome	Ultima modifica	Tipo	Dimensione
legal	11/11/2023 23:34	Cartella di file	
org	11/11/2023 23:34	Cartella di file	
resources	11/11/2023 23:34	Cartella di file	
script-dir	11/11/2023 23:34	Cartella di file	
allclasses-index.html	11/11/2023 23:34	Microsoft Edge H...	5 KB
allpackages-index.html	11/11/2023 23:34	Microsoft Edge H...	4 KB
copy.svg	11/11/2023 23:34	Microsoft Edge H...	2 KB
element-list	11/11/2023 23:34	File	1 KB
help-doc.html	11/11/2023 23:34	Microsoft Edge H...	10 KB
index.html	11/11/2023 23:34	Microsoft Edge H...	4 KB
index-all.html	11/11/2023 23:34	Microsoft Edge H...	8 KB
jquery-ui.overrides.css	11/11/2023 23:34	File di origine CSS	2 KB
member-search-index.js	11/11/2023 23:34	JSFile	1 KB
module-search-index.js	11/11/2023 23:34	JSFile	1 KB
overview-summary.html	11/11/2023 23:34	Microsoft Edge H...	1 KB
overview-tree.html	11/11/2023 23:34	Microsoft Edge H...	4 KB
package-search-index.js	11/11/2023 23:34	JSFile	1 KB
script.js	11/11/2023 23:34	JSFile	11 KB
search.html	11/11/2023 23:34	Microsoft Edge H...	4 KB
search.js	11/11/2023 23:34	JSFile	17 KB
search-page.js	11/11/2023 23:34	JSFile	12 KB
stylesheet.css	11/11/2023 23:34	File di origine CSS	28 KB
tag-search-index.js	11/11/2023 23:34	JSFile	1 KB
type-search-index.js	11/11/2023 23:34	JSFile	1 KB



OVERVIEW PACKAGE CLASS USE TREE INDEX HELP
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD
SEARCH Search

Package org.ingsw.model

Class User

java.lang.Object
    org.ingsw.model.User

```
public class User
extends Object
```

User model class

Author:

Sergio Dimeglio

Constructor Summary

Constructors

Constructor	Description
`User(Integer id, String username, String password)`	Creates a new user

Method Summary

All Methods · Instance Methods · Concrete Methods

Modifier and Type	Method	Description
`Integer`	`getId()`	
`String`	`getPassword()`	
`String`	`getUsername()`	

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Details

User

```
public User(Integer id,
            String username,
            String password)
```

Creates a new user

Parameters: