



# Deploy java applications

# Java Archive (Jar) files

- fundamental packaging format in Java.
- used to **aggregate and compress multiple files into a single archive**,
- distribute and deploy Java applications.

<https://naveen-metta.medium.com/demystifying-jar-fat-jar-war-and-ear-files-in-java-d6802af9beda>

# JAR file

- simply ZIP files that contain all of the resources required to run a Java application.
- A JAR file typically contains:
  - Java source files
  - Property files
  - XML files
  - Image files
  - A manifest file

<https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Run-JAR-file-example-windows-linux-ubuntu>

# Manifest.mf

- Text file located in the jar META-INF/MANIFEST.MF

Manifest-Version: 1.0

Created-By: 1.7.0\_06 (Oracle Corporation)

Main-Class: MyPackage.MyClass

- Can define which class will be executed

```
java -jar MyJar.jar
```

- It will exec MyPackage.Myclass within MyJar.java

<https://docs.oracle.com/javase/tutorial/deployment/jar/manifestindex.html>

# Manifest.mf

- Text file located in the jar META-INF/MANIFEST.MF

Manifest-Version: 1.0

Created-By: 1.7.0\_06 (Oracle Corporation)

Main-Class: **MyPackage.MyClass**

- Can define which class will be executed

```
java -jar MyJar.jar
```

- It will exec **MyPackage.Myclass** within MyJar.java

<https://docs.oracle.com/javase/tutorial/deployment/jar/manifestindex.html>

# If I want to run a specific class?

and if I want to run a specific class within the jar ?

Great question! If you want to run a **specific class within a JAR**, not necessarily the one defined as the **Main-Class** in the manifest, you can do it like this:

## ✓ Command to Run a Specific Class from a JAR

bash

Copy

Edit

```
java -cp your-jar-file.jar your.package.ClassName
```



- **-cp** (or **-classpath**) tells Java to use your JAR for classes.
- Replace **your-jar-file.jar** with the name of your JAR.
- Replace **your.package.ClassName** with the **fully qualified name** of the class you want to run.



# Creating a jar manually

## Creating a Jar File:

To create a basic Jar file, you can use the following command:

```
jar cf MyJar.jar -C /path/to/classes .
```

## More details on jar command

<https://docs.oracle.com/javase/tutorial/deployment/jar/manifestindex.html>

<https://joshiadvait8.medium.com/how-to-create-jar-file-easy-way-of-jar-from-java-files-b8c916ab291e>

# Jar commands: some examples

- Run a JAR file
  - `jar -jar jarfilename.jar`
- Create a JAR file
  - `jar cf jarfiletocreate.jar file-list`
- Open a JAR file
  - `jar xf jar-file-to-open.jar`
- Run a JAR without a manifest
  - `java -cp jarfilename.jar com.exe`

More details on jar command

<https://docs.oracle.com/javase/tutorial/deployment/jar/manifestindex.html>

<https://joshiadvait8.medium.com/how-to-create-jar-file-easy-way-of-jar-from-java-files-b8c916ab291e>



# Asked the chatGPT...

can you give the command line to run the class within the jar

Certainly! To run a Java class from within a **JAR file** that uses an embedded Jetty server, you need to:

1. **Package your project into a JAR** with all dependencies included.
2. **Ensure the JAR has a manifest file** that specifies the main class.
3. **Run the JAR** using the `java -jar` command.



# Asked the chatGPT...

can you give the command line to run the class within the jar

Certainly! To run a Java class from within a **JAR file** that uses an embedded Jetty server, you need to:

1. Package your project into a JAR with all dependencies included.
2. Ensure the JAR has a manifest file that specifies the main class.
3. Run the JAR using the `java -jar` command.



# Dependencies in java

- External Java libraries required for Project
  - Can be jars
  - Managed by user
- Must be within execution classpath and or Jar
- If not we get ClassNotFoundException

**\*\* Problem scenario #2: ClassLoader.loadClass() \*\***

Missing Java class

```
java.lang.ClassNotFoundException: org.ph.javaee.training5.ClassB
    at java.net.URLClassLoader$1.run(URLClassLoader.java:366)
    at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:423)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:356)
    at org.ph.javaee.training5.ClassNotFoundExceptionSimulator.main(Cl
```

Caller or referencing class

# Dependencies in java

- External Java libraries required for Project
  - Can be jars
  - Managed by user
- **Must be within execution classpath** and or Jar
- If not we get ClassNotFoundException

```
** Problem scenario #2: ClassLoader.loadClass() **
```

Missing Java class

```
java.lang.ClassNotFoundException: org.ph.javaee.training5.ClassB
    at java.net.URLClassLoader$1.run(URLClassLoader.java:366)
    at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:423)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:356)
    at org.ph.javaee.training5.ClassNotFoundExceptionSimulator.main(Cl
```

Caller or referencing class

# Dependencies in java

- External Java libraries required for Project
  - Can be jars
  - Managed by user
- Must be within execution classpath and **or Jar**
- If not we get ClassNotFoundException

**\*\* Problem scenario #2: ClassLoader.loadClass() \*\***

Missing Java class

```
java.lang.ClassNotFoundException: org.ph.javaee.training5.ClassB
    at java.net.URLClassLoader$1.run(URLClassLoader.java:366)
    at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:423)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:356)
    at org.ph.javaee.training5.ClassNotFoundExceptionSimulator.main(Cl
```

Caller or referencing class

# In default java projects in VSCode

- Place dependencies jar in
  - <project>/libs (default)
  - Path within settings.json field
    - "java.project.referencedLibraries"



# Create jar with dependencies

## Step-by-Step Instructions

1. Package into a **fat JAR** (with dependencies)

# Create jar with dependencies

## Step-by-Step Instructions

1. Package into a **fat JAR** (with dependencies)



# Fat Jar Files

- include **all dependencies within the archive**.
  - self-contained approach
  - simplifies deployment,
  - as the application can be run **without relying on external libraries**.
- Also referred as Uber Jars or executable Jars,

# Fat Jar Files

- include **all dependencies within the archive**.
  - self-contained approach
  - simplifies deployment,
  - as the application can be run **without relying on external libraries**.
- Also referred as **Uber Jars** or executable Jars,

## Creating a Fat Jar File:

Using tools like **Maven** or **Gradle**, you can generate a Fat Jar easily.



- software project management and comprehension tool. Based on the concept of a project object model (POM), **Maven** can **manage** a **project's build**, reporting and documentation from a central piece of information.



<https://maven.apache.org/>

# What is Maven Used For?

- **easily build a project using maven.**
- **add jars and other dependencies of the project easily using the help of maven.**
- Maven is very helpful for a project while **updating the central repository of JARs and other dependencies.**
- With the help of Maven, **we can build any number of projects into output types like the JAR, WAR, etc** without doing any scripting.
- helps in managing the project's build lifecycle, including tasks like **compiling, testing, packaging, and deploying the code.**
- **provides a standard project structure**, making it easy for developers to understand the layout of the project and locate specific files.
- plugins can be used to add **additional functionality to the build process**, such as code coverage analysis, static code analysis, and more.
- simplifies **the process of managing project dependencies**, ensuring that the correct versions of libraries and frameworks are used throughout the project.

# Why Maven in Java?

- Solves the dependency management problem.
- Downloads third-party JAR files that some apps require.
- Provides an extensive, third-party plugin architecture.
- Enables switching between development profiles.
- Integrates with DevOps tools such as Jenkins and Docker.

# Hands on java with maven



Try [agent mode](#) in VS Code!

Overview

SETUP

GET STARTED

CONFIGURE

EDIT CODE

BUILD, DEBUG, TEST

SOURCE CONTROL

TERMINAL

GITHUB COPILOT

LANGUAGES

NODE.JS /  
JAVASCRIPT

TYPESCRIPT

PYTHON

**JAVA**

Getting Started

Navigate and Edit

Refactoring

Formatting and Linting

Project Management

**Build Tools**

# Java build tools in VS Code

Edit

IN THIS ARTICLE

Maven

Gradle

Additional resources

Next steps

 [RSS Feed](#)

 [Ask questions](#)

 [Follow @code](#)

 [Request features](#)

 [Report issues](#)

 [Watch videos](#)

This document is an overview of how to work with your Java build tools in Visual Studio Code. It covers the [Maven for Java](#) and [Gradle for Java](#) extensions as well as other tools.

If you run into any issues when using the features below, you can contact us by entering an [issue](#).

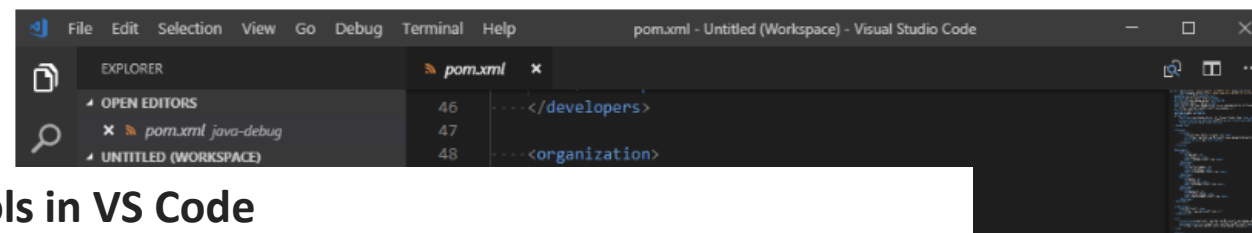
## Maven

[Maven](#) is a software tool that helps you manage Java projects and automate application builds. The [Maven for Java](#) extension for Visual Studio Code provides fully integrated Maven support, allowing you to explore Maven projects, execute Maven commands, and perform the goals of build lifecycle and plugins. We recommend installing the [Extension Pack for Java](#), which includes Maven support and other important Java development features.

[Install the Extension Pack for Java](#)

## Exploring Maven project

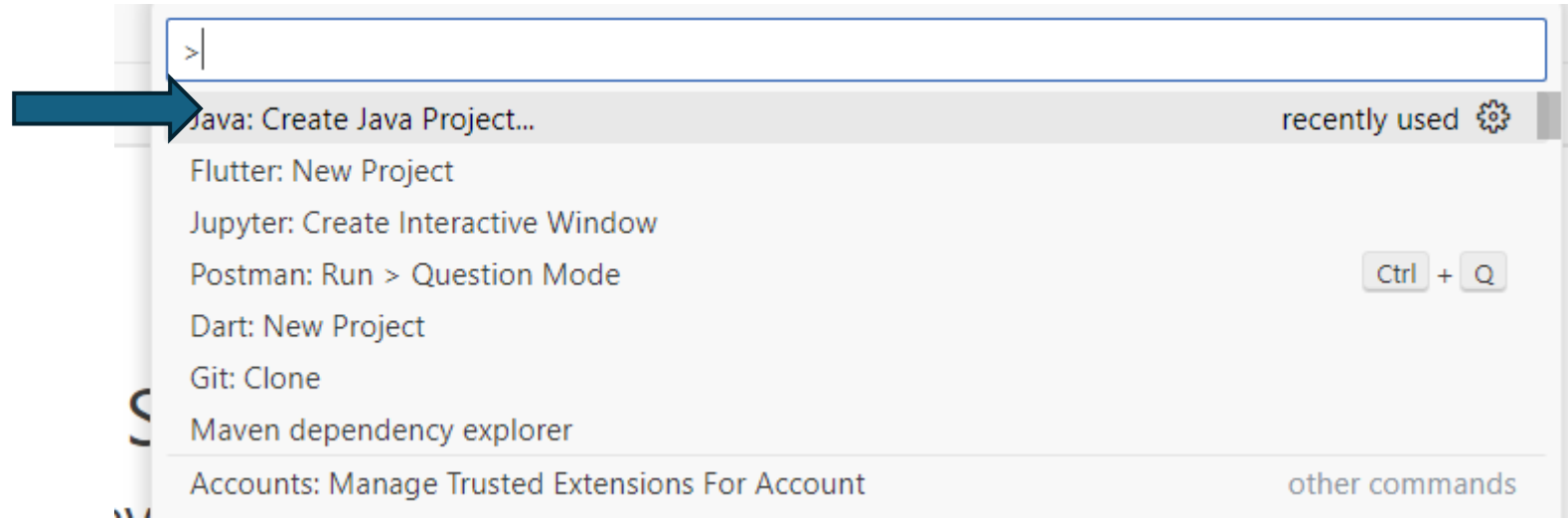
Once a Maven project is loaded, the extension will be activated and it will automatically scan for `pom.xml` files in your workspace and displays all Maven projects and their modules in the side bar.



## Java build tools in VS Code

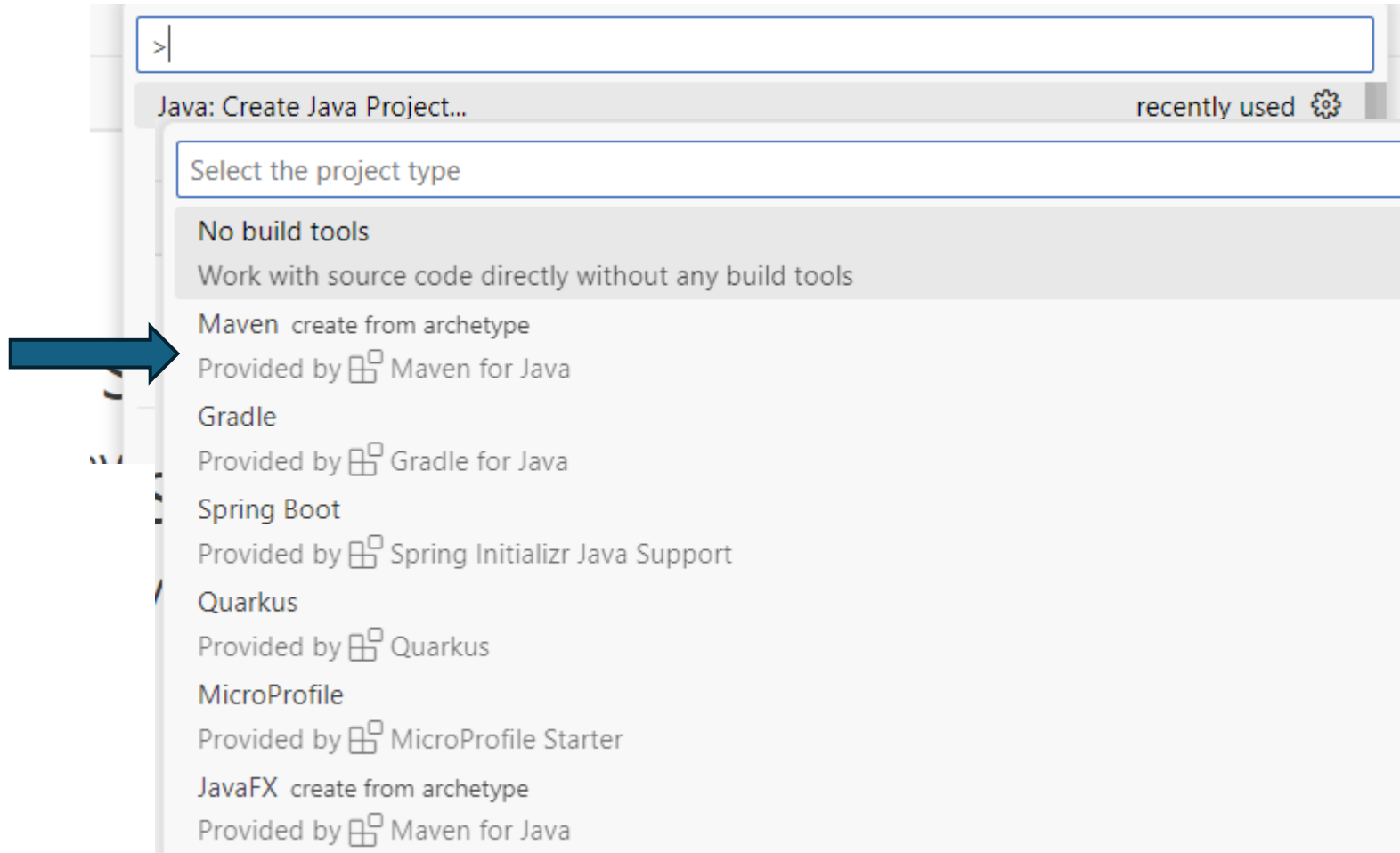
<https://code.visualstudio.com/docs/java/java-build>

# Create a project ( VSCode)

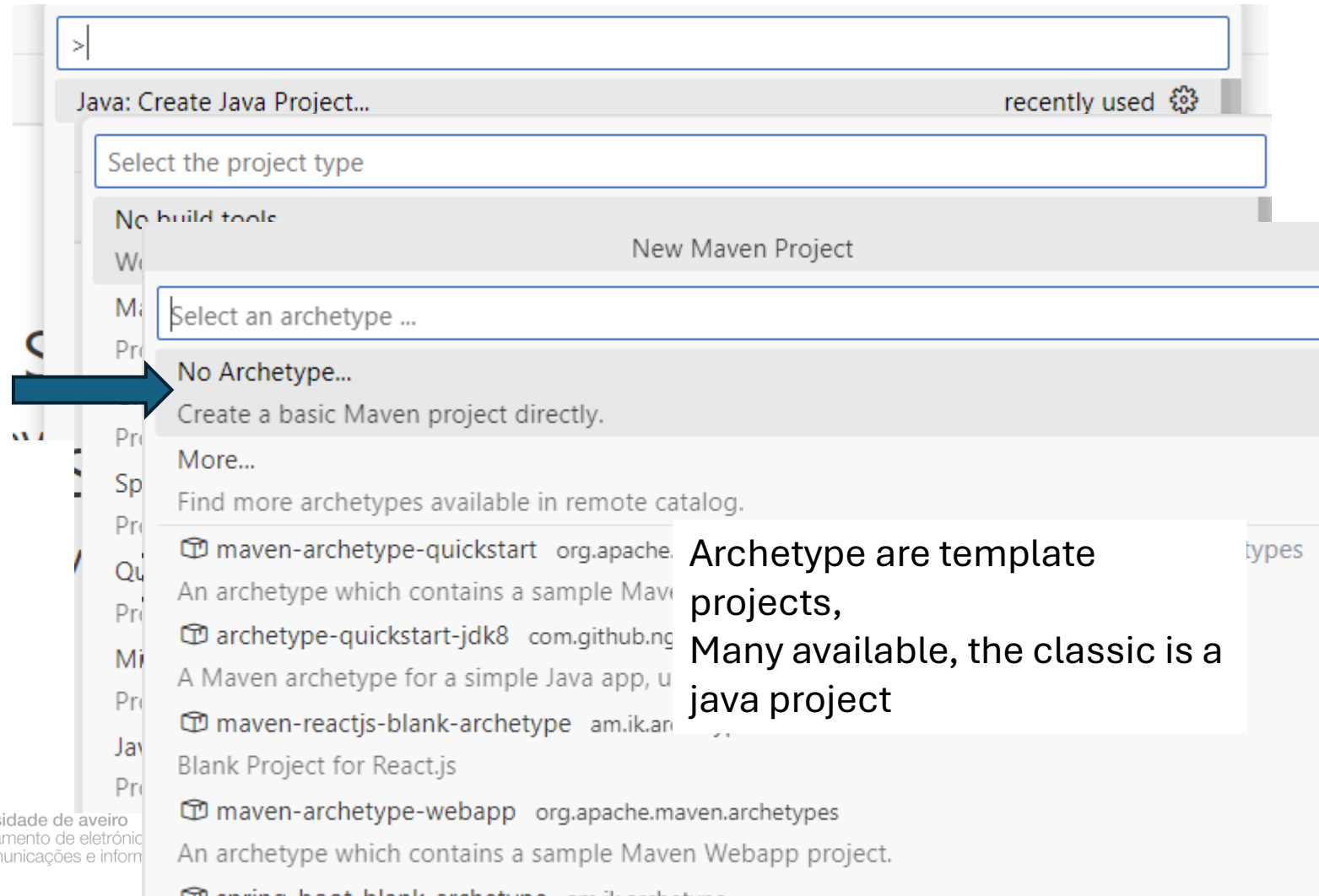


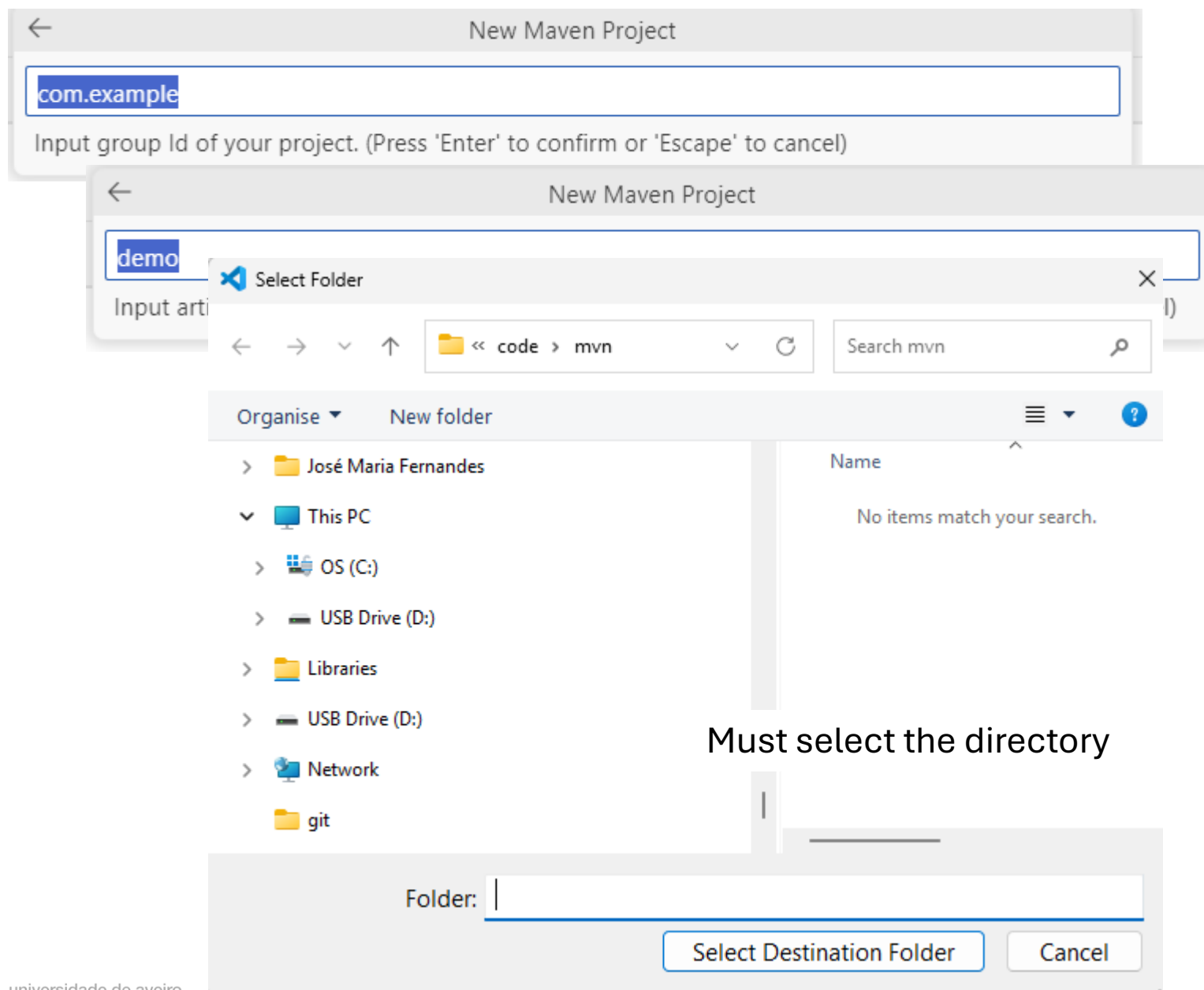


# Create a project ( VSCode)

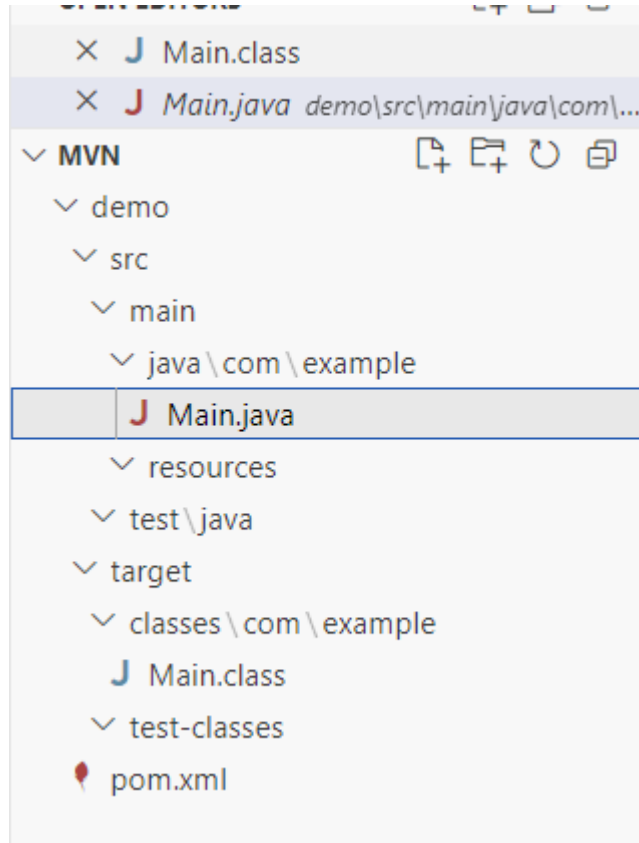


# Create a project ( VSCode)





# The structure



Maven creates the structure –  
this is the common structure  
of a shared by java archetypes

```
my-app
|-- pom.xml
`-- src
    |-- main
    |   |-- java
    |       |-- com
    |           |-- mycompany
    |               |-- app
    |                   |-- App.java
    |-- test
    |   |-- java
    |       |-- com
    |           |-- mycompany
    |               |-- app
    |                   |-- AppTest.java
```

# The structure

```
cd my-app
```

```
my-app
|-- pom.xml
`-- src
    |-- main
    |   |-- java
    |   |   |-- com
    |   |   |   |-- mycompany
    |   |   |   |   |-- app
    |   |   |   |   |   |-- App.java
    |-- test
    |   |-- java
    |   |   |-- com
    |   |   |   |-- mycompany
    |   |   |   |   |-- app
    |   |   |   |   |   |-- AppTest.java
```

src/main/java	Application/Library sources
src/main/resources	Application/Library resources
src/main/filters	Resource filter files
src/main/webapp	Web application sources
src/test/java	Test sources
src/test/resources	Test resources
src/test/filters	Test resource filter files
src/it	Integration Tests (primarily for plugins)
src/assembly	Assembly descriptors
src/site	Site
LICENSE.txt	Project's license
NOTICE.txt	Notices and attributions required by libraries that the project depends on
README.txt	Project's readme

<https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html>

# The structure

```
cd my-app
```

```
my-app
|-- pom.xml
`-- src
    |-- main
    |   |-- java
    |   |   |-- com
    |   |   |   |-- mycompany
    |   |   |   |   |-- app
    |   |   |   |   |   |-- App.java
    |-- test
    |   |-- java
    |   |   |-- com
    |   |   |   |-- mycompany
    |   |   |   |   |-- app
    |   |   |   |   |   |-- AppTest.java
    |-- it
    |-- assembly
    |-- site
    |-- LICENSE.txt
    |-- NOTICE.txt
    |-- README.txt
```

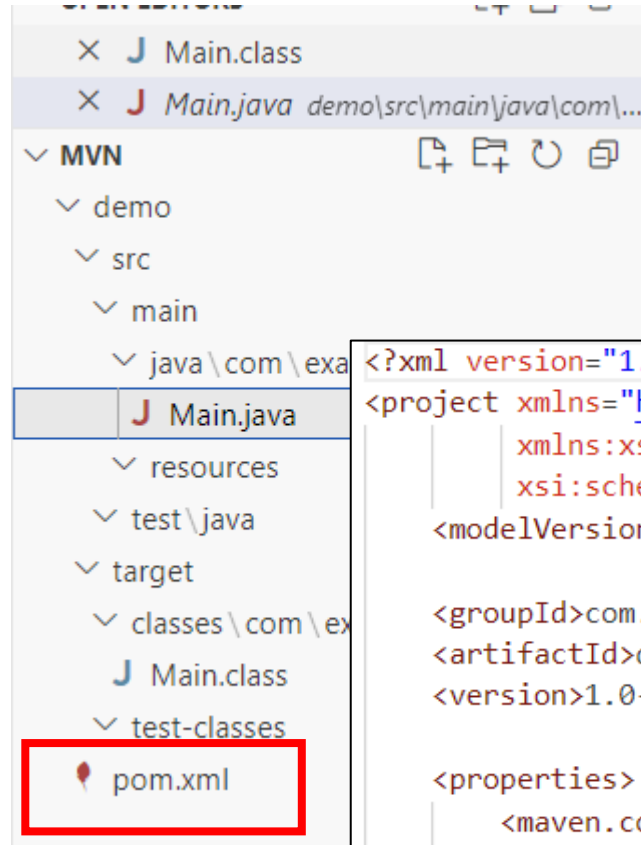
src/main/java	Application/Library sources
src/main/resources	Application/Library resources
src/main/filters	Resource filter files
src/main/webapp	Web application sources
src/test/java	Test sources
src/test/resources	Test resources
src/test/filters	Test resource filter files
src/it	Integration Tests (primarily for plugins)
src/assembly	Assembly descriptors
src/site	Site
LICENSE.txt	Project's license
NOTICE.txt	Notices and attributions required by libraries that the project depends on
README.txt	Project's readme

<https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html>

# pom.xml (Project Object Model)

- **Fundamental** file in Maven
- Xml file
- Contains:
  - configuration,
  - dependencies, and
  - project information required for building a Java project.
- details about
  - the project and
  - dependencies (libraries or frameworks),
  - plugins,
  - goals,
  - and build lifecycle.

# The pom.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>

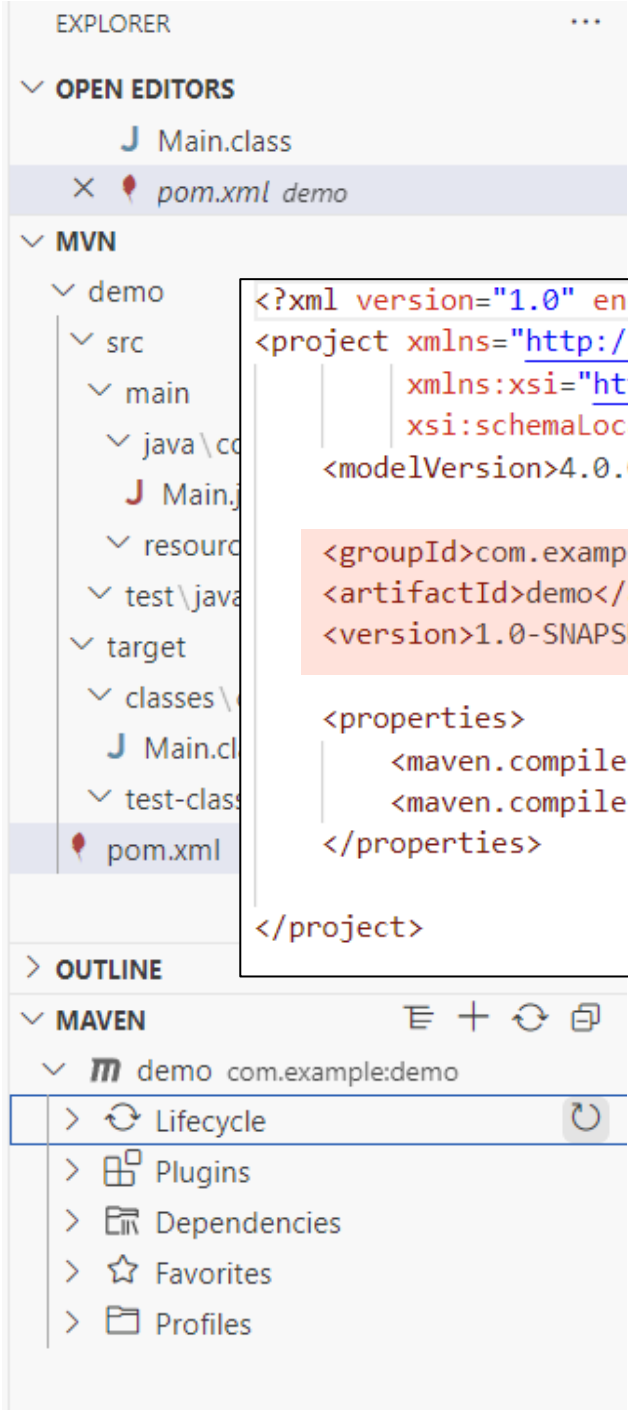
  <groupId>com.example</groupId>
  <artifactId>demo</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
  </properties>

</project>
```



# The pom.xml



The screenshot shows an IDE interface. The Explorer on the left lists the project structure: **EXPLORED**, **OPEN EDITORS** (Main.class, pom.xml demo), **MVN** (demo, src, main, java, Main.j, resource, test, java, target, classes, Main.cl, test-class, pom.xml), and **OUTLINE** (MAVEN, demo com.example:demo, Lifecycle, Plugins, Dependencies, Favorites, Profiles). The pom.xml file is open in the editor, showing the following XML content:

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

  <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
  </properties>

</project>
```

Every pom.xml should have

- groupId - the id of the project's group.
- artifactId - the id of the artifact (project)
- version - the version of the artifact under the specified group

EXPLORER

OPEN EDITORS

- Main.class
- pom.xml demo

MVN

- demo
  - src
    - main
      - java\com\example
        - Main.java
        - resources
      - test\java
    - target
      - classes\com\exa
        - Main.class
      - test-classes
    - pom.xml

OUTLINE

MAVEN

- demo com.examp
  - Lifecycle
  - Plugins
  - Dependencies
  - Favorites
  - Profiles

New Maven Project

com.example

Input group Id of your project. (Press 'Enter' to confirm or 'Escape' to cancel)

New Maven Project

demo

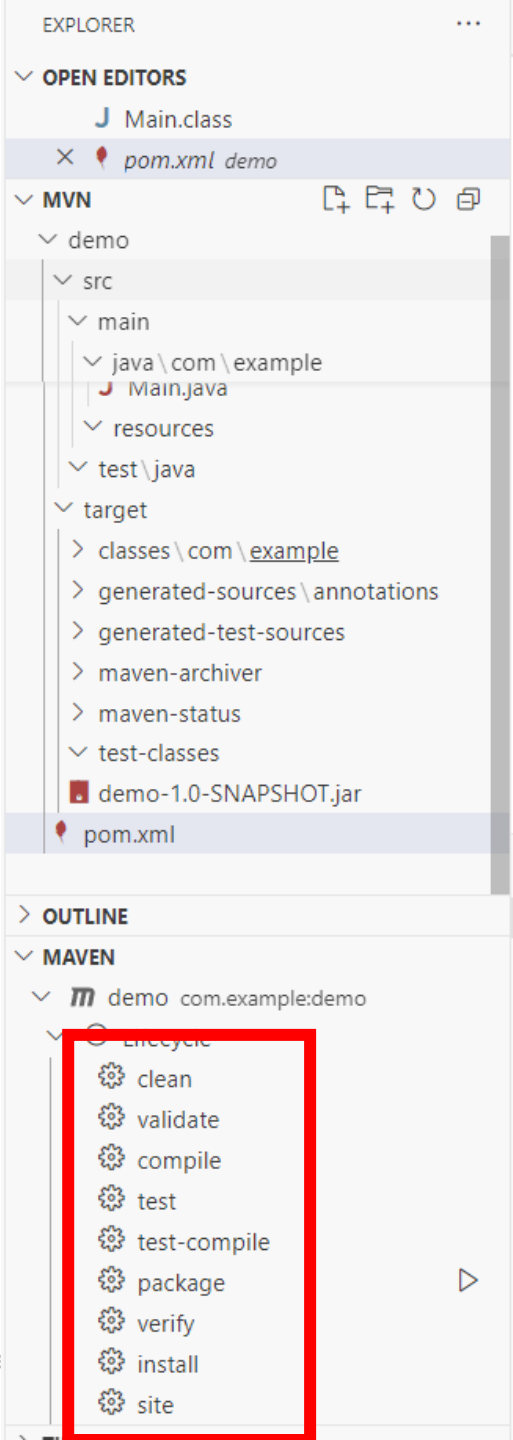
Input artifact Id (also as project name) of your project. (Press 'Enter' to confirm or 'Escape' to cancel)

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

  <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
  </properties>

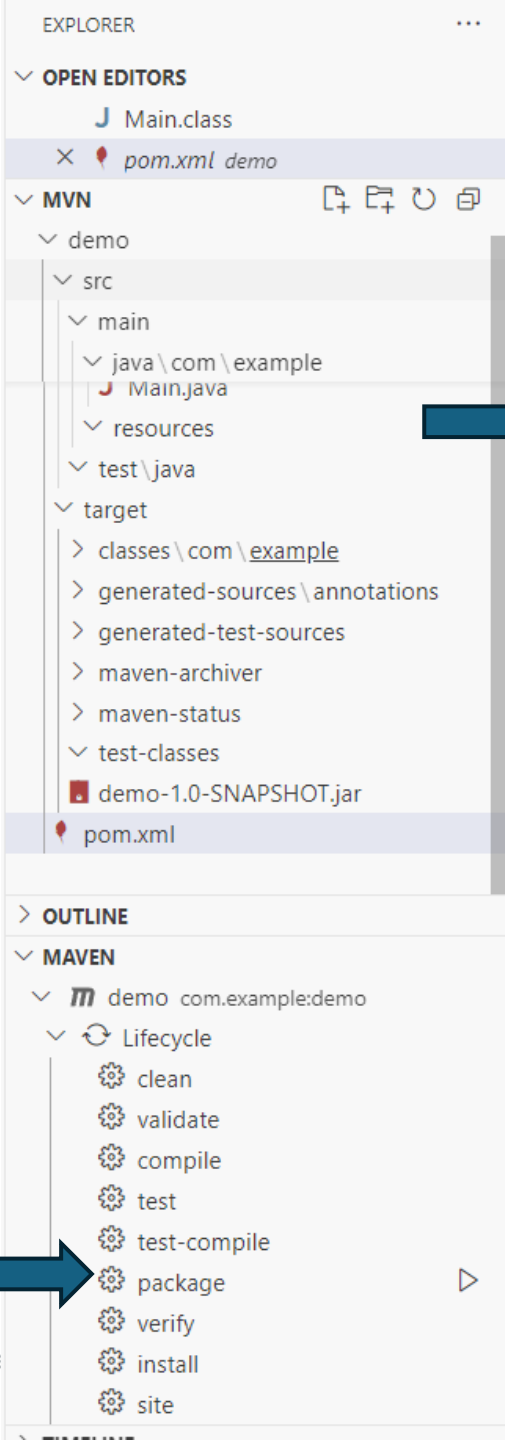
</project>
```



# lifecycle

- clean – deletes the target folder.
- compile – turns source files into class files.
- test – runs a project's unit tests.
- package – creates an artifact such as a JAR, ZIP or WAR file.
- install – moves a created artifact into a Maven repository.
- And much more ...

# Mvn package

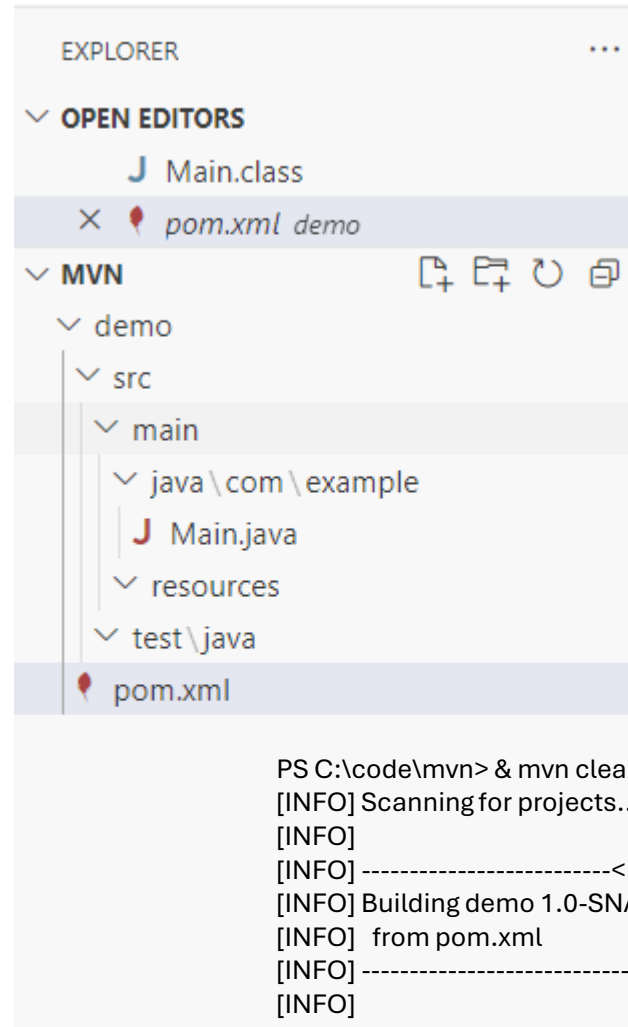
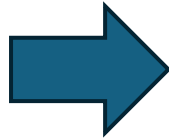
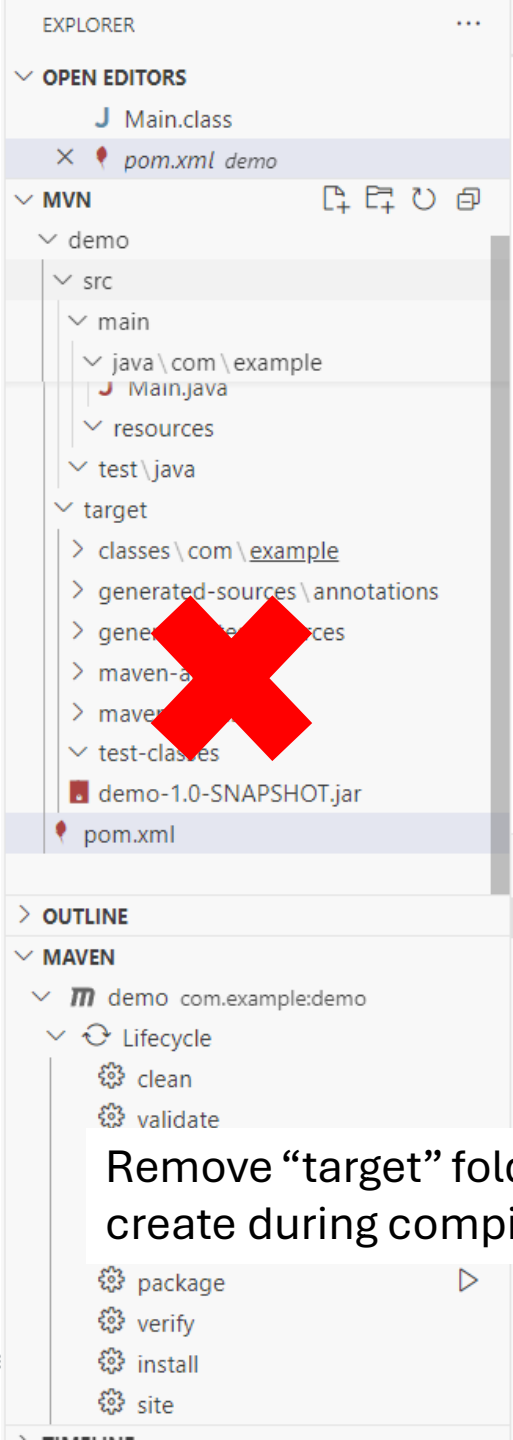


```
mvn package -f "c:\code\mvn\demo\pom.xml"
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.example:demo >-----
[INFO] Building demo 1.0-SNAPSHOT
[INFO] from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ demo ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource from src\main\resources to target\classes
[INFO]
[INFO] --- compiler:3.11.0:compile (default-compile) @ demo ---
[INFO] Changes detected - recompiling the module! :input tree
[WARNING] File encoding has not been set, using platform encoding Cp1252, i.e. build is platform dependent!
[INFO] Compiling 1 source file with javac [debug target 17] to target\classes
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ demo ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory c:\code\mvn\demo\src\test\resources
[INFO]
[INFO] --- compiler:3.11.0:testCompile (default-testCompile) @ demo ---
[INFO] Changes detected - recompiling the module! :dependency
[WARNING] File encoding has not been set, using platform encoding Cp1252, i.e. build is platform dependent!
[INFO]
[INFO] --- surefire:3.1.2:test (default-test) @ demo ---
[INFO]
[INFO] --- jar:3.3.0:jar (default-jar) @ demo ---
[INFO] Building jar: c:\code\mvn\demo\target\demo-1.0-SNAPSHOT.jar
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 4.719 s
[INFO] Finished at: 2025-05-06T14:33:45+01:00
[INFO]
PS C:\code\mvn>
```

# Mvn package

Create a jar file containing the compiled class ( to include dependencies may need to use a plugin )

```
mvn package -f "c:\code\mvn\demo\pom.xml"
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.example:demo
[INFO] Building demo 1.0-SNAPSHOT
[INFO] from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ demo ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource from src/main/resources to target/classes
[INFO]
[INFO] --- compiler:3.11.0:compile (default-compile) @ demo ---
[INFO] Changes detected - recompiling the module! :input tree
[WARNING] File encoding has not been set, using platform encoding Cp1252, i.e. build is platform dependent!
[INFO] Compiling 1 source file with javac [debug target 17] to target/classes
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ demo ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory c:\code\mvn\demo\src\test\resources
[INFO]
[INFO] --- compiler:3.11.0:testCompile (default-testCompile) @ demo ---
[INFO]
[INFO] --- jar:3.3.0:jar (default-jar) @ demo ---
[INFO] Building jar: c:\code\mvn\demo\target\demo-1.0-SNAPSHOT.jar
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 4.719 s
[INFO] Finished at: 2025-05-06T14:33:45+01:00
[INFO]
PS C:\code\mvn>
```



Remove “target” folder i.e. remove all files create during compilation and deployment

```
PS C:\code\mvn> & mvn clean -f "c:\code\mvn\demo\pom.xml"
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.example:demo >-----
[INFO] Building demo 1.0-SNAPSHOT
[INFO] from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- clean:3.2.0:clean (default-clean) @ demo ---
[INFO] Deleting c:\code\mvn\demo\target
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 0.384 s
[INFO] Finished at: 2025-05-06T14:40:02+01:00
[INFO]
PS C:\code\mvn>
```

# Mvn clean

# Some details on Pom.xml

# The pom.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 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>

  <name>my-app</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.release>17</maven.compiler.release>
  </properties>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.junit</groupId>
        <artifactId>junit-bom</artifactId>
        <version>5.11.0</version>
```



# The pom.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 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
```

```
<groupId>com.mycompany.app</groupId>
```

```
<artifactId>my-app</artifactId>
```

```
<version>1.0-SNAPSHOT</version>
```



The information of project

```
<name>my-app</name>
```

```
<!-- FIXME change it to the project's website -->
```

```
<url>http://www.example.com</url>
```



The official application name

```
<properties>
```

```
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
```

```
<maven.compiler.release>17</maven.compiler.release>
```

```
</properties>
```



Java environment settings

```
<dependencyManagement>
```

```
<dependencies>
```

```
<dependency>
```

```
<groupId>org.junit</groupId>
```

```
<artifactId>junit-bom</artifactId>
```

```
<version>5.11.0</version>
```

# The pom.xml: the environment

```
<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>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>

  <name>my-app</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.release>17</maven.compiler.release>
  </properties>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.junit</groupId>
        <artifactId>junit-bom</artifactId>
        <version>5.11.0</version>
```

Java environment settings



# The pom.xml: the environment

```
<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>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>

  <name>my-app</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.release>17</maven.compiler.release>
  </properties>

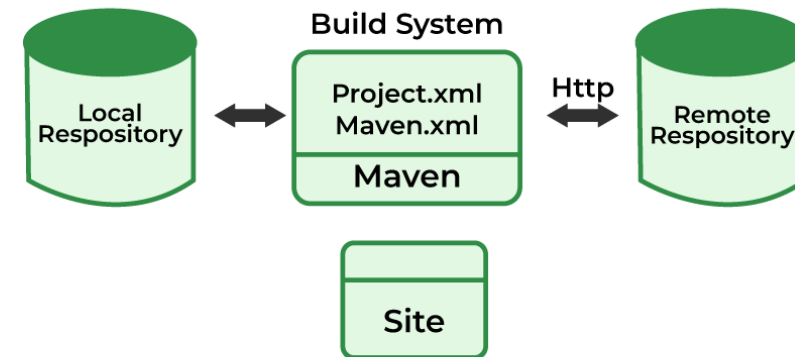
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.junit</groupId>
        <artifactId>junit-bom</artifactId>
        <version>5.11.0</version>
      </dependency>
    </dependencies>
  </dependencyManagement>
</project>
```

Java environment settings



# Maven and Dependencies repositories

- Repositories = Directories of packaged JAR files.
- local repository
  - directory on your machine's hard drive.
- Maven looks for dependencies
  - First in local Maven repository
  - Then from a central Maven repository and puts them in your local repository.



# The pom.xml: dependencies

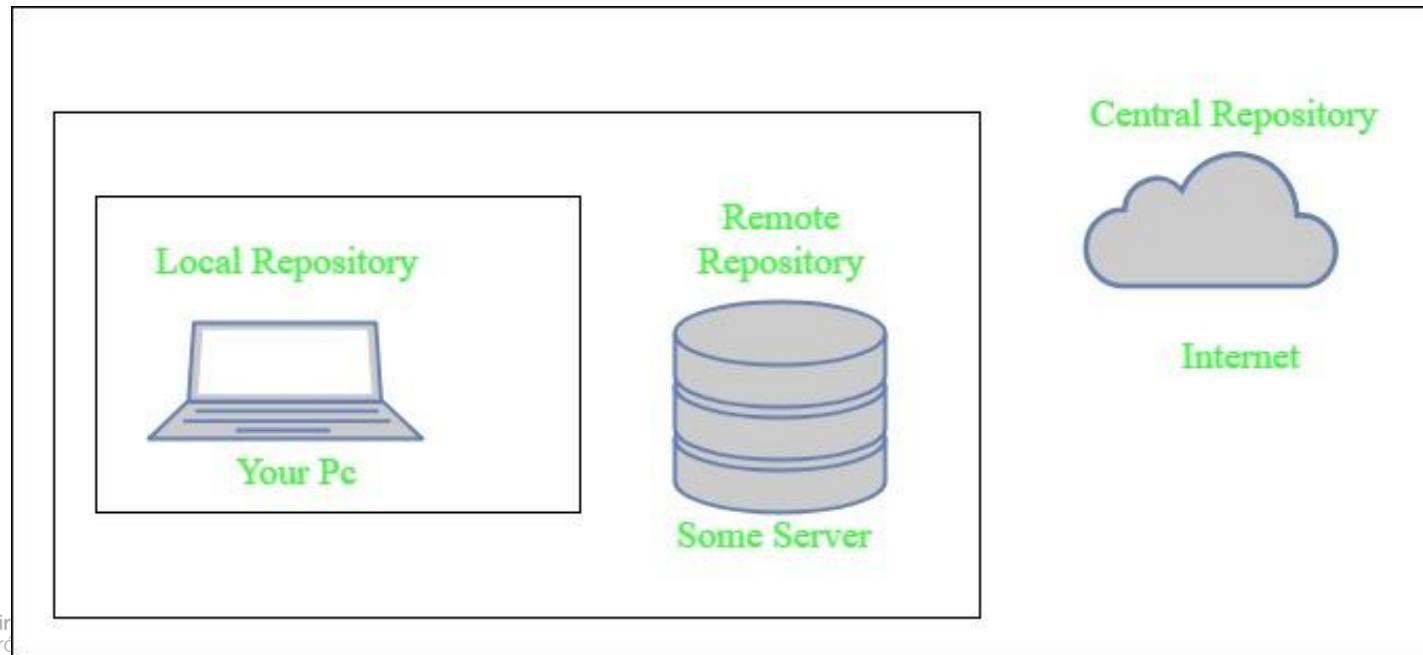
```
<dependencies>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <scope>test</scope>
  </dependency>
  <!-- Optionally: parameterized tests support -->
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-params</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>

<build>
  <pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->
    ... lots of helpful plugins
  </pluginManagement>
</build>
</project>
```

Depend on specific packages

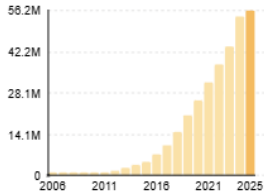
# Look for dependencies

- Locally
- Remote repository
- Internet



# https://mvnrepository.com/


**MVN REPOSITORY**

**Indexed Artifacts (56.2M)**

**Popular Categories**


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

**What's New in Maven**

**Jeddict AI Assistant**  
[io.github.jeddict](#) » [jeddict-ai](#) » 2.9Apache


Jeddict AI Assistant is a powerful and intuitive plugin designed for Apache NetBeans IDE. It integrates AI-powered assistance to help developers work more efficiently with Jakarta EE. This plugin provides real-time code suggestions, documentation insights, and best practices, significantly reducing development time. It is ideal for developers working on complex enterprise applications using Jakarta EE technologies, offering tailored recommendations based on the context ...

Last Release on May 4, 2025

**Markdown**  
[io.github.edadma](#) » [markdown](#) » 0.0.7ISC


markdown

Last Release on May 4, 2025

**Markdown**  
[io.github.edadma](#) » [markdown](#) » 0.0.7ISC

markdown

Last Release on May 4, 2025

**Markdown**  
[io.github.edadma](#) » [markdown](#) » 0.0.7ISC

markdown

Last Release on May 4, 2025

You can find here dependencies by using:

- groupId - the id of the project's group.
- artifactId - the id of the artifact (project)

# Search for dependency

The screenshot shows the Maven Repository search results for the term "jackson". The search bar at the top contains "jackson" and the "Search" button. Below the search bar, it says "Found 2332 results". The results are sorted by "Relevance". The first result is "Jackson Databind" by "com.fasterxml.jackson.core", with 33,255 usages and an Apache license. The "name" field is highlighted. Below the search results, there is a section for "Indexed Artifacts (56.2M)" and "Popular Categories". The "version" field is highlighted. A large blue arrow points from the "The jar" text to the "jar" file in the "Files" section of the "Jackson Databind" artifact details. The "Place in pom.xml" text is shown next to the XML snippet for the dependency.

```
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-core</artifactId>
  <version>2.19.0</version>
</dependency>
```

The jar

Place in pom.xml

```
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-core</artifactId>
  <version>2.19.0</version>
</dependency>
```



# The pom.xml: dependencies

```
<dependencies>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <scope>test</scope>
  </dependency>
  <!-- Optionally: parameterized tests -->
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-params</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>

<build>
  <pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->
    ... lots of helpful plugins
  </pluginManagement>
</build>
</project>
```

Depend on specific packages

```
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-core</artifactId>
  <version>2.19.0</version>
</dependency>
```



# https://central.sonatype.com/



## Find OSS Components

As stewards of Central for nearly 20 years and inventors of both software supply chain management and Next Repository, Sonatype knows that the integrity of your build is critical.

José Maria Fernandes (jfernand@ua.pt) is signed in

[Advanced Options](#)

## 2024 State of the Software Supply Chain

Open source consumption has exploded, with estimates placing **this year's downloads at over 6.6 trillion**. This reliance on open source components, now making up to 90% of the modern software application,



# https://central.sonatype.com/

The screenshot shows the Maven Central Repository page for the **jackson-core** artifact. The page has a dark blue header with the "maven central repository" logo and a search bar. Below the header, the artifact name "jackson-core" is displayed, along with its version "2.19.0" and a note that it is "Used in: 1566 components". The page is divided into tabs: "Overview" (selected), "Versions", "Dependents", and "Dependencies". The "Overview" section contains a "Description" box stating "Core Jackson processing abstractions (aka Streaming API), implementation for JSON" and a "Snippets" box showing an Apache Maven dependency snippet. A "Copy to clipboard" button is next to the snippet. The snippet content is: 

```
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-core</artifactId>
  <version>2.19.0</version>
</dependency>
```

maven central repository

maven central repository Search

**jackson-core** 2.19.0 pkg:maven/com.fasterxml...

Used in: 1566 components

Overview Versions Dependents Dependencies

Overview

**Description**

Core Jackson processing abstractions (aka Streaming API), implementation for JSON

**Snippets** Apache Maven Copy to clipboard

```
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-core</artifactId>
  <version>2.19.0</version>
</dependency>
```

Feedback

EXPLORER

OPEN EDITORS

- Main.class
- pom.xml demo**

MVN

- demo
  - src
    - main
      - java\com\example
        - Main.java**
        - resources
      - test\java
    - target
      - classes\com\example
        - Main.class
      - test-classes
    - pom.xml**

OUTLINE

MAVEN

  - demo com.example:demo
    - > Lifecycle
    - > Plugins
    - > Dependencies
    - > Favorites
    - > Profiles

Search bar: jackson-core

Input keywords to search artifacts from Maven Central Repository. (Press 'Enter' to confirm or 'Escape' to cancel)

URL: <http://maven.apache.org/POM/1.0.0/>

File Edit Selection View Go Run Terminal Help

mvn

EXPLORER

OPEN EDITORS

- Main.class
- pom.xml demo

MVN

- demo
  - src
    - main
      - java\com\example
        - Main.java
      - resources
    - test\java
    - target
      - classes
      - generated-sources
      - generated-test-sources
      - maven-archiver
      - maven-status
      - test-classes
  - demo-1.0-SNAPSHOT.jar
  - pom.xml

OUTLINE

MAVEN

- demo com.example:demo
  - Lifecycle
  - Plugins
  - Dependencies
    - com.fasterxml.jackson.core:jack...
  - Favorites
  - Profiles

Main.class pom.xml

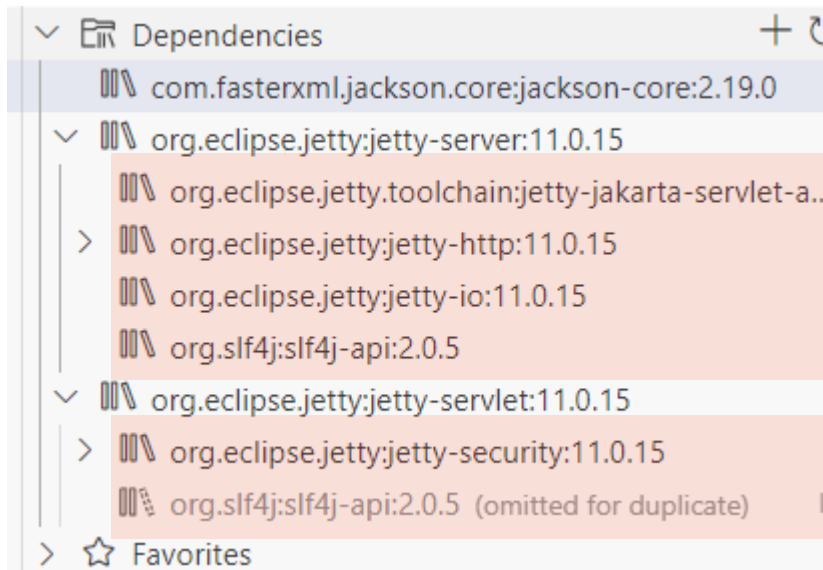
```
demo > pom.xml
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
10
11 <properties>
12 |   <maven.compiler.source>17</maven.compiler.source>
13 |   <maven.compiler.target>17</maven.compiler.target>
14 </properties>
15
16 <dependencies>
17 |   <dependency>
18 |     <groupId>com.fasterxml.jackson.core</groupId>
19 |     <artifactId>jackson-core</artifactId>
20 |     <version>2.19.0</version>
21 |   </dependency>
22 </dependencies>
23
24
25 <build>
26 |   <plugins>
27 |     <plugin>
28 |       <groupId>org.apache.maven.plugins</groupId>
29 |       <artifactId>maven-compiler-plugin</artifactId>
```

PROBLEMS 1

1 pom.xml demo

The build file has been changed and may need reload to make it effective. Java(0) [Ln 1, Col 1]

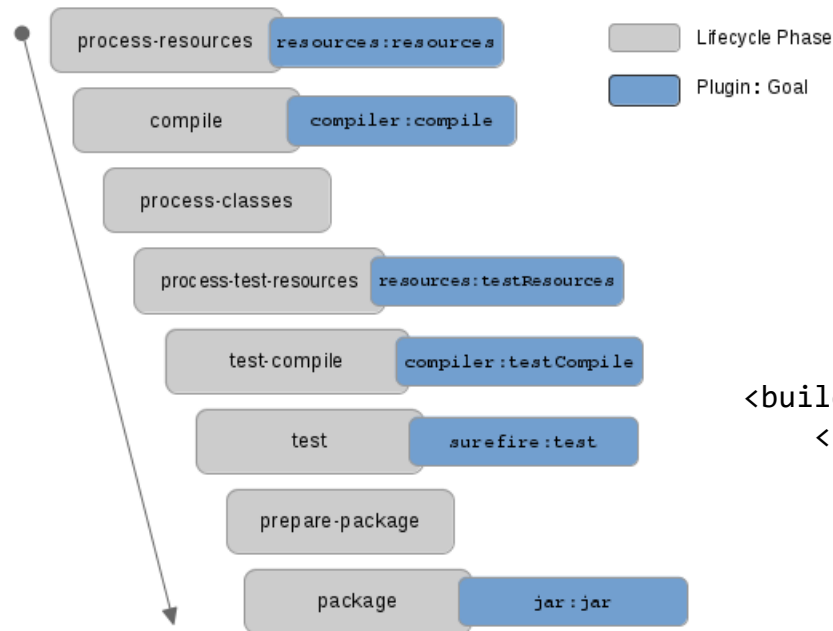
# Maven identifies (**in**)direct dependencies



```
<dependencies>
  <dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-core</artifactId>
    <version>2.19.0</version>
  </dependency>
  <dependency>
    <groupId>org.eclipse.jetty</groupId>
    <artifactId>jetty-server</artifactId>
    <version>11.0.15</version>
  </dependency>
  <dependency>
    <groupId>org.eclipse.jetty</groupId>
    <artifactId>jetty-servlet</artifactId>
    <version>11.0.15</version>
  </dependency>
</dependencies>
```

# Some notes on plugins

# The pom.xml: plugins

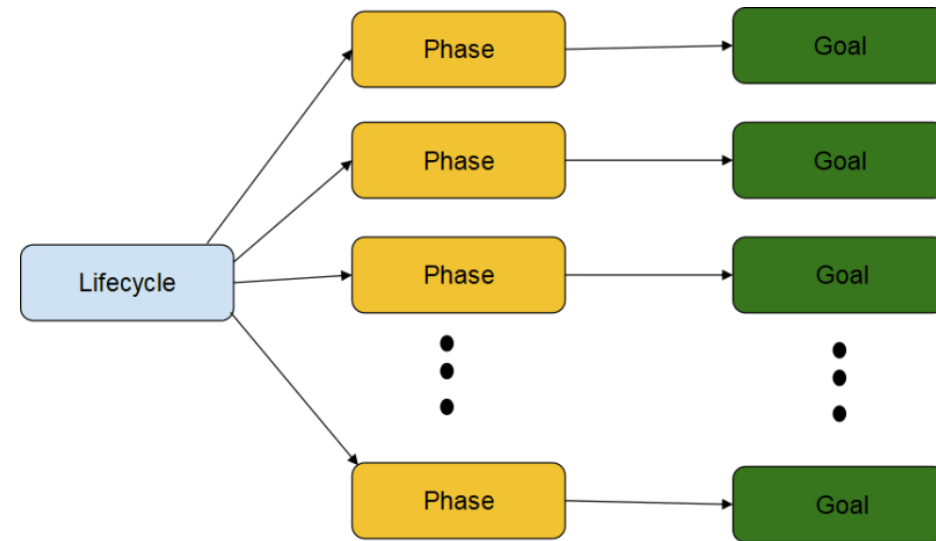


```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
      <configuration>
        <source>17</source>
        <target>17</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```



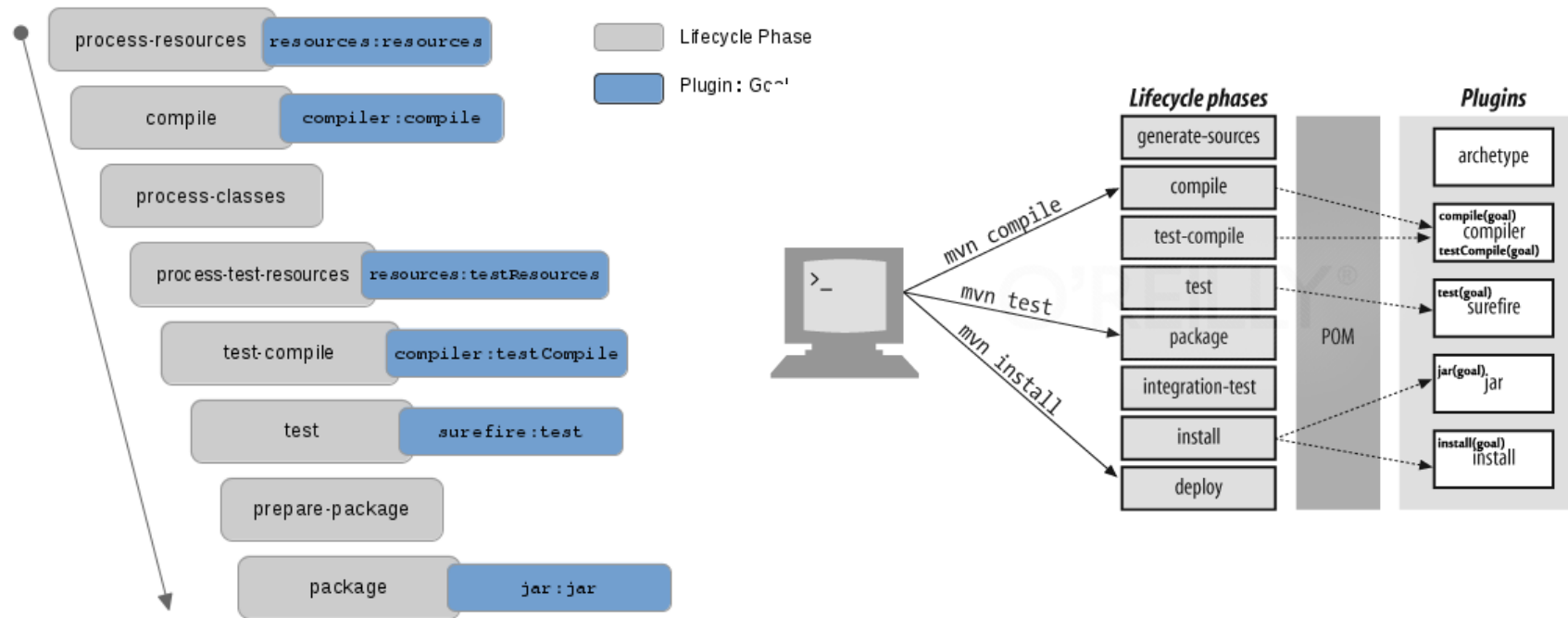
# Lifecycle and Phases

- To perform various build operations, Apache Maven provides a variety of phases that admins can invoke on a project through the mvn command.



**Maven™**

# Running MAVEN – lifecycle or phases



<https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html>

Welcome

[License](#)

ABOUT MAVEN

[What is Maven?](#)

[Features](#)

[Installation](#)

[Downloads](#)

[Use](#)

[Release Notes](#)

[Maven Daemon](#)

DOCUMENTATION

**Maven Plugins**

[Maven Extensions](#)

[Maven Tools](#)

[Index \(category\)](#)

[User Centre](#)

[Plugin Developer  
Centre](#)

[Maven Repository  
Centre](#)

[Maven Developer  
Centre](#)

[Books and Resources](#)

[Security](#)

COMMUNITY

## Available Plugins

Maven is - at its heart - a plugin execution framework; all work is done by plugins. Looking for a specific goal to execute? This page lists the core plugins and others. There are the build and the reporting plugins:

- **Build plugins** will be executed during the build and they should be configured in the `<build/>` element from the POM.
- **Reporting plugins** will be executed during the site generation and they should be configured in the `<reporting/>` element from the POM. Because the result of a Reporting plugin is part of the generated site, Reporting plugins should be both internationalized and localized. You can read more about the [localization of our plugins](#) and how you can help.

## Supported By The Maven Project

To see the most up-to-date list browse the Maven repository, specifically the [org/apache/maven/plugins](https://maven.apache.org/plugins) subdirectory. (Plugins are organized according to a directory structure that resembles the standard Java package naming convention)

Plugin	Type*	Version	Release Date	Description	Source Repository	Issue Tracking
Core plugins				Plugins corresponding to default core phases (ie. clean, compile). They may have multiple goals as well.		
<a href="#">clean</a>	B	3.4.1	2025-02-15	Clean up after the build.	<a href="#">Git / GitHub</a>	<a href="#">GitHub Issues</a>
<a href="#">compiler</a>	B	3.14.0	2025-02-17	Compiles Java sources.	<a href="#">Git / GitHub</a>	<a href="#">GitHub Issues</a>
<a href="#">deploy</a>	B	3.1.4	2025-02-23	Deploy the built artifact to the remote repository.	<a href="#">Git / GitHub</a>	<a href="#">GitHub Issues</a>
<a href="#">failsafe</a>	B	3.5.3	2025-03-31	Run the JUnit integration tests in an isolated classloader.	<a href="#">Git / GitHub</a>	<a href="#">GitHub Issues</a>
<a href="#">install</a>	B	3.1.4	2025-02-24	Install the built artifact into the local repository.	<a href="#">Git / GitHub</a>	<a href="#">GitHub Issues</a>
<a href="#">resources</a>	B	3.3.1	2023-03-21	Copy the resources to the output directory for including in the JAR.	<a href="#">Git / GitHub</a>	<a href="#">GitHub Issues</a>

Welcome

[License](#)

ABOUT MAVEN

[What is Maven?](#)

[Features](#)

[Installation](#)

[Downloads](#)

[Use](#) >

[Release Notes](#)

[Maven Daemon](#)

DOCUMENTATION

**Maven Plugins**

[Maven Extensions](#)

[Maven Tools](#)

[Index \(category\)](#)

[User Centre](#) >

[Plugin Developer  
Centre](#) >

[Maven Repository  
Centre](#) >

[Maven Developer  
Centre](#) >

[Books and Resources](#)

[Security](#)

COMMUNITY

## Available Plugins

Maven is - at its heart - a plugin execution framework; all work is done by plugins. Looking for a specific goal to execute? This page lists the core plugins and others. There are the build and the reporting plugins:

- **Build plugins** will be executed during the build and they should be configured in the `<build/>` element from the POM.
- **Reporting plugins** will be executed during the site generation and they should be configured in the `<reporting/>` element from the POM. Because the result of a Reporting plugin is part of the generated site, Reporting plugins should be both internationalized and localized. You can read more about the [localization of our plugins](#) and how you can help.

## Supported By The Maven Project

To  
str

**Focus on those to ease the  
deploy of jar**

ding to a directory

							Issue Tracking
Core plugins				Plugins corresponding to default core phases (ie. clean, compile). They may have multiple goals as well.			
<code>clean</code>	B	3.4.1	2025-02-15	Clean up after the build.	<a href="#">Git / GitHub</a>		<a href="#">GitHub Issues</a>
<code>compiler</code>	B	3.14.0	2025-02-17	Compiles Java sources.	<a href="#">Git / GitHub</a>		<a href="#">GitHub Issues</a>
<code>deploy</code>	B	3.1.4	2025-02-23	Deploy the built artifact to the remote repository.	<a href="#">Git / GitHub</a>		<a href="#">GitHub Issues</a>
<code>failsafe</code>	B	3.5.3	2025-03-31	Run the JUnit integration tests in an isolated classloader.	<a href="#">Git / GitHub</a>		<a href="#">GitHub Issues</a>
<code>install</code>	B	3.1.4	2025-02-24	Install the built artifact into the local repository.	<a href="#">Git / GitHub</a>		<a href="#">GitHub Issues</a>
<code>resources</code>	B	3.3.1	2023-03-21	Copy the resources to the output directory for including in the JAR.	<a href="#">Git / GitHub</a>		<a href="#">GitHub Issues</a>

# Fat Jar Files

- Uber Jars or executable Jars,
- take the concept of Jar files a step further by
- include **all dependencies within the archive.**
- self-contained approach
  - simplifies deployment,
  - as the application can be run **without relying on external libraries.**

## Creating a Fat Jar File:

Using tools like Maven or Gradle, you can generate a Fat Jar easily.

# Use proper plugin in pom.xml

If you're using Maven, use the `maven-shade-plugin` in your `pom.xml`:

```
<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>
            <transformers>
              <transformer
                implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">

                <mainClass>EmbeddedJettyExample</mainClass> <!-- Replace with your full
class name -->

              </transformer>
            </transformers>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

# Create the jar

```
bash
```

[Copy](#)[Edit](#)

```
mvn clean package
```

This will produce a **fat JAR** (e.g., `your-artifact-name-jar-with-dependencies.jar`) in the `target/` directory.

# run

## 2. Run the JAR

```
bash
```

[Copy](#)[Edit](#)

```
java -jar target/your-artifact-name-jar-with-dependencies.jar
```

Replace the file name with the actual one Maven generates.

It will run the class plugin field...

```
<mainClass>EmbeddedJettyExample</mainClass><!-- Replace with  
your full class name -->
```



# and if I want to run a specific class?

and if I want to run a specific class within the jar ?

Great question! If you want to run a **specific class within a JAR**, not necessarily the one defined as the **Main-Class** in the manifest, you can do it like this:

## ✓ Command to Run a Specific Class from a JAR

bash

Copy

Edit

```
java -cp your-jar-file.jar your.package.ClassName
```



- **-cp** (or **-classpath**) tells Java to use your JAR for classes.
- Replace **your-jar-file.jar** with the name of your JAR.
- Replace **your.package.ClassName** with the **fully qualified name** of the class you want to run.



# and if I want to run a specific class?

and if I want to run a specific class within the jar ?

Great question! If you want to run a **specific class within a JAR**, not necessarily the one defined as the **Main-Class** in the manifest, you can do it like this:

## ✓ Command to Run a Specific Class from a JAR

bash

Copy

Edit

```
java -cp your-jar-file.jar your.package.ClassName
```

- **-cp** (or **-classpath**) tells Java to use your JAR for classes.
- Replace **your-jar-file.jar** with the name of your JAR.
- Replace **your.package.ClassName** with the **fully qualified name** of the class you want to run.

# If your jar does not have all dependencies...

If the class requires external dependencies (like Jetty), and your JAR is **not a fat JAR**, then you'll need to add those dependencies to the classpath too, like this:

```
java -cp "myserver.jar:lib/*" com.example.EmbeddedJettyExample
```

The jar with your code ...

`myserver.jar`

directory(ies) containing all dependencies that are OUTSIDE the jar

`lib/*`

# If your jar does not have all dependencies...

If the class requires external dependencies (like Jetty), and your JAR is **not a fat JAR**, then you'll need to add those dependencies to the classpath too, like this:

```
java -cp "myserver.jar:lib/*" com.example.EmbeddedJettyExample
```

On Windows, use `;` instead of `::`:

```
bash
```

```
java -cp "myserver.jar;lib/*" com.example.EmbeddedJettyExample
```

# If your jar does not have all dependencies...

If the class requires external dependencies (like Jetty), and your JAR is **not a fat JAR**, then you'll need to add those dependencies to the classpath too, like this:

```
java -cp "myserver.jar:lib/*" com.example.EmbeddedJettyExample
```

The jar with your code ...

`myserver.jar`

directory(ies) containing all dependencies that are OUTSIDE the jar

`lib/*`

The

# How to Create an Executable JAR with Maven

## 1. Overview

In this quick tutorial, we'll focus on **packaging a Maven project into an executable Jar file**.

When creating a *jar* file, we usually want to run it easily, without using the IDE. To that end, we'll discuss the configuration and pros/cons of using each of these approaches for creating the executable.

# How to Create an Executable JAR with Maven

The most important aspect here is the type — to create an executable *jar*, double-check the configuration uses a *jar* type.

- Now we can start using the various solutions.

```
<modelVersion>4.0.0</modelVersion>  
<groupId>com.baeldung</groupId>  
<artifactId>core-java</artifactId>  
<version>0.1.0-SNAPSHOT</version>  
<packaging>jar</packaging>
```

# Goal – copy dependencies

- specify the **goal copy-dependencies**,
- copy dependencies into the specified **outputDirectory** i.e. a folder **libs** inside the project build directory (which is usually the **target** folder).

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-dependency-plugin</artifactId>
  <executions>
    <execution>
      <id>copy-dependencies</id>
      <phase>prepare-package</phase>
      <goals>
        <goal>copy-dependencies</goal>
      </goals>
      <configuration>
        <outputDirectory>
          ${project.build.directory}/libs
        </outputDirectory>
      </configuration>
    </execution>
  </executions>
</plugin>
```



# Maven jar plugin

- the *manifest* configuration.  
**We add a classpath, with all dependencies (folder *libs/*), and provide the information about the main class.**
- need to provide a fully qualified name of the class, which means it will include the package name.

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-jar-plugin</artifactId>
  <configuration>
    <archive>
      <manifest>
        <addClasspath>true</addClasspath>
        <classpathPrefix>libs/</classpathPrefix>
        <mainClass>
          com.baeldung.executable.ExecutableMavenJar
        </mainClass>
      </manifest>
    </archive>
  </configuration>
</plugin>
```

# Apache Maven Assembly Plugin

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-assembly-plugin</artifactId>
  <executions>
    <execution>
      <phase>package</phase>
      <goals>
        <goal>single</goal>
      </goals>
      <configuration>
        <archive>
          <manifest>
            <mainClass>
              com.baeldung.executable.ExecutableMavenJar
            </mainClass>
          </manifest>
        </archive>
        <descriptorRefs>
          <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Similarly to the manual approach, we need to provide the information about the main class. The difference is that the Maven Assembly Plugin will automatically copy all required dependencies into a *jar* file.

In the *descriptorRefs* part of the configuration code, we provided the name that will be added to the project name.

Output in our example will be named *core-java-jar-with-dependencies.jar*.

- **pros** – dependencies inside the *jar* file, one file only
- **cons** – basic control of packaging our artifact, for example, there is no class relocation support

# Apache Maven Shade Plugin

There are three main parts to this configuration.

First, `<shadedArtifactAttached>` marks all dependencies to be packaged into the *jar*.

Second, we need to specify [the transformer implementation](#); we used the standard one in our example.

Finally, we need to specify the main class of our application.

The output file will be named *core-java-0.1.0-SNAPSHOT-shaded.jar*, where *core-java* is our project name followed by snapshot version and plugin name.

- **pros** – dependencies inside the *jar* file, advanced control of packaging our artifact, with shading and class relocation
- **cons** – complex configuration (especially if we want to use advanced features)

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-shade-plugin</artifactId>
  <executions>
    <execution>
      <goals>
        <goal>shade</goal>
      </goals>
      <configuration>
        <shadedArtifactAttached>true</shadedArtifactAttached>
        <transformers>
          <transformer implementation=
            "org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
            <mainClass>com.baeldung.executable.ExecutableMavenJar</mainClass>
          </transformer>
        </transformers>
      </configuration>
    </execution>
  </executions>
</plugin>
```

# One Jar Maven Plugin

```
<plugin>
  <groupId>com.jolira</groupId>
  <artifactId>onejar-maven-plugin</artifactId>
  <executions>
    <execution>
      <configuration>
        <mainClass>org.baeldung.executable.
          ExecutableMavenJar</mainClass>
        <attachToBuild>true</attachToBuild>
        <filename>
          ${project.build.finalName}.${project.packaging}
        </filename>
      </configuration>
      <goals>
        <goal>one-jar</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Another option to create an executable *jar* is the One Jar project.

- This provides a custom class loader that knows how to load classes and resources from jars inside an archive, instead of from *jars* in the filesystem.

As shown in the configuration, we need to specify the main class and attach all dependencies to the build, by using *attachToBuild = true*.

Also, we should provide the output filename. Moreover, the goal for Maven is *one-jar*. Please note that One Jar is a commercial solution that will make dependency *jars* not expanded into the filesystem at runtime.

- **pros** – clean delegation model, allows classes to be at the top level of the One Jar, supports external *jars*, and can support Native libraries
- **cons** – not actively supported since 2012

# What was addressed

- Maven ( focus in java )
  - pom.xml
  - dependencies and plugins
  - adding dependencies
- deploy a java application
  - jar and fat jar
  - the maven pluggins to deploy
-

# The END

# Some links on maven

- Apache Maven Series
  - <https://www.baeldung.com/maven-series>
- Getting Started With Maven - A Beginner's Guide to Efficient Java Build Management
  - <https://hackernoon.com/getting-started-with-maven-a-beginners-guide-to-efficient-java-build-management>
- Maven in 5 Minutes
  - <https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html>
- Maven Getting Started Guide
  - <https://maven.apache.org/guides/getting-started/>