

# Maven

## Introduction

Maven est un outil de construction pour Java. Il permet d'importer des dépendances, et d'automatiser la construction et l'empaquetage d'un artéfact Java.

Tous les IDE Java intègrent Maven (nativement ou via des plugins) et permettent d'utiliser Maven via leur interface graphique. Ce tutoriel utilisera des instructions de l'interface en ligne de commande de Maven afin d'être indépendant de l'IDE.

## Installation

### Avec Chocolatey

Vous pouvez utiliser chocolatey pour installer maven :

```
choco install maven -y
```

### Installation manuelle

Dirigez vous sur [ce lien](#) et téléchargez la "binary zip archive". Décompressez là ensuite et stockez son contenu dans l'endroit où vous rangez vos outils ( `C:/tools` par exemple).

Ajoutez ensuite le dossier `bin` contenu dans le répertoire de maven que vous venez de décompresser au PATH.

Testez ensuite votre installation en tapant la commande suivante dans une invite de commande :

```
mvn --version
```

# Archétypes

Maven permet de créer des projets à partir de d'archétypes pour gagner du temps sur la mise en place de l'arborescence de fichier du projet, en utilisant une convention et donc les bonnes pratiques de la communauté Java.

Pour créer un projet selon un archétype, utilisez la commande suivante :

```
mvn archetype:generate -DarchetypeArtifactId=<nom archétype>
```

A l'exécution de cette commande, plusieurs informations vous sont demandées :

- `groupId` : Votre espace de noms. Par convention il s'agit de votre nom de domaine personnel, ou alors votre pseudo précédé d'une extension de domaine : `fr.ombrelin`
- `artifactId` : Le nom de votre projet
- `version` : Version du projet (défaut : `1.0-SNAPSHOT`)
- `package` : Nom du package (défaut : le `groupId`)

L'achétype le plus utilisé est `maven-archetype-quickstart`. Il définit un projet Java SE basique. Pour créer un project selon cet artéfact :

```
mvn archetype:generate -DarchetypeArtifactId=maven-archetype-quickstart
```

## Project Object Model

Après la création du projet depuis un archétype, maven génère un dossier contenant le projet. Ce dossier contient l'arborescence des fichiers ainsi qu'un fichier, le fichier `pom.xml`. Ce fichier contient la configuration de maven.

La balise `<project>` est la racine du `pom.xml`.

## Méta données du projet

Les premières balises contiennent les méta données du projet, notamment les informations que vous avez enseignées au moment de se création.

## Gestion des dépendances

La balise `<dependencies>` permet de lister les dépendances du projet, c'est à dire les librairies qui sont utilisées par le projet. Pour installer une dépendance, il suffit d'ajouter la balise `<dependency>` qui lui correspond ici ; maven s'occupera ensuite de télécharger les archives jar de la librairie et de les ajouter au classpath du projet.

Par exemple, pour ajouter JUnit 5 au projet ajoutez cette balise aux dépendances :

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-engine</artifactId>
  <version>5.5.2</version>
  <scope>test</scope>
</dependency>
```

puis exécutez `mvn dependency:resolve` afin de télécharger et ajouter au classpath les dépendances listées.

# Lifecycle Goals

La construction avec maven si un certain nombre de *lifecycles*. On peut exécuter différent *goals* de *lifecycles* afin de faire différentes actions. Les plus utiles sont :

- `compile` : Compile le code source en bytecode
- `test` : Exécute les tests unitaires
- `package` : Package le projet en archive JAR

# Plugins

Les plugins permettant d'ajouter de nouveaux *goals* afin de faire différentes actions, comme par exemple générer un JAR exécutable. Pour ce faire, ajoutez la balise build suivante à votre `pom.xml` :

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-assembly-plugin</artifactId>
      <version>3.1.0</version>
      <configuration>
```

```
<descriptorRefs>
  <descriptorRef>jar-with-dependencies</descriptorRef>
</descriptorRefs>
<archive>
  <manifest>
    <mainClass>le nom complet de votre classe principale (nomdupackage.nomdelaclass)</mainClass>
  </manifest>
</archive>
</configuration>
<executions>
  <execution>
    <id>make-assembly</id>
    <phase>package</phase>
    <goals>
      <goal>single</goal>
    </goals>
  </execution>
</executions>
</plugin>
</plugins>
</build>
```

Exécutez ensuite un `mvn clean package` . L'archive JAR exécutable sera générées dans le dossier target.

---

Revision #4

Created 21 December 2020 21:22:53 by Arsène Lapostolet

Updated 21 December 2020 22:55:00 by Arsène Lapostolet