# Maven

Olivier Cailloux

LAMSADE, Université Paris-Dauphine

Version du 11 août 2016

LAMSADE
UMR CNRS 7243
laboratoire d'analyse et modélisation de systèmes pour l'aide à la décision

DAUPHINE
UNIVERSITÉ PARIS

PSL
RESEARCH
UNIVERSITY

# Présentation

- Apache Maven
- Outil de gestion de projet
- Principalement gestion de dépendances
- Description de votre projet via POM (Project Object Model)
- Convention over configuration : peu de configuration grâce aux valeurs par défaut
- Dépôt central avec publications open source
- Fortement basé sur plugins

# Le POM

- Fichier XML
- Décrit un projet ou module et comment le construire (*build*)
- Un projet *peut* être composé de modules

## Exemple de POM

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="…"  xsi:schemaLocation="…">
  <modelVersion>4.0.0</modelVersion>
  <groupId>myGroupId</groupId>
  <artifactId>myArtifactId</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</project>
```

# Structure du projet

Structure déterminée normalement par *conventions*

## Arborescence de base

```
pom.xml
/src
    /main
        /java
        /resources
    /test
        /java
        /resources
```

## Répertoires de base

| | |
|---|---|
| `/src/main/...` | fichiers code et ressources « normales » |
| `/src/test/...` | fichiers code et ressources pour tests |
| `.../java` | code java |
| `.../resources` | images, etc., devant être dans classpath |

# Accès aux ressources en Java

Rappel : accès aux ressources en Java (indépendant de Maven)

## Exemple d'accès aux ressources

- `MyClass` dans package `com.mypkg`
- `URL url = MyClass.class.getResource("ploum.txt")`
- `url` désigne la ressource de chemin `/com/mypkg/ploum.txt`
- Ensuite : `url.openStream()` (ou toute autre exploitation)

## Positionnement de ressources avec Maven

Placer le fichier dans
`/src/main/resources/com/mypkg/ploum.txt`

# Cycles de vie Maven

- Maven utilise des cycles de vie
- Cycles embarqués : default ; clean ; site
- Cycle : ensemble ordonné de *phases*
- Cycle “clean” contient essentiellement phase “clean”
- Cycle “site” contient essentiellement phase “site”
- Lors exécution de Maven, préciser une phase (Maven en déduit le cycle)

# Cycle “default”

Phases (non exhaustif) dans cycle “default” :

| | |
|---|---|
| validate | valide informations du projet |
| process-resources | copie vers destination |
| compile | compilation du code source |
| test | lancement des tests |
| package | création d’un paquet |
| integration-test | tests d’intégration |
| verify | vérification de la validité du paquet |
| install | installation en local |
| deploy | déploiement dans dépôt configuré |

# Phases

- Chaque phase associée à un ensemble de plugins et d'objectifs (*goals*)
- Phase process-resources associée par défaut à plugin **Resources**, objectif `resource`
- Phase test associée par défaut à plugin **Surefire**, objectif `test`
- Phase package associée par exemple à plugin **JAR**, objectif `JAR`

## Exécution

Lancement de Maven avec `mvn phasechoisie` :

- Maven détecte de quel cycle il s'agit
- Maven exécute toutes les phases jusqu'à "phasechoisie"
- Exemple : exécution systématique de `test` avant `package`

# Dépendances

- Maven permet de gérer les « dépendances »
- Bibliothèques dont votre code dépend
- Pour compiler (dépendance statique) ; s'exécuter ; pour tests uniquement...
- Une bibliothèque dont vous dépendez peut elle-même avoir des dépendances
- Maven gère ces dépendances transitives pour vous !
- Dépendances prises par défaut dans Maven Central Repository
- Dans POM : section `<dependencies>`
- Dans cette section : ajouter une section `<dependency>` pour chaque dépendance à gérer

# Dépendances : exemples

## Exemple : dépendance vers Google Guava

```
<dependency>
  <groupId>com.google.guava</groupId>
  <artifactId>guava</artifactId>
  <version>19.0</version>
</dependency>
```

## Exemple : dépendance vers junit

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
```

# Dépendances dans POM

- Trouver `groupId` et `artifactId` : voir site du projet
- Trouver `version` : voir Central
- Presque tous les projets Java récents font une release Maven

Portées (liste non exhaustive) :

`compile` Par défaut

`test` Bibliothèque incluse uniquement lors phase tests

`runtime` Bibliothèque incluse uniquement lors exécution, pas lors compilation

# Configuration des plugins

- Voir Liste pour plugins de Apache
- Configuration parfois utile
- Dans POM : ajouter section
  `<build><plugins><plugin>…</plugin></plugins></build>`
- Exemple : pour configurer la compilation, voir la page “Apache Maven Compiler Plugin”

# Propriétés

- Propriété `ma propriété` : accessible via `${ma propriété}`
- Nommage souvent hiérarchique : `catégorie.sous-catégorie.nom-propriété`

Dans POM :

```
<properties>
  <cat.etc.prop1>valeur1</cat.etc.prop1>
  <cat.etc.prop2>valeur2</cat.etc.prop2>
</properties>
(...)
<balise-quelconque>${cat.etc.prop1}</balise-quelconque>
```

# Conventions et configurations classiques

- Utiliser comme groupeId un nom unique : généralement un nom de domaine inversé
- Le paquet de base de toutes les classes doit être ce nom
- Indiquer propriété `project.build.sourceEncoding` avec valeur `UTF-8`
- Indiquer propriétés `maven.compiler.source` et `maven.compiler.target` avec valeur `1.8`
- Canevas simples disponibles ici

# Maven et Eclipse

- M2Eclipse (m2e) fournit support Maven pour Eclipse
- Maven embarqué
- Wizards pour démarrer ou importer un projet maven
- Conseil : utiliser l'option `Maven / Update project / Update project configuration from pom.xml` pour configuration correcte du projet dans Eclipse

# Références

- Tutoriel Apache Maven
- Apache Maven Cookbook, Raghuram Bharathan, 2015

Omis dans cette présentation :

- Archetypes (maven-archetype-plugin)
- Packaging
- Assembly (maven-assembly-plugin)

# Licence