université Paris Ouest Nanterre La Défense

Programmation Orientée Objet

Introduction à Maven

Lom Messan Hillah

Université Paris Ouest Nanterre La Défense
UFR SEGMI - 2013/2014

Objectifs

- Qu'est-ce que Maven ?
- Cycles de production
- Architecture
- Plugins
- Repositories
- Project Object Model

Sommaire

Sommaire

Les outils de compilation avant Maven

- Utilisation de `javac` pour compiler les projets
- Utilisation d'un IDE pour compiler les projets (Eclipse, NetBeans, IntelliJ)
- Utilisation de Apache Ant (Another Neat Tool, http://ant.apache.org)
- Utilisation de GNU Make (http://www.gnu.org/software/make/)

La compilation de projets Java avant Maven

- Chaque projet a son propre fichier de compilation Ant
- Pas de réutilisation
- Les jars des dépendances étaient versionnés dans les SCM (CVS, SVN)
- Pas de réutilisation de jars entre projets
- Déploiement par scripts ad hoc dans différents langages, rendant la maintenance difficile

Origine et objectif de Maven

- Projet Apache depuis 2001, initialement pour simplifier la phase de production du projet Jakarta Turbine
- Génère des artefacts déployables à partir de code source
- Gestion unifiée des dépendances, compilation, test, empaquetage et distribution automatiques de logiciel en Java
- Maven est beaucoup plus un outil de **gestion de production de projet logiciel** qu'un simple outil de compilation, ou de "build".
- Maven peut également générer de la documentation : rapports et sites web.
- Gestion de projets multi-modules

Pourquoi utiliser Maven ?

- Gestion unifiée de toutes les dépendances de votre projet et de son cycle de production
- Production automatique de rapports de test, incluant la couverture.
- Élimination du stockage ad hoc des dépendances sur une machine de développement, elles sont récupérées d'un dépôt central
- Privilégie la convention à la configuration (qui reste néanmoins possible), ce qui signifie moins de configuration
- Gestion de production de grands logiciels (nombreux artefacts et modules)
- Configuration d'assemblages types de logiciels ayant le même profil (e.g projets web app).
- Utilisation d'un fichier de configuration de projet en XML, facile à partager.

Maven selon la Fondation Apache

Unifier la gestion de production du logiciel

Maven permet la production d'un projet logiciel en utilisant son fichier de configuration POM (Project Object Model), ainsi qu'un ensemble de greffons (plugins) qui sont réutilisés par tous les projets utilisant Maven. Une fois que vous savez comment produire un projet logiciel avec Maven, vous saurez comment Maven réalise la gestion de production des projets. Ceci vous permettra de gagner du temps lorsque vous naviguerez dans beaucoup de projets similaires.

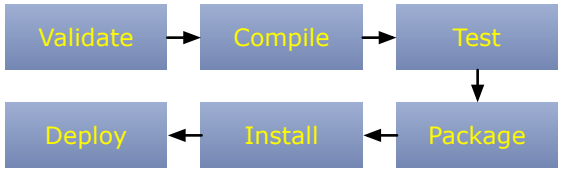
Sommaire

Cycles de production de Maven

- La production de projets avec Maven repose sur la mise en œuvre de workflows définissant le processus de production et de distribution des artefacts du projet (e.g. jar, war).
- Il existe 3 cycles pré-définis :
  - **Default** : gère la production du logiciel et son déploiement
  - **Clean** : gère le nettoyage du projet
  - **Site** : gère la production du site (web) du projet

Phases dans les cycles de production

- Chaque cycle comporte des **phases**
- Un exemple d'enchaînement de phases dans le cycle Default pourrait être :



Invocation d'une phase

- L'invocation d'une phase spécifique provoque également l'exécution des phases précédant la phase invoquée.
- Ex. : dans l'enchaînement de phases présenté sur la diapositive précédente, l'invocation de `$ mvn test`[1], provoque l'exécution des phases :

1. $ symbolise le prompt de la ligne de commande

Phases du cycle de production Default (1/4)

Phase	Description
`validate`	Valide la structure du projet et la configuration nécessaire pour sa production
`generate-sources`	Génère tout code source utile pour inclusion dans la phase compilation sa classe
`process-sources`	Effectue divers traitement sur le code source, par exemple expansion de propriétés de configuration ou filtrage de valeurs
`generate-resources`	Génère les ressources pour inclusion dans le paquetage
`process-resources`	Copie les fichiers ressources dans le répertoire destination, et les prépare au packaging
`compile`	Compile le code source
`process-classes`	Post-traitement des classes générées à la compilation, par exemple pour optimiser le bytecode

Phases du cycle de production Default (2/4)

Phase	Description
`generate-test-sources`	Génère tout code source de test pour inclusion dans la phase de compilation des tests
`process-test-sources`	Effectue divers traitement sur le code source des tests, par exemple expansion de propriétés de configuration ou filtrage de valeurs
`generate-test-resources`	Crée les ressources pour le test
`process-test-resources`	Copie les fichiers ressources des tests dans le répertoire destination
`test-compile`	Compile le code source
`test`	Exécute les tests, en utilisant le framework de test adapté. Ces tests ne requièrent pas que le code soit empaqueté ou déployé.

Phases du cycle de production Default (3/4)

Phase	Description
`prepare-package`	Effectue toutes les opérations nécessaires à l'empaquetage du logiciel avant qu'il ne soit effectivement empaqueté. Cela donne lieu à une version décompressée du paquetage
`package`	Empaquette le logiciel compilé dans le format de distribution choisi : jar, war ou ear
`pre-integration-test`	Effectue toutes les opérations requises avant que les tests d'intégration ne soient exécutés, par exemple configurer l'environnement requis
`integration-test`	déploie si nécessaire le logiciel empaqueté dans un environnement où les tests d'intégration sont exécutés
`post-integration-test`	Effectue toutes les opérations requises après l'exécution des tests d'intégration, par exemple nettoyer l'environnement de test

Phases du cycle de production Default (4/4)

Phase	Description
`verify`	Effectue toutes les vérifications nécessaires sur le paquetage pour s'assurer qu'il est valide et répond à des critères de qualité
`install`	Installe le paquetage dans le dépôt Maven local, afin qu'il puisse être utilisé comme dépendance dans d'autres projets, locallement
`deploy`	Copie le paquetage final dans le dépôt distant, afin de le partager avec d'autres développeurs et projets. Ceci est particulièrement utile pour le travail en équipe et lors de sorties officielles de versions du logiciel

Phases du cycle de nettoyage

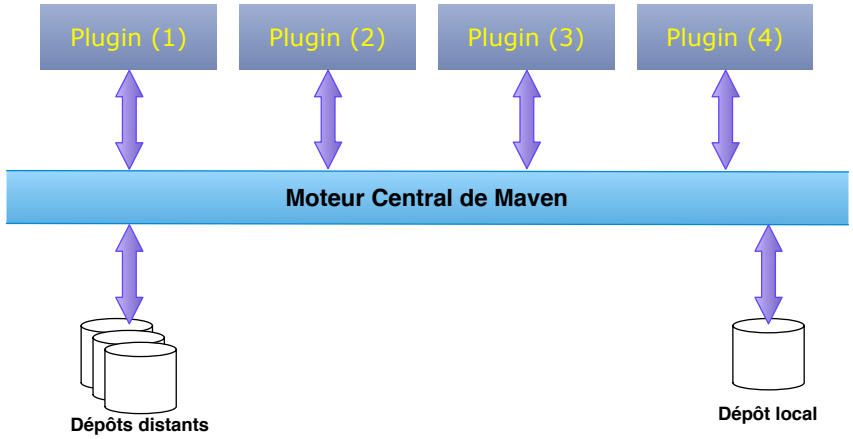
Phase	Description
`pre-clean`	Effectue toutes les opérations nécessaires avant que le nettoyage proprement dit ne soit exécuté
`clean`	Nettoie le répertoire destination, là où le logiciel compilé a été placé
`post-clean`	Effectue toutes les opérations nécessaires après le nettoyage

Phases du cycle de création du site de documentation

Phase	Description
`pre-site`	Exécute toute cible nécessaire avant la création du site
`site`	Crée le site
`post-site`	Exécute toute cible nécessaire après la création du site
`site-deploy`	Déploie le site

Sommaire

Architecture de Maven



Architecture de Maven : Moteur central

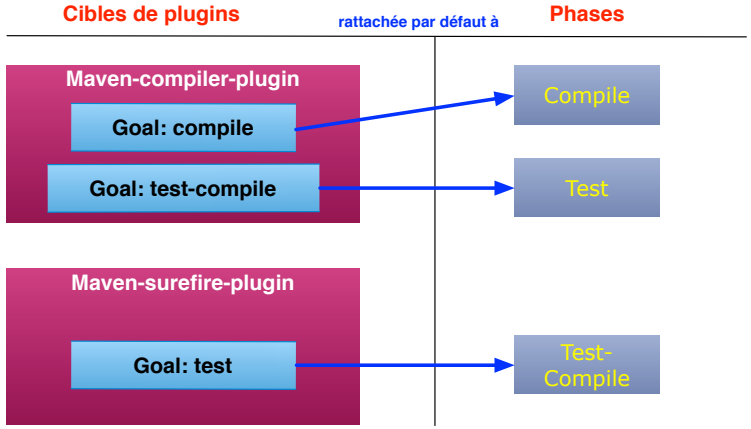
- Configuration du projet selon le fichier de configuration
- Gestion du cycle de production selon les phases (vues plus haut)
- Framework pour le déploiement de plugins

Architecture de Maven : Plugins

- Opérations de traitement pour la production du projet. Par ex., pour créer un jar, le plugin maven-jar-plugin sera utilisé.
- Chaque plugin fournit un ou plusieurs cibles (**goals**)
- Une cible (goal) effectue une opération particulière sur le projet. Par ex., compiler, créer un jar, déployer le logiciel dans un conteneur (e.g. Tomcat)
- Les cibles sont rattachés aux phases du cycle de production
  - On parle de **binding**

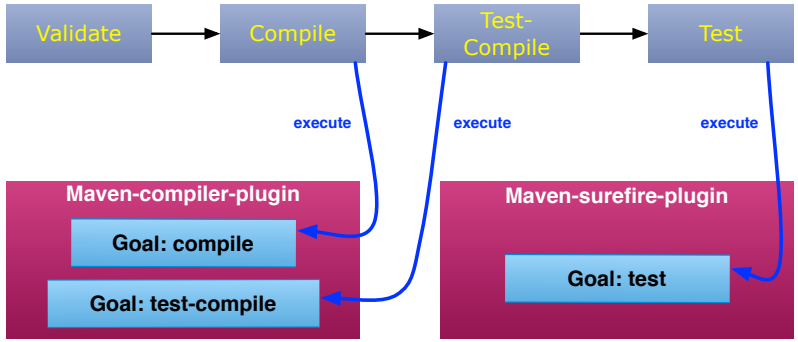
Rattachement des cibles aux phases

- Les cibles des plugins sont rattachés par défaut à des phases



Rattachement des cibles aux phases

- Ex. : `$ mvn test`
- Ex. : `$ mvn clean install` (☞ best practice)



Rattachement des cibles aux phases

- Configuration dans le fichier du **Project Object Model** (pom.xml)

```
1  <plugin>
2     <groupId>com.mycompany.example</groupId>
3     <artifactId>display-maven-plugin</artifactId>
4     <version>1.0</version>
5     <executions>
6       <execution>
7         <phase>process-test-resources</phase>
8         <goals>
9           <goal>time</goal>
10        </goals>
11      </execution>
12    </executions>
13 </plugin>
```


Cycles, plugins et cibles

- Un cycle de production est composé d'une série de phases
- Une phase est composée de cibles (goals)
- Les cibles sont fournis par les plugins
- Chaque phase a des `bindings` par défaut

`Bindings` par défaut des phases

- http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html

Clean lifecycle

Phase	Goal
`clean`	clean :clean

Bindings par défaut des phases

Default lifecycle - Packaging ejb, ejb3, jar, par, rar, war

Phase	Goal
process-resources	resources :resources
compile	compiler :compile
process-test-resources	resources :testResources
test-compile	compiler :testCompile
test	surefire :test
package	ejb :ejb or ejb3 :ejb3 or jar :jar or par :par or rar :rar or war :war
install	install :install
deploy	deploy :deploy

Bindings par défaut des phases

Default lifecycle - Packaging ear

Phase	Goal
generate-resources	ear :generate-application-xml
process-resources	resources :resources
package	ear :ear
install	install :install
deploy	deploy :deploy

Bindings par défaut des phases

Default lifecycle - Packaging maven-plugin

Phase	Goal
generate-resources	plugin :descriptor
process-resources	resources :resources
compile	compiler :compile
process-test-resources	resources :testResources
test-compile	compiler :testCompile
test	surefire :test
package	jar :jar and plugin :addPluginArtifactMetadata
install	install :install
deploy	deploy :deploy

Bindings par défaut des phases

Default lifecycle - Packaging pom

Phase	Goal
package	site :attach-descriptor
install	install :install
deploy	deploy :deploy

Bindings par défaut des phases

Site lifecycle

Phase	Goal
site	site :site
site-deploy	site :deploy

Sommaire

Dépôts (Repositories) Maven

- Stockage des artefacts issus de la production de logiciels par Maven et des dépendances pour les projets Java
- 2 types de dépôt
  - local
  - distant
- Pourquoi ne pas stocker les artefacts dans un SCM (tel que CVS, SVN, Git) ?
  - Pas de duplication de jars (pour chaque projet) dans le SCM
  - Allège la taille des projets dans le SCM (checkout plus rapide)
  - Pas de versionnement des jars, qui sont des binaires : le changement de version est reflété dans le nom du jar.

Dépôts distants Maven

- Accessibles par `http, ftp, file` ou tout autre protocole supporté par le serveur du dépôt
- Fournis par une tierce partie :
  - Maven Central : http://search.maven.org/
  - Sonatype = https://oss.sonatype.org/index.html
- Fournis par votre organisation, pour distribuer vos propres artefacts et jouer le rôle de proxy des autres dépôts distants.
  - MIAGE : http://miage08.u-paris10.fr:8081/nexus/index.html
  - LIP6 : https://teamcity-systeme.lip6.fr/nexus/index.htmle
- Serveur de dépôt le plus répandu : Nexus (http://www.sonatype.org/nexus/)
- Autre : Apache Archiva (http://archiva.apache.org/index.cgi)

Dépôt distant par défaut de Maven

- Maven utilise un dépôt par défaut : http://repo.maven.apache.org/maven2/
- On peut configurer d'autres dépôts (pas de limitation)

```
1  <repositories>
2   <repository>
3    <id>kepler</id>
4    <url>http://download.eclipse.org/releases/kepler</url>
5    <layout>p2</layout>
6   </repository>
7   <repository>
8    <id>jboss</id>
9    <url>https://repository.jboss.org/nexus</url>
10   <layout>p2</layout>
11  </repository>
12 </repositories>
```


Dépôts distants Maven pour plugins

- Les dépôts distants peuvent également héberger les artefacts des plugins Maven
- Il est possible de configurer les dépôts de plugins :

```
1  <pluginRepositories>
2   <pluginRepository>
3    <id>tycho</id>
4    <url>https://oss.sonatype.org/content/groups/public/</url>
5    <releases>
6     <enabled>false</enabled>
7    </releases>
8    <snapshots>
9     <enabled>true</enabled>
10   </snapshots>
11  </pluginRepository>
12 </pluginRepositories>
```


Dépôt local Maven

- Cache des dépendances et artefacts de compilation utilisés ou produits par votre projet, stockés sur la machine où Maven est installé
- Emplacement par défaut : `${user.home}/.m2/repository`
  - Peut être modifié dans le fichier : `${user.home}/.m2/settings.xml`

```
1  <localRepository>chemin/vers/depot/local</localRepository>
```


Sommaire

Artefacts (`Artifacts`) Maven

- Un artefact est un fichier résultant de la compilation et empaquetage d'un projet
- Peut être au format jar, war, ear, xml
- Ex. : exécutable (jar), web application (war), javadoc (jar)
- Les artefacts sont déployés dans des dépôts Maven, afin de pouvoir être utilisés comme dépendances dans d'autres projets

Identification d'artefact Maven

- Un artefact est identifié par 3 coordonnées (aka `artifact coordinates`) :
  - **Group Id** : identifiant unique pour un groupe d'artefacts provenant d'un même projet ou organisation, habituellement désignés comme les paquetages java (ex. : fr.p10.miage.si)
  - **Artifact Id** : identifiant unique de l'artefact même, dans le contexte d'un Group Id, désignant généralement un projet particulier (ex. : edt)
  - **Version** : numéro de version de l'artefact

Sommaire

Project Object Model (POM)

- Fichier de configuration en XML, situé à la racine de tout projet Maven
  - **Information** concernant le projet
  - Configuration pour **compiler** le projet
  - **Valeurs par défaut** pour certains éléments de configuration : `source dir, target dir`
  - Dépendances du projet
  - Configuration de plugins et leurs cibles
  - Dépôts utilisés
  - etc.

Configuration minimale du POM

- La configuration minimale d'un POM contient :
  - **project root**
  - **modelVersion** (4.0.0)
  - **groupId** : id du groupe du projet
  - **artefactId** : id de l'artefact (projet)
  - **version** : version de l'artefact

```
1  <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">
2    <modelVersion>4.0.0</modelVersion>
3    <groupId>fr.lip6.move</groupId>
4    <artifactId>ecore2rncparent</artifactId>
5    <version>0.0.7-SNAPSHOT</version>
6  </project>
```

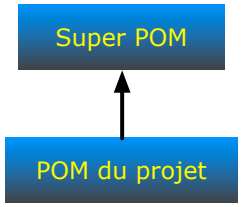

Configuration minimale du POM : packaging

- Il est également intéressant de déclarer le type de l'artefact du projet par le tag **packaging** : pom, jar, war, eclipse-plugin, eclipse-feature, eclipse-repository (nécessite le plugin `org.eclipse.tycho`)

```
1  <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">
2    <modelVersion>4.0.0</modelVersion>
3    <groupId>fr.lip6.move</groupId>
4    <artifactId>ecore2rncparent</artifactId>
5    <version>0.0.7-SNAPSHOT</version>
6    <packaging>pom</packaging>
7  </project>
```

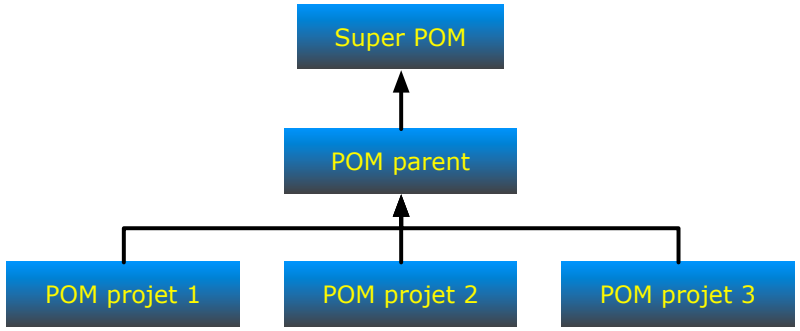

Super POM

- Tout POM possède un super POM
- Toute configuration requise pour compiler un projet, et qui n'est pas spécifiée dans le POM, est héritée du super POM



Héritage de POM

- Tout POM hérite par défaut du super POM
- Les POM peuvent hériter d'autres POM comme dans le cas de l'héritage entre classes en Java



Exemple d'héritage de POM

- Dans l'exemple ci-dessous, `groupId` est hérité
- `artefactId` est le minimum requis

```
1  <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/maven-v4_0_0.xsd">
2    <modelVersion>4.0.0</modelVersion>
3    <parent>
4      <relativePath>../ecore2rncparent/pom.xml</relativePath>
5      <groupId>fr.lip6.move</groupId>
6      <artifactId>ecore2rncparent</artifactId>
7      <version>0.0.7-SNAPSHOT</version>
8    </parent>
9    <artifactId>fr.lip6.move.ecore2rnc.plugin</artifactId>
10 </project>
```


Redéfinition de valeurs héritées

- Tout POM peut redéfinir des valeurs de configurations héritées de POMs de son ascendance
- Le **POM effectif** est la somme de toutes les configurations dans la hiérarchie de POMs (il est calculé)
- Le POM effectif peut être visualisé à l'aide de la commande `$ mvn help:effective-pom`
- Ex. : Je vous invite à regarder le POM effectif d'un module d'un de mes projets : `../images/effectivePOMEcoreToRncPlugin.xml`

POM multi module

- Un POM peut agréger plusieurs modules
- Un POM peut agréger plusieurs modules et être leur POM parent en même temps
- C'est un type de configuration très répandu (e.g. architectures n-tiers)
- Ex. : Je vous invite à regarder le POM effectif d'un module d'un de mes projets : `../images/effectivePOMEcoreToRncPlugin.xml`

Déclaration des modules de Maven 3.1.0

```
1  <modules>
2      <module>maven-plugin-api</module>
3      <module>maven-model</module>
4      <module>maven-model-builder</module>
5      <module>maven-core</module>
6      <module>maven-settings</module>
7      <module>maven-settings-builder</module>
8      <module>maven-artifact</module>
9      <module>maven-aether-provider</module>
10     <module>maven-repository-metadata</module>
11     <module>maven-embedder</module>
12     <module>maven-compat</module>
13     <module>apache-maven</module>
14   </modules>
```

Définition minimale du module apache-maven

```
1  <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">
2    <modelVersion>4.0.0</modelVersion>
3    <parent>
4      <groupId>org.apache.maven</groupId>
5      <artifactId>maven</artifactId>
6      <version>3.1.0</version>
7    </parent>
8
9    <artifactId>apache-maven</artifactId>
10 </project>
```


Maven 3.1.0 a lui-même un POM parent

```
1  <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">
2    <modelVersion>4.0.0</modelVersion>
3
4    <parent>
5      <groupId>org.apache.maven</groupId>
6      <artifactId>maven-parent</artifactId>
7      <version>23</version>
8      <relativePath>../pom/maven/pom.xml</relativePath>
9    </parent>
10
11   <artifactId>maven</artifactId>
12   <version>3.1.0</version>
13 </project>
```


Sommaire

Gestion des dépendances

- C'est LE point fort de Maven
- Gérer manuellement des centaines de dépendances est une tâche fastidieuse
  ☞ Maven facilite grandement cette tâche
- Les dépendances sont stockées dans votre dépôt Maven local
- Les dépendances transitives sont automatiquement gérées
- Les portées (scope) d'inclusion des dépendances sont également gérées

Déclaration des dépendances

- Il faut fournir les 3 coordonnées de l'artefact de la dépendance
  - GroupId
  - ArtefactId
  - Version
- Cette déclaration s'effectue dans le fichier de configuration **pom.xml**

```
1  <dependencies>
2    <dependency>
3      <groupId>org.testng</groupId>
4      <artifactId>testng</artifactId>
5      <version>6.8</version>
6      <scope>test</scope>
7    </dependency>
8  </dependencies>
```


Mécanisme de gestion des dépendances

- Centraliser la gestion des dépendances
- Toute la déclaration principale et la configuration des dépendances est placée dans le POM parent (version, exclusions, scope, type)
- Les POM enfants déclarent simplement une référence vers la dépendance déjà déclarée et sa configuration
- Cela allège grandement la configuration des POM enfants et évite la duplication de déclaration et configuration

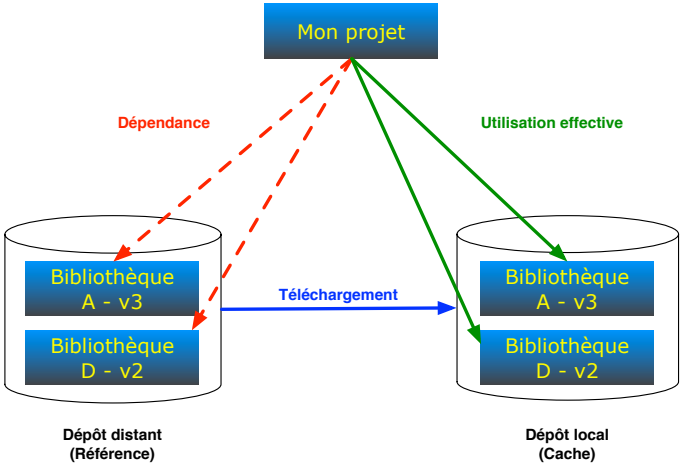
Centralisation de la déclaration de dépendances

```
1  <dependencyManagement>
2   <dependencies>
3    <dependency>
4     <groupId>javax</groupId>
5     <artifactId>javaee-web-api</artifactId>
6     <version>6.0</version>
7     <scope>provided</scope>
8    </dependency>
9    <dependency>
10    <groupId>com.oracle</groupId>
11    <artifactId>ojdbc6</artifactId>
12    <version>11.2.0.2.0</version>
13    <type>jar</type>
14   </dependency>
15  </dependencies>
16 </dependencyManagement>
```


Simplification de la déclaration de dépendances dans POM enfants

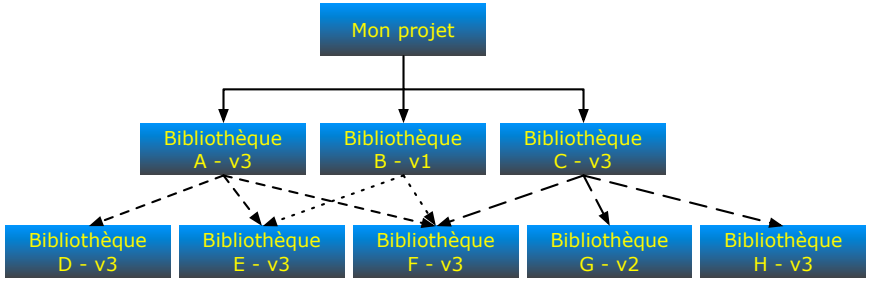
```
1  <parent>
2     <groupId>parent-groupId</groupId>
3     <artifactId>parent-artifactId</artifactId>
4     <version>1.0</version>
5  </parent>
6  <dependencies>
7    <dependency>
8     <groupId>javax</groupId>
9     <artifactId>javaee-web-api</artifactId>
10   </dependency>
11   <dependency>
12    <groupId>com.oracle</groupId>
13    <artifactId>ojdbc6</artifactId>
14    <type>jar</type>
15   </dependency>
16 </dependencies>
```

Gestion des dépendances par les caches

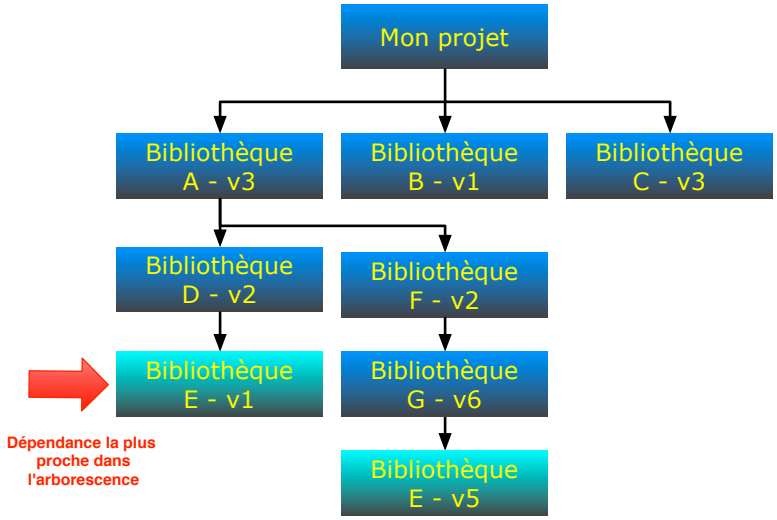


Gestion des dépendances transitives

- Maven gère également les dépendances de vos dépendances (et ainsi de suite)



Résolution des versions des dépendances



Portées (Scope) des dépendances

- Inclusion de dépendance seulement à l'étape voulue lors de la production
- Toutes les dépendances ne sont pas utiles à toutes les étapes
- Ex. 1 : Il n'est pas nécessaire de distribuer JUnit si on ne distribue pas les tests du logiciel
- Ex. 2 : les dépendances déjà fournies par les serveurs d'application ne doivent pas être incluses dans votre war ou ear

Portées des dépendances

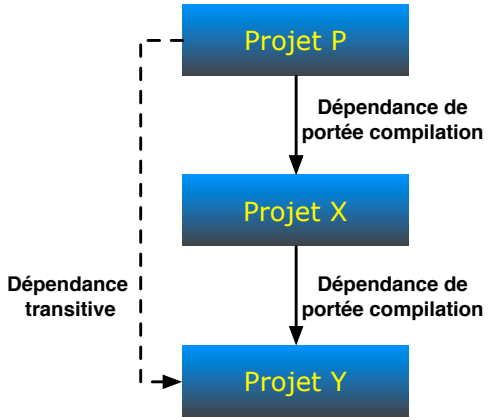
6 portées disponibles :

- `Compile`
- `Provided`
- `Runtime`
- `Test`
- `System`
- `Import`

Dépendance de portée Compilation

- C'est la portée par défaut
- La dépendance est exposée à tous les classpaths du cycle de production (phases compile, test-compile, test, package)
- La dépendance est incluse dans l'artefact final
- Elle est fournie transitivement à tout projet qui dépend du projet qui la déclare
  - Ex. : Projet X dépend de projet Y (portée compilation). Tout projet P qui dépend de X aura également Y comme dépendance, transitivement.

Dépendance de portée Compilation



Dépendance de portée Provided

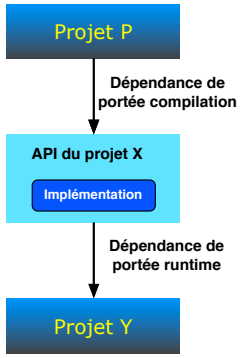
- Exposée aux classpaths des phases `compile` et `test`
- La dépendance n'est pas incluse dans l'artefact final
- Le conteneur où l'artefact sera utilisé est supposé fournir cette dépendance
  - Ex. : **servlet-api** est utilisée pour la compilation et les tests, mais ne sera pas distribuée avec l'artefact final, car le conteneur de Servlet est supposé le fournir.

Dépendance de portée Runtime

- Requise pendant l'exécution de l'application
- N'est donc pas requise pour compiler le projet
- Elle est cependant requise pendant les tests, puisque ceux-ci exécuteront le programme.

Dépendance de portée Runtime

- L'API du `Projet X` ne requiert pas le `Projet Y`
- Cependant, son implémentation dépend du `Projet Y`
- `Projet Y` est donc requis pendant l'exécution du `Projet P` mais pas pour la compilation de ce dernier



Dépendance de portée Test

- Requise pendant la compilation et l'exécution des tests
- Ne sera pas assemblé avec l'artefact final (jar, war, ear, etc.)
- N'est requise ni pour la compilation, ni pendant l'exécution de l'application

Dépendance de portée System

- Même signification que la dépendance de portée Provided
- N'est pas résolue en la cherchant dans le dépôt local
- Elle est supposée exister dans la machine de développement
- Il faut donc **fournir son chemin système** en utilisant la balise `<systemPath>`
- Utilisée dans de rares cas où la production du logiciel dépend vraiment de la machine où le processus de fabrication a lieu
- Pratique non recommandée

Exemple de dépendance de portée System

```
1  <dependencies>
2     <dependency>
3       <groupId>javax.sql</groupId>
4       <artifactId>jdbc-stdext</artifactId>
5       <version>2.0</version>
6       <scope>system</scope>
7       <systemPath>${java.home}/lib/rt.jar</systemPath>
8     </dependency>
9   </dependencies>
```


Dépendance de portée Import

- Importation de dépendances d'autres POMs
- Utilisée dans la section <dependencyManagement>
- Utile pour centraliser toute la gestion des dépendances dans un seul projet

Dépendance de portée Import : Projet X

```
1   <groupId>maven</groupId>
2   <artifactId>X</artifactId>
3   <packaging>pom</packaging>
4   <version>1.0</version>
5   <dependencyManagement>
6     <dependencies>
7       <dependency>
8         <groupId>test</groupId>
9         <artifactId>a</artifactId>
10        <version>1.1</version>
11      </dependency>
12      <dependency>
13        <groupId>test</groupId>
14        <artifactId>b</artifactId>
15        <version>1.0</version>
16        <scope>compile</scope>
17      </dependency>
18    </dependencies>
19  </dependencyManagement>
```


Dépendance de portée Import : Projet Y

```
1   <groupId>maven</groupId>
2   <artifactId>Y</artifactId>
3   <packaging>pom</packaging>
4   <version>1.0</version>
5   <dependencyManagement>
6     <dependencies>
7       <dependency>
8         <groupId>test</groupId>
9         <artifactId>a</artifactId>
10        <version>1.2</version>
11      </dependency>
12      <dependency>
13        <groupId>test</groupId>
14        <artifactId>c</artifactId>
15        <version>1.0</version>
16        <scope>compile</scope>
17      </dependency>
18    </dependencies>
19  </dependencyManagement>
```

Dépendance de portée Import : Projet Z importe X et Y

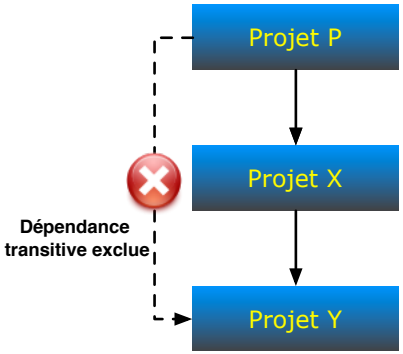
```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>maven</groupId>
      <artifactId>X</artifactId>
      <version>1.0</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
    <dependency>
      <groupId>maven</groupId>
      <artifactId>Y</artifactId>
      <version>1.0</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```


Exclusion de dépendances transitives

- Il est possible d'exclure les dépendances transitives de l'arbre des dépendances

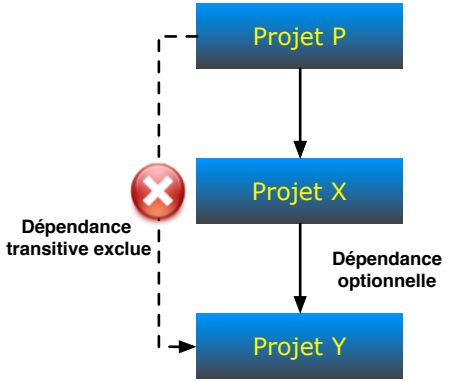
```
<dependencies>
  <dependency>
   <groupId>groupX</groupId>
   <artifactId>artifactX</artifactId>
   <exclusions>
    <exclusion>
     <groupId>groupY</groupId>
     <artifactId>artifactY</artifactId>
    </exclusion>
   </exclusions>
  </dependency>
</dependencies>
```


Exclusion de dépendances transitives



Exclusion de dépendances optionnelles

- Les dépendances optionnelles sont exclues par défaut



Sommaire

Archétypes Maven

- Un archétype est un modèle (**template**) de projet
- Factorise une configuration commune pour des projets de même profil
- Il est possible de créer un archétype à partir d'un projet
- Les archétypes sont déployés dans les dépôts Maven

Utilisation d'archétype

- Instancier un archétype :
  - `$ mvn archetype:generate`
- Créer un archétype du projet courant :
  - `$ mvn archetype:create-from-project`

Sommaire

Maven dans les IDE

- Un plugin Eclipse : **m2eclipse**
  - http://eclipse.org/m2e/download/
- Intégration dans NetBeans
  - https://platform.netbeans.org/tutorials/nbm-maven-quickstart.html
- Intégration dans ItelliJIDEA
  - http://www.jetbrains.com/idea/features/ant_maven.html

Références

- http://maven.apache.org
- http://maven.apache.org/pom.html
- http://books.sonatype.com/mvnref-book/reference/