Diginamic Formation

« Dynamisez votre carrière digitale ! »

Formation : Apache Maven 3

Votre formateur

- Ronan Mounès, société CodeVallée (http://codevallee.fr).
- Mes technos : Java / Docker / Android / Xamarin / Web / Linux
- Quelques missions : BNP Paribas, Crédit du Nord, Société Générale, JC Decaux, Ministère des Affaires sociales et de la Santé, Editions Francis Lefebvre, etc.
- J'ai été entre autres : développeur, chef de projets, architecte, formateur, intégrateur, etc.

Et vous ?

Plan

- ✓Construire un projet Java
- ✓Présentation
- ✓Le POM
- ✓Gestion des dépendances
- ✓Le cycle de vie d'un projet
- ✓Les référentiels
- ✓Les plugins
- ✓Les profils
- ✓Les projets multi-modules

Construire un projet Java sans MAVEN

TP 0

✓Récupérer le fichier TP00 - Projet sans MAVEN.pdf depuis le drive.

Bilan ?

- Des jar téléchargés et copiés sur le disque dur par le developpeur.
- Comment reproduire ces manipulations sur un autre poste, dans d’autres environnements ?
- Comment effectuer la maintenant de l’application ?

Présentation

Définition

- Logiciel de gestion de projet et l'automatisation de production des projets logiciels.
- Elément central dans la gestion d'une infrastructure de projet informatique.
- Maven est capable de gérer toute la vie du projet :
  - gestion des bibliothèques logicielles de dépendances,
  - génération de la documentation et rapport du projet,
  - construction du livrable final,
  - déploiement des versions de livraison sur les plateformes cibles (avec Jenkins par exemple)

Historique

- 2001
  - Jason van Zyl, son créateur, propose le premier prototype au sein du projet *Apache Jakarta Alexandria*.
  - Usage réel avec le projet *Jakarta Turbine*
- 2003
  - *Maven* devient *Apache Maven*
- 2004
  - Sortie de *Apache Maven 1*
- 2005
  - Sortie de *Apache Maven 2.0*
- 2010
  - *Sortie de Apache Maven 3.0*
- 2017
  - *Sortie de Apache Maven 3.5*

diginamic FORMATION
« Votre passeport pour l'emploi numérique »

Convention plutôt que configuration !

- ✓Basé sur le paradigme "Convention over Configuration« (CoC)
- ✓Maven intègre des comportements par défaut et normalisés

```
<project xmlns="http://maven.apache.org/POM/4.0.0">
  <modelVersion>4.0.0</modelVersion>
  <groupId>fr.codevallee.formation</groupId>
  <artifactId>maven-exemple</artifactId>
  <version>1.0.0-SNAPSHOT</version>
</project>
```

Ça c'était avant… Ant

✓Ant est un outil puissant pour construire des projets

```
<project name="my-project" default="dist" basedir=".">
    <description>
    simple example build file
    </description>
    <!-- set global properties for this build -->
    <property name="src" location="src/main/java"/>
    <property name="build" location="target/classes"/>
    <property name="dist" location="target"/>
    <target name="init">
        <!-- Create the time stamp -->
        <tstamp/>
        <!-- Create the build directory structure used by compile -->
        <mkdir dir="${build}"/>
    </target>
    <target name="compile" depends="init"
    description="compile the source " >
        <!-- Compile the java code from ${src} into ${build} -->
        <javac srcdir="${src}" destdir="${build}"/>
    </target>
    <target name="dist" depends="compile"
    description="generate the distribution" >
        <!-- Create the distribution directory -->
        <mkdir dir="${dist}/lib"/> <!-- Put everything in ${build} into the MyProject-
${DSTAMP}.jar file -->
        <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
    </target>
    <target name="clean"
    description="clean up" >
        <!-- Delete the ${build} and ${dist} directory trees -->
        <delete dir="${build}"/>
        <delete dir="${dist}"/>
    </target>
```

Pour résumer …

Buts de Maven

- Simplifier le build
- Système de build uniforme
- Information de qualité sur le projet
- Best practices
- Intégrations de nouvelles fonctionnalités

Le POM

Le POM (*Project Object Model*)

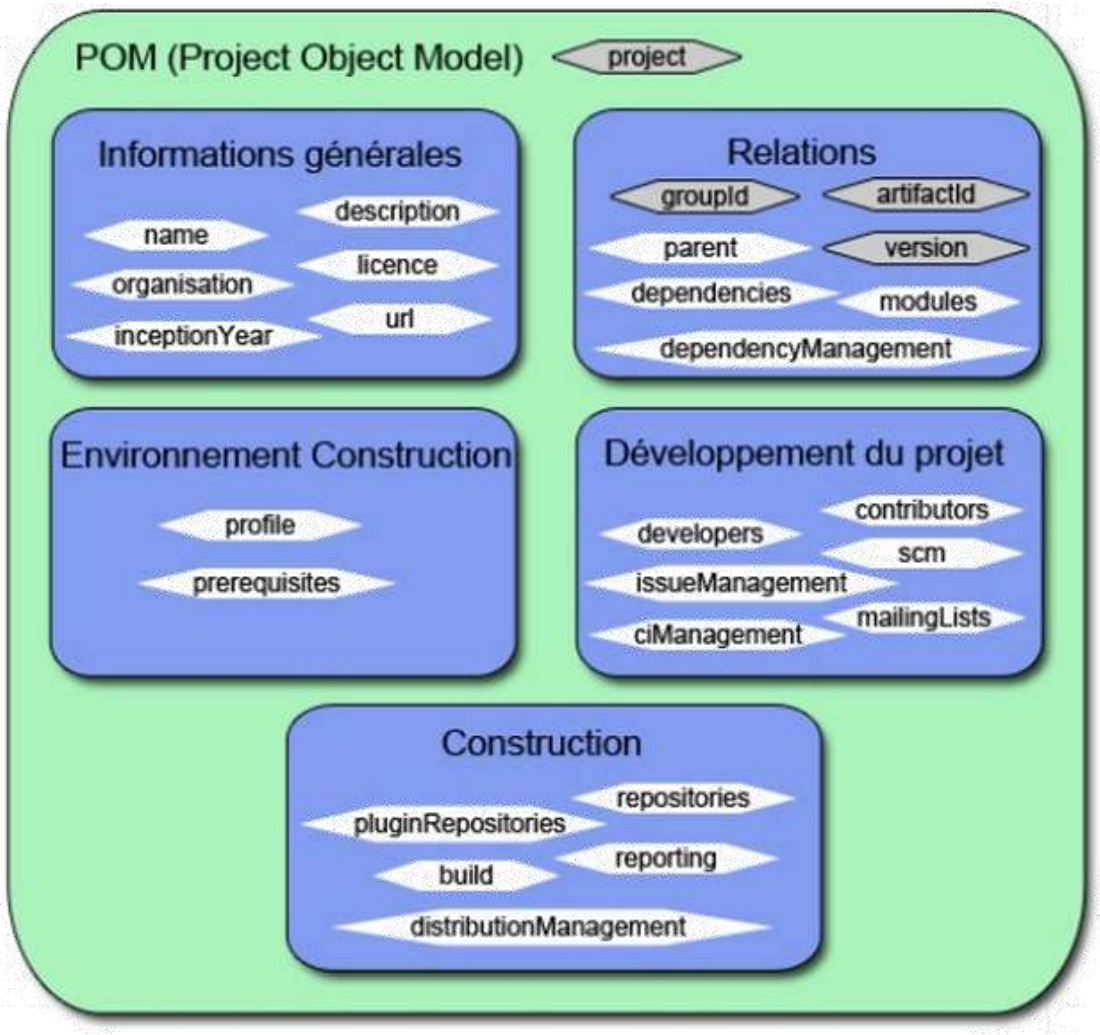
- ✓Le POM est le descripteur d'un projet Maven.
- ✓Fichier au format XML.
- ✓Ce fichier, nommé pom.xml, définit un projet Maven.
- ✓Il propose un modèle de données pour identifier les éléments indispensables du projet.

Format du fichier POM

- Le POM se compose de quatre catégories de description et de configuration :
  - Informations générales sur le projet,
  - Relations entre POM.
  - Configuration du build (construction),
  - Environnement du build (profil),
  - Informations sur l'équipe du projet

Source : http://www.sonatype.com

Format du fichier POM



Exemple de POM minimal

✓Les éléments obligatoires dans un fichier POM sont les suivants :

- `modelVersion :` version de l'objet modèle utilisée dans ce POM.
- `groupId :` indique l'identifiant unique de l'organisation ou groupe qui a créé le projet.
- `artifactId :` indique le nom de base unique de l'artefact principal généré dans ce projet.
- `version :` version de l'artefact générée par le projet.

Les coordonnées Maven

✓Les coordonnées Maven définissent un ensemble d'identifiants qui permet de définir de manière unique un projet, une dépendance ou un plugin.

✓Chaque projet Maven est identifié par ses coordonnées :

- Group (G)
- Artifact (A)
- Packaging (P)
- Version (V)

```
<groupId>:<artifactId>:<packaging>:<version>
```

La notion d'artéfact

- ✓Maven définit comme artéfact le fichier de sortie que le projet va générer au final.
- ✓Chaque projet ne doit fournir qu'un seul artéfact dont le format est déterminé par la valeur de l'élément `<packaging />` du POM.
- ✓Les recommandations de Maven précisent que la forme de l'artéfact de sortie est :

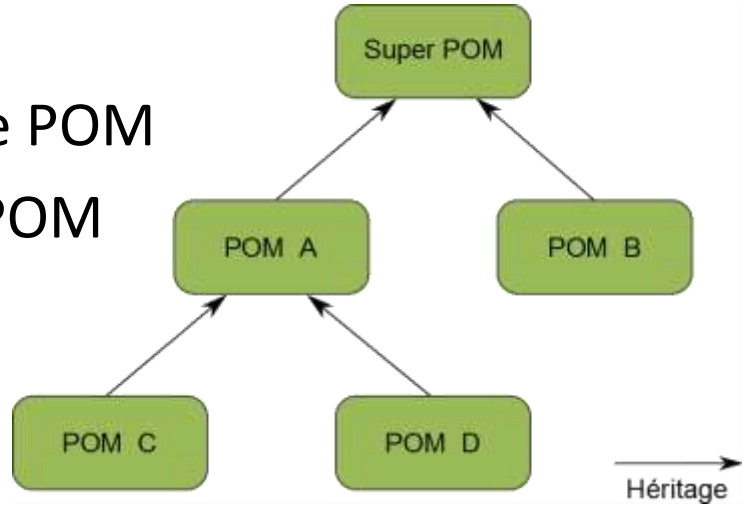
```
<artifactId>-<version>.<packaging>
```

Le Super POM

- A l'instar de toutes les classes Java qui héritent implicitement de la classe `java.lang.Object`, tous les POM héritent d'un POM commun : le **Super POM.**

- Ce POM définit les valeurs par défaut.

- Toutes les valeurs présentes dans ce POM peuvent être surchargées dans les POM qui en héritent.



Le Super POM

- On y trouve :
  - Pointeur vers le repository central de MAVEN, cette valeur peut être surchargé en modifiant le fichier settings.xml
  - Configuration par défaut des builds selon les standards MAVEN

Le POM effectif

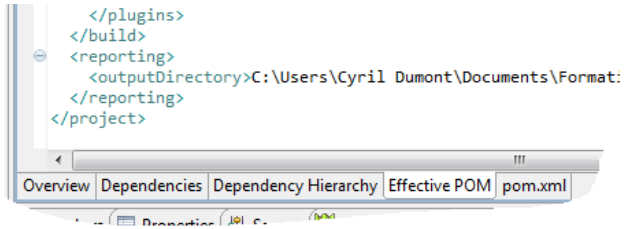
- Un POM Maven est la combinaison du Super POM, de tous les POMs parents intermédiaires et enfin du POM du projet en cours.

- Pour obtenir le POM effectif d'un projet, Maven commence par surcharger la configuration du Super POM avec un ou plusieurs POMs parents. Puis, il surcharge la configuration résultante avec les valeurs du POM du projet en cours.

- Il est possible d'obtenir le POM effectif d'un projet en exécutant la commande suivante :

```
$ mvn help:effective-pom
```

- Eclipse propose un onglet pour voir le POM effectif d'un projet.



Structure d’un projet

Structure d'un projet

✓Maven impose des standards *modifiables* :
- Structure et organisation des fichiers,
- Convention.

✓Il est conseillé de respecter les standards de Maven autant que possible :
- le fichier POM est plus court et plus simple grâce aux valeurs par défaut,
- le projet est plus simple à comprendre, à maintenir lors de changement dans l'équipe,
- l'intégration de plug-ins est plus simple.

diginamic FORMATION
« Votre passeport pour l'emploi numérique »

Structure d'un projet (2)

✓La structure d'un projet Maven de base est définit comme suit (la variable `${project.basedir}` est utilisée pour définir la racine du projet):

Répertoire	Contenu
`${project.basedir}`**`/src/main/java`**	Fichiers du code source Java
`${project.basedir}`**`/src/main/resources`**	Ressources du projet (.properties, .xml …)
`${project.basedir}`**`/src/test/java`**	Fichiers du code source Java des tests
`${project.basedir}`**`/src/test/resources`**	Fichiers de configuration des tests
`${project.basedir}`**`/target`**	artéfact du projet
`${project.basedir}`**`/target/classes`**	Classes compilées et ressources filtrées
`${project.basedir}`**`/target/test-classes`**	Classes de test compilées et ressources filtrée
`${project.basedir}`**`/target/site`**	Fichiers générés du site Web associé au projet

diginamic FORMATION
« Votre passeport pour l'emploi numérique »

TP 1 : découverte d'un projet MAVEN

✓Récupérer le fichier maven_tp1.txt sur le drive.

Gestion des dépendances

La gestion des dépendances

- La gestion des dépendances est un élément essentiel dans la configuration d'un projet Maven.

- Les librairies externes utilisées par le projet ne doivent être que des liens vers d'autres artefacts Maven et surtout pas copiées dans les répertoires du projet.

- Il est possible d'ajouter des dépendances dans le POM du projet. Une dépendance est identifiée comme tout artéfact par un `groupId`, un `artifactId` et un numéro de `version` (coordonnée) .

```
<dependencies>
   <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-entitymanager</artifactId>
      <version>4.1.8.Final</version>
      <scope>compile</scope>
   </dependency>
</dependencies>
```

Les champs d'application *(scope)*

- ✓Pour chaque dépendance, il faut définir son champs d'application (*scope*). Il précise la façon dont la dépendance est utilisée lors des différentes phases du projet.
- ✓Cette information a des conséquences sur la présence ou non de la librairie lors de la compilation des classes du projet ou lors de la création de l'artéfact.
- ✓Le champs d'application par défaut est compile.

diginamic FORMATION
« Votre passeport pour l'emploi numérique »

Les différents champs d'application

Scope	Description
compile	La librairie est nécessaire dans le processus de compilation. La libraire est incluse dans le CLASSPATH lors de la compilation et lors de l'exécution.
provided	Similaire à **compile** à l'exception que la librairie est disponible dans le conteneur de déploiement de l'application. La librairie n'est pas incluse dans le CLASSPATH lors de l'exécution.
runtime	La librairie n'est pas nécessaire lors de la compilation des sources mais uniquement lors de l'exécution du projet.
test	La librairie est incluse dans le CLASSPATH pour la compilation et l'exécution des tests.
system	Permet au projet de dépendre d'une librairie qui ne se trouve pas dans un référentiel mais qui est disponible sur le serveur dans lequel se trouve le projet. Il fonctionne exactement comme **provided**.
import	Utilisé uniquement pour définir une dépendance vers un POM de type pom. Il permet d'inclure dans un POM la gestion des dépendances d'un autre POM.
optional	la librairie est ajouté dans le MANIFEST mais pas dans le répertoire lib du WAR.

Les dépendances transitives

- ✓Maven permet de résoudre les dépendances transitives
- ✓Si un artefact `A` dépend d'un artefact `B` qui dépend d'un artefact `C`, la résolution des dépendances de `A` trouvera `B` et `C`
  - Déclaration de la dépendance de l'artefact `A` dans le pom.xml
  - Maven se charge des artefacts `B` et `C`
- ✓La commande suivante affiche les dépendances transitives :

```
$ mvn dependency:tree
```

```
[INFO] --- maven-dependency-plugin:2.1:tree (default-cli) @ mabiblio-persistence ---
[INFO] org.formation.mabiblio:mabiblio-persistence:jar:1.0.1-SNAPSHOT
[INFO] \- org.hibernate:hibernate-entitymanager:jar:4.1.8.Final:compile
[INFO]    +- org.jboss.logging:jboss-logging:jar:3.1.0.GA:compile
[INFO]    +- org.jboss.spec.javax.transaction:jboss-transaction-api_1.1_spec:jar:1.0.0.Final:compile
[INFO]    +- dom4j:dom4j:jar:1.6.1:compile
[INFO]    +- org.hibernate:hibernate-core:jar:4.1.8.Final:compile
[INFO]    |  \- antlr:antlr:jar:2.7.7:compile
[INFO]    +- org.hibernate.javax.persistence:hibernate-jpa-2.0-api:jar:1.0.1.Final:compile
[INFO]    +- org.javassist:javassist:jar:3.15.0-GA:compile
[INFO]    \- org.hibernate.common:hibernate-commons-annotations:jar:4.0.1.Final:compile
```

Gérer les conflits de dépendances

- Lors de la résolution de la transitivité sur un projet, il est possible que le graphe donne lieu à des conflits entre plusieurs dépendances.
- Un conflit intervient lorsque au moins deux dépendances avec les mêmes coordonnées sont disponibles dans l'arborescence dans plusieurs versions différentes.
- Un conflit entre deux dépendances directes se règle en supprimant la dépendance qui pose problème.
- Dans le cas d'un conflit entre deux dépendances directes issues de POM différents avec l'héritage, c'est la version de la dépendance déclarée dans le POM enfant qui a la priorité sur la version déclarée dans le POM parent.

Gérer les conflits de dépendances (2)

- La grande majorité des conflits apparaît avec les dépendances transitives.
- Maven définit des règles afin de gérer les cas de potentiels conflits.
- L'ordre de critères de priorité est le suivant :
  - La dépendance transitive la plus proche du projet en termes de profondeur d'arborescence.
  - La dépendance transitive issue de la dépendance directe déclarée en premier dans l'ordre des dépendance du POM.

diginamic FORMATION
« Votre passeport pour l'emploi numérique »

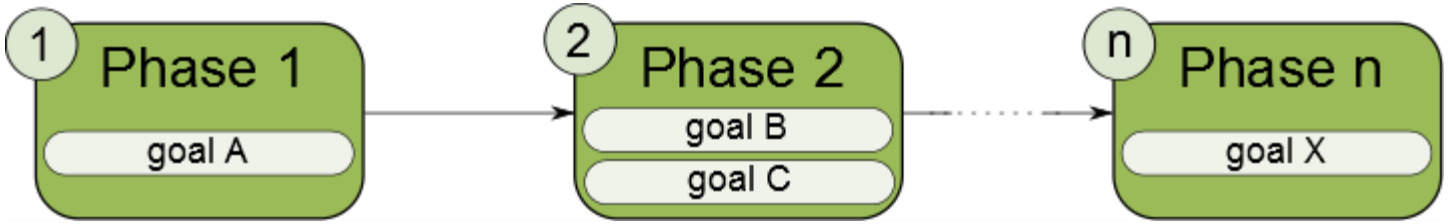
TP 2 les dépendances

✓Récupérer le fichier maven_tp2.txt sur le drive

Le cycle de vie d’un projet

Phases / Goals

- ✓Maven liste toutes les étapes nécessaires à la réussite d'un projet.
- ✓Ces étapes sont définies comme les phases du projet.
- ✓Chaque phase peut réaliser des actions (goals).
- ✓La succession des phases est définie comme le cycle de vie du projet.



Les différents cycles de vie

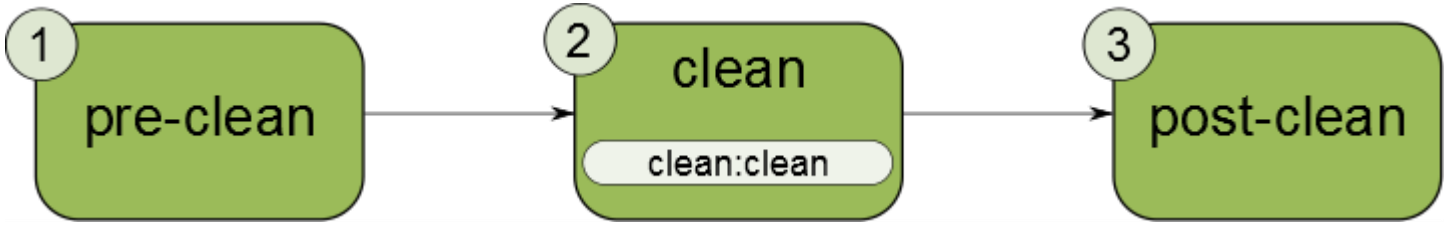
✓Maven a identifié 3 cycles de vie spécifique dans l'existence d'un projet :

- Le cycle de vie pour le nettoyage du projet (*clean life cycle*).
- Le cycle de vie par défaut (*default life cycle*).
- Le cycle de vie pour le site du projet (*site life cycle*).

clean life cycle

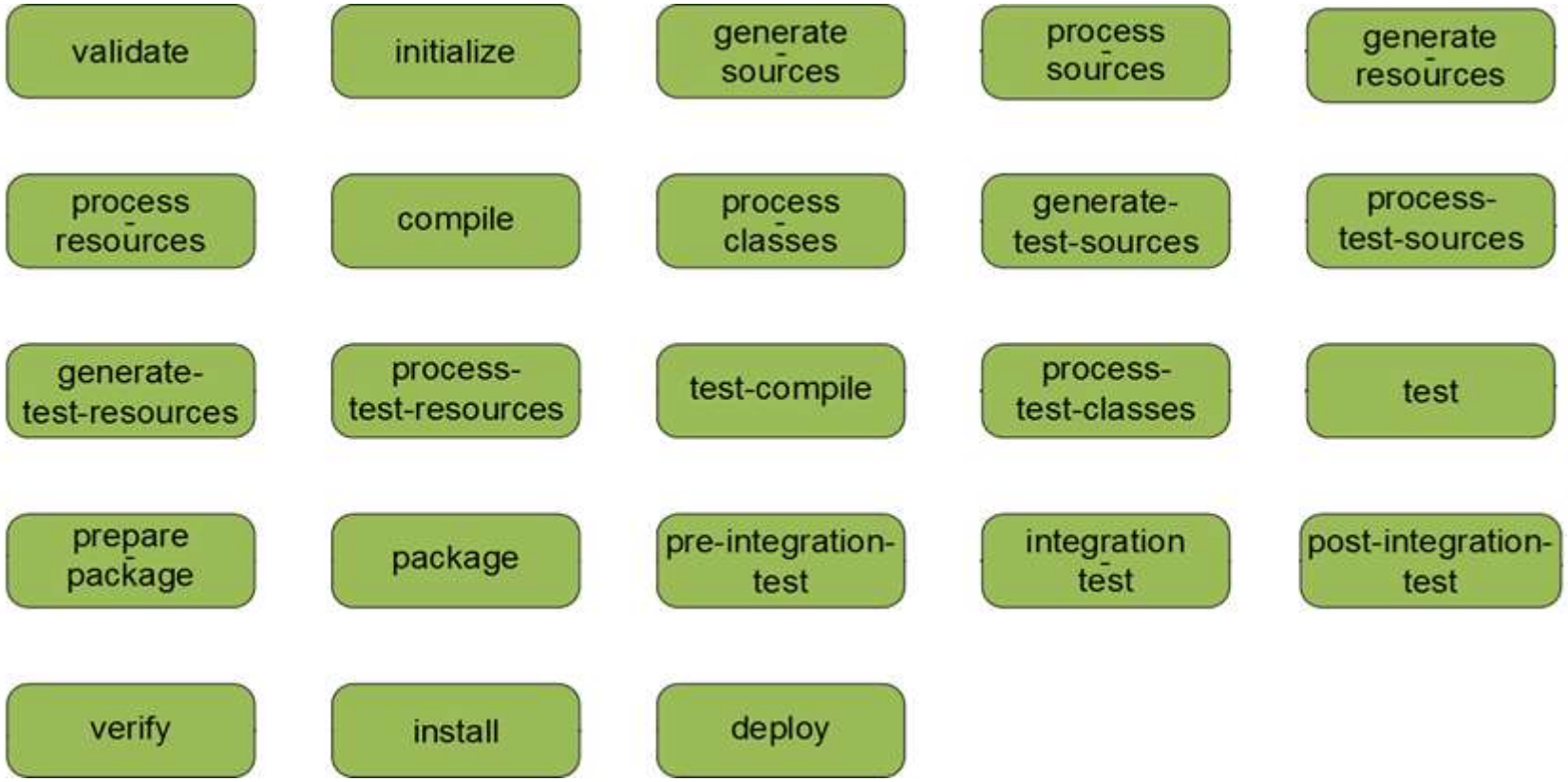
- Cycle de vie qui assure d'avoir tous les dossiers de travail vides avant la construction d'un nouvel artéfact.

- Il définit **3 phases** :
  - **pre-clean** : initialisation du processus de nettoyage.
  - **clean** : suppression des fichiers générés par un build précédent.
  - **post-clean** : finalisation du processus de nettoyage.



Le cycle de vie par défaut

✓Il est composé de 23 phases :

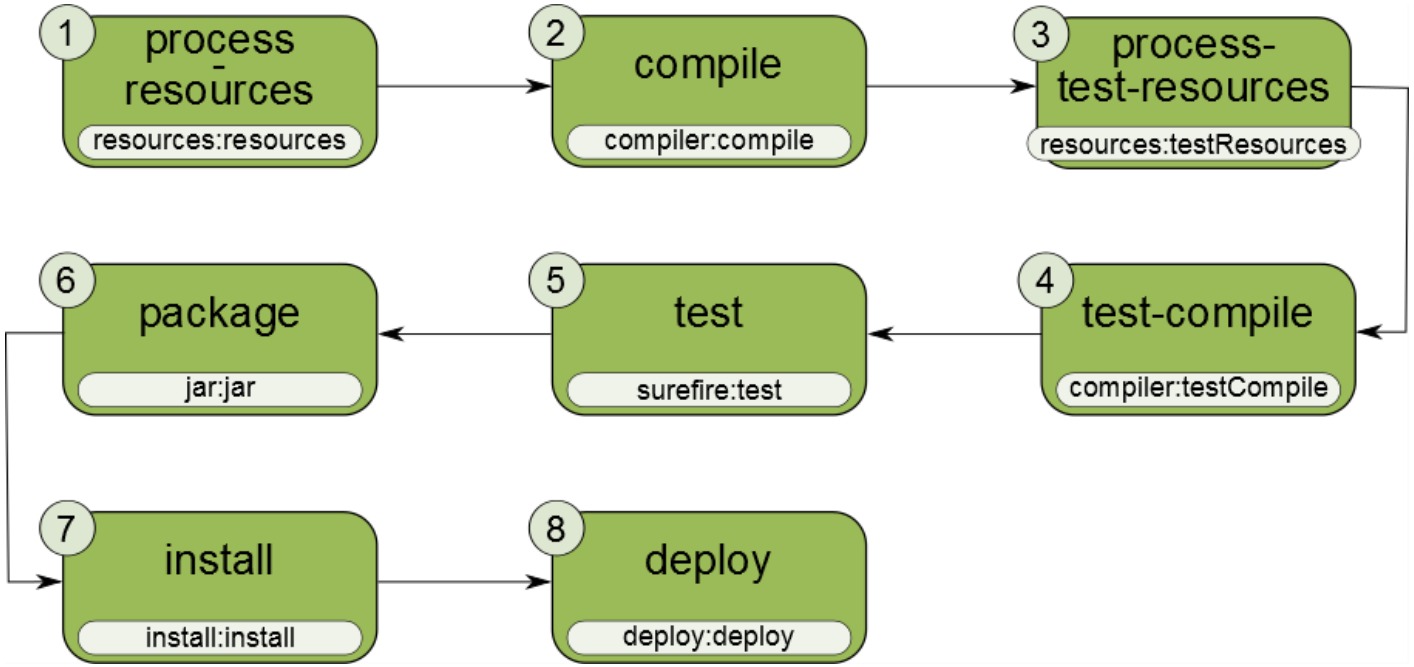


Le cycle de vie par défaut (2)

- ✓Ces phases ne réalisent pas les mêmes opérations selon la configuration du projet.
- ✓Chaque type de projet doit définir les goals à exécuter lors de chaque phase. C'est-à-dire associer ses propres traitements aux phases du cycle de vie par défaut.
- ✓Il n'est pas obligatoire d'associer des goals à toutes les phases du cycle de vie par défaut.

Création d'une archive Java

✓Le cycle de vie d'un projet pour la création d'une archive Java (JAR) redéfinit 8 phases du cycle de vie par défaut :



diginamic FORMATION
« Votre passeport pour l'emploi numérique »

Création d'une archive Java (2)

Phases	Description	Plugins
process-resources	Copie les fichiers, hors code source, dans le répertoire de construction finale en les filtrant si nécessaire.	maven-resources-plugin
compile	Compile le code source du projet dans le dossier de construction du JAR.	maven-compiler-plugin
process-test-resources	Copie les fichiers de test, hors code source, dans le répertoire de construction finale en les filtrant si nécessaire.	maven-resources-plugin
test-compile	Compile les classes de tests.	maven-compiler-plugin
test	Exécute les tests unitaires.	maven-surefire-plugin
package	Crée l'archive JAR à partir du dossier contenant les sources compilées et les fichiers de ressources filtrés.	maven-jar-plugin
install	Copie l'archive JAR dans le référentiel local.	maven-install-plugin
deploy	Déploie l'archive JAR dans le référentiel distant.	maven-deploy-plugin

Gestion des erreurs

- Les options suivantes permettent de contrôler le comportement de Maven lors de l'échec d'un build :

```
$ mvn -X -e deploy
```

  - `-X` pour activer le **mode debug,**
  - `-e` pour consulter la **trace Java complète de l'erreur.**

- Maven a mis l'accent sur l'affichage d'information lorsqu'une erreur se produit. Il est important d'étudier attentivement ces messages pour déterminer l'origine de l'erreur.

Exécuter un traitement

- Par exemple, il est possible d'exécuter un traitement de deux manières différentes :
  - Par l'appel d'un goal : `$ mvn jar:jar`

    Ici le traitement réalise simplement l'archivage dans un fichier jar du contenu du répertoire `target`. Le fichier peut alors être vide si le répertoire `target` du projet ne contient aucun fichier **.**

  - Par l'appel d'une phase : `$ mvn package`

    Ici toutes les phases précédant la phase package sont également exécutées. À partir du moment où le projet contient des sources Java, le fichier Jar ne peut être vide.

diginamic FORMATION
« Votre passeport pour l'emploi numérique »

TP 3 : compilation avec MAVEN

✓Récupérer le fichier maven_tp3.txt sur le drive

Les référentiels

Les référentiels

✓Un référentiel est un emplacement sur un système de fichiers qui contient des artefacts organisés selon une arborescence spécifique à Maven (basée sur les notions de `groupId`, `artifactId` et `version`) et indexées à l'aide de fichiers XML (`metadata.xml`).

✓Il existe deux types de référentiels :

- Le référentiel local (*local repository*),
- Les référentiels distants (*remote repository*).

Le référentiel local

- Le référentiel local est l'emplacement sur le système de fichiers local où sont stockées et indexées toutes les librairies et plugins utilisés par Maven et les projets en cours de développement sur la machine locale.

- L'emplacement par défaut du référentiel local est `${user.home}/.m2/repository`

- Il est possible de modifier cet emplacement dans le fichier `settings.xml` :

```
<localRepository>C:\Formations\Maven3\repo</localRepository>
```

Les référentiels distants (1)

✓Un référentiel distant permet d'alimenter le référentiel local.

✓Maven différencie les référentiels contenant les plugins de ceux contenant les librairies.

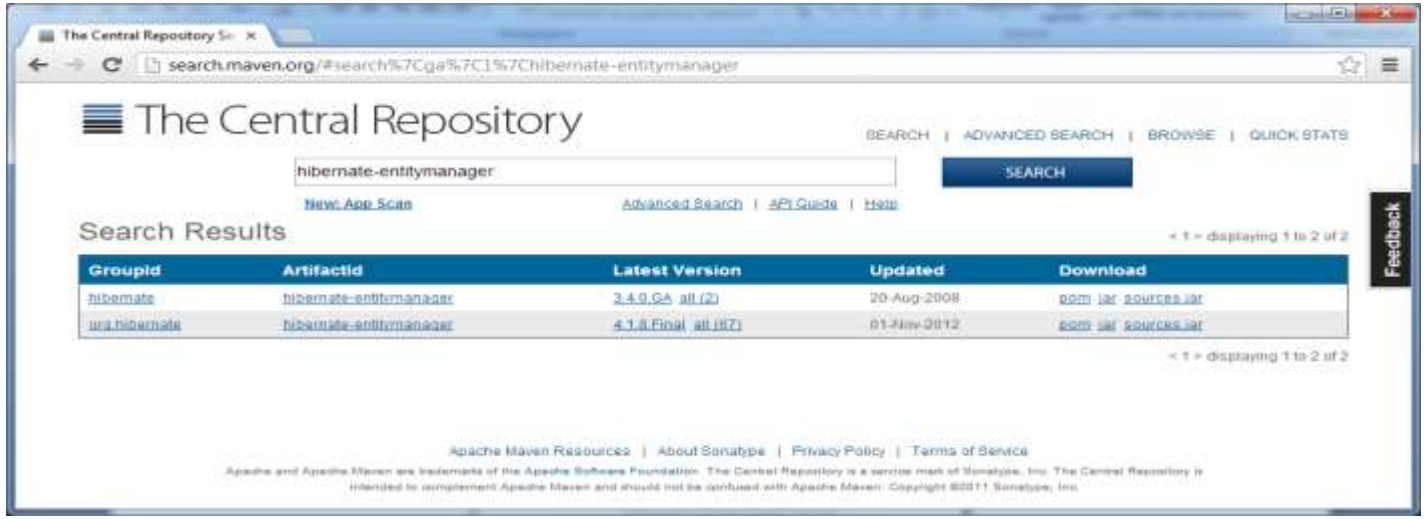
✓Le Super POM définit deux référentiels par défaut : un pour les librairies et un pour les plugins.

✓Il est possible d'ajouter d'autres référentiels distants dans un POM.

Le référentiel *Maven Central*

- ✓Le référentiel distant le plus utilisé et le plus complet pour la gestion des dépendances est le référentiel *central.*

- ✓Un site Web est disponible à l'adresse http://search.maven.org et propose de rechercher des artéfacts disponibles dans ce référentiel.



Installation dans un référentiel local

- L'installation d'un artéfact dans le référentiel local se fait lors la phase `install`.

- le référentiel local doit être considéré comme un emplacement de stockage temporaire qui peut être vidé de son contenu sans conséquence majeure. Il ne doit en aucun cas contenir des données du projet non stockées dans un emplacement de sauvegarde officiel.

- Lors de chaque nouvelle installation en local, l'artéfact précédent est supprimé et remplacé par le nouveau. Les dates sont mises à jour dans les fichiers XML.

diginamic FORMATION
« Votre passeport pour l'emploi numérique »

TP 4 : déploiement local

✓Récupérer le fichier maven_tp4.txt sur le drive.

Les plugins

Utilisation d'un plugin

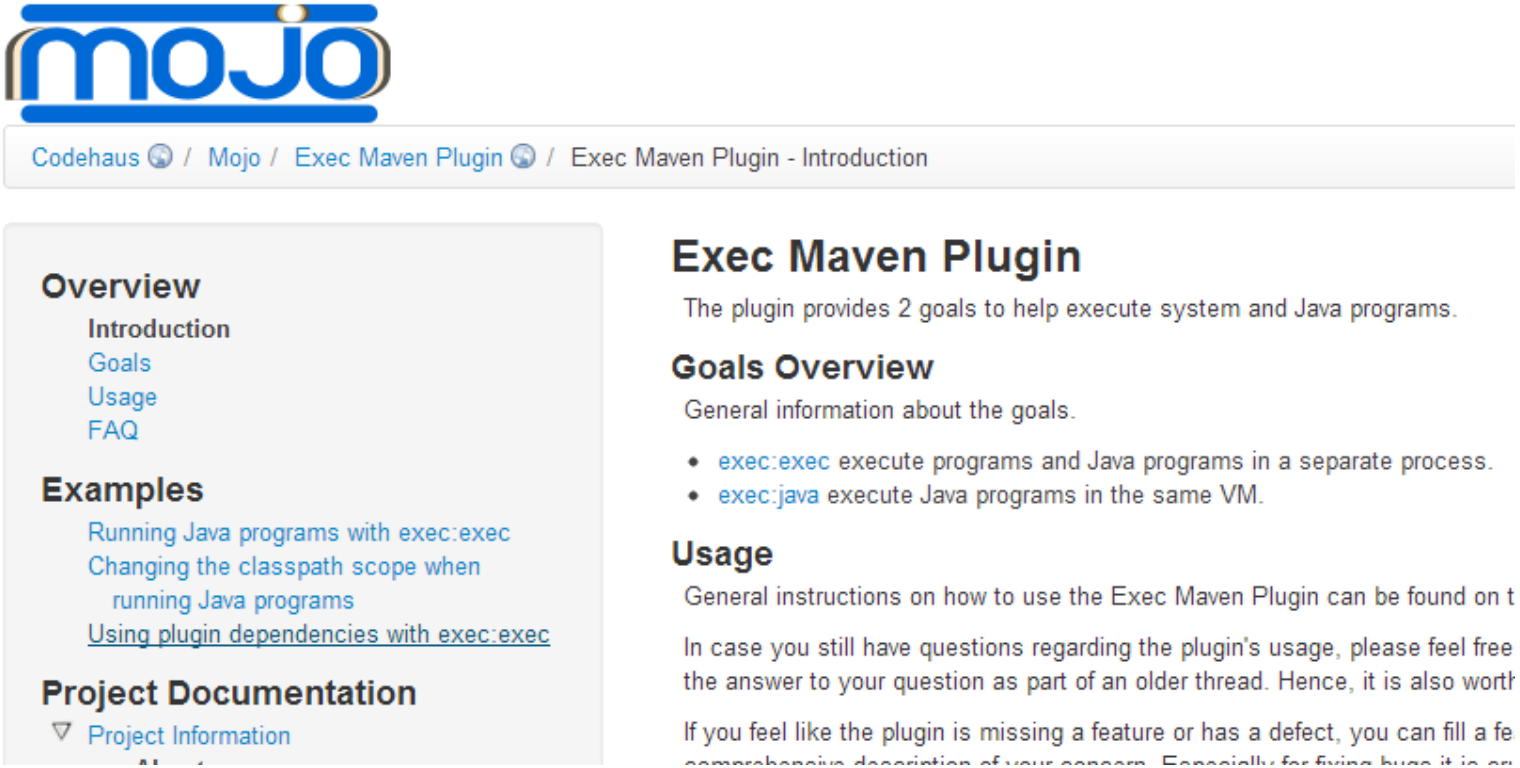
- ✓Les plugins sont configurés par l'intermédiaire de propriétés définies par goals.
- ✓Pour afficher la description d'un plugin, il est possible d'utiliser la commande suivante :

```
mvn help:describe -Dcmd=compiler:compile -Ddetail
```

```
$ mvn help:describe -Dcmd=compiler:compile
[INFO] [help:describe {execution: default-cli}]
[INFO] 'compiler:compile' is a plugin goal (aka mojo).
Mojo: 'compiler:compile'
compiler:compile
 Description: Compiles application sources
```

Utilisation d'un plugin (2)

✓Pour chaque plugin, il existe un site de documentation



Configuration d'un plugin

✓Configuration des paramètres globaux d'un plugin :

```
<plugin>
   <groupId>org.codehaus.mojo</groupId>
   <artifactId>exec-maven-plugin</artifactId>
   <configuration>
     <mainClass>org.formation.maven.mabiblio.App</mainClass>
   </configuration>
</plugin>
```

✓Modification des paramètres spécifique à une exécution :

```
<plugin>
   <groupId>org.codehaus.mojo</groupId>
   <artifactId>exec-maven-plugin</artifactId>
   <executions>
      <execution>
         <phase>test</phase>
         <goals>
            <goal>java</goal>
         </goals>
         <configuration>
           <mainClass>org.formation.maven.mabiblio.App</mainClass>
         </configuration>
     </execution>
   </executions>
</plugin>
```

diginamic FORMATION
« Votre passeport pour l'emploi numérique »

TP 5 : Quelques plugins

✓Récupérer le fichier maven_tp5.txt sur le drive

Le plugin versions-maven-plugin

- La gestion des numéros de version peut devenir une tâche contraignante lorsque les projets disposent de nombreux modules ou si l'héritage se réalise sur plusieurs niveaux.

- Pour optimiser la gestions des numéros de version, il existe le plugin `versions-maven-plugin`.

- Il propose plusieurs goals très utiles :
  - `versions:set` : modifie la version dans tous les POM du projet.
  - `versions:update-parent` : vérifie si le POM parent du projet est disponible dans une version plus récente et met à jour le POM du projet si nécessaire.
  - `versions:display-dependency-updates` : affiche la liste des dépendances du projet qui sont disponibles dans une version plus récente.
  - `versions:display-plugin-updates` : affiche la liste des plugins utilisés dans le projet qui sont disponibles dans une version plus récente que celle utilisée.

Héritage et multi-modules

Le `Corporate POM`

- ✓Le POM parent à tous les projets d'une application est appelé le `Corporate POM`.
- ✓Il est souvent identifié par un `artifactId` contenant le mot clé *parent*.
- ✓La mise en place d'un `Corporate POM` est l'occasion de renseigner les informations générales de l'application.

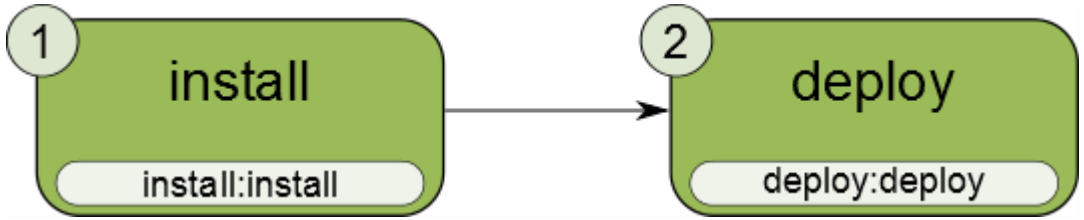
```
<project>
  . . .
  <groupId>org.formation.mabiblio</groupId>
  <artifactId>mabiblio-parent</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  . . .
</project>
```

Le type POM

- Pour créer un POM dont le but unique est de factoriser les informations communes à tous les projets, Maven propose le type de projet pom.

- Ce type de projet possède uniquement le fichier `pom.xml`.

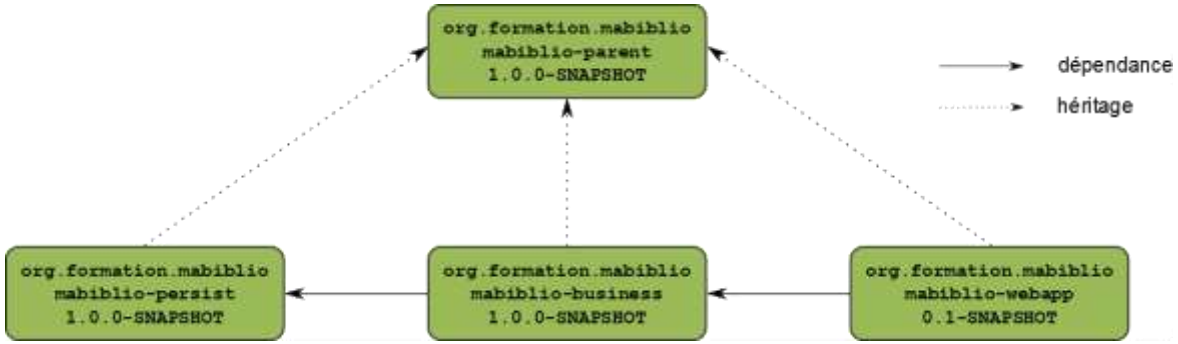
- Le cycle de vie d'un tel projet est minimal et contient uniquement deux phases :



- L'artéfact de sortie est le fichier `pom.xml`.

Le besoin d'héritage

- ✓Le développement d'une application peut nécessiter la création de plusieurs projets Maven.
- ✓Plusieurs informations peuvent être redondantes dans les POM des différents projets.
- ✓Au même titre que tous les POM héritent du Super POM, il devient nécessaire de créer un POM parent commun à tous les POM de l'application.



Héritage d'un POM

- Les POM des projets enfants doivent déclarer en premier lieu les coordonnées du POM parent afin d'activer le processus d'héritage grâce à l'élément `<parent />`.

```
<project>
  <parent>
    <groupId>org.formation.mabiblio</groupId>
    <artifactId>mabiblio-parent</artifactId>
    <version>1.0.0-SNAPSHOT</version>
  </parent>
  <artifactId>mabiblio-persist</artifactId>
  . . .
</project>
```

- Lors d'un héritage, les éléments `<groupId />` et `<version />` ne sont pas obligatoires dans le POM enfant. Les valeurs sont alors initialisées avec celles du POM parent. Néanmoins, Il est possible de les redéfinir.

Factoriser les dépendances

- La mise en place d'un `Corporate POM` permet de centraliser des informations communes.

- La notion d'héritage permet de déclarer les dépendances communes à tous les POM enfants directement dans le POM parent.

- Il est possible de redéfinir la version dans les POM enfants. La version d'une dépendance déclarée dans le POM enfant est prioritaire sur la version du POM parent.

Centraliser les informations

- ✓Il est possible de centraliser toutes les informations concernant les dépendances afin de maîtriser et d'alléger les dépendances des POM enfants.
- ✓L'élément `<dependencyManagement />` permet de centraliser :
  - Les versions des dépendances à utiliser,
  - Les dépendances transitives à exclure,
  - Les champs d'application des dépendances.

```
<dependencyManagement>
   <dependencies>
      <dependency>
         <groupId>log4j</groupId>
         <artifactId>log4j</artifactId>
         <version>1.2.13</version>
         <scope>compile</scope>
      </dependency>
   </dependencies>
</dependencyManagement>
```

```
<dependencies>
   <dependency>
     <groupId>log4j</groupId>
     <artifactId>log4j</artifactId>
   </dependency>
</dependencies>
```

Centraliser les informations (2)

- ✓L'élément `<pluginManagement />` est l'équivalent pour la gestion des plugins de l'élément `<dependencyManagement />` pour les dépendances du POM.

- ✓Il permet de centraliser les versions et les configurations des plugins pour l'ensemble des POM enfants.

```
<pluginManagement>
   <plugins>
      <plugin>
         <groupId>org.apache.maven.plugins</groupId>
         <artifactId>maven-compiler-plugin</artifactId>
         <version>3.0</version>
         <configuration>
            <source>1.6</source>
            <target>1.6</target>
         </configuration>
      </plugin>
   </plugins>
</pluginManagement>
```

```
<plugins>
   <plugin>
     <groupId>org.apache.maven.plugin</groupId>
     <artifactId>maven-compiler-plugin</artifactId>
   </plugin>
</plugins>
```

Les projets multi-modules

- ✓L'organisation des projets et les dépendances qu'ils possèdent les uns envers les autres peut devenir compliquer à gérer.
- ✓De nombreuses manipulations peuvent être nécessaires dès lors qu'un projet est modifié.
- ✓Maven propose une solution pour éviter ces manipulations en associant des projets ensemble à travers le concept de modules.

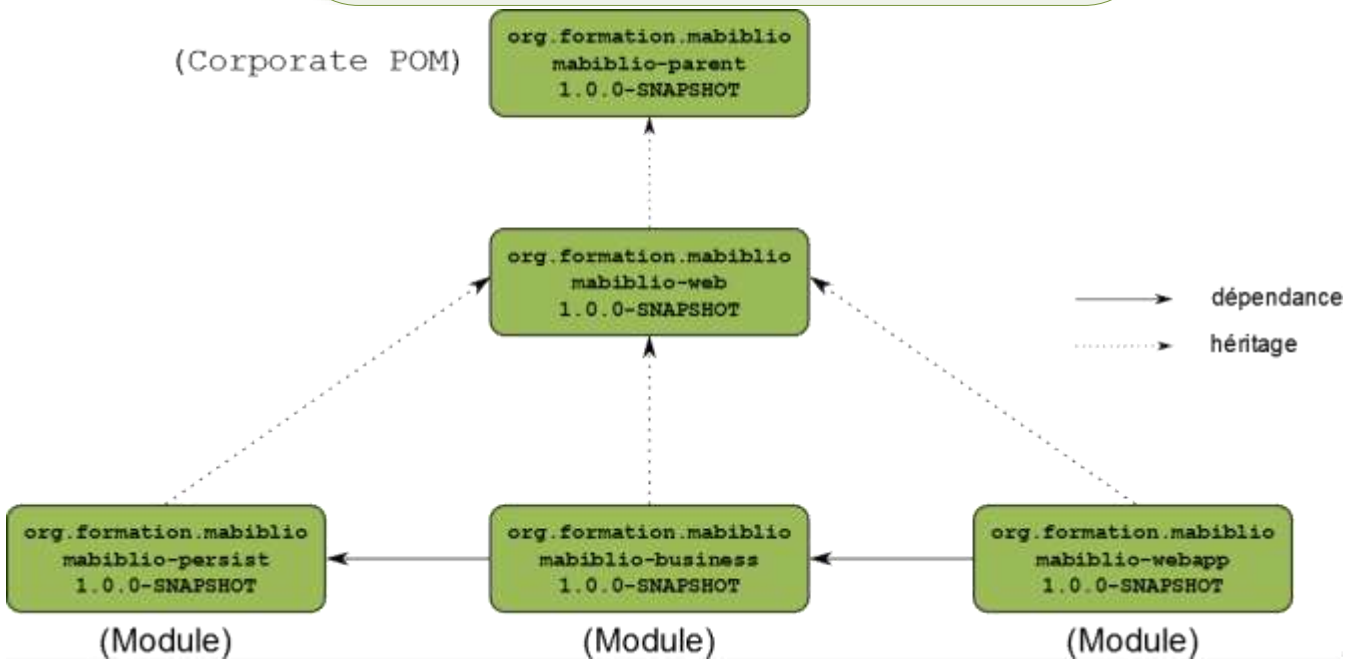
Les projets multi-modules

- ✓Maven permet de créer un projet de type pom, dont le but unique est de lister les projets qu'il identifie comme ses modules.
- ✓Une seule commande appelée sur ce POM effectue les phases sur chaque projet en respectant leurs dépendances et donc l'ordre correct d'exécution.
- ✓Maven utilise ce que l'on appel le "reactor" pour orchestrer dans un projet l'appel des différents modules en fonction de leurs dépendances.

Organisation d'un projet multi-modules

```
mabiblio-web
|
+- mabiblio-persist/
| +- src/
| +-pom.xml
|
+- mabiblio-business/
| +- src/
| +-pom.xml
|
+- mabiblio-webapp/
| +- src/
| +-pom.xml
|
+- pom.xml
```

```
<project>
   . . .
   <modules>
      <module>mabiblio-webapp</module>
      <module>mabiblio-persistence</module>
      <module>mabiblio-business</module>
   </modules>
   . . .
</project>
```

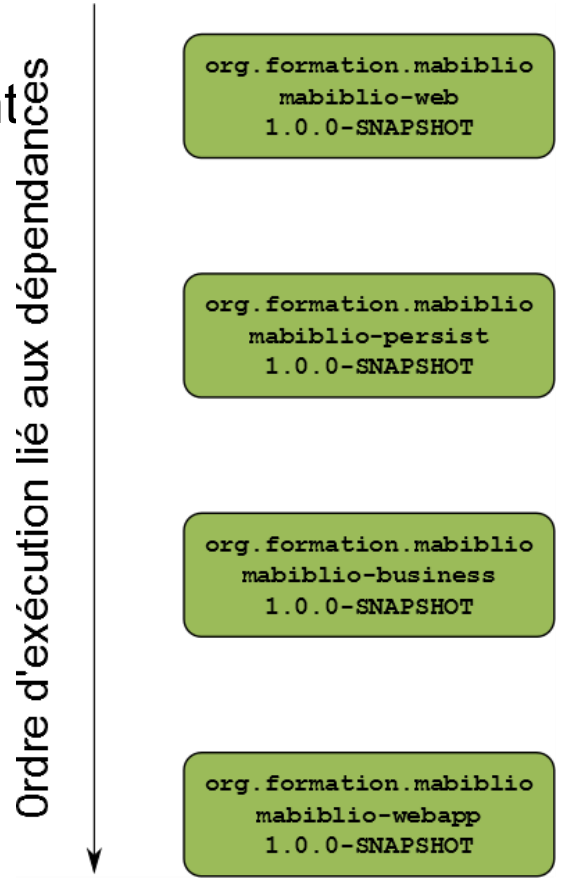


Traitement d'un projet multi-modules

```
$ mvn install
```

✓L'exécution de cette commande sur le projet donne le résultat suivant

```
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------
[INFO] Reactor Build Order:
[INFO]
[INFO] mabiblio-web
[INFO] mabiblio-persistence
[INFO] mabiblio-business
[INFO] mabiblio-webapp
[INFO]
[INFO] ------------------------------------------------------------
[INFO] Building mabiblio-web 1.0.0-SNAPSHOT
[INFO] ------------------------------------------------------------
```



TP 6 Création d'un projet multi-modules

✓Récupérer le fichier maven_tp6.txt sur le drive

Les propriétés et profils

Les propriétés

- ✓Il est possible d'utiliser des propriétés existantes ou de déclarer de nouvelles propriétés dans le POM.
- ✓Pour déclarer une propriété, il faut utiliser l'élément `<properties />`
- ✓Pour utiliser une propriété, la syntaxe est la suivante :

```
${nom.de.la.propriété}
```

- ✓Le nom de la balise XML est alors le nom de la propriété.

```
<properties>
   <log4j.version>1.2.13</log4j.version>
</properties>
```

Les propriétés (2)

- Maven fournit trois variables implicites qui peuvent être utilisées pour accéder aux variables d'environnement, aux informations du POM et à votre configuration de Maven :
  - `project` : permet d'accéder aux différents éléments du POM en utilisant une notation pointée pour référencer la valeur d'un élément du POM.

    Ex : `${project.build.filename}`
  - `env` : permet d'accéder aux variables d'environnement du système d'exploitation. Ex : `${env.PATH}`
  - `settings` : permet d'accéder aux informations de la configuration de Maven en utilisant une notation pointée pour référencer la valeur d'un élément du fichier `settings.xml`. Ex : `${settings.offline}`

- Toutes les propriétés accessibles via la méthode `getProperties()` de la classe `java.lang.System` sont visibles comme propriétés du POM
  - Ex : `${user.name}`, `${user.home}`, `${java.home}`, et `${os.name}`

Les filtrage de ressources

- Les valeurs des propriétés définies dans le POM peuvent être insérées dans les fichiers de configuration du projet.

- Cette fonctionnalité est possible grâce au plugin *maven-resources-plugin*. Pour que la configuration du plugin soit active, il faut autoriser les ressources du projet à être filtrées

- Les propriétés au format Maven présentes dans les fichiers contenus dans les ressources filtrées sont remplacées par leur valeur lors de l'appel du goal `resource:resource`

Les filtrage de ressources (2)

- Exemple :

```
log4j.rootLogger=${logger.level}, A1
. . .
```

src/main/resources/log4j.properties

```
logger.level=DEBUG
. . .
```

src/main/filters/dev.properties

```
<build>
   . . .
   <resources>
      <resource>
         <directory>src/main/resources</directory>
         <filtering>true</filtering>
      </resource>
   </resources>
   . . .
</build>
```

```
<build>
   . . .
   <filters>
      <filter>src/main/filters/dev.properties</filter>
   </filters>
   . . .
</build>
```

- Il est possible de configurer dans le POM des fichiers définissant des propriétés utilisées lors du filtrage des ressources à l'aide de l'élément `<filters />`.

Les profils

- ✓Un profil est un sous-modèle du POM.
- ✓Il peut être déclaré dans le fichier `settings.xml` ou dans le POM du projet.
- ✓Contrairement aux autres éléments du POM, les éléments compris dans les profils ne vont pas être obligatoirement pris en compte lors de l'exécution d'une commande Maven sur un projet.

```
<profiles>
   <profile>
      <id>profil-1</id>
      . . .
   </profile>
   <profile>
      . . .
   </profile>
   . . .
</profiles>
```

Configuration des profils

- Il est important d'identifier les éléments mis à disposition selon l'emplacement de la déclaration du profil.

- Eléments du profil dans le POM :
  - Le modèle de données du profil est aussi complet que son élément parent à l'exception des coordonnées qui identifie le projet et des informations générales du projet.

- Eléments du profil dans le `settings.xml` :
  - Il y a moins d'éléments. Il est possible de définir des propriétés et des référentiels.

Activation de profils

✓Activation par leurs identifiants à l'aide du paramètre

```
$ mvn -P profil-1 install
```

✓Activation par rapport :

- Au système d'exploitation
- À la version du JDK
- À la présence ou non d'une propriété
- À la présence ou non d'un fichier

```
<profile>
<id>profil-1</id>
<activation>
   <os>
      <name>windows 7</name>
      <arch>x86</arch>
      <family>windows</family>
      <version>6.1</version>
   </os>
   <jdk>1.6</jdk>
   <property>
      <name>...</name>
      <value>...</value>
   </property>
   <file>
      <exists>...</exists>
      <missing>...</missing>
   </file>
</activation>
. . .
</profile>
```

Activation de profils (2)

- Il est possible d'activer un profil par défaut.

```
<profile>
   . . .
   <activation>
      <activeByDefault>true</activeByDefault>
   </activation>
. . .
</profile>
```

- Le plugin *maven-help-plugin* propose des goals qui permettent :
  - De lister les profils définis pour le projet :
  - De lister les profils actifs

```
$ mvn help:all-profiles
$ mvn help:active-profiles
```

diginamic FORMATION
« Votre passeport pour l'emploi numérique »

TP 7 : Ressources et profils

✓Récupérer le fichier maven_tp7.txt sur le drive

En conclusion

- En cours développement, utiliser des snaphots
- Utiliser dependencyManagement
- Utiliser pluginManagement
- Utiliser les modules et les héritages
- Utiliser le même numéro de version dans tout le projet
- Comprendre son fonctionnement
- Utiliser un serveur d'intégration continue (Hudson, Jenkins, Continuum, …)

Pour aller plus loin …

- http://maven.apache.org/
- http://maven-guide-fr.erwan-alliaume.com/

diginamic FORMATION
« Votre passeport pour l'emploi numérique »

Restons en contact

Votre contact

DIGINAMIC formation
Lionel Cabon, Directeur
contact@diginamic.fr

N° SIRET : 818 241 978 00019
N° Déclaration OF : 91 34 08867 34
Nantes, Paris, Montpellier
www.diginamic.fr