Chapitre 3
Organisation et dépendances des projets

1. La gestion des dépendances

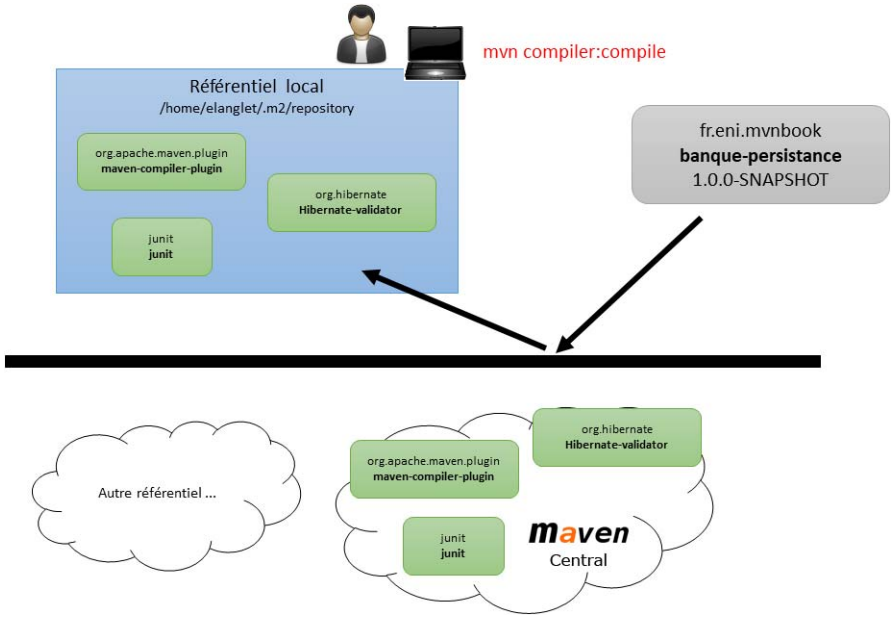
1.1 Définition et problématiques de la gestion des dépendances

La gestion de dépendances est essentielle dans l'univers d'Apache Maven. C'est une notion qui paraît simple au départ mais qui peut générer des problèmes et des incompréhensions lors de la construction ou de l'exécution des projets si elle n'est pas correctement maîtrisée.

Pour appréhender au mieux cette gestion des bibliothèques logicielles internes ou externes au projet, il était nécessaire de comprendre la philosophie d'Apache Maven autour des cycles de vie des projets et des plugins. Les deux premiers chapitres ont été réalisés dans ce sens.

Le **principe général des dépendances** dans Apache Maven est le suivant : les classes du projet ont besoin d'autres classes dans leur processus de fonctionnement (compilation, exécution), il faut donc **identifier et importer des référentiels distants vers le référentiel local** la ou les **bibliothèques logicielles** qui contiennent ces classes. Le projet possède alors des dépendances vers ces bibliothèques qui seront au final **ajoutées au chemin de génération** (*Class-Path*) du projet.

Ce fonctionnement est schématisé pour le cas du projet de persistance.



1.1.1 Les éléments du POM concernés

Les exemples du projet `banque-persistance` ont mis en évidence dans les premiers chapitres quelques éléments au niveau du POM. Les éléments centraux impliqués dans la gestion des dépendances sont les suivants :

```
<project>
  ...
  <dependencyManagement/>
  <dependencies/>

  <repositories/>
  ...
  <profiles/>
</project>
```

Remarque

À noter que les éléments déterminants `<dependencyManagement>`, `<dependencies>` *et* `<repositories>` *peuvent aussi être déclarés dans les profils (élément* `<profiles>`*) du POM. La notion de profil est détaillée ultérieurement dans le livre.*

1.1.2 Illustration des dépendances

Le plugin officiel de la fondation Apache centré sur la notion de dépendances dans Apache Maven est le plugin `maven-dependency-plugin`. Il est extrêmement utile pour afficher toute sorte d'informations liées au graphe de dépendances des projets. Il sera donc souvent utilisé dans ce chapitre avec notamment le MOJO `dependency:tree` qui donne une vue synthétique des dépendances résolues.

Il utilise le mécanisme de gestion des dépendances basée sur l'API Aether incluse dans le cœur du produit depuis Apache Maven 3.0. Cette nouvelle résolution des dépendances par rapport à Maven 2.x est plus robuste et ajoute d'importantes fonctionnalités, pour les constructions notamment, qui sont détaillées dans ce chapitre.

Le projet `banque-persistance` déclare, à cette étape du livre, uniquement quatre dépendances :

```
<project>
...
  <artifactId>banque-persistance</artifactId>
...
  <dependencies>
   <dependency>
     <groupId>org.hibernate</groupId>
     <artifactId>hibernate-validator-annotation-processor
</artifactId>
     <version>5.4.2.Final</version>
   </dependency>
   <dependency>
     <groupId>org.hibernate</groupId>
     <artifactId>hibernate-validator</artifactId>
     <version>5.4.2.Final</version>
   </dependency>
   <dependency>
```

```
      <groupId>org.glassfish.web</groupId>
      <artifactId>el-impl</artifactId>
      <version>2.2</version>
    </dependency>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
...
</project>
```

Pour autant, l'affichage des dépendances sous la forme d'une arborescence recense sept bibliothèques logicielles nécessaires au projet :

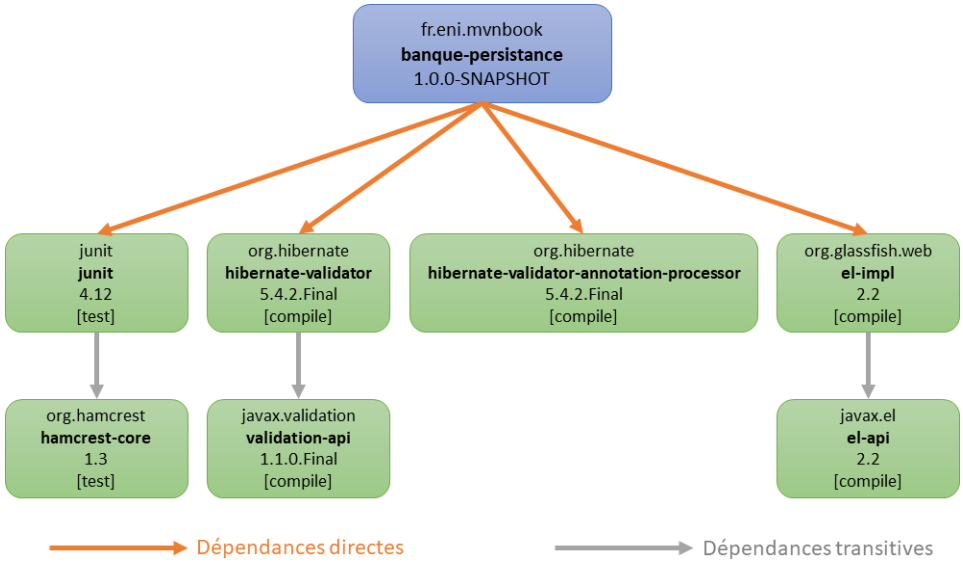
```
$ mvn dependency:tree
...
[INFO] --- maven-dependency-plugin:2.8:tree (default-cli) @ banque-persis-
tance ---[INFO] fr.eni.mvnbook:banque-persistance:jar:1.0.0-SNAPSHOT
[INFO] +- junit:junit:jar:4.12:test
[INFO] |  \- org.hamcrest:hamcrest-core:jar:1.3:test
[INFO] +- org.hibernate:hibernate-validator:jar:5.4.2.Final:compile
[INFO] |  \- javax.validation:validation-api:jar:1.1.0.Final:compile
[INFO] +- org.hibernate:hibernate-v-a-processor:jar:5.4.2.Final:compile
[INFO] +- org.glassfish.web:el-impl:jar:2.2:compile
[INFO] |  \- javax.el:el-api:jar:2.2:compile
...
```

Cet exemple est détaillé et utilisé dans les paragraphes suivants afin d'illustrer les notions d'Apache Maven qu'il met en œuvre.

Remarque

Pour une meilleure lisibilité du graphe, la dépendance avec l'`artifactId` `hibernate-validation-annotation-processor` *est notée de la façon suivante :* `hibernate-v-a-processor`.

Les commandes Apache Maven, dont le résultat consiste à afficher les relations des dépendances du projet, peuvent aussi être illustrées dans la suite du chapitre par des schémas tels que le suivant.



Les notions de dépendances directes et transitives sont expliquées dans ce chapitre.

1.2 Les champs d'application (scope) des dépendances

1.2.1 Définition

Chaque dépendance doit définir son **champ d'application (`scope`) au sein du POM**, c'est-à-dire préciser la façon dont elle est utilisée dans les différents processus du projet. Cette information a notamment des **conséquences sur la présence de la bibliothèque** dans les chemins de génération et d'exécution des classes (`Class-Path`) ainsi que lors du packaging du projet en vue du déploiement sur un serveur.

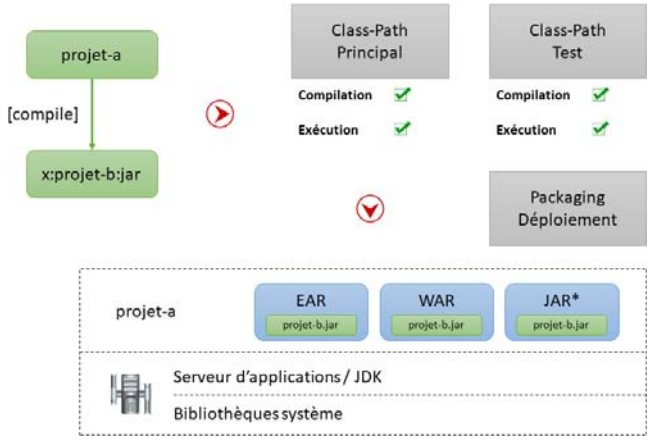
1.2.2 Le scope compile

Le scope **`compile`** est la **valeur par défaut** donnée à une dépendance qui ne définit pas de champ d'application. Comme son nom l'indique, il précise que la **bibliothèque** en question est nécessaire dans le processus de compilation du projet. En résumé, l'archive représentant la dépendance est **incluse dans tous les `Class-Path`** du projet.

Dans l'exemple du projet, toutes les annotations utilisées pour valider les objets du modèle (`@NotNull`, `@Size`...) sont importées dans le code et font références à des classes et interfaces définies dans la bibliothèque `hibernate-validator-annotation-processor`.

En règle générale, lors de l'import d'une classe Java externe dans le code source d'un projet, il est commun de définir la dépendance vers la bibliothèque associée avec le champ d'application `compile`.

Le schéma suivant illustre les impacts du scope `compile` sur la présence de la bibliothèque logicielle associée, dans les processus de `Class-Path`, de packaging et de déploiement des projets Apache Maven.



La notation *JAR**, sur le schéma précédent, signifie que cette archive JAR n'est pas créée avec le plugin `maven-jar-plugin`, qui, lui, englobe uniquement les fichiers sources compilés et les fichiers ressources du projet. C'est l'utilisation du plugin `maven-shade-plugin` qui permet la création d'une archive Java complète contenant en surplus les dépendances définies dans le POM.