Formation Apache Maven

TP 01 - Prise en main

Table of Contents

Installation

- Télécharger Maven (http://maven.apache.org/download.cgi, fichier : apache-maven-3.x.x-bin.zip).
- Décompresser l'archive (à la racine par exemple du lecteur C:).
- Créer une variable d'environnement JAVA_HOME avec comme valeur le chemin vers votre JDK installé (par exemple : *C:\Program Files\Java\jdk.1.8_45*).
- Mettre à jour la variable PATH pour que la commande *mvn* (situé dans le répertoire *bin*) soit accessible.
- Vérifier avec une invite de commande que la commande *mvn* fonctionne.

```
mvn --version
Apache Maven 3.5.0 (ff8f5e7444045639af65f6095c62210b5713f426; 2017-04-
03T21:39:06+02:00)
Maven home: /usr/local/Cellar/maven/3.5.0/libexec
Java version: 1.8.0_45, vendor: Oracle Corporation
Java home: /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "mac os x", version: "10.12.5", arch: "x86_64", family: "mac"
```

Etape 1 - Génération d'un projet Maven Simple

- Maven intègre le plugin *Archetype* qui permet de générer un projet à partir d'un squelette.
- Lancer la commande suivante :

```
mvn archetype:generate
```

Dans la commande ci-dessus, *archetype* représente le nom court du plugin, et *generate* la tâche invoquée du plugin.

La tâche *generate* lance le processus de génération de projet.

- Le plugin vous demande choisir un squelette parmi les 1800+ disponibles. Nous allons utiliser le plugin le plus simple (configuré par défaut). A cette étape, appuyer sur la touche entrée.
- Aller jusqu'au bout de la génération avec les informations suivantes :
  - Number : **6**
  - groupId : **dev**
  - artifactId : **premier-projet-maven**
  - version : **1.0-SNAPSHOT**

    - package : **dev**

- Vous devrier obtenir le squelette suivant :

```
/premier-projet-maven
    pom.xml                         ①
    /src
        /main
            /java
                /dev
                    App.java        ②
        /test
            /java
                /dev
                    AppTest.java    ③
```

① : fichier de configuration Maven

② : fichier java main

③ : fichier de test unitaire JUnit

    - Sauriez-vous expliquer la configuration contenu dans le fichier *pom.xml* ?

```
<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">
  <modelVersion>4.0.0</modelVersion>

  <groupId>dev</groupId>
  <artifactId>premier-projet-maven</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>premier-projet-maven</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

- **PAUSE GIT** : Sauvegarder dans Github.

Etape 2 - Compilation

- Compiler le projet avec Maven (en ligne de commande) :
  - *mvn compile* (visualiser le contenu du répertoire *target*)
  - Modifier une classe (*App.java*) pour introduire une erreur de compilation
  - *mvn compile*
  - *mvn clean* (visualiser le contenu du répertoire *target*)
  - *mvn clean compile* (visualiser le contenu du répertoire *target*)
  - Corriger l'erreur de compilation
  - *mvn clean compile*
- **PAUSE GIT** : Le répertoire *target* ne doit jamais faire partie de vos commits Git. Configurer Git pour que ce répertoire ne fasse pas partie des commits.

Etape 3 - Assembler

- Assembler un projet avec Maven

```
mvn package
```

- Visualiser le contenu du répertoire *target* et le fichier JAR généré.

Etape 4 - Installer un projet

- Lancer la commande suivante :

```
mvn install
```

- Vérifier que la présence de l'artefact dans le répertoire :

```
~/.m2/repository/dev/premier-projet-maven/1.0-SNAPSHOT/...
```

Etape 5 - Propriétés et variables

- Créer à présent une nouvelle propriété :

```
<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">
  <modelVersion>4.0.0</modelVersion>

  ...

  <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
      <junit.version>3.8.1</junit.version> ①
  </properties>

  ...
</project>
```

① : création d'une propriété dont le nom est *junit.version* et la valeur *3.8.1*.

- Réutiliser cette propriété dans la définition de la dépendance :

```
<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">
  <modelVersion>4.0.0</modelVersion>

    ...

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <junit.version>3.8.1</junit.version>
  </properties>

   ...

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>${junit.version}</version> ①
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

① Réutilisation de la propriété *junit.version*.

- Nous allons à présent personnaliser le nom du livrable généré par la commande *mvn package*. Cette opération s'éffectue avec la configuration suivante :

```
<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">
  <modelVersion>4.0.0</modelVersion>

  ...

  <build>
        <finalName>xxx</finalName> ①
  </build>

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

① *xxx* représente le nom du livrable (fichier xxx.jar).

Nous souhaitons que le nom du fichier jar soit le même que celui de l'artefactId (ici : *premier-projet-maven*).

- Modifier la configuration comme suit :

```
<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">
  <modelVersion>4.0.0</modelVersion>

  ...

  <build>
        <finalName>${project.artifactId}</finalName> ①
  </build>

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

① la propriété *project* est générée par défaut par Maven. Elle représente le modèle objet du projet courant.

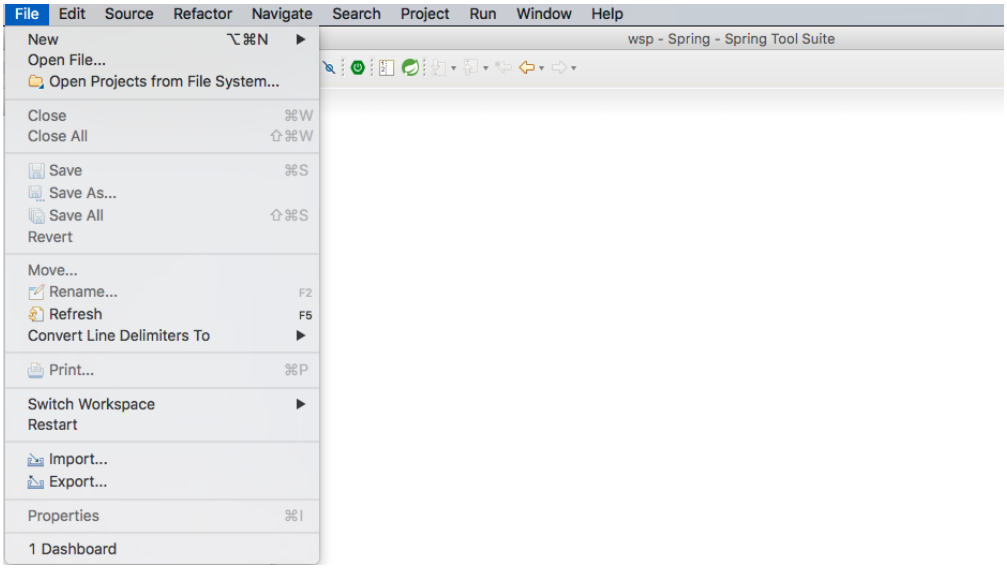
- Assembler à nouveau :

```
mvn package
```

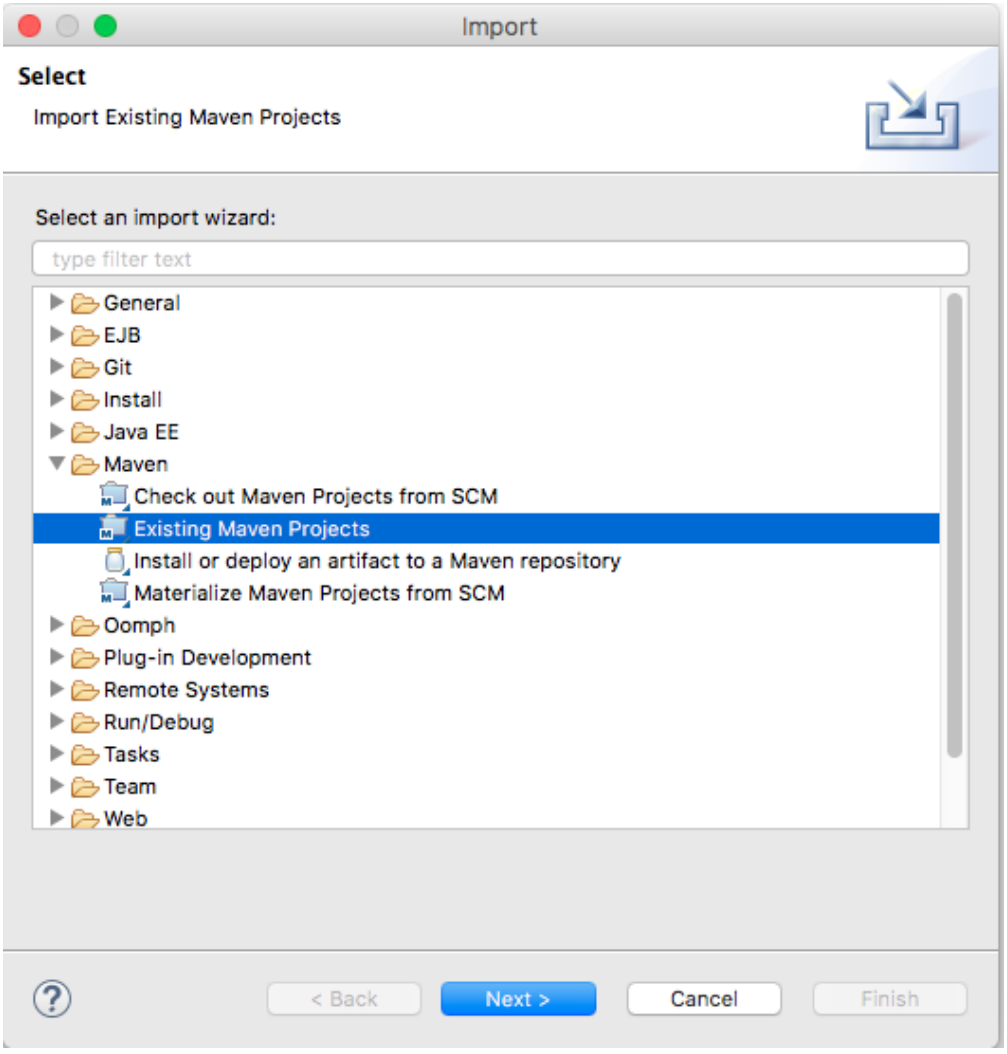
- Visualiser le contenu du répertoire *target* et le fichier JAR généré. Que constatez-vous ?
- **PAUSE GIT** : Sauvegarder dans Github.

Etape 6 : Intégration IDE

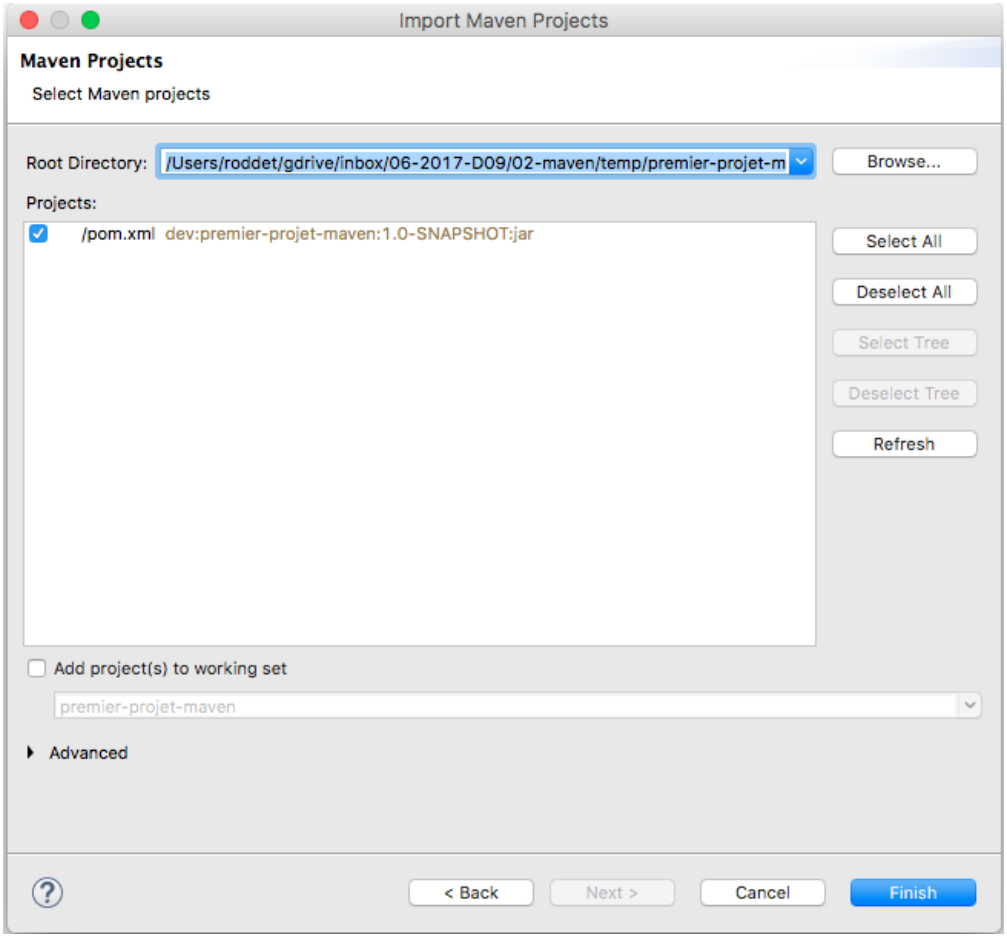
- Lancer *Spring Tools Suite*.
- Cliquer sur le menu *File* > *Import...*.



- Se rendre dans la rubrique *Maven* et choisir *Existing Maven Projects*.



- Sélectionner le répertoire projet puis cliquer sur Finish.



- Vous rendre dans le menu *Window* > *Preferences...* puis la rubrique *Java* > *Installed JREs*.
- Vérifier que le JDK (et non un JRE) est référencé. Si ce n'est pas le cas :
  - Cliquer sur *Add...*
  - Choisir l'option *Standard VM*
  - Choisir le répertoire d'installation du JDK (par exemple : *C:\Program Files\Java\jdk.1.8_45*).
  - Cliquer sur *Finish*
  - Assurer vous que c'est bien le JDK qui est sélectionné par défaut.
- Créer une tâche Maven pour assembler le projet :
  - Clic droit sur le projet > Run As > Maven build...
  - Dans la rubrique *Goals*, compléter : *clean package*.
  - Cliquer sur *Run*
  - Vérifier que la tâche s'est bien déroulée.
- Ouvrir le fichier *pom.xml* et parcourir tous les onglets. Des questions ?
- Le projet est correctement configurée :
  - Pour lancer l'application : clic droit sur *App.java* > Run As > Java Application.
  - Pour lancer le test unitaire : clic droit sur *AppTest.java* > Run As > JUnit Test.
- **PAUSE GIT** : Ignorer toutes les fichiers et répertoires générés par STS (.settings, .project, .classpath, ...). Sauvegarder dans Github.

Etape 7 - Intégration d’une librairie tierce

Nous allons la librairie *Jfiglet* pour afficher un titre à l’application de la forme suivante :

```
    _
   / \    _ __   _ __
  / _ \  | '_ \ | '_ \
 / ___ \ | |_) || |_) |
/_/   \_\| .__/ | .__/
         |_|    |_|
```

Sauriez-vous le mettre en oeuvre uniquement à l’aide de la documentation du projet : https://lalyos.github.io/jfiglet/ ?

- **PAUSE GIT** : Sauvegarder dans Github.