Atelier 01. Introduction à Maven
Gestion des dépendances

Apache Maven est un logiciel de gestion de projet. Maven peut gérer la construction, le reporting et la documentation d'un projet à partir d'un fichierXML.

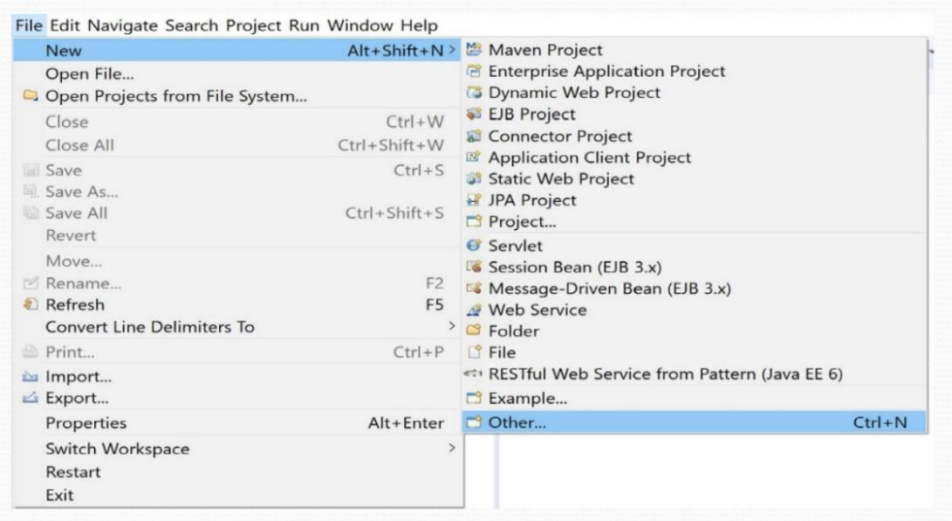
POM est un fichier XML (Project Object Model) qui contient des informations et les détails de configuration nécessaires sur le projet. Il contient des valeurs par défaut pour la plupart des projets. Les informations qui peuvent être définies dans le POM sont les dépendances du projet, les plugins qui peuvent être exécutés et, bien sûr, les profils de construction.

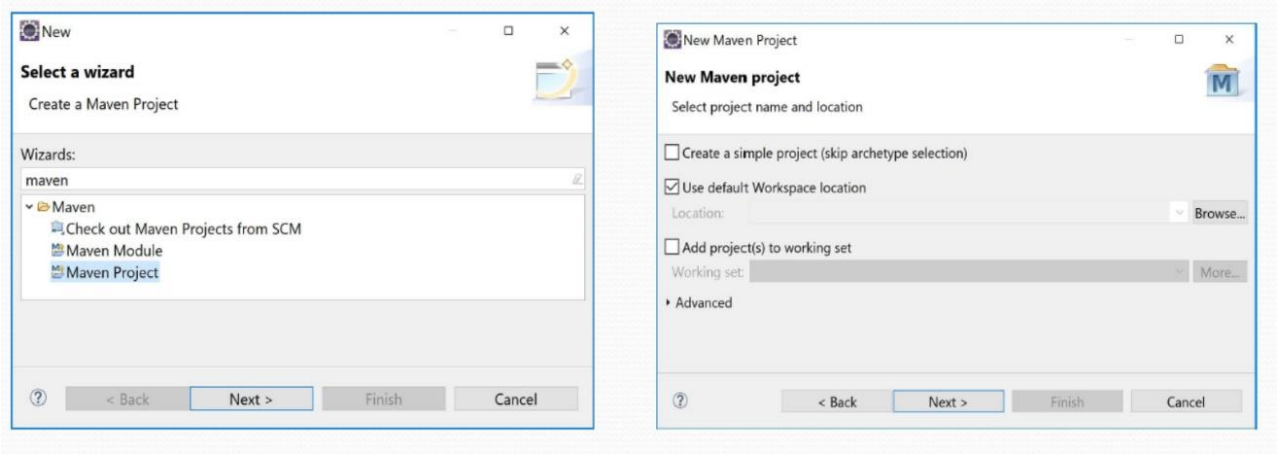
Éléments utilisés lors de la création du fichier pom.xml :

- **project**- Project est l'élément racine du fichier pom.xml.
- **modelVersion** - Cela signifie la version du modèle POM avec laquelle vous travaillez.
- **groupId** - Cela implique l'identifiant du groupe de projet. Il est unique et, le plus souvent, vous appliquerez un ID de groupe lié au nom du package Java racine.
- **artifactId** - Ceci est utilisé pour fournir le nom du projet que vous construisez.
- **Version** - Cet élément se compose du numéro de version du projet. Si votre projet a été publié en différentes versions, il est pratique de présenter la version de votre projet.

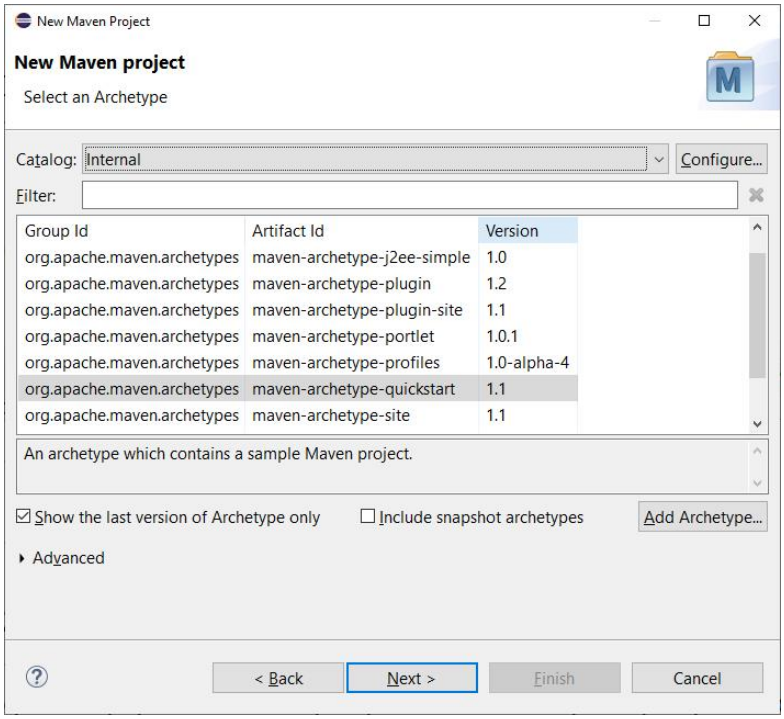
1.1 Création d'un projet Maven

- Maven permet de créer des projets typé (java, javeEE, web…).
- File->New->Other -> Maven -> Maven Project

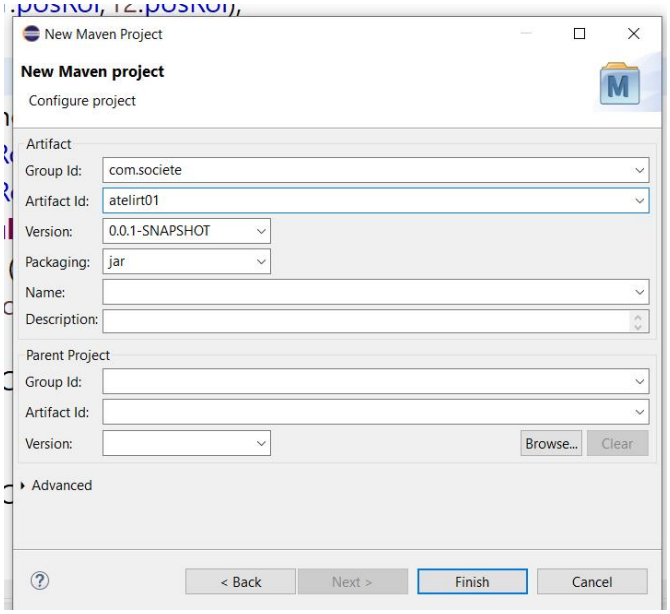




Sélectionner Maven project puis next



Sélectionner maven-archetype-quickstart



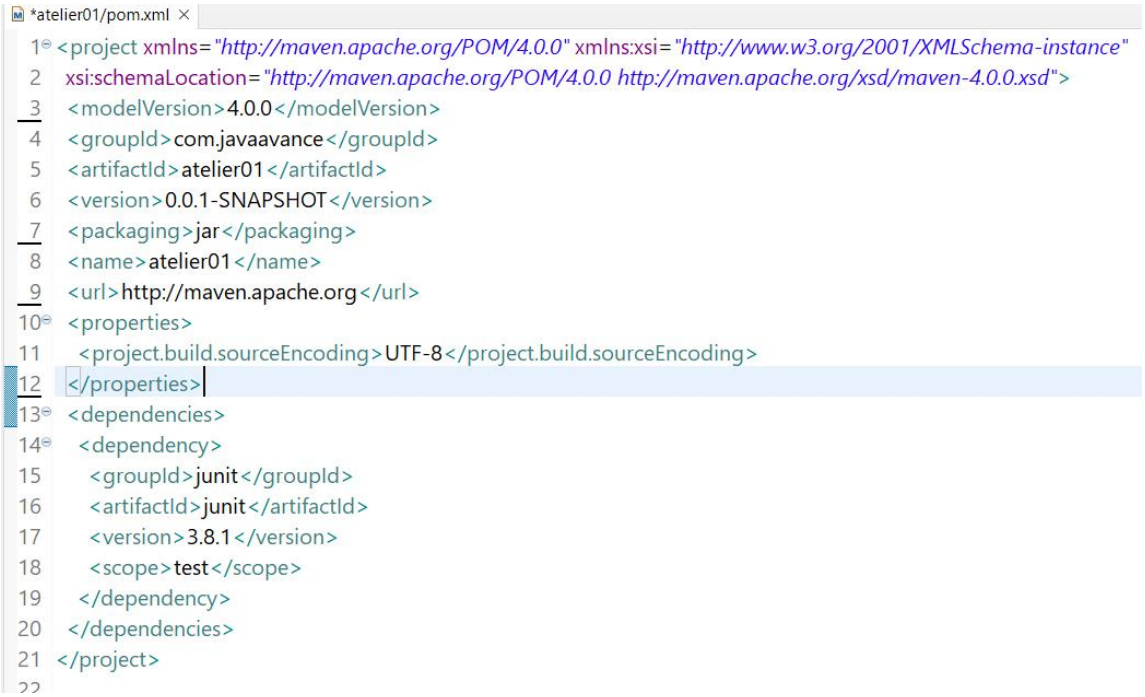
Remplir les champs

1.2 La structure d'un projet Maven

Le projet est créé avec une structure standard. Regardez la structure.

- Maven utilise une structure standard pour les ressources du projet :
- src/main/java : les src java
- src/test/java : les tests java
- target : les fichiers créés lors de la compilation

La structure d'un projet maven



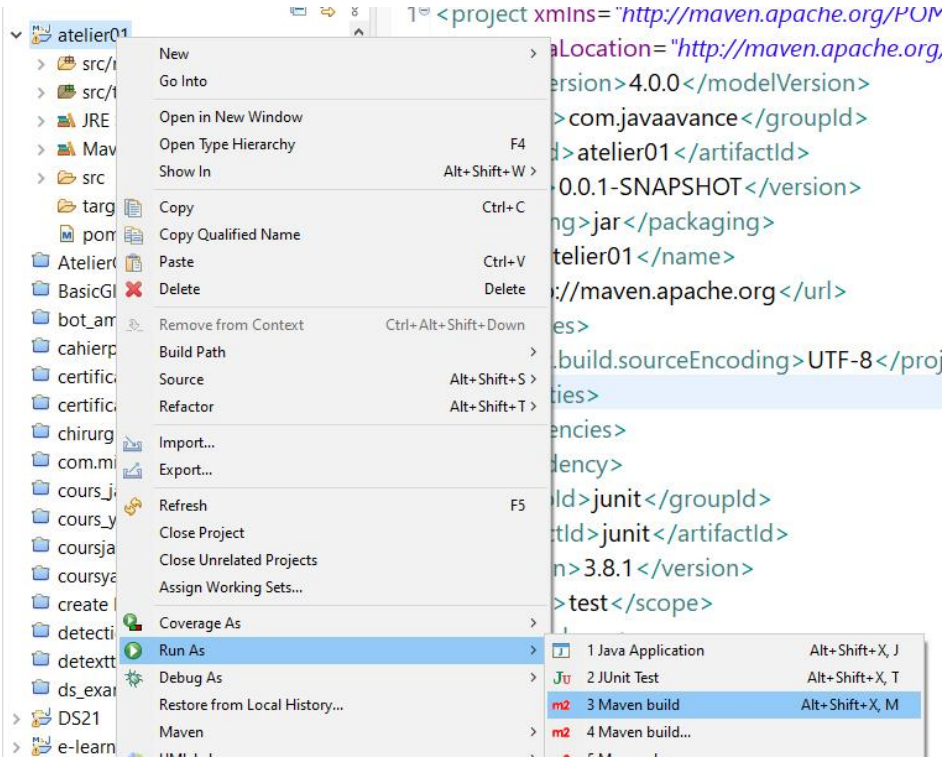
La structure du fichier XML Pom

Le fichier pom.xml est le cœur de la configuration du projet. Il contient toutes les informations nécessaires au build du projet.

1.3 Exécution du projet

Sous eclipse

- Run As->Maven buid



Il y a 3 build lifecycles de base dans Maven :

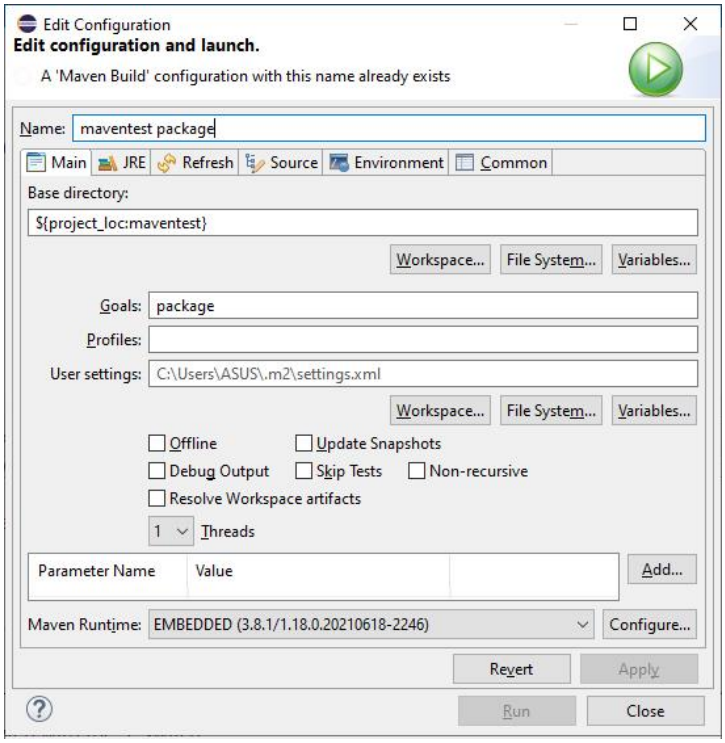
- default : qui permet de construire et déployer le projet
- clean : qui permet de nettoyer le projet en supprimant les éléments issus de la construction de celui-ci
- site : qui permet de créer un site web pour le projet

Ces build lifecycles sont découpés en phases qui sont exécutées séquentiellement les unes à la suite des autres.

Si je prends l'exemple du build lifecycledefault, nous y retrouvons, entre autres, les phases :

- **validate** : vérifie que la configuration projet est correcte (POM, pas d'éléments manquants...)
- **compile** : compile les sources du projet
- test : teste le code compilé avec les classes de tests unitaires contenues dans le projet
- **package** : package les éléments issus de la compilation dans un format distribuable (JAR, WAR...)
- **install** : installe le package dans votre repository local
- **deploy** : envoie le package dans le repository distant défini dans le POM

Ainsi, la construction du projet (le build lifecycle) est un enchaînement d'étapes (les phases) permettant d'obtenir le résultat final.



Avec l'option package : Maven crée un dossier de travail nommé target dans lequel il stocke les fichiers produits, compiler les sources, les tests unitaires, exécuter les tests unitaires et créer le fichier jar.

1.4 Dépendances

Maven ajoute la possibilité de gérer automatiquement les dépendances logicielles. Pour développer une application, nous allons avoir besoin de bibliothèques externes (les fichiers .jar en Java). Plutôt que d'aller les télécharger une à une depuis le Web et de les ajouter dans Eclipse, nous allons signaler à Maven l'identifiant des dépendances dont nous aurons besoin et il va se charger pour nous de les télécharger depuis un référentiel centralisé (Maven central repository), de les stocker dans un cache sur la machine et de les associer à notre projet.

Consulter le site : https://mvnrepository.com/

Dans notre cas

Gson est une bibliothèque open source développée par Google pour convertir un objet Java dans sa représentation JSON et vice versa.

```
<dependency>
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.9.0</version>
</dependency>
```

Convertir une chaine vers liste d'objet

```
public static void main(String[] args)     {
  Gson gson = new Gson();
  Person[] persons = gson.fromJson(getGson(), Person[].class);
  for(Person person : persons)
    System.out.println(person.toString());
  }
```

Code de la méthode main

```
public static String getGson() {
return "["
+"{\"id\":8484, \"nom\":\"Ali\",\"salaire\":1732};"
+"{\"id\":8485, \"nom\":\"salah\", \"salaire\":1837}"
+ "]";
}
```

Code de la méthode getGson

- Tester le programme