

Déployer le projet sur le serveur

Signer le code

Installer le projet en local

Exécuter les tests d'intégration

Générer la documentation du projet

Générer le jar, le war, ... avec le code compilé

Exécuter les tests

Générer le jar, avec le code source

Compiler les tests

Compiler les sources

Gérer les dépendances du projet

TODO!

# Fichier Jar



<https://docs.oracle.com/en/java/javase/11/docs/specs/jar/jar.html>

# Module JAVA 9

## Descripteur de module

module-info.java  
module-info.class

- Nom
- Les dépendances à d'autres modules
- Les package accessibles aux autres modules
- Les services offerts
- Les services consommés
- Le packages accessibles par réflexion

<https://www.oracle.com/corporate/features/understanding-java-9-modules.html>

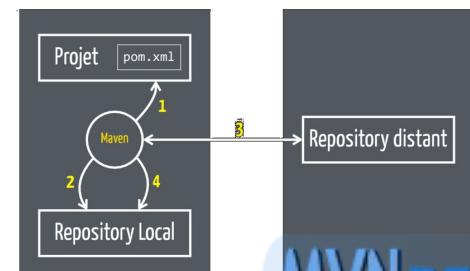


convention  
configuration

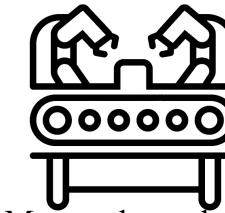
POM

# Maven™

Apache Maven Project  
<http://maven.apache.org/>



MVNREPOSITORY



# Installation de Maven



Télécharger le fichier compresser de la dernière version de Maven



Décompresser le fichier



Ajouter le sous dossier bin à la variable PATH de windows



Créer une variable d'environnement JAVA\_HOME pointant vers le dossier racine de votre JDK



Tester votre instalation en console grâce à la commande :  
`mvn -v`



<https://maven.apache.org/download.cgi>

# mvn Syntaxe

```
mvn [options] [goal(s)] [phase(s)]
```

**options**

-v : affiche la version de Maven

-h : affiche une description de toutes les options

**goal(s)**

mvn groupId:artifactId[:version]:goal (version est optionnel)

mvn PluginPrefix:goal



<https://maven.apache.org/guides/introduction/introduction-to-plugin-prefix-mapping.html>

**phase(s)**

validate, initialize, generate-sources, process-sources, generate-resources, process-resources, compile, process-classes, generate-test-sources, process-test-sources, generate-test-resources, process-test-resources, test-compile, process-test-classes, test, prepare-package, package, pre-integration-test, integration-test, post-integration-test, verify, install, deploy, pre-clean, clean, post-clean, pre-site, site, post-site, site-deploy.

# Cycle de vie

Maven possède par défaut 3 cycle de vie :

- default,
- clean,
- site .



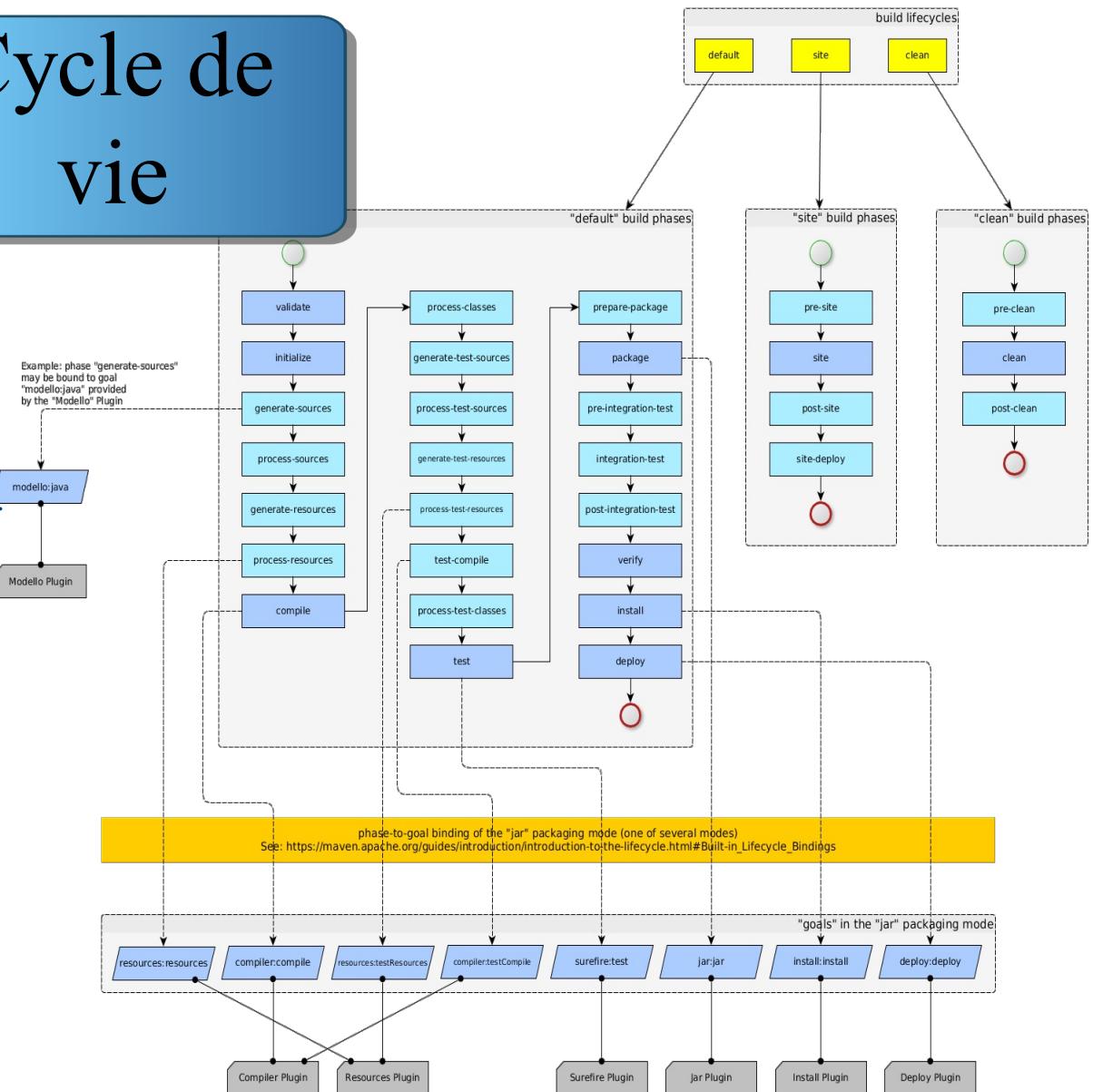
Chaque cycle de vie est composé d'une suite de phases :  
28 phases - default 21, clean 3 et site 4.



Quand une phase est invoquée avec la commande mvn, toutes les phases précédentes et celle demandée sont invoquée séquentiellement.



Chaque exécute les tâches des plugins auxquels elles sont reliées.





## Clone des sources

git clone https://github.com/YannickBoogaerts/myFrameWork2021.git



### Création d'un premier pom.xml

groupId : be.technifutur  
artifactId : myFrameWork  
version : 0.0.1-SNAPSHOT



Modifier la strucure des répertoires pour que Maven retrouve les fichier sources.

indice : Pom effectif



Modifier le pom.xml pour spécifier :

- que les sources sont codée en UTF-8
- que le code source est écrit en java 11
- que le bytecode est écrit en Java 11



Ajouter des dépendances dans le fichier pom.xml

- org.slf4j:slf4j-api:1.8.0-beta4

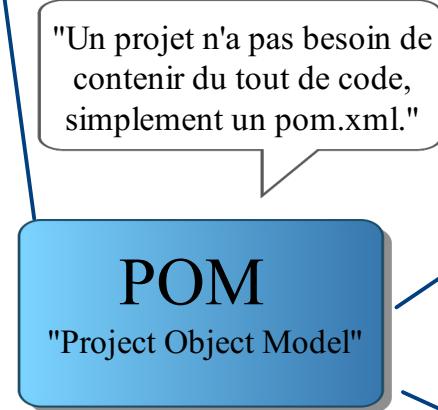
# Exercice fil rouge



Objectif : exécuter  
**mvn compile**  
sans erreur ni warning

## Représentation XML d'un projet Maven

- Les fichiers de configuration,
- Les développeurs impliqués et les rôles qu'ils jouent,
- Le système de suivi par défauts,
- L'organisation et les licences,
- L'URL de l'emplacement du projet,
- Les dépendances du projet
- et tous les autres petits éléments.



### Fichier pom.xml minimal

#### Informations de base

```
<project>
  <modelVersion>
  <groupId>
  <artifactId>
  <version>
```

- Identification du projet
- Spécification des dépendances requises
- Spécification des éléments de build du projet
  - localisation des ressources
  - configuration des plugins
  - la localisation des sources
  - les extensions utiles au build du projet
- Spécification des éléments de reporting du projet

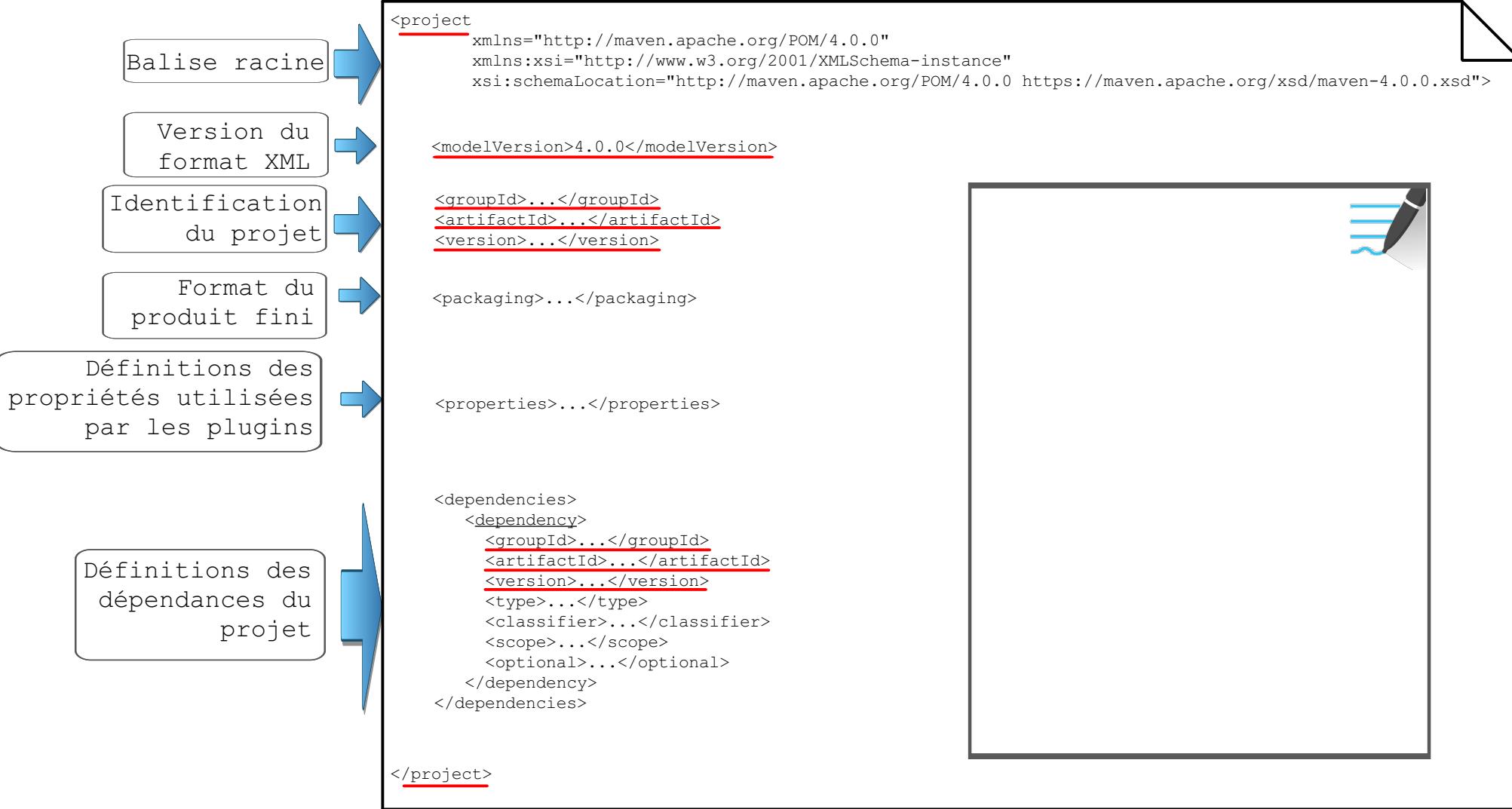
### Pom effectif

```
mvn help:effective-pom
[-Doutput=pom.txt]
[-Dverbose=true]
```

#### Fusion entre :

- super-pom
- livecycle-binding
- pom.xml

## Fichier pom.xml (basique)





## Maven coordinates

<https://maven.apache.org/pom.html#maven-coordinates>

### Gestion sémantique de version

<https://semver.org/lang/fr/>

### The SNAPSHOT Qualifier

[https://docs.oracle.com/middleware/1212/core/MAVEN/maven\\_version.htm#MAVEN401](https://docs.oracle.com/middleware/1212/core/MAVEN/maven_version.htm#MAVEN401)

[https://maven.apache.org/guides/getting-started/index.html#What\\_is\\_a\\_SNAPSHOT\\_version](https://maven.apache.org/guides/getting-started/index.html#What_is_a_SNAPSHOT_version)



## Properties

<https://maven.apache.org/pom.html#properties>

### Setting the -source and -target of the Java Compiler

<https://maven.apache.org/plugins/maven-compiler-plugin/examples/set-compiler-source-and-target.html>

### Specifying a character encoding scheme

<https://maven.apache.org/plugins/maven-resources-plugin/examples/encoding.html>



## Dependencies

<https://maven.apache.org/pom.html#dependencies>

# Exécution d'un plugin Maven

Un plugin est capables d'exécuter de 1 à n tâches les Goals

Les goals peuvent attendre des paramètres

<https://maven.apache.org/guides/mini/guide-configuring-plugins.html>

Syntaxe d'appel d'un goal

mvn groupId:artifactId[:version]:goal (version est optionnel)

mvn PluginPrefix:goal (si le groupId est repris dans la balise <pluginGroups> du fichier settings.xml)

•Préfix automatique si le artefacId suit le pattern :

- \${prefix}-maven-plugin ou
- maven-\${prefix}-plugin

•Préfix défini par la balise plugin/configuration/goalPrefix dans Pom.xml

<https://maven.apache.org/guides/introduction/introduction-to-plugin-prefix-mapping.html>

Plugins utilisables

<https://maven.apache.org/plugins/index.html>

<https://www.mojohaus.org/plugins.html>

# Exercice fil rouge



Ajouter des dépendances dans le fichier pom.xml

- org.junit.jupiter:junit-jupiter-api:5.3.1
- org.junit.jupiter:junit-jupiter-engine:5.3.1
- org.easymock:easymock:4.0.2



Visualiser le graphe des dépendances grâce

plugin : org.apache.maven.plugins:maven-dependency-plugin  
goal : tree

*Tester les 2 syntaxes : générale et préfixe*



Modifier la version du plugin maven-surefire-plugin.

utiliser la dernière version



Ajouter la dépendance au gestionnaire de log log4j.

- org.apache.logging.log4j:log4j-slf4j18-impl:2.17.1



Modifier les versions des plugins utilisés :

- maven-dependency-plugin
- maven-compiler-plugin
- maven-resources-plugin



Objectif : exécuter  
**mvn test**  
sans erreur ni warning

# 3 types de repository Maven

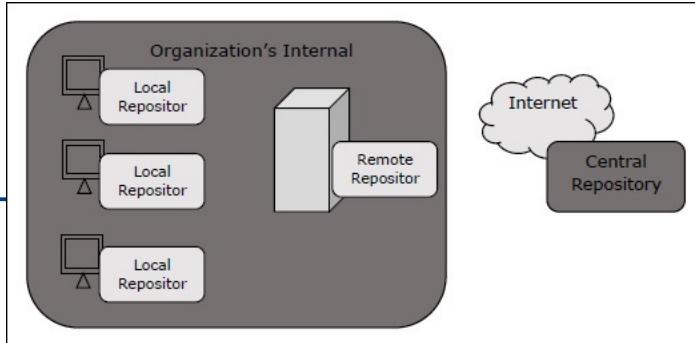
## Repository Locale

Créer automatiquement par maven sur votre machine

Contient toutes les dépendances dont Maven à besoin.

Il est localisé :

par défaut : \${user.home}/.m2/repository  
dans le fichier : settings.xml



Lors de la création d'un build, Maven recherche et charge les librairies dont il a besoin.

- 1 dans le repository local
- 2 dans le repository central et met à jour le locale.
- 3 Dans les repository distant configuré.

## Repository central

Maven central repository est fourni par la communauté Maven.

Concepts clés :

- This repository is managed by Maven community.
- It is not required to be configured.
- It requires internet access to be searched.

Contient énormément de librairies utilisées régulièrement

URL : <https://repo1.maven.org/maven2/>



## Introduction aux repositories

<https://maven.apache.org/guides/introduction/introduction-to-repositories.html>

# Exercice fil rouge

Installer la librairie dans le repository local  
- mvn install .

Localiser sur le disque dur le jar de la librairie installée  
Voir configuration dans settings.xml

Créer un nouveau projet avec son Pom.xml

- identification du projet
- configuration des propriétés
- ajout de la dépendance au projet myFrameWork



Objectif : installer et utiliser la librairie MyframeWork



Visualiser le graphe des dépendances grâce  
plugin : dependency  
goal : tree