

Maven par la pratique

Installation et Configuration de Maven

Téléchargez Apache Maven depuis le site officiel : <https://maven.apache.org/download.cgi>

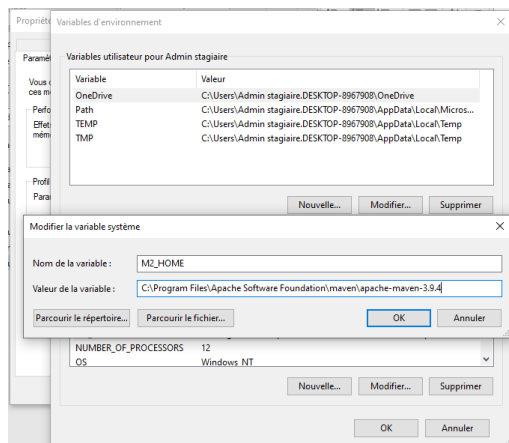
Prenez la version :

- **Binary tar.gz archive** si vous êtes sous Linux ou Mac OS
- **Binary zip archive** si vous êtes sous Windows

Décompresser l'archive dans un répertoire du système

Définir les variables d'environnement :

- Si cela n'a pas été déjà fait : définir le chemin vers le JDK grâce à la variable d'environnement `JAVA_HOME` et ajouter les binaires du JDK et de Maven au `PATH` :
 - `JAVA_HOME` : `C:\Program Files\Java\jdk-17.0.5`
 - `PATH` = `%JAVA_HOME%\bin`
- Créer la variable d'environnement `M2_HOME` qui pointe sur le répertoire contenant Maven :
`M2_HOME` = répertoire de maven
- Ajouter le chemin `M2_HOME/bin` à la variable `PATH` du système
 - `PATH` : `%M2_HOME%\bin`



Le répertoire de Maven contient plusieurs sous-répertoires :

- **bin** : Ce répertoire contient les scripts et exécutables nécessaires pour lancer Maven. Le fichier le plus important est **mvn**, qui est le script principal utilisé pour exécuter des commandes Maven en ligne de commande. En utilisant ce script, vous pouvez lancer Maven depuis n'importe quel répertoire de projet.
- **conf** : Ce répertoire contient la configuration par défaut de Maven, y compris le fichier **settings.xml**. Le fichier settings.xml permet de configurer divers paramètres de Maven, tels que les *référentiels (repositories)*, les *profils (profiles)*, et d'autres configurations globales.
- **lib** : Ce répertoire contient les bibliothèques (JAR) qui constituent le cœur de Maven. Il s'agit des bibliothèques Java nécessaires à l'exécution de Maven. Maven utilise ces bibliothèques pour gérer les dépendances, compiler des projets, exécuter des plugins, etc. Vous ne devriez généralement pas avoir besoin de modifier ce répertoire, car les bibliothèques sont gérées automatiquement par Maven.

Tester l'installation :

- Sous Windows : ouvrez une nouvelle console (Win + R, puis taper cmd, puis Entrée)
- Exécutez la commande suivante : `mvn -v`

```
Apache Maven 3.9.4 (dfbb324ad4a7c8fb0bf182e6d91b0ae20e3d2dd9)
Maven home: C:\Program Files\Apache Software Foundation\maven\apache-maven-3.9.4
Java version: 17.0.5, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-17.0.5
Default locale: fr_FR, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

Configurez Apache Maven

La configuration de Maven se fait dans le fichier `.m2/settings.xml`. C'est un fichier de configuration global qui permet de définir des paramètres et des configurations par défaut pour tous les projets Maven. Ce fichier n'existe pas par défaut, copiez celui contenu dans le répertoire `conf` de l'installation de Maven

Sous Windows, par défaut, il est situé dans le répertoire `.m2/repository`.

Vous trouverez plus de détails sur la configuration de Maven dans la documentation officielle :

- Configuration : <https://maven.apache.org/configure.html>
- Fichier settings.xml : <https://maven.apache.org/settings.html>

Créer un projet Maven avec Eclipse

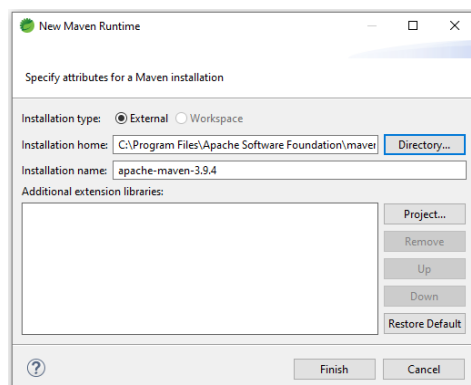
Configuration

Ouvrez les préférences (*Window > Preferences*) :

- **Maven**
 - Installations :

Maven est intégré à Eclipse (version EMBEDDED), ce qui évite d'installer Maven
On peut installer notre propre version depuis

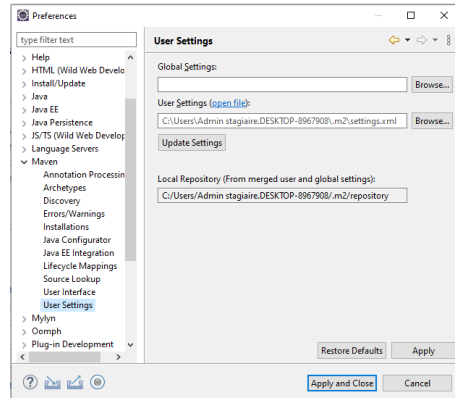
1. Ajoutez une nouvelle installation externe :



2. Cochez la nouvelle installation que vous venez de créer pour qu'elle devienne celle par défaut.

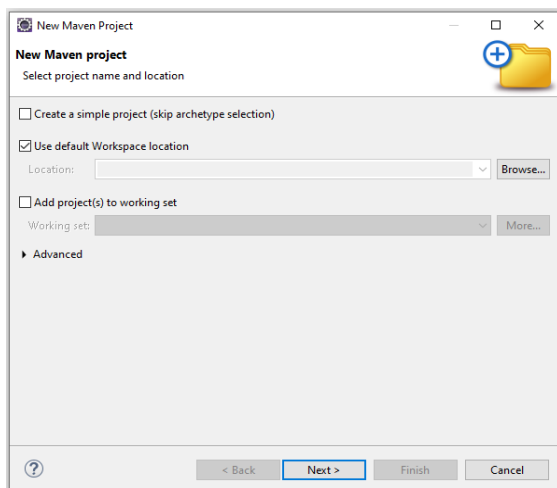
○ User Settings :

- User Settings : C:/Users/Admin stagiaire.DESKTOP-8967908/.m2/repository



Générer un projet Maven avec Eclipse

Choisissez File -> New -> Project..., puis choisissez Maven -> Maven Project



1. Cliquez sur l'option « Create a simple project (skip **archetype** selection) », puis Next.

- Si vous cochez cette option, Eclipse va créer un projet Maven sans vous demander de choisir un archetype (un modèle de projet Maven préconfiguré). Le projet sera généré avec une structure minimale et sans exemple de code.
- Cette option est utile si vous savez exactement ce que vous voulez faire et que vous préférez configurer manuellement votre projet Maven avec les dépendances et la structure de répertoire appropriées.

GAV (Maven théorie : Artefacts)

Packaging : Le packaging spécifie le type de conditionnement utilisé pour l'artefact généré lors de la construction du projet.

Il peut prendre différentes valeurs :

- **jar** (pour les bibliothèques Java),
- **war** (pour les applications web),
- **"pom"** (pour les projets de type pom)

Le packaging détermine le type de fichier généré dans le répertoire "target" lors de la construction du projet.

Name : Contient le nom complet du projet

Il peut être utile pour identifier rapidement le projet dans un environnement de développement ou de gestion de projet.

Description : La description est un texte court qui fournit une description brève mais informative de ce que fait votre projet.

2. Ne pas cocher l'option « Create a simple project (skip archetype selection) », puis Next.

• Les archétypes

Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-plugin-site	1.1
org.apache.maven.archetypes	maven-archetype-portlet	1.0.1
org.apache.maven.archetypes	maven-archetype-profiles	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.1
org.apache.maven.archetypes	maven-archetype-site	1.1
org.apache.maven.archetypes	maven-archetype-site-simple	1.1
org.apache.maven.archetypes	maven-archetype-webapp	1.0

Les modèles (archetype) sont récupérés localement ou à distance depuis un catalogue :

• Window > Preferences

○ Maven > Archetypes

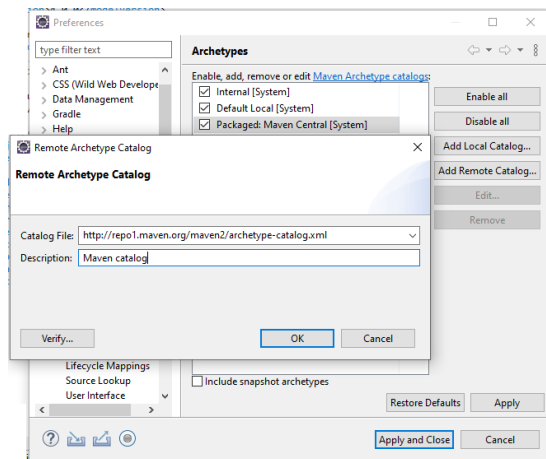
■ Add Remote Catalog

Catalog file : <http://repo1.maven.org/maven2/archetype-catalog.xml>

Description : Maven catalog

Voir : <https://maven.apache.org/archetype/maven-archetype-plugin/specification/archetype-catalog.html>

Le fichier **archetype-catalog.xml** répertorie les archétypes disponibles dans le référentiel Maven central (<https://repo1.maven.org/maven2/>)

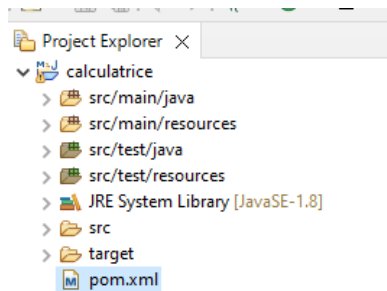


Structure d'un projet maven

Maven utilise une approche dite « convention over configuration » (ou **convention** plutôt que configuration en français). Cela signifie que Maven a établi un certain nombre de conventions et que si vous les respectez, beaucoup de choses seront automatiques.

Une des premières conventions concerne l'arborescence d'un projet Maven.

<https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html>



Répertoire	Contenu
/src/main/java	Le code source java de l'application (sera compilé dans /target/classes)
/src/main/resources	Les fichiers de ressources (fichiers de configuration, images, ...). Le contenu de ce répertoire est copié dans target/classes pour être inclus dans l'artéfact généré
/src/main/webapp	Les fichiers de l'application web
/src/test/java	Le code source des tests Junit (sera compilé dans /target/test-classes)
/src/test/resources	Les fichiers de ressources pour les tests
/target	Les fichiers générés pour les artéfacts et les tests (ce répertoire ne doit pas être inclus dans le gestionnaire de sources)
/target/classes	Les classes compilées
/target/test-classes	Les classes compilées des tests unitaires
/target/site	Site web contenant les rapports générés et des informations sur le projet
/pom.xml	Le fichier POM de description du projet

• Pom.xml

Le fichier POM (Project Object Model) contient la description du projet Maven. C'est un fichier XML nommé pom.xml ; Il pilote la configuration de Maven. Il contient les informations nécessaires à la génération du projet : identification de l'artéfact, déclaration des dépendances, ses plugins, définition d'informations relatives au projet, ...

- Le fichier POM doit être à la racine du répertoire du projet.
- Le tag racine est le tag <project>

```

1<project xmlns="http://maven.apache.org/POM/4.0.0"
2  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4
5<!-- La version du format du fichier pom. Actuellement la dernière version
6  est la 4.0.0. -->
7  <modelVersion>4.0.0</modelVersion>
8
9  <!-- ===== Informations du projet ===== -->
10 <!-- Informations du projet -->
11 <!-- ===== Informations Maven ===== -->
12 <!-- Informations Maven -->
13 <groupId>fr.dawan</groupId>
14 <artifactId>calculator</artifactId>
15 <version>1.0-SNAPSHOT</version>
16
17 <!-- Type d'archive à générer -->
18 <packaging>jar</packaging>
19
20 <!-- ===== Informations générales ===== -->
21 <name>calculator</name>
22 <url>http://maven.apache.org</url>
23 <description>
24   Projet Calculator : permettant d'effectuer plusieurs opérations mathématiques
25 </description>
26
27 <organization>
28   <name>Dawan</name>
29   <url>https://www.dawan.fr</url>
30 </organization>
31
32 <!-- ===== Propriétés ===== -->
33 <!-- Propriétés -->
34 <!-- ===== -->
35 <properties>
36   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
37   <maven.compiler.source>17</maven.compiler.source>
38   <maven.compiler.target>17</maven.compiler.target>
39 </properties>
40
41 <dependencies>
42   <dependency>
43     <groupId>junit</groupId>
44     <artifactId>junit</artifactId>
45     <version>3.8.1</version>
46     <scope>test</scope>
47   </dependency>
48 </dependencies>
49 </project>

```

Tag	Rôle
<modelVersion>	Préciser la version du modèle de POM utilisée
<groupId>	Préciser le groupe ou l'organisation qui développe le projet. C'est une des clés utilisées pour identifier de manière unique le projet et ainsi éviter les conflits de noms
<artifactId>	Préciser la base du nom de l'artéfact du projet
<packaging>	Préciser le type d'artéfact généré par le projet (jar, war, ear, pom, ...). Le packaging définit aussi les différents goals qui seront exécutés durant le cycle de vie par défaut du projet. La valeur par défaut est jar
<version>	Préciser la version de l'artéfact généré par le projet. Le suffixe -SNAPSHOT indique une version en cours de développement
<name>	Préciser le nom du projet utilisé pour l'affichage
<description>	Préciser une description du projet
<url>	Préciser une url qui permet d'obtenir des informations sur le projet
<licenses>	Permet de spécifier les informations sur la licence associée au projet, tels que le nom de la licence, l'URL de la licence, et le type de distribution.
<properties>	Permet de définir des propriétés personnalisées utilisées dans le fichier POM. Ces propriétés peuvent être référencées ailleurs dans le POM
<dependencies>	Définir l'ensemble des dépendances du projet
<dependency>	Déclarer une dépendance en utilisant plusieurs tags fils : <groupId>, <artifactId>, <version> et <scope>

Créer un projet Maven en ligne de commande

Générer un squelette de projet Maven

Afin de générer le squelette d'un projet, Maven s'appuie sur des archétypes (ce sont des sortes de modèles). Ici, on va tout simplement demander à Maven de me générer un squelette à partir de l'archétype quickstart

<https://maven.apache.org/archetypes/maven-archetype-quickstart/>

Voici comment générer le squelette en mode console :

1. Ouvrez un terminal (ou une console) et placez-vous dans le répertoire où vous voulez créer le projet (Maven y créera un sous-répertoire pour votre nouveau projet).

```
cd C:\Users\Admin stagiaire.DESKTOP-8967908\Desktop\MiseEnPratique\FormationMaven\Me\calculator_maven
```

2. Lancez la génération à partir de l'archétype :

- Lancer la commande :

```
mvn archetype:generate -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4
```

3. Maven va vous poser des questions afin de personnaliser la génération de votre projet :

```
groupId : fr.dawan
artifactId : calculator
version (1.0-SNAPSHOT) : laissez vide
package (fr.dawan.calculator) : laissez vide
```

4. Ensuite Maven vous demande de confirmer les paramètres, il vous suffit donc de presser la touche **Entrée**.
5. Maven crée le squelette du projet : vous devriez voir un résultat comme celui-ci :

```
Administrateur : Invite de commandes
Define value for property 'version' 1.0-SNAPSHOT: :
Define value for property 'package' fr.dawan: : fr.dawan.calculator_cmd
Confirm properties configuration:
groupId: fr.dawan
artifactId: calculator_cmd
version: 1.0-SNAPSHOT
package: fr.dawan.calculator_cmd
Y: : y
[INFO] -----
[INFO] Using following parameters for creating project from Archetype: maven-archetype-quickstart:1.4
[INFO] -----
[INFO] Parameter: groupId, Value: fr.dawan
[INFO] Parameter: artifactId, Value: calculator_cmd
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: package, Value: fr.dawan.calculator_cmd
[INFO] Parameter: packageInPathFormat, Value: fr/dawan/calculator_cmd
[INFO] Parameter: package, Value: fr.dawan.calculator_cmd
[INFO] Parameter: groupId, Value: fr.dawan
[INFO] Parameter: artifactId, Value: calculator_cmd
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Project created from Archetype in dir: C:\Users\Admin stagiaire.DESKTOP-8967908\Desktop\MiseEnPratique\FormationMaven\Me\calculator_maven\calculator_cmd
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 01:09 min
[INFO] Finished at: 2023-10-04T12:17:05+02:00
[INFO] -----
C:\Users\Admin stagiaire.DESKTOP-8967908\Desktop\MiseEnPratique\FormationMaven\Me\calculator_maven>
```

- Si on veut utiliser un archetype, par exemple « maven-archetype-quickstart »
 - <https://maven.apache.org/archetypes/maven-archetype-quickstart/>
- Les informations à fournir :

```
groupId : fr.dawan
artifactId : calculatrice_lignecmd
version : par défaut (1.0-SNAPSHOT)
package : fr.dawan.calculatrice_lignecmd
Après confirmer les propriétés : Y
```

```
Invite de commandes
Define value for property 'version' 1.0-SNAPSHOT: :
Define value for property 'package' fr.dawan: : fr.dawan.calculatrice
Confirm properties configuration:
groupId: fr.dawan
artifactId: calculatrice_lignecmd
version: 1.0-SNAPSHOT
package: fr.dawan.calculatrice
Y: : Y
[INFO] -----
[INFO] Using following parameters for creating project from Archetype: maven-archetype-quickstart:1.4
[INFO] -----
[INFO] Parameter: groupId, Value: fr.dawan
[INFO] Parameter: artifactId, Value: calculatrice_lignecmd
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: package, Value: fr.dawan.calculatrice
[INFO] Parameter: packageInPathFormat, Value: fr/dawan/calculatrice
[INFO] Parameter: package, Value: fr.dawan.calculatrice
[INFO] Parameter: groupId, Value: fr.dawan
[INFO] Parameter: artifactId, Value: calculatrice_lignecmd
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Project created from Archetype in dir: C:\Users\Admin stagiaire.DESKTOP-8967908\Desktop\MiseEnPratique\FormationMaven\Me\calculatrice_lignecmd
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 02:04 min
[INFO] Finished at: 2023-09-26T19:41:49+02:00
[INFO] -----
C:\Users\Admin stagiaire.DESKTOP-8967908\Desktop\MiseEnPratique\FormationMaven\Me>
```

6. Importer le projet dans Eclipse

- Pour éditer le projet généré avec Eclipse, nous avons besoin de demander à Maven de générer les fichiers **.project** et **.classpath**, nécessaires à un projet Eclipse
- Nous utilisons pour cela le **plugin eclipse**
- Exécuter la commande suivante : `mvn eclipse:eclipse`

D'autres liens utiles :

- <https://maven.apache.org/archetypes/>
- <https://maven.apache.org/archetype/maven-archetype-plugin/generate-mojo.html>

Création d'une application Calculator

Une classe **Calculator** qui contient des méthodes :

- **sum** qui permet de retourner la somme de deux nombres
- **multiply** qui permet de retourner le produit de deux nombres
- **subtract** qui permet de retourner la différence de deux nombres
- **divide** qui permet de retourner le quotient de deux nombres

Un Test unitaire qui permet de tester les méthodes

Nous demanderons ensuite à maven de :

- Compiler toutes les classes
- Exécuter tous les tests unitaires
- Installer le jar du projet dans le repository local de maven

Le cycle de vie du build lifecycle

De quoi est composé le build lifecycle ?

Afin d'automatiser la construction d'un projet, *Maven* s'appuie sur des cycles de vie de construction appelés **build lifecycle** dans le jargon de *Maven*.

<https://maven.apache.org/ref/3.9.4/maven-core/lifecycles.html>

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 on prend l'exemple du **build lifecycle default**, nous y retrouvons, entre autres, les phases :

- **validate** : vérifie que la configuration du 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.

Lancer un build lifecycle

Quand vous lancez une construction *Maven* en ligne de commande, vous précisez simplement une *phase* d'un des **build lifecycles** et *Maven* se charge d'exécuter, dans l'ordre, toutes les phases qui composent le build lifecycle jusqu'à la phase indiquée.

- **mvn package** : Cette commande, indique la phase package du *build lifecycle default*. Maven exécute dans l'ordre les phases *validate*, *compile*, *test* et enfin *package*.
- **mvn deploy** : Cette commande, indique la phase deploy du build lifecycle default. Maven exécute dans l'ordre les phases *validate*, *compile*, *test*, *package* et enfin *deploy*.

Pour les **projets multi-modules**, quand vous lancez la commande mvn sur le projet parent, Maven lance le build lifecycle dans chaque sous-projet (module), les uns à la suite des autres, en respectant l'ordre des dépendances inter-modules.

Il vous est aussi possible de chaîner l'exécution de plusieurs *build lifecycles* dans une seule commande Maven. Si vous lancez mvn clean package par exemple, *Maven* va :

1. Dans un premier temps, nettoyer le projet en exécutant le build lifecycle clean ;
2. Puis, il va lancer le build lifecycle default jusqu'à la phase package.

Les goals

Nous avons vu qu'un *build lifecycle* est constitué d'une série de *phases*. Mais la granularité de l'exécution de *Maven* est encore plus fine. **En effet, les phases sont découpées en tâches. Chaque tâche est assurée par un plugin Maven.** Dans le jargon de *Maven*, ces tâches s'appellent des **goals**.

Suivant le build lifecycle et le packaging utilisé, différents goals sont câblés par défaut aux différentes phases : <https://maven.apache.org/ref/3.9.4/maven-core/default-bindings.html>

Lien Plugins : <https://maven.apache.org/plugins/>
<https://maven.apache.org/plugins/maven-compiler-plugin/>

```

1 <phases>
2 <process-resources>
3   org.apache.maven.plugins:maven-resources-plugin:2.6:resources
4 </process-resources>
5 <compile>
6   org.apache.maven.plugins:maven-compiler-plugin:3.1:compile
7 </compile>
8 <process-test-resources>
9   org.apache.maven.plugins:maven-resources-plugin:2.6:testResources
10 </process-test-resources>
11 <test-compile>
12   org.apache.maven.plugins:maven-compiler-plugin:3.1:testCompile
13 </test-compile>
14 <test>
15   org.apache.maven.plugins:maven-surefire-plugin:2.12.4:test
16 </test>
17 <package>
18   org.apache.maven.plugins:maven-jar-plugin:2.4:jar
19 </package>
20 <install>
21   org.apache.maven.plugins:maven-install-plugin:2.4:install
22 </install>
23 <deploy>
24   org.apache.maven.plugins:maven-deploy-plugin:2.7:deploy
25 </deploy>
26 </phases>

```

Par exemple, la phase *test* est réalisée par défaut par le goal ***surefire:test***, c'est-à-dire le goal *test* du plugin *surefire*.

Compilation

mvn compile

```

C:\Users\Admin stagiaire.DESKTOP-8967908\Desktop\MiseEnPratique\FormationMaven\Me\calculator_maven\calculator>mvn compile
[INFO] Scanning for projects...
[INFO]
[INFO] -----< fr.dawan:calculator >-----
[INFO] Building calculator 1.0-SNAPSHOT
[INFO] from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ calculator ---
[INFO] skip non existing resourceDirectory C:\Users\Admin stagiaire.DESKTOP-8967908\Desktop\MiseEnPratique\FormationMaven\Me\calculator_maven\calculator\src\main\resources
[INFO]
[INFO] --- compiler:3.11.0:compile (default-compile) @ calculator ---
[INFO] Changes detected - recompiling the module! :input tree
[INFO] Compiling 1 source file with javac [debug target 1.8] to target\classes
[WARNING] bootstrap class path not set in conjunction with -source 8
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 0.784 s
[INFO] Finished at: 2023-10-04T14:58:25+02:00
[INFO]

```

- Associé au plugin ***maven-compiler-plugin***.

Après exécution de la commande « mvn compile » :

1. Copie les ressources de l'application
2. Compile les fichiers sources .java, et copie les .class dans target/classes

```

project/
├─ src/
│  ├─ main/
│  │  └─ java/
│  │     └─ (fichiers source Java)
│  │  └─ resources/
│  │     └─ (autres ressources)
│  └─ test/
│     └─ java/
│        └─ (fichiers source de tests Java)
│     └─ resources/
│        └─ (ressources de test)
├─ target/
│  └─ classes/
│     └─ (fichiers .class résultants)
│  └─ (autres dossiers et fichiers générés)
└─ (autres dossiers et fichiers du projet)
└─ pom.xml

```

Test unitaire

mvn test

Tests unitaires : classes permettant de tester le bon fonctionnement de notre code.

Intérêt : éviter des régressions de code

Pour exécuter des tests unitaires dans un projet Maven, vous devez suivre quelques étapes spécifiques :

- **Écrire des tests unitaires** : Créez des classes de test pour vos composants ou classes que vous souhaitez tester. Les classes de test doivent être placées dans le package `src/test/java`.
- **Utiliser un framework de test** : Choisissez un framework de test unitaire tel que JUnit ou TestNG. Vous pouvez ajouter ces frameworks en tant que dépendances dans la section `<dependencies>` de votre fichier POM.

```

<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter</artifactId>
  <version>5.6.0</version>
</dependency>

```

- Créer la classe de test : `CalculatorTest`
- Exécuter les tests : `mvn test`
 - `mvn test -Dtest=NomDeLaClasseDeTest`

Le goal **mvn test** est associé au plugin **maven-surefire-plugin**.

Après exécution de la commande « mvn test » :

1. Copie les ressources de l'application
2. Compile les fichiers sources `.java`, et copie les `.class` dans `target/classes`
3. Lance des tests unitaires : les fichiers `.class` des fichiers test (contenus dans le dossier `src/test/java`) sont placés dans le répertoire **target/test-classes**. Ces fichiers `.class` contiennent les classes de test compilées et sont utilisés pour exécuter les tests unitaires

```
C:\Users\Admin stagiaire.DESKTOP-8967908\Desktop\MiseEnPratique\FormationMaven\Me\calculator_maven\calculator>mvn test
[INFO] Scanning for projects...
[INFO]
[INFO] -----< fr.dawan:calculator >-----
[INFO] Building calculator 1.0-SNAPSHOT
[INFO] from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ calculator ---
[INFO] skip non existing resourceDirectory C:\Users\Admin stagiaire.DESKTOP-8967908\Desktop\MiseEnPratique\FormationMaven\Me\calculator_maven\calculator\src\main\resources
[INFO]
[INFO] --- compiler:3.11.0:compile (default-compile) @ calculator ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ calculator ---
[INFO] skip non existing resourceDirectory C:\Users\Admin stagiaire.DESKTOP-8967908\Desktop\MiseEnPratique\FormationMaven\Me\calculator_maven\calculator\src\test\resources
[INFO]
[INFO] --- compiler:3.11.0:testCompile (default-testCompile) @ calculator ---
[INFO] Changes detected - recompiling the module! :source
[INFO] Compiling 2 source files with javac [debug target 17] to target\test-classes
[INFO]
[INFO] --- surefire:3.1.2:test (default-test) @ calculator ---
[INFO] Using auto detected provider org.apache.maven.surefire.junitplatform.JUnitPlatformProvider
[INFO]
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running fr.dawan.calculator.CalculatorTest
[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.041 s -- in fr.dawan.calculator.CalculatorTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.009 s
[INFO] Finished at: 2023-10-04T15:39:41+02:00
[INFO] -----
```

```
project/
├─ src/
│  ├─ main/
│  │  ├─ java/
│  │  │  └─ (fichiers source Java)
│  │  └─ resources/
│  │     └─ (autres ressources)
│  └─ test/
│     ├─ java/
│     │  └─ (fichiers source de tests Java)
│     └─ resources/
│        └─ (ressources de test)
├─ target/
│  ├─ classes/
│  │  └─ (fichiers .class résultants du code source principal)
│  └─ test-classes/
│     └─ (fichiers .class résultants du code source de test)
└─ (autres dossiers et fichiers générés)
└─ pom.xml
```

Construction du projet (archive)

mvn package

La construction d'un projet (archive) dans le contexte de Maven consiste à générer un artefact, généralement sous la forme d'un fichier JAR (Java Archive) ou WAR (Web Application Archive) pour les projets web.

Le goal **mvn package** est associé au plugin **maven-jar-plugin**

Après exécution de la commande mvn package :

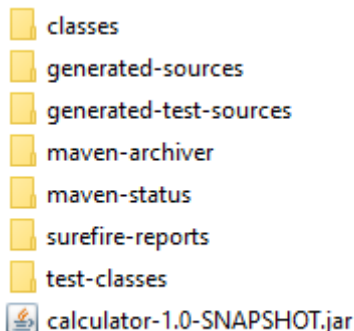
1. Copie les ressources de l'application
2. Compile les fichiers sources *.java*, et copie les *.class* dans *target/classes*
3. Lance des tests unitaires : les fichiers *.class* résultants des fichiers source de test sont placés dans le répertoire *target/test-classes*. Ces fichiers *.class* contiennent les classes de test compilées et sont utilisés pour exécuter les tests unitaires

4. Génère généralement le répertoire target à la racine de votre projet. Ce répertoire target contient les fichiers générés lors du processus de

```
C:\Users\Admin stagiaire.DESKTOP-8967908\Desktop\WiseEnPratique\FormationMaven\Me\calculator_maven\calculator>mvn package
[INFO] Scanning for projects...
[INFO] -----< fr.dawan:calculator >-----
[INFO] Building calculator 1.0-SNAPSHOT
[INFO] from pom.xml
[INFO] -----[ jar ]-----
[INFO] --- resources:3.3.1:resources (default-resources) @ calculator ---
[INFO] skip non existing resourceDirectory C:\Users\Admin stagiaire.DESKTOP-8967908\Desktop\WiseEnPratique\FormationMaven\Me\calculator_maven\calculator\src\main\resources
[INFO] --- compiler:3.11.0:compile (default-compile) @ calculator ---
[INFO] Nothing to compile - all classes are up to date
[INFO] --- resources:3.3.1:testResources (default-testResources) @ calculator ---
[INFO] skip non existing resourceDirectory C:\Users\Admin stagiaire.DESKTOP-8967908\Desktop\WiseEnPratique\FormationMaven\Me\calculator_maven\calculator\src\test\resources
[INFO] --- compiler:3.11.0:testCompile (default-testCompile) @ calculator ---
[INFO] Changes detected - recompiling the module! :source
[INFO] Compiling 2 source files with javac [debug target 17] to target\test-classes
[INFO] --- surefire:3.1.2:test (default-test) @ calculator ---
[INFO] Using auto detected provider org.apache.maven.surefire.junitplatform.JUnitPlatformProvider
[INFO]
[INFO] T E S T S
[INFO]
[INFO] Running fr.dawan.calculator.CalculatorTest
[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.041 s -- in fr.dawan.calculator.CalculatorTest
[INFO] Results:
[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] --- jar:3.3.0:jar (default-jar) @ calculator ---
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 2.066 s
[INFO] Finished at: 2023-10-04T16:10:33+02:00
[INFO]
```

Le **répertoire target** contient :

- Le fichier JAR : target/calculator-1.0-SNAPSHOT.jar
- **Répertoire classes** : Ce répertoire contient les fichiers .class résultants de la compilation de vos fichiers source Java.
- **Répertoire test-classes** : Ce répertoire contient les fichiers .class résultants de la compilation de vos fichiers source de test unitaire



Vous pouvez lancer votre application Java :

```
java -cp target/calculator-1.0-SNAPSHOT.jar fr.dawan.calculator.App
```

cp : (classpath) pour spécifie le chemin vers le JAR et les classes de test
java

jar :

Un fichier JAR (Java Archive) est un format de fichier qui permet de regrouper plusieurs fichiers Java (classes, métadonnées, ressources) en une seule archive.

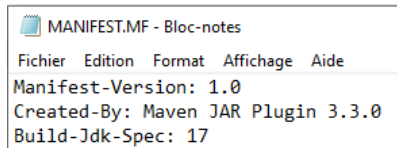
Il a été configuré pour être exécutable indépendamment, sans avoir besoin d'autres dépendances externes.

Les fichiers JAR sont utilisés pour distribuer et déployer des applications Java autonomes.

Manifest avec Classe Principale :

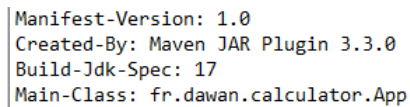
Un **JAR autonome** doit inclure un fichier **META-INF/MANIFEST.MF** qui spécifie la classe principale à exécuter lorsqu'on lance le JAR. Cette information est généralement définie dans l'attribut **Main-Class** du Manifest.

- **Contenu du Manifest sans Main-Class :**



```
MANIFEST.MF - Bloc-notes
Fichier  Edition  Format  Affichage  Aide
Manifest-Version: 1.0
Created-By: Maven JAR Plugin 3.3.0
Build-Jdk-Spec: 17
```

- **Contenu du Manifest avec Main-Class :**



```
Manifest-Version: 1.0
Created-By: Maven JAR Plugin 3.3.0
Build-Jdk-Spec: 17
Main-Class: fr.dawan.calculator.App
```

- Avec Maven pour créer un jar pour créer un fichier JAR exécutable, vous devez configurer le plugin Maven JAR pour inclure les informations du manifeste, notamment la classe principale (Main-Class)

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <version>3.3.0</version>
      <configuration>
        <archive>
          <manifest>
            <mainClass>fr.dawan.calculator.App</mainClass>
          </manifest>
        </archive>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Installation locale

mvn install

Le goal mvn install est associé au plugin **maven-install-plugin**

L'opération **mvn install** a pour but d'installer l'**artefact** localement dans le référentiel local de Maven (par exemple, ~/.m2/repository sous Windows) afin qu'il puisse être utilisé comme dépendance par d'autres projets Maven sur votre machine. Une fois installé localement, le projet peut être référencé comme une dépendance dans d'autres projets Maven sans avoir besoin d'être déployé vers un référentiel distant.

```
C:\Users\Admin stagiaire.DESKTOP-8967908\Desktop\MiseEnPratique\FormationMaven\Me\calculator_maven\calculator>mvn install
[INFO] Scanning for projects...
[INFO]
[INFO] -----< fr.dawan:calculator >-----
[INFO] Building calculator 1.0-SNAPSHOT
[INFO] from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ calculator ---
[INFO] skip non existing resourceDirectory C:\Users\Admin stagiaire.DESKTOP-8967908\Desktop\MiseEnPratique\FormationMaven\Me\calculator_maven\calculator\src\main\resources
[INFO]
[INFO] --- compiler:3.11.0:compile (default-compile) @ calculator ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ calculator ---
[INFO] skip non existing resourceDirectory C:\Users\Admin stagiaire.DESKTOP-8967908\Desktop\MiseEnPratique\FormationMaven\Me\calculator_maven\calculator\src\test\resources
[INFO]
[INFO] --- compiler:3.11.0:testCompile (default-testCompile) @ calculator ---
[INFO] Changes detected - recompiling the module! :source
[INFO] Compiling 2 source files with javac [debug target 17] to target\test-classes
[INFO]
[INFO] --- surefire:3.1.2:test (default-test) @ calculator ---
[INFO] Using auto detected provider org.apache.maven.surefire.junitplatform.JUnitPlatformProvider
[INFO]
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running fr.dawan.calculator.CalculatorTest
[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.042 s -- in fr.dawan.calculator.CalculatorTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- jar:3.3.0:jar (default-jar) @ calculator ---
[INFO] Building jar: C:\Users\Admin stagiaire.DESKTOP-8967908\Desktop\MiseEnPratique\FormationMaven\Me\calculator_maven\calculator\target\calculator-1.0-SNAPSHOT.jar
[INFO]
[INFO] --- install:3.1.1:install (default-install) @ calculator ---
[INFO] Installing C:\Users\Admin stagiaire.DESKTOP-8967908\Desktop\MiseEnPratique\FormationMaven\Me\calculator_maven\calculator\pom.xml to C:\Users\Admin stagiaire.DESKTOP-8967908\.m2\repository\fr\dawan\calculator\1.0-SNAPSHOT\calculator-1.0-SNAPSHOT.pom
[INFO] Installing C:\Users\Admin stagiaire.DESKTOP-8967908\Desktop\MiseEnPratique\FormationMaven\Me\calculator_maven\calculator\target\calculator-1.0-SNAPSHOT.jar to C:\Users\Admin stagiaire.DESKTOP-8967908\.m2\repository\fr\dawan\calculator\1.0-SNAPSHOT\calculator-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.137 s
[INFO] Finished at: 2023-10-04T16:39:04+02:00
[INFO] -----
```

	Organiser	Nouveau	Ouvrir
jmin stagiaire.DESKTOP-8967908 >	.m2 > repository > fr > dawan > calculatrice > 0.0.1-SNAPSHOT		
Nom	Modifié le	Type	Taille
_remote.repositories	27/09/2023 11:57	Fichier REPOSITOR...	1 Ko
calculatrice-0.0.1-SNAPSHOT.jar	27/09/2023 10:46	Executable Jar File	4 Ko
calculatrice-0.0.1-SNAPSHOT.pom	27/09/2023 10:12	Fichier POM	3 Ko
maven-metadata-local.xml	27/09/2023 11:57	Document XML	1 Ko

Déploiement sur un serveur distant

1. Utilisation d'un fichier jar (Java Archive)

[https://en.wikipedia.org/wiki/JAR_\(file_format\)](https://en.wikipedia.org/wiki/JAR_(file_format))

- **Programme Java exécutable (jar)**

Un programme Java exécutable peut être regroupé dans un fichier JAR, avec toutes les bibliothèques utilisées par le programme. Les fichiers JAR exécutables ont le manifeste spécifiant la classe du point d'entrée **Main-Class: fr.dawan.calculator.App** avec un chemin de classe explicite (et l'argument -cp est ignoré). Certains systèmes d'exploitation peuvent les exécuter directement lorsque vous cliquez dessus. L'invocation typique s'effectue **java -jar calculator.jar** à partir d'une ligne de commande.

- **Manifest**

Un fichier manifeste est un fichier [de métadonnées](#) contenu dans un JAR. Il définit les données liées aux extensions et aux packages. Il contient [des paires nom-valeur](#) organisées en sections. Si un fichier JAR est destiné à être utilisé comme fichier exécutable, le fichier manifeste spécifie la classe principale

de l'application. Le fichier manifeste est nommé `MANIFEST.MF`. Le répertoire manifeste doit être la première entrée de l'archive compressée.

Le manifeste apparaît à l'emplacement [canonique](#) `META-INF/MANIFEST.MF`. Il ne peut y avoir qu'un seul fichier manifeste dans une archive et il doit se trouver à cet emplacement.

Si une application est contenue dans un fichier JAR, la [machine virtuelle Java](#) doit connaître le point d'entrée de l'application. Un point d'entrée est n'importe quelle classe possédant une méthode **public static void main(String[] args)**. Ces informations sont fournies dans l'en-tête du manifeste `Main-Class`, qui a la forme générale :

```
Main-Class: fr.dawan.calculator
```

Dans cet exemple `fr.dawan.calculator.App.main()`, s'exécute au lancement de l'application.

2. Création d'un projet web avec SpringBoot

- Intégrer la dépendance dans notre projet :

```
<dependency>
  <groupId>fr.dawan</groupId>
  <artifactId>calculatrice</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</dependency>
```

`mvn install` : Il va faire les trois opérations, compile>test>génère le package> et l'installe l'artefact (`calculatrice_jar-0.0.1-SNAPSHOT.jar`) dans le repository local.

Un fichier JAR autonome est une application Java indépendante qui peut être exécutée localement sur une machine Java sans nécessiter de serveur externe ; car elle inclut toutes les dépendances requises, y compris les bibliothèques Java, et une version de tomcat embarquée.

L'exécution d'un fichier JAR autonome se fait généralement à l'aide de la commande `java -jar` suivie du nom du fichier JAR:

- `java -jar calculator-web-1.0-SNAPSHOT.jar`
- `java -jar calculator-web-1.0-SNAPSHOT.jar --server.port=8081`

Cela exécutera l'application autonome en utilisant la machine virtuelle Java locale.

Correction des erreurs

Parameter 0 of constructor in fr.dawan.calculator.metier.CalculatorMetier required a bean of type 'fr.dawan.calculator.Calculator' that could not be found.

```
/*
 * Configurer Spring pour Gérer les Beans
 * Dans notre classe de configuration Spring (annotée avec @Configuration),
 * créez un bean pour la classe Calculator.
 * */
@Bean
public Calculator calculator() {
    return new Calculator();
}
```

org.springframework.test.context.support.AnnotationConfigContextLoaderUtils -- Could not detect default configuration classes for test class [fr.dawan.calculatir.CalculatorJarApplication Tests]: CalculatorJarApplicationTests does not declare any static, non-private, non-final, nested classes

annotated with @Configuration. [ERROR] Tests run: 1, Failures: 0, Errors: 1, Skipped: 0, Time elapsed: 0.199 s <<< FAILURE! - in fr.dawan.calculatir.CalculatorJarApplicationTests

```
/*
 * L'erreur que vous rencontrez indique que Spring ne peut pas détecter les classes de configuration par
 * défaut pour votre classe de test CalculatorJarApplicationTests. Pour résoudre ce problème, vous
 * pouvez spécifier explicitement les classes de configuration à utiliser lors de l'exécution des tests
 */
@SpringBootTest(classes= {CalculatorJarApplication.class})
class CalculatorJarApplicationTests {
    @Test
    void contextLoads() {
    }
}
```

MANIFEST.MF

Le manifeste est généré automatiquement par le plugin Maven JAR

Champ	Explication
Manifest-Version: 1.0	Indique la version du format du manifeste. Dans ce cas, c'est la version 1.0.
Created-By: Maven JAR Plugin 3.3.0	Indique la version du plugin Maven JAR qui a créé ce fichier JAR (version 3.3.0 dans ce cas).
Build-Jdk-Spec: 17	La version minimale du JDK requise pour exécuter ce JAR (Java Development Kit).
Implementation-Title: calculator_jar	Le titre de l'implémentation du JAR, dans ce cas, "calculator_jar".
Main-Class: org.springframework.boot.loader.launch.JarLauncher	Indique la classe principale utilisée pour lancer le JAR. Dans ce cas, c'est une classe interne de Spring Boot (JarLauncher).
Start-Class: fr.dawan.calculator_jar.CalculatorJarApplication	Indique la classe principale de votre application Spring Boot.
Spring-Boot-Classes: BOOT-INF/classes/	Indique le répertoire dans lequel se trouvent les classes de l'application.
Spring-Boot-Lib: BOOT-INF/lib/	Indique le répertoire dans lequel se trouvent les bibliothèques (JAR) de l'application.
Spring-Boot-Classpath-Index: BOOT-INF/classpath.idx	Référence à un fichier d'index qui contient des informations sur le classpath.
Spring-Boot-Layers-Index: BOOT-INF/layers.idx	Référence à un fichier d'index qui contient des informations sur les couches (layers).

Si on veut déployer notre application sur autre machine/serveur tel qu'Apache Tomcat, on doit utiliser un fichier WAR (Web Application Archive). Une application Web contenue dans un fichier WAR nécessite un serveur d'applications pour son exécution, car elle dépend de l'infrastructure fournie par le serveur pour gérer les requêtes HTTP, gérer la configuration, etc.

3. Utilisation d'un fichier war (Web Archive)

On va transformer notre projet jar en war :

Dans le pom.xml : `<packaging>war</packaging>`

Un application jar intègre un **tomcat embarqué**, il va falloir exclure tomcat.

Le serveur d'application existe déjà (qui est notre serveur externe dans lequel on va déployer tomcat)

Pour cela ajouter cette dépendance, et **provided** pour le compléter :

Pourquoi provided ? Cette dépendance sera fournie par le serveur d'application sur lequel votre application sera déployée au moment de l'exécution. Cela permet de réduire la taille du fichier WAR de votre application, car il ne contient pas de bibliothèques déjà présentes sur le serveur d'application.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-tomcat</artifactId>
  <scope>provided</scope>
</dependency>
```

- **Servlet : classe ServletInitializer**

Dans une application web il nous faut une servlet qui démarre l'application Spring Boot.

Si on était dans une application Web Dynamique, il faudrait accéder au fichier « web.xml ».

Dans une application Spring Boot, on n'a pas de fichier « web.xml ». Une classe a donc été générée qui se nomme « WebInitializer » et qui hérite de « SpringBootServletInitializer ». En redéfinissant la méthode « configure », on lui passe « CalculatorWarApplication.class »

```
public class ServletInitializer extends SpringBootServletInitializer {
    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
        return application.sources(CalculatorWarApplication.class);
    }
}
```

- Dans une application web (war) la première classe qui va démarrer est WebInitializer qui hérite d'une servlet.
- Ensuite la classe WebInitializer démarre springBoot.
- Donc un fichier war, la servlet démarre en premier en suite springBoot
- Tomcat (ou un autre conteneur Web) doit déjà être en cours d'exécution pour déployer un fichier WAR. Le serveur de servlet (comme Tomcat) est responsable de l'exécution de votre application Spring Boot.
- Dans un fichier JAR autonome créé par Spring Boot, Spring Boot démarre en premier, ensuite springboot démarre tomcat

```
mvn install
```

Dans target il génère un fichier war. Ce war ne peut pas être exécuté comme le jar, car ce n'est pas une application autonome. Pour l'exécuter il nous faudra un serveur d'application (par exemple tomcat)

- **Contenu du fichier war**

Si nous regardons notre fichier war sa structure est différente du fichier jar

- Il contient un dossier **WEB-INF** (Car c'est un projet Web-Dynamic)
- Dans le dossier WEB-INF on a un dossier **lib-provided**. Ce sont des librairies qui sont déjà fourni par le serveur d'application. Il ne va pas les prendre en considération.
- Dans le même dossier on a les « classes »
- Dans le dossier « lib » on a les fichiers JAR (Java Archive) qui représentent des bibliothèques Java (par exemple, des dépendances tierces) nécessaires au fonctionnement de l'application.
- Le fichier MANIFEST.MF qui est dans le dossier META-INF permet à l'application Spring Boot d'être correctement exécutée et de charger toutes ses dépendances lorsqu'elle est déployée en tant que fichier WAR.

4. Comment déployer notre projet sur un serveur distant (ex : tomcat)

A. Installer tomcat 10.1.13

- Lien : <https://tomcat.apache.org/download-10.cgi>
- Choisir le « Core »

B. Démarrer « tomcat » en utilisant le fichier en utilisant le fichier « startup.bat »

- Lien du fichier « startup.bat » :


C:\Program Files\Apache Software Foundation\tomcat\apache-tomcat-10.1.13\bin

- Exécuter startup.bat en ligne de commande : **startup.bat**
- Tomcat démarre sur notre machine Window. On peut faire démarrer tomcat sur une machine distante (comme un serveur d'hébergement), pour cela on doit utiliser un accès distant (par exemple, SSH) pour accéder à cette machine.
- Tomcat par défaut démarre sur le port 8080. Vérifier que tomcat nous répond sur le port 8080, en tapant « localhost:8080 »
- Voici notre serveur tomcat :

Home Documentation Configuration Examples Wiki Mailing Lists Find Help

Apache Tomcat/9.0.62

If you're seeing this, you've successfully installed Tomcat. Congratulations!



Recommended Reading:

- [Security Considerations How-To](#)
- [Manager Application How-To](#)
- [Clustering/Session Replication How-To](#)

[Server Status](#)

[Manager App](#)

[Host Manager](#)

Developer Quick Start

- [Tomcat Setup](#)
- [First Web Application](#)
- [Realms & AAA](#)
- [JDBC Data Sources](#)
- [Examples](#)
- [Servlet Specifications](#)
- [Tomcat Versions](#)

Managing Tomcat

For security, access to the [manager webapp](#) is restricted. Users are defined in:

```
$CATALINA_HOME/conf/tomcat-users.xml
```

In Tomcat 9.0 access to the manager application is split between different users. [Read more...](#)

[Release Notes](#)

[Changelog](#)

[Migration Guide](#)

[Security Notices](#)

Documentation

[Tomcat 9.0 Documentation](#)

[Tomcat 9.0 Configuration](#)

[Tomcat Wiki](#)

Find additional important configuration information in:

```
$CATALINA_HOME/RUNNING.txt
```

Developers may be interested in:

- [Tomcat 9.0 Bug Database](#)
- [Tomcat 9.0 JavaDocs](#)
- [Tomcat 9.0 Git Repository at GitHub](#)

Getting Help

FAQ and Mailing Lists

The following mailing lists are available:

- [tomcat-announce](#)
Important announcements, releases, security vulnerability notifications. (Low volume).
- [tomcat-users](#)
User support and discussion
- [taglibs-user](#)
User support and discussion for [Apache Taglibs](#)
- [tomcat-dev](#)
Development mailing list, including commit messages

C. Déployer tomcat (avec Manager-App)

Pour déployer tomcat, on va utiliser **Manager-App** qui est une interface Web permettant de gérer le déploiement et la gestion des applications déployées sur Tomcat.

- **Authentification :**

Pour se connecter à [Manager-App](#) on doit fournir des informations d'identification. Par défaut, Tomcat utilise les informations d'identification configurées dans le fichier **tomcat-users.xml** situé dans le répertoire **conf**

Modifier le fichier tomcat-users.xml :

- Créer un utilisateur nommé "admin" avec le mot de passe "admin". Cet utilisateur sera associé aux rôles "manager-gui", "admin-gui" et "manager-script"
- Il a des droits d'accès aux fonctionnalités graphiques et de gestion de Tomcat, ainsi qu'aux fonctionnalités de script.

```
<role rolename="manager-gui"/>
<role rolename="admin-gui"/>
<role rolename="manager-script"/>
<user username="admin" password="admin" roles="manager-gui,admin-gui, manager-script"/>
```

- **Interface Manager-App :** <http://localhost:8080/manager/html>

On peut déployer un war en choisissant le fichier pour le déploiement

Dans target, choisissez le fichier .war :

```
C:\Users\Admin stagiaire.DESKTOP-8967908\Desktop\MiseEnPratique\FormationMaven\Me\calculator_maven\calculator_war\target
```

Tomcat ne se trouve pas en localhost mais sur un serveur distant. On est en train de déployer notre fichier **.war** sur un serveur distant

Après le déploiement notre application (l'artefact : calculator_war-0.0.1-SNAPSHOT.war) s'affiche dans la zone « Applications ». Elle est démarrée.

Double-clique sur l'application : **État HTTP 404 – Non trouvé**

Message :	OK
-----------	----

Gestionnaire			
Lister les applications	Aide HTML Gestionnaire	Aide Gestionnaire	Etat du serveur

Applications					
Chemin	Version	Nom d'affichage	Fonctionnelle	Sessions	Commandes
/	Aucun spécifié	Welcome to Tomcat	true	0	Démarrer Arrêter Recharger Retirer Expirer les sessions inactives depuis ≥ 30 minutes
/calculator_app	Aucun spécifié		true	0	Démarrer Arrêter Recharger Retirer Expirer les sessions inactives depuis ≥ 30 minutes
/host-manager	Aucun spécifié	Tomcat Host Manager Application	true	0	Démarrer Arrêter Recharger Retirer Expirer les sessions inactives depuis ≥ 30 minutes
/manager	Aucun spécifié	Tomcat Manager Application	true	1	Démarrer Arrêter Recharger Retirer Expirer les sessions inactives depuis ≥ 30 minutes

Deployer	
Emplacement du répertoire ou fichier WAR de déploiement sur le serveur	
Chemin de contexte (requis) :	<input type="text"/>
Version (pour les déploiements en parallèle) :	<input type="text"/>
URL du fichier XML de configuration :	<input type="text"/>
URL vers WAR ou répertoire :	<input type="text"/>
Deployer	
Fichier WAR à déployer	
Choisir le fichier WAR à téléverser <input type="button" value="Choisir un fichier"/> Aucun fichier choisi <input type="button" value="Deployer"/>	

D. Déploiement avec Maven

L'interface Manager-App de Tomcat est une interface web qui vous permet de déployer des applications manuellement sur un serveur Tomcat. Ce qui signifie que vous devez vous connecter à l'interface Web et télécharger votre application WAR à la main.

En revanche, l'utilisation du plugin Maven Tomcat vous permet d'automatiser le déploiement de votre application directement depuis la ligne de commande, ce qui est beaucoup plus efficace dans un environnement de développement ou de déploiement continu.

Avec le plugin Maven Tomcat, vous pouvez simplement exécuter une commande Maven, comme « mvn tomcat7:deploy », et Maven se chargera de télécharger votre application sur le serveur Tomcat

Pour automatiser le déploiement avec Maven, vous devez ajouter le plugin Tomcat à votre fichier pom.xml

- Déclaration du plugin dans le pom.xml :
- Une fois que vous avez ajouté le plugin et configuré votre fichier pom.xml, vous pouvez déployer votre application en exécutant la commande Maven « **mvn tomcat7:deploy** ».

```

58< build>
59< finalName>calculator_app</finalName>
60< plugins>
61< plugin>
62< groupId>org.springframework.boot</groupId>
63< artifactId>spring-boot-maven-plugin</artifactId>
64</plugin>
65< plugin>
66< groupId>org.apache.tomcat.maven</groupId>
67< artifactId>tomcat7-maven-plugin</artifactId>
68< version>2.2</version>
69< configuration>
70< url>http://localhost:8080/manager/text</url>
71< username>admin</username>
72< password>admin</password>
73< path>/calculator</path>
74</configuration>
75</plugin>
76</plugins>
77</build>
78

```

Décrivez votre projet dans le model POM

Les informations de base

```

1<?xml version="1.0" encoding="UTF-8"?>
2<project xmlns="http://maven.apache.org/POM/4.0.0"
3  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5
6<!-- La version du format du fichier pom. Actuellement la dernière version
7  est la 4.0.0. -->
8  <modelVersion>4.0.0</modelVersion>
9
10
11<!-- ===== Informations du projet -->
12<!-- ===== Informations Maven ===== -->
13<!-- ===== Informations générales ===== -->
14<!-- ===== Organisation ===== -->
15<!-- ===== Licences ===== -->
16<parent>
17  <groupId>org.springframework.boot</groupId>
18  <artifactId>spring-boot-starter-parent</artifactId>
19  <version>3.1.4</version>
20  <relativePath /> <!-- lookup parent from repository -->
21</parent>
22
23<groupId>fr.dawan</groupId>
24<artifactId>calculator_war</artifactId>
25<version>0.0.1-SNAPSHOT</version>
26<packaging>war</packaging>
27
28<!-- ===== Informations générales ===== -->
29<name>calculator_war</name>
30<description>Projet Calculator : permettant d'effectuer plusieurs opérations mathématiques </description>
31<url>http://www.dawan.fr/calculator</url>
32
33<!-- ===== Organisation ===== -->
34<organization>
35  <name>Dawan</name>
36  <url>https://www.dawan.fr</url>
37</organization>
38
39<!-- ===== Licences ===== -->
40<licenses>
41  <license>
42    <name>Apache License, Version 2.0</name>
43    <url>https://www.apache.org/licenses/LICENSE-2.0.txt</url>
44  </license>
45</licenses>

```

Les coordonnées du projet Maven

```
<groupId>fr.dawan</groupId>
<artifactId>calculator_war</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>war</packaging>
```

Un premier bloc définit les informations du projet relatives à Maven, avec :

- **groupId** : identifiant de l'organisation gérant le projet. Cet identifiant reprend la notation des packages Java. En général, celui-ci correspond au package de base de l'application, mais ce n'est pas obligatoire.
- **artifactId** : identifiant du projet
- **version** : version du projet.
- **packaging** : type de packaging devant être généré par Maven (jar, war, ear...).

En combinant les trois éléments (groupId, artifactId et version), vous obtenez un ***identifiant unique*** pour un **artefact** donné. Par exemple, l'identifiant unique "**fr.dawan: calculator-0.0.1-SNAPSHOT.jar**" indique que l'artefact est créé par l'organisation "fr.dawan", s'appelle "calculator" et est en version "1.0".

Un ***artéfact*** est un composant packagé possédant un identifiant unique composé de trois éléments : un groupId, un artifactId et un numéro de version.

Les artefacts sont stockés dans des référentiels Maven (comme le Central Repository) et sont téléchargés automatiquement par Maven lorsqu'ils sont spécifiés comme dépendances dans le fichier POM (Project Object Model) de votre projet

Par convention, une version en cours de développement d'un projet voit son numéro de version suivi d'un -SNAPSHOT. Et une fois la version terminée, prête à être déployée, vous devez enlever ce suffixe pour en faire une version release.

Après création de notre premier projet maven, on va maintenant plus loin dans sa définition.

Les informations générales du projet

```
<!-- ===== Informations générales ===== -->
<name>calculator_war</name>
<description>Projet Calculator : permettant d'effectuer plusieurs opérations mathématiques </description>
<url>http://www.dawan.fr/calculator</url>
```

On définit les informations générales du projet, avec les balises :

name : le nom du projet
description : la description du projet
url : URL du projet ou de l'application en production.

L'organisation gérant le projet

```
<!-- ===== Organisation ===== -->
<organization>
  <name>Dawan</name>
  <url>https://www.dawan.fr</url>
</organization>
```

Vous pouvez préciser des informations concernant l'organisation gérant le projet, avec la balise **organization**. Si vous travaillez comme prestataire de service, vous mettrez ici les informations du client pour lequel vous développez le projet.

Les licences du projet

```
<!-- ===== Licences ===== -->
<licenses>
  <license>
    <name>Apache License, Version 2.0</name>
    <url>https://www.apache.org/licenses/LICENSE-2.0.txt</url>
  </license>
</licenses>
```

Vous pouvez indiquer la ou les licences du projet grâce à la balise **<licenses>** et une sous-balise **<license>** par licence. Ceci est important si le projet est disponible publiquement ou s'il est sous licence libre.

<name> contient le nom de la licence
<url> contient l'URL vers le texte intégral de la licence.

Toutes ces informations seront reprises, entre autres, lors de la génération du site descriptif du projet par Maven.

Les propriétés

Les propriétés permettent d'ajouter un peu de généricité

Les propriétés sont souvent utilisées pour stocker des valeurs qui peuvent être réutilisées à plusieurs endroits dans le fichier pom.xml. On peut définir des propriétés spécifiques au projet ou des propriétés standard à Maven pour paramétrer la construction du projet.

Les propriétés sont des sortes de constantes qui sont remplacées par leur valeur lors de l'exécution de Maven en utilisant la notation **`${maPropriete}`** (qui sera remplacée par la valeur de la propriété `maPropriete`).

Vous pouvez définir les propriétés directement dans le fichier pom.xml, grâce à la balise <properties> :

```
<!-- ===== -->
<!-- Properties -->
<!-- ===== -->
<properties>
  <java.version>17</java.version>
  <maven.compiler.source>17</maven.compiler.source>
  <maven.compiler.target>17</maven.compiler.target>
</properties>
```

Voici un exemple d'utilisation des propriétés : on voudrait utiliser un Framework composé de plusieurs bibliothèques mais ne définir qu'une seule fois sa version.

```
<properties>
  <java-version>1.8</java-version>
  <org.springframework-version>5.2.1.RELEASE</org.springframework-version>
  <org.aspectj-version>1.6.10</org.aspectj-version>
  <org.slf4j-version>1.6.6</org.slf4j-version>
  <maven.compiler.target>1.8</maven.compiler.target>
  <maven.compiler.source>1.8</maven.compiler.source>
</properties>
<dependencies>
  <!-- Spring -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${org.springframework-version}</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${org.springframework-version}</version>
  </dependency>

  <!-- AspectJ -->
  <dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjrt</artifactId>
    <version>${org.aspectj-version}</version>
  </dependency>

  <!-- Logging -->
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>${org.slf4j-version}</version>
  </dependency>
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>jcl-over-slf4j</artifactId>
    <version>${org.slf4j-version}</version>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>${org.slf4j-version}</version>
    <scope>runtime</scope>
  </dependency>
</dependencies>
```

Exemple : Dans le projet calculator utiliser une propriété pour la version de JUnit Jupiter dans votre fichier pom.xml

1. Définir une propriété (par exemple junit.jupiter.version) dans la section <properties>
2. Référencez cette propriété dans la section <dependencies>.

```

<properties>
  <!-- Définissez la propriété pour la version de JUnit Jupiter -->
  <junit.jupiter.version>5.6.0</junit.jupiter.version>
</properties>

<dependencies>
  <!-- Utilisez la propriété pour spécifier la version de JUnit Jupiter -->
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>${junit.jupiter.version}</version>
    <scope>test</scope>
  </dependency>

  <!-- ... autres dépendances ... -->
</dependencies>

```

L'utilisation des propriétés n'est pas limitée au fichier pom.xml. En effet, il est possible de demander à Maven de remplacer les propriétés aussi dans les fichiers dits **resource**. Cela s'appelle [filter des fichiers ressource](#) dans le jargon de Maven. Ceci sera montré plus en détail dans la section *Le build*.

<https://maven.apache.org/guides/getting-started/index.html#how-do-i-filter-resource-files>

Propriétés pré-définies

En plus des propriétés que vous pouvez définir vous-même grâce à la balise **<properties>**, il existe également des [propriétés pré-définies](#) :

- **project.basedir** : donne le chemin vers le répertoire de base du projet, c'est-à-dire la racine de votre projet où se trouve le fichier pom.xml.
- **project.baseUri** : donne le chemin vers le répertoire de base du projet, mais sous forme d'URI.
- **maven.build.timestamp** : donne l'horodatage du lancement du build *Maven*

Propriétés particulières

Vous pouvez aussi accéder à des propriétés particulières grâce aux préfixes suivants :

- **env.** : permet de renvoyer la valeur d'une variable d'environnement. Par exemple, `${env.PATH}` renvoie la valeur de la variable d'environnement PATH.
- **project.** : renvoie la valeur d'une balise dans le fichier pom.xml du projet, en utilisant le point (.) comme séparateur de chemin pour les sous-balises. Par exemple,

```

<project>
  <organization>
    <name>Dawan</name>
  </organization>
</project>

```

est accessible via **`${project.organization.name}`**

- **settings.** : renvoie la valeur d'une balise dans le(s) fichier(s) settings.xml utilisé(s) par *Maven*. Utilise la notation pointée comme pour les propriétés de project.
- **java.** : renvoie la valeur d'une propriété système de Java. Ces propriétés sont les mêmes que celles accessibles via `java.lang.System.getProperties()`. Par exemple, `${java.version}` renvoie la version de Java

Le Build

En plus des informations de base et des propriétés, vous pouvez **paramétrer les différents éléments du processus de construction de Maven**, qu'on appelle le **build**.

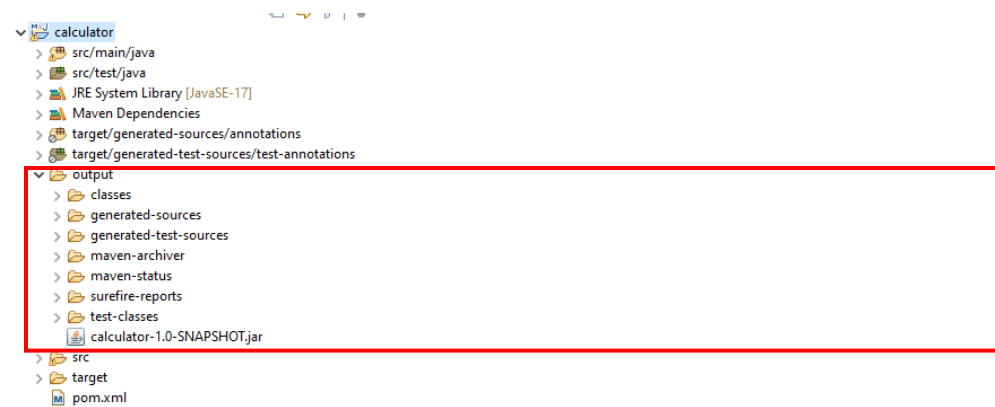
La configuration du build se fait grâce à la balise <build> et ses sous-balises

Exemple 1 :

1. Dans la balise <build> définir un chemin de sortie autre que celui par défaut :
`${project.basedir}/target`

```
63 <!-- ===== -->
64 <!-- Build -->
65 <!-- ===== -->
66 <build>
67   <directory>${project.basedir}/output</directory>
68 </build>
```

2. Ensuite **mvn package**



3. Il nous génère un autre chemin de sortie autre que target

Exemple 2 :

Si on prend le premier projet Maven que nous avons créé, on peut, par exemple, rendre le JAR généré exécutable en demandant à Maven d'indiquer la classe Main dans le Manifest du JAR :

Rappel : Les fichiers JAR exécutables ont le manifeste spécifiant la classe du point d'entrée **Main-Class: fr.dawan.calculator.App** avec un chemin de classe explicite (et l'argument **-cp** est ignoré). L'invocation typique s'effectue **java -jar calculator.jar** à partir d'une ligne de commande.

```

<!-- ===== -->
<!-- Build -->
<!-- ===== -->
<build>
  <!-- Gestion des plugins (version) -->
  <pluginManagement>
    <plugins>
      <!-- Plugin responsable de la génération du fichier JAR -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <version>3.0.2</version>
      </plugin>
    </plugins>
  </pluginManagement>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <configuration>
        <archive>
          <!-- Création du Manifest pour la définition de la classe Main -->
          <manifest>
            <mainClass>fr.dawan.calculator.App</mainClass>
          </manifest>
        </archive>
      </configuration>
    </plugin>
  </plugins>
</build>

```

Après avoir relancé la construction du projet avec **mvn package**, vous pourrez désormais lancer votre application Java sans indiquer la class main dans la ligne de commande :

```
java -jar target/calculator-1.0-SNAPSHOT.jar
```

Filtrer des fichiers ressources

<https://maven.apache.org/guides/getting-started/index.html#how-do-i-filter-resource-files>

Les **fichiers ressources** sont des fichiers qui n'ont pas à être compilés, mais simplement copiés dans le livrable généré par Maven. Par convention, ces fichiers se trouvent dans le répertoire **src/main/resources** (répertoire à créer si besoin).

Il est possible de dire à Maven de remplacer les propriétés par leurs valeurs dans des fichiers en filtrant les fichiers ressource.

Exemple :

Créer un fichier **info.properties** qui sera chargé par notre application, on mettra dans ce fichier une propriété contenant la version du projet.

1. Vous mettez ce fichier dans le répertoire **src/main/resources**.
Rafraichir le projet pour que Maven prenne en compte le répertoire resources
Clic droit sur le projet-> Maven -> Update project
2. Dans la classe App, ajouter :

```

Properties properties = new Properties();
InputStream inputStream = null;

try {
    inputStream = App.class.getResourceAsStream("/info.properties");
    properties.load(inputStream);
} catch (Exception e) {
    e.printStackTrace();
} finally {
    if (inputStream != null) {
        try {
            inputStream.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

System.out.println("Application version : " + properties.getProperty("fr.dawan.calculator.version", "?"));
}

```

- Contenu du fichier *info.properties* : **fr.dawan.calculator.version=1.0**
- Tester : **Clic droit sur le projet -> Run As -> Java Application**
- L'objectif ce n'est pas de générer cette valeur manuellement, mais plutôt d'aller chercher la valeur de la propriété version du projet dans le fichier pom.xml. Il est possible de faire cela avec maven, en faisant **du filtrage de ressource**
- Vous indiquez à Maven, dans le fichier pom.xml, dans la section build de filtrer les fichiers du répertoire src/main/resources avec la balise <filtering> qui sera ajouté dans la balise <resources>:

```

<resources>
  <resource>
    <directory>src/main/resources</directory>
    <filtering>true</filtering>
  </resource>
</resources>

```

Désormais, à la construction de l'application par Maven, les fichiers contenus dans src/main/resources seront filtrés. C'est-à-dire que Maven va chercher à l'intérieur de ces fichiers des propriétés, et va les remplacer par leur valeur. Comme il ferait dans le fichier pom.xml

- Modifier le fichier info.properties, ajouter : **fr.dawan.calculator.version=\${project.version}**
- Dans ce fichier, vous utilisez la même syntaxe que dans le fichier pom.xml :

Remarque :

Si vous avez des fichiers à ne pas filtrer dans le répertoire src/main/resources, il est possible de faire des sous-répertoires dans ce dernier afin d'organiser vos fichiers.

Voici un exemple où seulement les fichiers du sous-dossier filtered doivent être filtrés :

```

v src/main/resources
  v raw
    app.properties
  v filtered
    info.properties

```

```

<resources>
  <resource>
    <directory>src/main/resources/filtered</directory>
    <filtering>true</filtering>
  </resource>
  <resource>
    <directory>src/main/resources/raw</directory>
    <filtering>false</filtering>
  </resource>
</resources>

```

Les profils

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

Les **profils** permettent de créer des options *dans le build Maven*.

Vous pouvez par exemple envisager deux environnements cibles, deux fichiers propriétés (environnement de test, environnement de production) et embarquer, dans le JAR généré, des fichiers de configurations différents en fonction de la cible. Il suffit de créer deux profils (test et prod), chacun définissant un répertoire de fichiers ressources différent :

```

<!-- Profils -->
<!-- ===== -->
<profiles>
  <!-- Profil pour l'environnement de test -->
  <profile>
    <id>test</id>
    <build>
      <resources>
        <resource>
          <directory>src/main/resources/conf-test</directory>
          <filtering>true</filtering>
        </resource>
      </resources>
    </build>
  </profile>
  <!-- Profil pour l'environnement de production -->
  <profile>
    <id>prod</id>
    <build>
      <resources>
        <resource>
          <directory>src/main/resources/conf-prod</directory>
          <filtering>true</filtering>
        </resource>
      </resources>
    </build>
  </profile>
</profiles>

```

Remarque : Ne plus utiliser info.properties , et supprimer ou mettre en commentaire la section « resources » du build principal

1. Dans le dossier `src/main/ressources` créer deux sous-répertoires
 - Un répertoire pour la configuration de l'environnement de Test
 - Un répertoire pour la production

Il s'agit maintenant de dire à Maven de prendre les fichiers ressources qui sont dans les répertoires conf-prod lorsqu'il s'agit de construire un livrable pour l'environnement de production, et de sélectionner les fichiers qui sont dans le répertoire conf-test lorsqu'il s'agit de construire un livrable pour l'environnement de test

2. Construire nos livrable :

mvn clean package

Notre jar contient les deux dossiers créés, conf-prod et conf-test. Par convention tout ce qui se trouve dans src/main/resources est recopié dans le jar final

Pour activer le bon profil lors du lancement du build Maven on utilise l'option **-P** :

```
# Pour construire un livrable pour l'environnement de test :
```

```
mvn package -P test
```

```
# Pour construire un livrable pour l'environnement de production :
```

```
mvn package -P prod
```

Exemple : Disons à Maven que nous voulons construire un environnement de test

On utilise la même commande en ajoutant l'option « -P test » pour activer le profile test

mvn package -P test

- Résultat après exécution : fr.dawan.calculator.version = 0.0.1-SNAPSHOT TEST

Construisons un jar pour la Prod : mvn package -P prod

- Résultat après exécution : fr.dawan.calculator.version = 0.0.1-SNAPSHOT PROD

Générez un site pour votre projet

Précédemment, nous avons vu comment utiliser *Maven* pour

- Construire votre projet
- Gérer ses dépendances
- Et générer les livrables

Je vais vous montrer maintenant qu'il est aussi possible d'utiliser *Maven* pour générer un site web pour votre projet.

Vous y retrouverez :

- Des informations générales sur celui-ci,
- La Javadoc,
- Des rapports sur l'exécution des tests, la qualité du code...

Convention sur l'arborescence

Par convention, les sources servant à la génération du site du projet se trouvent dans le répertoire **src/site**, lui-même organisé de cette manière :

<https://maven.apache.org/plugins/maven-site-plugin/examples/creating-content.html>

```
my-project
|-- src
|   |-- site
|       |-- apt
|           |-- index.apt
|       |-- fml
|           |-- index.fml
|       |-- markdown
|           |-- index.md
|       |-- resources      (Répertoire de ressources pour la documentation)
|           |-- images
|               |-- logo.png
|           |-- styles
|               |-- style.css
|-- pom.xml
...
```

<https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html>

- Le fichier **src/site/site.xml** est le *site descriptor*. Il permet de définir la structure du site.
- Les répertoires apt, markdown, fml... hébergent les fichiers source des pages du site qui seront converties en HTML par Maven. Ces répertoires accueillent des fichiers de leur format respectif :
 - **Apt** (*Almost Plain Text*) : <https://maven.apache.org/doxia/references/apt-format.html>
 - **Fml** (FAQ Markup Language) : <https://maven.apache.org/doxia/references/fml-format.html>
 - **markdown** : <https://daringfireball.net/projects/markdown/>
 - **autres formats supportés par Maven** : https://maven.apache.org/plugins/maven-site-plugin/examples/creating-content.html#Documentation_formats

- Le répertoire ressources pour les autres ressources :
<https://maven.apache.org/plugins/maven-site-plugin/examples/creating-content.html#adding-extra-resources>
 - css/style.css : fichier CSS permettant d'ajuster le style par défaut
 - images/pic1.jpg

Le site descriptor

<https://maven.apache.org/plugins/maven-site-plugin/examples/sitedescriptor.html>

Le fichier **src/site/site.xml** est le site descriptor. Il permet de définir la structure du site.

Je crée donc ce fichier site.xml :

```
project xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://maven.apache.org/DECORATION/1.7.0"
  xsi:schemaLocation="http://maven.apache.org/DECORATION/1.7.0 http://maven.apache.org/xsd/decoration-1.7.0.xsd">

  <!-- Utilisation du template de site fluïdo -->
  <skin>
    <groupId>org.apache.maven.skins</groupId>
    <artifactId>maven-fluido-skin</artifactId>
    <version>1.6</version>
  </skin>

  <!-- Affichage de la date et de la version à droite dans le bandeau du haut -->
  <publishDate position="right"/>
  <version position="right"/>

  <body>
    <!-- Ajout d'un fil d'ariane -->
    <breadcrumbs>
      <item name="Accueil" href="index.html"/>
    </breadcrumbs>

    <!-- ===== Menus ===== -->
    <!-- Ajout d'un menu vers le projet parent -->
    <menu ref="parent" inherit="top"/>
    <!-- Ajout d'un menu vers les différents modules du projet -->
    <menu ref="modules" inherit="top"/>
  </body>
</project>
```

Configuration du site dans le POM

Pour générer un site il faut définir l'URL de déploiement du site, cela se fait dans la section ***distributionManagement***.

L'ajout de ces informations est requis par Maven, même si vous ne comptez pas déployer le site automatiquement, et que vous voulez l'utiliser en local sur votre poste. Dans ce cas, vous pouvez utiliser une URL de cette forme : **scp://localhost/tmp/**.

```
<!-- ===== -->
<!-- DistributionManagement -->
<!-- ===== -->
<distributionManagement>
  <site>
    <id>site-projet</id>
    <url>scp://localhost/tmp/</url>
  </site>
</distributionManagement>
```

L'élément **<distributionManagement>** dans le fichier pom.xml de Maven est utilisé pour spécifier comment les artefacts générés par le projet doivent être distribués. Il est couramment utilisé pour configurer le déploiement d'artefacts dans un référentiel distant (tel que Nexus, Artifactory, ou un

autre référentiel Maven distant) ou pour déployer des artefacts sur un serveur distant via SSH, FTP, etc.

Générer le site

Ajout du plugin maven-site-plugin

Afin de générer le site, vous allez utiliser le plugin maven-site-plugin.

```
<!--Generation du site -->
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-site-plugin</artifactId>
  <version>3.7.1</version>
  <configuration>
    <!-- Je veux le site en français -->
    <locales>fr</locales>
  </configuration>
</plugin>
```

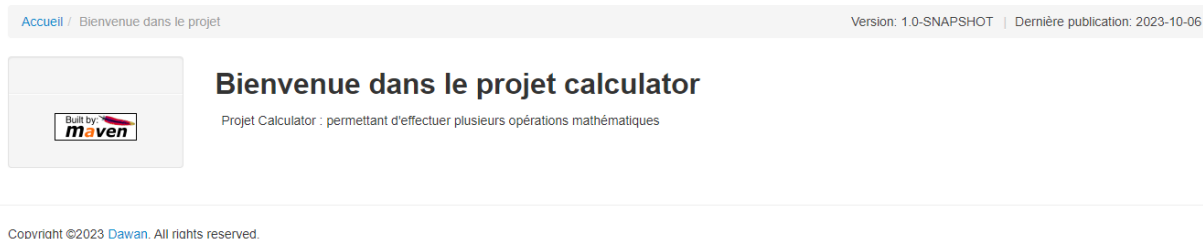
Génération du site

Pour générer le site, nous allons utiliser le build lifecycle [site](#).

```
mvn package site
```

Il ne vous reste plus qu'à ouvrir le fichier target/site/index.html :

calculator



On peut remarquer que les informations supplémentaires sur le projet sont indiquées sur le site. Par exemple la description, l'organisation et son url.

Ajouter des pages personnalisées

Maintenant que vous savez générer le site, ajoutons-y vos propres pages.

C'est ici que vous allez utiliser les dossiers apt, fml, markdown...

Je vous propose d'ajouter 2 pages dans le site :

- Une page de documentation expliquant comment est organisé le projet
- Une foire aux questions

Ajout d'une page de documentation

Pour la page de documentation, je vais utiliser le format [Markdown](#).

Je crée donc le répertoire `src/site/markdown`, et j'y crée un fichier `architecture.md` :

```

1 ## Architecture du projet
2
3 ### Généralités
4
5 Lorem ipsum dolor sit amet, consectetur adipisicing elit,
6 sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
7
8
9 Une image :
10
11 ![Dépendances entre les modules](img/diagramme_de_classe_calculator.png)
12
13
14 ### L'application web
15
16 Lorem ipsum dolor sit amet, consectetur adipisicing elit,
17 sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
18
19
20 ### Les batches
21
22 Lorem ipsum dolor sit amet, consectetur adipisicing elit,
23 sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
24

```

On va utiliser une image dans notre page : `img/ diagramme_de_classe_calculator.png`. Je la mets donc dans le répertoire `resources/img`.

Enfin, j'ajoute une entrée de menu dans le *site descriptor* pour pouvoir accéder à cette page :

```

23 <!-- Menu personnalisé -->
24 <!-- ===== Menus ===== -->
25 <!-- Ajout d'un menu vers la documentation -->
26 <menu name="Documentation">
27   <!-- Entrée de menu vers la page Architecture -->
28   <item name="Architecture" href="architecture.html" />
29 </menu>
30

```

Vous remarquez ici que le lien pointe vers ***architecture.html*** et non pas ***architecture.md***. En effet, Maven va convertir le fichier `architecture.md` en une page HTML `architecture.html`.

Il ne reste plus qu'à relancer la génération du site pour obtenir les éléments que nous venons d'ajouter. `mvn clean package site`

Ajout d'une FAQ

Pour ajouter une page de Foire aux questions, il suffit de suivre la même démarche que pour l'ajout d'une page, sauf que vous pouvez utiliser le format ***FAQ Markup Language***, plus adapté à l'écriture d'une FAQ.

Je crée le fichier `src/site/fml/faq.fml`

J'ajoute une entrée de menu dans le *site descriptor* pour pouvoir accéder à cette FAQ :

```

<!-- Menu personnalisé -->
<!-- ===== Menus ===== -->
<!-- Ajout d'un menu vers la documentation -->
<menu name="Documentation">
  <!-- Entrée de menu vers la page Architecture -->
  <item name="Architecture" href="architecture.html" />
  <!-- Entrée de menu vers la page FAQ -->
  <item name="FAQ" href="faq.html"/>
</menu>

```

`mvn clean package site`

Générez des rapports

<https://maven.apache.org/plugins/maven-site-plugin/examples/configuring-reports.html>

Rapports de base

Maven permet de générer, de base, un certain nombre de rapports sur votre projet :

- **Projet :**
 - Résumé du projet (nom, version, description, organisation...)
 - Liste des modules du projet
 - Membres du projet (contributeur, développeurs...)
 - Licence du projet (section `<licenses>`)
 - Gestion de la distribution du projet (section `<distributionManagement>`)
 - Listes de diffusion (section `<mailingLists>`)
 - Dépôt des sources du projet (section `<scm>`)
 - Intégration continue (section `<ciManagement>`)
 - Gestion des anomalies (section `<issueManagement>`)
- **Plugins :**
 - Gestion des plugins (section `<pluginManagement>`)
 - Liste des plugins utilisés dans le projet/module
- **Dépendances :**
 - Gestion des dépendances (section `<dependencyManagement>`)
 - Liste des dépendances utilisées dans le projet/module
 - Convergence des dépendances entre les différents modules du projet

Pour ajouter ces rapports au site, il faut ajouter un menu dans le fichier `site.xml`

```
<body>
...
<!-- ===== Menus ===== -->
<!-- Ajout d'un menu vers les différents rapport -->
<menu ref="reports" inherit="top"/>
</body>
```

Attention, ce plugin est un *plugin de rapport*. Il doit donc être ajouté à la section `<reporting>` et non `<build>`

```
<!-- ===== -->
<!-- Gestion des rapports -->
<!-- ===== -->
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-project-info-reports-plugin</artifactId>
      <version>2.7</version>
    </plugin>
  </plugins>
</reporting>
```

mvn package site

Sélectionner certains rapports seulement

Par défaut lorsque vous ajoutez un plugin de rapport tous ces goals seront exécutés, donc tous les rapports seront générés. Si vous le souhaitez, vous pouvez ne lancer que certains goals en ajoutant une section **<reportSets>** :

```
<!-- ===== Gestion des rapports -->
<!-- Gestion des rapports -->
<!-- ===== -->
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-project-info-reports-plugin</artifactId>
      <version>2.7</version>
      <reportSets>
        <reportSet>
          <reports>
            <report>index</report>
            <report>summary</report>
            <report>plugins</report>
          </reports>
        </reportSet>
      </reportSets>
    </plugin>
  </plugins>
</reporting>
```

Ces reportSet permet de sélectionner que certains rapports (index, summary et plugins)

mvn clean package site

Ajouter d'autres rapports

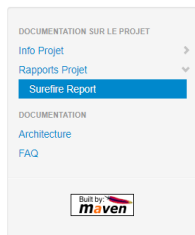
Pour générer des rapports supplémentaires, il suffit d'ajouter les plugins correspondants :

Liste des plugins (les plugins de rapports sont notés Type : R) <https://maven.apache.org/plugins/>

1. Par exemple, pour **générer un rapport sur les tests** nous allons utiliser le plugin **surefire-report**:

```
<!-- ===== Rapport sur les tests ===== -->
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-report-plugin</artifactId>
  <version>2.20</version>
  <configuration>
    <!-- Configuration spécifique indiquant de ne pas générer les liens
    XRef. dites cross-reference links -->
    <linkXRef>false</linkXRef>
  </configuration>
  <reportSets>
    <reportSet>
      <reports>
        <!-- Goal report du plugin surefire-report -->
        <report>report</report>
      </reports>
    </reportSet>
  </reportSets>
</plugin>
```

mvn clean package site



Surefire Report

Summary

[Summary] [Package List] [Test Cases]

Tests	Errors	Failures	Skipped	Success Rate	Time
5	0	0	0	100%	0,051

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Package List

[Summary] [Package List] [Test Cases]

Package	Tests	Errors	Failures	Skipped	Success Rate	Time
fr.dawan.calculator	5	0	0	0	100%	0,051

Note: package statistics are not computed recursively, they only sum up all of its testsuites numbers.

fr.dawan.calculator

	Class	Tests	Errors	Failures	Skipped	Success Rate	Time
☀	CalculatorTest	5	0	0	0	100%	0,051

Test Cases

[Summary] [Package List] [Test Cases]

CalculatorTest

2. Générer un rapport sur la qualité du code

Pour générer un rapport sur la qualité du code, nous allons utiliser **checkstyle**

<https://maven.apache.org/plugins/maven-checkstyle-plugin/examples/custom-checker-config.html>

```
<!-- ===== Rapport sur la qualité (l'analyse) du code par Checkstyle ===== -->
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-checkstyle-plugin</artifactId>
  <version>2.17</version>
  <configuration>
    <configLocation>src/build/checkstyle.xml</configLocation>
    <linkXRef>>false</linkXRef>
  </configuration>
  <reportSets>
    <reportSet>
      <reports>
        <report>checkstyle</report>
      </reports>
    </reportSet>
  </reportSets>
</plugin>
```

Il faudra préciser quel est le fichier de configuration de **checkstyle** à utiliser.

1. Créer un fichier **checkstyle.xml**
2. Créer un dossier **build** dans **src**, puis ajouter le fichier **checkstyle.xml**

Le fichier **checkstyle.xml** configure des règles pour l'outil de vérification de code source. Ce fichier est utilisé pour spécifier les règles que **Checkstyle** doit appliquer lors de l'analyse du code source Java. Les règles définissent divers aspects du style de codage, de la mise en forme du code et des bonnes pratiques de programmation.

<https://checkstyle.org/config.html#Checker>

