DevOps

Maven

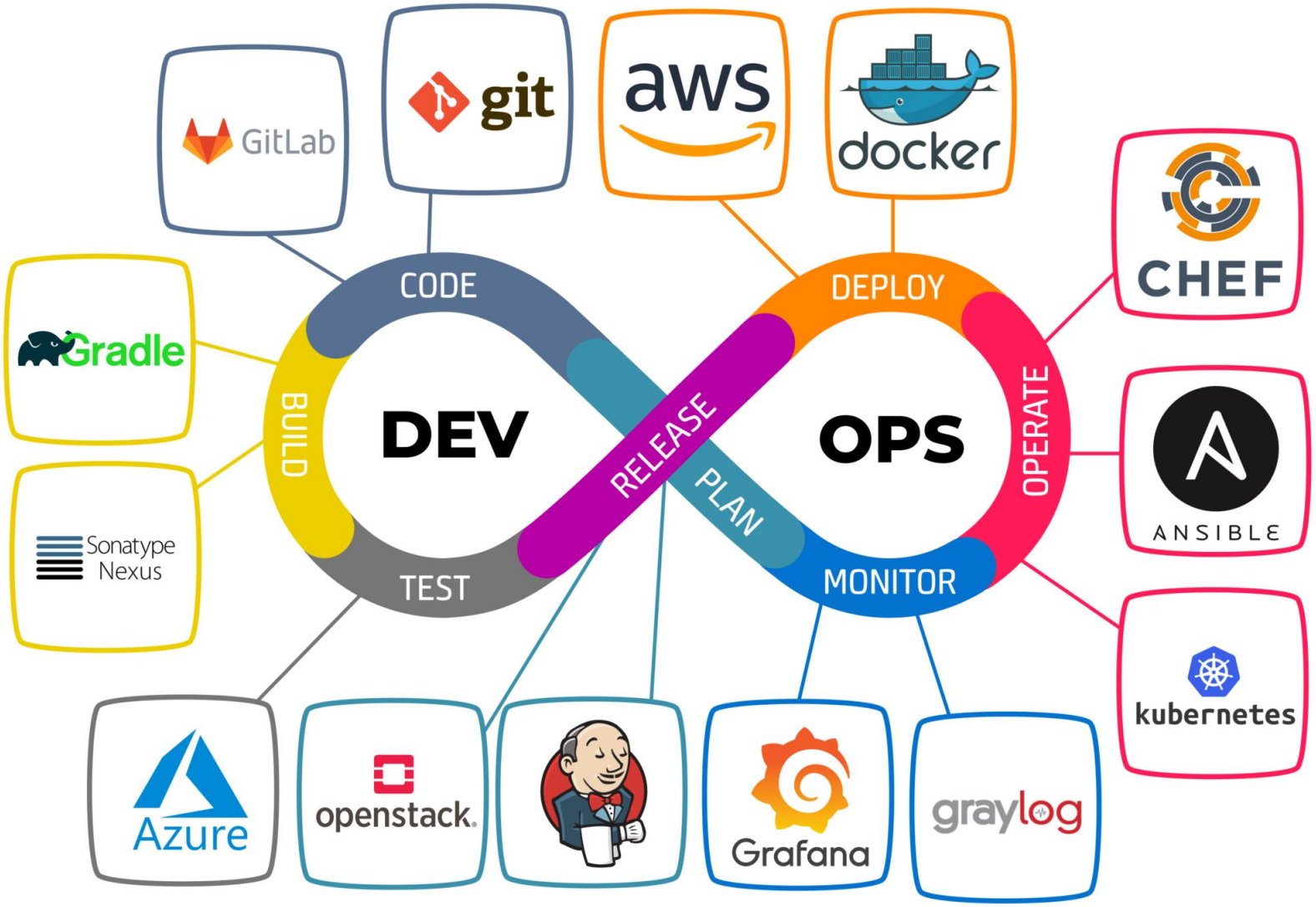
Jamel ESSOUSSI: Architecte logiciel

jamel.essoussi@gmail.com

https://github.com/jessoussi

2025

Outils du DevOps



Plan du cours

I. Introduction

II. Les principes de Maven

III. Le pom.xml

IV. Les plugins, les tâches, les cycles de vie

V. Coordonnées, dépôts et dépendances

VI. Héritage & Aggrégation

VII. Archetype

Introduction

Ce que c'est:

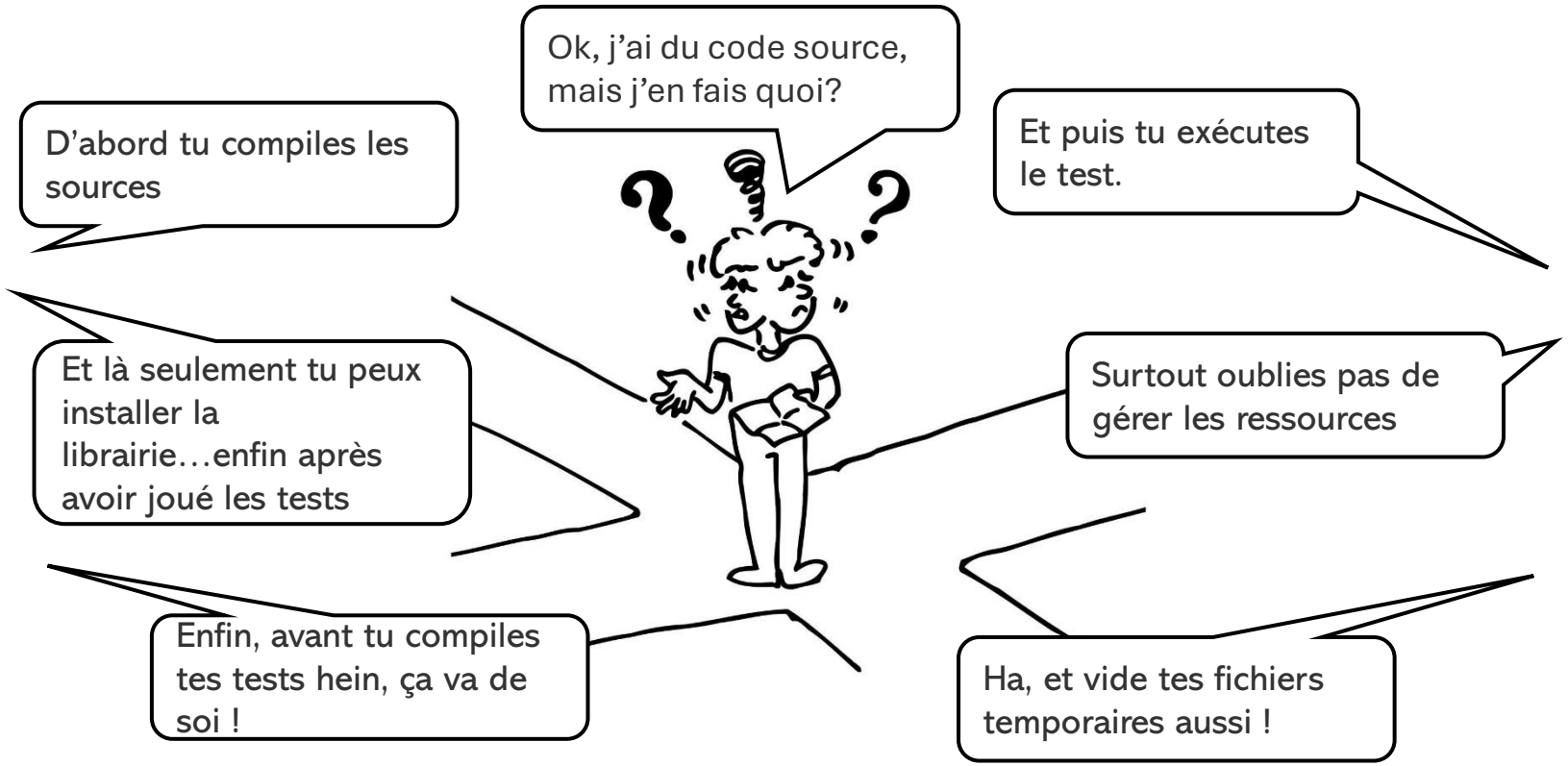
- Outils pour la gestion et l'automatisation de production de projets logiciels qui va chercher à produire un logiciel à partir de ses sources, en optimisant les tâches et en garantissant le bon ordre de fabrication.
- Cible principalement Java et en particulier les applications J2EE.

Introduction

- Maven est développé en 2003 par Apache Apache Software Foundation.
- La dernière version est la 3.9.11
- Maven 4 en cours de développement

Introduction

- Pas de cycle de vie standard de projet
- Gestion manuelle des bibliothèques dont dépendent le projet
- Pas de structure standard de projet



Introduction

Maven permet de: « Aider le développeur »

- Créer des builds customisables,
- Gérer l'utilisation des dépendances vers des librairies externes,
- D'accéder facilement à un large (et évolutif) répertoire de librairies,
- D'établir une convention uniforme,
- Faire du templating de code archetype,
- Faire du source code/release management,
- Faire de la gestion documentaire (distribution, mailing lists …).

Plan du cours

I. Introduction

II. Les principes de Maven

III. Le pom.xml

IV. Les plugins, les tâches, les cycles de vie

V. Coordonnées, dépôts et dépendances

VI. Héritage & Aggrégation

VII. Archetype

Les principes de Maven

Convention plutôt que configuration
(**Convention Over Configuration**)

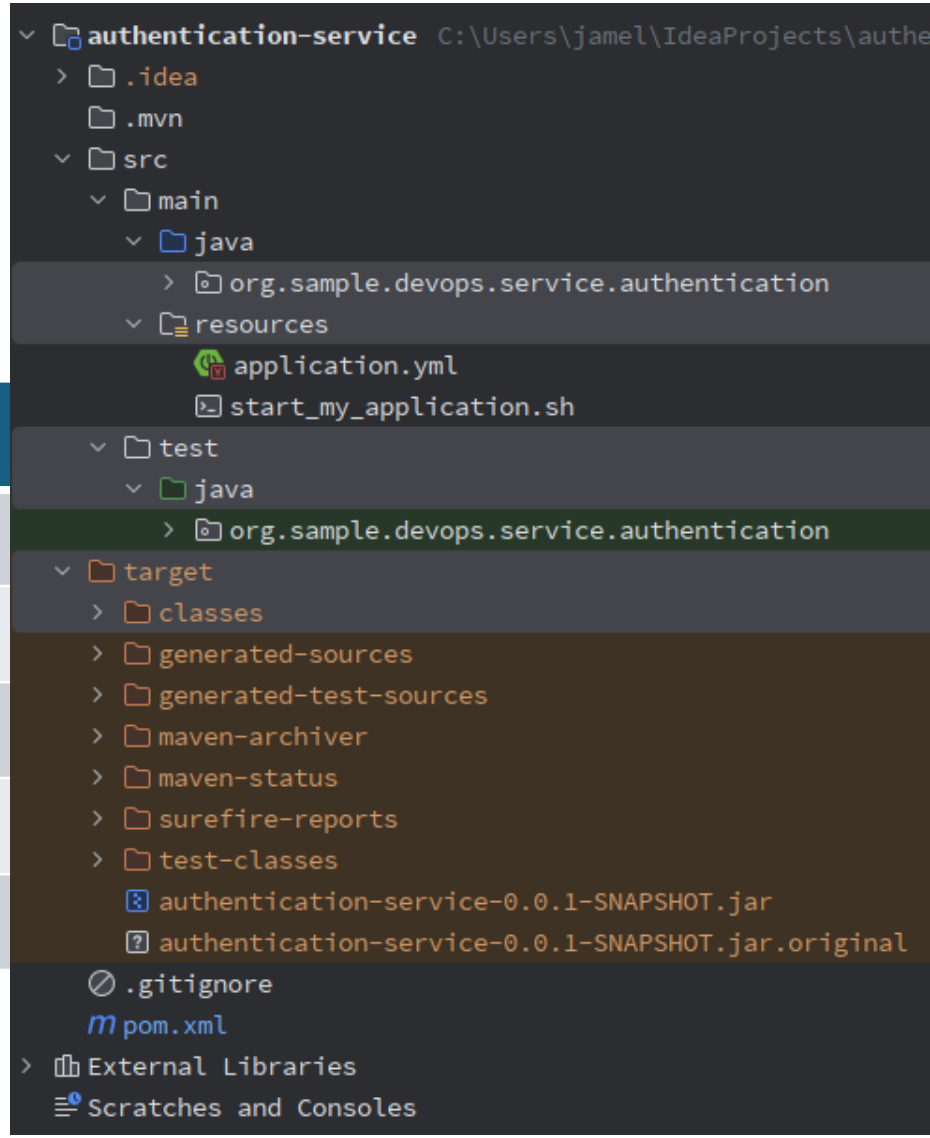
- « Tant que l'on suit la convention, pas besoin de préciser quoi que ce soit »
- Par défaut, tous les projets se ressemblent
  - Initialiser, Compiler, Tester, Assembler, . . .
- Maven définit une structure de projet par défaut
  - Ensemble de conventions raisonnables,
  - Il faut préciser ce qui ne suit pas les conventions

Les principes de Maven

Les conventions Maven

Convention	Valeur par défaut
Code source	${basedir}/src/main/java
Ressources	${basedir}/src/main/resources
Tests	${basedir}/src/test
Code compilé	${basedir}/target
Fichiers class	${basedir}/target/classes

Un plugin Maven permet d'initialiser tout ceci automatiquement



Les principes de Maven

Décrire plutôt que programmer

- ❑ Approche déclarative
- ❑ On indique les particularités du projet et non la manière de le construire

descripteur du projet

pom.xml

Plan du cours

I. Introduction

II. Les principes de Maven

III. Le pom.xml

IV. Les plugins, les tâches, les cycles de vie

V. Coordonnées, dépôts et dépendances

VI. Héritage & Aggrégation

VII. Archetype

Le pom.xml

- Project Object Model
- C'est un fichier au format xml
- Il contient la totalité des informations utiles au build d'un projet
- modelVersion définit le modèle de structure de projet.
  - 4.0.0 est le modèle par défaut
- pom.xml étend un fichier super-POM définit dans Maven
  - On peut aussi avoir des pom.xml parent.

Le pom.xml: sections principales

- groupId
  - Identifiant du groupe ayant créé le projet
- artifactId
  - Nom de l'artifact généré par le projet
- version
  - Numéro de version de l'artifact
- packaging
  - Type de l'application (jar/war/ear/pom)
- properties
  - Définition de propriétés/constantes
- dependencies
  - Définition des dépendances du projet
- build
  - Déclaration et configuration des plugins
  - Définition de propriétés du projet

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns=http://maven.apache.org/POM/4.0.0
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>…</modelVersion>
    <groupId>…</groupId>
    <artifactId>…</artifactId>
    <packaging>…</packaging >
    <version>…</version>
    <properties>…</properties>
    <dependencies>…</dependencies>
    <build>…</build>

</project>
```

Le pom.xml: sections documentation

- Certaines informations ne sont pas nécessaires au build. Elles sont uniquement présentes à titre documentaire. Elles sont facultatives.

  - name : Nom (fonctionnel) du projet
  - description : Descriptif du projet
  - url : URL du projet (s'il en possède une)
  - inceptionYear : Année de création du projet
  - organization : Nom de l'organisation
  - licenses : Type de licence
  - developers : Nom des développeurs du projet
  - contributors : Nom des contributeurs du projet

- Bien que facultatives, ces informations peuvent être utilisés par certains plugins pour générer des fichiers de licences.

Le pom.xml sections principals Exemple du TD

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.5.6</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>org.sample.devops</groupId>
    <artifactId>authentication-service</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>Authentication Service</name>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

Plan du cours

I. Introduction

II. Les principes de Maven

III. Le pom.xml

IV. Les plugins, les tâches, les cycles de vie

V. Coordonnées, dépôts et dépendances

VI. Héritage & Aggrégation

VII. Archetype

Plugin

- Fragment de logiciel qui se spécialise dans une tâche donnée.
- Ex: compilation, tests

Goals (Tâches)

- Un plugin peut exécuter un ensemble de goals (tâches unitaires)
- Pour exécuter un goal, on utilise la notation suivante: mvn plugin:goal
- Ex: compile du plugin Compiler, test du plugin surefire, …
- Ex: mvn depedency:tree, mvn archetype:generate

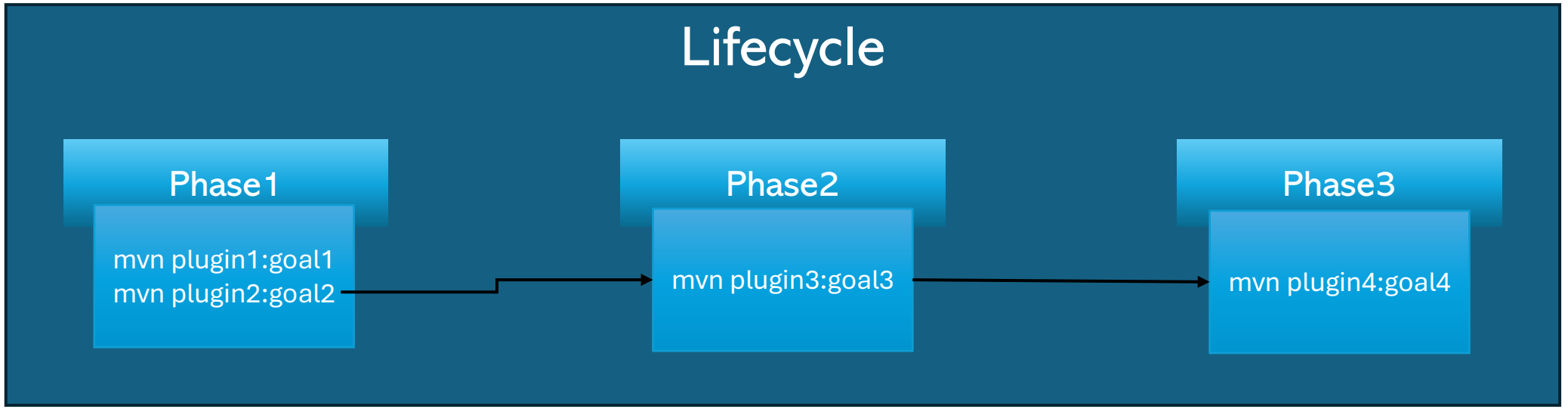
Phase

- Une phase peut contenir zéro à plusieurs plugin goals
- Pour exécuter une phase, on utilise la notation suivante: >mvn phase
- Ex: mvn clean -> Exécute la phase clean

Lifecycle (Cycle de vie)

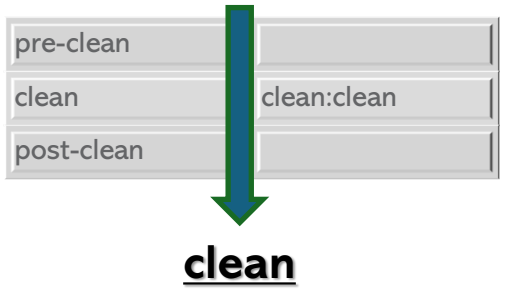
- Pour identifier et enchaîner les tâches de base dans un projet,
- Maven se base sur :
  - Les plugins et les tâches associées,
  - Un cycle de vie

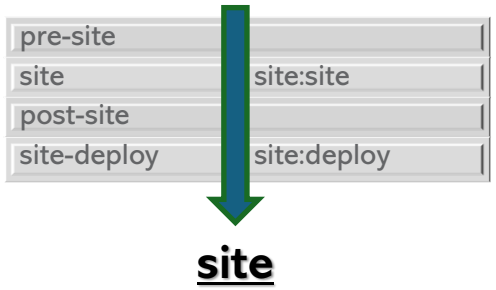
Lifecycle (Cycle de vie)

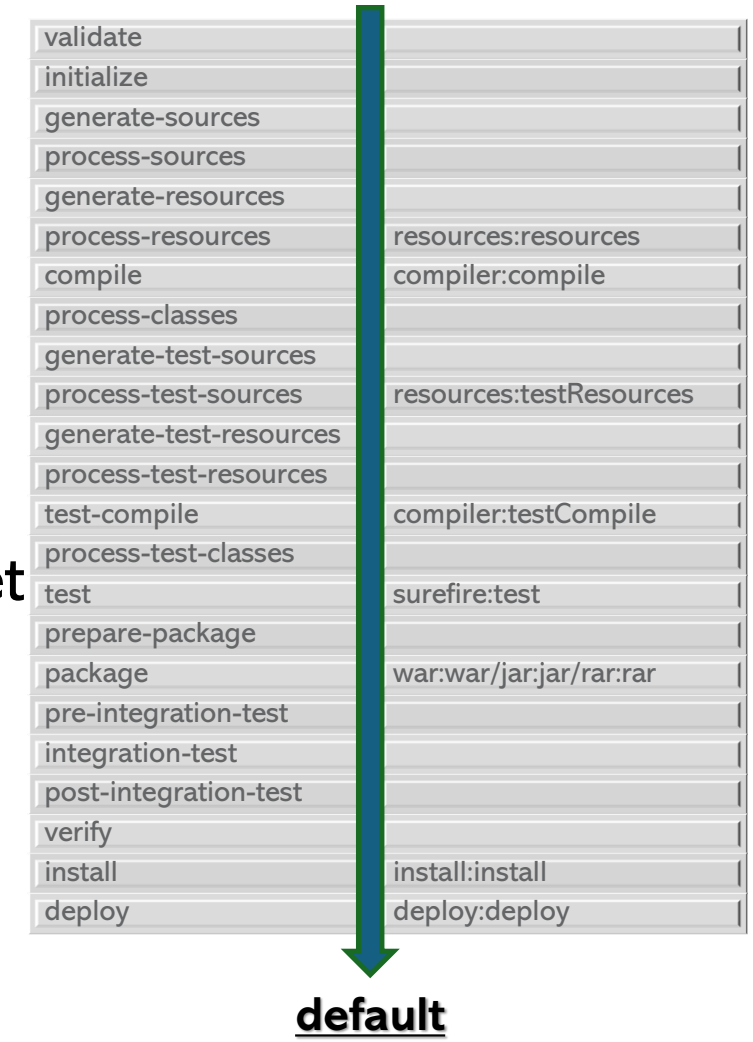


Lifecycle

- 3 cycles de vie prédéfinis avec 30 phases:
  - **default** : construire le projet
  - **clean** : nettoyage du projet
  - **site** : création de la documentation du projet







Lifecycle default

- Les phases principales du cycle de vie par d´efaut1:
  - validate: valide que le projet est correct et que toutes les infos nécessaires sont disponibles
  - compile
  - test
  - package: package les sources compilées dans un format distribuable (par ex JAR)
  - integration-test
  - verify: Lance les tests pour vérifier la qualité du package
  - install: Installe le package dans le dépôt local
  - deploy: Copie le package final dans un dépôt distant pour le partager

Lifecycle

- Demander l'exécution d'une phase d'un cycle entraîne l'exécution de toutes les phases précédentes.
  - mvn deploy
    - Exécute toutes les phases du cycle par défaut
  - mvn clean install
    - Exécute la phase clean (et précédentes) puis install (et précédentes)

- Qu'est ce qui est exécuté par une phase?
  - Les tâches qui lui ont été associées

Lifecycle

- Il est possible d'appeler plusieurs cycles en une même commande:
  - mvn clean deploy
    - Maven va alors les exécuter dans l'ordre indiqué

Plan du cours

I. Introduction

II. Les principes de Maven

III. Le pom.xml

IV. Les plugins, les tâches, les cycles de vie

V. Coordonnées, dépôts et dépendances

VI. Héritage & Aggrégation

VII. Archetype

Coordonnées, dépôts et dépendances

- Chaque projet est identifié de manière unique
  - ✓ Coordonnées
- Les artifacts d'un projet peuvent être publiés vers un dépôt maven
  - ✓ Local ou distant
- Les artifacts nécessaires à un projet (dépendances) sont téléchargés automatiquement par maven
  - ✓ Gestion automatique des dépendances transitives

Coordonnées

- Le fichier POM fournit un ensemble d'identifiants uniques du projet:

```
<groupId>org.sample.devops</groupId>
<artifactId>authentication-service</artifactId>
<packaging>jar</packaging>
<version>1.0-SNAPSHOT</version>
```

- groupId:artifactId:packaging:version identifie de manière unique le projet
  - groupId: identifie l'entité qui gère le projet
  - artifactId: identifie le projet
  - version: Numéro de version du projet
    - SNAPSHOT: Mot clé indiquant à Maven que le projet est en cours de développement

Coordonnées

- Les dépôts maven (publics, privés, locaux) sont organisés autour de ces coordonnées,
- Lorsqu'un projet est installé localement, il devient disponible pour tout autre projet local,
- Il suffit de déclarer une dépendance en utilisant les coordonnées de l'artifact.

Les dépôts Maven

- Un dépôt Maven stocke des artifacts:
  - Stocke un ensemble d'artifacts de projets rangés selon une structure de répertoires correspondant aux coordonnées Maven
  - Dépôt distant par défaut:
    - https://repo.maven.apache.org/maven2/
  - Dépôt local par défaut: $HOME/.m2/repository
  - Possibilité d'ajouter des dépôts
  - Possibilité de créer des dépôts privés

Les dépôts Maven

Les dépôts Maven

- Les plugins et dépendances sont obtenues depuis les dépôts
  - Si un artifact n'est pas dans le dépôt local, recherche dans le dépôt distant
  - Stockage dans le dépôt local pour résolution locale lors du prochain appel
  - Attention: Grand nombre de téléchargements lors des premières utilisations
  - mvn install installe le projet dans le dépôt local
- Si un artifact a le tag SNAPSHOT, vérification à chaque appel qu'une version plus récente n'est pas disponible sur le dépôt distant

Les dépendances

- Afin d'utiliser du code externe sans avoir à le dupliquer, on utilise les dépendances :

```
<dependencies>
   <dependency>
      <groupId>[groupID]</groupId>
      <artifactId>[artifactID]</artifactId>
      <version>[version]</version>
      <scope>[scope]</scope>
   </dependency>
   (...)
</dependencies>
```

- Définition des dépendances dans la section dependencies
- Maven gère les dépendances transitives,
- Les dépendances ont une portée (scope)

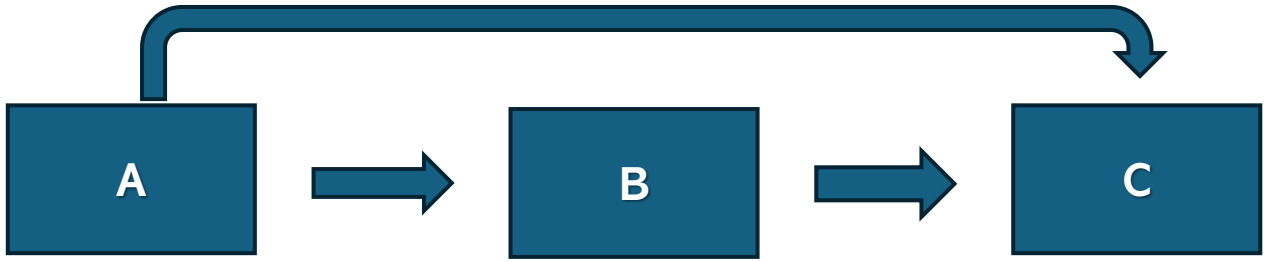
Les dépendances: exemple springboot

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        <version>3.5.6</version>
        <scope>compile</scope>
    </dependency>
</dependencies>
```

- Une dépendance avec la librairie spring boot web est définie, on peut maintenant utiliser les classes inclues dans cette librairie.

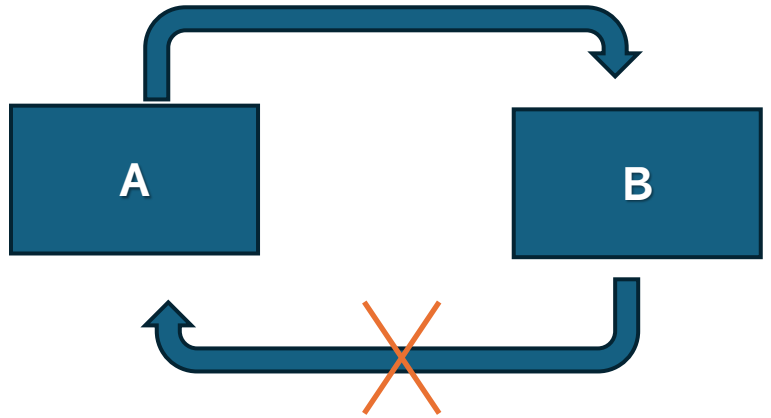
Les dépendances transitives

- Il se passe quoi si une librairie utilise d'autres librairies ?
- A chaque artifact est associé un fichier pom.xml dans lequel sont définies ses dépendances.
  - Si mon projet A dépend de B et que B dépend de C, mon projet dépend de C
  - Maven installera automatiquement C



Les dépendances transitives

- Il n'y a pas de limites au nombre de niveaux
- Il est possible de couper la transitivité grâce au tag:
  - <optional>true</optional>
- Utilisation de la balise <exclusion> pour exclure des dépendances
- Attention aux dépendances cycliques



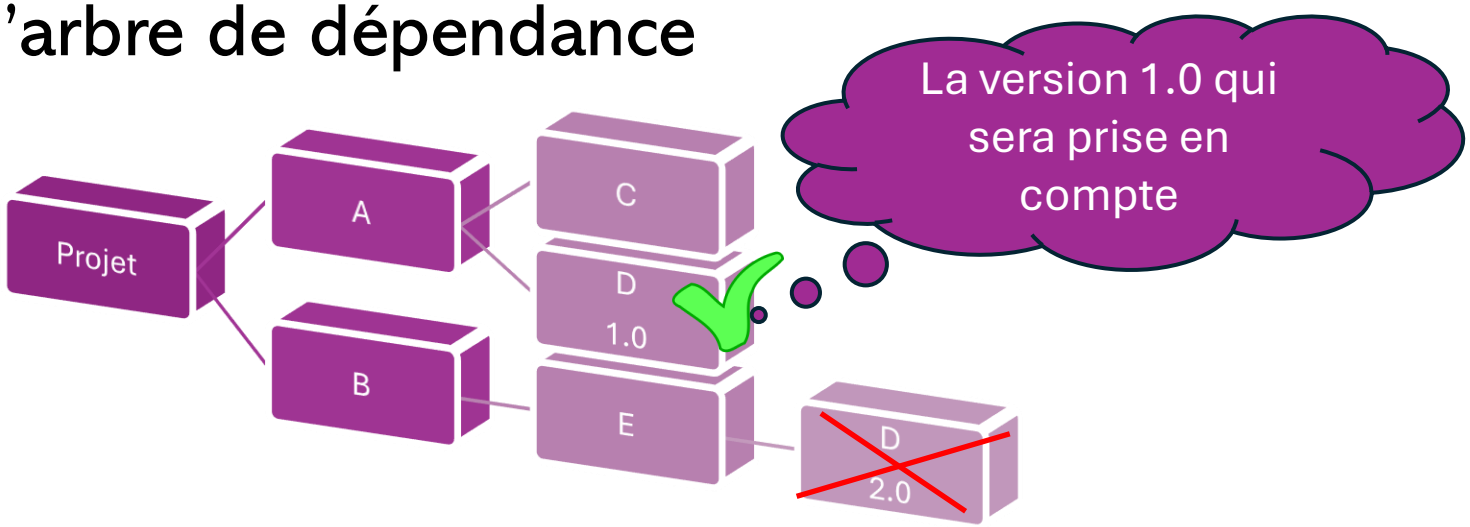
Les dépendances

Commande mvn dependency:tree pour étudier les dépendances en cas de conflits

```
$ mvn dependency:tree
MINGW support requires --add-opens java.base/java.lang=ALL-UNNAMED
MINGW support requires --add-opens java.base/java.lang=ALL-UNNAMED
[INFO] Scanning for projects...
[INFO]
[INFO] -------------< org.sample.devops:authentication-service >--------------
[INFO] Building authentication-service 0.0.1-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- dependency:3.8.1:tree (default-cli) @ authentication-service ---
[INFO] org.sample.devops:authentication-service:jar:0.0.1-SNAPSHOT
[INFO] +- org.springframework.boot:spring-boot-starter-web:jar:3.5.6:compile
[INFO] |  +- org.springframework.boot:spring-boot-starter:jar:3.5.6:compile
[INFO] |  |  +- org.springframework.boot:spring-boot:jar:3.5.6:compile
[INFO] |  |  +- org.springframework.boot:spring-boot-autoconfigure:jar:3.5.6:compile
[INFO] |  |  +- org.springframework.boot:spring-boot-starter-logging:jar:3.5.6:compile
[INFO] |  |  |  +- ch.qos.logback:logback-classic:jar:1.5.18:compile
[INFO] |  |  |  |  \- ch.qos.logback:logback-core:jar:1.5.18:compile
[INFO] |  |  |  +- org.apache.logging.log4j:log4j-to-slf4j:jar:2.24.3:compile
[INFO] |  |  |  |  \- org.apache.logging.log4j:log4j-api:jar:2.24.3:compile
[INFO] |  |  |  \- org.slf4j:jul-to-slf4j:jar:2.0.17:compile
[INFO] |  |  +- jakarta.annotation:jakarta.annotation-api:jar:2.1.1:compile
[INFO] |  |  \- org.yaml:snakeyaml:jar:2.4:compile
[INFO] |  +- org.springframework.boot:spring-boot-starter-json:jar:3.5.6:compile
[INFO] |  |  +- com.fasterxml.jackson.core:jackson-databind:jar:2.19.2:compile
[INFO] |  |  |  +- com.fasterxml.jackson.core:jackson-annotations:jar:2.19.2:compile
[INFO] |  |  |  \- com.fasterxml.jackson.core:jackson-core:jar:2.19.2:compile
[INFO] |  |  +- com.fasterxml.jackson.datatype:jackson-datatype-jdk8:jar:2.19.2:compile
[INFO] |  |  +- com.fasterxml.jackson.datatype:jackson-datatype-jsr310:jar:2.19.2:compile
[INFO] |  |  \- com.fasterxml.jackson.module:jackson-module-parameter-names:jar:2.19.2:compile
[INFO] |  +- org.springframework.boot:spring-boot-starter-tomcat:jar:3.5.6:compile
[INFO] |  |  +- org.apache.tomcat.embed:tomcat-embed-core:jar:10.1.46:compile
[INFO] |  |  +- org.apache.tomcat.embed:tomcat-embed-el:jar:10.1.46:compile
[INFO] |  |  \- org.apache.tomcat.embed:tomcat-embed-websocket:jar:10.1.46:compile
[INFO] |  +- org.springframework:spring-web:jar:6.2.11:compile
[INFO] |  |  +- org.springframework:spring-beans:jar:6.2.11:compile
[INFO] |  |  \- io.micrometer:micrometer-observation:jar:1.15.4:compile
[INFO] |  |     \- io.micrometer:micrometer-commons:jar:1.15.4:compile
[INFO] |  \- org.springframework:spring-webmvc:jar:6.2.11:compile
[INFO] |     +- org.springframework:spring-aop:jar:6.2.11:compile
[INFO] |     +- org.springframework:spring-context:jar:6.2.11:compile
[INFO] |     \- org.springframework:spring-expression:jar:6.2.11:compile
[INFO] \- org.springframework.boot:spring-boot-starter-test:jar:3.5.6:test
[INFO]    +- org.springframework.boot:spring-boot-test:jar:3.5.6:test
[INFO]    +- org.springframework.boot:spring-boot-test-autoconfigure:jar:3.5.6:test
[INFO]    +- com.jayway.jsonpath:json-path:jar:2.9.0:test
[INFO]    |  \- org.slf4j:slf4j-api:jar:2.0.17:compile
[INFO]    +- jakarta.xml.bind:jakarta.xml.bind-api:jar:4.0.2:test
[INFO]    |  \- jakarta.activation:jakarta.activation-api:jar:2.1.4:test
```

Les dépendances transitives

- Si deux librairies utilisent (transitivement) des versions différentes d'une librairie, Maven prend la version la plus proche dans l'arbre de dépendance

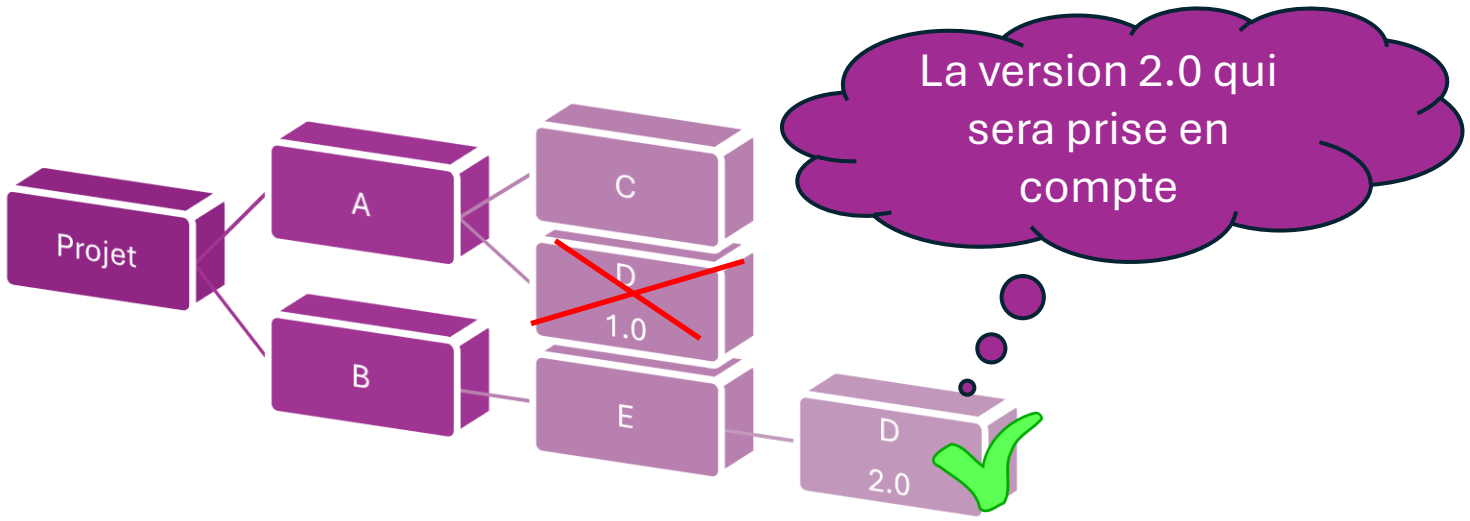


- Si la distance est la même, Maven prend en compte l'ordre des déclarations

Les dépendances transitives

- Dans le cas où l'on ne veut pas importer une dépendance par transitivité, il faut le spécifier grâce au tag <exclusions>

```xml
<dependencies>
    <dependency>
        <groupId>com.groupid</groupId>
        <artifactId>A</artifactId>
        <version>1.0-SNAPSHOT</version>
        <exclusions>
            <exclusion>
            <groupId>com.groupid</groupId>
                <artifactId>D</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
</dependencies>
```



Les dépendances Managements

- Il est possible de forcer Maven à utiliser une version spécifique grâce au <dependencyManagement>

- Si une version est précisée, elle sera écrasée

- Permet de ne plus avoir à spécifier la version de la librairie à utiliser

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>2.0.17</version>
        </dependency>
    </dependencies>
</dependencyManagement>
```

Portée des dépendances - scope

- Chaque dépendance a une portée (<scope>):
  - Permet de couper l'arbre des dépendances
  - Influence le classpath utilisé dans chaque phase
- 6 portées possibles:
  - compile (portée par défaut)
    - Disponibles dans tous les classpaths
    - Dépendances propagées aux projets dépendants (transitivité)
    - Permet la transitivité
  - provided
    - Classpath pour la compilation et les tests
    - Ne permet pas la transitivité
    - Dépendance résolue pour la compilation et le test mais supposée déjà disponible dans le contexte d'exécution

Portée des dépendances - scope

- runtime
  - Classpath pour les tests et au runtime:
  - Nécessaire au runtime mais pas à la compilation (code chargé dynamiquement, JDBC drivers etc…), Permet la transitivité
- test
  - Rend disponible la librairie uniquement en classpath de test
  - Ne permet pas la transitivité
- system
  - Spécifie l'emplacement de la librairie sur le file system avec une propriété <systemPath>
  - Mêmes propriétés que le scope compile
- import
  - Uniquement pour le <dependencyManagement>, importe le dependencyManagement d'un autre pom

Plan du cours

I. Introduction

II. Les principes de Maven

III. Le pom.xml

IV. Les plugins, les tâches, les cycles de vie

V. Coordonnées, dépôts et dépendances

VI. Héritage & Aggrégation

VII. Archetype

Hiérarchie de projets: Héritage

- Utilisation: Factorisation de plusieurs projets avec des configurations similaires
- Définition d'un POM parent
- Modèle objet: un POM hérite des attributs de son parent sauf si il les redéfinit:
  - Les dépendances
  - Les plugins
  - Configuration des plugins
  - . . .
- Arborescence de fichiers
  - pom.xml (parent)
  - my-module/pom.xml

Hiérarchie de projets: Héritage

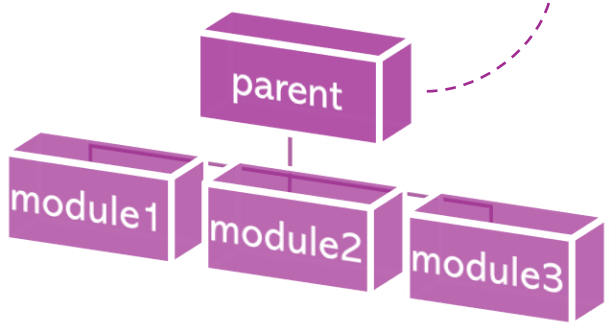
- Comme en Java, Maven implémente un mécanisme d'héritage :
  - Un enfant ne peut avoir qu'un seul parent
  - Un parent peut avoir plusieurs enfants

- Les enfants référencent leur parent de la manière suivante :

```
<parent>
   <groupId>groupIdParent</groupId>
   <artifactId>artifactIdParent</artifactId>
   <version>versionParent</version>
<parent>
```

Attention ! Pour pouvoir être hérité, le type de package doit être défini en « pom »

```
<packaging>pom</packaging>
```



Héritage: Exemple

pom.xml

```
<project>
    <parent>
        <groupId>com.mycompany.app</groupId>
        <artifactId>my-app</artifactId>
        <version>1</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>my-module</artifactId>
</project>
```

pom.xml du parent

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.mycompany.app</groupId>
    <artifactId>my-app</artifactId>
    <version>1</version>
</project>
```

Hiérarchie de projets: Aggrégation

- Utilisation: Regrouper un ensemble de projets à construire ensemble
- Définition d'un POM connaissant un ensemble des modules
- Quand une commande est exécutée sur le POM parent, elle est aussi exécutée sur les modules du parent
  - Le POM parent doit avoir le packaging pom
  - Peut être combiné avec de l'héritage
- Arborescence de fichiers
  - pom.xml (parent)
  - my-module/pom.xml

Aggrégation: Exemple

pom.xml

```
<project>
	<modelVersion>4.0.0</modelVersion>
	<groupId>com.mycompany.app</groupId>
	<artifactId>my-module</artifactId>
	<version>1</version>
</project>
```

pom.xml du parent

```
<project>
	<modelVersion>4.0.0</modelVersion>
	<groupId>com.mycompany.app</groupId>
	<artifactId>my-app</artifactId>
	<version>1</version>
	<packaging>pom</packaging>
	<modules>
		<module>my-module</module>
	</modules>
</project>
```

Plan du cours

I. Introduction

II. Les principes de Maven

III. Le pom.xml

IV. Les plugins, les tâches, les cycles de vie

V. Coordonnées, dépôts et dépendances

VI. Héritage & Aggrégation

VII. Archetype

Création d'un projet - Plugin archetype

- Permet à l'utilisateur de créer un projet à partir d'un template
- Création du projet pour le TD
  - mvn archetype:generate
    -DgroupId=org.sample.devops
    -DartifactId=authentication-service
    -DarchetypeArtifactId=maven-archetype-quickstart
    -Dversion=1.0-SNAPSHOT -DinteractiveMode=false
  - archetype est l'identifiant du plugin
  - generate est l'identifiant du goal

- Exemple de passage de paramètres au plugin via la ligne de commande
  - Sans l'option maven-archetype-quickstart, Maven nous aurait demandé quel type de projet créer

Merci