

# MAVEN

# PLAN

- I. Environnement de Maven
- II. Maven et projets
- III. Maven et IDE
- IV. Conclusion

# Environnement de Maven

- Présentation
- Le maven build
- Architecture de maven
- P.O.M et Goals
- Découpage en module
- Gestion des versions
- Conventions et standards

# Maven et projets

- Installation
- Le cycle de vie du projet
- Principales phases du projet maven
- Les fichiers de configurations
- Les archetypes
- Gestion de dépendances
- Les repositories
- Les plugins maven

# Maven et IDE

- Intégration avec Eclipse
- Importer un projet maven avec Eclipse
- Plugin m2e

# Environnement de Maven

PARTIE I

# Environnement de Maven

○ Présentation

# Présentation

- Un outil qui automatise la construction d'un projet (build)
- Un ensemble de standards et conventions
- Une collection de composants interdépendants
- Une manière de construire, tester et déployer des projets

# Présentation

- Automatisation des tâches répétitives (compilation, tests unitaires et déploiement )
- Gérer les dépendances vis-à-vis des bibliothèques nécessaires au projet
- Générer des documentations concernant le projet
- Il est extensible grâce à un mécanisme de plugins qui permettent d'ajouter des fonctionnalités.

# Présentation

- Réécriture du cœur de Maven V3 (plugin m2eclipse, qualité du code)
- Rétrocompatibilité
- Amélioration de la verbosité de son code XML

# Environnement de Maven

- Le maven build

# Le maven build

- Reconstruction automatique des applications
- Valider l'intégration des modifications et exécuter une série de tâches
- Tests unitaires, vérifications de normes, documentation, ...
- Assurer la qualité du code dans le temps
- Automatisation (toutes les semaines, jours, heures, commits, ...)

# Le maven build

- Chaque développeur avait sa propre approche du build
- Eviter l'intégration manuelle pour construire un projet
- Faciliter les tâches rébarbatives
- Automatisation (documentation, tests, rapports d'exécution (métriques), déploiement)

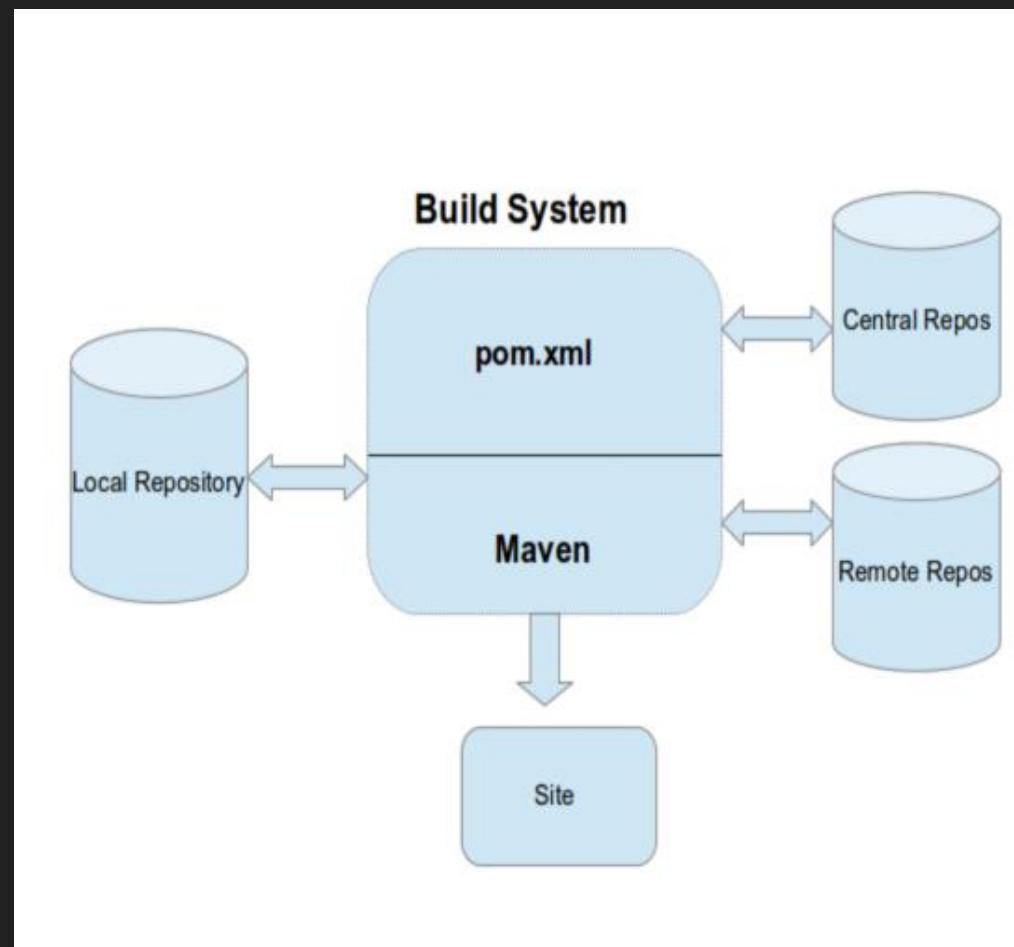
# Le maven build

- Un build est réussi quand :
  - Réalisé sur une machine dédiée
  - Référentiel contenant un projet « complet »
  - Les tâches exécutées produisent un rapport disponible

# Environnement de Maven

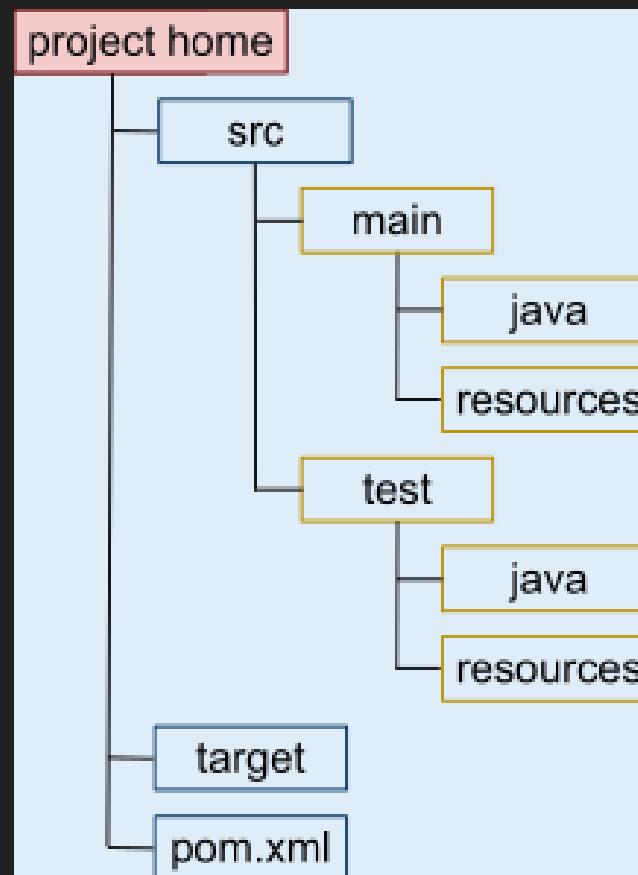
- L'Architecture de maven

# Architecture de Maven



# Architecture de Maven

- Le répertoire « target » contient les fichiers générés par le build (résultat de compilation, résultat de tests, artefact produit du build, ...)



# Architecture de Maven

## Vocabulaire

- **POM** : Project Object Model (descripteur de projet maven)
- **Artefact** : Fichier résultat du build d'un projet
- **Group** : Espace de nom permettant le regroupement d'artefacts
- **Version** : Identification d'une version d'un artefact
- **Repository** : Référentiel (dépôt) d'artefacts
- **Plugin** : Composant attaché dans le processus de build
- **Archétype** : Modèle (prototype) de projet
- **Goal** : Objectif du build

# Environnement de Maven

- P.O.M et Goals

# P.O.M et Goals

```
<project>
    <!-- model version is always 4.0.0 for Maven 2.x POMs -->
    <modelVersion>4.0.0</modelVersion>

    <!-- project coordinates, i.e. a group of values which
        uniquely identify this project -->

    <groupId>com.mycompany.app</groupId>
    <artifactId>my-app</artifactId>
    <version>1.0</version>

    <!-- Library dependencies -->

    <dependencies>
        <dependency>

            <!-- coordinates of the required library -->

            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>

            <!-- this dependency is only used for running and compiling tests -->

            <scope>test</scope>

        </dependency>
    </dependencies>
</project>
```

# P.O.M et Goals

## Le Project Object Model

- Il contient les informations concernant le projet : nom, scope, n° version, etc...
- Les plugins qui apportent des possibilités au build du projet
- Obéit à une structure qui exécute des tâches pré-définies
- Réalise un ensemble de tâches dont certaines standards
- Décrit des dépendances du projet

# P.O.M et Goals

## Le Project Object Model

- Est orienté objet
- Peut faire référence à un POM parent
- Le POM fils peut surcharger les valeurs héritées du POM parent

# P.O.M et Goals

- Le développeur déclare dans le POM toute les actions effectuées (build, test, déploiement, profils, etc...)
- Extensible : Prévoit l'ajout de fonctionnalités via les plugins
- XML : Maven fournit un langage commun à tous
- Le POM doit être exhaustif et doit pouvoir être interprété par un tiers

# P.O.M et Goals

## Fonctionnement du POM

- Ordonnancement des tâches
- Récupération par internet des plugins nécessaires, dépendances...
- Détection des erreurs (dépendances erronées)
- Ordres (ligne de commandes) : « mvn package » crée le livrable du projet
- Maven est plutôt dans l'esprit de configuration que d'écriture

# P.O.M et Goals

## Héritage entre fichiers POM

**Il est fréquent d'avoir des projets avec des valeurs communes**

- « groupId » identique
- Dépendances et versions communes
- Configurations de plugins communes

# P.O.M et Goals

## Héritage entre fichiers POM

- **Le pom « parent » = héritage de valeurs communes**
- On évite ainsi la recopie de valeurs dans chaque projet
- Héritage simple : un seul pom parent par projet Maven (i.e. par pom.xml)

```
<project ...>
  <parent>
    <groupId>com.company</groupId>
    <artifactId>projetWeb</artifactId>
    <version>1.0.2</version>
  </parent>
```

# P.O.M et Goals

- **Un pom parent peut avoir lui aussi un pom parent**
- **Une pratique usuelle est d'avoir :**
- Un pom grand parent qui contient les infos générales de l'entreprise
  - Nom du projet, des responsables, logins/pwd, adresses des serveurs...
- Des poms parents pour chaque type de projet
  - Un pom parent pour les war, un pour les jar ...
- Et enfin un pom.xml pour chaque projet
- **Maven contient lui-même un pom parent implicite : le super POM**

# P.O.M et Goals

- **Un pom parent possède un packaging particulier : pom**

Équivalent à la notion de classe abstraite des langages de programmation

- **Un tel projet ne produit pas d'artefact**

Il sert seulement à définir des éléments communs (valeurs, dépendances, ...)

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.apache.maven2.proficio</groupId>
    <artifactId>proficio</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>pom</packaging
```

# P.O.M et Goals

## Un Goal (Objectif)

- C'est une tâche à exécuter, une commande à lancer
- Il donne un ordre à maven
- Souvent sous la forme mvn « commande » Ex : maven compile

# P.O.M et Goals

## Un Goal (Objectif)

- Différent des systèmes de construction traditionnels (moins de script : Ant)
- Passage du « procédural » au « déclaratif »
- Permet de se concentrer sur la logique de développement
- L'infrastructure du build est gérée par Maven
- La description du build est gérée par le développeur

# P.O.M et Goals

## Un Goal (Objectif)

- Possibilité de lancer plusieurs goals :

mvn <goal 1> <goal 2> <goal 3> Exemple : mvn clean compile

- Plugins : Les goals

mvn <nom-plugin>:<libellé goal> Exemple : mvn eclipse:eclipse ou mvn gwt:eclipse

# Le cycle de vie du projet

- validate : valider le projet, voir s'il est correct
- compile : compilation du projet
- test : compiler les tests unitaires et les faire passer
- package : Former le livrable (jar/war)
- integration-test : Tests qui doivent passer sur un environnement dédié
- verify : Vérifier la qualité du livrable suivant une série de critères
- install : Déposer le package livrable sur le Repository Local
- deploy : Publier le package final sur un environnement partagé

# Environnement de Maven

- Découpage en module

# Découpage en module

- **La complexité des gros projets nécessite de les décomposer**

C'est la notion classique de module (ou sous projet)

On peut ainsi spécialiser les équipes de développement, offrir de la réutilisation, diminuer la complexité de maintenance, ...

Découpage classique : modèle métier, présentation, persistance, infrastructure

- **Problème : comment builder un gros projet en une commande ?**

Il faut pouvoir déclarer les éléments (modules) de ce projet

Le build du projet lancera le build de ses modules

# Découpage en module

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.company.formation</groupId>
    <artifactId>ProjetModule</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>pom</packaging>
    <name>Mon projet en différents modules</name>
    ...
    <modules>
        <module>app</module>
        <module>webapp</module>
        <module>backoffice</module>
    </modules>
    ...
</project>
```

# Environnement de Maven

- Gestion des versions

# Gestion des versions

- 1iere solution : On fixe des propriétés dans le pom parent et on hérite dans les pom enfants

Artifacts specified in the <dependencies> section **will ALWAYS be included** as a dependency of the child module(s).

# Gestion des versions

- 2ieme solution : <dependencyManagement> dans le pom parent

Artifacts specified in the <dependencyManagement> section, **will only be included** in the child module if they were also specified in the <dependencies> section of the child module itself.
- Pour le pom enfant :
  - Pas besoin de spécifier la version à utiliser
  - Ce numéro de version a été défini au niveau parent

# P.O.M Parent

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.8.1</version>
      <scope>compile</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

# P.O.M Enfant

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Même notion pour les plugins « `<pluginManagement/>` »

# Environnement de Maven

- Conventions et standards

# Conventions et standards

- Maven propose une structure, une convention. On bénéficie vraiment de la standardisation dans ce cas
- Un projet maven est assez préconfiguré depuis le début
- Maven utilise beaucoup de valeurs par défaut
- L'ajout d'une fonctionnalité est relativement simple à configurer
- Le but est de rester dans les conventions Maven

# Conventions et standards

- Structuration standard des sources du projet
- Un projet maven fournit une seule sortie : l'artefact
- Répertoires organisés de la même manière sur tous les projets
- L'ordre des tâches du build est constante
- Une telle structure facilite le travail et la collaboration

# Maven et projets

PARTIE II

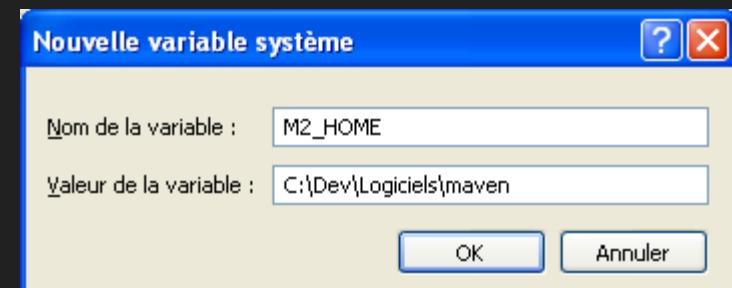
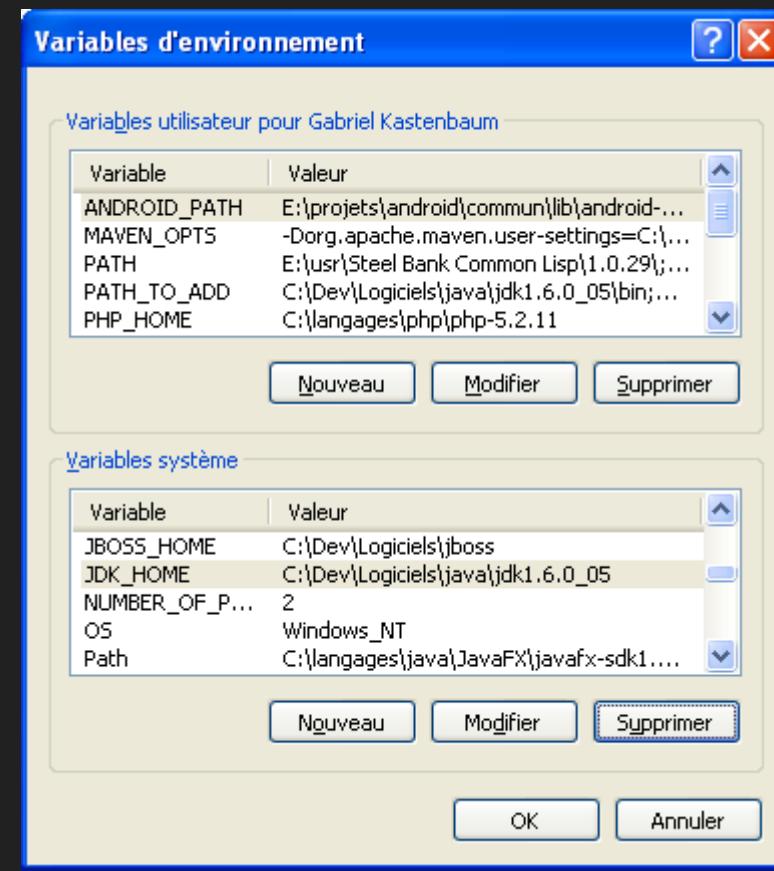
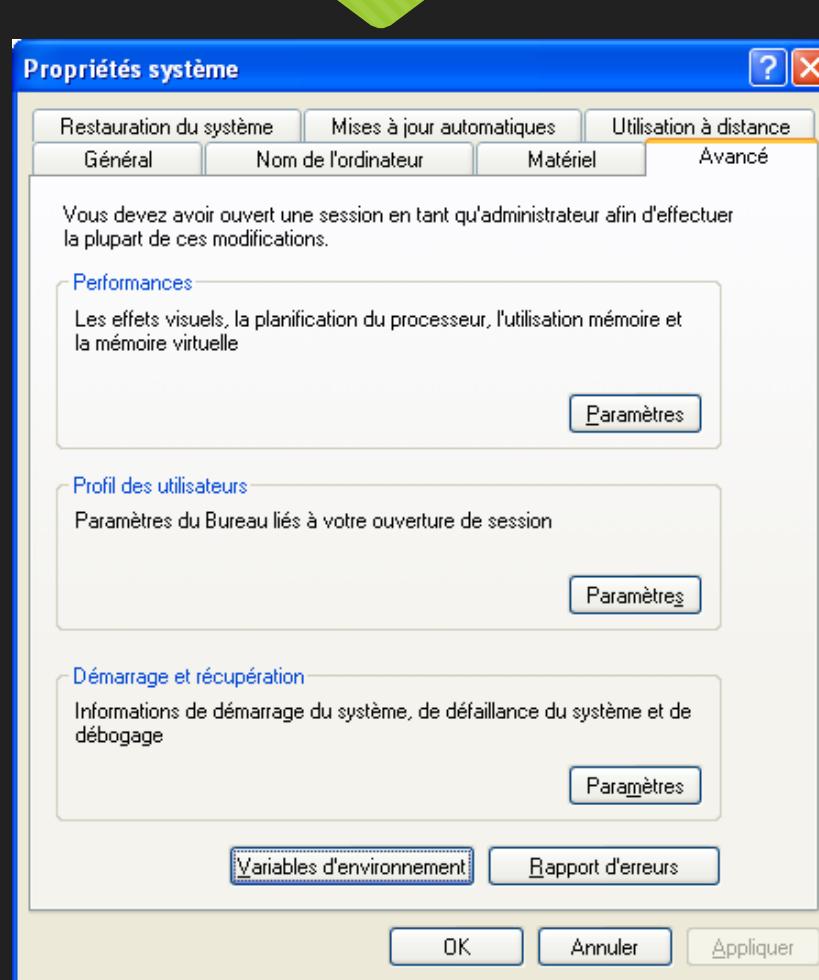
# Maven et projets

## ○ Installation de Maven

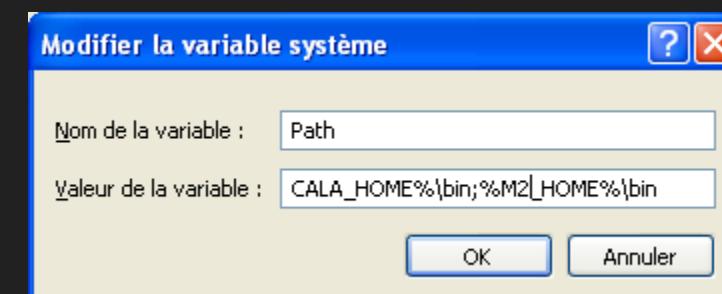
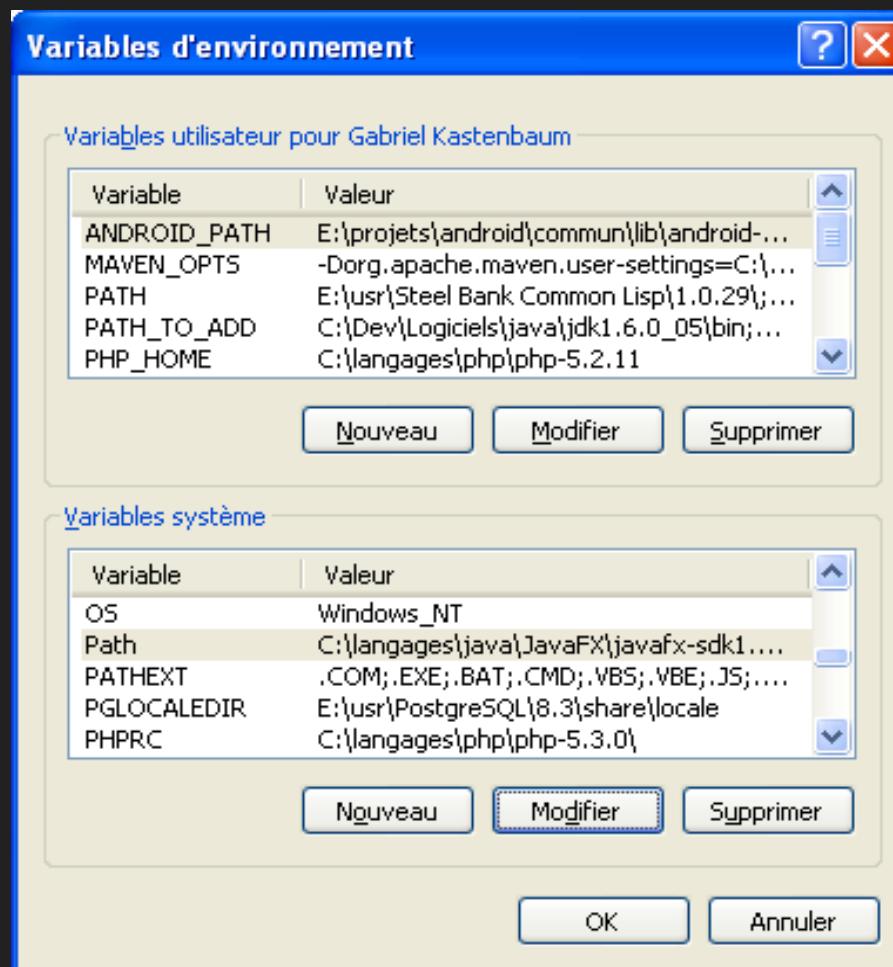
# Installation de Maven

- Télécharger Maven depuis le site officiel <http://maven.apache.org/download.cgi>
- Prendre la version : Binary zip archive sur le poste de travail:
- S'assurer d'avoir installé et configuré un JDK, lancez « java –version » en ligne de commande
- Configurer la variable d'environnement Path
- Ajouter le chemin vers le répertoire « bin » de maven
- En général, on déclare une variable M2\_HOME (voir ci-après)
- Commande « mvn –version » pour vérifier l'installation de maven
- L'intégration dans les IDE permet de se passer de cette installation manuelle

# Installation de Maven



# Installation de Maven



```
c:\tmp\testmaven>mvn -v
Apache Maven 2.2.1 (r80177; 2009-08-06 21:16:01+0200)
Java version: 1.6.0_20
Java home: C:\langages\java\jdk1.6.0_20\jre
Default locale: fr_FR, platform encoding: UTF-8
OS name: "windows xp" version: "5.1" arch: "x86" Family: "windows"
c:\tmp\testmaven>
```

# Maven et projets

- Le cycle de vie du projet

# Le cycle de vie du projet

- **Le build Maven suit un cycle ordonné et prédéfini**
- Des goals différents y sont rattachés par la suite, suivant le projet

# Le cycle de vie du projet

- validate : valider le projet, voir s'il est correct
- compile : compilation du projet
- test : compiler les tests unitaires et les faire passer
- package : Former le livrable (jar/war)
- integration-test : Tests qui doivent passer sur un environnement dédié
- verify : Vérifier la qualité du livrable suivant une série de critères
- install : Déposer le package livrable sur le Repository Local
- deploy : Publier le package final sur un environnement partagé

# Le cycle de vie du projet

- L'exécution d'une tâche maven exécute les tâches en amont  
Quand on lance « mvn install » : Maven lance auparavant : validate, compile, test, package ...  
Puis install
- Il existe des tâches hors « cycle de vie »  
clean, site, assembly:assembly

# Le cycle de vie du projet

## Goals les plus souvent utilisés

- `clean:clean` = Supprime tous les artefacts et fichiers intermédiaires créés
- `eclipse:eclipse` = Génère les fichiers de configuration pour Eclipse
- `javadoc:javadoc` Génère la documentation Java du projet
- `checkstyle:checkstyle` = Génère le rapport sur le style de codage du projet
- `site:site` = Crée un site web de documentation du projet
- `clean install` = Recompile le projet depuis le début et générer l'artefact

# Maven et projets

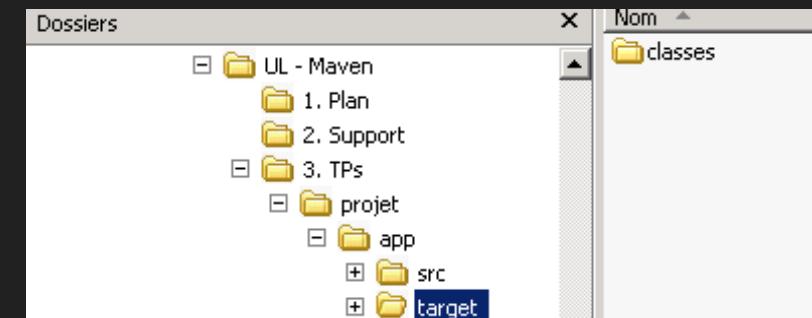
- Principales phases du projet maven

# Principales phases du projet maven

## Compilation

- Se positionner sur le répertoire créé et lancez « mvn compile »
- Un répertoire « target » est créé pour accueillir les fichiers .class

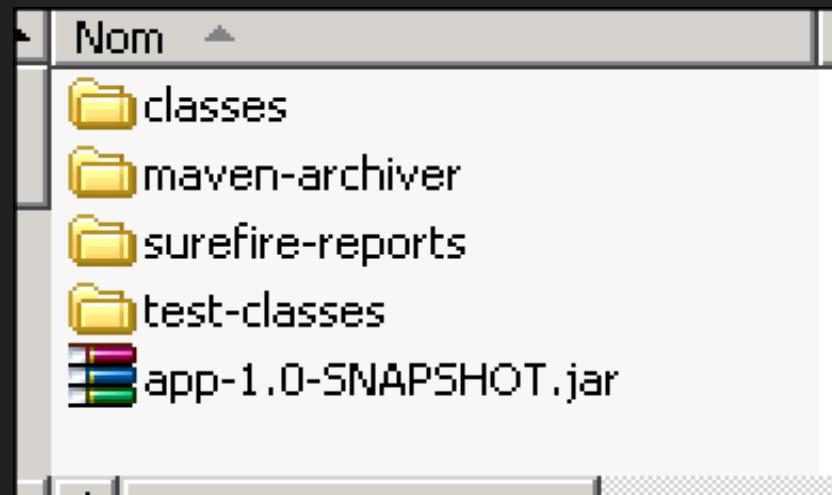
```
c:\C:\WINDOWS\system32\cmd.exe
Downloading: http://repo1.maven.org/maven2/org/codehaus/plexus/plexus-utils/1.0.4/plexus-utils-1.0.4.jar
159K downloaded
Downloading: http://repo1.maven.org/maven2/org/codehaus/plexus/plexus-compiler-api/1.5.3/plexus-compiler-api-1.5.3.jar
19K downloaded
Downloading: http://repo1.maven.org/maven2/org/codehaus/plexus/plexus-compiler-javac/1.5.3/plexus-compiler-javac-1.5.3.jar
13K downloaded
[INFO] [compiler:compile]
[INFO] Compiling 1 source file to D:\Formation Oxiane 2007\ure\UL - Maven\5. TP\projet\app\target\classes
[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 20 seconds
[INFO] Finished at: Tue Jan 29 17:22:07 CET 2008
[INFO] Final Memory: 3M/8M
[INFO] -----
```



# Principales phases du projet maven

## Packaging de l'application

- Se positionner sur le répertoire et lancez « mvn package »
- L'artefact est injecté dans le répertoire « target »



# Maven et projets

- Les fichiers de configurations

# Les fichiers de configurations

## O Fichier principal « pom.xml »

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.company.formation</groupId>
  <artifactId>my-app</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>

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

### <modelVersion>

Version de pom

Permet la cohabitation avec d'anciens projets maven

### Identification du projet

Ce projet identifié « my-app » dans le groupe « com.company.formation » produit un « jar » à la version « 1.0- SNAPSHOT »

### Identification de dépendance

Ce projet nécessite junit (un artefact identifié « junit » dans le groupe « junit » à la version « 3.8.1 », uniquement pour compiler et exécuter ses tests unitaires

# Les fichiers de configurations

- Fichier de configuration « settings.xml » global  
%MVN\_HOME%/conf  
C'est le fichier par défaut pour tous les utilisateurs
- Fichier de configuration « settings.xml » de l'utilisateur  
%USER\_HOME%/.m2/settings.xml (par défaut)  
Surcharge du global par l'utilisateur

# Maven et projets

- Les archetypes

# Les archetypes

**Un archétype est un modèle (pattern) constitué**

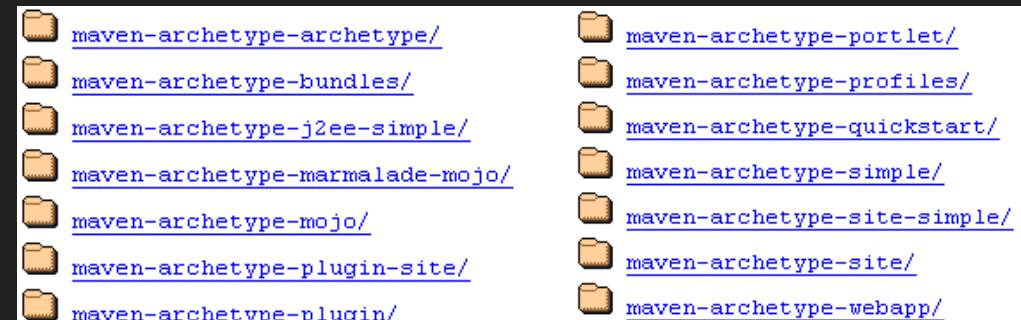
D'un descripteur (archetype.xml)

De fichiers qui seront copiés d'un pom

**Il existe des centaines d'archétypes déjà construits**

Lancer la commande « mvn archetype:generate »

Indiquez le nom d'archétype avec « –DarchetypeArtifactId="nom" »



# Les archetypes

- **Le plugin « archetype » propose des « modèles » de projets**
- Exemple : pour créer un projet web, jar, etc.
- **Il est même possible de créer ses propres archétypes**
- <http://maven.apache.org/guides/mini/guide-creating-archetypes.html>

# Les archetypes

## ○ Pour créer un projet web simple (1/2)

Choisir son « groupId » : com.company.formation.maven

Choisir son « artifactId » : mavenapp

Prendre un « archetypeArtifactId » simple : maven-archetype-webapp

## ○ Lancer la commande sur une console

```
mvn archetype:generate  
  -DgroupId=com.company.formation.maven  
  -DartifactId=mavenapp  
  -DarchetypeArtifactId=maven-archetype-webapp
```

# Les archetypes

- Pour créer un projet web simple (2/2)
- Un répertoire mavenapp est créé
- Avec son pom.xml et ses dépendances
- Avec l'arborescence web

# Maven et projets

- Gestion de dépendances

# Gestion de dépendances

- **Nous apercevons une dépendance vers junit**
- Habituellement, nous téléchargeons junit.jar
- Puis nous le plaçons dans un répertoire du projet
- Et enfin nous l'ajoutons dans le chemin de compilation du projet
- **Maven va faire cela automatiquement**
- En recherchant la dépendance dans ses « repository »

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

# Gestion de dépendances

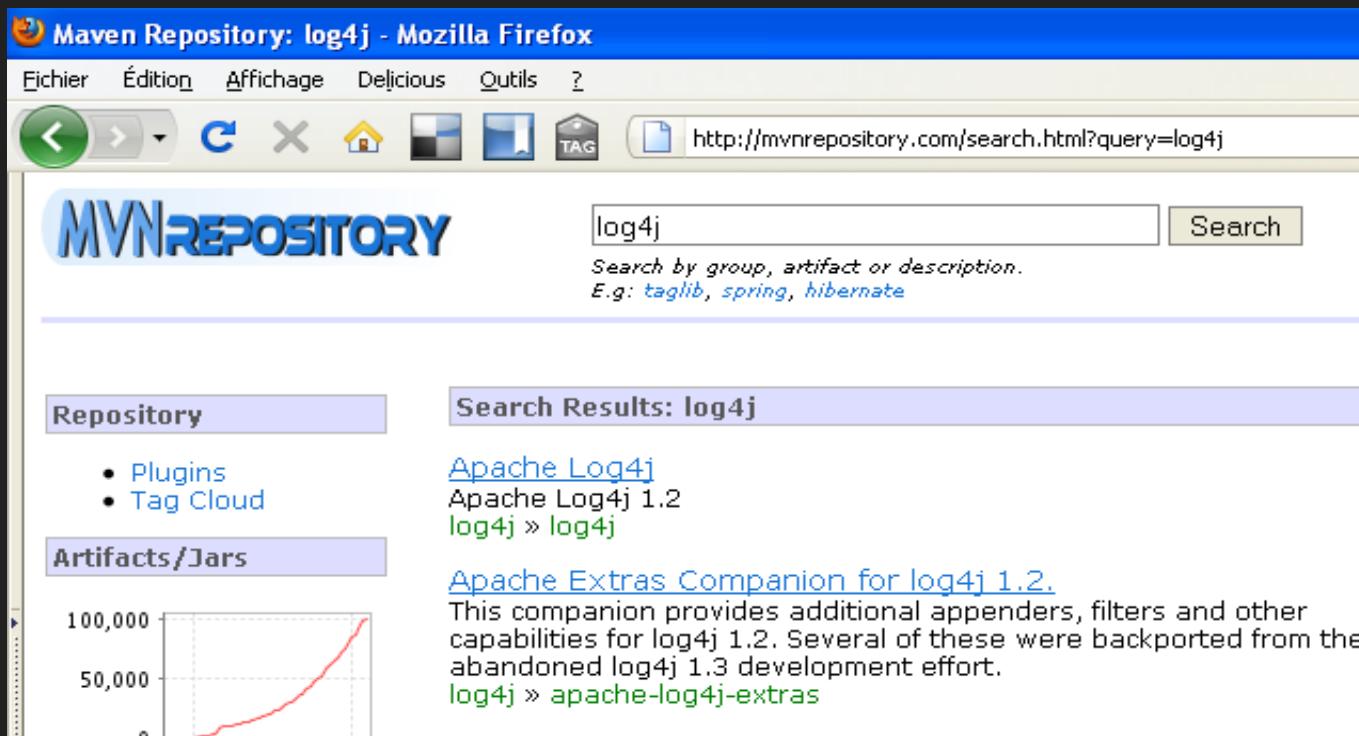
- Une dépendance est une référence à un artefact dans un repository
- On n'indique pas où sont les dépendances, on ne fait que les déclarer
- Maven possède un moteur de résolution de dépendances
- Il va chercher les dépendances dans des référentiels (repository ou dépôt)

# Gestion de dépendances

- **Repositories et artefacts dans Maven**
- Si la dépendance n'est pas trouvée en local, elle est recherchée en distant
- Elle est alors copiée dans le repository local
- **Maven porte une liste de référentiels distants**
- Par défaut <http://repo.maven.apache.org/maven2>
- On peut indiquer une liste dans le fichier de configuration settings.xml
- On peut aussi ajouter des repositories dans le pom.xml

# Gestion de dépendances

- Plusieurs sites permettent de rechercher une dépendance
- <http://mvnrepository.com> <http://search.maven.org>



# Gestion de dépendances

- On choisit la version que l'on veut dans son projet
- Le site donne le morceau de code xml pour déclarer la dépendance

Apache Log4j 1.2  
tags: [logging](#)

Available versions		
Version	Download	Details
<a href="#">1.2.16</a>	<a href="#">Binary (471 KB)</a>	<a href="#">Details</a>
<a href="#">1.2.15</a>	<a href="#">Binary (383 KB)</a>	<a href="#">Details</a>
<a href="#">1.2.14</a>	<a href="#">Binary (359 KB)</a>	<a href="#">Details</a>
<a href="#">1.2.13</a>	<a href="#">Binary (350 KB)</a>	<a href="#">Details</a>
<a href="#">1.2.12</a>	<a href="#">Binary (350 KB)</a>	<a href="#">Details</a>
<a href="#">1.2.11</a>	<a href="#">Binary (343 KB)</a>	<a href="#">Details</a>
<a href="#">1.2.9</a>	<a href="#">Binary (345 KB)</a>	<a href="#">Details</a>
<a href="#">1.2.8</a>	<a href="#">Binary (345 KB)</a>	<a href="#">Details</a>

POM Dependency

Apache Maven

```
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.14</version>
</dependency>
```

# Gestion de dépendances

- **Une dépendance est utile suivant l'environnement**
- Un jar peut être utile en test mais pas utile en production
- **On peut préciser le scope sur chaque dépendance (ci-après)**

```
<project>
<!-- ... -->
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>3.8.1</version>
        <scope>runtime*</scope>
    </dependency>
</dependencies>
</project>
```

\* This scope indicates that the dependency is not required for compilation, but is for execution.

# Gestion de dépendances

- **Compile (par défaut, niveau utilisé si non précisé)**
- La dépendance est nécessaire pour la compilation et l'exécution.
- Dépendances fournies dans le packaging final (par exemple le war)
- **Provided**
- La dépendance est nécessaire mais ne fait pas partie du « packaging » final
- **Runtime**
- Pas nécessaire pour la compilation, mais l'est pour l'exécution
- Ex : implémentations d'API externes, utilisées à l'exécution (Pilote JDBC,...)
- **Test**
- Dépendances requises uniquement pour les tests unitaires
- **System**
- Similaire à « provided » mais la dépendance doit être fournie manuellement

# Gestion de dépendances

## ○ Dépendance transitive

- **Une des fonctionnalités puissante de maven**
- Une dépendance peut amener d'autres dépendances plus profondes
- **Exemple**
- Projet C dépend de Projet B qui, lui même, dépend de Projet A
- Projet C peut utiliser une classe de Projet A
- On décide de supprimer la dépendance vers Projet B
- Le projet est alors en erreur
- **Depuis la version 2.0.9, il est possible de contrôler cela finement**

# Gestion de dépendances

## ○ Résoudre un conflit de dépendance

- Exclure une dépendance avec la balise « exclusion »

```
...  
  <dependency>  
    <groupId>org.codehaus.plexus</groupId>  
    <artifactId>plexus-containers-default</artifactId>  
    <version>1.0-alpha-9</version>  
    <exclusions>  
      <exclusion>  
        <groupId>org.codehaus.plexus</groupId>  
        <artifactId>plexus-utils</artifactId>  
      </exclusion>  
    </exclusions>  
  </dependency>  
...
```

```
...  
  <dependencies>  
    <dependency>  
      <groupId>org.codehaus.plexus</groupId>  
      <artifactId>plexus-utils</artifactId>  
      <version>1.1</version>  
      <scope>runtime</scope>  
    </dependency>  
  </dependencies>  
...
```

# Maven et projets

- Les repositories

# Les repositories

## Les repositories Maven

A la première utilisation de Maven, un repository local est créé. Il se remplit de dépendances au fur et à mesure des utilisations. Le repository est créé par défaut dans :

`%HOMEPATH%/.m2/repository`

# Les repositories

## Recherche et emplacement

- Génération d'un chemin de recherche sur le repository local :
- Exemple : pour le groupId : junit, artifactId : junit, version : 3.8.1, il cherchera : %HOMEPATH%/.m2/repository/junit/junit/3.8.1/junit-3.8.1.jar
- Dépendance absente en local → Recherche sur les repository distant

# Maven et projets

- Les plugins maven

# Les plugins maven

- Un noyau fournissant les fonctionnalités de base pour gérer un projet
- Des plugins qui implémentent les tâches de construction du projet
- Les plugins sont là pour implémenter les actions à exécuter

Ex: Créer des fichiers (JAR, WAR, ...), compiler, créer les tests, créer la documentation, etc.

- Elle permet la modularité des configurations Maven

# Les plugins maven

- **Le comportement d'un plugin est paramétrable**
- A l'aide de paramètres que l'on spécifie dans sa déclaration dans le POM
- Il faut regarder la documentation à chaque fois qu'on utilise un plugin
- **Un des plugins les plus simples de Maven est le plugin Clean**
- Fonction : supprimer les objets créés par Maven : artefacts, fichiers compilés  
Suivant le standard Maven, ces objets sont placés sous le répertoire « target »
- « mvn clean:clean » nettoie le répertoires « target »

# Maven et IDE

PARTIE III

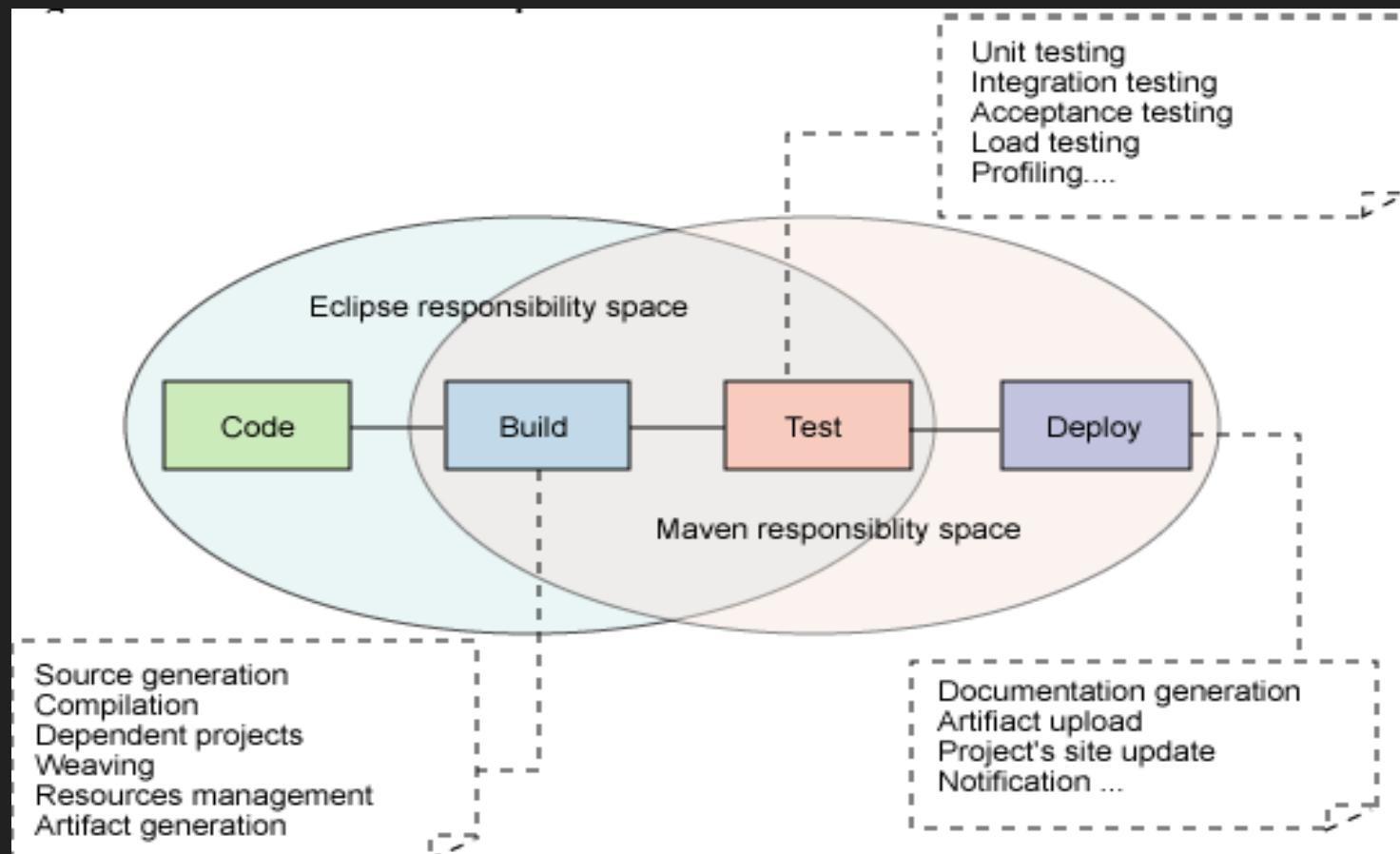
# Maven et IDE

- Intégration avec Eclipse

# Intégration avec Eclipse

- **Le développement de projet se fait généralement avec un IDE**
- Integrated Development Environment : Eclipse, Netbeans, IntelliJ, ...
- **Un développeur passe 90% de son temps sur son IDE**
- L'IDE est utilisé pour compiler, packager, déployer en test

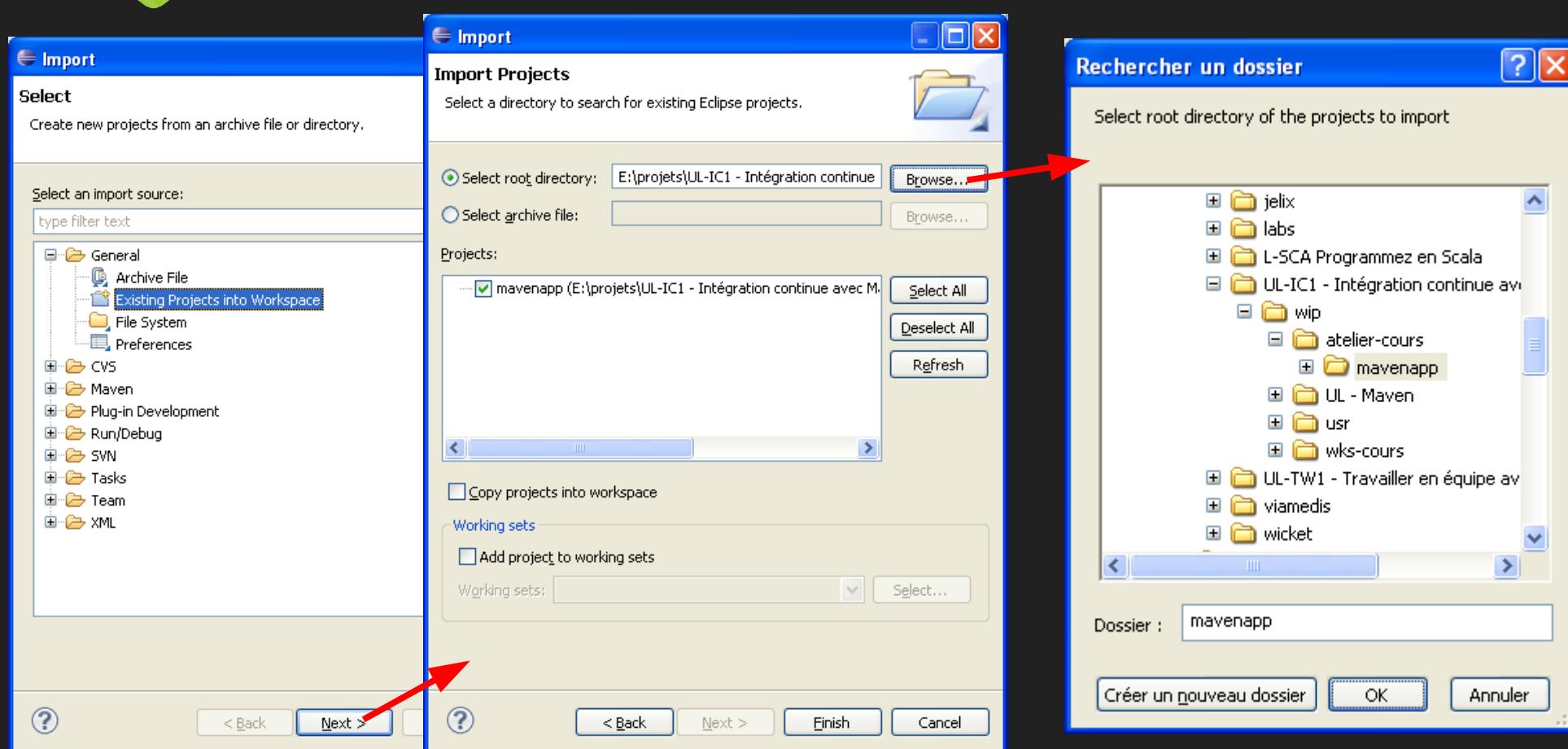
# Intégration avec Eclipse



# Maven et IDE

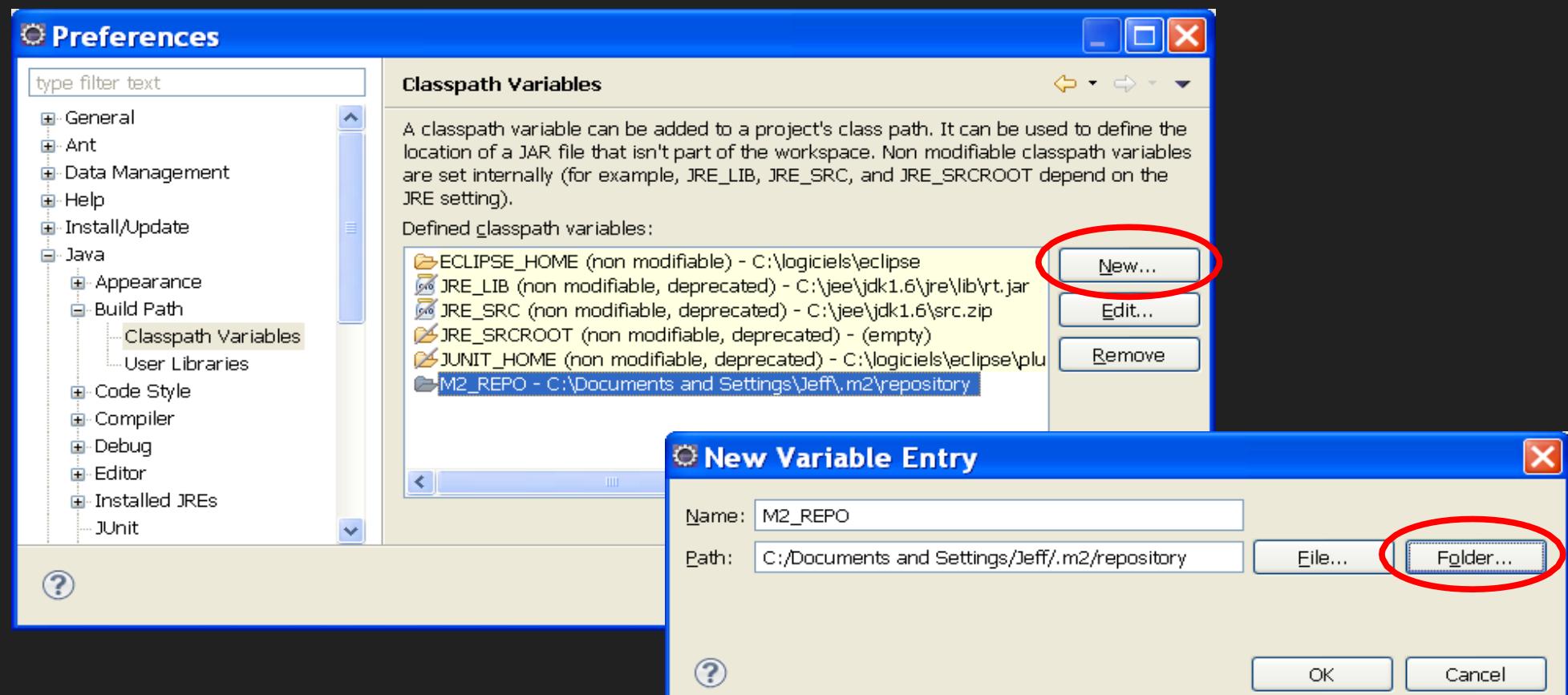
- Importer un projet maven avec Eclipse

# Importer un projet maven avec Eclipse



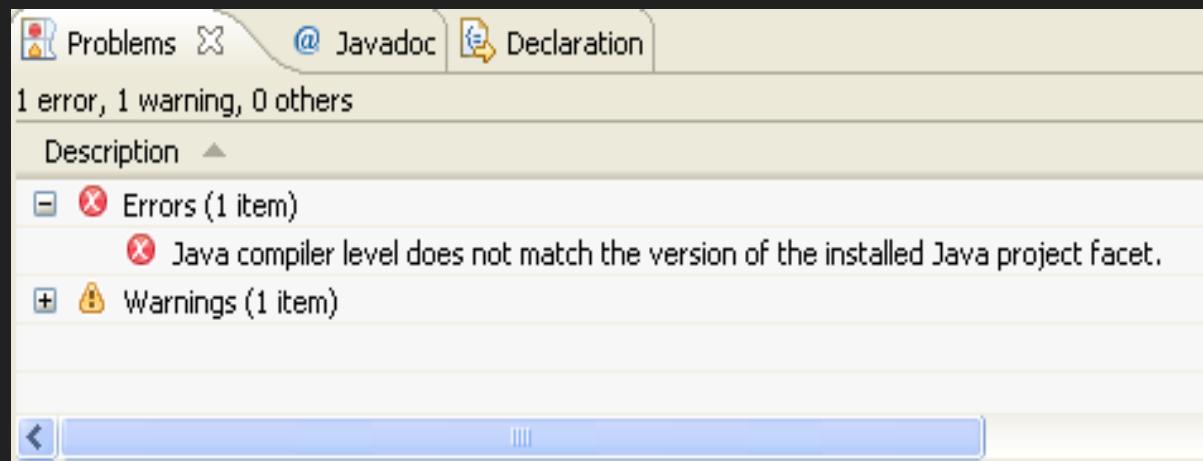
# Importer un projet maven avec Eclipse

Il peut subsister des erreurs si le projet utilise des dépendances



# Importer un projet maven avec Eclipse

Le problème vient de la version de Java considérée par défaut par Maven

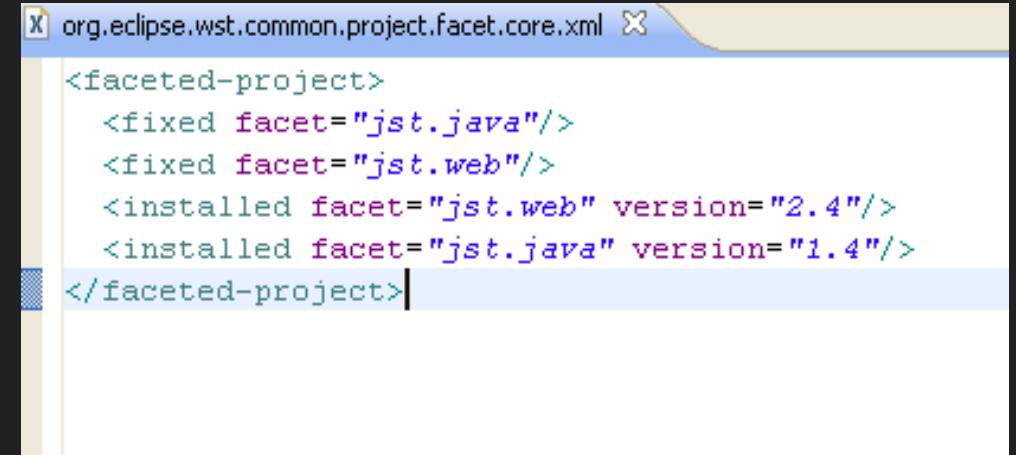


# Importer un projet maven avec Eclipse

**Une mauvaise solution consiste à modifier le fichier incriminé au sein du projet. Mauvaise solution :**

Le fichier sera écrasé lors du prochain « mvn eclipse:eclipse »

On exécute cette commande régulièrement (à chaque modification du POM)



The screenshot shows a code editor window titled "org.eclipse.wst.common.project.facet.core.xml". The content of the editor is an XML snippet defining facets for a project:

```
<faceted-project>
    <fixed facet="jst.java"/>
    <fixed facet="jst.web"/>
    <installed facet="jst.web" version="2.4"/>
    <installed facet="jst.java" version="1.4"/>
</faceted-project>
```

# Importer un projet maven avec Eclipse

La bonne solution : indiquer à maven que le projet est en Java 8 par exemple

```
...
<build>
  <pluginManagement>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.1</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </pluginManagement>
</build>
...
```

# Conclusion

## **Un outil incontournable dans l'entreprise**

Centralisation des éléments

Standardisation

Réutilisation

## **Une grande richesse mais qui nécessite un apprentissage**

Changement de culture

Changement des habitudes

## **Concurrencé mais les concurrents se doivent d'être compatibles**

La structure des référentiels reste le cœur de l'édifice