

# TP 7

## Introduction à l'intégration continue

### Objectifs du TP

Avant de travailler sur des TP de test logiciel, vous allez utiliser dans ce premier TP des outils liés au processus de développement logiciel, en l'occurrence Maven et des outils d'intégration continue souvent associés tels que Sonar et Jenkins.

Le but de ce TP est donc de :

- **Comprendre le fonctionnement de Maven**
- **Utiliser les artefacts**
- **Configurer un projet IntelliJ avec Maven**
- **Générer des rapports Maven**
- **Utiliser des outils d'intégration continue**

### Liens utiles

- Site de Maven : <http://maven.apache.org/>
- Plugin Checkstyle :  
<http://maven.apache.org/plugins/maven-checkstyle-plugin/>
- FAQ maven developpez.com : <http://java.developpez.com/faq/maven/>

### Environnement

Selon le 3ième lien donné ci-dessus, Maven est essentiellement un outil de gestion et de compréhension de projet. Maven offre des fonctionnalités de :

- Construction, compilation
  - Documentation
  - Rapport
  - Gestion des dépendances
  - Gestion des sources
  - Mise à jour de projet
  - Déploiement
-

Utiliser Maven consiste à définir dans chaque projet à gérer un script Maven appelés POM : *pom.xml*. Nous allons voir dans ce TP qu'un POM permet de définir des dépendances, des configurations pour notamment construire, tester, mettre en paquet des artefacts logiciels (exécutables, tests, documentations, archives, etc.). Pour cela, Maven récupère sur des dépôts maven les outils dont il a besoin pour exécuter le POM. Utiliser Maven requière donc : une (bonne) connexion à Internet car il télécharge beaucoup de choses ; de l'espace disque pour la même raison. Les artefacts qu'il télécharge sont originellement stockés dans le dossier *.m2* dans votre home-dir. Ce dossier contient également le fichier de configuration Maven : *settings.xml*. La première étape consiste donc à configurer Maven pour changer l'endroit où les artefacts téléchargés seront stockés afin d'éviter des problèmes d'espace disque. Pour ce faire, utilisez le fichier *settings.xml* suivant à mettre donc dans le dossier *.m2* :

```
<?xml version="1.0" encoding="UTF-8"?>
<settings>
  <localRepository>w:\mavenrepository</localRepository>
  <offline>false</offline>
</settings>
```

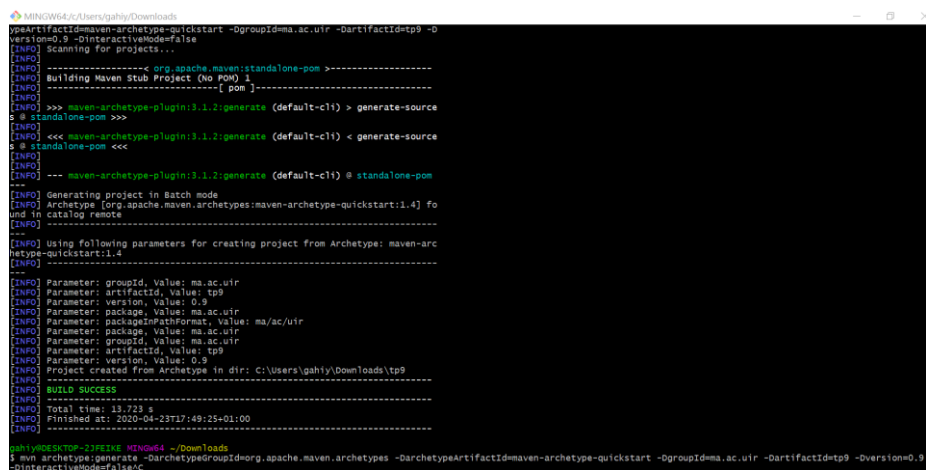
## Partie 1 : Utilisation de maven

Pour initialiser un projet Java basique, vous pouvez utiliser l'archetype maven : *maven-archetype-quickstart*. Un archetype est une sorte de modèle dont le but est de faciliter la mise en place de projets. Vous avez juste à fournir un *groupid* et un *artefactid*. Un *groupid* est l'identifiant du groupe de projets dont le projet fait partie. Un *artefactid* est l'identifiant du projet au sein de son groupe (depuis la console).

```
mvn archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -
DarchetypeArtifactId=maven-archetype-quickstart -DgroupId=ma.ac.uir -
DartifactId=tp9 -Dversion=0.9 -DinteractiveMode=false
```

Vous obtenez alors la structure de projet suivante :

```
-- src
|-- main
|  |-- java
|  |   |-- [your project's package]
|  |   |-- App.java
|  |-- test
|  |   |-- java
|  |   |-- [your project's package]
|  |   |-- AppTest.java
-- pom.xml
```



```

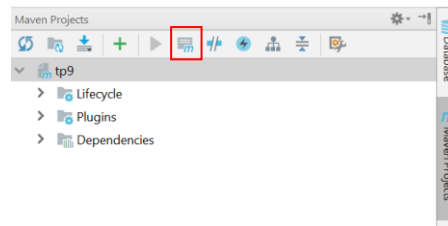
C:\Users\gahy\Downloads
mvn archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -DarchetypeArtifactId=maven-archetype-quickstart -DgroupId=ma.ac.uir -DartifactId=tp9 -Dversion=0.9 -DinteractiveMode=false
[INFO] Scanning for projects...
[INFO]
[INFO] -----< org.apache.maven:standalone-pom >-----
[INFO] Building Maven Stub Project (No POM) 1
[INFO] -----[ pom ]-----
[INFO]
[INFO] >>> maven-archetype-plugin:3.1.2:generate (default-cli) > generate-source
$ @ standalone-pom >>>
[INFO] <<< maven-archetype-plugin:3.1.2:generate (default-cli) < generate-source
$ @ standalone-pom <<<
[INFO]
[INFO] --- maven-archetype-plugin:3.1.2:generate (default-cli) @ standalone-pom
---
[INFO] Generating project in Batch mode
[INFO] Archetype [org.apache.maven.archetypes:maven-archetype-quickstart:1.4] found in catalog remote
[INFO]
[INFO] Using following parameters for creating project from Archetype: maven-archetype-quickstart:1.4
[INFO]
[INFO] Parameter: groupId, Value: ma.ac.uir
[INFO] Parameter: artifactId, Value: tp9
[INFO] Parameter: version, Value: 0.9
[INFO] Parameter: package, Value: ma.ac.uir
[INFO] Parameter: packageInPathFormat, Value: ma/ac/uir
[INFO] Parameter: package, Value: ma.ac.uir
[INFO] Parameter: groupId, Value: ma.ac.uir
[INFO] Parameter: artifactId, Value: tp9
[INFO] Parameter: version, Value: 0.9
[INFO] Project created from Archetype in dir: C:\Users\gahy\Downloads\tp9
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 11.723 s
[INFO] Finished at: 2020-04-23T17:49:25+01:00
[INFO]
C:\Users\gahy\Downloads
$ mvn archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -DarchetypeArtifactId=maven-archetype-quickstart -DgroupId=ma.ac.uir -DartifactId=tp9 -Dversion=0.9 -DinteractiveMode=false

```

## Partie 2 : Configuration d'IntelliJ

Eclipse ne supporte pas Maven par défaut, et nécessite un plugin externe assez capricieux. Nous allons donc utiliser un autre IDE : IntelliJ. Normalement tout est déjà disponible sur les machines à votre disposition. Ouvrez IntelliJ, puis faites « File » et « Open... ». Choisissez le répertoire contenant le projet maven créé dans la Partie 1, et validez.

IntelliJ possède nativement une façon d'appeler Maven facilement. Pour cela, faites « View », « Tool Windows », « Maven Projects ». Un panneau s'affiche à droite avec la liste des projets maven détectés, et l'ensemble des actions Maven qu'il est possible de faire sur le projet.



Effectuer ces actions en double cliquant dans ce panneau est équivalent à taper en ligne de commande « mvn <action> ». (clean, package, site...)

Pour exécuter les commandes Maven, il faut choisir « Execute maven goal » dans les options du Maven projects.

## Partie 3 : Génération de rapports

### Générer la javadoc

Ajoutez des commentaires dans le code de votre projet. Ajoutez le code suivant dans le *pom.xml* de votre projet.

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-project-info-reports-plugin</artifactId>
      <version>2.6</version>
    </plugin>
  </plugins>
</reporting>
```

Puis lancez : *mvn site*. Ce goal crée un site Web pour votre projet. Par défaut, les goals maven générant des fichiers dans le dossier *target* se trouvant au même niveau que le fichier *pom.xml*. Allez dans le dossier *target/site* et ouvrez le fichier *index.html*. Vous pouvez regarder la Javadoc générée en cliquant sur *Project reports*.

### Valider la qualité du code avec le plugin *checkstyle*

Ajoutez à la section <plugins> dans <reporting> le plugin *checkstyle* :

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-checkstyle-plugin</artifactId>
  <version>2.11</version>
</plugin>
```

Lancez *mvn clean site* (le goal *clean* vide le dossier *target*). Une nouvelle section *Checkstyle* a été ajouté

dans *Project reports*.

## Rapport croisé de source

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-jxr-plugin</artifactId>
  <version>2.3</version>
</plugin>
```

Lien utile : <http://maven.apache.org/plugins/maven-jxr-plugin/>.

Désormais vous pouvez passer du rapport *CheckStyle* au code source en cliquant sur le numéro de ligne associé au commentaire *CheckStyle*.

## Connaître l'activité du projet

But : avoir des informations sur les fichiers modifiés et leurs auteurs. Pour cela vous allez versionner en local votre projet avec *git*. Aller dans le dossier de votre projet, puis :

- créez un dépôt git : *git init* ;
- le dossier *target* ne doit pas être commité, de même que les fichiers créés par IntelliJ : créez un fichier « .gitignore », et mettez y :

```
target
*.iml
.idea
```
- versionnez tous les autres fichiers : *git add --all* . Notez qu'on se permet de faire ça uniquement parce qu'on a bien pris soin de créer un *.gitignore* qui empêche de versionner les fichiers inutiles ! Plus généralement, on utilise très peu le *--all*, à part éventuellement pour un premier commit, comme ici.
- committez : *git commit -m "first commit"*.

Ajoutez ensuite dans la section <plugins> de <reporting> du pom le plugin *changelog* :

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-changelog-plugin</artifactId>
  <version>2.2</version>
</plugin>
```

Ajoutez dans le bloc <project> le lien vers votre projet :

```
<scm>
  <connection>scm:git:file:///c:/Users/Downloads/tp9/</connection>
</scm>
```

Lancez *mvn clean site*. Le dossier */target/site* contient maintenant trois rapports d'activité :

- *changelog* : rapport indiquant toutes les activités sur le SCM.
- *dev-activity* : rapport indiquant par développeur le nombre de commits, de fichiers modifiés. Pour que ce rapport fonctionne, il faut déclarer les développeurs du projet dans le bloc <project>.
- *file-activity* : rapport indiquant les fichiers qui ont été révisés



Allez à l'adresse <http://localhost:9000/>. Loguez-vous avec le login *admin* et le mot-de-passe *admin*. Allez dans *Quality Profiles* et changez les règles de qualités utilisées puis relancez *mvn sonar:sonar*.

Baladez-vous dans Sonar pour explorer ces différentes fonctionnalités.

Essayez de créer d'autres classes avec du contenu et relancer la commande *mvn sonar:sonar*.

## Intégration avec Jenkins

Télécharger Jenkins à partir du lien suivant :

<https://www.jenkins.io/download/thank-you-downloading-windows-installer-stable>

Démarrer l'installation en lançant l'installateur windows : *jenkins.msi*

Allez dans votre navigateur : <http://localhost:8080/>. (le mot de passe du compte admin est récupérable depuis le chemin : *C:\Program Files (x86)\Jenkins\secrets\initialAdminPassword*)



Par défaut, Jenkins ne contient pas le plugin pour gérer des repository Git, Il vous faut installer le plugin “*Git Plugin*”.

	Allows to ignore GitHub branches/pull requests/commits based on the content of commit messages	0.4.0
Misc (git)		
<input checked="" type="checkbox"/>	<a href="#">Git</a> This plugin integrates <a href="#">Git</a> with Jenkins.	4.2.2
<input type="checkbox"/>	<a href="#">Git Automerger</a> Tool for merging release branches into master.	0.5.1
<input type="checkbox"/>	<a href="#">Git Forensics</a> Jenkins plug-in that mines and analyzes data from a Git repository.	0.7.0
<input type="checkbox"/>	<a href="#">pry</a> This plugin is a sample to explain how to write a Jenkins plugin.	1.1
Misc (github)		

Il faudra ajouter Maven

Filtre: <input type="text" value="Maven Integration"/>		
Mises à jour	Disponibles	Installés
Installer ↓		
<input checked="" type="checkbox"/>	<a href="#">Maven Integration</a> This plug-in provides, for better and for worse, a deep integration of Jenkins and Maven: Automatic triggers between projects depending on SNAPSHOTS, automated configuration of various Jenkins publishers (JUnit, ...).	3.6
<input type="checkbox"/>	<a href="#">Pipeline Maven Integration</a> This plugin provides integration with Pipeline, configures maven environment to use within a pipeline job by calling sh mvn or bat mvn. The selected maven installation will be configured and prepended to the path.	3.8.2
Update information obtained: 13 mn ago		
Vérifier		

Vous devez configurer Maven (voir *Configure System*).

**Configuration des projets Maven**

MAVEN_OPTS global	<input type="text"/>	?
Local Maven Repository	Default (~/.m2/repository)	?
Nb d'exécuteurs	2	
Libellés	<input type="text"/>	
Utilisation	Utiliser ce noeud autant que possible	?
Période d'attente	5	?
Nombre de tentatives de checkout SCM	0	
<input type="checkbox"/> Restreindre le nommage de projet		

## Configuration globale des outils

**Configuration Maven**

Fournisseur de réglages par défaut	Fichier de réglages Maven sur le système de fichiers	
File path	file://C:/apache-maven-3.5.3	?
Fournisseur de réglages globaux par défaut	Utiliser les réglages globaux Maven par défaut	

**JDK**

Installations JDK	<div>Ajouter JDK</div>	
	JDK	
	Nom	jdk1.8.0_161
	<input checked="" type="checkbox"/> Install automatically	?

Enregistrer

Appliquer

Pour créer un job, vous aurez besoin d'indiquer à Jenkins où se trouve votre *repository* git. Il y a deux façons de procéder :

- (a) Étant donné que git est un système décentralisé, votre répertoire de projet un *repository* git fonctionnel, même s'il est seulement accessible localement. Vous pouvez donc indiquer à Jenkins un chemin local vers votre répertoire de projet.
- (b) Ceci étant, dans le cadre d'un développement collaboratif, on utilise souvent un repository « central » par convention (sur un serveur géré par le groupe de développement, ou bien sur github, etc.). Dans ce cas, on indiquera à Jenkins d'utiliser ce repo principal pour l'intégration continue. Vous pouvez donc si vous le souhaitez mettre votre code sur github de cette manière :
- créez un nouveau *repository* via l'interface github
  - liez votre dépôt local au distant : `git remote add origin git@github.com:login/nomRepo.git`
  - mettez votre code sur ce dépôt : `git push origin master`

De retour dans Jenkins créez un job (le menu Nouveau Item), et définissez le *repository* git à utiliser soit (a) en indiquant un chemin local (e.g. `/home/toto/monProjet`), soit (b) en indiquant l'url du repository git que vous avez préalablement créé sur github (i.e. `https://github.com/login/nomRepo.git`) et enfin définissez les goals maven pour le build : pour commencer *clean package install*.

Une fois tout cela configuré, lancez un build.

Dans l'historique des builds, un icône bleu doit apparaître à la fin de la construction pour désigner la construction correcte de l'artefact (bleu car le développeur de Jenkins est Japonais et au Japon le bleu équivaut au vert chez nous, d'ailleurs un plugin Jenkins existe pour afficher des icône verte et non bleue...). Cliquez ensuite sur le lien sous « Construction du module », les artefacts créés par jenkins en utilisant le POM du projet sont visibles dont un jar.



