

Installation de maven

Table des matières

Propos.....	3
Pré-requis.....	3
Installer Maven.....	3
Installer m2Eclipse (uniquement pour Luna).....	7
Configurer maven externe comme étant le maven d'Eclipse.....	7
Migrer un projet Classique vers un projet Maven.....	9
Créer le projet vide.....	9
Importer le projet depuis Eclipse.....	9
Copier vos fichiers.....	13
Vérifier que votre build fonctionne correctement.....	13
Créer un projet java exécutable.....	14
Création du projet sous eclipse.....	15
Test du projet	20
Référence le jar dans un autre projet.....	22

Propos

Ce document n'est utile qu'à partir du cours 6 de JEE. Il propose un manuel d'installation et de configuration de maven.

Pré-requis

Il faut déjà avoir installé les éléments évoqués dans la documentation se trouvant avec le cours 1.

Installer Maven

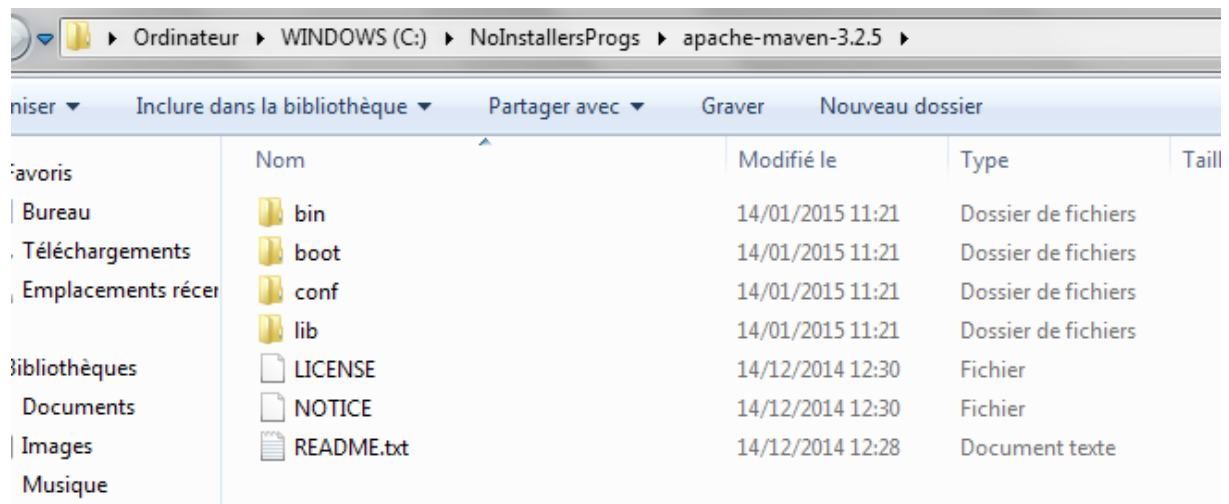
Rendez-vous à l'url suivante :

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

Vous trouverez un zip comme sur la capture ci-dessous. La version la plus récente de maven au moment de l'écriture de cette documentation est la 3.2.9 :

	Link	Checksum
Binary tar.gz archive	apache-maven-3.3.9-bin.tar.gz	apache-maven-3.3.9-bin.tar.gz.md5
Binary zip archive	apache-maven-3.3.9-bin.zip	apache-maven-3.3.9-bin.zip.md5
Source tar.gz archive	apache-maven-3.3.9-src.tar.gz	apache-maven-3.3.9-src.tar.gz.md5
Source zip archive	apache-maven-3.3.9-src.zip	apache-maven-3.3.9-src.zip.md5

Il suffit ensuite de dézipper maven dans un répertoire de votre choix :

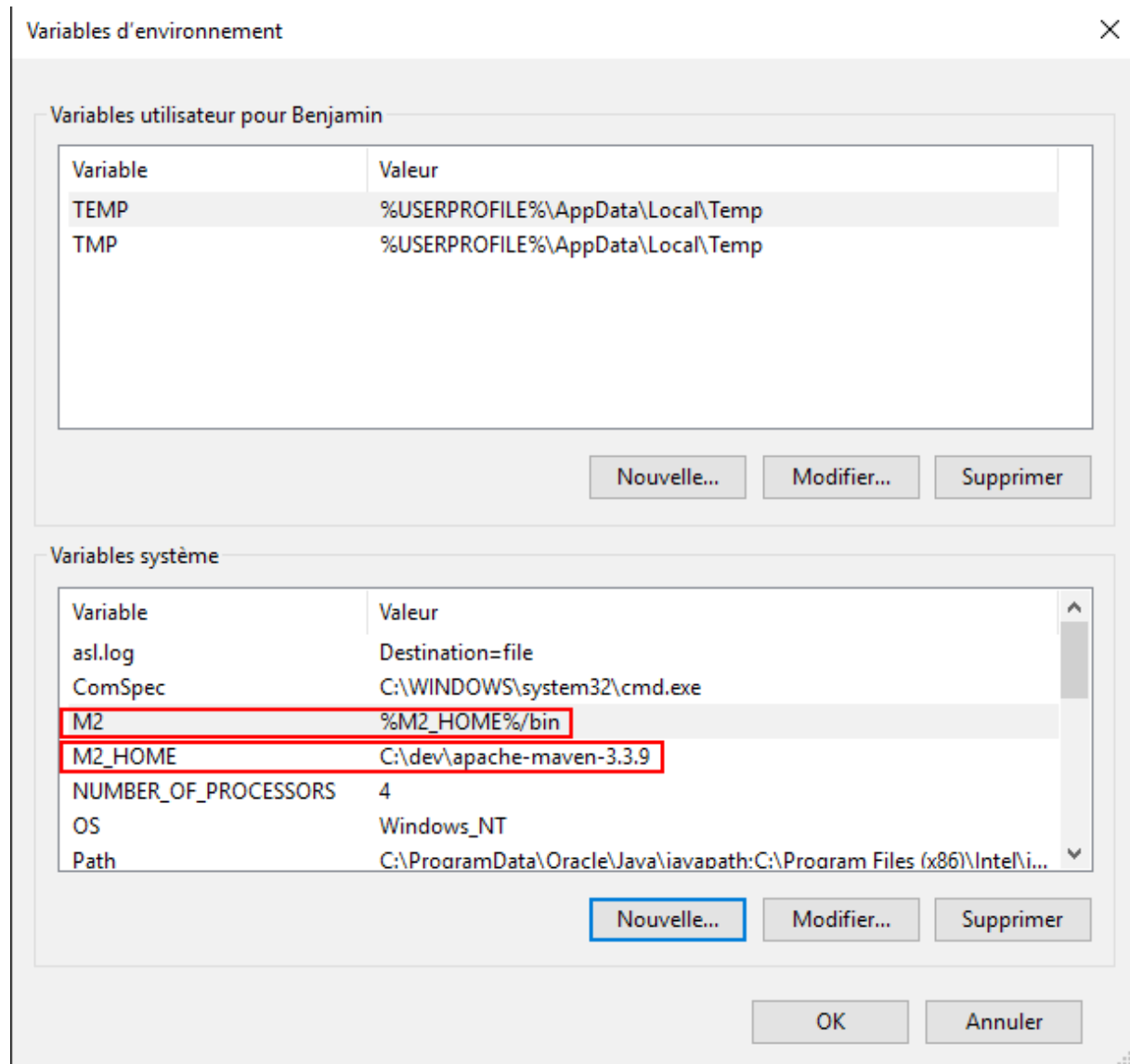


Pour commencer nous allons créer deux variables d'environnement :

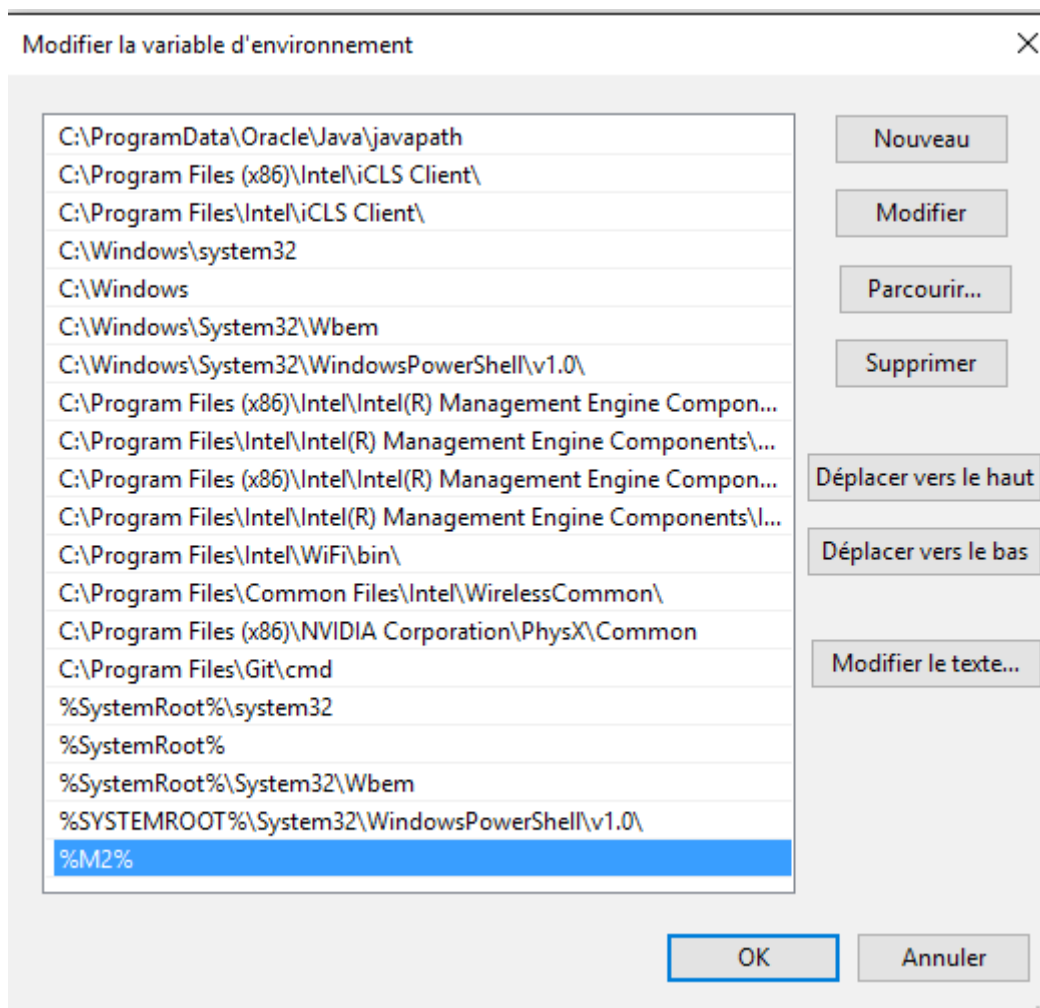
- M2_HOME : avec le chemin vers le repertoire Apache ;
- M2 : %M2_HOME%/bin

Faites démarrer -> clic droit sur ordinateurs -> paramètres systèmes avancés.

Les captures suivantes sont issus de Windows 10, elles diffèrent un peu de 7.

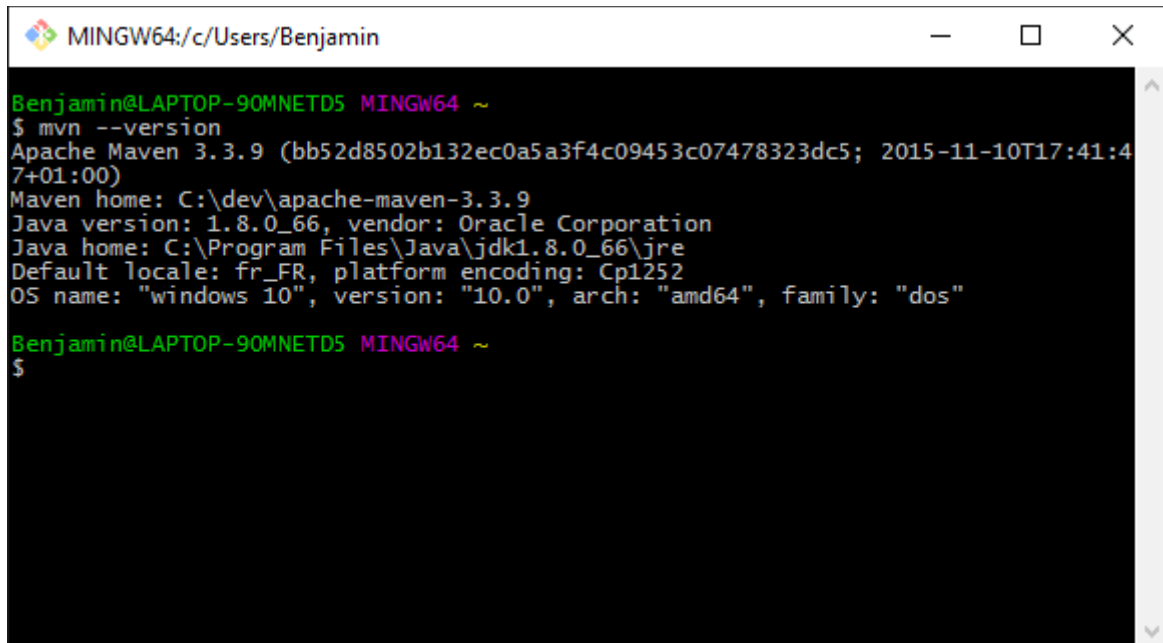


Il faut ensuite ajouter dans la variable PATH, la référence à M2 pour pouvoir utiliser facilement maven en ligne de commande :



Sous windows 7, quand vous modifierez la variable PATH, vous aurez une suite de valeur séparés par des ;. Ajouter le %M2% à la fin en prenant garde de ne pas supprimer les variables existantes !

Une fois ceci fait, vérifiez votre installation en faisant un `mvn --version` dans une invite de commande :



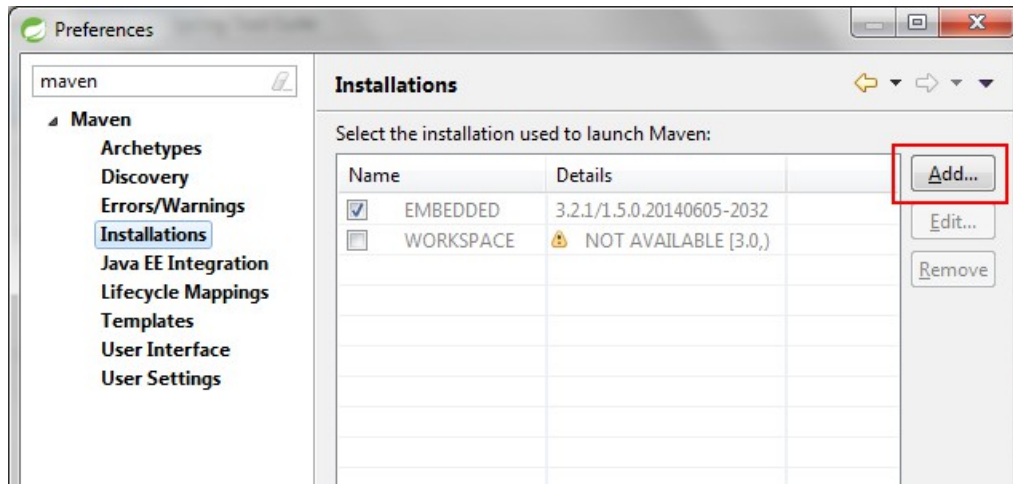
```
MINGW64:/c/Users/Benjamin

Benjamin@LAPTOP-90MNETD5 MINGW64 ~
$ mvn --version
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-10T17:41:47+01:00)
Maven home: C:\dev\apache-maven-3.3.9
Java version: 1.8.0_66, vendor: Oracle Corporation
Java home: C:\Program Files\Java\jdk1.8.0_66\jre
Default locale: fr_FR, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "dos"

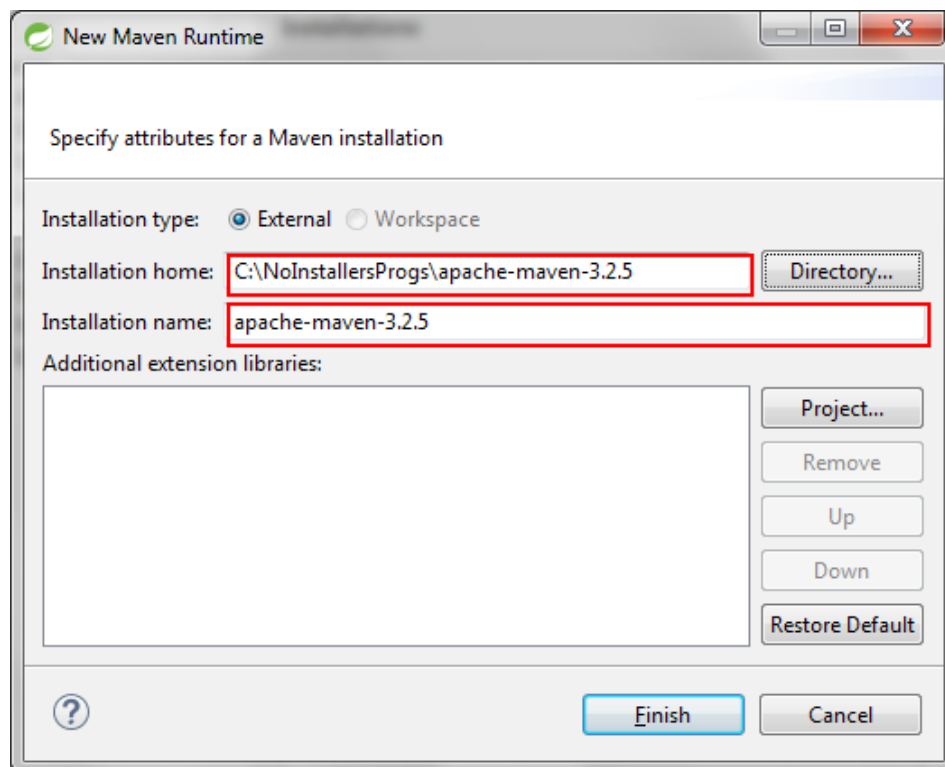
Benjamin@LAPTOP-90MNETD5 MINGW64 ~
$
```

Configurer maven externe comme étant le maven d'Eclipse

Puisque nous utilisons maven en ligne de commande, il est important que celui-ci soit le même pour Eclipse. Pour pouvoir utiliser un maven personnalisé, il faut se rendre dans windows->préférences->maven->installations. Vous obtiendrez l'écran suivant. Cliquez sur le bouton « add » :

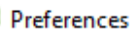


Vous obtiendrez un écran, où vous devrez sélectionner l'endroit où vous avez dézippé votre installation de maven.



Cliquez sur « finish ». Vous revenez sur l'écran de préférence.

N'oubliez pas de sélectionner votre installation dans la liste :



Archetypes

Archetypes

Discovery

Errors/Warnings

Installations

Java EE Integration

Lifecycle Mappings

Templates

User Interface

User Settings

[illegible]Add...

Edit...

Remove

Note: Embedded runtime is always used for dependency resolution

Restore Defaults

Apply



OK

Cancel

Migrer un projet Classique vers un projet Maven

Voici les différentes étapes permettant de migrer correctement un projet existant sous maven.

Créer le projet vide

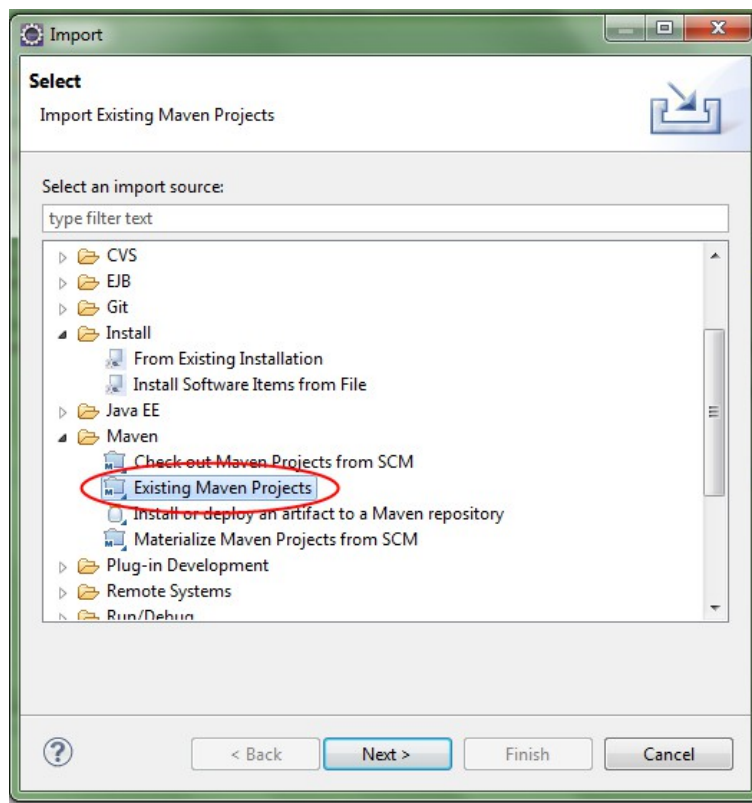
Plutôt que de convertir le projet existant en projet Maven, il est plus simple de créer un projet neuf et propre et copier les sources existantes dans ce nouveau projet.

Voici la ligne de commande utilisée pour créer le projet du cours. Placez-vous dans votre workspace avec un cd et prenez la commande suivante comme modèle :

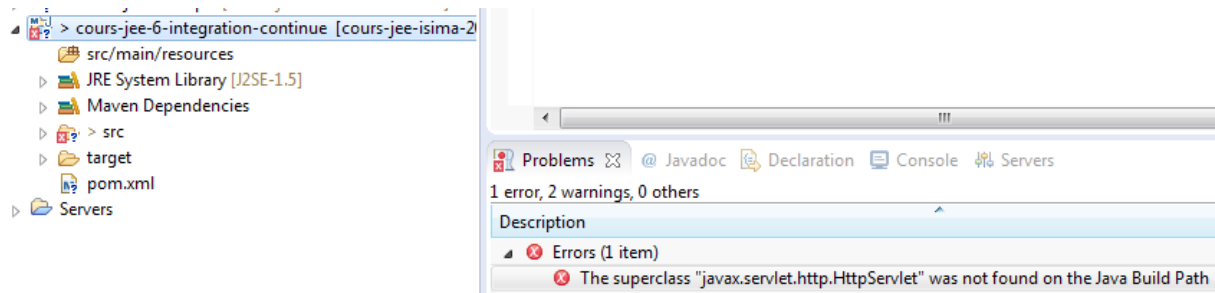
```
mvn archetype:generate -DgroupId={project-packaging}  
-DartifactId={project-name}  
-DarchetypeArtifactId=maven-archetype-webapp  
-DinteractiveMode=false
```

Importer le projet depuis Eclipse

Pour importer le projet sous Eclipse, il suffit d'utiliser le wizard adapté :



Vous obtiendrez le résultat suivant. Comme vous pouvez le constater, il y a une erreur et des sources folders manquent. L'archétype utilisé ne fait pas tout le job correctement et il reste quelques opérations manuelles à effectuer :



Pour commencer il faut modifier le pom.xml afin de résoudre la dépendance à HttpServlet (si vous le faites via les dépendances Eclipse comme vu au 1^{er} Tp, votre serveur d'intégration continue sera bien en peine de compiler votre projet) :

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-core</artifactId>
    <version>8.0.17</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>org.apache.tomcat</groupId>
    <artifactId>tomcat-jsp-api</artifactId>
    <version>8.0.17</version>
    <scope>provided</scope>
  </dependency>
</dependencies>
```

Vous noterez l'utilisation d'une version plus récente de Junit.

De plus il ne faut pas oublier d'ajouter les librairies que vous avez déjà utilisées, jstl et json/jee :

```
<dependency>
  <groupId>javax.json</groupId>
  <artifactId>javax.json-api</artifactId>
  <version>1.0</version>
</dependency>
<dependency>
  <groupId>jstl</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>
```

Par défaut, le plugin de création de war demande à ce qu'on ajoute un fichier web.xml. Pour éviter cela (nos projets n'en utilisent pas et il faut supprimer celui que vous trouvez dans le projet) :

```
<build>
  <finalName>cours-jee-6-integration-continue</finalName>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-war-plugin</artifactId>
      <configuration>
        <failOnMissingWebXml>>false</failOnMissingWebXml>
      </configuration>
    </plugin>
  </plugins>
</build>
```

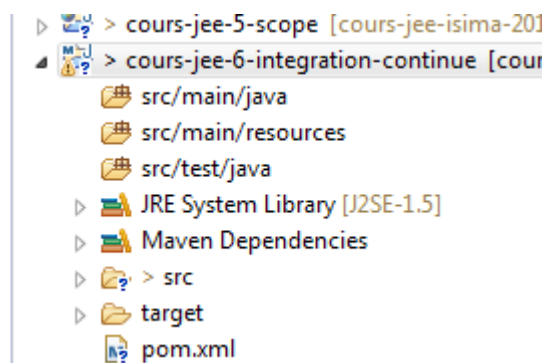
Profitons-en pour préciser le niveau de compilation que Maven doit appliquer quand il compile l'application, en rajoutant dans « plugins » l'instruction suivante :

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <configuration>
    <source>1.8</source>
    <target>1.8</target>
  </configuration>
</plugin>
```

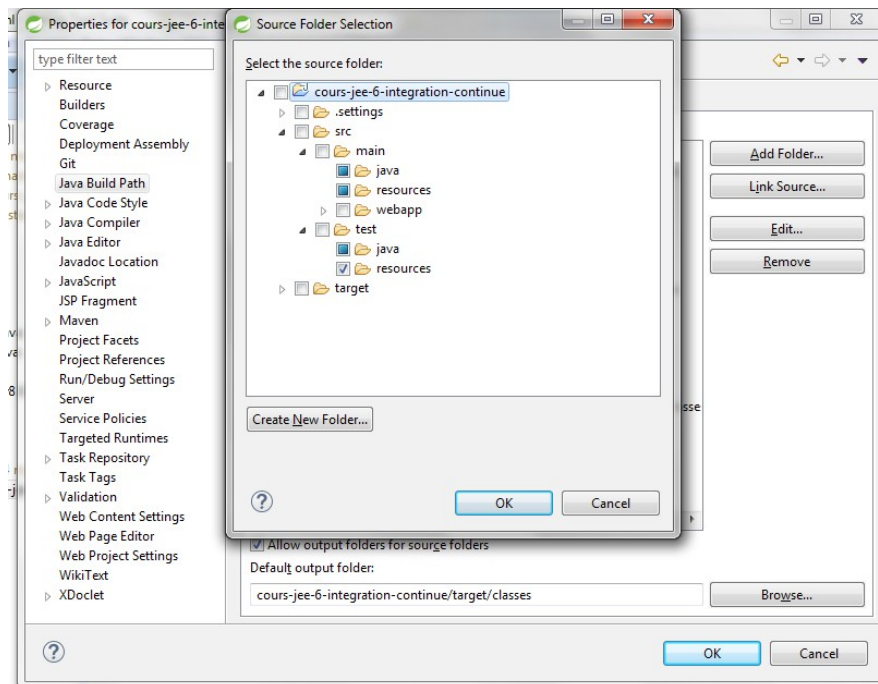
Il faut maintenant résoudre nos problèmes de « source folders » afin de se conformer aux projets Maven classiques, Il manque :

- src/main/java
- src/test/java
- src/test/resources

Créer les répertoires sur le disque, et faites un refresh. Il manque toujours le répertoire « src/test/resources » sur le projet :



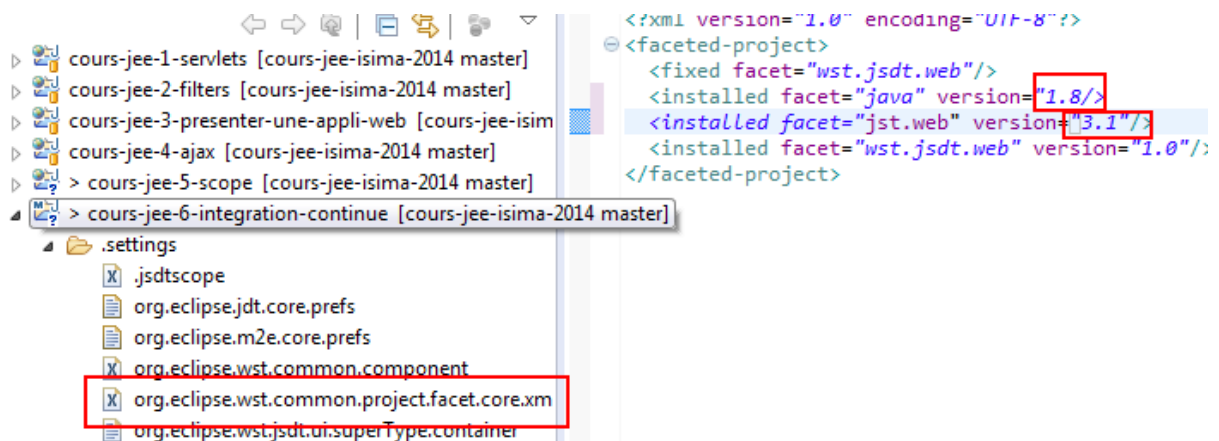
La raison est que l'archétype utilisé ne référence pas ce répertoire. Il suffit de la rajouter en tant que « source folder ». Pour ce faire un clic droit sur les propriétés du projet (alt+entrée), et le menu java build path :



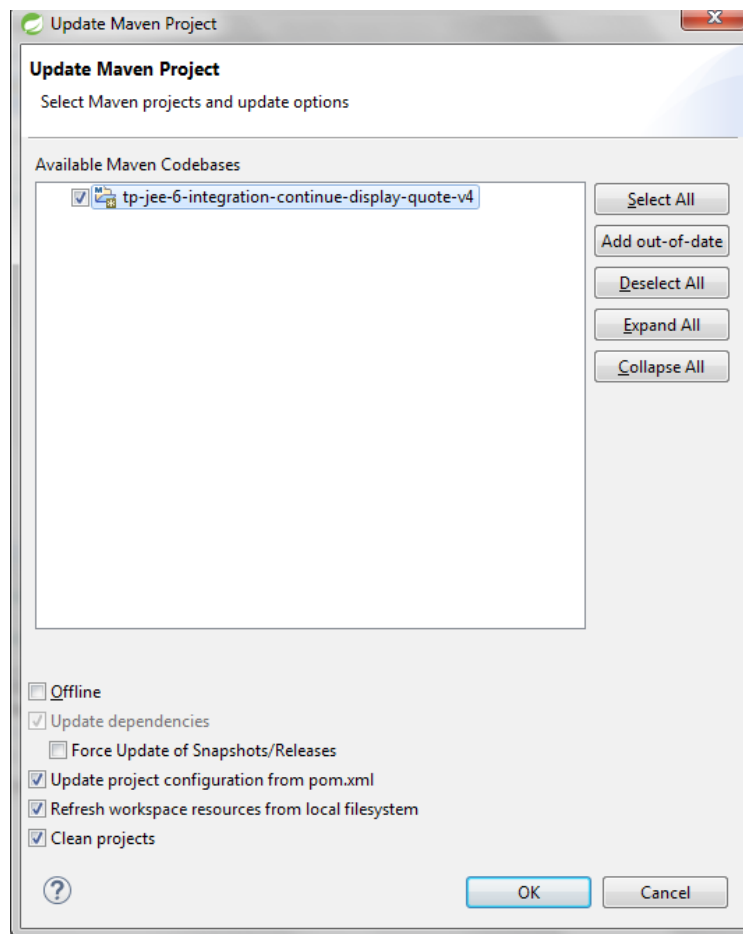
Il vous reste à copier les sources.

Des erreurs peuvent encore subsister... nous allons vérifier la conf Eclipse :

Positionnez-vous en vue « navigator » (window -> show view -> navigator) et vérifiez que le fichier org.eclipse.wst.common.project.facet.core.xml est bien rempli comme suit.



Eclipse va vous demander de rafraîchir la configuration en faisant un alt+F5 sur le projet. La fenêtre suivante apparaît, cliquez simplement sur ok :



Si vous avez suivi correctement l'ensemble des étapes décrites dans le document, votre projet doit maintenant compiler sans erreur. Il ne vous reste qu'à copier vos sources, comme précisé dans le paragraphe suivant.

Copier vos fichiers

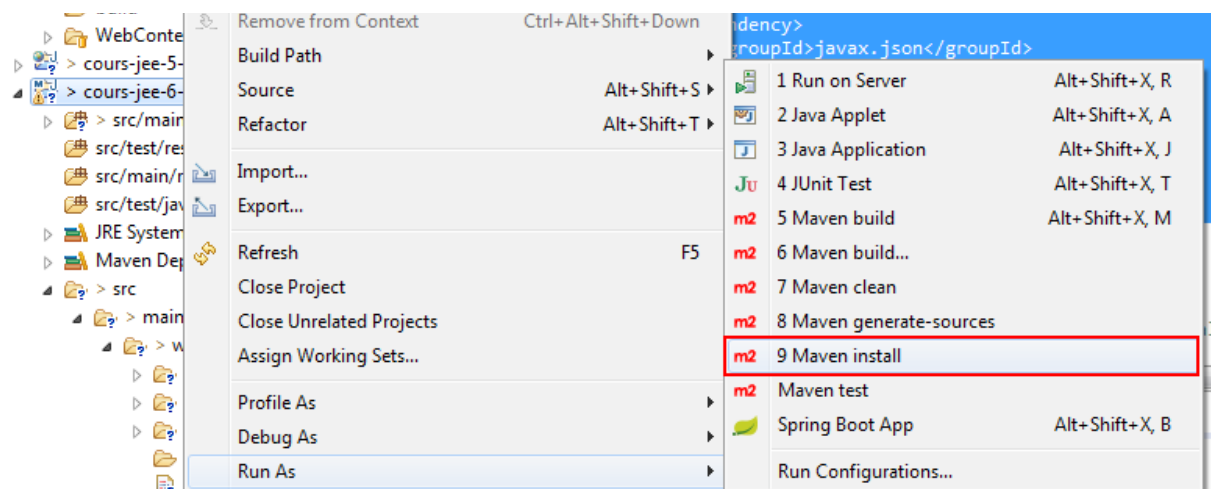
Il suffit de faire une copie de vos sources en suivant la méthode suivante :

- src -> src/main/java en omettant les fichiers de configuration ;
- src->src/main/resources pour les fichiers de configuration
- WebContent->src/main/webapp

Attention à supprimer le fichier « web.xml » qui est importé dans l'archétype que nous avons utilisé. Il se trouve dans src/main/webapp/WEB-INF/web.xml

Vérifier que votre build fonctionne correctement

Une fois que vous aurez terminé, exécutez un maven install pour vérifier que le build fonctionne.



Vous pourrez voir dans la vue console si le build s’est terminé avec succès :

```
[INFO] Building war: C:\Users\Benjamin\Documents\cours\onGitHub\cours-jee-isima-2014\sources\cours\cours-jee-6-integration-continue\cours-jee-6-integration-continue.war
[INFO] --- maven-install-plugin:2.4:install (default-install) @ cours-jee-6-integration-continue ---
[INFO] Installing C:\Users\Benjamin\Documents\cours\onGitHub\cours-jee-isima-2014\sources\cours\cours-jee-6-integration-continue\cours-jee-6-integration-continue.war to C:\Users\Benjamin\Documents\cours\onGitHub\cours-jee-isima-2014\sources\cours\cours-jee-6-integration-continue\cours-jee-6-integration-continue.war
[INFO] BUILD SUCCESS
[INFO] Total time: 3.501 s
[INFO] Finished at: 2015-01-23T23:37:06+01:00
[INFO] Final Memory: 11M/114M
```

Créer un projet java exécutable

Java permet de créer des “exécutables” (il faut tout de même un jdk pour les lancer). Cela permet par exemple de faire des programmes utilitaires. Les programmes exécutables sont des jars comme les libraries : ils ont simplement une classe avec une méthode main, comme celle-ci :

```
package fr.tp.isima;

public class Test {

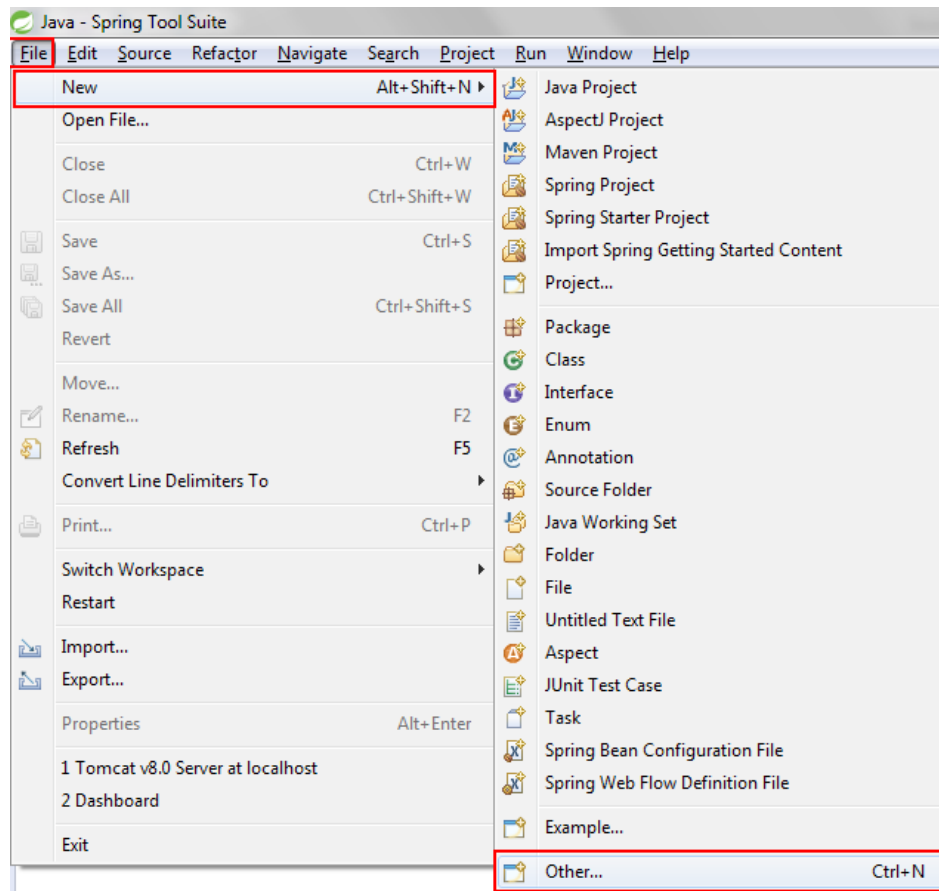
    public static void main(String[] args) {
        //TODO run the program
    }
}
```

Le wizard de création de classe permet de générer cette méthode avec la bonne signature de méthode.

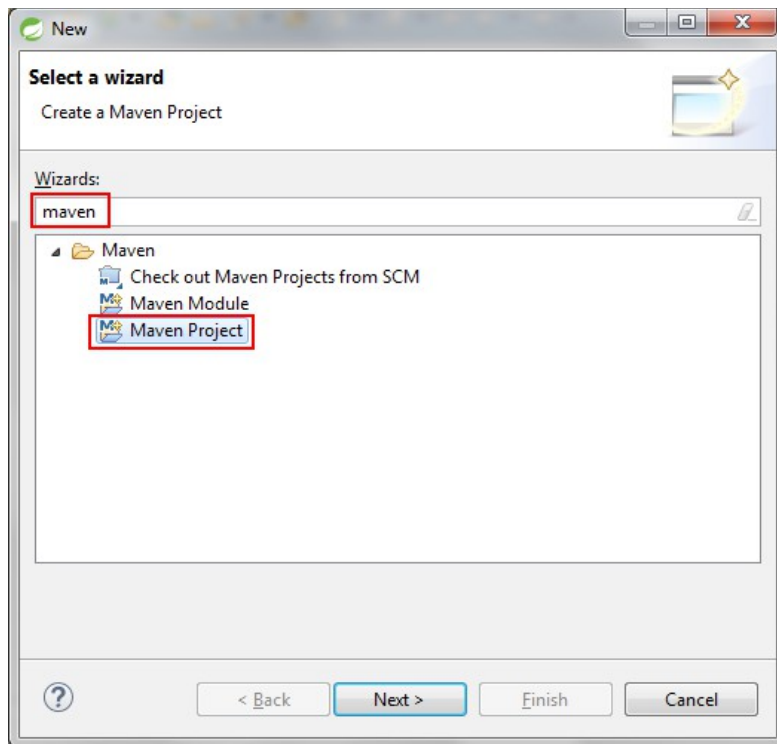
Cette documentation présente la méthode pour créer un jar exécutable qui affiche le premier argument passe en paramètre ou rien si aucun n’est transmis.

Création du projet sous eclipse

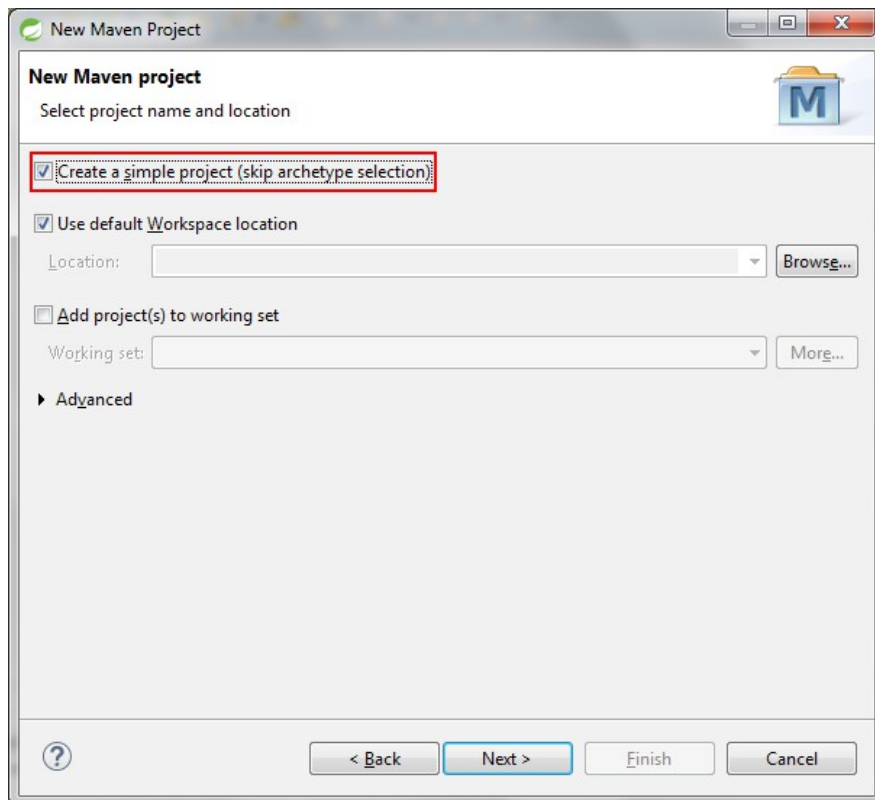
Allez dans le menu montré dans la capture ci-dessous : (file->new->other)



Vous allez voir la fenêtre de sélection des wizard, comme dans la capture, saisissez maven pour filtrer les wizards intéressants :



En appuyant sur le bouton « next », vous obtiendrez l'écran suivant. Cochez bien « create a simple project (skip archetype selection) ». Cliquez ensuite sur le bouton next :



Sur l'écran suivant vous pouvez remplir les informations qui vont servir dans votre fichier pom.xml pour déclarer votre projet !

New Maven Project

Configure project

Artifact

Group Id: fr.isima.exemple

Artifact Id: display-arguments

Version: 0.0.1-SNAPSHOT

Packaging: jar

Name:

Description:

Parent Project

Group Id:

Artifact Id:

Version:

Browse... Clear

Advanced

< Back Next > Finish Cancel

En ouvrant le fichier pom.xml, vous trouverez un fichier presque vide.

```
<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>4.0.0</modelVersion>
  <groupId>fr.isima.exemple</groupId>
  <artifactId>display-arguments</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</project>
```

Commençons par créer notre classe de test. Ne pas oublier de cocher dans le class wizard, la coche montrée dans la capture :

New Java Class

Java Class

Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☒ `public static void main(String[] args)`

☐ Constructors from superclass

☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

Le code est extrêmement simple, comme vous le voyez :

```
package fr.isima.exemple;

public class DisplayArguments {

    public static void main(String[] args) {
        if (args.length > 0) {
            System.out.println("Argument num 1 : " + args[0]);
        } else {
            System.err.println("Aucun argument fourni");
        }
    }
}
```

Ajoutez dans le pom.xml, la configuration suivante :

```
<properties>
    <main-jar-class>fr.isima.exemple.DisplayArguments</main-jar-class>
</properties>
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-jar-plugin</artifactId>
            <version>2.3.1</version>
            <configuration>
                <archive>
                    <manifest>
                        <addClasspath>>true</addClasspath>
                        <mainClass>${main-jar-
class}</mainClass>
                    </manifest>
                </archive>
            </configuration>
        </plugin>
        <plugin>
            <artifactId>maven-assembly-plugin</artifactId>
            <configuration>
                <descriptorRefs>
                    <descriptorRef>jar-with-
dependencies</descriptorRef>
                </descriptorRefs>
                <archive>
                    <manifest>
                        <mainClass>${main-jar-
class}</mainClass>
                    </manifest>
                </archive>
            </configuration>
            <executions>
                <execution>
                    <id>make-my-jar-with-dependencies</id>
                    <phase>package</phase>
                    <goals>
                        <goal>single</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
```

```

        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>3.2</version>
          <configuration>
            <source>1.8</source>
            <target>1.8</target>
          </configuration>
        </plugin>
      </plugins>
    </build>

```

Cette configuration soulève un des problèmes des fichiers pom.xml, ils sont souvent très verbeux. Ce paramétrage permet :

- de créer un jar avec la classe main DisplayArgumenets ;
- de créer un super jar avec toutes les dépendances incluses en un. Cela facilitera considérablement le déploiement de nos solutions ;
- de régler la compilation en java8.

Maven permet de définir des propriétés, qui facilitent la réutilisation de cette configuration dans un projet ultérieur :

```

<properties>
  <main-jar-class>fr.isima.exemple.DisplayArguments</main-jar-class>
</properties>

```

Il suffit de l'utiliser plus loin dans le fichier de configuration de la façon suivante, comme une EL :

```

<mainClass>${main-jar-class}</mainClass>

```

Test du projet

Pour tester le projet commencez par faire un maven install. Vous verrez alors dans votre console, la sortie suivante :

```

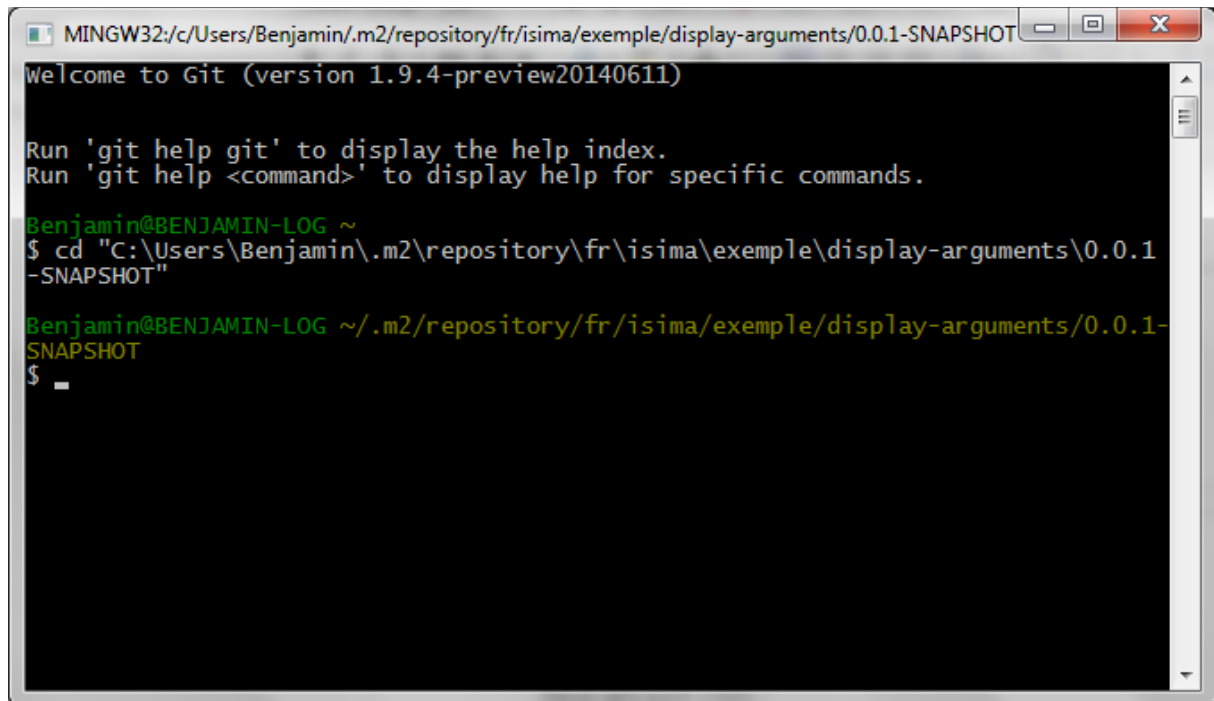
[INFO] --- maven-install-plugin:2.4:install (default-install) @ display-arguments
---
[INFO] Installing C:\Users\Benjamin\Documents\cours\onGitHub\cours-jee-isima-
2014\sources\exemples\display-arguments\target\display-arguments-0.0.1-
SNAPSHOT.jar to C:\Users\Benjamin\.m2\repository\fr\isima\exemple\display-
arguments\0.0.1-SNAPSHOT\display-arguments-0.0.1-SNAPSHOT.jar
[INFO] Installing C:\Users\Benjamin\Documents\cours\onGitHub\cours-jee-isima-
2014\sources\exemples\display-arguments\pom.xml to
C:\Users\Benjamin\.m2\repository\fr\isima\exemple\display-arguments\0.0.1-
SNAPSHOT\display-arguments-0.0.1-SNAPSHOT.pom
[INFO] Installing C:\Users\Benjamin\Documents\cours\onGitHub\cours-jee-isima-
2014\sources\exemples\display-arguments\target\display-arguments-0.0.1-SNAPSHOT-
jar-with-dependencies.jar to
C:\Users\Benjamin\.m2\repository\fr\isima\exemple\display-arguments\0.0.1-
SNAPSHOT\display-arguments-0.0.1-SNAPSHOT-jar-with-dependencies.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 14.175 s
[INFO] Finished at: 2015-01-24T14:52:24+01:00
[INFO] Final Memory: 16M/97M
[INFO] -----

```

Deux jars sont créés :

- le jar classique sans les dépendances ;
- le jar complet que nous pouvons utiliser pour l'exécution.

Ouvrons un shell, en se mettant sur le répertoire qui convient :



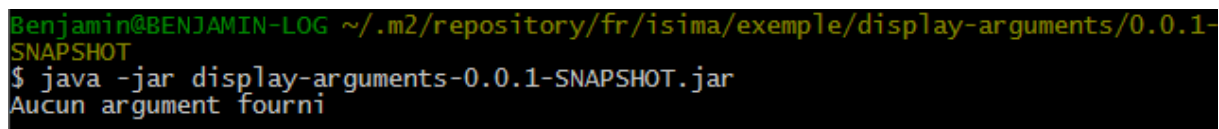
```
MINGW32:/c/Users/Benjamin/.m2/repository/fr/isima/exemple/display-arguments/0.0.1-SNAPSHOT
Welcome to Git (version 1.9.4-preview20140611)

Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

Benjamin@BENJAMIN-LOG ~
$ cd "C:\Users\Benjamin\.m2\repository\fr\isima\exemple\display-arguments\0.0.1-SNAPSHOT"
Benjamin@BENJAMIN-LOG ~/.m2/repository/fr/isima/exemple/display-arguments/0.0.1-SNAPSHOT
$
```

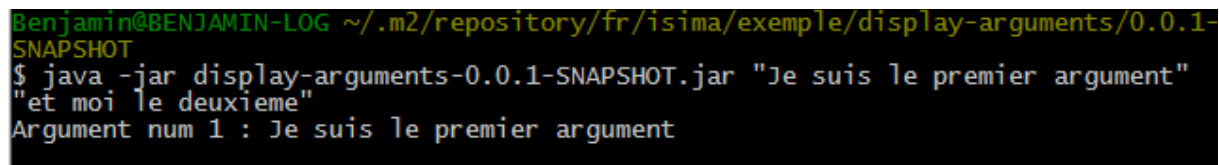
Exécutons le jar with dependencie avec la commande suivante :

"java -jar display-arguments-0.0.1-SNAPSHOT.jar"



```
Benjamin@BENJAMIN-LOG ~/.m2/repository/fr/isima/exemple/display-arguments/0.0.1-SNAPSHOT
$ java -jar display-arguments-0.0.1-SNAPSHOT.jar
Aucun argument fourni
```

Ajoutons deux arguments, pour finir le test de notre programme :



```
Benjamin@BENJAMIN-LOG ~/.m2/repository/fr/isima/exemple/display-arguments/0.0.1-SNAPSHOT
$ java -jar display-arguments-0.0.1-SNAPSHOT.jar "Je suis le premier argument"
"et moi le deuxieme"
Argument num 1 : Je suis le premier argument
```

Le séparateur entre les arguments est l'espace. Les guillemets permettent d'éviter que l'espace ne soit considéré comme un séparateur.

Référence le jar dans un autre projet

Une force de maven est que nous pouvons utiliser le jar dans un autre projet. Après avoir créé le projet « use-display-arguments », voici le pom.xml du projet :

```
<dependencies>
  <dependency>
    <groupId>fr.isima.exemple</groupId>
    <artifactId>display-arguments</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </dependency>
</dependencies>
```

La librairie est récupérée depuis le repository local. Voici un exemple d'utilisation de la classe DisplayArguments créée auparavant :

```
import fr.isima.exemple.DisplayArguments;

public class UseDisplayArguments {

    public static void main(String[] args) {
        DisplayArguments.main(args);
    }
}
```

Nous avons donc des jars exécutables qui peuvent faire office de librairie. De plus maven a la délicatesse de télécharger toutes les dépendances liées à display-arguments. Cela évite de faire des copies dans le pom.xml.