STI 3e année – Outils de développement

TD: Maven

J.-F. Lalande

Dans ce TD, nous proposons de développer une application en s'appuyant sur Maven. Maven permet de gérer les dépendances de projet et la compilation de celui-ci.

Pensez à créer un compte rendu PDF pour ce TD ! Il est attendu, dans ce compte rendu, de mentionner le dépot github associé.

1 Mon premier projet Maven

Pour que tout se déroule correctement, il faudra utiliser Eclipse Luna ou supérieur combiné à une JDK 1.8. Quelques références sont données en annexe.

Exercice 1 Si Maven n'est pas présent (testez en ligne de commande la commande "mvn"), récupérez les binaires de maven et décompressez les dans un coin. Changez votre PATH afin de pouvoir exécuter la commande "mvn".

Exercice 2 Créer un répertoire ProjetsMaven dans lequel vous taperez la commande de génération d'un projet "jf" (vous pouvez changer "jf" par autre chose !) :

```
mvn archetype:generate -DgroupId=jf -DartifactId=jf -DarchetypeArtifactId=maven-archetype-
    quickstart
```

Déplacez-vous dans le répertoire généré pour la suite.

Exercice 3 Regardez les fichiers générés. Transformez votre projet en projet git. Faites un push de celui-ci vers un dépôt github. Précisez l'url du dépôt github dans votre compte rendu.

Exercice 4 Compilez votre projet :

```
mvn compile
```

Exercice 5 Vérifier ce qu'il y a dans target. Executer la classe App en utilisant la commande maven :

```
mvn exec:java -Dexec.mainClass="jf.App"
```

Dans les récentes versions, Maven est supporté (par exemple dans Mars). Si vous êtes dans une veille version d'Eclipse, suivre la procédure décrite en Annexe.

Exercice 6 Importer ce projet dans eclipse **comme un projet Maven** et pas comme un projet Java (utiliser le répertoire racine de votre projet, celui contenant pom.xml).

Exercice 7 Faites quelques modifications. Exécutez votre classe App depuis Eclipse. Vérifiez que vous pouvez bien exécuter votre application modifiée en console avec Maven.

Exercice 8 Ajoutez une méthode :

```
public int max(int a, int b)
{
        return a > b ? a : b;
}
```

Exercice 9 Appelez la depuis votre méthode main et afficher son résultat pour l'appel à max(4,5) ;

Exercice 10 Dans votre class AppTest, dans la méthode testApp(), codez deux tests unitaires pour tester l'appel à max, sur un objet que vous instanciez à l'occasion. Appelez max(4,5) et max(5,4) et vérifiez que c'est toujours égal à 5 grâce à assertEquals.

Exercice 11 Demandez à Maven d'effectuer les tests unitaires à l'aide de la commande mvn test.

Exercice 12 Récupérer le contenu du tag "build" à partir de : `http://www.mkyong.com/maven/how-to-create-a-jar-file-with-maven/`. Adaptez votre finalName en marelease et votre mainClass et spécifier le numéro de version 1.8 pour source et target du plugin org.apache.maven.plugins (version de java pour la compilation).

Exercice 13 Construisez votre release :

Exercice 14 Testez la release de votre projet en exécutant le jar généré.

```
java -jar target/marelease.jar
```

Exercice 15 Modifiez votre implémentation de max, en :

```
public int max(int a, int b)
{
        return a; // Le stagiaire est passe par la
}
```

Exercice 16 Faites une release de votre projet. Que se passe-t-il ?

Exercice 17 Créez un répertoires releases et copiez marelease.jar dans ce répertoire en le nommant marelease-v1.jar. Ajoutez ce fichier à votre dépôt git et faites un push.

2 OpenCSV

Nous proposons de lire un fichier csv contenant des nombres. Nous allons chercher le max des nombres de ce fichier. Pour ne pas faire du parsing inutile nous allons utiliser une bibliothèque qui gère les csv et c'est Maven qui va se charger de l'intégration.

Exercice 18 Créez un fichier data.csv et mettez le à la racine de votre projet.

Exercice 19 Allez sur le site d'OpenCSV et récupérez le code "<dependency> ... </dependency>" à ajouter à votre pom.xml pour intégrer OpenCSV à votre projet.

Exercice 20 Ajoutez le code suivant à votre main qui va permettre de lire les données depuis votre fichier data.csv. Parcourez les valeurs de myEntries pour les afficher.

```
int monmax = 0;
CSVReader reader = new CSVReader(new FileReader("data.csv"));
List<String[]> myEntries = reader.readAll();
// parcourt
// System.out.println("Nombre lu: " + nb);
```

Exercice 21 Pour calculer le max des valeurs, nous proposons de remplacez le System.out.println à un appel à votre méthode max (elle n'est pas statique, donc créez un objet) afin de faire le max entre monmax et le chiffre courant nb. Affichez le max en fin d'algorithme.

Exercice 22 Réactivez le bug du stagiaire et vérifiez que le max est faux... C'est utile les tests unitaires !

Exercice 23 Faites une release de votre nouveau projet qui dépend maintenant d'un nouvel artefact. Que se passe-t-il si vous exécutez le jar généré en ligne de commande ?

Exercice 24 Pour résoudre le problème précédent, il faut ajouter un plugin `maven-assembly-plugin` qui va générer un jar et inclure les dépendances. Pour se faire, ajouter en fin de votre pom.xml le plugin suivant :

```
<plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <version>3.0.0</version>
                <configuration>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                    <archive>
                        <manifest> <mainClass>jf.App</mainClass>
                        </manifest>
                    </archive>
                </configuration>
                <executions>
                    <execution>
                        <id>make-assembly</id> <!-- this is used for inheritance merges -->
                        <phase>package</phase> <!-- bind to the packaging phase -->
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
</plugin>
```

Exercice 25 Faites une nouvelle release que vous pousserez dans releases/marelease-v2.jar.

3 Commons Collection

Dans cette partie, on souhait réaliser un filtre sur les éléments du fichier data.csv. Pour chaque ligne du fichier, on souhaite construire la collection contenant les éléments filtrés, par exemple ne garder que les éléments inférieurs à 50.

Exercice 26 Ajoutez la classe CollectionUtils version 4.1 à votre pom.xml et regénérez la configuration Eclipse :

```
    <dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-collections4</artifactId>
    <version>4.1</version>
    </dependency>
```

Exercice 27 Pendant que vous parcourez vos nombres dans les boucles convertissez votre ligne de nombre depuis le type `String[]` en une `List` grâce à l'appel à `list = Arrays.asList( ... )`.

Exercice 28 Utilisez alors un appel à `CollectionUtils.select` pour sélectionner les éléments d'un ensemble et construire un autre ensemble en utilisant un prédicat à coder. Votre appel à select sera :

```
Vector<String> out = new Vector<String>();
CollectionUtils.select(list, new MonPredicat<String>(), out);
System.out.println("OUT:" + out);
```

Le prédicat sélectionnera tous les nombres inférieurs à 50.

Exercice 29 Affichez l'ensemble filtré obtenu pour chaque ligne du fichier data.csv.

Exercice 30 En convertissant votre vecteur out en un `String[]` à l'aide de la méthode `.toArray(new String[0])`, écrivez votre ligne filtrée dans un nouveau fichier data-filtered.csv à l'aide d'un `CSVWriter`.

Exercice 31 Que constatez-vous sur le fichier produit ? Comme résoudre le problème ?

Exercice 32 Faites une nouvelle release que vous pousserez dans releases/marelease-v3.jar.

4 Annexes

4.1 Références

- `http://thierry-leriche-dessirier.developpez.com/tutoriels/java/importer-projet-maven-dans-ecl`
- `http://www.mkyong.com/maven/how-to-create-a-java-project-with-maven/`
- `http://matthieu-lux.developpez.com/tutoriels/java/maven/?page=eclipse`

4.2 Dans les veilles versions d'Eclipse

Exercice 33 Générez le .projet à l'aide du plugin eclipse :eclipse qui servira à Eclipse à savoir comment se configurer pour utiliser ce projet :

```
mvn eclipse:eclipse
```

Cette commande peut être réutilisée si vous changez la configuration de votre projet dans pom.xml. Le plugin mettra à jour le .projet.

Exercice 34 Importer ce projet dans eclipse comme un projet Maven (utiliser le répertoire contenant le .project comme répertoire à importer). Dans "Window>Preferences", puis dans "Java>Build Path>Classpath Variables", ajouter la variable d'environnement M2_REPO avec pour valeur */promo.../votre_login/.m2/repository*. Cela doit résoudre le problème d'Eclipse.

Exercice 35 Aller vérifier dans les propriétés du projet, Java Build Path, onglet Source, que la configuration spécifie bien que les classes se trouvent dans le sous répertoire *jf/target/classes*. C'est ce que Maven a spécifié lorsque vous avez généré le .project pour Eclipse.

Exercice 36 (optionnel car automatique si votre projet est reconnu par eclipse comme un projet Maven) Changez la chaine de compilation par défaut de votre projet : dans Propriétés > Builder, décochez Java Builder et crééz un nouveau Builder utilisant le binaire de maven (mvn), votre projet comme Working directory, et "compile" comme argument à ce builder. Reconstruisez votre proejt : vous devriez voir l'output de maven, identique à la commande mvn compile.