Conception et réalisation d'un outil de construction de projets Java

Christian Rinderknecht

4 février 2015

Un petit projet en Java est contenu dans un répertoire et est constitué de fichiers d'extension `.java`. Ces fichiers contiennent au moins une classe statique de même nom que le fichier et d'éventuelles classes auxiliaires. Les classes d'un fichier peuvent faire usage d'autres classes et objets définis dans d'autres fichiers, créant ainsi des dépendances utilisation-définition entre fichiers. Ces mêmes dépendances constituent des dépendances de compilation. En effet, si un fichier `A.java` dépend de `B.java`, alors il faut compiler `B.java` avant `A.java`. Le compilateur javac se charge seul de déterminer ces dépendances et, en fonction de celles-ci, compile l'ensemble des fichiers.

Lors de la première construction des *byte-codes* (`*.class`), en supposant que les sources sont livrés, tous les fichiers doivent et sont alors effectivement compilés. Mais si l'on développe soi-même une application, la commande `javac *.java` ne recompile alors que les fichiers qui ont besoin de l'être pour construire le *byte-code*. Par exemple, si, après avoir compilé notre exemple, seul `A.java` est modifié, il faut le recompiler mais il n'y a pas besoin de recompiler `B.java`. En général, lorsque l'on construit le *byte-code* d'un fichier, il faut déterminer s'il est à jour ou non. Il est à jour si et seulement si il est plus récent que le source et, si le source en question dépend d'autres sources, si ces sources sont à jour eux aussi. Un *byte-code* absent est considéré comme n'étant pas à jour, dans le but de déclencher la compilation de son source.

Il est possible que des fichiers Java soient mutuellement dépendants, par exemple `A.java` et `B.java`. Dans ce cas aussi, la commande `javac A.java B.java` traite la situation. Les *byte-codes* d'un ensemble de sources mutuellement dépendants sont tous à jour ou tous pas à jour. Si un des sources est plus récent que son *byte-code*, alors tous les sources de l'ensemble doivent être recompilés, sinon rien n'est fait.

Le but de ce projet est de concevoir et de réaliser un petit outil qui, étant donné l'ensemble des dépendances de compilation, effectue les compilations nécessaires — et seulement celles-ci — dans le bon ordre et compile ensemble les fichiers mutuellement dépendants. Les instructions de compilations ont donc deux formes :

— `javac A.java`
— `javac A.java B.java` si et seulement si `A.java` et `B.java` dépendent mutuellement l'un de l'autre (dans le cas général le nombre de fichiers mutuellement dépendants n'est pas limité *a priori*).

On propose que les dépendances soient consignées dans un fichier textuel constitué de lignes de la forme par exemple `A: B C`, signifiant que le fichier `A.java` dépend de `B.java` et `C.java` (la dépendance entre `B.java` et `C.java` n'est pas spécifiée). Un autre exemple serait :

```
X: Y
Y: Z T
Z: T
```

pour lequel l'unique suite de compilations pour construire `X.class` est (si aucun *byte-code* n'est à jour) :

```
javac T.java
javac Z.java
javac Y.java
javac X.java
```

En général, l'ordre est-il unique ? Ou encore

```
X: Y
Y: Z T
Z: T
T: Z
```

pour lequel les compilations pour construire `X.class` sont :

```
javac Z.java T.java
javac Y.java
javac X.java
```

L'outil devra se nommer `build`, prendre en argument sur la ligne de commande le nom du fichier de *byte-code* à construire et le nom du fichier de dépendance *via* une option `-d`, par exemple : `java build -d depend X`. Il faudra implanter une option `-v` pour afficher en plus les commandes de compilation et `-n` pour seulement afficher les commandes sans les envoyer à l'interprète de commandes sous-jacent.