UNIVERSITÉ CAEN NORMANDIE

UFR des Sciences

Université de Caen Normandie
Département mathématiques-informatique
2022–2023
Bruno Zanuttini

L2 informatique

Mémo. Commandes Java

Unité SMINFL3A : Programmation Java orientée objet

---

1 Compilation et exécution

Note De façon générale, le caractère « `:` » est un séparateur pour l'argument `-cp` (*classpath*) ; par exemple, `-cp ".:y/x.jar"` représente le répertoire courant (caractère « `.` ») et l'archive `x.jar`, située dans le répertoire `y`. Sous Windows, le séparateur est le caractère « `;` » au lieu de « `:` ».

Commandes

— Compiler toutes les classes du répertoire `monpackage` et placer les fichiers compilés (`.class`) dans le répertoire `build` :

```
javac -d build monpackage/*.java
```

— Exécuter la classe `monpackage.MaClasse` sachant que les fichiers compilés sont dans le répertoire `build` :

```
java -cp build monpackage.MaClasse
```

— Exécuter la classe `monpackage.MaClasse` en lui passant les arguments `mon-argument` et `arg avec espaces`, sachant que les fichiers compilés sont dans le répertoire `build` :

```
java -cp build monpackage.MaClasse mon-argument "arg␣avec␣espaces"
```

— Compiler toutes les classes du répertoire `monpackage` en utilisant les classes de l'archive `x.jar` située dans le répertoire `y` et placer les fichiers compilés (`.class`) dans le répertoire `build` :

```
javac -d build -cp y/x.jar monpackage/*.java
```

— Compiler toutes les classes du répertoire `monpackage` et les classes nécessaires des autres *packages* situés dans le répertoire courant, en utilisant les classes de l'archive `x.jar` située dans le répertoire `y`, et placer les fichiers compilés (`.class`) dans le répertoire `build` :

```
javac -d "build" -cp ".:y/x.jar" monpackage/*.java
```

Remplacer « `:` » par « `;` » dans le *classpath* sous Windows.

— Exécuter la classe `monpackage.MaClasse` en utilisant l'archive `x.jar` située dans le répertoire `y`, sachant que les fichiers compilés sont dans le répertoire `build` :

```
java -cp "build:y/x.jar" monpackage.MaClasse
```

Remplacer « `:` » par « `;` » dans le *classpath* sous Windows.

— Exécuter la classe `monpackage.MaClasse` en utilisant l'archive `x.jar` située dans le répertoire `y` et l'archive `t.jar` située dans le répertoire `u`, sachant que les fichiers compilés sont dans le répertoire `build` :

```
java -cp "build:y/x.jar:u/t.jar" monpackage.MaClasse
```

Remplacer « `:` » par « `;` » dans le *classpath* sous Windows.

2 Archives et Javadoc

— Créer une archive nommée `x.jar` et contenant tous les fichiers compilés du répertoire `build` :

```
cd build/
jar cf x.jar .
```

— Créer une archive nommée `x.jar` contenant tous les fichiers compilés du répertoire `build`, et qui soit exécutable avec comme classe principale `package.MaClasse` :

```
jar cfe x.jar package.MaClasse .
```

— Exécuter l'archive `x.jar` (qui doit être exécutable) :

```
java -jar x.jar
```

À noter, on ne peut pas exécuter l'archive `x.jar` (si elle a été créée comme ci-dessus) en utilisant également les classes de l'archive `y.jar`, on doit écrire :

```
java -cp "y.jar:x.jar" package.MaClasse
```

Remplacer « `:` » par « `;` » dans le *classpath* sous Windows.

— Lister le contenu de l'archive `x.jar` :

```
jar tf x.jar
```

— Générer la documentation de toutes les classes des *packages* `monpackage1` et `monpackage2`, et placer cette documentation dans le répertoire `doc` :

```
javadoc -d doc monpackage1/*.java monpackage2/*.java
```

3 Exemple d'arborescence de fichiers

On peut par exemple créer un répertoire par TP, ou un répertoire pour l'unité, au choix, l'important étant de séparer

— ce qu'on produit (code source, c'est-à-dire les fichiers `.java`),
— ce que le compilateur produit automatiquement (code compilé, c'est-à-dire les fichiers `.class`),
— les librairies externes utilisées (essentiellement des archives Java, c'est-à-dire des fichiers `.jar`).

L'arborescence suivante donne un exemple d'organisation (pour les exemples du premier cours) ; c'est le résultat de la commande `tree` exécutée dans le répertoire où sont stockés tous ces exemples (`demo_pour_cours`) :

```
.
|__ build
|__ lib
|   |__ personstests.jar
|__ src
    |__ computations
    |   |__ Demo.java
    |__ hello
    |   |__ Demo.java
    |__ persons
        |__ Demo.java
        |__ Person.java
        |__ Test.java
```

En supposant que c'est cette architecture qui est utilisée, les commandes suivantes, toutes exécutées depuis le répertoire `demo_pour_cours/src`, couvrent les principaux cas d'usage :

```
javac -d ../build hello/Demo.java
java -cp ../build hello.Demo
javac -d ../build computations/*.java
java -cp ../build computations.Demo
javac -d ../build -cp ".:../lib/personstests.jar" persons/*.java
java -cp "../build:../lib/personstests.jar" persons.Demo
java -cp "../build:../lib/personstests.jar" persons.Test
```

Remplacer « `:` » par « `;` » dans le *classpath* sous Windows.

On peut également ajouter à cette architecture un répertoire `scripts`, dans lequel on peut placer des scripts exécutant les commandes dont on a souvent besoin (par exemple `compile.sh`, `test.sh`, etc.). Ceci peut se révéler particulièrement pertinent pour le fil rouge.