

Compiler, interpréter, distribuer, exécuter

Olivier Cailloux

LAMSADE, Université Paris-Dauphine

Version du 18 mars 2023

Introduction

- Compiler : transformer un langage en un autre
- Interpréter : traduire et exécuter les instructions
- C++ est le plus souvent compilé en code machine (puis interprété par le CPU, ou « exécuté »)
- Java est généralement compilé vers un langage de haut niveau
- ... ou directement interprété (?)
- ... ou compilé en code machine
- Compilation-puis-interprétation est devenu courant dans les langages de haut niveau (Python, Ruby, C#, ...) avec exceptions (Go)

Introduction

- Compiler : transformer un langage en un autre
- Interpréter : traduire et exécuter les instructions
- C++ est le plus souvent compilé en code machine (puis interprété par le CPU, ou « exécuté »)
- Java est généralement compilé vers un langage de haut niveau
- ... ou directement interprété (jshell)
- ... ou compilé en code machine
- Compilation-puis-interprétation est devenu courant dans les langages de haut niveau (Python, Ruby, C#, ...) avec exceptions (Go)

Compilation

- Compilateur fourni avec le JDK
- JDK ?
- Transforme le code source en code compilé
- Par convention, le code source est dans un fichier `.java`
- Le code compilé est dans un fichier `.class`
- Appelé *bytecode*
- Une paire de fichiers par classe de premier niveau

Compilation

- Compilateur fourni avec le JDK
- JDK ? Java Development Kit
- Transforme le code source en code compilé
- Par convention, le code source est dans un fichier `.java`
- Le code compilé est dans un fichier `.class`
- Appelé *bytecode*
- Une paire de fichiers par classe de premier niveau

Interprétation

- Bytecode destiné à être interprété par une machine virtuelle Java (JVM)
 - Les fichiers `.class` ne dépendent *pas* de l'OS de destination
 - La JVM dépend de l'OS de destination
- ⇒ fichiers `.class` + JVM linux = programme exécutable sous linux
- ⇒ fichiers `.class` + JVM windows = programme exécutable sous windows
- ⇒ ...

Distribution et utilisation

Comment rendre son programme Java utilisable ?

- ❶ JVM unique + fichiers `.class`, séparément
 - Installer la JDK sur le système, inclut la JVM [U]
 - Récupérer les `.class` [U]
 - Exécuter avec la JVM à l'aide d'une commande [U]
 - Ou inclure un lanceur, une icône à cliquer... (qui exécute la commande) [D]
- ❷ JVM embarquée avec les fichiers `.class`
 - Emballer la JVM, les fichiers `.class` et un lanceur dans un installeur (cf. `jlink`) [D]
 - Installer puis exécuter le lanceur [U]
- ❸ La JVM tourne sur un serveur et communique avec l'utilisateur par le réseau

Dans ce cours : JVM unique (1), peut-être serveur (3)

Commandes : Compiler

- Compiler : invoquer javac
 - donner les chemins des fichiers à compiler en arguments
 - Extension de ces fichiers ?
 - Exemple :
-
- Fichiers compilés généralement dans répertoires \neq des sources

Commandes : Compiler

- Compiler : invoquer javac
- donner les chemins des fichiers à compiler en arguments
- Extension de ces fichiers? .java
- Exemple :
- Fichiers compilés généralement dans répertoires \neq des sources

Commandes : Compiler

- Compiler : invoquer `javac`
- donner les chemins des fichiers à compiler en arguments
- Extension de ces fichiers? `.java`
- Exemple : `javac "MainCls.java" "RequiredCls.java"`,
produisant `MainCls.class` et `RequiredCls.class`
- Fichiers compilés généralement dans répertoires \neq des sources

Commandes : Interpréter

- Invoquer java
- donner le *nom de la classe* à exécuter en argument
- Que manque-t-il ?
- Préciser le « Class path » avec `-cp` chemins
- Exemple : `java -cp "." "MainClass"`
- Pas de class path précisé : `.` par défaut
- Séparer les chemins par `:` sous Linux et par `;` sous Windows
- Exemple : `java -cp "folder1:folder2" "MainClass"`

Mais que manque-t-il encore ?

Commandes : Interpréter

- Invoquer java
- donner le *nom de la classe* à exécuter en argument
- Que manque-t-il ? L'endroit où trouver les classes !
- Préciser le « Class path » avec `-cp` chemins
- Exemple : `java -cp "." "MainClass"`
- Pas de class path précisé : `.` par défaut
- Séparer les chemins par `:` sous Linux et par `;` sous Windows
- Exemple : `java -cp "folder1:folder2" "MainClass"`

Mais que manque-t-il encore ?

Commandes : Interpréter

- Invoquer java
- donner le *nom de la classe* à exécuter en argument
- Que manque-t-il ? L'endroit où trouver les classes !
- Préciser le « Class path » avec `-cp` chemins
- Exemple : `java -cp "." "MainClass"`
- Pas de class path précisé : `.` par défaut
- Séparer les chemins par `:` sous Linux et par `;` sous Windows
- Exemple : `java -cp "folder1:folder2" "MainClass"`

Mais que manque-t-il encore ? La méthode à exécuter !

Main

- Par convention, méthode à exécuter déclarée par : `public static void main(String[] args)`
- L'interpréteur reçoit le nom de la classe à exécuter
- Il exécute la méthode `main` de cette classe
- Les arguments suivant le nom de la classe à exécuter sont passés dans le paramètre `args`
- Exemple : `java -cp "." "MainClass" "argument 1" "argument 2"`

Note :

- On peut avoir un `main` dans plusieurs classes
- ⇒ multiples points d'entrée possibles

Classes dans packages

- Commandes précédentes valables quand pas de package défini (package par défaut)
- Quid si classe `MyClass` définie dans package `me.myapp.mytheme` ?
- Donner à `javac` les fichiers à compiler : ?
- Donner à `java` le nom *complet* de la classe à exécuter : ?
- Mais que manque-t-il ?
- Class path : les chemins que la JVM utilise pendant son exécution pour charger les classes (sauf celles de l'API Java)
- Class path fourni à la JVM au démarrage
- Exemple : ?

Classes dans packages

- Commandes précédentes valables quand pas de package défini (package par défaut)
- Quid si classe `MyClass` définie dans package `me.myapp.mytheme` ?
- Donner à `javac` les fichiers à compiler : `javac "My project/src/me/myapp/mytheme/MyClass.java"`
- Donner à `java` le nom *complet* de la classe à exécuter : ?
- Mais que manque-t-il ?
- Class path : les chemins que la JVM utilise pendant son exécution pour charger les classes (sauf celles de l'API Java)
- Class path fourni à la JVM au démarrage
- Exemple : ?

Classes dans packages

- Commandes précédentes valables quand pas de package défini (package par défaut)
- Quid si classe `MyClass` définie dans package `me.myapp.mytheme` ?
- Donner à `javac` les fichiers à compiler : `javac "My project/src/me/myapp/mytheme/MyClass.java"`
- Donner à `java` le nom *complet* de la classe à exécuter : `java "me.myapp.mytheme.MyClass"`
- Mais que manque-t-il ?
- Class path : les chemins que la JVM utilise pendant son exécution pour charger les classes (sauf celles de l'API Java)
- Class path fourni à la JVM au démarrage
- Exemple : ?

Classes dans packages

- Commandes précédentes valables quand pas de package défini (package par défaut)
- Quid si classe MyClass définie dans package `me.myapp.mytheme` ?
- Donner à javac les fichiers à compiler : `javac "My project/src/me/myapp/mytheme/MyClass.java"`
- Donner à java le nom *complet* de la classe à exécuter : `java "me.myapp.mytheme.MyClass"`
- Mais que manque-t-il ? Où trouver cette classe !
- Class path : les chemins que la JVM utilise pendant son exécution pour charger les classes (sauf celles de l'API Java)
- Class path fourni à la JVM au démarrage
- Exemple : ?

Classes dans packages

- Commandes précédentes valables quand pas de package défini (package par défaut)
- Quid si classe `MyClass` définie dans package `me.myapp.mytheme` ?
- Donner à `javac` les fichiers à compiler : `javac "My project/src/me/myapp/mytheme/MyClass.java"`
- Donner à `java` le nom *complet* de la classe à exécuter : `java "me.myapp.mytheme.MyClass"`
- Mais que manque-t-il ? Où trouver cette classe !
- Class path : les chemins que la JVM utilise pendant son exécution pour charger les classes (sauf celles de l'API Java)
- Class path fourni à la JVM au démarrage
- Exemple : `java -cp "My project/src/" "me.myapp.mytheme.MyClass"`

License

This presentation, and the associated \LaTeX code, are published under the MIT license. Feel free to reuse (parts of) the presentation, under condition that you cite the author. Credits are to be given to Olivier Cailloux, Université Paris-Dauphine.