

Partagez simplement vos Java CLI Apps

Pierre-Yves Fourmond

 Human Talks Paris

 ?? ?? 2025

Qui suis-je ?



Pierre-Yves Fourmond

Développeur Back

Consultant Senior ( Tribu « Java / Kotlin »)




 Membre de Paris JUG

 [grumpyf0x48](#)  [pyfourmond.bsky.social](#)

#Java #Linux #Bash #Scripting #CLI

Le besoin

Votre équipe a identifié un nouveau besoin :

- Adressable par une CLI
- Facile à installer et à utiliser
-  Avec la volonté d'aller vite

Quel type d'application ?

Des outils pour les techs / QA :

- Générer des données de test
- Des commandes pour la CI
- Comparer des fichiers dans un double run

Un Proof Of Concept

Comment adresser ce besoin ?

 Et si on utilisait du scripting ?

- Il n'y a pas de compilation
- Le code **source** est **interprété**


Il est :

- **Livré**
- **Modifiable**

On va vraiment faire du scripting en Java ?



Avec quel outil ?

- Utiliser 
- Installer JBang sur la machine cible 🤔
- 🙅 Restons sur le JDK

Commençons avec Java 8

```
class Hello {  
    public static void main(String... args) {  
        System.out.println("Hello " + String.join(",", args));  
    }  
}
```

```
$ javac Hello.java  
$ java Hello Human Talks
```

```
Hello Human,Talks
```

- On a deux étapes successives :
 - **1** Compilation
 - **2** Exécution
- **✗ Ce n'est pas du scripting**




Passons à Java 11

Arrivée de la JEP 330 : [Launch Single-File Source-Code Programs](#)

```
$ java Hello.java Human Talks
```

```
Hello Human, Talks
```

- On n'appelle plus le compilateur `javac`
-  **C'est bien du scripting**



Scripting avec Java 11 et Linux

Utilisons un **Shebang** pour lancer notre programme :

```
$ cat Hello
```

```
#!/usr/bin/java --source 11
```

```
class Hello {  
    public static void main(String... args) {  
        System.out.println("Hello " + String.join(", ", args));  
    }  
}
```

```
$ ./Hello Human Talks
```

```
Hello Human,Talks
```

Java 11 et Linux 🤔

Pourquoi cette façon de faire pose un problème aux devs ?

```
#!/usr/bin/java --source 11

class Hello {
    public static void main(String... args) {
        System.out.println("Hello " + String.join(", ", args));
    }
}
```

- Le nom du fichier source n'a plus l'extension `.java`
- La première ligne `#!/usr/bin/java ...` n'est pas valide en Java
- 💡 **Et si on nommait le fichier `Hello.java` comme au début ?**

Java 11, Linux et une extension `.java`

😞 Perdu !

```
$ ./Hello.java
```

```
./Hello.java:1: error: illegal character: '#'  
#!/usr/bin/java --source 11  
^  
./Hello.java:1: error: class, interface, or enum expected  
#!/usr/bin/java --source 11  
^  
2 errors  
error: compilation failed
```

✗ Le compilateur ne comprend pas la première ligne

Scripting avec Java 11 et Linux

- Notre script se lance dans le terminal
- On ne sait pas l'éditer 😞
- **On a besoin d'un mode de lancement compris par Bash et ignoré de Java**

- ```
#!/usr/bin/java --source 11 "$0" "$@"; exit $?

class Hello {
 public static void main(String... args) {
 System.out.println("Hello " + String.join(", ", args));
 }
}
```

- ```
$ ./Hello.java Human Talks
```

```
Hello Human,Talks
```

Une CLI App pour générer des données de test

```
///usr/bin/java --source 21 --enable-preview --class-path lib/picocli-4.7.6.jar:lib/commons-lang3-3.14.0.jar "$@" "$@"; exit $?  
  
import ...  
  
@Command(name = "GenerateData", version = "0.1")  
class GenerateData implements Callable<Integer> {  
  
    @Option(names = {"-c", "--column"}, description = "Map a column with a fixed value, mapping function or value from a file")  
    String[] columnMappings;  
  
    @Option(names = {"-n", "--count"}, description = "Number of lines to generate", defaultValue = "100")  
    int lineCount;  
  
    @Parameters(arity = "1", description = "The file containing the SQL create table request")  
    File sqlRequestFile;  
  
    void main(String... args) {  
        System.exit(new CommandLine(new GenerateData()).execute(args));  
    }  
  
    @Override  
    public Integer call() throws IOException {  
        System.out.println(TableData.generate(sqlRequestFile, columnMappings, lineCount).toSQLInserts());  
        return 0;  
    }  
  
    record TableData(TableDefinition tableDefinition, TableRows tableRows) implements Exportable { ... }  
  
    ...  
}
```

L'application en mode dev

```
create table commandes (  
    id            uuid not null constraint "commandes_pk" primary key,  
    numero_client varchar,  
    date_commande timestamp,  
    montant       integer  
);
```

```
dev@equipe:~/sources/generate-data$ ./GenerateData.java
```

Missing required parameter: '<sqlRequestFile>'

Usage: **GenerateData** [-n=<lineCount>] [-c=<columnMappings>]... <sqlRequestFile>

<sqlRequestFile> The file containing the SQL create table request

-c, --column=<columnMappings>
Map a column with a fixed value, mapping function
or value from a file

-n, --count=<lineCount> Number of lines to generate

```
dev@equipe:~/sources/generate-data$ ./GenerateData.java --column id::random --column numero_client::file::clients.txt --column montant::random --count 10 create_table.sql
```

```
INSERT INTO commandes (id, numero_client, date_commande, montant)
```

```
VALUES
```

```
('484f7c61-83a9-4b1d-9f72-7a78fabff1bc', '10850', null, 350),  
( '4eec07f9-a707-418c-a0d3-0a403b5f192e', '11204', null, 150),  
( 'b6d66906-a533-427a-b6fe-187ccdd2224a', '12311', null, 300),  
( '14568230-6a6c-47ed-a646-0d4651458355', '12355', null, 150),  
( '9b046a0d-1305-40f4-afce-32c90d693149', '12714', null, 200),  
( '53467cca-b944-47c0-9926-c05efef3927e', '13011', null, 500),  
( '80660c2e-6d00-4e87-bb2d-beb3d29560b9', '13256', null, 250),  
( '933f34e6-b11d-40da-87d3-e3f9438df5bd', '13394', null, 100),  
( '49d9c9b0-4fe7-48cc-9336-82791b99fa2d', '13433', null, 100),  
( '971956d6-450a-4ede-a403-c7df61758fd6', '14405', null, 100);
```

Le livrable de l'application

👉 Créer une archive au format `zip` avec des répertoires `src`, `lib` et `bin` :

```
$ unzip -l build/GenerateData.zip
```

```
Archive:  build/GenerateData.zip
 Length      Date    Time    Name
-----
         0  2023-12-23  15:35  generate-data/
         0  2023-12-23  15:35  generate-data/src/
      11431  2023-12-23  15:35  generate-data/src/GenerateData.java
         0  2023-12-23  15:35  generate-data/lib/
     657952  2023-12-23  15:35  generate-data/lib/commons-lang3-3.14.0.jar
     415128  2023-12-23  15:35  generate-data/lib/picocli-4.7.6.jar
         0  2023-12-23  15:35  generate-data/bin/
        333  2023-12-23  15:35  generate-data/bin/GenerateData.sh
-----
    1084844
           8 files
```

Le Shell de lancement

Permet de changer le classpath de :

```
lib/picocli-4.7.6.jar:lib/commons-lang3-3.14.0.jar
```

à :

```
$APP_DIR/lib/picocli-4.7.6.jar:$APP_DIR/lib/commons-lang3-3.14.0.jar
```

pour être indépendant du répertoire de lancement 😎

Packager notre application

On a juste besoin de quelques commandes dans un Makefile :

```
APP_NAME := GenerateData
APP_DIR := generate-data

BUILD := build
BUILD_APP := $(BUILD)/$(APP_DIR)

package: build
    cd $(BUILD) && zip --recurse-paths $(APP_NAME).zip $(APP_DIR)

build: prepare
    cp --recursive --update src lib bin $(BUILD_APP)

prepare:
    mkdir --parents $(BUILD_APP)/src $(BUILD_APP)/lib $(BUILD_APP)/bin
```

L'application installée




```
autre-dev@equipe:~$ unzip -d /home/installApps /home/Téléchargements/GenerateData.zip
Archive:  /home/installApps/Téléchargements/GenerateData.zip
  creating: /home/installApps/generate-data/
  creating: /home/installApps/generate-data/lib/
  inflating: /home/installApps/generate-data/lib/picocli-4.7.5.jar
  inflating: /home/installApps/generate-data/lib/commons-lang3-3.14.0.jar
  creating: /home/installApps/generate-data/bin/
  inflating: /home/installApps/generate-data/bin/GenerateData.sh
  creating: /home/installApps/generate-data/src/
  inflating: /home/installApps/generate-data/src/GenerateData.java

autre-dev@equipe:~$ export PATH=/home/installApps/generate-data/bin:/usr/lib/jvm/java-1.21.0-openjdk-amd64/bin:$PATH

autre-dev@equipe:~$ which GenerateData.sh
/home/installApps/generate-data/bin/GenerateData.sh

autre-dev@equipe:~$ GenerateData.sh --column id::random --column numero_client::file::clients.txt --column montant::random --count 1
0 create_table.sql
INSERT INTO commandes (id, numero_client, date_commande, montant)
VALUES
('4228132f-4ca1-4cf5-8355-fd175dab0c1a', '10850', null, 200),
('0a97e5fc-49e6-4ab4-a2ad-bf25f7f9ed94', '11204', null, 200),
('efcab680-6519-4991-b203-32799da99d3f', '12311', null, 250),
('66bd6562-52ab-4af5-843d-63988080d438', '12355', null, 200),
('1f9100fd-b495-4b42-8152-0e469bb3b6de', '12714', null, 300),
('b3a7f8ed-d515-485f-a092-e2e9891fdf14', '13011', null, 250),
('d853196e-92a2-439e-8fd3-aade090dac5b', '13256', null, 400),
('5a5b456c-ab38-43e6-96f9-a1c910ebb219', '13394', null, 100),
('823aba91-0f62-4c51-b62f-3254de2abf76', '13433', null, 50),
('fcf98b9a-ddea-48b0-b3a5-9df4d343742a', '14405', null, 200);
```

Pourquoi choisir cette approche ?

-  Code source modifiable
-  Packaging simplifié
-  On ne dépend que du JDK

Pour aller plus loin

- Limitation à un fichier source
 - 🖱️ Utiliser Java 22 et la JEP 458 : [Launch Multi-File Source-Code Programs](#)
- Le code est compilé à chaque fois
 - 🕒 C'est l'affaire d'une seconde !
- Gestion des dépendances
 - 🤖 Comment éviter de les gérer manuellement ?

Résoudre les dépendances

💡 Utilisons Gradle pour packager **sources**, **dépendances** et **Shell de lancement** :


```
distributions {
    create("scripts") {
        contents {
            from("src").include("*.java").into("src")
            from(configurations.runtimeClasspath).into("lib")
            from("bin").include("*.sh").into("bin")
        }
    }
}

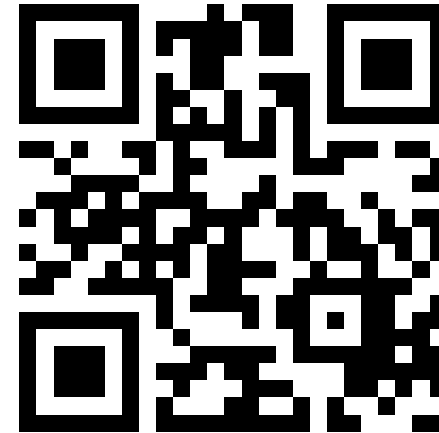
tasks.named<Zip>("scriptsDistZip") {
    archiveFileName.set("GenerateData.zip")
}
```

```
$ ./gradlew scriptsDistZip
```




Envie d'essayer ?

Vous trouverez sur <https://github.com/java-cli-apps>

- Exemple du talk
- Templates  pour démarrer
- Les slides



Autres approches possibles

Utiliser ce qu'on connaît déjà    :

- Une **application compilée** avec Maven ou Gradle
- Une application et son JRE dédié avec `jlink`
- Un **binaire natif** avec GraalVM et `native-image`
- Un **paquet système** (rpm, deb, pkg, ...) avec `jpackage`

Avez-vous des questions ?

Contact

 pyfourmond@gmail.com

 pyfourmond.bsky.social

 <https://www.linkedin.com/in/pyfourmond>

Evènements

 <https://octo.com/evenements#agenda>

Publications

 <https://www.octo.com/publications>