

Partagez simplement vos Java CLI Apps


Pierre-Yves Fourmond

 Human Talks Paris

 11.03.2025

Le contexte

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 ?

- Comparer des fichiers dans un double run
- **Générer des données SQL de test**

Comment adresser ce besoin ?

 Et si on utilisait du scripting ?

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


Il est donc :

- **Livré**
- **Modifiable**

On va vraiment faire du scripting en Java ?



Avec quel outil ?

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

Commençons par 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 : **Compilation** puis **Exécution**
- **✗ Ce n'est pas du scripting**



Passons à Java 11

Une nouvelle façon de lancer du Java :

- La JEP 330 : [Launch Single-File Source-Code Programs](#)

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

```
Hello Human,Talks
```

- 🖱️ On n'appelle plus le compilateur Java
- ✅ **C'est bien du scripting**



#! Scripting avec Java 11 et Linux

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

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

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

```
$ chmod u+x Hello
```

```
$ ./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`




```
$ ./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
```

 **Perdu !**

 **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 pour générer des données SQL de test

```
///usr/bin/java --source 21 --class-path lib/picocli-4.7.6.jar:lib/commons-lang3-3.17.0.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 = "1000")
    int lineCount;

    @Parameters(description = "The file containing the SQL create table request")
    File sqlRequestFile;

    public static 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 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);  
  
█
```

Comment partager l'application ?

👉 Créer une archive :

```
$ 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-lang3-3.17.0.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
```

Pourquoi un Shell de lancement ?

 Référencer les librairies avec un chemin complet :

- Lance le fichier `GenerateData.java`
- Définit le répertoire d'installation utile au `CLASSPATH` :

```
$APP_DIR/lib/picocli-4.7.6.jar:$APP_DIR/lib/commons-lang3-3.17.0.jar
```

Construire le package

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 ?

-  Le code source est modifiable
-  Le packaging est simplifié
-  On ne dépend que du JDK

Pour aller plus loin

- Limitation à un fichier source
 -  [JEP 330](#): Launch **Single-File** Source-Code Programs
 -  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")
        }
    }
}

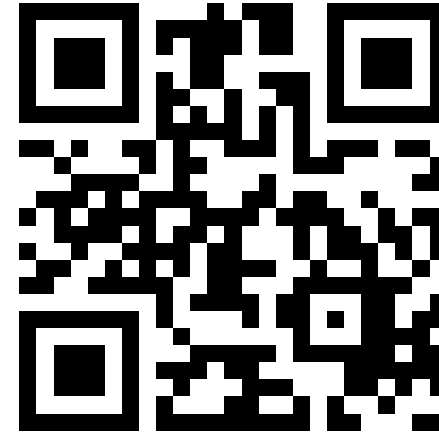
tasks.named<Zip>("scriptsDistZip") {
    archiveFileName.set("GenerateData.zip")
}
```

```
$ ./gradlew scriptsDistZip
```




Envie d'essayer ?

Vous trouverez sur l'organisation  [java-cli-apps](#) :

- Des templates d'applications pour débiter
 - [basic-java-23-quickstart](#)
 - [java-23-quickstart](#)
- L'exemple de code du talk
- Les slides du talk



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`

Merci de votre attention



Pierre-Yves Fourmond

Développeur Back

Consultant Senior (👤 Tribu « Java / Kotlin »)



Membre de Paris JUG

#Java #Linux #Bash #Scripting #CLI

🙋 Avez-vous des questions ?

Slides



Contact

✉ pyfourmond@gmail.com

🦋 pyfourmond.bsky.social

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

Prochains Meetups

Crafting Data Science - 17.03.2025

PyLadies Paris - 25.03.2025

School of Product (à Lille) - 25.03.2025

eXtreme Agilité - 27.03.2025

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

Publications

📖 <https://octo.com/publications>