

# Partagez simplement vos Java CLI Apps

Pierre-Yves Fourmond

 Conf Name

 Conf Date

## Qui suis-je ?



Pierre-Yves Fourmond

Développeur Back

Consultant Senior (Tribu « Java / Kotlin »)




X / GitHub : [@grumpyf0x48](#)

**#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 :

- Générer des données de test
- Des commandes pour la CI

Une étude / Un POC

## Comment adresser ce besoin ?

🤔 Et si on utilisait du scripting ?

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

Il sera :

- **Livré** donc **Modifiable**

## On va faire du scripting en Java ?

Avec quel outil ?

-  JBang!

Peut-on l'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));  
    }  
}
```

```
java-8:$ javac Hello.java
```

```
java-8:$ java Hello Conference Name  
Hello Conference Name
```

- On a deux étapes successives : compilation puis exécution
- **Ce n'est pas du scripting**



## Passons à Java 11

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

```
java-11:$ java Hello.java Conference Name  
Hello Conference Name
```

- 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 :

```
java-11:$ cat Hello
```

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

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

```
java-11:$ ./Hello Conference Name  
Hello Conference Name
```

## 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 avant ?**

## Java 11, Linux et une extension `.java`

😞 Perdu !

```
java-11:$ ./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));  
    }  
}
```

- ```
java-11:$ ./Hello.java Conference Name  
Hello Conference Name
```

## /// **Ceci n'est pas un Shebang !**

Un Shebang :

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

Une façon de lancer une commande :

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

utilisée par **JBang** et Go <https://golangcookbook.com/chapters/running/shebang> :

```
///usr/bin/go run "$0" "$@"; exit $?  
  
package main  
  
func main() {  
    println("Hello Conference Name")  
}
```

## 🤔 Pourquoi un flag `--source ...` ?

Il est utile pour :

- Préciser à `java` qu'on est en mode *source-file*
  - Pas besoin si le fichier source a l'extension `.java`
- Indiquer la version de Java utilisée avec `--enable-preview`

# 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` :

```
~/sources/generate-data$ 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

Lancer l'application avec un environnement modifié :

```
APP_DIR=... $APP_DIR/src/GenerateData.java ...
```

Le classpath devient :

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

 **On est 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
```

## Résoudre les dépendances

 Utilisons Gradle :

```
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
```

## 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

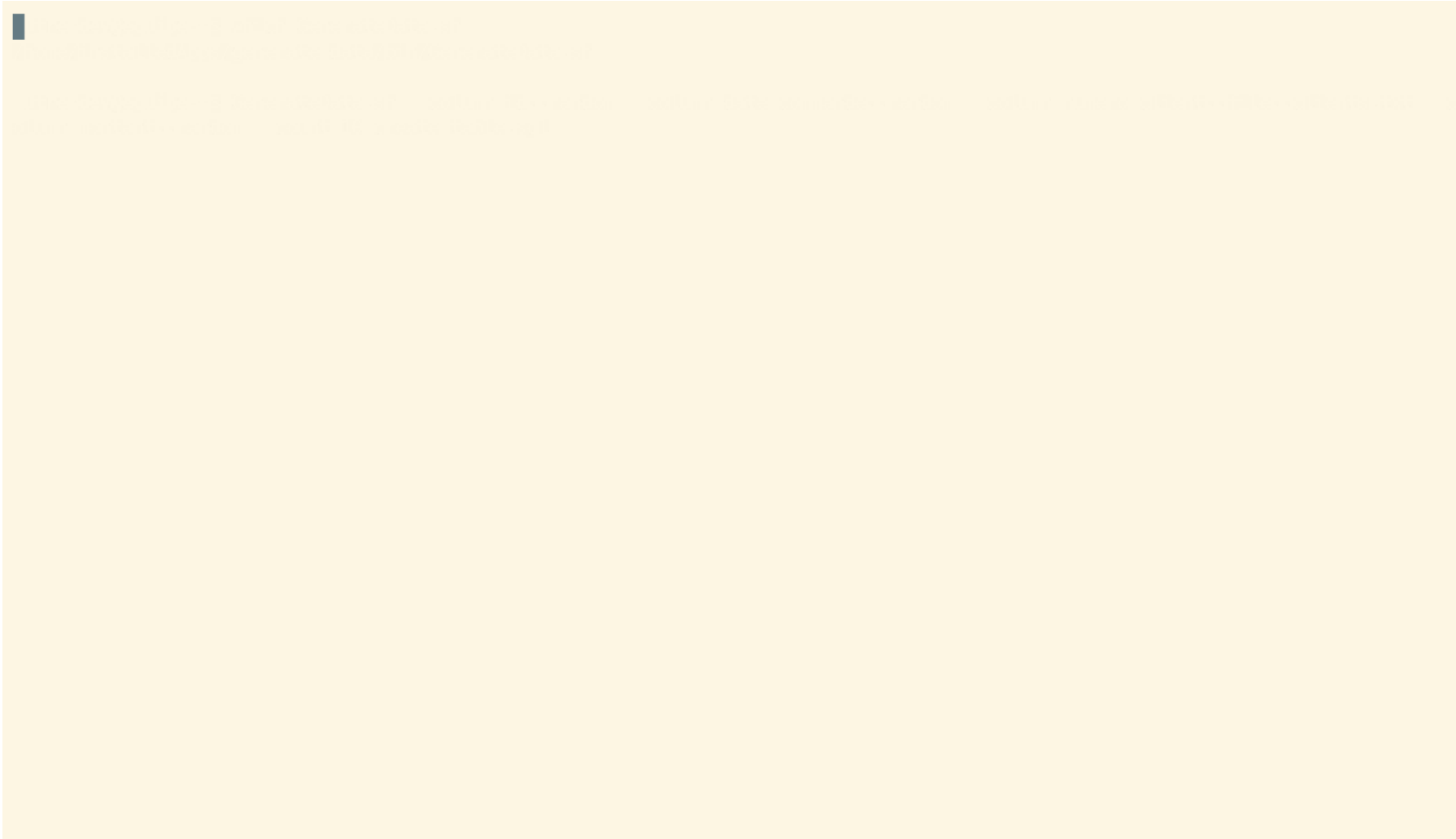
autre-dev@equipe:~$ which GenerateData.sh
/home/installApps/generate-data/bin/GenerateData.sh

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 accessible et modifiable
- Packaging simplifié
- On ne dépend que du JDK

## **Modifier l'application installée**



## Pour aller plus loin

- Limitation à un fichier source
  - 🖱️ Utiliser Java 22 et la JEP 458 : [Launch Multi-File Source-Code Programs](#)
- Gestion des dépendances
  - 📁 Packager sources, dépendances et Shell de lancement avec Gradle
- Le code est compilé à chaque fois
  - 🧑 C'est l'affaire d'une seconde !



## Un template d'application avec make

```
basic-java-22-quickstart:$ tree
```

```
.
├── application.iml
├── bin
│   └── Application.sh
├── lib
│   ├── jemoji-1.3.3.jar
│   ├── ...
├── Makefile
├── src
│   ├── Application.java
│   └── language
│       ├── en
│       │   └── Hello.java
│       └── fr
│           └── Bonjour.java
```

 <https://github.com/java-cli-apps/basic-java-22-quickstart>

## Utilisation du template

Créer un dépôt GitHub à partir de `java-cli-apps/basic-java-22-quickstart.git` :

```
gh repo create <nom-du-nouveau-depot> \  
  --public --template git@github.com:java-cli-apps/basic-java-22-quickstart.git
```

Cloner le dépôt :

```
gh repo clone git@github.com:<votre-compte-github>/<nom-du-nouveau-depot>
```

```
cd <nom-du-nouveau-depot>
```

On est prêt : 😎

```
nom-du-nouveau-depot:$ make  
install          Installer le package de l'application  
package         Packager l'application dans un fichier .zip  
...
```

## Un template d'application avec gradle

```
java-22-quickstart:$ tree
```

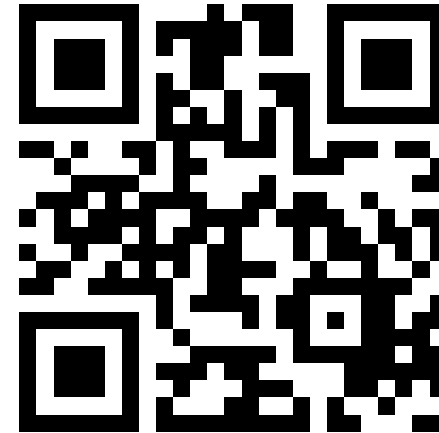
```
.
├── bin
│   └── Application.sh
├── build.gradle.kts
├── gradle
│   ├── libs.versions.toml
│   └── ...
├── gradlew
├── src
│   └── main
│       └── java
│           ├── Application.java
│           └── language
│               ├── en
│               │   └── Hello.java
│               └── fr
│                   └── Bonjour.java
```

 <https://github.com/java-cli-apps/java-22-quickstart>

## Envie d'essayer ?

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

- Exemple de CLI du talk
- Templates d'applications
- Les slides



## Alternatives

- 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**

## Questions



## Publications

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

## Contact

 [fopy@octo.com](mailto:fopy@octo.com)

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

 <https://twitter.com/grumpyf0x48>