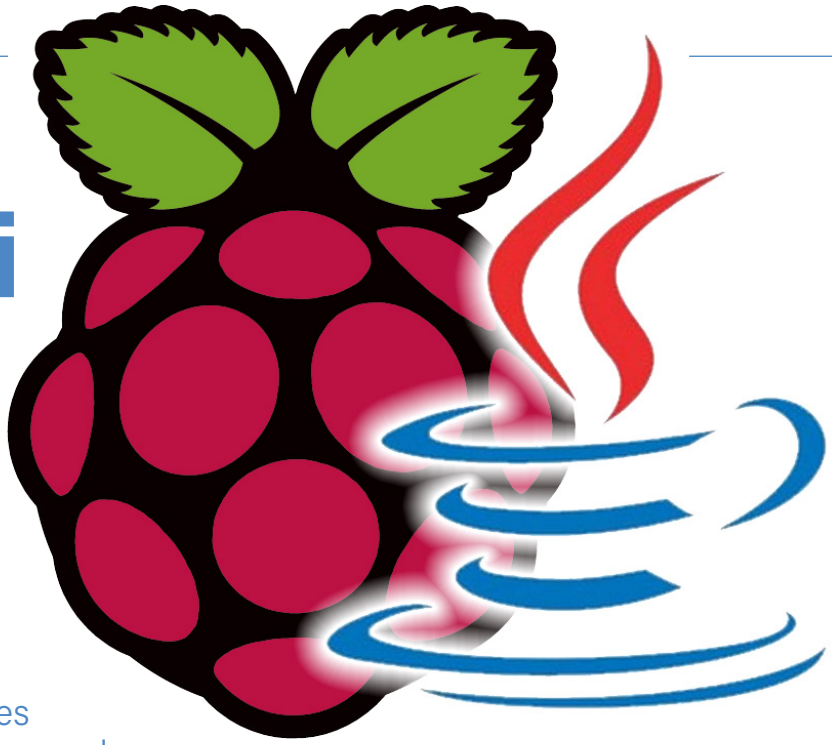
Java sur Raspberry Pi

Partie 1 : les broches GPIO



Frank Delporte (Belgique)

Même si le langage de programmation Java existe depuis longtemps, il reste très adapté au développement de code pour les plates-formes informatiques récentes telles que le Raspberry Pi. Pour démontrer ses capacités, la première partie de cette série fournit quelques informations sur ce langage de programmation et examine comment Java peut commander et lire les broches GPIO.

Brève introduction à Java

Java fait partie des langages de programmation qui commencent à dater. La première version est sortie en 1995 [1], en même temps que JavaScript, PHP et Qt. Python est un peu plus ancien, avec une première version en 1991. Lorsque la marque Java est passée de Sun Microsystems à Oracle, le développement est passé en mode *open source* et a été transféré sur GitHub en 2020 [2]. Depuis 2018, deux nouvelles versions de Java sortent chaque année, ce qui permet de mettre plus régulièrement à jour nos machines avec les correctifs et les nouvelles fonctions. Cela inclut des améliorations qui ciblent les plates-formes embarquées telles que le Raspberry Pi.

On pourrait penser que « Java est mort », mais les multiples projets et conférences (même si elles sont toutes devenues virtuelles en ces temps de Corona) liés à Java sont la preuve que ce langage est toujours vivant. Les principaux contributeurs au développement de Java sont une longue liste de sociétés bien connues, dont Oracle, Microsoft, SAP, Google et d'autres [3]. La rumeur veut également que la moitié du nuage Azure de Microsoft fonctionne en Java ! Aujourd'hui, les distributions Java sont disponibles auprès de différents fournisseurs de logiciels libres (jdk.java.net, adoptopenjdk.net) et commerciaux (Azul, BellSoft, Oracle, et d'autres).

Java sur Raspberry Pi ?!

Le Raspberry Pi n'a-t-il pas été conçu pour être programmé en Python ? Peut-être que oui, mais cela ne veut pas dire que vous ne pouvez pas utiliser d'autres langages. Bien qu'il s'agisse d'un article sur Java, nous ne voulons pas insinuer qu'il est préférable à Python, C ou tout autre langage ! Chaque projet a ses propres exigences

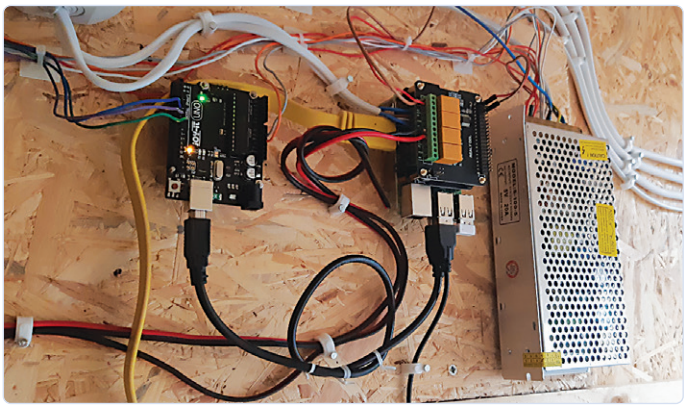


Figure 1. Le cœur du contrôleur à écran tactile pour la cabine acoustique de batterie.

Figure 2. Interface de commande à écran tactile pour le relais et les rubans de LED.

pour lesquelles un langage spécifique pourrait être parfaitement adapté, ou mieux correspondre aux compétences de l'équipe de développement.
Dans mon cas, en tant que développeur Java, cela m'intéressait vraiment de voir si je pouvais appliquer mes connaissances au Raspberry Pi. C'est arrivé après ma première expérience de programmation d'un jeu de pong en Python, qui s'est mal terminée avec une interface utilisateur décevante. Justement, les interfaces utilisateur peuvent être créées avec JavaFX, un projet indépendant [4, 5] qui complète Java avec un canevas pour construire des GUI (*Graphical User Interface*). Il fournit tous les composants de base typiques (boutons, étiquettes, tableaux, graphiques), et beaucoup de bibliothèques gratuites et à code source ouvert peuvent enrichir cette liste.
C'est au moment de construire une interface à écran tactile pour la cabine acoustique de la batterie de mon fils (**fig. 1** et **fig. 2**) que j'ai trouvé l'association parfaite entre Java et le Raspberry Pi. Combiné à une carte relais, un Arduino et quelques rubans de LED, cela constituerait la base de mon expérimentation dans un nouveau monde de programmation Java embarquée.

Se préparer

Pour reproduire les expériences de cet article, vous aurez besoin d'un Raspberry Pi récent avec un processeur ARMv7 ou ARMv8. Les cartes plus anciennes avec un ARMv6 ont une constitution interne différente pour laquelle le Java par défaut ne fonctionnera pas. Pour utiliser Java sur une carte ARMv6, il y a une solution : le JDK Azul Zulu [6]. De plus, pour vous éviter une fastidieuse saisie, tous les exemples de code accompagnant cette série sont disponibles sur un dépôt GitHub [7].
Commencez par préparer une carte SD avec l'outil « Imager » [8] et sélectionnez « Raspberry Pi OS Full (32-bit) » (**fig. 3**). Une fois votre Raspberry Pi lancé, ouvrez le terminal et tapez `java -version` pour vérifier que la version est bien « 11 ». Comme vous le verrez, OpenJDK est préinstallé dans cette version complète de l'OS :

```
$ java -version
openjdk version "11.0.9" 2020-10-20
OpenJDK Runtime Environment (build
   11.0.9+11-post-Raspbian-1deb10u1)
OpenJDK Server VM (build 11.0.9+11-post-Raspbian-
   1deb10u1, mixed mode)
```

Code pour Visual Studio

Vous pouvez développer, tester et exécuter votre code Java sur un PC avant de le transférer sur le Raspberry Pi une fois terminé. Mais il existe une autre approche : Visual Studio Code [9]. Cet EDI (Environnement de Développement Intégré) gratuit de Microsoft possède une longue liste d'extensions qui en font le compagnon idéal pour tout projet de programmation. Il existe une version pour les systèmes ARM à 32 bits (comme Raspberry Pi OS) et ARM à 64 bits (nouveau Raspberry Pi OS, encore en cours de développement) ou même Ubuntu Desktop [10].
Il existe aussi un « Java Extension Pack » qui ajoute plusieurs extensions Java à l'EDI pour en faire un EDI complet pour les développeurs Java (**fig. 4**) !

Essai avec *Hello World!*

Essayons notre tout premier programme Java sur le Raspberry Pi.

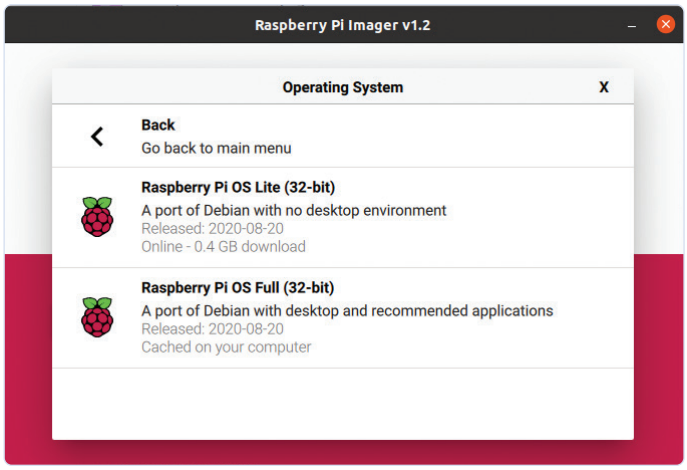


Figure 3. L'outil Raspberry Pi Imager – la meilleure façon de préparer une carte SD.

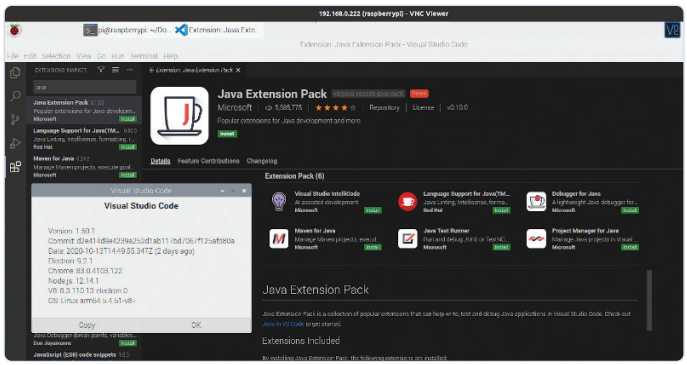


Figure 4. On peut utiliser Visual Studio Code sur Raspberry Pi avec un pack d'extensions Java.

Avec Visual Studio Code, un éditeur de texte, ou dans le terminal, créez un nouveau fichier texte nommé « HelloWorld.java » avec ce contenu :

```
public class HelloWorld {
  public static void main(String args[]) {
    System.out.println("Hello World!");
  }
}
```

Tout notre code appartient à la classe `HelloWorld` qui, par convention, porte le même nom que le fichier. Un programme Java démarre avec la méthode publique `static void main(String args[])`. La seule chose que nous faisons ici est d'imprimer « Hello World! » comme sortie du programme.
Avec Java 11, il est possible d'exécuter ce genre de programme simple sans compiler le code. Dans le terminal, depuis le répertoire contenant votre fichier Java, exécutez la commande `java HelloWorld.java` pour obtenir le résultat suivant :

```
pi@raspberrypi:~/elektor $ java HelloWorld.java
Hello World!
```

Listage 1. Code CPUtemp.java pour lire la température du processeur du Raspberry Pi [7].

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.List;
import java.util.ArrayList;

public class CPUTemp {
    private static final String FILE = “/sys/class/thermal/thermal_zone0/temp”;
    private static final List<Integer> values = new ArrayList<>();

    public static void main(String[] args) throws InterruptedException {
        while(true) {
            checkTemp();
            Thread.sleep(1000);
        }
    }

    private static void checkTemp() {
        try (BufferedReader br = new BufferedReader(new FileReader(FILE))) {
            int value = Integer.valueOf(br.readLine());
            values.add(value);
            int total = values.stream().mapToInt(Integer::valueOf).sum();
            System.out.println(“Now: “ + value
                + “ - Average: “ + (total / values.size())
                + “ - Number of measurements: “ + values.size());
        } catch (Exception ex) {
            System.err.println(“Error during temperature check: “
                + ex.getMessage());
        }
    }
}
```

Certes, `print()` en Python c'est plus court que `System.out.println()` en Java, mais pardonnons ce « péché de jeunesse ».

Lecture de la température de l'UC

De nombreuses valeurs du système, entrées et sorties, sont accessibles via le répertoire `/sys/` du système de fichiers Linux. Les valeurs stockées peuvent simplement être lues comme un fichier texte. Essayons de trouver la température du processeur du Raspberry Pi à partir d'un des fichiers de ce répertoire `sys`. Le nommage dans ce répertoire n'est pas toujours très clair, et il n'est pas toujours simple de déterminer comment interpréter les valeurs, mais c'est un bon point de départ pour apprendre à connaître un peu le système Linux. Exécutez la commande suivante :

```
$ ls -l /sys/
total 0
drwxr-xr-x   2 root root 0 Dec  2 15:44 block
drwxr-xr-x  29 root root 0 Feb 14  2019 bus
drwxr-xr-x  64 root root 0 Feb 14  2019 class
drwxr-xr-x   4 root root 0 Feb 14  2019 dev
drwxr-xr-x  10 root root 0 Feb 14  2019 devices
drwxr-xr-x   3 root root 0 Feb 14  2019 firmware
drwxr-xr-x   8 root root 0 Jan  1  1970 fs
```

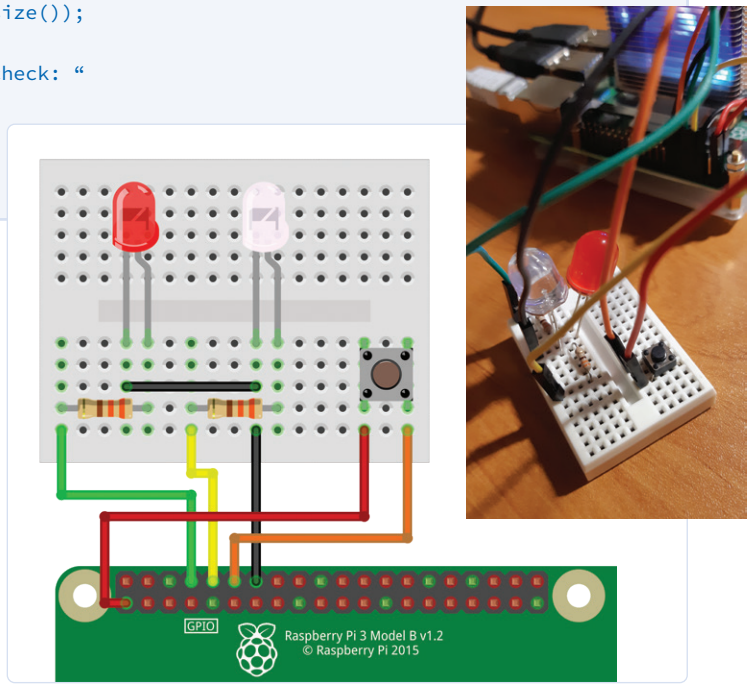


Figure 5. Diagramme Fritzing (à gauche) pour les LED et le bouton-poussoir, avec une photo (à droite).

```
drwxr-xr-x  12 root root 0 Jan  1  1970 kernel
```

Des questions, des commentaires ?
Envoyez un courriel à l'auteur (javaonraspberrypi@webtechie.be) ou contactez Elektor (redaction@elektor.fr).

```
drwxr-xr-x 122 root root 0 Feb 14  2019 module
drwxr-xr-x   2 root root 0 Dec 15 11:39 power
```

On dirait qu'on peut trouver beaucoup d'informations ici ! Essayons d'en approfondir certaines :

```
$ cat /sys/firmware/devicetree/base/cpus/cpu@0/
   compatible
arm,cortex-a72
$ cat /sys/class/net/wlan0/address
dc:a6:32:c5:b7:9d
$ cat /sys/class/bluetooth/hci0/uevent
DEVTYPE=host
$ cat /sys/class/thermal/thermal_zone0/temp
30667
```

La dernière ressemblerait vraiment à une température en degrés si on la divisait par 1000 ! Écrivons un programme simple pour lire cette valeur chaque seconde et calculer la température moyenne. Le code du **listage 1** utilise plus de méthodes Java, Il faut donc commencer par plusieurs importations. Les méthodes `io` sont utilisées pour lire un fichier `sys` comme un fichier texte. Les méthodes `List` et `ArrayList` servent à conserver une liste de toutes les valeurs mesurées. À partir de la méthode `main()`, nous appelons une méthode séparée `checkTemp()` pour acquérir et stocker la température dans la liste. On garde ainsi le code propre, car il vaut mieux isoler chaque fonction dans sa propre méthode. Comme on peut le voir, `Thread.sleep(1000)` est utilisé pour attendre une seconde entre chaque lecture de la température. Dans le cadre de cette méthode, la moyenne est calculée en additionnant la liste des valeurs à l'aide d'un flux. Les flux ont été introduits dans Java 8 et constituent un moyen très puissant de travailler avec des listes d'éléments. Comme auparavant, le nom du fichier est aussi le nom de la classe « CPUTemp.java ».
Comme auparavant, il s'agit d'une simple classe Java sans dépendances supplémentaires, ce qui nous permet de l'exécuter sans avoir à la compiler. La sortie peut être interrompue en utilisant « CTRL+C » :

```
pi@raspberrypi:~/elektor $ java CPUTemp.java
Now: 36998 - Average: 36998 - Number of measurements: 1
Now: 34563 - Average: 35780 - Number of measurements: 2
Now: 35537 - Average: 35699 - Number of measurements: 3
Now: 36024 - Average: 35780 - Number of measurements: 4
Now: 35537 - Average: 35731 - Number of measurements: 5
```

Commander une LED et lire un bouton-poussoir

Créons une autre application Java à un seul fichier pour faire clignoter deux LED et lire l'état d'un bouton. Le schéma de câblage est assez simple – un circuit rudimentaire pour relier deux LED et un bouton à différentes broches GPIO. Le bouton a besoin de 3,3 V, et non de 5,0 V, alors attention à utiliser la bonne broche d'alimentation ! Dans cette configuration, les LED sont connectées aux broches BCM 14 et 15, tandis que le bouton-poussoir est connecté à BCM 18 (**fig. 5**).

Essais avec le Terminal

Grâce aux commandes intégrées au Raspberry Pi OS, on peut accéder aux broches GPIO depuis le terminal. Utilisons-le pour tester notre câblage. Exécutez les commandes suivantes pour configurer la broche 14 en sortie (op) et la placer à l'état haut (dh) ou bas (dl). Essayez la même chose pour l'autre LED (BCM 15) :

```
$ raspi-gpio set 14 op
$ raspi-gpio set 14 dh
$ raspi-gpio set 14 dl
```

On peut utiliser une approche similaire pour tester le bouton en configurant la broche en entrée et en demandant l'état de l'entrée :

```
$ raspi-gpio set 18 ip
$ raspi-gpio get 18
GPIO 18: level=0 fsel=0 func=INPUT pull=DOWN
$ raspi-gpio get 18
GPIO 18: level=1 fsel=0 func=INPUT pull=DOWN
```

Lorsque le bouton est enfoncé, la valeur « `level` » passe à 1.

Idem en Java

Adoptons maintenant la même approche de base que celle utilisée dans « CPUTemp.java » et utilisons les commandes du terminal dans un programme Java en utilisant `Runtime.getRuntime().exec(cmd)`. En ajoutant la classe `Scanner` de saisie de l'utilisateur, on peut aussi acquérir le résultat de la commande et l'utiliser pour vérifier l'état du bouton. Ceci est montré dans le **listage 2**.
Le code utilise une énumération `Actions` pour simplifier le traitement des commandes. Dans sa forme la plus simple, une énumération n'est qu'une liste de noms prédéfinis, mais dans cet exemple, nous ajoutons une valeur à chaque nom. C'est un avantage important des énumérations, car elles permettent de rendre le code beaucoup plus lisible tout en le simplifiant dans de nombreux cas. Comme on peut le voir dans la méthode `doAction`, l'énumération `Action` est utilisée comme paramètre et sa valeur sert à constituer la commande qui doit être exécutée.
Les commentaires dans le code devraient suffire, car c'est du Java simple pour illustrer l'utilisation des broches GPIO en entrée et en sortie. La partie la plus complexe est la méthode `runCommand` qui utilise des méthodes combinées pour exécuter la commande, lire le résultat et gérer les éventuelles erreurs. Si ce n'est pas clair pour l'instant, pas de soucis – ça le sera quand vous exécuterez le code ! Comme cet exemple n'utilise pas non plus de dépendances tierces, on peut l'exécuter sans compilation. Le résultat devrait être évident puisque vous devriez voir vos LED s'allumer et s'éteindre à différents intervalles. Lorsqu'elles cessent de clignoter, vous pouvez appuyer sur le bouton et son état sera examiné dix fois à intervalle d'une seconde. Voici le résultat attendu :

PRODUITS

- **F. Delporte, « Getting Started with Java on the Raspberry Pi » (livre en anglais)**
www.elektor.fr/19292

- **Kit de démarrage du Raspberry Pi 4**
www.elektor.fr/19427

Listage 2. Code RaspiGpio.java pour commander les LED et obtenir l'état du bouton [7].

```
import java.io.IOException;
import java.util.Scanner;

public class RaspiGpio {

    private static final String LED_1 = “14”;
    private static final String LED_2 = “15”;
    private static final String BUTTON = “18”;

    private enum Action {
        OUTPUT_PIN(“op”),
        INPUT_PIN(“ip”),
        DIGITAL_HIGH(“dh”),
        DIGITAL_LOW(“dl”);
        private final String action;
        Action(String action) {
            this.action = action;
        }
        String getAction() {
            return action;
        }
    }

    public static void main(String[] args) throws InterruptedException {
        // Configurer en sorties les deux broches des LED
        doAction(LED_1, Action.OUTPUT_PIN);
        doAction(LED_2, Action.OUTPUT_PIN);

        // Configurer en entrée la broche du bouton
        doAction(BUTTON, Action.INPUT_PIN);

        // Faire clignoter les LED à différents intervalles
        for (int i = 0; i < 20; i++) {
            System.out.println(“Blink loop: “ + i);
            if (i % 2 == 0) {
                doAction(LED_1, Action.DIGITAL_HIGH);
            } else {
                doAction(LED_1, Action.DIGITAL_LOW);
            }
```

```
$ java RaspiGpio.java
Executing: raspi-gpio set 14 op
Executing: raspi-gpio set 15 op
Executing: raspi-gpio set 18 ip
Blink loop: 0
Executing: raspi-gpio set 14 dh
Executing: raspi-gpio set 15 dh
Blink loop: 1
Executing: raspi-gpio set 14 dl
Executing: raspi-gpio set 15 dl
...
Executing: raspi-gpio get 18
Button check 0: GPIO 18: level=0 fsel=0 func=INPUT pull=DOWN
 - PUSHED: false
...
Button check 3: GPIO 18: level=1 fsel=0 func=INPUT pull=DOWN
 - PUSHED: true
...
```

Préparez-vous pour la prochaine fois !

Avec quelques notions de base en poche, nous avons réussi à couvrir une petite introduction à Java sur le Raspberry Pi. Pour améliorer votre compréhension, n'hésitez pas à vous amuser avec le code en exemple et à lire la documentation en suivant les liens fournis. Nous espérons que cela simplifiera les étapes suivantes du prochain article, où nous développerons une application Java complète pour relier nos broches GPIO à une page web.

(200617-04)

Contributeurs
Idée, texte et illustrations : **Frank Delporte**
Rédaction : **Stuart Cording**
Mise en page : **Giel Dols**
Traduction : **Denis Lafourcade**

```
            if (i % 3 == 0) {
                doAction(LED_2, Action.DIGITAL_HIGH);
            } else {
                doAction(LED_2, Action.DIGITAL_LOW);
            }
            Thread.sleep(500);
        }
        doAction(LED_1, Action.DIGITAL_LOW);
        doAction(LED_2, Action.DIGITAL_LOW);

        // Examiner 10 fois l'état du bouton
        for (int j = 0; j < 10; j++) {
            String result = runCommand("raspi-gpio get " + BUTTON);
            System.out.println("Button check " + j
                + ": " + result
                + " - PUSHED: " + (result.contains("level=1")));
            Thread.sleep(1000);
        }
    }

    private static void doAction(String pin, Action action) {
        runCommand("raspi-gpio set " + pin + " " +
              action.getAction().toLowerCase());
    }

    private static String runCommand(String cmd) {
        System.out.println("Executing: " + cmd);
        Scanner s = null;
        try {
            s = new Scanner(Runtime.getRuntime().exec(cmd).getInputStream())
                .useDelimiter("\\A");
            return s.hasNext() ? s.next() : "";
        } catch (Exception ex) {
            System.err.println("Error during command: " + ex.getMessage());
            return "";
        } finally {
            if (s != null) {
                s.close();
            }
        }
    }
}
```

LIENS

[1] **Andrew Binstock, « Java's 20 Years Of Innovation », Forbes** : http://bit.ly/3qeIpVu

[2] **Projet OpenJDK** : https://github.com/openjdk

[3] **Liam Tung, Oracle, « Programming language Java 14 is out with these 16 major feature improvements », ZDNet** : http://zd.net/35wVW2O

[4] **Projet OpenJFX** : https://github.com/openjfx

[5] **OpenJFX** : https://openjfx.io/

[6] **Frank Delporte, « How to install and use Java 11 and JavaFX 11 on Raspberry Pi boards with ARMv6 processor »** : http://bit.ly/35yRrFc

[7] **Code des exemples sur GitHub** : http://bit.ly/3i9bP4v

[8] **Outil Raspberry Pi Imager** : http://bit.ly/3i58seU

[9] **Frank Delporte, « Visual Studio Code on the Raspberry Pi (with 32 and 64-bit OS) »** : http://bit.ly/3qeh9GN

[10] **Téléchargement de Visual Studio Code pour Arm** : http://bit.ly/2LqUng3