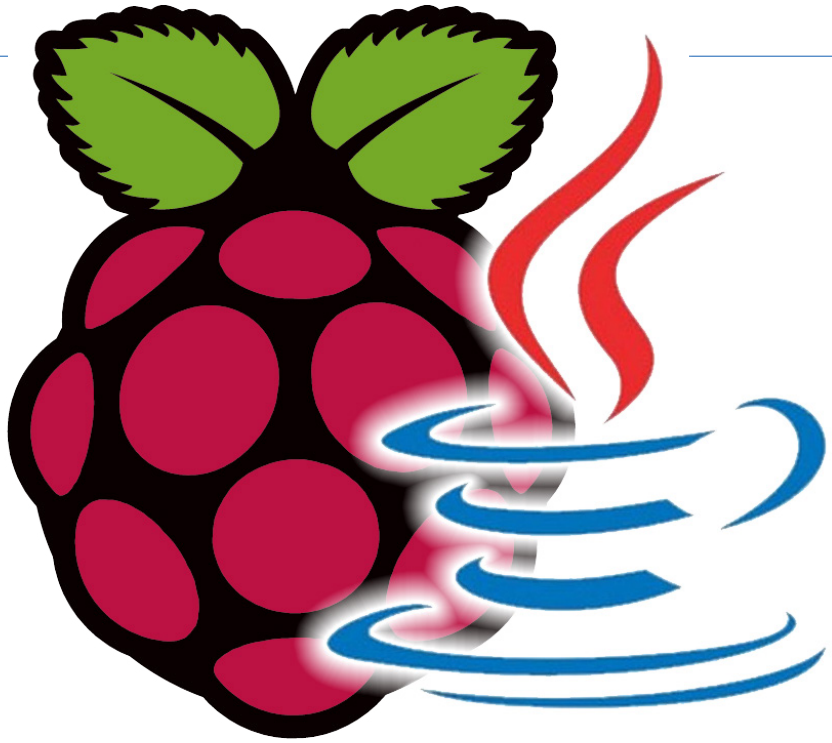
Java op de Raspberry Pi

deel 1: GPIO's



Frank Delporte (België)

De programmeertaal Java bestaat al lang, maar hij is nog steeds heel geschikt voor het ontwikkelen van code voor moderne rekenplatforms zoals de Raspberry Pi. In deze serie willen we de mogelijkheden tonen. In het eerste deel geven we enige achtergrondinformatie over de taal en gaan we onderzoeken hoe Java GPIO-pinnen kan inlezen en aansturen.

Java: een korte inleiding

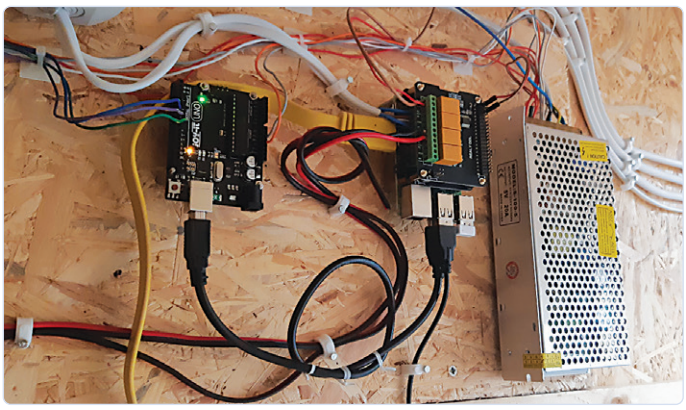
Java is één van die programmeertalen die al heel lang meedoen. De eerste versie kwam in 1995 uit [1], hetzelfde jaar waarin JavaScript, PHP en Qt het licht zagen. Python is iets ouder, dat deed zijn intrede in 1991.

Hoewel de handelsnaam 'Java' van Sun Microsystems naar Oracle is gegaan, vindt de ontwikkeling sinds 2020 via GitHub als open source-project plaats [2]. Sinds 2018 worden jaarlijks twee nieuwe releases van Java uitgebracht, zodat correcties en nieuwe features regelmatig onze machines bereiken. Dat geldt ook voor verbeteringen voor embedded platforms zoals de Raspberry Pi.

Bij sommigen lijkt het gevoel te bestaan dat 'Java dood is', maar de vele conferenties (ook al zijn die in deze Corona-tijden virtueel) en Java-gerelateerde projecten bewijzen dat de taal nog altijd springlevend is. De belangrijkste bijdragen aan de ontwikkeling van Java komen van een lange lijst van bekende bedrijven zoals Oracle, Microsoft, SAP, Google en anderen [3]. Er circuleert ook een gerucht dat de helft van de Microsoft Azure-cloud onder Java draait! Java-distributies zijn tegenwoordig beschikbaar bij verschillende open source (jdk.java.net, adoptopenjdk.net) en commerciële leveranciers (Azul, BellSoft, Oracle en anderen).

Java op de Raspberry Pi?!

Maar de Raspberry Pi was toch ontworpen om er Python op te draaien? Misschien, maar dat betekent niet dat hij geen andere talen aan kan. En, hoewel dit artikel over Java gaat, willen we zeker niet beweren dat Java beter is dan Python, C of een andere taal! Ieder project stelt zijn eigen eisen, waar een specifieke taal



Figuur 1. Het hart van het aanraakscherm van de drumcabine.

Figuur 2. De aanraakscherm-interface bestuurt de relais en de LED-strips.

de perfecte oplossing voor kan zijn of waar het ontwikkelteam de meeste ervaring mee heeft.
Zelf ben ik Java-ontwikkelaar en ik was nieuwsgierig of ik mijn kennis zou kunnen toepassen op de Raspberry Pi. Ik kwam op dat idee toen mijn eerste poging mislukte om in Python een pong-spel te bouwen. Ik vond de gebruikersinterface erg tegenvallen. Gelukkig kun je gebruikersinterfaces genereren met JavaFX, een onafhankelijk project [4, 5] dat Java uitbreidt met een framework om GUI's (grafische gebruikersinterfaces) te bouwen. Het biedt alle bekende basiscomponenten (drukknoppen, labels, tabellen, grafieken) en er is een lange lijst van gratis open source-bibliotheken die nog meer mogelijkheden bieden.
Ik vond de perfecte aansluiting tussen Java en de Raspberry Pi, toen ik een touchscreen-interface wilde bouwen voor de drumcabine van mijn zoon (**figuren 1 en 2**). In combinatie met een relaiskaart, een Arduino en enkele LED-strips werd dit de basis voor mijn eerste stappen in een nieuwe wereld van embedded programmeren in Java.

De voorbereiding

Voor de experimenten die ik in dit artikel met u wil delen, hebt u een recente Raspberry Pi met een ARMv7- of ARMv8-processor nodig. Oudere boards met een ARMv6 hebben een andere interne layout waar standaard-Java niet op werkt. Als u Java wilt gebruiken op zo'n ARMv6-kaart, is er wel een oplossing die met de Azul Zulu JDK [6] werkt. En als u geen zin hebt om veel te typen, kun u gebruik maken van een GitHub-repository [7] waar alle codevoorbeelden uit deze serie beschikbaar zijn.
Bereid als eerste de SD-kaart voor met het 'Imager'-tool [8] en kies "Raspberry Pi OS Full (32-bit)" (**figuur 3**). Als u de Raspberry Pi hebt gestart, opent u een terminal-venster en typt u `java -version` om te controleren of u versie '11' hebt. U zult zien dat OpenJDK is voorgeïnstalleerd in deze volledige versie van het OS:

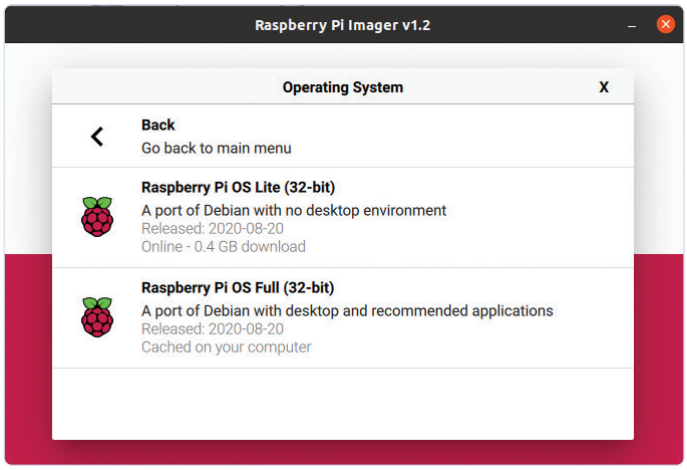
```
$ java -version
openjdk version "11.0.9" 2020-10-20
OpenJDK Runtime Environment (build
   11.0.9+11-post-Raspbian-1deb10u1)
OpenJDK Server VM (build 11.0.9+11-post-Raspbian-
   1deb10u1, mixed mode)
```

Visual Studio Code

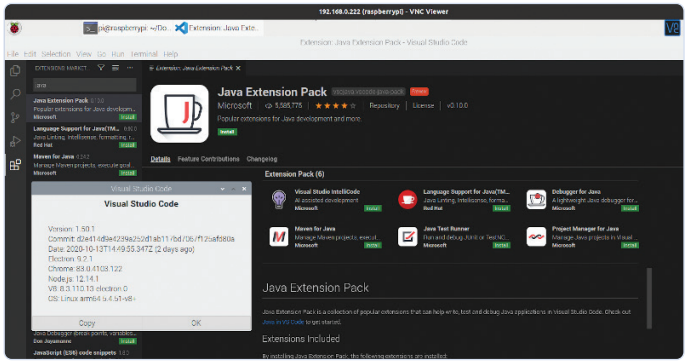
U kunt uw Java-code ontwikkelen, testen en uitvoeren op een PC voordat u die overbrengt naar de Raspberry Pi. Maar er is ook een andere manier: Visual Studio Code [9]. Er is voor deze gratis IDE (Integrated Development Environment) van Microsoft een groot aantal uitbreidingen beschikbaar, wat het tot een perfect gereedschap maakt voor elk programmeerproject. Er is een versie beschikbaar voor 32-bits ARM-systemen (zoals Raspberry Pi OS), voor 64-bits ARM-systemen (het nieuwe Raspberry Pi OS, dat nog in ontwikkeling is) en zelfs voor Ubuntu Desktop [10]. Er bestaat ook een 'Java Extension Pack' dat verschillende Java-uitbreidingen toevoegt aan de IDE zodat het een volledige Java-IDE (**figuur 4**) wordt!

Experimenteren met Hello World!

Laten we ons eerste Java-programma op de Raspberry Pi proberen. Met Visual Studio Code, een teksteditor of een terminalvenster maken we een nieuw tekstbestand aan met de naam "HelloWorld.java" en met de volgende inhoud:



Figuur 3. Raspberry Pi Imager tool: de beste manier om een SD-kaart voor te bereiden.



Figuur 4. Visual Studio Code is te gebruiken op een Raspberry Pi en heeft een Java Extension Pack in de aanbieding.

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

Al onze code maakt deel uit van de klasse `HelloWorld`. De conventie is dat die klasse dezelfde naam heeft als het bestand. Een Java-programma start met de methode `public static void main(String args[])`. Het enige dat we hier doen, is "Hello World!" afdrukken als uitvoer van het programma. Omdat we met Java 11 werken, kunnen we zo'n eenvoudig programma draaien zonder dat we het hoeven te compileren. Geef in de terminal, in de map waar uw Java-bestand staat, het commando `java HelloWorld.java`. U krijgt dan het volgende resultaat:

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

Natuurlijk is `print()` in Python korter dan `System.out.println()` in Java; laten we dat echter beschouwen als een 'jeugdzonde'.

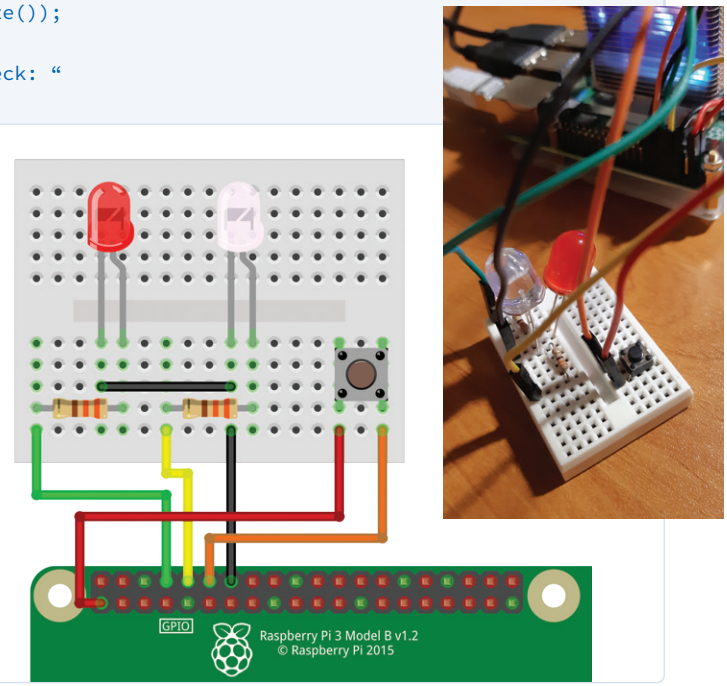
Listing 1. De code voor CPUtemp.java om de processortemperatuur van de Raspberry Pi in te lezen [7].

```
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());
        }
    }
}
```

De temperatuur van de CPU uitlezen

Veel systeemparameters, inputs en outputs, zijn toegankelijk via de map `/sys/` van het Linux-bestandssysteem. De opgeslagen waarden kunnen als gewoon tekstbestand worden gelezen. Laten we eens kijken of we de temperatuur van de Raspberry Pi-processor kunnen vinden in één van de bestanden in de map `sys`. De naamgeving in het bestandssysteem is niet altijd even duidelijk, en het is ook niet altijd eenvoudig om de uitgelezen waarden te interpreteren, maar het is een goede manier om iets te weten te komen over het Linux-systeem. Voer het volgende commando in:

```
$ 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
drwxr-xr-x  12 root root 0 Jan  1  1970 kernel
drwxr-xr-x 122 root root 0 Feb 14  2019 module
drwxr-xr-x   2 root root 0 Dec 15 11:39 power
```



Figuur 5. Fritzing-layout (links) voor de LED's en de drukknop. Rechts een foto van de opbouw.

Vragen of opmerkingen?
Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de auteur of naar de redactie van Elektor, via redactie@elektor.com.

Zo te zien is daar een hoop informatie te vinden! Laten we dat eens nader onderzoeken.

```
$ 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
```

Dat laatste zou best eens een temperatuur in graden Celsius kunnen zijn als we het door 1000 delen! Met een eenvoudig programma kunnen we die waarde elke seconde uitlezen en de gemiddelde temperatuur berekenen.
De code in **listing 1** maakt gebruik van wat meer Java-methoden dus moeten we beginnen met meerdere imports. De `io`-methoden worden gebruikt om een `sys`-file uit te lezen als een tekstfile. `List` en `ArrayList` gebruiken we om een lijst van alle gemeten waarden bij te houden. We roepen vanuit de methode `main()` een aparte methode `checkTemp()` aan om de temperatuur op te halen en die op te slaan in de lijst. Zo houden we de code netjes. Het is een goede gewoonte om elke functionaliteit te isoleren in een eigen methode. Zoals u ziet gebruiken we `Thread.sleep(1000)` om één seconde te wachten tussen de temperatuurcontroles. In onze methode wordt het gemiddelde berekend door de som van de lijst van waarden te berekenen met behulp van een stream. Streams werden geïntroduceerd in Java 8 en vormen een uiterst krachtige manier om met reeksen items te werken. Net als bij het eerste programma is de naam van de klasse gelijk aan de bestandsnaam: "CPUTemp.java". Ook dit is een eenvoudige Java-klasse zonder extra afhankelijkheden, dus we kunnen hem uitvoeren zonder te compileren. U kunt de uitvoer stoppen met '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
```

Een LED aansturen en een drukknop inlezen

We maken nog een Java-toepassing die uit één bestand bestaat. Deze kan twee LED's aan- en uitschakelen en de toestand van een drukknop inlezen. De bedrading is heel eenvoudig: we sluiten gewoon twee LED's en één drukknop aan, elk op een eigen GPIO. De drukknop werkt op 3,3 V, niet op 5,0 V, dus let op dat u de juiste voedingspin gebruikt! De LED's zijn verbonden met pinnen BCM 14 en 15 en de drukknop zit op BCM 18 (**figuur 5**).

Testen met de terminal

We kunnen de GPIO's aansturen vanuit de terminal met de ingebouwde commando's van Raspberry Pi OS. Daarmee testen we onze bedrading. Voer de volgende commando's in om pin 14 te configureren als uitgang (op) en maak hem hoog (dh) of laag (dl). Doe hetzelfde voor de andere LED (BCM 15):

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

Op soortgelijke wijze kunnen we de drukknop testen: we configureren de pin als een ingang en vragen diens toestand op:

```
$ 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
```

Als de drukknop is ingedrukt, krijgt 'level' de waarde 1.

En dan nu in Java

Op dezelfde manier als in "CPUTemp.java" gebruiken we de terminal-commando's in een Java-programma met behulp van `Runtime.getRuntime().exec(cmd)`. Met de gebruikers-inputklasse `Scanner` kunnen we het resultaat van het commando inlezen en dat gebruiken om de toestand van de drukknop te controleren. Dit is te zien in **listing 2**.
De code maakt gebruik van een enum `Action` om het verwerken van de commando's te vergemakkelijken. In zijn eenvoudigste vorm is een enum niet meer dan een lijst van voorgedefinieerde namen, maar in dit voorbeeld voegen we aan elke naam een waarde toe. Dat is een groot voordeel van het gebruik van enums, want ze maken de code veel beter leesbaar en vaak ook nog eenvoudiger. In de methode `doAction` wordt de enum `Action` gebruikt als een parameter en wordt een commando gegenereerd afhankelijk van de waarde van die parameter.
Het commentaar in de code zou voldoende moeten zijn om alles duidelijk te maken. Dit is gewoon eenvoudige Java-code om te illustreren hoe we de ingangs- en uitgangs-GPIO's kunnen gebruiken. Het ingewikkeldste deel is de methode `runCommand` die gebruik maakt van een combinatie van methoden om het commando uit te voeren, het resultaat te lezen en eventuele fouten af te handelen. Als het niet meteen helder is bij het lezen, dan komt dat wel goed als u de code gaat draaien!
Ook in dit voorbeeld wordt geen gebruik gemaakt van externe afhankelijkheden, dus we kunnen het uitvoeren zonder te compileren. De uitvoer geeft duidelijk aan wat er gebeurt, als het goed is ziet u LED's knipperen met verschillende tussenpozen. Als de LED's ophouden met knipperen, kunt u op de knop drukken, die wordt tien keer ingelezen met een interval van1 seconde. De verwachte uitvoer is als volgt:

Listing 2. Code voor RaspiGpio.java voor het besturen van de LED's en het inlezen van de drukknop [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 {
        // Configure the two LEDs as output pins
        doAction(LED_1, Action.OUTPUT_PIN);
        doAction(LED_2, Action.OUTPUT_PIN);

        // Configure the button as input pin
        doAction(BUTTON, Action.INPUT_PIN);

        // Blink the LEDs with different interval
        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
...
```

Voorbereiding voor de volgende aflevering

We hebben nu wat basiskennis opgedaan als introductie tot Java op de Raspberry Pi. Ga vooral lekker experimenteren met de voorbeeldcode en lees alvast wat achtergrondinformatie door de links te volgen. Hopelijk maakt dat de volgende stappen in het tweede artikel beter te begrijpen. We gaan dan een volledige Java-toepassing maken en onze GPIO's koppelen met een internetpagina.

200617-03

Een bijdrage van
Idee, tekst en afbeeldingen:
Frank Delporte

Redactie: **Stuart Cording**
Vertaling: **Evelien Snel**
Layout: **Giel Dols**

```
            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);

        // Read the button state 10 times
        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();
            }
        }
    }
}
```

WEBLINKS

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

[2] **OpenJDK project**: 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] **OpenJFX project**: 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] **Bijbehorende codevoorbeelden op GitHub**: http://bit.ly/3i9bP4v

[8] **Raspberry Pi Imager tool**: 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] **Visual Studio Code downloads voor ARM**: http://bit.ly/2LqUng3