Java et le Raspberry Pi 3 B – Eyrolles 2019

Boichat Jean-Bernard – Dernières corrections: 20 Décembre 2018

Programmer en Java avec un Raspberry Pi

Pistes et solutions des exercices

Certains des exercices seront de véritables petits projets informatiques. Nous ne donnerons donc que des pistes et de nombreuses références à du code que nous trouverons dans le livre.

Tous les fichiers Java ou Python se trouvent dans le fichier Exercices.zip dans un répertoire par chapitre. Dans ces répertoires il y a parfois aussi un fichier texte par exercice.

L'utilisation de Notepad++ sous Windows est impératif pour cet ouvrage.
Des outils comme le Blocnotes ou WordPad sont inutilisables dans la plupart des cas.
Notepad++ va nous garantir que nos fichiers texte préparés et/ou modifiés sous Windows et/ou le Raspberry Pi fonctionneront dans tous les cas.

Les exercices en Java nécessitent en principe de travailler dans Eclipse. Il faudra donc les utiliser soit dans les projets existants, voir dans de nouveaux. Cette dernière démarche est conseillée pour notre apprentissage d'Eclipse et de Java. Un seul projet pour plusieurs chapitres d'exercice irait aussi.

La création de projets et le mise en place des librairies sont expliqués dans le livre.

Les fichiers .java, .py ou autres fichiers texte, ne sont pas dans ce PDF. Ils sont livrés séparemment dans le fichier exercices.zip dans une structure de sous-répertoire par chapitre. Les commentaires présents ici se trouve aussi dans ces sous-répertoires et dans des fichiers `exerciceN.txt`.

Certains exercices n'ont pas de fichier `.txt` car le code source suffit.

Chapitre 1 - Eclipse et Java sous Windows

Exercice 1

Fichier classe Java: **`BonjourMonsieur.java`**

Exercice 2

Difficile de trouver de bons tutoriels sur le Web.

Mais est-ce nécessaire!?

Un des aspects les plus importants est le fait que des onglets sont disponibles. Nous voyons tous les fichiers en travail ou récents. Au prochain démarrage, ils seront toujours là. C'est très pratique.

De temps à autre, nous cliquerons sur les petites croix des onglets pour fermer les fichiers les plus anciens. Les onglets sont déplaçables dans la barre supérieure.

Comme décrit dans le livre, copier / coller entre différentes sources marchent très bien, même avec Linux dans une fenêtre Putty sur le Raspberry Pi.

Chapitre 2 - Maîtriser Eclipse d'abord

Exercice 1

Se référer au chapitre 2 où le Refactor est expliqué.

Fichier classe Java: **`PasAPas1.java`**

Exercice 2

Fichier classe Java: **`JavaHome.java`**

Chapitre 3 - Installation du Raspberry Pi 3 B

Exercice 1

Suivre les instructions du chapitre 3.
Il serait judicieux de restaurer l'image sur une seconde carte microSD.

Chapitre 4 - Putty et WinScp

Exercice 1

```
cp HelloWorld.class HelloWorld2.class
```

Possible, mais HelloWorld2.class ne pourra être exécuté.

```
mkdir repdetest
cp HelloWorld.class repdetest
```

Copie dans un répertoire :

```
cp *.class repdetest
```

va copier tous les fichiers `.class` dans un répertoire créé ci-dessus.

Chercher sur Internet:

```
$ cp -R /home/pi /home/back
$ cp -f /home/pi /home/back
```

Exercice 2

`cd /home/pi`

Peut-être pas nécessaire: entrer `pwd`

`mkdir cpTest`

C'est aussi possible avec WinScp:

Dans la fenêtre de la liste sur `/home/pi`, s'y déplacer si nécessaire, sélectionner avec le bouton droite de la souris : Nouveau et répertoire.

Exercice 3

Faire ce travail avec WinScp avec les bons répertoires à gauche et à droite

Exercice 4

Dans WinScp, aller dans la liste de gauche et sélectionner par exemple le disque `D:` en allant sur sa racine.

Dans la fenêtre de la liste de `D:`, sélectionner avec le bouton droite de la souris : Nouveau et répertoire

Entrer: `TestJava`

Transférer `HelloWorld` du panneau de droite (Raspberry Pi) à gauche (PC)

Exercice 5

Par exemple

`D:`

`cd TestJava`

`java HelloWorld`

Chapitre 5 - Le Raspberry Pi et son port GPIO

Exercice 1

Avant de spécifier et de déterminer quelles nouvelles broches seront utilisées, il faudra consulter l'annexe.

Nous devrons d'abord créer le schéma tel qu'il est défini.

En cas de difficulté, contactez-moi et je pourrais vous fournir le fichier schéma si nécessaire. Cependant, le faire soi-même reste tout de même une nécessité pour vraiment maîtriser l'outil.

Le déplacement des broches se fera en cliquant à l'endroit du fil connecté sur le port GPIO, en gardant le bouton gauche de la souris pressé avant le déplacement sur une nouvelle broche.

Les couleurs resteront les mêmes.

Il est ensuite possible d'utiliser le script blink2led2.py du chapitre 6, d'y modifier le numéro des pins (broches) et de l'exécuter sur le Raspberry Pi.

Chapitre 6 - Python pour vérifier les fonctions GPIO

Exercice 1

En lisant le morceau de code Java inclus dans ce chapitre, nous remarquerons que les deux leds clignotent en parallèle.

Nous ferons l'édition dans Notepad++, le transfert avec WinScp et les tests avec Putty.

Nous utiliserons le nom blink2leds_ex1.py et le déposerons à l'endroit habituel des scripts Python sur le Raspberry Pi.

Exercice 2

Dans le script, il faudra modifier le numéro des broches qui étaient

LedPin1 = 12
LedPin2 = 15

Ce sont le numéro des broches physiques.

Chapitre 7 - Le Pi4J sur le Raspberry Pi 3 B

Exercice 1

grep -nr '/opt/pi4j/examples/' -e 'toggle'

Exercice 2

Pour pouvoir comprendre et effectuer les instructions de cet exercice ... il faudra s'accrocher et pas mal feuilleter le livre. Mais ça marche.

Sur le Raspberry Pi: `GpioOutputExample.java` est un bon exemple

```
pi@raspberrypi:~/python $ grep -nr '/opt/pi4j/examples/' -e 'toggle'
/opt/pi4j/examples/GpioOutputExample.java:141:        // toggle the
current state of gpio pin (from HIGH to LOW)
....
pi@raspberrypi:~/python $ cd
pi@raspberrypi:~ $ mkdir testawk
```

On en prend plus:

```
pi@raspberrypi:~ $ ls /opt/pi4j/examples/G*.java | awk '{print "cp "
$1 " /home/pi/testawk" }'
cp /opt/pi4j/examples/GpioInputAllExample.java /home/pi/testawk
cp /opt/pi4j/examples/GpioInputExample.java /home/pi/testawk
cp /opt/pi4j/examples/GpioListenAllExample.java /home/pi/testawk
cp /opt/pi4j/examples/GpioListenExample.java /home/pi/testawk
cp /opt/pi4j/examples/GpioOutputExample.java /home/pi/testawk
cp /opt/pi4j/examples/GpioTest.java /home/pi/testawk
pi@raspberrypi:~ $ ls /opt/pi4j/examples/G*.java | awk '{print "cp "
$1 " /home/pi/testawk" }' >copy.sh
pi@raspberrypi:~ $ chmod +x copy.sh
```

```
pi@raspberrypi:~ $ ./copy.sh
pi@raspberrypi:~ $ ls -lrt testawk/
total 48
-rw-r--r-- 1 pi pi 4783 sep  6 10:28 GpioInputAllExample.java
-rw-r--r-- 1 pi pi 5041 sep  6 10:28 GpioInputExample.java
-rw-r--r-- 1 pi pi 6372 sep  6 10:28 GpioListenAllExample.java
-rw-r--r-- 1 pi pi 5615 sep  6 10:28 GpioListenExample.java
-rw-r--r-- 1 pi pi 7252 sep  6 10:28 GpioOutputExample.java
-rw-r--r-- 1 pi pi 6470 sep  6 10:28 GpioTest.java
pi@raspberrypi:~ $ cd testawk/
pi@raspberrypi:~/testawk $
```

Pour voir le fichier (view == vi en lecture seul): view GpioOutputExample.java

Et c'est parti pour la compilation sur le Raspberry Pi:

```
javac -classpath '.:/opt/pi4j/lib/*'  GpioOutputExample.java
pi@raspberrypi:~/testawk $ ls -l *.class
-rw-r--r-- 1 pi pi 1481 sep  6 10:42 GpioOutputExample$1.class
-rw-r--r-- 1 pi pi 3729 sep  6 10:42 GpioOutputExample.class
```

Et finalement exécuter la classe qui marche à merveille:

```
pi@raspberrypi:~/testawk $ java -Dpi4j.linking=dynamic -classpath
'.:/opt/pi4j/lib/*'  GpioOutputExample
************************************************************
************************************************************

                 <-- The Pi4J Project -->

                    GPIO Output Example

************************************************************
************************************************************

-----------------------------
```

```
|    PRESS CTRL-C TO EXIT    |
------------------------------
 ... Successfully provisioned output pin: "My Output" <GPIO 1>
-------------------------------------------------------------------
|  The GPIO output pin states will cycle HIGH and LOW states now.  |
-------------------------------------------------------------------
--> ["My Output" <GPIO 1>] state was provisioned with state = HIGH
Setting output pin state is set to LOW.
 --> GPIO PIN STATE CHANGE: "My Output" <GPIO 1> = LOW
Setting output pin state from LOW to HIGH.
 --> GPIO PIN STATE CHANGE: "My Output" <GPIO 1> = HIGH

Toggling output pin state from HIGH to LOW.
 --> GPIO PIN STATE CHANGE: "My Output" <GPIO 1> = LOW

Pulsing output pin state HIGH for 1 second.
 --> GPIO PIN STATE CHANGE: "My Output" <GPIO 1> = HIGH
 --> GPIO PIN STATE CHANGE: "My Output" <GPIO 1> = LOW
Blinking output pin state between HIGH and LOW for 3 seconds with a
blink rate of 250ms.
 --> GPIO PIN STATE CHANGE: "My Output" <GPIO 1> = HIGH
 --> GPIO PIN STATE CHANGE: "My Output" <GPIO 1> = LOW
 --> GPIO PIN STATE CHANGE: "My Output" <GPIO 1> = HIGH
 --> GPIO PIN STATE CHANGE: "My Output" <GPIO 1> = LOW
 --> GPIO PIN STATE CHANGE: "My Output" <GPIO 1> = HIGH
 --> GPIO PIN STATE CHANGE: "My Output" <GPIO 1> = LOW
```

```
 --> GPIO PIN STATE CHANGE: "My Output" <GPIO 1> = HIGH
 --> GPIO PIN STATE CHANGE: "My Output" <GPIO 1> = LOW
 --> GPIO PIN STATE CHANGE: "My Output" <GPIO 1> = HIGH
 --> GPIO PIN STATE CHANGE: "My Output" <GPIO 1> = LOW
 --> GPIO PIN STATE CHANGE: "My Output" <GPIO 1> = HIGH
 --> GPIO PIN STATE CHANGE: "My Output" <GPIO 1> = LOW
 --> GPIO PIN STATE CHANGE: "My Output" <GPIO 1> = HIGH
 --> GPIO PIN STATE CHANGE: "My Output" <GPIO 1> = LOW
************************************************************
                           GOODBYE
************************************************************
pi@raspberrypi:~/testawk $
```

Notre led verte, si correctement disponible comme décrit dans le livre, clignotera!

Avec la commande suivante, se sera la led rouge!

```
java -Dpi4j.linking=dynamic -classpath '.:/opt/pi4j/lib/*'
GpioOutputExample -p 3
```

Il faut regarder le code, `-p` pour le Pin en numérotation GPIO.

Chapitre 8 - Le Raspberry Pi clignote avec le Pi4J sous Eclipse

Exercice 1

Sur le Raspberry Pi:

```
cd /home/pi/java/
cp /opt/pi4j/lib/pi4j-core.jar .
```

Fichier `ClignoteLedAmelEx1.sh`:

```
java -Dpi4j.linking=dynamic -classpath .:* ClignoteLedAmel
```

Exercice 2

Fichier classe Java: **`ClignoteLedAmelEx2.java`**

Ne pas utiliser `blink()`, c'est clair et expliqué dans le livre.

Les boucles `while()` et `for (;;)` sont expliquées dans le chapitre 9.

D'autres constructions viendront aussi plus loin.

Cet exemple, sans utilisation de `blink()` ne nous permettrait pas d'exécuter autre chose en parallèle. Voir le chapitre 12 dédié au threads.

Chapitre 9 - Nos débuts en Java

Exercice 1

Nous chercherons le répertoire avec l'Explorer de Windows et copierons avec Ctlt-C son adresse depuis la barre supérieure:

`D:\EclipseWorkspace\VariablesEnJava\bin`

Avec `CMD` (DOS de Windows):

`D:`

Avec `CD Ctrl-V`

`CD D:\EclipseWorkspace\VariablesEnJava\bin`

Exécuter `java.exe` (ou `java`) avec différents paramètres

Ouvrir le code dans Eclipse, comprendre les messages et jouer avec différents paramètres

Exercice 2

Fichier classe Java: **`NosLedsEx2.java`**

Exercice 3

Juste un petit exemple :
Fichier classe Java: **`JavaTest1Ex3.java`**

Exercice 4

Pour apprendre un langage comme Java et ses nombreuses constructions, il faut bien y ajouter un peu plus dans l'exercice du prof! Et Internet est là pour nous aider!

Fichier classe Java: **`TableauDeuxDimEx4.java`**

Exercice 5

Il ne faudra pas trop chercher. Simplement comprendre le code qui suit en le relisant ou encore un pas à pas en mode Debug.

Fichier classe Java: **`FarfeluEx5.java`**

Exercice 6

Fichier classe Java: **`MesIntEx6.java`**

Exercice 7

Les exercices 7 et 9 sont presque du copier coller de l'exercice 6

Fichier classe Java: **`MesCharsEx7.java`**

Exercice 8

Fichier classe Java: **`MesIntCharsEx8.java`**

Exercice 9

Fichier classe Java: **`MesIntEx9.java`**

Exercice 10

Fichier classe Java: **`SwitchEx10.java`**

Exercice 11

Fichier classe Java: **`CmdArgsEx11.java`**

Exercice 12

Fichier classe Java: **`TableauIntEx12.java`**

Exercice 13

Avec la logique adoptée ici, il est important de tester un tableau de dimension impaire. Donc nous essayerons aussi avec int mesInt[] = new int[11];

Fichier classe Java: **`TableauIntEx13.java`**

Exercice 14

L'auteur s'est amusé à montrer plusieurs solutions. D'autres solutions existent évidemment.

Il faudra vérifier avec des espaces devant et derrière, ainsi que des doubles espaces.

Fichier classe Java: **`BonjourLaFamEx14.java`**

Exercice 15 et 16

Nous donnerons quelques pistes pour ces deux exercices 15 et 16 farfelus, mais combien intéressant pour notre apprentissage de Java.

L'exercice 14 nous a montré différentes manières de faire. Nous choisirons une ou l'autre des méthode.

La première lettre doit être considérée, sans doute séparément. Il faudra donc identifier les espaces avant de se positionner au caractère suivant qui pourrait être déjà une majuscule ou autre chose qu'une lettre. Ce dernier cas pourrait être plus simplement rejeté par une erreur.

Nous pourrions aussi ignorer les caractères qui ne sont pas des a-z et A-Z.

Les lettres en majuscule après la première lettre des mots pourraient passer en minuscule.

Nous utiliserons sans doute `Character.toUpperCase(char a).`

Donc **"**`Bonjour la famille`**"** deviendra **"**`BonjourLaFamille *`**"** si la longueur de **"**`Bonjour la famille`**"**, c'est à dire 18, est plus grande que la longueur totale des 10 phrases, avant traitement, divisé par 10.

Si nous n'avons qu'un tableau, il faudra commencer par ce calcul et sans doute créer un second tableau, par exemple de `boolean`, pour préparer ce * qui viendra en fin de traitement.

Oui c'est farfelu ... mais du joli code après pas mal de brainstorming!

Exercice 17

Fichier classe Java: **`DivMultEx17.java`**

Chapitre 10 - Une classe Java pour lire et écrire des fichiers

Fichier de test: **`config.txt`**

Exercice 1

Fichier classe Java: **`ConfigParam1Ex1.java`**

Exercice 2

Fichier classe Java: **`ConfigParam1Ex2.java`**

Exercice 3

Fichier classe Java: **`ConfigParamEx3.java`**

Exercice 4

Fichier classe Java: **`ConfigParamEx4.java`**

Exercice 5

Fichier classe Java: **`ConfigParamEx5.java`**

Exercice 6

Fichier classe Java: **`ConfigParamEx4_5_6.java`**

L'auteur y a inséré quelques tests et manières de faire pour montrer comment varier son code. Le lecteur choisira donc de programmer suivant son choix ou ses besoins.

En exécutant ConfigParamEx4_5_6 et le dernier message en particulier, nous comprendrons mieux les différentes manières de faire et de programmer ces cas d'erreur.

Dans le fichier config.txt, nous y retrouverons les paramètres utilisés :

mvtSensorActive avec la valeur true
mvtDelay avec la valeur 91
afterSunset avec la valeur 22:40
autre la valeur 1.0
param avec la valeur param1=3.0
mvtDelay avec la valeur 91

java.util.NoSuchElementException: Le paramètre pasDéfini n'existe pas

Valeur float: 12.3

Le param unautre n'existe pas: Le paramètre unautre n'existe pasDéfini

La classe ConfigParam pourrait être reprogrammée à l'infini.

Exercice 7

Fichiers classe Java: **`ReadWriteEx7.java`** et **`ReconstructEx7.java`**

Fichier de test: **`test.jpg`**

Cette exercice est particulièrement complexe. Le lecteur devra s'accrocher, soit pour écrire le code lui-même, soit pour le comprendre.

L'auteur a volontairement omis toute documentation dans le code.

Nous avons deux classes ici: ReadWriteEx7 et ReconstructEx7.

L'auteur considère qu'un tel exercice devrait déjà correspondre pour le lecteur à une très bonne maîtrise de Java. Il y a beaucoup de petits détails techniques indispensables au bon fonctionnement de ces deux outils.

Le lecteur pourrait étendre encore ce code avec d'autres options comme le passage de paramètres en argument du main().

La photo test.jpg prise par l'auteur à NYC a été réduite, mais suffisamment grande pour satisfaire les besoins de l'exercice.

La vérification en option, après reconstruction, a été laissée au lecteur (par exemple comme paramètre d'args).

Exercice 8

Nous ne donnerons que des pistes ici, car il y a évidemment quelques alternatives.

La description de l'exercice est suffisante pour retrouver les lignes de code des fichiers existants. Le lecteur choisira sans doute la version ConfigParamEx4_5_6 des exercices précédents.

Nous y donnerons un autre nom de classe.

Le lecteur utilisera sans doute un objet de LinkedHashMap, comme pour le code existant.

Il faudra considérer correctement les cas où aucune documentation est définie à l'origine. Ce n'est pas une erreur sauf si le lecteur en décide autrement.

Nous montrerons aussi sur la console d'Eclipse le commentaire avec le nom et la valeur du paramètre.

Chapitre 11 - Les Exceptions en Java

Exercice 1

Fichier classe Java: **`DesCalculs.java`**

Fichier de test: **`noscalculs.txt`**

Chapitre 12 - Java Thread – Des tâches en parallèle

Exercice 1

Fichier classe Java: **`MonThreadEx1.java`**

Exercice 2

Fichier classe Java: **`WatchdogEx2.java`**

Uniquement la version Windows est montrée dans la classe WatchdogEx2.

Nous étendrons cette classe avec le paramètre isTarget utilisé régulièrement dans le livre depuis le chapitre 2. Un sudo reboot fera le nécessaire pour redémarrer le Raspberry Pi: voir la classe RaspWebServer du chapitre 22.

Exercice 3

Il faudra définir un concept par application.

Pour chaque application, il faudra définir à quel endroit définir son watchdog, par exemple le main(),

donc le processus principal. Les exercices 1 et 2 sont de bonnes piste.

Il faudra considérer le timing.Si le processus principal prend une action toutes les 30 secondes, on ne va pas définir un concept de watchdog chaque seconde.

Nous pouvons aussi, en plus, relancer le Raspberry Pi régulièrement, par exemple toutes les jours, pour permettre en autres de libérer des ressources qui non jamais été libérées et provoquant une augmentation de la mémoire utilisées.

Chapitre 13 - Exécuter des scripts Python depuis Java

Exercice 1

Fichier classe Java: **`ProcessBuilderPythonTEx1.java`**

Il faudra s'assurer que le fichier salut.py est bien dans le répertoire racine du projet Eclipse.

Suivant le contenu du retour, il faudra adapter la vérification.

Si par exemple une température est reçue, il faudra vérifier son contenu plus précisément.

Ici le début de la réponse devrait suffire dans un premier temps.

Exercice 2
Exercice 3

Fichier classe Java: **`ProcessBuilderPythonEx2_3.java`**

Les solutions des deux exercices sont dans la même classe.

Une partie du code pourra être vérifié sous Eclipse, mais l'exécution finale du script Python blinkleds.py devra se faire sur le Raspberry Pi 3.

Exercice 4

Fichier classe Java: **`TwoPythonScriptsEx4.java`**

Exercice 5

C'est un exercice conséquent. L'auteur ne s'y est pas attaqué.

Nous commencerons par https://docs.oracle.com/javase/7/docs/api/java/lang/ProcessBuilder.html et l'exemple indiqué au début.

Il y a une foule de références sur Internet dédié à ProcessBuilder, voir associé à Redirect, comme

https://www.programcreek.com/java-api-examples/?api=java.lang.ProcessBuilder.Redirect.

Le lecteur commencera avec Eclipse avant de s'attaquer au Raspberry Pi et d'en vérifier le bon fonctionnement.

Chapitre 14 - Le jour ou la nuit

Les solutions des trois exercices sont dans la même classe.

Le développement et les tests ne se feront que sous Eclipse.

Ce sera plus simple de créer un projet Eclipse spécifique. Nous aurons juste besoin de la classe SunSetRise du projet RelaisEclairage.

Exercice 1
Exercice 2
Exercice 3

Fichier classe Java: **`MorningEveningPeriodsEx.java`**

Chapitre 15 - Un relais 5V - 220V

A partir de ce chapitre 15 et jusqu'à la fin du livre, nous ne donnerons en principe que des pistes pour résoudre les exercices.

Le code présent dans les différents chapitres du livre devrait suffire. Pour chaque exercice, nous donnerons évidemment la référence du chapitre concerné, voir des classes ou des méthodes à utiliser.

Les exercices seront parfois relativement simples, voir devenir de vrais petits travaux pratiques en informatique.

Exercice 1

Fichier script Python : **`relayEx1.py`**

Exercice 2

Fichier script Python : **`relayEx2.py`**

Utiliser le script blink1led.py du chapitre 6 pour copier le code.

Nous utiliserons les mêmes broches (principe du livre).

Exercice 3

Fichier classe Java: **`RelayEx3.java`**

Exercice 4

Nous ne donnerons évidemment que quelques pistes.

Le chapitre 10 est la référence.

Nous établirons un ensemble de paramètres qui nous semblent essentiels. Certains peuvent aussi rester dans le code.

L'exercice 3 et sa class RelayEx3 pourrait être une nouvelle base, avec perCent aussi défini dans config.txt.

La classe ConfigParam sera utilisée avec les méthodes get() appropriées.

Exercice 5

Le chapitre 22, avec sa classe RaspWebServer, est ici la référence.

Le point de départ c'est notre script PresenceSimulation.sh.

Comme pour le raspWebServer.sh du chapitre 22, nous devrons éditer le fichier /etc/rc.local pour y inclure PresenceSimulation.sh.

Si nous décidions plus tard d'y ajouter aussi un serveur Web, ce serait plus simple de lancer ce dernier depuis le main() de PresenceSimulation, voir d'une méthode appropriée permettant de partager des ressources à la fois d'une classe WebServer et d'une extension de PresenceSimulation.

Exercice 6

Au chapitre 14, nous avons utilisé la classe Calendar.

La méthode get(Calendar.HOUR_OF_DAY) devrait nous permettre d'identifier le passage entre 23 heures et minuit.

Ce test pourrait se faire dans le main() et utiliser "sudo reboot" comme dans la classe RaspWebServer du chapitre 22.

Pour développer le code sous Windows nous pourrions tester le passage sur une heure et minute spécifique ... et juste le montrer sur la console.

Une extension de cet exercice pourrait se faire avec une heure déterminée, par exemple 04:30 définie dans config.txt.

Exercice 7

Au démarrage de l'application, dans le main() sans doute, nous pourrions jouer avec notre led ou nos leds.

Nous trouverons comment changer l'état d'une led en fin de chapitre 8.

Chapitre 16 - Un capteur de lumière

A partir du chapitre 15 et jusqu'à la fin du livre, nous n'avons donné et ne donnerons en principe que des pistes pour résoudre les exercices (voir ci-dessus au chapitre 15).

Exercice 1
Exercice 2

Fichier classe Java: **`LightReadEx1_2.java`**

Les deux exercices sont dans la même classe.

Des corrections majeures y seront apportées après le codage de l'exercice 3 (fonctionnement en Thread, isTarget et withPrint).

Exercice 3

Fichier classe Java: **`LightReadEx3.java`**

Nous utiliserons la classe LightReadEx1_2 des deux exercices précédents.

En principe ces exercices devraient être dans des projets différents. La classe LightRead fonctionnerait aussi.

Très joli exercice qui demanderait une gros effort de description si inclus dans le livre.

Exercice 4

Fichier classe Java: **`LightReadEx4.java`**

Exercice 5

Nous ne donnerons que des pistes ici, car il y a beaucoup d'alternatives.

Le main() de la classe LightReadEx3 sera sans doute la base de notre application, car cette classe devrait utiliser un Thread.

Voici les points à considérer :

- Il faudra déterminer la valeur du niveau retourné par getValue() de la classe LightReadEx1_2 ;
- Nous ferons des essais avec LightReadEx1_2 et identifierons la valeur de déclenchement ;
- Nous ferons sans doute 3 ou 4 mesures très rapprochées et vérifier si elles sont dans le même rang de valeurs, par exemple 10-20% ;
- La lampe sera allumée en utilisant le relais du chapitre 15 (classe Relay et méthode activateOurRelay) ;
- Après enclenchement de la lampe nous ferons une pause, par exemple 5 ou 10 secondes ;
- Ensuite nous reprendrons des mesures de la lumière pour identifier quand cela redescend au dessous d'un certain niveau ;
- Donc, pour les deux cas de mesure, nous écrirons une méthode pour faire cette identification. Nous y passerons le niveau, le nombre de lecture, le timing et l'option en dessus ou en dessous ;
- Après avoir éteint la lampe, nous referons une pause, moins longue, avant de reprendre les mesures et recommencer dans une boucle for infinie.

Exercice 6

Nous relirons ce chapitre 16 pour faire ce travail

Nous utiliserons la combinaison Alt-Shift J.

La balise @author ne sera que dans la partie de la description de la classe et générée automatiquement avec un Alt-Shift J sur le nom de la classe.

Chapitre 17 - Un senseur de mouvement et un buzzer

A partir du chapitre 15 et jusqu'à la fin du livre, nous n'avons donné et ne donnerons en principe que des pistes pour résoudre les exercices (voir ci-dessus au chapitre 15).

Avec le chapitre 17, nous avons encore plus de composants et de possibilités d'écrire de petites applications.

Les exercices mentionnés auraient demandé à l'auteur des semaines de travail pour écrire des solutions complètes et vraiment présentable.

Certains de ces exercices pourraient même devenir de véritables travaux d'étude en y ajoutant d'autres fonctions imaginées ou requises par le lecteur ou son professeur.

Un assemblage rapide est relativement facile, mais dès qu'une solutions est un place, le lecteur aura évidemment la tentation d'y apporter des améliorations, des extensions, et voir de restructurer ses propres classes et méthodes.

Exercice 1

Fichier script Python : **`pir_relay.py`**

Après le déclenchement du relais, le capteur de mouvement aurait tendance à détecter trop vite un nouveau mouvement.

Ce n'est pas très clair! Et pourquoi pas chercher le problème en adaptant le script.

Exercice 2

Le point de départ sera les deux main() des classes Relay du chapitre 15, et de GetPirMovement ici.

Dans un premier temps, nous attendrons la détection d'un mouvement pour enclencher relais.

Après une pause de 3 minutes, nous stopperons le relais et attendrons le prochain mouvement pour recommencer.

Nous ne serons pas trop content de cette première solution rapide!

Nous utiliserons alors currentTimeMillis() de la classe MesureDeTemps du chapitre 9, tout en continuant de détecter un nouveau mouvement, pour remettre à 3 minutes une variable nous permettant de laisser le relais enclenché si nécessaire.

L'utilisation d'un Thread n'est pas nécessaire ici, mais pourrait faire l'objet d'un très bon exercice.

Exercice 3

Le chapitre 10 est la référence.

Cet exercice est similaire à l'exercice 4 du chapitre 15, où la classe ConfigParam sera utilisée avec les méthodes get() appropriées.

Nous reprendrons le code de l'exercice 2 en y rajoutant cette fonctionnalité.

Exercice 4

C'est tout à fait similaire à l'exercice 2.

Nous y intégrerons la classe MorningEveningPeriods avec ces méthodes isNight() et isMorningEvening() qui nous permettrons de savoir à quel moment nous pourrons lire le senseur de mouvement et éventuellement activer le relais.

Il est clair que si nous nous baladons constamment à minuit devant le détecteur de mouvement, nous ne devrions jamais voir notre relais se déclencher et s'enclencher immédiatement.

Exercice 5

C'est tout à nouveau une construction à partir de classes et d'exemples des chapitres précédents.

Le chapitre 15 pour le relais, le 16 pour le senseur de lumière, et celui-ci pour la détection de mouvement.

Cela vaudra la peine d'écrire une jolie classe avec le moins de code possible dans le main. Ce serait sans doute un très joli projet informatique.

Comme il y a pas mal de composants, nous pourrions aussi considérer l'exercice 7 du chapitre 19, voir le l'"exercice conséquent" du chapitre 23.

Exercice 6

C'est tout à nouveau une construction à partir de classes et d'exemples des chapitres précédents.

Cette exercice est encore plus complexe que le précédent où nous utiliserons la classe PresenceSimulation du chapitre 14.

Le chapitre 15 pour le relais, le 16 pour le senseur de lumière, et celui-ci pour la détection de mouvement.

Cela vaudra la peine d'écrire une jolie classe avec le moins de code possible dans le main.

C'est à nouveau un très joli projet informatique.

Comme il y a pas mal de composants, nous pourrions aussi considérer l'exercice 7 du chapitre 19, à cause du second relais, voir le l'"exercice conséquent" du chapitre 23.

Exercice 7
Exercice 8
Exercice 9

L'auteur pense que cette série de trois exercices est avant tout excellent pour améliorer notre expérience en programmation Java.

La classe AlarmBuzzer, avec juste un main(), n'est pas vraiment une classe.

Pour faire ces exercices, en plusieurs étapes, nous resterons avant tout sur Eclipse, avant de passer à la partie Raspberry Pi et les tests finaux.

Dès qu'une partie du code dédié au Raspberry Pi sera reprogrammé, nous l'essayerons sur la bête.

La meilleure façon de procéder est de commencer par de petits morceaux, même dans le main().

Ensuite nous définirons des méthodes dans notre nouvelle classe AlarmBuzzer.

Nous verrons rapidement le besoin d'écrire une classe Morse séparée.

Dans cette dernière, nous aurons par exemple une méthode pour la simulation dans la console de Windows.

Cette simulation peut même être paramétrée avec un withPrintMorse pour une exécution dans en mode console sur le Raspberry Pi.

Notre nouvelle classe AlarmBuzzer aura évidemment des méthodes pour jouer les points et traits du Morse avec les bonne pauses, aussi entre les lettres!

Exercice 10

Nous utiliserons évidemment le code présenté en début de chapitre.

Le code du main() de la classe GetPirMovement() sera copié / collé dans un premier temps.

Ce nous permettra aussi de vérifier notre nouvelle classe développée pour les exercices 7, 8 et 9.

Cette classe devra avoir une méthode pour "jouer" un mot en Morse.

Nous vérifierons aussi que "jouer" une phrase serait aussi possible et déjà programmée.

Le lecteur pourrait ensuite s'attaquer à l'apprentissage du Morse et comprendre ce que le Raspberry Pi peut bien nous raconter!?!

Chapitre 18 - Un senseur de température

A partir du chapitre 15 et jusqu'à la fin du livre, nous n'avons donné et ne donnerons en principe que des pistes pour résoudre les exercices (voir ci-dessus au chapitre 15).

Exercice 1
Exercice 2
Exercice 3

Nous introduirons d'abord le code suivant par exemple dans la classe JavaCode du projet du même nom au chapitre 9.

```
//22187, 22260, et 22755 donneraient 22.0, 22.5 et 23.0 respectivement.
String t1Val = "22755";
double t1Dec     = Double.parseDouble(t1Val)/1000;
float arrondi = (float)(Math.round(t1Dec * 2) / 2.0);
System.out.println(arrondi);
```

C'est avec le *2 et /2 que nous passerons en .0 ou .5

Nous introduirons ensuite le code dans la classe DS18x20read.

Il nous faudra les fichiers : **`w1_slave`**, **`config.txt`** et la classe Java **`ConfigParam.java`**.

Pour les cas de la vérification des 5 chiffres, sans doute pas nécessaire à cause du protocole, nous laisserons le lecteur écrire la solution et y ajouter un code d'erreur supplémentaire.

Exercice 4

Se référer aux exercices 7, 8 et 9 du chapitre précédent, le 17.

Le code de ces exercices sera nécessaire.

La classe pourrait s'appeler AlarmTemperature.

Nous lirons sans doute la température à partir d'un thread.

Ce qui est important sera sans doute de bien définir les écarts de température.

Utiliser le résultat des exercices 1, 2 et 3 semble une bonne chose.

Une simulation sous Windows sera nécessaire pour répéter convenablement le SOS.

Nous pourrions tester par exemple une limite à 23.0 avec un tableau de float 23.0 23.5 24.0 23.5 23.0 22.5 23.0 22.5 23.0 22.5 23.0 … et un timer.

Lors de variations rapides entre 22.5 et notre limite, nous aurions un autre timer pour ne pas répéter continuellement le SOS.

Exercice 5

Pour ajouter des leds à notre exercice précédent, nous nous référerons au chapitre 8.

Tout en fin de ce chapitre-là nous trouverons comment changer l'état d'une led.

Chapitre 19 - Ultrasons et bouton

A partir du chapitre 15 et jusqu'à la fin du livre, nous n'avons donné et ne donnerons en principe que des pistes pour résoudre les exercices (voir ci-dessus au chapitre 15).

Bien que le développement se fait sous Eclipse, il sera sans doute difficile de tout simuler sous Windows. Il nous faudra sans doute définir un certain nombre de withPrint différents pour tester sur le Raspberry Pi les différentes classes et méthodes.

Plus nous ajoutons de composants, plus notre imagination nous conduirait à créer des applications encore plus complexes, comme par exemple utiliser le capteur ultrasonique pour l'exercice 5 du chapitre 17 (`LampAuto`) : n'allumer qui si la personne est à une certaine distance (attention à l'angle).

Exercice 1

Fichier classe Java: **`ultrasonic_speed.java`**

Exercice 2

Nous créerons donc une classe `ButtonPython` dans un projet Eclipse du même nom. La référence est évidemment l'exemple du livre `BoutonDistance`.

Cette classe aura évidemment une méthode mesure() qui nous retournera la distance mesurée avec le capteur ultrasonique décrit dans ce chapitre. La classe `Button` de ce même chapitre pourra être certainement utilisée, voir avec quelques adaptations : il faudra déterminer l'intervalle désiré entre deux "clics" de boutons et ignorer le second s'il prend trop de temps.

Nous reprendrons le chapitre 13 qui nous explique comment recevoir la mesure du script Python. Le script `ultrasonic.py` de ce chapitre-ci devrait convenir. Si trop de variations sont visibles, où que la réponse est trop lente, il faudra adapter le script Python. Ce ne serait pas une bonne idée de faire une moyenne en Java en exécutant plusieurs fois le script Python (délais trop élevés).

Exercice 3

Il faudra étudier le code de la classe `BoutonDistance` pour examiner si cela fonctionne déjà ou non et faire la correction avec le changement d'état de avec `.getState()`. Nous pourrions décider de ne pas utiliser de `Listener` du tout.

Dans l'exercice 1 du chapitre 18, nous avions déjà joué avec des arrondis.

Exercice 4

C'est une extension des exercices 2 et 3. Nous pourrions définir les fonctionnalités dans un Thread (chapitre 12).

Un état pourrait être défini avec des valeurs indiquant par exemple 4 états comme : clic court, double clic court, pression longue, deux pressions longues, voir bouton reste pressé pour plus longtemps que x secondes.

Exercice 5

C'est une utilisation direct de l'exercice 4. Nous utiliserons le buzzer du chapitre 17 et des exercices 7 ou suivants.

Le `sudo reboot` de la méthode `RebootHandler` de la classe `RaspWebServer` du chapitre 22 à venir passera très bien.

Exercice 6

Cet exercice est sans doute un des plus difficiles de cet ouvrage.

Cette classe `CapteurUltrason` devra considérer une foule de paramètres et à se demander si c'est vraiment faisable. La classe `BoutonDistance` de ce chapitre ne nous aidera pas beaucoup, si ce n'est comment obtenir la distance à un objet.

L'idée de l'auteur aurait été, avec le capteur de mouvements qui a détecté une présence, de déterminer si la personne s'éloigne, pour éteindre plus rapidement notre relais.

Il faudra absolument tester cette classe de manière précise et pragmatique. L'angle de mesure est faible et l'objet pourrait zigzaguer. Ce n'est pas du tout le cas si nous devions détecter sur un circuit de train électrique, si le train arrive ou repart : l'angle est précis et le capteur relativement facile à positionner.

Nous devrons sans doute introduire une valeur indiquant la distance jusqu'au mur, ou autre objet fixe, suivant le positionnement du capteur. Des mouvements trop lointain ou trop proche pourrait être ignorés.

Si le lecteur arrive a le faire fonctionner, il pourrait encore ajouter un double-clic sur le bouton pressoir pour fixer le paramètre de distance sans objets en mouvement.

Exercice 7

Le chapitre 10 est la référence.

L'exercice 4 du chapitre 15, par exemple, nous montre comment définir des paramètres pour une application, en utilisant la classe ConfigParam et les méthodes get() appropriées.

Mais ici, c'est un peu différent. Le lecteur pourra imaginer différentes solutions pour résoudre cet exercice.

Imaginons que le lecteur a tout retiré ces fils sur son breadbord ou alors un seul fil est sorti: cela arrive tout le temps. Comment retrouver la broche sans devoir consulter le code!

Prenons le chapitre 15 avec le relais. Il nous semble qu'un `config.txt` avec:

```
defrelay_1=pin16 ; GPIO4
                 ; 5V pin2
                 ; ground pin6
```

devrait suffire (_1: nous pourrions avoir plusieurs relais). Les pins paires sont à droite, et le 16 est donc le huitième.

En se référant au chapitre 15 et la classe Relay, nous avons des instructions comme:

```
relayPin = gpio.provisionDigitalOutputPin(pinPi4j);
```

```
Relay ourRelay1 = new Relay(isTarget, RaspiPin.GPIO_04, true);
```

Il nous faux donc pouvoir convertir defrelay en pin16 physique et ensuite en RaspiPin.GPIO_04.

Le premier se fera avec le get() de la classe ConfigParam, et le second pourrait se faire avec une nouvelle méthode (voir une nouvelle classe Java) qui nous convertirait automatiquement les broches physiques dans la notation Pi4J avec une table dédiée.

Ce serait sans doute un joli exercice à faire en Python, en utilisant aussi le fichier config.txt.

Nous rappellerons que ce livre n'est pas consacré à Python et que ce genre d'exercice dépasserait les exemples simples présentés. C'est comme si nous présentions d'abord le code Java et de demander d'en faire la traduction en Python!

Exercice 8

Le chapitre 13 est la référence.

Pour préserver les scripts existants, si nous devons les adapter, nous les copierons de /home/pi/python dans /home/pi/python2.

Uniquement le cœur de la classe sera testé sous Eclipse. Il n'est pas sensé de simuler chaque script Python sous Windows.

La lecture d'un fichier texte se fera comme dans la classe ConfigParam du chapitre 10.

Nous laisserons le lecteur adapter chaque script si nécessaire et adapter le timing pour permettre un test fluide entre tous les scripts. Aucun de ces scripts ne devra évidemment être intégré dans des boucles continues.

Exercice 9

Il faudra consulter la documentation sur Internet sur le site de Python.

Ici c'est juste une question d'ajouter des parenthèses dans la fonction print du langage Python.

Une bon site :
https://stackoverflow.com/questions/32122868/python-3-print-without-parenthesis

Chapitre 20 - Java et les fichiers .jar

A partir du chapitre 15 et jusqu'à la fin du livre, nous n'avons donné et ne donnerons en principe que des pistes pour résoudre les exercices (voir ci-dessus au chapitre 15).

Exercice 1

Il faudra suivre les instructions décrites dans le chapitre. Nous sélectionnerons évidemment un projet Eclipse dans le livre avec une classe Java où nous trouverons au moins un composant GPIO. Si nous avons programmé un joli exemple dans les exercices des chapitres précédents de 15 à 19, nous nous le choisirons.

Chapitre 21 - Pi Camera

A partir du chapitre 15 et jusqu'à la fin du livre, nous n'avons donné et ne donnerons en principe que des pistes pour résoudre les exercices (voir ci-dessus au chapitre 15).

Exercice 1

A la classe `ProcessBuilderCamera`, nous ajouterons une méthode comme par exemple `execVideo()`. Nous définirons sans doute un `String[] videoParam` avec les paramètres requis (voir la partie Prise d'une vidéo). Après avoir vérifié sous Eclipse en mode simulation, nous passerons sur le Raspberry Pi pour vérifier le fonctionnement, voir faire ensuite les dernières adaptations.

Exercice 2

Beaucoup de fonctionnalités et d'améliorations à la classe `ProcessBuilderCamera` sont possibles.

Ce genre de classes sont souvent écrites d'un premier jet et nous sommes contents lorsqu'elles fonctionnent. Ensuite il faut faire une petite analyse. Nous voyons que le -o n'est pas vraiment nécessaire dans le `main()` et en plus pas joli joli. La construction de `String[] cameraParam` peut ce faire dans une méthode. Il faudra décider si nous passons la dimension de l'image par le constructeur ou une méthode, voir au dernier moment !

Une méthode pour appliquer une rotation à la prise de photo serait intéressante. La référence de la documentation se trouve dans le livre.

Le code de l'exercice 1 sera évidemment inclus.

Exercice 3

Pour écrire cette classe DiskSpace, nous consulterons le chapitre 13 pour exécuter avec `ProcessBuilder` la commande `df`.

Comme il y a un gros travail de parsing, nous exécuterons `df` sur le Raspberry Pi, récupérerons le résultat, et le déposerons, en mode simulation sous Windows, dans un `String` équivalent aux résultats de la méthode `pyOutput()` utilisée dans nos exemples du chapitre 13.

C'est la ligne du système de fichiers `/dev/root` qu'il faudra utilisée. De la colonne `Util%` (pour cent utilisé) nous retirerons en Java la valeur que nous devrons convertir en % restant.

Exercice 4

Encore un très bon exercice en Java. Nous utiliserons le résultat des exercices 1 et 2 pour y ajouter une méthode pour activer le paramètre monochrome `--colfx 128:128` à la commande `raspistill`.

Nous ajouterons une méthode pour nous retourner la dimension du fichier image généré. Le `main()` fera le travail de prise de photos et de calcul de gain. Le résultat sera analysé évidemment visuellement après transfert sur le PC.

Exercice en plus : rechercher si `colfx` est possible en vidéo, voir autrement et y introduire le code.

Chapitre 22 - Un serveur Web sur le Raspberry Pi 3 B

A partir du chapitre 15 et jusqu'à la fin du livre, nous n'avons donné et ne donnerons en principe que des pistes pour résoudre les exercices (voir ci-dessus au chapitre 15).

Exercice 1

Le fichier contiendra la température sur la première ligne et la date et l'heure sur la suivante.

Nous écrirons donc une classe basée sur la `DS18B20read` du chapitre 18 pour lire cette température en continu, par exemple chaque 10 secondes, et la stocker sur deux lignes dans un fichier texte (voir la classe `ConfigParam` et le code de la méthode saveConfig() du chapitre 10).

L'instance de la classe du serveur Web se fera **pas** à partir d'un thread. Donc le client Web devra attendre un instant avant de recevoir le résultat. Ce n'est pas trop bon et une meilleure manière de faire sera décrite dans l'exercice 3.

Exercice 2

Cet exercice se fera entièrement sous Eclipse. Cela prendra un certain temps avant d'avoir adapter nos méthodes de classe pour y introduire des balises HTML. Lorsque nous afficherons notre page Web pour la première fois, nous y découvrirons sans doute des erreurs et nous pourrions alors examiner le code source avant de l'adapter avec l'outil de l'esplorateur (sous Firefox et Chrome : bouton droite de la souris et option dans le menu).

La méthode `sendResponse()` de la classe `RaspWebServer`, reçoit juste un `String`. Ce String pourrait inclure des balises HTML, ou alors `sendResponse()` emballer le `String` dans des balises.
Le lecteur décidera et pourra même y ajouter des balises contenant des parties en gras, des listes, des URLs ou autres. Le `String` pourrait être associé à un type de balise (écrire une classe!).

Un exemple de l'ajout de balises se trouve dans le code du gestionnaire `FormHandler` de la classe `RaspWebSetServer` de ce chapitre.

Exercice 3

Nous utiliserons tout le matériel des exercices 1 et 2.

Le serveur Web sera dans un thread. L'instance de la classe du serveur Web se fera à partir d'un thread (chapitre 12) : les deux processus seront actifs en parallèle. La méthode qui sauvera le contenu dans le fichier température sera `synchronized` (mot clé décrit dans ce chapitre).

Il est clair que nous avons le choix de ne pas avoir la lecture de la température dans un thread, mais cette application pourrait faire d'autres chose en parallèle comme de la simulation de présence.

Exercice 4

Il faudra donc créer deux nouveau gestionnaires pour `...:8080/lampeOn` et `...:8080/lampeOff`.

Pour simplifier, dans chaque gestionnaire, nous créerons une instance de `Relay` et utiliserons la méthode `activateOurRelay()` du chapitre 15.

C'est vraiment simpliste et nous pourrions étendre notre serveur Web à l'infini, par exemple en spécifiant un relais spécifique, en passant un argument au serveur, comme dans la classe `RaspWebSetServer`, ou encore un relais qui ferait partie d'une application active dans un thread.

Passer à `lampeOn` un paramètre indiquant combien de temps la lumière resterait allumer, nous obligerait à définir un processus parallèle pour faire l’extinction après ce délais.

Exercice 5

La référence, c'est l'exercice 3 du chapitre 21.

Nous allons créer un gestionnaire pour `/diskspace` qui va nous calculer l'espace libre au moment du questionnement. Nous donnerons l'espace utilisé et restant en %, et pourquoi pas leurs valeurs en Moctets. Les valeurs restantes pourraient être présentées entre les balise "grasses" voir avec une méthode développée dans le cadre de l'exercice 2.

Exercice 6

C'est vraiment un petit monstre cet exercice. Il y a tellement de possibilités d'amélioration et de peaufinage, que nous pourrions en faire un concours dans une classe d'informatique.

Même du `css` pourrait être mis en place pour colorer notre page ou les champs présentant des erreurs.

Le point de départ sera sans doute, comme décrit dans ce chapitre, un fichier `form.html` que nous introduirons dans un gestionnaire, comme le `FormHandler` du chapitre.

La difficulté sera de présenter les erreurs après avoir identifié quels paramètres semblent erronés. Nous ne pourrons évidemment pas tout vérifier, comme par exemple si l'adresse Email est atteignable. Dans un cas une simple vérification des positionnements du @ et du point, ainsi que la présence de caractères normaux, pourrait suffire.

Avec beaucoup de code Java, c'est un excellent exercice aussi côté design en développant de petites méthodes claires et concises dans notre classe.

Exercice 7

C'est en retirant le `throws Exception` de la classe `LectureUrl` que nous remarquerons non seulement le désastre, avec 4 fautes, et que c'est vraiment le bon moment d'écrire une vraie classe instanciable.

Pour chaque cas d'erreur, les méthodes nous retournerons une indication et la source de l'erreur et pourquoi pas avec une méthode `getErrorMsg()`.

Exercice 8

Le `stringLine` contiendra la réponse à notre requête et nous analyserons chaque ligne pour en sortir une paire clé / valeur identique en fait à la lecture du fichier `config.txt` par la méthode `readConfig()` la classe `ConfigParam1`.

Il est évident que si nous avions étendu notre classe de ce serveur Web, pour nous retourner du code HTML, et non du simple texte, nous aurions de grosses difficultés de parsing sur les balises.

Cette classe sera développée sous Eclipse avec le serveur précédemment lancé depuis Eclipse aussi, mais pas nécessairement dans le même projet!

Exercice 9

C'est le même exercice que le précédent, mais avec une température. Nous resterons tout d'abord sur le PC. Nous pourrons ensuite utiliser un vrai Raspberry Pi qui possède un capteur de température.

Les exercices 8 et 9, et tout ce chapitre, nous ont montré en fait des moyens de communiquer entre 2 Raspberry PI voir depuis un PC. Sur ce dernier, nous pourrions par exemple développer sous Eclipse ou Netbeans, une jolie application Swing pour jouer avec notre ou nos Raspberry Pi, voir d'autres objets connectés.

Un autre exercice conséquent

Créer une nouvelle classe `RaspbWeb` incluant les fonctionnalités de `RaspWebServerAmel` et de `RaspWebSetServer`.

Choisir un des exercices précédents avec au moins un composant GPIO et développer une application composée de plusieurs classes.

Un des meilleurs exemples serait sans doute celui basé sur la classe `PresenceSimulation` avec un relais, voire un capteur de lumière en plus.

Les paramètres de l'application seraient modifiables avec un formulaire HTML et sauvegardés dans notre fichier de configuration.

Toutes ces fonctionnalités ont déjà été décrites dans les chapitres précédents.

Définir et vérifier les algorithmes permettant de recharger correctement ces nouveaux paramètres dans l'application concernée.

Chapitre 23 - Base de données relationnelle SQLite

Pour ces exercices, il nous faudra consulter le chapitre 23, créer un nouveau projet et y importer et définir les librairies JDBC comme expliquer.

Exercice 1

Fichier classe Java: **`CreateTableEvents.java`**

En début de classe, nous effacerons le fichier de la base de données, car l'instruction CREATE, comme définie, ne supprimera pas la table, ni son contenu d'ailleurs, si elle existe déjà.

Après la création du projet Eclipse, il n'est pas nécessaire de créer un fichier eventsSqlitePi.db, même vide.

Nous utiliserons le DB Browser pour SQLite, comme expliquer dans ce chapitre, pour visionner la structure de notre table.

Si nous exécutons à nouveau la classe dans Eclipse, il est possible que le DB Browser soit encore actif empêchant d'effacer le fichier de base de données.

Exercice 2

Fichier classe Java: **`CreateTableEvents.java`**

Fichier texte délimité: **`delim.txt`**

Le lecteur pourra reprendre cette exercice, lorsque terminé, pour en faire une jolie classe avec des méthodes bien définies pour chacune des fonctions nécessaires, comme par exemple l'extraction des champs.

L'analyse de la date et de l'heure devrait être aussi pointue que possible.

Nous allons reprendre la classe de l'exercice précédent pour aussi créer la table Events.

Pour la lecture du fichier délimité, nous avons pratiquement un copier / coller de la classe ConfigParam du chapitre 10.

La méthode isStringField() est vraiment simpliste, mais nous aide tout de même ici, pour simplifier le code.

Il faudra noter que l'instruction

INSERT INTO Events VALUES 'Temperature1','2018/06/04','16:00:00',27.1

fonctionnera aussi si les trois premiers champs sont compris entre deux " et non deux '.

Nous utiliserons le DB Browser pour SQLite, comme expliquer dans ce chapitre, pour visionner la structure et le contenu de notre table.

Exercice 3

Pour cet exercice conséquent, nous ne donnerons que quelques pistes.

Nous avons à disposition l'exercice 2 et la classe RaspbEvent décrite dans ce chapitre.

Nous savons comment insérer des données dans la base de données SQLite.

Nous créerons un objet de la classe RaspbEvent et pourrons ensuite en extraire la valeur des 4 champs pour construire une instruction SQL.

La suite se fera avec les méthodes createStatement() et executeQuery() décrites dans ce chapitre.

Nous laisserons les détails pour l'exercice suivant.

Exercice 4

Pour cet exercice plus que conséquent, nous ne donnerons évidemment que quelques pistes.

Nous avons à disposition les exercices 2 et 3, ainsi que la classe RaspbEvent décrite dans ce chapitre.

Nous considérerons tous les cas de figures, comme le fichier SQLite qui n'existe pas ou qui contient déjà notre table, voir des données aussi.

Il nous faudra garder une collection d'objets de RaspbEvent et pouvoir déterminer si un même contenu est déjà inclus dans la base de données.

Nous avons plusieurs choix possibles, avec des collections en Java, voir un simple tableau de RaspbEvent.

Il nous faudra coder des outils de comparaison pour identifier si un objet est déjà présent.

La manière la plus simple devrait être de construire un objet de RaspbEvent à partir d'une ligne du fichier delim.txt et ensuite de vérifier s'il existe déjà dans notre collection.

Si ce n'est pas le cas, nous pourrions alors l'insérer dans notre base de données sqlitePi.db.

Lorsque nous aurons terminé notre application, il devrait être suffisant de modifier une ligne de delim.txt, avec des valeurs différentes que précédemment, et de ré-exécuter le programme. Un seul nouvel enregistrement devrait être visible dans notre DB Browser pour SQLite.

Exercice 5

C'est un joli projet informatique que nous laisserons au lecteur.

Nous utiliserons le résultat de l'exercice 4 et le chapitre 10.

Notre table de paramètres devrait contenir au moins 2 champs, son nom et sa valeur. Un troisième champ serait pratique pour stocker l'information de commentaire du fichier config.txt après le ;.

Lorsqu'une première version sera plus ou moins fonctionnelle, nous introduirons les fonctionnalités décrites dans les exercices du chapitre 10.

Une méthode pour créer un fichier config.txt à partir de la base de données SQLite serait aussi un excellent exercice.

Chapitre 24 - Envoyer des Emails

A partir du chapitre 15 et jusqu'à la fin du livre, nous n'avons donné et ne donnerons en principe que des pistes pour résoudre les exercices (voir ci-dessus au chapitre 15).

Exercice 1

Nous pouvons aussi créer un nouveau projet Eclipse que nous utiliserons pour les exercices suivants.

Au début du main() de la classe EmailSqlite, il nous suffira d'introduire le code suivant:

```
File dbFile = new File("sqlitePi.db");
if (!dbFile.exists()) {
  System.out.println("Le fichier sqlitePi.db n'existe pas");
  return;
}
```

La classe EmailSqlite pourrait être améliorée en utilisant une variable comme dbFileName pour le non du fichier. Nous aurions alors des:

```
String url = "jdbc:sqlite:" + dbFileName;£
eEnv.envoyer("Données + dbFileName + " d'hier", mailMsg, "adress_email");
```

Exercice 2
Exercice 3

Nous les mettrons ensemble.

Fichier classe Java: **`EmailImageEx2_3.java`**

Pour ces exercices 2 et 3 nous définirons un nouveau projet Eclipse.

Il faudra y importer les classes .jar ainsi que la classe EmailEnvoi.java qui sera utilisé pour l'exercice 3.

La méthode static File.separator est intéressante et c'est la première fois que nous l'utilisons.

Elle permet d'ajouter le caractère définissant le séparateur pour les sous-répertoire.

Il est différent sous Windows et Linux: \ et /.

Exercice 4

Nous ne donnerons que quelques pistes au lecteur.

Nous laisserons le programmeur copié et renommé PirEnvoiEmail à nos souhaits dans le nouveau projet créé pour les exercices de ce chapitre.

Nous pensons que l'envoi d'une image peut aussi y rester en y ajoutant les paramètres nécessaire.

La classe PirEnvoiEmail deviendra une jolie classe avec un ensemble de méthodes et de paramètres, comme les adresses emails, pour gérer cette petite application.

Nous retournerons au chapitre 21 et à la classe ProcessBuilderCamera pour y retrouver et adapter le code nécessaire pour correspondre à la manière de faire de PirEnvoiEmail.

Les 3 exercices suivants ne seront que brièvement décrits vu leurs complexités.

Exercice 5

Il faudra consulter le chapitre 22 pour définir deux nouveaux gestionnaires.

Nous aurons peut-être déjà un serveur Web en développement et nous le réutiliserons.

Des améliorations apportées dans les exercices du chapitre 22 peuvent y être aussi ajoutées comme une belle présentation en HTML.

Un thread tout simple pourrait détecter un mouvement et enregistrer l'image.

Pour chaque nouveau mouvement, la même image pourrait être réécrite.

Attention que l'image soit prête.

Comme extension, nous pourrions ne prendre une photo que si le détecteur à ultrason (chapitre 19) a détecté que la personne s'approche.

Exercice 6

Nous laisserons le lecteur retrouver chaque fonction et du code existant dans les différents chapitres de cet ouvrage.

De jolis tableaux de température pourrait y être présenté, voir avec les minimum et maximum dans une journée.

Cet exercice pourrait être ensuite étendu en utilisant le précédent,comme de rendre chaque ligne écrite en HTML réactive pour activer l'envoi par email de l'image correspondante à la détection. Une requête YES/NO semble nécessaire.

Exercice 7

Comme pour l'exercice précédent, nous laisserons le lecteur retrouver chaque fonction et du code existant dans les différents chapitres de cet ouvrage.

Un certain nombre de fonctionnalités auront aussi déjà été considérées dans les nombreux exercices de cet ouvrage.

Il est évident qu'une interface config.txt, voir par le Web pourrait être développée pour entrer les adresses Emails, les mots de passe et les noms de serveurs dédiés.