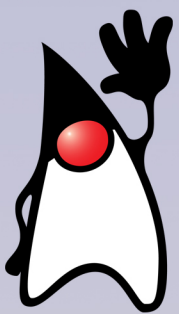
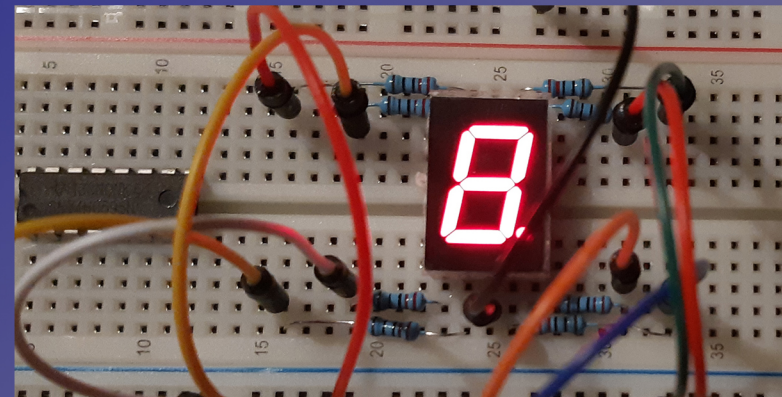
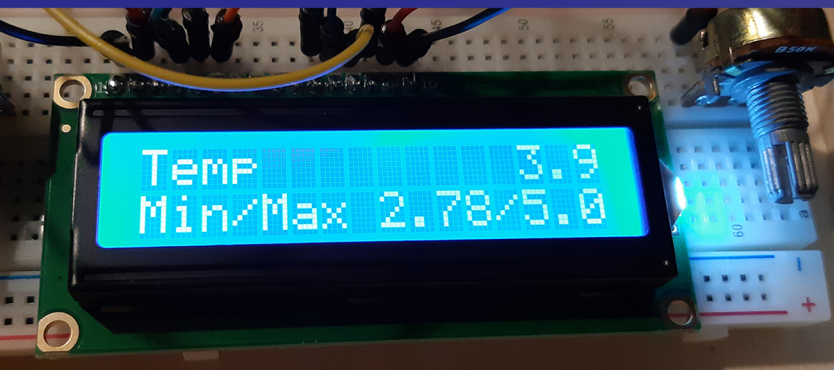
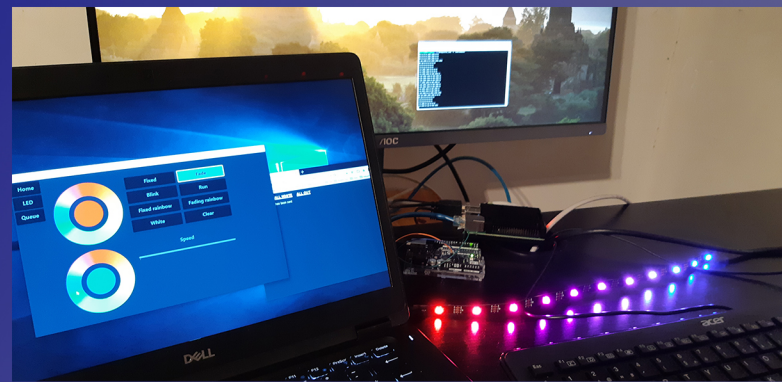
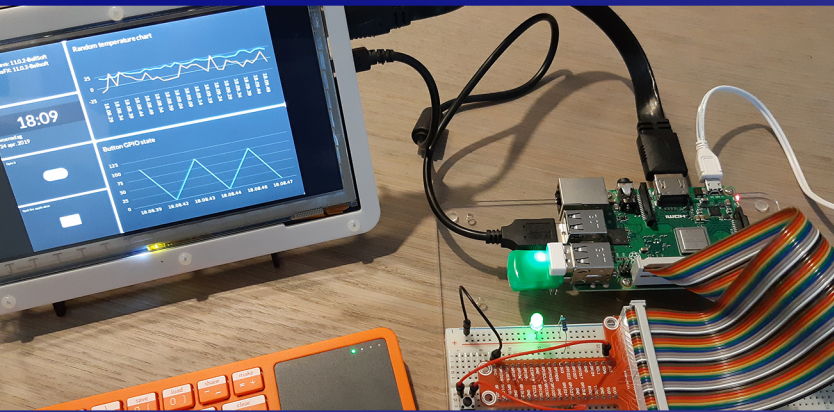
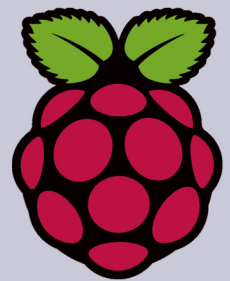


# Getting started with Java on Raspberry Pi



OpenJDK  
JavaFx



Frank Delporte  
webtechie.be

# Getting started with Java on the Raspberry Pi

A lot of small and bigger examples to introduce you to Java (11+), JavaFX (11+), Pi4J, Spring, Queues... with hardware projects on the Raspberry Pi.

Frank Delporte

This book is for sale at <http://leanpub.com/gettingstartedwithjavaontheraspberrypi>

This version was published on 2020-05-05



Leanpub

This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2019 - 2020 Frank Delporte

# Tweet This Book!

Please help Frank Delporte by spreading the word about this book on [Twitter](#)!

The suggested tweet for this book is:

[#JavaOnRaspberryPi](#) combined with [#JavaFX](#), [#Spring](#), [#Pi4J](#) and hardware components is fun! Find out in this book with a lot of examples: "Getting started with Java on the Raspberry Pi" by [@FrankDelporte](#) - <https://leanpub.com/gettingstartedwithjavaontheraspberrypi>

The suggested hashtag for this book is [#JavaOnRaspberryPi](#).

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

[#JavaOnRaspberryPi](#)

# Contents

<b>Chapter 1: Introduction</b> . . . . .	<b>1</b>
Content overview . . . . .	2
About me . . . . .	3
Sources and scripts used in this book . . . . .	4
Where to find them . . . . .	4
Get the sources . . . . .	4
The styling used in the book . . . . .	6
Read the README! . . . . .	7
Guidelines . . . . .	7
Example . . . . .	7
What's next? . . . . .	8
Thanks to... . . . . .	9
<b>Chapter 2: Tools and hardware used in this book</b> . . . . .	<b>10</b>
Raspberry Pi . . . . .	11
Prepare the Pi . . . . .	11
Connections between Pi and breadboard . . . . .	15
Software tools on the Pi . . . . .	18
Linux commands crash course . . . . .	18
Firefox . . . . .	22
VNC server . . . . .	22
Enable SSH on the Pi . . . . .	23
Free software tools on PC . . . . .	25
Integrated development environment aka IDE . . . . .	25
Remote connection to a Raspberry Pi (SSH) . . . . .	25
Wiring diagrams . . . . .	27
Schematic drawings . . . . .	28
Hardware components . . . . .	29
Resistors . . . . .	29
LEDs . . . . .	30
RGB-LED . . . . .	31
LED strips . . . . .	32
Electronic kit with Arduino board . . . . .	35

## CONTENTS

<b>Just a thought: Learn by educating</b> . . . . .	<b>36</b>
Interview with Karen Mouws . . . . .	37
<b>Chapter 3: Choosing an IDE</b> . . . . .	<b>39</b>
IntelliJ IDEA . . . . .	40
Using IntelliJ IDEA with the example projects . . . . .	41
Interview with Trisha Gee . . . . .	44
Visual Studio Code (VSC) . . . . .	46
VSCodium the free non-tracking alternative to VSC . . . . .	47
Java development with Visual Studio Code . . . . .	48
Using Visual Studio Code on the PC with code on the Pi . . . . .	49
Interview with Xiaokai He . . . . .	52
<b>Chapter 4: About Java</b> . . . . .	<b>53</b>
History . . . . .	55
Java files versus byte code . . . . .	56
JVM versus JRE versus JDK . . . . .	57
JVM = Java Virtual Machine . . . . .	57
JRE = Java Runtime Environment . . . . .	57
JDK = Java Development Kit . . . . .	57
Version history . . . . .	58
JDK providers . . . . .	59
Oracle . . . . .	59
AdoptOpenJDK . . . . .	59
Azul Zing and Zulu . . . . .	59
BellSoft Liberica . . . . .	60
Interview with Alexander Belokrylov . . . . .	61
Installing the Java JDK . . . . .	62
Install Java JDK on a Windows PC . . . . .	62
Install Java JDK on a Linux PC with SDKMAN . . . . .	63
Install Java JDK on a Raspberry Pi . . . . .	65
Some of the changes between Java versions . . . . .	67
Changes between Java 8 and 11 . . . . .	67
What's next after Java 11? . . . . .	68
Java crash course . . . . .	72
HelloWorld! Running a single-file Java-application . . . . .	72
Using the start-up arguments . . . . .	73
Working with numbers . . . . .	74
If, Then, Else . . . . .	75
Enum and Switch . . . . .	77
Using methods . . . . .	78
Using objects . . . . .	79
Reading a text file . . . . .	82

## CONTENTS

Using streams . . . . .	85
What's next? . . . . .	87
Interview with Jakob Jenkov . . . . .	88
<b>Chapter 5: Raspberry Pi pinning . . . . .</b>	<b>90</b>
Raspberry Pi types . . . . .	92
Models . . . . .	92
Major versions . . . . .	92
Board versions . . . . .	93
Pin types . . . . .	95
Power and ground . . . . .	95
Digital GPIO . . . . .	95
Pin functions . . . . .	96
Universal Asynchronous Receiver and Transmitter (UART - Serial) . . . . .	96
General Purpose Clock (GPCLK) . . . . .	97
Inter Integrated Circuit (I <sup>2</sup> C) . . . . .	97
Serial Peripheral Interface (SPI) . . . . .	97
Pulse-Width Modulation (PWM) . . . . .	98
Header types . . . . .	99
40-pin header . . . . .	99
26-pin header - Type 1 and 2 . . . . .	101
8-pin header . . . . .	102
Different pinning numbering schemes . . . . .	103
Board (pin) number . . . . .	103
BCM number . . . . .	103
WiringPi number . . . . .	103
<b>Chapter 6: What is Maven? . . . . .</b>	<b>106</b>
Install Maven . . . . .	109
On Windows PC . . . . .	109
On Raspberry Pi . . . . .	109
Generate a new Maven project . . . . .	111
Project structure . . . . .	111
A minimal pom.xml example . . . . .	112
Maven pom-files used in this book . . . . .	114
Add application logging with Maven and log4j . . . . .	116
<b>Just a thought: Abbreviations . . . . .</b>	<b>119</b>
<b>Chapter 7: About JavaFX . . . . .</b>	<b>121</b>
History . . . . .	122
Interview with Johan Vos . . . . .	123
Sample libraries to extend JavaFX . . . . .	125
TilesFX . . . . .	125

## CONTENTS

FXRibbon . . . . .	125
ControlsFX . . . . .	126
PickerFX . . . . .	126
Interview with Gerrit Grunwald . . . . .	128
Minimal JavaFX 11 sample application . . . . .	129
Add new archetypes to Maven . . . . .	129
Creating an empty application . . . . .	129
Running the empty application from Visual Studio Code . . . . .	130
Running the application on the Pi . . . . .	132
Example 1: TilesFX dashboard . . . . .	134
Wiring and testing in terminal . . . . .	134
Blink an LED with Java . . . . .	136
Building our first JavaFX application . . . . .	137
Run the application on PC . . . . .	150
Run the application on the Pi . . . . .	151
Conclusion . . . . .	153
Start a Java application when the Pi starts up . . . . .	154
Disable screensaver . . . . .	155
Example 2: Interact with an I <sup>2</sup> C relay board . . . . .	156
Enable and test I <sup>2</sup> C . . . . .	156
Coding the I <sup>2</sup> C controller application . . . . .	157
Running the relay controller on the Pi . . . . .	162
Example 3: Build a UI with FXML . . . . .	164
Generate an empty FXML project as a starting point . . . . .	164
Scene Builder . . . . .	166
<b>Just a thought - Beware of the PAF . . . . .</b>	<b>168</b>
<b>Chapter 8: Bits and bytes . . . . .</b>	<b>169</b>
Convert bits to a numeric and hex value . . . . .	170
Calculate a byte value . . . . .	171
Value ranges in Java . . . . .	172
Difference between Byte, Short, Integer and Long . . . . .	172
Minimum and maximum values in Java . . . . .	172
Signed versus unsigned . . . . .	173
Conclusion . . . . .	174
What can we do with this? . . . . .	176
Web colors . . . . .	176
Controlling a numeric segment display . . . . .	176
<b>Chapter 9: PI4J . . . . .</b>	<b>195</b>
Installation . . . . .	196
Programming with Pi4J . . . . .	197
Sources . . . . .	197

## CONTENTS

Maven dependencies . . . . .	197
Running the examples . . . . .	199
Digital GPIO input and output examples . . . . .	202
Example 1: Digital output with RGB-LED . . . . .	202
Example 2: Digital input with a button . . . . .	205
Example 3: Distance sensor . . . . .	209
PWM example . . . . .	216
Wiring a single LED . . . . .	216
Code to control an LED with PWM . . . . .	216
Running the example . . . . .	218
SPI example with MAX7219 and 8x8 LED-matrix . . . . .	219
Wiring . . . . .	219
SPI example code . . . . .	220
Running the application and created matrix output . . . . .	232
SPI conclusion . . . . .	233
Serial communication example with an Arduino . . . . .	234
Wiring . . . . .	234
Arduino code . . . . .	235
Detecting the serial interface on the Pi . . . . .	237
Raspberry Pi code . . . . .	237
Running the Java serial application . . . . .	243
What's next . . . . .	244
LCD-display with the weather forecast . . . . .	245
Wiring to connect an LCD to the Pi . . . . .	245
Get an AppID on OpenWeatherMap . . . . .	246
Weather LCD application code . . . . .	247
Running the LCD weather application . . . . .	258
Conclusion . . . . .	259
<b>Just a thought: Switching social . . . . .</b>	<b>260</b>
<b>Chapter 10: Spring . . . . .</b>	<b>262</b>
What is Spring Boot? . . . . .	264
What is Spring Initializr? . . . . .	265
Interview with Mark Heckler . . . . .	268
Example 1: Minimal webserver on the Pi . . . . .	270
Start from the Initializr project and modify pom.xml . . . . .	270
Application properties . . . . .	271
Image controller . . . . .	271
Swagger config . . . . .	276
Run on the Pi . . . . .	277
Conclusion . . . . .	278
Example 2: Database REST-service for IoT data on Pi . . . . .	279



## CONTENTS

pom.xml settings . . . . .	279
Creating the database entities . . . . .	280
Storing data in the database . . . . .	285
Adding the REST-services . . . . .	285
Adding Swagger . . . . .	287
Running the application and using the REST-services . . . . .	288
Configuration to run on the Pi . . . . .	292
Conclusion . . . . .	294
Interview with Vlad Mihalcea . . . . .	295
Example 3: REST-service on the Pi to toggle an LED . . . . .	297
Info REST-controller . . . . .	297
GPIO Manager . . . . .	298
GPIO REST-controller . . . . .	301
Running the application on a Pi . . . . .	302
Conclusion . . . . .	306
Example 4: Reactive data . . . . .	307
The code . . . . .	307
Running the streaming application on the Pi . . . . .	311
Conclusion . . . . .	312
<b>Just a thought: Impostor Syndrome . . . . .</b>	<b>313</b>
<b>Chapter 11: Message Queues . . . . .</b>	<b>314</b>
Using Mosquitto on the Pi . . . . .	316
Installation . . . . .	316
Testing Mosquitto on the Pi . . . . .	317
Example 1: Share data between Pi and PC . . . . .	319
Modifying the pom and module-info . . . . .	319
Connecting and publishing to Mosquitto . . . . .	320
Subscribing to Mosquitto . . . . .	320
The user interface . . . . .	321
Example 2: Control Arduino from JavaFX via Mosquitto . . . . .	323
Defining the messages . . . . .	324
The Arduino part . . . . .	324
The Java application . . . . .	333
The finished setup . . . . .	346
Tip: Checking the network packages between Arduino and Pi . . . . .	346
<b>Conclusion . . . . .</b>	<b>348</b>

# Chapter 1: Introduction

Welcome! Whether you are a newbie in Java, and want to learn from examples, or know the language, and want to get started with electronics and a Raspberry Pi, I hope you will find valuable new information in here!

This is not a book to learn every aspect of the Java language. There are already a lot of those written by way better programmers than me, so please check your bookstore or an online course if you want to get a real deep knowledge of the Java programming language. My goal was to bundle a **lot of sample code and information**, I collected while learning and experimenting, and to get anyone interested in Java to **take a quick start by using the examples to set up their own experiments**. By following these examples, you will also learn the language bit by bit.

Over the last couple of years, I focused on Java in my professional job. But on the other hand, I also got involved in CoderDojo (coding club for children) where I first was able to make a blinking LED with some simple code on Arduino and a Pi.

If I see how kids learn to work with electronics and programming at CoderDojo, I'm jealous those things didn't exist when I was a kid. Yes, I managed to control a relay-board with my Commodore 64 but it took me years to collect all needed knowledge and components to reach that goal. Now thanks to online shopping and knowledge sharing that same result can be achieved in days or hours...

As a professional Java-developer with a love for open-source, I set myself as a personal goal for 2019 to run a recent Java version (Java 11 or newer) on the Pi, with a **JavaFX user interface, and control an LED** with it. Very basic I know, but it resulted in a series of blog posts about each step to reach this goal. Based on these blog posts, I also wrote an article for MagPi (in the Dutch and French July 2019 edition) and [produced an Udemy course](#)<sup>1</sup>.

And as a **blinking LED is the equivalent of a “Hello World” program**, I continued with experimenting and ended up with... this getting started book! A whole list of easy “recipes” to get you started with many different Java libraries and electronics to build your projects. Take one or more of these “recipes”, combine and mix them to your taste, experiment and explore!

Oh, and this is not an anti-Python or anti-C book! Java is the language I love and use the most, but **for every problem you need to select the best solution**. And in some cases, this could be Java or something else...

---

<sup>1</sup><https://www.udemy.com/course/use-java-11-and-java-fx-11-on-a-raspberry-pi>

## Content overview

You can read this book from start to end, but can also use it as a “cookbook” and select the topics of your interest. For instance, you don’t need to know the history of Java to work with it, but it’s here for your reference.

In between some of the chapters you find a short “Just a thought”-piece about the things I find important and I would like to share with you. I’m also very happy I could interview some of my heroes, included in the chapters they are related to.

**Chapter 1: Introduction:** On the next pages you can find more info about the sources used in this book and are available for free online, and some additional info about readme-files and a thank-you-list.

**Chapter 2: Tools and hardware used in this book:** Setup a Raspberry Pi with Raspbian and an overview of the software used on both Pi and PC. Also generic info about some of the most-used hardware components in this book.

**Chapter 3: Choosing an IDE:** Info about IntelliJ IDEA and Visual Studio Code.

**Chapter 4: About Java:** History of Java, different versions and tools within the eco-system and how to install it on your Pi and PC. And of-course a crash course so you get a grasp of the language if it’s completely new to you.

**Chapter 5: Raspberry Pi pinning:** Different types of Raspberry Pi-boards and the different headers, pins and pin types and how you can use them to connect different types of hardware.

**Chapter 6: What is Maven?:** More info on Maven, the tool we will use to test, build and run our Java projects on PC and Pi.

**Chapter 7: About JavaFX:** The visual Java framework we will use to build beautiful user interfaces to interact with our Pi and hardware.

**Chapter 8: Bits and bytes:** The magic of ones and zeros and how they are combined into longer values.

**Chapter 9: PI4J:** A framework that makes it easier to work with the GPIOs from Java with multiple hardware examples.

**Chapter 10: Spring:** Building a Java application which exposes our Pi as a web service to store data or control the GPIOs.

**Chapter 11: Message Queues:** Use your Pi as a message handler to receive and send messages to different devices like other Pi’s, PCs or Arduino’s.

## About me

I'm Frank Delporte ([@FrankDelporte](https://twitter.com/FrankDelporte)<sup>2</sup> on Twitter, °1973) who started programming at age 11 when I got a Commodore 64. With Basic as the programming language, some magnetic switches, and a relay board, I managed to control my Lego train. After studying at a film school, I started my professional career as a video editor, but quickly got involved in the first multimedia and online video projects in Belgium. Combining my interest in multimedia and programming, this resulted in a journey through C#, Flash, HTML/JavaScript, Flex, SQL, Qt, Java...

Currently, I work as a technical lead and software developer at Televic Rail in Belgium with a team of highly skilled hard- and software-engineers. We are building the next generation of Passenger Information Systems used onboard rail vehicles to inform the passengers in the best possible way so they are fully informed about their journey. Making a bridge between on- and off-board data sources empower us to combine information into nice looking and easy to read screens.

I always try to “**get things moving and done**” while trying to stick to the **KISS principle** (see “Important abbreviations”).

As the best way to learn is teaching, I was one of the first lead coaches of CoderDojo in Belgium and started two Dojo's (Roeselare and Ieper) where I first came into contact with Arduino, Pi and all those other amazing tools, all thanks to the inspiring colleague-coaches.

On my blog [webtechie.be](https://www.webtechie.be)<sup>3</sup>, some projects are shared, which resulted in an article for MagPi which was the starting point for this book.

I live together with my wife Conny and son Vik in Passendale (Belgium), a small town that was one of the main battlegrounds of WWI.

---

<sup>2</sup><https://twitter.com/FrankDelporte>

<sup>3</sup><https://www.webtechie.be>

## Sources and scripts used in this book

Within this book, many open-source projects are described and used. So **all the sources and scripts which are given as examples in this book are also shared as open source.**

### Where to find them

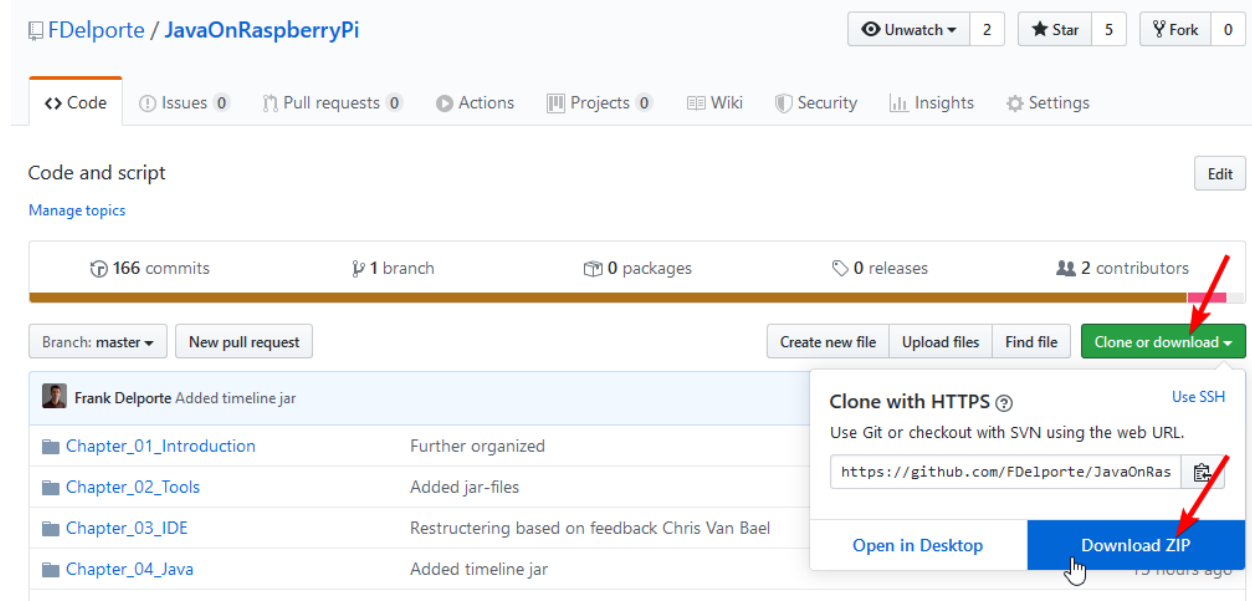
Everything is combined into one GitHub project:

[JavaOnRaspberryPi](https://github.com/FDelporte/JavaOnRaspberryPi)<sup>4</sup>. Keep an eye on this project for any changes and corrections. Changes will be listed in the file [CHANGES.md](https://github.com/FDelporte/JavaOnRaspberryPi/blob/master/CHANGES.md)<sup>5</sup>.

The sources are structured to match the chapters in this book so you can easily find each example.

### Get the sources

If you are new to GitHub (source control systems), just click the download button on the site and you will get a full copy of everything in one zip file.



Get sources from GitHub as ZIP

On the other hand, if you know how to checkout code, please do! It will give you a copy of all the code and scripts but also allow you to make changes and improvements and send them as pull requests if you know how to work with git. **It would be great if the samples could be further improved by you! I'm looking forward to your pull request.**

Here is how you check out the code into your home directory on the Pi:

<sup>4</sup><https://github.com/FDelporte/JavaOnRaspberryPi>

<sup>5</sup><https://github.com/FDelporte/JavaOnRaspberryPi/blob/master/CHANGES.md>

```
1 $ cd /home/pi
2 $ mkdir Book
3 $ cd Book
4 $ git clone https://github.com/FDelporte/JavaOnRaspberryPi.git
```

Inside most of the projects, you'll find a "target" directory with the jar-file which you can run directly if you don't want to modify the code and/or build the project yourself. More info on building and running later in this book...

## The styling used in the book

All the code and scripts in this book are formatted like this, where “...” means a portion of the code is skipped in this book, but fully available in the sources.

```
1 public class HelloWorldExample{
2     public static void main(String args[]){
3         System.out.println("Hello World !");
4         ...
5     }
6 }
```

Command and script examples have the same styling, and when the line starts with a “\$” that means it is a command you must type in in the terminal, the following lines are the output you will get, for example, for the command “java -version”:

```
1 $ java -version
2 openjdk 11-BellSoft 2018-09-25
3 OpenJDK Runtime Environment (build 11-BellSoft+0)
4 OpenJDK Server VM (build 11-BellSoft+0, mixed mode)
```

All references to the sources in the GitHub project will look like this, referring to the subdirectory in [the GitHub project](https://github.com/FDelporte/JavaOnRaspberryPi)<sup>6</sup>:



<https://github.com/FDelporte/JavaOnRaspberryPi>  
Chapter\_01\_Introduction > README.md

Tips will be displayed like this.



Don't forget to check the README file!

---

<sup>6</sup><https://github.com/FDelporte/JavaOnRaspberryPi>

## Read the README!

There is a README file in the root folder, but also in each chapter, sometimes also in the projects. This will help you to easily use the scripts, set up a project, etc.

The README of most chapters also has a “Read more” section where you can find links to more info about the topics of that chapter.

Adding a README file is a good practice, even when you are the only person working on a project! Minimally it needs to contain a title and description, how to work on the project and how to build, test and run it.

## Guidelines

A README uses a list of conventions for the formatting, the ones used most in this book:

- 1 \* `“##”`, `“###”`, `“####”` ... for titles
- 2 \* `“*”` for bullets
- 3 \* `“[title](link)”` for links
- 4 \* Three single ``` before and after a code block

## Example

This is a short example file:

```
1 # Chapter 3: Java
2 Chapter describing the history and current state of Java and how to install on a Pi.
3
4 ## Links
5 * [AdoptOpenJDK](https://adoptopenjdk.net/)
6 * [BellSoft Liberica JDK](https://bell-sw.com/)
7 * [Duke the Java Mascot, explained](https://jaxenter.com/duke-the-java-mascot-explained-118397.html)
8 * [SDKMAN](https://sdkman.io/)
9 * [SDKMAN on Windows](https://ngeor.com/2019/12/07/sdkman-windows.html)
10
11 ## Scripts
12 See the scripts directory to install specific Liberica SDK versions on the Pi.
13
14 To check the installed java version run this command in the terminal:
15
16 ```
17 $ java -version
18 ```
```

Screenshot of the source of a demo readme file

When you look at this file online it will look like this:





Screenshot of the demo readme file as shown on GitHub

## What's next?

In the source files you can find an overview of links with interesting articles on how and why to write a good README:



Chapter\_01\_Introduction > README.md

## Thanks to...

...all open-source contributors and bloggers who are constantly pushing Java, JavaFX, Pi4J and all those other marvelous tools, frameworks, libraries forward. Without them, we wouldn't be able to produce such easy to build and good-looking applications! Only a limited list is mentioned in this book, but there are a lot more and they are constantly sharing their work and knowledge on the internet. Keep an eye out for them!

...my colleagues who are always critical when doing pull requests and sharing their knowledge. I fully agree with the phrase "What one developer can do in one month, can be done by two developers in two months", but two experienced and critical developers will produce better code, than a solo player.



Read the book "[The Mythical Man-Month](https://en.wikipedia.org/wiki/The_Mythical_Man-Month)" by Fred Brooks<sup>7</sup> if you want to know why a late software project becomes even later when adding manpower.

Or even better, buy two copies of this book for your manager, so he can read it twice as fast ;-)

...Elektor who triggered me to start writing this book and publish it as a real paper book! A true bucket list thing achieved now.

...my wife and son, who I have neglected too much the last months. I promised them that would change once this book was finished. It's a pity they know me too well and immediately asked what my next project would be :-)

...the reviewers, interviewees and everyone who helped me to realize this book!

Reviews, contributions, tips, feedback, ideas... by Stijn Deroo, Nathalie Delporte, Lieven Baes, Trisha Gee, Vlad Mihalcea, Kasper Zutterman, Ludo Joris, Jan Lamote, Kim Harding, Catherine Van Damme, Chris Van Bael, Mark Heckler, Pieterjan Deconinck, Kasia Pranke, Marian Faydula, Wouter Vanhauwaert, Michael Egger and... you? Please send me your feedback via e-mail on [javaonraspberrypi@webtechie.be](mailto:javaonraspberrypi@webtechie.be)! You can also send me pull-requests on GitHub if you want to contribute to the examples.



### Legal stuff

Raspberry Pi and the associated logo are trademarks of The Raspberry Pi Foundation. The name and logo used throughout this book and their trademarked status are acknowledged here.

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

Other names may be trademarks of their respective owners.

---

<sup>7</sup>[https://en.wikipedia.org/wiki/The\\_Mythical\\_Man-Month](https://en.wikipedia.org/wiki/The_Mythical_Man-Month)

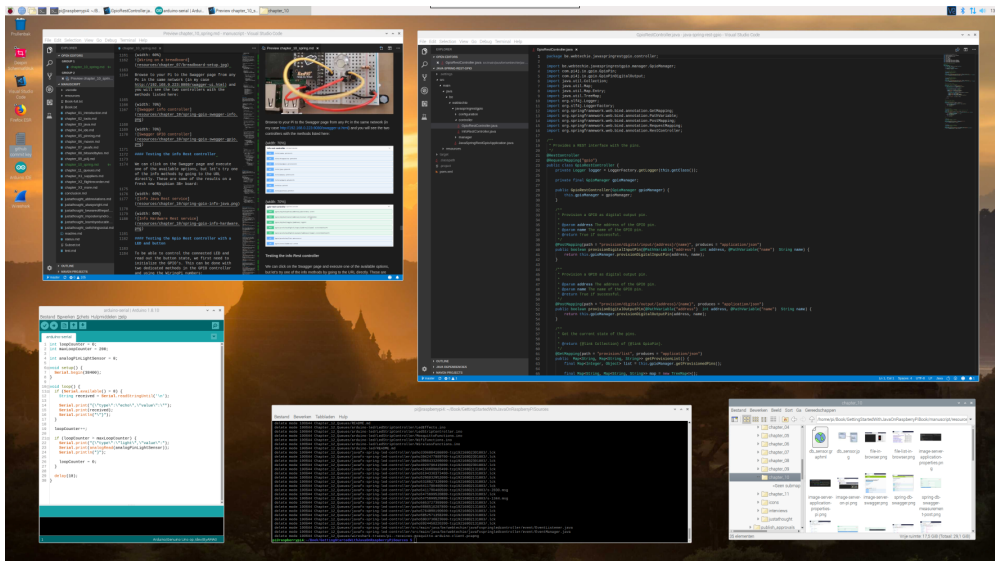
# Chapter 2: Tools and hardware used in this book

In the past, most good tools were expensive and as such, used by a limited user group. But thanks to open source soft- and hardware, and freeware developers, you can now build awesome stuff without the need to kill your wallet.

You will find many of these in this book. Are these the best? Are these the only ones to be used? No, definitely not! But they are what I have used and considered to be “good enough for the job” and easy to use, which is the goal of this book.

## Raspberry Pi

Just when I started working on this book, the 4th edition of the Pi became available. Combined with a 4K display this turned out to be a perfect companion while testing and building stuff. It's not required, but maybe it's a good idea to buy a fan as the board can become pretty hot. But WOW, what a magnificent device! Most of the writing and coding for this book was done on a Pi 4! And this was also the first time I used a 4K display, another WOW! Two code windows open on one single screen, each using less than half of that screen, and still space for some other tools like a terminal window.



Multiple programs open on a 4K display with Pi 4

As you always risk “burning” a GPIO pin during experiments, most of the experiments were done on some older - cheaper - Pi 3 boards.

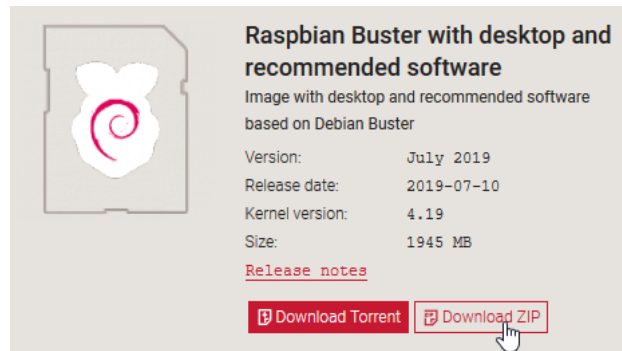
Keep in mind, you can use the same SD card with Raspbian and Java JDK on different Pi's (except the oldest ones with an ARMv6 processor, see “Chapter 4: Java” > “Install Java on a Pi”). When you have a working SD card with all the applications you need, you can take it out from a Pi 4 and use it with a Pi 3 board or vice-versa.

## Prepare the Pi

### Operating system on SD

When you start with a new Pi, you will need to prepare an SD card with the operating system. You can download this from the Raspberry Pi site by selecting “Raspbian with desktop and recommended software” from the [downloads page](https://www.raspberrypi.org/downloads/raspbian/)<sup>8</sup>.

<sup>8</sup><https://www.raspberrypi.org/downloads/raspbian/>



Download link on the Raspberry Pi website

Select the ZIP and wait until the download has finished. In the meantime, you can also download a tool to “burn” this file to the SD card. On PC I prefer to use [Rufus](https://rufus.ie/)<sup>9</sup>, which you can download in a portable version and run without the need to install anything.



On Linux you don’t need to install additional software to burn the downloaded OS to an SD card but can use the “dd” command in the terminal.

Make sure you select the correct location to write to, by following this [step-by-step tutorial on the Raspberry Pi website](#)<sup>10</sup>.

The current Raspbian OS with desktop and recommended software, needs around 7GB. If you use a micro SD card of 8GB, only 1GB is left, which is not enough space to experiment with Java and applications... So it’s recommended to use a card of 16GB or more. You will also need an adaptor to use the micro SD in most SD slots of a PC. So we put the small SD in the adaptor to write the operating system on it on a PC.

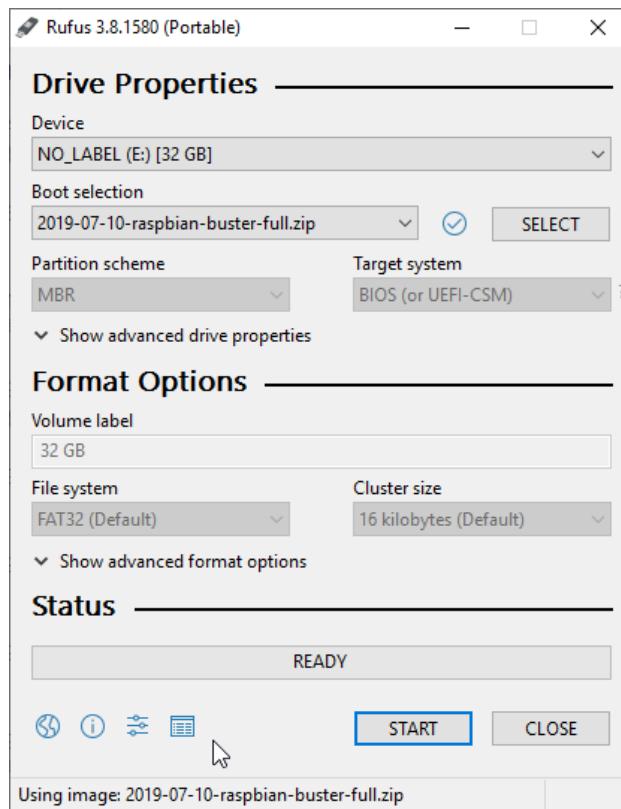


Micro SD card and adapter

<sup>9</sup><https://rufus.ie/>

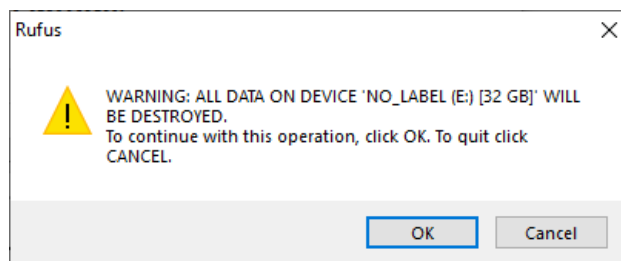
<sup>10</sup><https://www.raspberrypi.org/documentation/installation/installing-images/linux.md>

Run Rufus and select the “Device” (make sure to select the SD card and not the hard drive of your PC!!!) and the downloaded ZIP file with Raspbian as “Boot selection”.



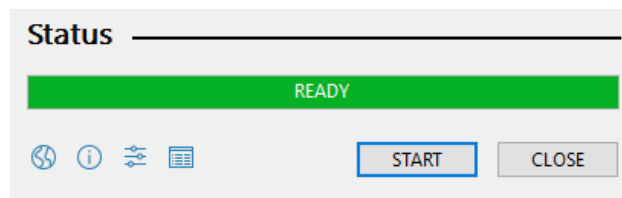
Rufus select SD and zip file

You can leave the default settings for the other inputs, and click “START”, after which you have to confirm.



Confirm overwriting the SD card

This is your last chance to make sure you selected the correct device! After clicking “OK” the Status bar will show the progress, wait until “READY” appears.



Bottom part of Rufus UI when finished

You can now remove the SD card from the PC and put the small card in the Pi. Watch out for the orientation, don't push too hard because that means you didn't correctly insert the card.

Now start your Pi and you will soon get your desktop view. Click around a bit to check the pre-installed programs. You can also configure the WiFi connection to connect it to your local network.

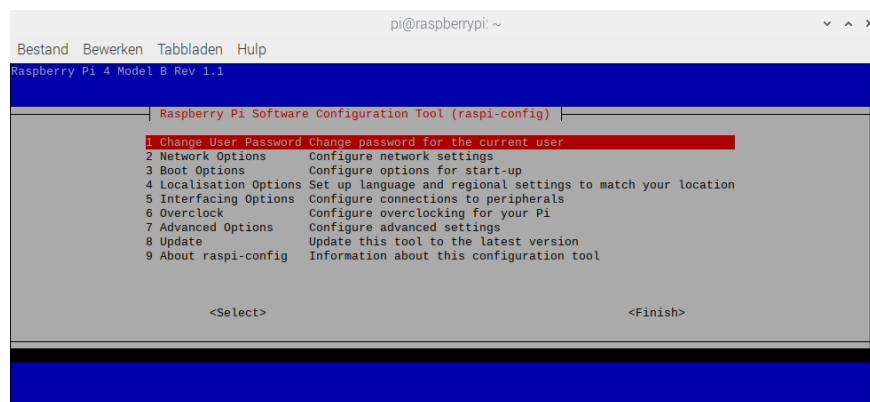


Don't just unplug the power, but turn off in Raspbian. Otherwise, you might get a corrupted SD card! This topic is discussed in detail on [StackExchange](https://stackoverflow.com/questions/381/how-do-i-turn-off-my-raspberry-pi)<sup>11</sup>.

## Resize the root partition of the SD card

When you start the Pi for the first time with a new SD card, the root partition will be resized automatically to use the maximum available capacity. If you need to do this manually, you can use “raspi-config” which provides a lot of configuration and maintenance options. Start it from the terminal on your Pi:

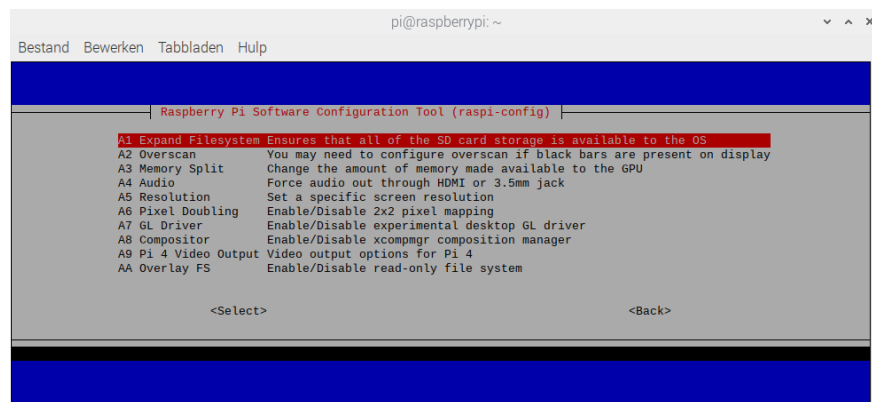
```
1 $ sudo raspi-config
```



Raspi-config main screen

To make sure you are using the latest version of this tool, start by selecting option “8 Update”. The tool will restart and now choose option “7 Advanced options”.

<sup>11</sup><https://raspberrypi.stackexchange.com/questions/381/how-do-i-turn-off-my-raspberry-pi>



Raspi-config advanced screen

In this advanced screen, select the first option “A1 Expand Filesystem”. Your Pi will need to restart and then you can check the disk usage with the following command in the terminal:

```
1 $ df -h
```

```

pi@raspberrypi:~ $ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root        29G   6.0G   22G   22% /
devtmpfs        459M   0   459M   0% /dev
tmpfs           464M   5.7M   458M   2% /dev/shm
tmpfs           464M   13M   451M   3% /run
tmpfs           5.0M   4.0K   5.0M   1% /run/lock
tmpfs           464M   0   464M   0% /sys/fs/cgroup
/dev/mmcblk0p1  253M   52M   201M   21% /boot
tmpfs           93M   0   93M   0% /run/user/1000
pi@raspberrypi:~ $

```

Disk usage of the SD card

The disk is split into multiple partitions, but the biggest part is used by the root partition where all programs, our files... are stored. In my case, using a 32GB card, the root partition is 29GB with around 22GB free space left.

## Connections between Pi and breadboard

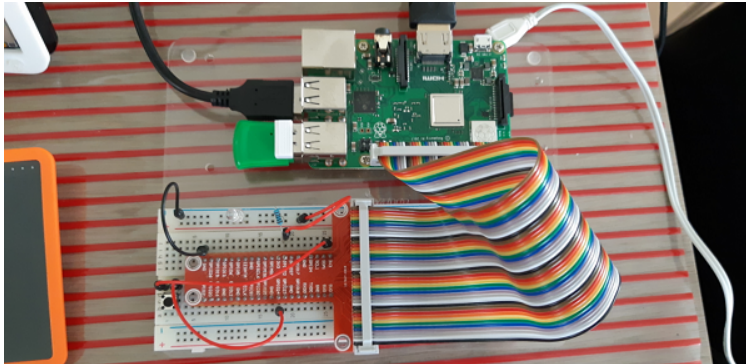
It's very easy to connect wires and hardware to the Pi, but the pin numbering can be confusing. Different solutions are available, of which I tested these:

### T-board with extension cable

This T-shaped board can be plugged in directly in a breadboard and you can easily read the pinning numbers from it. The T-board is an identical copy of the connector on the Pi. The drawback is the



big cable which takes a lot of space. You can find a lot of different ones online, for example, on ebay when searching for “rasberry pi t-shaped”.

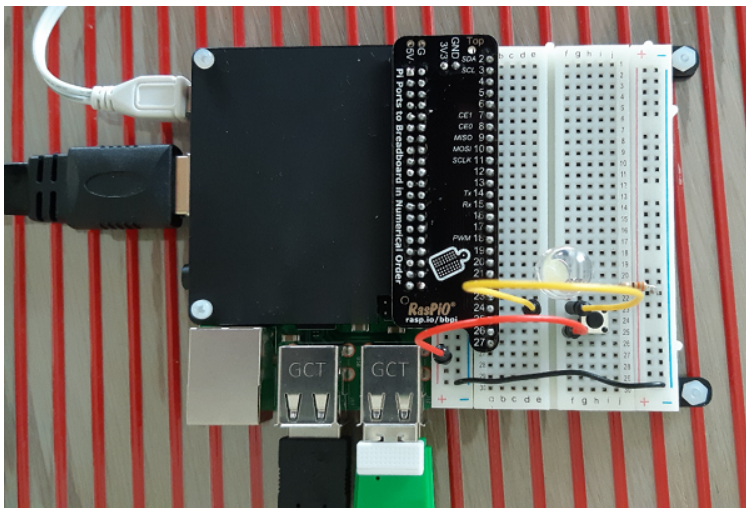


T-board with cable to Pi

## Breadboard Pi Bridge

For [this board](#)<sup>12</sup> you have to nicely align the Pi and the breadboard before you push it down, trying not to bend the pins (been there, done that...). If you follow the [online video manual](#)<sup>13</sup>, you will get a nice clean result and the GPIO pinning numbers will now end up in numerical order!

Without the flat cable, it is much more compact and my preferred solution. And the additional cover on the Pi avoids damaging it.



Pi bridge board

My test setup consists of this bridge-board on a Pi 3, some additional breadboards, and a 7” touchscreen. All this glued together on a wooden board to make it easier to put away when not

<sup>12</sup><https://shop.rasp.io/products/breadboard-pi-bridge-pi-ports-to-breadboard-in-numerical-order>

<sup>13</sup><https://youtu.be/imxyzSrO0wI>

needed while keeping everything nicely organized. The orange wireless keyboard was part of a [Kano kit](https://kano.me/uk)<sup>14</sup> with a much older Pi, which can not run Java.



Pi 3 test setup with multiple breadboards

---

<sup>14</sup><https://kano.me/uk>

## Software tools on the Pi

### Linux commands crash course

**Clemens Valens**<sup>15</sup> of Elektor shared this overview [in a blog post](#)<sup>16</sup> and was happy to have it included in this book. Thanks a lot, Clemens!

Because it is almost impossible to work on the Raspberry Pi, or, for that matter, on Linux in general, without ever needing to enter commands in a terminal, here is a list of frequently used commands, which are executed by the command interpreter “Bash”. A terminal is that black window in which you can only type text. Sometimes it is also called a Command Line Interface or CLI.

There are many commands and most commands accept all sorts of parameters and arguments. To find out more on a specific command, add “-help” (two dashes) to it, e.g.:

```
1 rm --help
```

In the following “[path]” refers to a relative or absolute path. An absolute path starts with “/”, e.g.

```
1 /home/pi
```

Please note that the table below is not exhaustive and your favorite commands may be missing. Some of these commands are only available on Debian-based Linux systems like Raspbian for Raspberry Pi.

Command	Description
pwd	Display the name of the current working directory.
ls	List the content of the current directory.
ls [path]	List the content of the specified directory.
ls -l	List the content of the current directory with additional information.
ls -a	List all files including hidden files beginning with ‘.’ (i.e. dotfiles).
ls -h	List all files and display the file sizes in a human readable format.
cd [path]	Change the current directory to [path].
cd ..	Change to the parent directory (note the space between ‘cd’ and ‘..’).
cd /	Change to the root directory (note the space between ‘cd’ and ‘/’).
cd ~	Change to the home directory (determined by \$HOME environment variable).
mkdir [name]	Create the directory [name] in the current working directory.

<sup>15</sup>[https://twitter.com/Clemens\\_Elektor](https://twitter.com/Clemens_Elektor)

<sup>16</sup><https://www.elektormagazine.com/news/bash-command-cheat-sheet>

Command	Description
<code>rmdir [name]</code>	Remove the empty directory [name] from the current working directory.
<code>rm [name]</code>	Remove the specified file.
<code>rm *</code>	Remove all the files from the current working directory.
<code>rm -r *</code>	Remove all the files and directories from the current working directory.
<code>cp [from] [to]</code>	Copy a file from source [from] to destination [to].
<code>cp -r [from] [to]</code>	Copy everything including directories from source [from] to destination [to].
<code>mv [from] [to]</code>	Move a file from source [from] to destination [to].
<code>mv -r [from] [to]</code>	Move everything including directories from source [from] to destination [to].
<code>find</code>	Search for files matching certain patterns.
<code>sudo [command]</code>	Superuser do. Execute [command] with elevated privileges. Allows you to do things you are not entitled to. Common examples include:
<code>sudo raspi-config</code>	Launch the Raspberry Pi configuration tool.
<code>sudo reboot</code>	Safely restart your system.
<code>sudo shutdown -h now</code>	Safely shutdown your system now.
<code>sudo apt-get install [package]</code>	Install a package.
<code>sudo apt-get update</code>	Update the list of packages without installing anything.
<code>sudo apt-get upgrade</code>	Upgrade the installed packages to the versions obtained with 'apt-get update'
<code>sudo chown pi:root [name]</code>	Change the owner of [name] to 'pi' and set the group to 'root'.
<code>sudo su</code>	Become Superuser for more than one command.
<code>cat [name]</code>	Show the contents of a file.
<code>head [name]</code>	Show the beginning of a file.
<code>tail [name]</code>	Show the end of a file.
<code>chmod [who][+,-,=][permissions] [name]</code>	Change the permissions for a file.
<code>chmod u+x [name]</code>	Add execute permission for the owner of the file.
<code>chmod 777 [name]</code>	Allow every user to read, write and execute the file [name].
<code>tar -cvzf [name] [path]</code>	Create compressed file [name] from the contents in [path].
<code>tar -xvzf [name]</code>	Extract the contents of a compressed file.
<code>wget [uri]</code>	Download a file from the Internet.
<code>man [command]</code>	Show the manual page for a command.
<code>man man</code>	View the manual page of the 'man' command.
<code>nano [path]</code>	Open text editor with given file.
<code>grep 'string' [name]</code>	Search inside one or more files for occurrences of 'string'.



Jan Lamote (one of the reviewers of this book) pointed out to me that “apt-get” can also be done with “apt”, as this is the command now being recommended. You can find more info and a comparison between the two on [this nice post by Abhishek Prakash](https://itsfoss.com/apt-vs-apt-get-difference/)<sup>17</sup>.

Let's try out a few examples, open a terminal and let's start with a directory listing “ls”. By default,

<sup>17</sup><https://itsfoss.com/apt-vs-apt-get-difference/>

the terminal will start in your home location and this command will show you the content of the current directory, so in this case the home directory “/home/pi”.

```
1 $ ls
2 Desktop Documents Downloads MagPi Music Pictures Public Templates Videos
```

By adding a path to the ls-command, we can get the content for a given directory, for example for the root of the system “/” or for the home directory “/home”:

```
1 $ ls /
2 bin dev home lost+found mnt proc run srv tmp var
3 boot etc lib media opt root sbin sys usr
4
5 $ ls /home
6 pi
```

To get more info we can do the same with the additional “-lha” parameters to include:

- l: add additional information (user rights, owner, size, etc.)
- a: including hidden files beginning with “.” (i.e. dotfiles)
- h: display the file sizes in a human readable format.

```
1 $ ls -lha
2 total 41M
3 drwxr-xr-x 18 pi pi 4.0K Jan 7 18:59 .
4 drwxr-xr-x 3 root root 4.0K Sep 26 01:09 ..
5 -rw-r--r-- 1 pi pi 220 Sep 26 01:09 .bash_logout
6 -rw-r--r-- 1 pi pi 3.5K Sep 26 01:09 .bashrc
7 drwxr-xr-x 7 pi pi 4.0K Jan 7 19:00 .cache
8 drwx----- 6 pi pi 4.0K Jan 7 19:00 .config
9 drwxr-xr-x 2 pi pi 4.0K Sep 26 01:46 Desktop
10 drwxr-xr-x 2 pi pi 4.0K Sep 26 01:46 Documents
11 drwxr-xr-x 2 pi pi 4.0K Sep 26 01:46 Downloads
12 drwx----- 3 pi pi 4.0K Sep 26 01:46 .gnupg
13 drwxr-xr-x 3 pi pi 4.0K Sep 26 01:18 .local
14 drwxr-xr-x 2 pi pi 4.0K Sep 26 01:46 Music
15 drwxr-xr-x 2 pi pi 4.0K Sep 26 01:46 Pictures
16 drwx----- 3 pi pi 4.0K Jan 7 18:59 .pki
17 -rw-r--r-- 1 pi pi 807 Sep 26 01:09 .profile
18 drwxr-xr-x 2 pi pi 4.0K Sep 26 01:46 Public
19 drwxr-xr-x 2 pi pi 4.0K Sep 26 01:46 Templates
```

```

20 drwxr-xr-x  2 pi  pi  4.0K Sep 26 01:46 Videos
21 drwx----- 3 pi  pi  4.0K Jan  7 18:29 .vnc
22 -rw-r--r--  1 pi  pi   165 Jan  7 18:54 .wget-hsts
23 -rw-----  1 pi  pi    56 Sep 26 01:46 .Xauthority
24 -rw-----  1 pi  pi   2.3K Sep 26 01:46 .xsession-errors

```

In this book we will be using the text editor “nano”, so let’s take a look into the manual with “man nano”:

```

1  $ man nano
2  NANO(1)                General Commands Manual          NANO(1)
3
4  NAME
5      nano - Nano's ANOther editor, an enhanced free Pico clone
6
7  SYNOPSIS
8      nano [options] [[+line[,column]] file]...
9
10 DESCRIPTION
11     nano is a small and friendly editor. It copies the look and feel of Pico,
12     but is free software, and implements several features that Pico lacks,
13     such as: opening multiple files, scrolling per line, undo/redo, syntax
14     coloring, line numbering, and soft-wrapping overlong lines.
15
16     When giving a filename on the command line, the cursor can be put on a
17     specific line by adding the line number with a plus sign (+) before
18     the filename, and even in a specific column by adding it with a comma.
19
20     As a special case: if instead of a filename a dash (-) is given, nano
21     will read data from standard input.

```

The actual manual is longer, this is only the initial part. Press q to exit from it. Now let’s take a few steps to:

- Create a directory “experimenting” in the root-temp directory “/tmp”.
- Start the text editor with the file “mytextfile” and type in “Hello!”.
- To exit and save your changes:
  - \* CTRL+X -> exit
  - \* Y -> to save the changes
  - \* Enter -> to confirm the filename
- Check the content of the created file with “head”.
- Check the size of the created file with “ls -l”.
- Remove the file again.

```
1 $ mkdir /tmp/experimenting
2 $ ls -l /tmp/experimenting/
3 total 0
4 $ nano /tmp/experimenting/mytextfile
5 $ head /tmp/experimenting/mytextfile
6 Hello!
7 $ ls -l /tmp/experimenting/
8 total 4
9 -rw-r--r-- 1 pi pi 7 Dec 15 22:07 mytextfile
10 $ rm /tmp/experimenting/mytextfile
11 $ ls -l /tmp/experimenting/
12 total 0
```

Some additional tips:

- Use the tab key to auto-complete a path, e.g. type in “ls -l /t” and hit the tab-key. This will autocomplete to “ls -l /tmp/” as there is only one directory starting with “/t”.
- Use the up and down key in the terminal to reuse one of the commands you have typed in before.

## Firefox

Raspbian comes with Chromium (from Google) pre-installed, but that’s not my preferred browser (see “Just a thought: Switching social”), so I installed Firefox with following commands:

```
1 $ sudo apt-get update
2 $ sudo apt-get upgrade
3 $ sudo apt-get install firefox-esr
```

## VNC server

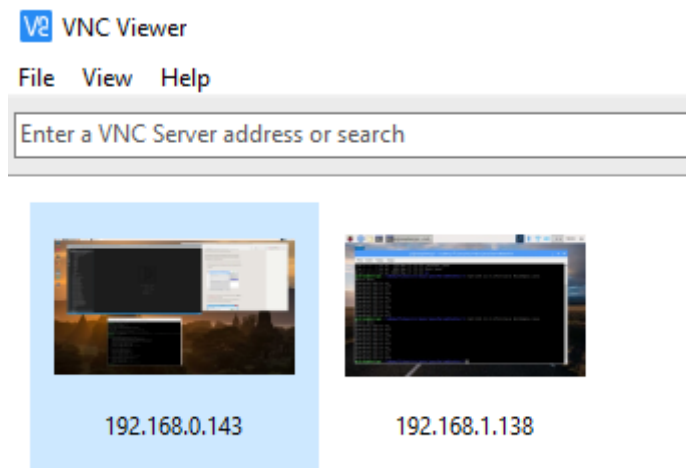
When you use multiple Pi’s, only have one monitor and/or your Pi is somewhere in an unreachable place, you can still easily work with it using a remote desktop tool. VNC server is the easiest one to use.

Raspbian includes VNC Server by default. Only if you want to add the viewer, you will need to run the following command in the terminal, in case you want to use the viewer on the Pi itself to start a session to another one.

- 1 `$ sudo apt-get update`
- 2 `$ sudo apt-get install realvnc-vnc-viewer`
- 3 `$ sudo raspi-config`

In raspi-config select “5 Interfacing options” > “VNC” > “Yes” to enable remote VNC connections.

On your PC you’ll need to install [VNC Viewer](https://www.realvnc.com/en/connect/download/viewer/)<sup>18</sup> to connect to your Pi remotely. This application will keep a list of all your connections. You can find the IP of your Pi with the terminal and the “ifconfig” command.



VNC Viewer

When you first connect to a new Pi, the credentials to be used are “pi” for the username, and “raspberrypi” for the password. As with all default passwords, you should change them ASAP! Open the terminal and type the following command:

- 1 `$ passwd`

You are asked for the current password, and a new one and you’re done! BTW, you can also do the same via “raspi-config”, option “1 Change user password”.

## Enable SSH on the Pi

As of the November 2016 release, Raspbian has the SSH server disabled by default. It can be enabled manually from the desktop:

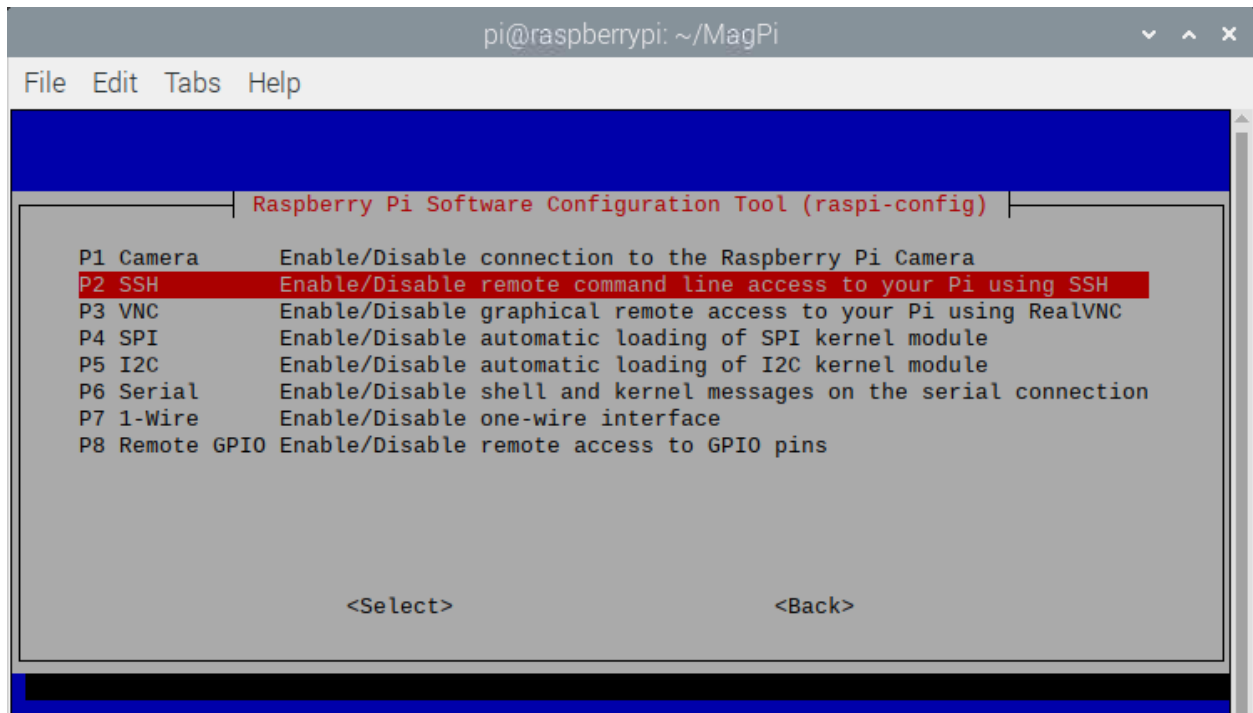
- Launch Raspberry Pi Configuration from the Preferences menu

---

<sup>18</sup><https://www.realvnc.com/en/connect/download/viewer/>



- Navigate to the Interfaces tab
- Select Enabled next to SSH
- Click OK



Enable SSH in raspi config

## Free software tools on PC

Most of these tools are available in versions for Windows, Apple and/or Linux, or similar applications can easily be found.

## Integrated development environment aka IDE

I'm a visual developer. While "die-hards" stick to text only and the terminal, I always use a lot of different programs that show you what's happening. Even for back-end applications that run somewhere hidden on a far-away server, I always try to have some kind of simple web page to at least know the program is running...

Yes, you are allowed to call me old fashioned ;-)

There are a lot of different IDE's you can use for Java development, but in this book, Visual Studio Code and IntelliJ IDEA are used. More about that later in "Chapter 3: Choosing an IDE".

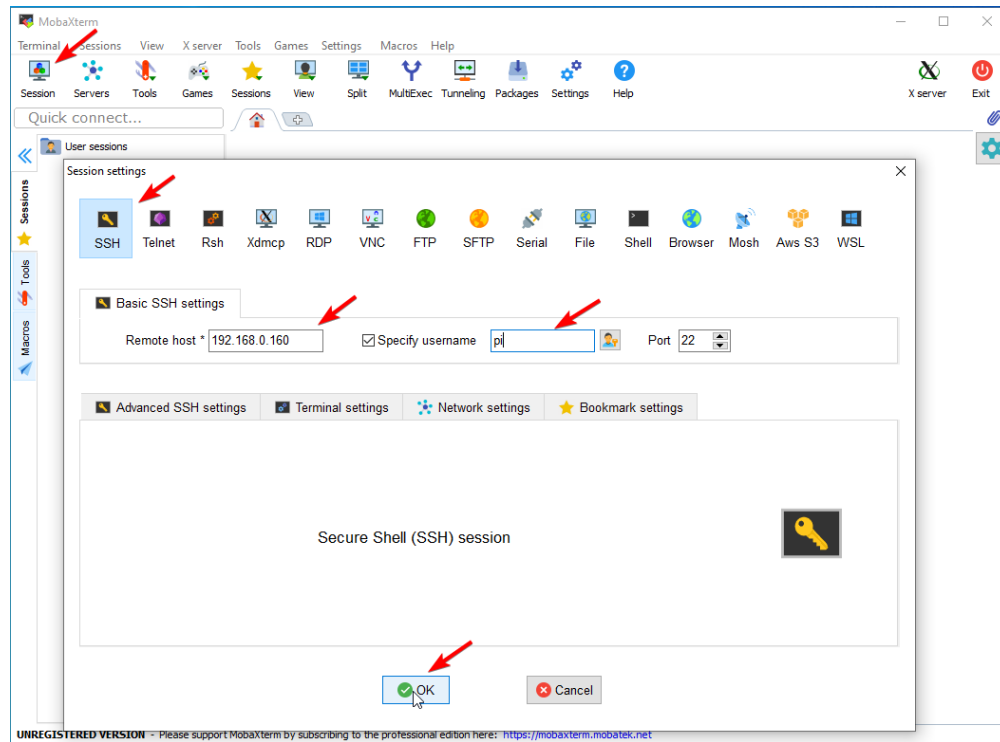
## Remote connection to a Raspberry Pi (SSH)

To work on your Pi from a PC, or upload a file, SSH (Secure Shell) is the way to go (see "Enable SSH on the Pi" earlier in this chapter). [MobaXterm](https://mobaxterm.mobatek.net/)<sup>19</sup> is a great choice for Windows.

To start a new session, click on the "Session" button and in the popup select "SSH", fill in the IP address of your Pi (which you can find on the Pi itself in the terminal by typing in "ifconfig"), specify the username ("pi" by default) and click "OK".

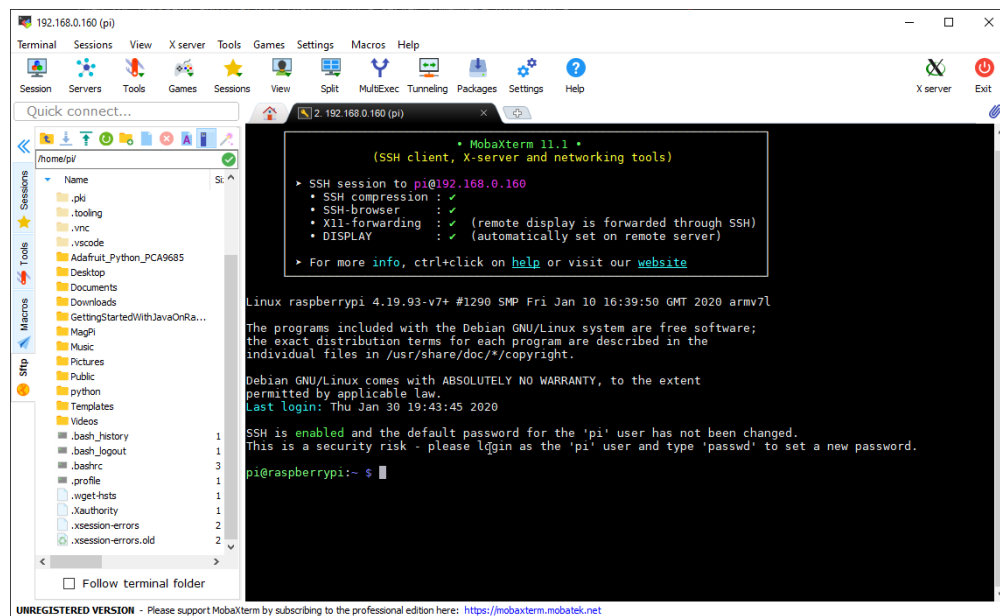
---

<sup>19</sup><https://mobaxterm.mobatek.net/>



Setting up a new connection in MobaXterm

You will be asked for the password which is “raspberrypi” by default, but you should have changed that when you first booted your Pi for obvious security reasons... When this is OK you get both a directory listing and terminal window, both in your home directory (“/home/pi”).



MobaXterm after login on Pi

To upload a file, e.g. after you compiled a Java jar-application, you can easily drag it from your PC into the directory listing window of MobaXterm.

If you are using Linux as your workstation, you can connect directly from the terminal with the command “ssh pi@IPADDRESS”. When this is the first login, you will need to confirm. E.g. in my case:

```

1  $ ssh pi@192.168.0.160
2  The authenticity of host '192.168.0.160 (192.168.0.160)' can't be established.
3  ECDSA key fingerprint is SHA256:w4QRlobqhj98enNnqTbIpAJ6NpWiLUXLCbweHCA1Em8.
4  Are you sure you want to continue connecting (yes/no)? y
5  Please type 'yes' or 'no': yes
6  Warning: Permanently added '192.168.0.160' (ECDSA) to the list of known hosts.
7  pi@192.168.0.160's password:
8  Linux raspberrypi 4.19.93-v7+ #1290 SMP Fri Jan 10 16:39:50 GMT 2020 armv7l
9
10 The programs included with the Debian GNU/Linux system are free software;
11 the exact distribution terms for each program are described in the
12 individual files in /usr/share/doc/*/copyright.
13
14 Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
15 permitted by applicable law.
16 Last login: Sat Feb  1 12:06:00 2020 from 192.168.0.135
17
18 SSH is enabled and the default password for the 'pi' user has not been changed.
19 This is a security risk - please login as the 'pi' user and type 'passwd' to set a n\
20 ew password.
21 pi@raspberrypi:~ $

```

You see you even get a warning when you didn't change the password! Oops, now you know I didn't do that, but in my defense, I started with a fresh Raspbian a few times for this book to clean up my test environment, so none of these default password versions survived long... ;-)

Copying a file to and from a Pi can also be done in the Linux terminal with the “scp SOURCE DESTINATION” command. These are a few examples to create a txt file, copy it to a Pi and copying it back to the PC:

```

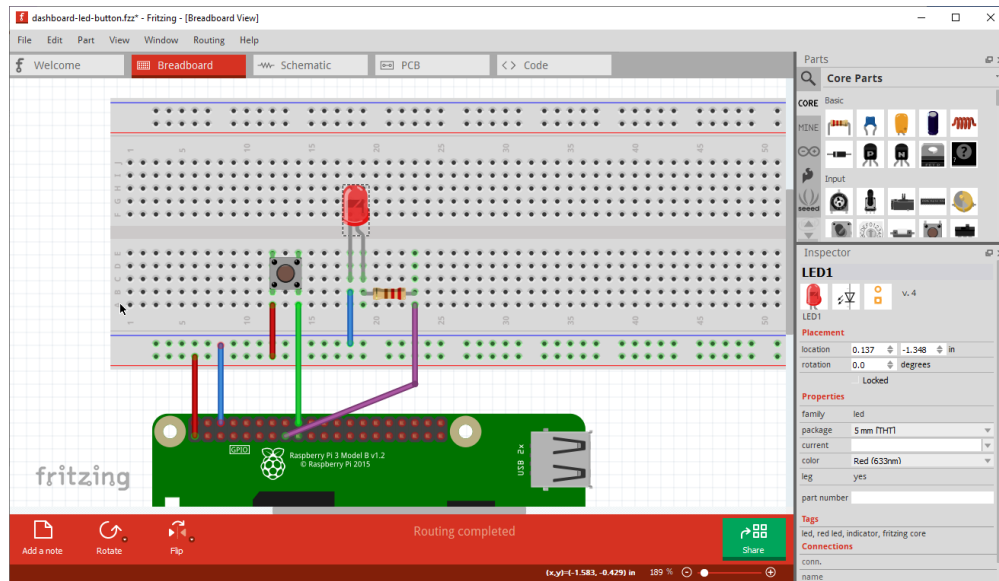
1  $ nano test.txt
2  $ scp /home/pi/test.txt pi@192.168.0.160:/home/pi/copied.txt
3  $ scp pi@192.168.0.160:/home/pi/copied.txt /home/pi/copied_back.txt

```

## Wiring diagrams

The wiring diagrams in this book are made with [Fritzing](https://fritzing.org/)<sup>20</sup>

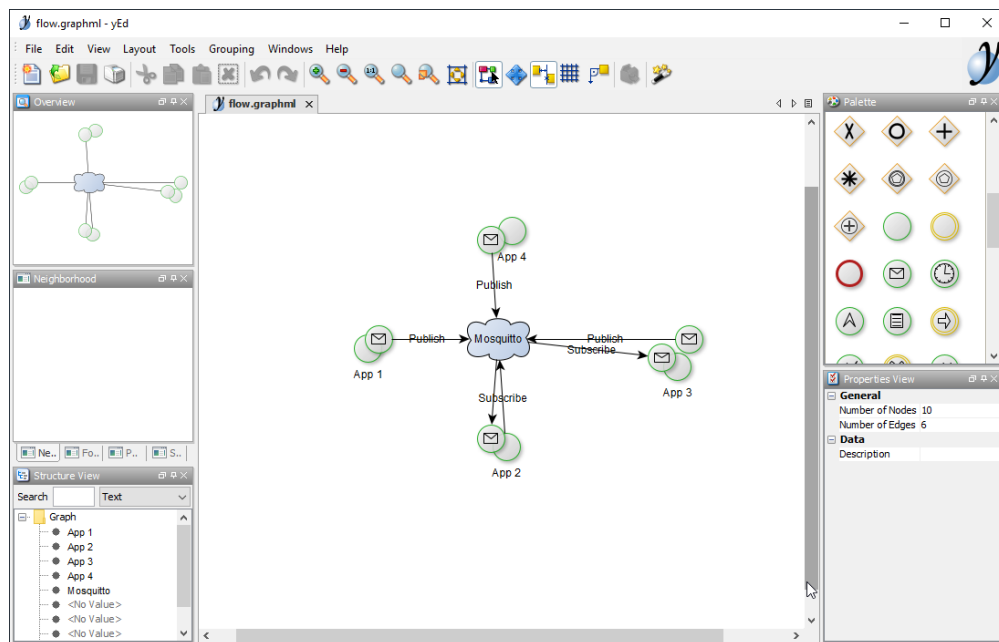
<sup>20</sup><https://fritzing.org/home/>



Screenshot Fritzing

## Schematic drawings

Diagrams for this book were made with [yEd Graph Editor](https://www.yworks.com/) by [yworks](https://www.yworks.com/)<sup>21</sup>, a free tool to quickly draw diagrams. They also provide commercial libraries (for Java, .NET, web) to include such drawing tools into your software.















Screenshot yEd

<sup>21</sup><https://www.yworks.com/downloads#yEd>

## Hardware components

### Resistors

These are the base components we will need in all of our projects. And as it is impossible to keep them separated, you will need to be able to calculate their value based on the color-coding. Most of the resistors in use have 4 color rings and you will need this table to calculate the value.

Name	Color	Value	Multiplier	Tolerance	Temp. Coeff.
BLACK	 #000000	0	1Ω		250ppm/K
BROWN	 #761d1d	1	10Ω	1%	100ppm/K
RED	 #cc0000	2	100Ω	2%	50ppm/K
ORANGE	 #ffa500	3	1KΩ		15ppm/K
YELLOW	 #ffff00	4	10KΩ	0%	25ppm/K
GREEN	 #008000	5	100KΩ	0.5%	20ppm/K
BLUE	 #0000cc	6	1MΩ	0.25%	10ppm/K
VIOLET	 #ee82ee	7	10MΩ	0.1%	5ppm/K
GREY	 #808080	8	100MΩ	0.25%	1ppm/K
WHITE	 #ffffff	9	1GΩ	0%	
GOLD	 #ffd700		0.1Ω	5%	
SILVER	 #c0c0c0		0.01Ω	10%	
NONE				20%	

Color coding table



This color-coding table is created as a JavaFX application. JavaFX is explained later in this book. But you can take a look at the sources if you want to experiment with it or use it to calculate the values of a resistor as described below.

Chapter\_02\_Tools > javafx-resistors

The most used type of resistor used in Pi- and Arduino-projects is a 330Ω to limit the power for an LED.



330Ω

Gold can never be the first one, so we need to start on the other side.

- the first two bands give us the value = orange-orange = 33
- the third band is a multiplier = brown = 10Ω
- the fourth band is the tolerance = gold = 5%

- so  $(33 * 10) = 330\Omega$  with 5% tolerance

By using the JavaFX application we indeed get the same result:

**Resistor value calculator (3, 4, 5 or 6 bands)**

ORANGE ORANGE BROWN GOLD

Resistor value is 330Ω with tolerance 5.0%

Calculate the resistor value for orange - orange - brown - gold



1MΩ

Let's take another one:

- brown (1), black (0), green (100KΩ) multiplier, gold (5) tolerance
- $10 * 100K\Omega$  with 5% tolerance
- = 1MΩ with 5% tolerance

**Resistor value calculator (3, 4, 5 or 6 bands)**

BROWN BLACK GREEN GOLD

Resistor value is 1MΩ with tolerance 5.0%

Calculate the resistor value for brown - black - green - gold

## LEDs



Mix of different LEDs

Next to resistors, LEDs are also most used in the projects in this book. They come in all colors and sizes. A Light-Emitting Diode, as the name says, emits light when current flows through the diode, blocking the current flowing in the opposite direction. The voltage drop caused by the LED mostly lays between 1.2 to 3.6 volts with a recommended current between 10 to 30 mA.

They have a positive and negative pin and it's easy to remember **which is the positive one. The longest one! You have more of it ;-)** In case the pins are cut off, the flat edge of the plastic cap is the negative side. The current in the LED can only flow from the positive (anode) to the negative (cathode) side!

As a Pi outputs 3.3V on the GPIOs and most Arduinos output 5V, we will need a resistor to limit the voltage so we don't damage the LED.



Formula to calculate the resistor value:

$$\text{resistorValue} = (\text{sourceVoltage in V} - \text{ledVoltage in V}) / \text{ledCurrent in A}$$

For example for a 3.3V GPIO with an LED of 2.2V and current 20mA.

You can find these values on the technical sheet or packaging of your LEDs.

$$(3.3\text{V} - 2.2\text{V}) / 0.02\text{A} = 55\Omega$$

To be on the “safe side”, in most cases we will use a 330 $\Omega$  resistor.

The JavaFX application also has a calculation tool for this:

**Led resistor calculator**

Input voltage (V) 3,3 Led voltage (V) 2,2 Led current (A) 0,02

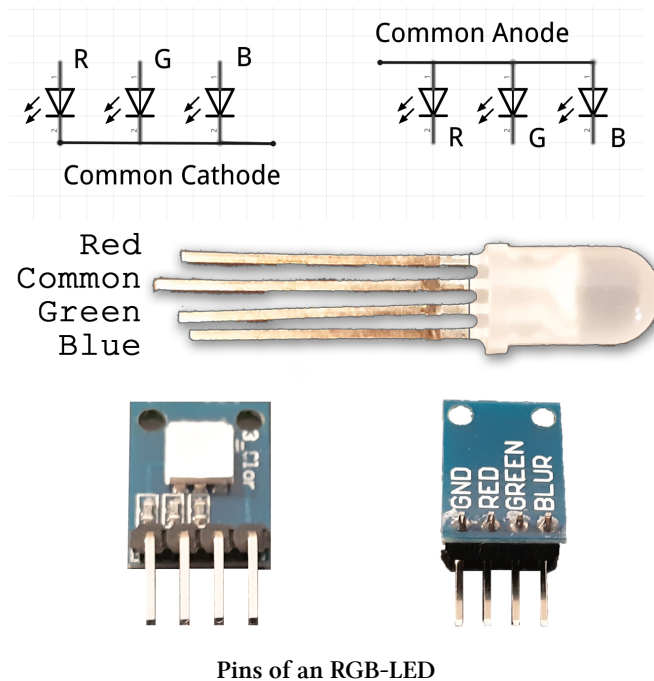
**A resistor with value 55 $\Omega$  is needed**

Calculate the resistor value for an LED

## RGB-LED

RGB-LEDs combine three different colors in one component. They share either a common cathode (-) or anode (+). Luckily you can't damage them by connecting them the wrong way, so you can quickly check with a fixed power pin from the Pi (don't forget the resistor!). As you will see in “Chapter 9: Pi4J > Digital GPIO input and output examples > Example 1: Digital output with RGB-LED” the different colors don't have the same brightness and you'll need to check the datasheet to define the correct resistors to be used for every pin.





The common pin is the longest one, in the order red-common-green-blue.

Some (Arduino) starter kits, contain an RGB-LED attached to a small PCB with breadboard pins. In that case, you need to check the board itself for the correct connections. In the picture above you can see an example that has clear indications on the back.

## LED strips

Led strips are available in these two most used “flavors”: WS2812B (improved version of the WS2811, also known as the AdaFruit NeoPixel) versus WS2813 and their main difference is the number of connections (3 versus 4).

Two important notes for LED strips:

- it’s not enough to connect power to them, they won’t do anything until a controller sends data.
- they need a lot of power so don’t take power from the Pi, but use an external power supply.

## Power supply for LED strips

To know which power supply you need, some calculation is required... Check the datasheet of the strip you are using to know how much current a single LED in the strip uses at maximum brightness. As an example, take one which uses 60mA max. per LED. By the way, this is only the case for full white, so the real use will be lower if you are using colors only.

Current/LED	LED/m	Total 1m	Total 5m
60mA	30	1.8A	9A
60mA	60	3.6A	18A
60mA	144	8.64A	43.2A

If we only want to do some testing, we can use a short piece with e.g. only 10 LEDs. So this would require 0.6A (600mA) at full brightness. But even this is too much for the Pi. The 5V pin gets its power from the USB power supply, which is also used by the Pi-system itself, of course, leaving only a few hundred mA left for this pin.

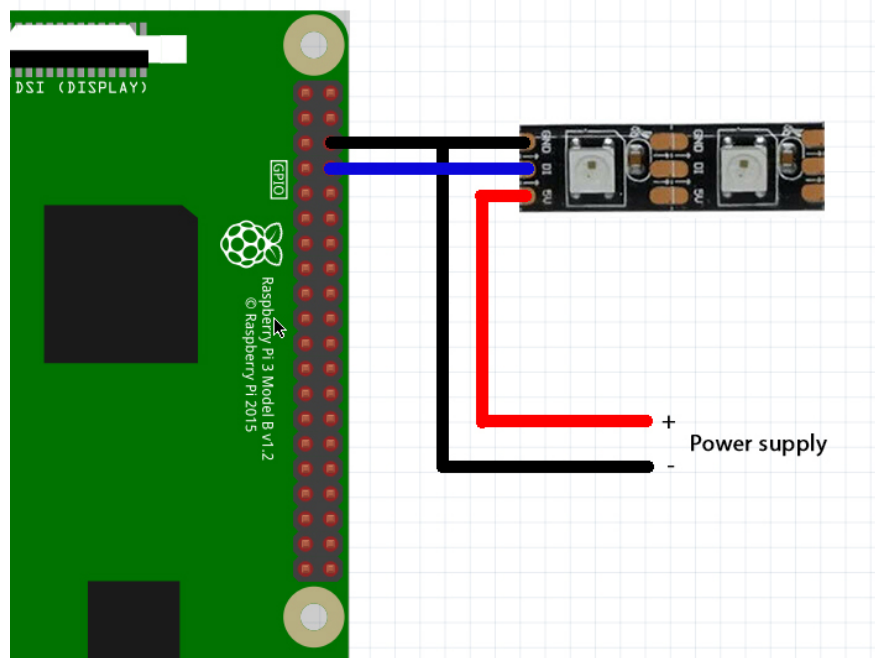
For an Arduino, it depends on the type and power supply but about 400mA to 800mA should be available. And indeed as I found out myself while setting up the example project described in “Chapter 11: Message Queues”, a short strip can be powered directly from an Arduino.



Side remark, that Arduino example didn't work at some point because I had the board connected to my PC. So the board was powered by my PC, while I used a long LED-strip with a separate power supply. Because they didn't share the same ground, the data was completely messed up and the LEDs didn't show the effect I was expecting. So remember to ground them together!

## Conclusions

- Pi: connect the power for the led strip in all cases to a separate power supply.
- Arduino: OK for testing a short strip.
- Make sure board and strip share the same ground.

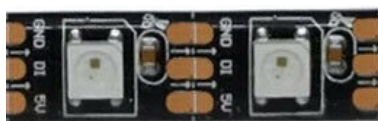


Connect LED strip to Pi with external power source



Watch out for eye damage while experimenting with strips as they can be very bright!

### WS2812B = controlled with one connection

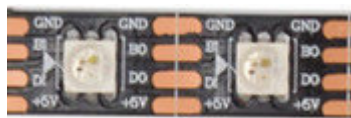


WS2812B

maximum.

Every LED in such a strip can be separately controlled because each one has a WS2812 chip embedded in it, notice the small dark spot inside the LED. In most of the cases, this is a 5050 LED (= 50x50 mm). Controlling the LEDs is done via one single data wire. These strips can be controlled with a refresh rate of 400Hz

### WS2813 = controlled with two connections



WS2813

The WS2813 is an updated version of the WS2812B and uses two data wires, so the rest of the strip will continue to work if one LED breaks. Another advantage is the higher refresh rate of 2000Hz, but this is specifically important for display usage, while in this book we will just do “fun LED-strip” experiments.

Disadvantage: they are slightly more expensive compared to WS2812B-strips.

## Electronic kit with Arduino board

We will also be using Arduino, LEDs, resistors... in the examples in this book. If you buy a good starter kit, you will get most of the components needed for the examples in this book.

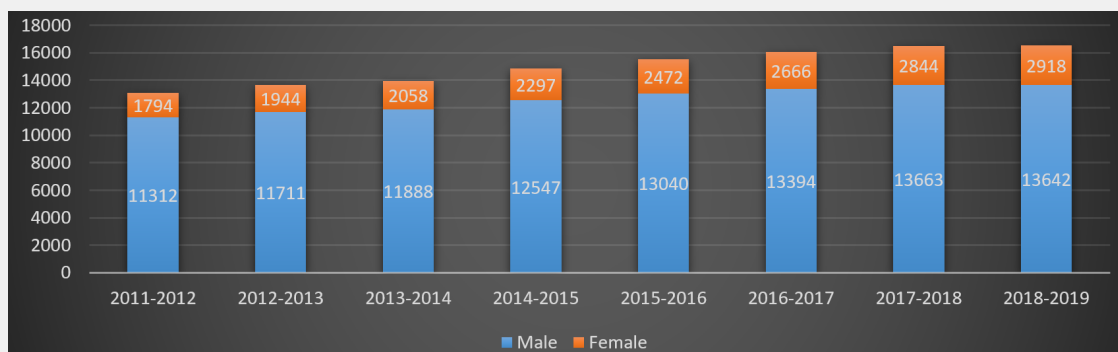
Make sure you have a breadboard and wires so it is easy to quickly set up the examples.



Arduino learning kit available at [Elektor.com](http://Elektor.com)

# Just a thought: Learn by educating

I started two [CoderDojo](#)<sup>a</sup> clubs in Belgium (Roeselare and Ieper) at the start of 2014 and today still organize the one in Ieper. CoderDojo is a club for kids to learn to program. Not with real courses, but by guiding the kids to experiment and discover. The coaches are only there to help to find a solution when all else fails. At that time in 2014, there was a lot of “fuzz” about the lack of engineers for software and electronics projects.



Engineering students in Flanders

This chart is based on the number of engineering students (electromechanics, digital arts, and entertainment, electronics-ICT, multimedia and communication technology,...) in Flanders, Belgium<sup>b</sup>. Thanks to CoderDojo and other STEAM-initiatives<sup>c</sup>, this kind of education gained popularity and the female group grew from 15% to 21% in all these years.

Of all the students (240.332) who started high school in 2018-2019, only 16.560 or 7% started in this engineering direction, while it almost guarantees you to find a fun and challenging job! As STEAM will become more and more important in the future, in a society that relies heavily on technology, we need to inspire more people...

A long intro to point out the fact I started “educating” kids not only in CoderDojo, but also at work with children of colleagues in a [First Lego League](#)<sup>d</sup> team. That’s the moment you discover you know nothing... or at least not what kids are asking. Also, the moment you start experimenting with new things and a new world opens its doors for you. For me, that was the moment I discovered Arduino, Raspberry Pi, Mindstorms and many more. Because I believe in the power of knowledge sharing, I also started blogging, writing, presenting and again discovered new worlds.

You can only explain something if you master it or research, try out, fail, restart, document. Exactly what I tried to do for this book...

<sup>a</sup><https://coderdojo.com/>

<sup>b</sup>Publications on [www.vlaanderen.be/publicaties](http://www.vlaanderen.be/publicaties) > “Hoger onderwijs in cijfers”

<sup>c</sup>Science, Technology, Engineering, Arts and Mathematics

<sup>d</sup><http://www.firstlegoleague.org/>

## Interview with Karen Mouws



**Karen Mouws @LimsKaren<sup>a</sup>, Educational Designer by day, Maker Educator whenever I can. Part-time adult and proud of it.**

*The IT world is mainly a (white) male environment. Why is this a problem in your opinion?*

I firmly believe that diversity in a team or company will always have benefits. Bringing multiple perspectives to the table helps identify hidden problems, novel solutions and challenges existing thinking patterns. A lack of diversity can cause a narrow scope in the long term and start excluding users if they don't fit the profile of the happy flow.

Getting more women into IT is a mission well on its way: there are so many initiatives out there, helping women 'break into' the IT sector. A bigger challenge will be keeping them there. It is never easy to be "the odd one" on the team or to bump into metaphorical walls, such as your voice not being heard or being constantly underestimated as a professional.

*What is your "plan of attack" to inspire kids to get them involved in the digital world?*

Children and teens are often already involved in the digital world, now more than ever, but can get "stuck" on the consumption levels. Only a small group of them will start creating content, from funny videos to small games. I think we need to break this pattern on multiple levels: introduction; creative production and visibility.

First of all, you can't love what you don't know: if no one introduces you to code as a fun toolbox, how will you ever try it out? That's why I have such a big love for CoderDojo and WeGoSTEM, who reach thousands of children every year.

But we also need to move beyond the first, fun intro: how do we keep them engaged? I think that's where long term passion projects come into play. Other interests and hobbies will intermingle with learning to code or physical computing. For example, if you have a dog or cat at home, why not try to build a game that is controlled by them? I rarely opt for "more serious" projects, as I love the power of silly ideas to challenge yourself.

Visibility is super important as well: we need more role models. It is so easy to think "oh that is not for me" or "programming is for boys" if all the role models you see are Adam Savage or Elon Musk. As inspiring as they are, I would love to see Sophy Wong or Simone Giertz on posters on the wall as well.

*Free tools like [Scratch<sup>b</sup>](#), [Code.org<sup>c</sup>](#), [Blockly<sup>d</sup>](#) and many more, allow children of any age to get to learn the principles of coding. When do you think they should switch from such drag-and-drop based tools to "real" programming languages?*

Block-based coding is an amazing entry point for programming projects, as you can focus on the idea and the creativity, without the fear of making syntax mistakes or the frustration of missing a ";" somewhere deep in your code. I'd say moving on to written code will come up around 12-14 years

old, but someone who has been in a CoderDojo since age 6 might be a lot faster to transition. Projects like [EduBlocks<sup>e</sup>](#) can be an easy step up, to take away the “fear” of written code, by combining blocks and Python into one editor. [MakeCode<sup>f</sup>](#) by Microsoft offers a similar function with Javascript and so on. Seeing the effects of your block-based code on the written code and vice versa is a very strong learning experience.

*You are “Raspberry Pi Certified”. What does this mean and how did you achieve this?*

Raspberry Pi has a training program for educators interested in teaching physical computing called Picademy. For three days, a group learns all about using the Pi in an educational context, how to code basic things, but also how to explain them and utilize the Pi in projects. I love that it combines technical, creative and didactical knowledge into one awesome program. I would suggest anyone into code education (teachers, Dojo leads, library educators...) to sign up. It is free, as long as you are willing to travel to the UK for it. I was the first Belgian educator to participate, but I’m happy that others have followed and I hope many more will.

*Why is now the perfect time to learn to program?*

We live in a day where tech has more and more become a closed system. We use phones with apps on them all the time, we have smart technology in our houses. But very few people understand how it works, and feel a lack of control over them. To me, learning to program means taking back that control and breaking open the black box. Not everyone should become a professional programmer, but every child, teen or adult can benefit from understanding (basic) coding logic.

*Which DIY-programming-electronics-project are you working on, or is on your “if I ever have time” list?*

I would need more hours in a day/week but I’ve been wanting to assemble an RGB light-up scarf for a while now, as well as a Raspberry Pi-powered selfie booth for my dog.

---

<sup>e</sup><https://twitter.com/LimsKaren>

<sup>h</sup><https://scratch.mit.edu/>

<sup>i</sup><https://code.org/>

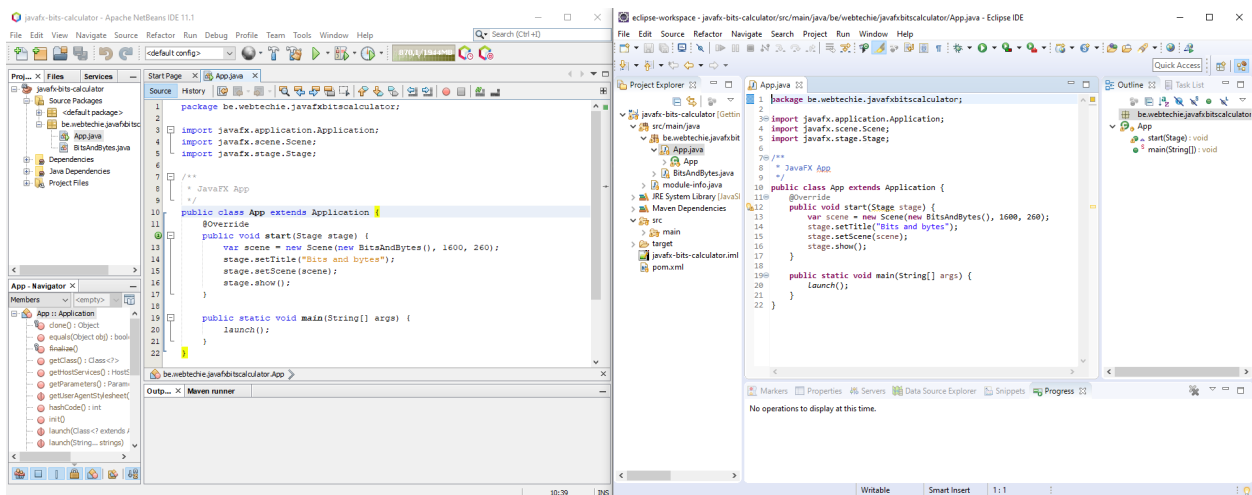
<sup>j</sup><https://blockly.games>

<sup>k</sup><https://edublocks.org/>

<sup>f</sup><https://www.microsoft.com/en-us/makecode>

# Chapter 3: Choosing an IDE

An IDE (= Integrated Development Environment) is your best friend while developing applications. It will help you to write the correct code and compile it to a running application. Some of the most used ones are IntelliJ IDEA, Visual Studio Code, NetBeans and Eclipse.



NetBeans and Eclipse side-by-side with the same project

Apache NetBeans<sup>22</sup> is a free and open-source IDE which you can use for Java, JavaScript, PHP, C/C++... The same goes for the Eclipse IDE<sup>23</sup>. Both are very popular editors but I will focus on two other ones. Selecting an IDE is a very personal thing. Once you are familiar with one and know the shortcuts to work fast, it will be hard to switch. So try them out, find which one fits best to your needs, go for it and don't look back ;-)

I started using Eclipse when I first programmed Java but switched a few years ago to IntelliJ IDEA (for Java) and Visual Studio Code (for Java, JavaScript, and experimenting). These tools just seem to work better for (with) me.

<sup>22</sup><https://netbeans.org/>

<sup>23</sup><https://www.eclipse.org/ide/>



## IntelliJ IDEA

IntelliJ IDEA is a commercial product built by the company JetBrains, headquartered in the Czech Republic. They sell a lot of developer tools, but IntelliJ IDEA is the most known one and it is the most used Java IDE by professional developers. There is a free version (the Community edition) which can be used for Java and Android development, while the Ultimate edition is a paid version with additional features, as you can see on the [download page](#)<sup>24</sup>.

Code completion (= hinting) and powerful refactoring tools are the main advantages of an IDE which help you to write your code faster as the possible methods are shown, including the parameters.

```
HBox bitSelection = new HBox();
bitSelection.setSpacing(10);
```

Code hinting in IntelliJ IDEA

Another great feature of IDEA is the visualization of the parameter name in your code, so you can easily check if you used the correct value when calling a method.

```
HBox bitSelection = new HBox();
bitSelection.setSpacing(10);
bitSelection.setPadding(new Insets(topRightBottomLeft: 5));
```

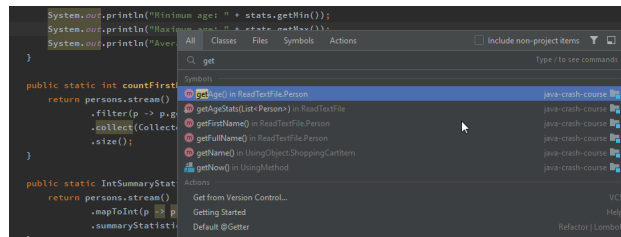
Parameter name hints in IntelliJ IDEA



The magic key of IntelliJ IDEA: double-tap shift

You can speed up your development by learning the shortcuts. CTRL-C and CTRL-V are probably the best known to copy and paste. For every IDE there is a long list, but if you only remember one for IntelliJ IDEA it's double tapping the shift key for "Search Everywhere". It will show you a popup where you can type in anything you are searching for like a class name, a method you've written somewhere in your code, a setting of the IDE...

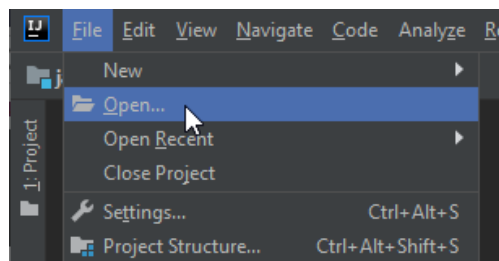
<sup>24</sup><https://www.jetbrains.com/idea/download/>



Search Everywhere in IntelliJ IDEA

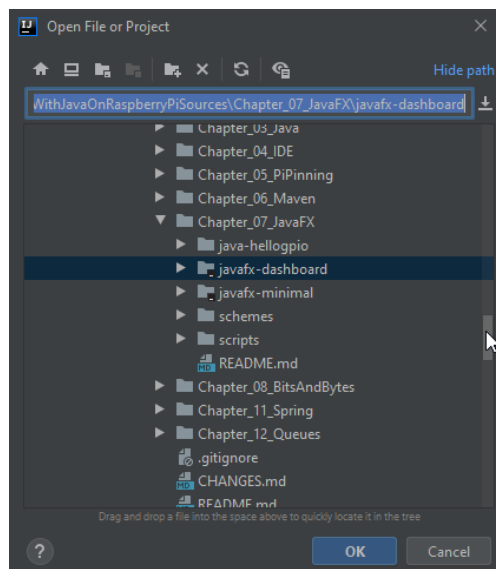
## Using IntelliJ IDEA with the example projects

To open one of the example projects from the source, just click File > Open and select the source directory.



Opening a project

In the next window select the directory of the project you want to open and click OK.



Selecting the project

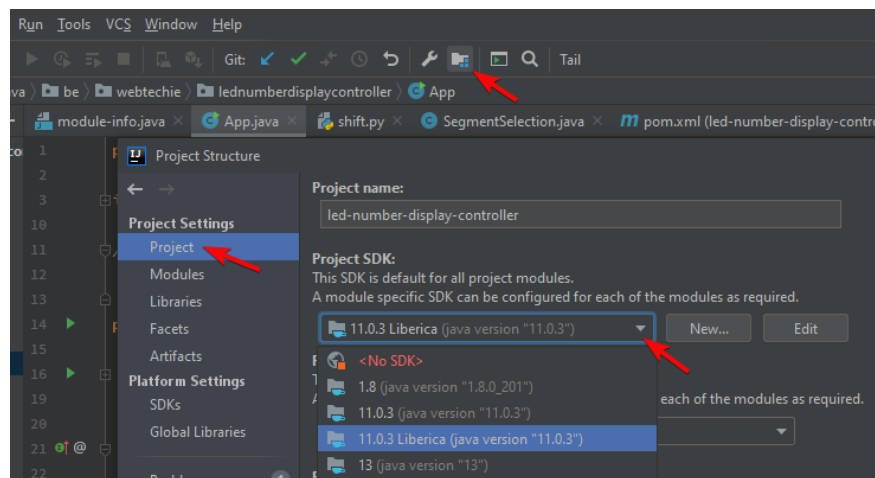
IntelliJ IDEA will need some time to load the Maven dependencies (see Chapter 6 for more info), and when this is finished everything is ready to explore the files and start coding yourself.

In case you have multiple Java SDK's installed, you may need to help IntelliJ IDEA to define which one to use.



Warning to select the Java SDK

Keep in mind you will need at least JDK 11 for the Java example projects. You can use the same one we will be using on the Pi (Liberica JDK from BellSoft, see Chapter 4 on how to get and install this JDK) or use the latest one from [AdoptOpenJDK](https://adoptopenjdk.net/)<sup>25</sup>. You can define the SDK to be used for every project via the “Project Structure” button (SHIFT+CTRL+ALT+S).



Select the Java SDK for a project

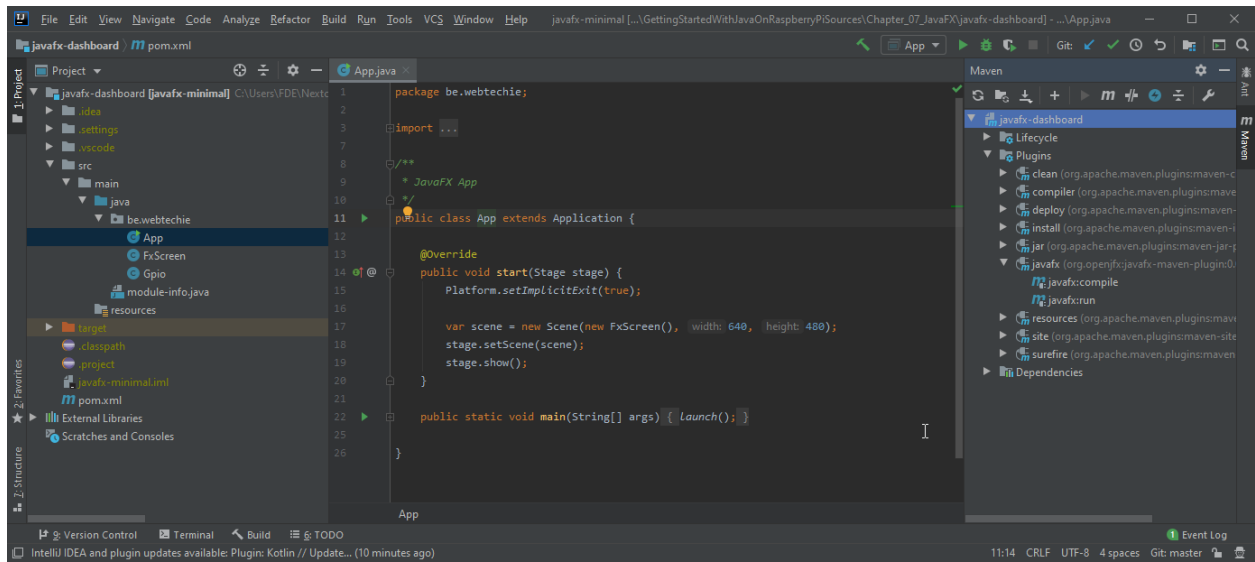
In the next screenshot, you can see an open JavaFX project from the examples.

- Left - Project window: here you can browse all the files in your project
- Middle - Editor window: containing an open Java file
- Right - Maven window: more info in Chapter 6, but click here on
  - \* “Plugins > javafx > javafx:compile” to compile the code to bytecode which will also reveal if the code is OK
  - \* “Plugins > javafx > javafx:run” to start the application



In some cases when the code doesn't want to start or build, select “Build” in the top menu and click “Rebuild Project” to clean up everything inside the project. When this is done, you can try to run your project again.

<sup>25</sup><https://adoptopenjdk.net/>



Project in IntelliJ IDEA

## Interview with Trisha Gee



**Trisha Gee, Coder, blogger, speaker, Developer Advocate at JetBrains, @trisha\_gee<sup>a</sup>**

*Java is changing a lot with the new releases every 6 months. What is the impact for IntelliJ IDEA?*

Obviously, we have to keep up with the changes. It's particularly challenging keeping up with preview features that might change in only 6 months. However, since we have three releases a year, we can usually support the key features from each release of Java before the actual release, thanks to Early Access versions of both the JDK and IntelliJ IDEA, which help us to work iteratively with

feedback from developers and also allows us to feedback to the JDK developers.

*IntelliJ is the leading IDE for Java development. How do you look at a free competitor like Visual Studio Code? What are the main reasons to work with IntelliJ?*

I personally don't like to think of VSCode as a competitor. As developers it's all about using the right tools for the right job - after all, a carpenter has more than one screwdriver (one would hope!). IntelliJ IDEA is great as a full IDE (Integrated Development Environment) - it provides not only code highlighting, code completion, suggestions, and code generation, of course, but also: automatic refactoring for pretty much everything in Martin Fowler's famous Refactoring book (for more language than just Java); full support for Version Control Systems like Git, so I no longer need the command line or a separate tool to do everything I want; complete tooling for automated testing, debugging, etc. For me, personally, IntelliJ IDEA gives me all the functionality I need as a developer so I don't need to context switch into any other tool or environment. Do I use it to make a change to a README file on a project when I don't want to download the whole codebase? No, I use a text editor for that.

*What are the most important recent and upcoming changes in the "Java World" according to you?*

I think the key thing is to have a vague idea of what's happening in the Java world, rather than a deep knowledge of one (or more) area(s). I expect people to have a competent idea whatever it is they need for their day job, of course! And that will vary from job to job. The key to staying up to date is not to know everything about everything, that's not possible, but to have a "TL;DR" idea of what's going on. In the Java world, I would say what you need to be aware of is:

- 6 monthly release cadence since Java 9.
- Difference between "feature releases" (9, 10, 12, 13, 14...) and "Long Term Support releases" (8, 11).
- What's a preview feature (also why you should try it and why you should not use it in production code!)
- If you're a team lead/decision-maker you should probably understand the licensing/support changes since Java 11. Otherwise, you might end up with a big bill to pay Oracle, and no-one wants that!

If you're interested in Java features since Java 8, then you probably want to look at:

- The updates to Streams and Optional since Java 8
- “var” (specifically when not to use it!)
- Switch expressions.

In terms of up-and-coming things, I'm interested in Records and Pattern Matching. Also, I think it's worth taking a look at Kotlin, it's a very pleasing language to code in.

*Why is now the perfect time to learn Java?*

Java is not going anywhere any time fast! And even if no-one developed anything new in Java from now on (which is certainly not the case!), there are so many Java applications and so much Java code out there that there's plenty to keep us in well-paid employment for as long as we want to be coders! Maybe that's not the most exciting reason to learn a language but it's certainly the most pragmatic. But on top of that, the six-monthly release cadence is really helping to move the language forward faster, and with feedback from real developers like us, so it's a really great time to be involved with the language and help to drive it in a direction that works for us.

*Which DIY-programming-electronics-project are you working on, or is on your “if I ever have time” list?*

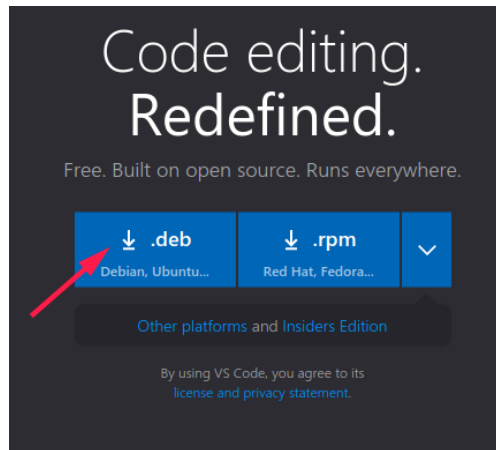
Er none! I've traditionally always used my so-called free time for community and advocacy, so I tended to go to user groups, write blog posts, and work on my presentation skills when I wasn't doing my 9-5 job. Now, I don't even really do that because the reality of having small children is I just don't have the time or energy for anything other than trying to keep them alive and possibly entertained. I have to scratch my development/technology itches during my day job. Luckily, as a developer advocate, I have a reasonable amount of flexibility in what I spend my time on, and I can use work time to learn new stuff, especially if it's relevant to the Java/JVM community.

<sup>a</sup>[https://twitter.com/trisha\\_gee](https://twitter.com/trisha_gee)

## Visual Studio Code (VSC)

Another great IDE is Visual Studio Code developed by Microsoft. The [source code is available on GitHub](#)<sup>26</sup>. It started as an editor for web applications, but thanks to its flexible approach with plugins, it gained a lot of attraction and almost all languages can now be written in VSC.

For Windows and other 64bit-systems, you can get it from [the official site](#)<sup>27</sup>:



Install VSC for Linux

As described further, you can use VSC on PC but work with files on the Pi, but you can also install VSC on the Pi itself if you want to program on it. As the official version is only for 64-bit PCs and Raspbian only has a 32-bit operating system version - although the Pi 4 has a 64-bit processor - this will not work. Luckily we can still find an earlier version of VSC which is 32-bit and compiled for the Pi! You can install in three easy steps:

- 1 `$ cd /home/pi/`
- 2 `$ wget https://github.com/stevedesmond-ca/vscode-arm/releases/download/1.28.2/vscode-1.28.2.deb`
- 3 `vscode-1.28.2.deb`
- 4 `$ sudo apt install ./vscode-1.28.2.deb`

<sup>26</sup><https://github.com/Microsoft/vscode>

<sup>27</sup><https://code.visualstudio.com/Download>



Java projects in VSC can be easily started with the popup “Run|Debug” buttons. They appear automatically above the main-method. If you use VSC on the Pi you need to keep in mind it will take some time before this is shown after loading a project!

```

14 public class Launcher {
15     Run | Debug
16     public static void main(String[] args) {
17         Main.main(args);
18     }
19 }

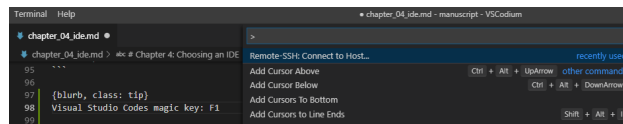
```

Run Java application from within VSC



The magic key of Visual Studio Code: F1

This shortcut gives you the “Show All Commands” popup where you can type in the function you are looking for. It will also show you which shortcuts you can use to access them directly.



Show All Commands in VSC

## VSCodium the free non-tracking alternative to VSC

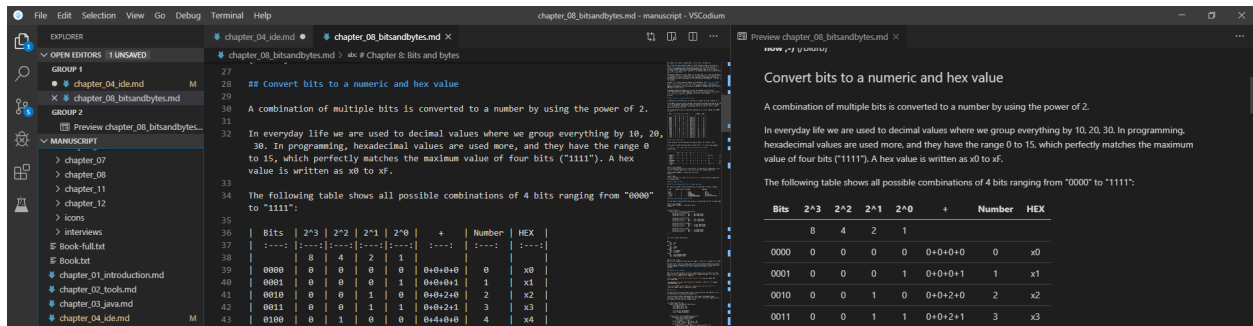
The sources of VSC are available on GitHub as open-source, but the packaged application Visual Studio Code is licensed under a non-free license and contains telemetry and tracking components of Microsoft. If you care about your privacy (you should, see “Just a thought: Switching Social”) and don’t want to be tracked, you can download the sources and compile the program yourself, or take the easy path and [download VSCodium](#)<sup>28</sup>. It’s a fully functional VSC-version that works the same way and can be extended with the same plugins. It is shared under the MIT license and telemetry is disabled. The only drawback I found until now is the missing support for Remote Development as described further.

As you may have seen in some screenshots... this book is even written with VSCodium! Because I worked with [LeanPub](#)<sup>29</sup> book generation tools, this has been written in the Markua-format (an extended README-format) and an IDE is perfect to write and visualize at the same time.

<sup>28</sup><https://vscodium.com/>

<sup>29</sup><https://leanpub.com/>

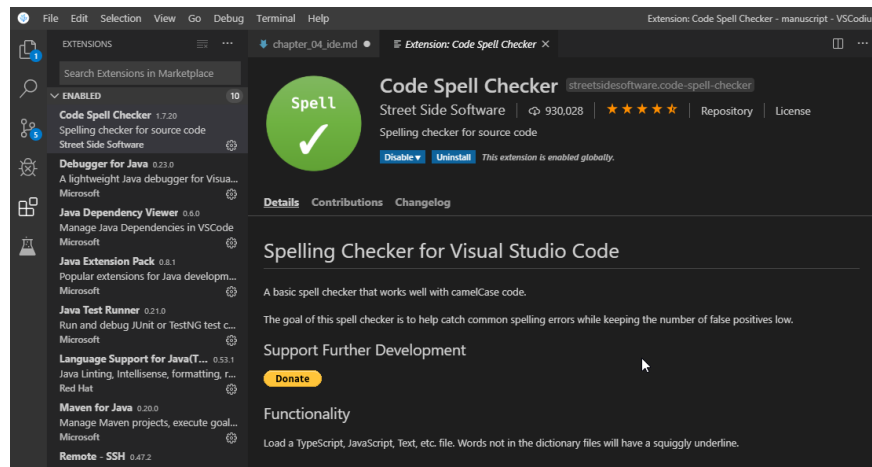




Editing a README file in VSCode

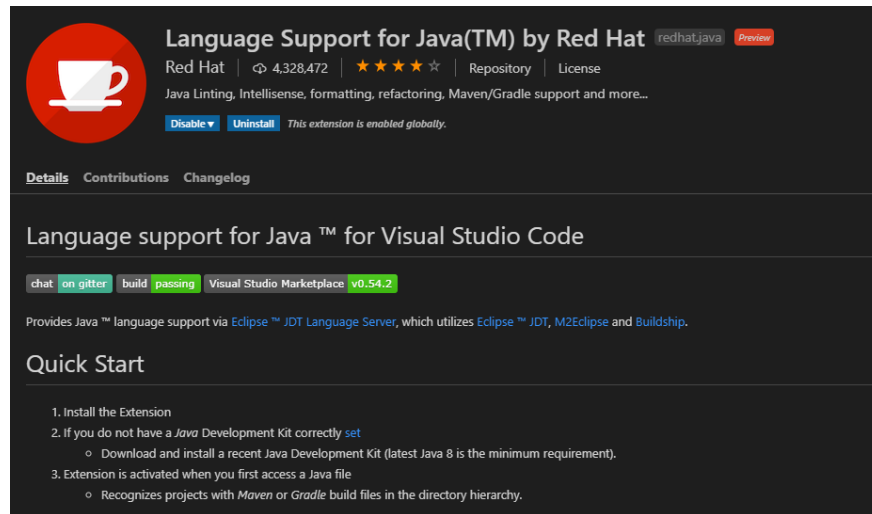
## Java development with Visual Studio Code

VSC and VSCode can be extended with euh... "extensions" to support specific languages or additional functions. As I'm not a native English speaker I started with installing a spell checker ;-)



Plugins section of VSC

Click on the "Extensions" button in the left bar. With the search box on top, you can find extensions in the Marketplace. For Java development, you will need some additional ones. Not all are required but as you can see I installed a bunch of them. "Language Support for Java" by RedHat is the most important one.



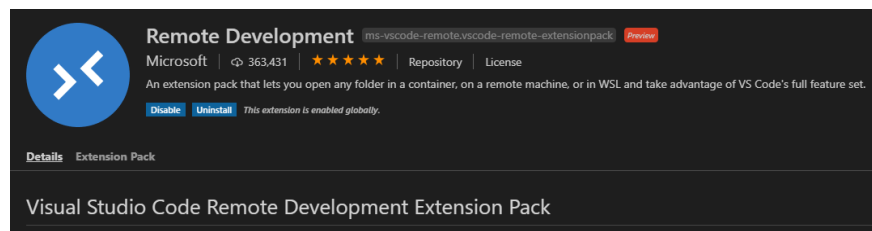
### Plugin Language Support for Java

Take a look at this [Java info page on the VSC website](#)<sup>30</sup> for more detailed info on this topic.

## Using Visual Studio Code on the PC with code on the Pi

The main advantage of Visual Studio Code - and why I use it a lot - is the remote development plugin. This allows you to work on your PC, but with files located on a Pi. That means you can take full advantage of the power of your PC but still work on the Pi. Although with the Pi 4 the difference between Pi and PC got a lot smaller!

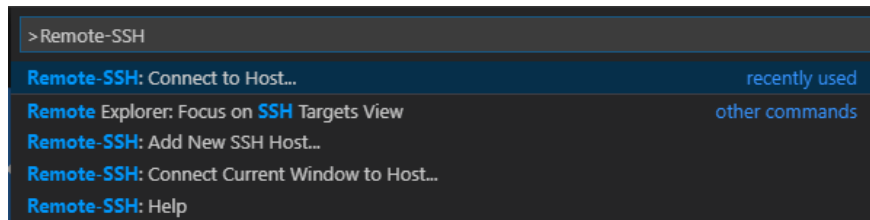
As mentioned before this plugin doesn't work with VSCodeium, so you need to use Visual Studio Code, the "official" Microsoft version. Go to the "extensions" menu and search for "remote development" and click install.



### Remote Development plugin in VSC

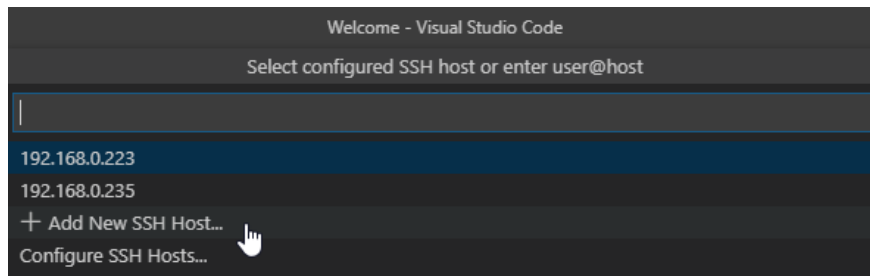
When this is done, hit F1 for the "Search Everywhere" popup on top and search for "Remote-SSH" and hit enter.

<sup>30</sup><https://code.visualstudio.com/docs/languages/java>



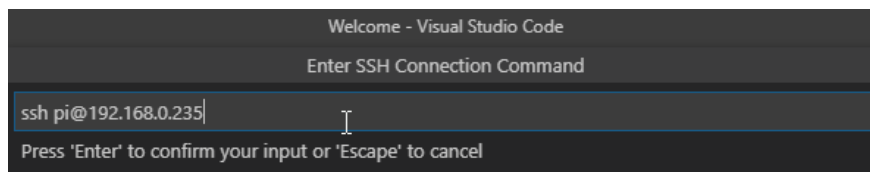
Search for Remote-SSH

In my case, I already have two Pi's addresses configured, but to add one click on "Add New SSH Host..."



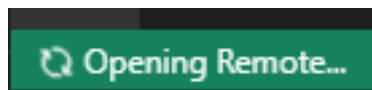
Add new SSH host

You will need to know the IP address if your Pi and SSH needs to be enabled (see Chapter 2: Tools). By default you can connect with username "pi" and password "raspberr", but don't forget to change that if you care about security (and we all do... but still forget to change default passwords...).



Adding an SSH connection

When you have entered the ssh login command with the IP address of your Pi, in my case "ssh pi@192.168.0.235", you will be asked for the password. It will take some time for VSC to install the necessary files on the board so it can be used for remote development. You will see on the lower left of the IDE that it's busy...



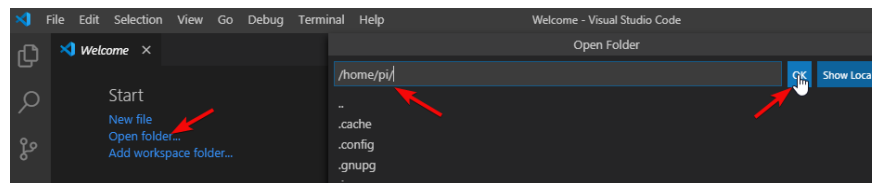
Opening Remote is busy

... or ready.



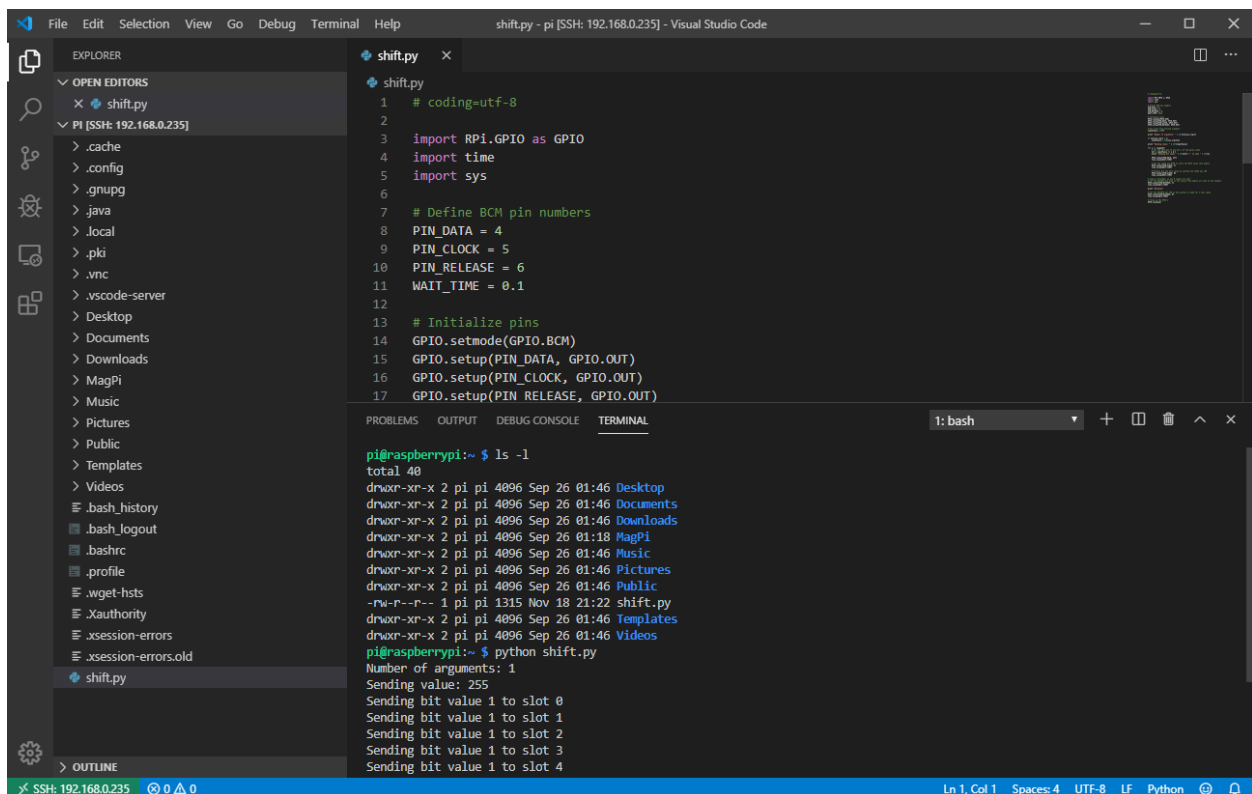
Connection established

Now you can open a file or directory from the Pi on your PC in VSC just as you would be working on the Pi itself.



Opening a project directory

Running the code can be done the same way in the terminal of VSC and will be executed on the board itself. In this screenshot, you see the Python code from “Chapter 8: Bits and Bytes”.



Running Python script with Remote Development in VSC

Indeed Python! As mention before this is not an anti-Python book ;-)

## Interview with Xiaokai He



Xiaokai He, Senior Program Manager, Java on Visual Studio Code and Azure, [@XiaokaiHe](https://twitter.com/XiaokaiHe)<sup>a</sup>

*Visual Studio Code is a perfect tool to develop and build Java applications with additional plugins. Can we expect even more Java-specific new features?*

We started the Java support work for VS Code in 2016 along with developers from other companies such as Red Hat and IBM using the then newly release language server protocol. In the following years, we see more and more developers using VS Code for their Java development, and we decided to double down our collaboration with our core partners. We're surely committed to delivering more Java specific new features to VS Code in the next few years. As a lightweight editor, our goal is to make VS Code great for lightweight Java workloads, from student's first assignment to microservices in today's cloud-native paradigm. Given its wide usage and active community, we believe Java will continue to attract new developers to learn and keep evolving in the foreseeable future.

*Which Microsoft and/or Azure technology do you think would be a perfect match with IoT, Java and the Raspberry Pi.*

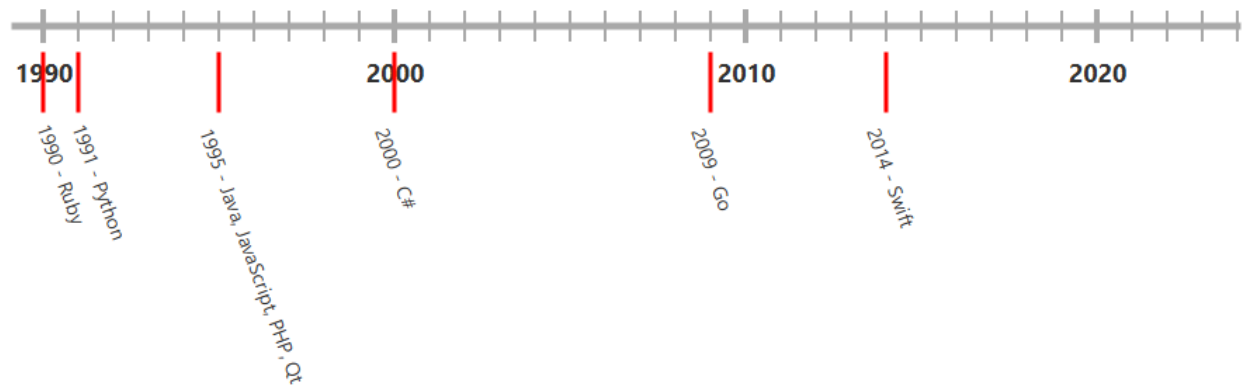
Talking about IoT, we also have VS Code extensions for Azure IoT Hub and IoT Edge. You can use them to deploy your application to a device and connect to other Azure services.

---

<sup>a</sup><https://twitter.com/XiaokaiHe>

# Chapter 4: About Java

In many (on-line) discussions, Java is considered to be an old (outdated?!) language compared to other popular ones like PHP and JavaScript. But if we look at their first appearance dates on Wikipedia we discover this...



Timeline of some popular programming languages

Yes, indeed, they have the same age! :-)

Java was initially created for digital television, but it was too advanced and evolved to solve big problems on huge servers on enterprise levels. And indeed a lot of computer power and memory was needed for such applications. But a lot has changed, and you can now use and run Java on embedded platforms and small PCs like the Raspberry Pi.

More than ever, Java is a “write once, run everywhere” language, even for smartphone apps!



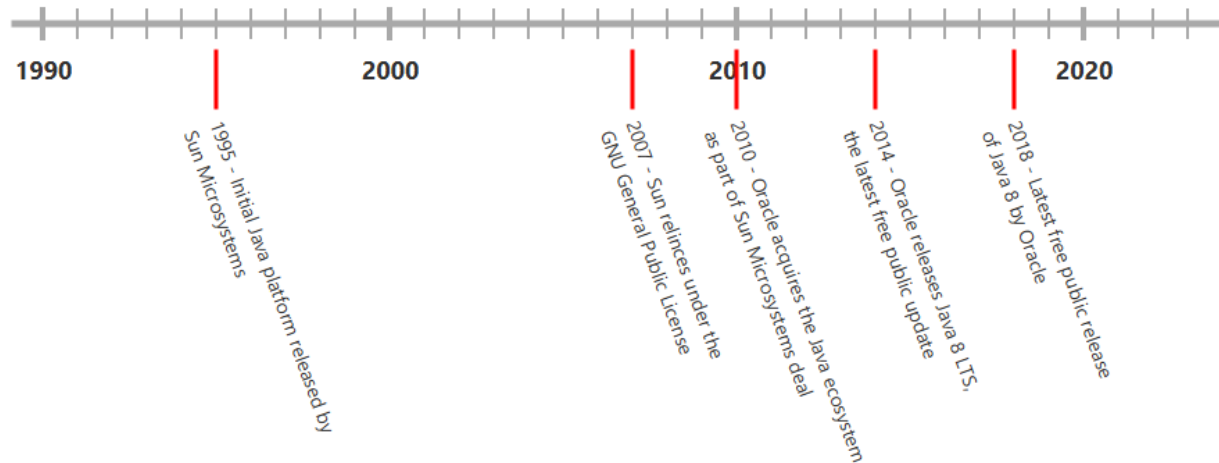
When working on this chapter, I was wondering if I should create a timeline in Photoshop or Illustrator. But being a developer, it needed to be done with code of course! And it took me only two hours to make something which I can easily change to different data sets. Most of that time was needed to research how to draw with “javafx.scene.shape.Line” as I hadn’t done this before.

The finished application is part of the sources:  
chapter\_04\_Java > java-timeline

Take a look at the sources in DataSets.java and you will see how easy it was to change the markers on the timeline to create the different images for this chapter, for example for the above image:

```
1 public static final Map<Integer, String> LANGUAGE_BIRTHDAYS = Map.ofEntries(  
2     entry(1990, "Ruby"),  
3     entry(1991, "Python"),  
4     entry(1995, "Java, JavaScript, PHP, Qt"),  
5     entry(2000, "C#"),  
6     entry(2014, "Swift")  
7 );
```

## History

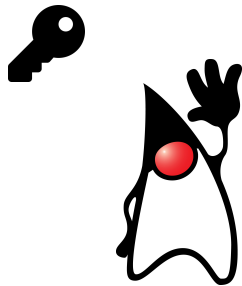


Timeline of Java

The development of the Java language started in 1991 with the first public release by Sun Microsystem in 1995. From 2007 on, the major part of the Java core source code has been available as open-source.

When Oracle acquired Sun Microsystems in 2009-2010 they also became the owner of Java.

Google used Java as the main programming language for Android applications, as Android systems are based on Linux.



Duke

Say hello to Duke, designed by artist Joe Palrang while he was Art Director at Sun. It was originally a “software agent” designed to perform tasks for the user, just like Clippy the animated paper clip from Microsoft.

Duke is now the official mascot for the Java language even appearing in comics, on coffee cups, t-shirts, etc. The graphic specification of Duke has been [open-sourced](https://hg.openjdk.java.net/duke/duke/)<sup>31</sup> by Sun on November 13, 2016.

<sup>31</sup><https://hg.openjdk.java.net/duke/duke/>



## Java files versus byte code

The Java platform has different ‘‘layers’’. The written code is only one of them. This is done in Java files, for example, ‘‘HelloWorldExample.java’’ which could look like this:

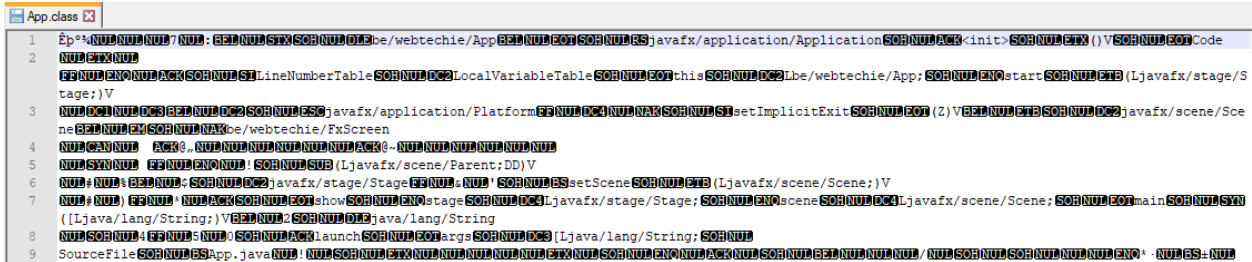
```
1 public class HelloWorldExample {
2     public static void main(String args[]) {
3         System.out.println("Hello World!");
4     }
5 }
```

This code would result in the following output in the console:

```
1 Hello World!
```

This .java-file is the file you will be working in to write code (for example on a Windows PC in IntelliJ IDEA). But to run it on different platforms (for example on the Raspberry Pi), it needs to be converted to byte code. The byte code file (with the extension .class) is the result of compiling your source code file(s) to something which can be used by the Java runtime on each platform.

When opening such a class-file in e.g. Notepad++ you get this:



A class file is not human-readable as it is compiled to be run by a JVM

It’s no longer readable for us but optimized for the runtime environment.

## **JVM versus JRE versus JDK**

Java code is platform-independent, but to execute it, we need specific versions for every platform (Windows, Linux, iOS...) and processor (Intel, ARM, 32bit, 64bit...).

### **JVM = Java Virtual Machine**

The JVM is not a physical machine, but a specification to provide a runtime environment to execute Java byte code on each platform.

This enables you to develop your code on one machine and compile it to bytecode, but run it on many different types of computers without the need to change anything in your code!

### **JRE = Java Runtime Environment**

The JRE contains everything needed to run a (compiled) Java program. It includes the implementation of the JVM and a collection of libraries, tools, and software that the JVM uses at runtime to run Java code and applications.

You cannot use the JRE to create new programs.

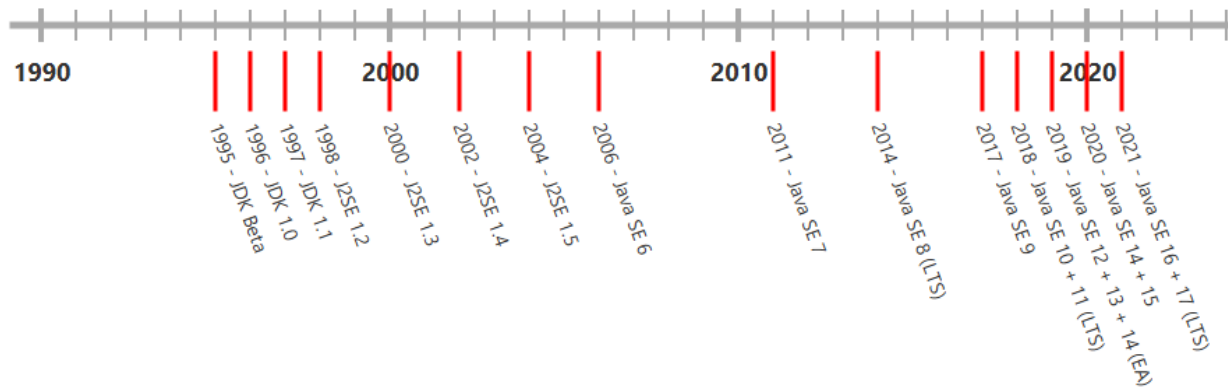
### **JDK = Java Development Kit**

The JDK is a software package you need on your PC to develop Java applications. The JDK includes the JVM, JRE, an interpreter to load java applications, a compiler to convert java-files to byte code and many other tools.

If you install a JDK you have everything to build and run Java applications. And that's what we will do on our Pi.

## Version history

The Java-architects always made sure improvements and new features in Java didn't break existing applications. Because of this, you can still run old Java applications on newer versions with none or minimal changes. This is one of the big differences with Python where applications made for Python 2 are incompatible with the completely refactored version 3. On the other hand, the version number and naming of the Java releases took some strange turns as you can see in this timeline:



Timeline of Java

Abbreviations used:

- J2SE: Java Platform version 2, Standard Edition
- SE: Standard Edition
- LTS: Long Time Supported
- RC: Release Candidate
- EA: Early Access

Between the versions 5 till 9, new versions of Java were released with big intervals. With current fast changes with cloud platforms, big data, artificial intelligence, etc. a faster release cycle was needed to be able to align new Java features with new demands.

This led to a big change in the whole Java ecosystem in recent years. A lot of companies contribute to the OpenJDK project pushing it forward and releases are now scheduled every six months! All new features which are finished at the release date, get included, others just have to wait till the next one. This is a completely different approach compared to the earlier days where a release was only done when all scheduled new features were finished.

If you keep an eye on the changes in new releases, you can decide for yourself if you want to update your development and runtime JDK to the latest one, or just stick to the one you are using if you don't need one of the new features. But if you care about security and the best performance, following the release cycle will offer you the best results as Java gets improved continuously!

## JDK providers

All the sources of the Java Development Kit itself are open source and can be found on [GitHub](#) > [OpenJDK](#)<sup>32</sup>. So if you would like, you can build your own JDK from these sources. Luckily a lot of others have done this already and provide up-to-date compiled versions.

A full list is available on the [jchoice website](#)<sup>33</sup>. Let's take a quick look at some of them.

### Oracle

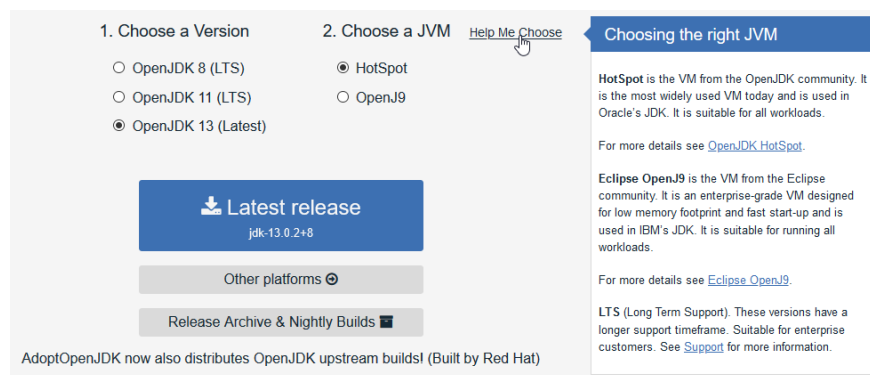
This is the “mother” of all JDK's, but the most expensive one for commercial use as you need a contract with Oracle for non-personal and non-development use.

On the other hand, Oracle is still the main contributor to OpenJDK. There is a nice overview of the differences between Oracle JDK and OpenJDK on this [post on the Baeldung site](#)<sup>34</sup>.

### AdoptOpenJDK

If you want to have the most up-to-date JDK for your development PC, [this is the place to be](#)<sup>35</sup>. All the binaries and scripts provided here are free.

On the AdoptOpenJDK website, you will find multiple versions for different platforms, all ready to install with a few clicks.



AdoptOpenJDK - Help Me Choose

### Azul Zing and Zulu

[Azul](#)<sup>36</sup> provides commercial support with Zing if you need a JVM to run a big application in a cluster to handle big loads of data. They also have Zulu which is a build of OpenJDK and is available for

<sup>32</sup><https://github.com/openjdk/jdk>

<sup>33</sup><http://www.jchoice.eu/>

<sup>34</sup><https://www.baeldung.com/oracle-jdk-vs-openjdk>

<sup>35</sup><https://adoptopenjdk.net/>

<sup>36</sup><https://www.azul.com/>

different platforms. For embedded systems, they can provide “tailor-made” JDK versions tested for that specific platform.

Zulu is used on Microsoft Azure to run Java applications in the cloud.

## BellSoft Liberica

Similar to the other ones, [BellSoft](https://bell-sw.com/)<sup>37</sup> compiles from the OpenJDK sources, but they include JavaFX into the JDK in specific platform versions. JavaFX was once a part of Java but has become a separate project.

Above all, BellSoft also provides a version specifically made for the Raspberry Pi. Further in this book, we will use JavaFX to build GUIs (Graphical User Interface) for the Pi, so this is the SDK we will be using on the Pi in this book.



### Liberica JDK version 12.0.2

[Release Notes](#)

Liberica is a 100% open-source Java 12 implementation. It is built from OpenJDK which BellSoft contributes to, is thoroughly tested and passed the JCK provided under the license from OpenJDK. The versions of Liberica for Windows x86\_64, Windows x86, Mac x86\_64, Linux x86\_64 and ARMv7 also contain JavaFX 12. The version for Linux ARMv7 contains Device IO API as additional module and JavaFX with hardware-accelerated EGL support. We currently provide binaries suitable for different hardware and OS combinations:

- **Windows x86\_64** (64-bit version for Microsoft Windows)
- **Windows x86** (32-bit version for Microsoft Windows)
- **macOS x86\_64** (64-bit version for Apple macOS)
- **Linux x86\_64** (64-bit version for Linux servers and desktops)
- **Linux ARMv8** (64-bit version for ARMv8 servers and embedded)
- **Linux ARMv7 HardFloat** (32-bit version for Raspbian on Raspberry Pi 2 and Raspberry Pi 3)
- **Solaris SPARC** (SPARCv9 version for Solaris)
- **Solaris x86\_64** (x86\_64 version for Solaris)

### JDK selection on the BellSoft website

<sup>37</sup><https://bell-sw.com/>

## Interview with Alexander Belokrylov



Alexander Belokrylov, [@gigabel](https://twitter.com/gigabel)<sup>a</sup>, CEO of BellSoft, Java enthusiast

*BellSoft makes a living by providing Java services, so why did you decide to distribute Liberica JDK for free?*

Java by its nature should be **Free and Open Source**. BellSoft is the company committed to freedom by releasing Liberica JDK. Our goal is to create the most useful OpenJDK binary distribution which makes the life of developers easier and we do it the open source way with full transparency. On top of that, we provide commercial support to companies who want to have additional assurance and a top OpenJDK.

*You are the only JDK provider who has a Raspberry Pi version and integrates JavaFX, why did you go this extra mile?*

We are passionate about making Liberica JDK fully functional on Raspberry Pi. Our company has started this journey with the first release of Liberica JDK 9 for the Raspberry Pi platform. We continue to **actively support the ARM32 port within OpenJDK to bring Raspberry Pi developers all modern JDK features!**

*In a blog post you announced support for video in JavaFX on the Pi. Can we expect even more of these additions from BellSoft?*

There was a high demand from the community and commercial companies for video in JavaFX on Raspberry Pi. We are carefully listening to the voice of the community. **As soon as we see Liberica JDK lacks some capabilities required by our users we act accordingly.**

*Which DIY-programming-electronics-project are you working on, or is on your “if I ever have time” list?*

On our blog, we already talked about the support of Liberica JDK for the Nvidia Jetson Nano<sup>b</sup>. We implemented a demo application that provides a data-driven evaluation of the booth effectiveness on an exhibition floor. The setup was tested at Code One conference in San Francisco and brought lots of attention from participants.

Technically we have a Jetson Nano board with a 4K camera that captures video and uses DeepStream library for real-time video analytics & people identification. A Java application is used to perform data analysis, and the OpenJFX application is visualizing the statistics. The next step for us is to use Project Panama<sup>c</sup> on Nvidia for interoperability between a Java application and Cuda native app<sup>d</sup>.

<sup>a</sup><https://twitter.com/gigabel>

<sup>b</sup>NVIDIA® Jetson™ System-On-Module systems provide the performance and power efficiency to run autonomous machines software, faster and with less power.

<sup>c</sup>Interconnecting JVM and native code by improving and enriching the connections between the Java™ virtual machine and well-defined but “foreign” (non-Java) APIs, including many interfaces commonly used by C programmers.

<sup>d</sup>CUDA is a parallel computing platform and programming model invented by NVIDIA.

# Installing the Java JDK

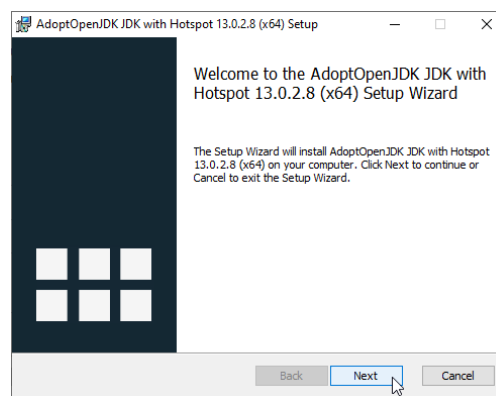
## Install Java JDK on a Windows PC

On Windows I prefer to work with the SDK provided by [AdoptOpenJDK](https://adoptopenjdk.net/)<sup>38</sup>. Just select the most recent version, download, run and done!



Selection the version on the AdoptOpenJDK website

Run the downloaded file.

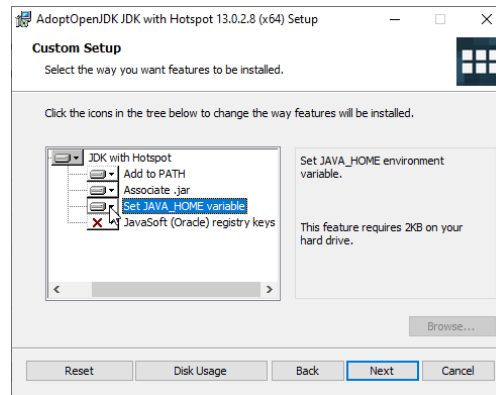


Installer of AdoptOpenJDK on Windows

In one of the next steps, make sure to select the option “Set JAVA\_HOME variable”. By doing so, we will be able to start Java easily in the terminal.

---

<sup>38</sup><https://adoptopenjdk.net/>



Select JAVA\_HOME option

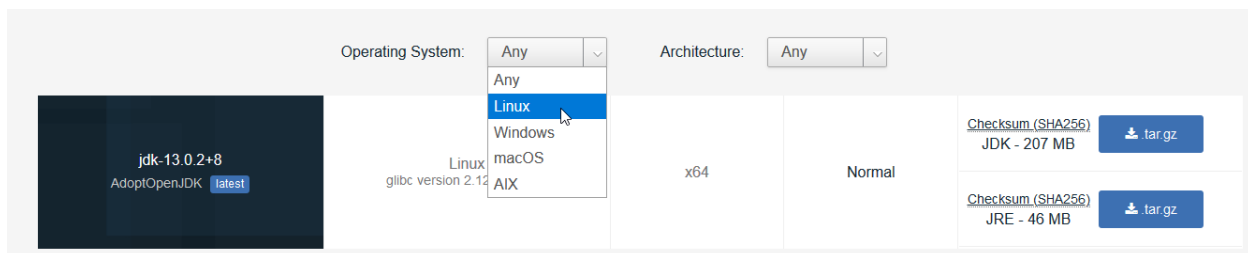
Once the installation is completed, you can open a “Command Prompt” via the start menu. Type “java –version” and you will get the version.

```
C:\>java --version
openjdk 13.0.2 2020-01-14
OpenJDK Runtime Environment AdoptOpenJDK (build 13.0.2+8)
OpenJDK 64-Bit Server VM AdoptOpenJDK (build 13.0.2+8, mixed mode, sharing)
```

Check installed Java version on Windows

## Install Java JDK on a Linux PC with SDKMAN

AdoptOpenJDK also provides installers for Linux and other platforms.



Platform selection on the AdoptOpenJDK website

But you can also use [SDKMAN](https://sdkman.io/)<sup>39</sup> to change between different SDK versions easily. According to [this blog post](https://ngeor.com/2019/12/07/sdkman-windows.html)<sup>40</sup>, it also works on Windows, but I didn’t try it out myself. The tool also provides a list overview of the available SDKs through SDKMAN:

<sup>39</sup><https://sdkman.io/>

<sup>40</sup><https://ngeor.com/2019/12/07/sdkman-windows.html>



```

1 $ sdk list java
2 =====
3 Available Java Versions
4 =====
5 Vendor      | Use | Version      | Dist  | Status  | Identifier
6 -----
7 AdoptOpenJDK |     | 13.0.1.j9    | adpt  |         | 13.0.1.j9-adpt
8             |     | 13.0.1.hs    | adpt  |         | 13.0.1.hs-adpt
9             |     | 12.0.2.j9    | adpt  |         | 12.0.2.j9-adpt
10            |     | 12.0.2.hs    | adpt  |         | 12.0.2.hs-adpt
11            |     | 11.0.5.j9    | adpt  |         | 11.0.5.j9-adpt
12            |     | 11.0.5.hs    | adpt  |         | 11.0.5.hs-adpt
13            |     | 8.0.232.j9   | adpt  |         | 8.0.232.j9-adpt
14            |     | 8.0.232.hs   | adpt  |         | 8.0.232.hs-adpt
15 Amazon      |     | 11.0.5       | amzn  |         | 11.0.5-amzn
16 ...
17            |     | 6.0.119     | zulu  |         | 6.0.119-zulu
18 Azul ZuluFX  |     | 11.0.2       | zulufx |         | 11.0.2-zulufx
19            |     | 8.0.202     | zulufx |         | 8.0.202-zulufx
20 BellSoft    |     | 13.0.1       | librca | installed | 13.0.1-librca
21            |     | 12.0.2       | librca |         | 12.0.2-librca
22            |     | 11.0.5       | librca |         | 11.0.5-librca
23 ...
24 SAP         |     | 12.0.2       | sapmchn |         | 12.0.2-sapmchn
25            |     | 11.0.4       | sapmchn |         | 11.0.4-sapmchn
26 =====
27 Use the Identifier for installation:
28     $ sdk install java 11.0.3.hs-adpt
29 =====

```

And within the shell you can choose to use a different SDK version, for instance, if you want to test if your application can run with another one:

```

1 $ sdk use java 11.0.5-open
2 Using java version 11.0.5-open in this shell.

```

## Install Java JDK on a Raspberry Pi



The Java versions used here, can only be used on the Pi's with an ARMv7 or ARMv8 processor. The oldest Pi's have an ARMv6 that has a completely different architecture which is not supported by these Java JDKs.

On the [Wikipedia site you can find a nice overview<sup>41</sup>](#) of the different Raspberry Pi models and which instruction set they use. The ones with ARMv6 are not a perfect fit for our Java experiments.

### Raspbian (Buster) has Java 11 pre-installed

In [the release notes of Raspbian<sup>42</sup>](#) you can see that the version of 2019-06-20 includes OpenJDK Java 11.

- 1 2019-06-20:
- 2 \* Based on Debian Buster
- 3 \* Oracle Java 7 and 8 replaced with OpenJDK 11

So we are good to go! A recent version of Java is ready to use! But... as we will be using JavaFX later in this book, we will replace the pre-installed JDK with Liberica JDK provided by BellSoft, which includes JavaFX.

### Downloading and testing BellSoft Liberica JDK

Installing a new JDK is easy once you know where to get it. These are the steps needed to install version 13 of Liberica JDK.

First, we will check the current version so we can validate it after the installation. With Raspbian Buster this will be the result:

- ```
1 $ java -version
2 openjdk version "11.0.3" 2019-04-16
3 OpenJDK Runtime Environment (build 11.0.3+7-post-Raspbian-5)
4 OpenJDK Server VM (build 11.0.3+7-post-Raspbian-5, mixed mode)
```

Now we will download the version we want, and replace the existing one. Check the Liberica JDK pages on the [BellSoft website<sup>43</sup>](#) and copy the link to the deb-file of the version for the Pi you want to use. This is the example for the JDK 13 version:

<sup>41</sup>[https://en.wikipedia.org/wiki/Raspberry\\_Pi#Specifications](https://en.wikipedia.org/wiki/Raspberry_Pi#Specifications)

<sup>42</sup>[http://downloads.raspberrypi.org/raspbian/release\\_notes.txt](http://downloads.raspberrypi.org/raspbian/release_notes.txt)

<sup>43</sup><https://bell-sw.com>

```
1 $ cd /home/pi
2 $ wget https://download.bell-sw.com/java/13/bellsoft-jdk13-linux-arm32-vfp-hflt.deb
3 $ sudo apt-get install ./bellsoft-jdk13-linux-arm32-vfp-hflt.deb
4 $ sudo update-alternatives --config javac
5 $ sudo update-alternatives --config java
```

When this is done, we can check the version again and it should look like this:

```
1 $ java --version
2 openjdk version "13-BellSoft" 2019-09-17
3 OpenJDK Runtime Environment (build 13-BellSoft+33)
4 OpenJDK Server VM (build 13-BellSoft+33, mixed mode)
```



chapter\_04\_Java > scripts > a file is provided per Liberica version

If you downloaded the sources to a Pi in the home directory, this is what you need to do in the terminal:

```
1 $ cd /home/pi/JavaOnRaspberryPi/chapter_04_Java/scripts
2 $ sh install_liberica_13.sh
```

## Some of the changes between Java versions

As the Java maintainers make sure the core functionality doesn't change fundamentally, you can use the same code with different Java versions. But with each newer version, features are added, or existing ones get improved and extended.

For business applications, usually, only LTS (= Long Time Supported) versions are used, like 8 and 11. But it's always a good idea to keep track of the changes between versions, so you have an idea which new features become available.

### Changes between Java 8 and 11

As Java 8 was the latest free LTS version released by Oracle, it took some time before a next LTS became available, which was the 11 version.

In-between those LTS-versions, Java 9 introduced modules that enable us to build a smaller distributable and better define which code is available for other applications. [Check this page<sup>44</sup>](#) for full info about Java Modules benefits and basics.

“jshell” was also added in version 9 which enables to quickly test Java code. To start, just type in jshell in the terminal:

```
1 $ jshell
2 | Welcome to JShell -- Version 11.0.4
3 | For an introduction type: /help intro
4
5 jshell> /help intro
6 |
7 |                               intro
8 |                               =====
9 |
10 | The jshell tool allows you to execute Java code, getting immediate results.
11 | You can enter a Java definition (variable, method, class, etc), like: int x = 8
12 | or a Java expression, like: x + x
13 | or a Java statement or import.
14 | These little chunks of Java code are called 'snippets'.
15 |
16 | There are also the jshell tool commands that allow you to understand and
17 | control what you are doing, like: /list
18 |
19 | For a list of commands: /help
20
```

---

<sup>44</sup><http://tutorials.jenkov.com/java/modules.html>

```
21 jshell> var txt = "Hello World!"
22 txt ==> "Hello World!"
23
24 jshell> txt
25 txt ==> "Hello World!"
26
27 jshell> txt + (5*4)
28 $3 ==> "Hello World!20"
29
30 jshell> txt.substring(2,5)
31 $4 ==> "llo"
32
33 jshell> /exit
34 | Goodbye
```

In version 11 the JavaFX library was removed from the JDK and moved to a separate project (See Chapter 7).

On [this website made by Marc R. Hoffmann<sup>45</sup>](#) you can compare the differences between the Java versions in a very detailed way.

## What's next after Java 11?

12? Yes indeed! That was easy ;-)

But above all, because of the faster release cycle and the big community pushing Java “fast-forward”, new features become available much faster and are a real answer to the most requested functionalities.

Let's take a look at some of the main changes in the new versions by listing some of the JEP's that were included. A JEP is a “JDK Enhancement Proposal” written by one or more persons with a request to add something to Java. If the proposal is clear enough, has enough support and gets implemented, it can become part of a new version. I only describe a few that I find very attractive as they will lead to cleaner code.

Be careful with “Preview” features as they are still in development and not part of the specification, so they must be extra enabled when compiling (“javac -enable-preview”) and running (“java -enable-preview”).

As new JDK versions are now released every six months, what's described here, are just a few features that were introduced as preview-features and will be or are already fully integrated into a later version.

---

<sup>45</sup><https://javaalmanac.io/>

## Switch Expressions

### Java 12 first improvements

#### [JEP 325: Switch Expressions \(Preview\)](#)<sup>46</sup>

This is a first step to reduce the code you need to write for a switch statement. The example given in the JEP is to be able to change this:

```
1  switch (day) {
2      case MONDAY:
3      case FRIDAY:
4      case SUNDAY:
5          System.out.println(6);
6          break;
7      case TUESDAY:
8          System.out.println(7);
9          break;
10     case THURSDAY:
11     case SATURDAY:
12         System.out.println(8);
13         break;
14     case WEDNESDAY:
15         System.out.println(9);
16         break;
17 }
```

into a shorter code block like this:

```
1  switch (day) {
2      case MONDAY, FRIDAY, SUNDAY -> System.out.println(6);
3      case TUESDAY                 -> System.out.println(7);
4      case THURSDAY, SATURDAY     -> System.out.println(8);
5      case WEDNESDAY              -> System.out.println(9);
6  }
```

### Further extended in Java 13

#### [JEP 354: Switch Expressions \(Preview\)](#)<sup>47</sup>

The improved switch expression in Java 12 were further extended so you can also use them to define a value:

---

<sup>46</sup><http://openjdk.java.net/jeps/325>

<sup>47</sup><http://openjdk.java.net/jeps/354>

```

1  int numLetters = switch (day) {
2      case MONDAY, FRIDAY, SUNDAY -> 6;
3      case TUESDAY                 -> 7;
4      case THURSDAY, SATURDAY     -> 8;
5      case WEDNESDAY              -> 9;
6  };

```

## Text blocks introduced in Java 13

[JEP 355: Text Blocks \(Preview\)](#)<sup>48</sup>

This allows you to define strings in multiple lines without additional “” and ‘+’.

Before:

```

1  String htmlBlock = "<html>" +
2      "  <body>" +
3      "    <p>Welcome on my site</p>" +
4      "  </body>" +
5      "</html>";

```

Now you can use triple “” to start and end a String definition, but these must be on separate lines. You can add tabs to structure your text. Here is an example:

```

1  String htmlBlock = """
2      <html>
3          <body>
4              <p>Welcome on my site</p>
5          </body>
6      </html>
7      """ ;

```

## Helpful NullPointerExceptions in Java 14

[JEP 358: Helpful NullPointerExceptions](#)<sup>49</sup>

Improve the usability of NullPointerExceptions generated by the JVM by describing precisely which variable was null.

NullPointerExceptions are hard to debug because they throw a message like this

---

<sup>48</sup><https://openjdk.java.net/jeps/355>

<sup>49</sup><https://openjdk.java.net/jeps/358>

```
1 Exception in thread "main" java.lang.NullPointerException at
2   Main.main(Unknown Source)
```

With this new functionality, the error will provide more info about the object which caused the error, so you can quickly spot the error in your code. For instance:

```
1 Exception in thread "main" java.lang.NullPointerException:
2   Cannot assign field "textValue" because "textBox" is null at
3   Main.main(Unknown Source)
```



## Java crash course

From Java version 11 on, you can run simple Java code without the need to compile it, the same way as you can run a Python or Bash script. And that’s exactly what we are going to use for this high-speed, trial-and-error crash course! If you are new to Java, follow these steps to get a first look at how to write Java code.

**For these examples, no IDE is required and it will work with any JDK from version 11 or higher.** So you can start with a complete new SD card with the latest Raspbian OS as it comes with AdoptOpenJDK 11 pre-installed! Of course, you can do the same on your PC with a simple text editor (check-out [Notepad++<sup>50</sup>](#)) and a JDK version 11 or higher.



All the examples used here can be found in  
chapter\_04\_Java > java-crash-course

## HelloWorld! Running a single-file Java-application

Traditionally we start our first experiment with a “Hello World” application.

```
1 $ cd /home/pi
2 $ nano HelloWorld.java
```

In this new file we add the following code:

```
1 public class HelloWorld {
2     public static void main (String[] args) {
3         String txt = "Hello World";
4         System.out.println(txt);
5     }
6 }
```

Java requires us to “package” our code in a class. This has the same name as the file, so in this case, we need to start with “public class HelloWorld”.

A Java application also needs an “entry point”, the main class which is started and can call all other methods, that’s the second line we need to add: “public static void main (String[] args)”.

Here we can add any code we want. In this case, we start with creating a String variable “txt” which holds “Hello World”. Notice each line needs to end with a “;”.

“System.out.println(txt)” is similar to “console.log” in JavaScript or “print” in Python.

---

<sup>50</sup><https://notepad-plus-plus.org/>

When you are doing this on the Pi, and are finished with typing the code in “nano”, click “CTRL+X” to exit, “Y” to save and “ENTER” to write to the file.

To execute this code, we need to start Java with the name of the file we just created, as an argument:

```
1 $ java HelloWorld.java
2 Hello World
```

And there it is... our first working Java code on a Pi! :-)



Although you can write all the code from these examples directly on the Pi or your PC with nano or any other text file editor, you can of course also do the same in an IDE.

When doing so, you will immediately notice the advantage of the code hinting.

For instance, if you want to get the first 4 characters of the txt-variable, you are presented with a list of possible methods you can use on a String.

Code hints for a String in IntelliJ IDEA

## Using the start-up arguments

Let’s go a little step further and use the start-up arguments assigned in “main (String[] args)”. We need to create a new Java file.

```
1 $ cd /home/pi
2 $ nano MainArguments.java
```

And this is the code:

```
1 public class MainArguments {
2     public static void main (String[] args) {
3         System.out.println("Number of arguments: " + args.length);
4
5         if (args.length > 0) {
6             System.out.println("First argument: " + args[0]);
7         }
8
9         for (int i = 0; i < args.length; i++) {
10            System.out.println("Argument " + (i + 1) + ": " + args[i]);
11        }
12    }
13 }
```

Now we can start the application and provide it any number of extra arguments, which are provided to the Java code as an array of Strings (= String[]). From this array, we can get the number of items (= args.length) and use a for-loop to cycle through all the arguments.

```
1 $ java MainArguments.java
2 Number of arguments: 0
3
4 $ java MainArguments.java "Hello World" "Bye"
5 Number of arguments: 2
6 First argument: Hello World
7 Argument 1: Hello World
8 Argument 2: Bye
```

## Working with numbers

Ok, up to the next one...

Java has different number types of which a few of them are further explained in “Chapter 8: Bits and Bytes”. In this example we use:

- Integer: rounded numbers only
- Float: can hold 8 decimal numbers
- Double: can hold 16 decimal numbers

Depending on the type of number, they use more or less memory. In most cases ints are used, but when you need to store e.g. prices, you would use a float. When assigning a value to either a float or a double, the number has to end with an “F” or “D” (see example). This helps Java to understand what you want to achieve.

We create both the example float and double with 20 decimals to see the number of decimals that are really stored in the variable.

```
1 $ cd /home/pi
2 $ nano NumberValues.java
3
4 public class NumberValues {
5     public static void main (String[] args) {
6         int intValue = 2;
7         float floatValue = 1.12345678901234567890F;
8         double doubleValue = 1.12345678901234567890D;
9
10        System.out.println("Integer: " + intValue);
11        System.out.println("Float: " + floatValue);
12        System.out.println("Double: " + doubleValue);
13
14        System.out.println("Multiply: " + (intValue * floatValue)
15            + ", rounded: " + Math.round(intValue * floatValue));
16    }
17 }
18
19 $ java NumberValues.java
20
21 Integer: 2
22 Float: 1.1234568
23 Double: 1.1234567890123457
24 Multiply: 2.2469137, rounded: 2
```

## If, Then, Else

An important function we will need a lot is comparing different values, which is done with if/then/else.

```
1 $ cd /home/pi
2 $ nano IfThenElse.java
3
4 public class IfThenElse {
5     public static void main (String[] args) {
6         // Compare integer value
7         int piHeaderVersion = 1;
8
9         // We don't change the variable in this example,
10        // but here could be some code to define this variable based on,
11        // for instance, a hardware check.
12
```

```

13     if (piHeaderVersion == 1) {
14         System.out.println("Header version 1 is used on original Model B");
15     } else if (piHeaderVersion == 2) {
16         System.out.println("Header version 2 is used on Model A and Model B"
17             + " (revision 2)");
18     } else if (piHeaderVersion == 3) {
19         System.out.println("Header version 3 is used on Model A+, B+, Pi Zero,"
20             + " Pi Zero W, Pi2B, Pi3B, Pi4B");
21     } else {
22         System.out.println("Sorry, header version " + piHeaderVersion
23             + " is not known");
24     }
25
26     // Compare strings
27     String string1 = "Hello world";
28     String string2 = "Hello" + " " + "world";
29     String string3 = "Hello World";
30
31     System.out.println("Are string1 and string2 equal? "
32         + string1.equals(string2));
33     System.out.println("Are string1 and string3 equal? "
34         + string1.equals(string3));
35     System.out.println("Are string1 and string3 equal ignoring the case? "
36         + string1.equalsIgnoreCase(string3));
37
38     if (string1.equalsIgnoreCase(string3)) {
39         System.out.println("string1 and string3 are equal ignoring the case");
40     }
41 }
42 }
43
44 $ java IfThenElse.java
45
46 Header version 1 is used on original Model B
47 Are string1 and string2 equal? true
48 Are string1 and string3 equal? false
49 Are string1 and string3 equal ignoring the case? true
50 string1 and string3 are equal ignoring the case

```

The line starting with “//” is a comment line, so it is ignored by Java. By changing the piHeaderVersion to 2, 3 or something else, you will see the text output of the several if blocks.

The second part of this example script shows how to compare string-values. Because a String is an

object, you can only correctly compare the content (= text) of the String by using the “equals()” or “equalsIgnoreCase()” methods.

## Enum and Switch

In the previous example, we use an integer value to define piHeaderVersion although we know we only can have three different possibilities instead of any integer value.

A better way to do this is by defining an “enum” with the available choices. Enums are by convention typed in uppercase.

```
1 $ cd /home/pi
2 $ nano IfThenElse.java
3
4 public class EnumSwitch {
5     public static void main (String[] args) {
6         // Compare integer value
7         HEADER_VERSION piHeaderVersion = HEADER_VERSION.TYPE_2;
8
9         // Same as before, we don't change the variable in this example.
10
11        switch(piHeaderVersion) {
12            case TYPE_1:
13                System.out.println("Header version 1 is used on original Model B");
14                break;
15            case TYPE_2:
16                System.out.println("Header version 2 is used on Model A and Model B"
17                    + " (revision 2)");
18                break;
19            case TYPE_3:
20                System.out.println("Header version 3 is used on Model A+, B+,"
21                    + " Pi Zero, Pi Zero W, Pi2B, Pi3B, Pi4B");
22                break;
23            default:
24                System.out.println("Sorry, header version " + piHeaderVersion
25                    + " is not known");
26        }
27    }
28
29    enum HEADER_VERSION {
30        TYPE_1, TYPE_2, TYPE_3, UNKNOWN;
31    }
32 }
```

```

33
34 $ java EnumSwitch.java
35 Header version 2 is used on Model A and Model B (revision 2)

```

Instead of if/then/else we are now using switch/case/default, resulting in more readable code. This is also easier to use and maintain as we know exactly which are the possible values for the header versions.

Don't forget to add the "break;" lines in each case. Try removing them and re-run the example.

```

1 $ java EnumSwitch.java
2 Header version 2 is used on Model A and Model B (revision 2)
3 Header version 3 is used on Model A+, B+, Pi Zero, Pi Zero W, Pi2B, Pi3B, Pi4B
4 Sorry, header version TYPE_2 is not known

```

Not what we wanted! All the cases after the valid one are returned now... That's also the reason we don't need to add a break in the default-block as this is the last one anyhow.

## Using methods

Now let's use some separated methods so we can keep the code simple and easy to read and understand. Each method does one specific thing, with or without input values.

```

1 $ cd /home/pi
2 $ nano UsingMethod.java
3
4 import java.text.SimpleDateFormat;
5 import java.util.Date;
6
7 public class UsingMethod {
8     public static void main (String[] args) {
9         System.out.println("2 x Raspberry Pi 4 4Gb, price: "
10             + getTotal(2, 59.95F) + " Euro");
11
12         System.out.println("Current date and time is: " + getNow());
13     }
14
15     public static float getTotal(int quantity, float price) {
16         return quantity * price;
17     }
18
19     public static String getNow() {
20         return new SimpleDateFormat("yyyy.MM.dd HH:mm:ss").format(new Date());

```

```

21     }
22 }
23
24 $ java UsingMethod.java
25
26 2 x Raspberry Pi 4 4Gb, price: 119.9000015258789 Euro
27 Current date and time is: 2019.12.03 12:35:23

```

First look at the import-lines. Because we use some methods which are not part of basic Java, we need to tell our program which additional classes need to be imported.

Then two methods are defined:

- “getTotal(int quantity, float price)” which returns a calculated value
- “getNow()” which returns the current timestamp as a readable formatted String

By calling these methods from the main method, we can keep the code in this main-method very clean and readable.

## Using objects

Java is an object-oriented programming language. This means we can turn any part of our code into an object with its variables and methods. For instance, let’s create a program containing a shopping cart list with items.



This example uses some methods which are not part of “base” Java. That’s why our code starts with the import-lines to make clear where ArrayList and List can be found.

Further in this book, not all imports will be included in the code examples, to keep everything nice and readable. If you use an IDE, you will be warned of the missing imports. Look into the GitHub sources of each example for the full code.

```

1 $ cd /home/pi
2 $ nano UsingObject.java
3
4 import java.util.ArrayList;
5 import java.util.List;
6
7 public class UsingObject {
8     public static void main (String[] args) {
9         List<ShoppingCartItem> items = new ArrayList<>();
10        items.add(new ShoppingCartItem("Raspberry Pi 4, 4Gb", 1, 59.95F));

```



```
11     items.add(new ShoppingCartItem("Micro-HDMI cable", 2, 5.9F));
12     items.add(new ShoppingCartItem("Raspberry Pi 4 power supply", 1, 9.95F));
13
14     double total = 0D;
15     for (ShoppingCartItem item : items) {
16         System.out.println(item.getName());
17         System.out.println("    " + item.getQuantity() + "\tx\t"
18             + item.getPrice() + "\t= " + item.getTotal() + " Euro");
19         total += item.getTotal();
20     }
21
22     System.out.println("\nTotal for shopping cart:\n    " + total + " Euro");
23 }
24
25 public static class ShoppingCartItem {
26     // These values are final as they should not be changed
27     private final String name;
28     private final int quantity;
29     private final float price;
30
31     public ShoppingCartItem(String name, int quantity, float price) {
32         this.name = name;
33         this.quantity = quantity;
34         this.price = price;
35     }
36
37     public String getName() {
38         return name;
39     }
40
41     public int getQuantity() {
42         return quantity;
43     }
44
45     public float getPrice() {
46         return price;
47     }
48
49     public float getTotal() {
50         return quantity * price;
51     }
52 }
53 }
```

```
54
55 $ java UsingObject.java
56 Raspberry Pi 4, 4Gb
57     1 x      59.95   = 59.95 Euro
58 Micro-HDMI cable
59     2 x      5.9     = 11.8 Euro
60 Raspberry Pi 4 power supply
61     1 x      9.95    = 9.95 Euro
62
63 Total for shopping cart:
64     81.70000076293945 Euro
```

The class “ShoppingCartItem” is an object which can hold the data for each item on the shopping list. The constructor “ShoppingCartItem(String name, int quantity, float price)” enables us to make an item that has a name, quantity, and price. The method “getTotal()” inside the item will return the total cost for the item based on quantity and price.

In the main method, we can now easily create a list with objects of type “ShoppingCartItem” and add a few of them in the list. With a for-loop, we can read all the items inside the list and calculate the total value.

In the “System.out.println” two special characters are used for better readable output:

- “\n” to jump to a new line
- “\t” to insert a tab



Readable, readable, READABLE

Hmm, I seem to have repeated that word a few times already.

There is nothing wrong with long methods with a lot of code inside of them. Java doesn't care. But the one looking at your code does! In many cases that will be only you and maybe you don't care either...

But imagine you've written a complex DIY-project and next year you want to improve or extend it. When you have structured your code into small methods that do a very specific, small thing only, you will be able to dive-in again very quickly and find the exact spot where you need to work in.

Another major topic is the way you name your variables. Which are the clearest ones from this example?

```

1  int var1 = 10;
2  int var2 = 21;
3
4  int outsideTemperature = 10;
5  int insideTemperature = 21;

```

So remember, breaking your code into smaller portions, with “real” variable names will help you and your colleagues to understand, maintain, improve and extend the code.

## Reading a text file

With a [free online tool](#)<sup>51</sup> we create a test CSV file with random data. This file is stored in a subdirectory “resources” as “testdata.csv”.

The file in the source code contains comma-separated values for counter, firstname, lastname, age, Street, City, State, ZIP:

```

1  1,Ada,Gomez,40,Mabvob Pike,Radafso,LA,60500
2  2,Bernard,Jordan,28,Dotcu Court,Cewbufbim,MS,17422
3  3,Mittie,Vaughn,64,Nandac Mill,Patunif,RI,81182
4  4,Miguel,Clarke,39,Liac Boulevard,Deguci,NH,32207
5  ...

```

Let's write a program to read this CSV-file line by line and convert each line to an object which is added to a list of persons:

<sup>51</sup><http://www.convertcsv.com/generate-test-data.htm>

```
1 $ cd /home/pi
2 $ nano UsingObject.java
3
4 import java.io.BufferedReader;
5 import java.io.File;
6 import java.io.FileReader;
7 import java.util.ArrayList;
8 import java.util.List;
9
10 public class ReadTextFile {
11     public static void main (String[] args) {
12         List<Person> persons = loadPersons();
13
14         System.out.println("Number of persons loaded from CSV file: "
15             + persons.size());
16
17         for (Person person : persons) {
18             System.out.println(person.getFullName() + ", age: " + person.getAge());
19         }
20     }
21
22     public static List<Person> loadPersons() {
23         List<Person> list = new ArrayList<>();
24
25         File file = new File("resources/testdata.csv");
26
27         try (Scanner scanner = new Scanner(file)) {
28             while (scanner.hasNextLine()) {
29                 list.add(new Person(scanner.nextLine()));
30             }
31         } catch (FileNotFoundException ex) {
32             System.err.println("Could not find the file to be loaded");
33         }
34
35         return list;
36     }
37
38     public static class Person {
39         private int id;
40         private String firstName;
41         private String lastName;
42         private int age;
43         private String street;
```

```
44     private String city;
45     private String state;
46     private int zip;
47
48     public Person(String csvLine) {
49         String[] data = csvLine.split(",");
50
51         this.id = Integer.valueOf(data[0]);
52         this.firstName = data[1];
53         this.lastName = data[2];
54         this.age = Integer.valueOf(data[3]);
55         this.street = data[4];
56         this.city = data[5];
57         this.state = data[6];
58         this.zip = Integer.valueOf(data[7]);
59     }
60
61     public String getFirstName() {
62         return firstName;
63     }
64
65     public String getFullName() {
66         return firstName + " " + lastName;
67     }
68
69     public int getAge() {
70         return age;
71     }
72 }
73 }
74
75 $ java ReadTextFile.java
76 Number of persons loaded from CSV file: 100
77 Ada Gomez, age: 40
78 Bernard Jordan, age: 28
79 Mittie Vaughn, age: 64
80 Miguel Clarke, age: 39
81 ...
```

Just like in the UsingObjects-example, we use an object to store the data of each line in the CSV file, in this case, the object “Person”.

Within the loadPersons-method:

- the file is opened
- read line by line
- the text of each line is used to create a Person-object
- this object is added to the list
- the list is returned.

Within the main-method, the resulting list is used:

- to count the number of persons in the list
- to print each person's full name and age.

## Using streams

Streams were introduced as a new feature in Java 8. These allow you to perform a series of steps or actions on an object.

Let's extend the ReadTextFile example with some extra functions...

First, we need to add extra imports:

```
1 import java.util.IntSummaryStatistics;
2 import java.util.stream.Collectors;
```

And we add some output in the main method:

```
1 public static void main (String[] args) {
2     List<Person> persons = loadPersons();
3
4     System.out.println("Number of persons loaded from CSV file: " + persons.size());
5
6     for (Person person : persons) {
7         System.out.println(person.getFullName() + ", age: " + person.getAge());
8     }
9
10    System.out.println("-----");
11    System.out.println("Number of persons with first name");
12    System.out.println("    * 'Matthew':    "
13        + countFirstName(persons, "Matthew"));
14    System.out.println("    * 'Charlotte':    "
15        + countFirstName(persons, "Charlotte"));
16
17    System.out.println("-----");
18    IntSummaryStatistics stats = getAgeStats(persons);
```

```

19     System.out.println("Minimum age: " + stats.getMin());
20     System.out.println("Maximum age: " + stats.getMax());
21     System.out.println("Average age: " + stats.getAverage());
22 }

```

Of course, we need to add the methods “countFirstName” and “getAgeStats” which are used in this extra code in the “main”-method:

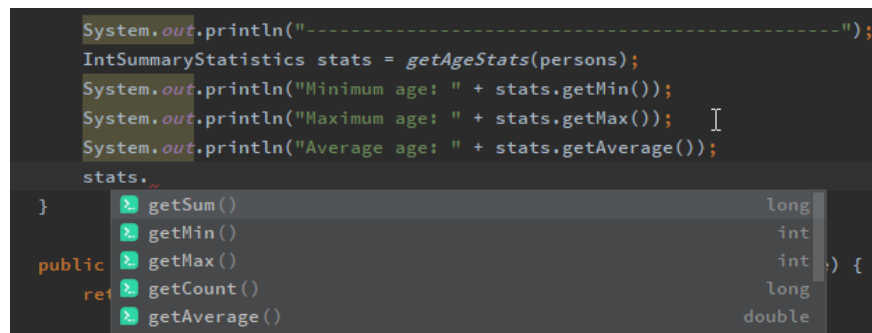
```

1  public static int countFirstName(List<Person> persons, String firstName) {
2      return (int) persons.stream()
3          .filter(p -> p.getFirstName().equals(firstName))
4          .count();
5  }
6
7  public static IntSummaryStatistics getAgeStats(List<Person> persons) {
8      return persons.stream()
9          .mapToInt(Person::getAge)
10         .summaryStatistics();
11 }

```

As you see, we start by converting the list into a stream, by adding “.stream()” behind the “persons”-list. We can further handle that stream step-by-step to obtain certain results.

- countFirstName
  - \* filter is used to get all the persons with the given first name
  - \* collect is used to make a list of all the matching persons
  - \* size returns the number of items in the collected list
- getAgeStats
  - \* mapToInt is used to only use the ages from all the persons
  - \* summaryStatistics returns an IntSummaryStatistics-object from the list from which we can get different values



The screenshot shows a code editor with the following code snippet:

```

System.out.println("-----");
IntSummaryStatistics stats = getAgeStats(persons);
System.out.println("Minimum age: " + stats.getMin());
System.out.println("Maximum age: " + stats.getMax());
System.out.println("Average age: " + stats.getAverage());
stats.
}
public
ret

```

A dropdown menu is open under the `stats.` prefix, showing the following methods and their return types:

| Method                    | Return Type |
|---------------------------|-------------|
| <code>getSum()</code>     | long        |
| <code>getMin()</code>     | int         |
| <code>getMax()</code>     | int         |
| <code>getCount()</code>   | long        |
| <code>getAverage()</code> | double      |

The methods provided by IntSummaryStatistics

When we run this same application again now, we get this output:

```
1 $ java ReadTextFile.java
2 Number of persons loaded from CSV file: 100
3 Ada Gomez, age: 40
4 Bernard Jordan, age: 28
5 Mittie Vaughn, age: 64
6 Miguel Clarke, age: 39
7 ...
8 Alexander McCoy, age: 46
9 Callie Fitzgerald, age: 32
10 -----
11 Number of persons with first name
12     * 'Matthew':      2
13     * 'Charlotte':   4
14 -----
15 Minimum age: 18
16 Maximum age: 65
17 Average age: 42.78
```

## What's next?

Have fun! Experiment! Extend and break the examples! ;-)  
You can only learn a programming language by using it...!



chapter\_04\_Java > README.md

In the README file of this chapter, you'll find a bunch of links where you can learn more and in detail about Java.



## Interview with Jakob Jenkov

*One of the best Java learning sites is [tutorials.jenkov.com](http://tutorials.jenkov.com)<sup>a</sup>. If you want to learn more about the Java “eco-system”, you’ll find plenty of information on this site! This site is a project of Jakob Jenkov, who is one of those experts which in almost every Java-experts-to-follow-list on Twitter or blogs.*

**Jakob Jenkov, [@jjenkov](https://twitter.com/jjenkov)<sup>b</sup>, Independent Software Architect, owns [jenkov.com](http://jenkov.com)<sup>c</sup>, Co-founder + CTO of [Nanosai](http://nanosai.com)<sup>d</sup>**

*What is your history in the Java world?*

I learned Java during university in 1995-1997 and started working with Java professionally from 1999. I started writing Java tutorials around 2005-2006 because I felt there were areas of Java that could be covered better.

*How did you start and maintain your tutorial website?*

I just started with a few topics I felt needed a bit more coverage - and which were interesting for me to study in more detail. Then the content just grew from there. However, as the body of content grows larger, keeping it up-to-date with new developments in the languages, APIs, frameworks, methods, etc. - is a growing task. For instance, from Aug. 2018 and until now (Feb. 2020) the majority of the effort I have put into the website has gone into updating existing tutorials. That is why regular readers might not have seen so many new topics added. I mostly maintain it in my spare time, and time in between consulting contracts.

*What is your motivation to do this?*

Well, at first I just wanted to see if there would be any readers. As I started getting readers, I was hoping that maybe it could be a living income. However, it takes *a lot* of traffic to make money enough from ads to make a living comparable to having a normal developer job (if you live in Europe or the US at least). Most websites don’t have that level of traffic. Now I mostly write about what I anyways need to learn e.g. for work or other projects, so I am not just plowing through tons of technology without a purpose. And - every time I need to Google something for a problem at work and find somebody else’s thorough solution to that problem, it confirms to me that sharing knowledge with everyone else is a good thing :-)

I don’t have a wife and children, so I have more extra time than many others.

*What are the biggest recent changes in Java?*

In my view, Java is evolving towards a smaller footprint (smaller JDK), higher performance and more control. Java is also evolving towards building standalone applications with the Java platform included. The various APIs become more “optional”, meaning if you don’t use them you can leave them out of your application distribution. APIs like Java EE and JavaFX which were considered “core” Java before, have been separated out and will take on a life of their own. Both are still very popular though.

I believe this development is good in that sometimes what we believe is “the best way” to do something now, turns out to be a bad way when we get more experience with it. This way those

bad parts are not an integral part of the JDK anymore. This will unfortunately also lead to more fragmentation of the community though, among various frameworks and toolkits.

*Why is now the perfect time to learn Java?*

The “minification” evolution of Java means that you don’t have to learn a thousand topics anymore to learn the basics of Java. With serverless Java support in e.g. AWS Lambdas and in Google App Engine you might not even have to learn all of Java EE anymore, or learn how to use a Java web container like Tomcat, Jetty, Wildfly, etc. In other words, you need to learn less of Java to be effective with Java than you did 10 years ago.

Java is already a strong platform for web applications, microservices, data streaming, and other things server-side. Additionally, JavaFX can now run on Windows, Linux, Mac, iOS, Android / Chromebook and Raspberry Pi. This makes Java a viable native GUI app platform and language too.

*Which DIY-programming-electronics-project are you working on?*

None. I am primarily a “software enthusiast” :-). Though I find mobile computing very interesting since we almost always carry this device around with us everywhere. But I don’t see myself inventing a new mobile phone :-D

---

<sup>a</sup><http://tutorials.jenkov.com>

<sup>b</sup><https://twitter.com/jjenkov>

<sup>c</sup><http://jenkov.com>

<sup>d</sup><http://nanosai.com>

# Chapter 5: Raspberry Pi pinning

Connecting electronic components to the Pi is done via one or more of the 40 pins in the so-called header.

Let's take a look at the Raspberry Pi history and its versions to better understand what is possible with these GPIOs. The following images and tables in this chapter are all generated from the open-sourced Java library [PiHeaders on GitHub](https://github.com/FDelporte/PiHeaders)<sup>52</sup> and can be used as a [Maven dependency](https://mvnrepository.com/artifact/be.webtechie/pi-headers)<sup>53</sup>.

|         |       |                    |        |                              |                                    |
|---------|-------|--------------------|--------|------------------------------|------------------------------------|
| 3.3 VDC |       | PIN 1              | PIN 2  | 5.0 VDC                      |                                    |
| BCM WPI | 2 8   | SDA1 (I2C)         | PIN 3  | PIN 4                        | 5.0 VDC                            |
| BCM WPI | 3 9   | SCL1 (I2C)         | PIN 5  | PIN 6                        | Ground                             |
| BCM WPI | 4 7   | GPCLK0             | PIN 7  | PIN 8                        | UART TxD WPI BCM<br>15 14          |
| Ground  |       | PIN 9              | PIN 10 | UART RxD WPI BCM<br>16 15    |                                    |
| BCM WPI | 17 0  | PIN 11             | PIN 12 | PCM_CLK/PWM0 WPI BCM<br>1 18 |                                    |
| BCM WPI | 27 2  | PIN 13             | PIN 14 | Ground                       |                                    |
| BCM WPI | 22 3  | PIN 15             | PIN 16 | 4 23                         |                                    |
| 3.3 VDC |       | PIN 17             | PIN 18 | 5 24                         |                                    |
| BCM WPI | 10 12 | MOSI (SPI)         | PIN 19 | PIN 20                       | Ground                             |
| BCM WPI | 9 13  | MISO (SPI)         | PIN 21 | PIN 22                       | WPI BCM<br>6 25                    |
| BCM WPI | 11 14 | SCLK (SPI)         | PIN 23 | PIN 24                       | CE0 (SPI) WPI BCM<br>10 8          |
| Ground  |       | PIN 25             | PIN 26 | CE1 (SPI) WPI BCM<br>11 7    |                                    |
| BCM WPI | 0 30  | SDA0 I2C ID EEPROM | PIN 27 | PIN 28                       | SCL0 I2C ID EEPROM WPI BCM<br>31 1 |
| BCM WPI | 5 21  | GPCLK1             | PIN 29 | PIN 30                       | Ground                             |
| BCM WPI | 6 22  | GPCLK2             | PIN 31 | PIN 32                       | PWM0 WPI BCM<br>26 12              |
| BCM WPI | 13 23 | PWM1               | PIN 33 | PIN 34                       | Ground                             |
| BCM WPI | 19 24 | PCM_FS/PWM1        | PIN 35 | PIN 36                       | WPI BCM<br>27 16                   |
| BCM WPI | 26 25 | PIN 37             | PIN 38 | PCM_DIN WPI BCM<br>28 20     |                                    |
| Ground  |       | PIN 39             | PIN 40 | PCM_DOUT WPI BCM<br>29 21    |                                    |

|                                       |                          |
|---------------------------------------|--------------------------|
| <span style="color: red;">■</span>    | Power                    |
| <span style="color: black;">■</span>  | Ground                   |
| <span style="color: green;">■</span>  | Digital                  |
| <span style="color: yellow;">■</span> | Digital and PWM          |
| <span style="color: purple;">■</span> | Digital without pulldown |

40-pins header

<sup>52</sup><https://github.com/FDelporte/PiHeaders>

<sup>53</sup><https://mvnrepository.com/artifact/be.webtechie/pi-headers>



The image visualizing the header pins of a Raspberry Pi is a screenshot from a JavaFX project which you can find in:

Chapter\_05\_PiPinning > javafx-pinning-info

In one of the next chapters we will dive into JavaFX, but feel free to look around in the code for some first insights.

This application uses this Maven library:

```
1 <dependency>
2     <groupId>be.webtechie</groupId>
3     <artifactId>pi-headers</artifactId>
4     <version>0.1.1</version>
5 </dependency>
```

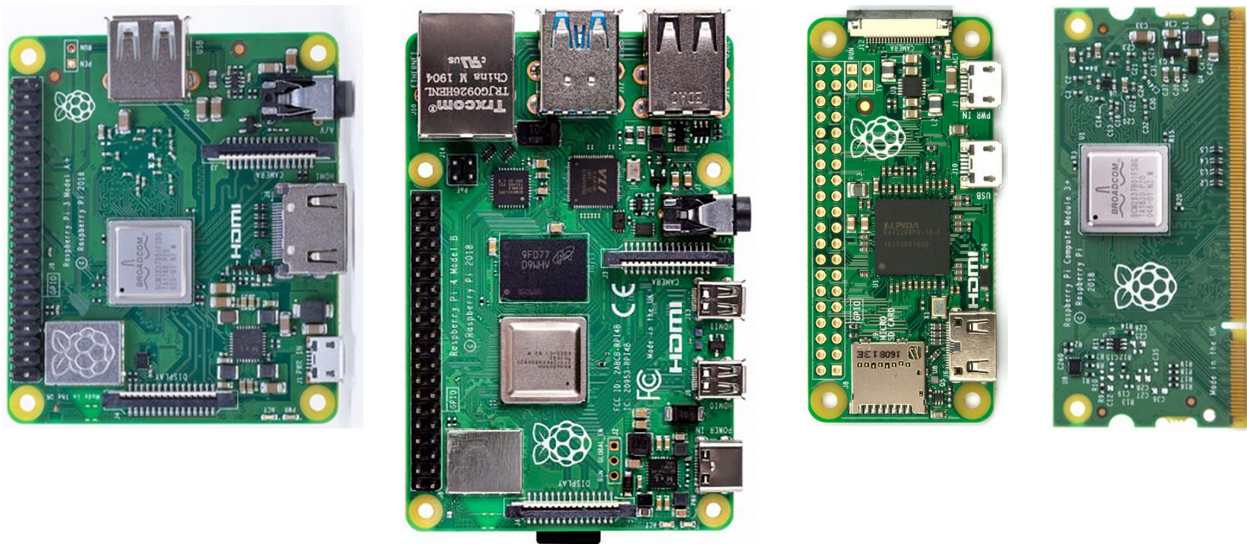
## Raspberry Pi types

### Models

Raspberry Pi boards are available in different models, enabling you to select the right one for your project. Model B is most known and used as it has the most connections. If you don't need a cabled network connection, you can use a Model A. The Zero is the smallest one, but not applicable for Java projects as it doesn't have the correct processor (see "Chapter 4: Java").

The Compute module is aimed at professional use as it needs to be integrated on a board that can be fully adjusted to fit in a specific product. It is already used in industrial controllers and computers, home automation, panel PCs, ...

| Name           | Description                                                                 |
|----------------|-----------------------------------------------------------------------------|
| Model A        | Without ethernet connector                                                  |
| Model B        | With ethernet connector                                                     |
| Zero           | Smaller size and reduced GPIO capabilities                                  |
| Compute Module | Pi on a 200-pin DDR2-memory-like module for integration in embedded devices |



Model A - Model B - Zero - Compute Module

### Major versions

Both Model A and B evolved and had different versions. Each version extended and improved the boards. These were the main changes:

| Name      | Model  | Description                                | Headers                    |
|-----------|--------|--------------------------------------------|----------------------------|
| Version 1 | A & B  | First generation                           | 26pin header               |
| Version 2 | B only | Added more RAM                             | 26pin header + 8pin header |
| Version 3 | A & B  | Including WiFi and Bluetooth               | 40pin header               |
| Version 4 | B only | Gigabit ethernet, USB 3.0 and dual monitor | 40pin header               |

## Board versions

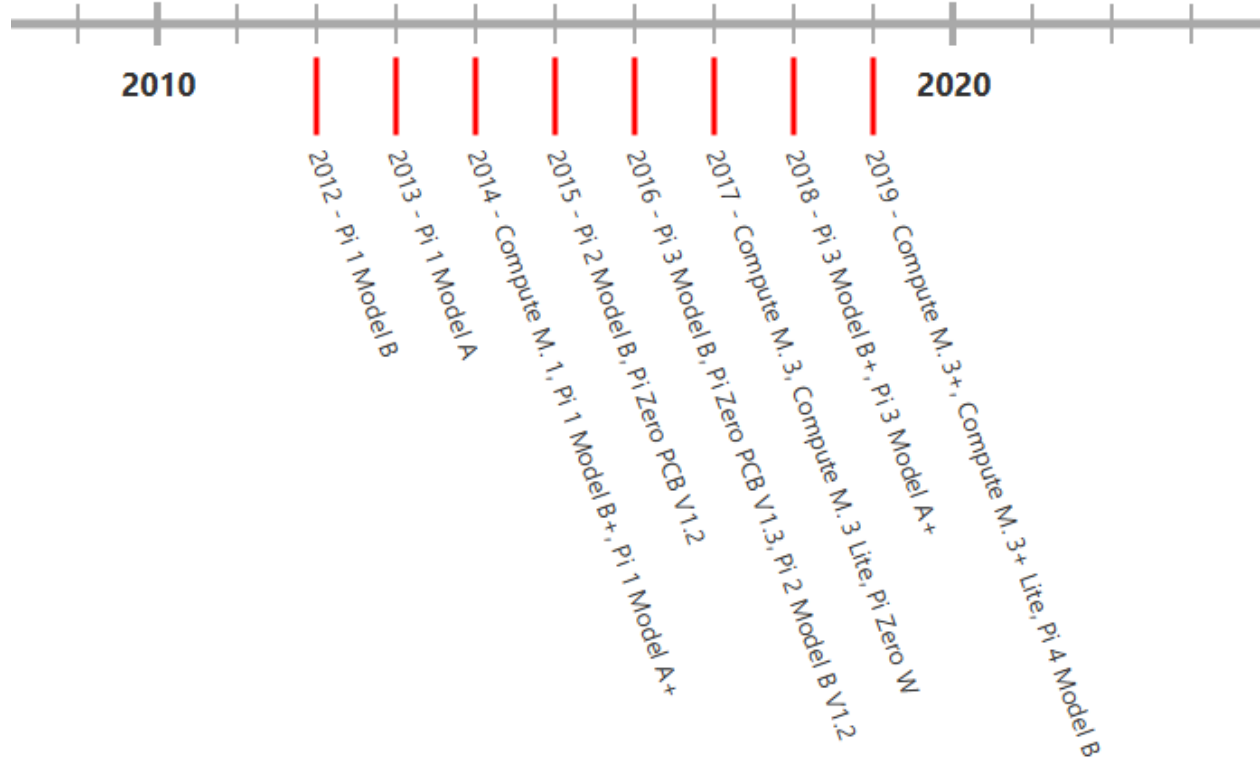
The full list of board versions with the models and major versions gives this list:

| Name                   | Model          | Version   | Release date |
|------------------------|----------------|-----------|--------------|
| Pi 1 Model A           | Model A        | Version 1 | 2013-02      |
| Pi 1 Model A+          | Model A        | Version 1 | 2014-11      |
| Pi 3 Model A+          | Model A        | Version 3 | 2018-11      |
| Pi 1 Model B           | Model B        | Version 1 | 2012-04      |
| Pi 1 Model B+          | Model B        | Version 1 | 2014-07      |
| Pi 2 Model B           | Model B        | Version 2 | 2015-02      |
| Pi 2 Model B V1.2      | Model B        | Version 2 | 2016-10      |
| Pi 3 Model B           | Model B        | Version 3 | 2016-02      |
| Pi 3 Model B+          | Model B        | Version 3 | 2018-03      |
| Pi 4 Model B           | Model B        | Version 4 | 2019-06      |
| Compute Module 1       | Compute Module |           | 2014-04      |
| Compute Module 3       | Compute Module |           | 2017-01      |
| Compute Module 3 Lite  | Compute Module |           | 2017-01      |
| Compute Module 3+      | Compute Module |           | 2019-01      |
| Compute Module 3+ Lite | Compute Module |           | 2019-01      |
| Pi Zero PCB V1.2       | Zero           |           | 2015-11      |
| Pi Zero PCB V1.3       | Zero           |           | 2016-05      |
| Pi Zero W              | Zero           |           | 2017-02      |



By using this board version list of the Maven dependency in combination with the JavaFX timeline application, that we already used in “Chapter 4: Java”, you get this nice timeline with the Raspberry Pi model history.

```
Chapter_04_Java > java-timeline
```



Timeline with the Raspberry Pi models

## Pin types

The type of header is different between the Pi board versions and contains 8, 26 or 40 pins. These pins have different functions:

### Power and ground

Both 5V and 3.3V are available as power pins and, of course, also ground pins. Anytime the board is powered you have a fixed power supply available for your components. As mentioned in “Chapter 2: Tools and hardware used in this book” > “Hardware components” > “LED strips”, you have to take into account not to connect devices that need a lot of current!

### Digital GPIO

The other ones are “General-Purpose Input/Output” (GPIO) pins. These pins can be addressed with software to act as input or output for an application. They use 3.3V, meaning an output pin will be set to 0V (low) or 3.3V (high) and an input pin will read 0V as low and 3.3V as high.

Most of the GPIOs have an internal pull-up or pull-down resistor which can be enabled in software. The use of this is illustrated with the use of a push-button in “Chapter 5: Pi4J > Digital input with a button”.



## Pin functions

Certain GPIOs can be used for specific functions. The functionality is described here shortly, while in “Chapter 7: JavaFX” and “Chapter 9: Pi4J” examples are provided to make their specific functionalities more clear.

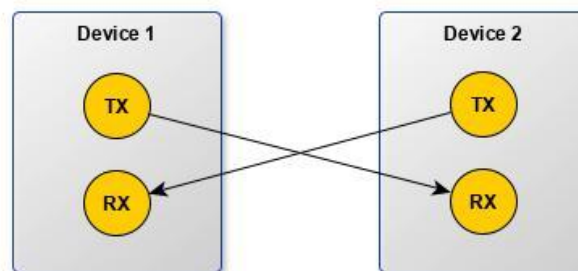
| Name  | Label                                           | Description                                |
|-------|-------------------------------------------------|--------------------------------------------|
| UART  | Universal Asynchronous Receiver and Transmitter | Asynchronous serial communication protocol |
| GPCLK | General Purpose Clock                           | Output a fixed frequency                   |
| I2C   | Inter Integrated Circuit                        | Synchronous serial computer bus            |
| SPI   | Serial Peripheral Interface                     | Four-wire serial bus                       |
| PWM   | Pulse-Width Modulation                          | Generate analog output with a digital one  |

### Universal Asynchronous Receiver and Transmitter (UART - Serial)

UART is the system that enables us to send and receive serial data by using a shift register (for more info, see “Chapter 8: Bits and Bytes > Controlling a numeric segment display”).

Serial communication is a way to transfer data, bit by bit in a sequence, through a single wire from a transmitter (= TX) to a receiver (= RX) where the bits are combined again to bytes.

For two-way communication, two wires are needed between RX and TX from both sides:



Connection between two devices

The communication between these two devices can happen in different ways:

- simplex: one direction only without a message back to confirm the receiving
- half-duplex: devices can only send or receive at once
- full-duplex: both devices can send and receive at the same time

Serial communication is done at a predefined speed which needs to be known on both sides, as both the sender and receiver need to handle the data at the same speed. This speed is called the “baud rate” with a value indicating the number of bits per second (bps). The most used speed is 9600 bps,

but you can use lower (1200, 2400, 4800) or higher (19200, 38400, 57600, 115200) speeds, depending on the speed which can be handled reliably by the device.

The Pi has two built-in UART connections to GPIOs BCM number 14 (TX) and 15 (RX). Identical to the other GPIOs, they use 3.3V so make sure, when you connect other devices, the same voltage is used.

## General Purpose Clock (GPCLK)

General Purpose Clock pins can be configured to continuously output a fixed frequency. So once your software has started this output, no further control is needed. The clock signal as very stable as it is generated by hardware components.

The GPCLK uses three pins with BCM numbers 4, 5 and 6.

## Inter Integrated Circuit (I<sup>2</sup>C)

I<sup>2</sup>C, invented by Philips (1982), is a serial computer bus to transfer data with these features:

- synchronous: exchange of data based on a clock signal
- multi-master and multi-slave: multiple controllers can be used together with multiple devices
- packet-switched: data is grouped in packets with a header and payload
- single-ended: varying voltage represents the data compared to a reference voltage (the ground)

It can be used to transfer (low-speed) data between boards over a short distance.

I<sup>2</sup>C needs two cables and two pins are designed for this purpose, GPIOs with BCM numbers 2 (data) and 3 (clock).

## Serial Peripheral Interface (SPI)

Just like I<sup>2</sup>C, SPI uses separate cables for the data and the clock to make sure the data is read at exactly the right time. Where I<sup>2</sup>C needs two cables, SPI needs four but is faster while using less power.

SPI was invented by Motorola in the mid-1980s. Data is transferred in full-duplex mode, using a single master and one or more slaves.

In most hardware diagrams and on SPI-components you'll find these names:

- MOSI = Master Out Slave In = Communication from Pi to component
- MISO = Master In Slave Out = Communication from component to Pi

## Pulse-Width Modulation (PWM)

As a digital GPIO can only be low (0V) or high (3.3V) it is not possible to have a real analog output, for example, if you want to fade an LED to half brightness.

In these cases, PWM is the perfect solution! By switching fast enough between low and high and using different durations for the low and/or high state, a “semi-analog output” can be achieved by creating an averaged value. The values to be used in this case are:

- On-time: duration the output is high
- Off-time: duration the output is low
- Period: on-time + off-time
- Duty cycle: percentage of the time the output is high

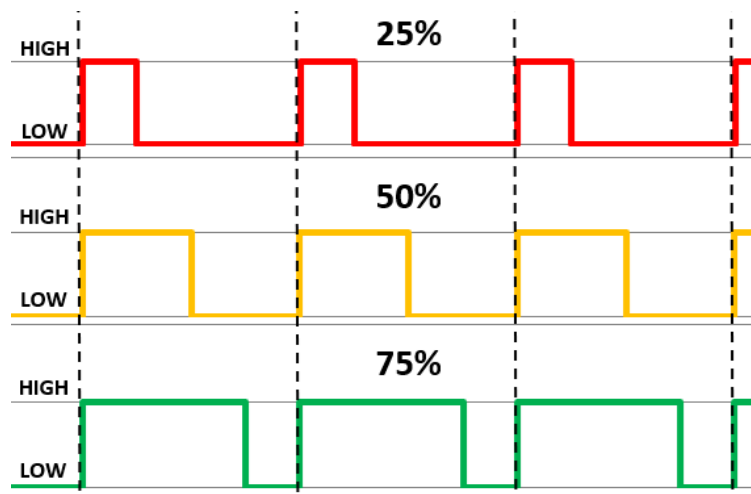


Chart showing a 25%, 50% and 75% PWM signal



I was going to make a JavaFX application to generate this chart, just like the one used for the timeline images in “Chapter 4: Java”. However, to avoid the “NIH Syndrome” (see “Just a thought: Important abbreviations”) I decided to first look for an existing solution. After a quick search, I found an [instructable by Artworker<sup>54</sup>](https://www.instructables.com/id/Make-Digital-Graphs-in-Excel/) in which a nice and ready to use Excel is shared to generate the above image.

PWM can be generated with software (e.g. using timers) on all GPIO pins, but a more stable signal can be produced with hardware functions provided on GPIOs with BCM numbers 12, 13, 18 and 19. These are marked in the PIN table with the type “DIGITAL\_AND\_PWM” (yellow).






<sup>54</sup><https://www.instructables.com/id/Make-Digital-Graphs-in-Excel/>

## Header types

Depending on the type of board one or two headers are available with different GPIO functions on the pins:

| Description                                        | Headers                             |
|----------------------------------------------------|-------------------------------------|
| Original Model B                                   | 26pin header - type 1               |
| Model A and Model B (revision 2)                   | 26pin header - type 2 + 8pin header |
| Model A+, B+, Pi Zero, Pi Zero W, Pi2B, Pi3B, Pi4B | 40pin header                        |

The colors in the header and table view indicate the type of pin:

|                                                                                   |                          |
|-----------------------------------------------------------------------------------|--------------------------|
|  | Power                    |
|  | Ground                   |
|  | Digital                  |
|  | Digital and PWM          |
|  | Digital without pulldown |

Color legend header pin types

## 40-pin header

You can find a header visualization at the start of this chapter. That same data can be used in the same JavaFX application to generate a table view. This is the pin list of the 40-pin header:

| Pin n° | WiringPi n° | BCM n° | Name                | Type                 | Function | Remark                                    |
|--------|-------------|--------|---------------------|----------------------|----------|-------------------------------------------|
| 1      |             |        | 3.3 VDC             | POWER                |          |                                           |
| 2      |             |        | 5.0 VDC             | POWER                |          |                                           |
| 3      | 8           | 2      | SDA1 (I2C)          | DIGITAL_NO_PULL_DOWN | I2C      | SDA.1 pin has a physical pull-up resistor |
| 4      |             |        | 5.0 VDC             | POWER                |          |                                           |
| 5      | 9           | 3      | SCL1 (I2C)          | DIGITAL_NO_PULL_DOWN | I2C      | SCL.1 pin has a physical pull-up resistor |
| 6      |             |        | Ground              | GROUND               |          |                                           |
| 7      | 7           | 4      | GPCLK0              | DIGITAL              | GPCLK    |                                           |
| 8      | 15          | 14     | UART TxD            | DIGITAL              | UART     |                                           |
| 9      |             |        | Ground              | GROUND               |          |                                           |
| 10     | 16          | 15     | UART RxD            | DIGITAL              | UART     |                                           |
| 11     | 0           | 17     |                     | DIGITAL              | SPI      |                                           |
| 12     | 1           | 18     | PCM_CLK/PWM0        | DIGITAL_AND_PWM      | SPI      | Supports PWM0 [ALT5]                      |
| 13     | 2           | 27     |                     | DIGITAL              |          |                                           |
| 14     |             |        | Ground              | GROUND               |          |                                           |
| 15     | 3           | 22     |                     | DIGITAL              |          |                                           |
| 16     | 4           | 23     |                     | DIGITAL              |          |                                           |
| 17     |             |        | 3.3 VDC             | POWER                |          |                                           |
| 18     | 5           | 24     |                     | DIGITAL              |          |                                           |
| 19     | 12          | 10     | MOSI (SPI)          | DIGITAL              | SPI      |                                           |
| 20     |             |        | Ground              | GROUND               |          |                                           |
| 21     | 13          | 9      | MISO (SPI)          | DIGITAL              | SPI      |                                           |
| 22     | 6           | 25     |                     | DIGITAL              |          |                                           |
| 23     | 14          | 11     | SCLK (SPI)          | DIGITAL              | SPI      |                                           |
| 24     | 10          | 8      | CE0 (SPI)           | DIGITAL              | SPI      |                                           |
| 25     |             |        | Ground              | GROUND               |          |                                           |
| 26     | 11          | 7      | CE1 (SPI)           | DIGITAL              | SPI      |                                           |
| 27     | 30          | 0      | SDA0 I2C ID EEPR... | DIGITAL_NO_PULL_DOWN | I2C      | SDA.0 pin has a physical pull-up resistor |
| 28     | 31          | 1      | SCL0 I2C ID EEPROM  | DIGITAL_NO_PULL_DOWN | I2C      | SDC.0 pin has a physical pull-up resistor |
| 29     | 21          | 5      | GPCLK1              | DIGITAL              | GPCLK    |                                           |
| 30     |             |        | Ground              | GROUND               |          |                                           |
| 31     | 22          | 6      | GPCLK2              | DIGITAL              | GPCLK    |                                           |
| 32     | 26          | 12     | PWM0                | DIGITAL_AND_PWM      |          | Supports PWM0 [ALT0]                      |
| 33     | 23          | 13     | PWM1                | DIGITAL_AND_PWM      |          | Supports PWM1 [ALT0]                      |
| 34     |             |        | Ground              | GROUND               |          |                                           |
| 35     | 24          | 19     | PCM_FS/PWM1         | DIGITAL_AND_PWM      | SPI      | Supports PWM1 [ALT5]                      |
| 36     | 27          | 16     |                     | DIGITAL              | SPI      |                                           |
| 37     | 25          | 26     |                     | DIGITAL              |          |                                           |
| 38     | 28          | 20     | PCM_DIN             | DIGITAL              | SPI      |                                           |
| 39     |             |        | Ground              | GROUND               |          |                                           |
| 40     | 29          | 21     | PCM_DOUT            | DIGITAL              | SPI      |                                           |

Table view of the pins in the header



The above table view threw an error while developing because JavaFX PropertyValueFactory was unable to receive property from a class located in a different package.

This was caused by the module system which has been added in Java version 9 and only allows modules to access other modules when permission is defined for it.

In this project this was fixed by adding the “opens” line in “module-info.java”:

```

1 module be.webtechie {
2     requires javafx.controls;
3     requires be.webtechie.piheaders;
4     opens be.webtechie to javafx.graphics;
5 }

```

## 26-pin header - Type 1 and 2

Both types are almost identical, except some of the BCM numbers for pin 3, 5 and 13:

| 3.3 VDC |     |            |     | PIN | 5.0 VDC      |     |     |              |
|---------|-----|------------|-----|-----|--------------|-----|-----|--------------|
| BCM     | WPI | PIN        | PIN | 1   | 2            | 3   | 4   | 5.0 VDC      |
| 0       | 8   | SDA1 (I2C) | 3   | 4   | 5.0 VDC      |     |     |              |
| BCM     | WPI | PIN        | PIN | 5   | 6            |     |     | Ground       |
| 1       | 9   | SCL1 (I2C) | 5   | 6   | Ground       |     |     |              |
| BCM     | WPI | PIN        | PIN | 7   | 8            |     |     | UART Tx/D    |
| 4       | 7   | GPCLK0     | 7   | 8   | UART Tx/D    | WPI | BCM | 15 14        |
|         |     | Ground     | PIN | PIN |              |     |     | UART Rx/D    |
|         |     | Ground     | 9   | 10  | UART Rx/D    | WPI | BCM | 16 15        |
| BCM     | WPI | PIN        | PIN | 11  | 12           |     |     | PCM_CLK/PWM0 |
| 17      | 0   |            | 11  | 12  | PCM_CLK/PWM0 | WPI | BCM | 1 18         |
| BCM     | WPI | PIN        | PIN | 13  | 14           |     |     | Ground       |
| 21      | 2   |            | 13  | 14  | Ground       |     |     |              |
| BCM     | WPI | PIN        | PIN | 15  | 16           |     |     | 4 23         |
| 22      | 3   |            | 15  | 16  | 4 23         |     |     |              |
|         |     | 3.3 VDC    | PIN | PIN |              |     |     | WPI          |
|         |     | 3.3 VDC    | 17  | 18  | 5 24         |     |     | 5 24         |
| BCM     | WPI | PIN        | PIN | 19  | 20           |     |     | Ground       |
| 10      | 12  | MOSI (SPI) | 19  | 20  | Ground       |     |     |              |
| BCM     | WPI | PIN        | PIN | 21  | 22           |     |     | WPI          |
| 9       | 13  | MISO (SPI) | 21  | 22  | 6 25         |     |     | 6 25         |
| BCM     | WPI | PIN        | PIN | 23  | 24           |     |     | WPI          |
| 11      | 14  | SCLK (SPI) | 23  | 24  | CE0 (SPI)    | WPI | BCM | 10 8         |
|         |     | Ground     | PIN | PIN |              |     |     | WPI          |
|         |     | Ground     | 25  | 26  | CE1 (SPI)    | WPI | BCM | 11 7         |

26 pins header type 1 (left) and 2 (right)

## 8-pin header

|                  |          |          |                  |
|------------------|----------|----------|------------------|
| 5.0 VDC          | PIN<br>1 | PIN<br>2 | 3.3 VDC          |
| BCM WPI<br>28 17 | PIN<br>3 | PIN<br>4 | WPI BCM<br>18 29 |
| BCM WPI<br>30 19 | PIN<br>5 | PIN<br>6 | WPI BCM<br>20 31 |
| Ground           | PIN<br>7 | PIN<br>8 | Ground           |

8 pins header

## Different pinning numbering schemes

As you can see in the header tables, there are three different numbers.

### Board (pin) number

These are the logical numbers in the header where one row has all the even numbers and the odd ones are on the other row.



Pi pin header with the indication of pin 1, 2, 3, 37, 39 and 40

### BCM number

BCM refers to the “Broadcom SOC channel” number, which is the numbering inside the chip which is used on the Raspberry Pi.

These numbers changed between board versions as you can see in the previous tables for the 26-pin header type 1 versus 2, and or not sequential.

### WiringPi number

[WiringPi](http://wiringpi.com)<sup>55</sup> is developed by [Gordon Henderson](https://twitter.com/drogon)<sup>56</sup> and installed by default on the Pi when you use Raspbian. You can easily check the version and pins with the commands “`gpio -v`” and “`gpio readall`”. This is the result on a Pi 3B+:

---

<sup>55</sup><http://wiringpi.com>

<sup>56</sup><https://twitter.com/drogon>



```

1  $ gpio -v
2  gpio version: 2.50
3  Copyright (c) 2012-2018 Gordon Henderson
4  This is free software with ABSOLUTELY NO WARRANTY.
5  For details type: gpio -warranty
6
7  Raspberry Pi Details:
8  Type: Pi 3B+, Revision: 03, Memory: 1024MB, Maker: Sony
9  * Device tree is enabled.
10 *--> Raspberry Pi 3 Model B Plus Rev 1.3
11 * This Raspberry Pi supports user-level GPIO access.
12
13 $ gpio readall
14 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
15 | BCM | wPi | Name | Mode | V | Physical | V | Mode | Name | wPi | BCM |
16 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
17 |    |    | 3.3v |    |  | 1 || 2 |    |    | 5v  |    |    |
18 |  2 |  8 | SDA.1 | IN | 1 | 3 || 4 |    |    | 5v  |    |    |
19 |  3 |  9 | SCL.1 | IN | 1 | 5 || 6 |    |    | 0v  |    |    |
20 |  4 |  7 | GPIO.7 | IN | 1 | 7 || 8 | 0 | IN | TxD  | 15 | 14 |
21 |    |    | 0v  |    |  | 9 || 10 | 1 | IN | RxD  | 16 | 15 |
22 | 17 |  0 | GPIO.0 | IN | 0 | 11 || 12 | 0 | IN | GPIO.1 | 1 | 18 |
23 | 27 |  2 | GPIO.2 | IN | 0 | 13 || 14 |    |    | 0v  |    |    |
24 | 22 |  3 | GPIO.3 | IN | 0 | 15 || 16 | 0 | IN | GPIO.4 | 4 | 23 |
25 |    |    | 3.3v |    |  | 17 || 18 | 0 | IN | GPIO.5 | 5 | 24 |
26 | 10 | 12 | MOSI | IN | 0 | 19 || 20 |    |    | 0v  |    |    |
27 |  9 | 13 | MISO | IN | 0 | 21 || 22 | 0 | IN | GPIO.6 | 6 | 25 |
28 | 11 | 14 | SCLK | IN | 0 | 23 || 24 | 1 | IN | CE0  | 10 | 8  |
29 |    |    | 0v  |    |  | 25 || 26 | 1 | IN | CE1  | 11 | 7  |
30 |  0 | 30 | SDA.0 | IN | 1 | 27 || 28 | 1 | IN | SCL.0 | 31 | 1  |
31 |  5 | 21 | GPIO.21 | IN | 1 | 29 || 30 |    |    | 0v  |    |    |
32 |  6 | 22 | GPIO.22 | IN | 1 | 31 || 32 | 0 | IN | GPIO.26 | 26 | 12 |
33 | 13 | 23 | GPIO.23 | IN | 0 | 33 || 34 |    |    | 0v  |    |    |
34 | 19 | 24 | GPIO.24 | IN | 0 | 35 || 36 | 0 | IN | GPIO.27 | 27 | 16 |
35 | 26 | 25 | GPIO.25 | IN | 0 | 37 || 38 | 0 | IN | GPIO.28 | 28 | 20 |
36 |    |    | 0v  |    |  | 39 || 40 | 0 | IN | GPIO.29 | 29 | 21 |
37 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
38 | BCM | wPi | Name | Mode | V | Physical | V | Mode | Name | wPi | BCM |
39 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

As you can see “gpio readall” gives you a nice overview of the different pin numberings and functions.

The WiringPi numbering can be a bit confusing if you don’t have an overview table. But this

numbering has a “historical reason”. When development for the very first Raspberry Pi’s was ongoing, only 8 pin numbers were foreseen. But when the designs further evolved and more pins were added, the numbering in WiringPi was extended to be able to address the extra pins.

As Pi4J (see Chapter 9) uses WiringPi “under the hood”, it’s important to know this numbering scheme is used. In the example code, the BCM number is also added to make things more clear.

**Important, if you are using a Raspberry Pi 4, you will need to make sure you are on the latest version 2.52.** Because the internal wiring of the processor of this type is different compared to the previous boards, an updated gpio is available. Check the version with “gpio -v” in the terminal and install the new version with the following commands:

```
1 $ gpio -v
2 gpio version: 2.50
3 $ cd /tmp
4 $ wget https://project-downloads.drogon.net/wiringpi-latest.deb
5 $ sudo dpkg -i wiringpi-latest.deb
6 $ gpio -v
7 gpio version: 2.52
```

Unfortunately the maker of WiringPi [announced in August 2019, he stopped further development of this tool](#)<sup>57</sup>. His decision is based on the problem many open-source developers of popular tools are facing. A lot of people use it and ask for free support or even worse, re-use and sell products using the code without giving proper credits to the maker.

That’s the main reason this book and the README-files in the sources contain all the links of the sites and tools used while writing this book. Without all those people who publish and share their knowledge, creating the examples for this book would not have been possible.

---

<sup>57</sup><http://wiringpi.com/wiringpi-deprecated/>

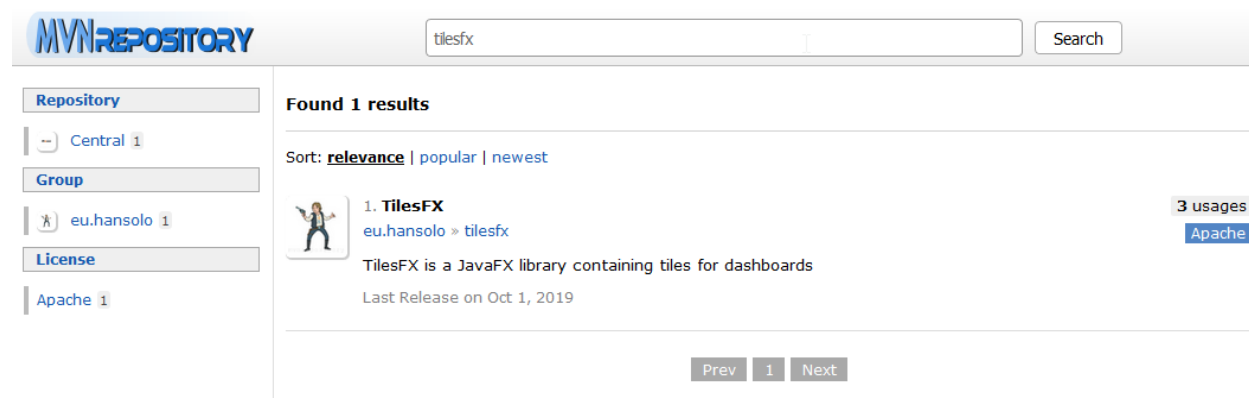
# Chapter 6: What is Maven?

**Maven**<sup>58</sup> is a tool to help you to manage the dependencies (external libraries) of your Java project and build the application for testing and distribution. It works by using a predefined structure to organize your code and to describe a project in a pom.xml (Project Object Model) file.

Based on different polls, about 50 to 60% of Java developers use Maven. There are other tools available to do the same thing, with Gradle being the second most popular one (20 to 30%). As I'm most familiar with Maven and it's the most used one, we will only work with it in this book.

Next to the development tool, there is also the on-line **Maven Repository**<sup>59</sup> where you can find a long list of versioned libraries.

Let's take a look for example at the TilesFX library we will be using in chapter 7. Go to the mvnrepository.com website and search for TilesFX.



TilesFX on Maven repository

If we click on the “tilesfx” link, we get the page with all the available versions of this library.

<sup>58</sup><https://maven.apache.org/>

<sup>59</sup><https://mvnrepository.com/>

Home » eu.hansolo » tilesfx

**TilesFX**  
TilesFX is a JavaFX library containing tiles for dashboards

License: Apache 2.0  
Used By: 3 artifacts

Central (78) Spring Plugins (1)

| Version                       | Repository | Usages | Date      |
|-------------------------------|------------|--------|-----------|
| 11.14.x <a href="#">11.14</a> | Central    | 0      | Oct, 2019 |
| 11.13.x <a href="#">11.13</a> | Central    | 0      | Sep, 2019 |
| 11.12.x <a href="#">11.12</a> | Central    | 0      | Sep, 2019 |
| 11.11.x <a href="#">11.11</a> | Central    | 0      | Sep, 2019 |
| 11.10.x <a href="#">11.10</a> | Central    | 0      | Sep, 2019 |
| 11.9.x <a href="#">11.9</a>   | Central    | 0      | Sep, 2019 |
| 11.8.x <a href="#">11.8</a>   | Central    | 0      | Sep, 2019 |
| 11.7.x <a href="#">11.7</a>   | Central    | 0      | Sep, 2019 |

TilesFX versions on Maven repository

By selecting the version you want to use (in most cases the latest and most recent one), you can copy and paste the xml dependency code to add to the pom.xml file.

Home » eu.hansolo » tilesfx » [11.14](#)



## TilesFX » 11.14

TilesFX is a JavaFX library containing tiles for dashboards

|              |                                                                                               |
|--------------|-----------------------------------------------------------------------------------------------|
| License      | Apache 2.0                                                                                    |
| HomePage     | <a href="https://github.com/HanSolo/tilesfx/wiki">https://github.com/HanSolo/tilesfx/wiki</a> |
| Date         | (Oct 01, 2019)                                                                                |
| Files        | <a href="#">jar (1.7 MB)</a> <a href="#">View All</a>                                         |
| Repositories | Central                                                                                       |
| Used By      | 3 artifacts                                                                                   |

Maven [Gradle](#) [SBT](#) [Ivy](#) [Grape](#) [Leiningen](#) [Buildr](#)

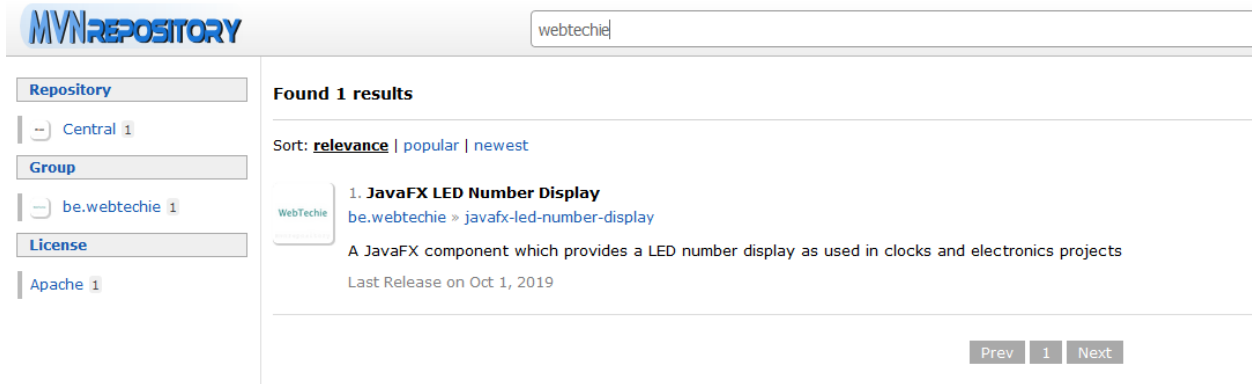
```

<!-- https://mvnrepository.com/artifact/eu.hansolo/tilesfx -->
<dependency>
  <groupId>eu.hansolo</groupId>
  <artifactId>tilesfx</artifactId>
  <version>11.14</version>
</dependency>

```

TilesFX dependency XML

Also the “LED number display” JavaFX library created for ‘Chapter 8 Bits & Bytes’ can be found in the Maven Repository:



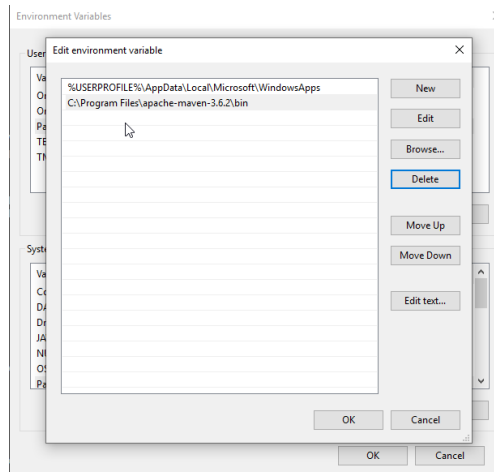
The screenshot shows the Maven Repository search interface. At the top left is the 'Maven Repository' logo. A search bar at the top right contains the text 'webtechie'. On the left side, there are filters for 'Repository' (Central 1), 'Group' (be.webtechie 1), 'License' (Apache 1), and 'Apache 1'. The main content area displays 'Found 1 results' and 'Sort: relevance | popular | newest'. The first result is '1. JavaFX LED Number Display' by 'WebTechie', with a link to 'be.webtechie » javafx-led-number-display'. The description reads: 'A JavaFX component which provides a LED number display as used in clocks and electronics projects' and 'Last Release on Oct 1, 2019'. At the bottom right of the results area, there are navigation buttons: 'Prev', '1', and 'Next'.

### JavaFX LED number display on Maven repository

# Install Maven

## On Windows PC

- First install an up-to-date version of Java
- Download the ZIP from the [Apache Maven website](https://maven.apache.org/download.cgi)<sup>60</sup>
- Unzip and move the directory to e.g. “C:\Program Files\apache-maven”
- Right-click on “My PC” in explorer and select “Properties” > “Advanced system settings” > “Environment variables”



Setting Environment variable in Windows

- Add the location of the “Maven” > “bin” directory to the “User variables” > “Path” settings
- Click “OK” till all boxes are closed
- Sign out and in again (or restart)
- Check the version in cmd

```

1 $ mvn -v
2 Apache Maven 3.6.2 (40f523331...; 2019-08-27T17:06:16+02:00)
3 Maven home: C:\Program Files\apache-maven-3.6.2\bin\..
4 Java version: 11.0.4, vendor: AdoptOpenJDK, runtime:
5   C:\Program Files\AdoptOpenJDK\jdk-11.0.4.11-hotspot
6 Default locale: en_US, platform encoding: Cp1252
7 OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"

```

## On Raspberry Pi

We only need to call the “apt” program with the correct parameters from the terminal to add Maven to our Pi:

<sup>60</sup><https://maven.apache.org/download.cgi>

```
1 $ sudo apt install maven
```

Check the installation by requesting the version:

```
1 $ mvn -v
2 Apache Maven 3.6.0
3 Maven home: /usr/share/maven
4 Java version: 13-BellSoft, vendor: BellSoft, runtime:
5   /usr/lib/jvm/bellsoft-java13-arm32-vfp-hflt
6 OS name: "linux", version: "4.19.66-v71+", arch: "arm", family: "unix"
```

## Generate a new Maven project

Let's use Maven to generate a new project which we can use as a starting point for every application we want to build. Run the following command in the terminal. You can change the groupId and artifactId to the name you would like to use.

```
1 mvn archetype:generate
2   -DgroupId=be.webtechie
3   -DartifactId=java-maven-minimal
4   -DarchetypeArtifactId=maven-archetype-quickstart
5   -DinteractiveMode=false
```



This script can also be found in:  
Chapter\_06\_Maven > scripts > start-new-maven-project.sh

## Project structure

As you can see in the next screenshot, the created Maven project has a clear structure. The pom.xml file is in the root directory. All code is in the directory structure src > main > java > be > webtechie (based on the artifactId). There is also a unit test file available in src > test > java > be > webtechie.

```
File Edit Selection View Go Debug Terminal Help App.java - java-maven-minimal - Visual Studio Code
EXPLORER
OPEN EDITORS
  App.java src/main/java/be/webtechie
JAVA-MAVEN-MINIMAL
  .vscode
  src
    main
      java
        be
          webtechie
            App.java
  test
    java
      be
        webtechie
          AppTest.java
  target
  pom.xml
App.java
src > main > java > be > webtechie > App.java > ...
1 package be.webtechie;
2
3 /**
4  * Hello world!
5  */
6
7 public class App
8 {
9     Run | Debug
10    public static void main( String[] args )
11    {
12        System.out.println( "Hello World!" );
13    }
14 }
```

Maven project structure



## A minimal pom.xml example

Let's take a look at the small example pom.xml file. All other available elements are described on [the Apache Maven guide](#)<sup>61</sup>. In the file generated by Maven itself, there is no info added, so below you can find a version with additional comments to describe the purpose of the elements.

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4   http://maven.apache.org/maven-v4_0_0.xsd">
5   <modelVersion>4.0.0</modelVersion>
6   <!--
7   The "project" element contains info about the layout of the document
8   and we leave it as is. The same goes for "modelVersion".
9   -->
10
11  <!--
12  Minimum required info about this project.
13  -->
14  <groupId>be.webtechie</groupId>
15  <artifactId>java-maven-minimal</artifactId>
16  <version>1.0-SNAPSHOT</version>
17
18  <!--
19  Info about the project.
20  -->
21  <name>java-maven-minimal</name>
22  <url>http://maven.apache.org</url>
23
24  <!--
25  How the application needs to be packaged for distribution by Maven.
26  -->
27  <packaging>jar</packaging>
28
29  <!--
30  Define extra dependencies needed in the application.
31  In this example the JUnit test library is added.
32  -->
33  <dependencies>
34    <dependency>
35      <groupId>junit</groupId>
```

---

<sup>61</sup><https://maven.apache.org/guides/introduction/introduction-to-the-pom.html>

```
36         <artifactId>junit</artifactId>
37         <version>3.8.1</version>
38         <scope>test</scope>
39     </dependency>
40 </dependencies>
41 </project>
```

## Maven pom-files used in this book

There are different ways to package your application into a jar-file. Depending on the fact if you want a minimal package size by providing the dependencies on the device itself, or want to include all the dependencies inside the jar so you don't need to worry about them. Each of these options has its use-case, with the size of the jar-file as an important factor.

For ease-of-use, in the examples in this book, we will include the dependencies into the jar-file. To do this, we will use “maven-assembly-plugin”. Keep in mind you will need to update the “mainClass” value to reference the class with your main-method. The value “be.webtechie.App” used here, is only an example.

```
1 <build>
2   <plugins>
3     <plugin>
4       <groupId>org.apache.maven.plugins</groupId>
5       <artifactId>maven-compiler-plugin</artifactId>
6       <version>3.8.0</version>
7       <configuration>
8         <release>11</release>
9       </configuration>
10    </plugin>
11
12    <plugin>
13      <artifactId>maven-assembly-plugin</artifactId>
14      <version>2.2.1</version>
15      <configuration>
16        <descriptorRefs>
17          <descriptorRef>jar-with-dependencies</descriptorRef>
18        </descriptorRefs>
19        <archive>
20          <manifest>
21            <addClasspath>true</addClasspath>
22            <mainClass>be.webtechie.App</mainClass>
23          </manifest>
24        </archive>
25      </configuration>
26      <executions>
27        <execution>
28          <id>make-assembly</id>
29          <phase>package</phase>
30          <goals>
31            <goal>single</goal>
```

```
32         </goals>
33     </execution>
34 </executions>
35 </plugin>
36 </plugins>
37 </build>
```

With these plugin included in the “pom.xml”-file, we can build the application with the following command in the terminal inside your application directory:

```
1 $ mvn clean package
```

Don't worry if this is not fully clear at this moment! In the GitHub sources of each example used in this book, you can find the correct pom-file and the generated jar-file you can run on the Pi. So you have the choice to generate the package yourself, or use the provided one. How to build and run each example is also described in the examples.

## Add application logging with Maven and log4j

In the examples in this book “System.out.println()” and “System.err.println()” are used to output log messages which can be used for debugging. These log messages are only available when you start your Java application from the terminal and are lost when you end your application.

For a better approach, use log4j which can be configured to, for example, both log into the console similar to “System.out” and to a file, so you can still access the log messages after the application has finished.



A simple sample application to illustrate the initialization and use of log4j logging can be found in:

Chapter\_06\_Maven > java-maven-logging

The pom-file needs to be extended with this Maven dependency:

```
1 <dependency>
2   <groupId>log4j</groupId>
3   <artifactId>log4j</artifactId>
4   <version>1.2.17</version>
5 </dependency>
```

The initialization and some example logging is done in “App.java”:

```
1 package be.webtechie;
2
3 import org.apache.log4j.ConsoleAppender;
4 import org.apache.log4j.Level;
5 import org.apache.log4j.Logger;
6 import org.apache.log4j.PatternLayout;
7 import org.apache.log4j.RollingFileAppender;
8
9 /**
10  * Hello world!
11  *
12  */
13 public class App {
14     private static Logger logger = Logger.getLogger(App.class);
15
16     public static void main( String[] args ) {
17         Logger.getRootLogger().getLoggerRepository().resetConfiguration();
18         initLog();
```

```
19
20     logger.debug("Debug message");
21     logger.info("Info message");
22     logger.warn("Warn message");
23     logger.error("Error message");
24 }
25
26 private static void initLog() {
27     PatternLayout logPattern =
28         new PatternLayout("%d{yyyyMMdd HH:mm:ss,SSS} | %-5p | [%c{1}] | %m%n");
29
30     // Log to the console, similar to System.out
31     ConsoleAppender console = new ConsoleAppender();
32     console.setName("ConsoleLogger");
33     console.setLayout(logPattern);
34     console.setThreshold(Level.DEBUG);
35     console.activateOptions();
36     Logger.getRootLogger().addAppender(console);
37
38     // Log to a file to store it for later reference,
39     // creating max 5 files of 10MB
40     RollingFileAppender file = new RollingFileAppender();
41     file.setName("FileLogger");
42     file.setFile("logs/app.log");
43     file.setLayout(logPattern);
44     file.setThreshold(Level.INFO);
45     file.setAppend(true);
46     file.activateOptions();
47     file.setMaxFileSize("10MB");
48     file.setMaxBackupIndex(5);
49     Logger.getRootLogger().addAppender(file);
50 }
51 }
```

Please check “[Java Logging](http://tutorials.jenkov.com/java-logging/index.html)” on [jenkov.com](http://jenkov.com)<sup>62</sup> for other and more detailed ways to configure the logger. Full log4j documentation is available on [logging.apache.org/log4j/1.2/apidocs](https://logging.apache.org/log4j/1.2/apidocs/)<sup>63</sup>.

When running this example application, the output will look like this:

---

<sup>62</sup><http://tutorials.jenkov.com/java-logging/index.html>

<sup>63</sup><https://logging.apache.org/log4j/1.2/apidocs/index.html>

```
1 log4j: reset attribute= "false".
2 log4j: Threshold ="null".
3 log4j: Level value for root is [debug].
4 log4j: root level set to DEBUG
5 log4j: Class name: [org.apache.log4j.RollingFileAppender]
6 log4j: Setting property [file] to [demoApplication.log].
7 log4j: Parsing layout of class: "org.apache.log4j.PatternLayout"
8 log4j: Setting property [conversionPattern] to
9     [%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n].
10 log4j: setFile called: demoApplication.log, true
11 log4j: setFile ended
12 log4j: Adding appender named [fileAppender] to category [root].
13 log4j: setFile called: /tmp/app.log, true
14 log4j: setFile ended
15 20200219 20:23:37,152 | DEBUG | [App] | Debug message
16 20200219 20:23:37,153 | INFO | [App] | Info message
17 20200219 20:23:37,154 | WARN | [App] | Warn message
18 20200219 20:23:37,154 | ERROR | [App] | Error message
19
20 Process finished with exit code 0
```



If you run an application in IntelliJ IDEA with this logging configuration and the [Grep Console plugin](#)<sup>64</sup>, the warn and error lines will be highlighted, which makes it very easy to debug your code!

A screenshot of the IntelliJ IDEA Run console. The console window is titled 'Run: App x'. It displays a list of log messages. The messages are: '20200219 20:23:37,152 | DEBUG | [App] | Debug message', '20200219 20:23:37,153 | INFO | [App] | Info message', '20200219 20:23:37,154 | WARN | [App] | Warn message', and '20200219 20:23:37,154 | ERROR | [App] | Error message'. The 'WARN' and 'ERROR' messages are highlighted in yellow. At the bottom, it says 'Process finished with exit code 0'. On the left side of the console, there are several icons: a play button, a pencil, a square, a double arrow pointing down, a camera, an up arrow, a refresh icon, a down arrow, and a search icon.

Highlighted log messages in IntelliJ IDEA

<sup>64</sup><https://plugins.jetbrains.com/plugin/7125-grep-console>

# Just a thought: Abbreviations

As a developer, there are some “important” abbreviations you need to know.

## YAGNI

“[You Aren’t Gonna Need It](#)”<sup>a</sup> is used to point out the fact **you don’t need to add code until you need it**. Sometimes it feels right to add some additional methods which could one day be useful. But as long as you didn’t finish and test the code you need, you cannot be sure what additional code you’ll need in the future. And **it doesn’t make sense to write code which may never be used at all**.

## KISS principle

At work (only?) I mention from time to time I love to KISS. But at work that means “[Keep It Simple, Stupid](#)”<sup>b</sup>. The goal of the KISS principle is to **keep your code as simple as possible**. Breaking complex methods in multiple smaller ones, is one of the easiest ways to achieve this. If your method has a name like “doThisAndThisAndReturnThat”, you already should notice you could break it into three smaller ones.

## POGE

Always keep the [Principle Of Good Enough](#)<sup>c</sup> in mind when you develop software. Users will go for anything which fulfills their needs, even if more advanced solutions are available. It’s **better to go for a quick good solution, than to keep working towards perfection**, that unreachable goal... At my CoderDojo club a boy is already working on the same Minecraft project for years, and it never gets finished as he keeps adding new features. That same problem exists in many software teams. Deliver fast, fail fast, improve even faster and never forget: “[Perfect is the enemy of good](#)”<sup>d</sup>.

## NIH

Engineers tend to suffer the “[Not Invented Here](#)”<sup>e</sup> Syndrome. They keep reinventing the wheel while they often could benefit by adopting existing solutions. Often this syndrome is caused by the idea it is cheaper to do something yourself instead of paying for a license or hiring someone with experience in the matter. As a good developer, if you need to do something new, your first step should be to **investigate if there are existing frameworks, methodologies, program languages... which can solve your problem**.

## DRY

The “[Don’t repeat yourself](#)”<sup>f</sup> principle, aims to **divide your application into small blocks with a single responsibility**. Each block may only appear once in your system. Tools like [SonarLint](#)<sup>g</sup> which integrate with your IDE, will alert you if you have identical or similar code at multiple places. But you don’t always need a tool, if you feel you are repeating yourself, take a step back and look at how you can improve and simplify your code. If you aren’t following this approach, you end up with a “**WET solution**” aka “**Waste Everyone’s Time**”.



## TDD

“**Test Driven Development**”<sup>h</sup> is a way of coding, in which you **first write the minimal test for a new feature in your application**. As soon as your test succeeds you can refactor and improve your implementation. If you need more functionality, you first extend the test which has to turn green by adding the required code, and all previous tests should stay green.

Other DD-principles are “**Behavior Driven Development**”<sup>i</sup> and “**Domain Driven Design**”<sup>j</sup>.

BDD focusses on creating applications by a team in which each member has a different role (developer, tester, manager...) and fully understands the functionality to be implemented.

DDD is driven by a “domain expert” who can explain how the system should work which needs to be implemented and keeps an overview of the full development. Different subsystems can be developed but share the same “domain”. For instance, if you would design a software system for a shop you would have a database that stores the data, next to a program where clients and orders can be managed by two teams but share some functions for after-sales support. The domain expert has an overview of all the use cases and can guide the development team in the best approach.

While most teams aim to use a healthy mix of these and other principles, still “**Deadline Driven Development**” is the real-world approach in a lot of companies...

## RTFM

“**Read The Fucking Manual**”<sup>k</sup> is often used by a developer who gets frustrated by users asking the same question over and over again. But in my opinion, the user is always right. This means, if they keep asking how something works, you probably didn’t make it easy enough or the user is using it differently than what you assumed would happen.

That’s why you need to “**fail fast and often**”, meaning you need to **involve the end-user in your development process as soon as possible** so they can start using or testing it and give you feedback.

So next time you want to use “RTFM”, listen carefully to the question and what the user wants to do and why he doesn’t succeed...

<sup>h</sup>[https://en.wikipedia.org/wiki/You\\_aren%27t\\_gonna\\_need\\_it](https://en.wikipedia.org/wiki/You_aren%27t_gonna_need_it)

<sup>i</sup>[https://en.wikipedia.org/wiki/KISS\\_principle](https://en.wikipedia.org/wiki/KISS_principle)

<sup>j</sup>[https://en.wikipedia.org/wiki/Principle\\_of\\_good\\_enough](https://en.wikipedia.org/wiki/Principle_of_good_enough)

<sup>k</sup>[https://en.wikipedia.org/wiki/Perfect\\_is\\_the\\_enemy\\_of\\_good](https://en.wikipedia.org/wiki/Perfect_is_the_enemy_of_good)

<sup>l</sup>[https://en.wikipedia.org/wiki/Not\\_invented\\_here](https://en.wikipedia.org/wiki/Not_invented_here)

<sup>m</sup>[https://en.wikipedia.org/wiki/Don%27t\\_repeat\\_yourself](https://en.wikipedia.org/wiki/Don%27t_repeat_yourself)

<sup>n</sup><https://www.sonarlint.org/>

<sup>o</sup>[https://en.wikipedia.org/wiki/Test-driven\\_development](https://en.wikipedia.org/wiki/Test-driven_development)

<sup>p</sup>[https://en.wikipedia.org/wiki/Behavior-driven\\_development](https://en.wikipedia.org/wiki/Behavior-driven_development)

<sup>q</sup>[https://en.wikipedia.org/wiki/Domain-driven\\_design](https://en.wikipedia.org/wiki/Domain-driven_design)

<sup>r</sup><https://en.wikipedia.org/wiki/RTFM>

# Chapter 7: About JavaFX

From the Oracle site: “JavaFX is a set of graphics and media packages that enables developers to design, create, test, debug, and deploy rich client applications that operate consistently across diverse platforms.”

JavaFX was created by Sun Microsystems, later became part of Oracle Corporation and is now an open-source project on [openjfx.io](https://openjfx.io)<sup>65</sup> for which commercial support is available via [GluonHQ](https://gluonhq.com)<sup>66</sup>.

The sources are available on [GitHub](https://github.com)<sup>67</sup>.

JavaFX allows you to **quickly build a user interface while using the language and tools you already know as a Java developer**. This can be styled with a CSS syntax and is very flexible and extendable, or you can draw things with code as I’ve done in “Chapter 03: Java” for the timelines.

There are a lot of projects going on to make it very developer-friendly to be able to publish your application easily to different operating systems and platforms. Keep an eye on the news messages by [GluonHQ](https://gluonhq.com) and [GraalVM](https://www.graalvm.org)<sup>68</sup> who are working together on these solutions.

---

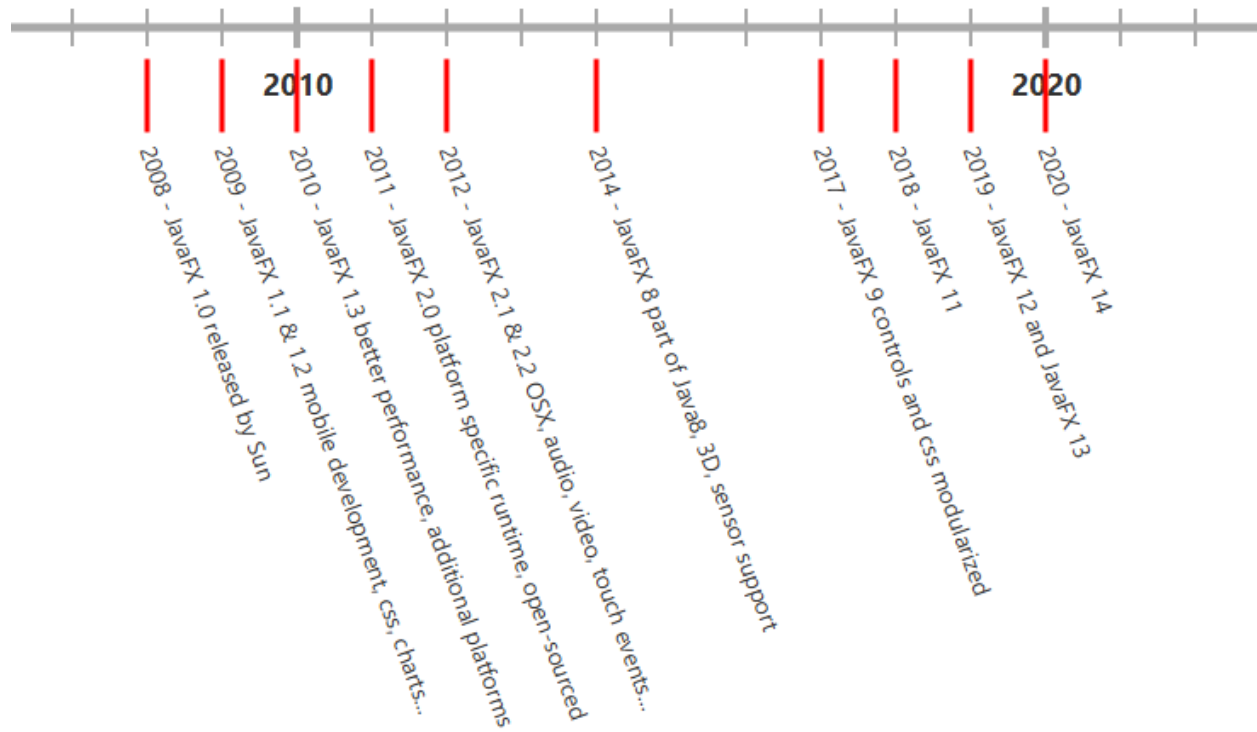
<sup>65</sup><https://openjfx.io/>

<sup>66</sup><https://gluonhq.com/>

<sup>67</sup><https://github.com/openjdk/jfx>

<sup>68</sup><https://www.graalvm.org/>

## History



Timeline of JavaFX

JavaFX was initially (version 1) a platform to develop Rich Internet Applications (RIA) to make it easy to create web-based interfaces with a scripting language.

After Oracle acquired Java and JavaFX from Sun Microsystems in 2009, it announced a new major version 2 in 2010. This was released in October 2011 and changed from a scripting language to a Java API.

The version numbering jumped to 8 when Java 8 itself introduced a lot of new APIs and language enhancements and JavaFX got improved to be in line with all these changes.

Since version 11 the JavaFX module moved out of the JDK and GluonHQ is maintaining openJFX with the support of many contributors (also still Oracle). Just like the new 6-month release cycle of Java, JavaFX development speed has increased pushing new features forward.

## Interview with Johan Vos



Johan Vos, [@johanvos<sup>a</sup>](#), Co-founder and CTO of [Gluon<sup>b</sup>](#), Java Champion, Leading the development of OpenJFX and OpenJDK for Mobile

*You are one of the main contributors to JavaFX (OpenJFX). What are the biggest changes ongoing in this framework?*

The most important criterium for the development of OpenJFX is stability. OpenJFX itself is not creating cool new visualization effects, but we mainly allow others (libraries, frameworks, projects) to leverage the JavaFX APIs and build added-value on top of it.

OpenJFX provides the top-level JavaFX APIs that allow developers to create UI applications and tools in Java. We keep working on those APIs, mainly enhancing the quality, and making sure they stay relevant in the evolving landscape.

Apart from the APIs, OpenJFX bridges the gap between those APIs and the low-level graphical systems on the different configurations. We make sure the APIs keep working first class on the underlying systems. This is a task that should not be underestimated.

In the Java world, and especially in OpenJDK projects, we are very cautious about adding functionality. One of the first questions we ask is: “Will this functionality still be relevant and maintained in 5 years from now?” We also have to make sure there is a broad consensus in the developer community to do something. Since JavaFX is widely used in very different areas, it is not always easy to come up with a set of APIs that are equally applicable in all domains. Hence, before adding or modifying APIs, we need lots of feedback.

Fortunately, there is a strong JavaFX ecosystem, with many small and large contributors that are very open and constructively discussing the future path.

*Do you believe JavaFX is the perfect match for user interfaces on the Raspberry Pi?*

Absolutely. The cross-platform character of Java makes it very easy to develop UI’s on a desktop system, and once you’re happy with it, you can run it on a Raspberry Pi and tune embedded-specific issues.

Especially with the [GraalVM native-image capabilities<sup>c</sup>](#), offered via the [Gluon Substrate plugin<sup>d</sup>](#), applications created with JavaFX will be very relevant.

*BellSoft announced in a blog post, support for video in JavaFX on the Pi. Can we expect more JavaFX features on the Raspberry Pi, like WebView support?*

Video support has been part of JavaFX since the beginning, as is WebView. Depending on the configuration and how you build, WebView might not be included in an embedded platform as an embedded component, but we rather use the existing webkit implementation on the device (which is what we do on iOS/Android for example). In general, everything that is available on the desktop is available as well on embedded and mobile.

*In your opinion, what is the most important change within the Java/JavaFX ecosystem in recent years?*

The decoupling of the JDK was an important change with a big impact. We knew this would shock several developers, but with the JDK and the ecosystem going modular, we had to do this with OpenJFX as well. We are now in pretty smooth waters, where the majority of the ecosystem embraced the modular approach and is using existing, familiar tools like Maven or Gradle to create JavaFX applications. For those developers and companies that prefer a separate SDK, we still offer that as well.

*Why is now the perfect time to learn Java?*

With Java, you learn a language that doesn't lock you in a particular market, area, or even language itself. Many languages are using the Java VM, and Java is used in a wide range of domains, in open-source and commercial software.

It is very mature, stable, and still very innovative, e.g. with the 6-month release cycles.

*Which DIY-programming-electronics-project are you working on, or is on your "if I ever have time" list?*

Alarm system for home, including AI and face recognition.

Automotive interface.

---

<sup>1</sup><https://twitter.com/johanvos>

<sup>2</sup><https://gluonhq.com/>

<sup>3</sup><https://www.graalvm.org/>

<sup>4</sup><https://github.com/gluonhq/substrate>

## Sample libraries to extend JavaFX

A lot of extra libraries are also available to extend JavaFX further to build very powerful UIs (user interfaces). A shortlist with some of them, but there are many more, check this project by [Hossein Rimazfor](#)<sup>69</sup> for a long list<sup>70</sup>!

### TilesFX

This great library is developed by [Gerrit Grunwald](#)<sup>71</sup> and provides tiles to build dashboards. Check [his blog](#)<sup>72</sup> for many more JavaFX examples!

You can find [the sources on GitHub](#)<sup>73</sup> and it is available as a [Maven dependency](#)<sup>74</sup>.



TilesFX example application

### FXRibbon

Inspired by the Microsoft Ribbon, this library developed by [Pedro Duque](#)<sup>75</sup>, provides a similar component in JavaFX. You can use this control to simplify the UI of applications with a significant number of actions.

<sup>69</sup><https://twitter.com/mhrimaz>

<sup>70</sup><https://github.com/mhrimaz/AwesomeJavaFX>

<sup>71</sup>[https://twitter.com/hansolo\\_](https://twitter.com/hansolo_)

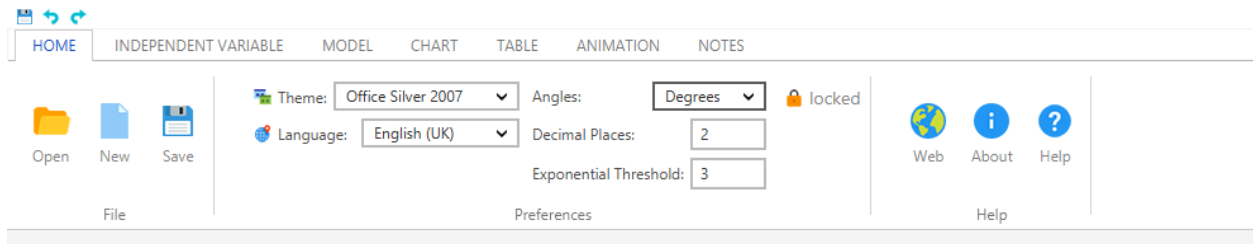
<sup>72</sup><https://harmoniccode.blogspot.com/>

<sup>73</sup><https://github.com/HanSolo/tilesfx>

<sup>74</sup><https://mvnrepository.com/artifact/eu.hansolo/tilesfx>

<sup>75</sup>[https://twitter.com/P\\_Duke](https://twitter.com/P_Duke)

You can find [the sources on GitHub](#)<sup>76</sup> and it is available as a [Maven dependency](#)<sup>77</sup>.

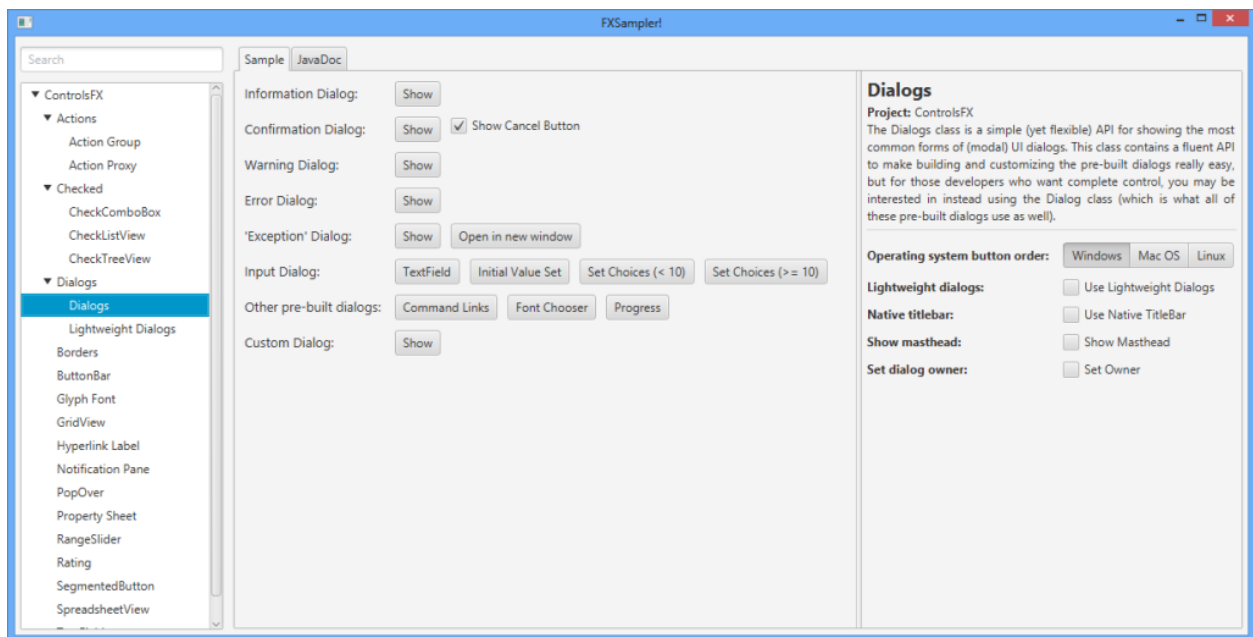


FXRibbon example application

## ControlsFX

Project by [Jonathan Giles](#)<sup>78</sup>.

You can find [the sources on GitHub](#)<sup>79</sup> and it is available as a [Maven dependency](#)<sup>80</sup>.



ControlsFX example application

## PickerFX

Project by one of those other JavaFX heroes [Dirk Lemmermann](#)<sup>81</sup>. He has many others, so check out his full [GitHub repo!](#)

<sup>76</sup><https://github.com/duke/FXRibbon>

<sup>77</sup><https://mvnrepository.com/artifact/com.pixelduke/FXRibbon>

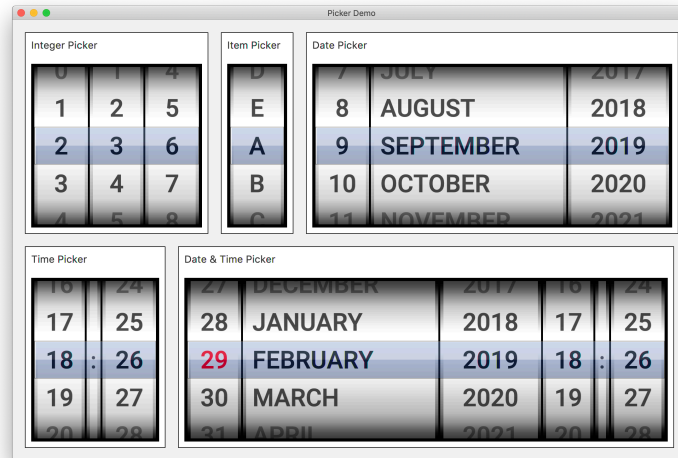
<sup>78</sup><https://twitter.com/JonathanGiles>

<sup>79</sup><https://github.com/controlsfx/controlsfx>

<sup>80</sup><https://mvnrepository.com/artifact/org.controlsfx/controlsfx>

<sup>81</sup><https://twitter.com/dlemmermann>

You can find [the sources on GitHub](#)<sup>82</sup> and it is available as a [Maven dependency](#)<sup>83</sup>. Make sure to also check [his blog with more awesome JavaFX stuff like JMetro](#)<sup>84</sup>!



**PickerFX example application**

<sup>82</sup><https://github.com/dlsc-software-consulting-gmbh/PickerFX>

<sup>83</sup><https://mvnrepository.com/artifact/com.dlsc.pickerfx/pickerfx>

<sup>84</sup><https://pixelduke.com/>



## Interview with Gerrit Grunwald



**Gerrit Grunwald, the Java Champion with the coolest Twitter name: [@hansolo](#)<sup>a</sup>, loves Java(FX) and IoT, UI guy**

*Why do you build such a nice framework as TilesFX as open-source and give it “for free” to everyone.*

I worked on a project where a dashboard for the management was needed. For this dashboard, I’ve created two different widgets. Once I started creating these widgets I realized that there was no good library to create dashboards in Java and at that moment TilesFX was born. I just added everything that came up my mind

and added it to the library. The funny thing is that I only used it for a little side project but never in any official project. But it seems a lot of people use it for their stuff.

*In your opinion, what is the most important change within the Java ecosystem in recent years?*

Well the most important change for me was the movement to the modularized JDK... which does not mean that I’m a big fan of it. It might be useful for some companies but I guess for many people the old structure would also still work. Of course, it helps in the world of containerized applications in combination with GraalVM (which is another important change in the Java ecosystem).

*What do you see as the next evolutions in the Java-world?*

I would love to see [Project Panama](#)<sup>b</sup> become reality soon to be able to make accessing native libs easier<sup>c</sup>.

*Why is now the perfect time to learn Java?*

Java still is one of the most widely used languages in the field. For me this is reason enough to recommend it to people when they ask me which programming language to learn. The other important thing is the huge number of libraries available and the strong community. Also, Java has shown that it was capable of “staying young” over all the years while evolving.



*Which DIY-programming-electronics-project are you working on, or is on your “if I ever have time” list?*

I’m working on a little gadget that helps my wife and myself to monitor the diabetes of our son. This is mainly based on ESP8266 stuff but for this, I also created a TileSkin for TilesFX :)

<sup>a</sup>[https://twitter.com/hansolo\\_](https://twitter.com/hansolo_)

<sup>b</sup><https://openjdk.java.net/projects/panama/>

<sup>c</sup>Project Panama aims to improve and enrich the connections between the Java virtual machine and well-defined but “foreign” (non-Java) APIs, including many interfaces commonly used by C programmers.

## Minimal JavaFX 11 sample application

GluonHQ has been working on a system to quickly get started with an “empty” JavaFX application with some plugins, in a few easy steps.

### Add new archetypes to Maven

First, we need to add some archetypes to our Maven. Open your terminal or cmd box and perform these commands:

```
1 $ git clone https://github.com/openjfx/javafx-maven-archetypes.git
2 $ cd javafx-maven-archetypes
3 $ mvn clean install
```



This script is available in:  
Chapter\_07\_JavaFX > scripts > gluonhq\_add\_archetypes.sh

The last step will take a while, but finally, you should have this in the log:

```
1 [INFO] -----
2 [INFO] Reactor Summary for Maven Archetypes for JavaFX 0.0.2-SNAPSHOT:
3 [INFO]
4 [INFO] Maven Archetypes for JavaFX ..... SUCCESS [ 1.883 s]
5 [INFO] Simple JavaFX Maven Archetype ..... SUCCESS [ 8.548 s]
6 [INFO] FXML JavaFX Maven Archetype ..... SUCCESS [ 0.304 s]
7 [INFO] -----
8 [INFO] BUILD SUCCESS
9 [INFO] -----
```

### Creating an empty application

In the terminal or cmd box you can now create a basic JavaFX project with the following command:

```
1 $ mvn archetype:generate
2     -DarchetypeGroupId=org.openjfx
3     -DarchetypeArtifactId=javafx-archetype-simple
4     -DarchetypeVersion=0.0.1
5     -DgroupId=be.webtechie
6     -DartifactId=javafx-minimal
7     -Dversion=0.0.1
```



This script is available in:  
Chapter\_07\_JavaFX > scripts > gluonhq\_create\_empty\_javafx\_project.sh

You can define the values for groupId, artifactId and version yourself, the given values above are just examples.

After running the above script a new Java Maven project is created for you with a pom-file and the first Java source files.



The project created with the above command is available in the sources in the directory:  
Chapter\_07\_JavaFX > javafx-minimal

## Running the empty application from Visual Studio Code

Make sure you are working with Java 11 JDK (or higher) on your development machine

```
1 $ java -version
2 openjdk version "11.0.4" 2019-07-16
3 OpenJDK Runtime Environment AdoptOpenJDK (build 11.0.4+11)
4 OpenJDK 64-Bit Server VM AdoptOpenJDK (build 11.0.4+11, mixed mode)
```

You can start the application with the “Run|Debug” popup from within Visual Studio Code.

```

src > main > java > be > webtechie > App.java > App > main(String[])
10  /**
11  */
12  public class App extends Application {
13
14      @Override
15      public void start(Stage stage) {
16          var javaVersion = SystemInfo.javaVersion();
17          var javafxVersion = SystemInfo.javafxVersion();
18
19          var label = new Label("Hello, JavaFX " + javafxVersion + ", running on Java " + javaVersion + ".");
20          var scene = new Scene(new StackPane(label), 640, 480);
21          stage.setScene(scene);
22          stage.show();
23      }
24
25      public static void main(String[] args) {
26          launch();
27      }
28
29  }

```

Starting the JavaFX application in Visual Studio Code

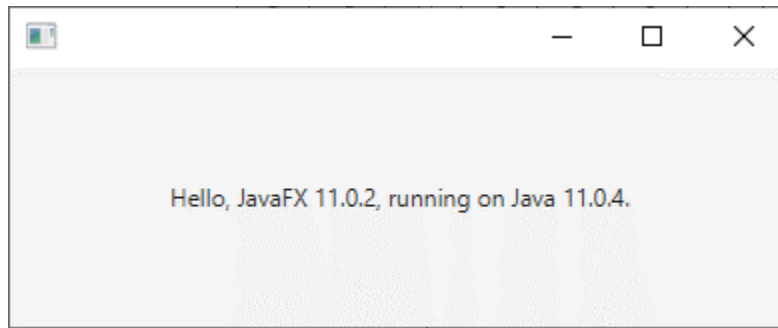
Or you can do the same with the Maven command “mvn javafx:run”:

```

1  PS D:\...\javafx-minimal> mvn javafx:run
2
3  [INFO] Scanning for projects...
4  [INFO]
5  [INFO] -----< be.webtechie:javapirecipes-javafx-minimal >-----
6  [INFO] Building javafx-minimal 0.0.1
7  [INFO] -----[ jar ]-----
8  [INFO]
9  [INFO] --- javafx-maven-plugin:0.0.1:run (default-cli) @ javafx-minimal ---
10 [INFO] -----
11 [INFO] BUILD SUCCESS
12 [INFO] -----
13 [INFO] Total time: 14.769 s
14 [INFO] -----

```

And the running application will appear on your screen:



Screenshot of the minimal JavaFX application

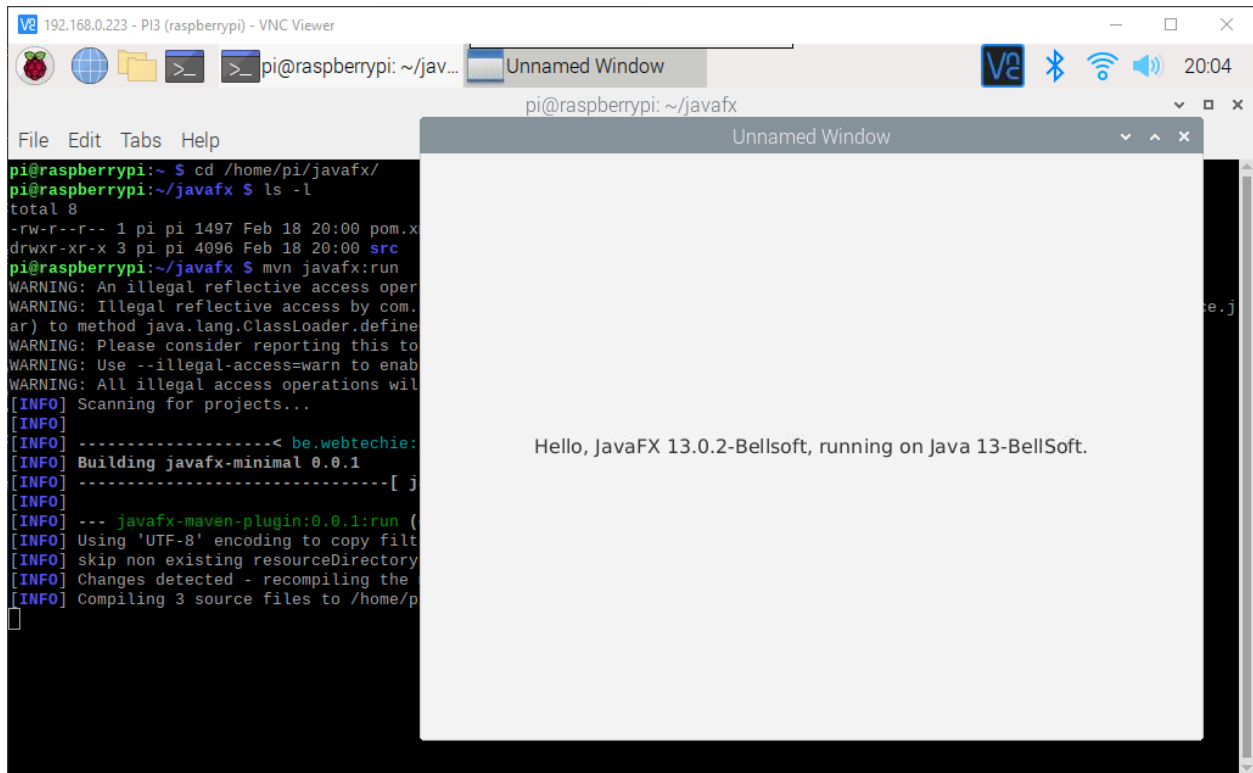
In the application, the version of your installed Java JDK and JavaFX are displayed.

## Running the application on the Pi

Just like you can run the application on the PC, you can do the same on a Raspberry Pi. Copy the project directory to the Pi (via SSH, see “Chapter 2: Tools > Remote connection to a Raspberry Pi (SSH)”), for example in “/home/pi/javafx”, or download the project from the GitHub sources.

In the terminal, we can now run this project with a few commands.

```
1 pi@raspberrypi:~ $ cd /home/pi/javafx/
2 pi@raspberrypi:~/javafx $ ls -l
3 total 8
4 -rw-r--r-- 1 pi pi 1497 Feb 18 20:00 pom.xml
5 drwxr-xr-x 3 pi pi 4096 Feb 18 20:00 src
6 pi@raspberrypi:~/javafx $ mvn javafx:run
```



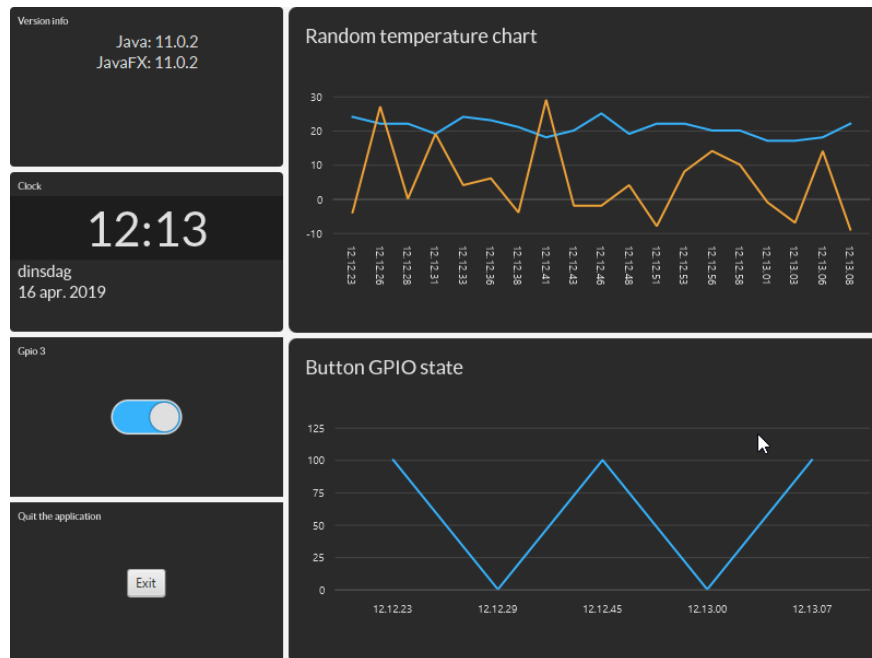
```
192.168.0.223 - Pi3 (raspberrypi) - VNC Viewer
pi@raspberrypi: ~/javafx
File Edit Tabs Help
pi@raspberrypi:~ $ cd /home/pi/javafx/
pi@raspberrypi:~/javafx $ ls -l
total 8
-rw-r--r-- 1 pi pi 1497 Feb 18 20:00 pom.xml
drwxr-xr-x 3 pi pi 4096 Feb 18 20:00 src
pi@raspberrypi:~/javafx $ mvn javafx:run
WARNING: An illegal reflective access operation detected
WARNING: Illegal reflective access by com.sun.javafx.fxml (unmodule class)
to method java.lang.ClassLoader.defineClass(java.lang.String,byte[],int,int,java.security.ProtectionDomain)
WARNING: Please consider reporting this to the vendor of the object
WARNING: Use --illegal-access=warn to enable warnings of this type.
WARNING: All illegal access operations will be denied in the future
[INFO] Scanning for projects...
[INFO] -----< be.webtechie:
[INFO] Building javafx-minimal 0.0.1
[INFO] -----[ j
[INFO] --- javafx-maven-plugin:0.0.1:run (
[INFO] Using 'UTF-8' encoding to copy filtered resources
[INFO] skip non existing resourceDirectory /home/pi/javafx/target
[INFO] Changes detected - recompiling the module
[INFO] Compiling 3 source files to /home/pi/javafx/target
Hello, JavaFX 13.0.2-Bellsoft, running on Java 13-BellSoft.
```

Running the minimal JavaFX application on Pi

## Example 1: TilesFX dashboard

Let's start with a first Java + JavaFX + GPIO example!

This is the application we are going to build: a dashboard with tiles (from TilesFX), an input from a push button and an output with an LED.



Screenshot of the running application



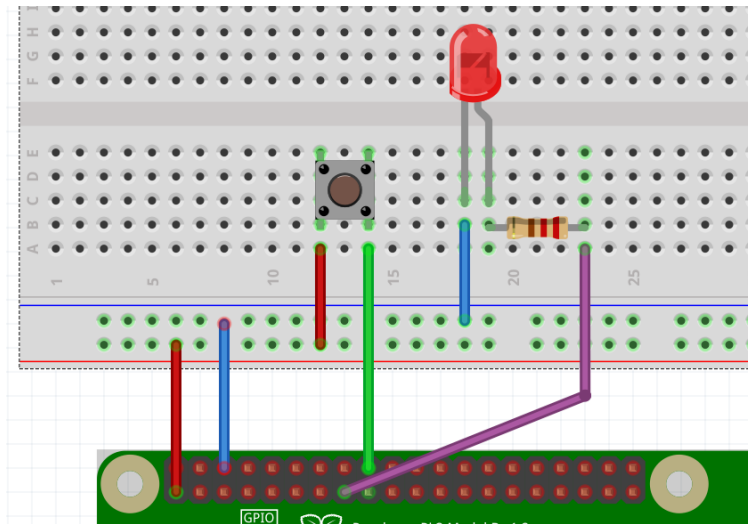
Chapter\_07\_JavaFX > javafx-dashboard

## Wiring and testing in terminal

Let's add some hardware to use some of the full power of the Pi. With some basic components we add an LED and a push-button, connected like this:

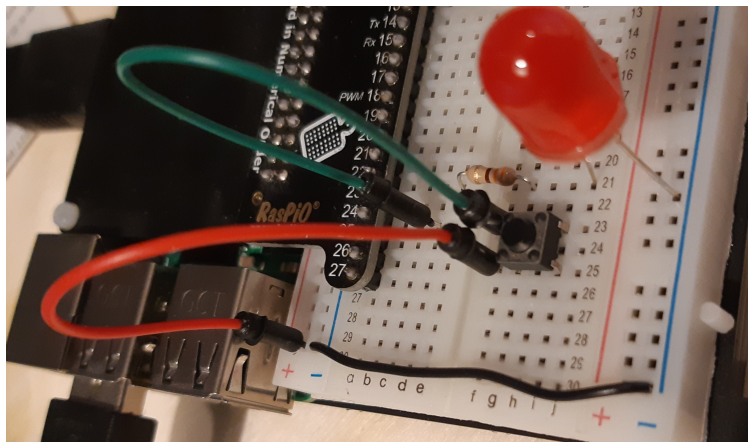
- GPIO22 (WiringPi n° 3) > resistor > LED > ground
- 3.3V > push button > GPIO24 (WiringPi n° 5)

To test if we connected the LED in the correct direction, we can plug in a cable between a 3.3V pin and GPIO22. If the LED doesn't turn on, we need to swap it.



Wiring for the JavaFX dashboard application

On my breadboard test setup it looks like this:



Wiring on a breadboard

To test the connections we will use “gpio” via the terminal. This program is part of Raspbian, so we don’t need to install anything to be able to do this. Make sure you are on the latest gpio-version if you are using a Raspberry Pi 4, see Chapter 5 > WiringPi number.

Turn LED on GPIO22 on (1) and off (0):

```
1 $ gpio mode 3 out
2 $ gpio write 3 1
3 $ gpio write 3 0
```

Read the button state (1 = pressed, 0 = not pressed):



```
1 $ gpio mode 5 in
2 $ gpio read 5
3 1
```

## Blink an LED with Java

This same thing can now be done with Java. Let's first try to do this directly on the Pi. We need to make a new file "HelloGpio.java" and add the following content.

```
1 $ nano HelloGpio.java
2
3 public class HelloGpio {
4     public static void main (String[] args) {
5         System.out.println("Hello Gpio");
6
7         try {
8             Runtime.getRuntime().exec("gpio mode 3 out");
9
10            var on = true;
11
12            for (var loopCounter = 0; loopCounter < 10; loopCounter++) {
13                System.out.println("Changing LED to " + (on ? "on" : "off"));
14                Runtime.getRuntime().exec("gpio write 3 " + (on ? "1" : "0"));
15
16                on = !on;
17
18                Thread.sleep(500);
19            }
20        } catch (Exception ex) {
21            System.err.println("Exception from Runtime: "
22                + ex.getMessage());
23        }
24    }
25 }
```

In this example code, we configure pin 3 to be an output-pin and toggle it 10 times between "on" and "off", with an interval of 500 milliseconds. Java requires us to catch possible exceptions, so that's why there is some extra try/catch code.

Make sure you are still in the same directory and run this code without compiling (because we use Java 11 or a newer version) with the following result:

```

1 $ ls -l
2 HelloGpio.java
3
4 $ java HelloGpio.java
5 Hello Gpio
6 Changing LED to on
7 Changing LED to off
8 Changing LED to on
9 ...

```

## Building our first JavaFX application

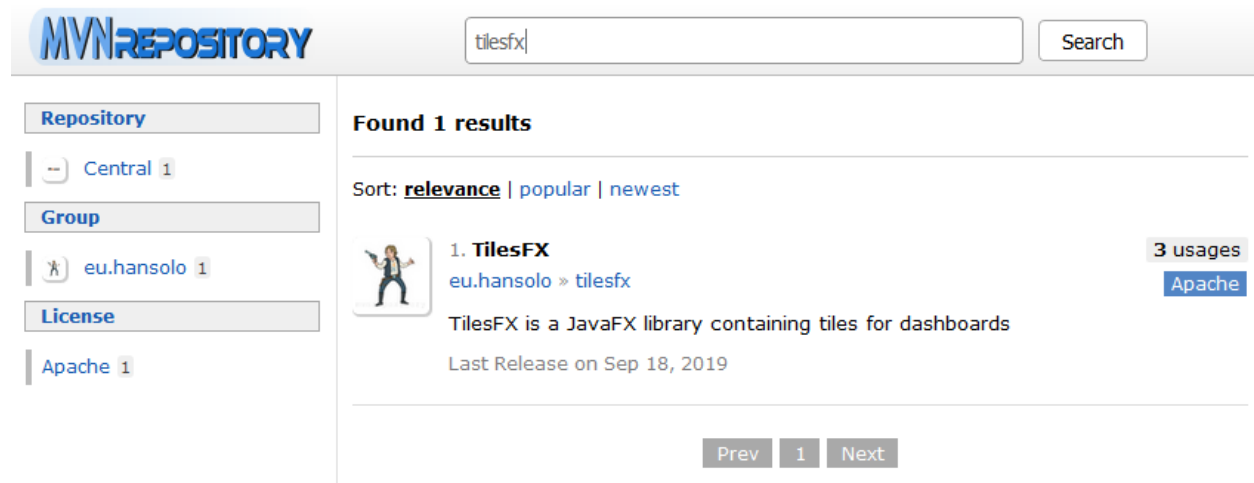
Running one Java-file is fun and enables us to quickly test something, but it's only a first step. To make a full Java-application, we need some additional steps...

Let's start by making a copy of the "javafx-minimal" application we created with the GluonHQ-tool (or take it from the examples GitHub of course).

### Extend the pom.xml

The application will use the TilesFX libraries to make a fancy dashboard next to the already used JavaFX library. We need to add the TilesFX dependency in the pom.xml file so our IDE will download it during development and we can call the methods of it.

Go to [the Maven repository website](https://mvnrepository.com/)<sup>85</sup> and search for TilesFX.



The screenshot shows the Maven Repository website interface. At the top, there is a search bar with the text 'tilesfx' entered and a 'Search' button. Below the search bar, there are filters for Repository (Central 1), Group (eu.hansolo 1), and License (Apache 1). The main content area displays 'Found 1 results' and a list of search results. The first result is '1. TilesFX' by 'eu.hansolo', with 3 usages and an Apache license. The description states: 'TilesFX is a JavaFX library containing tiles for dashboards' and 'Last Release on Sep 18, 2019'. Navigation buttons for 'Prev', '1', and 'Next' are visible at the bottom of the results list.

#### TileFX in the Maven repository

When you click through you can select the version and on its page, you will find the dependency to be added to the pom.xml file, where the javafx-controls dependency is already included. We also need to add the javafx-web dependency as this is used by TilesFX.

<sup>85</sup><https://mvnrepository.com/>

While we are working in the pom.xml file, we also change the artifactId as we copied this project from the javafx-minimal application and we add the build-plugins as described in “Chapter 6: Maven”.

```
1  ...
2  <artifactId>javafx-dashboard</artifactId>
3  ...
4  <dependencies>
5      <dependency>
6          <groupId>org.openjfx</groupId>
7          <artifactId>javafx-controls</artifactId>
8          <version>11.0.2</version>
9      </dependency>
10     <dependency>
11         <groupId>org.openjfx</groupId>
12         <artifactId>javafx-web</artifactId>
13         <version>11.0.2</version>
14     </dependency>
15     <dependency>
16         <groupId>eu.hansolo</groupId>
17         <artifactId>tilesfx</artifactId>
18         <version>11.13</version>
19     </dependency>
20 </dependencies>
21
22 <build>
23     <plugins>
24         <plugin>
25             <groupId>org.apache.maven.plugins</groupId>
26             <artifactId>maven-compiler-plugin</artifactId>
27             <version>3.8.0</version>
28             <configuration>
29                 <release>11</release>
30             </configuration>
31         </plugin>
32
33         <plugin>
34             <artifactId>maven-assembly-plugin</artifactId>
35             <version>2.2.1</version>
36             ...
37         </plugin>
38     </plugins>
39 </build>
```

40 ...

## Add a GPIO helper class

In HelloGpio.java we were able to turn an LED on and off with a small piece of code. But to be able to handle exceptions and read the state of a GPIO, we need some more code. We create a new class “Gpio.java”. This will use the “gpio” commands we used before in the terminal, but controlled with Java. This is just a first “quick-win” approach as in chapter 9 we will start using Pi4J which provides a better approach and many other methods, but at this moment we want to take a code-only approach.



This full class can be found in  
Chapter\_07\_JavaFX > javafx-dashboard > src > main > java > be > webtechie > Gpio.java

Let’s look at it, bit by bit.

We start with the package name and the imports we need. Within the class we will add the methods we need.



These are the last examples where the full code is documented within this book. Look into the GitHub sources of each example including package names and imports.

```

1 package be.webtechie;
2
3 import java.io.BufferedReader;
4 import java.io.IOException;
5 import java.io.InputStream;
6 import java.io.InputStreamReader;
7
8 /**
9  * Change a pin state using WiringPi and command line calls
10 * http://wiringpi.com/the-gpio-utility/
11 */
12 public class Gpio {
13
14 }
```

We need one big method to run the gpio commands and handle the result. E.g. if we run “gpio read 5” we want to know if “1” or “0” is returned by gpio. We call this the “execute” method. It is a private static method because it will only be used by other static methods inside Gpio.java. The return of this method is a String, so we can handle the value depending on the type of result we are expecting.

To avoid memory leaks where Java blocks memory which is no longer needed, the closing of the stream and reader is done in the finally block which gets called even if an error occurs.

```
1  /**
2  * Execute the given command, this is called by the public methods.
3  *
4  * @param cmd String command to be executed.
5  */
6  private static String execute(String cmd) {
7      Process p = null;
8      InputStream error = null;
9      BufferedReader input = null;
10
11     try {
12         // Get a process to be able to do native calls on the operating system.
13         // You can compare this to opening a terminal window and running a command.
14         p = Runtime.getRuntime().exec(cmd);
15
16         // Get the error stream of the process and print it
17         // so we will now if something goes wrong.
18         error = p.getErrorStream();
19         for (int i = 0; i < error.available(); i++) {
20             System.out.println("" + error.read());
21         }
22
23         // Get the output stream, this is the result of the command we give.
24         String line;
25         StringBuilder output = new StringBuilder();
26         input = new BufferedReader(new InputStreamReader(p.getInputStream()));
27         while ((line = input.readLine()) != null) {
28             output.append(line);
29         }
30         input.close();
31
32         System.out.println(cmd);
33
34         // Return the result of the command.
35         return output.toString();
36     } catch (IOException e) {
37         System.err.println(e.getMessage());
38
39         return "";
40     } finally {
41         if (p != null) {
42             p.destroy();
43         }
44     }
45 }
```

```
44
45     if (error != null) {
46         try {
47             error.close();
48         } catch (IOException ex) {
49             System.err.println("Error while closing the error stream");
50         }
51     }
52
53     if (input != null) {
54         try {
55             input.close();
56         } catch (IOException ex) {
57             System.err.println("Error while closing the input stream");
58         }
59     }
60 }
61 }
```



By using the “static” keyword we tell Java this method can be called without the need to create an instance of the class.

In short, we will be able to use this method from another class like this:

```
1 Gpio.initiatePin(5, "out");
```

Without the static keyword we need two lines as we need to create an instance of the class first:

```
1 Gpio gpio = new Gpio();
2 gpio.initiatePin(5, "out");
```

In this case, we can use static methods as we don’t need to define values used in all the methods of Gpio, or don’t need Gpio to do stuff first, before using one of its methods. In the next class, we will make “Fxsreen”, that works differently.

Now let’s use this “execute” to initiate a pin. For this, we need to set the mode and don’t need to handle the return. Again we define this as a static method.

```

1  /**
2  * Initialize the pin so it can be toggled later.
3  *
4  * @param pin The pin number according to the WiringPi numbering scheme
5  */
6  public static void initiatePin(final int pin, final String mode) {
7      execute("gpio mode " + pin + " " + mode);
8  }

```

Setting a pin to high or low also can be done without handling the return.

```

1  /**
2  * Set the state of the pin high or low.
3  *
4  * @param pin The pin number according to the WiringPi numbering scheme
5  * @param on True or False
6  */
7  public static void setPinState(final int pin, final boolean on) {
8      execute("gpio write " + pin + (on ? " 1" : " 0"));
9  }

```

Reading a pin state needs some extra handling, as the result needs to be parsed to return a boolean for true (=high) or false (=low).

```

1  /**
2  * Get the state of the pin.
3  *
4  * @param pin The pin number according to the WiringPi numbering scheme
5  * @return Flag if the pin is high (1 = true) or low (0 = false)
6  */
7  public static boolean getPinState(final int pin) {
8      final String result = execute("gpio read " + pin);
9
10     System.out.println("Getting pin state of " + pin + ", result: " + result);
11
12     return result.equals("1");
13 }

```

So in short, this class provides three static methods:

- `Gpio.initiatePin(final int pin, final String mode)`  
 \* This one needs to be called once for every pin we want to use as either an “in” or “out” pin.

- `Gpio.setPinState(final int pin, final boolean on)`  
\* This method allows us to turn a pin (LED) on (true) or off (false).
- `Gpio.getPinState(final int pin)`  
\* This is the method we need to call to read out a GIO state. It will return a boolean for the state of the pin: high (1 = true) or low (0 = false).

## TilesFX dashboard

When the application starts, we want it to load a dashboard screen. Thanks to TilesFX this can be created quite easily. We create a new class “FxScreen.java” with the following code.

Here we won’t be using static methods, as we want to initialize a lot of stuff when we use the class.



This full class can be found in  
Chapter\_07\_JavaFX > javafx-dashboard > src > main > java > be > webtechie > FxScreen.java

Again let’s look at it bit by bit.

The number of imports is bigger as we will be using more dependencies.

We are also using the “extends” keyword here, as we want our FxScreen class to be of the JavaFX-type “HBox”. This way, we can add visual components that are horizontally placed next to each other.

```

1  package be.webtechie;
2
3  import eu.hansolo.tilesfx.Tile.SkinType;
4  import eu.hansolo.tilesfx.TileBuilder;
5  import java.text.SimpleDateFormat;
6  import java.util.Date;
7  import java.util.Locale;
8  import java.util.Random;
9
10 import javafx.application.Platform;
11 import javafx.geometry.Pos;
12 import javafx.scene.chart.XYChart;
13 import javafx.scene.control.Button;
14 import javafx.scene.layout.HBox;
15 import javafx.scene.layout.VBox;
16
17 /**
18  * Helper class to create our graphical user interface.
19  */
20 public class FxScreen extends HBox {

```



```
21  
22 }
```

We also need some variables we will use in the methods later, so let's add them to the class.

```
1 public class FxScreen extends HBox {  
2     /**  
3      * The pins we are using in our example.  
4      */  
5     private static final int PIN_LED = 3;  
6     private static final int PIN_BUTTON = 5;  
7  
8     /**  
9      * Data series used on the charts.  
10    */  
11    private final XYChart.Series<String, Number> seriesTemperatureInside;  
12    private final XYChart.Series<String, Number> seriesTemperatureOutside;  
13    private final XYChart.Series<String, Number> seriesButton;  
14  
15    /**  
16     * Previous state of the button so we can update the chart when it changes.  
17     */  
18    private boolean buttonIsPressed = false;  
19  
20    /**  
21     * Flag to keep the threads running or stop them when close button is clicked.  
22     */  
23    private static boolean running = true;
```

Now let's add the constructor. This one gets called if you create a new instance of this class from another one. Later in the main class we will call this with:

```
1 new FxScreen();
```

Which will call this constructor and here we initialize the variables we will use later.

```

1  /**
2  * Constructor.
3  */
4  public FxScreen() {
5      // Initialize the pins
6      Gpio.initiatePin(PIN_LED, "out");
7      Gpio.initiatePin(PIN_BUTTON, "in");
8
9      // Setup the line chart data series
10     this.seriesTemperatureInside = new XYChart.Series();
11     this.seriesTemperatureInside.setName("Inside temperature");
12
13     this.seriesTemperatureOutside = new XYChart.Series();
14     this.seriesTemperatureOutside.setName("Outside temperature");
15
16     this.seriesButton = new XYChart.Series();
17     this.seriesButton.setName("Button pressed");
18
19     // Start thread which will generated dummy data
20     this.startDemoData();
21
22     // Start thread which read the button state
23     this.startButtonRead();
24
25     // Build the screen
26     this.buildScreen();
27 }

```

A good coding practice, is to keep your methods short and readable. That's why at the end we are calling other methods to start two threads and build up the screen. The first thread generates random temperature values every 2,5" to fill in one of the charts.

```

1  /**
2  * Thread to generate random test temperatures.
3  */
4  private void startDemoData() {
5      Thread t = new Thread(() -> {
6          while (running) {
7              var timeStamp = new SimpleDateFormat("HH.mm.ss").format(new Date());
8              this.seriesTemperatureInside.getData()
9                  .add(new XYChart.Data(timeStamp, this.randomNumber(17,25)));
10             this.seriesTemperatureOutside.getData()
11                 .add(new XYChart.Data(timeStamp, this.randomNumber(-10,30)));

```

```

12
13     try {
14         Thread.sleep(2500);
15     } catch (InterruptedException ex) {
16         System.err.println("Data thread got interrupted");
17     }
18 }
19 });
20
21 t.start();
22 }
23
24 /**
25  * Generate random number between the given limits.
26  *
27  * @param min
28  * @param max
29  * @return
30  */
31 private int randomNumber(int min, int max) {
32     Random rand = new Random();
33     return rand.nextInt((max - min) + 1) + min;
34 }

```

The second thread reads the button state every half second and updates the data for the second chart if the button state has changed. In the Pi4J chapter you can find code to achieve this with an event handler if you need immediate feedback on input changes instead of a check based on an interval as we use here.

```

1 /**
2  * Thread to read the button state.
3  */
4 private void startButtonRead() {
5     Thread t = new Thread(() -> {
6         while (running) {
7             var buttonPressed = Gpio.getPinState(PIN_BUTTON);
8
9             if (buttonIsPressed != buttonPressed) {
10                buttonIsPressed = buttonPressed;
11
12                var timeStamp = new SimpleDateFormat("HH.mm.ss")
13                    .format(new Date());
14                this.seriesButton.getData()

```

```

15         .add(new XYChart.Data(timestamp, buttonPressed ? 100 : 0));
16     }
17
18     try {
19         Thread.sleep(500);
20     } catch (InterruptedException ex) {
21         System.err.println("Data thread got interrupted");
22     }
23 }
24 });
25
26 t.start();
27 }

```

The only remaining big block is the creation of the screen itself. As we are using TilesFX, this is done by defining a bunch of different tiles. We want to have them in two different VBox's and these are added to the overall HBox.

```

1  /**
2  * Build the screen.
3  */
4  private void buildScreen() {
5      // Get the Java version info
6      final String javaVersion = System.getProperty("java.version");
7      final String javaFxVersion = System.getProperty("javafx.version");
8
9      // Define our local setting (used by the clock)
10     var locale = new Locale("nl", "be");
11
12     // Tile with the Java info
13     var textTile = TileBuilder.create()
14         .skinType(SkinType.TEXT)
15         .prefSize(250, 200)
16         .title("Version info")
17         .description("Java: " + javaVersion + "\nJavaFX: " + javaFxVersion)
18         .descriptionAlignment(Pos.TOP_CENTER)
19         .textVisible(true)
20         .build();
21
22     ...
23
24     // Tile with a switch button to turn our LED on or off
25     var ledSwitchTile = TileBuilder.create()

```

```
26         .skinType(SkinType.SWITCH)
27         .prefSize(250, 200)
28         .title("Gpio " + PIN_LED)
29         .roundedCorners(false)
30         .build();
31
32     ledSwitchTile.setOnSwitchReleased(e -> Gpio
33         .setPinState(PIN_LED, ledSwitchTile.isActive()));
34
35     // Tile with an exit button to end the application
36     var exitButton = new Button("Exit");
37     exitButton.setOnAction(e -> endApplication());
38
39     var exitTile = TileBuilder.create()
40         .skinType(SkinType.CUSTOM)
41         .prefSize(250, 200)
42         .title("Quit the application")
43         .graphic(exitButton)
44         .roundedCorners(false)
45         .build();
46
47     ...
48
49     // Line chart example which will get the button state from a thread
50     var buttonLineChartTile = TileBuilder.create()
51         .skinType(SkinType.SMOOTHED_CHART)
52         .prefSize(550, 400)
53         .title("Button GPIO state")
54         .smoothing(false)
55         .series(this.seriesButton)
56         .build();
57
58     var tilesColumn1 = new VBox(textTile, clockTile, ledSwitchTile, exitTile);
59     tilesColumn1.setMinWidth(250);
60     tilesColumn1.setSpacing(5);
61
62     var tilesColumn2 = new VBox(tempartureLineChartTile, buttonLineChartTile);
63     tilesColumn2.setSpacing(5);
64
65     this.getChildren().add(tilesColumn1);
66     this.getChildren().add(tilesColumn2);
67 }
68
```

```
69 /**
70  * Stop the threads and close the application.
71  */
72 private void endApplication() {
73     this.running = false;
74
75     Platform.exit();
76 }
```

## Start methods

To let the application start with the FxScreen.java we need to change the Main.java file which was part of javafx-minimal so it looks like this. “Platform.setImplicitExit(true);” is also added to make sure the application shuts down nicely.

```
1  package be.webtechie;
2
3  import javafx.application.Application;
4  import javafx.scene.Scene;
5  import javafx.stage.Stage;
6
7  /**
8   * JavaFX App
9   */
10 public class App extends Application {
11
12     @Override
13     public void start(Stage stage) {
14         Platform.setImplicitExit(true);
15
16         var scene = new Scene(new FxScreen(), 640, 480);
17         stage.setScene(scene);
18         stage.show();
19
20         // Make sure the application quits completely on close
21         stage.setOnCloseRequest(t -> {
22             Platform.exit();
23             System.exit(0);
24         });
25     }
26
27     public static void main(String[] args) {
28         launch();
29     }
30 }
```

```
29     }  
30 }
```

## Update module-info.java

Last small step! Add the TilesFx dependency in this file so it looks like this:

```
1 module be.webtechie {  
2     requires javafx.controls;  
3     requires eu.hansolo.tilesfx;  
4     exports be.webtechie;  
5 }
```

## Run the application on PC

Running the program in Visual Studio Code is very easy. When hovering over the main method, you will get the “Run|Debug” options from which you can start the application.

```

App.java x
src > main > java > be > webtechie > App.java > App > main(String[])
3  import javafx.application.Application;
4  import javafx.application.Platform;
5  import javafx.scene.Scene;
6  import javafx.stage.Stage;
7
8  /**
9   * JavaFX App
10  */
11  public class App extends Application {
12
13      @Override
14      public void start(Stage stage) {
15          Platform.setImplicitExit(true);
16
17          var scene = new Scene(new FxScreen(), 640, 480);
18          stage.setScene(scene);
19          stage.show();
20
21          // Make sure the application quits completely on close
22          stage.setOnCloseRequest(t -> {
23              Platform.exit();
24              System.exit(0);
25          });
26      }
27
28      public static void main(String[] args) {
29          launch();
30      }
31
32  }
  
```

Starting the application from Visual Studio Code

## Run the application on the Pi

### Package the jar for the Pi

By using Maven and pom.xml we can build our application in the terminal of our IDE.

```

1  $ mvn clean package
2  [INFO] Scanning for projects...
3  [INFO]
4  [INFO] -----< be.webtechie:javafx-dashboard >-----
5  [INFO] Building javafx-dashboard 0.0.1
6  [INFO] -----[ jar ]-----
7  [INFO]
8  [INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ javafx-dashboard ---
9  ...
  
```

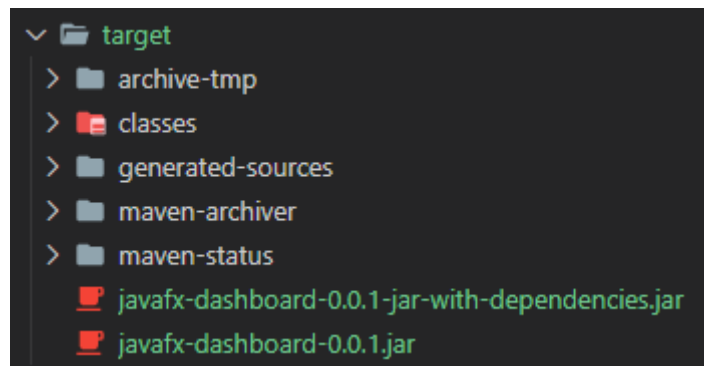


```

10 [INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ javafx-dashboard ---
11 [INFO] Building jar: D:\...\target\javafx-dashboard-0.0.1.jar
12 [INFO] -----
13 [INFO] BUILD SUCCESS
14 [INFO] -----
15 [INFO] Total time: 6.544 s
16 [INFO] -----

```

With “mvn clean package” the Java-files are converted to byte code class-files in the “target” directory of our project. The same process will also collect the compiled code into a jar-file and add the dependencies in “javafx-dashboard-0.0.1-jar-with-dependencies.jar” inside that target directory. This jar contains our full program and dependencies, so is about 65MB in size. This would not be ideal when we want to distribute our application, but is not an issue in our case, as we can just copy the file to the Pi through our local network if we are developing on PC.



The compiled JARs in the target directory

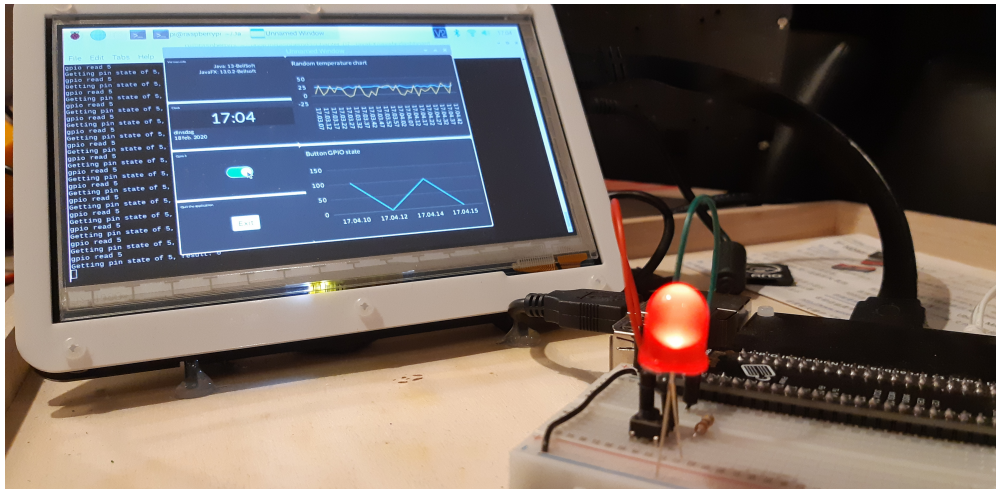
## Starting the application on the Pi

As the last step, we need to put the file “javafx-dashboard-0.0.1-jar-with-dependencies.jar” from the GitHub sources, or from the PC where you have built it yourself, on the Pi with SSH (see “Chapter 2: Tools > MobaXterm”). Or you build the jar on the Pi itself if you installed Maven on it.

Open the terminal, and go to the directory where the jar-file is located. Starting it is very easy when the Libarica JDK which includes JavaFX in installed as described in “Chapter 4: Java”:

```
1 $ java -jar javafx-dashboard-0.0.1-jar-with-dependencies.jar
```

First, some logging will be shown on the screen and a bit later the FX-screen is opened. The temperature chart will get a new random value every 2,5”.



Test setup with the running application

The LED can be toggled with the switch button on the screen in one of the tiles. The lower chart will get a new value every half second when the button is toggled.

## Conclusion

And that's it, our very first project using Java, JavaFX and the GPIOs!

Check the TilesFX documentation and go crazy with your own version of the dashboard screen...

## Start a Java application when the Pi starts up

In case you're building a dashboard-like application, as in the previous example, you'll probably want it to start automatically when your Raspberry Pi is powered. This can be done with a script that gets called after start-up.

1. Upload your jar with the application to the Pi, for example, in the directory “/home/pi/java-apps/”
2. Create a text file, for example, “start-script” and add the two lines to start the jar.

```
1 $ nano /home/pi/start-script
2
3 #! /usr/bin/bash
4 java -jar /home/pi/java-apps/my-app.jar
```

3. Test the script you created by running it manually.

```
1 $ bash /home/pi/start-script
```

4. Create a new file “startup.desktop” in “/etc/xdg/autostart/” to execute the script you created when the Pi starts.

```
1 $ sudo nano /etc/xdg/autostart/startup.desktop
2
3 [Desktop Entry]
4 Type=Application
5 Name=JavaApplication
6 Exec=bash /home/pi/start-script
```

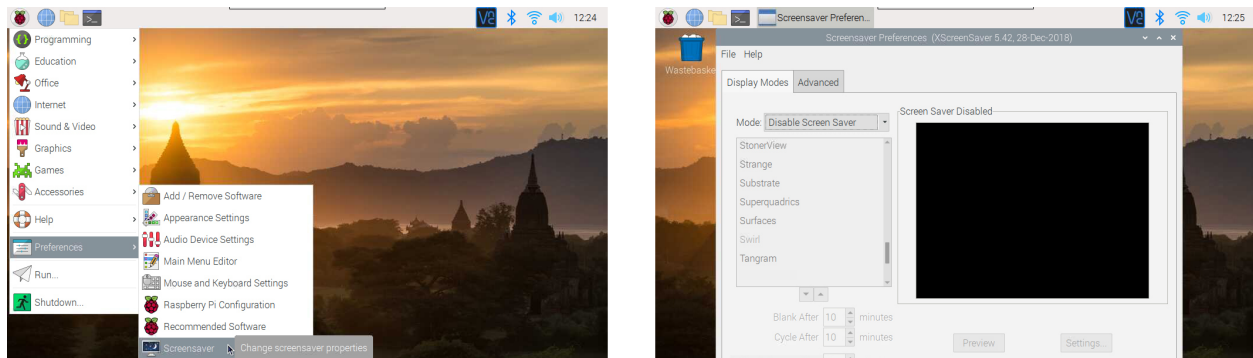
5. Restart the Pi and your application will be started automatically!

## Disable screensaver

When you build a dashboard-like application or, for example, a home automation interface, you maybe want to disable the screensaver so the screen is never put off. By default, there is no easy way to do this in Raspbian OS, but we can fix this easily by installing “xscreensaver”.

```
1 $ sudo apt install xscreensaver
```

After running this command, you have an extra program in “Preferences” with many options where you also can disable the screensaver completely.

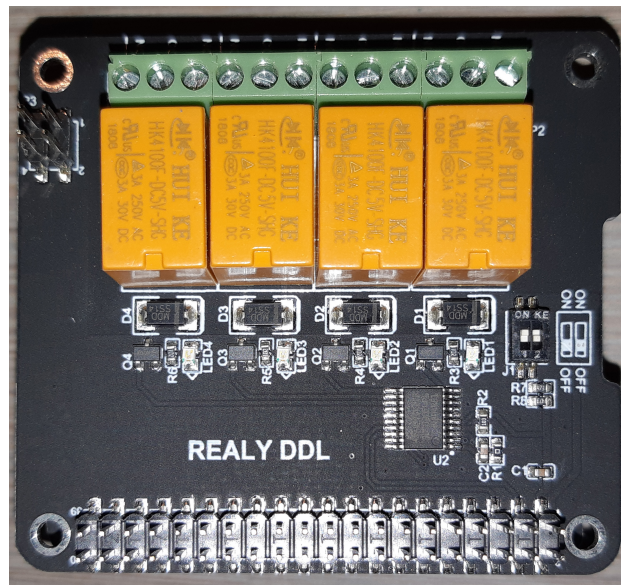


Screensaver settings screen

## Example 2: Interact with an I<sup>2</sup>C relay board

I<sup>2</sup>C (for more info see “Chapter 5: Raspberry Pi Pinning”) allows controlling different devices with only two pins if each one has a unique identifier.

For this example two “GeekPi Raspberry Pi Expansion Board 4 Channel Relay Board Modules”<sup>86</sup> are used. You can stack up to four of them on top of each other directly on the Pi. Two dip switches allow you to assign a different address (0x10, 0x11, 0x12 or 0x13) to each of the boards. This board has a clear [documentation page](#)<sup>87</sup> which is always important to be checked before you decide to buy a component!



Relay board

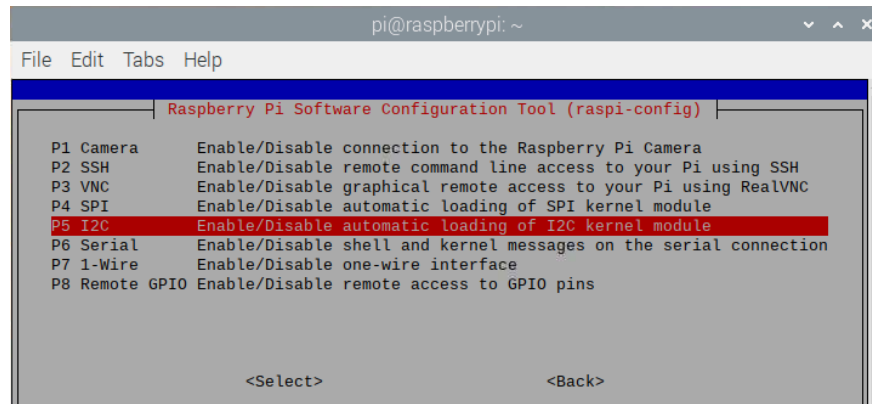
### Enable and test I<sup>2</sup>C

First, we need to enable I<sup>2</sup>C in “5 Interfacing Options” in raspi-config.

```
1 $ sudo raspi-config
```

<sup>86</sup>[https://www.amazon.com/gp/product/B07Q2P9D7K/ref=ppx\\_yo\\_dt\\_b\\_asin\\_title\\_o00\\_s00?ie=UTF8&psc=1](https://www.amazon.com/gp/product/B07Q2P9D7K/ref=ppx_yo_dt_b_asin_title_o00_s00?ie=UTF8&psc=1)

<sup>87</sup>[https://wiki.52pi.com/index.php/DockerPi\\_4\\_Channel\\_Relay\\_SKU:\\_EP-0099](https://wiki.52pi.com/index.php/DockerPi_4_Channel_Relay_SKU:_EP-0099)

Enable I<sup>2</sup>C in raspi-config

After it is enabled we can check which devices are detected. In this case, with two stacked boards you see addresses 0x10 and 0x11 are in use.

```

1 $ i2cdetect -y 1
2     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
3 00:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
4 10: 10 11  --  --  --  --  --  --  --  --  --  --  --  --  --
5 20:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
6 30:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
7 40:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
8 50:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
9 60:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
10 70:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --

```

We can check the relay state with “i2cget -y 1 ADDRESS RELAY” and toggle each relay in the terminal with the command “i2cset -y 1 ADDRESS RELAY STATE” where the state is 0x00 for off and 0xff for on.

```

1 $ i2cget -y 1 0x10 0x01
2 0x00
3 $ i2cset -y 1 0x10 0x01 0xff
4 $ i2cget -y 1 0x10 0x01
5 0xff

```

## Coding the I<sup>2</sup>C controller application



The code is separated in different small classes to make it easy to understand and extend. Please check the sources for the full code and structure. Check the pom.xml file for the correct dependencies and build plugins.

```
Chapter_07_JavaFX > javafx-i2c-relay
```

## Enums to make the code easier to use

To make it easy to define which boards are available and which relay we want to toggle in the application, three different enum classes are used.

The first one defines which boards are used. As you can stack four of them, but this example only uses two, those are the ones defined here with their address as configured with the dip switches.

```
1 public enum Board {
2     BOARD_1(0x10),
3     BOARD_2(0x11);
4
5     private final int address;
6
7     Board(int address) {
8         this.address = address;
9     }
10
11     public int getAddress() {
12         return this.address;
13     }
14 }
```

Each board has four relays, with each having a byte defining their channel.

```
1 public enum Relay {
2     RELAY_1((byte) 0x01),
3     RELAY_2((byte) 0x02),
4     RELAY_3((byte) 0x03),
5     RELAY_4((byte) 0x04);
6
7     private final byte channel;
8
9     Relay(byte channel) {
10         this.channel = channel;
11     }
12
13     public byte getChannel() {
14         return this.channel;
15     }
16 }
```

And finally, each relay has two states: on and off defined by a byte value.

```

1  public enum State {
2      STATE_ON((byte) 0xFF),
3      STATE_OFF((byte) 0x00);
4
5      private final byte value;
6
7      State(byte value) {
8          this.value = value;
9      }
10
11     public byte getValue() {
12         return this.value;
13     }
14 }

```

## I<sup>2</sup>C controller

To toggle a relay, we'll be using the same commands as we used before in the terminal, e.g. "i2cset -y 1 0x10 0x01 0xff".

Let's implement this in the class RelayController.java. We'll provide two methods to either toggle one relay on a specific board, or a list of relays on a list of boards.

```

1  /**
2   * Controller to set the relay states on different boards.
3   */
4  public class RelayController {
5      /**
6       * Set the state of all the relays on the boards.
7       *
8       * @param boards List of Enum Board
9       * @param relays List of Enum Relay
10      * @param state Enum State
11      */
12     public static void setRelays(List<Board> boards, List<Relay> relays,
13         State state) {
14         for (Board board : boards) {
15             for (Relay relay : relays) {
16                 setRelay(board, relay, state);
17             }
18         }
19     }
20 }

```



```

21  /**
22   * Set the state of the relay on the board.
23   *
24   * @param board Enum Board
25   * @param relay Enum Relay
26   * @param state Enum State
27   */
28  public static void setRelay(Board board, Relay relay, State state) {
29      String cmd = "i2cset -y 1"
30          + " " + String.format("0x%02X", board.getAddress())
31          + " " + String.format("0x%02X", relay.getChannel())
32          + " " + String.format("0x%02X", state.getValue());
33
34      execute(cmd);
35
36      System.out.println(relay + " on " + board + " set to " + state
37          + " with command: " + cmd);
38  }
39
40  private static void execute(String cmd) {
41      // Same as used in the JavaFX dashboard application.
42  }
43 }

```

## Toggle switch screen

The user interface contains eight ControlsFX ToggleSwitches which could be created eight times. But as “a good programmer, is a lazy programmer”, we use other methods to avoid duplicating the same code eight times.

```

1  /**
2   * Builder for a screen with two rows with four toggle switches.
3   */
4  public class ToggleSwitchScreen extends VBox {
5      public ToggleSwitchScreen() {
6          this.setSpacing(25);
7          this.setPadding(new Insets(25));
8
9          this.getChildren().add(this.createRow(Board.BOARD_1, 0));
10         this.getChildren().add(this.createRow(Board.BOARD_2, 4));
11
12         System.out.println("Toggle switch screen created");
13     }

```

```
14
15  /**
16   * Create a row with four toggle switches for the given board.
17   *
18   * @param board The board to be controlled
19   * @param offset Offset for the number showed in the label
20   * @return The created HBox
21   */
22  private HBox createRow(Board board, int offset) {
23      HBox row = new HBox();
24      row.setSpacing(25);
25
26      row.getChildren().add(this.createRelayToggleSwitch(
27          "Relay " + (offset + 1),
28          board, Relay.RELAY_1));
29      row.getChildren().add(this.createRelayToggleSwitch(
30          "Relay " + (offset + 2),
31          board, Relay.RELAY_2));
32      row.getChildren().add(this.createRelayToggleSwitch(
33          "Relay " + (offset + 3),
34          board, Relay.RELAY_3));
35      row.getChildren().add(this.createRelayToggleSwitch(
36          "Relay " + (offset + 4),
37          board, Relay.RELAY_4));
38
39      return row;
40  }
41
42  /**
43   * Create a ToggleSwitch which will call the RelayController on change.
44   *
45   * @param label Label for the toggle switch
46   * @param board The board to be controlled
47   * @param relay The relay on the board to be controlled
48   * @return The created ToggleSwitch
49   */
50  private ToggleSwitch createRelayToggleSwitch(String label,
51      Board board, Relay relay) {
52      ToggleSwitch toggleSwitch = new ToggleSwitch();
53      toggleSwitch.setText(label);
54      toggleSwitch.selectedProperty()
55          .addListener((observable, oldValue, selected) ->
56          RelayController.setRelay(board, relay,
```

```
57         selected ? State.STATE_ON : State.STATE_OFF));
58     return toggleSwitch;
59 }
60 }
```

## Main application class

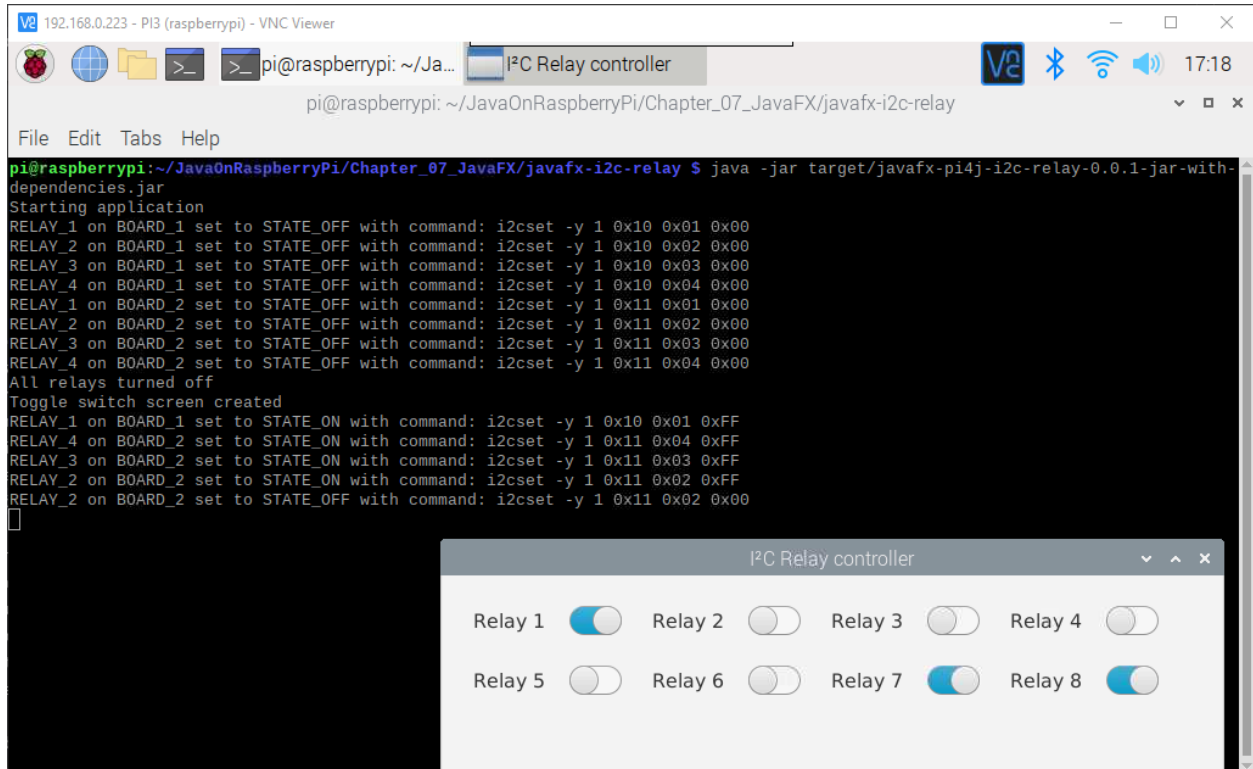
The main class could be very small, but here we added some additional code to make sure all relays are off when the application starts.

```
1  public class App extends Application {
2      @Override
3      public void start(Stage stage) {
4          System.out.println("Starting application");
5
6          // Set all relays out
7          RelayController.setRelays(
8              Arrays.asList(Board.BOARD_1, Board.BOARD_2),
9              Arrays.asList(Relay.RELAY_1, Relay.RELAY_2,
10                 Relay.RELAY_3, Relay.RELAY_4),
11              State.STATE_OFF);
12          System.out.println("All relays turned off");
13
14          var scene = new Scene(new ToggleSwitchScreen(), 640, 480);
15          stage.setScene(scene);
16          stage.setTitle("I2C Relay controller");
17          stage.show();
18      }
19
20     public static void main(String[] args) {
21         launch();
22     }
23 }
```

## Running the relay controller on the Pi

After building the application (on PC or Pi), we can start it from the terminal. When you run “mvn clean package” directly on the Pi via the terminal in the directory of this project, you need to tell Java it’s located in the “target” directory to start it.

- 1 \$ mvn clean package
- 2 \$ java -jar target/javafx-pi4j-i2c-relay-0.0.1-jar-with-dependencies.jar



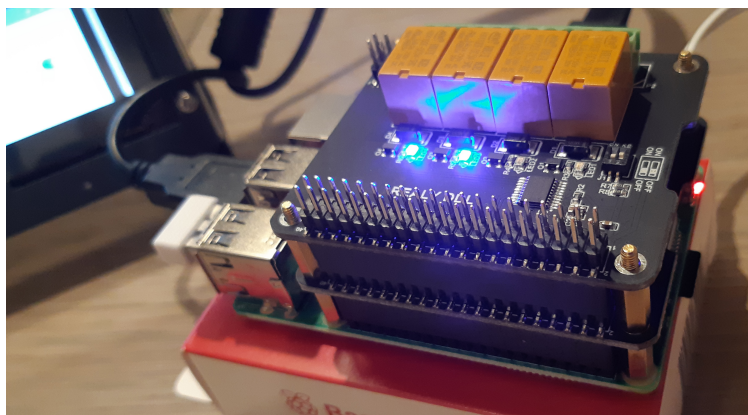
```

pi@raspberrypi: ~/JavaOnRaspberryPi/Chapter_07_JavaFX/javafx-i2c-relay
File Edit Tabs Help
pi@raspberrypi:~/JavaOnRaspberryPi/Chapter_07_JavaFX/javafx-i2c-relay $ java -jar target/javafx-pi4j-i2c-relay-0.0.1-jar-with-dependencies.jar
Starting application
RELAY_1 on BOARD_1 set to STATE_OFF with command: i2cset -y 1 0x10 0x01 0x00
RELAY_2 on BOARD_1 set to STATE_OFF with command: i2cset -y 1 0x10 0x02 0x00
RELAY_3 on BOARD_1 set to STATE_OFF with command: i2cset -y 1 0x10 0x03 0x00
RELAY_4 on BOARD_1 set to STATE_OFF with command: i2cset -y 1 0x10 0x04 0x00
RELAY_1 on BOARD_2 set to STATE_OFF with command: i2cset -y 1 0x11 0x01 0x00
RELAY_2 on BOARD_2 set to STATE_OFF with command: i2cset -y 1 0x11 0x02 0x00
RELAY_3 on BOARD_2 set to STATE_OFF with command: i2cset -y 1 0x11 0x03 0x00
RELAY_4 on BOARD_2 set to STATE_OFF with command: i2cset -y 1 0x11 0x04 0x00
All relays turned off
Toggle switch screen created
RELAY_1 on BOARD_1 set to STATE_ON with command: i2cset -y 1 0x10 0x01 0xFF
RELAY_4 on BOARD_2 set to STATE_ON with command: i2cset -y 1 0x11 0x04 0xFF
RELAY_3 on BOARD_2 set to STATE_ON with command: i2cset -y 1 0x11 0x03 0xFF
RELAY_2 on BOARD_2 set to STATE_ON with command: i2cset -y 1 0x11 0x02 0xFF
RELAY_2 on BOARD_2 set to STATE_OFF with command: i2cset -y 1 0x11 0x02 0x00

```

Running application on Pi with output in terminal

On the board, an LED indicates the enabled relays which makes it very easy to debug the link between the addresses you use in your application, and the toggled relays on the board.



Blue leds indicate the toggled relays

## Example 3: Build a UI with FXML

In the previous JavaFX example application and all the others in this book, a full code-only approach has been taken. This means all JavaFX layout is defined in code.

But JavaFX also provides a way of working where you can fully split the visual definition and the actions, by using FXML-files. These files are in XML format and only define how your interface needs to look, while the functionality is handled in code.

Let's look at an example again to explain this...

### Generate an empty FXML project as a starting point

We start by creating an empty project with the Maven archetype “javafx-archetype-fxml” which will prepare a starting-point application.

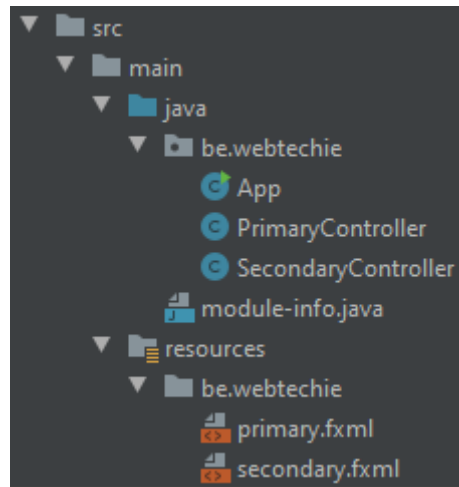
```
1 $ mvn archetype:generate
2     -DarchetypeGroupId=org.openjfx
3     -DarchetypeArtifactId=javafx-archetype-fxml
4     -DarchetypeVersion=0.0.1
5     -DgroupId=be.webtechie
6     -DartifactId=javafx-fxml-minimal
7     -Dversion=0.0.1
```



This script to create the JavaFX FXML start project is available in:  
Chapter\_07\_JavaFX > scripts > gluonhq\_create\_empty\_javafx\_fxml\_project.sh

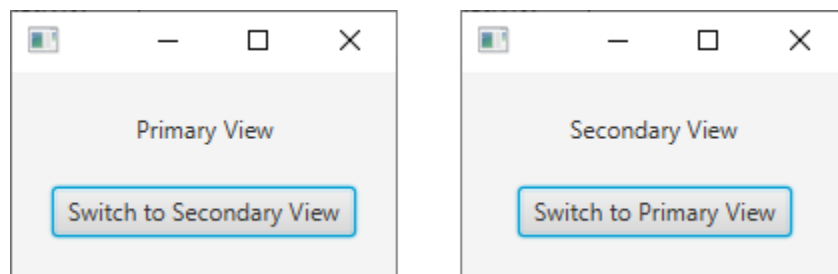
Or you can immediately download the generated project from:  
Chapter\_07\_JavaFX > javafx-minimal-fxml

When you open this project in the IDE you'll see that the visual part is in the resources-directory, while the logic is in the java-directory as controllers.



Project structure of FXML start project

Starting the application shows you a little box in which you can switch between two views.



FXML start project: primary and secondary view

As you can see in the project structure each view has an FXML file, for instance for the primary view:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <?import javafx.scene.layout.VBox?>
4 <?import javafx.scene.control.Label?>
5 <?import javafx.scene.control.Button?>
6 <?import javafx.geometry.Insets?>
7
8 <VBox alignment="CENTER" spacing="20.0" xmlns="http://javafx.com/javafx/8.0.171"
9     xmlns:fx="http://javafx.com/fxml/1"
10    fx:controller="be.webtechie.PrimaryController">
11    <children>
12        <Label text="Primary View" />
13        <Button fx:id="primaryButton" text="Switch to Secondary View"
14            onAction="#switchToSecondary"/>
15    </children>

```

```
16     <padding>
17         <Insets bottom="20.0" left="20.0" right="20.0" top="20.0" />
18     </padding>
19 </VBox>
```

And for each FXML-file a controller which handles the actions “#switchToSecondary” and “#switchToPrimary”:

```
1 public class PrimaryController {
2     @FXML
3     private void switchToSecondary() throws IOException {
4         App.setRoot("secondary");
5     }
6 }
```

and

```
1 public class SecondaryController {
2     @FXML
3     private void switchToPrimary() throws IOException {
4         App.setRoot("primary");
5     }
6 }
```

Such an approach is ideal when you are working on a bigger application where the visual work is done by different team members than the ones who do the implementation work. They both need to agree on the calls to be made, but the work can be handled separately.

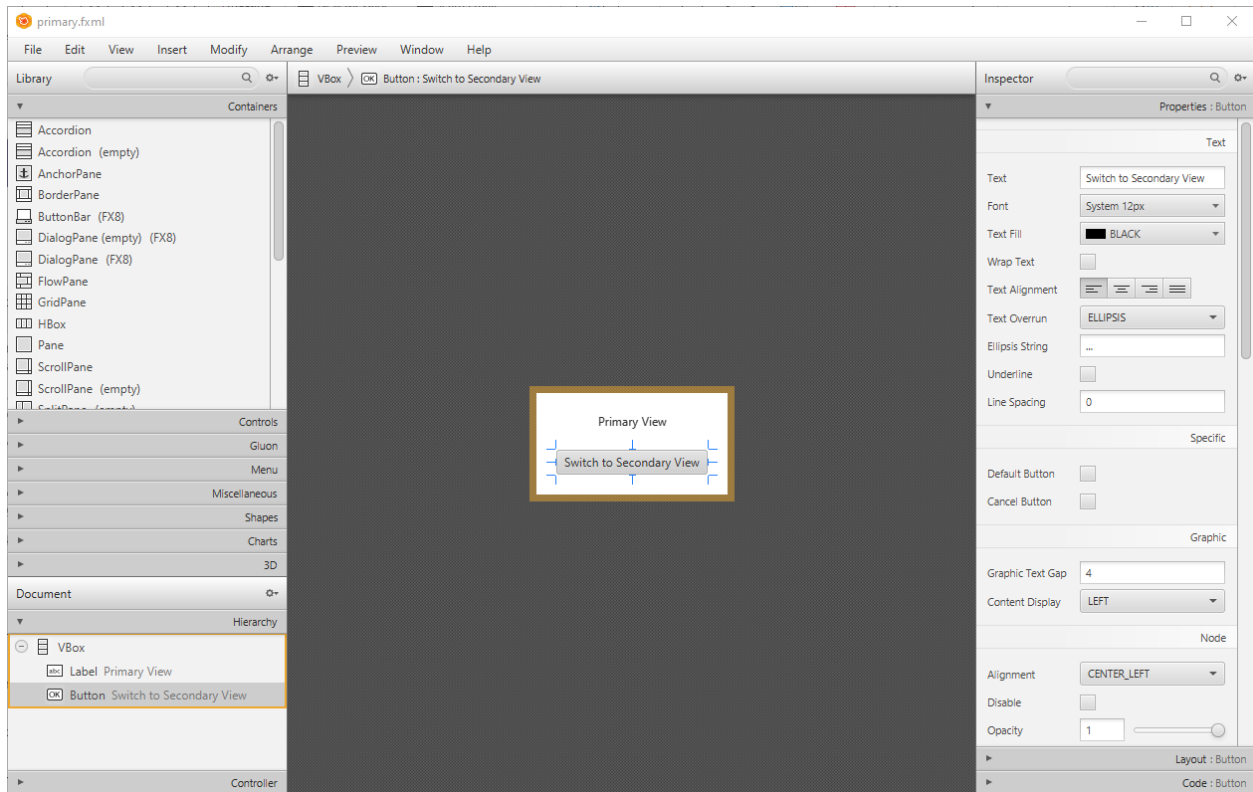
## Scene Builder

[Scene Builder](https://gluonhq.com/products/scene-builder/)<sup>88</sup> is a tool to help you build JavaFX user interfaces. It can read FXML files and will show you both the visual result as the properties you can define for each element.

Let’s, for instance, open the “primary.fxml” file from the generated start application. As you can see all the available components are listed on the left, and the properties of the selected component on the right.

---

<sup>88</sup><https://gluonhq.com/products/scene-builder/>



Scene Builder with primary.fxml

If you want to experiment with FXML to generate JavaFX user interfaces, Scene Builder is worth a try!



# Just a thought - Beware of the PAF

I first learned about the “WAF” at work during a lunch discussion between engineers, on self-made domotics projects. The major reason such a project fails or succeeds is the “**Wife Acceptance Factor**”. According to some, the same WAF is the most critical point in every decision when buying a new television, radio or any electronic device with a complex remote control.

But as I strongly believe technology is not a man-only thing, the “W” in “WAF” gives me even more grey hair than I already have... Technology is not male – or at least it should not be at all. So that’s why I officially rename “WAF” to “PAF” now, “**Partner Acceptance Factor**”!

I went through this renaming-process before. At my job, we were selling devices with the name “MMI”, “**Man-Machine Interface**”. It’s a touch screen device used by the train driver to select the journey, which announcements need to be played back, answer an emergency call, etc. But not all train drivers are male of course. So by introducing the new name “HMI”, “**Human-Machine Interface**” in one project, I was able to slowly break into the company’s history of product naming and turn this unfriendly name into a new one.

It’s only a small step of course, but as we also do with CoderDojo (a computer club for kids), we need to convince girls technology is fun! Diversifying the group of people working in technology is important. Good products can only be made if the people involved are a good mixture of age, gender, origin...

As STEAM (Science, Technology, Engineering, Arts and Mathematics) receives more and more attention in education, we see a slow change in the number of girls choosing for a scientific trajectory. This book is my own little attempt to make sure enough fun and simple getting-started projects are available to inspire “kids” of any age.

Oh, and as a side note, if anyone asks why you would build something yourself if you can buy it off-the-shelf, the correct answer is “**Because I can!**”. Even if you fail during the process, you will have learned new things!

While working on this book, I also encountered the “CAF”, the “**Community Acceptance Factor**”. Using Java on the Pi seems to be a polarizing subject according to Mark Heckler (Spring Developer & Advocate at Pivotal Software). You either totally love it or hate it, no middle ground. To some, a Pi is made for Python. Of course, I don’t agree ;-) Python is great and you can get started with it very quickly, but I don’t like the user interfaces you can make with it ([and I tried](#)<sup>¶</sup>). JavaFX provides way better code for this. On the other hand, I have to repeat myself... Why do I prefer Java on the Pi? “**BECAUSE WE CAN!**”

<sup>¶</sup><https://www.youtube.com/watch?v=yMtzJCo6nAk>

# Chapter 8: Bits and bytes

A lot of interactions with the GPIOs require some understanding of bits and bytes. Let's dive into a quick summary.

Very short:

- All logic inside the brain of a computer is a bit which can be 0 or 1 (off or on).
- When you combine 8 bits, you get a byte.



Bits make it very easy to handle data, true or false, nothing else.

But in the next years, quantum computers should become available, and they use **qubits which can be 0 AND 1 at the same time**, what is called **superposition**<sup>89</sup>. The first “Proof of Principle” quantum computers are already available and they use 20 to 50 qubits. This looks a very small number compared to the Gigabyte memory and storage we now have on PCs, even the Pi.

But quantum computers use their qubits at the same time, so if there are 20 qubits,  $2^{20}$  number of parallel computations can be achieved and can solve problems which are impossible to solve with current supercomputers. Breaking encryption keys, simulating the effect of new medicines, correctly predicting the weather... are just a few examples.

The goal is to reach a quantum computer with 300 qubits. And **the value  $2^{300}$  is bigger than the number of atoms in the whole universe**<sup>90</sup>! If a quantum computer with 50 qubits is capable of breaking all currently known encryptions in seconds, what will one with 300 qubits be able to do?!

Programming a quantum computer will be very challenging as even the most intelligent physicist and mathematicians find it **hard to understand the principles of all quantum stuff**<sup>91</sup>. So, let's just stick to our **0 OR 1** for now ;-)

---

<sup>89</sup>[https://en.wikipedia.org/wiki/Quantum\\_superposition](https://en.wikipedia.org/wiki/Quantum_superposition)

<sup>90</sup><https://www.quora.com/What-is-2-raised-to-the-power-of-300>

<sup>91</sup><https://hubpages.com/education/QuantumVagary>

## Convert bits to a numeric and hex value

A combination of multiple bits is converted to a number, by using the power of 2.

In everyday life, we are used to decimal values where we group everything by 10, 20, 30. In programming, hexadecimal values are used more, and they have the range 0 to 15, which perfectly matches the maximum value of four bits (“1111”). A hex value is written as x0 to xF.

The following table shows all possible combinations of 4 bits ranging from “0000” to “1111”:

| Bits | 2 <sup>3</sup> | 2 <sup>2</sup> | 2 <sup>1</sup> | 2 <sup>0</sup> | +       | Number | HEX |
|------|----------------|----------------|----------------|----------------|---------|--------|-----|
|      | 8              | 4              | 2              | 1              |         |        |     |
| 0000 | 0              | 0              | 0              | 0              | 0+0+0+0 | 0      | x0  |
| 0001 | 0              | 0              | 0              | 1              | 0+0+0+1 | 1      | x1  |
| 0010 | 0              | 0              | 1              | 0              | 0+0+2+0 | 2      | x2  |
| 0011 | 0              | 0              | 1              | 1              | 0+0+2+1 | 3      | x3  |
| 0100 | 0              | 1              | 0              | 0              | 0+4+0+0 | 4      | x4  |
| 0101 | 0              | 1              | 0              | 1              | 0+4+0+1 | 5      | x5  |
| 0110 | 0              | 1              | 1              | 0              | 0+4+2+0 | 6      | x6  |
| 0111 | 0              | 1              | 1              | 1              | 0+4+2+1 | 7      | x7  |
| 1000 | 1              | 0              | 0              | 0              | 8+0+0+0 | 8      | x8  |
| 1001 | 1              | 0              | 0              | 1              | 8+0+0+1 | 9      | x9  |
| 1010 | 1              | 0              | 1              | 0              | 8+0+2+0 | 10     | xA  |
| 1011 | 1              | 0              | 1              | 1              | 8+0+2+1 | 11     | xB  |
| 1100 | 1              | 1              | 0              | 0              | 8+4+0+0 | 12     | xC  |
| 1101 | 1              | 1              | 0              | 1              | 8+4+0+1 | 13     | xD  |
| 1110 | 1              | 1              | 1              | 0              | 8+4+2+0 | 14     | xE  |
| 1111 | 1              | 1              | 1              | 1              | 8+4+2+1 | 15     | xF  |

## Calculate a byte value

A byte consists of 8 bits and has the range of 0x00 (= 0) to 0xFF (= 255).

So we need to extend the table above to have 8 bits. Let's take a few examples:

| Bits     | 2 <sup>7</sup> | 2 <sup>6</sup> | 2 <sup>5</sup> | 2 <sup>4</sup> | 2 <sup>3</sup> | 2 <sup>2</sup> | 2 <sup>1</sup> | 2 <sup>0</sup> | +         | Totaal | HEX |
|----------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|-----------|--------|-----|
|          | 128            | 64             | 32             | 16             | 8              | 4              | 2              | 1              |           |        |     |
| 00001111 | 0              | 0              | 0              | 0              | 1              | 1              | 1              | 1              | 8+4+2+1   | 15     | x0F |
| 00011111 | 0              | 0              | 0              | 1              | 1              | 1              | 1              | 1              | 16+...+1  | 31     | x1F |
| 00100000 | 0              | 0              | 1              | 0              | 0              | 0              | 0              | 0              | 32        | 32     | x20 |
| 11111111 | 1              | 1              | 1              | 1              | 1              | 1              | 1              | 1              | 128+...+1 | 255    | xFF |



To fully understand how this is converted for all possible bit combinations, you can use the sample application from the sources with a JavaFX application:

Chapter\_08\_BitsAndBytes > javafx-bits-calculator

The image shows three screenshots of a JavaFX application interface for converting bit patterns to numbers. Each screenshot has a 'Select type of value' section with radio buttons for 'Integer', 'Short', and 'Byte', where 'Byte' is selected. Below this is a 'Power of 2' section with labels 2<sup>7</sup> through 2<sup>0</sup> and a 'Value' section with labels 128 through 1. A 'Select' row contains checkboxes for each bit position. The 'Bits' field shows the input bit pattern, the 'Hex value' field shows the result, and the 'Unsigned value' and 'Signed value' fields show the decimal ranges.

| Input Bits | Hex Value | Unsigned Value Range    | Signed Value Range          |
|------------|-----------|-------------------------|-----------------------------|
| 0000 1111  | 0x0F      | 15 - Min: 0 - Max: 255  | 15 - Min: -127 - Max: 127   |
| 0001 1111  | 0x1F      | 31 - Min: 0 - Max: 255  | 31 - Min: -127 - Max: 127   |
| 1111 1111  | 0xFF      | 255 - Min: 0 - Max: 255 | -127 - Min: -127 - Max: 127 |

Converting 0000 1111, 0001 1111 and 1111 1111 to number

## Value ranges in Java

### Difference between Byte, Short, Integer and Long

All these are numeric objects and each uses a fixed number of bytes in memory:

| Type    | N° of bytes | Minimum            | Maximum            |
|---------|-------------|--------------------|--------------------|
| Byte    | 1           | 0x00               | 0xFF               |
| Short   | 2           | 0x0000             | 0xFFFF             |
| Integer | 4           | 0x00000000         | 0xFFFFFFFF         |
| Long    | 8           | 0x0000000000000000 | 0xFFFFFFFFFFFFFFFF |

### Minimum and maximum values in Java

Let's go back to Java and check how values are represented with the following code:



Chapter\_08\_BitsAndBytes > java-value-limits

```

1  class PrintLimits {
2      public static void main(String[] args) {
3          System.out.println("Byte");
4          System.out.println("    Min: " + Byte.MIN_VALUE);
5          System.out.println("    Max: " + Byte.MAX_VALUE);
6
7          System.out.println("Short");
8          System.out.println("    Min: " + Short.MIN_VALUE);
9          System.out.println("    Max: " + Short.MAX_VALUE);
10
11         System.out.println("Integer");
12         System.out.println("    Min: " + Integer.MIN_VALUE);
13         System.out.println("    Max: " + Integer.MAX_VALUE);
14
15         System.out.println("Long");
16         System.out.println("    Min: " + Long.MIN_VALUE);
17         System.out.println("    Max: " + Long.MAX_VALUE);
18     }
19 }

```

As a result, we get these values:

```

1 Byte
2   Min: -128
3   Max: 127
4 Short
5   Min: -32768
6   Max: 32767
7 Integer
8   Min: -2147483648
9   Max: 2147483647
10 Long
11  Min: -9223372036854775808
12  Max: 9223372036854775807

```



Hmm, this is unexpected! Does a byte have the range of -128 to 127, instead of 0 to 255?!

On some forums, there was some confusion and someone thought he was doing things wrong as he expected a different value during testing. To understand this, we need to know **the difference between signed and unsigned values!**

## Signed versus unsigned

When you calculate the byte value to a signed number value, the major bit (the most left one) is handled as an indicator for a negative number (1) or a positive number (0), for example, 8 bits “1000 1111”:

|                       |                                     |                             |                                       |                          |                                     |                                     |                                     |                                     |
|-----------------------|-------------------------------------|-----------------------------|---------------------------------------|--------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| Select type of value: | <input type="radio"/> Integer       | <input type="radio"/> Short | <input checked="" type="radio"/> Byte |                          |                                     |                                     |                                     |                                     |
| Power of 2            | 2 <sup>7</sup>                      | 2 <sup>6</sup>              | 2 <sup>5</sup>                        | 2 <sup>4</sup>           | 2 <sup>3</sup>                      | 2 <sup>2</sup>                      | 2 <sup>1</sup>                      | 2 <sup>0</sup>                      |
| Value                 | 128                                 | 64                          | 32                                    | 16                       | 8                                   | 4                                   | 2                                   | 1                                   |
| Select                | <input checked="" type="checkbox"/> | <input type="checkbox"/>    | <input type="checkbox"/>              | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| Bits                  | 1000 1111                           |                             |                                       |                          |                                     |                                     |                                     |                                     |
| Hex value             | 0x8f                                |                             |                                       |                          |                                     |                                     |                                     |                                     |
| Unsigned value        | 143 - Min: 0 - Max: 255             |                             |                                       |                          |                                     |                                     |                                     |                                     |
| Signed value          | -15 - Min: -127 - Max: 127          |                             |                                       |                          |                                     |                                     |                                     |                                     |

As an example: byte 0x8f can be both 143 and -15!

The same goes for a short which consist of two bytes (= 16 bits), for example, “1000 0000 0000 1111”:

|                       |                                     |                                        |                            |                          |                          |                          |                          |                          |                          |                          |                          |                          |                                     |                                     |                                     |                                     |
|-----------------------|-------------------------------------|----------------------------------------|----------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| Select type of value: | <input type="radio"/> Integer       | <input checked="" type="radio"/> Short | <input type="radio"/> Byte |                          |                          |                          |                          |                          |                          |                          |                          |                          |                                     |                                     |                                     |                                     |
| Power of 2            | 2 <sup>15</sup>                     | 2 <sup>14</sup>                        | 2 <sup>13</sup>            | 2 <sup>12</sup>          | 2 <sup>11</sup>          | 2 <sup>10</sup>          | 2 <sup>9</sup>           | 2 <sup>8</sup>           | 2 <sup>7</sup>           | 2 <sup>6</sup>           | 2 <sup>5</sup>           | 2 <sup>4</sup>           | 2 <sup>3</sup>                      | 2 <sup>2</sup>                      | 2 <sup>1</sup>                      | 2 <sup>0</sup>                      |
| Value                 | 32,768                              | 16,384                                 | 8,192                      | 4,096                    | 2,048                    | 1,024                    | 512                      | 256                      | 128                      | 64                       | 32                       | 16                       | 8                                   | 4                                   | 2                                   | 1                                   |
| Select                | <input checked="" type="checkbox"/> | <input type="checkbox"/>               | <input type="checkbox"/>   | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| Bits                  | 1000                                | 0000                                   | 0000                       | 1111                     |                          |                          |                          |                          |                          |                          |                          |                          |                                     |                                     |                                     |                                     |
| Hex value             | 0x800f                              |                                        |                            |                          |                          |                          |                          |                          |                          |                          |                          |                          |                                     |                                     |                                     |                                     |
| Unsigned value        | 32,783 - Min: 0 - Max: 65,535       |                                        |                            |                          |                          |                          |                          |                          |                          |                          |                          |                          |                                     |                                     |                                     |                                     |
| Signed value          | -15 - Min: -32,768 - Max: 32,767    |                                        |                            |                          |                          |                          |                          |                          |                          |                          |                          |                          |                                     |                                     |                                     |                                     |

Hex value 0x800f is both 32873 and -15 depending on how you convert it

## Conclusion

You need to be sure how you handle numeric values when you read them out to not confuse between the signed and unsigned value!

Luckily we have the `Byte.toUnsignedInt(b)` and `Integer.toUnsignedString(i)` methods to make sure we are reading out the correct value when logging. Here a quick demo with some byte and integer values:

```

1 class HexIntegerToString {
2     public static void main(String[] args) {
3         convertByte((byte) 8);
4         convertByte((byte) 124);
5         convertByte((byte) 170);
6
7         convertInteger(0x7FFFFFFF);
8         convertInteger(0x80000001);
9     }
10
11     private static void convertByte(byte value) {
12         System.out.println("Byte Unsigned: "
13             + Byte.toUnsignedInt(value)
14             + "\tSigned: " + value);
15         System.out.println("    Hex value: 0x"
16             + Integer.toHexString(value & 0xFF));
17         System.out.println("    Binair:      "
18             + padLeftZero(Integer.toBinaryString(value & 0xFF), 8));
19     }

```

```

20
21     private static void convertInteger(int value) {
22         System.out.println("Integer Unsigned: "
23             + Integer.toUnsignedString(value)
24             + "\tSigned: " + Integer.toString(value));
25         System.out.println("    Hex value: 0x"
26             + Integer.toHexString(value));
27         System.out.println("    Binair:    "
28             + padLeftZero(Integer.toBinaryString(value), 32));
29     }
30
31     private static String padLeftZero(String txt, int length) {
32         StringBuilder rt = new StringBuilder();
33         for (int i = 0; i < (length - txt.length()); i++) {
34             rt.append("0");
35         }
36         return rt.append(txt).toString();
37     }
38 }

```

When running this code, we get this output with the expected and readable values:

```

1  $ java HexIntegerToString.java
2  Byte Unsigned: 8          Signed: 8
3      Hex value: 0x8
4      Binair:    00001000
5  Byte Unsigned: 124       Signed: 124
6      Hex value: 0x7c
7      Binair:    01111100
8  Byte Unsigned: 170       Signed: -86
9      Hex value: 0xaa
10     Binair:    10101010
11  Integer Unsigned: 2147483647 Signed: 2147483647
12     Hex value: 0x7fffffff
13     Binair:    01111111111111111111111111111111
14  Integer Unsigned: 2147483649 Signed: -2147483647
15     Hex value: 0x80000001
16     Binair:    10000000000000000000000000000001

```



## What can we do with this?

### Web colors

The most known use of hex bytes is the definition of colors in HTML. Colors are defined with three bytes for red, green and blue. Some examples:

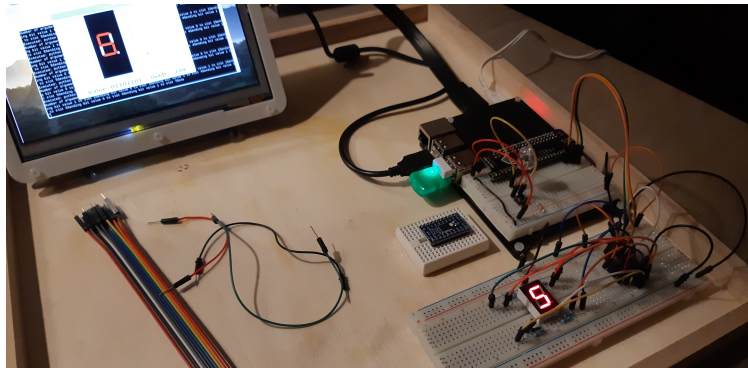
- #FFFFFF is full R+G+B = white
- #FF0000 is full red only
- #D3D3D3 is light grey with value 211 for red, green and blue

You can try this out on [this page of w3schools.com](https://www.w3schools.com/colors/colors_rgb.asp)<sup>92</sup>.

This is also the color-coding we can use to define the value for RGB-LEDs and LED strips. We will need this later in this book.

### Controlling a numeric segment display

Let's make a Java project to control a numeric segment display where we can fully use the possibilities of 8 bits in a byte.

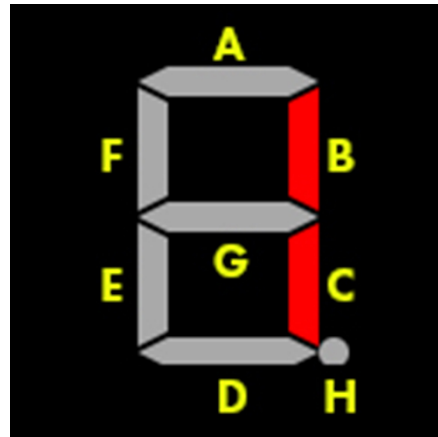
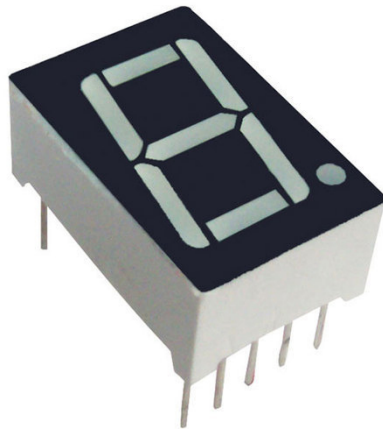


Full test set-up with running JavaFX application

### The LED numeric segment display

Each segment is an LED with its own pin so it can be controlled separately. To easily understand which segment will be used, they are identified as A till H.

<sup>92</sup>[https://www.w3schools.com/colors/colors\\_rgb.asp](https://www.w3schools.com/colors/colors_rgb.asp)



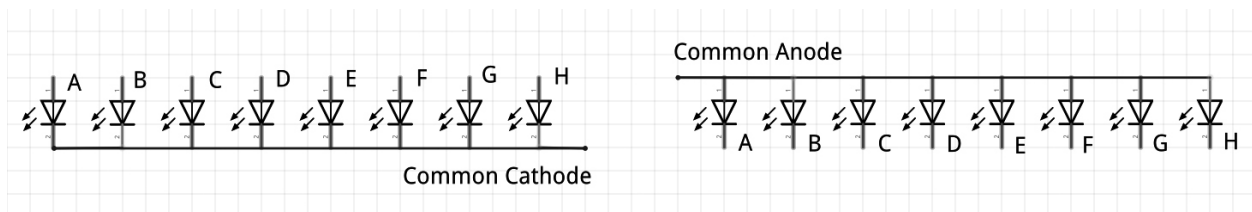
LED number display with the segments B and C on, showing the number 1

For the numbers 0 till 9 we can define the hex value to turn on the desired segments. E.g. for the 0 we need segments A till F, not G and H. So the 6 lowest bits 00111111 which is equal to 0x3f.

This is the full table to control the 8 segments with one hex value to indicate which of the 8 LED(s) we want to be on.

| LED display | h | g | f | e | d | c | b | a | Hex value | Number value |
|-------------|---|---|---|---|---|---|---|---|-----------|--------------|
| 0           | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0x3f      | 63           |
| 1           | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0x06      | 6            |
| 2           | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0x5b      | 91           |
| 3           | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0x4f      | 79           |
| 4           | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0x66      | 102          |
| 5           | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0x6d      | 109          |
| 6           | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0x7d      | 125          |
| 7           | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0x07      | 7            |
| 8           | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0x7f      | 127          |
| 9           | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0x6f      | 111          |

Just like RGB-LEDs (see “Chapter 2: Tools” > “Hardware” > “RGB-LED”) these displays exist in two “flavours” with common cathode (-) or anode (+). For this example, we need a common cathode type as each LED will get its power from a separate pin of the IC, so an [5101AS \(datasheet\)<sup>93</sup>](https://www.datasheets360.com/pdf/-5896711825166489141) was used.



Common cathode versus common anode

<sup>93</sup><https://www.datasheets360.com/pdf/-5896711825166489141>

## Using a bit shift register

The Pi has enough GPIO connections to control these 8 LED segments. And you can control them the same way as we have done before in the JavaFX chapter with the `Gpio.java` class. For instance, turning one segment on connected to pin with wiringpi number 5 would be done like this:

```
1 Gpio gpio = new Gpio();
2 gpio.initiatePin(5, "out");
3 gpio.setPinState(5, true);
```

But this means we need 8 pins for 8 segments and if we would be building a more complex project, we could quickly run out of GPIOs... Let's take a better approach and use a bit shift register. This is a digital circuit that can be used to extend the number of GPIOs of a Pi (or Arduino).

In our case, we will work with the SN74HC595, a Serial-In-Parallel-Out shift register. Instead of 8, we will only need three of the Pi GPIOs to control 8 LED segments.

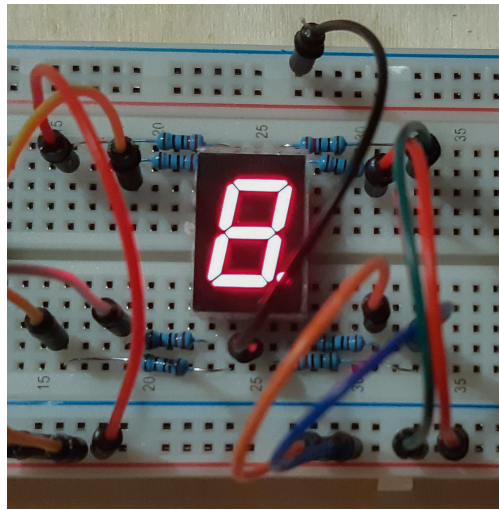
First, we take a look at the [technical sheet](#)<sup>94</sup> to be sure what we can do with it and how to connect it. This is the pin-listing with a reference to the variable we will use in the Python script "shift.py":

| Pin | Name  | Description                                                                                          | In shift.py    |
|-----|-------|------------------------------------------------------------------------------------------------------|----------------|
| 1   | Qb    | Output of B value                                                                                    |                |
| 2   | Qc    | Output of C value                                                                                    |                |
| 3   | Qd    | Output of D value                                                                                    |                |
| 4   | Qe    | Output of E value                                                                                    |                |
| 5   | Qf    | Output of F value                                                                                    |                |
| 6   | Qg    | Output of G value                                                                                    |                |
| 7   | Qh    | Output of H value                                                                                    |                |
| 8   | GND   | Ground                                                                                               |                |
| 9   | Qh'   | Output of H' value                                                                                   |                |
| 10  | SRCLR | Input of CLEAR. Not used in this example.                                                            |                |
| 11  | SRCLK | Input of the CLOCK. A pulse on this pin lets the register store the value of the data pin in memory. | PIN_SRCLK      |
| 12  | RCLK  | Input of LATCH. A pulse on this pin sets the stored value on the 8 output pins.                      | PIN_RCLK_LATCH |
| 13  | OE    | Input of output enable                                                                               |                |
| 14  | SER   | Input of the DATA which needs to be sent bit by bit.                                                 | PIN_DATA       |
| 16  | Qa    | Output of A value                                                                                    |                |
| 15  | VCC   | Power pin                                                                                            |                |

## Wiring

Before connecting the IC, you can first try all connections to the LED segment display with a fixed 3.3V input instead of a GPIO.

<sup>94</sup><https://www.ti.com/lit/ds/symlink/sn74hc595.pdf>



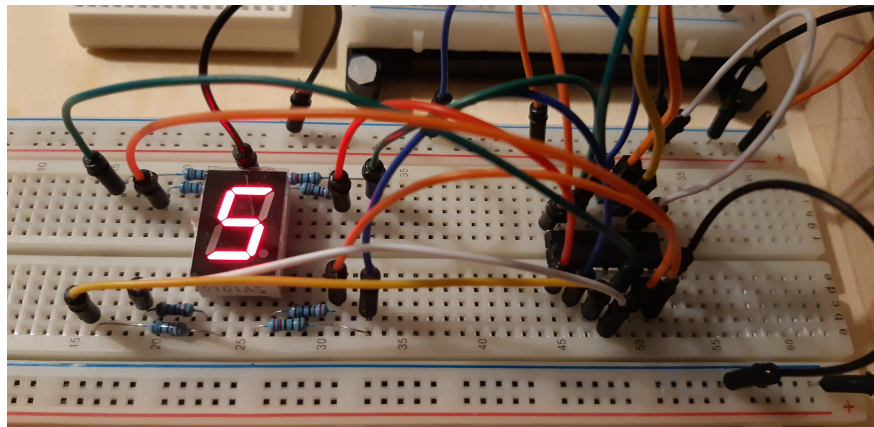
Testing the LED segment connections



Always double-check which resistors you need to use and the expected input voltage. Not all LEDs are forgiving when you are applying a too high voltage.

At least one component died during the writing of this book...

After we have checked the LED segments are working with the fixed power supply, we can start with the full connections including the IC.

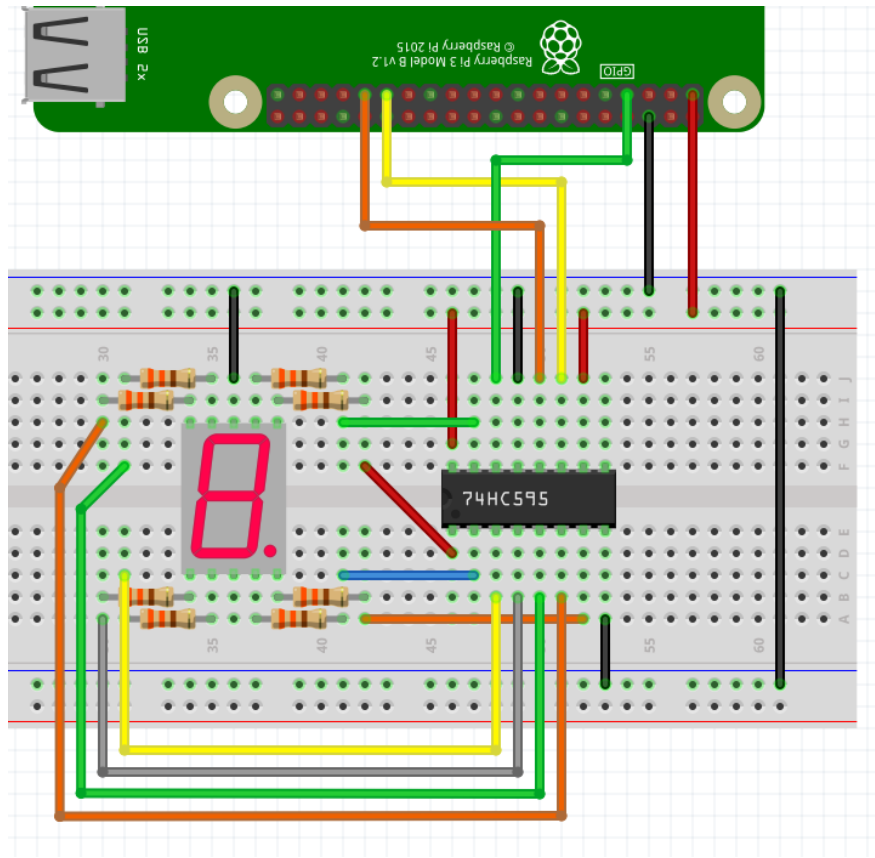


Full wiring for the example application

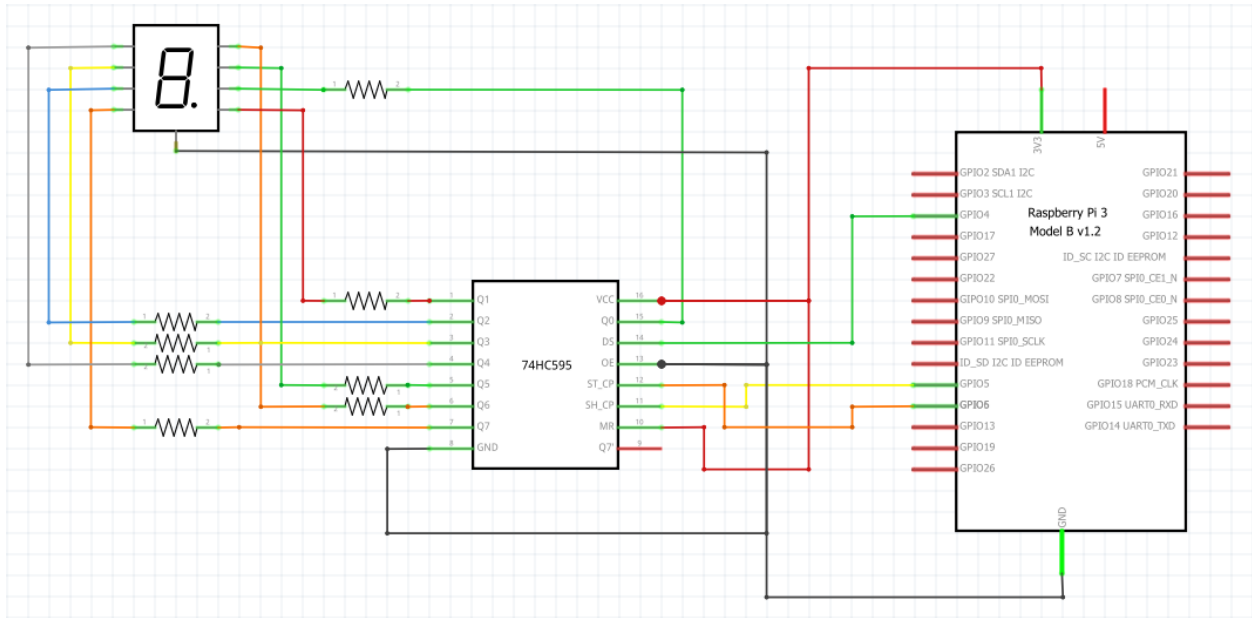


A picture, of course, is not very clear to get the correct wiring, so below you find a Fritzing breadboard and scheme which is also available in the sources:

Chapter\_08\_BitsAndBytes > schemes > 74NC595-led-number.fzz



This is the same wiring but another view to make things even more clear:



## Control with Python code

In the previous chapter, we used `Gpio.java` to execute terminal commands from Java to toggle and read GPIOs. In the next chapter, we will be working with `Pi4J` so we can control the GPIOs directly from within Java. But in this chapter, we take yet another approach and use a Python script to execute multiple GPIO interactions in one flow.

This script will be included in the Java application so it can be called from within the application and needs to execute these steps:

1. Parse the given number value to be sent to the LED segments
2. For all 8 bits in the number value:
  1. Send the first bit from the HEX value to the DATA pin
  2. Send a pulse to the CLOCK pin to store the DATA value
3. Send a pulse to the RELEASE pin to send all 8 stored values to the 8 output pins
4. Clean-up the GPIOs so we can use them again

Let us first create a Python file “`shift.py`” (e.g. in “`/home/pi`”) on the Pi, so we can start it in the terminal with a number value as start-up argument, e.g. number 109 (see the table before) to show a “5” on the segment display:

```
1 $ python shift.py 109
```

The Python script sets the bits of this value 109 to the 8 memory positions of the SN74HC595. This is the full script:

```
1 # coding=utf-8
2
3 import RPi.GPIO as GPIO
4 import time
5 import sys
6
7 # Define BCM pin numbers
8 PIN_DATA = 4
9 PIN_SRCLK = 5
10 PIN_RCLK_LATCH = 6
11 WAIT_TIME = 0.01
12
13 # Ignore GPIO warnings as we are re-using pins
14 GPIO.setwarnings(False)
15
16 # Initialize pins
17 GPIO.setmode(GPIO.BCM)
```

```
18 GPIO.setup(PIN_DATA, GPIO.OUT)
19 GPIO.setup(PIN_SRCLK, GPIO.OUT)
20 GPIO.setup(PIN_RCLK_LATCH, GPIO.OUT)
21
22 # Get value from startup argument
23 inputValue = 0xff
24
25 print "Number of arguments: " + str(len(sys.argv))
26
27 if len(sys.argv) > 1:
28     if sys.argv[1].startswith('0x'):
29         # This is used to parse a hex value in format "0x00" into an integer
30         inputValue = int(sys.argv[1], 16)
31     else:
32         # Parsing other values to integer
33         inputValue = int(sys.argv[1])
34
35 print "Sending value: " + str(inputValue)
36
37 # Set the RELEASE pin low so the values are stored in memory
38 GPIO.output(PIN_RCLK_LATCH, 0)
39 time.sleep(WAIT_TIME)
40
41 for y in range(8):
42     # Set the DATA pin to the bit y of the given value
43     bit = inputValue >> (7 - y) & 1
44     print "Sending bit value " + str(bit) + " to slot " + str(y)
45
46     # Prepare for the next value by setting the CLOCK pin LOW
47     GPIO.output(PIN_SRCLK, 0)
48     time.sleep(WAIT_TIME)
49
50     # Set the data (high or low state) for the pin y
51     GPIO.output(PIN_DATA, bit)
52     time.sleep(WAIT_TIME)
53
54     # Set the CLOCK pin HIGH to store the DATA value into memory
55     GPIO.output(PIN_SRCLK, 1)
56     time.sleep(WAIT_TIME)
57
58 # Loop is finished, so all 8 values are sent
59 # Set the RELEASE pin high so the values from memory are sent to the outputs
60 GPIO.output(PIN_RCLK_LATCH, 1)
```

```

61 time.sleep(WAIT_TIME)
62
63 print "Done"

```

With the comments in the code this should be pretty self-explaining, but let us take out one special line from the for-loop to get bit per bit from the `inputValue`:

```
1 bit = inputValue >> (7 - y) & 1
```

- “`inputValue >> (7 - y)`” moves the bits to the right
- “`& 1`” is used to get the most right single bit from the given value

To illustrate this we take the `inputValue` 109 which is `0x6d = 01101101`

| <code>inputValue</code> | <code>y</code> | <code>(7 - y)</code> | <code>inputValue &gt;&gt; (7 - y)</code> | <code>&amp; 1</code> |
|-------------------------|----------------|----------------------|------------------------------------------|----------------------|
| 01101101                | 0              | 7                    | 0                                        | 0                    |
| 01101101                | 1              | 6                    | 01                                       | 1                    |
| 01101101                | 2              | 5                    | 011                                      | 1                    |
| 01101101                | 3              | 4                    | 0110                                     | 0                    |
| 01101101                | 4              | 3                    | 01101                                    | 1                    |
| 01101101                | 5              | 2                    | 011011                                   | 1                    |
| 01101101                | 6              | 1                    | 0110110                                  | 0                    |
| 01101101                | 7              | 0                    | 01101101                                 | 1                    |

When we run this script in the terminal we get the following output matching this table:

```

1 $ python shift.py 109
2 Number of arguments: 2
3 Sending value: 109
4 Sending bit value 0 to slot 0
5 Sending bit value 1 to slot 1
6 Sending bit value 1 to slot 2
7 Sending bit value 0 to slot 3
8 Sending bit value 1 to slot 4
9 Sending bit value 1 to slot 5
10 Sending bit value 0 to slot 6
11 Sending bit value 1 to slot 7
12 Done

```

But we can also call the same script with a HEX value when we start it with “0x” as this is converted correctly inside the script:



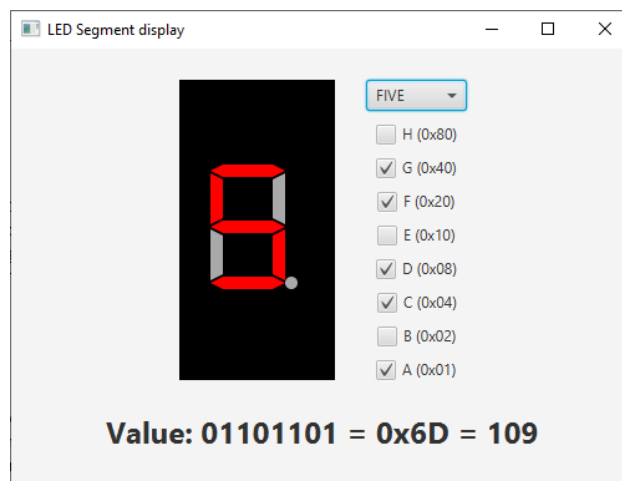
```

1 $ python shift.py 0x6d
2 Number of arguments: 2
3 Sending value: 109
4 Sending bit value 0 to slot 0
5 Sending bit value 1 to slot 1
6 Sending bit value 1 to slot 2
7 Sending bit value 0 to slot 3
8 Sending bit value 1 to slot 4
9 Sending bit value 1 to slot 5
10 Sending bit value 0 to slot 6
11 Sending bit value 1 to slot 7
12 Done

```

## Control with a Java application

Now let's make a user interface in JavaFX to easily test these bits to/from hex conversion.

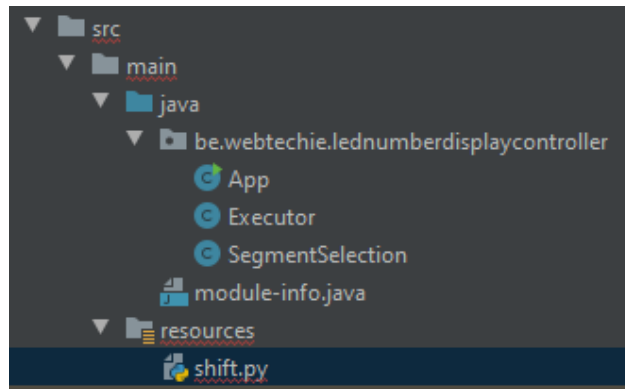


LED segment test application showing the selected segments as bits, hex value and number value



The finished application is available in:  
 Chapter\_08\_BitsAndBytes > javafx-led-number-display-controller

We can start again with a copy from the minimal JavaFX application from Chapter 7 and rename the project to “lednumberdisplaycontroller”. Copy the script “shift.py” we created before, into the resources directory of our Java project. This way, it will be packaged into the .jar-file with our application which we will run on the Pi.



Java project structure with the shift.py added in the resources directory

We need to change the pom-artifactId and add a dependency to the library “javafx-led-number-display” which has been published into the Maven repository, to easily visualize an LED segment display in JavaFX.

```

1 <groupId>be.webtechie</groupId>
2 <artifactId>led-number-display-controller</artifactId>
3
4 <dependencies>
5     ...
6     <dependency>
7         <groupId>be.webtechie</groupId>
8         <artifactId>javafx-led-number-display</artifactId>
9         <version>0.0.3</version>
10    </dependency>
11 </dependencies>
  
```



Because be.webtechie.javafx-led-number-display is a Maven dependency, it is not part of the sources of this book, but can be found in its own repository on <https://github.com/FDelporte/JavaFXLedNumberDisplay><sup>95</sup>

In this same pom.xml file we also need to add a plugin to be able to build a jar-file which includes the extra dependency and we don’t need to install it on the device where we want to run this application. If you use a different package name for the App-class you need to update the mainClass-value.

<sup>95</sup><https://github.com/FDelporte/JavaFXLedNumberDisplay>

```
1 <plugins>
2   ...
3   <plugin>
4     <artifactId>maven-assembly-plugin</artifactId>
5     <version>2.2.1</version>
6     <configuration>
7       <descriptorRefs>
8         <descriptorRef>jar-with-dependencies</descriptorRef>
9       </descriptorRefs>
10      <archive>
11        <manifest>
12          <addClasspath>true</addClasspath>
13          <mainClass>be.webtechie.lednumberdisplaycontroller.App</mainClass>
14        </manifest>
15      </archive>
16    </configuration>
17    <executions>
18      <execution>
19        <id>make-assembly</id>
20        <phase>package</phase>
21        <goals>
22          <goal>single</goal>
23        </goals>
24      </execution>
25    </executions>
26  </plugin>
27 </plugins>
```

In the start method of the App-class we call “new SegmentSelection(copyShiftScriptFile())” as the UI for the segment selection. The function “copyShiftScriptFile()” copies the shift.py file from inside the .jar-file to the temp-directory of the system. The path and file name where this file is saved, is given as an argument to SegmentSelection so we can use it there to run it each time we update the LED segments in the UI.

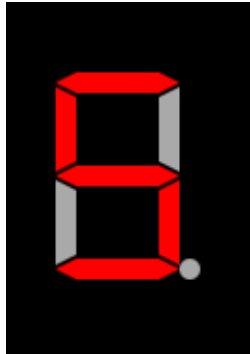
```
1 public class App extends Application {
2     public static void main(String[] args) {
3         launch();
4     }
5
6     @Override
7     public void start(Stage stage) {
8         var scene = new Scene(new SegmentSelection(copyShiftScriptFile()), 500, 350);
9         stage.setScene(scene);
10        stage.show();
11    }
12
13    /**
14     * Copy the Python file "shift.py" from the resources directory to
15     * the tmp directory.
16     *
17     * @return The path of the copied file.
18     */
19    private String copyShiftScriptFile() {
20        try (InputStream is = App.class.getResourceAsStream("/shift.py")) {
21            if (is == null) {
22                System.err.println("shift.py file not found in the jar");
23                return "";
24            }
25
26            File shiftFile = File.createTempFile("shift_", ".py");
27            try (OutputStream os = new FileOutputStream(shiftFile)) {
28                int readBytes;
29                byte[] buffer = new byte[4096];
30
31                while ((readBytes = is.read(buffer)) > 0) {
32                    os.write(buffer, 0, readBytes);
33                }
34            } catch (Exception ex) {
35                System.err.println("Error: " + ex.getMessage());
36            }
37
38            return shiftFile.toString();
39        } catch (Exception ex) {
40            System.err.println("Error: " + ex.getMessage());
41        }
42
43        return "";
44    }
45 }
```

```

44     }
45 }

```

Take a look at the full code of this class `SegmentSelection.java` and feel free to “adjust-and-mess-up” to see how this works. Only the most important parts are described here.

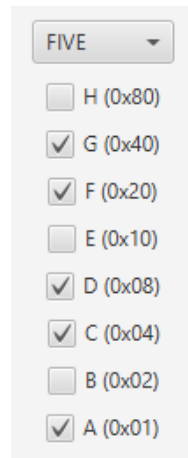


UI part generated by `generateHighlightSelection()`

```

1  /**
2  * Generate a black holder with an LED number display component from
3  * the Maven dependency be.webtechie.javaafx-led-number-display.
4  *
5  * @return {@link HBox} with the led number display
6  */
7  private HBox generateHighlightSelection() {
8      HBox holder = new HBox();
9      holder.setStyle("-fx-background-color: black;");
10     holder.setPadding(new Insets(25));
11     holder.setAlignment(Pos.CENTER);
12
13     this.ledNumberDisplay = new LedNumber(DisplaySkin.CLASSIC,
14         Color.BLACK, Color.DARKGRAY, Color.RED);
15     this.ledNumberDisplay.setMaxHeight(105);
16     holder.getChildren().add(this.ledNumberDisplay);
17
18     return holder;
19 }

```



UI part generated by generateBitSelection()

```

1  /**
2  * Generate the holder with a dropdown with highlight presets
3  * and 8 checkboxes to be able each segment separately.
4  *
5  * return {@link HBox} with combobox and checkboxes
6  */
7  private VBox generateBitSelection() {
8      VBox selectionsHolder = new VBox();
9      selectionsHolder.setSpacing(10);
10     selectionsHolder.setAlignment(Pos.CENTER);
11
12     this.selectHighLightType = new ComboBox<>();
13     this.selectHighLightType.getItems().setAll(HighlightType.values());
14     this.selectHighLightType.setOnAction(this::updateHighlights);
15     selectionsHolder.getChildren().add(this.selectHighLightType);
16
17     this.cbA = new CheckBox("A (0x01)");
18     this.cbA.setOnAction(this::updateFromBits);
19     this.cbB = new CheckBox("B (0x02)");
20     this.cbB.setOnAction(this::updateFromBits);
21     ...
22
23     selectionsHolder.getChildren().addAll(
24         this.cbH, this.cbG, this.cbF, this.cbE,
25         this.cbD, this.cbC, this.cbB, this.cbA
26     );
27
28     return selectionsHolder;

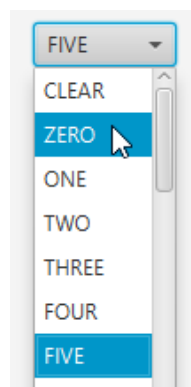
```

29 }

In the Maven library an enum is available with a list of possible HighlightTypes, which we use to fill a combobox. These HighlightTypes can be checked in the source of the Maven library (see link before) and look like this:

```

1  /**
2  * Enum with definition of the segments to be highlighted.
3  *
4  * Placing of the LEDs:
5  * AAAA
6  * F B
7  * GGGG
8  * E C
9  * DDDD
10 */
11 public enum HighlightType {
12     CLEAR(false, false, false, false, false, false, false),
13
14     ZERO(true, true, true, true, true, true, false),
15     ONE(false, true, true, false, false, false, false),
16     TWO(true, true, false, true, true, false, true),
17     ...
18 }
```



ComboBox using values from the enum HighlightType

When the user changes the selection in the combobox all the checkboxes are adjusted to represent the selected HighlightType. When this is done “setValue()” is called to send this new value to the Python script.

```

1  /**
2  * Change the states of all the {@link ComboBox} to match
3  * the selected {@link HighlightType}
4  */
5  private void updateHighlights(ActionEvent actionEvent) {
6      HighlightType highlightType = this.selectHighLightType.getValue();
7
8      if (highlightType == null) {
9          return;
10     }
11
12     this.ledNumberDisplay.highlight(highlightType, this.cbH.isSelected());
13
14     this.cbA.setSelected(highlightType.isA());
15     this.cbB.setSelected(highlightType.isB());
16     this.cbC.setSelected(highlightType.isC());
17     this.cbD.setSelected(highlightType.isD());
18     this.cbE.setSelected(highlightType.isE());
19     this.cbF.setSelected(highlightType.isF());
20     this.cbG.setSelected(highlightType.isG());
21
22     this.setValue();
23 }

```

Whenever a new HighlightType is selected in the dropdown or a CheckBox is (de)selected, the value is recalculated, and the Python script is executed. The calculated value is also shown in a label inside the UI as bits, hex and integer value.

**Value: 01101101 = 0x6D = 109**

Calculated value used for the Python script

```

1  /**
2  * Calculate the value base on the selected {@link ComboBox},
3  * display in the label of the UI and send to the hardware
4  * by calling the Python script.
5  */
6  private void setValue() {
7      int value = (this.cbA.isSelected() ? 0x01 : 0x00)
8          + (this.cbB.isSelected() ? 0x02 : 0x00)
9          + (this.cbC.isSelected() ? 0x04 : 0x00)
10         + (this.cbD.isSelected() ? 0x08 : 0x00)

```



```
11         + (this.cbE.isSelected() ? 0x10 : 0x00)
12         + (this.cbF.isSelected() ? 0x20 : 0x00)
13         + (this.cbG.isSelected() ? 0x40 : 0x00)
14         + (this.cbH.isSelected() ? 0x80 : 0x00);
15
16     this.lblValue.setText("Value: " + value
17         + " = 0x" + padLeftZero(Integer.toHexString(value).toUpperCase(), 2)
18         + " = " + padLeftZero(Integer.toBinaryString(value), 8));
19
20     Executor.execute("python " + this.scriptFilePath + " " + value);
21 }
```

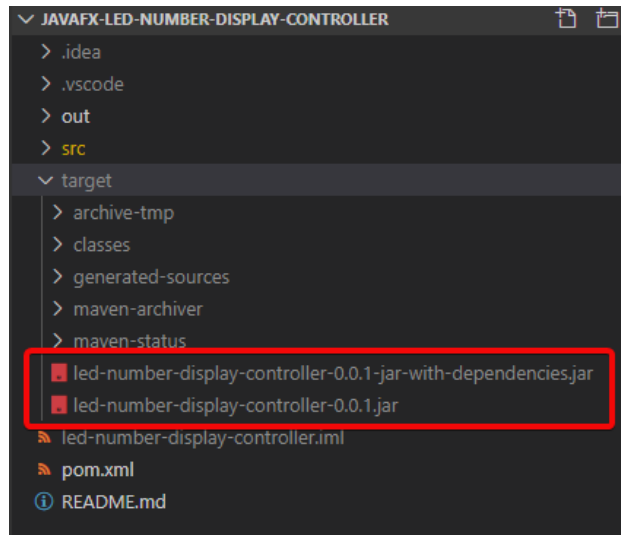
## Building and running on the Pi

Because we added “maven-assembly-plugin” into pom.xml, we can build a jar-package which includes the lednumber-dependency so we can run it on the Pi without the need to install any additional libraries.

In the terminal of your IDE the following Maven command must be executed

```
1 $ mvn clean package
2 ...
3
4 [INFO] -----
5 [INFO] BUILD SUCCESS
6 [INFO] -----
7 [INFO] Total time: 14.804 s
8 [INFO] Finished at: 2019-12-16T22:01:04+01:00
9 [INFO] -----
```

When this is finished, you’ll find two jar-files in the target directory of your project:



Jar-files generated by Maven

The one we need to copy to the Pi is “led-number-display-controller-0.0.1-jar-with-dependencies.jar”, where we can start it with “java -jar [filename]”. After the application has started, a log is shown with the location of the Python script file. After each UI change, this script is called with a number value and the script-output is also logged:

```

1 $ java -jar led-number-display-controller-0.0.1-jar-with-dependencies.jar
2
3 Initializing SegmentSelection with script file located copied to /tmp/shift_36889615\
4 36089611462.py
5
6 Executed: python /tmp/shift_3688961536089611462.py 125
7 Number of arguments: 2Sending value: 125Sending bit value 0 to slot 0Sending bit val\
8 ue 1 to slot 1Sending bit value 1 to slot 2Sending bit value 1 to slot 3Sending bit \
9 value 1 to slot 4Sending bit value 1 to slot 5Sending bit value 0 to slot 6Sending b\
10 it value 1 to slot 7Done
11
12 Executed: python /tmp/shift_3688961536089611462.py 61
13 Number of arguments: 2Sending value: 61Sending bit value 0 to slot 0Sending bit valu\
14 e 0 to slot 1Sending bit value 1 to slot 2Sending bit value 1 to slot 3Sending bit v\
15 alue 1 to slot 4Sending bit value 1 to slot 5Sending bit value 0 to slot 6Sending bi\
16 t value 1 to slot 7Done

```

And there we have it! A nice JavaFX UI, integrated use of a Python script and a fully functional hardware setup with an IC and LED segment display. Nice work! :-)

## What's next?

Just some ideas...

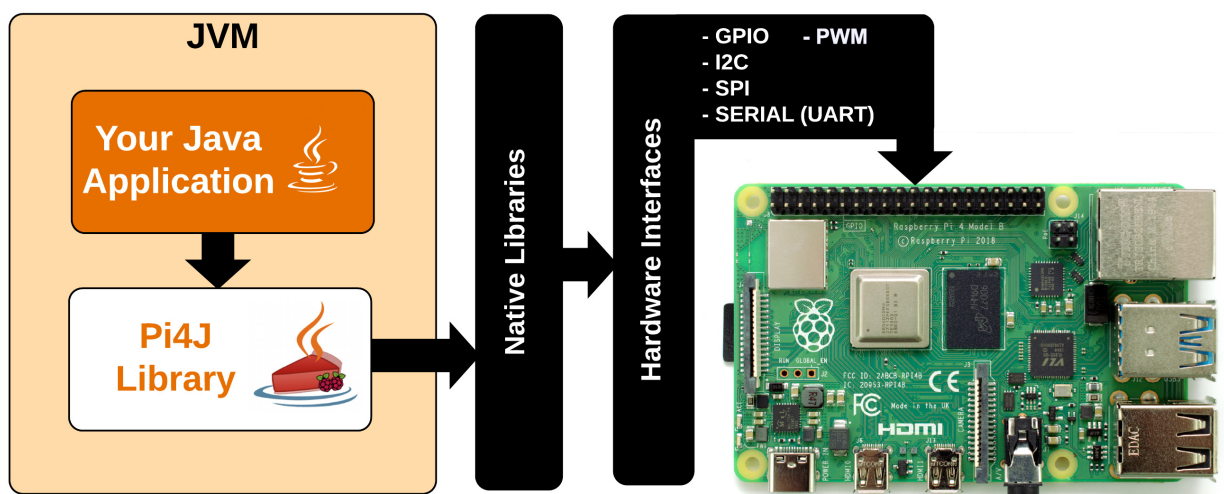
- Use 6 LEDs instead of the LED segment display to make an electronic dice.
- Use more of these ICs to connect even more LEDs or other output devices. On the [Arduino website you can find an example of this](#)<sup>96</sup> to use as a base to achieve the same with Java and Pi.
- Control a set of relays to turn devices on or off with higher voltage needs than the Pi can deliver.

---

<sup>96</sup><https://www.arduino.cc/en/Tutorial/ShiftOut>

# Chapter 9: Pi4J

Pi4J is the best library to link Java on the Raspberry Pi with the GPIOs. In our earlier experiments with the JavaFX dashboard (see “Chapter 7”), we created our own Gpio.java but this was only able to set a pin high or low and read a pin state via “terminal commands”. The Pi4J library offers a lot more methods directly connected to the hardware for optimal performance.



Pi4J is a layer between Java and the hardware

The open-source project was started by Robert Savage and Daniel Sendula and [the code is available on GitHub](#)<sup>97</sup>.

In 2019 a full rework of this library started to bring it more in line with modern Java and to be able to handle new types of Raspberry Pi’s more easily. Unfortunately, this new version is not available yet, so the following examples still use version 1.2 but should be easily adaptable later. All the examples in this chapter were developed on a Raspberry 3 B+.

As the Raspberry Pi 4 uses a new chip, some of the interior wirings are different compared to the previous Pi’s. Make sure to [update WiringPi](#)<sup>98</sup> if you want to experiment with Pi4J on a Pi 4, as this is the interface between Pi4J and the hardware, as described in Chapter 5 > WiringPi number.

All methods of Pi4J are documented with JavaDoc and the generated HTML documentation can be found on [www.pi4j.com/1.2/apidocs/index.html](http://www.pi4j.com/1.2/apidocs/index.html)<sup>99</sup>.

<sup>97</sup><https://github.com/Pi4J/pi4j/>

<sup>98</sup><http://wiringpi.com/download-and-install/>

<sup>99</sup><https://www.pi4j.com/1.2/apidocs/>

## Installation

Adding the Pi4J framework to your Pi can be done with one single command which will download and launch an installation script that performs the following steps:

- Adds the Pi4J APT repository to the local APT repositories
- Downloads and installs the Pi4J GPG public key for signature validation
- Invokes the 'apt-get update' command on the Pi4J APT repository to update the local package database
- Invokes the 'apt-get install pi4j' command to perform the download and installation

```
1 $ curl -sSL https://pi4j.com/install | sudo bash
```

By using this method, you can also update to newer versions later with the following commands:

```
1 sudo apt-get update
2 sudo update-get upgrade
```

## Programming with Pi4J

As all programs created with Pi4J can only run on the Pi itself, these examples were developed with Visual Studio Code directly on a Pi 3 B+ as this is the one which Pi4J V1.2 fully supports.

### Sources



All of the sources for these example applications are available in:  
Chapter\_09\_Pi4J

For example the first example with the RGB-LED:  
Chapter\_09\_Pi4J > java-pi4j-digital-ouput-led

The electronic schemes designed with Fritzing can be found in:  
Chapter\_09\_Pi4J > schemes

Some of the examples also use an Arduino and the code for these projects can also be found in this same directory, e.g.:  
Chapter\_09\_Pi4J > arduino-serial

### Maven dependencies

To add Pi4J to a Java Maven project we need to add at least this dependency in the pom file:

```
1 <dependency>
2     <groupId>com.pi4j</groupId>
3     <artifactId>pi4j-core</artifactId>
4     <version>1.2</version>
5 </dependency>
```

Some of the examples need additional dependencies. Check the pom.xml-file of each example in the sources to check the required dependencies. For example, when an LCD-display needs to be controlled, this additional one must be added:

```
1 <dependency>
2     <groupId>com.pi4j</groupId>
3     <artifactId>pi4j-device</artifactId>
4     <version>1.2</version>
5 </dependency>
```

The full list of Pi4J dependencies can be found in the [Maven Repository of com.pi4j<sup>100</sup>](#).

Also used in some of the examples: unit tests. We didn't use them a lot in this book yet, but they should be part of any good development! So where applicable, you can find some unit tests in this project not only as example code but also to validate the code of the examples itself of course. For the unit tests, this dependency is needed:

```
1 <dependency>
2   <groupId>junit</groupId>
3   <artifactId>junit</artifactId>
4   <version>4.12</version>
5   <scope>test</scope>
6 </dependency>
```

As you can see, an additional value is added “<scope>test</scope>” because this dependency should not be added to the final package, but is only needed during testing.

Some of the examples use jackson-databind to convert a JSON string to a Java object as further explained. This is the required dependency for these applications:

```
1 <dependency>
2   <groupId>com.fasterxml.jackson.core</groupId>
3   <artifactId>jackson-databind</artifactId>
4   <version>2.10.1</version>
5 </dependency>
```

---

<sup>100</sup><https://mvnrepository.com/artifact/com.pi4j/>



JSON = [JavaScript Object Notation](https://en.wikipedia.org/wiki/JSON)<sup>101</sup> is a commonly used way to structure data. For example:

```
1 {
2   "type": "measurement",
3   "data": {
4     "sensor1": 123,
5     "sensor2": 456
6   }
7 }
```

This format allows organizing data in a human-readable way which is also easy to be parsed by a programming language.

XML = [Extensible Markup Language](https://en.wikipedia.org/wiki/XML)<sup>102</sup> is another way to achieve this. The same data could be structured like this, but the formatting is up to the developer:

```
1 <message>
2   <type>measurement</type>
3   <dataset>
4     <data source="sensor1">123</data>
5     <data source="sensor2">456</data>
6   </dataset>
7 </message>
```

Which format is best, is open for discussion, but with the current frameworks, there is no big performance issue between the parsing of both. Just try to stick to one method.

## Running the examples

Some of the examples can be started directly from Visual Studio Code (VSC) on the Pi if you have Maven and the Java plugins installed in VSC.

---

<sup>101</sup><https://en.wikipedia.org/wiki/JSON>

<sup>102</sup><https://en.wikipedia.org/wiki/XML>



```

App.java - java-pi4j-digital-ouput-led - Visual Studio Code
File Edit Selection View Go Debug Terminal Help

EXPLORER
├─ OPEN EDITORS
│   └─ App.java src/main/java/be/webt...
├─ JAVA-PI4J-DIGITAL-OUPUT-LED
│   └─ .vscode
│       └─ src
│           └─ main
│               └─ java
│                   └─ be
│                       └─ webtechie
│                           └─ pi4jgpio
│                               └─ App.java
└─ target
    └─ pom.xml

App.java x
5  import com.pi4j.io.gpio.gpioPinDigitalOutput;
6  import com.pi4j.io.gpio.Pin;
7  import com.pi4j.io.gpio.PinState;
8  import com.pi4j.io.gpio.RaspiPin;
9
10 /**
11  * Based on https://www.pi4j.com/1.2/example/control.html
12  */
13 public class App {
14     private static final Pin PIN_RED = RaspiPin.GPIO_23; // BCM 13
15     private static final Pin PIN_GREEN = RaspiPin.GPIO_26; // BCM 12
16     private static final Pin PIN_BLUE = RaspiPin.GPIO_14; // BCM 11
17
18     public static void main( String[] args ) {
19         System.out.println("Starting output example...");
20
21         try {
22             // Initialize the GPIO controller
23             final GpioController gpio = GpioFactory.getInstance();

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```

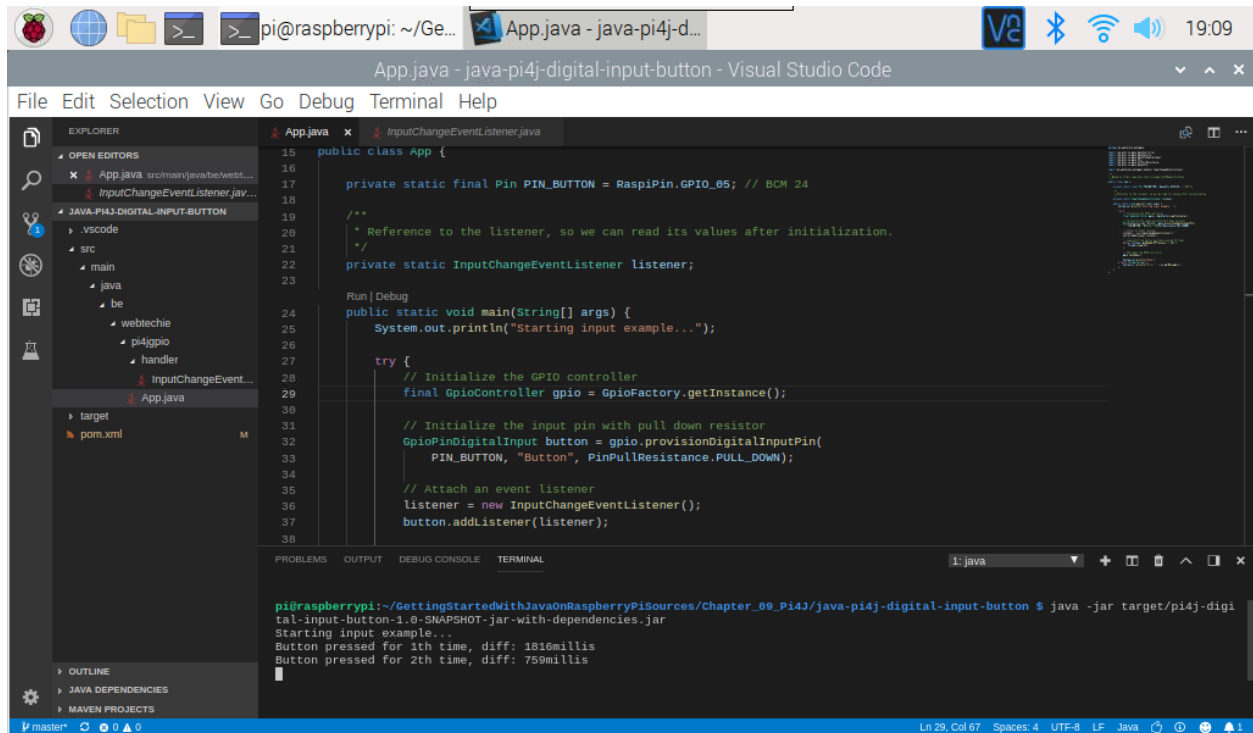
State of the LED BLUE has been toggled
State of the LED BLUE has been toggled
State of the LED BLUE has been toggled
State of the LED BLUE has been toggled
LED BLUE is off
All three on, check if this looks like white...

```

Start application from VSC on the Pi

Where the application needs more permissions, it needs to be started as a jar. You'll need to create the jar and start it from the terminal. For example:

- 1 \$ mvn clean package
- 2 \$ java -jar target/pi4j-digital-input-button-1.0-SNAPSHOT-jar-with-dependencies.jar
- 3 Starting...



```
public class App {
    private static final Pin PIN_BUTTON = RaspiPin.GPIO_05; // BCM 24

    /**
     * Reference to the listener, so we can read its values after initialization.
     */
    private static InputChangeListener listener;

    Run | Debug
    public static void main(String[] args) {
        System.out.println("Starting input example...");

        try {
            // Initialize the GPIO controller
            final GpioController gpio = GpioFactory.getInstance();

            // Initialize the input pin with pull down resistor
            GpioPinDigitalInput button = gpio.provisionDigitalInputPin(
                PIN_BUTTON, "Button", PinPullResistance.PULL_DOWN);

            // Attach an event listener
            listener = new InputChangeListener();
            button.addListener(listener);
        }
    }
}
```

```
pi@raspberrypi:~/GettingStartedWithJavaOnRaspberryPiSources/Chapter_09_PI4J/java-pi4j-digital-input-button $ java -jar target/pi4j-digital-input-button-1.0-SNAPSHOT-jar-with-dependencies.jar
Starting input example...
Button pressed for 1th time, diff: 1816millis
Button pressed for 2th time, diff: 759millis
```

Start application from VSC terminal as jar

For some of the applications you'll also need to start with sudo:

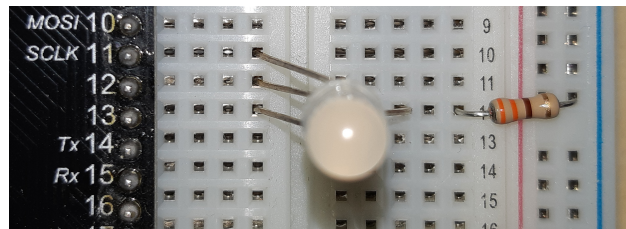
- 1 \$ sudo java -jar target/name-of-the-jar.jar

## Digital GPIO input and output examples

### Example 1: Digital output with RGB-LED

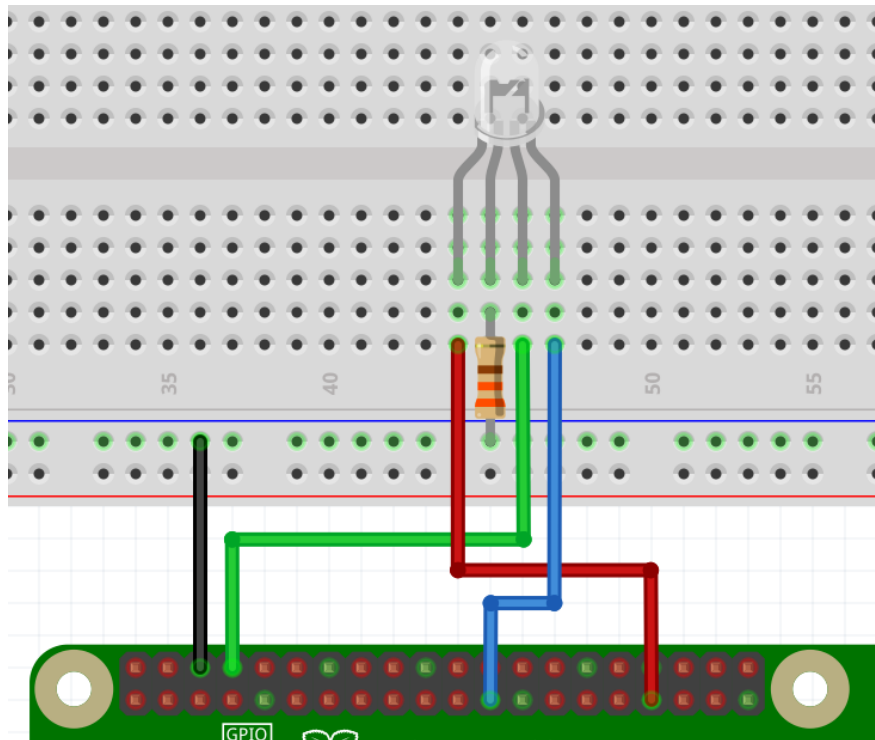
Let's start with the easiest example, using GPIOs as output pins. We will use three of them to control the red, green and blue of an RGB-LED.

#### Connecting an RGB-LED with a breadboard



Test setup with RGB-LED connected to Breadboard Pi Bridge

Make sure you have a common cathode LED by directly connecting one of the color pins to the 3.3V output of the Pi and use a resistor (330 $\Omega$  will work for most). Then connect each of the RGB pins to a GPIO and the common pin to a ground pin.



Digital output wiring with RGB-LED

## One class for all the code to control the RGB-LED

Follow the code to see how:

- The GPIO controller is initialized.
- The three GPIOs are initialized for each pin.
- Each color is toggled on/of 10 times.
- The three colors are all on for 5 seconds.
- The GPIO controller is shut down.

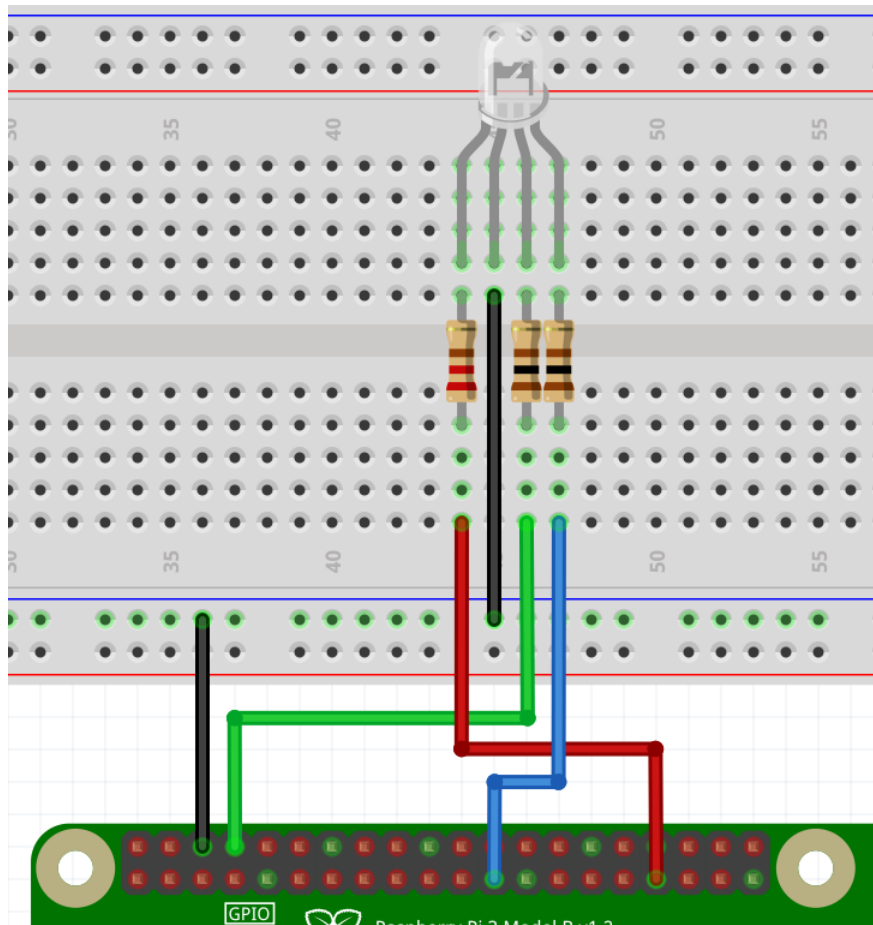
```
1 public class App {
2     private static final Pin PIN_RED = RaspiPin.GPIO_23;    // BCM 13
3     private static final Pin PIN_GREEN = RaspiPin.GPIO_26; // BCM 12
4     private static final Pin PIN_BLUE = RaspiPin.GPIO_14;  // BCM 11
5
6     public static void main( String[] args ) {
7         System.out.println("Starting output example...");
8
9         try {
10            // Initialize the GPIO controller
11            final GpioController gpio = GpioFactory.getInstance();
12
13            // Initialize the led pins as a digital output pin
14            // with initial low state
15            final GpioPinDigitalOutput ledRed =
16                gpio.provisionDigitalOutputPin(PIN_RED, "RED",
17                    PinState.LOW);
18            final GpioPinDigitalOutput ledGreen =
19                gpio.provisionDigitalOutputPin(PIN_GREEN, "GREEN",
20                    PinState.LOW);
21            final GpioPinDigitalOutput ledBlue =
22                gpio.provisionDigitalOutputPin(PIN_BLUE, "BLUE",
23                    PinState.LOW);
24
25            // Set the shutdown state
26            ledRed.setShutdownOptions(true, PinState.LOW);
27            ledGreen.setShutdownOptions(true, PinState.LOW);
28            ledBlue.setShutdownOptions(true, PinState.LOW);
29
30            // Toggle 10 times RED on and off
31            for (int led = 1; led <= 3; led++) {
32                GpioPinDigitalOutput useLed = led == 1 ?
```

```
33         ledRed : (led == 2 ? ledGreen : ledBlue);
34
35         for (int i = 0; i < 10; i++) {
36             useLed.toggle();
37             Thread.sleep(150);
38
39             System.out.println("State of the LED " + useLed.getName()
40                 + " has been toggled");
41         }
42
43         // Make sure the led is off
44         useLed.low();
45         System.out.println("LED " + useLed.getName() + " is off");
46     }
47
48     Thread.sleep(1000);
49
50     // All three on, should be white
51     ledRed.high();
52     ledGreen.high();
53     ledBlue.high();
54
55     System.out.println("All three on, check if looks white...");
56
57     Thread.sleep(5000);
58
59     // Shut down the GPIO controller
60     gpio.shutdown();
61
62     System.out.println("Done");
63 } catch (Exception ex) {
64     System.err.println("Error: " + ex.getMessage());
65 }
66 }
67 }
```

Depending on the type of RGB-LED each color needs another resistor value because each one has a different operating voltage. So this quick example with one resistor on the cathode side causes in my case to have a very dominant, bright red. When all three GPIOs are high, the mixed color is red instead of white.

## Correct wiring for a white result

To achieve a correct color mix, you will need to check the datasheet of your RGB-LED to calculate the correct resistor for each pin. But when setting up this experiment, as there was no article number or datasheet available, the trial-and-error approach has been used... With a  $220\Omega$  resistor between the GPIO and the red pin and a  $100\Omega$  one on the green and blue pin, the mixed color is nearly white. The adjusted wiring looks like this:



Digital output wiring with RGB-LED

## Example 2: Digital input with a button

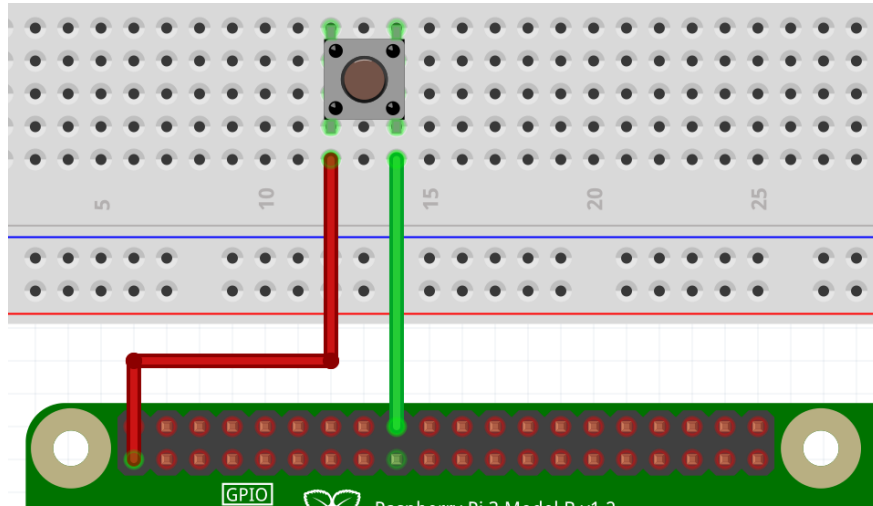
In this example, we use a push button to change the input state of a GPIO. When the button is pressed we show a log message in the console and after 10 button presses, the program is terminated.

Why? Good question... :-)

We will use this approach to prove why a GPIO input can and needs to be configured to use the integrated pull-down resistor!

## Wiring to connect a push-button

If you don't know how to connect this kind of button, use the fixed 3.3V and an LED to detect how to push the button into the breadboard. You have a 50% chance to be first-time-right. In my case second-time-right...



Digital input wiring with button

## Code to read a GPIO state

### Adding a change handler

To simplify the code we add a separate class that implements `GpioPinListenerDigital`. In this class, we can count the number of presses and log the time difference with the previous press.

```

1  /**
2   * Listener which will be called each time the button is pressed.
3   */
4  public class InputChangeListener implements GpioPinListenerDigital {
5      private int numberOfPresses = 0;
6      private long lastPress = System.currentTimeMillis();
7
8      /**
9       * Event handler
10     */
11     @Override
12     public void handleGpioPinDigitalStateChangeEvent(
13         GpioPinDigitalStateChangeEvent event) {
14
15         if (event.getState() == PinState.HIGH) {

```

```

16         this.numberOfPresses++;
17
18         long diff = System.currentTimeMillis() - this.lastPress;
19
20         System.out.println("Button pressed for "
21             + this.numberOfPresses + "th time, diff: "
22             + diff + "millis");
23
24         this.lastPress = System.currentTimeMillis();
25     }
26 }
27
28 /**
29  * @return the number of times the button has been pressed
30  */
31 public int getNumberOfPresses() {
32     return this.numberOfPresses;
33 }
34 }

```

## The main class to initialize the GPIO input

The main class contains all the initialization code:

- Define the input GPIO number, in this example WiringPi n° 5 = BCM 24.
- Initialize GpioPinDigitalInput and add the listener class.
- Wait until the button is pressed 10 times.

```

1 public class App {
2     private static final Pin PIN_BUTTON = RaspiPin.GPIO_05; // BCM 24
3
4     /**
5      * Reference to the listener, so we can read its values after initialization.
6      */
7     private static InputChangeListener listener;
8
9     public static void main(String[] args) {
10         System.out.println("Starting input example...");
11
12         try {
13             // Initialize the GPIO controller
14             final GpioController gpio = GpioFactory.getInstance();

```



```
15
16     // Initialize the input pin with pull down resistor
17     GpioPinDigitalInput button = gpio.provisionDigitalInputPin(
18         PIN_BUTTON, "Button", PinPullResistance.PULL_DOWN);
19
20     // Attach an event listener
21     listener = new InputChangeListener();
22     button.addListener(listener);
23
24     // Loop until the button has been pressed 10 times
25     while (listener.getNumberOfPresses() < 10) {
26         Thread.sleep(10);
27     }
28
29     // Shut down the GPIO controller
30     gpio.shutdown();
31
32     System.out.println("Done");
33 } catch (Exception ex) {
34     System.err.println("Error: " + ex.getMessage());
35 }
36 }
37 }
```

## Running this code

We test this application with “mvn clean package” and “java -jar”. When trying to push the button every second, the result looks like this:

```
1 $ mvn clean package
2 $ java -jar target/pi4j-dital-input-button-1.0-SNAPSHOT-jar-with-dependencies.jar
3 Starting input example...
4 Button pressed for 1th time, diff: 1635millis
5 Button pressed for 2th time, diff: 811millis
6 Button pressed for 3th time, diff: 944millis
7 Button pressed for 4th time, diff: 1073millis
8 Button pressed for 5th time, diff: 1009millis
9 Button pressed for 6th time, diff: 1079millis
10 Button pressed for 7th time, diff: 978millis
11 Button pressed for 8th time, diff: 236millis
12 Button pressed for 9th time, diff: 994millis
13 Button pressed for 10th time, diff: 1117millis
14 Done
```

## Without the PULL\_DOWN

Let's try what a pull-down resistor adds as functionality as described in "Chapter 5: Raspberry Pi pinning > Pin types > Digital GPIO".

Go back to the main class and change the initialization of `GpioPinDigitalInput` to:

```
1 GpioPinDigitalInput button = gpio.provisionDigitalInputPin(
2     PIN_BUTTON, "Button", PinPullResistance.OFF);
```

Now run the application the same way without touching the button and you'll get this kind of output:

```
1 $ mvn clean package
2 $ java -jar target/pi4j-digital-input-button-1.0-SNAPSHOT-jar-with-dependencies.jar
3 Starting input example...
4 Button pressed for 7th time, diff: 349millis
5 Button pressed for 2th time, diff: 229millis
6 Button pressed for 8th time, diff: 369millis
7 Button pressed for 6th time, diff: 330millis
8 Button pressed for 1th time, diff: 221millis
9 Button pressed for 5th time, diff: 314millis
10 Button pressed for 4th time, diff: 271millis
11 Button pressed for 3th time, diff: 249millis
12 Button pressed for 9th time, diff: 2millis
13 Button pressed for 10th time, diff: 20millis
14 Done
```

The GPIO toggles so fast the output gets completely messed up. As you can see the log output gives unreliable results.

When using a pull-up resistor, no button presses are detected at all.

```
1 gpioPinDigitalInput button = gpio.provisionDigitalInputPin(
2     PIN_BUTTON, "Button", PinPullResistance.PULL_UP);
```

So the `PULL_DOWN` value is the one to be used in this case!

## Example 3: Distance sensor

In this example, we'll be using an ultrasound distance sensor which you can find in a lot of starter kits for Arduino and Pi. This is a module known as "HC-SR04" for which you can find a lot of info and examples on-line. It needs both an input and output GPIO. This sensor works the same way as

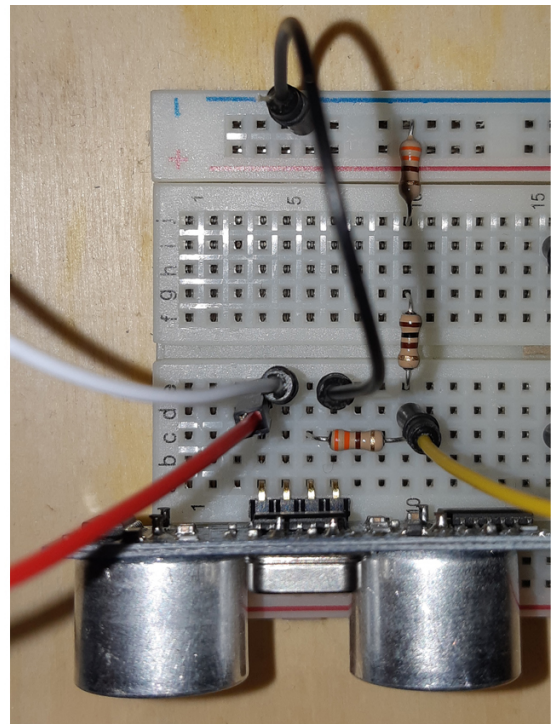
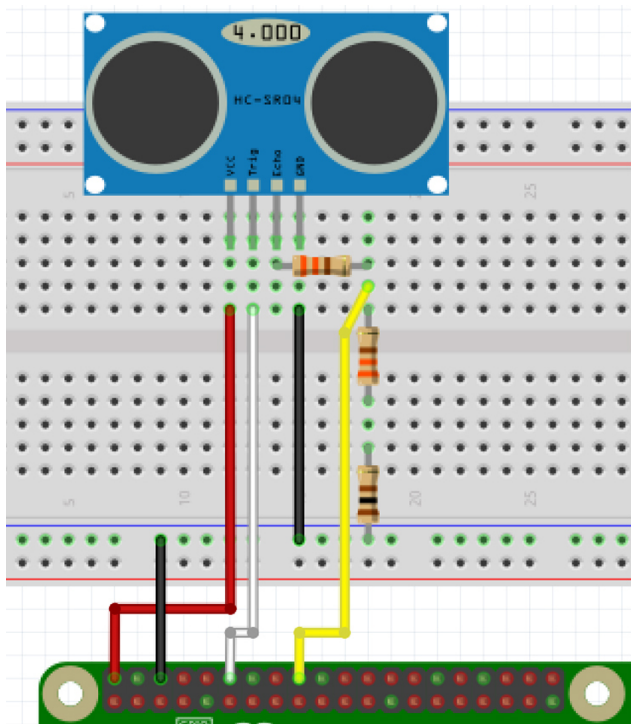
the system used by a bat to fly in the dark without hitting walls. The reflection of ultrasound is used to calculate the distance to an object on which the sound reflects.

To measure the distance, these steps must be taken:

- The module needs to be powered with 5V.
- The application needs to put the trigger pin high for at least 10 $\mu$ s.
- The module will send multiple (typically 8) 40kHz signals and detect when the signal is received back.
- The echo pin will be set high for the same duration as the ultrasound needed for returning.
- The application needs to measure the duration of the high state of the echo pin so it can calculate the distance based on the speed of sound.

### Wiring between Pi and distance sensor

As this sensor module works on 5V, we need to limit the current going back to the Pi to not damage the GPIO which expects max 3.3V. This is done with the resistors which also connect the echo pin to the ground to have a clear difference between high and low (pull-down).



Connections for the distance sensor

## Code blocks

### Calculation helper class

As we will need to do some calculations on the measured results, a separate “Calculation.java” class is used to provide these methods.

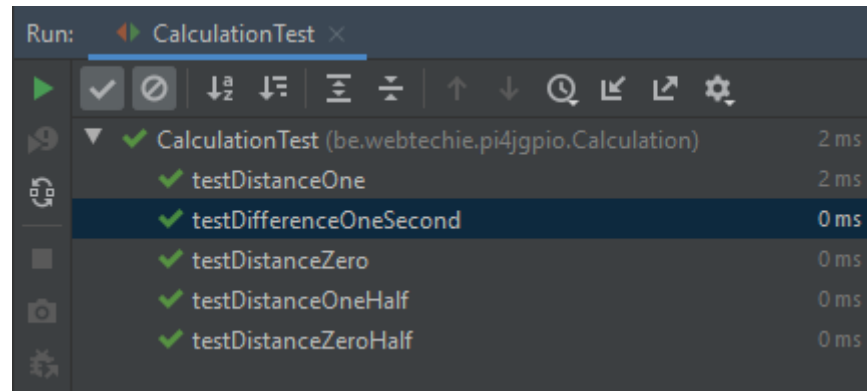
```
1  /**
2   * Helper class for duration and distance calculation.
3   */
4  public class Calculation {
5      /**
6       * Get the distance (in cm) for a given duration.
7       * The calculation is based on the speed of sound which is 34300 cm/s.
8       *
9       * @param seconds Number of seconds
10      * @param half Flag to define if the calculated distance must be divided
11      */
12     public static int getDistance(float seconds, boolean half) {
13         float distance = seconds * 34300;
14         return Math.round(half ? distance / 2 : distance);
15     }
16
17     /**
18      * Get the number of seconds between two nanosecond timestamps.
19      * 1 second = 1000000000 nanoseconds
20      *
21      * @param start Start timestamp in nanoseconds
22      * @param end End timestamp in nanoseconds
23      */
24     public static float getSecondsDifference(long start, long end) {
25         return (end - start) / 1000000000F;
26     }
27 }
```

### Unit test

To validate these functions, a unit test “CalculationTest.java” is also added to the example code which checks the results of these methods. “assertEquals” checks if the test result is OK.

```
1 public class CalculationTest {
2     @Test
3     public void testDistanceZero() {
4         assertEquals(0, Calculation.getDistance(0, false));
5     }
6
7     @Test
8     public void testDistanceZeroHalf() {
9         assertEquals(0, Calculation.getDistance(0, true));
10    }
11
12    @Test
13    public void testDistanceOne() {
14        assertEquals(34300, Calculation.getDistance(1, false));
15    }
16
17    @Test
18    public void testDistanceOneHalf() {
19        assertEquals(34300 / 2, Calculation.getDistance(1, true));
20    }
21
22    @Test
23    public void testDifferenceOneSecond() {
24        long now = System.nanoTime();
25        // Compare two floats, which in this case need to match
26        // with a precision (delta) 1
27        assertEquals(1F, Calculation.getSecondsDifference(now, now + 1000000000), 1);
28    }
29 }
```

When you run this class, your IDE will show the result for each test separately:



Unit test results in IntelliJ IDEA

## Application code

Now let's write the application code. In the "main"-method the pins are initialized. We keep looping to do continuous distance measurements every 2 seconds. The call to "measureDistance()" allows us to keep this method very clean and isolate the measurement code.

```

1 public class App {
2     private static final Pin PIN_TRIGGER = RaspiPin.GPIO_01;    // BCM 18
3     private static final Pin PIN_ECHO = RaspiPin.GPIO_05;      // BCM 24
4
5     private static GpioPinDigitalOutput trigger;
6     private static GpioPinDigitalInput echo;
7
8     public static void main(String[] args) {
9         System.out.println("Starting distance sensor example...");
10
11         try {
12             // Initialize the GPIO controller
13             GpioController gpio = GpioFactory.getInstance();
14
15             // Initialize the pins
16             trigger = gpio.provisionDigitalOutputPin(PIN_TRIGGER,
17             "Trigger", PinState.LOW);
18             echo = gpio.provisionDigitalInputPin(PIN_ECHO,
19             "Echo", PinPullResistance.PULL_UP);
20
21             // Loop and measure the distance 5 times per second
22             while (true) {
23                 measureDistance();
24

```

```

25         Thread.sleep(2000);
26     }
27     } catch (Exception ex) {
28         System.err.println("Error: " + ex.getMessage());
29     }
30 }
31 }

```

Of course, we still need to add the `measureDistance`-method. In the example code, it is part of the same `App`-class but you could also put it in a separate one. Within this method, each step is taken as described at the beginning of this example to put the trigger pin high and measure the duration between the state changes of the echo pin. Using the methods created before in the `Calculation`-class we can now log the distance between the distance sensor and the nearest object.

```

1  private static void measureDistance() {
2      try {
3          // Set trigger high for 0.01ms
4          trigger.pulse(10, PinState.HIGH, true, TimeUnit.NANOSECONDS);
5
6          // Start the measurement
7          while (echo.isLow()) {
8              // Wait until the echo pin is high,
9              // indicating the ultrasound was sent
10         }
11         long start = System.nanoTime();
12
13         // Wait till measurement is finished
14         while (echo.isHigh()) {
15             // Wait until the echo pin is low,
16             // indicating the ultrasound was received back
17         }
18         long end = System.nanoTime();
19
20         // Output the distance
21         float measuredSeconds = Calculation.getSecondsDifference(start, end);
22         System.out.println("Measured distance is: "
23             + Calculation.getDistance(measuredSeconds, true) + "cm"
24             + " for " + measuredSeconds + "s");
25     } catch (Exception ex) {
26         System.err.println("Error: " + ex.getMessage());
27     }
28 }

```

## Running the distance sensor application

When we build and run the application, we get a log-line every 2 seconds with the measurement between the distance sensor and ... euh ... for this example my hand going back and forth :-)

```
1 $ mvn clean package
2 $ java -jar target/pi4j-distancesensor-1.0-SNAPSHOT-jar-with-dependencies.jar
3 Starting distance sensor example...
4 Measured distance is: 29cm for 0.001689683s
5 Measured distance is: 16cm for 9.47862E-4s
6 Measured distance is: 6cm for 3.38697E-4s
7 Measured distance is: 13cm for 7.56821E-4s
8 Measured distance is: 23cm for 0.001337653s
9 Measured distance is: 30cm for 0.001767964s
10 Measured distance is: 279cm for 0.016285324s
```

Oh, and my test setup is about 3m from the wall as the last measurement shows.



## PWM example

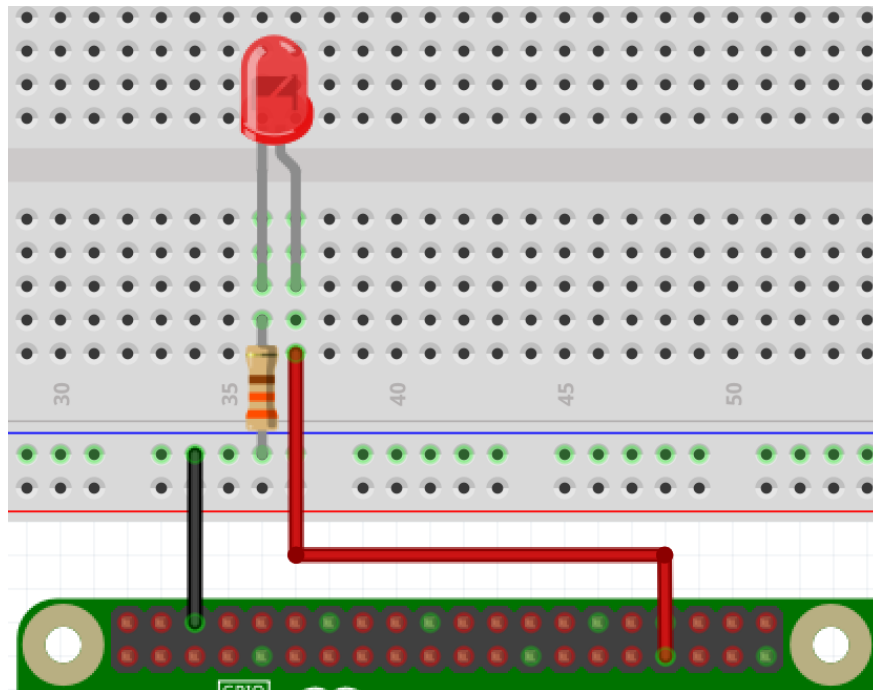
As described in “Chapter 5: Raspberry Pi pinning”, PWM (Pulse-Width Modulation) can be used to create a “semi-analog output”.

In this example, we are going to control the brightness of LED by using a PWM signal.

### Wiring a single LED

The simplest wiring diagram in this book! ;-)

We only need one LED and a resistor, use one of type  $330\Omega$  or calculate the exact type you need based on the datasheet of the LED and the formula you can find in “Chapter 2 > Hardware components > Resistors”. Connect the anode side (+) to WiringPi n° 1, BCM 18.



LED connected to BCM pin 18

### Code to control an LED with PWM

For this example, we only need the main class.

```
1 public class App {
2     private static final int MAX_PWM_VALUE = 1000;
3     private static final int FADE_STEPS = 10;
4     private static final Pin PIN_LED = RaspiPin.GPIO_01; // BCM 18
5
6     public static void main(String[] args) {
7         System.out.println("Starting PWM output example...");
8
9         try {
10            // Initialize the GPIO controller
11            GpioController gpio = GpioFactory.getInstance();
12
13            // All Raspberry Pi models support a hardware PWM pin on GPIO_01.
14            // Raspberry Pi models A+, B+, 2B, 3B also support hardware PWM pins:
15            // GPIO_23, GPIO_24, GPIO_26
16            GpioPinPwmOutput pwm = gpio.provisionPwmOutputPin(PIN_LED);
17
18            // You can optionally use these wiringPi methods to further customize
19            // the PWM generator see:
20            // http://wiringpi.com/reference/raspberry-pi-specifics/
21            com.pi4j.wiringpi.Gpio.pwmSetMode(com.pi4j.wiringpi.Gpio.PWM_MODE_MS);
22            com.pi4j.wiringpi.Gpio.pwmSetRange(1000);
23            com.pi4j.wiringpi.Gpio.pwmSetClock(50);
24
25            // Loop through PWM values 10 times
26            for (int loop = 0; loop < 10; loop++) {
27                for (int useValue = MAX_PWM_VALUE; useValue >= 0;
28                    useValue-=MAX_PWM_VALUE/FADE_STEPS) {
29                    pwm.setPwm(useValue);
30                    System.out.println("PWM rate is: " + pwm.getPwm());
31
32                    Thread.sleep(200);
33                }
34            }
35
36            // Shut down the GPIO controller
37            gpio.shutdown();
38
39            System.out.println("Done");
40        } catch (Exception ex) {
41            System.err.println("Error: " + ex.getMessage());
42        }
43    }
```

```
44 }
```

## Running the example

To be able to control the PWM, we need to start this application with “sudo”. The LED will be dimmed three times from the maximum to 0 value.

```
1 $ mvn clean package
2 $ sudo java -jar target/pi4j-pwm-output-led-1.0-SNAPSHOT-jar-with-dependencies.jar
3 Starting PWM output example...
4 PWM rate is: 1000
5 PWM rate is: 900
6 PWM rate is: 800
7 PWM rate is: 700
8 PWM rate is: 600
9 PWM rate is: 500
10 PWM rate is: 400
11 PWM rate is: 300
12 PWM rate is: 200
13 PWM rate is: 100
14 PWM rate is: 0
15 PWM rate is: 1000
16 PWM rate is: 900
17 ...
```

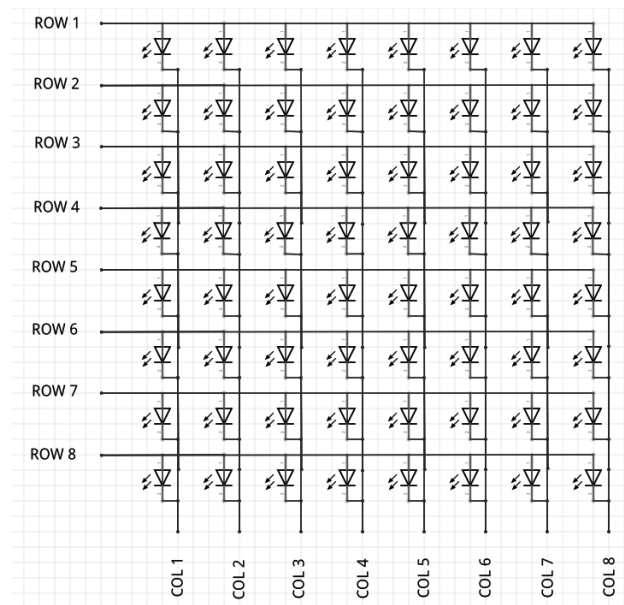
## SPI example with MAX7219 and 8x8 LED-matrix

A perfect component to experiment with the SPI connection (see “Chapter 5 > Pin functions > Serial Peripheral Interface (SPI)”) is the component you can also find in a lot of starter boxes which combines an 8x8 LED-matrix with a MAX7219 chip.

This chip can be controlled via SPI. You send it a command and don’t need to care about timings to control the LEDs as the chip takes care of this. Once the command is sent, and the chip is powered, the requested output is shown and you don’t need to control it anymore.

You can also connect multiple of these boards in a daisy-chain to create a long display. But make sure to use an external power supply to not draw too much current from the Pi. We will use one in this example and create some helper functions to work with images and characters.

The same MAX-chip is also available in combination with up to 8 LED-number displays. Or you can build your combination of multiple LEDs in any shape. There are a lot of examples projects you can find online with 3D cubes with 3x3x3 or more LEDs (search for “LED 3D cube”).

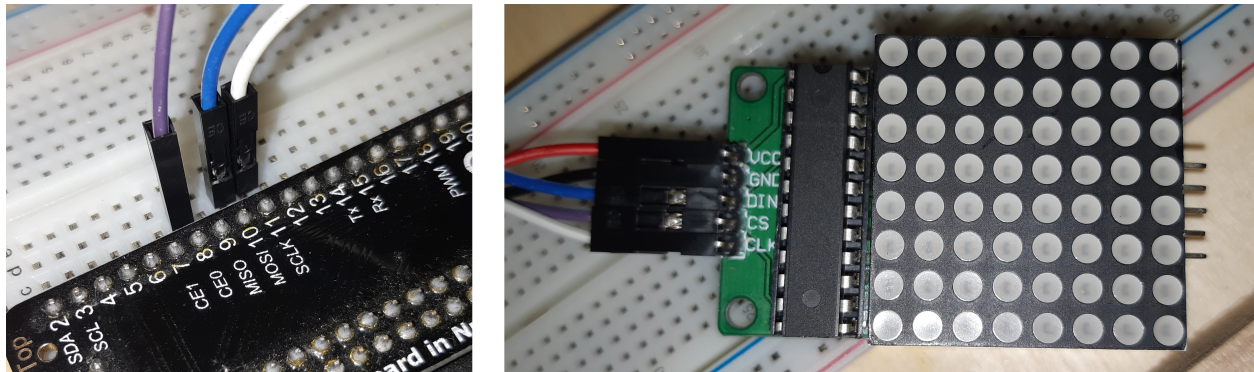


Connections between the LEDs in a matrix

The LEDs inside the matrix are connected as shown in the previous image. By putting power on a row, and connecting a column to the ground, a specific LED can be put on. When this is done faster than the eye can see, multiple LEDs look to be continuously powered while they are actually toggled very fast.

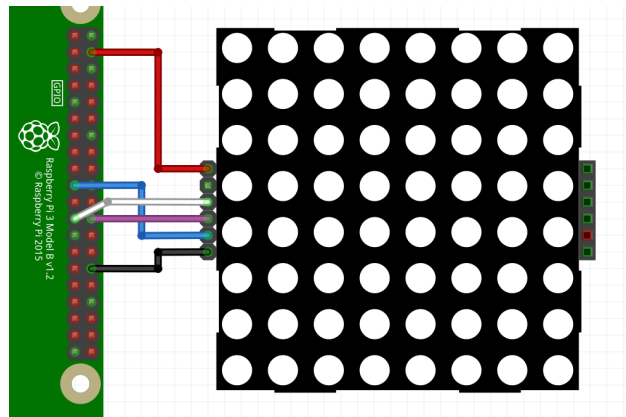
### Wiring

SPI requires four connections, but as we don’t read data back, we only connect three of them plus power and ground.



Test setup

- BCM 10 (MOSI) to DIN
- BCM 11 (SCLK) to CLK
- BCM 8 (CE0) to CS
- 5V to VCC
- Ground to ground



SPI wiring scheme

## SPI example code

The chip can be controlled by sending two byte-values. The first one is a command which is either the row number (1 to 8) or one of the predefined functions (see table). The next byte defines the action. For a row, it's the on or off value for each column as bits. We know from “Chapter 8: Bits and Bytes” that 8 bits fit in one HEX-value, so that's exactly the number of LEDs in a row.

The full specification of the MAX can be found in the [datasheet<sup>103</sup>](https://www.sparkfun.com/datasheets/Components/General/COM-09622-MAX7219-MAX7221.pdf), but this is a short version related to the example application we are going to build.

<sup>103</sup><https://www.sparkfun.com/datasheets/Components/General/COM-09622-MAX7219-MAX7221.pdf>

| Command | Description                                  | Values                                               |
|---------|----------------------------------------------|------------------------------------------------------|
| 0x01    | Row 1                                        | 0x00 - 0xFF                                          |
| 0x02    | Row 2                                        | 0x00 - 0xFF                                          |
| ...     | ...                                          | ...                                                  |
| 0x08    | Row 8                                        | 0x00 - 0xFF                                          |
| 0x09    | Decode mode                                  | 0x00 needed in our case to use all bits of the value |
| 0x0A    | Brightness                                   | 0x00 (min) - 0x0F (max)                              |
| 0x0B    | Scan limit                                   | 0x0F needed in our example                           |
| 0x0C    | Let the chip listen (or not) to SPI commands | 0x00: shutdown mode - 0x01: normal mode              |
| 0x0F    | Test mode                                    | 0x00 (off) - 0x01 (on)                               |

Again, to make everything more clear, the code is split into small blocks.

## Definitions

Let's first start with a list of commands, not including the rows as we will just use the row number later so don't need to define a command for them.

```

1  /**
2   * Reserved byte values to send a command.
3   */
4  public enum SpiCommand {
5      DECODE_MODE((byte) 0x09),
6      BRIGHTNESS((byte) 0x0A),
7      SCAN_LIMIT((byte) 0x0B),
8      SHUTDOWN_MODE((byte) 0x0C),
9      TEST((byte) 0x0F);
10
11     private final byte value;
12
13     SpiCommand(byte value) {
14         this.value = value;
15     }
16
17     public byte getValue() {
18         return value;
19     }
20 }

```

Two types of enums are also created: images and characters, with an image always being 8 by 8 in size, but the characters can have a different width, depending on the number of columns needed to draw the character.

Let's start with the images. Only two are included here, but there are more in the sources in the GitHub-repository and of course, you can add as many as you want! We need byte values, but it's more clear to create these as strings with 0s and 1s for each LED. These string values are converted to a byte for each row, so we need 8 of them for each image.

```

1  public enum Image {
2      HEART(Arrays.asList(
3          (byte) Integer.parseInt("00100100", 2),
4          (byte) Integer.parseInt("01111110", 2),
5          (byte) Integer.parseInt("11111111", 2),
6          (byte) Integer.parseInt("11111111", 2),
7          (byte) Integer.parseInt("01111110", 2),
8          (byte) Integer.parseInt("00111100", 2),
9          (byte) Integer.parseInt("00111100", 2),
10         (byte) Integer.parseInt("00011000", 2)
11     )),
12     ...
13     CROSS(Arrays.asList(
14         (byte) Integer.parseInt("10000001", 2),
15         (byte) Integer.parseInt("01000010", 2),
16         (byte) Integer.parseInt("00100100", 2),
17         (byte) Integer.parseInt("00011000", 2),
18         (byte) Integer.parseInt("00011000", 2),
19         (byte) Integer.parseInt("00100100", 2),
20         (byte) Integer.parseInt("01000010", 2),
21         (byte) Integer.parseInt("10000001", 2)
22     ));
23
24     private final List<Byte> rows;
25
26     Image(List<Byte> rows) {
27         this.rows = rows;
28     }
29
30     public List<Byte> getRows() {
31         return rows;
32     }
33 }

```

A little trick, when you do a “Find” on “1” in your IDE, the created image becomes a lot more clear!

```

36 )),
37  CROSS(Arrays.asList(
38     (byte) Integer.parseInt("0000001", radix: 2),
39     (byte) Integer.parseInt("0000010", radix: 2),
40     (byte) Integer.parseInt("0000100", radix: 2),
41     (byte) Integer.parseInt("0001000", radix: 2),
42     (byte) Integer.parseInt("0010000", radix: 2),
43     (byte) Integer.parseInt("0100000", radix: 2),
44     (byte) Integer.parseInt("1000000", radix: 2),
45     (byte) Integer.parseInt("0000001", radix: 2)
46 ));

```

Highlighting the 1s in the IDE

Similarly, the characters are created, with additional info about ASCII-code and the number of columns used for the character. Check for instance the difference between “B” (6 columns) and “T” (5 columns).

ASCII is the standard that defines a value for each character. The first 32 are unprintable control codes like backspace, tab, line break etc. Codes 32 till 127 are used for letters, digits, punctuation marks, miscellaneous symbols... Starting from 128 is extended ASCII with special characters like é, ®, ¾.

A quick overview:

| DEC | HEX  | Character |
|-----|------|-----------|
| 32  | 0x20 | Space     |
| ... |      |           |
| 40  | 0x28 | (         |
| 41  | 0x29 | )         |
| ... |      |           |
| 48  | 0x30 | 0         |
| 49  | 0x31 | 1         |
| 50  | 0x32 | 2         |
| ... |      |           |
| 65  | 0x41 | A         |
| 66  | 0x42 | B         |
| 67  | 0x43 | C         |
| ... |      |           |
| 97  | 0x61 | a         |
| 98  | 0x62 | b         |
| 99  | 0x63 | c         |

Building a character on a matrix can be challenging. So again only a few examples and more homework for you :-)



```
1 public enum AsciiCharacter {
2     ...
3     B(0x42, 6, Arrays.asList(
4         (byte) Integer.parseInt("11111000", 2),
5         (byte) Integer.parseInt("10000100", 2),
6         (byte) Integer.parseInt("10000100", 2),
7         (byte) Integer.parseInt("11111000", 2),
8         (byte) Integer.parseInt("10000100", 2),
9         (byte) Integer.parseInt("10000100", 2),
10        (byte) Integer.parseInt("10000100", 2),
11        (byte) Integer.parseInt("11111000", 2)
12    )),
13    ...
14    T(0x54, 5, Arrays.asList(
15        (byte) Integer.parseInt("11111000", 2),
16        (byte) Integer.parseInt("00100000", 2),
17        (byte) Integer.parseInt("00100000", 2),
18        (byte) Integer.parseInt("00100000", 2),
19        (byte) Integer.parseInt("00100000", 2),
20        (byte) Integer.parseInt("00100000", 2),
21        (byte) Integer.parseInt("00100000", 2),
22        (byte) Integer.parseInt("00100000", 2)
23    ));
24
25    private final int ascii;
26    private final int numberOfColumns;
27    private final List<Byte> rows;
28
29    AsciiCharacter(int ascii, int numberOfColumns, List<Byte> rows) {
30        this.ascii = ascii;
31        this.numberOfColumns = numberOfColumns;
32        this.rows = rows;
33    }
34
35    public int getAscii() {
36        return ascii;
37    }
38
39    public int getNumberOfColumns() {
40        return numberOfColumns;
41    }
42
43    public List<Byte> getRows() {
```

```

44     return rows;
45 }
46
47 public static AsciiCharacter getByAscii(int ascii) {
48     for (AsciiCharacter asciiCharacter : AsciiCharacter.values()) {
49         if (asciiCharacter.getAscii() == ascii) {
50             return asciiCharacter;
51         }
52     }
53     return null;
54 }
55
56 public static AsciiCharacter getByChar(char character) {
57     return getByAscii(character);
58 }
59 }

```

```

B( ascii: 0x42, numberOfColumns: 6, Arrays.asList(
    (byte) Integer.parseInt( s: "11111000", radix: 2),
    (byte) Integer.parseInt( s: "10000100", radix: 2),
    (byte) Integer.parseInt( s: "10000100", radix: 2),
    (byte) Integer.parseInt( s: "11111000", radix: 2),
    (byte) Integer.parseInt( s: "10000100", radix: 2),
    (byte) Integer.parseInt( s: "10000100", radix: 2),
    (byte) Integer.parseInt( s: "10000100", radix: 2),
    (byte) Integer.parseInt( s: "11111000", radix: 2)
)),

```

```

T( ascii: 0x54, numberOfColumns: 5, Arrays.asList(
    (byte) Integer.parseInt( s: "11111000", radix: 2),
    (byte) Integer.parseInt( s: "00100000", radix: 2),
    (byte) Integer.parseInt( s: "00100000", radix: 2),
    (byte) Integer.parseInt( s: "00100000", radix: 2),
    (byte) Integer.parseInt( s: "00100000", radix: 2),
    (byte) Integer.parseInt( s: "00100000", radix: 2),
    (byte) Integer.parseInt( s: "00100000", radix: 2),
    (byte) Integer.parseInt( s: "00100000", radix: 2)
));

```

Width of B versus T

The ASCII code could be used to find the matching enum for a given character if you want to extend the application with a method to display a certain text on the matrix display. To get that code we can use the “charAt”-method:

```

1  jshell> String test = "ABC"
2  test ==> "ABC"
3
4  jshell> test.charAt(0);
5  $2 ==> 'A'
6
7  jshell> char c = test.charAt(1);
8  c ==> 'B'
9
10 jshell> (int) c
11 $4 ==> 66
12

```

```

13 jshell> "0x" + Integer.toHexString(c);
14 $8 ==> "0x42"

```

## Helpers

Sending visual output to the matrix via the SPI through the chip is split into three classes, so you can easily play with it, add other methods, etc.

First, there is a general “Demo” one. Check the JavaDoc on each method and the logging as these explain what each one is doing.

```

1  public class DemoMode {
2      /**
3       * Highlight all rows one by one.
4       *
5       * @param spi SpiDevice
6       * @param waitBetween Number of milliseconds to wait between every row output
7       */
8      public static void showRows(SpiDevice spi, int waitBetween) {
9          try {
10             for (int onRow = 1; onRow <= 8; onRow++) {
11                 for (int row = 1; row <= 8; row++) {
12                     spi.write((byte) row,
13                             (onRow == row ? (byte) 0xff : (byte) 0x00));
14                 }
15                 System.out.println("Row " + onRow + " is on");
16                 Thread.sleep(waitBetween);
17             }
18         } catch (Exception ex) {
19             System.err.println("Error during row demo: " + ex.getMessage());
20         }
21     }
22
23     /**
24     * Highlight all columns one by one.
25     *
26     * @param spi SpiDevice
27     * @param waitBetween Number of milliseconds to wait between every column output
28     */
29     public static void showCols(SpiDevice spi, int waitBetween) {
30         try {
31             for (int onColumn = 0; onColumn < 8; onColumn++) {
32                 for (int row = 1; row <= 8; row++) {

```

```

33         spi.write((byte) row, (byte) (1 << (8 - onColumn)));
34     }
35     System.out.println("Col " + onColumn + " is on");
36     Thread.sleep(waitBetween);
37 }
38 } catch (Exception ex) {
39     System.err.println("Error during column demo: " + ex.getMessage());
40 }
41 }
42
43 /**
44  * Demo mode which generates specified number of cycles of random enabled LEDs.
45  *
46  * @param spi SpiDevice
47  * @param numberOfLoops Number of random outputs to be generated
48  * @param waitBetween Number of milliseconds to wait between random screens
49  */
50 public static void showRandomOutput(SpiDevice spi, int numberOfLoops,
51     int waitBetween) {
52     try {
53         Random r = new Random();
54         int min = 0;
55         int max = 255;
56
57         for (int loop = 1; loop <= numberOfLoops; loop++) {
58             for (int row = 1; row <= 8; row++) {
59                 spi.write((byte) row, (byte) (r.nextInt((max - min) + 1) + min));
60             }
61             System.out.println("Random effect " + loop);
62             Thread.sleep(waitBetween);
63         }
64     } catch (Exception ex) {
65         System.err.println("Error during random demo: " + ex.getMessage());
66     }
67 }
68 }

```

A similar class is available for the Images:

```
1 public class ImageMode {
2     /**
3      * Show all the images as defined in the enum.
4      *
5      * @param spi SpiDevice
6      * @param waitBetween Number of milliseconds to wait between every image output
7      */
8     public static void showAllImages(SpiDevice spi, int waitBetween) {
9         try {
10            for (Image image : Image.values()) {
11                showImage(spi, image);
12                System.out.println("Showing image " + image.name());
13                Thread.sleep(waitBetween);
14            }
15        } catch (Exception ex) {
16            System.err.println("Error during images: " + ex.getMessage());
17        }
18    }
19
20    /**
21     * Output the given image to the matrix.
22     *
23     * @param spi SpiDevice
24     * @param image Image to be shown
25     */
26    public static void showImage(SpiDevice spi, Image image) {
27        try {
28            for (int i = 0; i < 8; i++) {
29                spi.write((byte) (i + 1), image.getRows().get(i));
30            }
31        } catch (Exception ex) {
32            System.err.println("Error during images: " + ex.getMessage());
33        }
34    }
35 }
```

And at last for the ASCII characters.

```
1 public class AsciiCharacterMode {
2     /**
3      * Show all the characters as defined in the alphabet enum.
4      *
5      * @param spi SpiDevice
6      * @param waitBetween Milliseconds between every AsciiCharacter
7      */
8     public static void showAllAsciiCharacters(SpiDevice spi, int waitBetween) {
9         try {
10            for (int ascii = 32; ascii <= 126; ascii++) {
11                AsciiCharacter asciiCharacter = AsciiCharacter.getByAscii(ascii);
12                if (asciiCharacter != null) {
13                    showAsciiCharacter(spi, asciiCharacter);
14                    System.out.println("Written to SPI : " + asciiCharacter.name());
15                    Thread.sleep(waitBetween);
16                }
17            }
18        } catch (Exception ex) {
19            System.err.println("Error during Ascii: " + ex.getMessage());
20        }
21    }
22
23    /**
24     * Output the given alphabet character to the display.
25     *
26     * @param spi SpiDevice
27     * @param asciiCharacter AsciiCharacter to be shown
28     */
29    public static void showAsciiCharacter(SpiDevice spi,
30        AsciiCharacter asciiCharacter) {
31        try {
32            for (int row = 0; row < 8; row++) {
33                spi.write((byte) (row + 1), asciiCharacter.getRows().get(row));
34            }
35        } catch (Exception ex) {
36            System.err.println("Error during images: " + ex.getMessage());
37        }
38    }
39
40    /**
41     * Show all the characters as defined in the alphabet enum.
42     *
43     * @param spi SpiDevice
```

```

44     * @param waitBetweenMove Milliseconds between every column move
45     */
46     public static void scrollAllAsciiCharacters(SpiDevice spi, int waitBetweenMove) {
47         try {
48             for (int ascii = 32; ascii <= 126; ascii++) {
49                 AsciiCharacter asciiCharacter = AsciiCharacter.getByAscii(ascii);
50                 if (asciiCharacter != null) {
51                     scrollAsciiCharacter(spi, asciiCharacter, waitBetweenMove);
52                     System.out.println("Scrolled : " + asciiCharacter.name());
53                     Thread.sleep(250);
54                 }
55             }
56         } catch (Exception ex) {
57             System.err.println("Error during Ascii: " + ex.getMessage());
58         }
59     }
60
61     /**
62     * Scroll a character over the screen.
63     *
64     * @param spi SpiDevice
65     * @param asciiCharacter AsciiCharacter to be scrolled
66     * @param waitBetweenMove Milliseconds between every column move
67     */
68     public static void scrollAsciiCharacter(SpiDevice spi,
69         AsciiCharacter asciiCharacter, int waitBetweenMove) {
70         try {
71             for (int move = 0; move < (8 * 2); move++) {
72                 for (int row = 0; row < 8; row++) {
73                     int rowValue = 0xFF & asciiCharacter.getRows().get(row);
74                     if (move < 8) {
75                         rowValue = 0xFF & (rowValue >> (8 - move));
76                     } else {
77                         rowValue = 0xFF & (rowValue << (move - 8));
78                     }
79                     spi.write((byte) (row + 1), (byte) rowValue);
80                 }
81                 Thread.sleep(waitBetweenMove);
82             }
83         } catch (Exception ex) {
84             System.err.println("Error during images: " + ex.getMessage());
85         }
86     }

```

87 }

## Main application code

That was all the hard work! Initializing the SPI and sending content is now a small remaining step we can do in the main-method where we call the different methods from the DemoMode-, ImageMode- and AsciiCharacterMode-classes.

```
1 public class App {
2     public static void main(String args[]) {
3         try {
4             System.out.println("Starting SPI example...");
5
6             // Initialize the SpiFactory
7             SpiDevice spi = SpiFactory.getInstance(SpiChannel.CS0,
8                 SpiDevice.DEFAULT_SPI_SPEED, // default spi speed 1 MHz
9                 SpiDevice.DEFAULT_SPI_MODE); // default spi mode 0
10
11             spi.write(SpiCommand.TEST.getValue(), (byte) 0x01);
12             System.out.println("Test mode all on");
13             Thread.sleep(1000);
14
15             spi.write(SpiCommand.TEST.getValue(), (byte) 0x00);
16             System.out.println("Test mode all off");
17             Thread.sleep(1000);
18
19             spi.write(SpiCommand.DECODE_MODE.getValue(), (byte) 0x00);
20             System.out.println("Use all bits");
21
22             spi.write(SpiCommand.BRIGHTNESS.getValue(), (byte) 0x08);
23             System.out.println("Changed brightness to medium level"
24                 + " (0x00 lowest, 0x0F highest)");
25
26             spi.write(SpiCommand.SCAN_LIMIT.getValue(), (byte) 0x0f);
27             System.out.println("Configured to scan all digits");
28
29             spi.write(SpiCommand.SHUTDOWN_MODE.getValue(), (byte) 0x01);
30             System.out.println("Woke up the MAX7219, is off on startup");
31
32             DemoMode.showRows(spi, 250);
33             DemoMode.showCols(spi, 250);
34             DemoMode.showRandomOutput(spi, 5, 500);
35
```

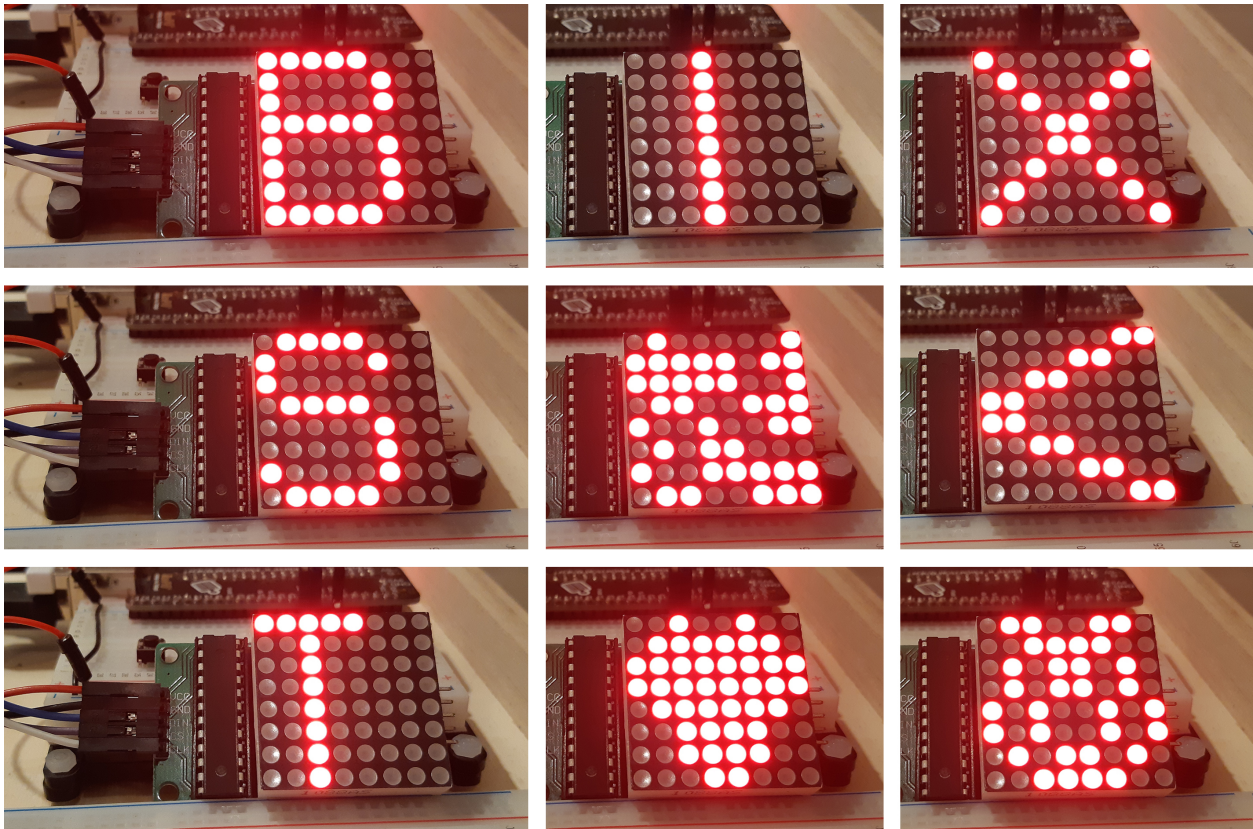


```
36         ImageMode.showAllImages(spi, 2000);
37         AsciiCharacterMode.showAllAsciiCharacters(spi, 750);
38         AsciiCharacterMode.scrollAllAsciiCharacters(spi, 50);
39     } catch (Exception ex) {
40         System.err.println("Error in main function: " + ex.getMessage());
41     }
42 }
43 }
```

## Running the application and created matrix output

As always, build the application, run it from the target-directory and check the result...

```
1 $ mvn clean package
2 $ java -jar target/pi4j-spi-1.0-SNAPSHOT-jar-with-dependencies.jar
3 Starting SPI example...
4 Disabled BCD mode
5 All LEDs ON
6 All LEDs OFF
7 Row 1 is on
8 ...
9 Col 7 is on
10 Random effect 1
11 ...
12 Showing image HEART
13 Showing image PI_LOGO
14 Showing image ARROW_LEFT
15 Showing image CROSS
16 Written to SPI : SPACE
17 Written to SPI : A
18 Written to SPI : B
19 ...
20 Scrolled : S
21 Scrolled : T
```



Some of the generated output on the 8x8 LED-matrix

## SPI conclusion

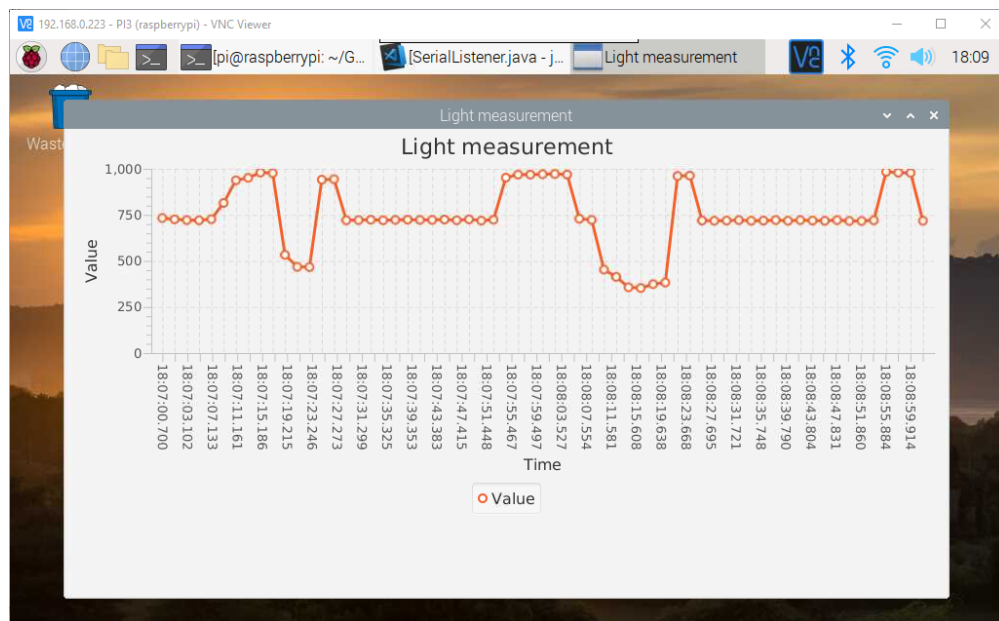
Homework left for you in this example! Not the complete ASCII-set is included and you can now only let one character scroll at a time, not a whole sentence... Hop hop, get to work now and experiment!

## Serial communication example with an Arduino

In this example, we will be using an Arduino board as a serial client for our Pi. With a light sensor, we'll do a measurement every few seconds and send the result as JSON data through serial communication via USB.

Two of the GPIO of the Pi can also be used for serial communication, but by default, they can't be used by an application as they are configured as console terminal port to be able to connect to the board without an ethernet connection. In the readme of this example project, some links are added in which the steps are described that you need to take to use the pins as the serial communication link, but this is not needed in our case as we use the USB connection.

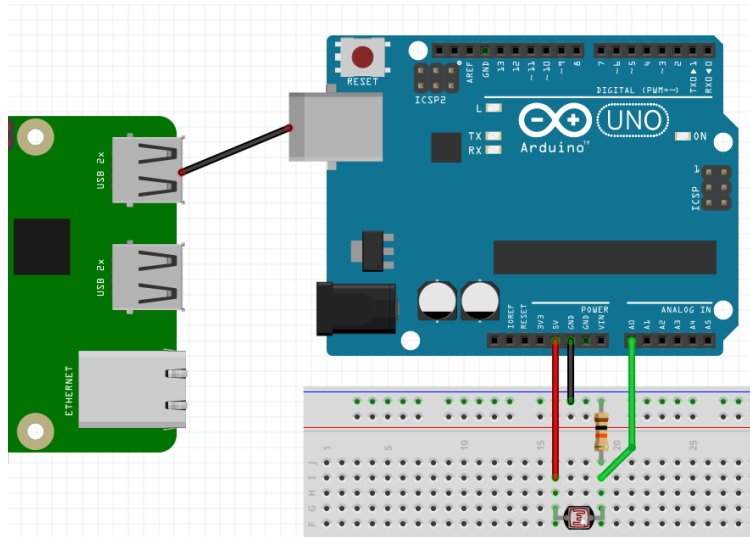
The finished application will show a line chart showing each measurement that is received from the Arduino board. The different values on the chart in the following screenshot were generated by switching between the bright light of my phone flashlight and covering the sensor.



Running serial example with chart showing the measurements

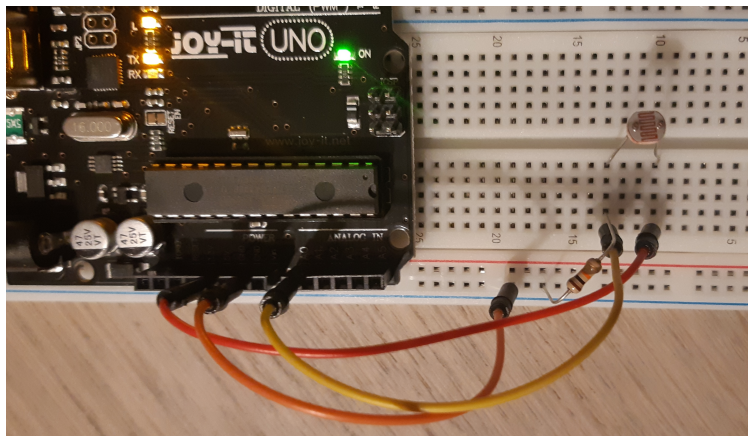
## Wiring

Connect the sensor to the 5V on one side, and the other side to the A0 analog pin of the Arduino board. With a 10k $\Omega$  resistor connected between the A0-connection and the ground, we add a pull-down system to have a good measurement.



Wiring scheme with the light sensor on Arduino

We will first connect the Arduino to the PC to program and test it. Later the same USB-cable is used to connect the Arduino board directly to the Pi.



Test setup

## Arduino code

For this example application, we use a simple Arduino script to:

- Initialize the serial connection to use 38400 Baud
- If serial data is received, return it as an echo message in JSON format
- On the predefined interval, send the sensor value also in JSON format

```
1  int loopCounter = 0;
2  int maxLoopCounter = 200;
3
4  int analogPinLightSensor = 0;
5
6  void setup() {
7    Serial.begin(38400);
8  }
9
10 void loop() {
11   if (Serial.available() > 0) {
12     String received = Serial.readStringUntil('\n');
13
14     Serial.print("{\"type\":\"echo\",\"value\":\"");
15     Serial.print(received);
16     Serial.println("\}");
17   }
18
19   loopCounter++;
20
21   if (loopCounter > maxLoopCounter) {
22     Serial.print("{\"type\":\"light\",\"value\":\"");
23     Serial.print(analogRead(analogPinLightSensor));
24     Serial.println("\}");
25
26     loopCounter = 0;
27   }
28
29   delay(10);
30 }
```

Please note the difference between “Serial.print” and “Serial.println”. The first one is used to construct parts of a sentence, while “println” ends the line by adding a line-break at the end. This is the identifier for the receiver to handle all the data it has received until that point.

When you upload this code to your Arduino board and open the “Serial Monitor”, you will immediately see the sensor value output. When you send a message, the echo also appears in the output window.

The screenshot shows the Arduino IDE interface. On the left, the code editor displays the following code:

```

1 int loopCounter = 0;
2 int maxCounter = 200;
3 int messageCounter = 0;
4
5 int analogPinLightSensor = 0;
6
7 void setup() {
8   Serial.begin(38400);
9 }
10
11 void loop() {
12   if (Serial.available() > 0) {
13     String received = Serial.readStringUntil('\n');
14
15     Serial.print("{\"type\":\"echo\",\"value\":\""});

```

On the right, the Serial Monitor window shows the following output:

```

talk to me [Send]
11:57:25.658 -> {"type":"light","value":645}
11:57:27.668 -> {"type":"light","value":645}
11:57:29.678 -> {"type":"light","value":645}
11:57:31.690 -> {"type":"light","value":645}
11:57:33.688 -> {"type":"light","value":644}
11:57:35.704 -> {"type":"light","value":635}
11:57:36.970 -> {"type":"echo","value":"test"}
11:57:37.720 -> {"type":"light","value":629}
11:57:39.727 -> {"type":"light","value":644}
11:57:41.740 -> {"type":"light","value":621}
11:57:41.979 -> {"type":"echo","value":"talk to me"}
11:57:43.753 -> {"type":"light","value":633}
11:57:45.766 -> {"type":"light","value":617}
11:57:47.786 -> {"type":"light","value":618}
11:57:49.799 -> {"type":"light","value":614}

```

Arduino code running and testing through “Serial Monitor”

## Detecting the serial interface on the Pi

We tested the serial communication in two directions with Arduino IDE on PC. Now you can unplug the USB connection from your PC and connect to the Raspberry Pi.

We will need to know the address of the Arduino to configure the serial communication in our Java application. Open the terminal on the Pi after you connected the Arduino and type in:

1 dmesg

This command will give you the latest logging of the Linux kernel which also contains the messages produced by the device drivers. We are looking for an address starting with ‘tty’:

```

[ 3515.388506] usb 1-1.1.2: USB disconnect, device number 5
[ 3518.756350] usb 1-1.1.2: new full-speed USB device number 8 using dwc_otg
[ 3518.900260] usb 1-1.1.2: New USB device found, idVendor=2341, idProduct=0043, bcdDevice= 0.01
[ 3518.900269] usb 1-1.1.2: New USB device strings: Mfr=1, Product=2, SerialNumber=220
[ 3518.900275] usb 1-1.1.2: Manufacturer: Arduino (www.arduino.cc)
[ 3518.900280] usb 1-1.1.2: SerialNumber: 7573630333635131B1A0
[ 3518.901043] cdc_acm 1-1.1.2:1.0: ttyACM0: USB ACM device
pi@raspberrypi:~$

```

Based on this output we know we will need to use “/dev/ttyACM0” further.

## Raspberry Pi code

Again for readability, maintainability, and testability, the code is split into different small classes.

Start from the javafx-minimal example and extend the pom-file with the pi4j-core and jackson-databind dependencies.

## Arduino message

As we saw in the Arduino code, every few seconds a new light measurement value is sent through the serial connection. The JSON format of this data is:

```
1 {"type": "light", "value": 394}
```

To be able to handle this easily in the application we need to convert it to a Java object and jackson-databind takes care of this for us with the `@JsonProperty`-annotations. Some additional get-methods are added as we could be using this type/value-pair for any type of data and values. For instance, the echo-type will contain a String, the light-type will contain an analog value between 0 and 1024.

```
1 public class ArduinoMessage {
2     @JsonProperty("type")
3     public String type;
4
5     @JsonProperty("value")
6     public String value;
7
8     public Integer getIntValue() {
9         if (this.value.matches("-?(0|[1-9]\\d*")) {
10             return Integer.parseInt(this.value);
11         }
12         return null;
13     }
14
15     public Float getFloatValue() {
16         if (this.value.matches("[+]?[0-9]*\\.?[0-9]+")) {
17             return Float.parseFloat(this.value);
18         }
19         return null;
20     }
21 }
```

## Arduino message mapper

In another class, we define the mapper to convert a JSON string to an `ArduinoMessage`-object. If you check the full code in the GitHub repo, you'll also find a unit test to validate this map-method.

```

1 public class ArduinoMessageMapper {
2     public static ArduinoMessage map(String jsonString) {
3         try {
4             ObjectMapper mapper = new ObjectMapper();
5             mapper.configure(
6                 DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES, false
7             );
8             return mapper.readValue(jsonString, ArduinoMessage.class);
9         } catch (IOException ex) {
10            System.err.println("Unable to parse string to Forecast: "
11                + ex.getMessage());
12            return null;
13        }
14    }
15 }

```

## Serial sender

This implements runnable as we will later start the class in a separate thread, to send a timestamp to the Arduino board to demonstrate the echo function.

```

1 public class SerialSender implements Runnable {
2     private static int INTERVAL_SEND_SECONDS = 5;
3
4     final Serial serial;
5
6     /**
7      * Constructor which gets the serial object to be used to send data.
8      *
9      * @param serial
10     */
11    public SerialSender(Serial serial) {
12        this.serial = serial;
13    }
14
15    @Override
16    public void run() {
17        // Keep looping until an error occurs
18        boolean keepRunning = true;
19        while (keepRunning) {
20            try {
21                // Write a text to the Arduino, as demo
22                this.serial.writeln("Timestamp: " + System.currentTimeMillis());

```



```

23
24         // Wait predefined time for next loop
25         Thread.sleep(INTERVAL_SEND_SECONDS * 1000);
26     } catch (Exception ex) {
27         System.err.println("Error: " + ex.getMessage());
28         keepRunning = false;
29     }
30 }
31 }
32 }

```

## Serial listener

Listener which will print out the data received on the serial connection

```

1  public class SerialListener implements SerialDataEventListener {
2      private final DateTimeFormatter formatter;
3      private final XYChart.Series<String, Integer> data;
4
5      /**
6       * Constructor which initializes the date formatter.
7       *
8       * @param data The data series to which the light values must be added
9       */
10     public SerialListener(XYChart.Series<String, Integer> data) {
11         this.data = data;
12         this.formatter = DateTimeFormatter.ofPattern("HH:mm:ss.SSS");
13     }
14
15     /**
16      * Called by Serial when new data is received.
17      */
18     @Override
19     public void dataReceived(SerialDataEvent event) {
20         try {
21             String received = event.getAsciiString()
22                 .replace("\t", "")
23                 .replace("\n", "");
24
25             ArduinoMessage arduinoMessage = ArduinoMessageMapper.map(received);
26             String timestamp = LocalTime.now().format(formatter);
27
28             if (arduinoMessage.type.equals("light")) {

```

```

29         // We need to use the runLater approach as this data is handled
30         // in another thread as the UI-component
31         Platform.runLater(() -> {
32             data.getData().add(
33                 new XYChart.Data(timestamp, arduinoMessage.getIntValue())
34             );
35         });
36     }
37
38     System.out.println(timestamp + " - Received: " + received);
39 } catch (IOException ex) {
40     System.err.println("Serial error: " + ex.getMessage());
41 }
42 }
43 }

```

## Measurement chart

Here we handle the generation of the user interface with the line chart, and initialize the serial connection as we want to link the serial listener to the chart data.

```

1 public class MeasurementChart extends VBox {
2     /**
3      * Constructor which will build the UI with the chart
4      * and start the serial communication
5      *
6      * @param serialDevice the serial device
7      */
8     public MeasurementChart(String serialDevice) {
9         // Initialize the data holder for the chart
10        XYChart.Series<String, Integer> data = new XYChart.Series<>();
11        data.setName("Value");
12
13        // Initialize the chart
14        CategoryAxis xAxis = new CategoryAxis();
15        xAxis.setLabel("Time");
16        NumberAxis yAxis = new NumberAxis();
17        yAxis.setLabel("Value");
18
19        LineChart lineChart = new LineChart(xAxis, yAxis);
20        lineChart.setTitle("Light measurement");
21
22        lineChart.getData().add(data);

```

```
23
24     this.getChildren().add(lineChart);
25
26     // Create an instance of the serial communications class
27     final Serial serial = SerialFactory.createInstance();
28
29     // Create and register the serial data listener
30     SerialListener serialListener = new SerialListener(data);
31     serial.addListener(serialListener);
32
33     this.startSerialCommunication(serial, serialDevice);
34
35     Thread t = new Thread(new SerialSender(serial));
36     t.start();
37 }
38
39 /**
40  * Start the serial communication
41  *
42  * @param serial Pi4J serial factory
43  * @param serialDevice the serial device
44  */
45 private void startSerialCommunication(Serial serial, String serialDevice) {
46     try {
47         // Create serial config object
48         SerialConfig config = new SerialConfig();
49         config.device(serialDevice)
50             .baud(Baud._38400)
51             .dataBits(DataBits._8)
52             .parity(Parity.NONE)
53             .stopBits(StopBits._1)
54             .flowControl(FlowControl.NONE);
55
56         // Display connection details
57         System.out.println("Connection: " + config.toString());
58
59         // Open the serial port with the configuration
60         serial.open(config);
61     } catch (Exception ex) {
62         System.err.println("Error: " + ex.getMessage());
63     }
64 }
65 }
```

## Application code

OK, we have all the elements now! Let's glue them together and start the application. The serial device is the address we found in the "dmesg" log.

```

1 public class App extends Application {
2     private static final String SERIAL_DEVICE = "/dev/ttyACM0";
3
4     @Override
5     public void start(Stage stage) {
6         System.out.println("Starting serial communication example");
7
8         var scene = new Scene(new MeasurementChart(SERIAL_DEVICE), 640, 480);
9         stage.setScene(scene);
10        stage.setTitle("Light measurement");
11        stage.show();
12    }
13
14    public static void main(String[] args) {
15        launch();
16    }
17 }

```

## Running the Java serial application

If everything went well, we can build and start the application. The logging output will show all messages received from the Arduino and the light-values will be used in the chart.

```

1 $ mvn clean package
2 $ java -jar target/pi4j-serial-1.0-SNAPSHOT-jar-with-dependencies.jar
3 Starting serial communication example
4 Connection: /dev/ttyACM0 (38400,8N1) {FC:NONE}
5 18:07:03.102 - Received: {"type":"light","value":722}
6 18:07:05.117 - Received: {"type":"light","value":721}
7 18:07:06.711 - Received: {"type":"echo","value":"Timestamp: 1580407626647"}
8 18:07:07.133 - Received: {"type":"light","value":727}
9 18:07:09.144 - Received: {"type":"light","value":815}
10 18:07:11.161 - Received: {"type":"light","value":938}
11 ...
12 18:08:16.735 - Received: {"type":"echo","value":"Timestamp: 1580407696668"}
13 18:08:17.626 - Received: {"type":"light","value":353}
14 18:08:19.638 - Received: {"type":"light","value":374}
15 18:08:21.649 - Received: {"type":"light","value":383}
16 ...

```

## What's next

The line chart can show different value sets. And the Arduino has a lot of analog and digital connections. Check which components you have available, connect them to the Arduino, extend the code and send their values as new types of data over the serial link.

By extending the `SerialListener` you can handle these different types and add them to additional data sets.

Have fun! :-)

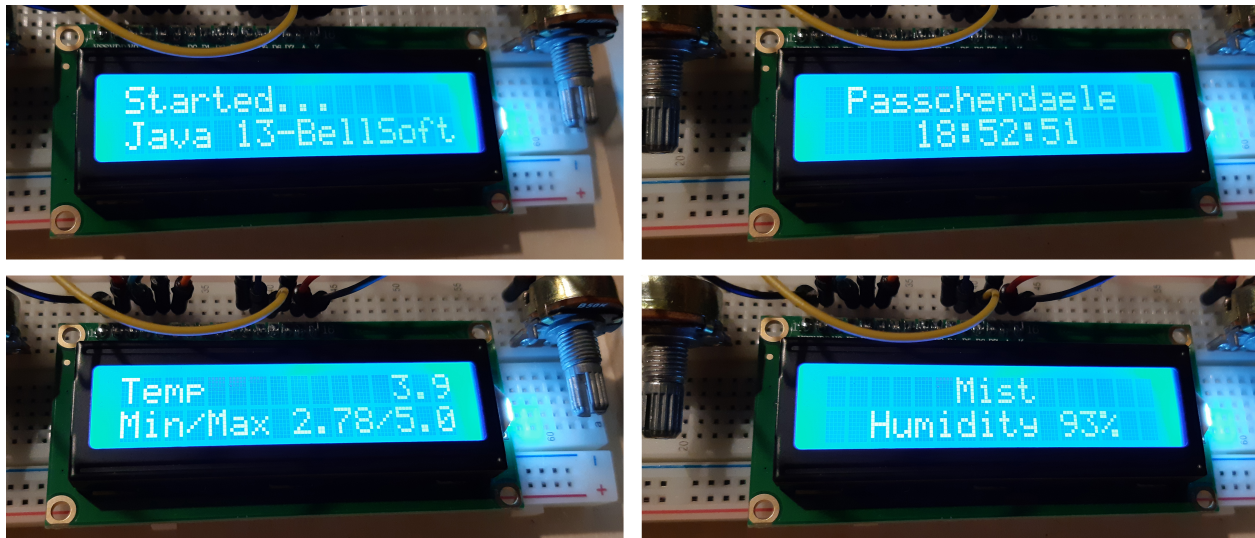
## LCD-display with the weather forecast

Pi4J also contains helper methods to minimize the work needed to use certain modules. As an example, we will be using “GpioLcdDisplay” to control an LCD with 2 rows of 16 characters (type “1602A”).



Very important remark before you power the test setup for this example! This module uses 5V so we don't want it to send back data to the Pi GPIOs. This must be done by connecting pin 5 (read/write) of the module to the ground.

We will be requesting the weather forecast from a public website to have dynamic data to visualize on the LCD-display.



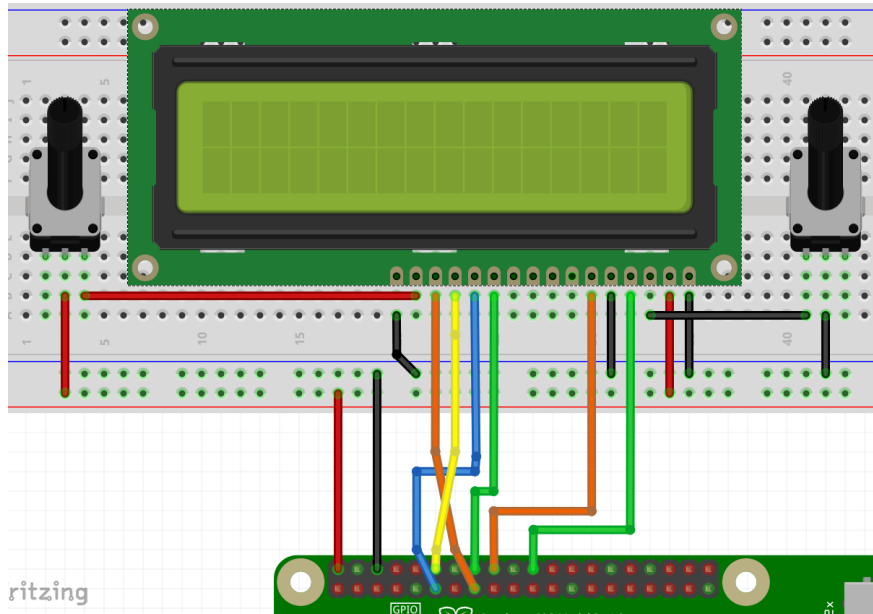
The four generated outputs on the LCD-screen

### Wiring to connect an LCD to the Pi

Not all these types of displays have completely the same connections, so you'll need to check your display to setup the wiring like this:

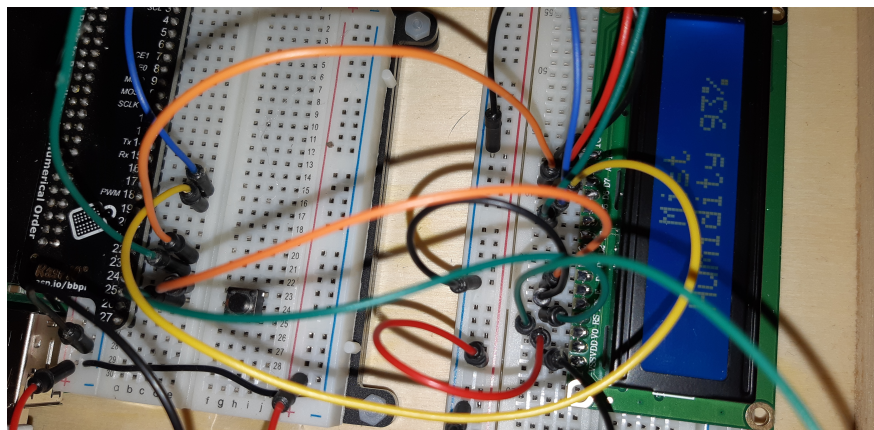
- VSS: ground
- VDD: 5V
- V0: with a variable resistor (potentiometer) connected to ground. Used to control the brightness of the characters. If you don't have such one, you can experiment with different resistors to find the best one for maximum readability.
- RS, E, D4, D5, D6 and D7: to GPIO as defined in the code

- RW: ground. Blocks sending back data to the board as we don't want to have 5V being sent to the board.
- K: ground
- A: with a variable resistor connected to 5V. Similar to V0 controls the background-LED brightness. This can also be replaced by a fixed resistor.



Connections between LCD and Pi

This is the final setup on my test board with the “Breadboard Pi Bridge”.



Connections between LCD and Pi

## Get an AppID on OpenWeatherMap

OpenWeatherMap API<sup>104</sup> is an online weather service where you can get a lot of data. If you check

<sup>104</sup><https://openweathermap.org/api>

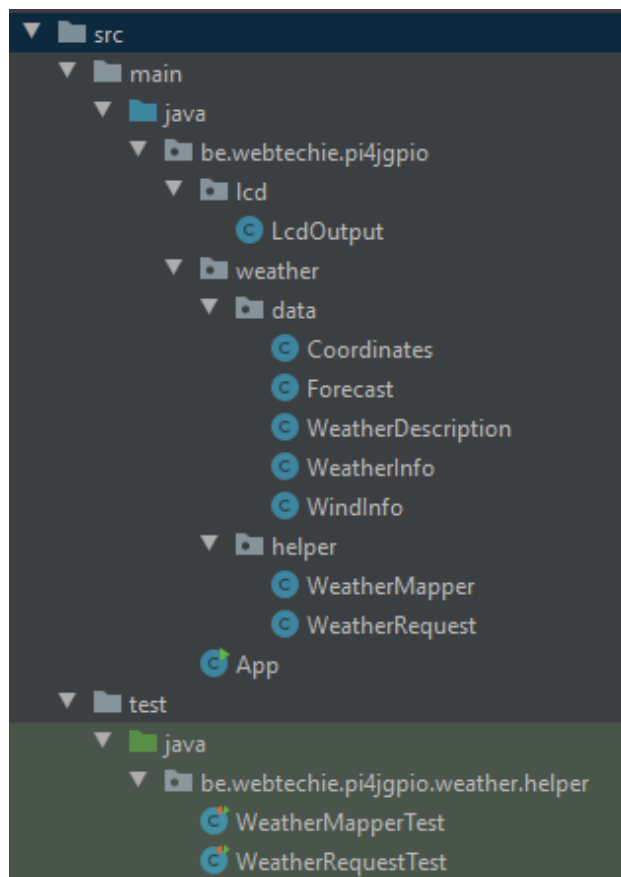
the “Price” page, you’ll see that only a small part can be used for free, but that’s still perfect for our example!

Start by signing up and you’ll receive an AppId in your mailbox, for example, “Your API key is 9f72246c2183b3e577fb925fafa0cfbf”. This is the value to be filled in in the code. Based on the key, the number of requests will be measured to check if you stay within the limits of the free account.

## Weather LCD application code

We start again from a minimal Java Maven project and add the dependencies for pi4j-core, pi4j-device and jackson-databind.

The final project is structured like this:



Code structure of the LCD-project

## Requesting the forecast from OpenWeatherMap

Let’s start with the forecast itself. We can build this as a separate class “WeatherRequest.java” which only handles this single task and returns the String received from the API. The location and appId are input parameters so we can easily test this in the next step.



```
1  /**
2   * Helper to get the forecast from OpenWeatherAPI.
3   */
4  public class WeatherRequest {
5      public static String getForecast(String location, String appId) {
6          StringBuilder rt = new StringBuilder();
7
8          try {
9              URL url = new URL("http://api.openweathermap.org/data/2.5/weather"
10                 + "?units=metric"
11                 + "&q=" + location
12                 + "&appid=" + appId);
13
14              HttpURLConnection conn = (HttpURLConnection) url.openConnection();
15              conn.setRequestMethod("GET");
16
17              int responseCode = conn.getResponseCode();
18              if (responseCode == HttpURLConnection.HTTP_OK) {
19                  BufferedReader in = new BufferedReader(
20                      new InputStreamReader(conn.getInputStream()));
21                  String readLine;
22                  while ((readLine = in.readLine()) != null) {
23                      rt.append(readLine);
24                  }
25                  in.close();
26              } else {
27                  System.err.println("Wrong response code: " + responseCode);
28              }
29          } catch (Exception ex) {
30              System.err.println("Request error: " + ex.getMessage());
31          }
32
33          return rt.toString();
34      }
35 }
```

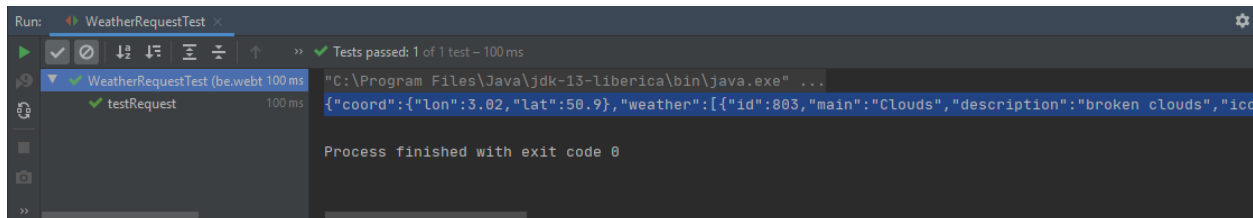
To make sure this code works as expected, we use a unit test in the test “test/java”-directory. To make this easy to maintain, the best practice is to use the same package-structure here as in “main/java”. The test class has the same name as the class which is tested + “Test”. In this case, we create “WeatherRequestTest.java” and check if we get a response from the weather service.

```

1 public class WeatherRequestTest {
2     // Create an app id by signing up on
3     // https://home.openweathermap.org/users/sign_up
4     private static final String APP_ID = "YOUR-KEY-HERE";
5
6     @Test
7     public void testRequest() {
8         String result = WeatherRequest.getForecast("Passendale", APP_ID);
9         assertNotNull(result);
10        System.out.println(result);
11    }
12 }

```

When we run this test, the result should be green and displaying the received forecast in the output.



Unit test result for WeatherRequestTest in IntelliJ IDEA

## Parsing the forecast to a Java object

The unit test from the previous step gives us a forecast in JSON format like this:

```

1 {
2     "coord": {
3         "lon": 3.02,
4         "lat": 50.9
5     },
6     "weather": [
7         {
8             "id": 803,
9             "main": "Clouds",
10            "description": "broken clouds",
11            "icon": "04d"
12        }
13    ],
14    "base": "stations",
15    "main": {
16        "temp": 6.16,

```

```
17     "feels_like":3.05,
18     "temp_min":3.89,
19     "temp_max":7.78,
20     "pressure":1032,
21     "humidity":87
22 },
23 "visibility":5000,
24 "wind":{
25     "speed":2.6,
26     "deg":70
27 },
28 "clouds":{
29     "all":75
30 },
31 "dt":1579779953,
32 "sys":{
33     "type":1,
34     "id":1233,
35     "country":"BE",
36     "sunrise":1579765021,
37     "sunset":1579796510
38 },
39 "timezone":3600,
40 "id":2794055,
41 "name":"Passchendale",
42 "cod":200
43 }
```

To be able to use this data in our application, we need to convert it to a Java object and the library “com.fasterxml.jackson.databind” will help us to achieve this.

First, we need to create the “Forecast.java” class to store every part of the data. As you can see every value in this class is linked to a specific value from the Json data, e.g. “coord” will be mapped to the coordinates variable.

```
1 public class Forecast {
2     @JsonProperty("coord")
3     public Coordinates coordinates;
4
5     @JsonProperty("weather")
6     public List<WeatherDescription> weatherDescription;
7
8     @JsonProperty("main")
9     public WeatherInfo weatherInfo;
10
11    @JsonProperty("wind")
12    public WindInfo windInfo;
13
14    @JsonProperty("name")
15    public String name;
16 }
```

Now let's add "WeatherDescription.java":

```
1 public class WeatherDescription {
2     @JsonProperty("id")
3     public long id;
4
5     @JsonProperty("main")
6     public String main;
7
8     @JsonProperty("description")
9     public String description;
10
11    @JsonProperty("icon")
12    public String icon;
13 }
```

And so on... As always you can find all the code as a finished project in the source code of this book on GitHub.

To fill these objects from a JSON string, we create the class "WeatherMapper.java" which takes a JSON String as input and returns a Forecast object:

```

1  /**
2   * Helper to convert the JSON received from OpenWeatherAPI to Java objects.
3   */
4  public class WeatherMapper {
5      public static Forecast getWeather(String jsonString) {
6          try {
7              ObjectMapper mapper = new ObjectMapper();
8              mapper.configure(
9                  DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES,
10                 false);
11             return mapper.readValue(jsonString, Forecast.class);
12         } catch (IOException ex) {
13             System.err.println("Unable to parse the given string to object: "
14                 + ex.getMessage());
15             return null;
16         }
17     }
18 }

```

At last, we add a unit test “WeatherMapperTest” to validate if an example JSON data string is converted as expected. In the init we need the full JSON string (truncated for readability here), and we create as many “@Test” functions as needed to check the parsing of all parts of the forecast.

```

1  public class WeatherMapperTest {
2      private Forecast forecast;
3
4      @Before
5      public void init() {
6          this.forecast = WeatherMapper.getWeather("{\"coord\":...\"");
7          assertNotNull(this.forecast);
8      }
9
10     @Test
11     public void testBase() {
12         assertEquals("Ypres", this.forecast.name);
13     }
14
15     @Test
16     public void testCoordinates() {
17         assertEquals(2.89, this.forecast.coordinates.longitude, 0.01);
18         assertEquals(50.85, this.forecast.coordinates.latitude, 0.01);
19     }
20 }

```

```

21     ...
22 }

```

## Put content on the LCD-screen

We will use a “Runnable” class to display the forecast output on the LCD-screen which will allow us to run this in a “Thread”. This has the advantage it will keep updating the screen uncoupled from the forecast request and we can implement functionality to continuously update the screen.

All this is integrated in “LcdOutput.java”. We start with some variables we will need later, a constructor with the “GpioLcdDisplay” as a parameter and a method to receive the “Forecast”. There is also a getter for the running state, so we can decide in the “main” class if the application needs to keep running.

```

1  /**
2   * Runnable class to continuously update the LCD-display with weather info.
3   */
4  public class LcdOutput implements Runnable {
5      private static final int LCD_ROW_1 = 0;
6      private static final int LCD_ROW_2 = 1;
7      private static final int CONTENT_INTERVAL = 2500;
8
9      private final GpioLcdDisplay lcd;
10
11     private boolean running = false;
12     private Forecast forecast = null;
13     private long lastUpdate = 0;
14     private int contentStep = 0;
15
16     private SimpleDateFormat formatter = new SimpleDateFormat("HH:mm:ss");
17
18     /**
19      * Constructor
20      *
21      * @param lcd Link to the GpioLcdDisplay
22      */
23     public LcdOutput(GpioLcdDisplay lcd) {
24         this.lcd = lcd;
25     }
26
27     /**
28      * Update the forecast used to show data on the LCD
29      *

```

```

30     * @param forecast The forecast
31     */
32     public void setForecast(Forecast forecast) {
33         this.forecast = forecast;
34     }
35
36     /**
37     * @return The running state.
38     */
39     public boolean isRunning() {
40         return this.running;
41     }
42 }

```

As this class implements “Runnable”, we need to add a run-method which will be called if this is started in a “Thread” which we will see later. This keeps running (except when an error occurs) and calls the “showContent” method on a predefined interval variable “CONTENT\_INTERVAL” which was already defined as 2500 milliseconds.

```

1  /**
2  * Method started by the tread which keeps looping until an error happens.
3  * On pre-defined interval, showContent is called to update the LCD output.
4  */
5  @Override
6  public void run() {
7      this.running = true;
8
9      try {
10         while (this.running) {
11             if (this.forecast != null
12                 && (System.currentTimeMillis() - this.lastUpdate)
13                     > CONTENT_INTERVAL) {
14                 this.showContent();
15                 this.lastUpdate = System.currentTimeMillis();
16             }
17
18             Thread.sleep(100);
19         }
20     } catch (Exception ex) {
21         System.err.println("Error in LCD output thread: " + ex.getMessage());
22
23         this.running = false;

```

```
24     }
25 }
```

Within the “showContent” method we handle the cycling of the content with the “contentStep” variable. If you want to add extra output, this is the place to be...

```
1  /**
2  * Periodically change the content on the LCD in different steps.
3  */
4  public void showContent() {
5      switch (this.contentStep) {
6          case 0:
7              this.showTimestamp();
8              break;
9          case 1:
10             this.showTemperatures();
11             break;
12          case 2:
13             this.showDescription();
14             break;
15          default:
16             System.err.println("Step not defined");
17     }
18
19     this.contentStep++;
20
21     if (this.contentStep > 2) {
22         this.contentStep = 0;
23     }
24 }
```

And of course, we now need to create the code for each of these methods which show a specific part of the forecast on the LCD-Screen. This is the code for the temperature output, you can find the other ones in the source code.



```

1  /**
2  * Show the forecast temperatures.
3  */
4  private void showTemperatures() {
5      System.out.println("Showing temperature " + forecast.weatherInfo.temperature);
6
7      try {
8          lcd.clear();
9          Thread.sleep(1000);
10
11         lcd.write(LCD_ROW_1, "Temp",
12                 LCDTextAlignment.ALIGN_LEFT);
13         lcd.write(LCD_ROW_1, String.valueOf(forecast.weatherInfo.temperature),
14                 LCDTextAlignment.ALIGN_RIGHT);
15         lcd.write(LCD_ROW_2, "Min/Max "
16                 + String.valueOf(forecast.weatherInfo.temperatureMinimum)
17                 + "/"
18                 + String.valueOf(forecast.weatherInfo.temperatureMaximum),
19                 LCDTextAlignment.ALIGN_CENTER);
20     } catch (Exception ex) {
21         System.err.println("Error while handling content for the LCD: "
22                 + ex.getMessage());
23     }
24 }

```

This is a long class, but by splitting up into multiple smaller methods, it's still very readable and easy to extend. As always, this is just one example, feel free to play with it and put on the screen whatever you want...

## Main class to get the forecast

Now we have all the building blocks to glue this together to one application in the "App.java" class and "main" method. This method will run in the first (main) thread.

We first initialize "GpioController" and "GpioLcdDisplay" from Pi4J. Then we can initialize our "LcdOutput" with the already created "GpioLcdDisplay" as a parameter and can start running it in a new (second) thread.

As a result, the App-class and the LcdOutput-class will run next to each other, so they can do their stuff without blocking the other one. And that's what we use by looping in the main-method to request a new forecast every number of seconds defined by REQUEST\_FORECAST\_SECONDS. But when an error happened in LcdOutput and it's not running anymore, we also stop the loop here and the application will exit.

```
1 public class App {
2     // Create an app id by signing up on
3     // https://home.openweathermap.org/users/sign_up
4     private static final String APP_ID = "YOUR-KEY-HERE";
5     private static final String LOCATION = "YOUR-LOCATION-HERE";
6     private static final int REQUEST_FORECAST_SECONDS = 60;
7
8     public static final int LCD_ROWS = 2;
9     public static final int LCD_COLUMNS = 16;
10
11     public static void main(String[] args) {
12         System.out.println("Starting LDC display example...");
13
14         try {
15             // Initialize the GPIO controller
16             final GpioController gpio = GpioFactory.getInstance();
17
18             // Initialize the LCD
19             final GpioLcdDisplay lcd = new GpioLcdDisplay(
20                 LCD_ROWS,           // Nr of rows
21                 LCD_COLUMNS,        // Nr of columns
22                 RaspiPin.GPIO_06,   // BCM 25: RS pin
23                 RaspiPin.GPIO_05,   // BCM 24: Strobe pin
24                 RaspiPin.GPIO_04,   // BCM 23: D4
25                 RaspiPin.GPIO_00,   // BCM 17: D5
26                 RaspiPin.GPIO_01,   // BCM 18: D6
27                 RaspiPin.GPIO_03    // BCM 22: D7
28             );
29
30             // Initial output to check if the wiring is OK
31             lcd.write(0, "Started...");
32             lcd.write(1, "Java " + SystemInfo.getJavaVersion());
33
34             // Initialize the LCD output and start it as a separate thread
35             final LcdOutput lcdOutput = new LcdOutput(lcd);
36             final Thread thread = new Thread(lcdOutput);
37             thread.start();
38
39             // Make sure the thread is started, before continuing
40             Thread.sleep(1000);
41
42             // Continuously get the weather forecast and show on the LCD
43             String apiReply;
```

```

44     while (lcdOutput.isRunning()) {
45         apiReply = WeatherRequest.getForecast(LOCATION, APP_ID);
46         if (!apiReply.isEmpty()) {
47             System.out.println("Received: " + apiReply);
48             lcdOutput.setForecast(WeatherMapper.getWeather(apiReply));
49         } else {
50             System.err.println("Could not get forecast");
51         }
52
53         Thread.sleep(REQUEST_FORECAST_SECONDS * 1000L);
54     }
55
56     // Shut down the GPIO controller
57     gpio.shutdown();
58
59     System.out.println("Done");
60 } catch (Exception ex) {
61     System.err.println("Error: " + ex.getMessage());
62 }
63 }
64 }

```

## Running the LCD weather application

Let's package this into a jar file and run it on the Pi.

```

1 $ mvn clean package
2 $ java -jar target/pi4j-lcddispla1.0-SNAPSHOT-jar-with-dependencies.jar
3 Starting LDC display example...
4 Received: {"coord":{"lon":3.02,"lat":50.9},"weather":[{"id":701,"main":"Mist","descr\
5 iption":"mist","icon":"50n"}],"base":"stations","main":{"temp":3.9,"feels_like":1.86\
6 ,"temp_min":2.78,"temp_max":5,"pressure":1037,"humidity":100},"visibility":1500,"win\
7 d":{"speed":1},"clouds":{"all":78},"dt":1579718566,"sys":{"type":1,"id":1233,"countr\
8 y":"BE","sunrise":1579678687,"sunset":1579710010},"timezone":3600,"id":2794055,"name\
9 ":"Passchendaele","cod":200}
10 Showing timestamp for Passchendaele
11 Showing temperature 3.9
12 Showing description mist
13 Received: ...

```

First “Started...” and “Java VERSION” will be shown on the LCD as defined in the main-method, but as soon as a forecast is received, the output generated by LcdOutput will be on the screen.



While testing this example on another day, an expected output on the temperature screen appeared with “4emp” where it should be displaying “Temp”.

Apparently, when the text is too long for one line, it overflows on the other line and the length of the string value needs to be checked before sending it to the LCD. We can use the “substring(beginIndex, endIndex)” method to achieve this, as displayed here with a test in “jshell”:

```
1  jshell> String test = "Min/Max 2.22/4.44";
2  test ==> "Min/Max 2.22/4.44"
3
4  jshell> test.length()
5  $2 ==> 17
6
7  jshell> test = test.substring(0, 16);
8  test ==> "Min/Max 2.22/4.4"
9
10 jshell> test.length()
11 $4 ==> 16
```

## Conclusion

In this example, you learned how easy it is to get data from a public source and show it on the LCD. Of course, you could also show a sensor measurement or a database value or ... Again, your imagination is the only limit!

# Just a thought: Switching social

I have to admit, yes, I'm socially addicted. Although I'm not the person who starts talking to everyone in real life. But on-line I'm networking fulltime with all means. I probably had an account on every service you can think of.

Twitter, Facebook, Google, LinkedIn and every new thing which seemed interesting for one or more reasons... Until a few years ago, when you were searching for test-users and I saw your tweet, you got my data within five minutes.

## Free is not Free

Of course, I realized a long time ago those services are free because they make a profit with your data. Your data is the oil that keeps their engine running. Google offers you all those user-friendly tools because they get to know you thanks to your gmails, documents in Google Drive, browsing and location history... And because of that, they can show you advertisements based on your interests so there is a bigger chance you click on them, buy and make the Google cash flow. That's also how Facebook knows your relation will end long before you change your status to "it's complicated" yourself. And LinkedIn knows you're searching for a new job, long before you request a final chat with your boss.

But what applies on-line, also happens off-line. Supermarkets know you're pregnant thanks to your card, based on your changed shopping habits, before you even tell your parents.

So we are constantly monitored and categorized, without seeing, and pushed in the most affordable direction for the advertisers.

## You are being sold

But it's not that bad, isn't it? That's also what I thought. Until I saw the talks of [Aral Balkan<sup>a</sup>](#). Years ago I saw him the first time at Multi-Mania in Kortrijk where he talked about user experience design and especially how dark-pattern-designs are used to trick users in choosing the most expensive option (low-cost flight operators seem to be most experienced in this matter...). Shortly after, he changed in an even more inspiring speaker, fighting for a new internet where the user becomes the owner of his own digital identity. He and his wife [Laura Kalbag<sup>b</sup>](#), who is an even important voice in the same battle, founded the [Small Technology foundation<sup>c</sup>](#) which advocates for, and builds small technology to protect personhood and democracy in the digital network age.

It's no longer only advertising we have to worry about. That whole system to get to know as much as possible about you has only one goal: sell your identity, a digital copy of you. If you don't pay for a service, you are the one being sold. You became the product being sold. You are being farmed!

The acronym [TANSTAAFL \(There ain't no such thing as a free lunch<sup>d</sup>](#) already exists since the 1930s, but it has taken a very bad turn in the last decade.

## Rehab

So I decided to rehab! I wanted to take back control of my online existence.

My Google e-mail account? All my mails are forwarded to my domain and everyone got my self-hosted address. I also stopped using Google Calendar and switched completely to my work calendar. Which makes it even easier to only have one place to plan both work and personal matters. Google search is replaced by [DuckDuckGo<sup>e</sup>](#) and [Ecosia<sup>f</sup>](#) because they don't track your history and even work better because they don't send you to the same sites over and over again because you clicked already a few times on them.

Many other services to go! There are still a few I need and use as long as I don't find an alternative, but that list gets shorter and shorter. Twitter, for instance, is my main source for information and news but [Mastodon<sup>g</sup>](#) is gaining a lot of traction these days!

### Switching Social

And don't forget you're not only tracked online. Adobe, Microsoft... all include trackers in the software you install on your PC. Luckily a site like [switching.software<sup>h</sup>](#) keeps a nice list of privacy-friendly alternatives. Take a look and check the nicely presented list per type of application.

### Grumpy old man

Did I become an old, grumpy, suspicious man? Maybe a bit. But years after Edward Snowden made public how the NSA and all those big internet companies work together to share our data, I found this beautiful quote of him: "Arguing that you don't care about privacy because you have nothing to hide is no different than saying you don't care about free speech because you have nothing to say."

---

<sup>a</sup><https://ar.al/>

<sup>b</sup><https://laurakalbag.com/posts/>

<sup>c</sup><https://small-tech.org/>

<sup>d</sup>[https://en.wikipedia.org/wiki/There\\_ain%27t\\_no\\_such\\_thing\\_as\\_a\\_free\\_lunch](https://en.wikipedia.org/wiki/There_ain%27t_no_such_thing_as_a_free_lunch)

<sup>e</sup><https://duckduckgo.com/>

<sup>f</sup><https://www.ecosia.org/>

<sup>g</sup><https://mastodon.social>

<sup>h</sup><https://switching.software/>

# Chapter 10: Spring

Spring is one of the leading frameworks for (enterprise) Java applications providing a set of features and programming conventions that allows developers to program faster and easier. Other such frameworks are [Micronaut<sup>105</sup>](#) and [Quarkus<sup>106</sup>](#).

Spring is developed by [Pivotal<sup>107</sup>](#), but available as an open-source framework. The goal of Spring is to **make it easier to develop applications by removing a lot of “boilerplate code”**. For example, reading and writing data in and from a database can be done with minimal code. The core idea of Spring is to use **dependency injection** which allows you to write **decoupled classes that are tight together at runtime when needed**.

Dependency injection is a system to provide **Inversion Of Control (IOC)<sup>108</sup>** which is used to manage the relationship between objects. As a developer, you don't need to take care of this relationship, as the framework will handle that for you.

By using Spring-annotations (e.g. “@Component”) simple POJO classes ([Plain Old Java Object<sup>109</sup>](#)) get injected into other classes so these can use their methods.

Let's look at an example where we have a class which can load sensors from a database:

```
1 @Component
2 public class SensorDao {
3     // We use the generic abbreviation for this class: DAO = Data Access Object
4     public List<Sensor> getSensors() {
5         // Database code needs to be added here
6     }
7 }
```

Now any other class in the application can use this Component by auto-wiring to it, without the need to initialize it or worry about the correct order of initialization.

---

<sup>105</sup><https://micronaut.io/>

<sup>106</sup><https://quarkus.io/>

<sup>107</sup><https://pivotal.io/>

<sup>108</sup>[https://en.wikipedia.org/wiki/Inversion\\_of\\_control](https://en.wikipedia.org/wiki/Inversion_of_control)

<sup>109</sup>[https://en.wikipedia.org/wiki/Plain\\_old\\_Java\\_object](https://en.wikipedia.org/wiki/Plain_old_Java_object)

```
1  @Component
2  public class HtmlHelper {
3      @Autowired
4      private SensorDao sensorDao;
5
6      public String getSensorsAsHtmlList() {
7          StringBuilder rt = new StringBuilder();
8          rt.append("<ul>");
9          for (Sensor sensor : this.sensorDao.getSensors()) {
10             rt.append("<li>").append(sensor.getName()).append("</li>");
11          }
12          rt.append("</ul>");
13          return rt.toString();
14      }
15 }
```

At startup, the framework will make sure the SensorDao is available for HtmlHelper when it needs it.



## What is Spring Boot?

Spring Boot is a “layer” on top of Spring which provides “off-the-shelf” packages to create stand-alone Spring-based applications that you can “just run”. This is achieved by a principle known as “**convention above configuration**”. You only need to configure the things you want to behave differently than the default way.

Most important features as listed on [the Spring Boot website](https://spring.io/projects/spring-boot)<sup>110</sup>:

- Easily create stand-alone Spring applications
- Embed a webserver inside your application (Tomcat, Jetty or Undertow)
- Provide opinionated ‘starter’ dependencies to simplify your build configuration
- Automatically configure Spring and 3rd party libraries whenever possible
- Provide production-ready features such as metrics, health checks, and externalized configuration
- Absolutely no code generation and no requirement for XML configuration

The Spring examples in this book are all based on Spring Boot.

---

<sup>110</sup><https://spring.io/projects/spring-boot>

## What is Spring Initializr?

[Spring Initializr](https://start.spring.io/)<sup>111</sup> is an online tool to quickly get an application that includes all the Spring Boot packages you need.

In the screenshot below the options are selected for the project we are going to use as a starting point. The additional “Spring Web” dependency is added to allow us to very easily build a [REST-application](#)<sup>112</sup> which allows us to connect to the application via webpages.

The screenshot shows the Spring Initializr web interface. On the left, there is a sidebar with the following sections: Project, Language, Spring Boot, and Project Metadata. The main content area is divided into two columns. The left column contains the following options: Project (Maven Project selected), Language (Java selected), Spring Boot (2.2.2 selected), and Project Metadata (Group: be.webtechie, Artifact: java-spring-rest). The right column contains the following options: Name (java-spring-rest), Description (Demo project for Spring Boot), Package Name (be.webtechie.java-spring-rest), Packaging (Jar selected), and Java (11 selected). At the bottom, there are three buttons: Generate - Ctrl + G, Explore - Ctrl + Space, and Share...

### Spring Initializr

Click on the “Generate” button and you will get a ZIP-file with a ready-to-use Maven project.



The unmodified downloaded sources from [start.spring.io](https://start.spring.io/) are included in the sources of this book:

Chapter\_10\_Spring > java-spring-rest-original

This project can be opened in Visual Studio Code and started by hitting the Run command in `JavaSpringRestApplication.java`

<sup>111</sup><https://start.spring.io/>

<sup>112</sup>[https://en.wikipedia.org/wiki/Representational\\_state\\_transfer](https://en.wikipedia.org/wiki/Representational_state_transfer)

```

src > main > java > be > webtechie > javaspringrest > JavaSpringRestApplication.java > JavaSpringRestApplication
1  package be.webtechie.jvaspringrest;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6  @SpringBootApplication
7  public class JavaSpringRestApplication {
8
9  public static void main(String[] args) {
10     SpringApplication.run(JavaSpringRestApplication.class, args);
11 }
12
13 }
14

```

Starting the Spring Initializr generated application

In the logging we will get this result (without the timestamps here for readability):

```

1  .   ____          _            __ _ _
2  /\\ / ___'_  _ ( _)_      _   /\\ / \\
3  ( ( )\___ | '_ | '_ | | '_ \ / \\ / \\
4  \\/ ___| |_) | | | | | |_) | ( ) / |
5  '  |___| |__| | | | | |__| | / / / /
6  =====|_|=====|__|=/_/_/_/
7  :: Spring Boot ::      (v2.2.2.RELEASE)
8
9  Starting JavaSpringRestApplication with PID 1260
10     (C:\...\java-spring-rest-original\target\classes
11     started by Frank in C:\...\java-spring-rest-original)
12  No active profile set, falling back to default profiles: default
13  Tomcat initialized with port(s): 8080 (http)
14  Starting service [Tomcat]
15  Starting Servlet engine: [Apache Tomcat/9.0.29]
16  Initializing Spring embedded WebApplicationContext
17  Root WebApplicationContext: initialization completed in 1607 ms
18  Initializing ExecutorService 'applicationTaskExecutor'
19  Tomcat started on port(s): 8080 (http) with context path ''
20  Started JavaSpringRestApplication in 4.582 seconds (JVM running for 7.196)
21  Initializing Spring DispatcherServlet 'dispatcherServlet'
22  Initializing Servlet 'dispatcherServlet'
23  Completed initialization in 9 ms

```

What does this tell us?

- JVM is running for 7 seconds of which 5 seconds was needed to start the application.
- Tomcat (the webserver) is embedded in the application and running on port 8080.

Nice, let's open a browser and go to <http://localhost:8080/>

## Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

There was an unexpected error (type=Not Found, status=404).  
No message available

Spring Initializr HTML error page

An error page? Indeed, we didn't create anything yet to be on this page! But as we get this error page, we know our application is working and acting as a webserver. It just doesn't know yet what to give us back.

Let's extend this "empty box" with some nice features we can use on the Pi in the next examples.

## Interview with Mark Heckler



**Mark Heckler, [@mkheck](#)<sup>a</sup>, Spring Developer & Advocate at Pivotal Software (VMware), Java Champion**

*Why has Pivotal (VMware) chosen to provide Spring framework as free and open software, while you are a company and need to make money out of this?*

Spring's origins are free and open. When Rod Johnson published his groundbreaking book *Expert One-on-One J2EE Design and Development* in 2002, he included source code showing how to better solve common problems and pain points in enterprise Java. This code served as the foundation for Interface21, which eventually became known as the Spring Framework. It's in our DNA!

VMware has many ways to make money, and not all offerings contribute directly to that goal... but there are many goals within a company and its portfolio. If you provide the best tools for developers that can truly solve their problems, even if many of those offerings come with no cost whatsoever, you're in a far better position to partner with those devs on problems where it makes sense for them to invest. By putting the community first, everyone wins.

*Now Pivotal has been acquired by VMWare, will this cause changes in the way Spring is developed and available?*

Nope! Even the [spring.io](#)<sup>b</sup> (including [start.spring.io](#)<sup>c</sup>) URLs are the same. Same team, same approach, same unmatched developer productivity. :-D

*As a Spring developer, do you believe Java and Spring are a good match with the Raspberry Pi?*

First let me say that none of the Spring portfolio of projects has ever specifically targeted the Raspberry Pi, to my knowledge. :-) That said, as an IoT enthusiast & advocate myself, I've been happily deploying Spring code to the Pi for a few years now. Like all things Spring, it just works. And with the Pi becoming ever more capable, there is no shortage of things you can do with Spring on the Pi... as your book demonstrates!

*Will the Spring platform be further extended for "small" computers like the Pi?*

There is nothing specifically in the works targeting System on a Chip (SoC) computers like the Pi within Spring, but there are some upcoming developments that will apply very nicely. Stay tuned! I think 2020 is going to be a very good year. :-)

*Why is now the perfect time to learn Java?*

Considering Java's origins, it was made with small devices in mind! Of course, it could handle massive problems and workloads, which is why it was naturally embraced in tackling enterprise-grade problems pretty early on. But Java is again becoming smaller, faster, more streamlined, and targeted toward efficiency. Between that and the faster release cadence, we're seeing some great innovations that make now perhaps the most exciting time for Java since its initial splash.

It's also an incredibly versatile language with a massive number of libraries available - many of which are free and open-source - to solve established problems & brand new ones, making a developer's job much easier and more enjoyable. It's like having a never-ending toolbox of options at hand, for a language that's constantly adding capabilities. What could be better than that???

*Which DIY-programming-electronics-project are you working on, or is on your "if I ever have time" list?*

I'm building a real-world distributed system with live aviation data streams using the Raspberry Pi and other small devices. I plan to showcase that in my upcoming book, "Spring Boot: Up & Running!", due later this year. As you might expect, Spring Boot & other Spring portfolio projects feature heavily in that. I'm really excited about it!

---

<sup>1</sup><https://twitter.com/mkcheck>

<sup>2</sup><https://spring.io/>

<sup>3</sup><https://start.spring.io/>

## Example 1: Minimal webserver on the Pi

Imagine we use the Pi as a storage device with pictures. How cool is it, to expose those images via a website, so we can look at the pictures from any computer in our house? Let's try...



Full sources can be found in:  
Chapter\_10\_Spring > java-spring-image-server

### Start from the Initializr project and modify pom.xml

We already have a running webserver, so let's make a copy of the Spring Initializr project and call it "java-spring-image-server".

First, we modify pom.xml to match our new project, in this case, we need to change "artifactId", "name" and "description":

```

1     <groupId>be.webtechie</groupId>
2     <artifactId>java-spring-image-server</artifactId>
3     <version>0.0.1-SNAPSHOT</version>
4     <name>java-spring-image-server</name>
5     <description>Spring Boot project to access pictures via the browser</description>

```

We also add some dependencies which will help us to test our application with [Swagger<sup>113</sup>](https://swagger.io/). This framework will automatically provide documentation webpages for all our web services.

```

1 <dependencies>
2   <dependency>
3     <groupId>org.springframework.boot</groupId>
4     <artifactId>spring-boot-starter-web</artifactId>
5   </dependency>
6
7   <dependency>
8     <groupId>io.springfox</groupId>
9     <artifactId>springfox-swagger2</artifactId>
10    <version>2.9.2</version>
11  </dependency>
12  <dependency>
13    <groupId>io.springfox</groupId>
14    <artifactId>springfox-swagger-ui</artifactId>

```

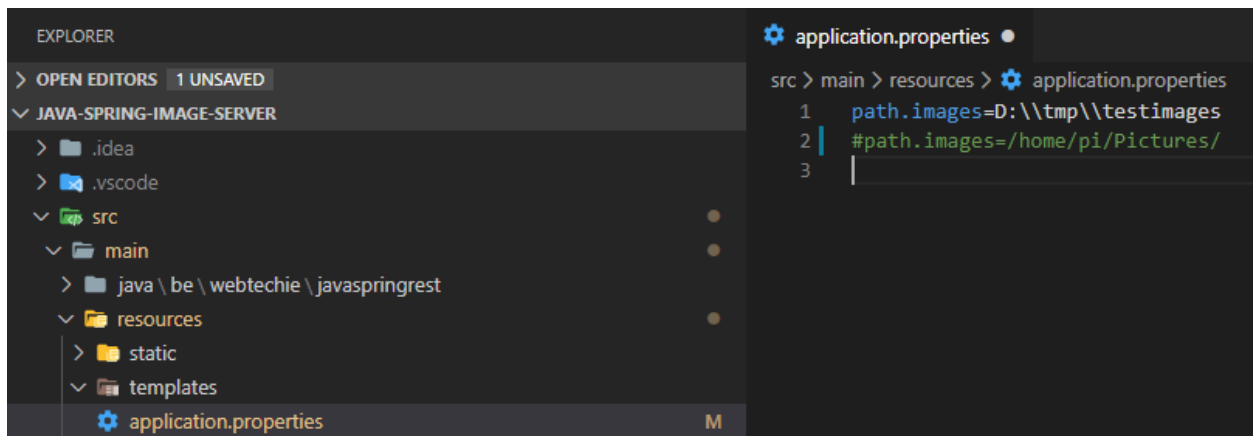
<sup>113</sup><https://swagger.io/>

```
15     <version>2.9.2</version>
16 </dependency>
17
18 <dependency>
19     <groupId>org.springframework.boot</groupId>
20     <artifactId>spring-boot-starter-test</artifactId>
21     <scope>test</scope>
22 </dependency>
23 </dependencies>
```

## Application properties

We will use a Spring property to make it easy to change the directory where the images are stored. Open the file “application.properties” from the resources directory. Create a directory on your PC with some test images and add the path to this file.

On Windows, directories are separated with a backslash, but we need to use a double one for correct parsing of this value, as you can see in the screenshot.



application.properties file

The second line in this screenshot is a comment (starts with “#”) and is the value we are going to use for this property when we’ve finished testing and want to build this application for the Pi. At that moment we move the “#” to the first line with the Windows setting.

## Image controller

And now we are going to add the image controller which needs to do two things: show a list of all the available images and provide a selected image. Again we add a new package “controller” and add a file “ImageController.java”. This is the initial code for this file which will generate a table with the files in the directory which is configured in the application.properties file with the key “path.images”.



Note a `StringBuilder` is used to construct the HTML table. This will work a lot faster compared to the use of a `String`. The extra whitespace and tabs in the different `“append”` lines are only there for better readability.

```

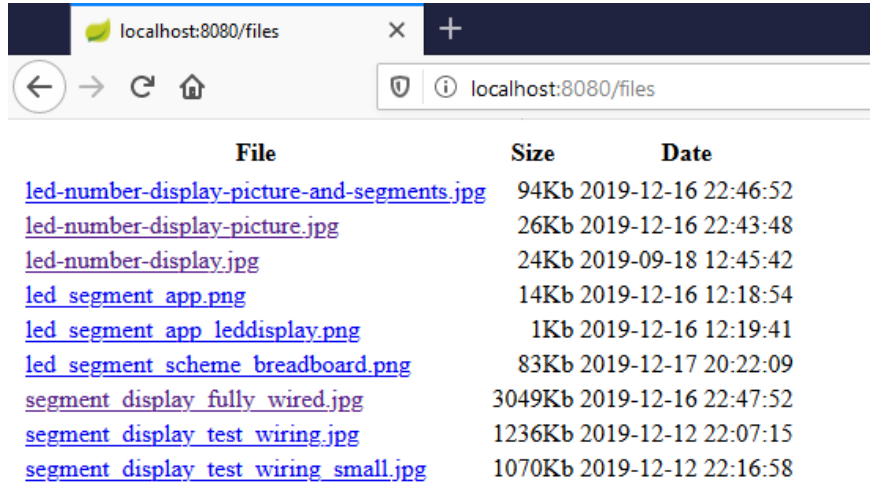
1  /**
2   * REST-calls to access the images on this device.
3   */
4  @RestController
5  public class ImageController {
6      /**
7       * Get the value from application.properties where we define
8       * the location of the images.
9       */
10     @Value("${path.images}")
11     private String pathImages;
12
13     /**
14      * Get a list of all the files.
15      * @return An HTML string with a list of all the files.
16      */
17     @GetMapping("/files")
18     public String getFiles() {
19         StringBuilder rt = new StringBuilder();
20
21         try {
22             rt.append("<html>")
23                 .append("    <body>")
24                 .append("        <table>")
25                 .append("            <tr>")
26                 .append("                <th>File</th>")
27                 .append("                <th>Size</th>")
28                 .append("                <th>Date</th>")
29                 .append("            <tr>");
30
31             File[] childFiles = (new File(this.pathImages)).listFiles();
32             for (File childFile : childFiles) {
33                 String relativePath = childFile.getName();
34                 rt.append("            <tr>")
35                     .append("                <td>")
36                     .append("<a href='file/'")
37                     .append(URLEncoder.encode(relativePath, "UTF-8"))
38                     .append("> target='_blank'>")
39                     .append(relativePath)

```

```
40         .append("</a></td>")
41     .append("          <td style='text-align: right;'>")
42     .append(getSize(childFile)).append("</td>")
43     .append("          <td>")
44     .append(getTimestamp(childFile)).append("</td>")
45     .append("          </tr>");
46     }
47
48     rt.append("    <table>");
49     rt.append(" </body>");
50     rt.append("</html>");
51
52     } catch (Exception ex) {
53         throw new RuntimeException("Error accessing requested file/directory: "
54             + ex.getMessage());
55     }
56
57     return rt.toString();
58 }
59
60 /**
61  * Converts the file size in bytes to Kb.
62  *
63  * @param file
64  * @return
65  */
66 private String getSize(File file) {
67     long sizeInBytes = file.length();
68     long sizeInKilobytes = sizeInBytes / 1024;
69     return sizeInKilobytes + "Kb";
70 }
71
72 /**
73  * Converts the last modified timestamp of the file to a readable format.
74  *
75  * @param file
76  * @return
77  */
78 private String getTimestamp(File file) {
79     long timestamp = file.lastModified();
80     SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
81     return dateFormat.format(new Date(timestamp));
82 }
```

83 }

This will provide us a webpage with the file list:



| File                                                        | Size   | Date                |
|-------------------------------------------------------------|--------|---------------------|
| <a href="#">led-number-display-picture-and-segments.jpg</a> | 94Kb   | 2019-12-16 22:46:52 |
| <a href="#">led-number-display-picture.jpg</a>              | 26Kb   | 2019-12-16 22:43:48 |
| <a href="#">led-number-display.jpg</a>                      | 24Kb   | 2019-09-18 12:45:42 |
| <a href="#">led segment app.png</a>                         | 14Kb   | 2019-12-16 12:18:54 |
| <a href="#">led segment app leddisplay.png</a>              | 1Kb    | 2019-12-16 12:19:41 |
| <a href="#">led segment scheme breadboard.png</a>           | 83Kb   | 2019-12-17 20:22:09 |
| <a href="#">segment display fully wired.jpg</a>             | 3049Kb | 2019-12-16 22:47:52 |
| <a href="#">segment display test wiring.jpg</a>             | 1236Kb | 2019-12-12 22:07:15 |
| <a href="#">segment display test wiring small.jpg</a>       | 1070Kb | 2019-12-12 22:16:58 |

The file list in the browser

Now let's add a second method to get the file itself. Some more code is needed here to make sure the file exists and we return the correct media type depending on the type of image.

```

1  /**
2  * Get the request file.
3  * @param fileName The filename
4  * @return The file as byte array
5  */
6  @GetMapping(
7      value = "/file/{filename}",
8      produces = MediaType.APPLICATION_OCTET_STREAM_VALUE)
9  public ResponseEntity<byte[]> getFile(@PathVariable("filename") String fileName) {
10     // Initiate the headers we will use in the return
11     HttpHeaders headers = new HttpHeaders();
12     headers.setCacheControl(CacheControl.noCache().getHeaderValue());
13
14     // Get the file
15     File file = new File(this.pathImages, fileName);
16
17     // Check if the file exists.
18     if (!file.exists()) {
19         // Immediately return error 404.
20         return new ResponseEntity(HttpStatus.NOT_FOUND);
21     }

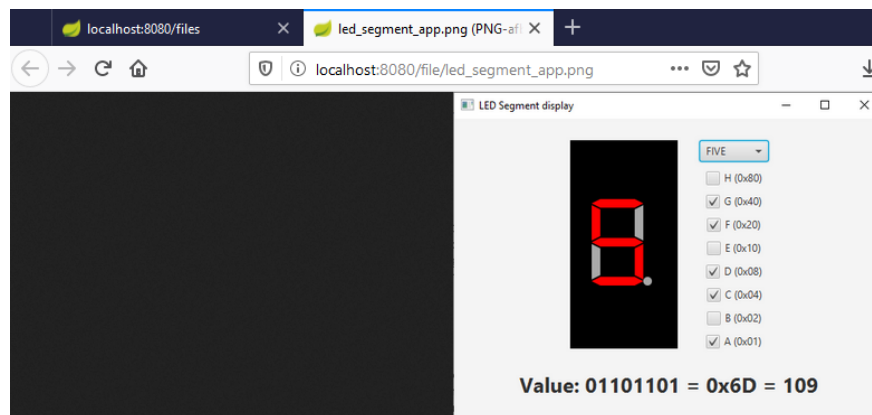
```

```

22
23 // Get the file as a byte array.
24 byte[] media = null;
25 try (InputStream in = new FileInputStream(file)) {
26     media = in.readAllBytes();
27 } catch (IOException ex) {
28     // Oops something went wrong, return error 500.
29     headers.setContentType(MediaType.TEXT_PLAIN);
30     return new ResponseEntity(ex.getMessage(), headers,
31         HttpStatus.INTERNAL_SERVER_ERROR);
32 }
33
34 // Check which type of file we are returning so we can correctly define
35 // the header content type. By doing this, the browser can show
36 // the image inside the browser, otherwise it will do a download.
37 if (fileName.toLowerCase().endsWith(".jpg")) {
38     headers.setContentType(MediaType.IMAGE_JPEG);
39 } else if (fileName.toLowerCase().endsWith(".png")) {
40     headers.setContentType(MediaType.IMAGE_PNG);
41 } else if (fileName.toLowerCase().endsWith(".gif")) {
42     headers.setContentType(MediaType.IMAGE_GIF);
43 }
44
45 // Everything OK, return the image.
46 return new ResponseEntity<>(media, headers, HttpStatus.OK);
47 }

```

This method will give us the image in the browser:



The file itself in the browser



If you want to extend this web-interface with more pages with a nicer look-and-feel, you should take a look at [Thymeleaf](https://www.thymeleaf.org/)<sup>114</sup>. This server-side Java template engine, allows you to separate HTML files from code. This results in cleaner code and easier to maintain web pages.

## Swagger config

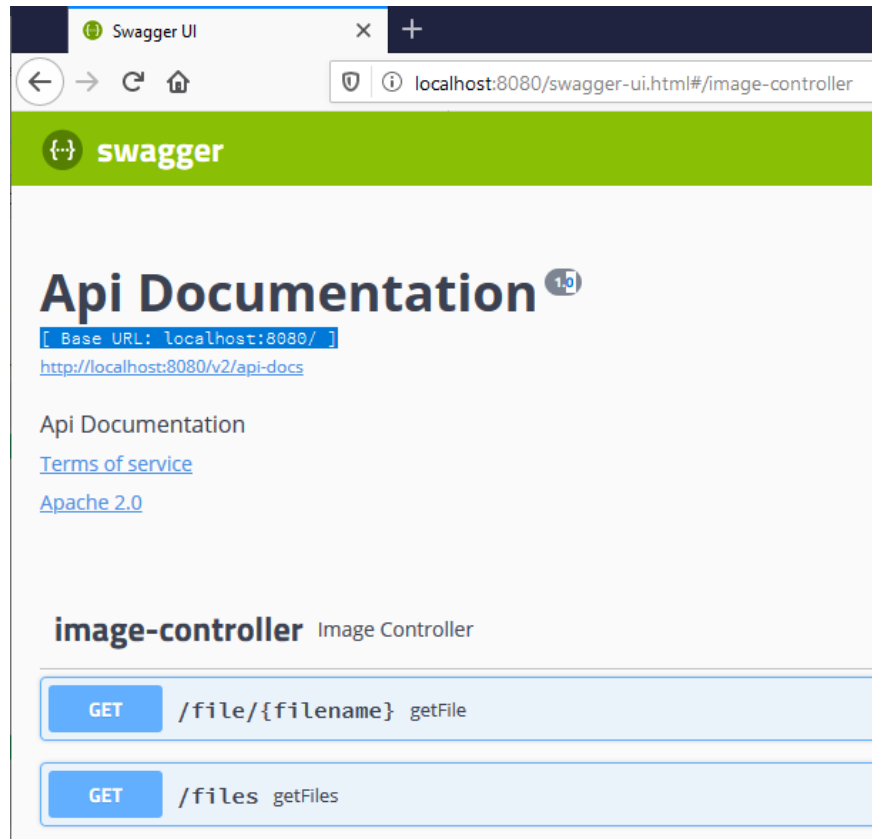
To configure Swagger, we need to add a class. For a clear structure, it is added in a new package “configuration” with the file “SwaggerConfig.java” which looks like this to automatically document all our web services:

```
1 @Configuration
2 @EnableSwagger2
3 public class SwaggerConfig {
4     @Bean
5     public Docket api() {
6         return new Docket(DocumentationType.SWAGGER_2)
7             .select()
8             .apis(RequestHandlerSelectors.any())
9             .paths(PathSelectors.any())
10            .build();
11    }
12 }
```

This simple config will automatically create a webpage with all the info of the web services we created in our application on “localhost:8080/swagger-ui.html”.

---

<sup>114</sup><https://www.thymeleaf.org/>



Swagger UI

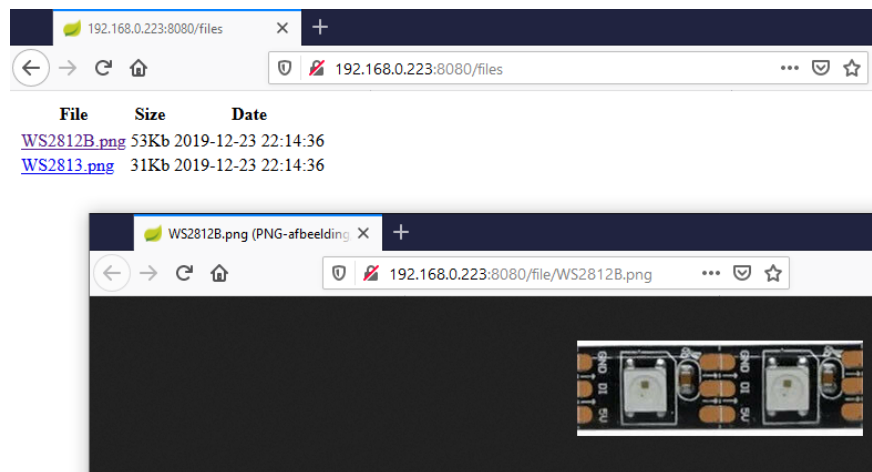
## Run on the Pi

Change the application.properties file to define a directory on your Pi from which the images must be loaded, for example:

```
application.properties ×
src > main > resources > application.properties
1 #path.images=D:\\tmp\\testimages
2 path.images=/home/pi/Pictures/
3
```

application.properties for Pi

Now build the application with “mvn clean package” and copy the file “java-spring-image-server-0.0.1-SNAPSHOT.jar” from the target directory to your Pi. Start the application with “java -jar java-spring-image-server-0.0.1-SNAPSHOT.jar” and browse with your PC to the IP of your Pi, e.g. in my case on “http://192.168.0.223:8080/files”:



Images served from a Pi

## Conclusion

And there it is, our image server! Start with an empty Spring project, add one class (Swagger is not really needed, it's only here as an example) and done!!!

## Example 2: Database REST-service for IoT data on Pi

This project is a proof-of-concept application to store data in a database on the Pi. The data can be provided by any source with REST-services.



Full sources can be found in:  
Chapter\_10\_Spring > java-spring-rest-db

The [H2 database](#)<sup>115</sup> we will be using is a full-Java-solution which doesn't need any additional installation but is fully part of the application itself.

There is a lot of “discussion” whether you should use H2 only for testing or can use it in production. In recent years it has evolved into a very stable database solution which in my opinion is a very good choice for embedded projects.

JPA is the persistence specification used to “communicate” with the database. It helps you to define which objects (entities) can be stored in a database in which structure (tables).

### pom.xml settings

Again starting from a minimal Spring project, we modify the pom.xml file with the correct settings and some more dependencies.

```
1 <groupId>be.webtechie</groupId>
2 <artifactId>java-spring-rest-db</artifactId>
3 <version>0.0.1-SNAPSHOT</version>
4 <name>java-spring-rest-db</name>
5 <description>Spring Boot project to store data in a H2 database</description>
6
7 <dependencies>
8     <dependency>
9         <groupId>org.springframework.boot</groupId>
10        <artifactId>spring-boot-starter-web</artifactId>
11    </dependency>
12    <dependency>
13        <groupId>org.springframework.boot</groupId>
14        <artifactId>spring-boot-starter-data-jpa</artifactId>
15    </dependency>
16    <dependency>
17        <groupId>org.springframework.boot</groupId>
18        <artifactId>spring-boot-starter-data-rest</artifactId>
```

---

<sup>115</sup><https://www.h2database.com>



```
19     </dependency>
20     <dependency>
21         <groupId>com.h2database</groupId>
22         <artifactId>h2</artifactId>
23         <scope>runtime</scope>
24     </dependency>
25
26     <dependency>
27         <groupId>io.springfox</groupId>
28         <artifactId>springfox-swagger2</artifactId>
29         <version>2.9.2</version>
30     </dependency>
31     <dependency>
32         <groupId>io.springfox</groupId>
33         <artifactId>springfox-swagger-ui</artifactId>
34         <version>2.9.2</version>
35     </dependency>
36
37     <dependency>
38         <groupId>org.springframework.boot</groupId>
39         <artifactId>spring-boot-starter-test</artifactId>
40         <scope>test</scope>
41     </dependency>
42 </dependencies>
```

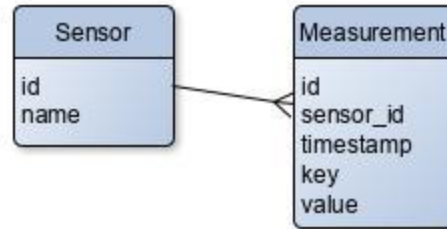
## Creating the database entities

The main advantage of a JPA is the fact we can define our database structure in code. We don't need to create and define tables; the code will take care of this.

This is the model we want to use in our example:

- Sensor
- Unlimited number of measurements per sensor

Which looks like this in a diagram:



UML diagram database model

This can be achieved by adding two classes to our project in a package (directory) “entity”. First, we create one for the sensors called “SensorEntity.java”. As you can see this code contains a lot of “annotations” (start with @) which will help the framework to map the code to the database.

```

1  /**
2   * Maps a database entity from the table SENSORS to a Java object.
3   * The ID is marked as the unique identifier.
4   */
5  @Entity
6  @Table(name = "SENSORS",
7         uniqueConstraints={@UniqueConstraint(
8             name="UN_SENSOR_ID",
9             columnNames={"ID"})})
10 public class SensorEntity {
11     /**
12      * Auto-generated identifier to have a unique key for this sensor.
13      */
14     @Id
15     @GeneratedValue
16     private Long id;
17
18     /**
19      * Name of the sensor, a required value.
20      */
21     @Column(nullable = false)
22     private String name;
23
24     /**
25      * Relationship between the sensor and a list of measurements.
26      */
27     @OneToMany(
28         mappedBy = "sensorEntity",
29         cascade = {CascadeType.MERGE},
30         fetch = FetchType.LAZY
  
```

```
31     )
32     private Set<MeasurementEntity> measurements = new HashSet<>();
33
34     /**
35      * No-argument constructor is needed for JPA.
36      */
37     public SensorEntity() {
38         // NOP
39     }
40
41     /**
42      * Constructor with a name value.
43      * @param name
44      */
45     public SensorEntity(String name) {
46         this.name = name;
47     }
48
49     // Getters and setters needed by JPA and the code.
50     public Long getId() {
51         return id;
52     }
53
54     public void setId(Long id) {
55         this.id = id;
56     }
57
58     public String getName() {
59         return name;
60     }
61
62     public void setName(String name) {
63         this.name = name;
64     }
65
66     public Set<MeasurementEntity> getDataEntries() {
67         return measurements;
68     }
69
70     public void setDataEntries(Set<MeasurementEntity> dataEntries) {
71         this.measurements = dataEntries;
72     }
73 }
```

The same way, we add a class “MeasurementEntity.java”

```
1  /**
2   * Maps a database entity from the table MEASUREMENTS to a Java object.
3   * The ID is marked as the unique identifier.
4   */
5  @Entity
6  @Table(name = "MEASUREMENTS",
7         uniqueConstraints={@UniqueConstraint(
8             name="UN_MEASUREMENT_ID",
9             columnNames={"ID"})})
10 public class MeasurementEntity {
11
12     /**
13      * Auto-generated identifier to have a unique key for this sensor.
14      */
15     @Id
16     @GeneratedValue
17     private Long id;
18
19     /**
20      * Relationship between the measurement and its sensor.
21      */
22     @ManyToOne
23     @JoinColumn(
24         name = "SENSOR_ID",
25         nullable = false,
26         foreignKey = @ForeignKey(name="FK_MEASUREMENT_SENSOR"))
27     private SensorEntity sensorEntity;
28
29     /**
30      * Timestamp of the measurement.
31      */
32     @Column(nullable = false)
33     private long timestamp;
34
35     /**
36      * Key for the type of measurement, e.g. "temperature", "distance"...
37      */
38     @Column(nullable = false)
39     private String key;
40
41     /**
```

```
42     * Value of the measurement
43     */
44     @Column(nullable = false)
45     private double value;
46
47     /**
48     * No-argument constructor is needed for JPA.
49     */
50     public MeasurementEntity() {
51         // NOP
52     }
53
54     /**
55     * Constructor with a name value.
56     * @param sensorEntity
57     * @param timestamp
58     * @param key
59     * @param value
60     */
61     public MeasurementEntity(SensorEntity sensorEntity, long timestamp,
62         String key, double value) {
63         this.sensorEntity = sensorEntity;
64         this.timestamp = timestamp;
65         this.key = key;
66         this.value = value;
67     }
68
69     // Getters and setters needed by JPA and the code.
70     ...
71
72     @JsonIgnore
73     public SensorEntity getSensor() {
74         return sensorEntity;
75     }
76
77     ...
78 }
```



Notice the “@JsonIgnore” on “getSensor()” in this entity class. It’s added here to avoid endless loops when we load the data with the REST-service. Otherwise, the application tries to nest measurements-with-sensors into sensors-with-measurements resulting in this error:

```

1 Request processing failed; nested exception is
2 org.springframework.http.converter.HttpMessageNotWritableException:
3 Could not write JSON: Infinite recursion (StackOverflowError);
4 nested exception is com.fasterxml.jackson.databind.JsonMappingException:
5 Infinite recursion (StackOverflowError) (through reference chain:
6 be.webtechie.jvaspringrestdb.entity.SensorEntity...)
```

These two classes are all we need to define the database tables and how data needs to be stored!

## Storing data in the database

We will do this with two repository-classes and minimal code! Let’s start again with the Sensors and create a package “repository” and a file “SensorRepository.java”. By extending from “JpaRepository” we get most CRUD-functionality (Create, Read, Update and Delete) for free, and only need to define the additional methods we need. By using the same parameter-names as the values in the entities, Spring will handle the functionality for us!

```

1 @Repository
2 public interface SensorRepository extends JpaRepository<SensorEntity, Long>{
3     Page<SensorEntity> findAll(Pageable pageable);
4
5     List<SensorEntity> findAllByName(String name);
6
7     SensorEntity findById(@Param("id") long id);
8 }
```

“MeasurementRepository.java” is even smaller:

```

1 @Repository
2 public interface MeasurementRepository extends
3     JpaRepository<MeasurementEntity, Long>{
4     Page<MeasurementEntity> findAll(Pageable pageable);
5 }
```

## Adding the REST-services

Now let’s expose our database functionality with REST-services so anyone can read and write data from and into the database!

Create a package “resource” and a file “SensorResource.java”. In here we define three services which will be available on an URL:

- GET localhost:8080/sensor: all the sensors from the database
- GET localhost:8080/sensor/id: one sensor from the database
- POST localhost:8080/sensor: add a sensor with the given name, after checking this name isn't already used

All this is done through the SensorRepository we created before:

```
1  @RestController
2  public class SensorResource {
3      @Autowired
4      private SensorRepository sensorRepository;
5
6      @GetMapping("/sensor")
7      public List<SensorEntity> retrieveAllSensors() {
8          return sensorRepository.findAll();
9      }
10
11     @GetMapping("/sensor/{id}")
12     public SensorEntity retrieveSensor(@RequestParam long id) {
13         return sensorRepository.findById(id);
14     }
15
16     @PostMapping("/sensor")
17     public ResponseEntity createSensor(@RequestParam String name) {
18         List<SensorEntity> sensorEntities = sensorRepository.findAllByName(name);
19
20         if (sensorEntities.size() > 0) {
21             return ResponseEntity.status(HttpStatus.BAD_REQUEST)
22                 .body("There is already a sensor with the name: " + name);
23         }
24
25         SensorEntity sensorEntity = new SensorEntity(name);
26         sensorRepository.save(sensorEntity);
27         return ResponseEntity.ok(sensorEntity);
28     }
29 }
```

We need to do the same for the measurements to provide these URLs:

- GET localhost:8080/measurement: all the measurements from the database

- POST localhost:8080/measurement: add a measurement for the given sensor ID, key and value, after checking the given sensor ID is defined in the database:

```
1 @RestController
2 public class MeasurementResource {
3     @Autowired
4     private SensorRepository sensorRepository;
5
6     @Autowired
7     private MeasurementRepository measurementRepository;
8
9     @GetMapping("/measurement")
10    public List<MeasurementEntity> retrieveAllMeasurements() {
11        return measurementRepository.findAll();
12    }
13
14    @PostMapping("/measurement")
15    public ResponseEntity createMeasurement(
16        @RequestParam long sensorId,
17        @RequestParam String key,
18        @RequestParam double value) {
19
20        SensorEntity sensorEntity = sensorRepository.findById(sensorId);
21
22        if (sensorEntity == null) {
23            return ResponseEntity.status(HttpStatus.BAD_REQUEST)
24                .body("No sensor defined with the ID: " + sensorId);
25        }
26
27        MeasurementEntity measurementEntity = new MeasurementEntity(
28            sensorEntity, System.currentTimeMillis(), key, value);
29        measurementRepository.save(measurementEntity);
30
31        return ResponseEntity.ok().build();
32    }
33 }
```

## Adding Swagger

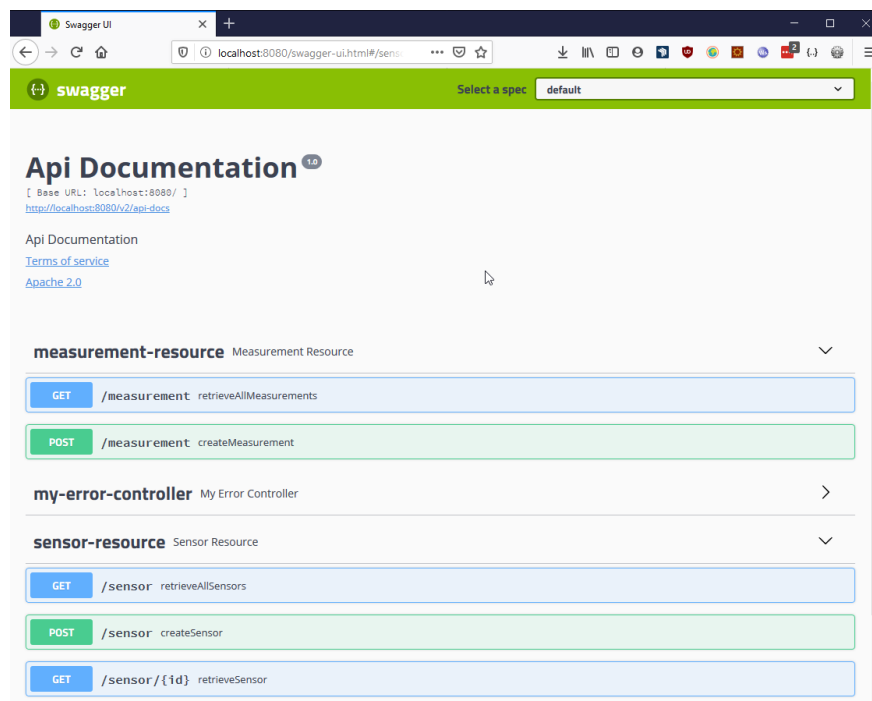
One final coding step, so we can use Swagger to quickly test our application! Create a package “configuration” with the file “SwaggerConfig.java” to expose our REST-resources:



```
1 @Configuration
2 @EnableSwagger2
3 public class SwaggerConfig {
4     @Bean
5     public Docket api() {
6         return new Docket(DocumentationType.SWAGGER_2)
7             .select()
8             .apis(RequestHandlerSelectors.any())
9             .paths(PathSelectors.any())
10            .build();
11    }
12 }
```

## Running the application and using the REST-services

Let's try it out! Hit run on the main function in "JavaSpringRestDbApplication.java" and browse to <http://localhost:8080/swagger-ui.html>



Swagger with the sensor and measurement resources

Let's start by creating some sensors by clicking on "POST /sensor" and "Try it out". Fill in some values (e.g. "temp1", "temp2"...) and each time hit "Execute".

POST /sensor createSensor

Parameters Cancel

| Name                                 | Description |
|--------------------------------------|-------------|
| name * required<br>string<br>(query) | name        |

temp1

Execute

Posting a sensor in Swagger

If you try to add the same name twice, an error will be returned:

Code: 400 Undocumented

Details: Error:

Response body

```
There is already a sensor with the name: temp2
```

Download

Posting error in Swagger when a sensor already exists

Now you can see the created records in Swagger by trying out the GET's for “/sensor” and “/sensor/”, but also by browsing directly to <http://localhost:8080/sensor>. The data is formatted in this screenshot of Firefox with the “JSON-formatter” plugin:

localhost:8080/sensor

localhost:8080/sensor

JSON Onbewerkte gegevens Headers

Opslaan Kopiëren Alles samenvouwen Alles uitvouwen JSON filteren

```

0:
  id: 1
  name: "temp1"
  dataEntries: []
1:
  id: 2
  name: "temp2"
  dataEntries: []

```

JSON data of the sensors in the browser

Next step: storing some measurements for one of our sensors (ID 1):

| Name                                               | Description   |
|----------------------------------------------------|---------------|
| key * required<br>string<br>(query)                | key<br>°C     |
| sensorId * required<br>integer(\$int64)<br>(query) | sensorId<br>1 |
| value * required<br>number(\$double)<br>(query)    | value<br>24   |

Posting a measurement in Swagger

Now we can get the measurements of this sensor by browsing to <http://localhost:8080/sensor/1>:

```

{
  "id": 1,
  "name": "temp1",
  "dataEntries": [
    {
      "id": 5,
      "timestamp": 1577089744799,
      "key": "°C",
      "value": 29
    },
    {
      "id": 3,
      "timestamp": 1577089738960,
      "key": "°C",
      "value": 24
    },
    {
      "id": 4,
      "timestamp": 1577089741950,
      "key": "°C",
      "value": 26
    }
  ]
}

```

JSON data of the sensors in the browser

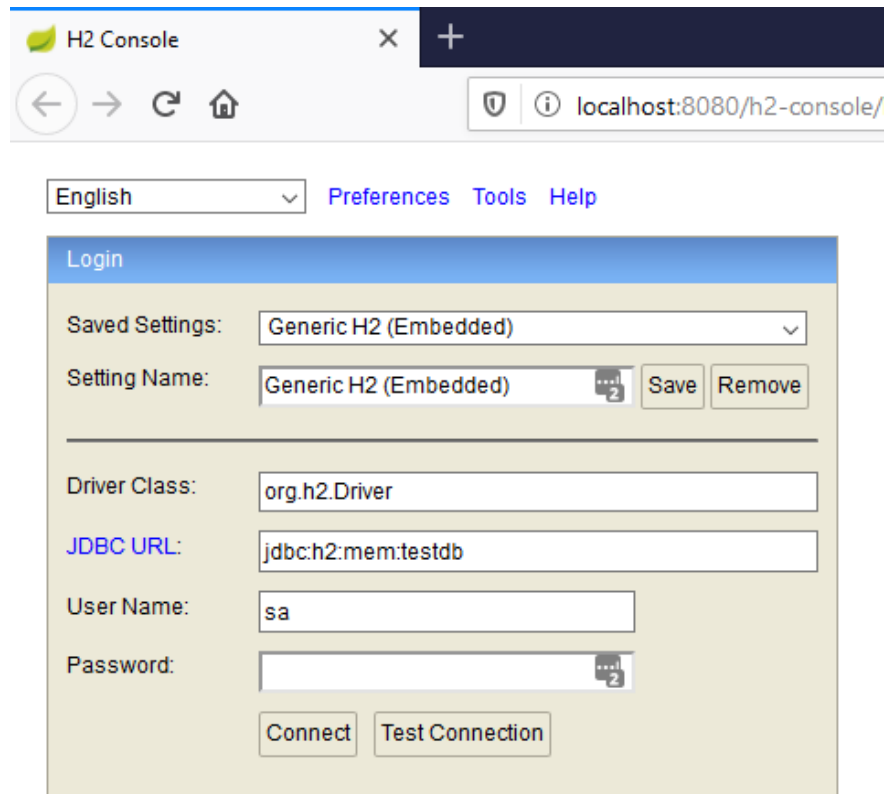
## Checking the data in H2-console

If you add an additional setting in “src > main > resources > application.properties” something magical happens. Out of the blue, our application is extended with a full database browser!

```
1 spring.h2.console.enabled=true
```

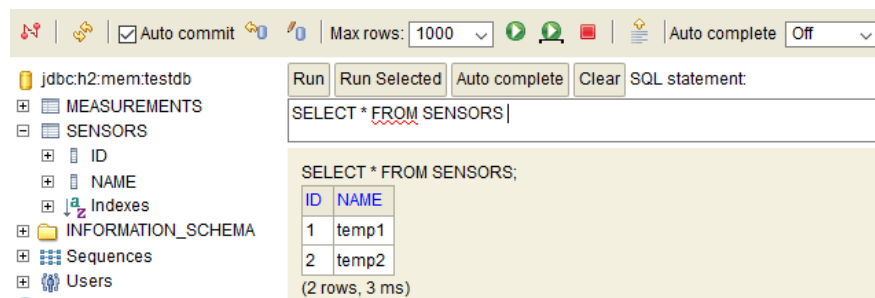
Restart your application so the new setting is applied. Because H2 by default stores everything in memory, our test data is gone, and we need to first generate some again with Swagger as described

before. Now browse to <http://localhost:8080/h2-console> and make sure “jdbc:h2:mem:testdb” is used as “JDBC URL”.

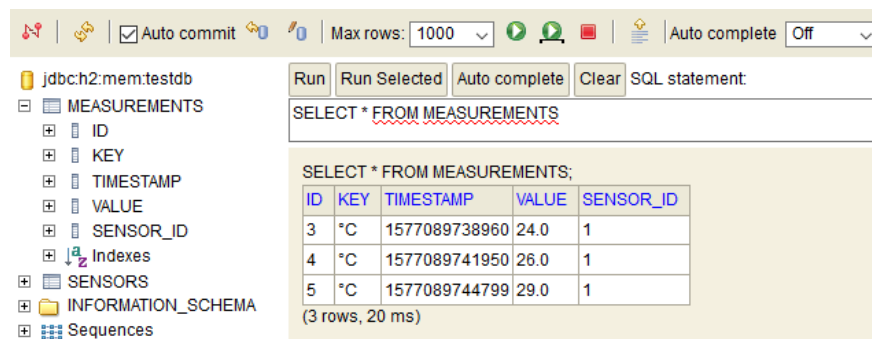


Login for H2 console

After login we see the database structure as we defined it in our code for both tables:



Sensors table in H2 console



Measurements table in H2 console

## Configuration to run on the Pi

Our application works on PC, let's prepare the Pi now. Spring will look for an application.properties file in the config directory at startup and if found, will use that one instead of the one included in the jar. Let's use this functionality to reconfigure the application to use a file-database instead of in-memory, so the data is not lost when restarting.

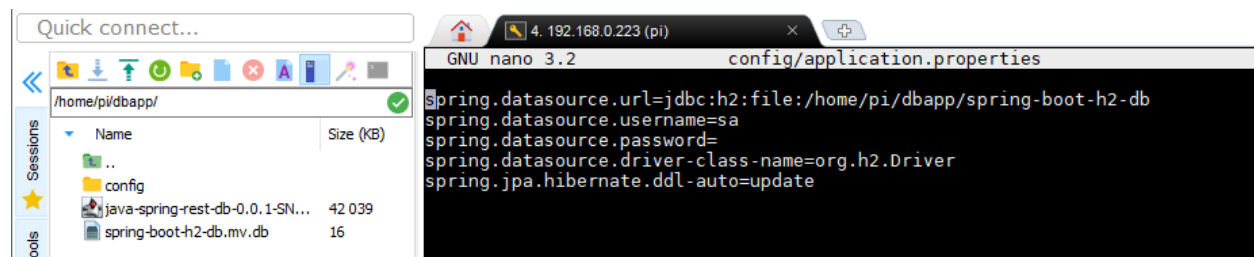
We will make a directory for the application and add the properties like this:

```

1 $ mkdir /home/pi/dbapp
2 $ cd /home/pi/dbapp
3 $ mkdir config
4 $ nano config/application.properties
5
6 spring.datasource.url=jdbc:h2:file:/home/pi/dbapp/spring-boot-h2-db
7 spring.datasource.username=sa
8 spring.datasource.password=
9 spring.datasource.driver-class-name=org.h2.Driver
10 spring.jpa.hibernate.ddl-auto=update

```

Build the application on your PC to a jar with “mvn clean package” and copy “java-spring-rest-db-0.0.1-SNAPSHOT.jar” from the “target” directory to your Pi in the “/home/pi/dbapp” directory.



Database application and properties on the Pi

In this screenshot, you also see the created database file “spring-boot-h2-db.mv.db” as defined in the properties file. When adding data via Swagger, you will see the size of this file grow.

Running the application gives the expected output similar to the one on the PC:

```

1  $ java -jar java-spring-rest-db-0.0.1-SNAPSHOT.jar
2
3
4  .   ____          _            __ _ _
5  /\\ / ___'_   _'  _/_/  ___/_ _/ ___/_ _/ ___/_ _/ ___/
6  (  ( / ___| | | |'_/_/_/  ___/_ _/ ___/_ _/ ___/_ _/ ___/
7  \\/  ___)| |_)| | | |'_/_/_/  ___/_ _/ ___/_ _/ ___/_ _/ ___/
8  =====|_|=====|__/=/_/_/_/
9  :: Spring Boot ::                (v2.1.8.RELEASE)
10
11 Starting JavaSpringRestDbApplication v0.0.1-SNAPSHOT on raspberrypi with PID 4557
12     (/home/pi/dbapp/java-spring-rest-db-0.0.1-SNAPSHOT.jar
13     started by pi in /home/pi/dbapp)
14 No active profile set, falling back to default profiles: default
15 Bootstrapping Spring Data repositories in DEFAULT mode.
16 Finished Spring Data repository scanning in 465ms. Found 2 repository interfaces.
17 ...
18 Tomcat started on port(s): 8080 (http) with context path ''
19 Started JavaSpringRestDbApplication in 63.416 seconds (JVM running for 67.637)

```

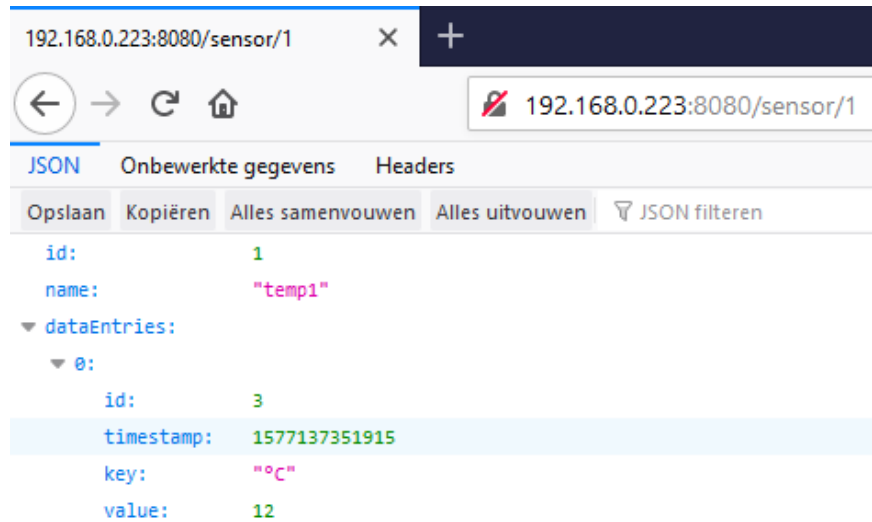
The big difference is this last line when compared to my development PC:

```

1 Started JavaSpringRestDbApplication in 5.452 seconds (JVM running for 7.338)

```

Indeed you'll need to take into account this application needs a longer start-up time on the Pi. But as soon as it runs, you can access Swagger and the REST-services from any PC within your network on the IP address of your Pi, in my case 192.168.0.223:



REST-service from the Pi

## Conclusion

Just a basic example with two tables, but it shows you how quickly and with minimal code, a database application can be built. The REST-services are available to all your devices who can connect to your Pi via the network so they can store data in one central place and/or can read that data.

The JSON data provided by the REST-services can be used by all different kinds of devices or applications to visualize the results. On the [Elektor website](https://www.elektormagazine.com/labs/9292-dutch-public-transport-monitor) you can find a nice example of a microcontroller doing this to show JSON data on a screen<sup>116</sup>.

<sup>116</sup><https://www.elektormagazine.com/labs/9292-dutch-public-transport-monitor>

## Interview with Vlad Mihalcea



Vlad Mihalcea [@vlad\\_mihalcea<sup>a</sup>](#), Java Champion, CEO of Hypersistence, JPA expert, one of the top Hibernate ORM committers.

*Java, Java Persistence API (JPA), and Hibernate allow building a database application with minimal code. All made possible by free and open-source frameworks! What is your motivation to contribute to these developments?*

JPA and Hibernate are, indeed, very convenient when building a data access layer, and this is the reason why they are used by the vast majority of enterprise applications.

Not only Java EE, but Spring Boot or JHipster use Hibernate by default, so, there's no wonder Hibernate is so popular.

My main motivation for contributing to Hibernate is the desire to teach people how to get the most out of JPA, Hibernate, or the database system they are using.

That's the reason I wrote [High-Performance Java Persistence<sup>b</sup>](#), and started projects such as [Hibernate Types<sup>c</sup>](#) or [Hypersistence Optimizer<sup>d</sup>](#).

*You share a lot of tips and tricks on Twitter, your site, StackOverflow, books, talks... I keep wondering how all these experts like you, achieve this combined with a job and family?*

Well, it was quite difficult in the beginning when I was working a full-time job and doing a lot of writing on my blog or answering questions on StackOverflow.

However, 4 years ago, I quit my job and turned my passion for high-performance database systems into a full-time job. Nowadays, blogging, training, and writing books is part of my job, and I manage to do all that in less than 40 hours per week.

Also, one great benefit of working from home is that it allows me to spend a lot of time with my family.

*The database example in this book works with some simple tables and an H2-database. This works very well on a Raspberry Pi. It's astonishing to me that the same technology used in major big data solutions can also run on a PC which costs less than 50€. Does this surprise you?*

No, not at all. That's exactly why Java is one of the most popular programming languages. The ecosystem is very diverse, making Java a very flexible choice in the long run.

Relational databases come in many forms. While banks use Oracle to handle billions of records, smartphones can use SQLite instead. Nevertheless, both Oracle and SQLite implement many features of the SQL Standard, hence the flexibility you get when using a relational database.

*In your opinion what are the biggest changes in Java and databases we had in the last years and are still to come?*



The new development cycle of Java, with a release every 6 months, allows the Java and JDK developers to introduce many useful features. And, there are also many great features to come, like Project Loom<sup>e</sup> or Valhalla<sup>f</sup>.

*Why is now the perfect time to learn Java?*

Because Java has a rich ecosystem of open-source frameworks and being able to run it on a great variety of systems, ranging from small devices to supercomputers, Java and the JVM are a great choice.

*Which DIY-programming-electronics-project are you working on, or is on your “if I ever have time” list?*

**Hypersistence Optimizer<sup>g</sup>** is what I’m currently working on, and I’m very excited about it as it allows you to automate the process of detecting JPA and Hibernate issues, so you can finally focus on data access logic instead of chasing performance-related issues.

<sup>e</sup>[https://twitter.com/vlad\\_mihalcea](https://twitter.com/vlad_mihalcea)

<sup>f</sup><https://vladmihalcea.com/books/high-performance-java-persistence/>

<sup>g</sup><https://github.com/vladmihalcea/hibernate-types>

<sup>h</sup><https://vladmihalcea.com/hypersistence-optimizer/>

<sup>e</sup>Project Loom wants to make it easier to write, debug, profile and maintain concurrent Java applications.

<sup>f</sup>Project Valhalla combines multiple sub-projects aiming to adapt the Java language and runtime to modern hardware.

<sup>g</sup><https://vladmihalcea.com/hypersistence-optimizer/>

## Example 3: REST-service on the Pi to toggle an LED

In this example, we will be working on top of Pi4J (see “Chapter 9: Pi4J”) to create a proof-of-concept application that combines Spring and Pi4J. Because of the dependency of this library, we can not run or test this application on the PC and need to run the produced jar directly on a Pi.



Full sources can be found in:  
Chapter\_10\_Spring > java-spring-rest-gpio

We start from the minimal Spring sources again and need to add this dependency:

```
1 <dependency>
2     <groupId>com.pi4j</groupId>
3     <artifactId>pi4j-core</artifactId>
4     <version>1.2</version>
5     <scope>compile</scope>
6 </dependency>
```

### Info REST-controller

The first thing we are going to expose with this application, is the information provided by the Pi4J library.

Create a package “controller” with a file “InfoRestController.java”. In the sources, you can find the full code, but this is a short piece. Each method is a REST-mapping which returns a specific set of key-value pairs with info about your Raspberry Pi.

```
1 /**
2  * Provides a REST-interface to expose all board info.
3  *
4  * Based on https://pi4j.com/1.2/example/system-info.html
5  */
6 @RestController
7 @RequestMapping("info")
8 public class InfoRestController {
9     private Logger logger = LoggerFactory.getLogger(this.getClass());
10
11     /**
12     * Get the OS info.
13     */
14     @GetMapping(path = "os", produces = "application/json")
```

```
15     public Map<String, String> getOsInfo() {
16         Map<String, String> map = new TreeMap<>();
17         try {
18             map.put("Name", SystemInfo.getOsName());
19         } catch (Exception ex) {
20             logger.error("OS name not available, error: {}", ex.getMessage());
21         }
22         try {
23             map.put("Version", SystemInfo.getOsVersion());
24         } catch (Exception ex) {
25             logger.error("OS version not available, error: {}", ex.getMessage());
26         }
27         return map;
28     }
29
30     /**
31      * Get the Java info.
32      */
33     @GetMapping(path = "java", produces = "application/json")
34     public Map<String, String> getJavaInfo() {
35         Map<String, String> map = new TreeMap<>();
36         map.put("Vendor ", SystemInfo.getJavaVendor());
37         map.put("VendorURL", SystemInfo.getJavaVendorUrl());
38         map.put("Version", SystemInfo.getJavaVersion());
39         map.put("VM", SystemInfo.getJavaVirtualMachine());
40         map.put("Runtime", SystemInfo.getJavaRuntime());
41         return map;
42     }
43 }
```

## GPIO Manager

Before we can start creating the GPIO REST-controller, we will add `GpioManager` which handles the Pi4J calls. We will use this manager to store the initialized GPIO pins and call the Pi4J methods to interact with the GPIOs. Check the sources for the full class.

```
1  /**
2   * Singleton instance for the {@link GpioFactory}.
3   * SCOPE_SINGLETON is the default value, but added for clarity.
4   */
5  @Component
6  @Scope(value = ConfigurableBeanFactory.SCOPE_SINGLETON)
7  public class GpioManager {
8      private Logger logger = LoggerFactory.getLogger(this.getClass());
9
10     /**
11      * The GPIO controller.
12      */
13     private final GpioController gpio;
14
15     /**
16      * List of the provisioned pins with the address as key.
17      */
18     private final Map<Integer, Object> provisionedPins = new HashMap<>();
19
20     /**
21      * Constructor which initializes the Pi4J {@link GpioController}.
22      */
23     public GpioManager() {
24         this.gpio = GpioFactory.getInstance();
25     }
26
27     /**
28      * Get the pin for the given address.
29      *
30      * @param address The address of the GPIO pin.
31      * @return The {@link Pin} or null when not found.
32      */
33     private Pin getPinByAddress(int address) {
34         Pin pin = RaspiPin.getPinByAddress(address);
35         if (pin == null) {
36             logger.error("No pin available for address {}", address);
37         }
38         return pin;
39     }
40
41     /**
42      * Provision a GPIO as digital output pin.
43      *
```

```
44     * @param address The address of the GPIO pin.
45     * @param name The name of the GPIO pin.
46     * @return True if successful.
47     */
48     public boolean provisionDigitalOutputPin(final int address, final String name) {
49         if (this.provisionedPins.containsKey(address)) {
50             throw new IllegalArgumentException("There is already a provisioned pin"
51                 + " at the given address");
52         }
53
54         final GpioPinDigitalOutput provisionedPin = this.gpio
55             .provisionDigitalOutputPin(
56                 this.getPinByAddress(address), name, PinState.HIGH);
57         provisionedPin.setShutdownOptions(true, PinState.LOW);
58
59         this.provisionedPins.put(address, provisionedPin);
60
61         return true;
62     }
63
64     ...
65
66     /**
67     * Toggle a pin.
68     *
69     * @param address The address of the GPIO pin.
70     * @return True if successful.
71     */
72     public boolean togglePin(final int address) {
73         logger.info("Toggle pin requested for address {}", address);
74
75         Object provisionedPin = this.provisionedPins.get(address);
76
77         if (provisionedPin == null) {
78             throw new IllegalArgumentException("There is no pin provisioned"
79                 + " at the given address");
80         } else {
81             if (provisionedPin instanceof GpioPinDigitalOutput) {
82                 ((GpioPinDigitalOutput) provisionedPin).toggle();
83
84                 return true;
85             } else {
86                 throw new IllegalArgumentException("The provisioned pin at"
```

```

87         + " the given address is not of the type GpioPinDigitalOutput");
88     }
89 }
90 }
91
92 ...
93
94 }

```

## GPIO REST-controller

Let's add a controller to expose the Pi4J GPIO-methods we integrated in the GpioManager.java class. Below only a few of the methods are included, you can find all of them in the sources:

```

1  /**
2   * Provides a REST-interface with the pins.
3   */
4  @RestController
5  @RequestMapping("gpio")
6  public class GpioRestController {
7      private Logger logger = LoggerFactory.getLogger(this.getClass());
8
9      private final GpioManager gpioManager;
10
11     public GpioRestController(GpioManager gpioManager) {
12         this.gpioManager = gpioManager;
13     }
14
15     ...
16
17     /**
18      * Provision a GPIO as digital output pin.
19      *
20      * @param address The address of the GPIO pin.
21      * @param name The name of the GPIO pin.
22      * @return True if successful.
23      */
24     @PostMapping(
25         path = "provision/digital/output/{address}/{name}",
26         produces = "application/json")
27     public boolean provisionDigitalOutputPin(@PathVariable("address") int address,
28         @PathVariable("name") String name) {

```

```
29     return this.gpioManager.provisionDigitalOutputPin(address, name);
30 }
31
32 ...
33
34 /**
35  * Toggle a pin.
36  *
37  * @param address The address of the GPIO pin.
38  * @return True if successful.
39  */
40 @PostMapping(
41     path = "digital/toggle/{address}",
42     produces = "application/json")
43 public boolean togglePin(@PathVariable("address") long address) {
44     return this.gpioManager.togglePin((int) address);
45 }
46
47 ...
48 }
```

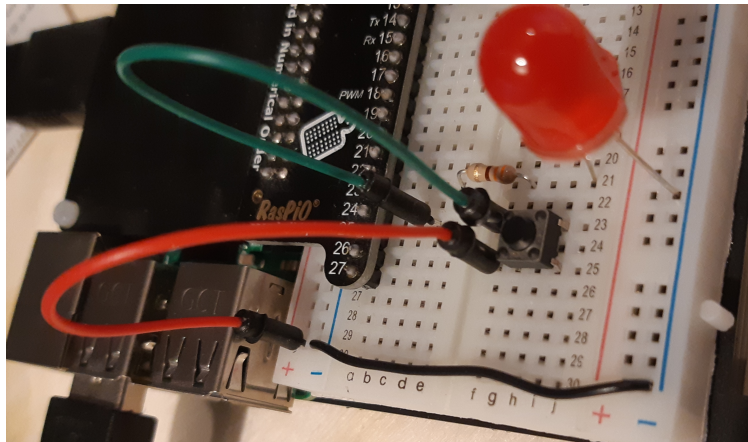
## Running the application on a Pi

By adding a Swagger config (see the sources and previous examples), we can easily test the REST-controllers.

Build the jar with “mvn clean package” and copy the jar file from the target directory to the Pi. Run it with “java -jar java-spring-rest-gpio-0.0.1-SNAPSHOT.jar”.

To demonstrate this application, we will be using the same wiring as the one used in “Chapter 7: JavaFX > Example 1: TilesFX dashboard” with:

- LED on GPIO 22 (WiringPi n° 3)
- Button on GPIO 24 (WiringPi n° 5)



Wiring on a breadboard

Browse to your Pi to the Swagger page from any PC in the same network and you will see the two controllers with the methods listed here:

| info-rest-controller Info Rest Controller |                                |
|-------------------------------------------|--------------------------------|
| GET                                       | /info/codec getCodecInfo       |
| GET                                       | /info/frequencies getClockInfo |
| GET                                       | /info/hardware getHardwareInfo |
| GET                                       | /info/java getJavaInfo         |
| GET                                       | /info/memory getMemoryInfo     |
| GET                                       | /info/network getSystemInfo    |
| GET                                       | /info/os getOsInfo             |
| GET                                       | /info/platform getPlatform     |

Swagger info-controller

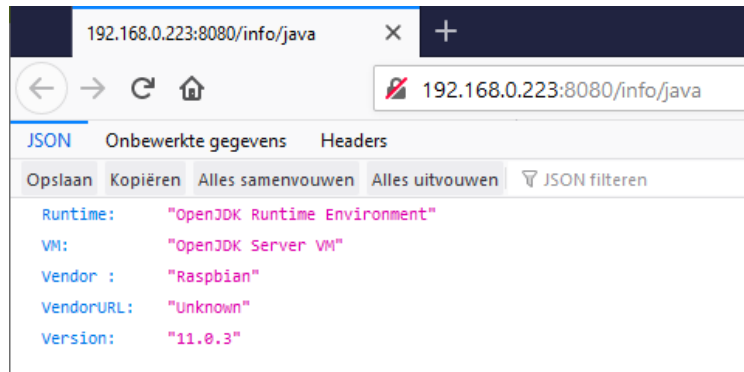
| gpio-rest-controller Gpio Rest Controller |                                                                           |
|-------------------------------------------|---------------------------------------------------------------------------|
| POST                                      | /gpio/digital/pulse/{address}/{duration} pulsePin                         |
| POST                                      | /gpio/digital/state/{address}/{value} setPinDigitalState                  |
| POST                                      | /gpio/digital/toggle/{address} togglePin                                  |
| POST                                      | /gpio/provision/digital/input/{address}/{name} provisionDigitalInputPin   |
| POST                                      | /gpio/provision/digital/output/{address}/{name} provisionDigitalOutputPin |
| GET                                       | /gpio/provision/list getProvisionList                                     |
| GET                                       | /gpio/state/{address} getState                                            |

Swagger GPIO-controller

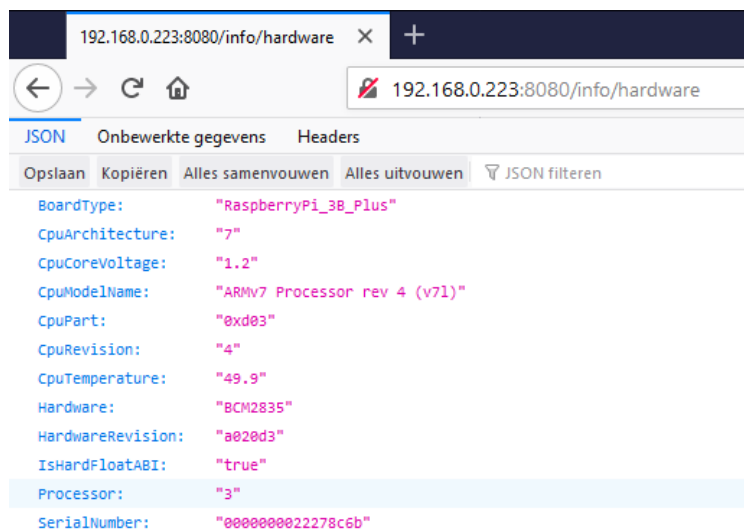
## Testing the info REST-controller

We can click on the Swagger page and execute one of the available options, but let's try an info method by going to the URL directly. These are some of the results on a fresh new Raspbian 3B+ board:





Info Java REST-service



Info Hardware REST-service

## Testing the GPIO REST-controller with an LED and button

To be able to control the connected LED and read out the button state, we first need to initialize the GPIOs. This can be done with two dedicated methods in the GPIO controller and using the WiringPi numbers:

**POST** /gpio/provision/digital/output/{address}/{name} provisionDigitalOutputPin

Parameters Cancel

| Name                                             | Description  |
|--------------------------------------------------|--------------|
| address * required<br>integer(\$int32)<br>(path) | address<br>3 |
| name * required<br>string<br>(path)              | name<br>LED  |

Execute

**POST** /gpio/provision/digital/input/{address}/{name} provisionDigitalInputPin

Parameters Cancel

| Name                                             | Description    |
|--------------------------------------------------|----------------|
| address * required<br>integer(\$int32)<br>(path) | address<br>5   |
| name * required<br>string<br>(path)              | name<br>Button |

Execute

### Initialize GPIO pins with Swagger

When the GPIOs are initialized we can get a list of them:

```

192.168.0.223:8080/gpio/provision/ | 192.168.0.223:8080/gpio/provision/list
JSON  Onbewerkte gegevens  Headers
Opslaan  Kopiëren  Alles samenvouwen  Alles uitvouwen  JSON filteren
▼ ProvisionedPin_3:
  address: "3"
  mode:    "output"
  name:    "LED"
  pinName: "GPIO 3"
  state:   "1"
  type:    "com.pi4j.io.gpio.impl.GpioPinImpl"
▼ ProvisionedPin_5:
  address: "5"
  mode:    "input"
  name:    "Button"
  pinName: "GPIO 5"
  state:   "0"
  type:    "com.pi4j.io.gpio.impl.GpioPinImpl"

```

### List of initialized GPIOs

As we checked the GPIOs are ready to use, we can now toggle the LED on and off by clicking on the “Execute”-button again and again:

**POST** /gpio/digital/toggle/{address} togglePin

Parameters Cancel

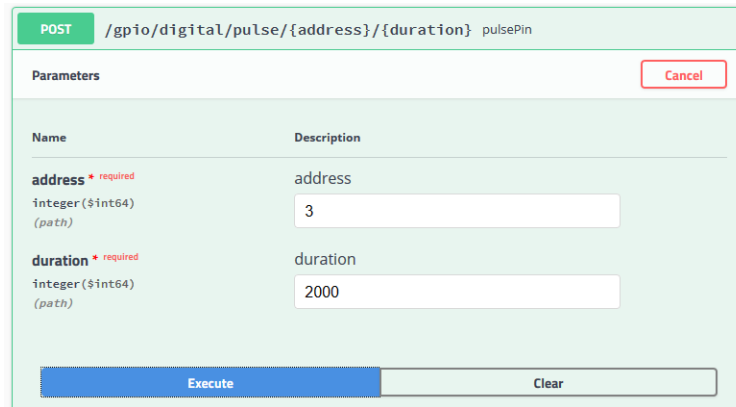
| Name                                             | Description  |
|--------------------------------------------------|--------------|
| address * required<br>integer(\$int64)<br>(path) | address<br>3 |

Execute
Clear

### Toggle the LED on or off

There is also an additional method to put the LED on for a given time, in this example 2 seconds,

but the duration must be provided in milliseconds:



POST /gpio/digital/pulse/{address}/{duration} pulsePin

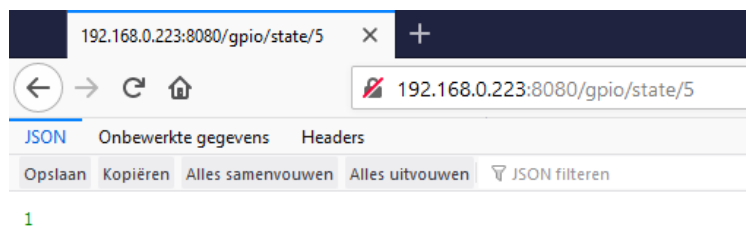
Parameters Cancel

| Name                                                     | Description |
|----------------------------------------------------------|-------------|
| <b>address</b> * required<br>integer(\$int64)<br>(path)  | address     |
| <b>duration</b> * required<br>integer(\$int64)<br>(path) | duration    |

Execute Clear

Toggle the LED on for a given duration

Requesting the state of the button can be done via Swagger but also directly via the URL. In this case, the button is pressed and returns 1:



Read the state of an input GPIO

## Conclusion

This application only exposes a few of the Pi4J methods as a REST-service to show the possibilities and power of this approach. Depending on the project you want to build, you can extend or rework this example to fit your exact needs.

## Example 4: Reactive data

[Reactive programming](#)<sup>117</sup> uses a different approach compared to the previous examples where data can be requested from a REST-service when it's needed. In a reactive system, continuous streams are used. For instance, to update a user interface based on the incoming data.

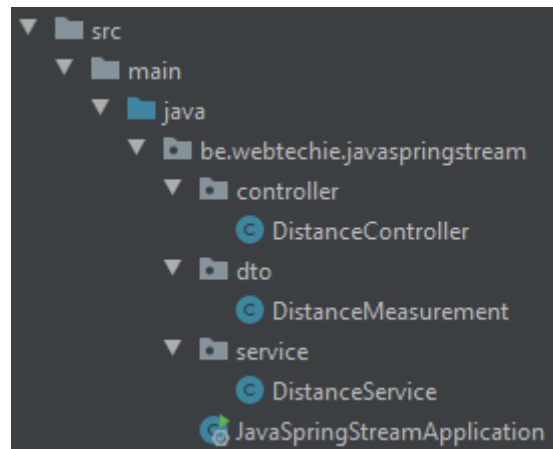
Trisha Gee (see the interview in “Chapter 3: Choosing an IDE”) and Josh Long (Spring Developer Advocate at Pivotal, [@starbuxman](#)<sup>118</sup>) worked together on a [blog series](#)<sup>119</sup> in which they showed the power of reactive data produced by a Spring application.

Their example uses a Kotlin service to continuously send stock values, a Java client to receive this data and a JavaFX client to visualize the updating prices in real-time on a line chart.

In this example application, we will achieve that same functionality on the Pi, based on the example from “Chapter 9: Pi4J” with the distance sensor. We use the same wiring with some rework of the code, integrated into a Spring application.

### The code

This project uses three classes in this structure:



Spring stream project structure



Full sources can be found in:  
Chapter\_10\_Spring > java-spring-stream

An extra dependency needs to be added to pom.xml:

<sup>117</sup>[https://en.wikipedia.org/wiki/Reactive\\_programming](https://en.wikipedia.org/wiki/Reactive_programming)

<sup>118</sup><https://twitter.com/starbuxman>

<sup>119</sup>[https://trishagee.github.io/presentation/coding\\_duel/](https://trishagee.github.io/presentation/coding_duel/)

```
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-webflux</artifactId>
4 </dependency>
```

## DTO (Data Transfer Object)

This object will be used to push data into the stream every second.

```
1 package be.webtechie.javaspringstream.dto;
2
3 /**
4  * Object containing a distance measurement.
5  */
6 public class DistanceMeasurement {
7     private final long timestamp;
8     private final int distance;
9     private final float duration;
10
11     /**
12      * Constructor which will add the current timestamp.
13      *
14      * @param distance The distance in centimeter
15      * @param duration The measurement duration in nanos
16      */
17     public DistanceMeasurement(int distance, float duration) {
18         this.timestamp = System.currentTimeMillis();
19         this.distance = distance;
20         this.duration = duration;
21     }
22
23     public long getTimestamp() {
24         return timestamp;
25     }
26
27     public int getDistance() {
28         return distance;
29     }
30
31     public float getDuration() {
32         return duration;
33     }
34 }
```

## Reactive controller

The controller provides the REST-endpoint which can be called from a client.

```

1  @RestController
2  @RequestMapping("/api")
3  public class DistanceController {
4
5      private final DistanceService distanceService;
6
7      public DistanceController(DistanceService distanceService) {
8          this.distanceService = distanceService;
9      }
10
11     @GetMapping(
12         value = "/distance",
13         produces = MediaType.APPLICATION_STREAM_JSON_VALUE
14     )
15     public Flux<DistanceMeasurement> distance() {
16         return this.distanceService.getDistances();
17     }
18 }

```

## Service to generate the stream

The magic happens in the service where the “Flux” is generated and gets new data on each interval. Most of the code has already been explained in the Pi4J chapter. The pins are initialized in the constructor and are used to do a measurement every second.

```

1  @Service
2  public class DistanceService {
3      private static final Logger logger = LoggerFactory
4          .getLogger(DistanceService.class);
5
6      private static final Pin PIN_TRIGGER = RaspiPin.GPIO_01;    // BCM 18
7      private static final Pin PIN_ECHO = RaspiPin.GPIO_05;      // BCM 24
8
9      private final GpioPinDigitalOutput trigger;
10     private final GpioPinDigitalInput echo;
11
12     public DistanceService() {
13         // Initialize the GPIO controller
14         GpioController gpio = GpioFactory.getInstance();

```

```
15
16     // Initialize the pins
17     this.trigger = gpio
18         .provisionDigitalOutputPin(PIN_TRIGGER, "Trigger", PinState.LOW);
19     this.echo = gpio
20         .provisionDigitalInputPin(PIN_ECHO, "Echo", PinPullResistance.PULL_UP);
21 }
22
23 public Flux<DistanceMeasurement> getDistances() {
24     return Flux
25         .fromStream(Stream.generate(() -> this.getDistanceMeasurement()))
26         .delayElements(Duration.ofSeconds(1));
27 }
28
29 private DistanceMeasurement getDistanceMeasurement() {
30     try {
31         // Set trigger high for 0.01ms
32         this.trigger.pulse(10, PinState.HIGH, true, TimeUnit.NANOSECONDS);
33
34         // Start the measurement
35         while (this.echo.isLow()) {
36             // Wait until the echo pin is high,
37             // indicating the ultrasound was sent
38         }
39         long start = System.nanoTime();
40
41         // Wait till measurement is finished
42         while (this.echo.isHigh()) {
43             // Wait until the echo pin is low,
44             // indicating the ultrasound was received back
45         }
46         long end = System.nanoTime();
47
48         // Output the distance
49         float measuredSeconds = (end - start) / 1000000000F;
50         int distance = Math.round(measuredSeconds * 34300 / 2);
51
52         logger.info("Measured distance is: {} for {}s",
53             distance, measuredSeconds);
54
55         return new DistanceMeasurement(distance, measuredSeconds);
56     } catch (Exception ex) {
57         logger.error("Error: {}", ex.getMessage());
```

```

58     }
59
60     return null;
61 }
62 }

```

## Running the streaming application on the Pi

After building the application with “mvn clean package” and copying the jar to the Pi, it can be started and will produce this output (timestamps and classes removed for readability) with the first “Measured distance” log lines appearing as soon as a client connects to the API:

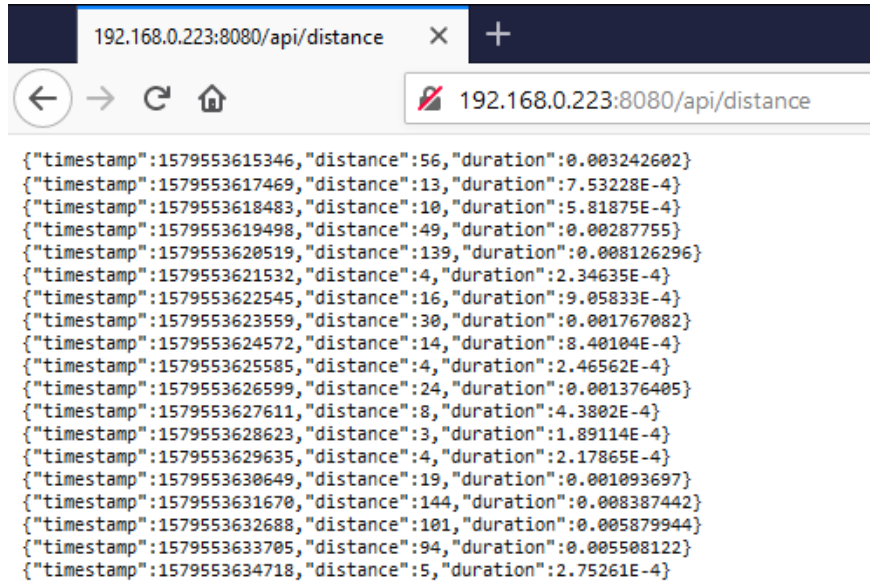
```

1  $ java -jar java-spring-stream-0.0.1-SNAPSHOT.jar
2
3
4  .  ____          _            __ _ _
5  /\\ / ___'_ __ _ _(_)' __ _ _/   \/_ \\  /
6  ( ( ) \___| '_ | '_| | '_ \ V  _ \ /\\ / /
7  \\/  ___| |_)| | | | |___| | \\|_| / / /
8  =====|_|=====|___/=/_/_/_/
9  :: Spring Boot ::                (v2.2.2.RELEASE)
10
11 Starting JavaSpringStreamApplication v0.0.1-SNAPSHOT on raspberrypi with PID
12 No active profile set, falling back to default profiles: default
13 ...
14 Completed initialization in 49 ms
15 ...
16 Measured distance is: 56 for 0.003242602s
17 Measured distance is: 13 for 7.53228E-4s
18 Measured distance is: 10 for 5.81875E-4s
19 Measured distance is: 49 for 0.00287755s
20 Measured distance is: 139 for 0.008126296s
21 ...

```

Browser output when calling the api service on the Pi (in my case on IP 192.169.0.223):





The screenshot shows a web browser window with the address bar displaying `192.168.0.223:8080/api/distance`. The browser's address bar includes navigation icons (back, forward, refresh, home) and a lock icon. The main content area of the browser displays a continuous stream of JSON objects, each representing a data point with a timestamp, distance, and duration. The data points are as follows:

```
{ "timestamp": 1579553615346, "distance": 56, "duration": 0.003242602 }
{ "timestamp": 1579553617469, "distance": 13, "duration": 7.53228E-4 }
{ "timestamp": 1579553618483, "distance": 10, "duration": 5.81875E-4 }
{ "timestamp": 1579553619498, "distance": 49, "duration": 0.00287755 }
{ "timestamp": 1579553620519, "distance": 139, "duration": 0.008126296 }
{ "timestamp": 1579553621532, "distance": 4, "duration": 2.34635E-4 }
{ "timestamp": 1579553622545, "distance": 16, "duration": 9.05833E-4 }
{ "timestamp": 1579553623559, "distance": 30, "duration": 0.001767082 }
{ "timestamp": 1579553624572, "distance": 14, "duration": 8.40104E-4 }
{ "timestamp": 1579553625585, "distance": 4, "duration": 2.46562E-4 }
{ "timestamp": 1579553626599, "distance": 24, "duration": 0.001376405 }
{ "timestamp": 1579553627611, "distance": 8, "duration": 4.3802E-4 }
{ "timestamp": 1579553628623, "distance": 3, "duration": 1.89114E-4 }
{ "timestamp": 1579553629635, "distance": 4, "duration": 2.17865E-4 }
{ "timestamp": 1579553630649, "distance": 19, "duration": 0.001093697 }
{ "timestamp": 1579553631670, "distance": 144, "duration": 0.008387442 }
{ "timestamp": 1579553632688, "distance": 101, "duration": 0.005879944 }
{ "timestamp": 1579553633705, "distance": 94, "duration": 0.005508122 }
{ "timestamp": 1579553634718, "distance": 5, "duration": 2.75261E-4 }
```

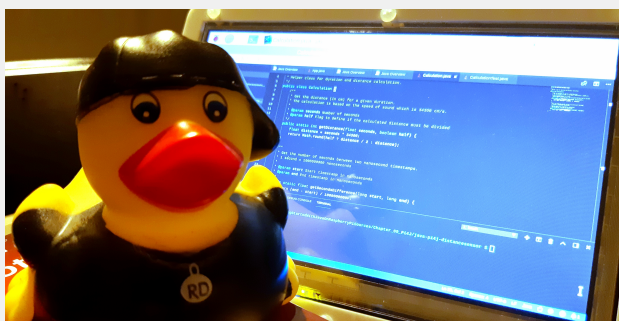
Stream output in the browser

## Conclusion

If we want to get data from a Pi sensor we can use reactive APIs and use the generated data on any other device which can handle and parse this streamed data.

# Just a thought: Impostor Syndrome

I have a confession to make. I'm suffering from a chronic disease, one which seems to be incurable, the **Impostor Syndrome**<sup>a</sup>. Every day I'm afraid of being exposed as a fraud, charlatan, hoax... People with the Impostor Syndrome are convinced their achievements are based on luck instead of knowledge or experience. They also believe other people think they are smarter than they are and will realize this sooner or later.



Although I often have this same feeling, I decided to embrace it! At work, I'm surrounded by colleagues who have a higher degree and are way smarter than me. I absorb all the knowledge they share. And we also share our failures and victories. If one of us gets stuck, we explain our problem to one another, which is the best first step to identify exactly what's the part of our code which

is not working as expected. And if no-one is available, we have a rubber duck<sup>b</sup> on our desk which is more than happy to listen to our puzzles.

Doubting yourself is a good thing, it makes you question your decisions, which proves you are worth doing the things you're doing. You got the job because you were the right person at the right moment. Use your doubts to improve your workflow, thinking, communication... And never forget that the opposite of an imposter is a narcissist, and I'm certain which of those two is my preferred type of colleague!

Do what you like to do, repeat it over and over again, just like professional sporters do. Maybe you're not an expert yet, but choose your battles wisely and one day you will be.

Rules to keep in mind...

***It's not a problem to not know, it's a problem to not have tried.***

***There are two types of decisions: good decisions and lessons learned.***

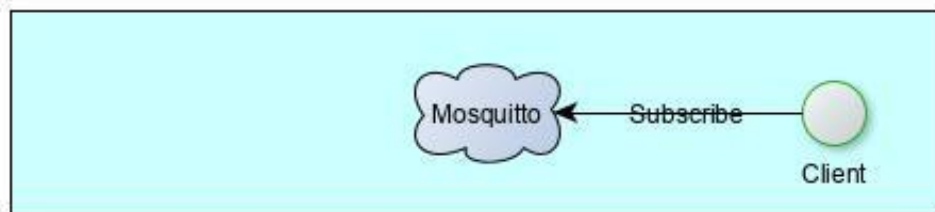
<sup>a</sup>[https://en.wikipedia.org/wiki/Impostor\\_syndrome](https://en.wikipedia.org/wiki/Impostor_syndrome)

<sup>b</sup>Using a **rubber duck** to explain your problem. By doing so you are forced to rethink the process which leads to your problem, which in many cases reveals the error. A good practice before involving (disturbing) a colleague.

# Chapter 11: Message Queues

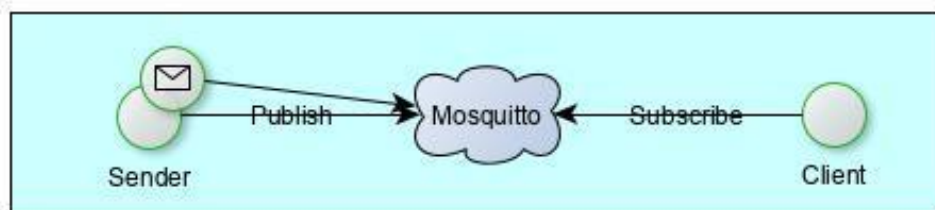
A message queue enables to decouple applications and devices, but let them interact with messages. In between them, we add a “Message Queue”. Some of the most popular ones are RabbitMQ, ActiveMQ, Mosquitto... These are software applications you install separately and act as a “post office” to receive messages and deliver them to all those who are interested in them.

In our case, we will install Mosquitto on a Pi and use it as our message delivery system. Once Mosquitto is running we can let an application subscribe to a topic in Mosquitto, for which we can freely define a name, e.g. “sensorMeasurement”, “ledCommands”.



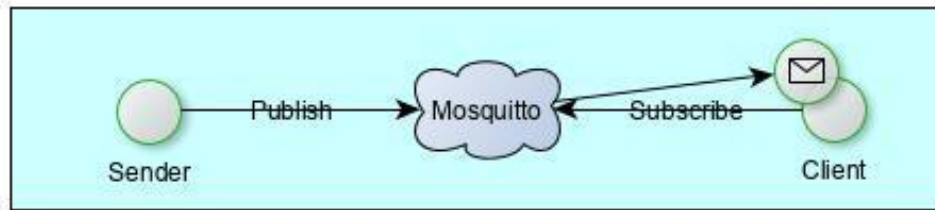
Subscribe on a topic in Mosquitto

Mosquitto will create the topic (= “mailbox”) as soon as one party wants to listen to it or publishes a message in it, so they don’t need to be configured in advance. Any application or device (or multiple ones) can publish a message to a topic.



Publish a message

Mosquitto will check who is subscribed to the topic and forward the message to all subscribed applications.



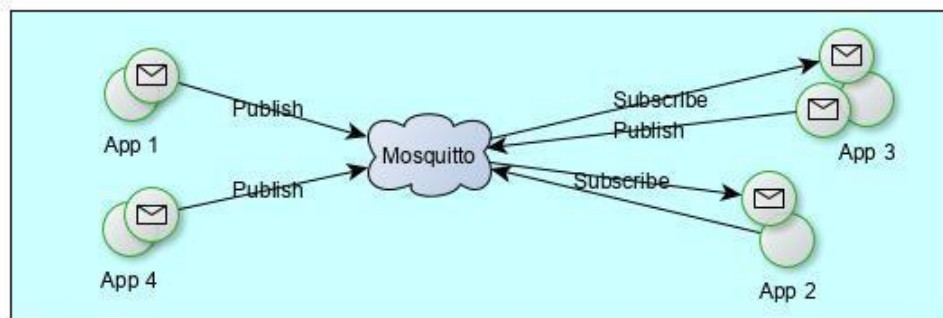
Deliver message to subscribers

This way we achieve full “loose coupling”, which means both the publishers and subscribers don’t need to know of the existence of each other. They only need to agree on the topic name and the content of the messages they exchange via the queue.

Some example use-cases:

- A temperature sensor publishes its measurements into a topic.
  - \* A database application is subscribed and stores every measurement in a database.
  - \* The air conditioning controller also is subscribed and if the temperature exceeds a given value for a few minutes, it turns on the cooling.
- A bunch of sensors all publish their measurements into the same topic.
  - \* A dashboard application is subscribed and visualizes the values in different charts.
- When someone rings your doorbell a message is published.
  - \* A Pi with a camera is subscribed and records a movie of the person at the door.
  - \* A subscribed switch powers the lights above the door.

As you can imagine based on these examples the drawing can become a bit more complicated as applications can be publisher, subscriber or both.



Mosquitto with multiple apps

# Using Mosquitto on the Pi

## Installation

To install the broker (= the Mosquitto application) we need to do three steps:

- Make sure the update list on the Pi is up-to-date
- Get and install the broker
- Let it start when the Pi boots so we are sure the broker is always running

```
1 $ sudo apt update
2 $ sudo apt install -y mosquitto mosquitto-clients
3 $ sudo systemctl enable mosquitto.service
```

Now we can check if it is installed correctly and running:

```
1 $ mosquitto -v
2 1569780732: mosquitto version 1.5.7 starting
3 1569780732: Using default config.
4 1569780732: Opening ipv4 listen socket on port 1883.
5 1569780732: Error: Address already in use
```

The error “Address already in use” apparently can be ignored...

We can also validate Mosquitto is listening to incoming messages on the default port 1883 with “netstat”:

```
1 $ netstat --listen
2 Active Internet connections (only servers)
3 Proto Recv-Q Send-Q Local Address           Foreign Address         State
4 tcp        0      0 0.0.0.0:1883             0.0.0.0:*               LISTEN
5 tcp6       0      0 [::]:1883               [::]:*                  LISTEN
```

Later we will need to know where other devices need to connect to. So we need the network IP address of the Pi on which we installed Mosquitto. We can check this with the following command:

```
1 $ hostname -I
2 192.168.0.213 2a02:1811:bc04:4900:412e:b96a:d93c:b0a2 2a02:1811:bc04:4900:11fb:315f:\
3 b8e1:f5dc
```

In this case, it’s “192.168.0.213”.

The same info can also be found with the following command which will give you more information about your network interfaces. In my case I’m using the wireless connection on “wlan”:

```

1  $ ifconfig
2  eth0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
3      ether dc:a6:32:21:c1:12 txqueuelen 1000 (Ethernet)
4      RX packets 0 bytes 0 (0.0 B)
5      RX errors 0 dropped 0 overruns 0 frame 0
6      TX packets 0 bytes 0 (0.0 B)
7      TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
8
9  lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
10     inet 127.0.0.1 netmask 255.0.0.0
11     inet6 ::1 prefixlen 128 scopeid 0x10<host>
12     loop txqueuelen 1000 (Local Loopback)
13     RX packets 0 bytes 0 (0.0 B)
14     RX errors 0 dropped 0 overruns 0 frame 0
15     TX packets 0 bytes 0 (0.0 B)
16     TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
17
18  wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
19     inet 192.168.0.213 netmask 255.255.255.0 broadcast 192.168.0.255
20     inet6 2a02:1811:bc04:4900:11fb:315f:b8e1:f5dc prefixlen 64 scopeid 0x0<glo\
21  bal>
22     inet6 2a02:1811:bc04:4900:412e:b96a:d93c:b0a2 prefixlen 128 scopeid 0x0<gl\
23  obal>
24     inet6 fe80::1f70:9d7c:a379:f0f3 prefixlen 64 scopeid 0x20<link>
25     ether dc:a6:32:21:c1:13 txqueuelen 1000 (Ethernet)
26     RX packets 59629 bytes 74393867 (70.9 MiB)
27     RX errors 0 dropped 0 overruns 0 frame 0
28     TX packets 28575 bytes 3084130 (2.9 MiB)
29     TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

## Testing Mosquitto on the Pi

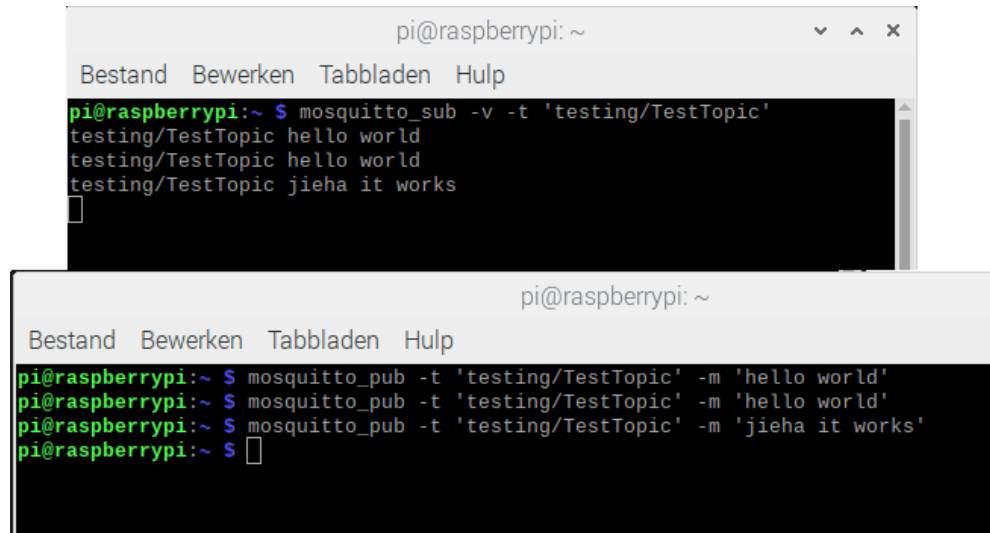
We can easily test if Mosquitto is running OK on the Pi, by opening two terminal windows. In the first one we start a listener on topic “testing/TestTopic”:

```
1  $ mosquitto_sub -v -t 'testing/TestTopic'
```

In the second terminal we send multiple commands with a message for this topic, like this:

- 1 \$ mosquitto\_pub -t 'testing/TestTopic' -m 'hello world'
- 2 \$ mosquitto\_pub -t 'testing/TestTopic' -m 'hello world'
- 3 \$ mosquitto\_pub -t 'testing/TestTopic' -m 'jieha it works'

Every “publish” from the second terminal window will appear in the first one.



```
pi@raspberrypi: ~
Bestand  Bewerken  Tabbladen  Hulp
pi@raspberrypi:~ $ mosquitto_sub -v -t 'testing/TestTopic'
testing/TestTopic hello world
testing/TestTopic hello world
testing/TestTopic jieha it works

```

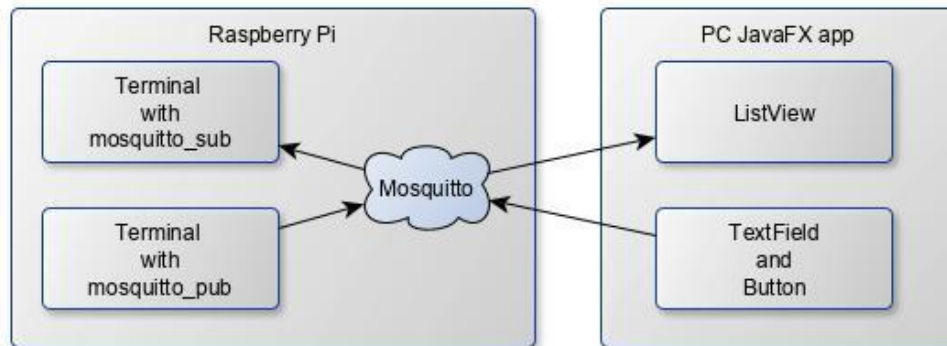
```
pi@raspberrypi: ~
Bestand  Bewerken  Tabbladen  Hulp
pi@raspberrypi:~ $ mosquitto_pub -t 'testing/TestTopic' -m 'hello world'
pi@raspberrypi:~ $ mosquitto_pub -t 'testing/TestTopic' -m 'hello world'
pi@raspberrypi:~ $ mosquitto_pub -t 'testing/TestTopic' -m 'jieha it works'
pi@raspberrypi:~ $

```

Testing Mosquitto on the Pi

## Example 1: Share data between Pi and PC

We know Mosquitto is running on the Pi and we can use the terminal to send and receive a message with the test programs. Now let's do the same from PC with a JavaFX application to make it more visually.



Publishing to and subscribing from Mosquitto from Pi and PC at the same time



Chapter\_11\_Queues > javafx-mosquitto

## Modifying the pom and module-info

Again we start from the minimal JavaFX application (Chapter 7) sources and rename the artifactId and add a dependency to [Eclipse Paho](https://www.eclipse.org/paho/)<sup>120</sup>, the library we will use for the [MQTT message protocol](http://mqtt.org/)<sup>121</sup> to talk to our Mosquitto:

```

1 <artifactId>javafx-mosquitto</artifactId>
2
3 ...
4
5 <dependency>
6   <groupId>org.eclipse.paho</groupId>
7   <artifactId>org.eclipse.paho.client.mqttv3</artifactId>
8   <version>1.2.2</version>
9 </dependency>
  
```

Paho also needs to be added to module-info.java

<sup>120</sup><https://www.eclipse.org/paho/>

<sup>121</sup><http://mqtt.org/>



```
1 module be.webtechie {
2     requires javafx.controls;
3     requires org.eclipse.paho.client.mqttv3;
4     exports be.webtechie.javafxmosquitto;
5 }
```

## Connecting and publishing to Mosquitto

To connect to Mosquitto we need to know the IP address of the Pi. There is also some exception handling required, but actually, we only need to define the MqttClient and call the connect-method.

```
1 MqttClient client = new MqttClient("tcp://" + ipAddress + ":1883",
2     MqttClient.generateClientId());
3 client.connect();
```

Once the client is connected, we can send a message (“Hello from PC”) to the topic (= mailbox, e.g. “testing/TestTopic”) with this code:

```
1 String messageText = "Hello from PC";
2 MqttMessage message = new MqttMessage();
3 message.setPayload(messageText.getBytes());
4 client.publish("testing/TestTopic", message);
```

The full code with error handling is available in the sources in “QueueClient.java”.

## Subscribing to Mosquitto

Subscribing to a topic is also done in the QueueClient script like this:

```
1 ObservableList<String> queueItems = FXCollections.observableArrayList();
2
3 client.setCallback(new ClientCallback(this.queueItems));
4 client.subscribe("testing/TestTopic");
```

We use an ObservableList so we can use this in the UI later to show the incoming messages.

As you see it uses another class ClientCallback which implements MqttCallback, that will get called each time a new message is published in “testing/TestTopic” and pushed to the subscribed applications:

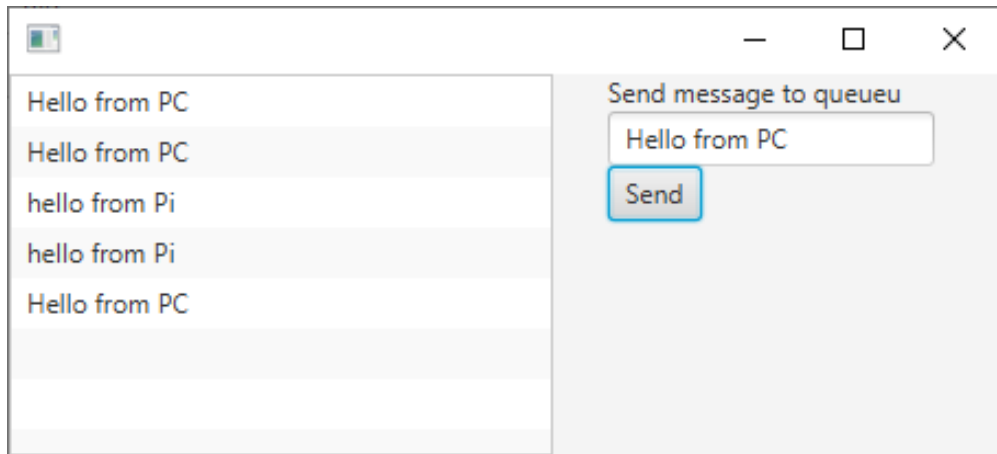
```
1 public class ClientCallback implements MqttCallback {
2     private ObservableList<String> queueItems;
3
4     public ClientCallback(ObservableList<String> queueItems) {
5         this.queueItems = queueItems;
6     }
7
8     @Override
9     public void connectionLost(Throwable throwable) {
10        System.out.println("Connection to MQTT broker lost!");
11    }
12
13    @Override
14    public void messageArrived(String s, MqttMessage mqttMessage) throws Exception {
15        String message = new String(mqttMessage.getPayload());
16
17        System.out.println("Message received:\n\t" + message);
18
19        Platform.runLater(() -> {
20            queueItems.add(message);
21        });
22    }
23
24    @Override
25    public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken) {
26        System.out.println("Delivery complete");
27    }
28 }
```

We need to use `Platform.runLater` to avoid Thread errors between the Java app running the connection, and the JavaFX thread which updates the screen.

## The user interface

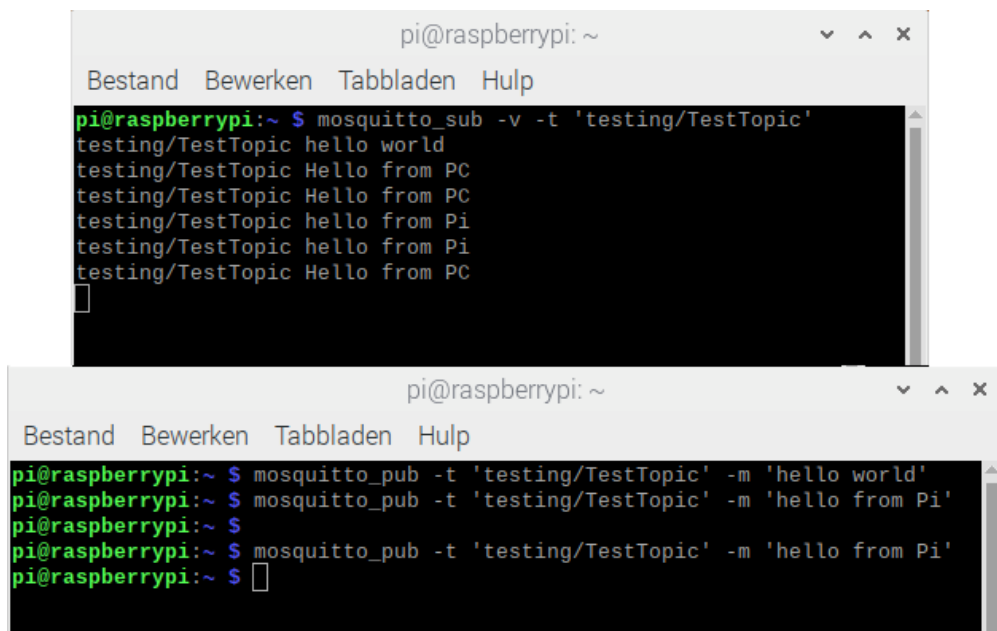
For the user interface, we are using a `ListView`, `TextInput` and `Button`.

When combining everything we get this app, e.g. running on a PC in the same network as the Pi.



Screenshot UI on PC

When we send a message to the queue, we also receive it back into the list as this application is both publisher and subscriber to the same topic in Mosquitto.

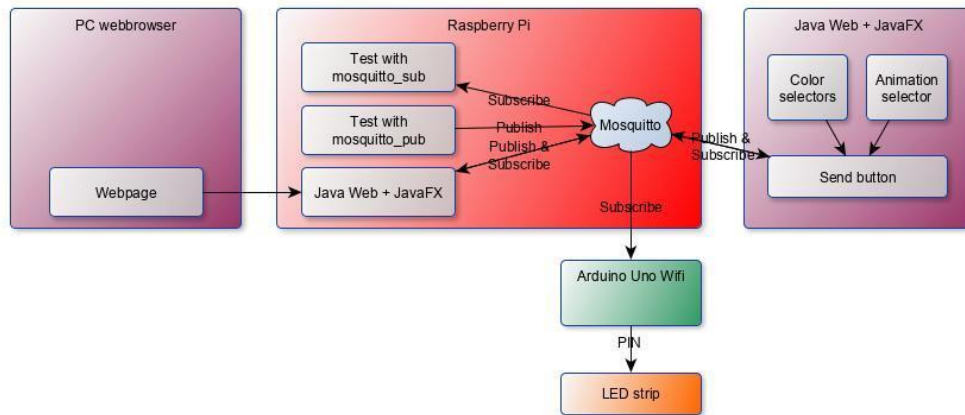


Screenshot UI on Pi

On the Pi we see the same messages appearing and we can also push messages to the topic which appear simultaneously on the PC!

## Example 2: Control Arduino from JavaFX via Mosquitto

Let's go one step further and add an Arduino to the chain to control an LED-strip. This is what we are going to build:



Controlling an LED strip on Arduino via Mosquitto

As you can see, this will be a more complex project using a lot of the knowledge we collected in the previous chapters. It includes:

- An Arduino (with WiFi) to control an LED-strip either with:
  - the full strip in one color
  - a color animation
- On the Pi we have:
  - Mosquitto broker application
  - A Java application
    - \* Webpage to select from a list of LED-effects
    - \* JavaFX UI where you can also select the color and animation
- One or more other PCs with a web browser who can load the web page
- The Java application from the Pi can also run on a PC and interact the same way with Mosquitto



The code is split into these projects  
 Chapter\_11\_Queuees > arduino-led  
 Chapter\_11\_Queuees > javafx-led-controller

This example project got a bit too big to describe all code here, so please take a look at the GitHub sources for the complete thing.

## Defining the messages

First, we need to agree on the message structure which will be exchanged via Mosquitto and the topic. We will be using a string with “:” as the separator between the different values. This is the structure which we will be using:

- The topic in Mosquitto is “ledCommand” in which new commands will be published
- The command itself is structured in the format “COMMAND\_ID:SPEED:R1:G1:B1:R2:G2:B2”
  - \* COMMAND\_ID: number of the LED effect we want, see table below
  - \* SPEED: milliseconds between effects
  - \* RGB: two sets of color values for red, green and blue between 0 and 255

So we can send two different colors in the command, but some effects don’t use colors at all, or only one and some don’t need the speed value. This is an overview of the available commands:

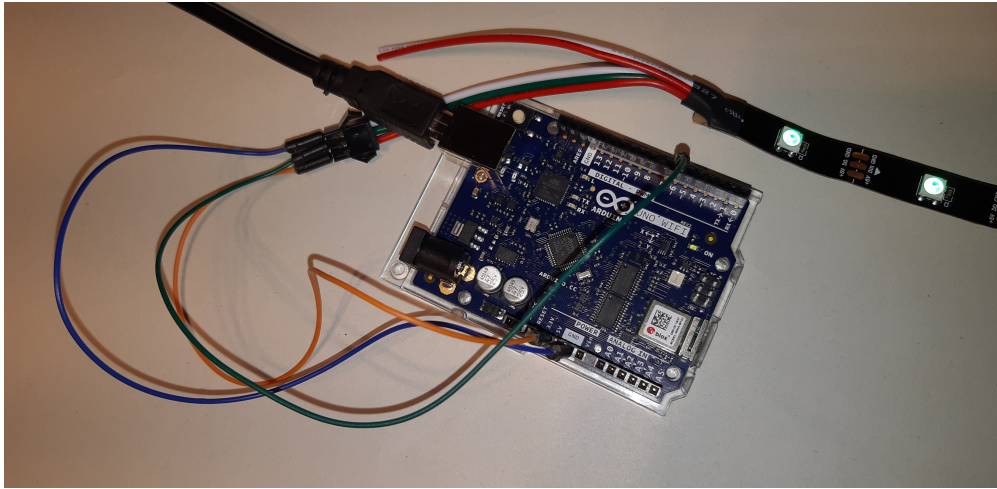
| ID | Name                                         | Uses speed | Uses RGB1 | Uses RGB2 |
|----|----------------------------------------------|------------|-----------|-----------|
| 1  | Fixed color (RGB1)                           | false      | true      | false     |
| 2  | Static fade from RGB 1 to 2                  | false      | true      | true      |
| 3  | Blinking (between color RGB1 and RGB2)       | true       | true      | true      |
| 4  | Running (RGB1 one by one, others fixed RGB2) | true       | true      | true      |
| 5  | Fading rainbow using a color gradient        | true       | false     | false     |
| 6  | Static rainbow from RGB1 to RGB2             | false      | true      | true      |
| 98 | All white                                    | false      | false     | false     |
| 99 | All out                                      | false      | false     | false     |

Some examples

- 1:20:255:0:0 = all LEDs fixed full red
- 3:2500:255:0:0:0:0:255 = change all leds from full red to full blue every 2,5 seconds
- 5:50:255:0:0:0:0:255 = all leds are blue, and one running led is red for 50 milliseconds

## The Arduino part

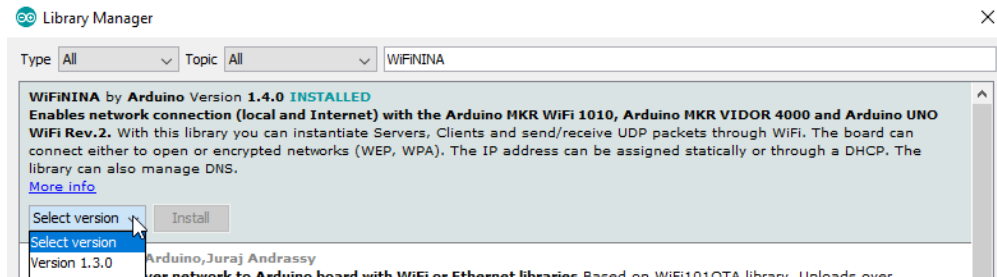
For the Arduino, we will be using an LED-strip of the type WS2812B which only uses one pin to control it, connected to pin 6. For the power, we connect to the 5V and ground pin. Make sure to not connect too many LEDs as this could require too much power (also see “Chapter 2: Tools > LED strips”). If you want to control a long strip, you will need a separate power supply. In that case, also connect the ground of the Arduino board to the ground of the power supply, or you will get surprising but unwanted LED effects because the timing of the strip and Arduino are not in sync.



Setup with the Arduino and LED strip

We will also need extra libraries to be added to the Arduino IDE. In the “Tools” menu, select “Manage libraries...” and search for these and click “Install”:

- [AdaFruit NeoPixel by Adafruit](#)<sup>122</sup>
- [WiFiNINA by Arduino](#)<sup>123</sup>
- [ArduinoMqttClient by Arduino](#)<sup>124</sup>

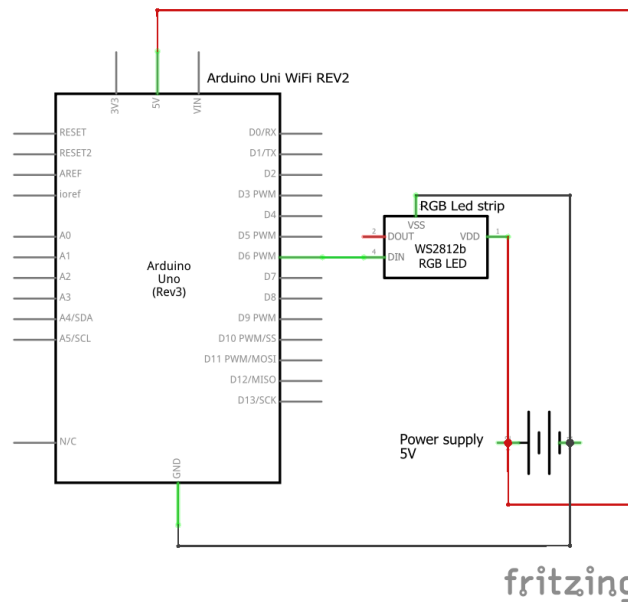


Installing the WiFiNINA library in Arduino IDE

<sup>122</sup>[https://github.com/adafruit/Adafruit\\_NeoPixel](https://github.com/adafruit/Adafruit_NeoPixel)

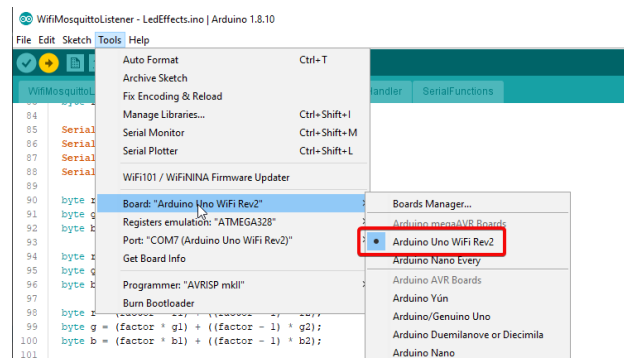
<sup>123</sup><https://github.com/arduino-libraries/WiFiNINA>

<sup>124</sup><https://github.com/arduino-libraries/ArduinoMqttClient>



Scheme with Arduino Uno WiFi REV3, led strip and power supply

The test setup for this example uses an Arduino Uno WiFi REV2. This can be a bit confusing as there are multiple Uno-types in the Arduino IDE from which you can select. Make sure to select the correct one!



Select the correct board in Arduino IDE

## Splitting up the Arduino code

To make the code of the Arduino easy to understand, the functions are split up in different “ino”-files with “WifiMosquitoListener.ino” being the main one:



Different files in an Arduino project

- WifiMosquitoListener.ino
  - \* Main file with the setup and loop function
- ConnectionHandler
  - \* Handler for incoming Mosquitto messages
- LedEffects
  - \* Code to generate the different LED animations
- MessageHandler
  - \* Converts the command which can be received from Mosquitto and Serial (for testing) into values which are used by LedEffects
- SerialFunctions
  - \* Function to check if serial data (with test command) is available

Take a look at these different files, some of them are further described here.

## Main Arduino code in setup and loop

An Arduino project has two required functions:

- setup()
  - \* Is called once when the board starts or resets.
  - \* Is used to initialize libraries, variables, pins...
- loop()
  - \* Loops continuously to actively control the board.

Let's take a look at how these two are used in our project. The actual code you find in the GitHub sources contains even more Serial debug info. At the end of the setup, we already provide a message so the LED strip starts with effect 2 (static fade).



```
1 void setup() {
2   // Configure serial speed and wait till it is available
3   // This is used to output logging info and can receive LED commands
4   Serial.begin(9600);
5   while (!Serial) {
6     ; // wait for serial port to connect. Needed for native USB port only
7   }
8
9   // Initialize the leds
10  initLeds();
11
12  // Connecting to WiFi
13  Serial.println("--- Connecting to WiFi ---");
14  while (WiFi.begin(ssid, pass) != WL_CONNECTED) {
15    // failed, retry
16    Serial.print(".");
17    delay(5000);
18  }
19
20  // Connecting to Mosquitto
21  Serial.println("--- Connecting to Mosquitto ---");
22  mqttClient.onMessage(onMqttMessage);
23  if (!mqttClient.connect(broker, port)) {
24    Serial.print("MQTT connection failed! Error code = ");
25    Serial.println(mqttClient.connectError());
26  } else {
27    mqttClient.subscribe(topic);
28    Serial.println("MQTT connection ready");
29  }
30
31  // Set the initial LED effect
32  String message = "2:0:14:255:0:255:5:0";
33  message.toCharArray(input, 50);
34 }
```

Once everything is configured and ready to use, the loop function will continuously check if new messages are available in Mosquitto or Serial, convert the message to variables and call the correct LED function.

```
1 void loop() {
2   mqttClient.poll();
3   checkSerial();
4   handleMessage();
5
6   currentLoop++;
7
8   // Only do LED effect when loop exceeds the defined animationSpeed
9   if (currentLoop >= animationSpeed) {
10    // Depending on the commandId, call the correct LED effect
11    if (commandId == 1) {
12      setStaticColor();
13    } else if (commandId == 2) {
14      setStaticFade();
15    } else if (commandId == 3) {
16      setBlinking();
17    } else if (commandId == 4) {
18      // ... call the correct function for each commandId
19    }
20
21    currentLoop = 0;
22  }
23
24  delay(5);
25 }
```

## Converting the command string to values

The incoming (string) message in Arduino needs to be parsed to numeric values. So this will for instance parse “2:25:255:0:0:0:0:250” to

- command id = 2
- speed = 25
- rgb1 = 0xFF0000
- rgb2 = 0x0000FF

This is done in MessageHandler.ino and the “magic” is in splitting the input on “:” and assign each part to a value:

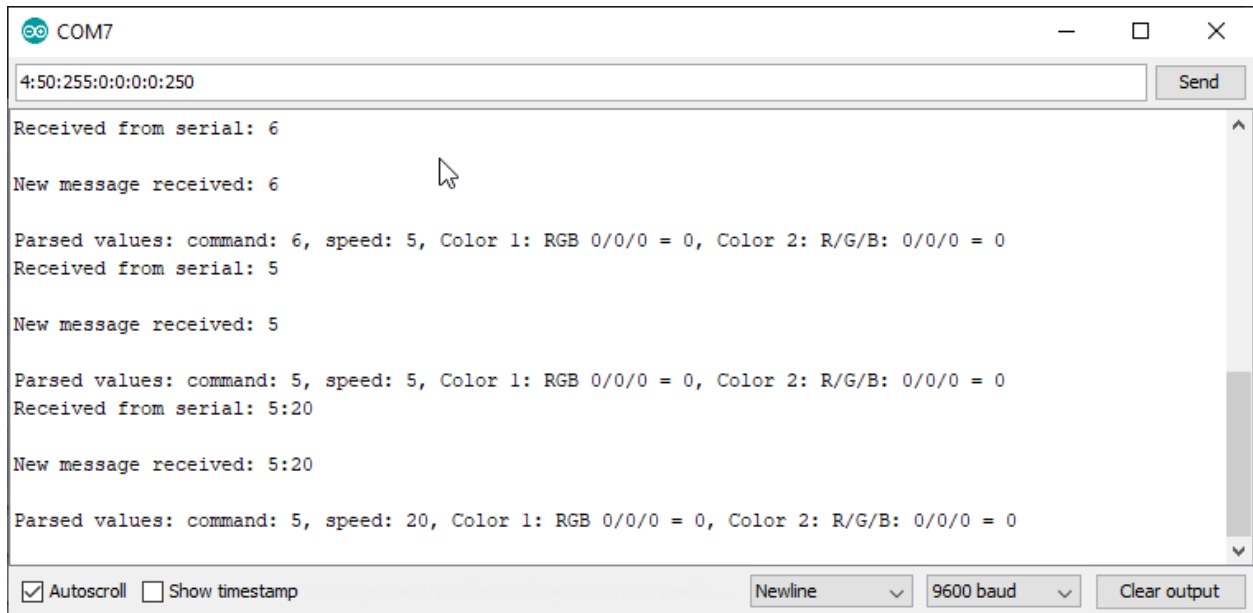
```
1 void handleMessage() {
2   char* part = strtok(input, ":");
3   int position = 0;
4
5   while (part != 0) {
6     int value = atoi(part);
7
8     if (position == 0) {
9       commandId = value;
10    } else if (position == 1) {
11      animationSpeed = value;
12    } else if (position == 2) {
13      r1 = value;
14    } else if (position == 3) {
15      // ... handle each part and assign to a value
16    }
17
18    position++;
19  }
20 }
```

## Receiving commands via MQTT and Serial

Mosquitto messages are received with a callback function which was configured in “setup()” - `mqttClient.onMessage(onMqttMessage);` - and we put the received message in the “input” variable which is later processed to values:

```
1 void onMqttMessage(int messageSize) {
2   while (mqttClient.available()) {
3     mqttClient.read(input, sizeof(input));
4     input[50] = 0;
5   }
6 }
```

For easy testing, we can also send the same commands via “Tools” > “Serial Monitor”. And as you can see in the screenshot, for the LED effects which don’t use speed or colors, we don’t need to send the full string. E.g. “6” starts the static rainbow.



Send LED command via the serial interface of the Arduino IDE

Serial messages are handled in a function which is called from “loop()”:

```

1 void checkSerial() {
2   if (Serial.available() > 0) {
3     String message = Serial.readString();
4
5     message.toCharArray(input, 50);
6     input[50] = 0;
7   }
8 }

```

In both functions, we make sure the last byte is 0. This is used later to make sure we don’t end up in an endless loop where the input is handled.

## LED effects

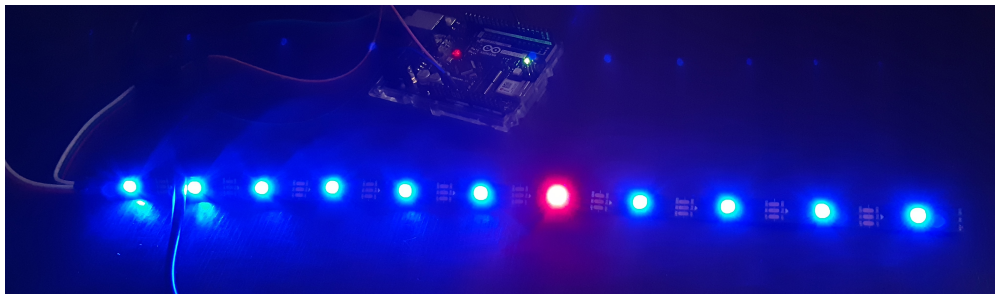
Only your imagination and programming skills are the limiting factor on what you can achieve... ;-) In “LedEffects.ino” the functions are included for the effects defined in the table before, but feel free to go crazy and extend these!

Just some examples. First, the LED strip must be initialized, this function is called from “main” > “setup()”:

```
1 void initLeds() {
2   strip.begin();
3   strip.setBrightness(100);
4   strip.clear();
5   strip.show(); // Initialize all pixels to 'off'
6 }
```

For a running LED, we need to put one LED on color 2 and show it. The value for this LED is then immediately put back to color 1, without showing it. That's what we do the next time the same function is called, and the next LED is put to color 2.

```
1 void setRunningLight() {
2   if (currentAction >= NUMBER_OF_LEDS) {
3     currentAction = 0;
4   }
5
6   // Show color 1
7   strip.setPixelColor(currentAction, rgb1);
8   strip.show();
9
10  // Reset to color 2 for next loop
11  strip.setPixelColor(currentAction, rgb2);
12
13  currentAction++;
14 }
```



Running light with red and blue

There are also two helper functions to calculate the gradient color and a rainbow color:

```
1 // Calculate a gradient color between color 1 and 2, for the given position
2 uint32_t getGradientColor(uint16_t pos) {
3     ...
4 }
5
6 // Pos from 0 to 255 to get colors from full-color wheel
7 // 0 - 85:    G - R
8 // 85 - 170:  R - B
9 // 170 - 255: B - G
10 uint32_t getWheelColor(byte pos) {
11     ...
12 }
```



Two color gradient from green to red

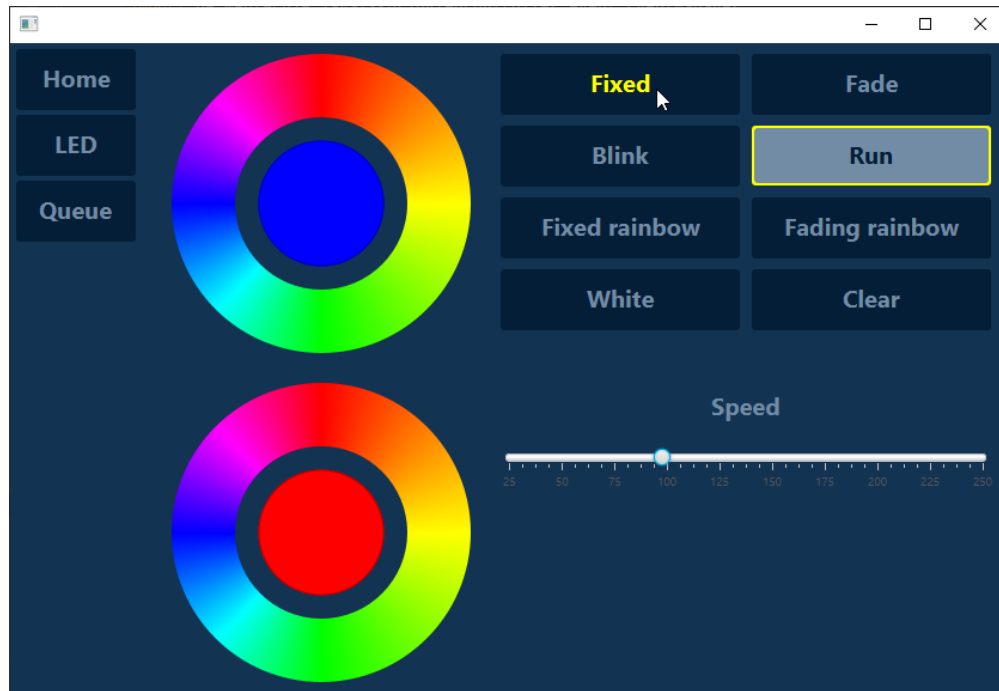
## The Java application

We are going to build a controller application with JavaFX and the Undertow webserver. By doing this, we can run the same application on both Pi (as a webserver so we can change the LEDs from any PC via the browser) and PC (for testing and just because we can...).

We use two libraries to have the perfect use-case for some of the required functionality:

- Undertow
  - \* Webserver in code, so no extra installation needed
  - \* Easy to add custom web functions
- JavaFX
  - \* Make a UI to select the color and animation
  - \* Visualize the active colors and animation on the Arduino

This is the user interface to control our LED-strip(s):



LED controller UI

## EventManager

Within the application, we add an EventManager. This will allow us to have two different UI-parts who listen to the incoming LED-commands.

```

1 public class EventManager {
2     /**
3      * The list with components to be notified of a new LedCommand
4      * received from Mosquitto.
5      */
6     private List<EventListener> eventListeners = new ArrayList<>();
7
8     /**
9      * Used by every component which wants to be notified of new events.
10    */
11    public void addListener(EventListener eventListener) {
12        this.eventListeners.add(eventListener);
13    }
14
15    /**
16     * Used by Mosquitto callback to forward a received message to all
17     * components in the application who were added as a listener.

```

```
18  *
19  * @param ledCommand {@link LedCommand}
20  */
21  public void sendEvent(LedCommand ledCommand) {
22      this.eventListeners.forEach(1 -> 1.onQueueMessage(ledCommand));
23  }
24  }
```

Each component we want to be an EventListener has to implement the interface:

```
1  public interface EventListener {
2      /**
3       * Whenever a new {@link LedCommand} is received from Mosquitto,
4       * all listeners will be notified so they can handle it for their own use.
5       *
6       * @param ledCommand {@link LedCommand}
7       */
8       void onQueueMessage(LedCommand ledCommand);
9  }
```

## Mosquitto connection

We reuse the code from the first example in this chapter. But instead of an ObservableList to store the incoming messages, we will use the EventManager to forward the message to the different components of the application which were added to the list of listeners. This is done when the LED panel is initialized:

```
1  LedControlPanel ledControlPanel = new LedControlPanel(queueClient);
2  eventManager.addListener(ledControlPanel);
3  this.led = new Group(ledControlPanel);
```

## LED effect selection UI

This UI will change the color wheels and speed values according to the received command. And of course, changing the color and pressing a button will send a command to Mosquitto so it will be received by the Arduino. It is build up with a set of JavaFX components with some additional CSS styling. This is the styling used for the LED-effect selection buttons:



```

1  .ledButton {
2    -fx-font: 20px "Sans";
3    -fx-padding: 10;
4    -fx-background-color: #041E37;
5    -fx-text-fill: #728CA6;
6    -fx-font-weight: bold;
7    -fx-min-width: 200;
8    -fx-min-height: 30;
9  }
10
11 .ledButton:hover {
12   -fx-text-fill: yellow;
13 }
14
15 .ledButton:selected {
16   -fx-text-fill: #041E37;
17   -fx-background-color: yellow, #728CA6;
18   -fx-background-insets: 0 0 0 0, 2 2 2 2;
19 }

```

As you can see in “.ledButton:selected”, JavaFX styling allows you to use multiple colors for the background. By adding different insets you can create borders and shadows.



LED controller UI

By changing this one CSS styling you can have even more combined colors in the background:

```

1  .ledButton:selected {
2    -fx-text-fill: #041E37;
3    -fx-background-color: white, yellow, red;
4    -fx-background-insets: 0 -6 -6 0, 0 0 0 0, 2 -2 -2 2;
5  }

```



LED controller UI

The color selection wheels are made by Gerrit Grunwald (see “Chapter 7: JavaFX”). As we want this component to act when a new message is received from Mosquitto, it needs to implement “EventListener”:

```
1 public class LedControlPanel extends HBox implements EventListener {
2     ...
3 }
```

An extract of the code for this UI below. The buttons and speed sliders are put in GridPane for a nice structure.

```
1 public LedControlPanel(QueueClient queueClient) {
2     this.queueClient = queueClient;
3
4     this.setSpacing(25);
5
6     VBox colorSelectors = new VBox();
7     colorSelectors.setSpacing(25);
8     this.getChildren().add(colorSelectors);
9
10    this.colorSelector1 = new ColorSelector();
11    this.colorSelector1.setPrefSize(250, 250);
12    this.colorSelector1.selectedColorProperty().addListener(e -> this.sendMessage());
13    this.colorSelector1.setSelectedColor(Color.BLUE);
14    colorSelectors.getChildren().add(this.colorSelector1);
15
16    ...
17
18    GridPane effectButtons = new GridPane();
19    effectButtons.setHgap(10);
20    effectButtons.setVgap(10);
21    this.getChildren().add(effectButtons);
22
23    this.btStatic = new Button("Fixed");
24    this.btStatic.getStyleClass().add("ledButton");
25    this.btStatic.setOnAction(e -> this.setEffect(LedEffect.STATIC));
26    effectButtons.add(this.btStatic, 0, 0);
27
28    this.btStaticFade = new Button("Fade");
29    this.btStaticFade.getStyleClass().add("ledButton");
30    this.btStaticFade.setOnAction(e -> this.setEffect(LedEffect.STATIC_FADE));
31    effectButtons.add(this.btStaticFade, 1, 0);
32
33    ...
34
35    Label lblSpeed = new Label("Speed");
36    lblSpeed.getStyleClass().add("ledSpeed");
```

```

37  GridPane.setHalignment(lblSpeed, HPos.CENTER);
38  effectButtons.add(lblSpeed, 0, 5, 2, 1);
39
40  this.slider = new Slider();
41  this.slider.setShowTickLabels(true);
42  this.slider.setShowTickMarks(true);
43  this.slider.valueProperty().addListener((o, old, new) -> this.sendMessage());
44  effectButtons.add(this.slider, 0, 6, 2, 1);
45
46  this.setEffect(LedEffect.ALL_OUT);
47 }

```

When a message is received from Mosquitto (via the `onQueueMessage` from the `EventListener` interface), the color wheels and slider are changed:

```

1  /**
2   * {@link LedCommand} received from Mosquitto.
3   * We block sending updates back to Mosquitto until the interface is updated fully
4   * to match the received command, to avoid infinite loops.
5   */
6  @Override
7  public void onQueueMessage(LedCommand ledCommand) {
8      this.blockSending = true;
9
10     this.setEffect(ledCommand.getLedEffect());
11     this.slider.setValue(ledCommand.getSpeed());
12     this.colorSelector1.setSelectedColor(ledCommand.getColor1());
13     this.colorSelector2.setSelectedColor(ledCommand.getColor2());
14
15     this.blockSending = false;
16 }

```

Depending on the selected `LedEffect` the color wheels are enabled or disabled, on the slider values updated:

```

1  /**
2  * Handle the chosen effect from a button or a Mosquitto message
3  * to enable/disable the available UI elements and
4  * highlight the button of the selected {@link LedEffect}.
5  */
6  private void setEffect(LedEffect ledEffect) {
7      this.selectedLedEffect = ledEffect;
8
9      this.colorSelector1.setDisable(!ledEffect.useColor1());
10     this.colorSelector2.setDisable(!ledEffect.useColor2());
11     this.slider.setDisable(!ledEffect.useSpeed());
12     this.slider.setMin(ledEffect.getMinimumSpeed());
13     this.slider.setMax(ledEffect.getMaximumSpeed());
14
15     this.btStatic.setSelected(ledEffect == LedEffect.STATIC);
16     this.btStaticFade.setSelected(ledEffect == LedEffect.STATIC_FADE);
17     this.btBlinking.setSelected(ledEffect == LedEffect.BLINKING);
18     this.btRunning.setSelected(ledEffect == LedEffect.RUNNING);
19     this.btStaticRainbow.setSelected(ledEffect == LedEffect.STATIC_RAINBOW);
20     this.btFadingRainbow.setSelected(ledEffect == LedEffect.FADING_RAINBOW);
21     this.btWhite.setSelected(ledEffect == LedEffect.ALL_WHITE);
22     this.btClear.setSelected(ledEffect == LedEffect.ALL_OUT);
23
24     this.sendMessage();
25 }

```

When a button is clicked, or color changed, a new command is published to Mosquitto:

```

1  /**
2  * Send a message to Mosquitto if a new effect and/or different parameters
3  * are selected.
4  */
5  private void sendMessage() {
6      if (this.blockSending) {
7          // Avoid sending the same command to Mosquitto again
8          // it to avoid infinite loops.
9          return;
10     }
11
12     LedCommand ledCommand = new LedCommand(
13         this.selectedLedEffect,
14         (int) this.slider.getValue(),
15         this.colorSelector1.getSelectedColor(),

```

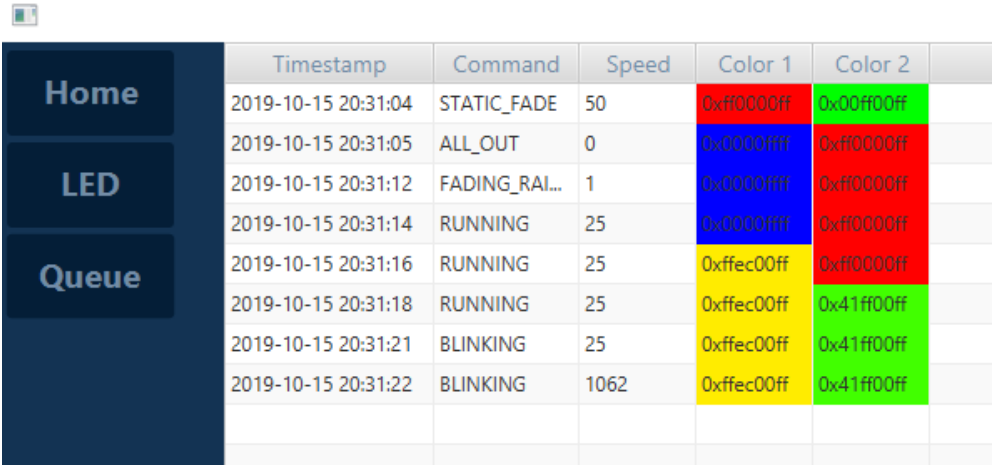
```

16         this.colorSelector2.getSelectedColor()
17     );
18
19     if (this.queueClient != null) {
20         this.queueClient.sendMessage(ledCommand);
21     }
22 }

```

## Queue log UI

As an example of a second `EventListener`, there is a “`QueueMessagesList`” UI which shows a table with the history of all received commands.



The screenshot shows a mobile application interface with a dark blue sidebar on the left containing three menu items: "Home", "LED", and "Queue". The "Queue" item is selected. To the right of the sidebar is a table displaying a log of received commands. The table has six columns: "Timestamp", "Command", "Speed", "Color 1", "Color 2", and an empty column. The data rows are as follows:

| Timestamp           | Command       | Speed | Color 1    | Color 2    |  |
|---------------------|---------------|-------|------------|------------|--|
| 2019-10-15 20:31:04 | STATIC_FADE   | 50    | 0xff0000ff | 0x00ff00ff |  |
| 2019-10-15 20:31:05 | ALL_OUT       | 0     | 0x0000ffff | 0xff0000ff |  |
| 2019-10-15 20:31:12 | FADING_RAI... | 1     | 0x0000ffff | 0xff0000ff |  |
| 2019-10-15 20:31:14 | RUNNING       | 25    | 0x0000ffff | 0xff0000ff |  |
| 2019-10-15 20:31:16 | RUNNING       | 25    | 0xffec00ff | 0xff0000ff |  |
| 2019-10-15 20:31:18 | RUNNING       | 25    | 0xffec00ff | 0x41ff00ff |  |
| 2019-10-15 20:31:21 | BLINKING      | 25    | 0xffec00ff | 0x41ff00ff |  |
| 2019-10-15 20:31:22 | BLINKING      | 1062  | 0xffec00ff | 0x41ff00ff |  |

Queue log UI

This class also implements the `EventListener` interface and can store all the received messages in an `ObservableList`

```

1 public class QueueMessagesList extends TableView implements EventListener {
2     private ObservableList<ReceivedMessage> list =
3         FXCollections.observableArrayList();
4
5     /**
6      * {@link LedCommand} received from Mosquitto.
7      * We put it on top of our internal list so it gets added to the table.
8      *
9      * @param ledCommand The {@link LedCommand}
10     */
11     @Override
12     public void onQueueMessage(LedCommand ledCommand) {
13         this.list.add(0, new ReceivedMessage(ledCommand));

```

```

14 }
15 }

```

ReceivedMessage is a just a data class which holds a timestamp and the LedCommand:

```

1  /**
2  * Helper class to add a {@link LedCommand} to a table with a timestamp.
3  */
4  public class ReceivedMessage {
5      private final String timestamp;
6      private final LedCommand ledCommand;
7
8      private final DateTimeFormatter dateFormat =
9          DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
10
11     public ReceivedMessage(LedCommand ledCommand) {
12         this.timestamp = LocalDateTime.now().format(dateFormat);
13         this.ledCommand = ledCommand;
14     }
15
16     public String getTimestamp() {
17         return timestamp;
18     }
19
20     public LedCommand getLedCommand() {
21         return ledCommand;
22     }
23 }

```

The table is initialized within the class constructor. Each column has its own “setCellFactory”, which can, for instance, change the background color of the cell to match the color of the LedCommand value.

```

1  public QueueMessagesList() {
2      TableColumn colTimestamp = new TableColumn("Timestamp");
3      colTimestamp.setStyle("-fx-alignment: TOP-LEFT;");
4      colTimestamp.setMinWidth(150);
5      colTimestamp.setCellValueFactory(new PropertyValueFactory<>("timestamp"));
6
7      TableColumn colCommand = new TableColumn("Command");
8      colCommand.setStyle("-fx-alignment: TOP-LEFT;");
9      colCommand.setMinWidth(100);

```

```

10 colCommand.setCellValueFactory(new PropertyValueFactory<>("ledCommand"));
11 colCommand.setCellFactory(column -> new TableCell<LedCommand, LedCommand>() {
12     @Override
13     protected void updateItem(LedCommand item, boolean empty) {
14         super.updateItem(item, empty);
15
16         if (item == null || empty) {
17             setText(null);
18             setStyle("");
19         } else {
20             setText(item.getLedEffect().name());
21         }
22     }
23 });
24
25 ...
26
27 TableColumn colColor1 = new TableColumn("Color 1");
28 colColor1.setStyle("-fx-alignment: TOP-CENTER;");
29 colColor1.setMinWidth(70);
30 colColor1.setCellValueFactory(new PropertyValueFactory<>("ledCommand"));
31 colColor1.setCellFactory(column -> new TableCell<LedCommand, LedCommand>() {
32     @Override
33     protected void updateItem(LedCommand item, boolean empty) {
34         super.updateItem(item, empty);
35
36         if (item == null || empty) {
37             setText(null);
38             setStyle("");
39         } else {
40             setText(item.getColor1().toString());
41             setStyle("-fx-background-color: " + item.getColor1().toString()
42                 .replace("0x", "#"));
43         }
44     }
45 });
46
47 ...
48
49 this.getColumns().addAll(
50     colTimestamp,
51     colCommand,
52     colSpeed,

```

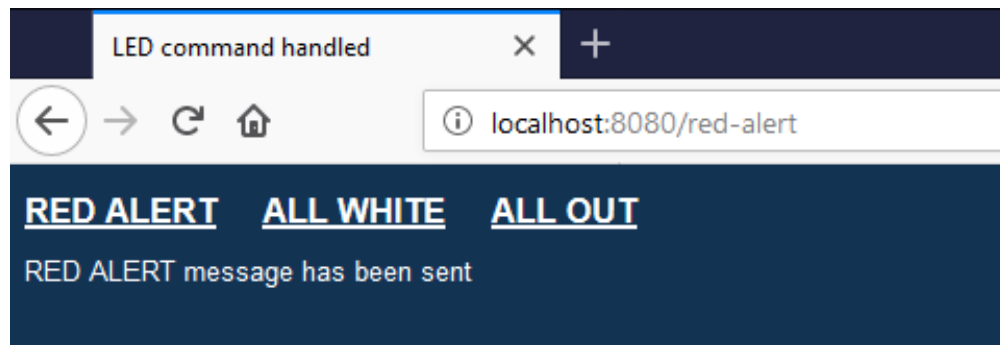
```

53     colColor1,
54     colColor2);
55
56     this.setItems(this.queueItems);
57 }

```

## The web interface

To be able to change the LED strip from any PC within the same network as the Pi, we want to use a webpage.



Web interface LED controller application

To achieve this, we need to add [Undertow<sup>125</sup>](http://undertow.io/). This is a light-weight but very flexible Java-webserver that you can fully integrate into an application. So let's add it as a dependency in our pom.xml file:

```

1 <dependency>
2   <groupId>io.undertow</groupId>
3   <artifactId>undertow-core</artifactId>
4   <version>2.0.26.FINAL</version>
5 </dependency>
6 <dependency>
7   <groupId>io.undertow</groupId>
8   <artifactId>undertow-servlet</artifactId>
9   <version>2.0.26.FINAL</version>
10 </dependency>

```

In the main start-function, we initialize and configure the server:

<sup>125</sup><http://undertow.io/>



```
1 Undertow server = Undertow.builder()
2   .addHttpListener(8080, "IP_ADDRESS_OF_YOUR_PI")
3   .setHandler(new WebHandler(queueClient))
4   .build();
5 server.start();
```

And the handler is something we need to implement as a class that will return a webpage, a REST-handler or whatever we want to provide. In this case, we are using a webpage where you select a few predefined LED commands. The code is very simple as we only provide three options and some checks. So you get the minimal idea but can extend as much as you want...

```
1 public class WebHandler implements HttpHandler {
2   final QueueClient queueClient;
3
4   public WebHandler(QueueClient queueClient) {
5     this.queueClient = queueClient;
6   }
7
8   /**
9    * Whenever a request is received by Undertow webserver,
10   * this handler will be called.
11   */
12   @Override
13   public void handleRequest(HttpServerExchange exchange) throws Exception {
14     String path = exchange.getRequestPath();
15
16     if (this.queueClient == null) {
17       this.returnError(exchange, StatusCodes.INTERNAL_SERVER_ERROR,
18         "Sorry, Message Queue Client is not available!"
19         + " Can not send your request...");
20     } else if (path.equals("/red-alert")) {
21       this.queueClient.sendMessage(
22         new LedCommand(LedEffect.BLINKING, 1000, Color.RED, Color.WHITE)
23       );
24       this.returnSuccess(exchange, "RED ALERT message has been sent");
25     } else if (path.equals("/all-white")) {
26       this.queueClient.sendMessage(new LedCommand(LedEffect.ALL_WHITE));
27       this.returnSuccess(exchange, "ALL WHITE message has been sent");
28     } else if (path.equals("/all-out")) {
29       this.queueClient.sendMessage(new LedCommand(LedEffect.ALL_OUT));
30       this.returnSuccess(exchange, "ALL OUT message has been sent");
31     } else {
32       this.returnError(exchange, StatusCodes.NOT_FOUND,
```

```
33         "The requested path is not available");
34     }
35 }
36
37 /**
38  * Return a success page.
39  */
40 private void returnSuccess(HttpServerExchange exchange, String message) {
41     this.returnPage(exchange, StatusCodes.OK, "LED command handled", message);
42 }
43
44 /**
45  * Return an error page.
46  */
47 private void returnError(HttpServerExchange exchange, int statusCode,
48     String message) {
49     this.returnPage(exchange, statusCode, "Error", message);
50 }
51
52 /**
53  * Create and return the page.
54  */
55 private void returnPage(HttpServerExchange exchange, int statusCode,
56     String title, String content) {
57     StringBuilder sb = new StringBuilder();
58
59     sb.append("<!DOCTYPE html>\n")
60         .append("<html>\n")
61         .append("    <head>\n")
62         .append("        <title>").append(title).append("</title>\n")
63         .append("        <style>\n")
64         .append("            // CSS styling")
65         .append("        </style>\n")
66         .append("    </head>\n")
67         .append("    <body>\n")
68         .append("        <ul>\n")
69         .append("            <li><a href='/red-alert'>RED ALERT</a></li>\n")
70         .append("            <li><a href='/all-white'>ALL WHITE</a></li>\n")
71         .append("            <li><a href='/all-out'>ALL OUT</a></li>\n")
72         .append("        </ul>\n")
73         .append("    ").append(content).append("\n")
74         .append("    </body>\n ")
75         .append("</html>");
```

```
76     exchange.getResponseHeaders().put(Headers.CONTENT_LENGTH, sb.toString()
77         .length());
78     exchange.getResponseHeaders().put(Headers.CONTENT_TYPE, "text/html");
79     Sender sender = exchange.getResponseSender();
80     exchange.setStatusCode(statusCode);
81     sender.send(sb.toString());
82 }
83 }
```

## The finished setup

When everything is combined you can use both the JavaFX UI and the web interface to control the LED strip with commands exchanged via Mosquitto on the Pi. When a command is selected on the web interface, the JavaFX UI shows the selected state.



JavaFX on PC, terminal on Pi with Mosquitto and Arduino with LED strip

### Tip: Checking the network packages between Arduino and Pi

If you don't manage to control the Arduino and are not sure if the connection with the Pi is working, you can use Wireshark to check the network packages. This application shows you the network packages which are sent and received on a network interface.

You need a few extra steps to configure some permissions for this application:

```

1 $ sudo apt-get install wireshark
2 $ sudo groupadd wireshark
3 $ sudo groupadd wireshark
4 $ sudo usermod -a -G wireshark pi
5 $ sudo chgrp wireshark /usr/bin/dumpcap
6 $ sudo chmod 750 /usr/bin/dumpcap
7 $ sudo setcap cap_net_raw,cap_net_admin=eip /usr/bin/dumpcap

```

After this, log-off or restart the Pi and now you can start Wireshark, select your cable or WiFi connection and watch the messages being transmitted over the network.



In the sources, you can find a Wireshark trace file with the communication between my Pi and Arduino test setup.

Chapter\_11\_Queuees > wireshark-traces > pi-receives-mosquitto-arduino-client.pcapng

With the filter “ip.addr == 192.168.0.213” we only get the messages to and from the Arduino in my test setup. In packet 13 you can see an LED-command message which was successfully sent from Pi (192.168.0.213) to Arduino (192.168.0.128). In the details of the packet (middle window) and the bytes overview (bottom window) you can find back the full command “4:2:255:22:0:0:1:255”.

The screenshot displays the Wireshark interface for the file 'pi-receives-mosquitto-arduino-client.pcapng'. The filter 'ip.addr == 192.168.0.213' is applied. The packet list shows several MQTT and TCP packets. Packet 13 is highlighted, showing an MQTT Publish Message from 192.168.0.213 to 192.168.0.128. The details pane for this packet shows the message type as Publish Message with a topic of 'ledCommand' and a message of '4:2:255:22:0:0:1:255'. The packet bytes pane shows the raw hex and ASCII representation of the message.

| No. | Time         | Source        | Destination   | Protocol | Length | Info                                            |
|-----|--------------|---------------|---------------|----------|--------|-------------------------------------------------|
| 4   | 0.006799720  | 192.168.0.128 | 192.168.0.213 | MQTT     | 84     | Connect Command                                 |
| 5   | 0.006899793  | 192.168.0.213 | 192.168.0.128 | TCP      | 54     | 1883 → 65488 [ACK] Seq=1 Ack=31 Win=29200 Len=0 |
| 6   | 0.007027772  | 192.168.0.213 | 192.168.0.128 | MQTT     | 58     | Connect Ack                                     |
| 7   | 0.012277667  | 192.168.0.128 | 192.168.0.213 | MQTT     | 71     | Subscribe Request (id=1) [ledCommand]           |
| 8   | 0.012485459  | 192.168.0.213 | 192.168.0.128 | MQTT     | 59     | Subscribe Ack (id=1)                            |
| 9   | 0.028976982  | 192.168.0.128 | 192.168.0.213 | TCP      | 54     | 65488 → 1883 [ACK] Seq=48 Ack=10 Win=5735 Len=0 |
| 10  | 17.63358815  | 192.168.0.135 | 192.168.0.213 | MQTT     | 90     | Publish Message (id=33) [ledCommand]            |
| 11  | 17.633769793 | 192.168.0.213 | 192.168.0.135 | MQTT     | 58     | Publish Ack (id=33)                             |
| 12  | 17.633975511 | 192.168.0.213 | 192.168.0.128 | MQTT     | 88     | Publish Message [ledCommand]                    |
| 13  | 17.634029085 | 192.168.0.213 | 192.168.0.128 | MQTT     | 88     | Publish Message [ledCommand]                    |
| 14  | 17.678032744 | 192.168.0.135 | 192.168.0.213 | TCP      | 54     | 57434 → 1883 [ACK] Seq=37 Ack=5 Win=512 Len=0   |

Details for Packet 13:

- Frame 13: 88 bytes on wire (704 bits), 88 bytes captured (704 bits) on interface 0
- Ethernet II, Src: dc:a6:32:21:c1:13 (dc:a6:32:21:c1:13), Dst: Espressi\_ac:d8:9c (24:0a:c4:ac:d8:9c)
- Internet Protocol Version 4, Src: 192.168.0.213, Dst: 192.168.0.128
- Transmission Control Protocol, Src Port: 1883, Dst Port: 65488, Seq: 10, Ack: 48, Len: 34
- MQ Telemetry Transport Protocol, Publish Message
  - Header Flags: 0x30, Message Type: Publish Message, QoS Level: At most once delivery (Fire and Forget)
  - Msg Len: 32
  - Topic Length: 10
  - Topic: ledCommand
  - Message: 4:2:255:22:0:0:1:255

Packet Bytes:

```

0000 24 0a c4 ac d8 9c dc a6 32 21 c1 13 08 00 45 00  $.....2!....E-
0010 00 4a b9 f1 40 00 40 06 fe 16 c0 a8 00 d5 c0 a8  .J..@. ....
0020 00 80 07 5b ff d0 8b ba 21 a2 00 00 19 9d 50 18  ...[.....!.....P-
0030 72 10 82 b9 00 00 30 20 00 0a 6c 65 64 43 6f 6d  r.....0 ..ledCom
0040 6d 61 6e 64 34 3a 32 3a 32 35 35 3a 32 32 3a 30  mand4:2: 255:22:0
0050 3a 30 3a 31 3a 32 35 35  :0:1:255

```

### Wireshark MQTT packet

So at this moment, we are sure the commands are exchanged between Pi and Arduino.

# Conclusion

Wow, what a journey... What started as some blog posts and a 4-page article for MagPi, turned into this book!

Building the examples and understanding what needed to be done on the hardware level has been pretty challenging sometimes. But in the end, all the examples I had in mind when starting this book and creating the first draft of the table of contents, are here. Especially thanks to all those people who volunteered to review and give feedback. I'm also honored the experts I admire, were willing to spend some of their precious time for an interview!

**Do I still think Java on Raspberry Pi is a good idea? You bet!!!** More than ever, I learned Java is the tool I love to create stuff, experiment and build. Is it the one-and-only-tool-for-everything? Of course not! As always you have to choose wisely but I hope you can find enough approaches in this book to combine the power of Java with the tools you love and know...

**JavaFX is a no-brainer when you need a tool to create user interfaces** and you can create visually appealing screens with very little effort. But Arduino is the best fit to control, for example, LED-strips. The GPIOs are wonderful to attach any kind of hardware and easy to control as shown in "Chapter 9: Pi4J", but sometimes it's easier to find an example Python script for more complex components. But you can start these scripts from Java with parameters as described in "Chapter 8: Bits and Bytes".

As with any good software project **you have to select the tools that fit your needs in the best possible way**. And most of the times it will be a combination of different things. But anyhow there is a big chance **the Raspberry Pi** will be the centerpiece of it. This small and cheap computer **has proven once more it's a "game-changer"!**

Now let me answer the question I asked all interviewees in this book...

*Which DIY-programming-electronics-project are you working on, or is on your "if I ever have time" list?*

The project I had in mind when starting my Java-on-Raspberry-Pi-experiments is a combination of some of the examples in this book. For my son, I've built a drum booth some time ago so he can continue his love for playing music without all the noise for my wife and myself. In that booth, I want to add a touch-screen with a user interface to control LED-strips on the ceiling attached to an Arduino, different types of lights with a relay-board on the Pi, a web-interface for us so that we can alarm him the food is on the table by turning those LED-strips into flashing red alarm signal... For this project, different examples from this book must be combined.

So one final note to you and myself: **hop hop get to work, build, experiment and most importantly... have fun!!!**