

IOT

— A practical introduction with Pharo and PharoThings —

Steven Costiou
steven.costiou@inria.fr

Allex Oliveira
allex.oliveira@inria.fr

October 31, 2019

Plan

- 1 About this course
- 2 IOT: some generalities
- 3 Sensors and actuators
- 4 The Raspberry-pi computer
- 5 IOT with PharoThings
- 6 PharoThings in practice
- 7 PharoThings exercises!

Plan

- 1 About this course
- 2 IOT: some generalities
- 3 Sensors and actuators
- 4 The Raspberry-pi computer
- 5 IOT with PharoThings
- 6 PharoThings in practice
- 7 PharoThings exercises!

About this course

This is a practical introduction to IOT: we will learn how to write high-level software for sensors and how to use them in applications deployed on Raspberry-pi computers.

About this course

This is a practical introduction to IOT: we will learn how to write high-level software for sensors and how to use them in applications deployed on Raspberry-pi computers.

We will learn to:

- ▶ Program with Pharo and PharoThings on Raspberry-pi computers,
- ▶ design and implement software models of sensors,
- ▶ write applications using sensors,
- ▶ use Pharo remote development tools to debug running IOT applications.

About this course

This is a practical introduction to IOT: we will learn how to write high-level software for sensors and how to use them in applications deployed on Raspberry-pi computers.

We will learn to:

- ▶ Program with Pharo and PharoThings on Raspberry-pi computers,
- ▶ design and implement software models of sensors,
- ▶ write applications using sensors,
- ▶ use Pharo remote development tools to debug running IOT applications.

Want more?

- ▶ A MOOC with more theoretical aspects:
<https://www.coursera.org/learn/iot/>
- ▶ PharoThings materials: <https://github.com/SquareBracketAssociates/Booklet-APharoThingsTutorial>

Plan

- 1 About this course
- 2 IOT: some generalities
- 3 Sensors and actuators
- 4 The Raspberry-pi computer
- 5 IOT with PharoThings
- 6 PharoThings in practice
- 7 PharoThings exercises!

What is IOT?

Internet of Things

- ▶ Objects inter-connected through the internet:
- ▶ Objects are unique entities sharing data in a network:
a person, a phone, a device...
- ▶ Device:
hardware (micro-computer, sensors, motors...) + software (*intelligence*),
- ▶ Objects are interrelated:
they transfer data between them through the network
- ▶ Objects use a protocol to communicate:
for example, an HTTP REST API through TCP/IP...
- ▶ Objects need a power source to be autonomous

What is IOT?

Examples of IOT systems/applications

- ▶ Personal health or sport monitoring systems
- ▶ Smart homes: your fridge now tells you when you need to refill!
- ▶ Logistics: automated warehouses and production chain monitoring
- ▶ Smart grids: optimizing energy production and routing
- ▶ Etc.

Plan

- 1 About this course
- 2 IOT: some generalities
- 3 Sensors and actuators**
- 4 The Raspberry-pi computer
- 5 IOT with PharoThings
- 6 PharoThings in practice
- 7 PharoThings exercises!

Sensors

"A sensor is a device, module, machine, or subsystem whose purpose is to detect events or changes in its environment and send the information to other electronics, frequently a computer processor."

— <https://en.wikipedia.org/wiki/Sensor>

- ▶ Transforms a physical information (e.g., the temperature of the room) to an analog or digital signal
- ▶ The analog/digital signals are processed by the computer processor and interpreted and used by user-applications
- ▶ Sensors are sources of data (inputs) in IOT applications

Sensors

"A sensor is a device, module, machine, or subsystem whose purpose is to detect events or changes in its environment and send the information to other electronics, frequently a computer processor."

– <https://en.wikipedia.org/wiki/Sensor>

- ▶ Transforms a physical information (e.g., the temperature of the room) to an analog or digital signal
- ▶ The analog/digital signals are processed by the computer processor and interpreted and used by user-applications
- ▶ Sensors are sources of data (inputs) in IOT applications

Examples

- ▶ Temperature sensor: measures temperature in an environment
- ▶ Camera
- ▶ Accelerometer: measures acceleration
- ▶ ...

What do sensors measure?

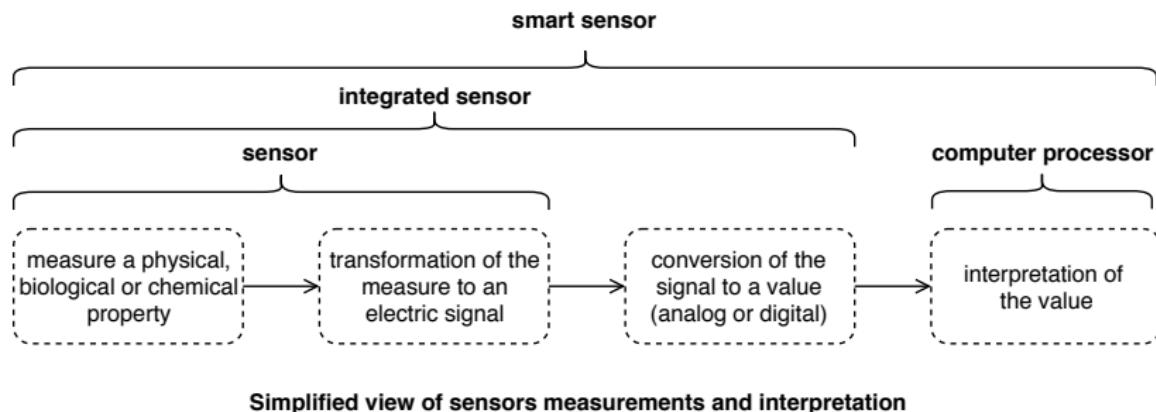
- ▶ One sensor only measure one specific property of its environment
- ▶ Ideally, a sensor should not interfere with the environment (*i.e.*, not influence the measured value)
- ▶ Sensors always work in association with an electronic system (in IOT)

Sensors

What do sensors measure?

- ▶ One sensor only measure one specific property of its environment
- ▶ Ideally, a sensor should not interfere with the environment (*i.e.*, not influence the measured value)
- ▶ Sensors always work in association with an electronic system (in IOT)

How do sensors measure?



Different kind of sensors

- ▶ **Basic sensor element:** transforms a physical phenomenon into an electronic signal

Example: RTD¹'s measure the resistance of metal, that changes with temperature

- ▶ **Integrated sensor:** sensor element with signal conditioning

Example: A sensor that transforms the resistance measured by a RTD into an analog signal (e.g., 0 to 1023)

- ▶ **Smart or intelligent sensor:** integrated sensor with additional features

Example: A sensor that transforms the analog value transformed from an RTD resistance into a human-readable temperature in Celsius

¹Resistance Temperature Detector

Actuators

Systems that physically interact with the environment

- ▶ Controlled by an electrical signal
 - ▶ Transforms the signal into a mechanical motion (output)
 - ▶ Requires a power source
- ▶ Has a direct impact on the environment
 - ▶ An actuator can affect sensors!

Examples

- ▶ LED: produces light
- ▶ Motor: induces movement
- ▶ Speaker: produces sound
- ▶ ...

Plan

- 1 About this course
- 2 IOT: some generalities
- 3 Sensors and actuators
- 4 The Raspberry-pi computer
- 5 IOT with PharoThings
- 6 PharoThings in practice
- 7 PharoThings exercises!

The Raspberry-pi single-board computer

Cheap general-purpose single-board computer

- ▶ ARM processor
- ▶ Linux based distributions or Windows IOT operating systems
- ▶ 40 GPIO (general-purpose input/output) pins to control sensors/actuators

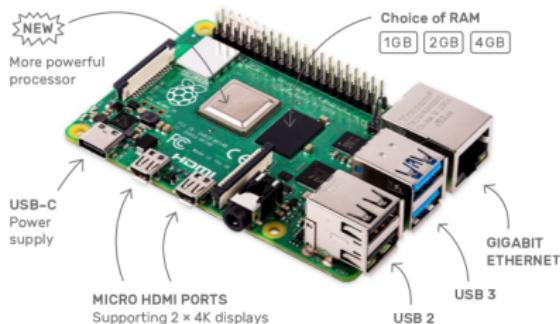


Figure: <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/>

General properties of GPIO pins

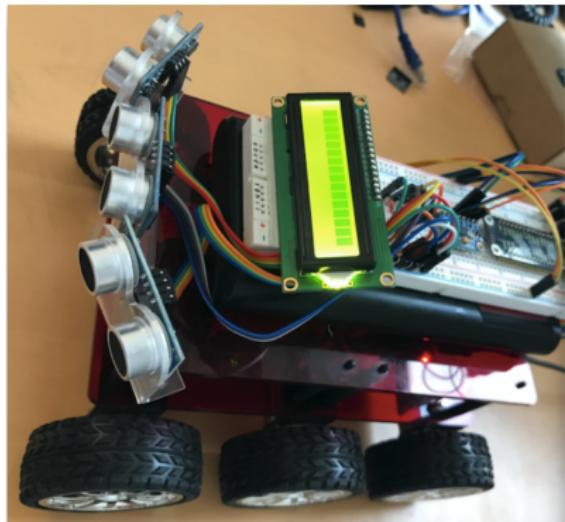
- ▶ Pins can be configured as input (e.g., to read sensors) or as output
- ▶ Values set to outputs or read as input can be set to high, or 1 (3.3V) or low, or 0 (0V)
- ▶ 4 hardware PWM (Pulse-Width Modulation) pins: control of the voltage by switching power on and off at a fast rate²
- ▶ SPI (Serial Peripheral Interface): synchronous serial communication interface between a master and a slave³
- ▶ I2C (Inter-Integrated Circuit): synchronous, multi-master, multi-slave, packet switched, single-ended, serial computer bus⁴
- ▶ 2 serial pins (transmission and reception)

²https://en.wikipedia.org/wiki/Pulse-width_modulation

³https://en.wikipedia.org/wiki/Serial_Peripheral_Interface

⁴<https://en.wikipedia.org/wiki/I%C2%B2C>

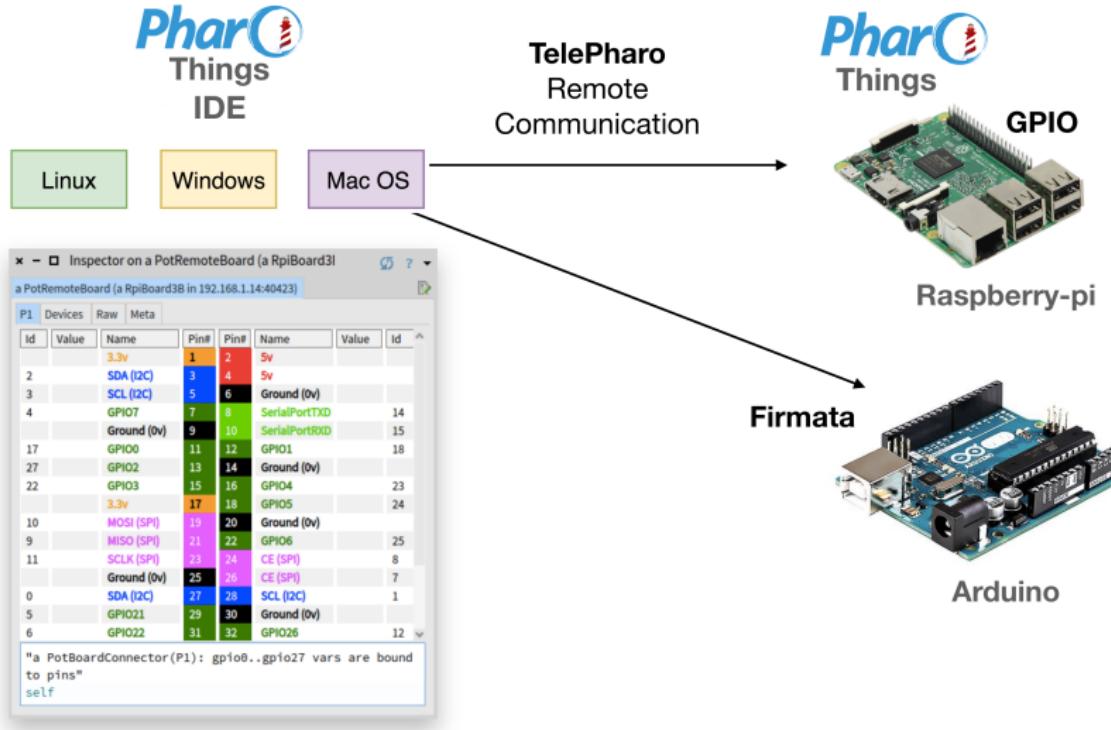
Example projects based on Raspberry-pi and Pharo-Things



Plan

- 1 About this course
- 2 IOT: some generalities
- 3 Sensors and actuators
- 4 The Raspberry-pi computer
- 5 IOT with PharoThings
- 6 PharoThings in practice
- 7 PharoThings exercises!

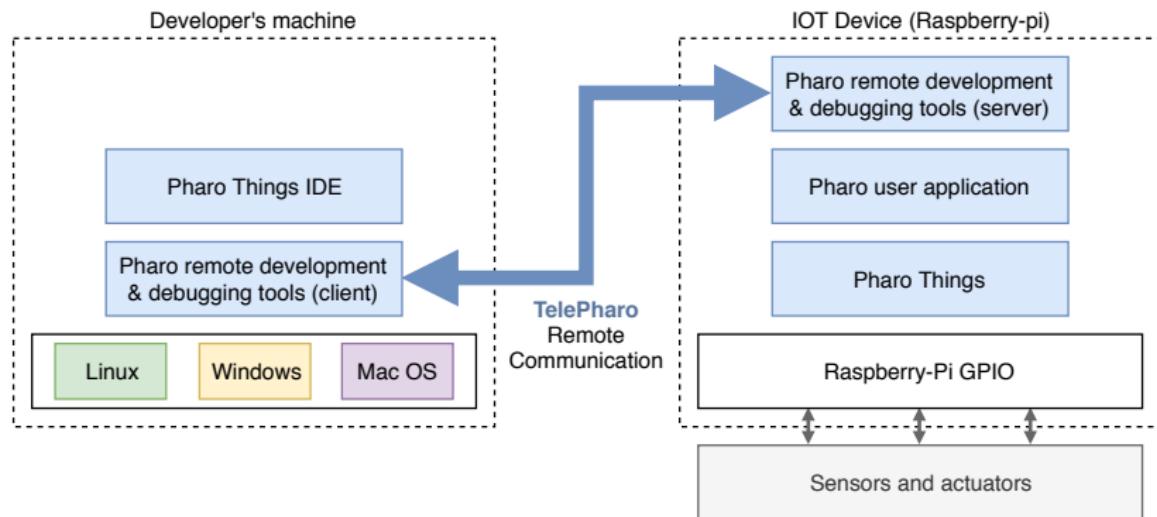
PharoThings



The PharoThings infrastructure

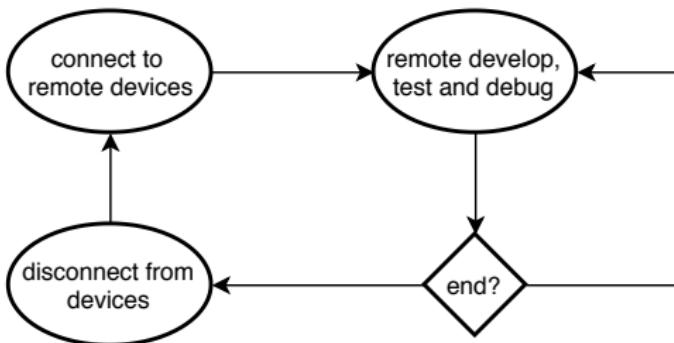
Developers (clients) connect to running IOT devices (servers)

- ▶ Developers connect to remote IOT devices running their application
- ▶ Pharo development and debugging tools are remotely usable through the TelePharo architecture
- ▶ Developers can connect to many devices at the same time from their development machine



PharoThings development process

1. Developers connect to running IOT application, deployed on IOT devices (Raspberry-pi)
2. Developers remotely and lively write, test and debug code
3. When development or debugging is finished, developers disconnect from the device
4. At any point developers can reconnect and start or continue developing their IOT app – which is still running!



Remote development and debugging tools

- ▶ Playground: write scripts and execute code
- ▶ Class browser: write classes and applications
- ▶ Inspector:
 - ▶ observe your application state
 - ▶ observe and interact with remote live objects
 - ▶ observe and control GPIO state
- ▶ Debugger: debug and fix your code (live!)
- ▶ Process browser: inspect and control running processes

Plan

- 1 About this course
- 2 IOT: some generalities
- 3 Sensors and actuators
- 4 The Raspberry-pi computer
- 5 IOT with PharoThings
- 6 PharoThings in practice
- 7 PharoThings exercises!

Downloading and starting PharoThings on the Raspberry-pi

On a running Raspbian on your Raspberry Pi

Open a terminal window in your Raspberry Pi (local or through remote SSH) and enter the following commands:

```
wget -O - get.pharoiot.org/server | bash
```

This downloads the server side and extracts the files to the `pharoiot-server` folder. The following commands will go to that folder and run the server:

```
cd pharoiot-server  
./pharo-server
```

- ▶ You should see this message: a `TlpRemoteUIManager` is registered on port 40423.
- ▶ This means that you have the TelePharo architecture running on your Raspberry on TCP port 40423 !
- ▶ Now you can use Pharo IoT on your computer to connect to your Raspberry and create IoT applications remotely.

Connecting to a remote device

On a running Raspbian on your Raspberry Pi

- ▶ First download the PharoThings IDE on your Windows/Linux/Mac computer get.pharoiot.org/multi.zip and run Pharo
- ▶ Open a playground in your local pharo image
- ▶ Write and doIt the following:

```
remotePharo := TIpRemoteIDE connectTo:  
    (TCPAddress ip: #[192 168 1 200] port: 40423).
```

If you don't receive any error, this means that you are connected. The `remotePharo` object allows you to interact with the remote device.

Inspecting the physical board (1)

We can inspect the software model of the remote device

- ▶ We must select the model corresponding to physical board
 - ▶ RpiBoardBRev1: Raspberry Pi Model B Revision 1
 - ▶ RpiBoardBRev2: Raspberry Pi Model B Revision 2
 - ▶ RpiBoard3B: Raspberry Pi Model B+, Pi2 Model B, Pi3 Model B, Pi3 Model B+
- ▶ In the playground on your local image, evaluate the following:

```
remoteBoard := remotePharo evaluate: [ RpiBoard3B current ].
```

- ▶ RpiBoard3B is a class from the remote system
- ▶ RpiBoard3B current returns the current instance of the board
- ▶ evaluate: remotely evaluates the block of code and returns the result
- ▶ remoteBoard stores that result, a remote instance of RpiBoard3B

Inspecting the physical board (2)

We can inspect the software model of the remote device

- ▶ Evaluate `remoteBoard inspect` to observe the state of the board
- ▶ The board inspector shows a layout of pins similar to Raspberry Pi docs
- ▶ This is a live tool which represents the current pins state

x - □ Inspector on a PotRemoteBoard (a RpiBoard3B in 192.168.1.106:40423)

a PotRemoteBoard (a RpiBoard3B in 192.168.1.106:40423)

P1	Devices	Raw	Meta				
Value	Function	Name	Pin#	Pin#	Name	Function	Value
3.3V		1	2	5v			
	SDA (I2C)	gpio2	3	4	5v		
	SCL (I2C)	gpio3	5	6	Ground		
	Clock	gpio4	7	8	gpio14	TxD (Serial)	
	Ground	9	10	gpio15	RxD (Serial)		
	gpio17	11	12	gpio18	PWM		
	gpio27	13	14	Ground			
	gpio22	15	16	gpio23			
3.3V		17	18	gpio24			
	MOSI (SPI)	gpio10	19	20	Ground		
	MISO (SPI)	gpio9	21	22	gpio25		
	SCLK (SPI)	gpio11	23	24	gpio8	CE (SPI)	
	Ground	25	26	gpio7	CE (SPI)		
	SDA (I2C)	gpio0	27	28	gpio1	SCL (I2C)	
	gpio5	29	30	Ground			
	gpio6	31	32	gpio12	PWM		
	PWM	gpio13	33	34	Ground		
	MISO (SPI)	gpio19	35	36	gpio16		
	gpio26	37	38	gpio20	MOSI (SPI)		
	Ground	39	40	gpio21	SCLK (SPI)		

Using GPIO: set a pin state (1)

- ▶ Digital pins can be set to HIGH (1) or LOW (0)
- ▶ They are shown with green/red icons which represent HIGH/LOW
- ▶ Digital pins' values are set using the value: message send to a pin
- ▶ Execute the following code in the lower pane of the board inspector: icons are updated according to pin value changes

```
pin := gpio4.  
pin beDigitalOutput.  
pin value: 1
```

Using GPIO: set a pin state (2)

- Results are visible in the board inspector:

The screenshot shows the BeagleBoard Inspector interface for a PotRemoteBoard (a RPiBoard3B). The main window displays a table of GPIO pins with columns for Value, Function, Name, Pin#, and Value. A status bar at the bottom shows the following code:

```
led1 := gpio4.  
led1 beDigitalOutput.  
led1 value: 1
```

Using GPIO: read a pin state (1)

- ▶ Digital pins can be configured as inputs, possible states: HIGH (1)/LOW (0)
- ▶ Digital pins' values are read using the value message send to a pin
- ▶ Output pins also respond to value, and return the last set state
- ▶ Write the following code in the lower pane of the board inspector, and execute it:

```
inputPin := gpio17.  
inputPin beDigitalInput.  
inputPin value inspect
```

Using GPIO: read a pin state (2)

- Results are visible in the board inspector:

The screenshot shows the 'Inspector' window for a 'PotRemoteBoard (a RpiBoard3B)'. The window has tabs for P1, Devices, Raw, and Meta. The P1 tab is selected, displaying a table of GPIO pins. The table includes columns for Value, Function, Name, Pin#, and Value. The Value column shows the current state of each pin (e.g., 3.3v, Ground, High, Low). The Function column indicates the pin's purpose (e.g., SDA (I2C), Clock, PWM, MOSI (SPI)). The Name column lists the pin names (e.g., gpio2, gpio3, gpio4, etc.). The Value column also displays the raw digital value (e.g., 1, 0, 2, 4, etc.). A status bar at the bottom shows the code: `inputPin := gpio17.
inputPin beDigitalInput .
inputPin value.] 0`.

Value	Function	Name	Pin#	Value
3.3v			1	2
SDA (I2C)	gpio2	3	4	5v
SCL (I2C)	gpio3	5	6	5v
out	Clock	gpio4	7	Ground
Ground		gpio17	9	10
in		gpio27	11	12
		gpio22	13	14
3.3v			15	16
MOSI (SPI)	gpio10	17	18	gpio18 PWM
MISO (SPI)	gpio9	19	20	Ground
SCLK (SPI)	gpio11	21	22	gpio25
Ground		23	24	gpio8 CE (SPI)
SDA (I2C)	gpio0	25	26	gpio7 CE (SPI)
		27	28	gpio1 SCL (I2C)
gpio5		29	30	Ground
gpio6		31	32	gpio12 PWM
PWM		gpio13	33	34
MISO (SPI)	gpio19	35	36	Ground
		gpio26	37	38
Ground			39	40
				gpio20 MOSI (SPI)
				gpio21 SCLK (SPI)

```
inputPin := gpio17.  
inputPin beDigitalInput .  
inputPin value.] 0
```

Scripting with the remote playground

Writing and reading GPIO from the remote playground

- ▶ We can interact with the GPIO in scripts and programs
- ▶ Open locally a remote playground: `remotePharo openPlayground`
- ▶ The remote playground opens, now get the current board instance:

```
board := RpiBoard3B current.
```

- ▶ Configure a new digital pin with the current board instance :

```
pin := PotGPIO id: 7.  
pin board: board.
```

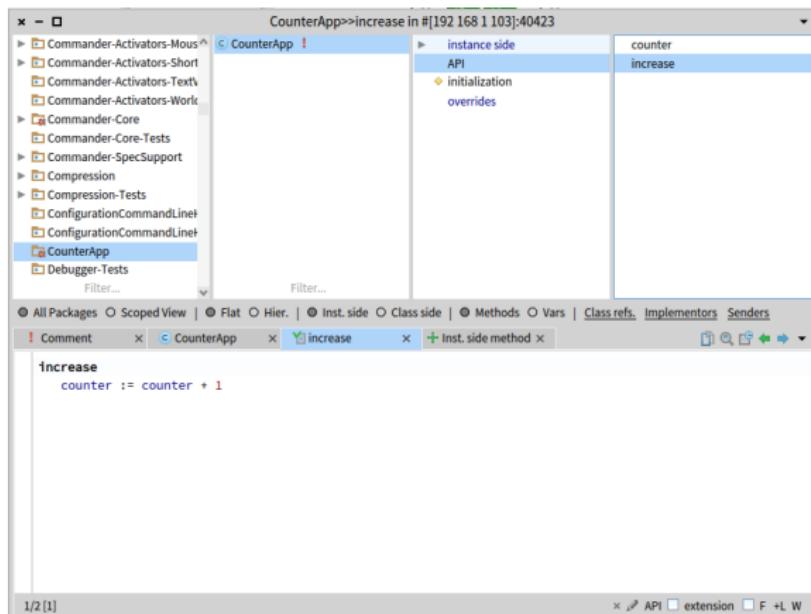
- ▶ Set the pin as output, set a HIGH value, inspect the state of the pin:

```
pin beDigitalOutput.  
pin value: 1.  
pin value inspect
```

Developing with the remote class browser

Develop your applications with the remote class browser

- ▶ Write a package
- ▶ Write a class with a method that increases a counter
- ▶ Use the remote playground to instantiate your class and play with your counter
- ▶ Save your remote code modifications: `remotePharo saveImage`



Remote debugging

Live debugging of a remote application

- ▶ In your remote counter class, write the following method:

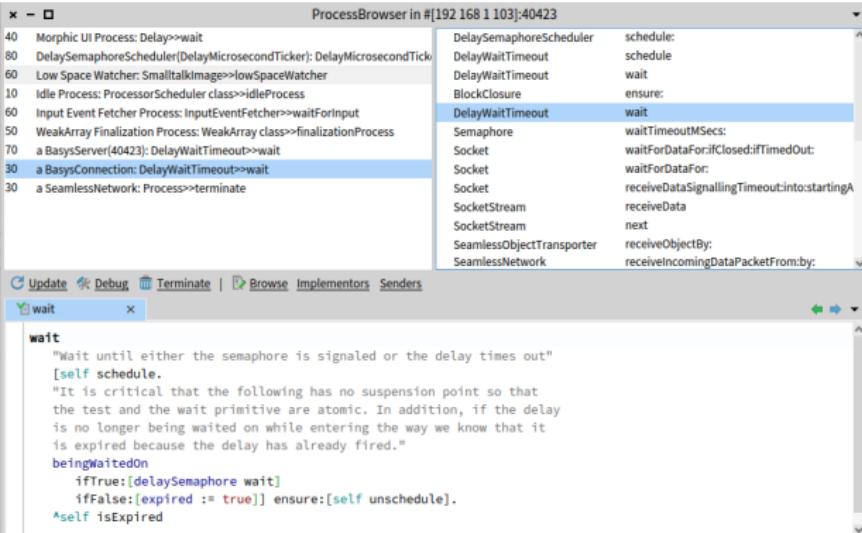
```
decrease  
    counter := counter / 0
```

- ▶ Instantiate your counter in the remote playground
- ▶ Call the decrease method: there is an error because of the division by 0!
- ▶ Click "debug": a remote debugger opens
- ▶ Fix the method, then click "proceed"
- ▶ The remote code is modified and the execution continues!

The remote process browser

Inspect and control processes running on your remote Pharo application

- ▶ Remotely open it using `remotePharo openProcessBrowser`
- ▶ Process priorities are on the left
- ▶ Process names are displayed in the first column
- ▶ Process stacks are on the right
- ▶ The action bar in the middle allows you to control processes



The screenshot shows the "ProcessBrowser" application window. The title bar reads "ProcessBrowser in #192.168.1.103:40423". The main pane displays a list of processes:

Priority	Process Name	Description
40	Morphic UI Process: Delay->wait	
80	DelaySemaphoreScheduler(DelayMicrosecondTicker): DelayMicrosecondTick	
60	Low Space Watcher: SmalltalkImage>lowSpaceWatcher	
10	Idle Process: ProcessorScheduler class->idleProcess	
60	Input Event Fetcher Process: InputEventFetcher->waitForInput	
50	WeakArray Finalization Process: WeakArray class->finalizationProcess	
70	a BayesServer(40423): DelayWaitTimeout->wait	
30	a BayesConnection: DelayWaitTimeout->wait	
30	a SeamlessNetwork: Process->terminate	

The process "a BayesConnection: DelayWaitTimeout->wait" is selected and highlighted with a blue selection bar.

The bottom pane shows the source code for the "wait" method:

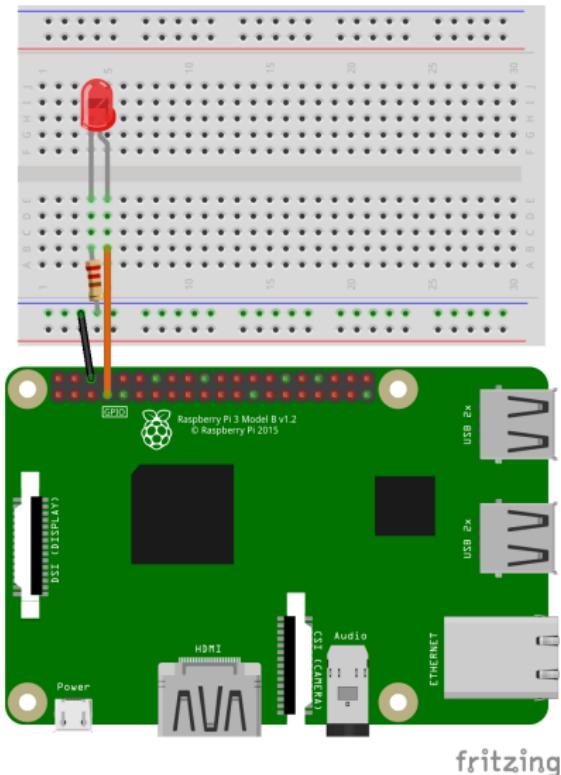
```
wait
    "Wait until either the semaphore is signaled or the delay times out"
    [self schedule].
    "It is critical that the following has no suspension point so that
     the test and the wait primitive are atomic. In addition, if the delay
     is no longer being waited on while entering the way we know that it
     is expired because the delay has already fired."
    beingWaitedOn
        ifTrue:[delaySemaphore wait]
        ifFalse:[expired := true] ensure:[self unschedule].
    ^self isExpired
```

Plan

- 1 About this course
- 2 IOT: some generalities
- 3 Sensors and actuators
- 4 The Raspberry-pi computer
- 5 IOT with PharoThings
- 6 PharoThings in practice
- 7 PharoThings exercises!

Turning LEDs on and off

Follow and adapt code from tutorial chapter 2



1. Build the LED circuit
2. Connect to the remote Pharo image
3. Open a remote Raspberry-pi board inspector
4. Set the LED pin to HIGH
5. Play with `toggleDigitalValue` to toggle the LED pin state

Blinking a LED

Follow and adapt code from tutorial chapter 3

1. Reuse the previous lesson to set an output pin for the LED
2. Write a loop that toggles the pin state
3. Use a delay to wait between each iteration
4. Create a process by forking the loop (and give it a name)

```
[10 timesRepeat: [
    led toggleDigitalValue.
    (Delay forSeconds: 1) wait
]
] forkNamed: 'BlinkerProcess'
```

Model your own LED blinder

Follow and adapt code from tutorial chapter 4

- ▶ We want a blinder object, that we can control!
1. Create a `Blinker` class in the remote browser
 2. The class must have a `process` instance variable
 3. The class must have a `delay` instance variable
 4. The class must have a `led` instance variable
 5. Create an `initialize` method to set the initial state of the blinder
 6. You must be able to start and stop the blinking

```
| blinder |
blinder := Blinker new.
blinder delay: 1 second.
blinder start. "start the blinder"
blinder stop. "stop the blinder"
```

Make your LED breath using PWM

Follow and adapt code from tutorial chapter 7

- We want a to make a LED light up then fade away

```
| breathingLed |
breathingLed := BreathingLed new.
breathingLed breatheDelay: 10 milliseconds.
breathingLed breatheFrom: 10 to: 25.
breathingLed startBreathing.
```

1. Create a BreathingLed class with the remote browser
2. Configure the LED pin to PWM output:

```
led function: PotPWMFunction new.
led bePWMOOutput.
```

3. The PWM value must be in the interval [0,1023]:

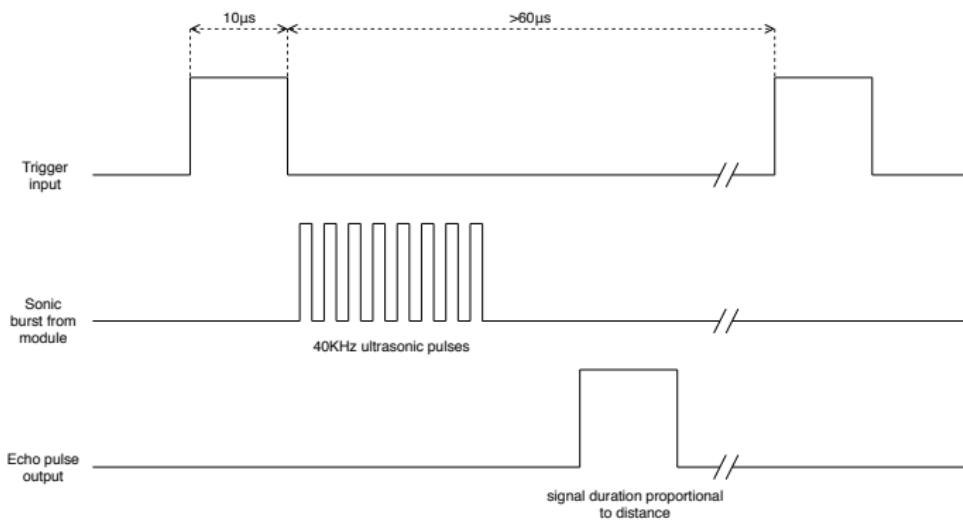
```
led writePWMValue: 42.
```

4. To make the led breath, you must increment the PWM value by 1 from a start value in a loop until it reaches a stop value. When it does, decrease the value by 1 until it reaches the start value again, and so on.

```
| breathingLed |
```

Ultra-Sonic sensor: basic concepts

- ▶ The sensor sends an ultrasonic pulse, triggered by a $10 \mu\text{s}$ 5V signal on the Trigger pin
- ▶ Pulse waves reflect on objects in front and are reflected back to the sensor
- ▶ The sensor detects these waves and sends a 5V signal on the Echo pin
- ▶ The duration of the echo signal is proportional to the distance: we can approximate the distance using the speed of sound ($\approx 340\text{m.s}^{-1}$)



Ultra-Sonic sensor: write your own driver!

Build the circuitry

- ▶ Follow instructions from chapter 10

Write your own driver

- ▶ Create a class with two instance variables: an echo and a trigger pins
- ▶ Create an instance creation method to initialize the pins:

```
sensor trigger: triggerPinId echo: echoPinId
```

- ▶ Write the sensor behavior:
 1. A method that sends a pulse
 2. A method that receives the pulse back signal and computes its duration
 3. A method that computes the distance from the pulse duration
 4. An interface method named `readDistance` that use all the previous method to return a distance (see next slide)
- ▶ Test your sensor!

Write your own driver: the pulse back signal reception (1/2)

Concept of a pulseIn⁵

- ▶ Reads a pulse (either HIGH or LOW) on a pin
- ▶ For example, if value is HIGH, waits for the pin to go from LOW to HIGH
- ▶ Starts a timer then waits for the pin to go LOW and stops timing
- ▶ Returns the length of the pulse in microseconds or gives up and returns 0 if no complete pulse was received within the timeout

In Pharo:

- ▶ We do not have the pulseIn mechanism in Pharo, therefore we have to simulate it!

⁵<https://www.arduino.cc/reference/en/language/functions/advanced-io/pulsein/>

Write your own driver: the pulse back signal reception (2/2)

Simulated pulseIn in Pharo

Function: pulseln

Data: The Echo pin *echo* set as digital input and with pull down resister enabled^a

Result: A duration

begin

```
  while read(echo) == 0 do
    | startTime ← currentTime
  end
  while read(echo) == 1 do
    | endTime ← currentTime
  end
  return endTime - startTime
end
```

^asend the enablePullDownResister message to the pin object

Write your own driver: the `readDistance` interface

Beware of timeouts!

- ▶ Use a semaphore to avoid infinite waiting on timeouts!

```
readDistance
    | travelTime semaphore |
    semaphore := Semaphore new.
    [ ... use the sensor to find travel time... ] fork.
    semaphore
        wait: 100 milliSeconds
        onCompletion: [... return the travel time... ]
        onTimeout: [ self rebootSensor. ^ -1 ]
```

- ▶ When the sensor times out, use the reboot trick:

```
rebootSensor
    echoPin beDigitalOutput; value: 1.
    1 milliSeconds wait.
    echoPin value: 0.
    echoPin beDigitalInput; enablePullDownResister.
    triggerPin beDigitalOutput; value: 0
```

Ultra-Sonic sensor: combine it with LED breathing!

Write an IOT application using an LED and an UltraSonic sensor

1. When there is nothing closer than 1 meter, the LED is off
2. When something is detected closer than 1 m, the light starts emitting light
3. The closer the obstacle, the brighter must be the LED!

Schedule 2h00 - 6h00

- ▶ (2h00 - 2h30) I2C protocol/connecting Rasps
- ▶ (2h30 - 3h00) Hands on BME280
- ▶ (3h00 - 3h40) Hands on LCD
- ▶ (3h40 - 4h00) Pause
- ▶ (4h00 - 5h00) Mini weather station cloud
- ▶ (5h00 - 6h00) Building a Webserver on Raspberry

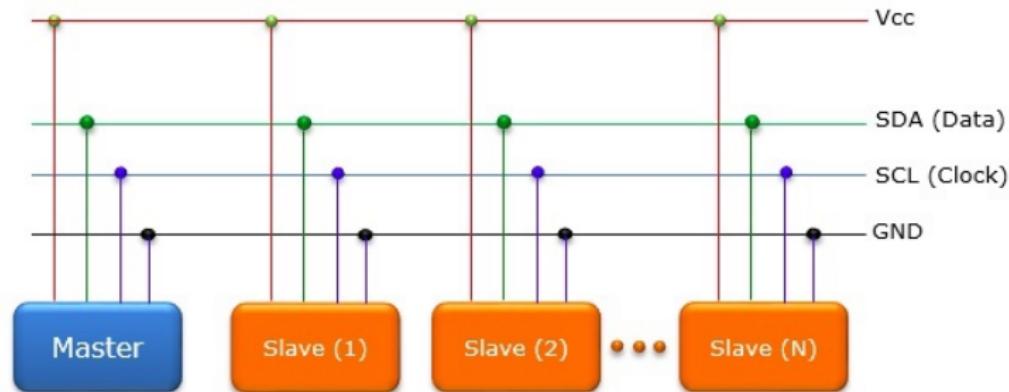
How I2C works?

- ▶ Use 2 wires SDA (serial data) and SCL (serial clock)
- ▶ Each device has set an ID or a unique address
- ▶ 112 devices using 7-bit addressing
- ▶ 1008 devices using 10-bit addressing

I2C Sensors

How I2C works?

- ▶ Use 2 wires SDA (serial data) and SCL (serial clock)
- ▶ Each device has set an ID or a unique address
- ▶ 112 devices using 7-bit addressing
- ▶ 1008 devices using 10-bit addressing



Examples

I2C Sensors

I2C and GPIO based devices

- ▶ PotLCDHD44780 class
- ▶ Written to work with both models

PotLCDHD44780I2C>>connect

The screenshot shows the Pharo IDE interface with the following details:

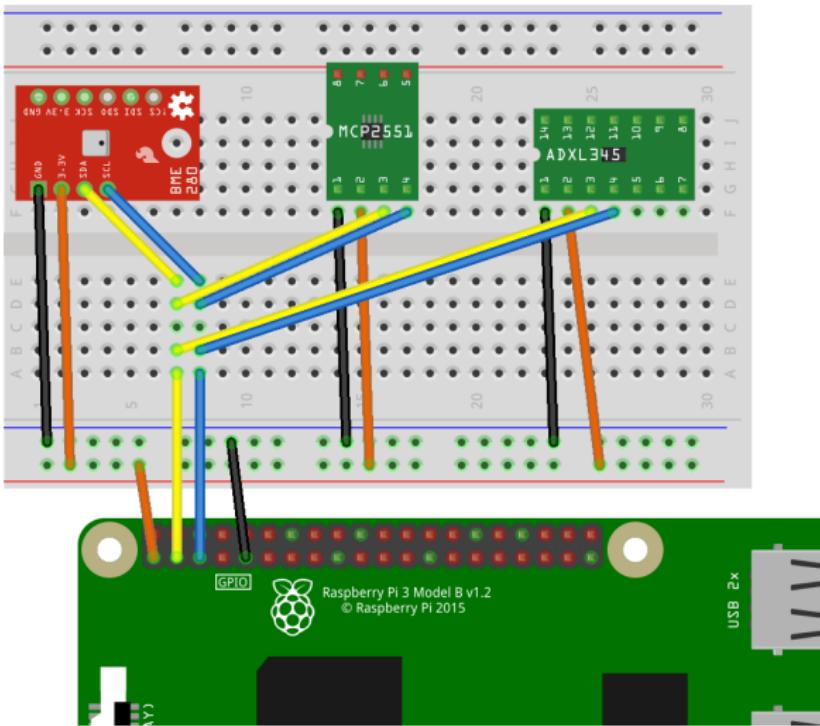
- Left pane:** A tree view of packages under "PharoThings". The "PharoThings-Devices-I2C" package is selected.
- Middle pane:** A list of methods for the selected class. The "connect" method is highlighted.
- Right pane:** A detailed view of the "connect" method's implementation. It includes:
 - Instance side methods: accessing, controlling, initialization, testing, writing, overrides.
 - Implementation details: configPeripherals, connect, initializeRegisters, isConnected, lower4Bits:, peripherals, syncClock:, upper4Bits:, write:mode:.
- Bottom pane:** A code editor showing the implementation of the "connect" method. The code is:

```
connect
    i2cConnection := board connectToI2CDevice: 16r27 .
    self initializeRegisters
```

I2C Sensors

BME280 sensor

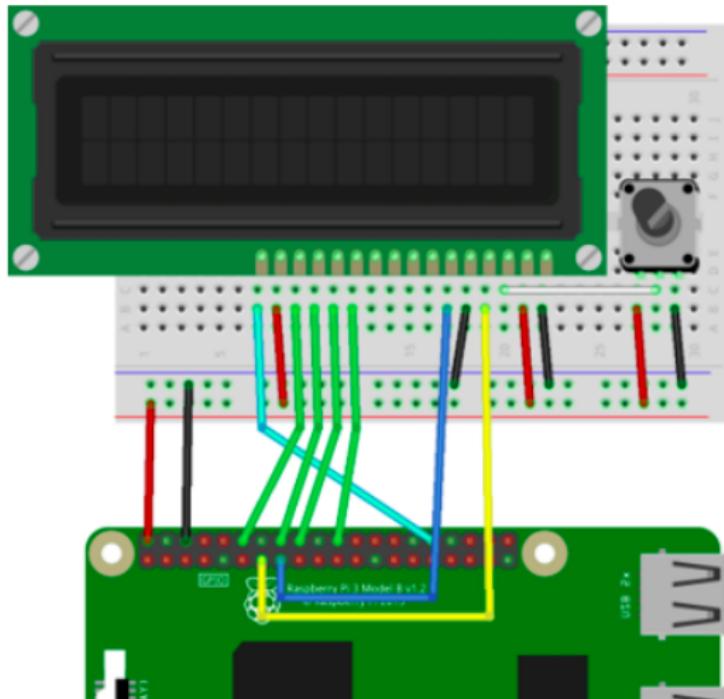
- ▶ PotBME280Device class



LCD screen

LCD 1602

- ▶ PotLCD1602Device class to GPIO device
 - ▶ PotLCD1602DeviceI2C class to I2C device



LCD screen

Write an IOT application using an LCD and an BME280 sensor

1. Shows the data of sensor on the LCD each 1 second

Chapter 12 of Bookletr

1. Create an account on Thingspeak
2. Create the class to show the sensor data on LCD
3. Create the class to send data to cloud

Chapter 13 of Bookletr

1. Create the html page and start the webserver
2. Create the methods to show the data on the web page
3. Exercises, connect the sensor and LED to the web page