Live Programming IoT devices

With Pharo Things

Do Hoang

University of Science And Technology of Hanoi

June 15, 2019

Section 1

Lesson 1 – Turning LED on/off

Lesson 1 – Turning LED on/off

Components

- 1 Raspberry Pi connected to your network (wired or wireless)
- 1 Breadboard
- 1 LED
- 1 Resistor (330 ohms)
- Jumper wires

Experimental procedure

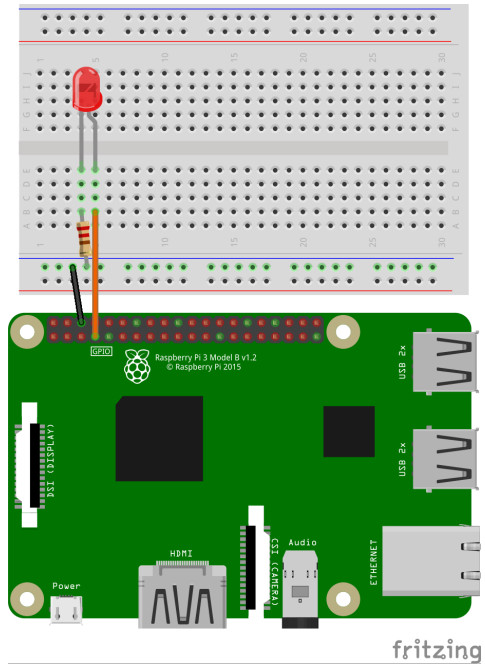

Figure 1: Physical connection LED

* The circuit consists of an LED that lights up when power is applied, a resistor to limit current and a power supply (the Rasp).

Physical connection LED detail

- Connect the Ground PIN from Raspberry in the breadboard blue rail (-). Raspeberry Pi models with 40 pins has 8 GPIO ground pins. You can connect with anyone. In this experiment we will use the PIN6 (Ground);
- Then connect the resistor from the blue rail on the breadboard (-) to a column on the breadboard
- Now push the LED legs into the breadboard, with the long leg (with the kink) on the right;
- And insert a jumper wire connecting the rigth column and the PIN7 (GPIO7).

Experimental code

Connecting remotely

- Run this code in Playground:

```
remotePharo := TlpRemoteIDE connectTo: (TCPAddress ip:
    #[193 51 236 167] port: 40423)
GTInspector enableStepRefresh

remoteBoard := remotePharo evaluate: [ RpiBoard3B
    current].

remoteBoard inspect.
```

$\Longrightarrow$ Make a new connection to your Rpi and Open the *Remote Playground*

Experimental code

- To control the LED we first introduce the named variable #led which we assigned to GPIO7 pin instance:

```
led := gpio7.
```

- Then we configure the pin to be in digital output mode and set the value:

```
led beDigitalOutput.
led value: 1.
```

$\Longrightarrow$ It turns the LED on.

Experimental code

- You can **notice** that gpio variables are not just numbers/ids.
- They are real **objects** with behaviour.
- For example you can ask pin to toggle a value:

```
led toggleDigitalValue.
```

- Or ask a pin for current value if you want to check it:

```
led value.
```

Result

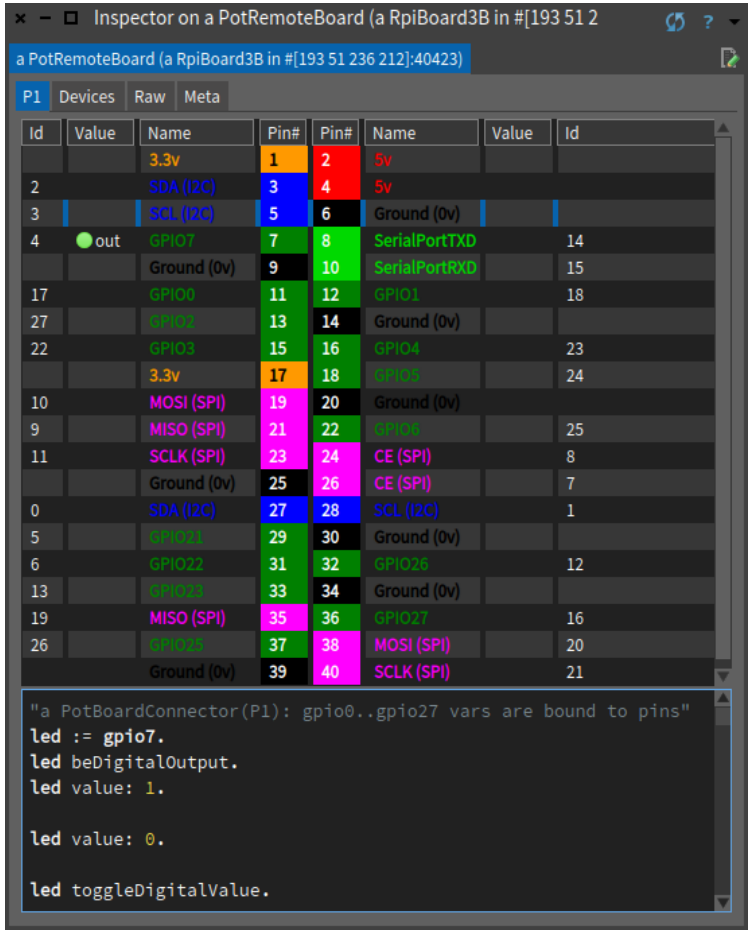

Figure 2: Remote Board Inspector

Section 2

Lesson 2 - Blinking LED

Experimental code

Connecting remotely

- Run this code in Playground:

```
remotePharo := TlpRemoteIDE connectTo: (TCPAddress ip:
    #[193 51 236 212] port: 40423)
GTInspector enableStepRefresh.

remoteBoard := remotePharo evaluate:
    [ RpiBoard3B current].

remoteBoard inspect.
```

$\Longrightarrow$ Make a new connection to your Rpi and Open the *Remote Playground*

Experimental code

- We still assigned the LED pin to GPIO7 pin:

```
led := gpio7.
led beDigitalOutput.
```

- To blink the LED, we create a loop to change the value of LED by time.
- We use the method *toggleDigitalValue* as previously.
- For example we blink the LED every 1 second by 10 times.

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

$\Longrightarrow$ Your LED is blinking now!

Section 3

Lesson 3 - Introduce to Pharo object-oriented

Blinking LED using OOP

Experimental code

```
|blinker|
blinker := Blinker new.
blinker timesRepeat: 10 waitForSeconds: 1.
```

Experimental code

- We declare the variable *blinker* in the first line.
- We will use this variable to create an object using the Blinker class.
- In the second line, we instantiate the Blinker class in the *blinker* variable
- In the third line, we send some messages to the blinker object to controll the times.

$\Longrightarrow$ This will make the GPIO behave according to the parameters

Create your own class remotely

- To create a class, we need first to create a package.
- In your local playground, call the Remote System Browser of your Raspberry Pi

```
remotePharo := TlpRemoteIDE connectTo: (TCPAddress ip:
    #[193 51 236 212] port: 40423).
remotePharo openBrowser.
```

Create a package

- Using the Remote Browser to create.
- *Right-click* the package area and enter the package name.
- For example, we create a package named *PharoThings-Lessons*

Create a class

- Edit the default class template by changing the *#NameOfSublass* to the name of new class
- For example, let's create the class *#Blinker*
- The class name begin with hash symbol (#) and a calpital leter.

```
Object subclass: #Blinker
instaceVariableNames: 'led'
classVariableNames: ''
package: 'PharoThings-Lessons'
```

- Right click on the code and select *Accept* option.
  $\Longrightarrow$ The class is compiled and added to the system.

Create a protocol

- Create a new protocol to organize the methods.
- The first protocol: `initialization`

Create a protocol

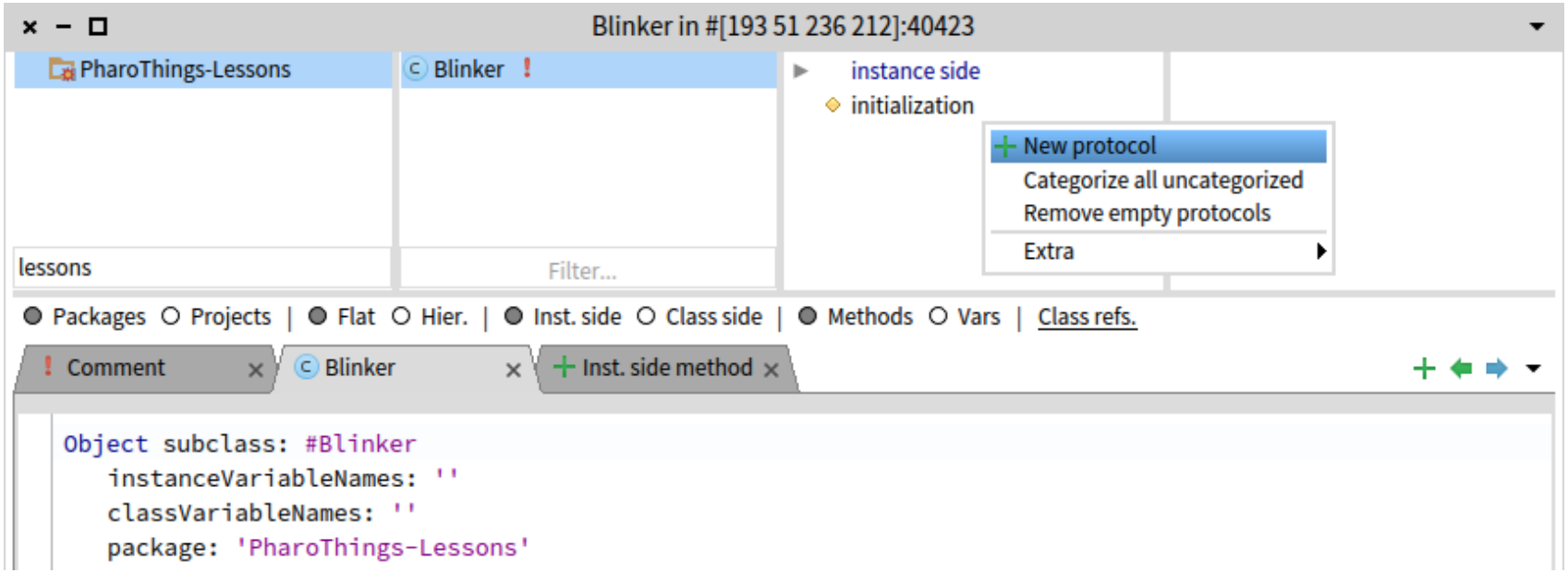

Figure 3: Creating a package remotely.

Create a method

- Inside this protocol, create an `initialize` method.
  $\Longrightarrow$ Everytime a new object was created using Blinker class, this method will be executed to define some variable in the new object.
- Let's use the instance variable `led`.
- The instance variable is private to the object and accessible by any methods inside this class.
- These methods can access this variable to get or set any value to it.

initialize

```
initialize
    led := PotClockGPIOPin id: 4 number: 7.
    led board: RpiBoard3B current; beDigitalOutput
```

Creating the initialize method

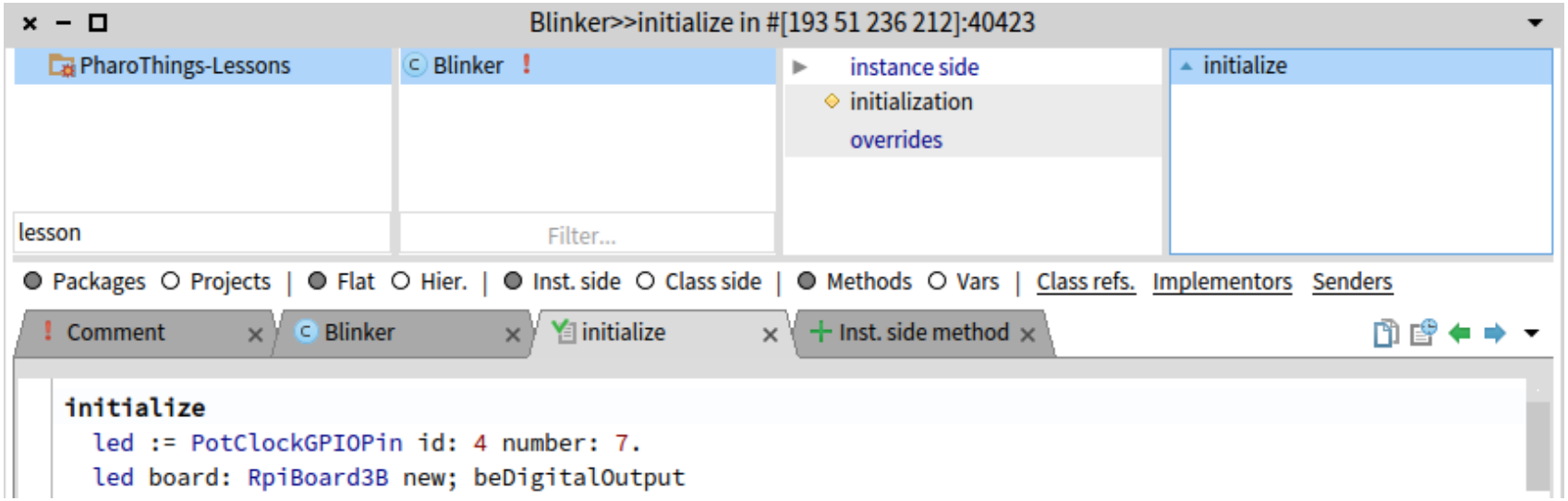

Figure 4: Creating the initialize method

Explanation

Explanation

- The first line defines the name of the method;
- In the second line, we configure the GPIO that we wanna use. Note that we need the GPIO number and ID. The ID is required to communicate with WiringPi Library. You can seethe ID and GPIO number in PotRemoteBoard inspector.
- In the third line, we define the model of the Raspberry board and con- figure this GPIO as beDigitalOutput. This means that when the GPIO change to `value:1`, the power will go out of the GPIO to power the LED.
- Compile your code (cmd + S) and the method will be shown in the remote browser.

Create a method to do actions

- Let's create a method to control the object led inside the class Blinker.
- Create a protocol `operations` and inside this protocol, create the following method

Method

```
timesRepeat: anInteger waitForSeconds: aNumber
        [ anInteger timesRepeat: [
            led toggleDigitalValue.
            (Delay forSeconds: aNumber) wait
        ] ] forkNamed: 'BlinkerProcess'.
```

Explanation

Explanation

- In the first line, we define the message with `timesRepeat:` and wait `ForSeconds:`. We inform the kind of value will be received, creating 2 variables: aNumber and anInteger;
- We replace these variables in the code and now we have the control to say how many times repeat and for how many seconds;
- We finished the code by putting everything inside a fork to create a process in Pharo. While the process is running, you can open the Remote Process Browser (remotePharo openProcessBrowser) and see the process. This is useful when you wanna kill the remote process.

Using the new class

- Now we can use the class that we created, the Blinker class. To do this, let's open the Remote Playground:

```
remotePharo openPlayground
```

Call blinker class

```
|blinker|
    blinker := Blinker new.
    blinker timesRepeat: 10 waitForSeconds: 1.
```

Save your work

Don't forget to save your work remotely. To do this, run this command on your local playground:

```
remotePharo saveImage.
```