A PharoThings Tutorial

Allex Oliveira

November 5, 2018

Copyright 2017 by Allex Oliveira.

The contents of this book are protected under the Creative Commons Attribution-ShareAlike 3.0 Unported license.

You are **free**:

- to **Share**: to copy, distribute and transmit the work,
- to **Remix**: to adapt the work,

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page:
http://creativecommons.org/licenses/by-sa/3.0/

Any of the above conditions can be waived if you get permission from the copyright holder. Nothing in this license impairs or restricts the author's moral rights.

Your fair dealing and other rights are in no way affected by the above. This is a human-readable summary of the Legal Code (the full license):
http://creativecommons.org/licenses/by-sa/3.0/legalcode

Layout and typography based on the `sbabook` LaTeX class by Damien Pollet.

Contents

Illustrations ii

1 A brief introduction to Pharo object-oriented language 1

1.1 Developing a simple LED blinker . . . 1
1.2 Our use case . . . 1
1.3 Create your own class remotely . . . 2
1.4 Create a package . . . 3
1.5 Create a class . . . 3
1.6 Create a protocol . . . 3
1.7 Creating an initialize method . . . 5
1.8 Using your new class . . . 6
1.9 Save your work . . . 7
1.10 Conclusion . . . 7

Illustrations

1-1 Remote System Browser. 2
1-2 Creating a package remotely. 3
1-3 Creating a class remotely. 4
1-4 Creating a package remotely. 4
1-5 Looking for ID and GPIO number on Remote Inspector. 5
1-6 Creating a initialize method. 6
1-7 Creating an operation method. 7
1-8 Remote playground. 7

CHAPTER 1

A brief introduction to Pharo object-oriented language

Pharo is a new generation reflective language and programming environment. The last code was executed inside the remote inspector. To get started using OOP (Object-oriented programming) with classes, methods, and instances, I invite you to implement a simple application to blink the LEDs. Pharo will fundamentally change your perception about what OOP is.

1.1 Developing a simple LED blinker

The following part of this chapter and application was based on the exercise `Developing a Simple Counter`, of Week 1 of Pharo MOOC (https://-mooc.pharo.org/). I strongly recommend that you read and do the "counter exercise" to better understand the concepts explained here. And, of course, do the MOOC to learn how to develop using Pharo and the OOP concept, Object-oriented programming.

1.2 Our use case

We want to create a blinker led using a few parameters such as time to repeat the blinking LED and how many seconds to wait between blinks. The following code should run in the playground when we finish this lesson:

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

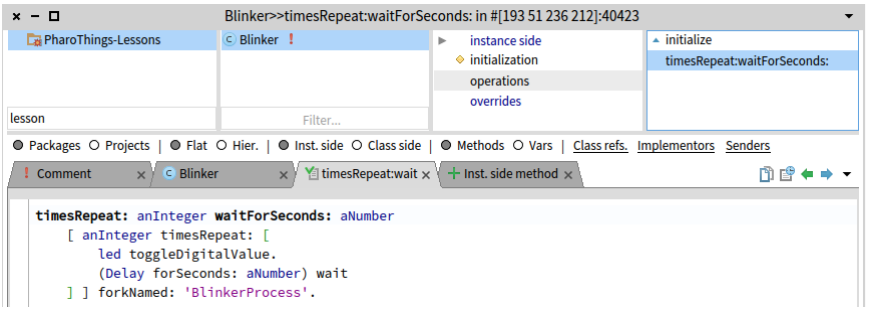

Figure 1-1 Remote System Browser.

Here is a short explanation of this code:

- In the first line, we declare the variable `blinker`. We can use any name. We will use this variable to create an object using the Blinker class;
- In the second line, we instantiate the Blinker class (with uppercase B) in the `blinker` variable, creating an object. In this lesson, we will create this class and methods to control the LED;
- In the third line, we send some messages to `blinker` object, for how long and how many times per second. This will make the GPIO behave according to the parameters sent.

Now we will develop all the mandatory class and methods to support this scenario.

1.3 Create your own class remotely

Let's create our first class. To create a class in Pharo, we need first to create a package. Inside the package, you can create many classes and inside the classes, you can create many methods. The methods are organized in protocols, to become more easily navigate between them. Take a look in Figure 1-1 to better understanding. *edit image, put name in windows

In your local playground, call the Remote System Browser of your Raspberry Pi. If you are already connected in your Raspberry Pharo, you do not need to run the first line below again. This will open a window as shown in Figure 1-1

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

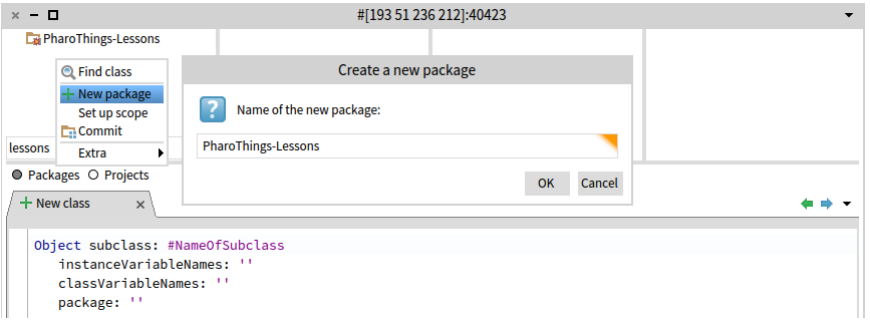

Figure 1-2 Creating a package remotely.

1.4 Create a package

Let's create a package using the Remote Browser. Right-click the package area and enter the package name, as shown in Figure 1-2. In this example, we will create a package named `PharoThings-Lessons`.

1.5 Create a class

To create a new class, edit the default class template by changing the #NameOfSubclass to the name of the new class. In this example let's create the class `#Blinker`. Take care that the class name begins with a capital letter and that you do not remove the octothorpe symbol (#) in front of NameOfSubClass.

You must then fill in the names of the instance variables for this class. We need an instance variable called `led`. Be careful to leave the strings quotes!

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

Now we need to compile it. Right click on the code area and select Accept option. The `Blinker` class is now compiled and added to the system, as shown in Figure 1-3.

1.6 Create a protocol

Let's create a new protocol to organize the methods. The first protocol we are going to create is `initialization`, as shown in Figure 1-4.

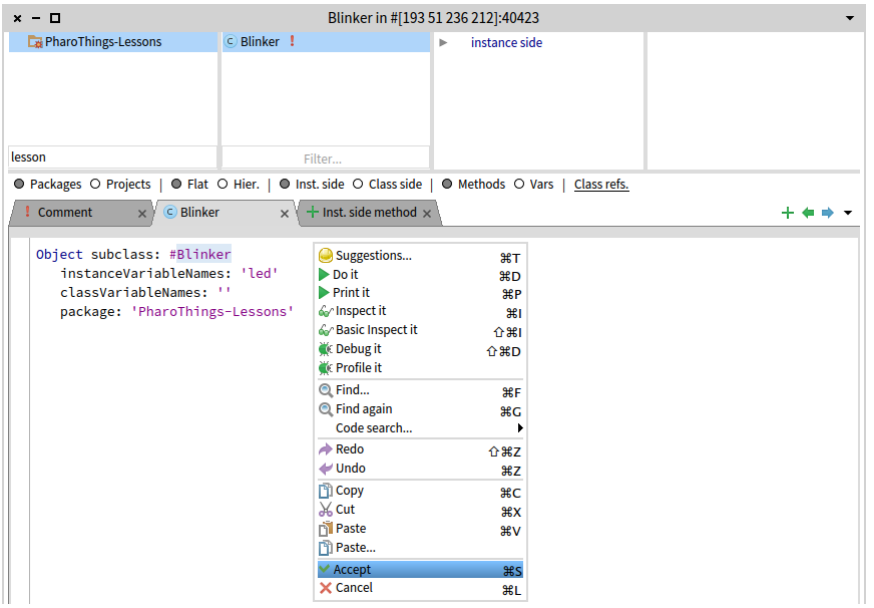

Figure 1-3 Creating a class remotely.

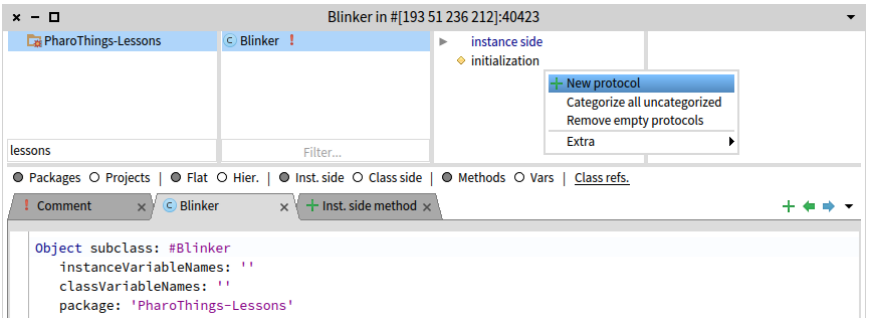

Figure 1-4 Creating a package remotely.

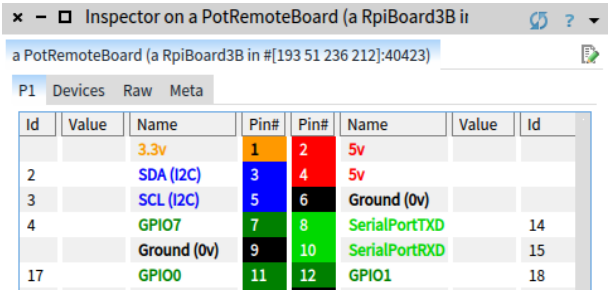

Figure 1-5 Looking for ID and GPIO number on Remote Inspector.

1.7 Creating an initialize method

Inside this protocol, we will create a `initialize` method. This means that every time we create a new object using this class, in this case, the Blinker class, this method will be executed to define some parameters in the new object.

Let's use the instance variable `led`, which we defined when we created the class. 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
  led := PotClockGPIOPin id: 4 number: 7.
  led board: RpiBoard3B new; beDigitalOutput
```

Here is a short explanation of this code:

- 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 Wiring Pi Library. You can see the `ID` and `GPIO number` in PotRemoteBoard inspector, as shown in Figure 1-5;
- In the third line, we define the model of the Raspberry board and configure 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, as shown in Figure 1-6:

Creating a method to do actions

Now let's create a method to control the object led inside the class Blinker. Let's take the code that we used in PotRemoteBoard inspector to do the LED

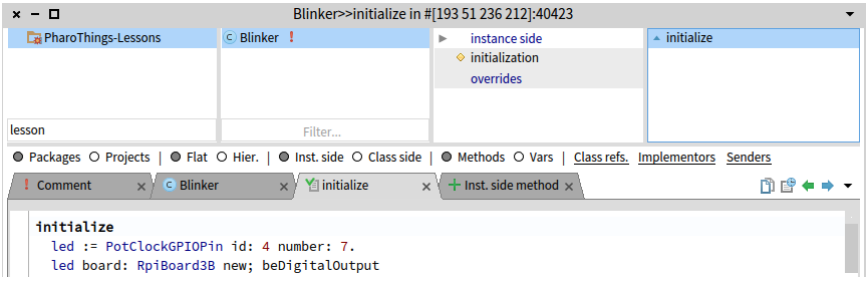

Figure 1-6 Creating a initialize method.

to blink and replace the numbers on code for two arguments. Create the protocol `operations` and inside this protocol, create the following method:

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

Here is a short explanation of this code:

- In the first line, we define the message with timesRepeat: and waitForSeconds:. 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.

Compile your code (cmd + S) and the method will be shown in the remote browser, as shown in Figure 1-7:

1.8 Using your 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.
```

and run the code that we saw in the begin of this lesson:

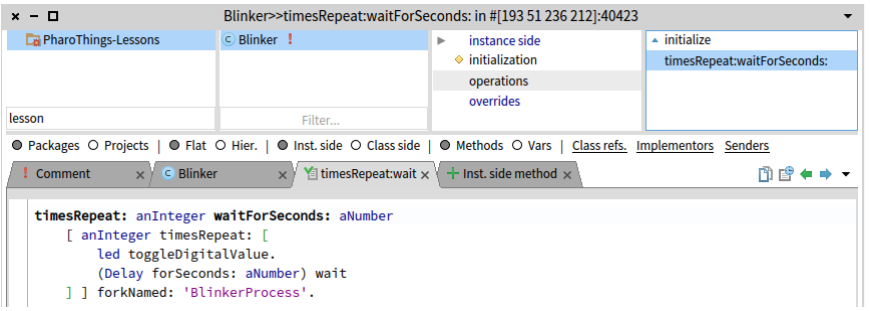

Figure 1-7 Creating an operation method.

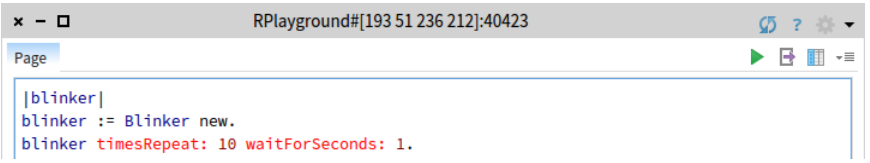

Figure 1-8 Remote playground.

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

Run this code, as shown in Figure 1-8 and... `cool`! Now your LED is blinking! And the better, you did this using Object-oriented programming!

1.9 Save your work

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

```
remotePharo saveImage.
```

1.10 Conclusion

In this tutorial, you learned how to define packages, classes, and methods. The flow of programming that we chose for this first tutorial is similar to most of the programming languages.

With PharoThings you can remotely develop and manage your Raspberry GPIOs. Very easy and powerful.

In the next lesson, let's use what we learned in this lesson and write a simple code to flowing lights using 4 LEDs.