CHAPTER 1

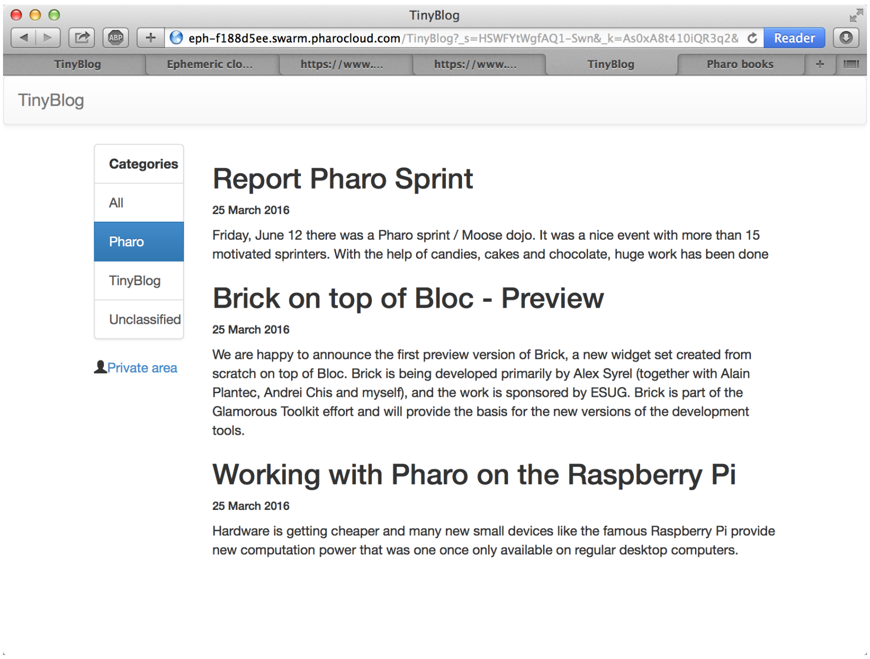
TinyBlog: presentation and model

In this project, we will guide you to develop a mini project: a small web application, named TinyBlog, that manages a blog system (see its final state in Figure 1.1). The idea is that a visitor of the web site can read the posts and that the post author could connect to the web site to admin the application and manage its posts (add, remove and modify existing ones).

TinyBlog is a small pedagogical application that will show you how to define and deploy a web application using Pharo / Seaside / Mongo and frameworks available in Pharo such as NeoJSON. We will show you how to expose your application via a REST server. Presented solutions are sometimes not the best to offer you a room for improvement. Our goal is not to be exhaustive. We present one way to develop TinyBlog nevertheless we invite the reader to read further references such as books or tutorials on Pharo to deepen his expertise and enhance his application.

1.1 Installing Pharo

In this project, we assume that you use Pharo 5.0 and since we will also use some extra libraries and frameworks dedicated to web development such as: Seaside, Magritte, Bootstrap, Voyage, VoyageMongo, ... we provide a specific Pharo web image that already includes all of these libraries. You can download it at the following address: http://mooc.pharo.org/. For this project, we suggest to always use this image because it contains all the packages you need.



Report Pharo Sprint

25 March 2016

Friday, June 12 there was a Pharo sprint / Moose dojo. It was a nice event with more than 15 motivated sprinters. With the help of candies, cakes and chocolate, huge work has been done

Brick on top of Bloc - Preview

25 March 2016

We are happy to announce the first preview version of Brick, a new widget set created from scratch on top of Bloc. Brick is being developed primarily by Alex Syrel (together with Alain Plantec, Andrei Chis and myself), and the work is sponsored by ESUG. Brick is part of the Glamorous Toolkit effort and will provide the basis for the new versions of the development tools.

Working with Pharo on the Raspberry Pi

25 March 2016

Hardware is getting cheaper and many new small devices like the famous Raspberry Pi provide new computation power that was one once only available on regular desktop computers.

Figure 1.1 Final state of the TinyBlog application

1.2 The model of posts

The model part of TinyBlog is really simple. We start here with the `TBPost` class.

TBPost Class

We adopt the following naming convention: all class names will be prefixed by `TB` (for TinyBlog). You must choose another prefix (e.g. `TBM`) if you want to be able to load the correction code in the same Pharo image as your code to compare both implementations.

Define the `TBPost` class using:

```
Object subclass: #TBPost
   instanceVariableNames: 'title text date category visible'
   classVariableNames: ''
   package: 'TinyBlog'
```

Description of a post

A blog post is described by 5 instance variables.

Variable	Signification
`title`	post title
`text`	post text
`date`	date of writing
`category`	name of the category of the post
`visible`	is the post publicly visible or not?

All of these variables have corresponding accessor methods in the ’accessing’ protocol.

```
TBPost >> title
   ^ title
```

```
TBPost >> title: anObject
   title := anObject
```

```
TBPost >> text
   ^ text
```

```
TBPost >> text: anObject
   text := anObject
```

```
TBPost >> date
   ^ date
```

```
TBPost >> date: anObject
   date := anObject
```

```
TBPost >> visible
   ^ visible
```

```
TBPost >> visible: anObject
   visible := anObject
```

```
TBPost >> category
   ^ category
```

```
TBPost >> category: anObject
   category := anObject
```

1.3 Post Visibility

We should add methods to make a post visible or not and also test if it is visible. Those methods are defined in the ’action’ protocol.

```
TBPost >> beVisible
   self visible: true
```

```
TBPost >> notVisible
   self visible: false
```

```
TBPost >> isVisible
   ^ self visible
```

1.4 Initialization

The `initialize` method ('initialization' protocol) sets the date to the current day and the visibility to false: the user must explicitly make a post visible. This allows him to write drafts and only publish when the post is finished. By default, a post belongs to the 'Unclassified' category. This category name is defined on class-side by the `unclassifiedTag` method.

```
TBPost class >> unclassifiedTag
   ^ 'Unclassified'
```

Pay attention the method `unclassifiedTag` should be defined on the class-side of the class `TBPost` (click on the `class` button to define it). The other methods are defined on the instance-side: it means that they will be applied to `TBBlog` instances.

```
TBPost >> initialize
   self category: TBPost unclassifiedTag.
   self date: Date today.
   self notVisible
```

In the above solution, it would be better that the `initialize` method does not hard code the reference to the `TBPost` class. We will present and discuss this point later in the MOOC.

1.5 Posts creation methods

On class-side, we add methods to ease posts creation for blogs that belong to a category or not.

```
TBPost class >> title: aTitle text: aText
   ^ self new
        title: aTitle;
        text: aText;
        yourself
```

```
TBPost class >> title: aTitle text: aText category: aCategory
   ^ (self title: aTitle text: aText)
            category: aCategory;
            yourself
```

1.6 Creating posts

You can now create some posts. Open the Playground tool and execute the following snippet:

```
TBPost
  title: 'Welcome in TinyBlog'
  text: 'TinyBlog is a small blog engine made with Pharo.'
  category: 'TinyBlog'
```

If you inspect the resulting object of the above expression (right click on the expression and select the ”inspect it” menu entry), you will get an inspector on the newly created `TBPost` object.

1.7 Testing post classification

We define a method for testing if a post has a category or not.

```
TBPost >> isUnclassified
   ^ self category = TBPost unclassifiedTag
```

Similarly to `initialize`, it would be better to not hard code the reference to the `TBPost` class.

1.8 Conclusion

We have now a first part of the model and it is a good time to save your Pharo image (menu item ’save’). In the next session we will show you how to save your code (package) with the Pharo versioning system.