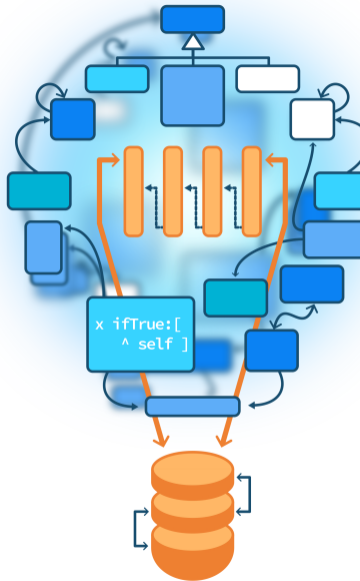


Blocks vs. Objects

Rethinking common abstractions

S.Ducasse, L. Fabresse, G. Polito, and P. Tesone



Goals

- Thinking about API
- **Rethinking** block usage
- Blocks are powerful and handy
- Small objects are **better** in the long run



Blocks are powerful

Blocks

- Central to Pharo syntax and object model
- Iterators
- New iterator definition
- DSL-like APIs



Central to message based syntax

- Remember that blocks freeze execution and give the power to decide when to execute
- Controlling behavior of block execution is the **key** for Pharo **compact** syntax

```
False >> ifTrue: trueAlternativeBlock ifFalse: falseAlternativeBlock  
  ^ falseAlternativeBlock value
```

```
True >> ifTrue: trueAlternativeBlock ifFalse: falseAlternativeBlock  
  ^ trueAlternativeBlock value
```



Iterators

Blocks are the cornerstone of iterators

```
#(1 2) allSatisfy: [ :each | each even ]
```

```
(String streamContents: [:s | #(1 2 3)]  
do: [:each | s << each asString]  
separatedBy: [s << ', ']])
```



New iterator definition

Blocks support definition of **new** iterators

SequenceableCollection >> pairsDo: aBlock

"Evaluate aBlock with my elements taken two at a time. If there's an odd number of items, ignore the last one. Allow use of a flattened array for things that naturally group into pairs. See also pairsCollect:"

1

```
to: self size // 2
```

```
do: [:index | aBlock
```

```
    value: (self at: 2 * index - 1)
```

```
    value: (self at: 2 * index) ]
```



DSL like APIs

```
GLMCompositePresentation new tabulator with: [ :t |
  t transmit from: #index; to: #details; andShow: [ :composite |
    composite text
      title: 'XML';
      display: [ :file | file contents ].
    composite list
      title: 'Targets';
      display: [ :file | (XMLDOMParser parse: file contents) // 'target' ];
      format: [ :xmlElement | xmlElement attributeAt: 'name' ].
    composite roassal2
      title: 'Dependencies';
      initializeView: [ RTMondrian new ];
      painting: [ :view :file |
        ...
      ]].
```



Stepping back

Blocks are **on the spot poor literal** objects

- What is the difference between a block and a simple object understanding value?
- With a block, no need to create a class, no need to define a method

But...



Analysis

Blocks are nice but not a panacea:

- Storing and changing state is **cumbersome**
- One single message: `value!`
- They do not **expose well** the arguments they need
- It makes scripting easy but extension difficult
- Having richer API is impossible

Let us study the limits!



Blocks are black boxes

- You can only send the messages `value*` to a block.
- It is **hard and cumbersome** to store and access the state in a block as in an object
 - Imagine passing a block around and wanting to accumulate information
 - You can't!



Arguments?

- What if you want optional arguments?
 - Then you are doomed to choose which arguments and which order
- `cull`: is reflective by nature
 - Avoid using it



Argument order requires to know the block definition!

Blocks do not expose well the arguments they need

```
aCol inject: default into: [:a :b | ... ]
```

What is a and b?



Block limits

- Saving blocks is **painful**
- Adding behavior (i.e., offering another message) is impossible
- Extension via superclass / hook of block behavior is impossible



Long blocks are missed reuse opportunity

- Impossible to turn into a template and modify
 - Remember that *sending a message is a plan for reuse*
- Long blocks are a plague



Long blocks are missed reuse opportunity

Instead of

```
... display: [:v |  
  | tmp |  
  tmp := v size + 100.  
  v  
  foo;  
  bar;  
  more ]
```

Prefer

```
method: v  
  | tmp |  
  tmp := v size + 100.  
  v  
  foo;  
  bar;  
  more  
... display: [:v | xxx method: v ]
```

This way you can override `method:` in subclasses.



Long blocks are missed reuse opportunity

```
... painting: [ :view :file |
| tags |
tags := XMLDOMParser parse: file.
view shape label text: [:each | each
stringValue].
view nodes: tags.
view shape line color: (Color gray alpha
: 0.5).
view edges connectFromAll: [:aTag |
... ]]
```

```
paintOnView: view file: file
| tags |
tags := XMLDOMParser parse: file.
view shape label text: [:each | each
stringValue].
view nodes: tags.
view shape line color: (Color gray alpha
: 0.5).
view edges connectFromAll: [:aTag |
... ]]

... painting: [ :view :file | self
paintOnView: view file: file ]
```



Is not a little object more powerful than a block?

With an object you can

- Design an API
- Accumulate state
- Specify optional / obligatory inputs
- Support extension by construction



Conclusion

- When you use blocks, keep them as small as possible
- Use them to script DSLs but NOT to define your domain model
- Create classes and pass their instances around
- You will learn in the long run



Produced as part of the course on <http://www.fun-mooc.fr>

Advanced Object-Oriented Design and Development with Pharo

A course by

S.Ducasse, L. Fabresse, G. Polito, and P. Tesone



Inria
LearningLab



Except where otherwise noted, this work is licensed under CC BY-NC-ND 3.0 France
<https://creativecommons.org/licenses/by-nc-nd/3.0/fr/>