# Global to parameter

## Basic but important

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

http://www.pharo.org

## Goals

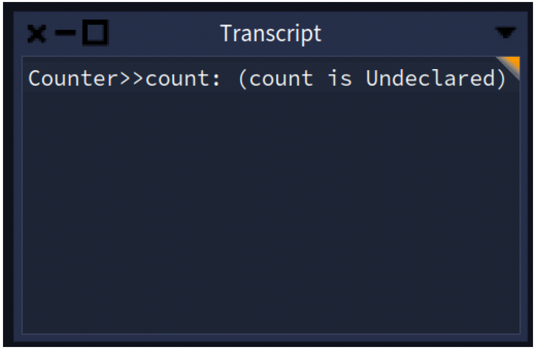
- Verify that globals are not a fatality
- Some can be turned into computation parameters (such as instance variables)
- Understand pros and cons

## Roadmap

- Example: Transcript usage
- Cure
- Stepping back
- Other analysis
- Related to Singleton Design Pattern plague

## The case: Transcript

Remember: Transcript is a global variable pointing to a log stream instance



```
Counter>>count: (count is Undeclared)
```

## Handy in development

```
myMethod
  Transcript show: 'foo' ; cr.
  self doSomething.
```

## The core of the problem on released soft

```
MicAbstractBlock >> iterate
  ...
  Transcript
    nextPutAll: 'Start ';
    nextPutAll: step asString;
    cr.
  ...
  Transcript
    nextPutAll: 'Stop ';
    nextPutAll: step asString;
    cr.
```

- What if I would like to have a specific log?
- What if we want to test that such logs are correct?

## Analysis

Some facts:

- You may not want the extra dependencies (such as Transcript) in your code
- Using Transcript, **your** log can be mixed with **other** logs
- You do not want to **dirty** build logs without a bit of control

Far worse and more important:

- You cannot reliably write tests to be sure that the log is correctly happening

## The solution: Use locality and encapsulation

- Think about object self-containment
- An object encapsulates a log stream
- Easy! Just add an instance variable to hold a stream

```
MicAbstractBlock >> initialize
  super initialize.
  logStream := WriteStream on: (String new: 1000)
```

- Use and write to THAT stream

```
MicAbstractBlock >> closeMe
  logStream << 'Closing ' << self class name; cr
```

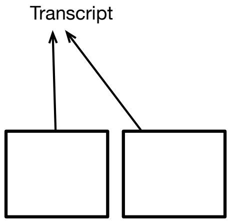
## Get the butter and the money

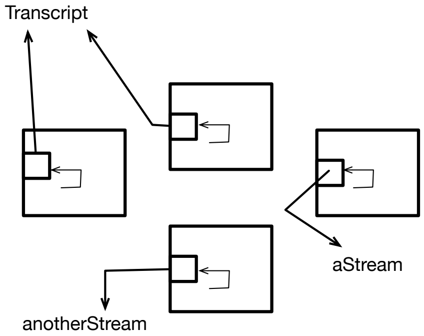
- Make sure that you can plug another stream as a logstream

```
MicAbstractBlock >> logStream: aStream
  logStream := aStream
```

- Now you can pass a Transcript and get the same as before but better.
- Bonus: You can write **tests in isolation**

## From monolithic to parametrizable





## Do you see the pattern?

```
RubScrollTextMorph >> defaultScrollTarget
  | textArea |
  textArea := self textAreaClass new.
  textArea backgroundColor: Color lightGray veryMuchLighter.
  ^ textArea
```

Why `Color lightGray veryMuchLighter` is hardcoded?

## A solution

Make it configurable!

```
RubScrollTextMorph >> defaultScrollTarget
  | textArea |
  textArea := self textAreaClass new.
  textArea backgroundColor: defaultBackgroundColor.
  ^ textArea
```

```
RubScrollTextMorph >> initialize
  defaultBackgroundColor := Color lightGray veryMuchLighter
```

## Supporting personalization

```
RubScrollTextMorph >> setBackgroundColor: aColor
  defaultBackgroundColor := aColor
```

Now each instance can have its specific value!

## Instance variables

- Instance variables are state of objects
- Instance variables are also **parameters** of your computation
- You can also share state with class scope variables (sharedVariables in Pharo)
- See lectures in Module **Sharing objects**

## About globals

Pros:

- You do not have to add an instance variable to your domain
- You do not have to initialize such global on your specific case

Cons:

- You have **only one** (e.g., if an entity belongs to one global model, you cannot have two entites living in different models)
- Testing requires care and is sometimes **not possible** or cumbersome because of **side effects**
- You cannot **initialize**, **specialize** the global for your context (there is only one)

## About parametrization

Sometimes you simply **cannot** add an instance variable to your objects

- Too many of them
- Fixed size inherited from old design
- About space consumption, check Lectures about Sharing and Flyweigth Design Pattern
- Factor the global usage to ease future changes

## In general: Avoid globals

- Avoid Singleton
- Avoid globals
- They make your code **less** modular, less **testable**
- Check lectures on Singleton and Disguised Singleton

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

2023