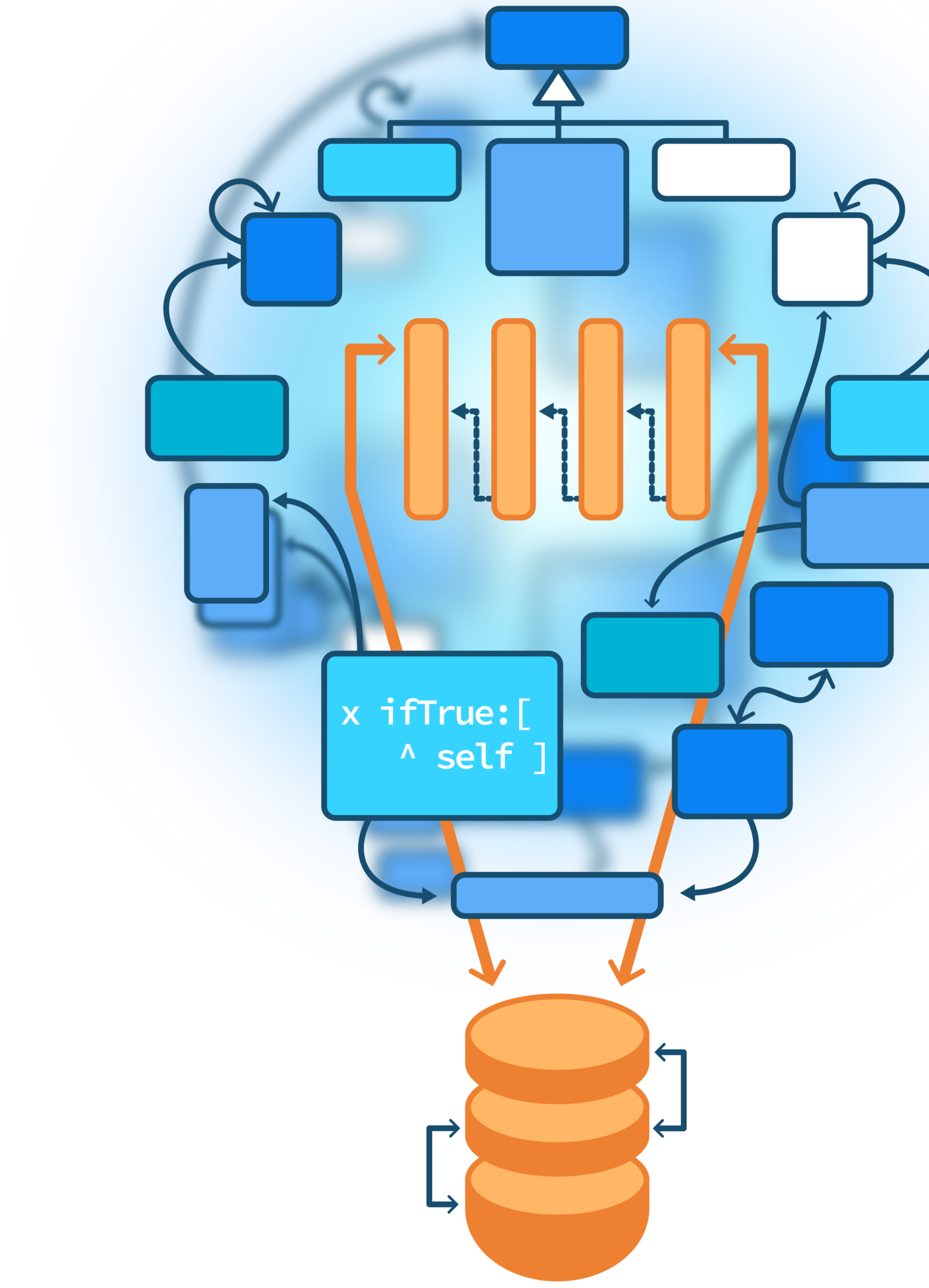
Advanced Object-Oriented Design

Essence of Dispatch

Taking Pharo Booleans as example

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



http://www.pharo.org

Objectives

- Understand of message passing (late binding) for **real** this time
- The **heart of Object-Oriented Design**
- Look at a beautiful implementation in Pharo

Context: Booleans

In Pharo, Booleans have a superb implementation!
You get the classical messages:

- &, |, not (eager)
- or:, and: (lazy)

And some less traditional ones:

- ifTrue:ifFalse:, ifFalse:ifTrue:
  - Yes, conditionals are messages sent to boolean objects

Three exercises

- Exo 1: Implement `not` (Not)
- Exo 2: Implement | (Or)
- Exo 3: What is the goal of these exercises?

Exercise 1: Implement Not

Propose an implementation of Not in a world where:

- You have: true, false objects
- You only have objects and messages

How would you implement the message not?

```
false not
-> true

true not
-> false
```

Hint 1: No conditionals

The solution does not use explicit conditionals (i.e., no `if`)

Hint 2: How do we express choices in OOP?

In OOP, the choice is expressed

- By defining classes with **compatible** methods
- By **sending** a message to an instance of such a class

Let the receiver decide!

Hint 2: An example of choice in OOP

```
x open
```

- x can be a file, a window, a tool,...
- The method is **selected** based on x's class

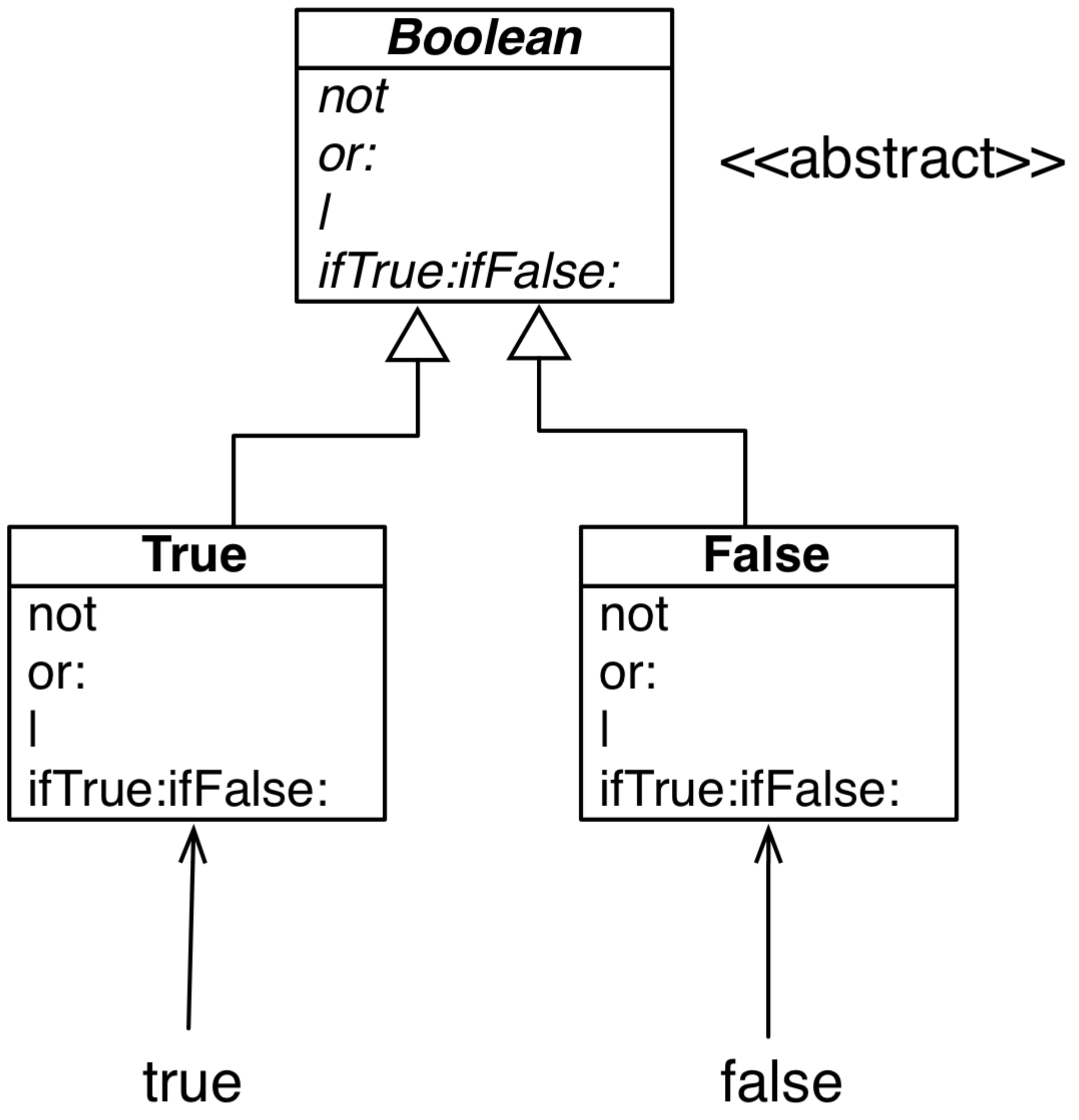
Hint 3: With at least two classes

- `true` is the singleton instance of the class `True`
- `false` is the singleton instance of the class `False`

The Pharo implementation uses three classes:

- The class `Boolean` (abstract), `True`, and `False`

Hint 3: With at least 2 classes and 2 methods



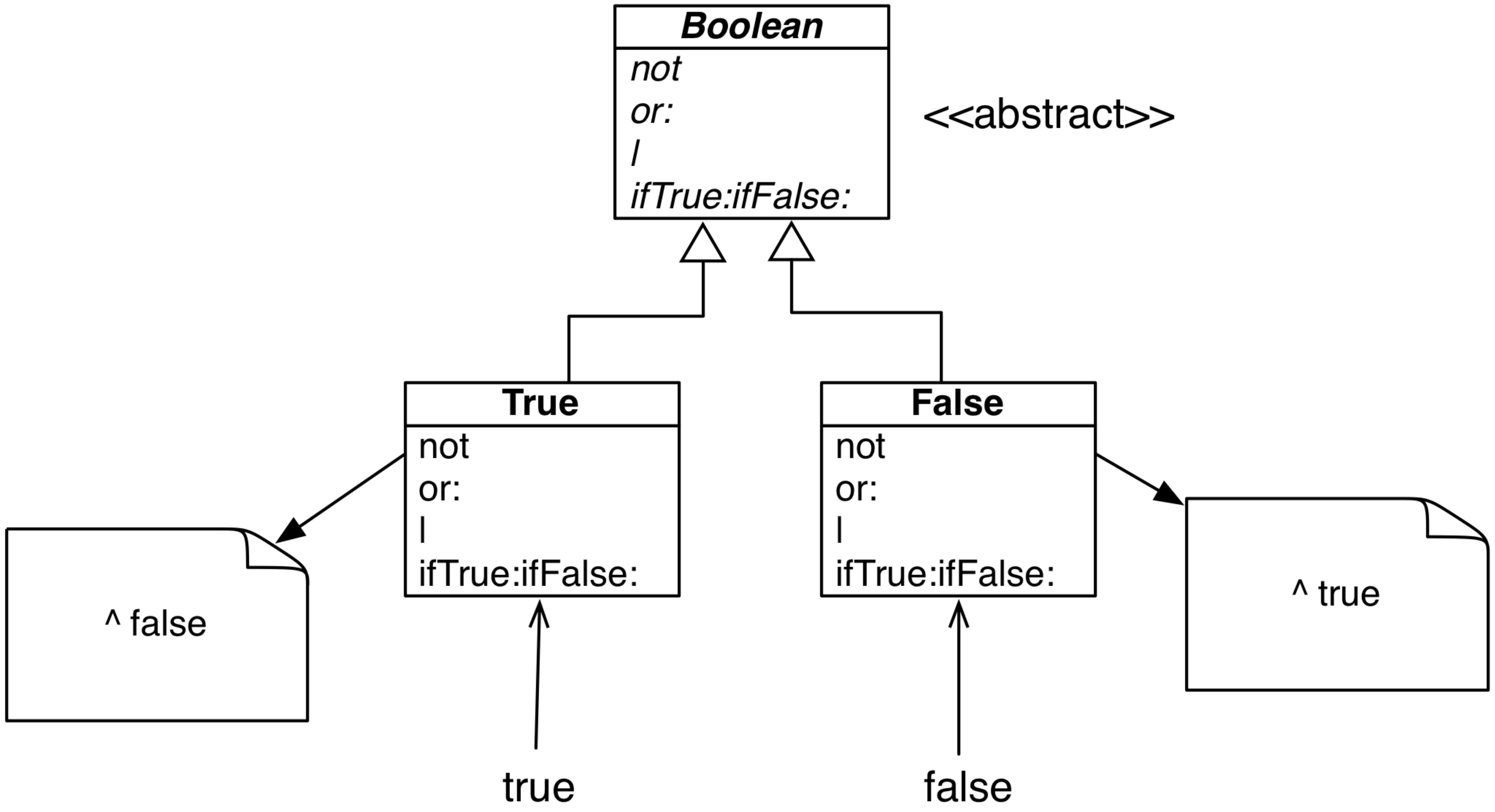
The class Boolean is not needed per se but it improves reuse

Implementation of Not in two methods

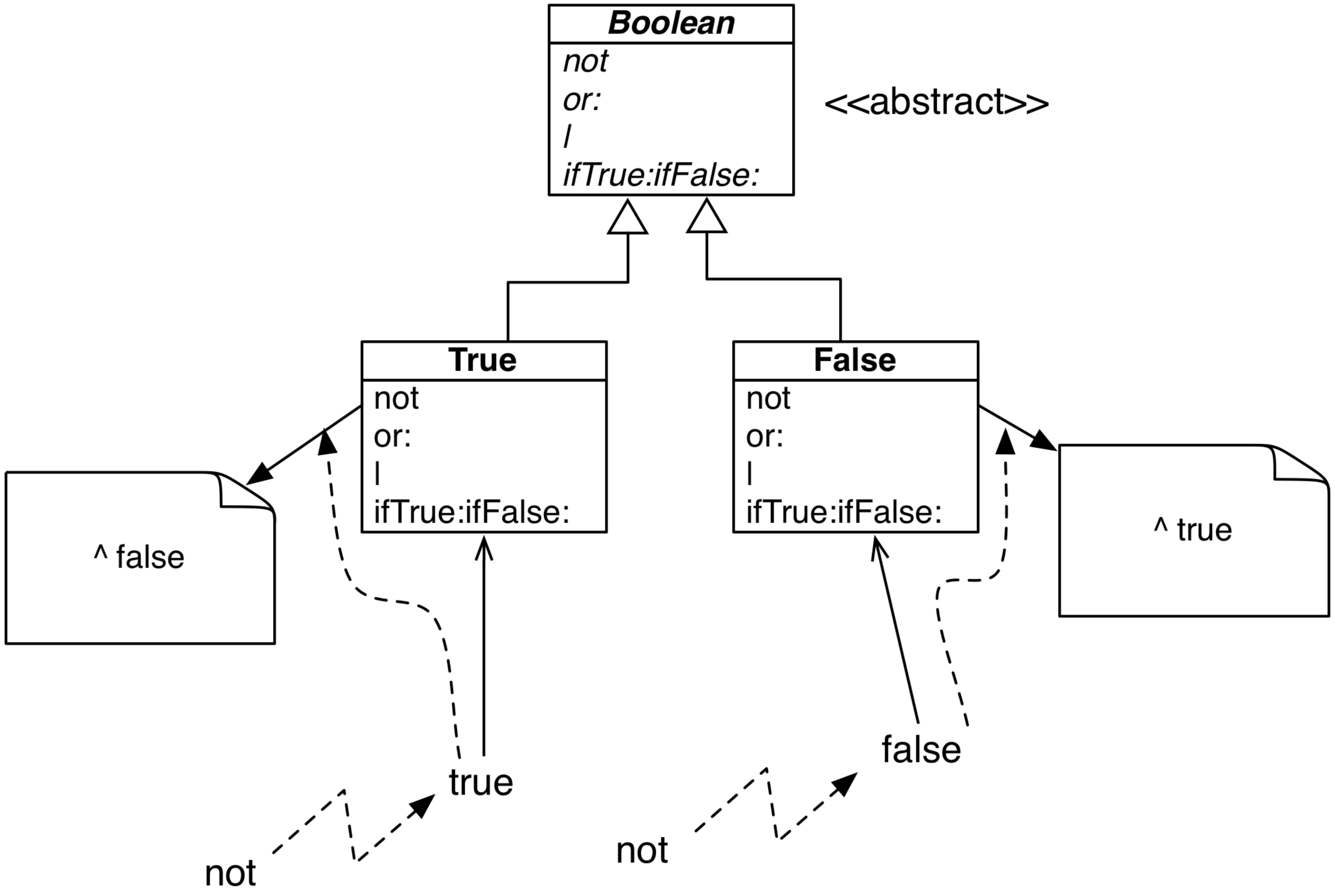
```
False >> not
  "Negation -- answer true since the receiver is false."
  ^ true
```

```
True >> not
  "Negation -- answer false since the receiver is true."
  ^ false
```

Implementation hierarchy



Message lookup chooses the right method



Boolean implementation

- The class Boolean is abstract
- The classes True and False implement
  - logical operations &, not
  - control structures and:, or:, ifTrue:, ifFalse:, ifTrue:ifFalse:, ifFalse:ifTrue:
  - reuse some logic from Boolean

Exercise 2: Implement Or

```
true | true -> true
true | false -> true
true | anything -> true

false | true -> true
false | false -> false
false | anything -> anything
```

Implementation of Or in Boolean

```
Boolean >> | aBoolean
  "Abstract method. Evaluating Or: Evaluate the argument.
  Answer true if either the receiver or the argument is true."
  self subclassResponsibility
```

Implementation of Or in class False

```
false | true -> true
false | false -> false
false | anything -> anything
```

Implementation of Or in class False

```
false | true -> true
false | false -> false
false | anything -> anything
```

```
False >> | aBoolean
  "Evaluating Or -- answer with the argument, aBoolean."
  ^ aBoolean
```

Implementation of Or in class True

```
true | true -> true
true | false -> true
true | anything -> true
```

Implementation of Or in class True

```
true | true -> true
true | false -> true
true | anything -> true
```

```
True >> | aBoolean
  "Evaluating Or -- answer true since the receiver is true."
  ^ true
```

Real implementation of Or in class True

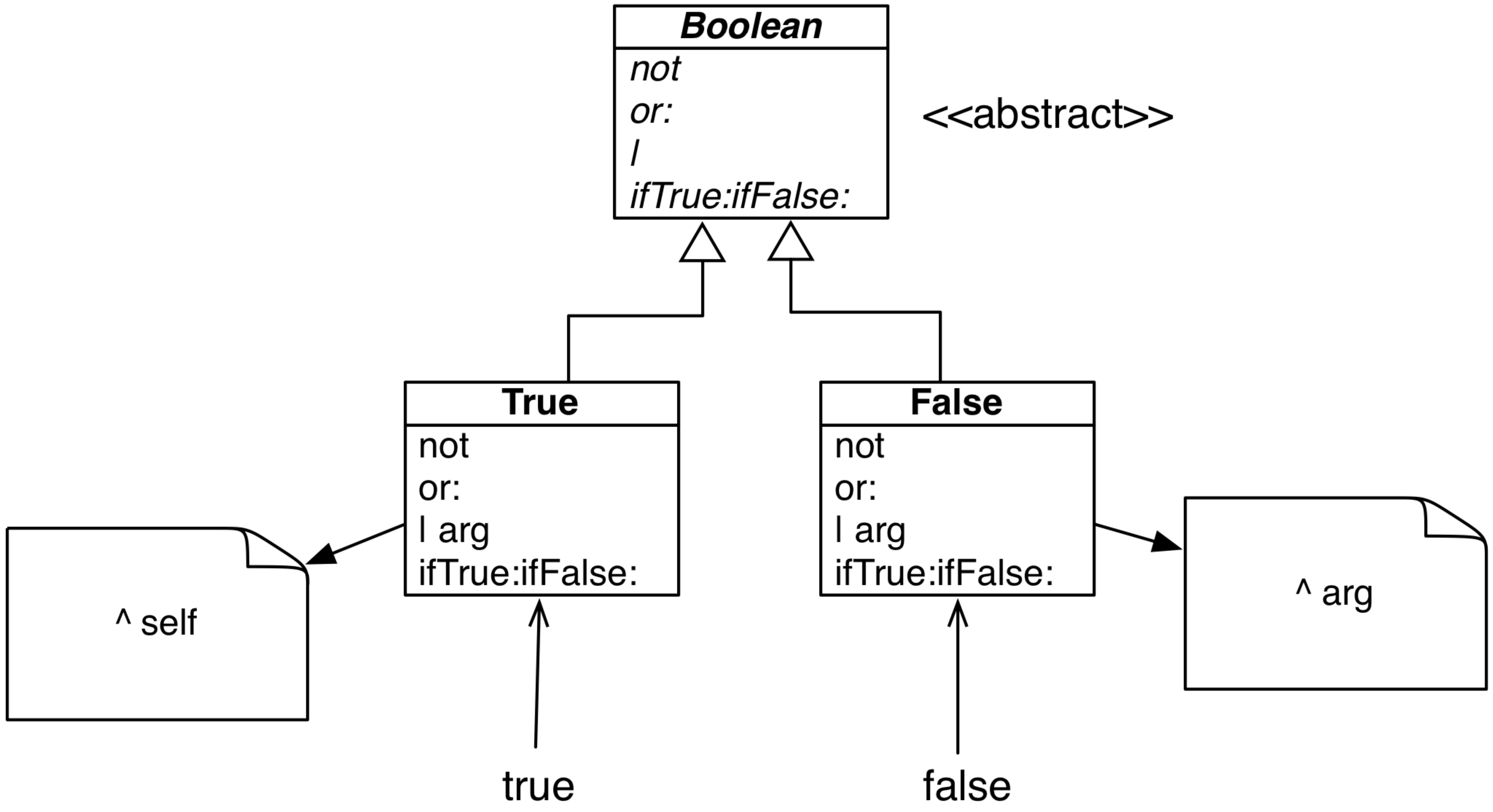
The object true is the receiver of the message!

```
True>> | aBoolean
  "Evaluating disjunction (Or) -- answer true since the receiver is true."
  ^ true
```

So we can write it like the following:

```
True >> | aBoolean
  "Evaluating disjunction (Or) -- answer true since the receiver is true."
  ^ self
```

Or Implementation in two methods



Step back

- An example of **Do not ask, tell** principle application
- Here:
  - We **delegate** to the correct `Boolean` **object**
  - Each subclass implements its **own** logic

Summary

We saw:

- The solution to implement boolean operations does NOT use explicit conditionals (if)
- **Sending a message is making a choice**

Remember two important principles

- **Do not ask, tell**
- **Let the receiver decide**

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

IMT Nord Europe
École Mines-Télécom
IMT-Université de Lille

2023