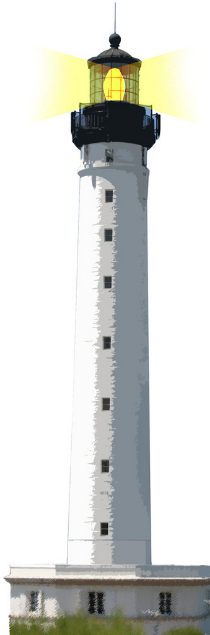
Advanced Object-Oriented Design

Thinking about Coupling

S. Ducasse

http://www.pharo.org



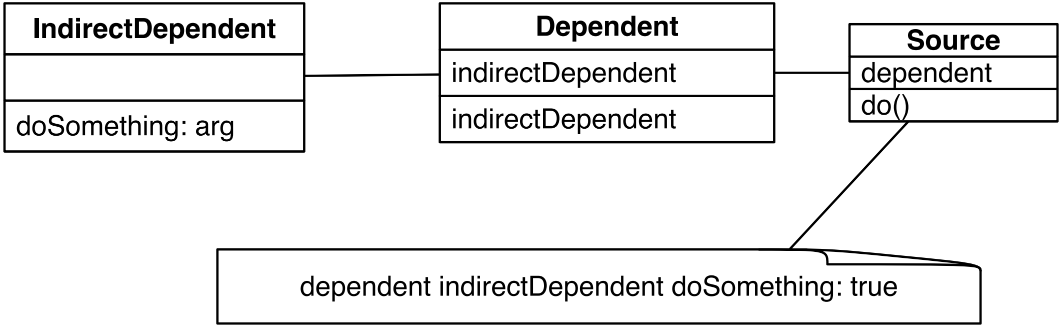
Goal

- Think about Coupling
- Law of Demeter
- Move Behavior close to Data
- Better seen as a Heuristic

Symptoms of costly coupling

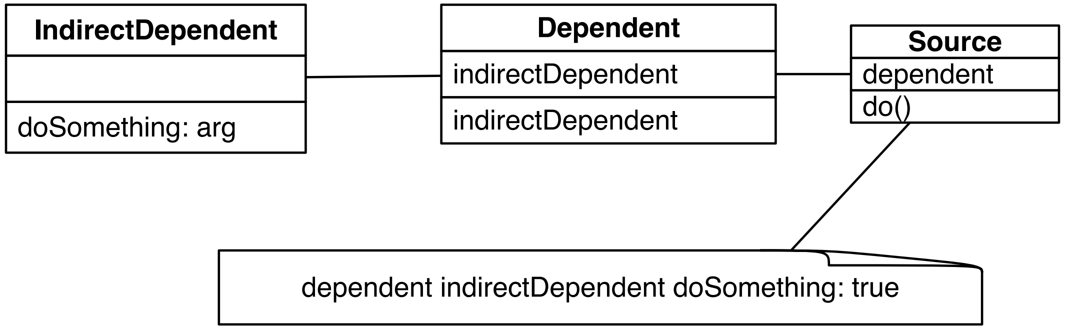
- **Reuse:** I cannot reuse this component in another application
- **Substition:** I cannot easily substitute this part for another one
- **Encapsulation:** when a change far away happens, I get impacted
- **Untestable:** I cannot test this part

Core of the Problem



Changes

- Changes are natural
- When you change, your dependents should update
- The problem is **waves of changes when dependents of dependents should change**



Waves

- Waves are created by leaks of references to far objects
- Basically violation of encapsulation
- How to limit wave creation?
- Do not leak far references!

Law of Demeter

You should only send messages to:

- an argument passed to you
- instance variables
- an object you create
- self, super your class

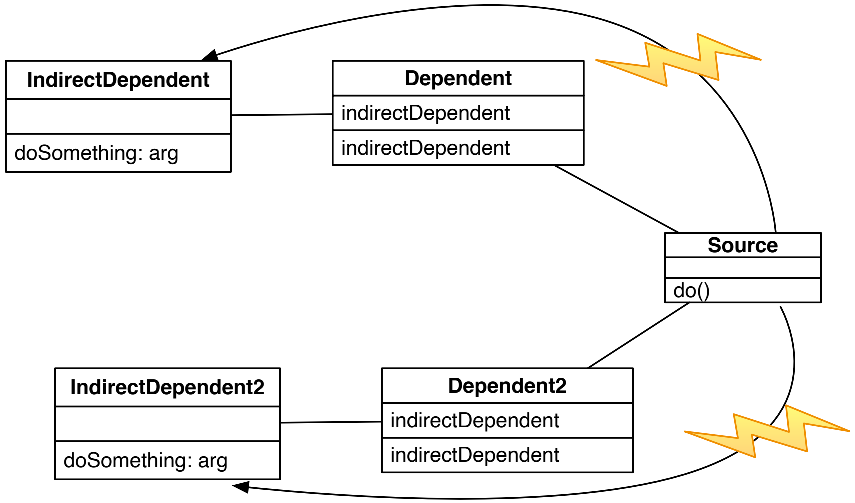
You should avoid

- global variables
- objects returned from message sends other than self

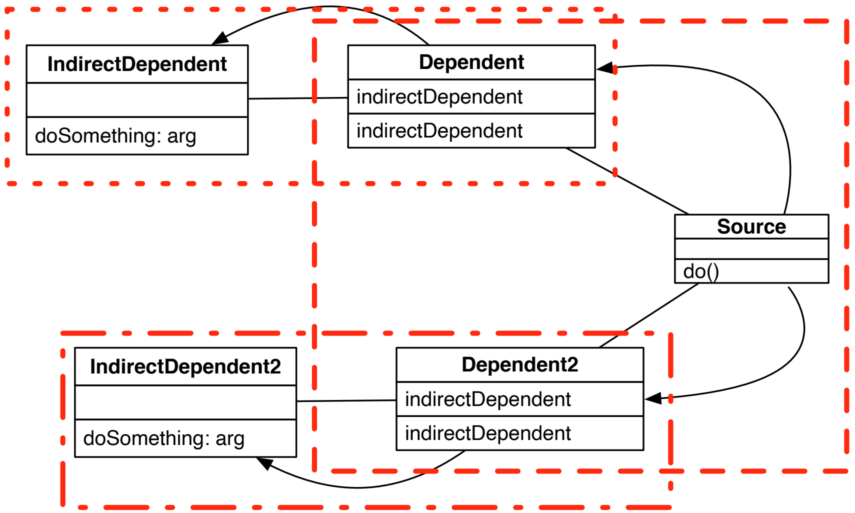
Only talk to your immediate friends

```
someMethod: aParameter
  self foo.
  super someMethod: aParameter.
  self class foo.
  self instVarOne foo.
  instVarOne foo.
  aParameter foo.
  thing := Thing new.
  thing foo
```

Don’t skip your intermediates



Solution: Respect encapsulation

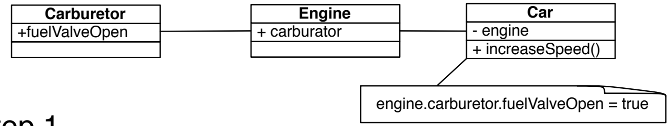


Move behavior close to data

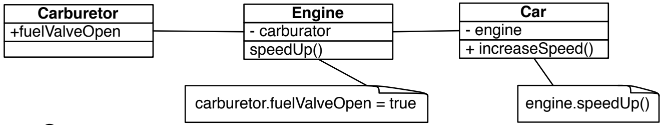
An object-oriented reengineering pattern (check the book)

- if data and behavior are not close
- then logic is distributed/duplicated in clients!

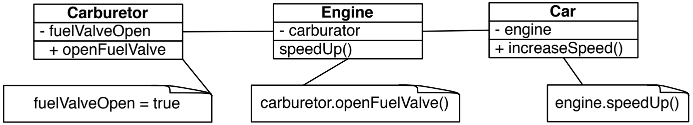
Move behavior close to data: Transformation



Step 1



Step 2



Real example

```
OSWindowMorphicEventHandler >> visitWindowResolutionChangeEvent: anEvent
  "Resolution (dpi) changed. For now just check for a new size."
  "We need to reset the render if the resolution changes."

  morphicWorld worldState worldRenderer window backendWindow renderer destroy.
  morphicWorld worldState worldRenderer window backendWindow renderer validate.
  morphicWorld worldState doFullRepaint.
  morphicWorld worldState worldRenderer window backendWindow renderer
      updateAll.
  morphicWorld worldState worldRenderer checkForNewScreenSize
```

Solution

```
OSWindowMorphicEventHandler >> visitWindowResolutionChangeEvent: anEvent
  morphicWorld worldState updateToNewResolution: anEvent
```

```
WorldState >> updateToNewResolution: originalEvent
  "We need to reset the render if the resolution changes."

  self doFullRepaint.
  self worldRenderer updateToNewResolution.
  self worldRenderer checkForNewScreenSize
```

```
OSSDL2BackendWindow >> updateToNewResolution
  "Force the regeneration of the renderer because we have a new resolution"
  renderer destroy.
  renderer validate.
  renderer updateAll.
```

```
NullWorldRenderer >> updateToNewResolution
  self
```

LOD is a **heuristic**

- Pay attention! A too strict application of the LOD can lead to over engineered design
- Encapsulating collections may produce large interfaces so not applying the LoD may help
- Understand when it is reasonable to leak

LOD can produce bloated APIs

```
Object subclass: #FMMethods
  instVar: 'senders'
  ...
```

```
FMMethods >> do: aBlock
  senders do: aBlock
FMMethods >> collect: aBlock
  ^ senders collect: aBlock
FMMethods >> select: aBlock
  ^ senders select: aBlock
FMMethods >> detect: aBlock
  ^ senders detect: aBlock
FMMethods >> isEmpty
  ^ senders isEmpty
...
```

Conclusion

- Think about impact of changes
- Avoid chaining messages
- Law of Demeter is a heuristic
- Move behavior close to data

A course by

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