Design Points - Law of Demeter

Stéphane Ducasse
stephane.ducasse@inria.fr
http://stephane.ducasse.free.fr/
Stéphane Ducasse --- 2005

About Coupling
- Why coupled classes is fragile design?
- Law of Demeter
- Thoughts about accessor use

The Law of Demeter
You should only send messages to:
- an argument passed to you
- instance variables
- an object you create
- self, super
- your class
- Avoid global variables
- Avoid objects returned from message sends other than self

Correct Messages
someMethod: aParameter
- self foo.
- super someMethod: aParameter.
- self class foo.
- self instVarOne foo.
- instVarOne foo.
- aParameter foo.
- thing := Thing new.
- thing foo

In other words
- Only talk to your immediate friends.
- In other words:
  - You can play with yourself. (this.method())
  - You can play with your own toys (but you can't take them apart). (field.method(), field.getX())
  - You can play with toys that were given to you. (arg.method())
  - And you can play with toys you've made yourself. (A = new A(); A.method())

Halt!

To not skip your intermediate

Solution

OO Following of LoD
Law of Demeter’s Dark Side

Class A

```smaller
instVar: myCollection
```

A>>do: aBlock
myCollection do: aBlock
A>>collect: aBlock
^ myCollection collect: aBlock
A>>select: aBlock
^ myCollection select: aBlock
A>>detect: aBlock
^ myCollection detect: aBlock
A>>isEmpty

Accessors are good for lazy initialization

Scheduler>>tasks
^ tasks

BUT accessors methods should be Protected by default at least at the beginning

About the Use of Accessors

Some schools say: “Access instance variables using methods”

But

Be consistent inside a class, do not mix direct access and accessor use
First think accessors as protected methods that should not be invoked by clients
Only when necessary put accessors in accessing protocol

Accessors open Encapsulation

The fact that accessors are methods doesn’t support a good data encapsulation.
You could be tempted to write in a client:

```smaller
ScheduledView>>addTaskButton
...
model addTask: newTask
```

What’s happen if we change the representation of tasks?

Use intention revealing names

Better

Scheduler>>taskCopy or copiedTasks
^ task copy
Provide a Complete Interface

Workstation>>accept: aPacket
   aPacket addressee = self name
   ... 
   It is the responsibility of an object to offer a complete interface that protects itself from client intrusion.
   Shift the responsibility to the Packet object
   Packet>>isAddressedTo: aNode
      ^ addressee = aNode name
   Workstation>>accept: aPacket