CS 535 Object-Oriented Programming & Design
Fall Semester, 2011
Doc 15 Some OO Advice
Oct 17 2011

References

Object-Oriented Design Heuristics, Riel

Principles of OO Design, or Everything I Know About Programming, I Learned from Dilbert, http://alanknightsblog.blogspot.com/2011/10/principles-of-oo-design-or-everything-i.html

Smalltalk Best Practice Patterns, Beck, Prentice Hall,1997

Principles of OO Design, or Everything I Know About Programming, I Learned from Dilbert

Alan Knight

1. Never do any work that you can get someone else to do for you

Example 1 Total of bills that have been paid this quarter for a factory

```
total := 0
aFactory billings do: [:each |
   (each status == #paid and: [each date > startDate])
      ifTrue: [total := total + each amount]].
```

versus

```
total := aPlant totalBillingsPaidSince: startDate.
```

1. Never do any work that you can get someone else to do for you

Excuse me Smithers. I need to know the total bills that have been paid so far this quarter. No, don't trouble yourself. If you'll just lend me the key to your filing cabinet I'll go through the records myself. I'm not that familiar with your filing system, but how complicated can it be? I'll try not to make too much of a mess.

Verses

SMITHERS! I need the total bills that have been paid since the beginning of the quarter. No, I'm not interested in the petty details of your filing system. I want that total, and I'll expect it on my desk within the next half millisecond.

1. Never do any work that you can get someone else to do for you

```
somebody clients add: Client new.
```

Less work

somebody just returns collection

verses

```
somebody addClient: Client new.
```

Needs

addClient:
removeClient:
more?

Encapsulation & Responsibility

Encapsulation is about responsibility

Who does the work

Who should do the work

2. Avoid Responsibilty

If you must accept a responsibility, keep it as vague as possible.

For any responsibility you accept, try to pass the real work off to somebody else.

```
Stack>>do: aBlock
    topOfStack do: aBlock
```

Kent Beck's Properties of Good Style

Kent Beck's Properties of Good Code Stype

Once and only once

Lots of little pieces

Replacing objects

Moving Objects

Rates of change

Once and Only Once

"In a program written with good style, everything is said once and only once"

If have

- several methods with same logic
- several objects with same methods

then rule is not satisfied

Lots of little pieces

"Good code invariably has small methods and small objects"

Small pieces allow you to satisfy "once and only once"

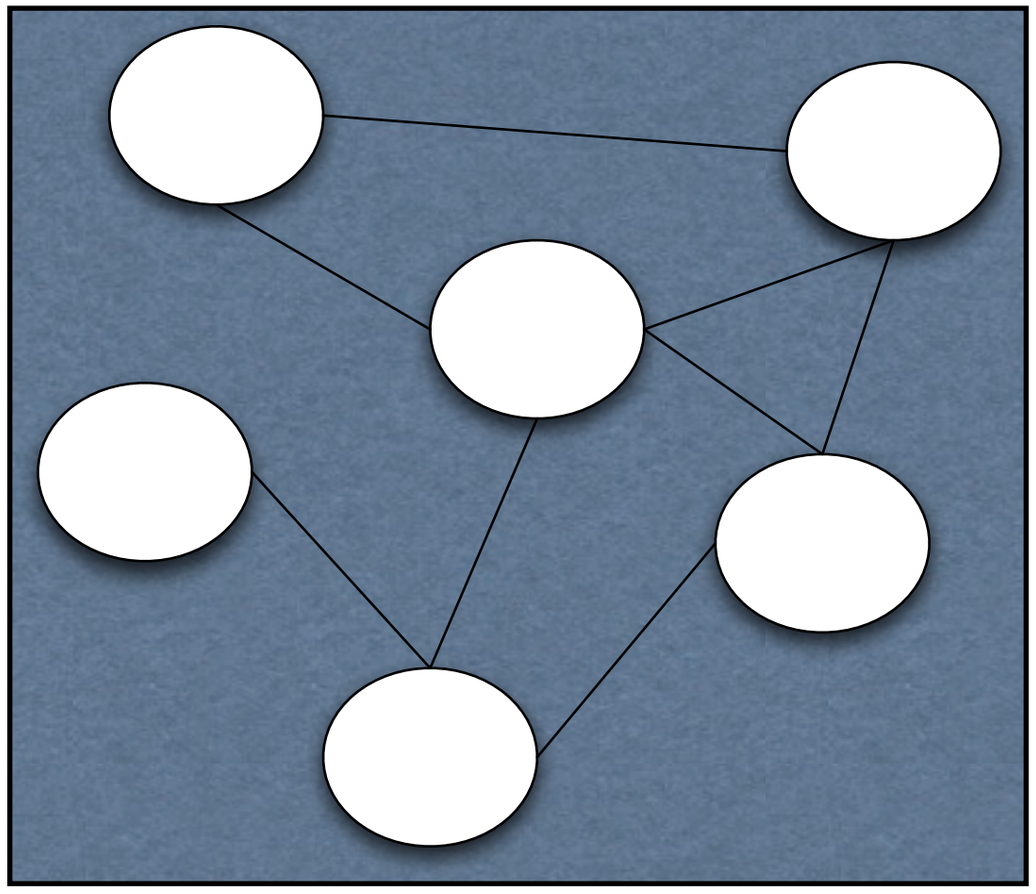
Replacing objects

Good style leads to easily replaceable objects

When you can extend a system soley by adding new objects without modifying exisiting objects, then you have system that is fexible and cheap to maintain

Needs lots of little pieces

Some heuristics



OrderedCollection
String
Dictionary
Characters
Streams
Trolls
etc.

Adventure open

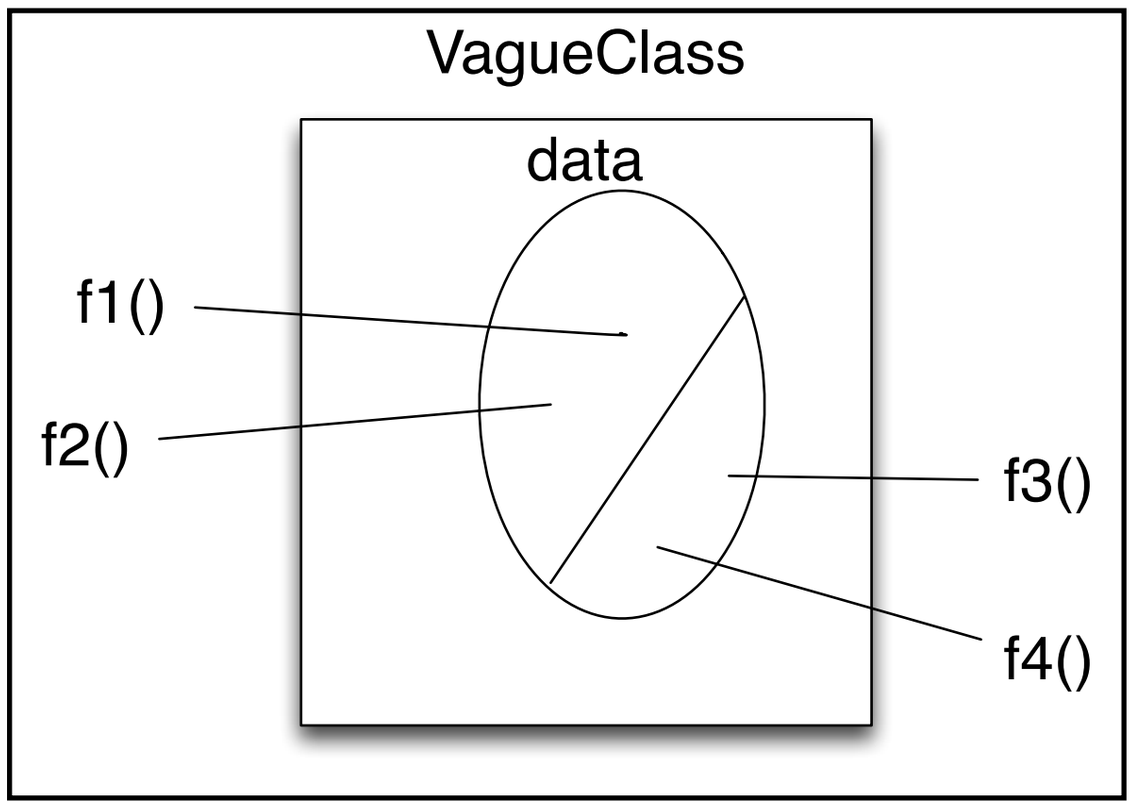
What does "main" in a program do? Think of a GUI application. Where is the main there?

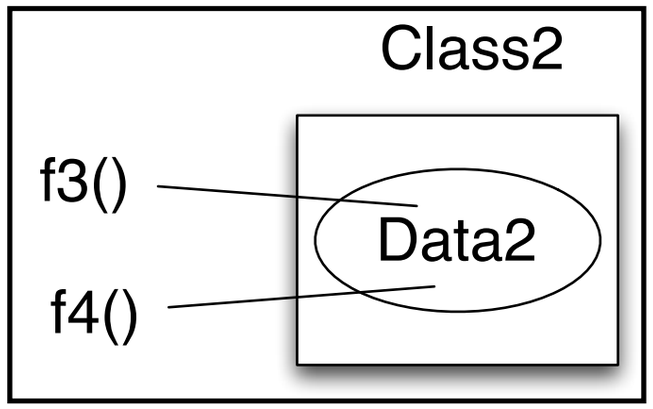
2.8 A class should capture one and only one key abstraction

This is the most important idea in OO

To perform an operation send a message to the object that contains the data

Spin off nonrelated information into another class



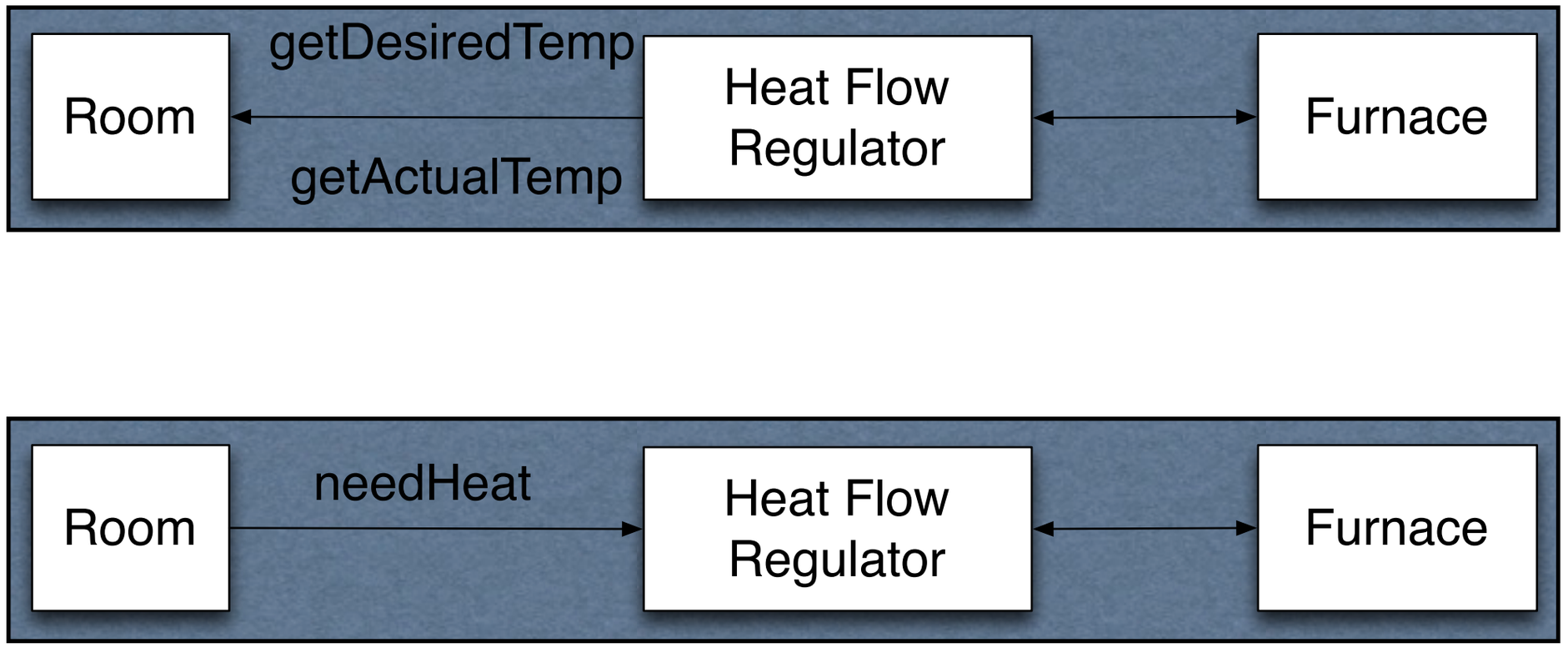


God object is an object that knows too much or does too much

Behavioral Form

Replaces the main
Does too much

Definition is from Wikipedia


Room
getDesiredTemp
getActualTemp
Heat Flow
Regulator
Furnace

Room
needHeat
Heat Flow
Regulator
Furnace

Distribute system intelligence horizontally as uniform as possible

Do not create god classes/objects
Be very suspicous of a class whose name contains Driver, Manager, System

Beware of classes that have many accessor methods defined in there public interface

Beware of classes that have too much noncommunicating behavoir

Model should not depend on the interface
The interface should depend on the model

So interface needs to access data in the model