SOLID Ruby

Design Principles for a Dynamic Language

Jim Weirich
Chief Scientist / EdgeCase
jim@edgecase.com
@jimweirich

EdgeCase
software artisans

http://github.com/jimweirich/presentation_solid_ruby

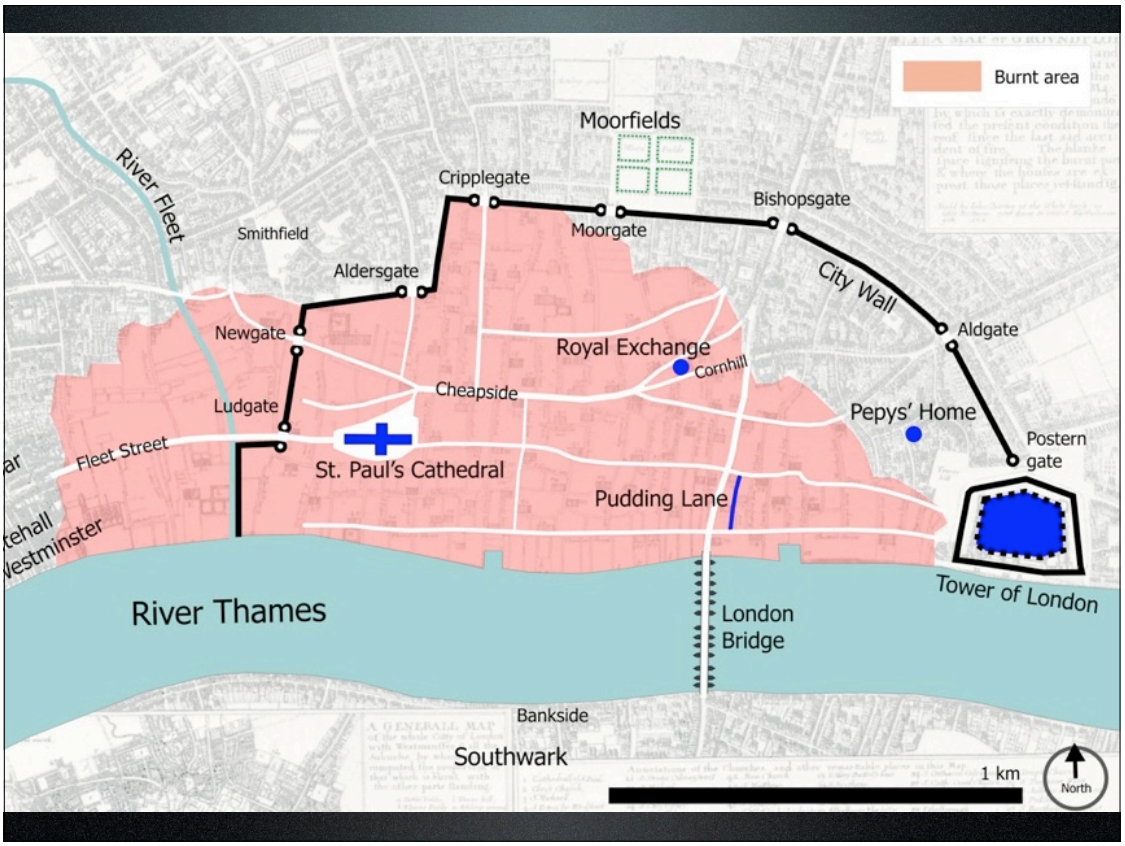
How do you recognized a good design?

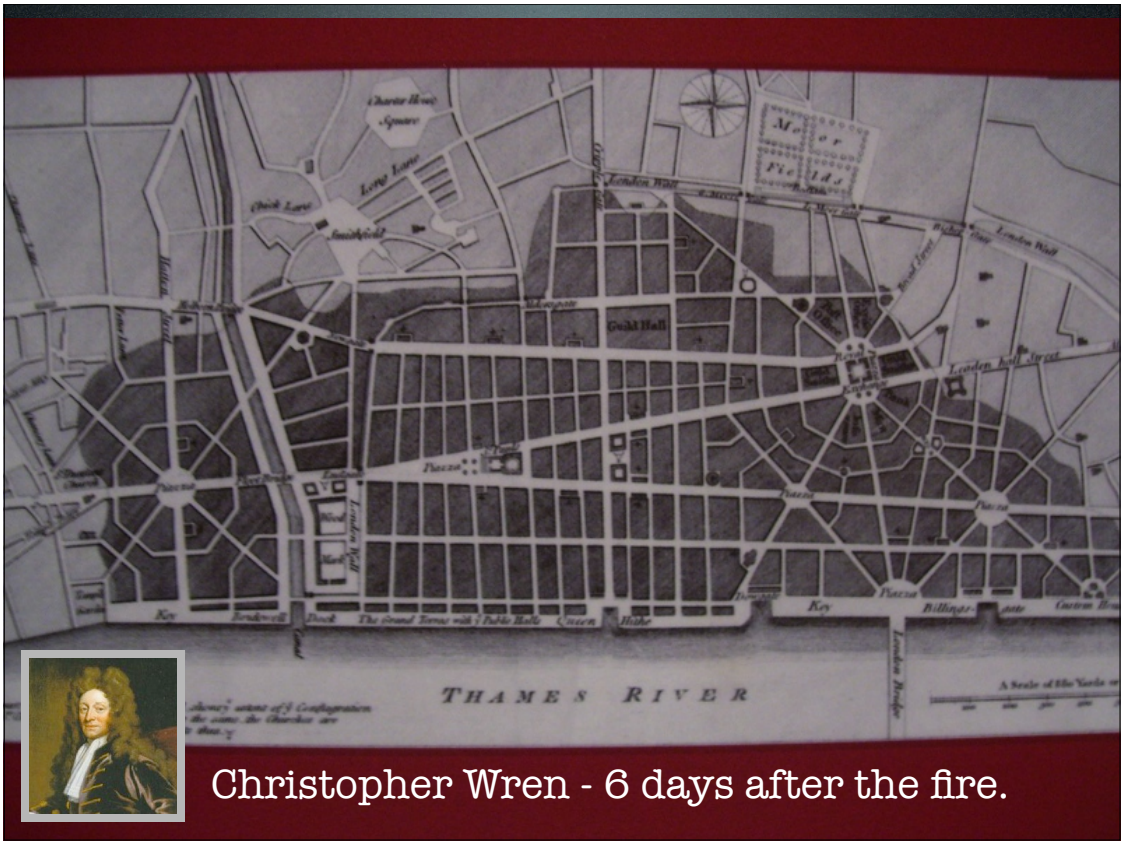
Huh?


from failblog.org

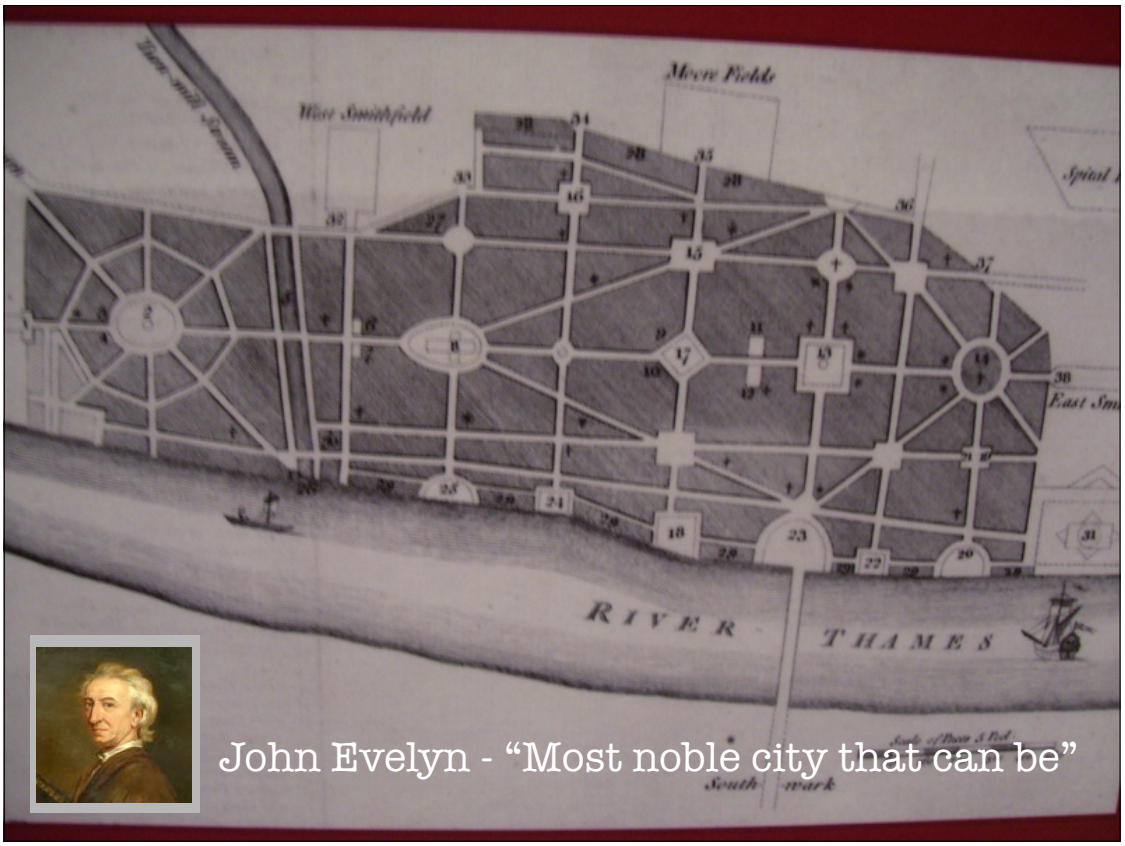

London
Pudding Lane
Sep 2, 1666

Great Fire of London

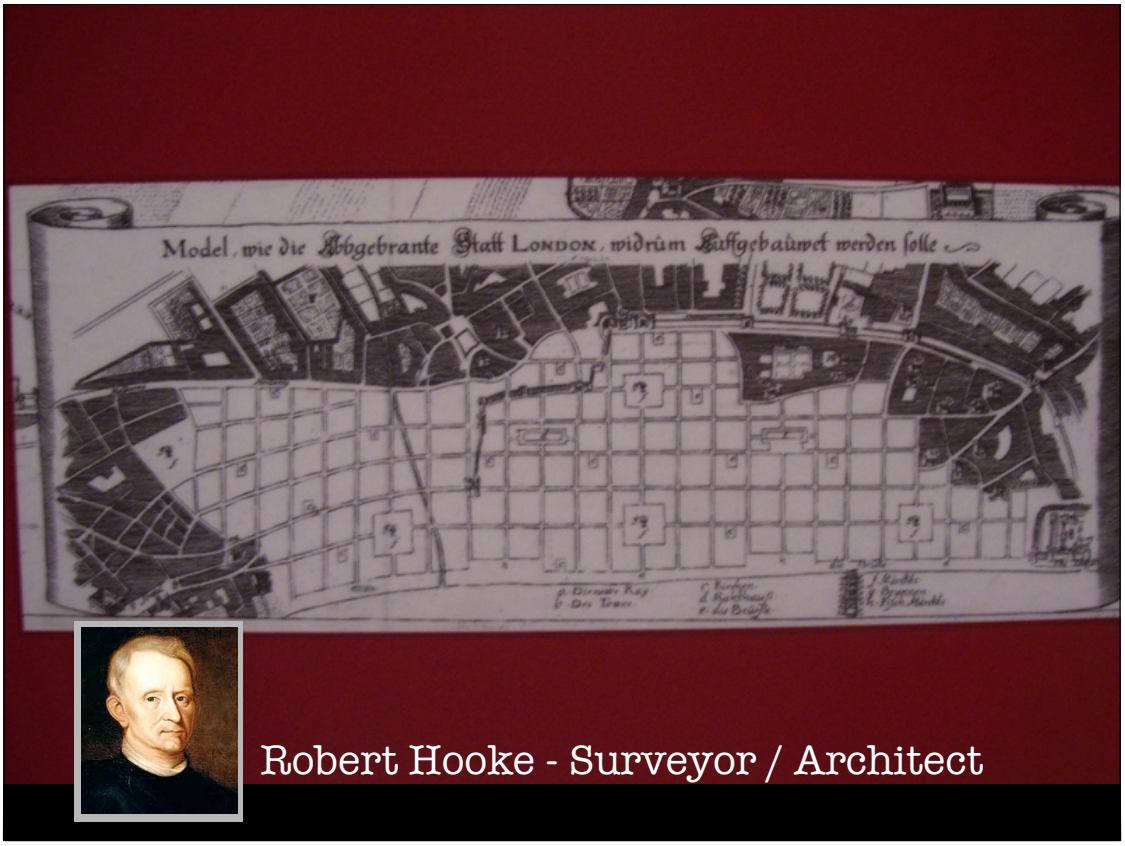




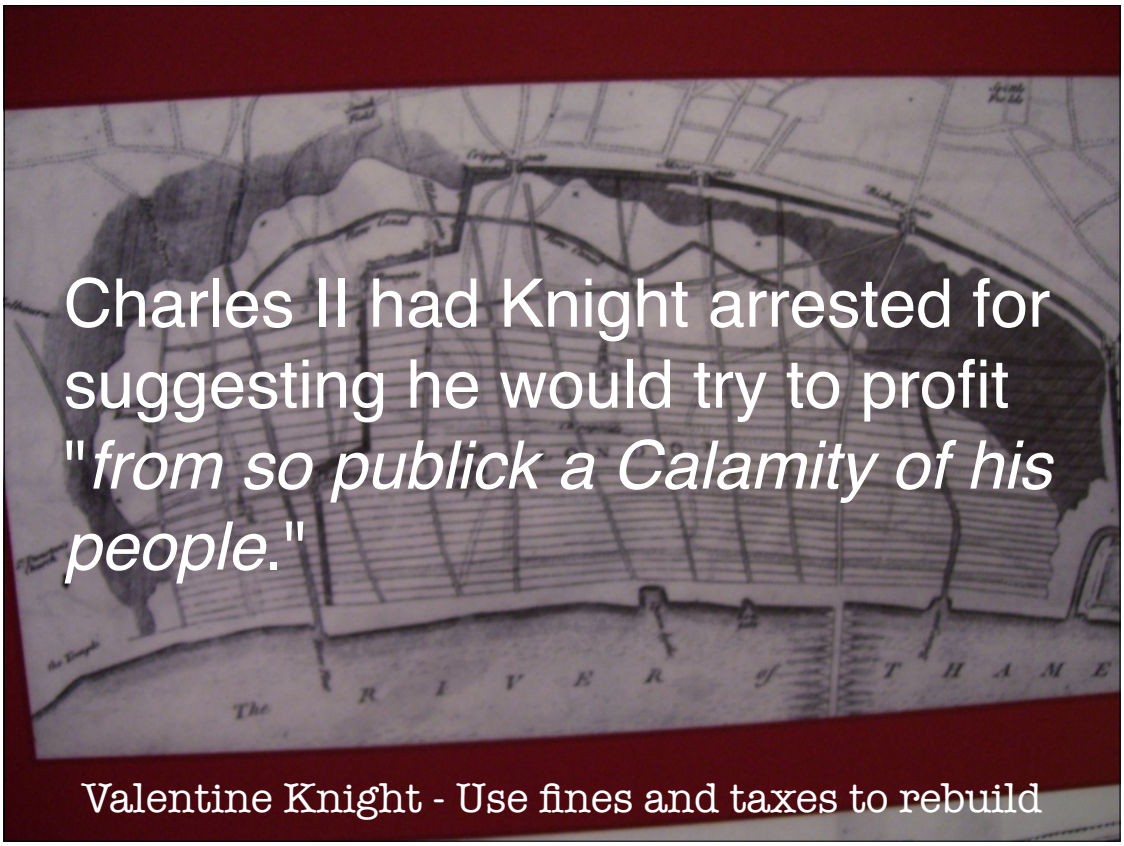
Christopher Wren - 6 days after the fire.

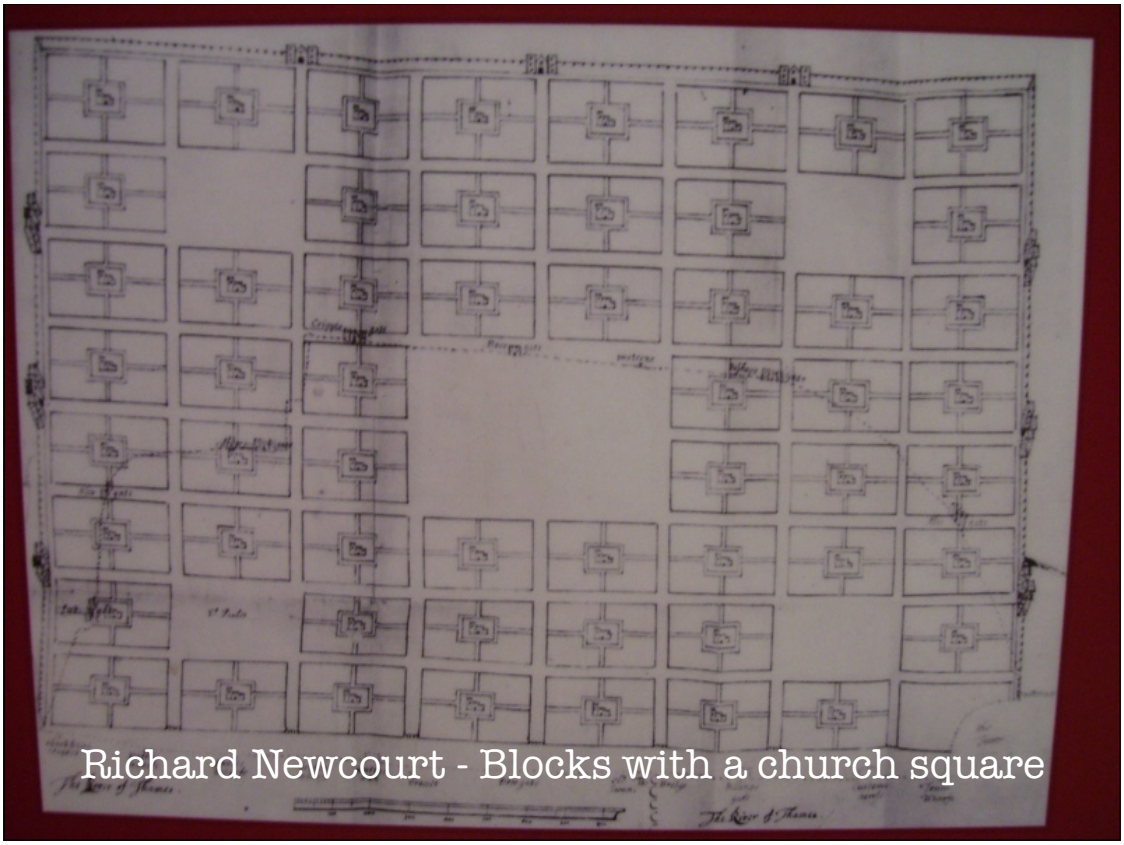


John Evelyn - “Most noble city that can be”



Robert Hooke - Surveyor / Architect


Charles II had Knight arrested for suggesting he would try to profit "*from so publick a Calamity of his people*."
Valentine Knight - Use fines and taxes to rebuild

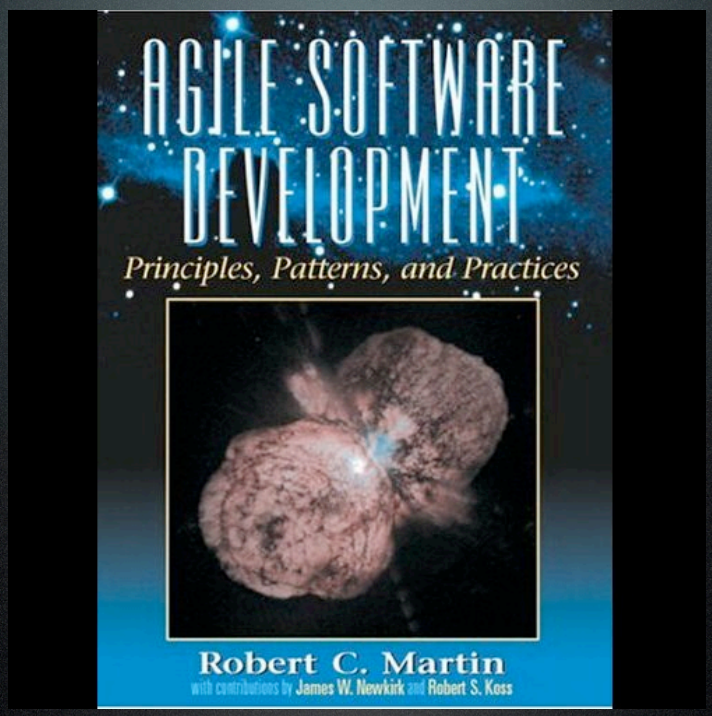

Richard Newcourt - Blocks with a church square

Which was chosen?

None of them!

Rebuilt Incrementally

First Agile Project



- Single Responsibility Principle
- Open/Closed Principle
- Liskov Substitution Principle
- Interface Segregation Principle
- Dependency Inversion Principle

- Single Responsibility Principle
- Open/Closed Principle
- Liskov Substitution Principle
- Interface Segregation Principle
- Dependency Inversion Principle

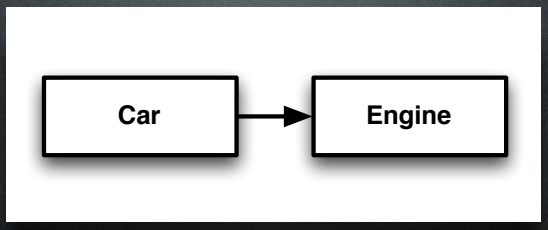
SOLID

Software Development is not a Jenga game

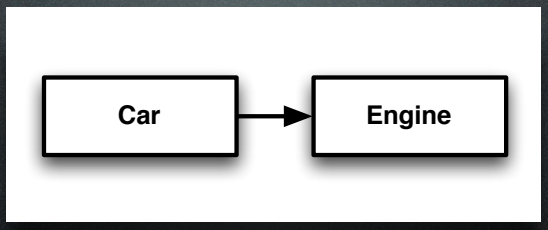
http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod

Dependencies

Cars depend on Engines

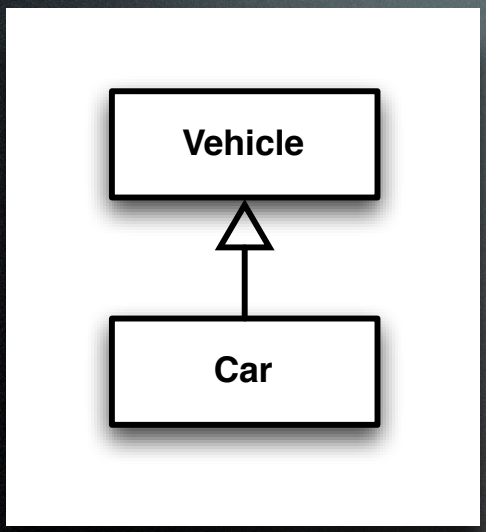


Cars depend on Engines



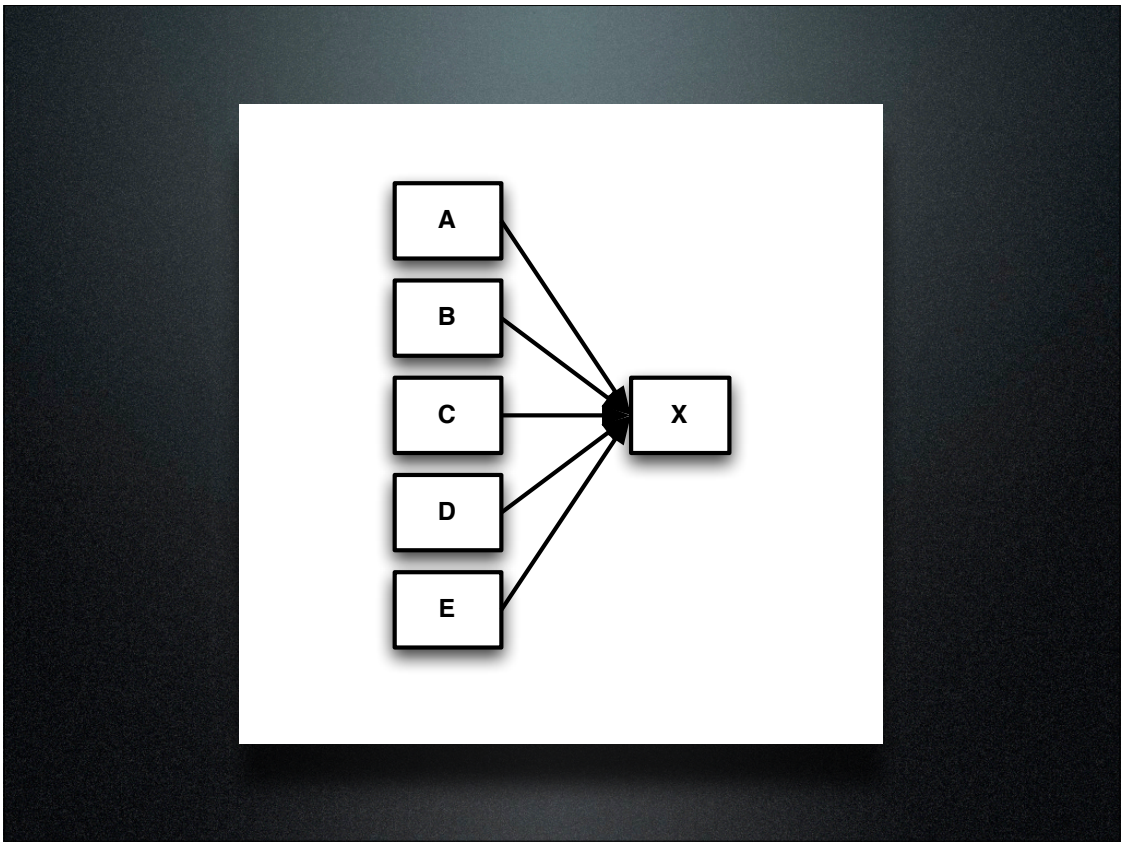
Car objects are allowed to call methods on engine objects.

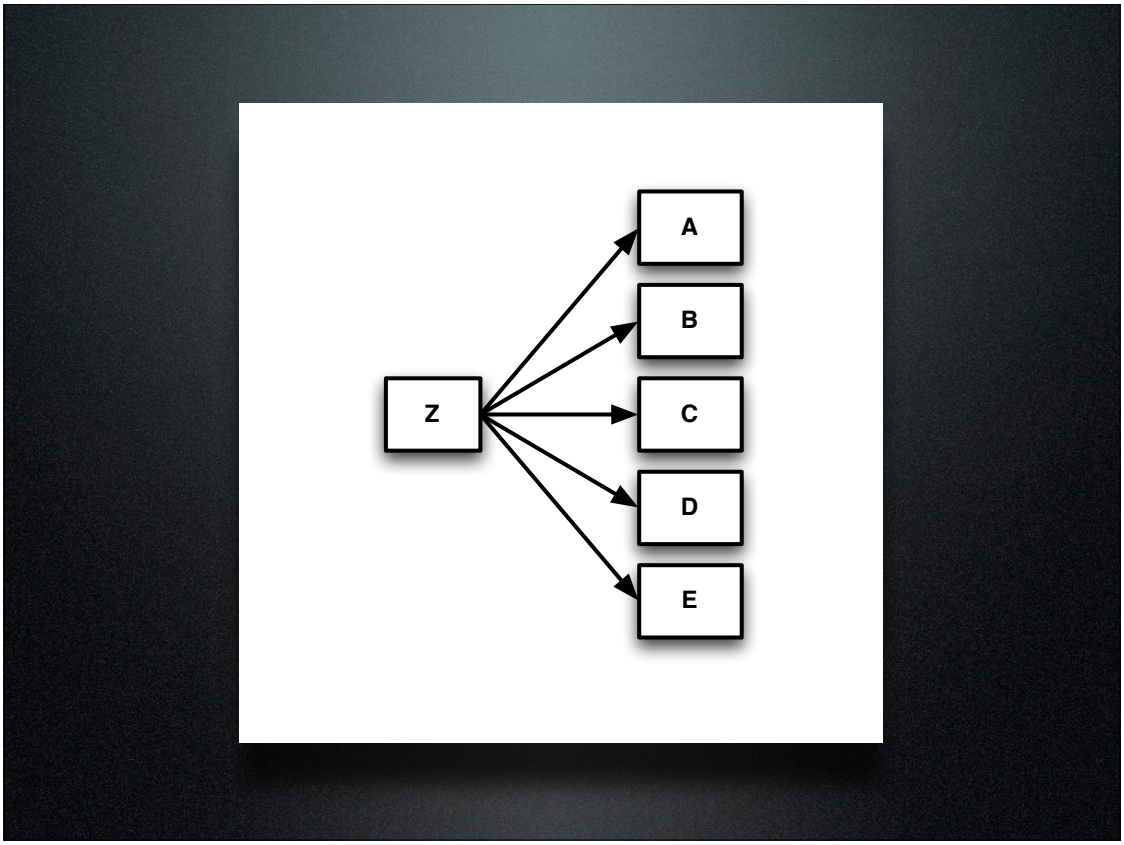
Car depends on Vehicle

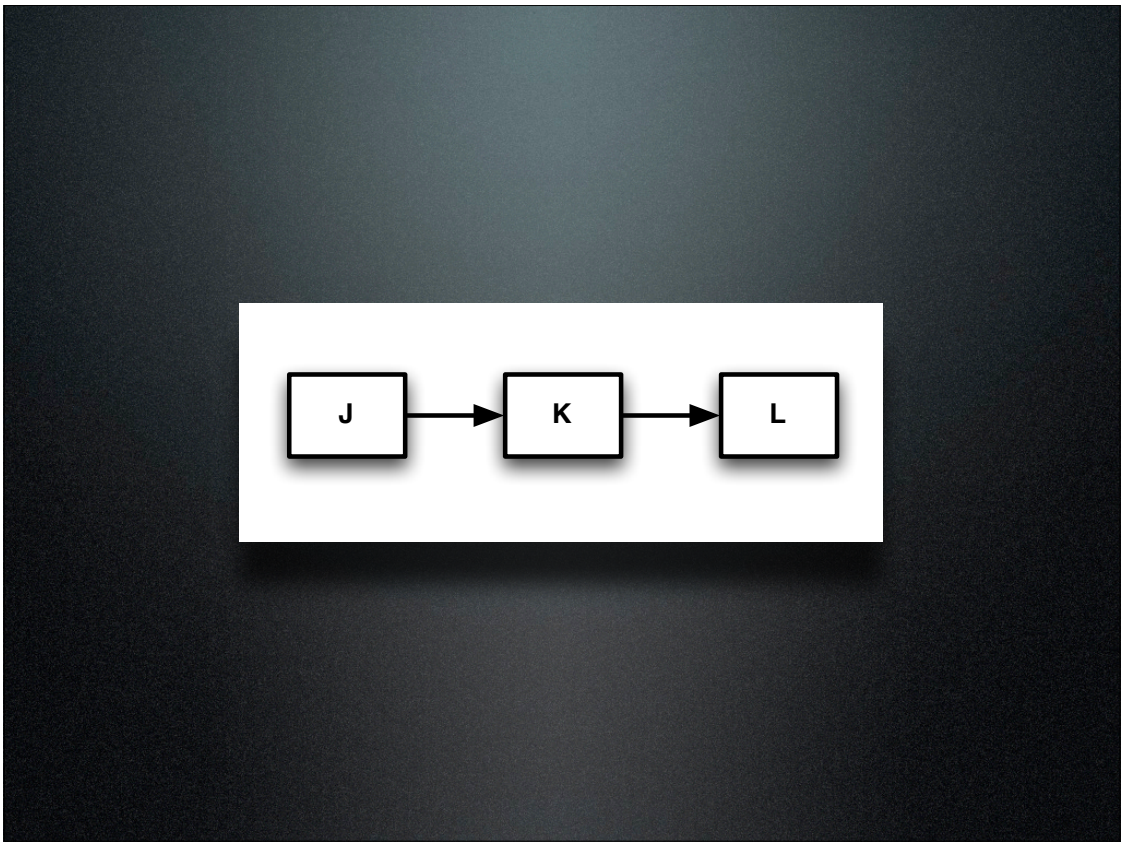


- The Car class inherits from Vehicle
- Cars depend on implementation provided by Vehicle

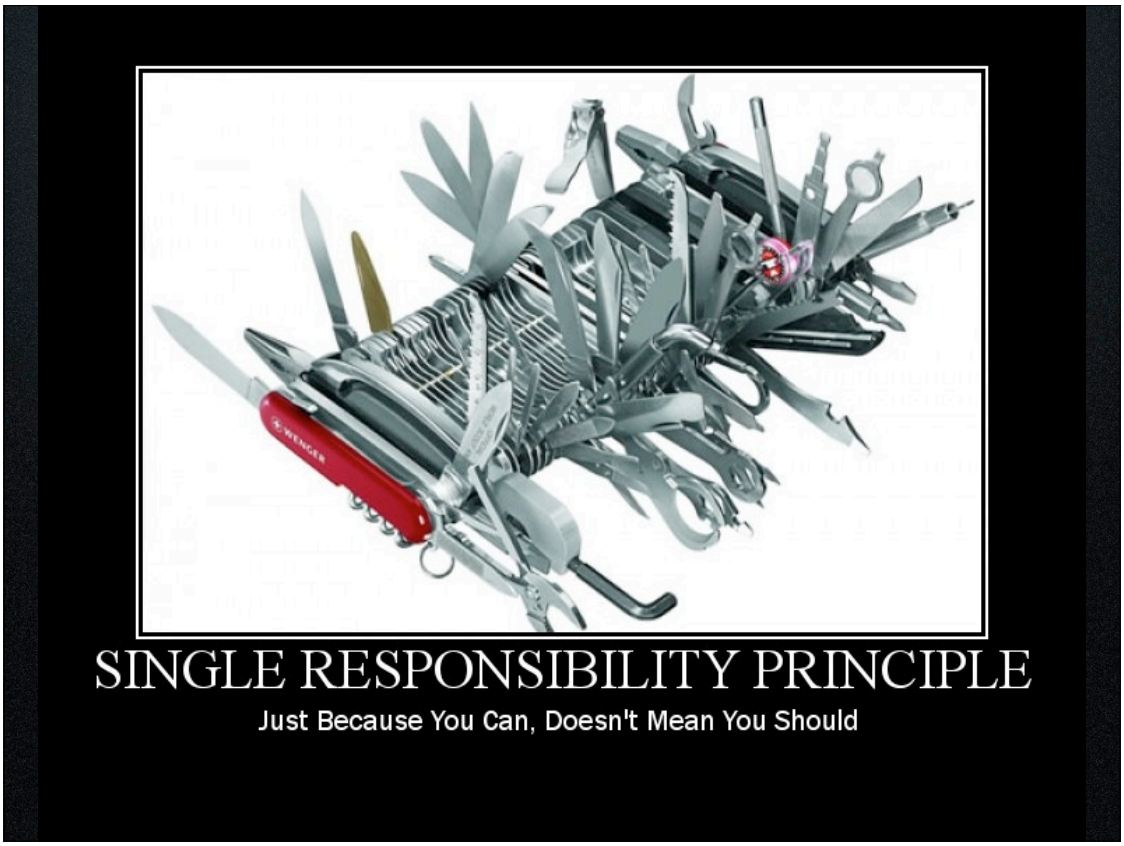
Why are Dependencies Important?


A
B
C
D
E
X


Z
A
B
C
D
E

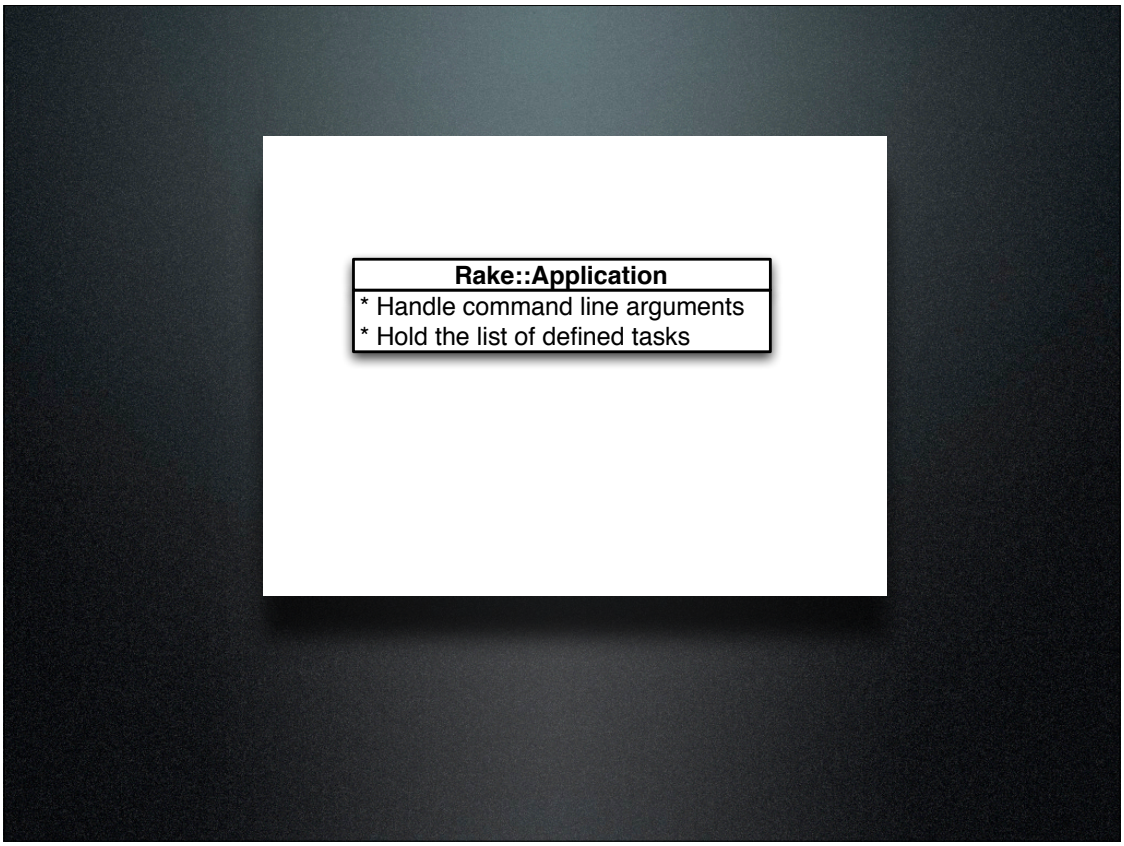


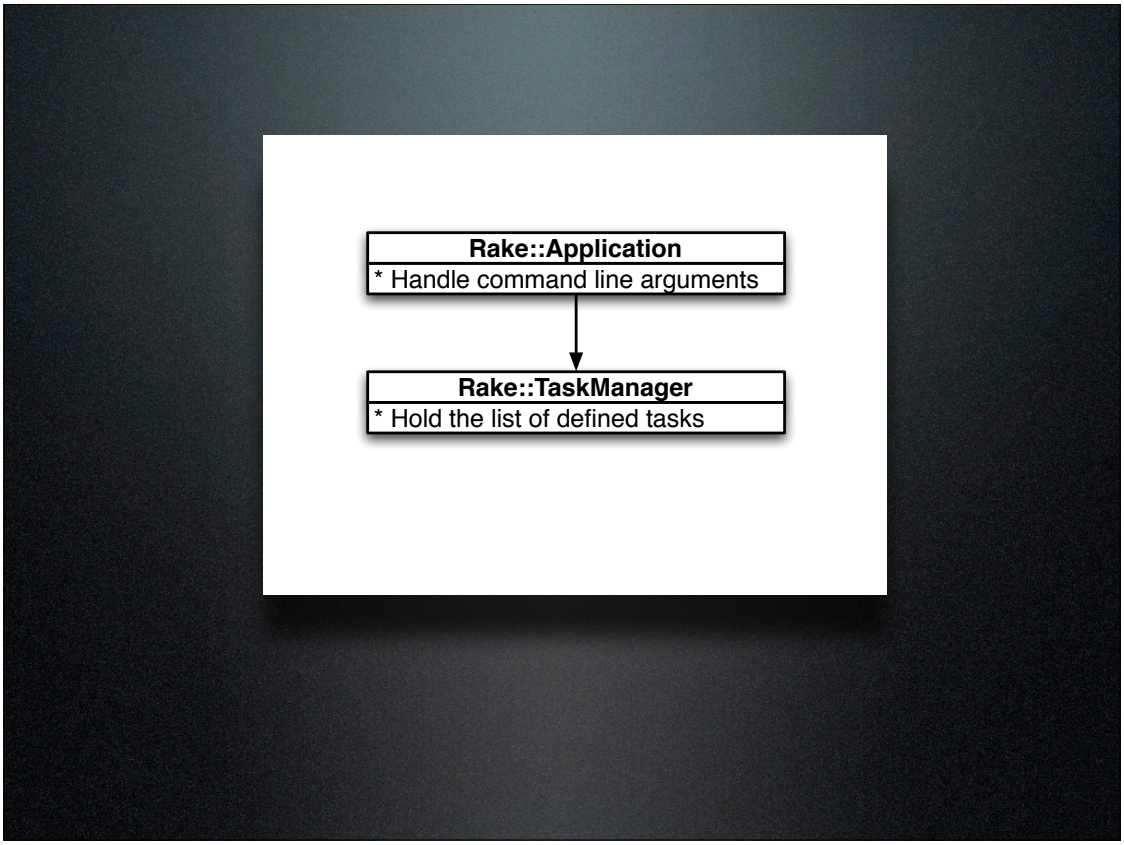
Dynamic VS Static

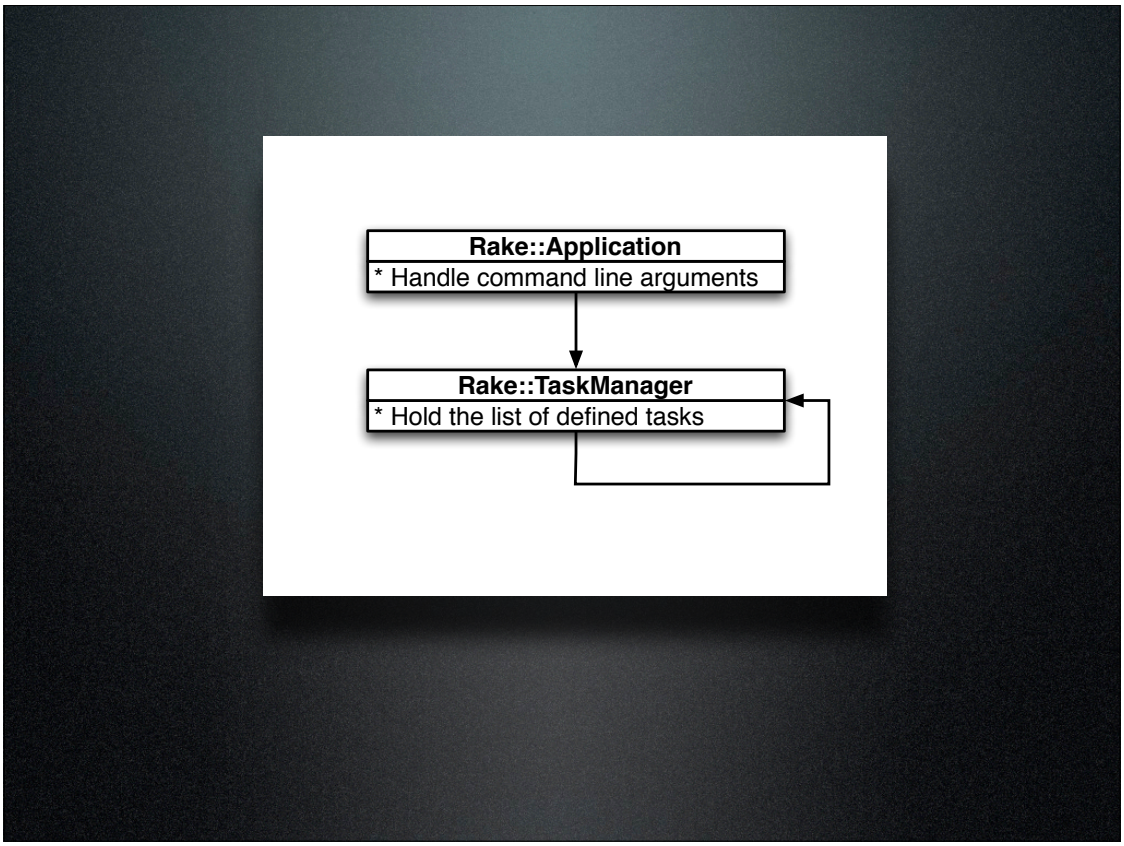


Single Responsiblity Principle

A class should have one, and only one, reason to change.


Rake::Application
* Handle command line arguments
* Hold the list of defined tasks

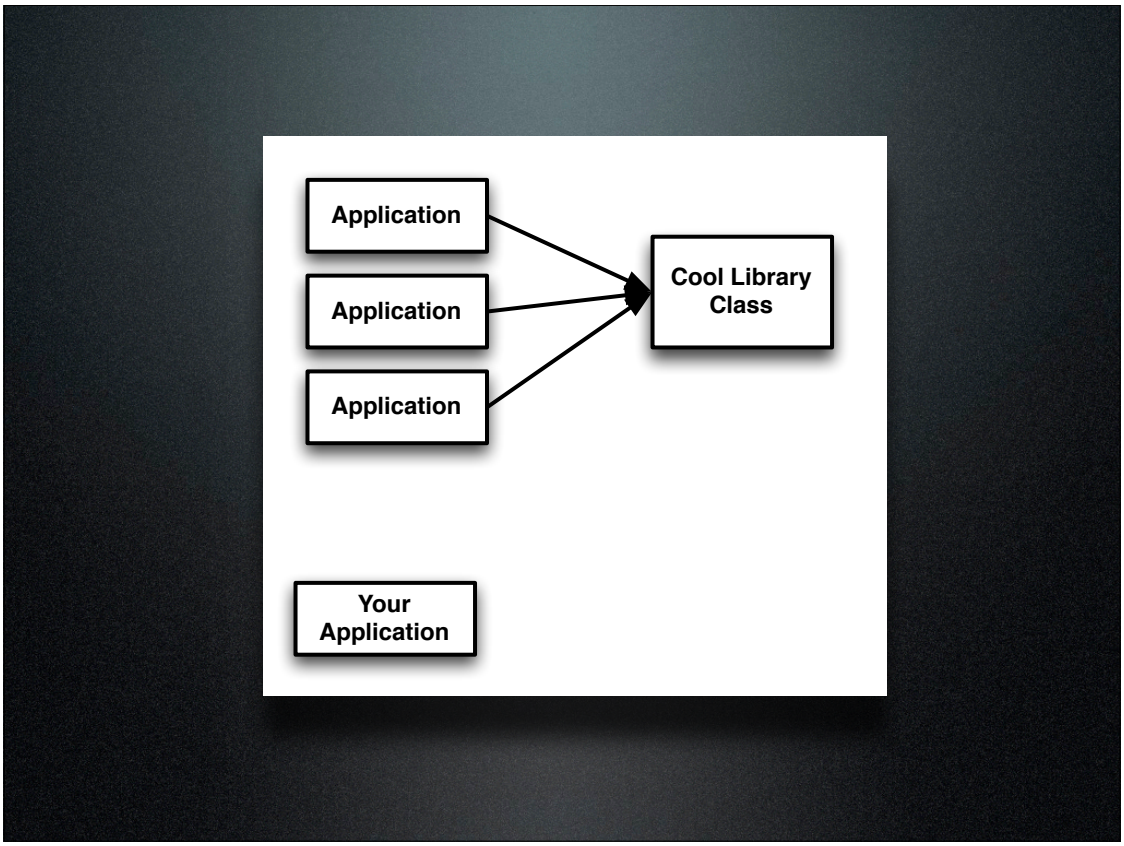

Rake::Application
* Handle command line arguments
Rake::TaskManager
* Hold the list of defined tasks

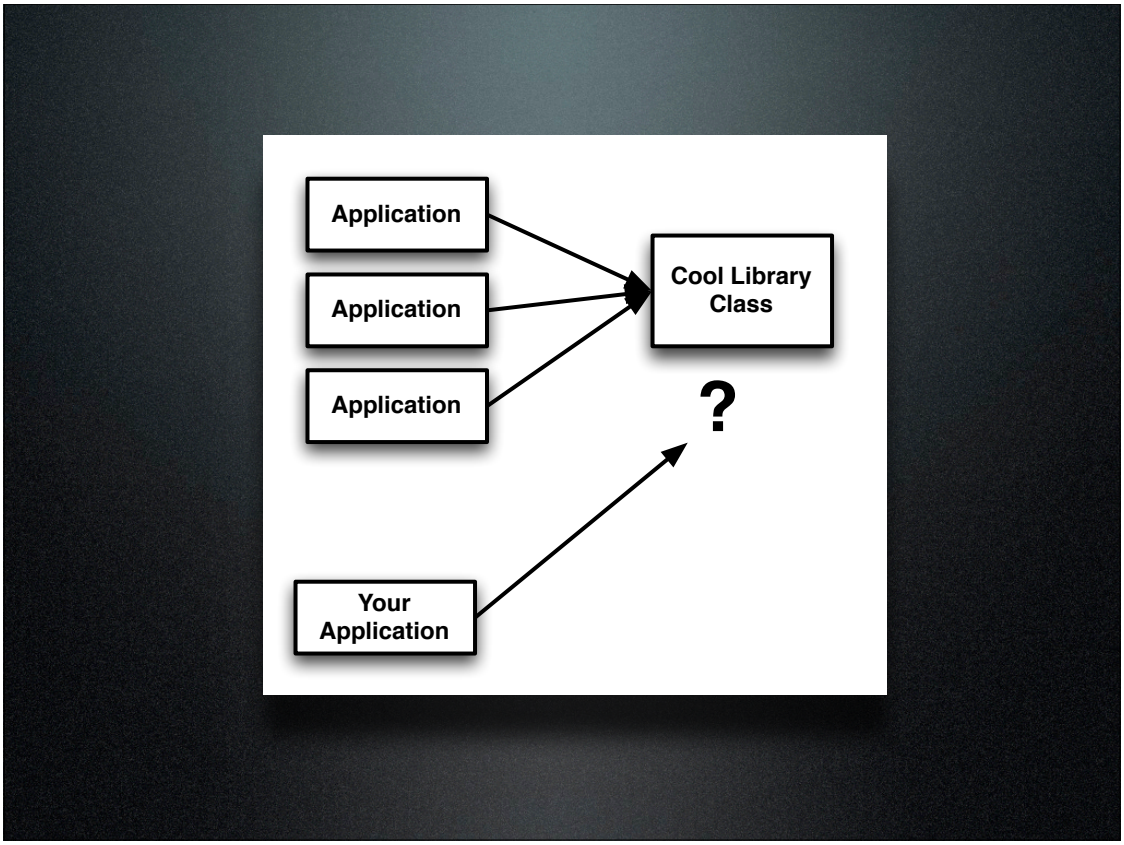

Rake::Application
* Handle command line arguments
Rake::TaskManager
* Hold the list of defined tasks

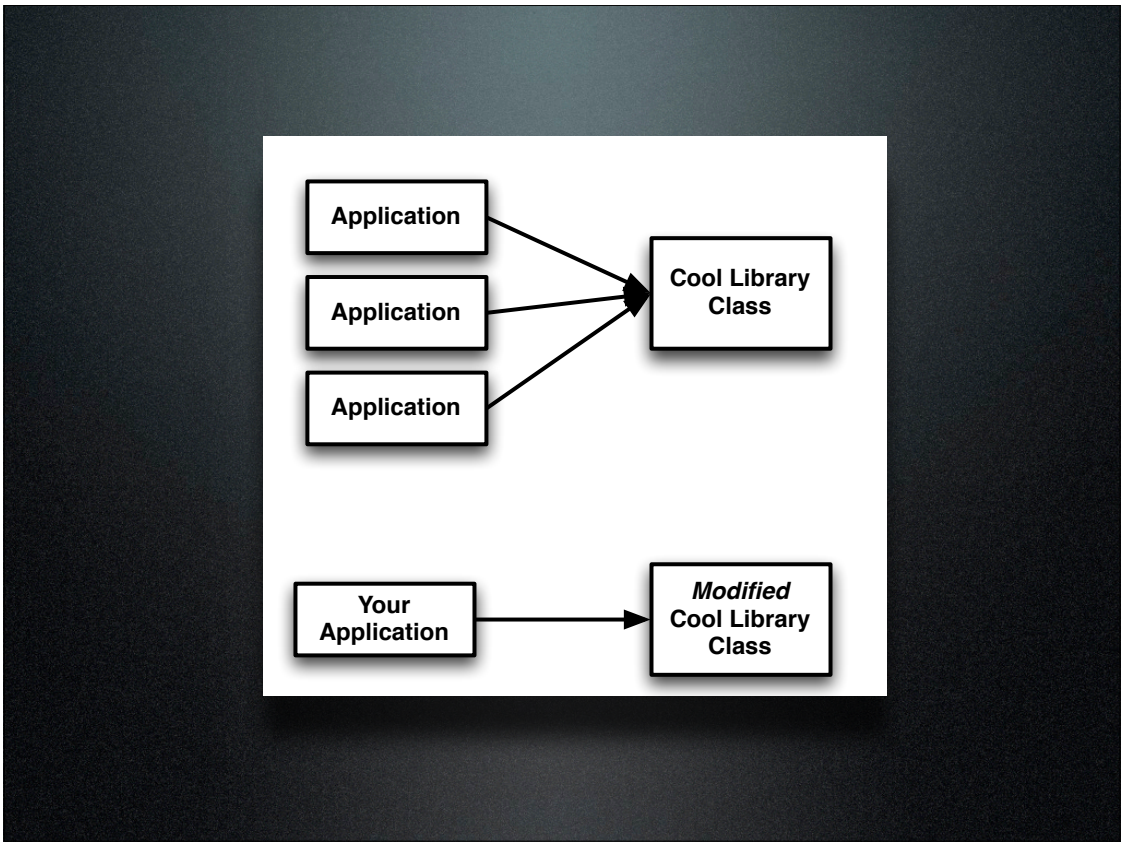
AND / OR

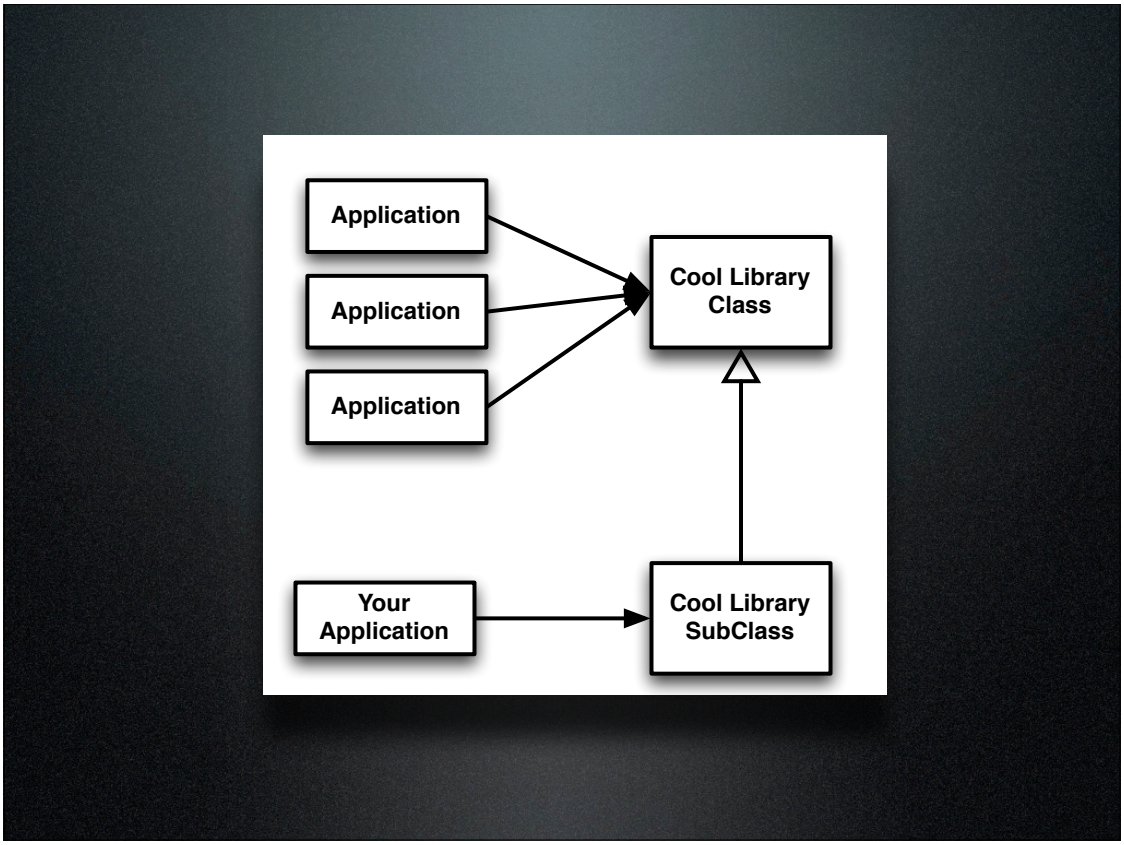
Open/Closed Principle

You should be able to extend a class's behavior, without modifying it.


Application
Application
Application
Cool Library Class
Your Application


Application
Application
Application
Cool Library Class
?
Your Application


Application
Application
Application
Cool Library Class
Your Application
Modified Cool Library Class


Application
Application
Application
Cool Library Class
Your Application
Cool Library SubClass

Open Classes?

Maybe

simple_logger.rb

```
class SimpleLogger
  attr_accessor :format_string

  def initialize
    @format_string = "%s: %s\n"
  end

  def log(msg)
    STDOUT.write(format(Time.now, msg))
  end

  def format(time, msg)
    @format_string % [
      time.strftime('%Y-%m-%d %H:%M:%S'),
      msg]
  end
end
```

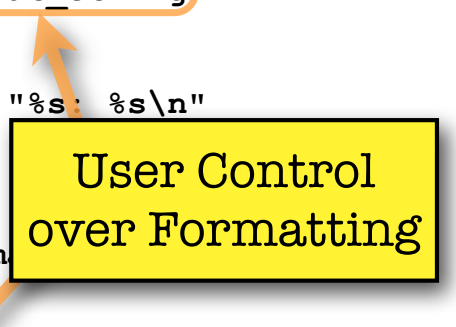

simple_logger.rb

```
class SimpleLogger
  attr_accessor :format_string

  def initialize
    @format_string = "%s: %s\n"
  end

  def log(msg)
    STDOUT.write(form
  end

  def format(time, msg)
    @format_string % [
      time.strftime('%Y-%m-%d %H:%M:%S'),
      msg]
  end
end
```



app.rb

```
require 'simple_logger'


logger = SimpleLogger.new
logger.format_string = "LOG: %s: %s\n"
logger.log("Hello, World")
```

Console:

```
$ ruby app.rb
LOG: 2009-08-06 14:56:46: Hello, World
```


app.rb

```
require 'simple_logger'
require 'cool_library'

logger = SimpleLogger.new
logger.format_string = "LOG: %s: %s\n"
logger.log("Hello, World")
```

Console:

```
$ ruby app.rb
06/08/09 15:14:22: Hello, World
```

What happened to our formatting?

cool_library.rb

```
# Open logger to fix it

class SimpleLogger
  def format(time, msg)
    "#{time.strftime("%d/%m/%y %H:%M:%S")}: " +
      #{msg}\n"
  end
end

class CoolLibrary
  # blah blah blah
end
```

Library Overrides Formatting!

Principle Of Least Surprise

Principle Of Most Heinous Arsenic Injection

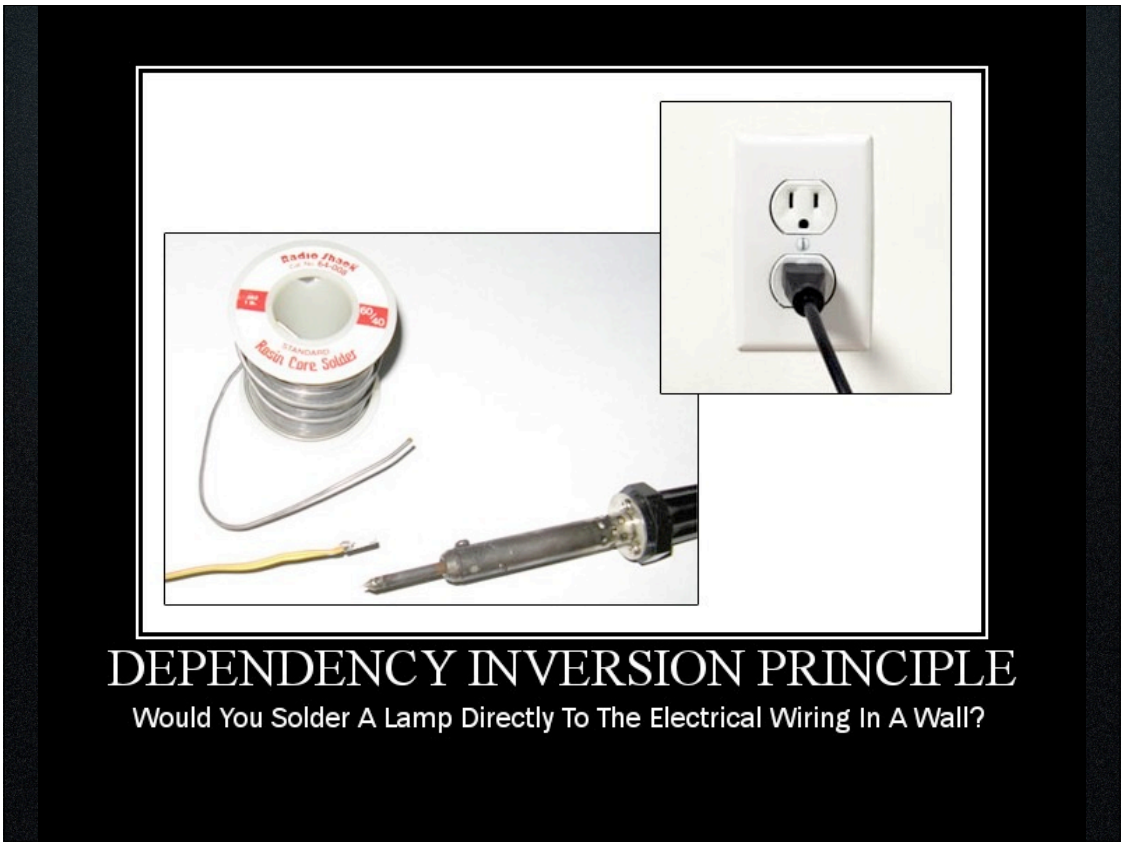
Better Way?

cool_library.rb

```
class CoolLogger < SimpleLogger
  def format(time, msg)
    "#{time.strftime("%d/%m/%y %H:%M:%S")}: " +
      "#{msg}\n"
  end
end

class CoolLibrary
  # blah blah blah
end
```


Prefer Subclassing or Wrapping over Reopening classes

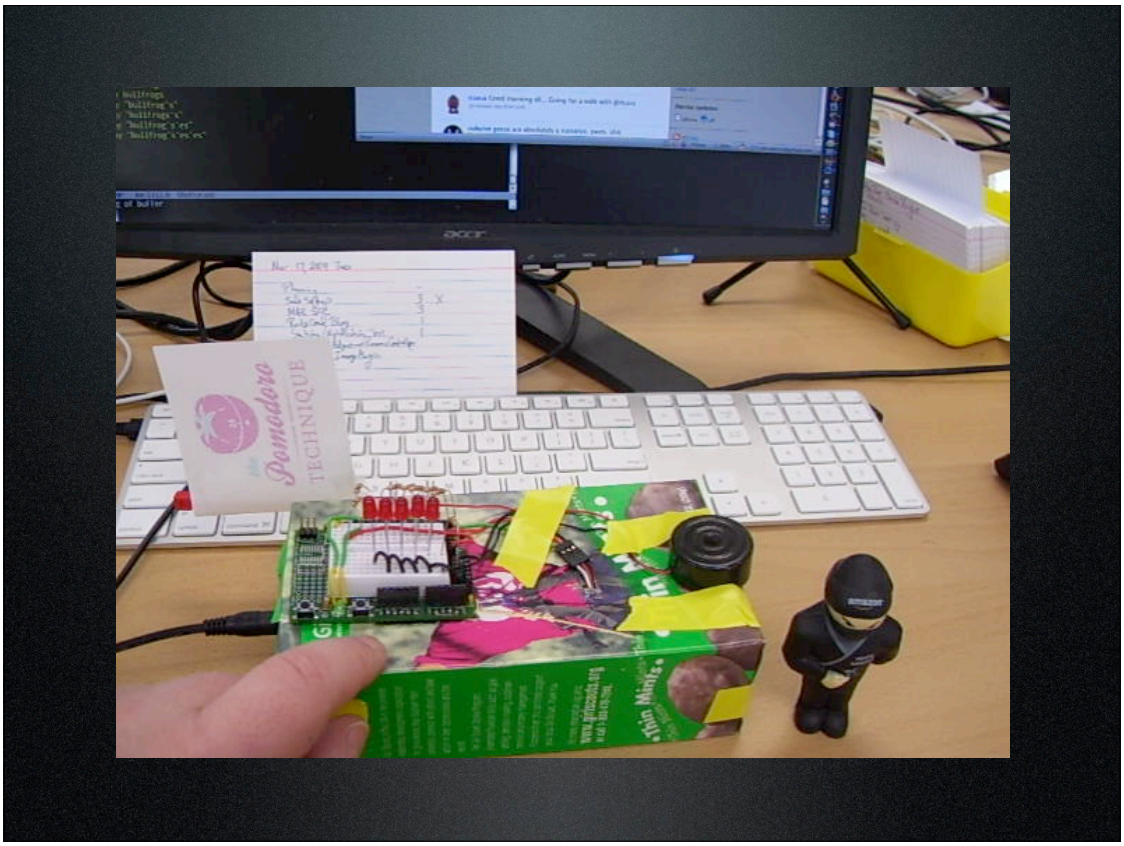


DEPENDENCY INVERSION PRINCIPLE

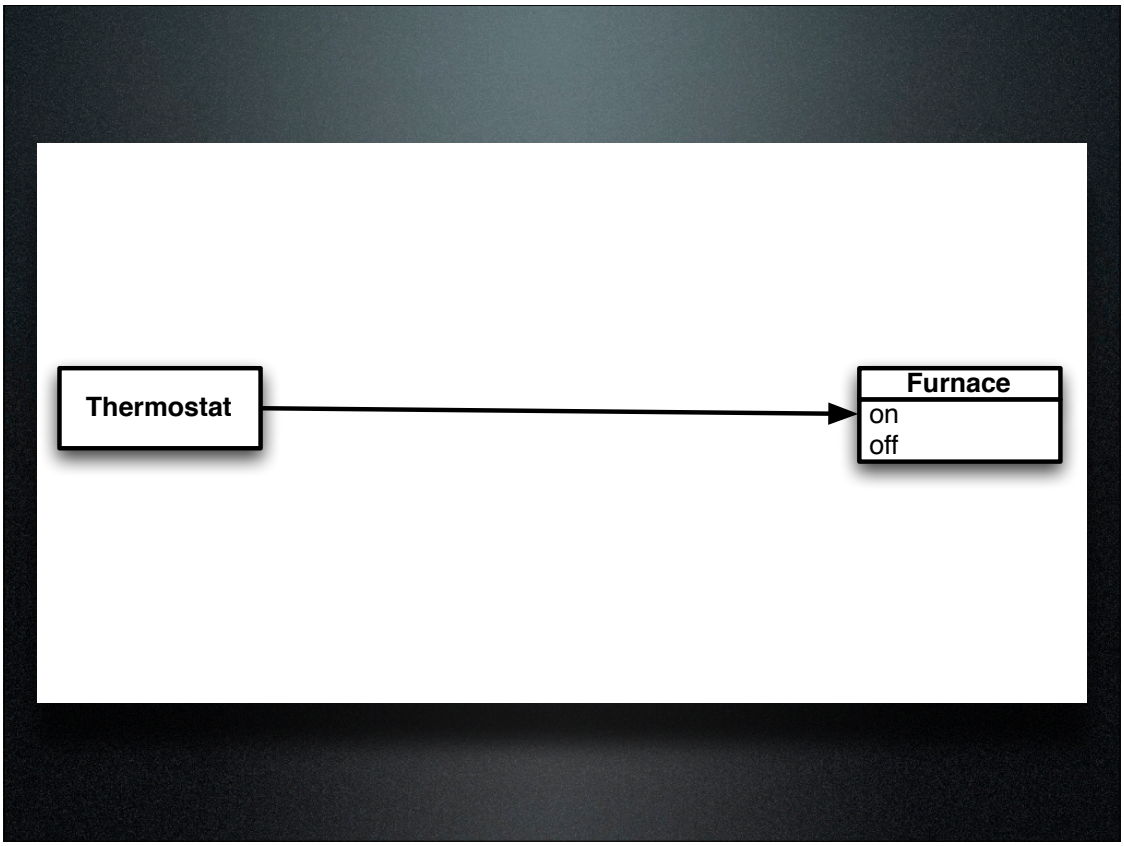
Would You Solder A Lamp Directly To The Electrical Wiring In A Wall?

Dependency Inversion Principle

Depend on abstractions, not on concretions.



The Thermostat Problem



Furnace.java

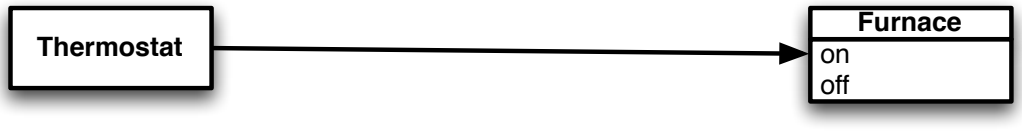
```
class Furnace {
    public Furnace() {
        off();
    }
    public void on() {
        // Code to turn furnace on.
    }
    public void off() {
        // Code to turn furnace off.
    }
}
```

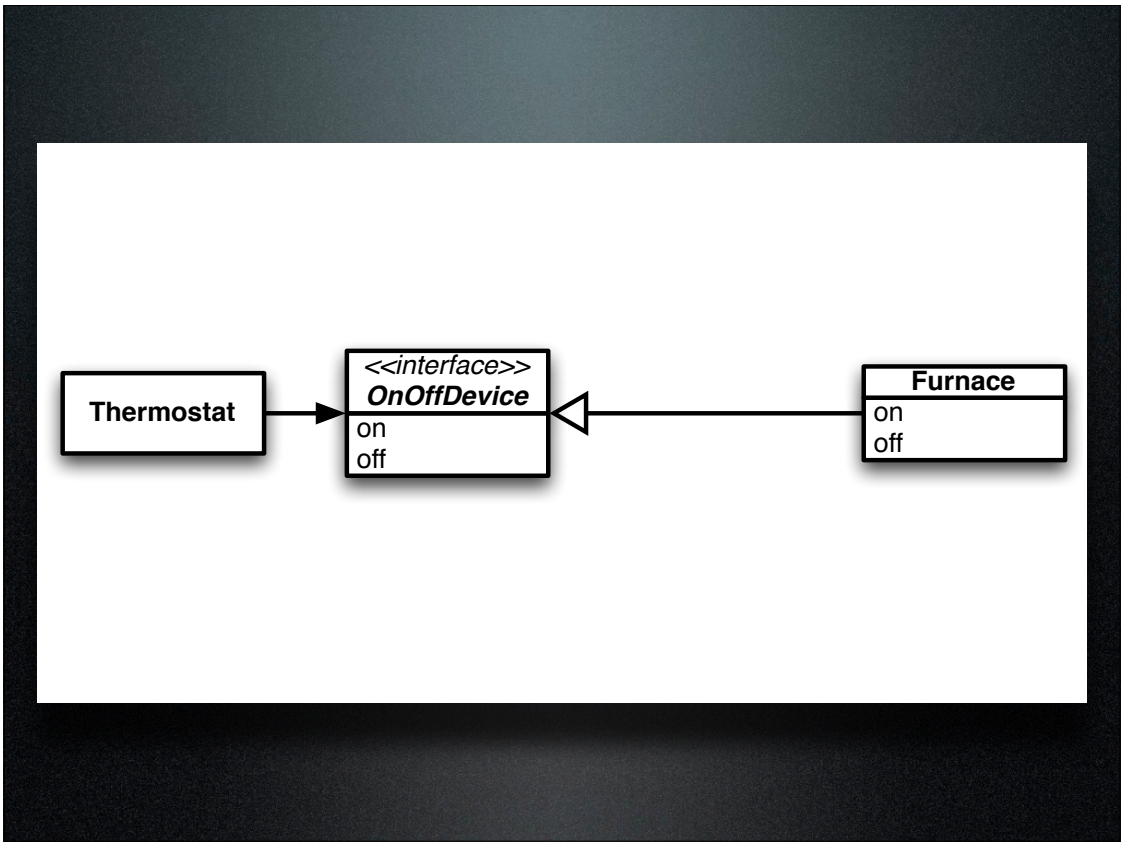
Thermostat.java

```
class Thermostat {
    public Thermostat(Furnace f) {
        this.furnace = f;
    }
    public void run() {
        if (should_be_on()) {
            this.furnace.on();
        } else {
            this.furnace.off();
        }
    }

    private Furnace furnace;
    //...
}
```

Explicit references to Furnace





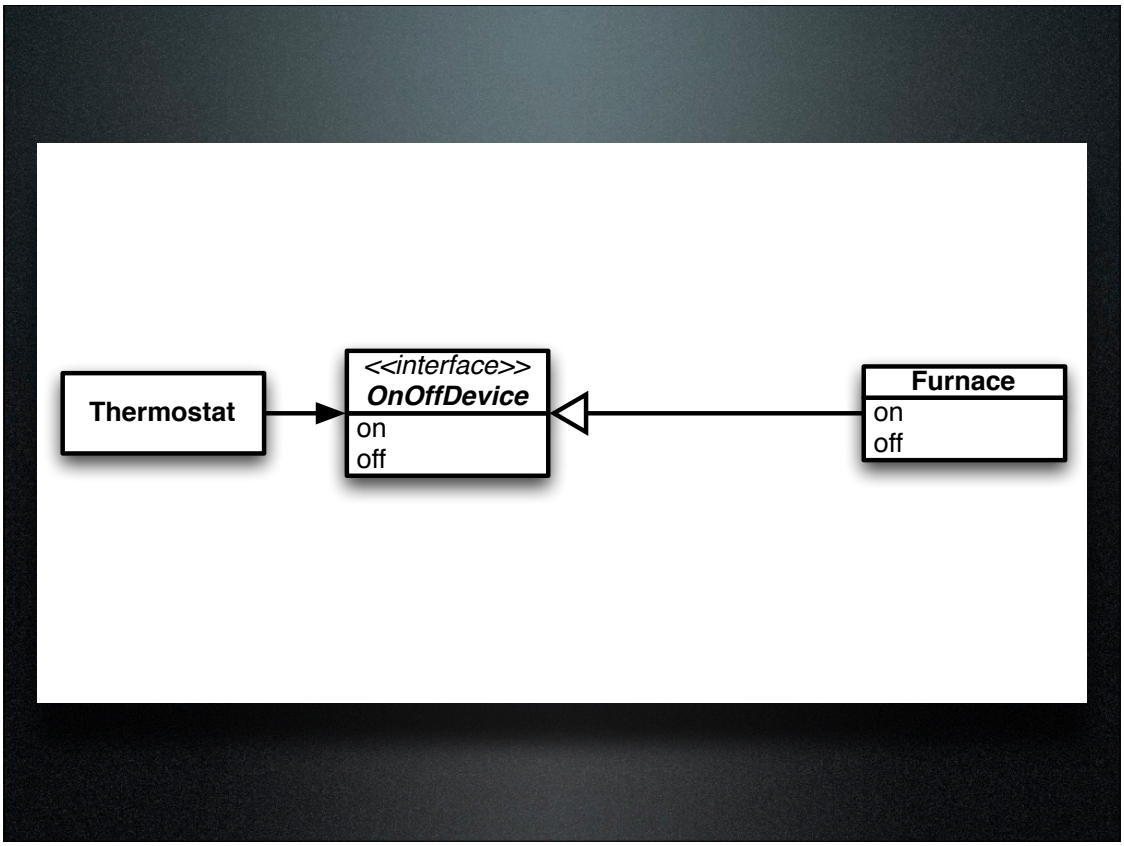
OnOffDevice.java

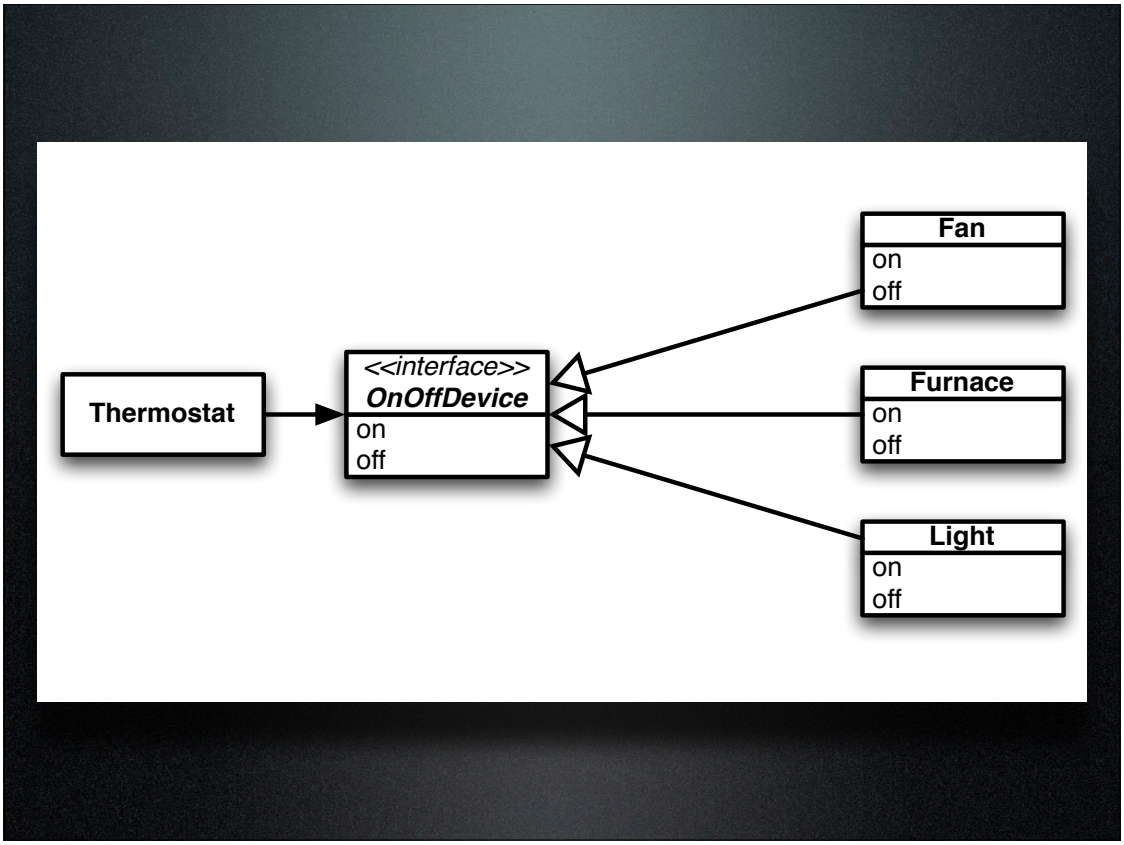
```
interface OnOffDevice {
    public void on();
    public void off();
}
```

Thermostat.java

```
class Thermostat {
    public Thermostat(OnOffDevice f) {
        this.furnace = f;
    }
    // ...

    private OnOffDevice furnace;
    //...
}
```


Thermostat
<<interface>>
OnOffDevice
on
off
Furnace
on
off


Thermostat
<<interface>>
OnOffDevice
on
off
Fan
on
off
Furnace
on
off
Light
on
off

Code to Interfaces

How about Ruby?

Furnace.rb

```
class Furnace
  def initialize
    off
  end
  def on
    # Code to turn furnace on
  end
  def off
    # Code to turn furnace off
  end
end
```


Thermostat.rb -- Pre-DIP

```
class Thermostat
  def initialize(furnace)
    @furnace = furnace
  end
  def run
    if should_be_on?
      @furnace.on
    else
      @furnace.off
    end
  end
  # ...
end
```

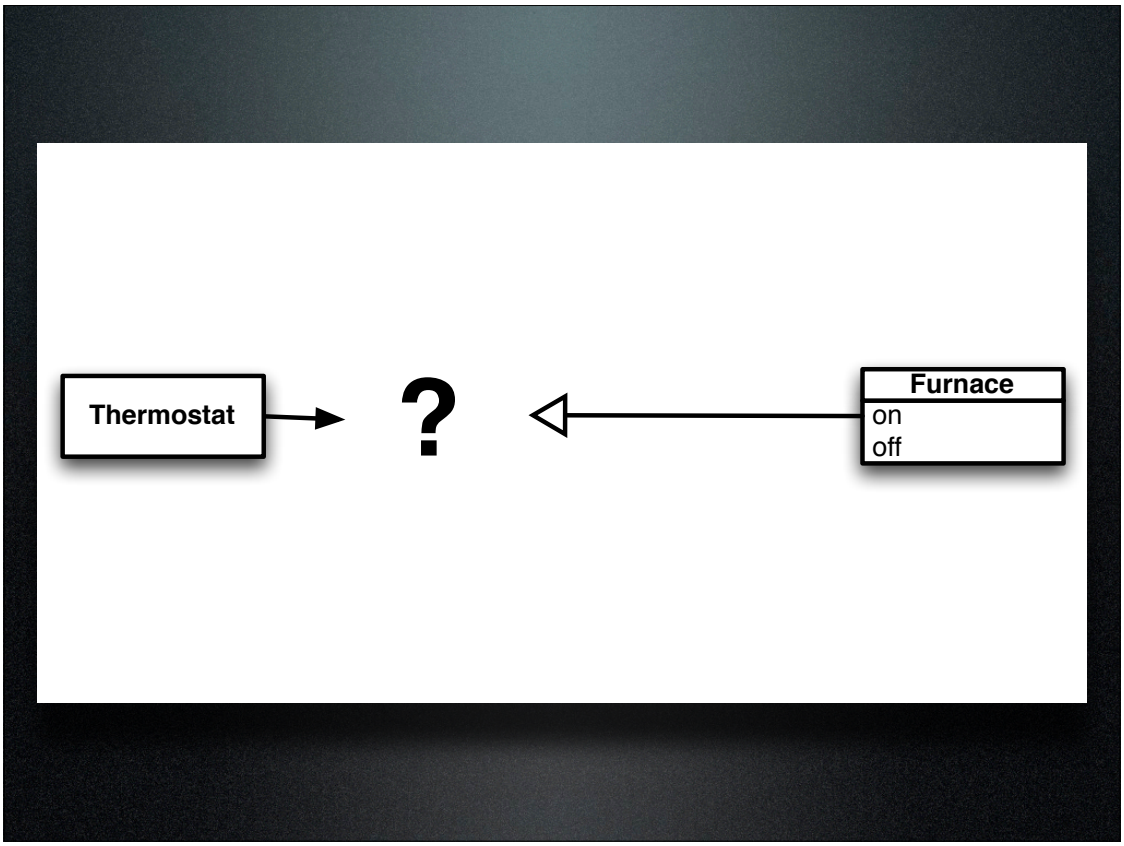
Thermostat.rb -- Post-DIP

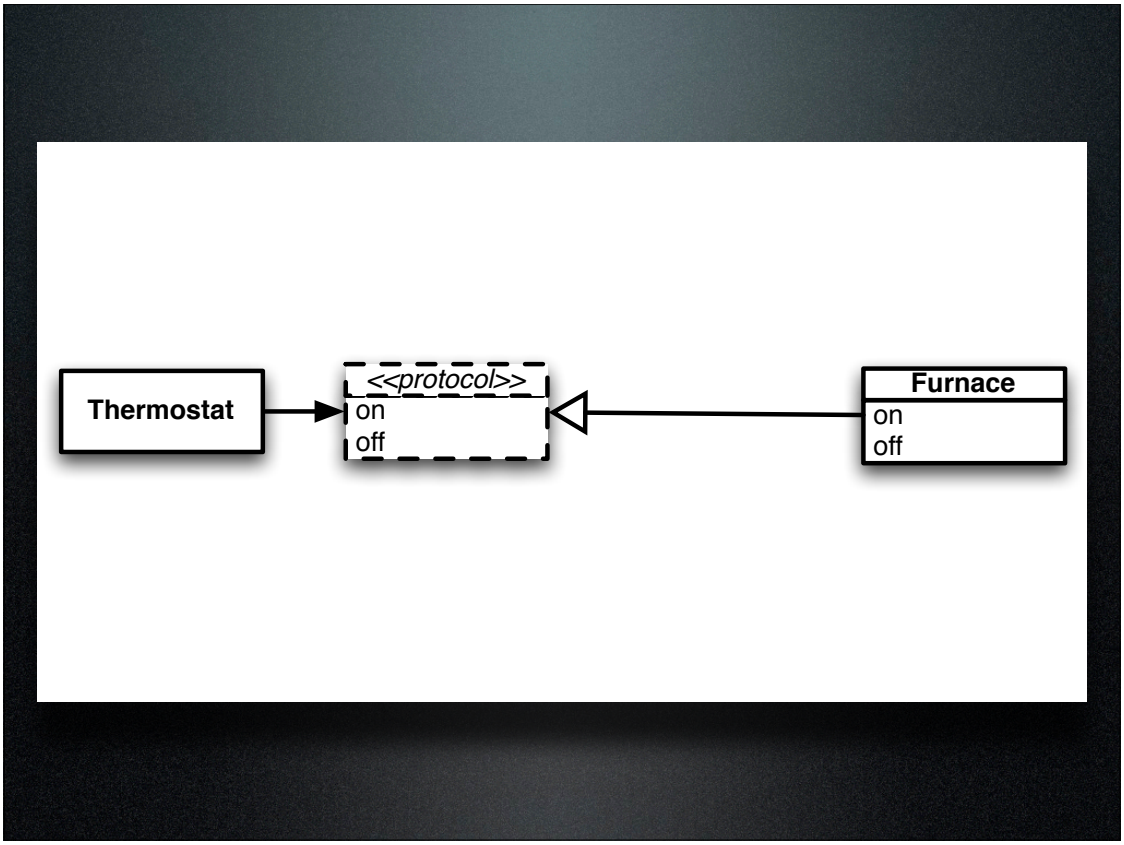
```
class Thermostat
  def initialize(furnace)
    @furnace = furnace
  end
  def run
    if should_be_on?
      @furnace.on
    else
      @furnace.off
    end
  end
  # ...
end
```


Thermostat.rb (an aside)

```
class Thermostat
  def initialize(furnace)
    fail "Must use a Furnace" unless
      furnace.is_a?(Furnace)
    @furnace = furnace
  end
  def run
    if should_be_on?
      @furnace.on
    else
      @furnace.off
    end
  end
  # ...
end
```

Explicit references to Furnace


Thermostat
?
Furnace
on
off


Thermostat
<<protocol>>
on
off
Furnace
on
off

Protocols are Important

(another aside)

Thermostat.rb

```
class Thermostat
  def initialize(furnace)
    fail "Must use a On/Off Device" unless
      [:on, :off].all?{|m| furnace.respond_to?(m)}
    @furnace = furnace
  end
  def run
    if should_be_on?
      @furnace.on
    else
      @furnace.off
    end
  end
  # ...
end
```

No Explicit Reference, but not much better

Chicken Typing

XmlBuilder

String Builder Example

```
sb = Builder::XmlMarkup.new(:target => "")
sb.person { sb.name("Jim") }
puts sb.target!
```


new(options={})

Create an XML markup builder. Parameters are specified by an option hash.

:target=>*target_object*: Object receiving the markup. `target_object` must respond to the `<<(a_string)` operator and return itself. The default target is a plain string target.

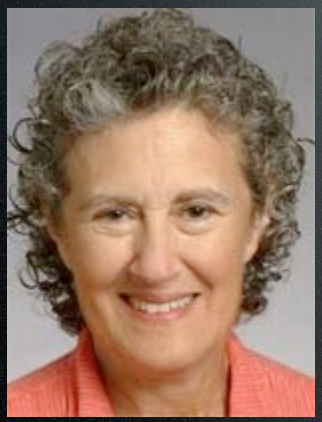
Shovel Operator

File Builder Example

```
fb = Builder::XmlMarkup.new(:target => STDOUT)
fb.person { fb.name("Bob") }
```


Code to Protocols

Liskov Substitution Principle



Barbara Liskov

If for each object o1 of type S there is an object o2 of type T such that for all programs P defined in terms of T, the behavior of P is unchanged when o1 is substituted for o2 then S is a subtype of T.

Liskov Substitution Principle

Derived classes must be substitutable for their base classes.

-- Robert Martin

Liskov Substitution Principle

Sort of like when they changed the actors who played "Darren" in Bewitched

-- Joey deVilla

Liskov Substitution Principle

Duck Typing

(If it walks like a duck ... etc.)

-- Dave Thomas

When is something... substitutable?

```
class NormalMath
  def sqrt(x)
    guess = 1.0
    while (x - guess*guess).abs > 0.001
      guess = (guess + (x/guess)) / 2.0
    end
    guess
  end
end
```


Substitutable?

```
class AccurateMath
  def sqrt(x)
    guess = 1.0
    while (x - guess*guess).abs > 0.00001
      guess = (guess + (x/guess)) / 2.0
    end
    guess
  end
end
```

How about this one?

```
class SloppyMath
  def sqrt(x)
    guess = 1.0
    while (x - guess*guess).abs > 0.1
      guess = (guess + (x/guess)) / 2.0
    end
    guess
  end
end
```


Two Questions

- What does the method require?
- What does the method promise?

```
def sqrt(x)
requires
  ?
promises
  ?
```

Two Questions

- What does the method require?
- What does the method promise?

```
def sqrt(x)
requires
  non_negative: x >= 0
promises
  ?
```


Two Questions

- What does the method require?
- What does the method promise?

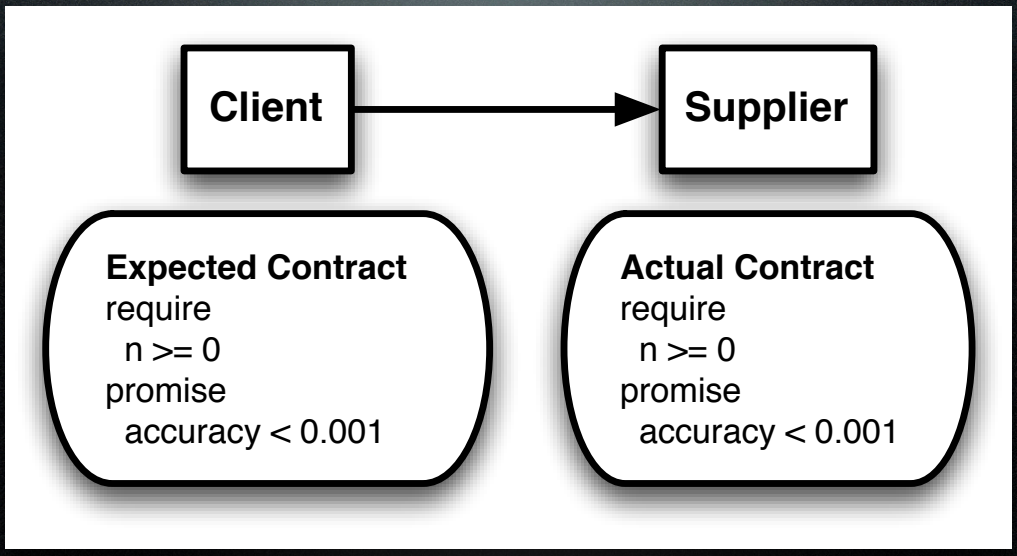
```
def sqrt(x)
requires
  non_negative: x >= 0
promises
  accurate: (x - result**2).abs <= 0.001
```

Client Software

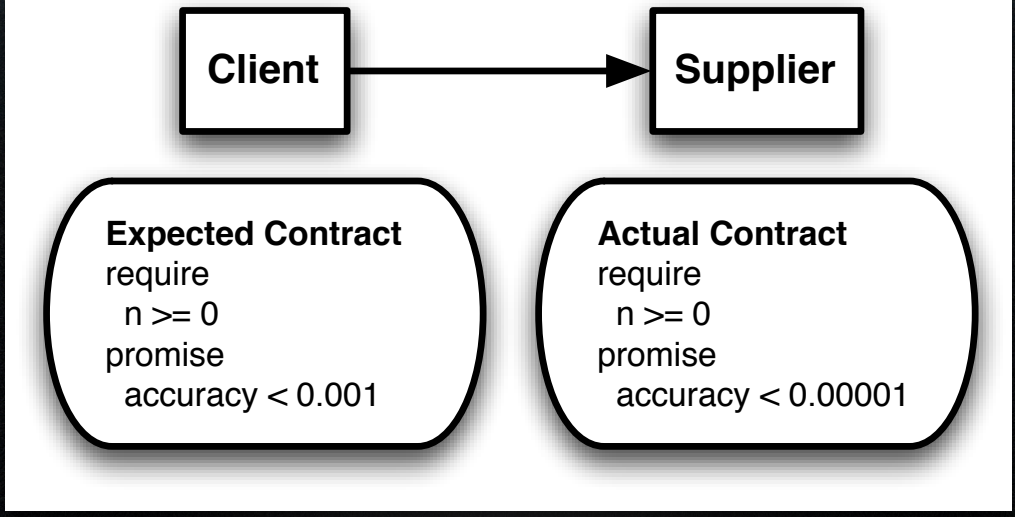
```
# Distance between points a and b

def distance(a, b, math)
  math.sqrt(
     (a.x - b.x)**2 +
     (a.y - b.y)**2)
end
```

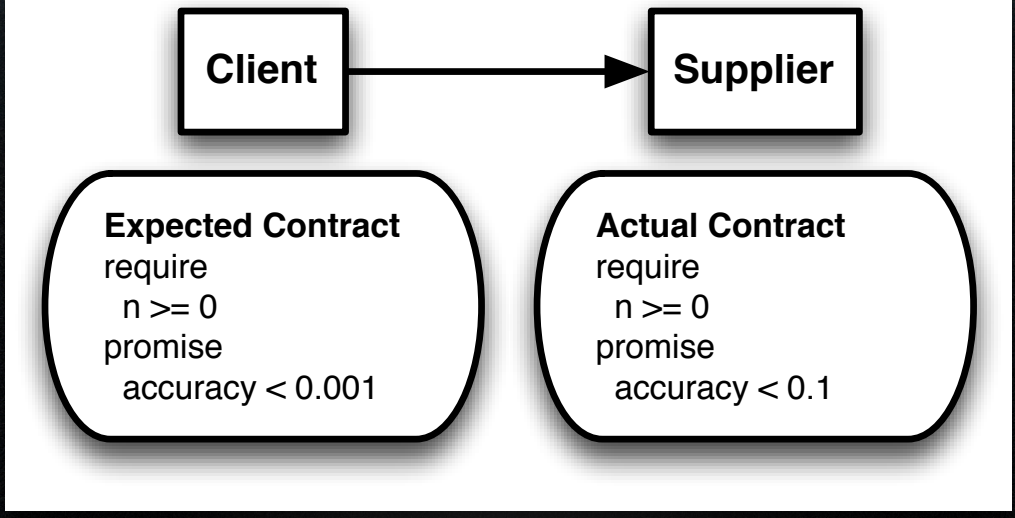

Same Contract - OK



Better Promise - OK



Worse Promise - FAIL



How about this one?

```
class ComplexSquare
  def root(x)
    guess = (x < 0) ? Complex(0, 1) : 1.0
    while (x - guess*guess).abs > 0.001
      guess = (guess + (x/guess)) / 2.0
    end
    guess
  end
end
```


How about this one?

```
class ComplexSquare
  def root(x)
    guess = (x < 0) ? Complex(0, 1) : 1.0
    while (x - guess*guess).abs > 0.001
      guess = (guess + (x/guess)) / 2.0
    end
    guess
  end
end
```

No Assumptions about x >= 0

How about this one?

```
class ComplexSquare
  def root(x)
    guess = (x < 0) ? Complex(0, 1) : 1.0
    while (x - guess*guess).abs > 0.001
      guess = (guess + (x/guess)) / 2.0
    end
    guess
  end
end
```

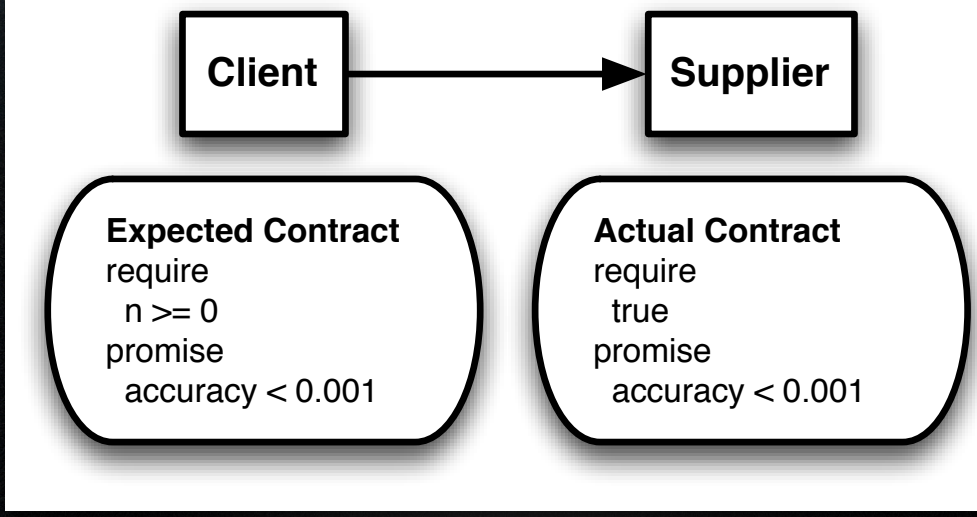
Same Accuracy Guarantee!

Complex Math Contract

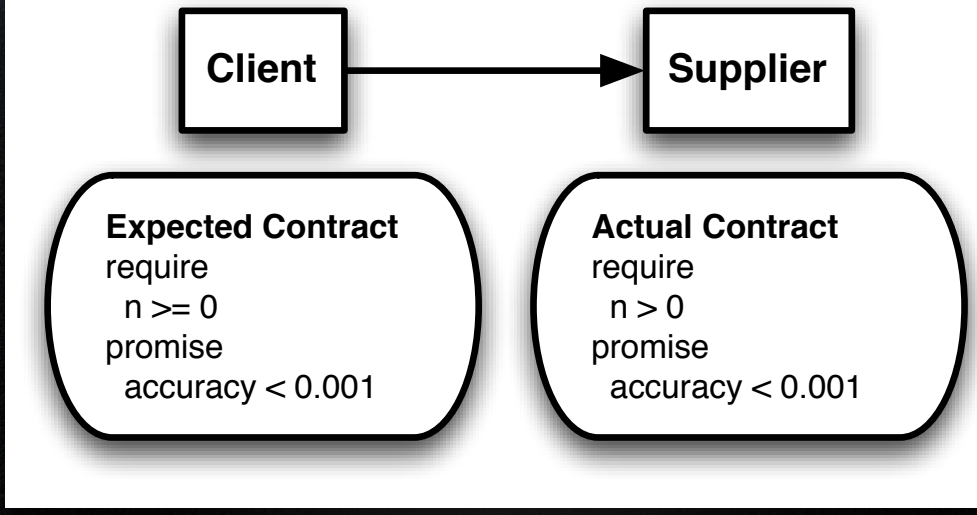
- No requirements on the value of x
  - Equivalent to a TRUE condition

```
def sqrt(x)
requires
  anything: true
promises
  accurate: (x - result**2).abs <= 0.001
```

Less Restrictive Requirement - OK

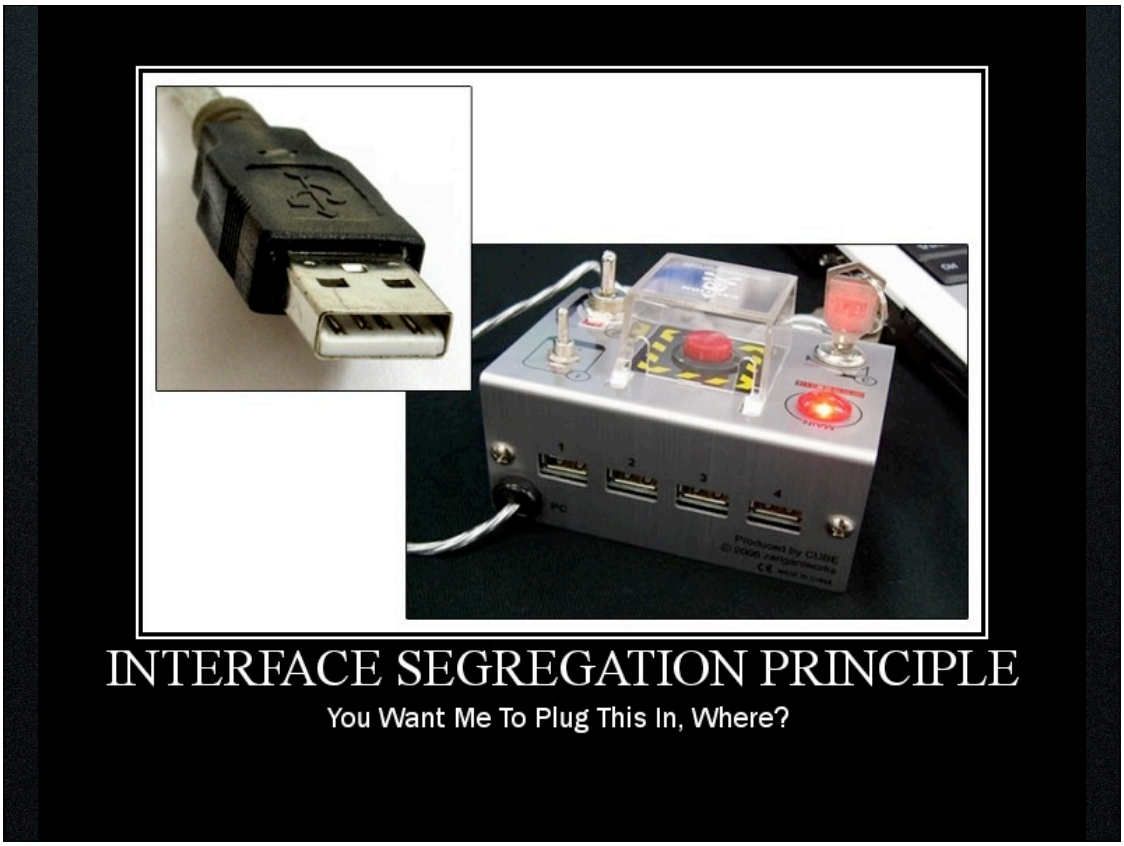


More Restrictive Requirement - FAIL



When is something... substitutable?

Require No More
Promise No Less

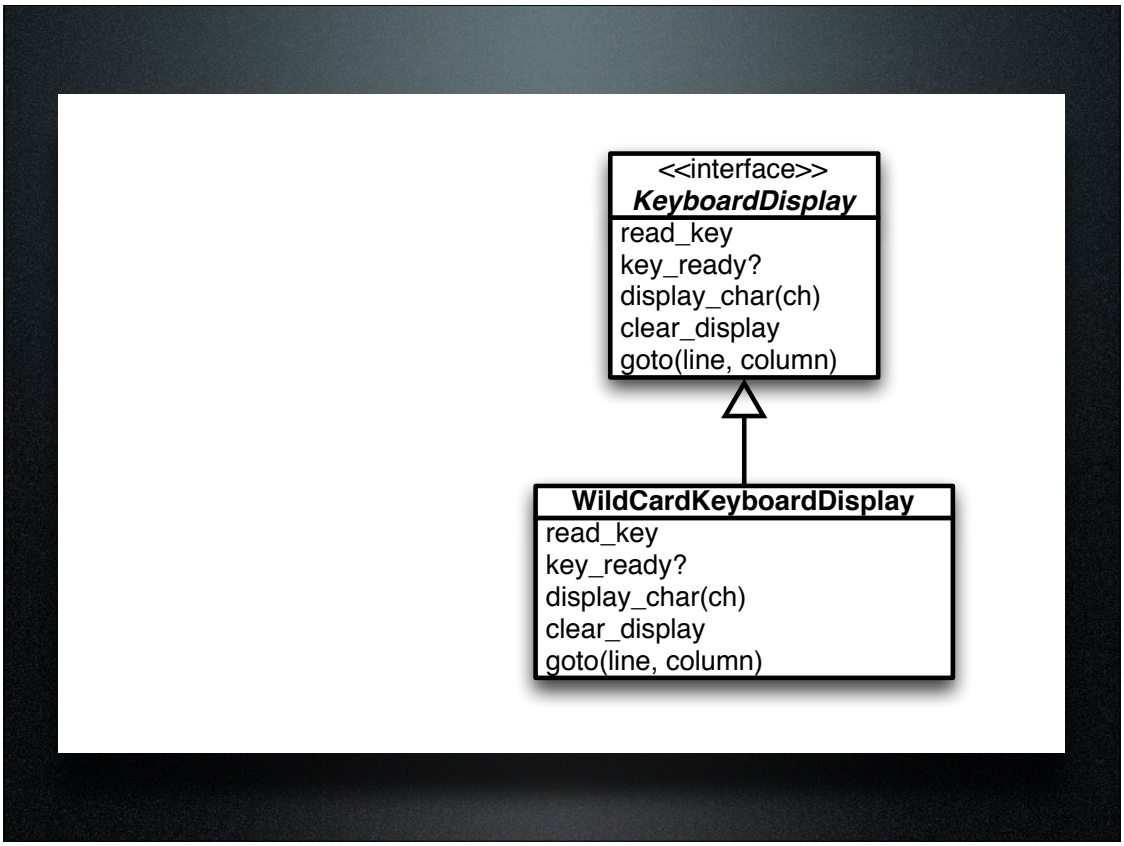


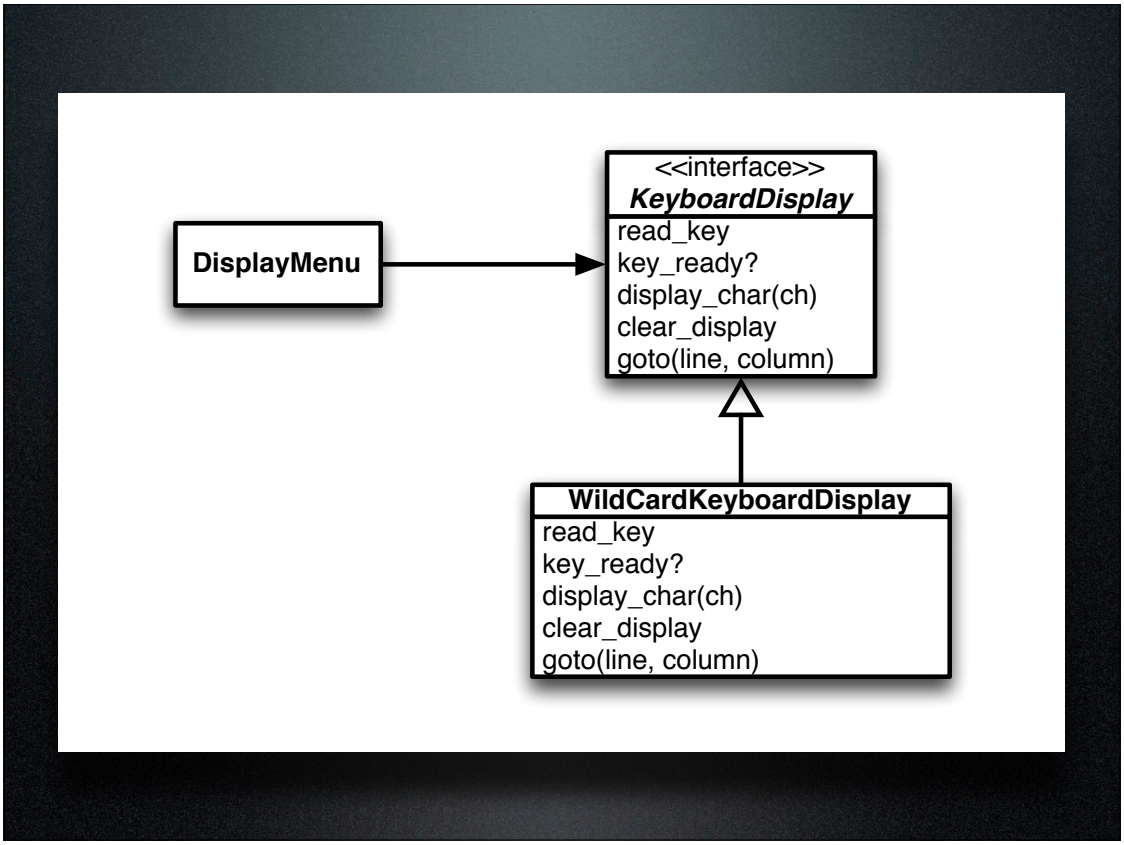
Interface Segregation Principle

Make fine grained interfaces that are client specific.

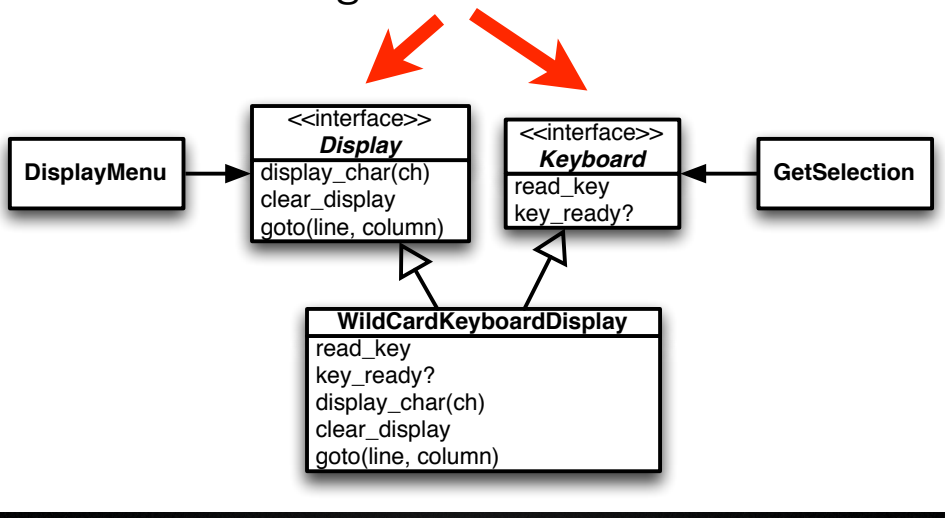
Wildcard Keyboard/Display

WildCardKeyboardDisplay
read_key
key_ready?
display_char(ch)
clear_display
goto(line, column)


<<interface>>
KeyboardDisplay
read_key
key_ready?
display_char(ch)
clear_display
goto(line, column)
WildCardKeyboardDisplay
read_key
key_ready?
display_char(ch)
clear_display
goto(line, column)

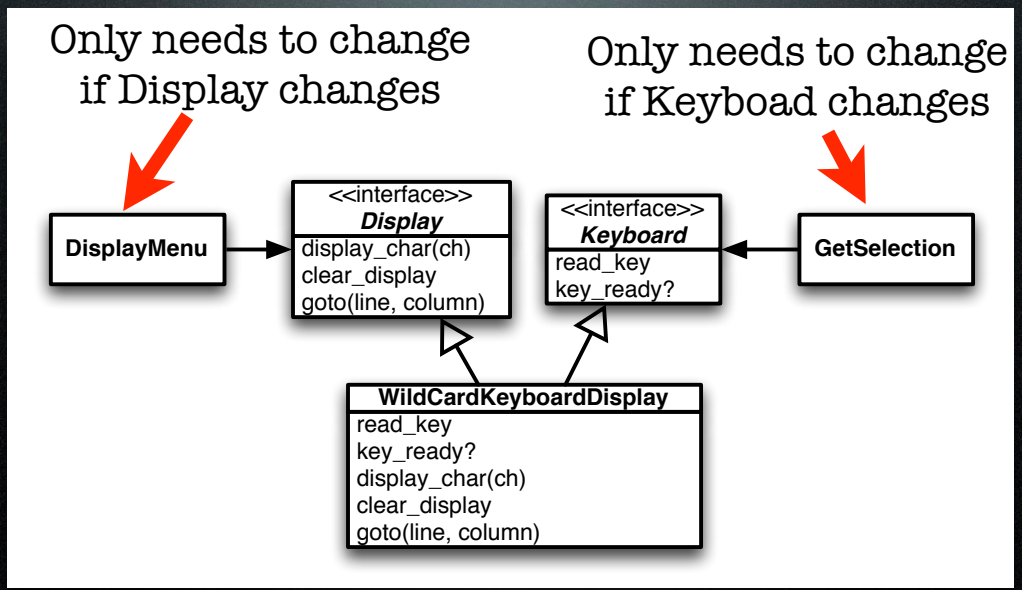

DisplayMenu
<<interface>>
KeyboardDisplay
read_key
key_ready?
display_char(ch)
clear_display
goto(line, column)
WildCardKeyboardDisplay
read_key
key_ready?
display_char(ch)
clear_display
goto(line, column)

Fine grained interfaces

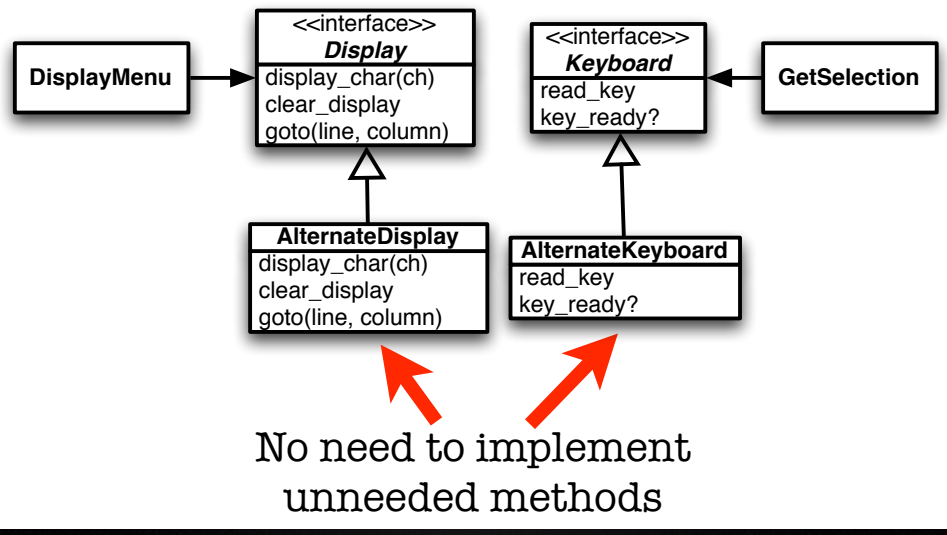


Why?

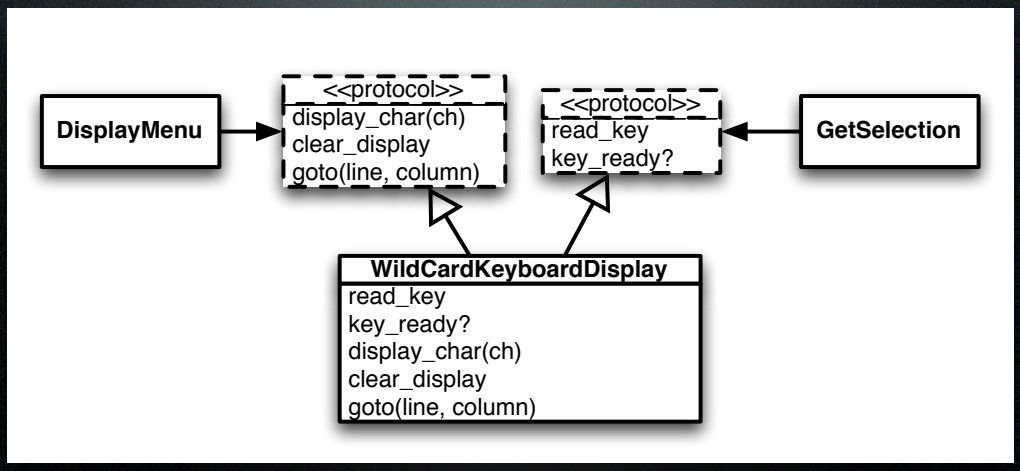
(1)
Client software depends on less (and therefore have fewer reasons to change)



(2)
Easier to implement alternative solutions



ISP with Protocols?



Interface Segregation Principle

Clients should depend on as narrow protocol as possible.

e.g.
Builder only depends on the << operator.

e.g.
If you are using an array as a stack ... only use the push/pop methods

For Discussion ...

For Discussion

ActiveRecord objects implement a domain concept and a persistence concept. Does this violate SRP?

For Discussion

Sometimes you can't express requirements/promises as logical expressions. What should you do?

For Discussion

How do you verify that your client software only uses the (narrow) protocol you expect?

For Discussion

Some SOLID principles are an awkward fit for Ruby.
Are there other SOLID-like principles that are specific to dynamic languages?

Summary

SOLID Principles are important for understanding good OO design

Although developed with static languages in mind ...

... Useful lessons can still be applied to dynamic languages (like Ruby)

Thank - You

Questions?

References

References

- More on SOLID can be found at:
  - http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod
  - http://www.lostechies.com/content/pablo_ebook.aspx
  - http://www.globalnerdy.com/2009/07/31/barbara-liskov-interviewed/ (interview with Barbara Liskov)

References

- Design by contract:
  - Eiffel, the Language
    - http://eiffel.com/
  - Hoare logic and stuff on pre/post conditions can be found at:
    - http://en.wikipedia.org/wiki/Hoare_logic

References

- Images are from Pablo's SOLID Software Development E-Book, available at:
  - http://www.lostechies.com/blogs/derickbailey/archive/2009/05/19/announcing-pablo-s-e-books-book-1-pablo-s-solid-software-development.aspx