Aspect-Oriented Programming with Module#prepend

Colin Kelley
CTO/co-founder, Invoca, Inc

SBonRails - July 8, 2014

Aspect-Oriented Programming

from Wikipedia:

Aspect-oriented programming (AOP) is patented programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns…

without modifying the code itself.

Fundamentals: Module vs. Class

```
>> Class.ancestors
=> [Class, Module, Object, Kernel, BasicObject]
```

- Module is a namespace for instance and module methods
- Class adds Class.**new** that calls **#initialize**

Fundamentals: include vs. Inheritance

```
class A < Base
  include C
  include D
end

>> A.ancestors
=> ???
```

Fundamentals:
inheritance vs. ActiveSupport::Concern

```
class Base
  def instance_method; puts "instance Base"; end
  def self.class_method; puts "class Base"; end
end

class A < Base
  def instance_method; puts "instance A\n"; super; end
  def self.class_method; puts "class A\n"; super; end
end

>> a = A.new
>> a.instance_method
>> a.class.class_method
=> ??
```

Fundamentals:
inheritance vs. ActiveSupport::Concern

```
module B
  extend ActiveSupport::Concern
  def instance_method; puts "instance B"; end
  module ClassMethods
    def class_method; puts "class B\n"; end
  end
end
class A
  include B
  def instance_method; puts "instance A\n"; super; end
  def self.class_method; puts "class A\n"; super; end
end
>> a = A.new
>> a.instance_method
>> a.class.class_method
>> ??
```

include vs. prepend

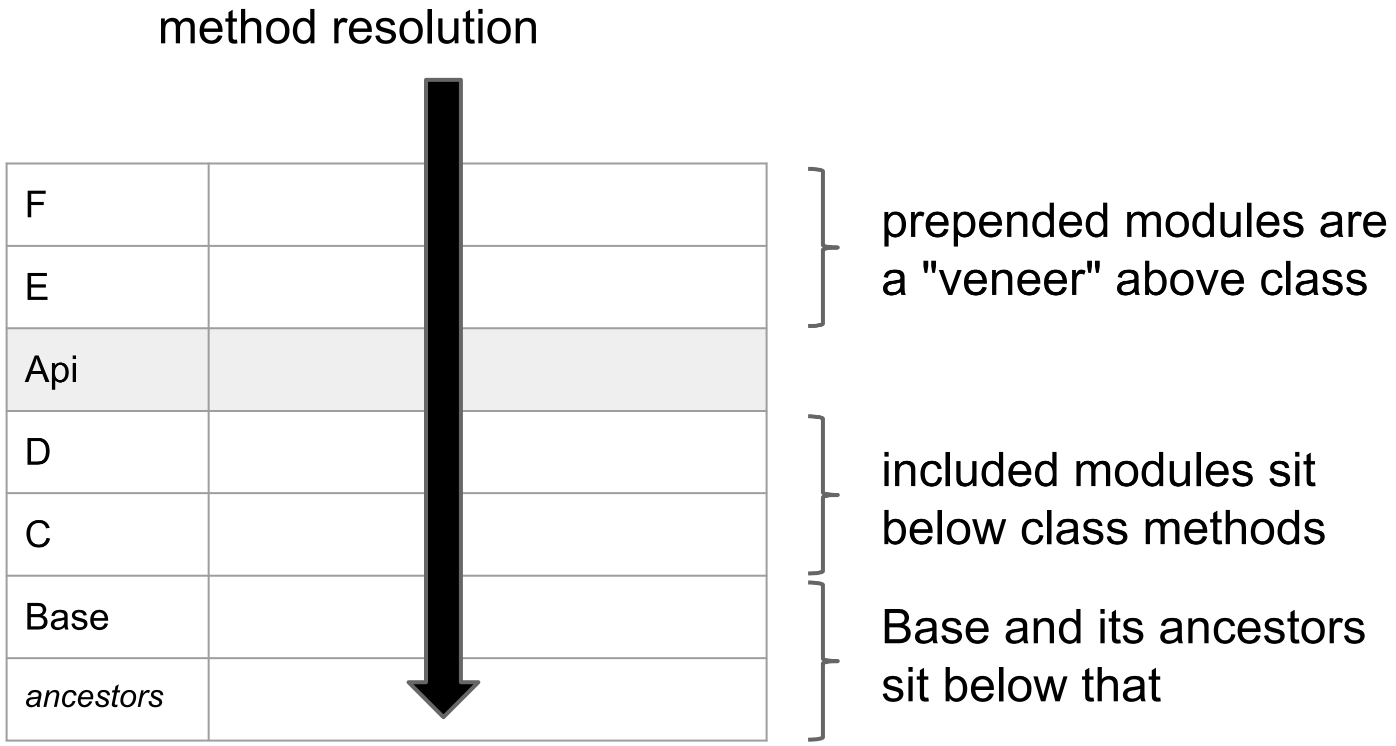
```
class Api < Base
  include C
  include D



end


>> Api.ancestors
=> ???
```

include vs. prepend


method resolution
F
E
Api
D
C
Base
ancestors
prepended modules are a "veneer" above class
included modules sit below class methods
Base and its ancestors sit below that

Case Study 1: ParseFigLeaf

Case Study 2: PublishSuccessMetric

Case Study 3: MethodJournal

?