Better Composition With

Traits

Joe Corcoran • corcoran.io • @josephcorcoran

```ruby
class Bike
  def wheels; 2 end
end
```

```ruby
class Bike
  def wheels; 2 end
end

class Track < Bike
  def gearing; :fixed end
  def brakes; [] end
end
```

```ruby
class Bike
  def wheels; 2 end
end

class Track < Bike
  def gearing; :fixed end
  def brakes; [] end
end

track = Track.new
track.wheels  # => 2
track.gearing # => :fixed
track.brakes  # => []
```

```ruby
class Track < Bike
  def gearing; :fixed end
  def brakes; [] end
end

class Fixie < Track
  def brakes; [:front, :back] end
end
```

```ruby
class Track < Bike
  def gearing; :fixed end
  def brakes; [] end
end

class Fixie < Track
  def brakes; [:front, :back] end
end

fixie = Fixie.new
fixie.wheels  # => 2
fixie.gearing # => :fixed
fixie.brakes  # => [:front, :back]
```

Composition

Prefer composition over inheritance.

– Somebody

```ruby
module Bike
  def wheels; 2 end
end

module Track
  def gearing; :fixed end
  def brakes; [] end
end

class Fixie
  include Bike
  include Track

  def brakes; [:front, :back] end
end
```

But

```ruby
module Bike
  def wheels; 2 end
  def brakes; [:front, :back] end
end

module Track
  def gearing; :fixed end
  def brakes; [] end
end

class Fixie
  include Bike
  include Track
end

fixie = Fixie.new
fixie.brakes # => ?
```

```ruby
module Bike
  def wheels; 2 end
  def brakes; [:front, :back] end
end

module Track
  def gearing; :fixed end
  def brakes; [] end
end

class Fixie
  include Bike
  include Track
end

fixie = Fixie.new
fixie.brakes # => []
```

```
Fixie.ancestors

# => [Fixie, Track, Bike, Object, ...]
```

Problems

Problems

- Implied design

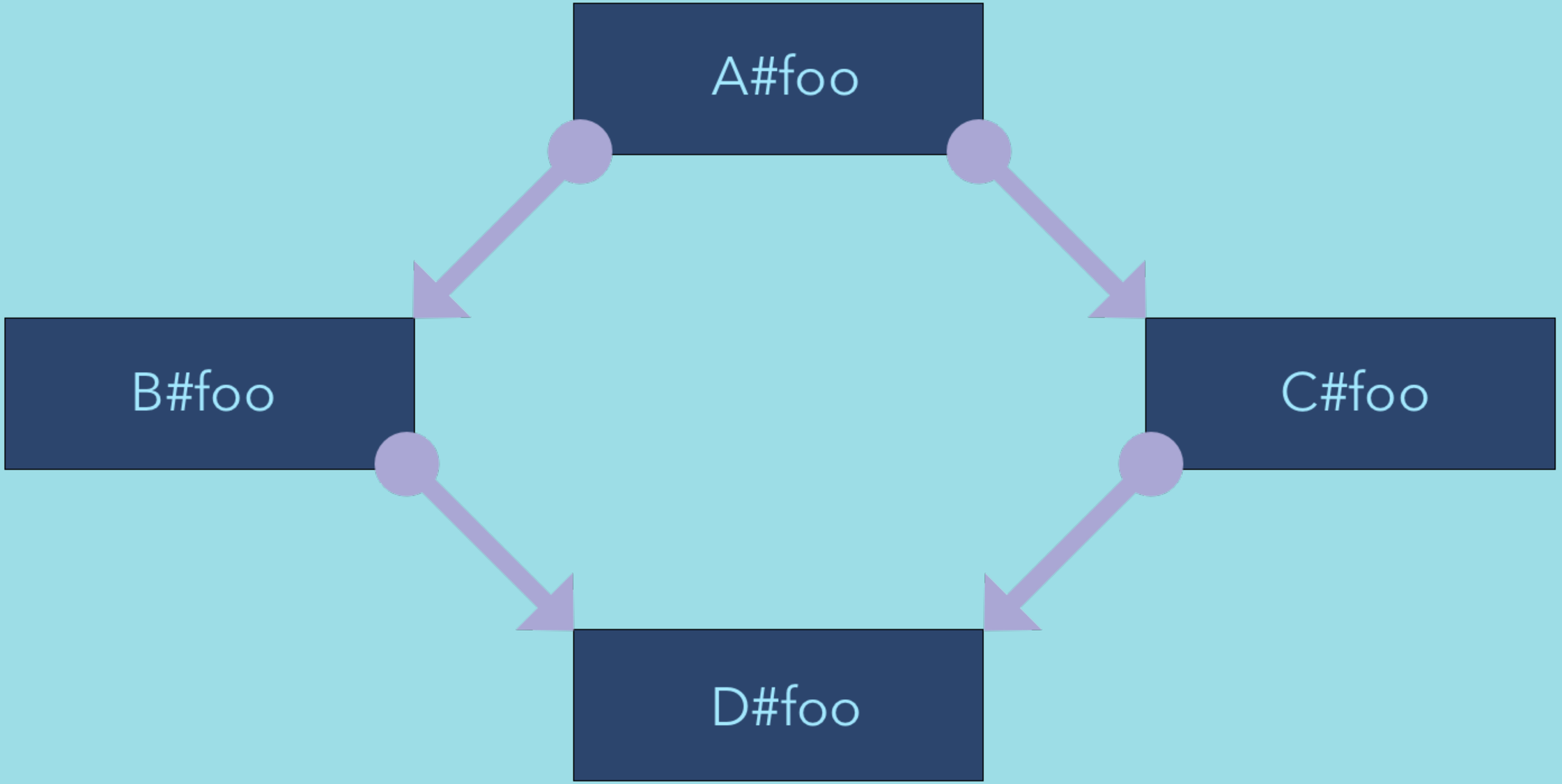
Problems

- Implied design
- Maintenance is harder

Problems

- Implied design
- Maintenance is harder
- We need tests to cover our basic design

The diamond problem


A#foo
B#foo
C#foo
D#foo

Multiple inheritance is good, but there is no good way to do it.

– Steve Cook

Traits

Traits: A Mechanism for Fine-grained Reuse

Ducasse, Nierstrasz, Schärli, Wuyts and Black

Software Composition Group
University of Berne, 2006

Traits

- Finite method dictionaries

Traits

- Finite method dictionaries
- Composable

Traits

- Finite method dictionaries
- Composable
- Can be summed with other traits

$$a \to m1$$

$$\{a \to m1\}$$

$$\{a \to m1, b \to m2\} + \{a \to m1, b \to m3\} = \{a \to m1, b \to \top\}$$

Resolving conflicts

Resolving conflicts

- Override the conflicting method

Resolving conflicts

- Override the conflicting method
- Exclude methods

Resolving conflicts

- Override the conflicting method
- Exclude methods
- Alias methods

Flattening

Flattening

- A *well-defined* class...

Flattening

- A *well-defined* class has no conflicting methods after composition

Flattening

- A *well-defined* class has no conflicting methods after composition
- No need for traits!

Scala

```
trait Bike {
  def wheels() : Int = {
    return 2
  }
  def brakes() : Vector[Symbol] = {
    return Vector('front, 'back)
  }
}

trait Track {
  def gearing() : Symbol = {
    return 'fixed
  }
  def brakes() : Vector[Symbol] = {
    return Vector()
  }
}

class Fixie extends Bike with Track {}
```

```
trait Bike {
  def wheels() : Int = {
    return 2
  }
  def brakes() : Vector[Symbol] = {
    return Vector('front, 'back)
  }
}

trait Track {
  def gearing() : Symbol = {
    return 'fixed
  }
  def brakes() : Vector[Symbol] = {
    return Vector()
  }
}

class Fixie extends Bike with Track {}
// => error: class Fixie inherits conflicting members
```

```
trait Bike {
  def wheels() : Int = {
    return 2
  }
  def brakes() : Vector[Symbol] = {
    return Vector('front, 'back)
  }
}

trait Track {
  def gearing() : Symbol = {
    return 'fixed
  }
  def brakes() : Vector[Symbol] = {
    return Vector()
  }
}

class Fixie extends Bike with Track {
  override def brakes() : Vector[Symbol] = {
    return Vector('front, 'back)
  }
}
```

```ruby
module Bike
  def wheels; 2 end
  def brakes; [:front, :back] end
end

module Track
  def gearing; :fixed end
  def brakes; [] end
end

class Fixie
  include Bike
  include Track
end
```

```ruby
module Bike
  def wheels; 2 end
  def brakes; [:front, :back] end
end

module Track
  def gearing; :fixed end
  def brakes; [] end
end

class Fixie
  compose Bike,
          Track.methods(exclude: :brakes)
end
```

```
source 'https://rubygems.org'

gem 'fabrik'
```

```ruby
class Bike
  extend Fabrik::Trait

  provides do
    def wheels; 2 end
    def brakes; [:front, :back] end
  end
end

class Track
  extend Fabrik::Trait

  provides do
    def gearing; :fixed end
    def brakes; [] end
  end
end

class Fixie
  extend Fabrik::Composer

  compose Bike,
          Track[exclude: :brakes]
end
```

```
class Fixie
  extend Fabrik::Composer

  compose Bike,
          Track[aliases: { brakes: :stopping_machines }]
end
```

```ruby
class Fixie
  extend Fabrik::Composer

  def brakes; [:front] end

  compose Bike, Track
end
```

```ruby
class Bike
  extend Fabrik::Trait

  provides do
    def wheels; 2 end
    def brakes; [:front, :back] end
  end
end
```

```ruby
module Foo
  def bar; :baz end
end

bar = Foo.instance_method(:bar)
# => #<UnboundMethod: Foo#bar>
```

```ruby
module Foo
  def bar; :baz end
end

bar = Foo.instance_method(:bar)

class Qux; end
Qux.send(:define_method, :quux, bar)

Qux.new.quux
# => :baz
```

```ruby
module Foo
  def bar; :baz end
end

bar = Foo.instance_method(:bar)

class Qux; end
Qux.send(:define_method, :quux, bar)
# => TypeError: bind argument must be a subclass of Foo
```

$$\{a \to m1, b \to m2\} + \{a \to m1, b \to m3\} = \{a \to m1, b \to \top\}$$

```ruby
module Foo
  def bar; :baz end
end

b1 = Foo.instance_method(:bar)
b2 = Foo.instance_method(:bar)

b1 == b2 # => true
```

```ruby
module Foo
  def bar; :baz end
end

class Qux
  extend Fabrik::Trait
  provides_from Foo, :bar
end
```

```ruby
module Paintwork
  def paint!
    [:red, :green, :blue].sample
  end
end

class Bike
  extend Fabrik::Trait
  provides_from Paintwork, :paint!
end

class Track
  extend Fabrik::Trait
  provides_from Paintwork, :paint!
end

class Fixie
  extend Fabrik::Composer
  compose Bike, Track
end

fixie = Fixie.new
fixie.paint! # => :green
```

Links

- Traits: A Mechanism for Fine-grained Reuse (PDF)
- github.com/joecorcoran/fabrik
- github.com/joecorcoran/talks/tree/master/traits
- corcoran.io
- @josephcorcoran