Managing Dependencies

Tight vs. Loose Coupling

Author: Torey Hickman

Phase 1: Day 9

Side Effects of Tight Coupling

- Rigidity
- Fragility
- Immobility

When is There a Dependency

- Knowing the name of another class
- Knowing the name of messages to send to other objects
- Knowing required arguments
- Knowing the order of arguments

Minimizing Effects of Dependence

- Isolate dependencies
- Remove argument order dependencies
- Rely on the more stable object
- Inject dependencies
- Law of Demeter

Isolate Dependencies

- Make the dependency obvious
- Localize the dependency

Isolate Dependencies

```
class Baker
  def prepare_batch
    self.batches << Batch.new
  end
end
```

Isolate Dependencies

```ruby
class Baker
  def prepare_batch
    self.batches << Batch.new
  end
end
```

```ruby
class Baker
  def prepare_batch
    self.batches << new_batch
  end

  def new_batch
    Batch.new
  end
end
```

Argument Order Dependencies

- Protect from changes in implementation
- More readable code

Argument Order Dependencies

```
class Oven
  def bake
    contents.bake(350, 10)
  end
end

class Cookie
  def bake(temp, time)
    # do something with args
  end
end
```

Argument Order Dependencies

```
class Oven
  def bake
    contents.bake(350, 10)
  end
end

class Cookie
  def bake(temp, time)
    # do something with args
  end
end
```

```
class Oven
  def bake
    contents.bake(350, 10)
  end
end

class Cookie
  def bake(time, temp)
    # do something with args
  end
end
```

Argument Order Dependencies

```
class Oven
  def bake
    contents.bake(350, 10)
  end
end

class Cookie
  def bake(temp, time)
    # do something with args
  end
end
```

```
class Oven
  def bake
    contents.bake(350, 10)
  end
end

class Cookie
  def bake(time, temp, rack)
    # do something with args
  end
end
```

Argument Order Dependencies

```
class Oven
  def bake
    contents.bake(temp: 350, time: 10)
  end
end

class Cookie
  def bake(args)
    #  Do something with args[:time] and args[:temp]
    #  args.fetch(:rack, :middle)
  end
end
```

Rely on More Stable Objects

- Depend on what is least likely to change

Rely on More Stable Objects

- Depend on what is least likely to change

  Ruby classes: Array, Hash, String, etc. …

Inject Dependencies

- Favor the abstract over the concrete

Inject Dependencies

```
class Oven
  def add
    contents = Cookie.new
  end

  def bake
    contents.bake
  end
end

oven = Oven.new
oven.add
```

Inject Dependencies

```
class Oven
  def add
    contents = Cookie.new
  end

  def bake
    contents.bake
  end
end

oven = Oven.new
oven.add
```

```
class Oven
  def add(bakeable)
    contents = bakeable
  end

  def bake
    contents.bake
  end
end

oven = Oven.new
oven.add(Cookie.new)
```

Inject Dependencies

- An oven doesn’t need cookies, it needs something that responds to bake.

Inject Dependencies

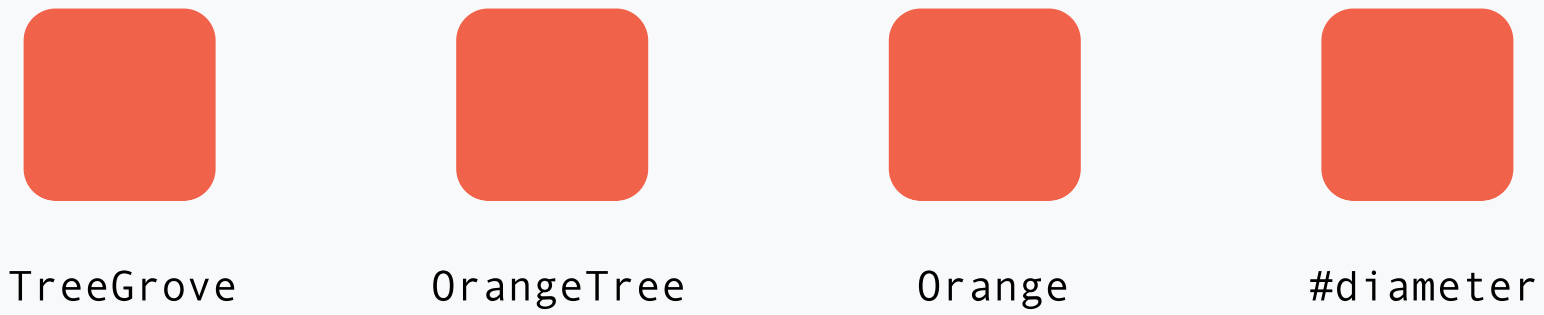
```ruby
class Oven
  def add(bakeable)
    contents = bakeable
  end

  def bake
    contents.bake
  end
end

oven = Oven.new
oven.add(Cheesecake.new)
```

```ruby
class Cookie
  def bake
  end
end

class Bread
  def bake
  end
end

class Cheesecake
  def bake
  end
end
```

Law of Demeter

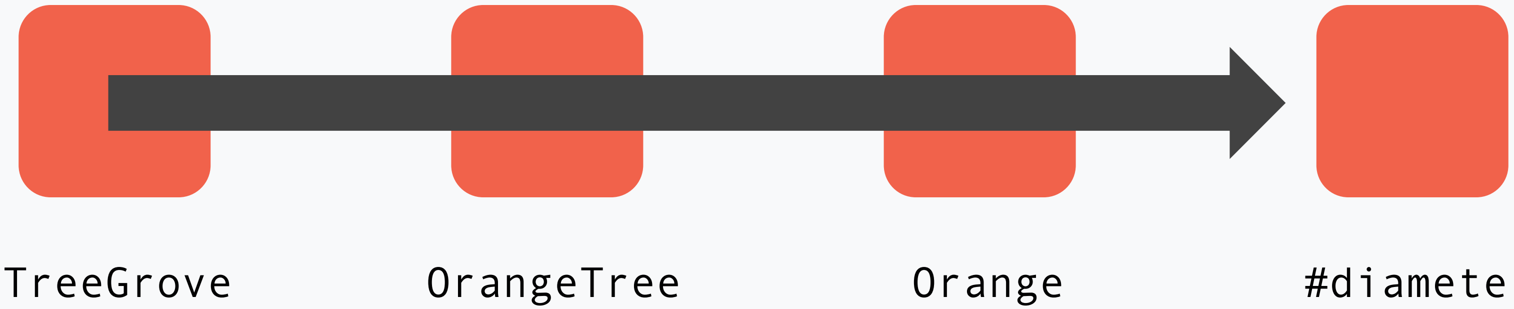
- Only talk to neighbors
- Limit how far you reach into objects

Law of Demeter



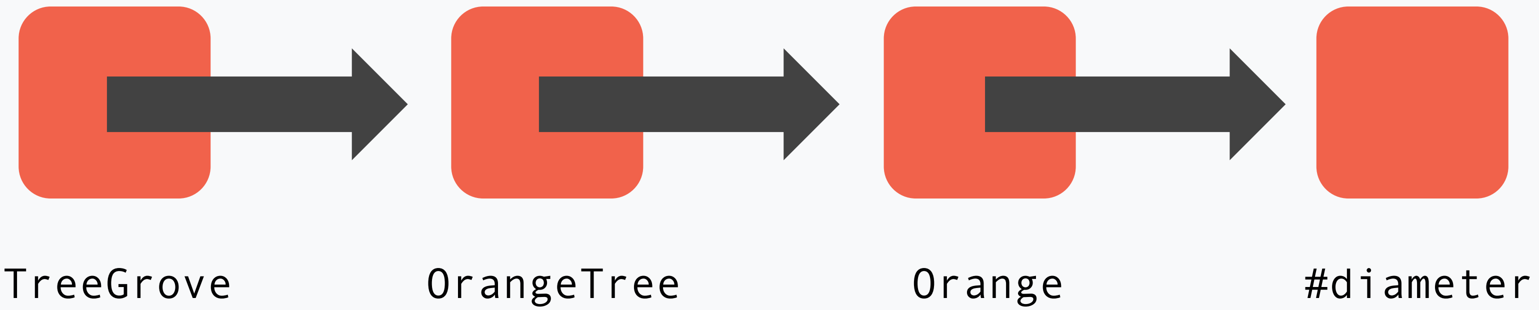
- How to get the diameter of any one orange?

Law of Demeter



TreeGrove asks directly

Law of Demeter



TreeGrove asks Tree asks Orange

Managing Dependencies