Reducing duplication

Designing and Maintaining Software (DAMS)

Louis Rose

Tactics

Accentuate similarities to find differences

Favour composition over inheritance

Know when to reach for advanced tools
(metaprogramming, code generation)

Accentuate similarities

Aim: make similar code identical to find differences

```ruby
class StuffedCrust
  def bake(dough)
    base = stuff(roll(dough))
    raw_pizza = top(base)
    cook(raw_pizza, 12.minutes)
  end
end
```

```ruby
class DeepPan
  def bake(dough)
    base = roll(dough)
    raw_pizza = top(base)
    cook(raw_pizza, 12.minutes)
  end
end
```

Accentuate similarities

Aim: make similar code identical to find differences

```
class StuffedCrust
  def bake(dough)
    base = prepare(dough)
    raw_pizza = top(base)
    cook(raw_pizza, 12.minutes)
  end
end
```

```
class DeepPan
  def bake(dough)
    base = prepare(dough)
    raw_pizza = top(base)
    cook(raw_pizza, 12.minutes)
  end
end
```

Accentuate similarities

Aim: make similar code identical to find differences

```
class StuffedCrust
  def bake(dough)
    base = prepare(dough)
    raw_pizza = top(base)
    cook(raw_pizza, 12.minutes)
  end

   def prepare(dough)
    stuff(roll(dough))
  end
end
```

```
class DeepPan
  def bake(dough)
    base = prepare(dough)
    raw_pizza = top(base)
    cook(raw_pizza, 12.minutes)
  end

   def prepare(dough)
     roll(dough)
  end
end
```

Once and Only Once

Now we can specify the baking logic in one place

```
class StuffedCrust < Pizza
    def prepare(dough)
   stuff(roll(dough))
  end
end
```

```
class DeepPan < Pizza
    def prepare(dough)
      roll(dough)
  end
end
```

```
class Pizza
  def bake(dough)
    base = prepare(dough)
    raw_pizza = top(base)
    cook(raw_pizza, 12.minutes)
  end
end
```

Template Method Pattern

Defer some parts of an algorithm to subclasses

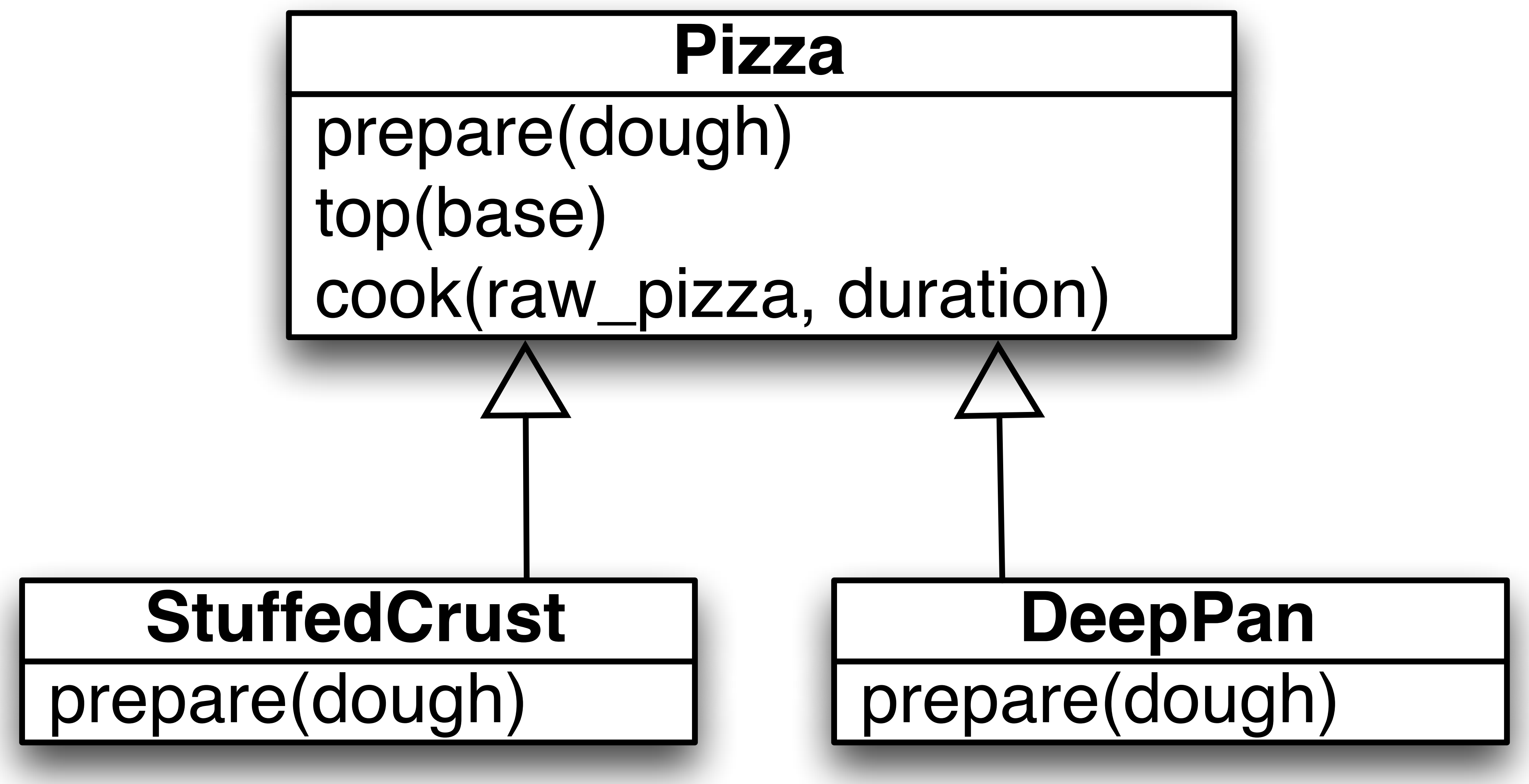
```
class StuffedCrust < Pizza
   def prepare(dough)
  stuff(roll(dough))
 end
end
```

```
class DeepPan < Pizza
   def prepare(dough)
     roll(dough)
 end
end
```

```
class Pizza
 def bake(dough)
  base = prepare(dough)
  raw_pizza = top(base)
  cook(raw_pizza, 12.minutes)
 end
end
```

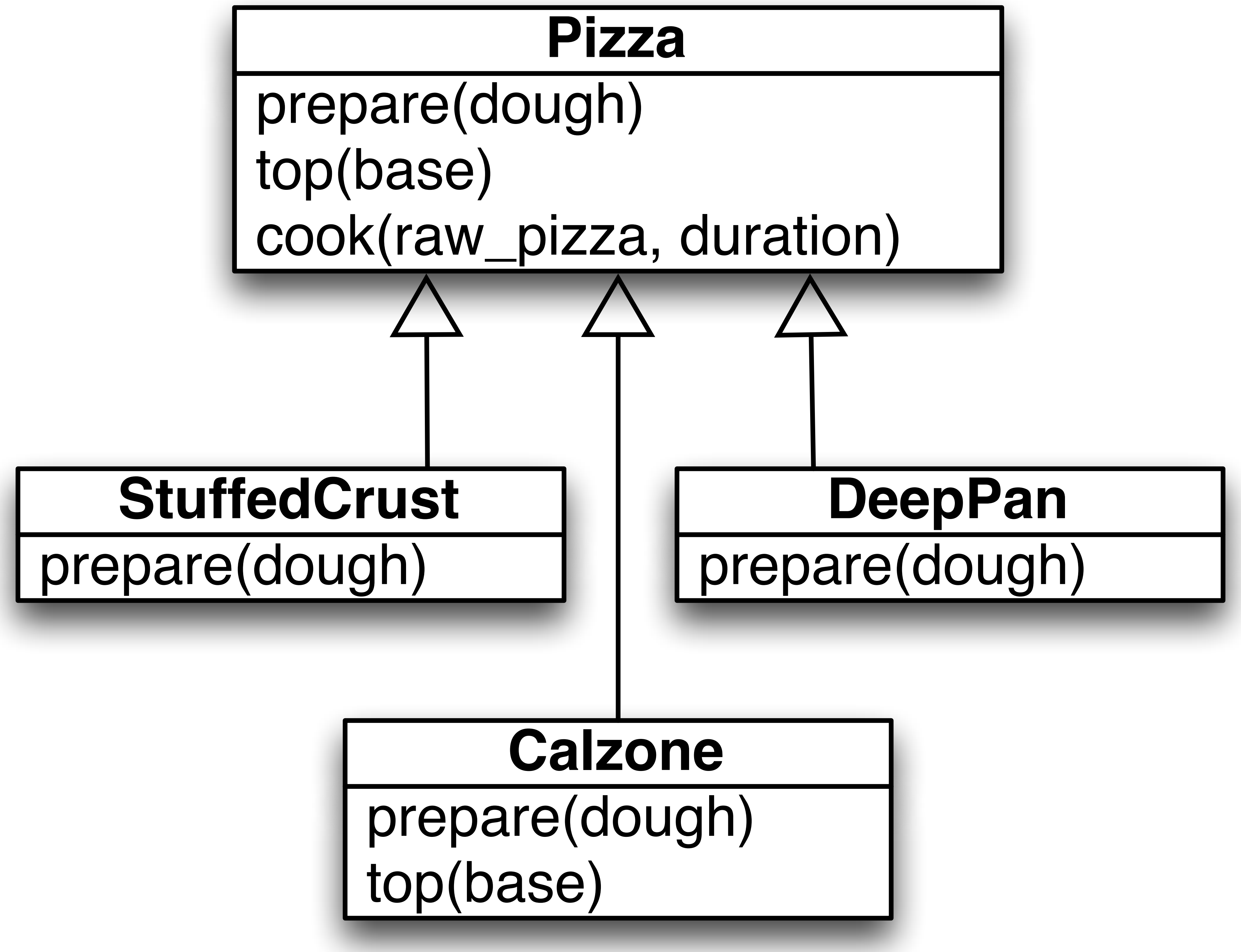
Template Method Pattern

Defer some parts of an algorithm to subclasses



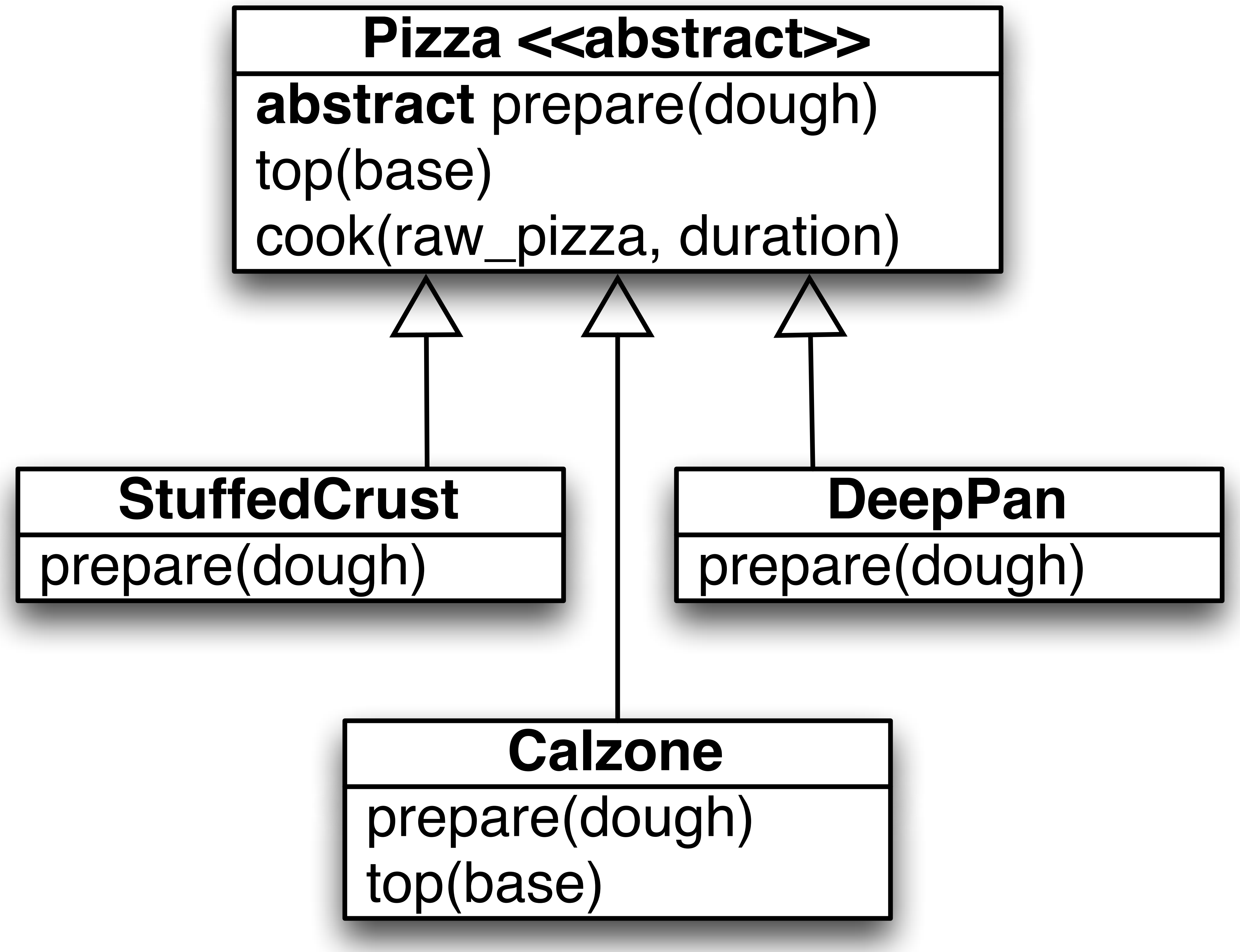
Template Method Pattern

Defer some parts of an algorithm to subclasses



Template Method Pattern

Defer some parts of an algorithm to subclasses



Template Method Caveats

Some OO languages don’t support abstract classes

Template Method Caveats

Some OO languages don’t support abstract classes

including Ruby

Template Method Pattern

The best we can is to raise in “abstract” methods

```
class Pizza
  def bake(dough)
    base = prepare(dough)
    raw_pizza = top(base)
    cook(raw_pizza, 12.minutes)
  end

   def prepare(dough)
    raise "No implementation”
  end
end
```

Template Method Caveats

Some OO languages don’t support abstract classes

Can be difficult to communicate which methods are to be overridden

Breaks down if there is more than one axis of change…

Feature Request

Some customers like their pizzas to be “well done”

Feature Request

Some customers like their pizzas to be “well done”

```
class Pizza
  def bake(dough)
    base = prepare(dough)
    raw_pizza = top(base)
    cook(raw_pizza, 12.minutes)
  end
end
```

```
class WellDone
  def bake(dough)
         base = prepare(dough)
    raw_pizza = top(base)
    cook(raw_pizza, 15.minutes)
  end
end
```

Feature Request

Some customers like their pizzas to be “well done”

```
class Pizza
  def bake(dough)
    base = prepare(dough)
    raw_pizza = top(base)
    cook(raw_pizza, bake_time)
  end

  def bake_time
    12.minutes
  end
end
```

```
class WellDone
  def bake(dough)
        base = prepare(dough)
    raw_pizza = top(base)
    cook(raw_pizza, bake_time)
  end

  def bake_time
    15.minutes
  end
end
```

Feature Request

Some customers like their pizzas to be “well done”

```
class Pizza
  def bake(dough)
    base = prepare(dough)
    raw_pizza = top(base)
    cook(raw_pizza, bake_time)
  end

  def bake_time
    12.minutes
  end
end
```

```
class WellDone < Pizza
  def bake_time
    15.minutes
  end
end
```

But what about prepare?

Recall that Pizza doesn’t provide an implementation

```
class Pizza
  def bake(dough)
    base = prepare(dough)
    raw_pizza = top(base)
    cook(raw_pizza, bake_time)
  end

  def bake_time
    12.minutes
  end

   def prepare(dough)
    raise "No implementation”
  end
end
```

```
class WellDone < Pizza
  def bake_time
    15.minutes
  end
end
```

Inheritance to the rescue

Recall that Pizza doesn’t provide an implementation

```
class WellDoneDeepPan < DeepPan
  def bake_time
    15.minutes
  end
end

class WellDoneStuffedCrust < StuffedCrust
  def bake_time
    15.minutes
  end
end
```

Once and Only Once

If only we had multiple inheritance…

```
class WellDoneDeepPan < DeepPan
  def bake_time
    15.minutes
  end
end

class WellDoneStuffedCrust < StuffedCrust
  def bake_time
    15.minutes
  end
end
```

~~Once and Only Once~~

~~Ruby modules to the rescue~~

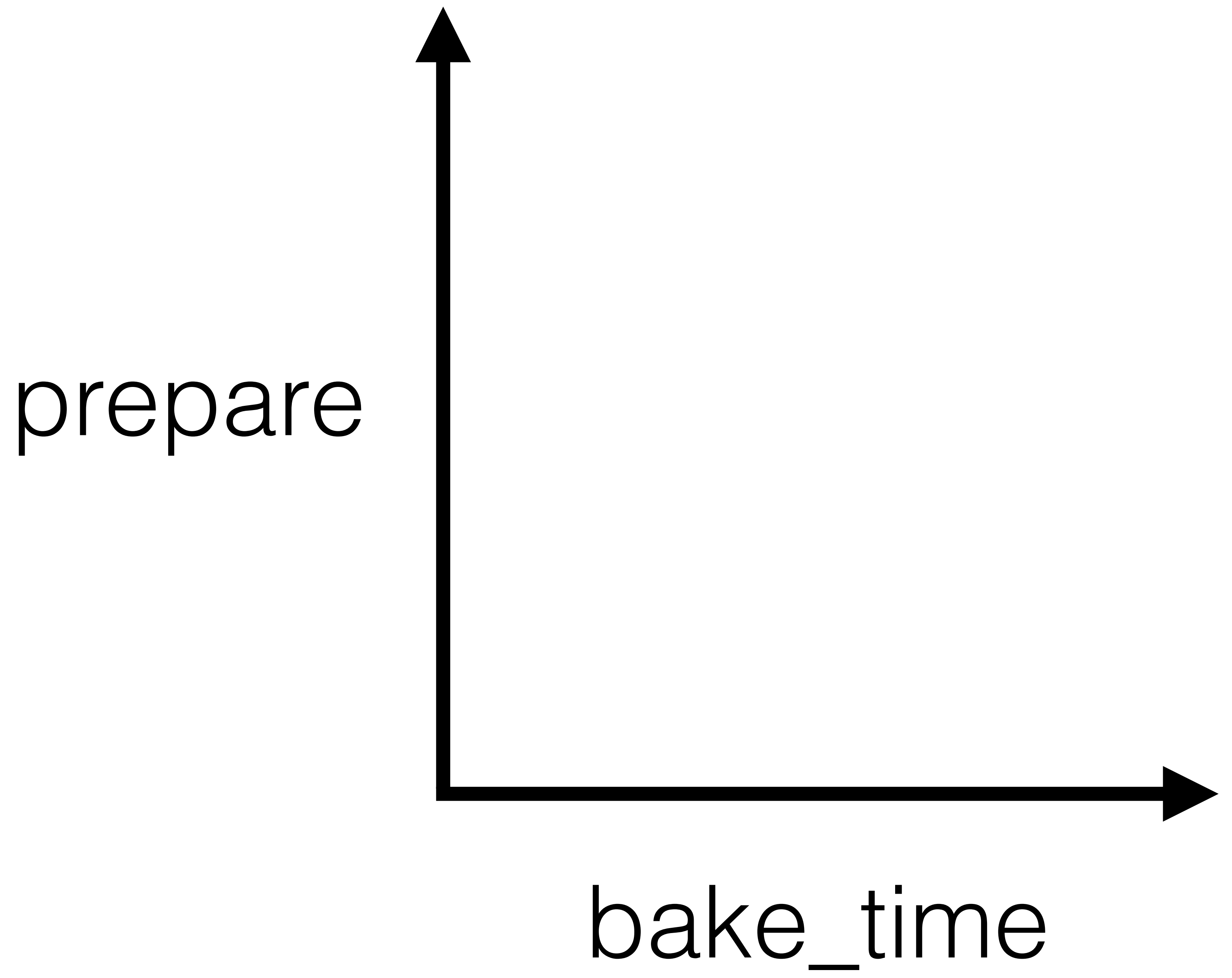
```
class WellDoneDeepPan < DeepPan
  include WellDone
end

class WellDoneStuffedCrust < StuffedCrust
  include WellDone
end

module WellDone
  def bake_time
    15.minutes
  end
end
```

Favour composition…

… over inheritance when there is >1 axis of change



StuffedCrust is-a Pizza?

Arguably not.

```
class Pizza
 def bake(dough)
   base = prepare(dough)
   raw_pizza = top(base)
   cook(raw_pizza, 12.minutes)
 end

  def prepare(dough)
   raise "No implementation"
 end
end
```

```
class StuffedCrust < Pizza
   def prepare(dough)
   stuff(roll(dough))
  end
end

class DeepPan < Pizza
   def prepare(dough)
     roll(dough)
  end
end
```

StuffedCrust is-a Pizza?

Arguably not.

```
class Pizza
 def initialize(recipe)
   @recipe = recipe
 end

 def bake(dough)
   base = @recipe.prepare(dough)
   raw_pizza = top(base)
   cook(raw_pizza, 12.minutes)
 end

  def prepare(dough)
   raise "No implementation”
 end
end
```

```
class StuffedCrust < Pizza
    def prepare(dough)
   stuff(roll(dough))
  end
end

class DeepPan < Pizza
    def prepare(dough)
      roll(dough)
  end
end
```

StuffedCrust is-a Pizza?

Arguably not.

```
class Pizza
  def initialize(recipe)
    @recipe = recipe
  end

  def bake(dough)
    base = @recipe.prepare(dough)
    raw_pizza = top(base)
    cook(raw_pizza, 12.minutes)
  end

   def prepare(dough)
    raise "No implementation”
  end
end
```

```
class StuffedCrust
    def prepare(dough)
    stuff(roll(dough))
  end
end

class DeepPan
    def prepare(dough)
      roll(dough)
  end
end
```

StuffedCrust is-a Pizza?

Arguably not.

```
class Pizza
  def initialize(recipe)
    @recipe = recipe
  end

  def bake(dough)
    base = @recipe.prepare(dough)
    raw_pizza = top(base)
    cook(raw_pizza, 12.minutes)
  end
end
```

```
class StuffedCrust
    def prepare(dough)
    stuff(roll(dough))
  end
end

class DeepPan
    def prepare(dough)
      roll(dough)
  end
end
```

StuffedCrust is-a Pizza?

Arguably not.

```
class Pizza
  def initialize(recipe)
    @recipe = recipe
  end

  def bake(dough)
    base = @recipe.prepare(dough)
    raw_pizza = top(base)
    cook(raw_pizza, 12.minutes)
  end
end
```

```
class StuffedCrust
    def prepare(dough)
    stuff(roll(dough))
  end
end

class DeepPan
    def prepare(dough)
      roll(dough)
  end
end
```

```
Pizza.new(StuffedCrust.new)
```

Back to WellDone

Recall that latest feature request…

```
class Pizza
  def initialize(recipe)
    @recipe = recipe
  end

  def bake(dough)
    base = @recipe.prepare(dough)
    raw_pizza = top(base)
    cook(raw_pizza, 12.minutes)
  end
end
```

```
class WellDone
    def bake_time
    15.minutes
  end
end
```

Back to WellDone

Recall that latest feature request…

```
class Pizza
  def initialize(recipe)
    @recipe = recipe
  end

  def bake(dough)
    base = @recipe.prepare(dough)
    raw_pizza = top(base)
    cook(raw_pizza, 12.minutes)
  end
end
```

```
class WellDone
    def bake_time
    15.minutes
  end
end

class Medium
    def bake_time
    12.minutes
  end
end
```

Back to WellDone

Recall that latest feature request…

```
class Pizza
  def initialize(recipe, doneness = Medium.new)
    @recipe = recipe
    @doneness = doneness
  end

  def bake(dough)
    base = @recipe.prepare(dough)
    raw_pizza = top(base)
    cook(raw_pizza, @doneness.bake_time)
  end
end
```

```
class WellDone
    def bake_time
    15.minutes
  end
end

class Medium
    def bake_time
    12.minutes
  end
end
```

Back to WellDone

Recall that latest feature request…

```
class Pizza
  def initialize(recipe, doneness = Medium.new)
    @recipe = recipe
    @doneness = doneness
  end
end
```

Back to WellDone

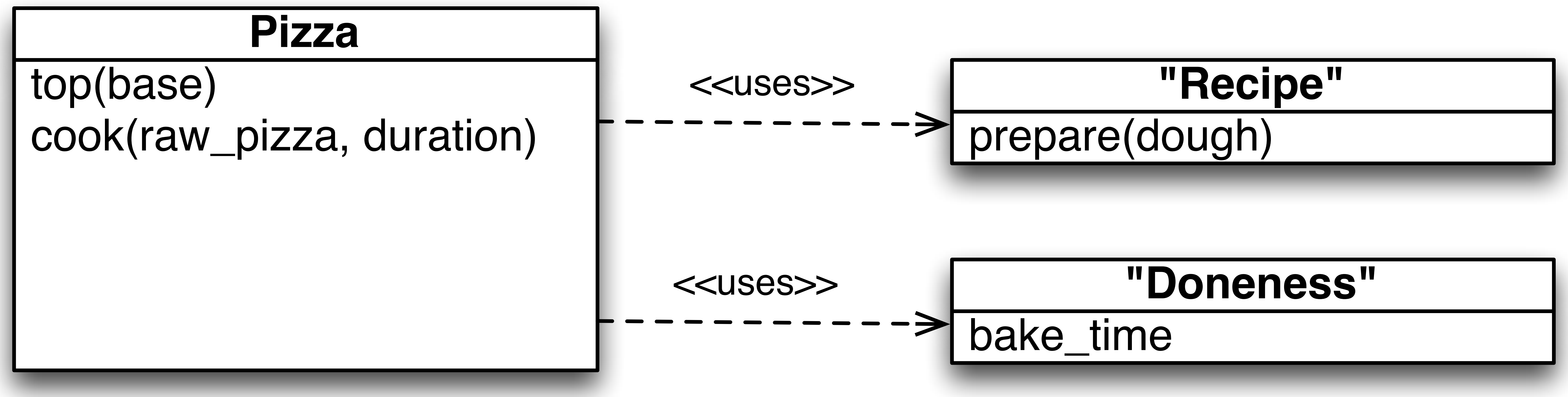
Recall that latest feature request…

```
class Pizza
  def initialize(recipe, doneness = Medium.new)
    @recipe = recipe
    @doneness = doneness
  end
end

Pizza.new(StuffedCrust.new)
Pizza.new(StuffedCrust.new, WellDone.new)
Pizza.new(DeepPan.new)
Pizza.new(DeepPan.new, WellDone.new)
```

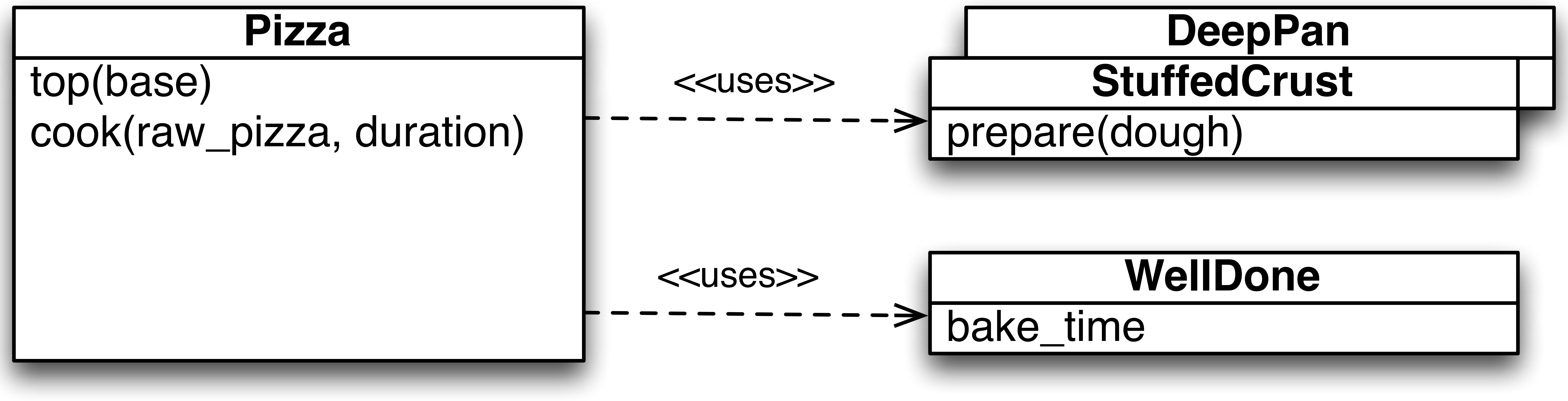
Strategy Pattern

Defer some parts of an algorithm to collaborators



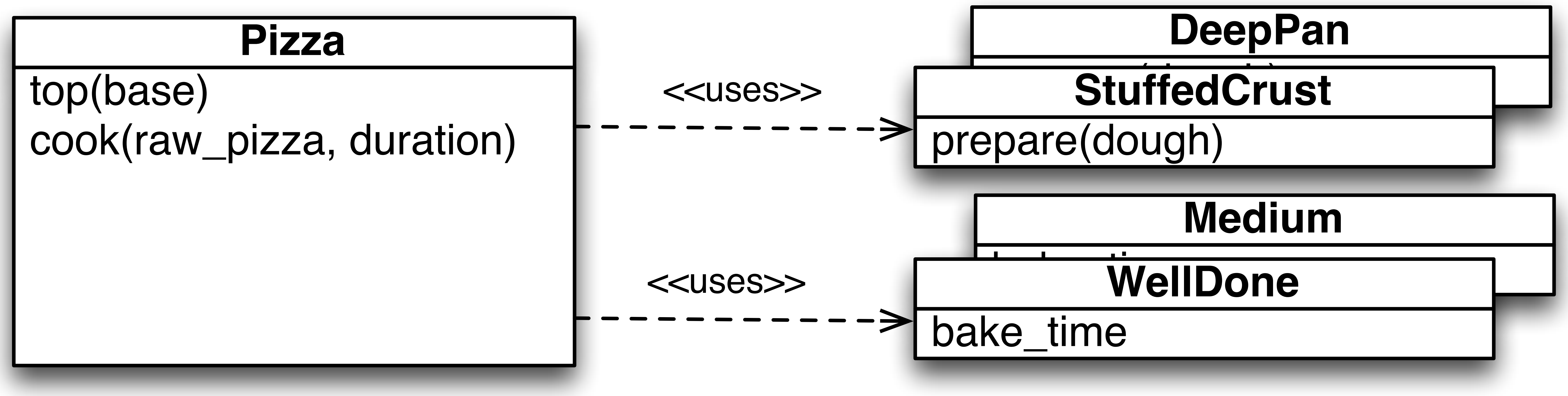
Open/Closed Principle

New functionality added without altering existing code



Open/Closed Principle

New functionality added without altering existing code



Summary

Refactor to isolate differences by
making code more alike

Use inheritance and Template Method
to isolate duplication single axis of change

Favour Strategies when
there are several axes of change

Also important

“Do nothing” can be a variation point
when accentuating similarities.

Null Object pattern is, essentially,
a special form of Strategy.

“Nothing is Something”
Sandi Metz (RailsConf 2015)

Why not inject data?

```
class Pizza
 def initialize(recipe, doneness = Medium.new)
   @recipe = recipe
   @doneness = doneness
 end

 def bake(dough)
   base = @recipe.prepare(dough)
   raw_pizza = top(base)
   cook(raw_pizza, @doneness.bake_time)
 end
end
```

```
class WellDone
    def bake_time
   15.minutes
  end
end

class Medium
    def bake_time
   12.minutes
  end
end
```

Why not inject data?

```
class Pizza
  def initialize(recipe, bake_time = 12)
    @recipe = recipe
    @bake_time = bake_time
  end

  def bake(dough)
    base = @recipe.prepare(dough)
    raw_pizza = top(base)
    cook(raw_pizza, @bake_time)
  end
end
```

```
class WellDone
    def bake_time
    15.minutes
  end
end

class Medium
    def bake_time
    12.minutes
  end
end
```

Why not inject data?

```
class Pizza
  def initialize(recipe, bake_time = 12)
    @recipe = recipe
    @bake_time = bake_time
  end

  def bake(dough)
    base = @recipe.prepare(dough)
    raw_pizza = top(base)
    cook(raw_pizza, @bake_time)
  end
end
```

Why not inject data?

```
class Pizza
  def initialize(recipe, bake_time = 12)
    @recipe = recipe
    @bake_time = bake_time
  end

  def bake(dough)
    base = @recipe.prepare(dough)
    raw_pizza = top(base)
    cook(raw_pizza, @bake_time)
  end
end
```

```
Pizza.new(StuffedCrust.new, 15)
```

Why not inject data?

```
class Pizza
  def initialize(recipe, bake_time = 12)
    @recipe = recipe
    @bake_time = bake_time
  end

  def bake(dough)
    base = @recipe.prepare(dough)
    raw_pizza = top(base)
    cook(raw_pizza, @bake_time)
  end
end
```

```
Pizza.new(StuffedCrust.new, 15)


Pizza.new(StuffedCrust.new, 17)
Pizza.new(StuffedCrust.new, 42)
```