CONDITIONAL CODE *Smells*

ERIC ROBERTS | @EROBERTS

WHAT IS CONDITIONAL LOGIC?

Conditional statements are features of a programming language which perform different computations or actions depending on whether a programmer-specified boolean condition evaluates to true or false. Apart from the case of branch predication, this is always achieved by selectively altering the control flow based on some condition.

– Wikipedia

```
if thing
  puts 'hello'
else
  puts 'goodbye'
end
```

```
thing ? puts 'hello' : puts 'goodbye'
```

```
puts 'hello' if thing

puts 'goodbye' unless thing
```

```
case thing
when Hello
  puts 'hello'
when Goodbye
  puts 'goodbye'
else
  puts 'uh oh..'
end
```

SO, WHY'S THIS BAD?

GOOD QUESTION

IT'S NOT

BUT IT CAN BE

WHAT ARE CODE SMELLS?

Code smells are not *technically* incorrect and do not prevent the program from functioning.

Instead, they *indicate weaknesses* in design that may be slowing down development or increasing the risk of bugs in the future.

– Wikipedia

LET'S TAKE A LOOK

```
if thing
  puts 'hello'
else
  puts 'goodbye'
end
```

```ruby
if thing
  puts 'hello'
elsif other_thing
  puts 'auf wiedersehen'
else
  puts 'goodbye'
end
```

```ruby
if thing
  puts 'hello'
elsif other_thing
  puts 'auf wiedersehen'
elsif another_thing
  puts 'bonjour'
else
  puts 'goodbye'
end
```

```
if thing
  puts 'hello'
elsif other_thing
  if going
    puts 'auf wiedersehen'
  else
    puts 'hallo'
  end
elsif another_thing
  puts 'bonjour'
else
  puts 'goodbye'
end
```

Well, this is just getting out of hand, isn't it?

```ruby
def pricing_range_fold_and_save(new_range)
  pos = 0
  pricings = self.pricings
  other_range = Pricing.new
  if pricings.length == 0
    pricings.push(new_range)
  else
    pricings.each_with_index do |e, i|
      if i == 0 && new_range.start_date < e.start_date
        pricings.unshift(new_range)
        if new_range.end_date <= e.end_date
          e.start_date = new_range.end_date + 1.day
          break
        else
          while pricings[1].present? && new_range.end_date >= pricings[1].end_date
            pricings[1].destroy
            pricings.delete pricings[1]
          end
          if pricings[1].present?
            pricings[1].start_date = new_range.end_date + 1.day
          end
          break
        end
      elsif new_range.start_date <= e.end_date && new_range.start_date >= e.start_date
        if new_range.end_date < e.end_date
          other_range = e.dup
          other_range.start_date = new_range.end_date + 1.day
          if new_range.start_date == e.start_date
            e.destroy
            pricings.delete e
            i -= 1
          else
            e.end_date = new_range.start_date - 1.day
          end
          pricings.insert(i+1, new_range)
          pos = i+1
          pricings.insert(i+2, other_range)
          break
        else
          if new_range.start_date == e.start_date
            e.destroy
            pricings.delete e
            i -= 1
          else
            e.end_date = new_range.start_date - 1.day
          end
```

```
        pricings.insert(i+1, new_range)
        pos = i+1
        while pricings[i+2].present? && new_range.end_date >= pricings[i+2].end_date
          pricings[i+2].destroy
          pricings.delete pricings[i+2]
        end
        if pricings[i+2].present?
          pricings[i+2].start_date = new_range.end_date + 1.day
        end
        break
      end

    elsif i == pricings.size-1
      pricings[i].end_date = new_range.start_date-1.day
      pricings.push(new_range)
      break
    end
  end
end

pricings.each_with_index do |pricing, i|
  if i != pricings.size-1 && pricing.price ==pricings[i+1].price
    pricing.end_date = pricings[i+1].end_date
  end
  if i != 0 && pricing.end_date == pricings[i-1].end_date
    pricing.destroy
    pricings.delete pricing
  end
  if pricing.end_date < pricing.start_date
    pricing.destroy
    pricings.delete pricing
  end
end

pricings.each do |pricing|
  if pricing != pricings[pos]
    pricing.currency = pricings[pos].currency
    pricing.save
  end
end
pricings[pos].save
return pricings
end
```

e cable attaching you to the *Maray* is
ed to its limit. You have come to rest on a
near the canyon in the ocean floor that
t myth says leads to the lost city of Atlantis.
have an experimental diving suit designed
tect you from the intense pressure of the
You should be able to leave the *Seeker* and
e the sea bottom. The new suit contains a
r of the latest microprocessors enabling a
of useful functions. It even has a built-in
vith laser communicator. You can cut loose
he cable; the *Seeker* is self-propelled. You
w in another world. Remember, this is a
ous world, an unknown world.
agreed, you signal the *Maray*, "All systems
s awesome down here."

---

you decide to explore the ledge where the Seeker has come to rest, turn to page 6.

If you decide to cut loose from the Maray and dive with the Seeker into the canyon in the ocean floor, turn to page 4.

Carefully maneuvering the *Seeker* betw
walls of the canyon, you discover a large
hole. A stream of large bubbles flows stea
of the hole. The *Seeker* is equipped with s
equipment to analyze the bubbles. It also ha
equipment that can measure depth. The
covers close to 90% of the earth and is
unknown. Who knows where this hole migl

---

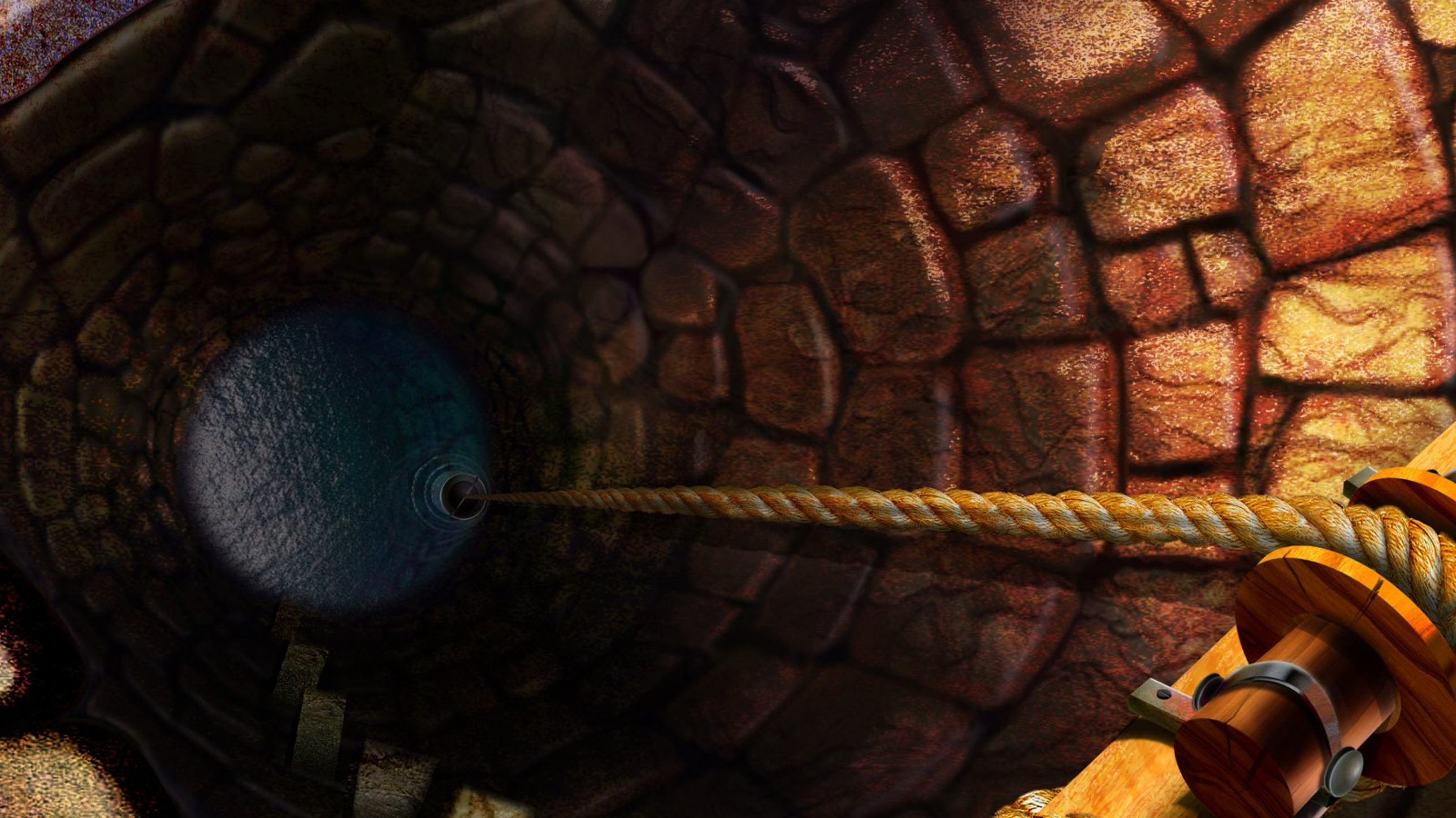
*If you decide to anal
bubbles, turn to*

*If you decide to tak
readings, turn to p*

REMEMBER CHOOSE YOUR OWN ADVENTURE BOOKS?

Conditional code can be like reading a choose your own adventure book from front to back.

This next bit is from Avdi Grimm's excellent book, Confident Ruby



SO, WE ALL KNOW WHAT'S GOING ON RIGHT?

OK, SO HOW DO I FIX IT?

COMMON INTERFACES

```ruby
class Mailing
  belongs_to :owner, polymorphic: true

  def company
  end
end
```

```ruby
class Company
  has_many :campaigns
  has_many :mailings, as: :owner
end

class Campaign
  belongs_to :company
  has_many :mailings, as: :owner
end
```

```ruby
class Mailing
  belongs_to :owner, polymorphic: true

  def company
    if owner.is_a? Company
      owner
    else
      owner.company
    end
  end
end
```

```ruby
class Mailing
  belongs_to :owner, polymorphic: true

  def company
    owner.company
  end
end
```

```ruby
class Mailing
  belongs_to :owner, polymorphic: true
  delegate :company, to: :owner
end
```

```ruby
class Company
  has_many :campaigns
  has_many :mailings, as: :owner

  def company
    self
  end
end
```

```
company = Company.new
mailing = Mailing.new(owner: company)
mailing.company

campaign = Campaign.new(company: company)
mailing = Mailing.new(owner: campaign)
mailing.company
```

USE INHERITANCE

Sandi Metz, ALL THE LITTLE THINGS

```ruby
class GildedRose
  attr_reader :name, :quality, :days_remaining

  def initialize(name, quality, days_remaining)
    @name, @quality, @days_remaining = name, quality, days_remaining
  en

  def tick
    [...]
  end
end
```

```
def tick
  days_remaining -= 1

  if name == 'Aged Brie'
    quality -= 1
  elsif name == 'Sulfuras, Hand of Ragnaros'
    quality += 1
  end
end
```

```ruby
class GildedRose
  attr_reader :name, :quality, :days_remaining

  def initialize(name, quality, days_remaining)
    @name, @quality, @days_remaining = name, quality, days_remaining
  en

  def tick
    days_remaining -= 1
  end
end
```

```ruby
class AgedBrie < GildedRose
  def tick
    super
    quality -= 1
  end
end
```

```ruby
class Sulfuras < GildedRose
  def tick
    super
    quality += 1
  end
end
```

```ruby
class GildedRose
  [...]

  def self.klass_for(name)
    {
      'Aged Brie' => AgedBrie,
      'Sulfuras, Hand of Ragnaros' => Sulfuras
    }
  end
end
```

```
thing = GildedRose.klass_for('Aged Brie')
thing.tick

thing = GildedRose.klass_for('Sulfuras, Hand of Ragnaros')
thing.tick
```

USE SPECIAL CASE OBJECTS

```
if current_user
  "Hello, #{current_user.name}"
else
  "Hello, Guest"
end
```

```ruby
def user
  current_user || GuestUser.new
end

class GuestUser
  def name
    "Guest"
  end
end
```

```
"Hello, #{user.name}"
```

USE HASH#FETCH

Avdi Grimm, CONFIDENT RUBY

```ruby
def create_user(options)
  login = options[:login]

  unless login
    raise ArgumentError, 'login must be supplied'
  end
end
```

```ruby
def create_user(options)
  login = options.fetch(:login)
end
```

```ruby
def create_user(options)
  login = options.fetch(:login, 'default_login')
end
```

```
things = {
  foo: {
    bar: :baz
  }
}
```

```ruby
if things[:foo]
  things[:foo][:bar]
end
```

```ruby
things.fetch(:foo, { bar: nil })[:bar]
```

USE ENUMERABLE

This one's going to take a little longer...

```ruby
def pricing_range_fold_and_save(new_range)
  pos = 0
  pricings = self.pricings
  other_range = Pricing.new
  if pricings.length == 0
    pricings.push(new_range)
  else
    pricings.each_with_index do |e, i|
      if i == 0 && new_range.start_date < e.start_date
        pricings.unshift(new_range)
        if new_range.end_date <= e.end_date
          e.start_date = new_range.end_date + 1.day
          break
        else
          while pricings[1].present? && new_range.end_date >= pricings[1].end_date
            pricings[1].destroy
            pricings.delete pricings[1]
          end
          if pricings[1].present?
            pricings[1].start_date = new_range.end_date + 1.day
          end
          break
        end
      elsif new_range.start_date <= e.end_date && new_range.start_date >= e.start_date
        if new_range.end_date < e.end_date
          other_range = e.dup
          other_range.start_date = new_range.end_date + 1.day
          if new_range.start_date == e.start_date
            e.destroy
            pricings.delete e
            i -= 1
          else
            e.end_date = new_range.start_date - 1.day
          end
          pricings.insert(i+1, new_range)
          pos = i+1
          pricings.insert(i+2, other_range)
          break
        else
          if new_range.start_date == e.start_date
            e.destroy
            pricings.delete e
            i -= 1
          else
            e.end_date = new_range.start_date - 1.day
          end
```

ADDING A NEW PRICING

SCREWED UP THE REST OF MY YEAR

Time to start reading...

```
def pricing_range_fold_and_save(new_range)
  pos = 0
  pricings = self.pricings
  other_range = Pricing.new
  if pricings.length == 0
    pricings.push(new_range)
  else
```

```
pricings.each_with_index do |e, i|
  if i == 0 && new_range.start_date < e.start_date
    pricings.unshift(new_range)
    if new_range.end_date <= e.end_date
      e.start_date = new_range.end_date + 1.day
      break
    else
      while pricings[1].present? && new_range.end_date >= pricings[1].end_date
        pricings[1].destroy
        pricings.delete pricings[1]
      end
      if pricings[1].present?
        pricings[1].start_date = new_range.end_date + 1.day
      end
      break
    end
```

OK, enough of that. What do the tests say?

```ruby
require 'spec_helper'

describe Property do
  pending "add some examples to (or delete) #{__FILE__}"
end
```

Alrighty then...

SO, WHAT IS THIS SUPPOSED TO DO?

MAGIC

1 - If the new price completely covers an old price, delete the old price.

2 - Adjust overlapping pricings.

3 - Split prices that fully contain the new price.

4 - Expand the range to meet the new price.

5 - Combine pricings where the pricings match.

```ruby
class PricingRange
  attr_reader :property

  def initialize(property)
    @property = property
  end

  def add(pricing)
    delete_complete_overlaps(pricing)
    adjust_overlaps(pricing)
    split_containing(pricing)
    expand_to_meet(pricing)
    coalesce_pricings
  end
end
```

```
class PricingRange
  [...]

  def add(pricing)
    delete_complete_overlaps(pricing)
    [...]
  end
end
```

DELETE COMPLETE OVERLAPS

```ruby
def delete_complete_overlaps(new)
  pricings.each do |old|
    if new.start_date <= old.start_date && new.end_date >= old.end_date
      pricings.delete old
    end
  end
end
```

```ruby
def delete_complete_overlaps(new)
  to_delete = pricings.select do |old|
    new.start_date <= old.start_date && new.end_date >= old.end_date
  end

  pricings.delete to_delete
end
```

```
new.start_date <= old.start_date && new.end_date >= old.end_date
```

```ruby
class PricingRange
  [...]

  def contains?(pricing1, pricing2)
    pricing1.start_date <= pricing2.start_date && pricing1.end_date >= pricing2.end_date
  end
end
```

```ruby
class Pricing
  [...]

  def contains?(other)
    self.start_date <= other.start_date && self.end_date >= other.end_date
  end
end
```

```ruby
def delete_complete_overlaps(new)
  to_delete = pricings.select { |old| new.contains? old }
  pricings.delete to_delete
end
```

```ruby
def delete_complete_overlaps(new)
  pricings.each do |old|
    if new.start_date <= old.start_date && new.end_date >= old.end_date
      pricings.delete old
    end
  end
end
```

```ruby
def delete_complete_overlaps(new)
  to_delete = pricings.select { |old| new.contains? old }
  pricings.delete to_delete
end
```

ADJUST OVERLAPS

```
class PricingRange
  [...]

  def adjust_overlaps(new)
    adjust_overlaps_of_start(new)
    adjust_overlaps_of_end(new)
  end
end
```

OVERLAPPING START

```ruby
def adjust_overlaps_of_start(new)
  pricings.each do |old|
    if new.start_date <= old.start_date && new.end_date >= old.start_date
      old.start_date = new.end_date + 1.day
      old.save!
     end
  end
end
```

```ruby
def adjust_overlaps_of_start(new)
  to_adjust = pricings.select do |old|
    new.start_date <= old.start_date && new.end_date >= old.start_date
  end

  to_adjust.each do |pricing|
    pricing.start_date = new.start_date + 1.day
    pricing.save!
  end
end
```

```ruby
class Pricing
  [...]

  def overlaps_start_of?(other)
    self.start_date <= other.start_date && self.end_date >= other.start_date
  end
end
```

```ruby
def adjust_overlaps_of_start(new)
  to_adjust = pricings.select { |old| new.overlaps_start_of? old }

  to_adjust.each do |pricing|
    pricing.start_date = new.start_date + 1.day
    pricing.save!
  end
end
```

```ruby
def adjust_overlaps_of_start(new)
  overlapped = pricings.find { |old| new.overlaps_start_of? old }
  overlapped.start_date = new.start_date + 1.day
  overlapped.save!
end
```

```ruby
def adjust_overlaps_of_start(new)
  overlapped = pricings.find { |old| new.overlaps_start_of? old }
  return unless overlapped

  overlapped.start_date = new.start_date + 1.day
  overlapped.save!
end
```

```ruby
def adjust_overlaps_of_start(new)
  to_adjust = pricings.select { |old| new.overlaps_start_of? old }

  to_adjust.each do |pricing|
    pricing.start_date = new.start_date + 1.day
    pricing.save!
  end
end
```

OVERLAPPING END

```ruby
def adjust_overlaps_of_end(new)
  pricings.each do |old|
    if new.start_date <= old.end_date && new.end_date >= old.end_date
      old.end_date = new.start_date - 1.day
      old.save!
     end
  end
end
```

```ruby
def adjust_overlaps_of_end(new)
  to_adjust = pricings.select { |old| new.overlaps_end_of? old }

  to_adjust.each do |pricing|
    pricing.end_date = new.start_date - 1.day
    pricing.save!
  end
end
```

```
class PricingRange
  [...]

  def add(pricing)
    delete_complete_overlaps(pricing)
    adjust_overlaps(pricing)
    split_containing(pricing)
    [...]
  end
end
```

SPLIT CONTAINING

```ruby
def split_containing(new)
  pricings.each do |old|
    if old.start_date <= new.start_date && old.end_date >= new.end_date
      old_end = old.end_date
      old.end_date = new.start_date - 1.day
      old.save!

      other_side = old.dup
      other_side.start_date = new.end_date + 1.day
      other_side.end_date = old_end

      pricings << other_side
    end
  end
end
```

```ruby
def split_containing(new)
  to_split = pricings.select { |old| old.contains? new }

  to_split.each do |left_side|
    right_side_end = left_side.end_date
    left_side.end_date = new.start_date - 1.day
    left_side.save!

    pricings << build_right_side(left_side, new.end_date + 1.day, right_side_end)
  end
end
```

```ruby
def build_right_side(left_side, middle, start_date, end_date)
  right_side = left_side.dup
  right_side.start_date = start_date
  right_side.end_date = end_date
  right_side
end
```

```
class PricingRange
  [...]

  def add(pricing)
    delete_complete_overlaps(pricing)
    adjust_overlaps(pricing)
    split_containing(pricing)
    expand_to_meet(pricing)
    [...]
  end
end
```

EXPAND TO MEET

```
def expand_to_meet(new)
  expand_to_meet_before(new)
  expand_to_meet_after(new)
end
```

```ruby
def expand_to_meet_before(new)
  pricings.each do |old|
    if old == pricings.order('start_date ASC').first && old.start_date > new.end_date
      old.start_date = new.end_date + 1.day
      old.save!
    end
  end
end
```

```ruby
def expand_to_meet_before(new)
  first_pricing_after_new = pricings.select do |old|
    old == pricings.order('start_date ASC').first && old.start_date > new.end_date
  end

  first_pricing_after_new.each do |old|
    old.start_date = new.end_date + 1.day
    old.save!
  end
end
```

```ruby
def first_pricing
  pricings.order('start_date ASC').first
end
```

```ruby
class Pricing
  [...]

  def is_after?(other)
    self.start_date > other.end_date
  end
end
```

```ruby
def expand_to_meet_before(new)
  first_pricing_after_new = pricings.select do |old|
    old == first_pricing && old.is_after?(new)
  end

  first_pricing_after_new.each do |old|
    old.start_date = new.end_date + 1.day
    old.save!
  end
end
```

```
def expand_to_meet_after(new)
  [...]
end
```

I think you probably get it...

```
class PricingRange
  [...]

  def add(pricing)
    delete_complete_overlaps(pricing)
    adjust_overlaps(pricing)
    split_containing(pricing)
    expand_to_meet(pricing)
    coalesce_pricings
  end
end
```

COALESCE PRICINGS

```ruby
def coalesce_pricings
  pricings.each do |pricing|
    previous = pricings.where(end_date: pricing.start_date - 1.day).first

    if previous && pricing.price == previous.price
      pricing.start_date = previous.start_date
      previous.destroy
      pricing.save!
    end
  end
end
```

```ruby
def coalesce_pricings
  delete_before = pricings.select do |pricing|
    previous = pricings.where(end_date: pricing.start_date - 1.day).first
    previous && pricing.price == previous.price
  end

  delete_before.each do |pricing|
    previous = pricings.where(end_date: pricing.start_date - 1.day).first
    pricing.start_date = previous.start_date
    previous.destroy
    pricing.save!
  end
end
```

```ruby
def previous_pricing_for(pricing)
  pricings.where(end_date: pricing.start_date - 1.day).first
end
```

```ruby
def coalesce_pricings
  delete_before = pricings.select do |pricing|
    previous = previous_pricing_for(pricing)
    previous && pricing.price == previous.price
  end

  delete_before.each do |pricing|
    previous = previous_pricing_for(pricing)
    pricing.start_date = previous.start_date
    previous.destroy
    pricing.save!
  end
end
```

```
class PricingRange
  [...]

  def add(pricing)
    delete_complete_overlaps(pricing)
    adjust_overlaps(pricing)
    split_containing(pricing)
    expand_to_meet(pricing)
    coalesce_pricings
  end
end
```

```
class PricingRange
  [...]

  def add(pricing)
    delete_complete_overlaps(pricing)
    adjust_overlaps(pricing)
    split_containing(pricing)
    expand_to_meet(pricing)

    pricings << pricing

    coalesce_pricings

    pricings
  end
end
```

DONE!

WRAP-UP

WRITE TESTS

Ugly, conditional code, is made more beautiful by tests.

DON'T EVER USE UNLESS/ELSE

```
unless thing
  do_something
else
  do_something_else
end
```

IF YOU READ THAT AND IT MADE SENSE, YOU ARE A ROBOT.

WE'RE ALL STILL GOING TO USE CONDITIONALS.

IT'S OK.

WHAT'S IMPORTANT IS THAT YOU FEEL BAD ABOUT IT.

That's all, folks!

Oh, and you should probably follow me on Twitter

@eroberts

THINGS YOU SHOULD CHECK OUT FOR MORE INFORMATION

- Confident Ruby by Avdi Grimm
- All the Little Things by Sandi Metz
- Practical Object Oriented Design in Ruby by Sandi Metz
- Therapeutic Refactoring by Katrina Owen