Object Oriented Rails

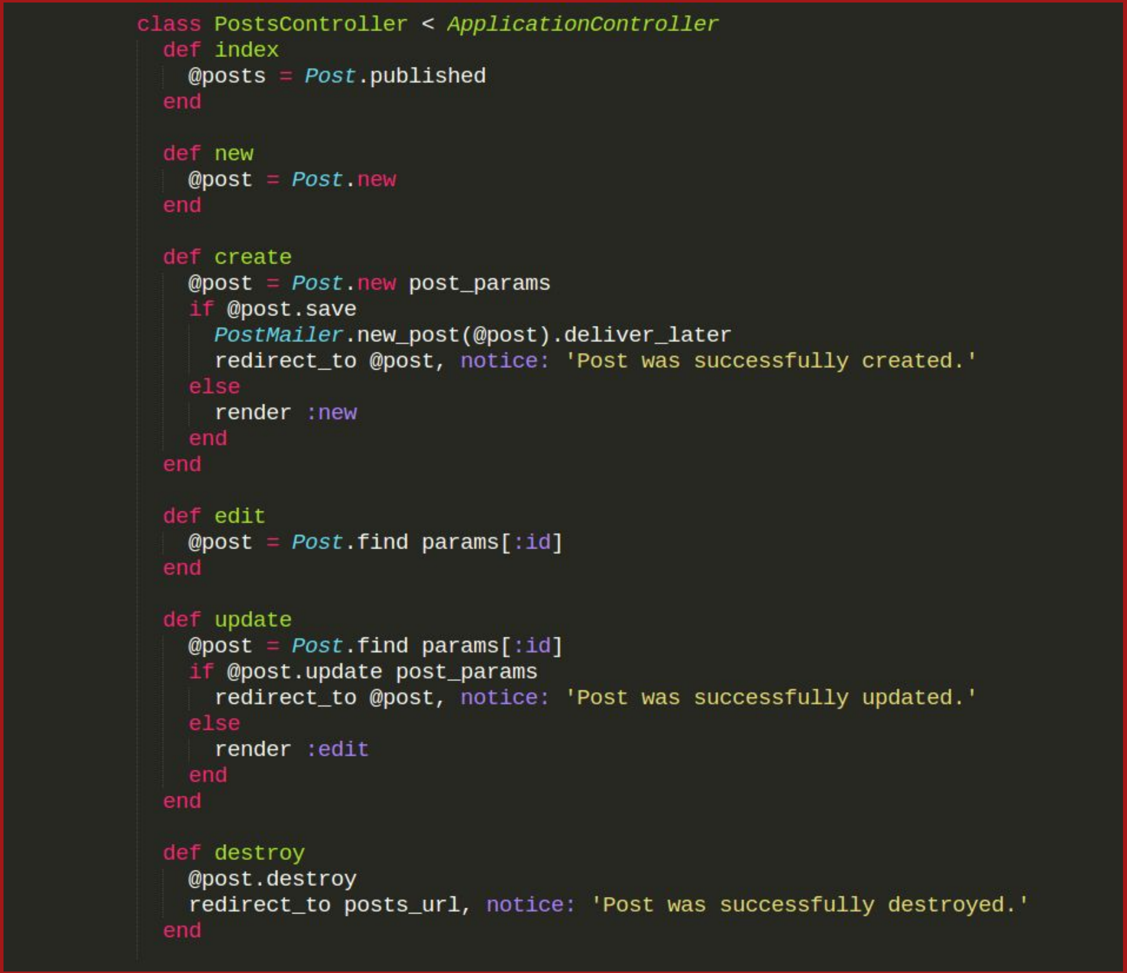
●●●

Karol Topolski

Agenda

1. Class based programming versus object oriented programming
2. RoR developer’s road to object oriented zen
3. Object Oriented Rails - different kind of objects

Class based programming

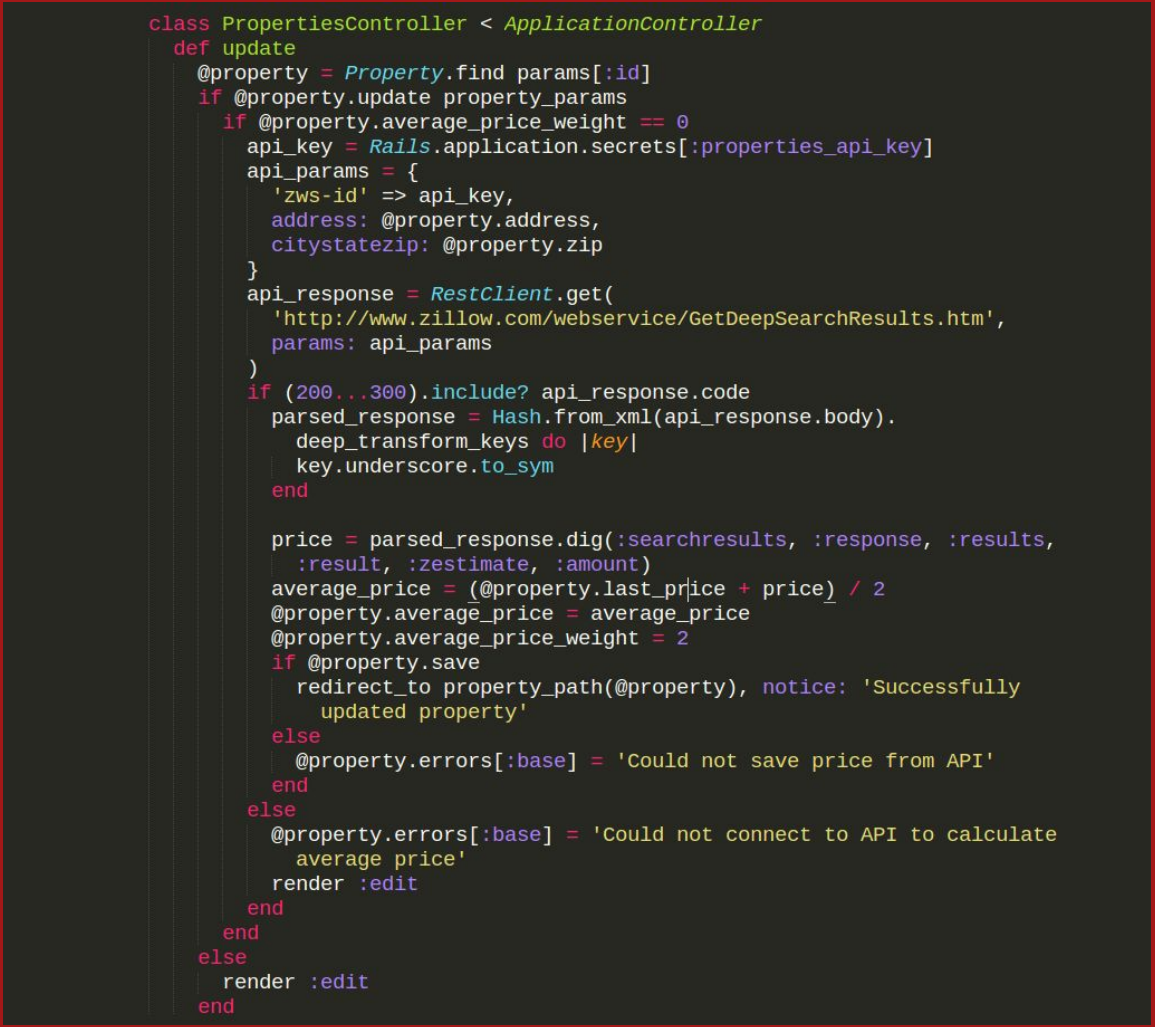


```ruby
class PostsController < ApplicationController
  def index
    @posts = Post.published
  end

  def new
    @post = Post.new
  end

  def create
    @post = Post.new post_params
    if @post.save
      PostMailer.new_post(@post).deliver_later
      redirect_to @post, notice: 'Post was successfully created.'
    else
      render :new
    end
  end

  def edit
    @post = Post.find params[:id]
  end

  def update
    @post = Post.find params[:id]
    if @post.update post_params
      redirect_to @post, notice: 'Post was successfully updated.'
    else
      render :edit
    end
  end

  def destroy
    @post.destroy
    redirect_to posts_url, notice: 'Post was successfully destroyed.'
  end
```

Typical Rails controller

Rails may be Object Oriented itself,
but it forces us to write mainly classes and
structural code.

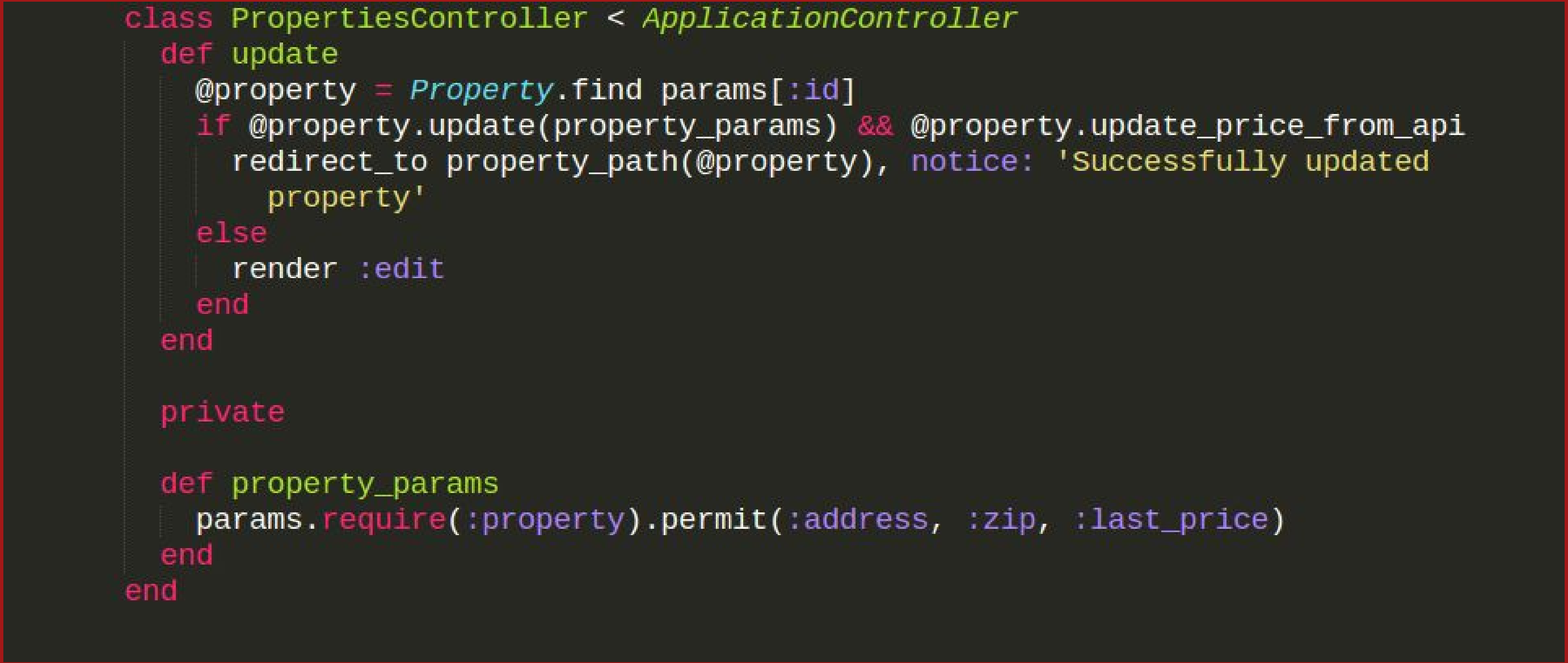
RoR developer’s way to Object Oriented zen



```ruby
class PropertiesController < ApplicationController
  def update
    @property = Property.find params[:id]
    if @property.update property_params
      if @property.average_price_weight == 0
        api_key = Rails.application.secrets[:properties_api_key]
        api_params = {
          'zws-id' => api_key,
          address: @property.address,
          citystatezip: @property.zip
        }
        api_response = RestClient.get(
          'http://www.zillow.com/webservice/GetDeepSearchResults.htm',
          params: api_params
        )
        if (200...300).include? api_response.code
          parsed_response = Hash.from_xml(api_response.body).
            deep_transform_keys do |key|
            key.underscore.to_sym
          end

          price = parsed_response.dig(:searchresults, :response, :results,
            :result, :zestimate, :amount)
          average_price = (@property.last_price + price) / 2
          @property.average_price = average_price
          @property.average_price_weight = 2
          if @property.save
            redirect_to property_path(@property), notice: 'Successfully
              updated property'
          else
            @property.errors[:base] = 'Could not save price from API'
          end
        else
          @property.errors[:base] = 'Could not connect to API to calculate
            average price'
          render :edit
        end
      end
    else
      render :edit
    end
```

We have to start somewhere…
This is where most of us did

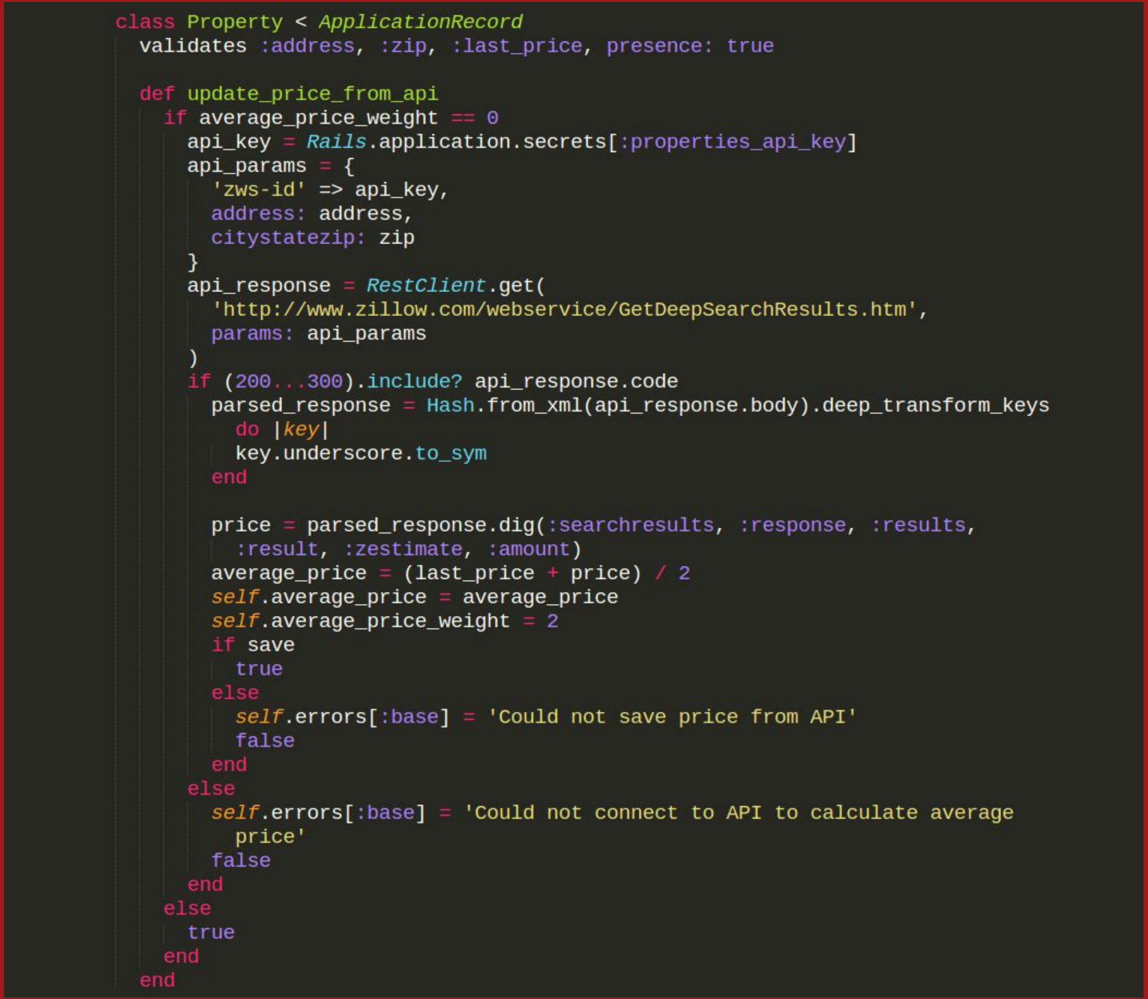
Fat models, thin controllers!



```ruby
class PropertiesController < ApplicationController
  def update
    @property = Property.find params[:id]
    if @property.update(property_params) && @property.update_price_from_api
      redirect_to property_path(@property), notice: 'Successfully updated
        property'
    else
      render :edit
    end
  end

  private

  def property_params
    params.require(:property).permit(:address, :zip, :last_price)
  end
end
```

Fat models, thin controllers!

Fat models, thin controllers!



```ruby
class Property < ApplicationRecord
  validates :address, :zip, :last_price, presence: true

  def update_price_from_api
    if average_price_weight == 0
      api_key = Rails.application.secrets[:properties_api_key]
      api_params = {
        'zws-id' => api_key,
        address: address,
        citystatezip: zip
      }
      api_response = RestClient.get(
        'http://www.zillow.com/webservice/GetDeepSearchResults.htm',
        params: api_params
      )
      if (200...300).include? api_response.code
        parsed_response = Hash.from_xml(api_response.body).deep_transform_keys
          do |key|
          key.underscore.to_sym
        end

        price = parsed_response.dig(:searchresults, :response, :results,
          :result, :zestimate, :amount)
        average_price = (last_price + price) / 2
        self.average_price = average_price
        self.average_price_weight = 2
        if save
          true
        else
          self.errors[:base] = 'Could not save price from API'
          false
        end
      else
        self.errors[:base] = 'Could not connect to API to calculate average
          price'
        false
      end
    else
      true
    end
  end
```

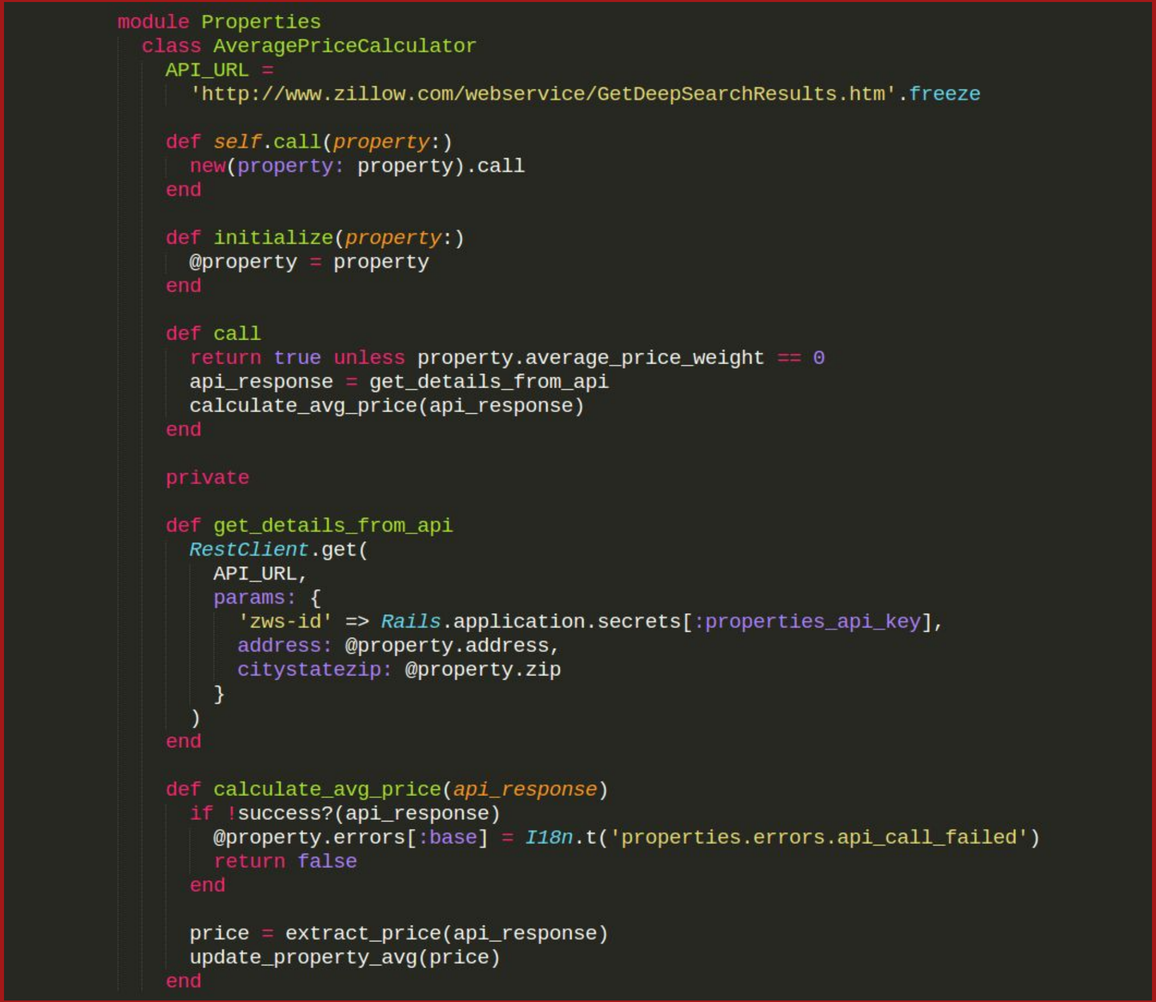
ActiveRecord, what are you?

"An object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data."

Martin Fowler

Design Patterns of Enterprise Application Architecture

Putting business logic into models is RailsWay™, but not really S.O.L.I.D...



```ruby
module Properties
  class AveragePriceCalculator
    API_URL =
      'http://www.zillow.com/webservice/GetDeepSearchResults.htm'.freeze

    def self.call(property:)
      new(property: property).call
    end

    def initialize(property:)
      @property = property
    end

    def call
      return true unless property.average_price_weight == 0
      api_response = get_details_from_api
      calculate_avg_price(api_response)
    end

    private

    def get_details_from_api
      RestClient.get(
        API_URL,
        params: {
          'zws-id' => Rails.application.secrets[:properties_api_key],
          address: @property.address,
          citystatezip: @property.zip
        }
      )
    end

    def calculate_avg_price(api_response)
      if !success?(api_response)
        @property.errors[:base] = I18n.t('properties.errors.api_call_failed')
        return false
      end

      price = extract_price(api_response)
      update_property_avg(price)
    end
```

Service object

Welcome to the world of Object Oriented Rails

Object Oriented Rails: Service Objects

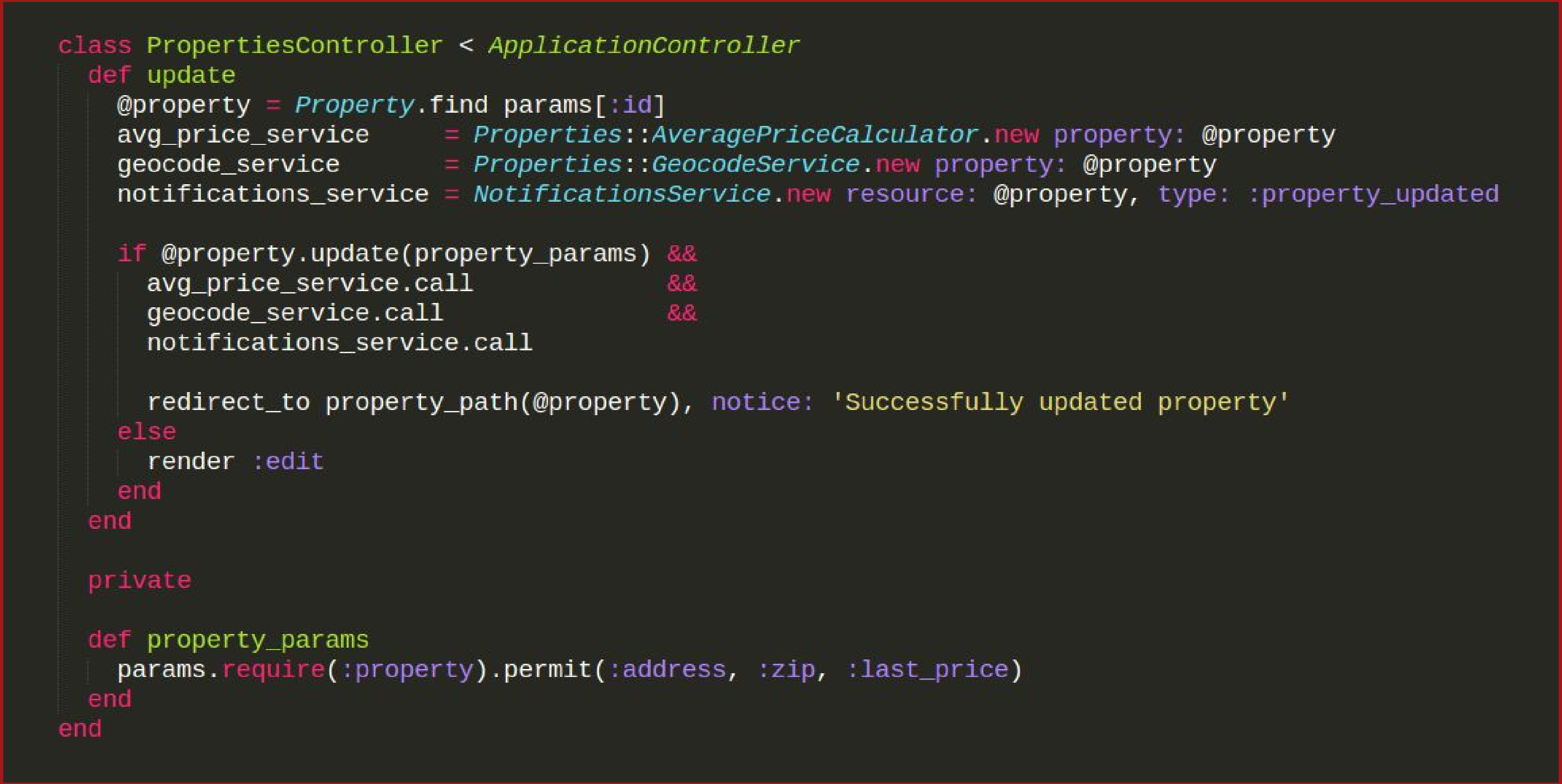
Service object - poor man’s definitions

Domain Driven Design definition:

“Operation offered as an interface that stands alone in the model, without encapsulating state(...)”

Rails community definition:

EVERYTHING THAT DOESN'T FIT IN MVC!
memegenerator.net

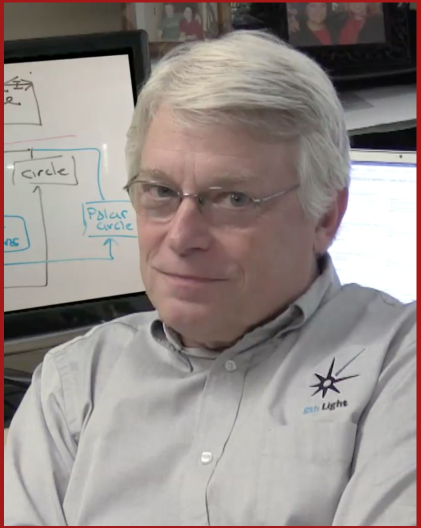


```ruby
class PropertiesController < ApplicationController
  def update
    @property = Property.find params[:id]
    avg_price_service     = Properties::AveragePriceCalculator.new property: @property
    geocode_service       = Properties::GeocodeService.new property: @property
    notifications_service = NotificationsService.new resource: @property, type: :property_updated

    if @property.update(property_params) &&
      avg_price_service.call             &&
      geocode_service.call               &&
      notifications_service.call

      redirect_to property_path(@property), notice: 'Successfully updated property'
    else
      render :edit
    end
  end

  private

  def property_params
    params.require(:property).permit(:address, :zip, :last_price)
  end
end
```

When services grow in numbers...

Object Oriented Rails: Interactors

Interactors

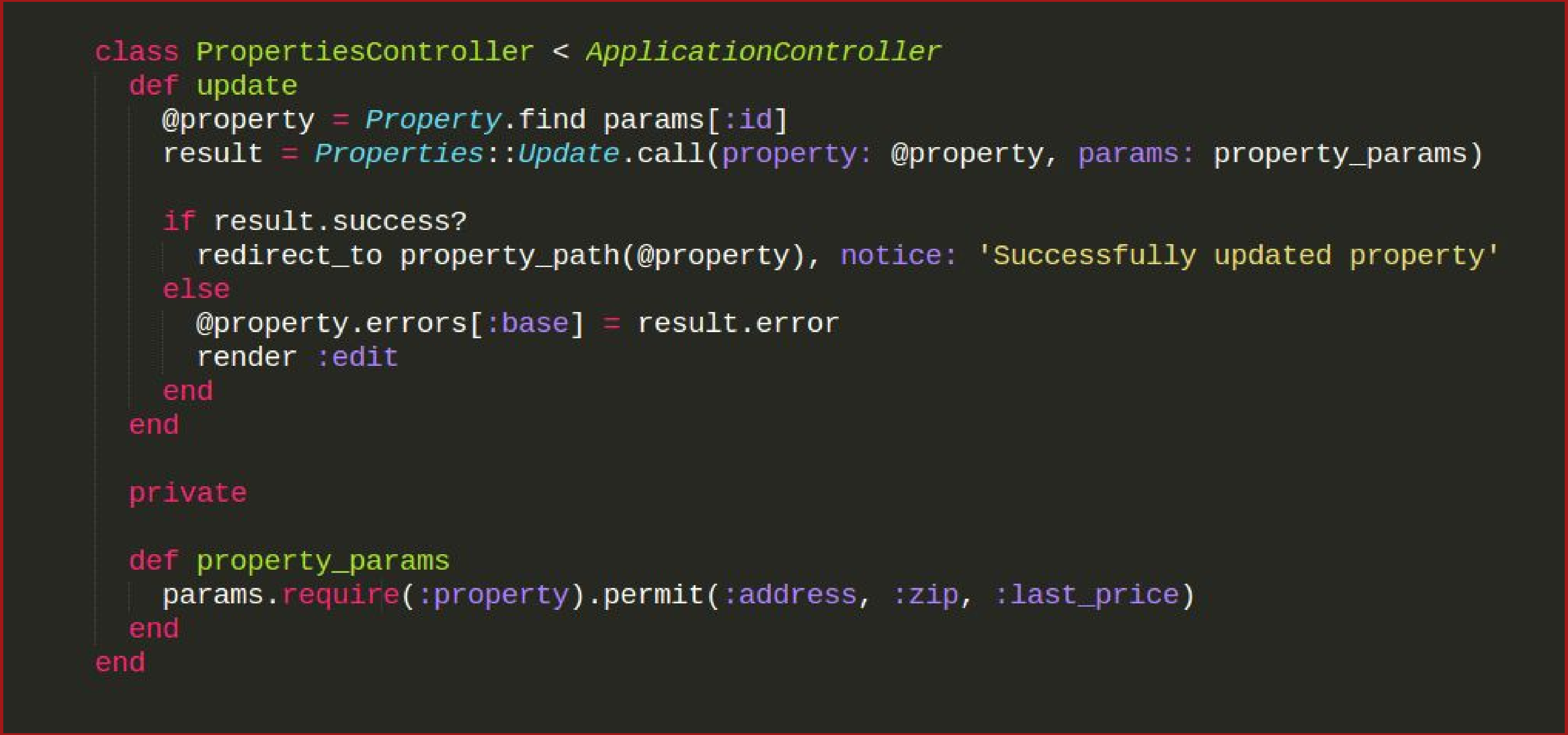


*“Interactor objects are objects that implement a **use case.**”*

*“Architecture is about **intent.**”*

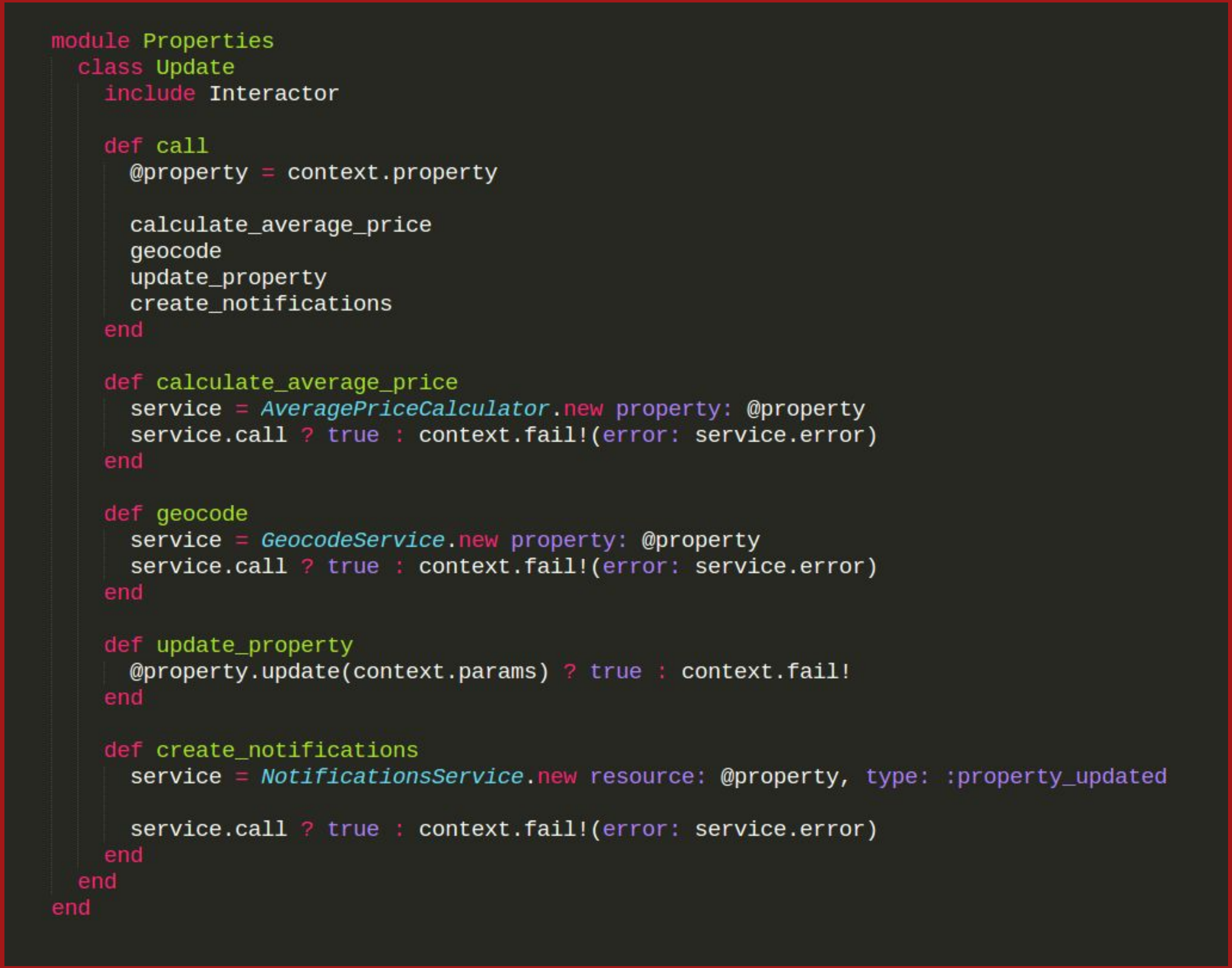
Robert C. Martin a.k.a. Uncle Bob,

Architecture - The Lost Years(@ Ruby Midwest 2011)



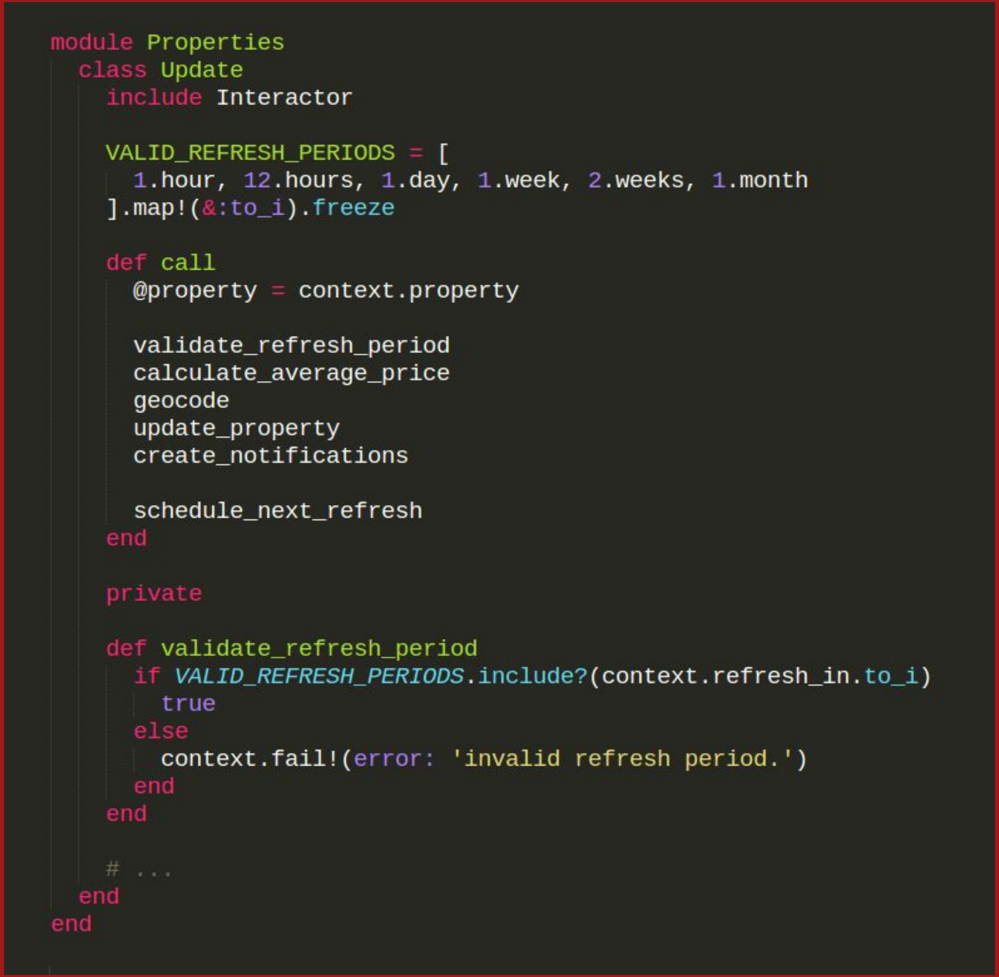
```ruby
class PropertiesController < ApplicationController
  def update
    @property = Property.find params[:id]
    result = Properties::Update.call(property: @property, params: property_params)

    if result.success?
      redirect_to property_path(@property), notice: 'Successfully updated property'
    else
      @property.errors[:base] = result.error
      render :edit
    end
  end

  private

  def property_params
    params.require(:property).permit(:address, :zip, :last_price)
  end
end
```

Same endpoint with interactor



```ruby
module Properties
  class Update
    include Interactor

    def call
      @property = context.property

      calculate_average_price
      geocode
      update_property
      create_notifications
    end

    def calculate_average_price
      service = AveragePriceCalculator.new property: @property
      service.call ? true : context.fail!(error: service.error)
    end

    def geocode
      service = GeocodeService.new property: @property
      service.call ? true : context.fail!(error: service.error)
    end

    def update_property
      @property.update(context.params) ? true : context.fail!
    end

    def create_notifications
      service = NotificationsService.new resource: @property, type: :property_updated

      service.call ? true : context.fail!(error: service.error)
    end
  end
end
```

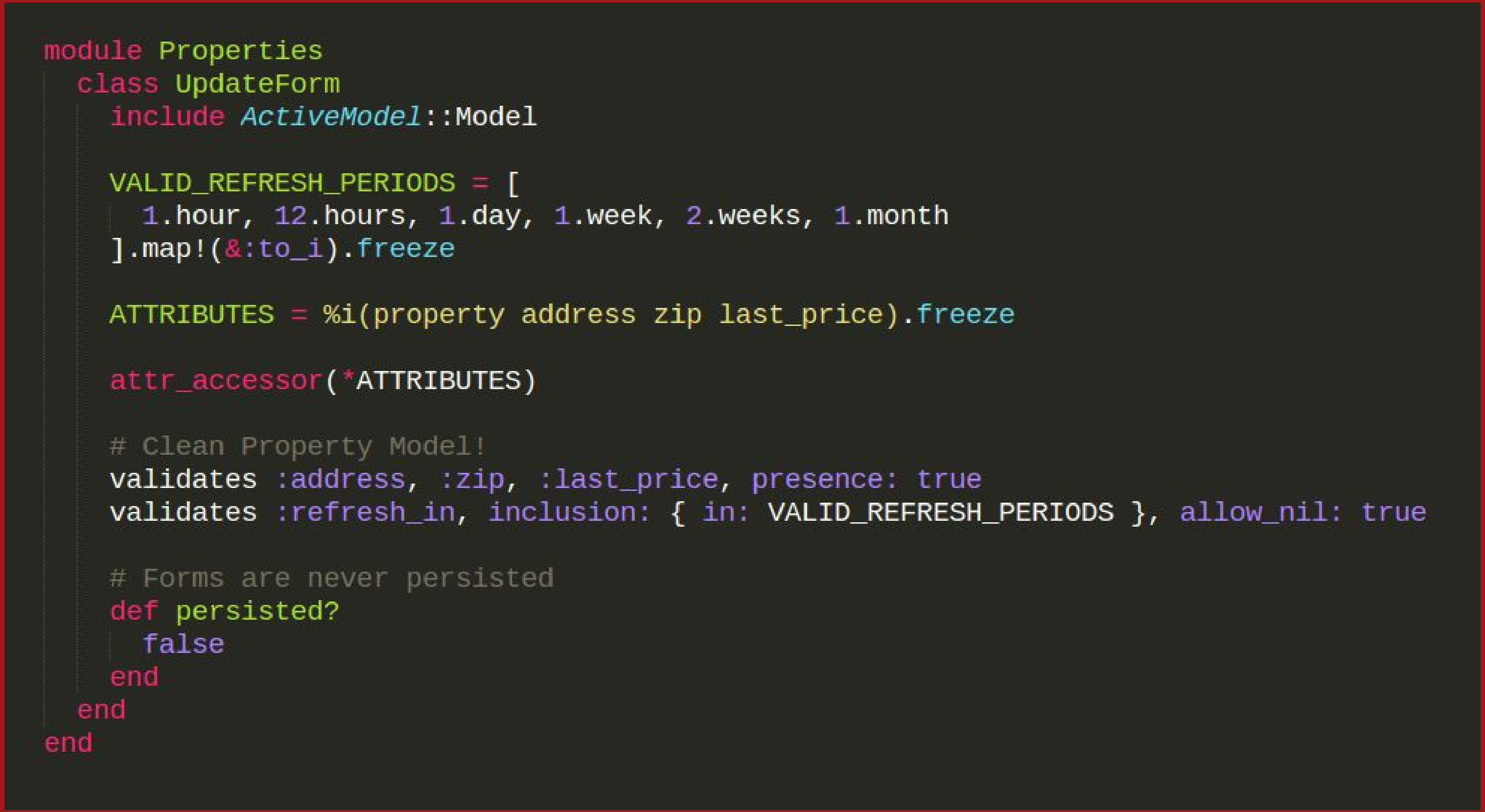
Interactor example



```
module Properties
  class Update
    include Interactor

    VALID_REFRESH_PERIODS = [
      1.hour, 12.hours, 1.day, 1.week, 2.weeks, 1.month
    ].map!(&:to_i).freeze

    def call
      @property = context.property

      validate_refresh_period
      calculate_average_price
      geocode
      update_property
      create_notifications

      schedule_next_refresh
    end

    private

    def validate_refresh_period
      if VALID_REFRESH_PERIODS.include?(context.refresh_in.to_i)
        true
      else
        context.fail!(error: 'invalid refresh period.')
      end
    end

    # ...
  end
end
```

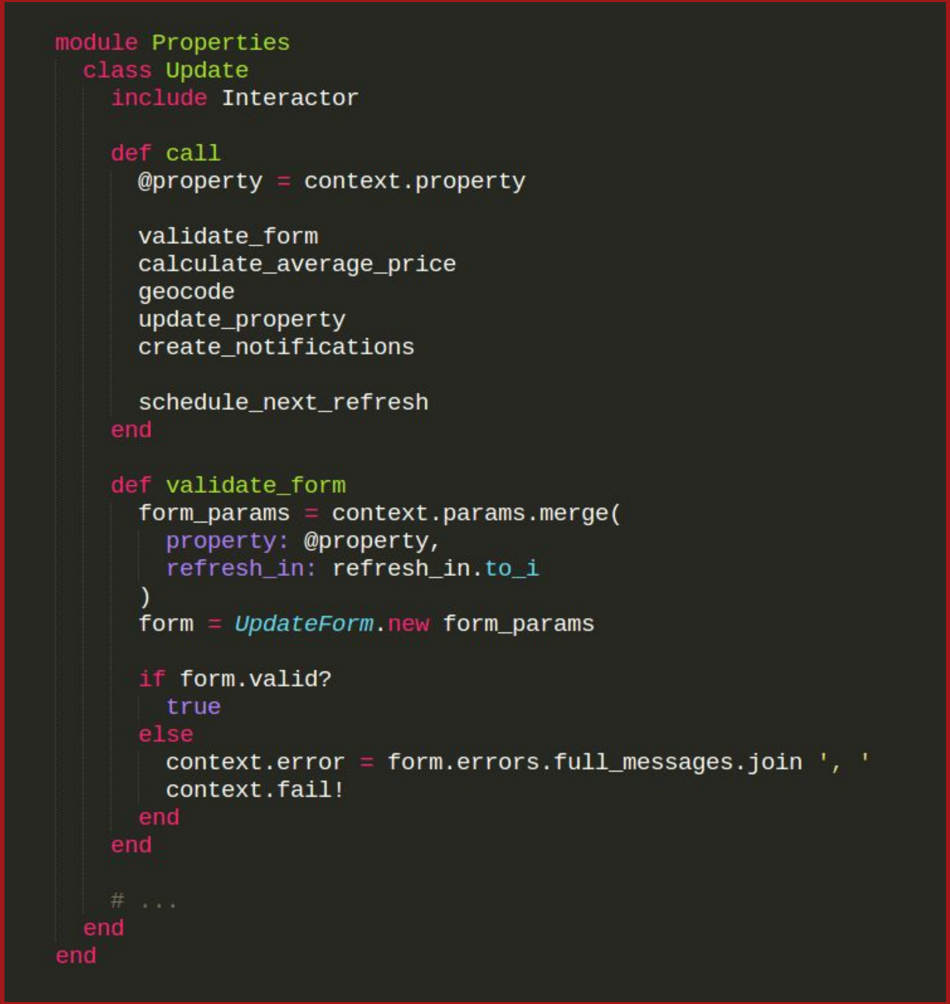
The params problem in interactor

Object Oriented Rails: Form Objects



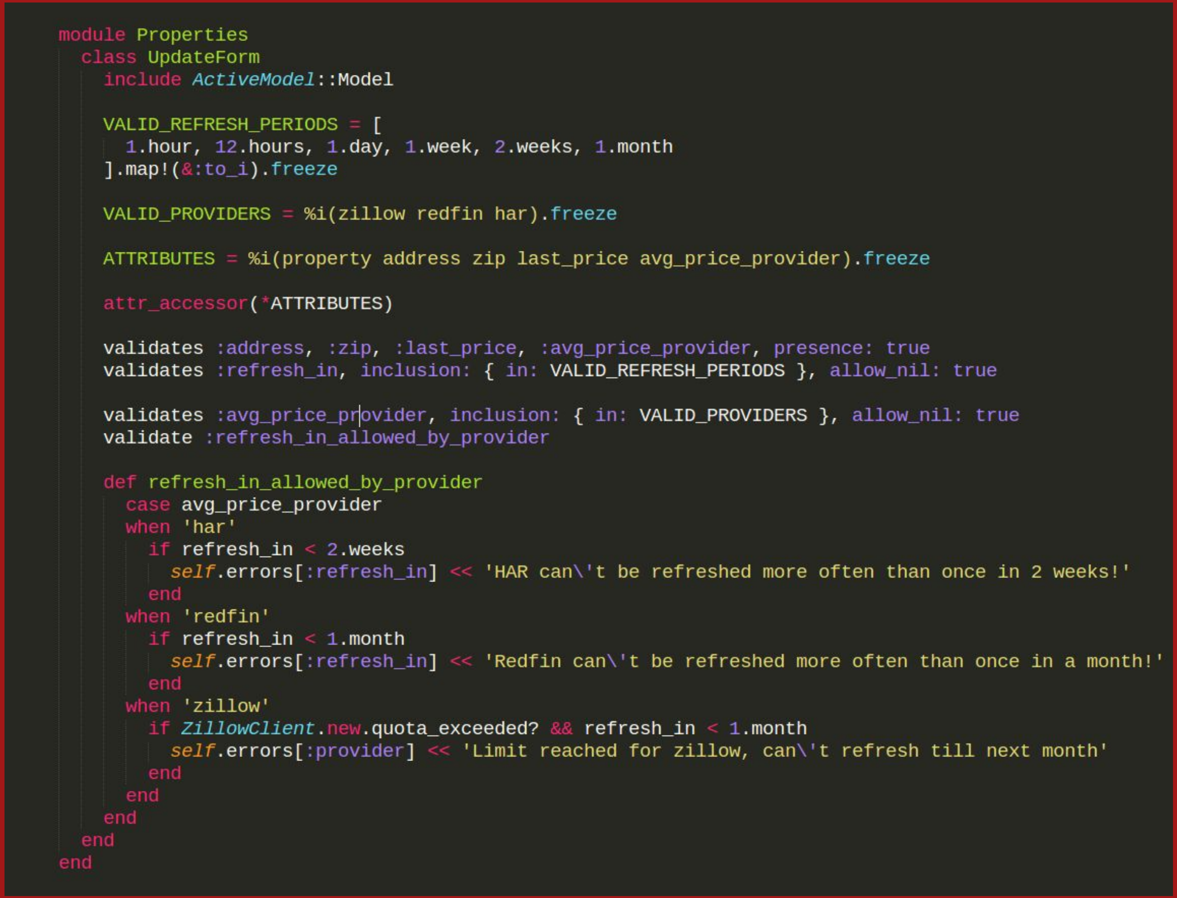
```ruby
module Properties
  class UpdateForm
    include ActiveModel::Model

    VALID_REFRESH_PERIODS = [
      1.hour, 12.hours, 1.day, 1.week, 2.weeks, 1.month
    ].map!(&:to_i).freeze

    ATTRIBUTES = %i(property address zip last_price).freeze

    attr_accessor(*ATTRIBUTES)

    # Clean Property Model!
    validates :address, :zip, :last_price, presence: true
    validates :refresh_in, inclusion: { in: VALID_REFRESH_PERIODS }, allow_nil: true

    # Forms are never persisted
    def persisted?
      false
    end
  end
end
```

Form Object using ActiveModel



```
module Properties
  class Update
    include Interactor

    def call
      @property = context.property

      validate_form
      calculate_average_price
      geocode
      update_property
      create_notifications

      schedule_next_refresh
    end

    def validate_form
      form_params = context.params.merge(
        property: @property,
        refresh_in: refresh_in.to_i
      )
      form = UpdateForm.new form_params

      if form.valid?
        true
      else
        context.error = form.errors.full_messages.join ', '
        context.fail!
      end
    end

    # ...
  end
end
```

Interactor with form object

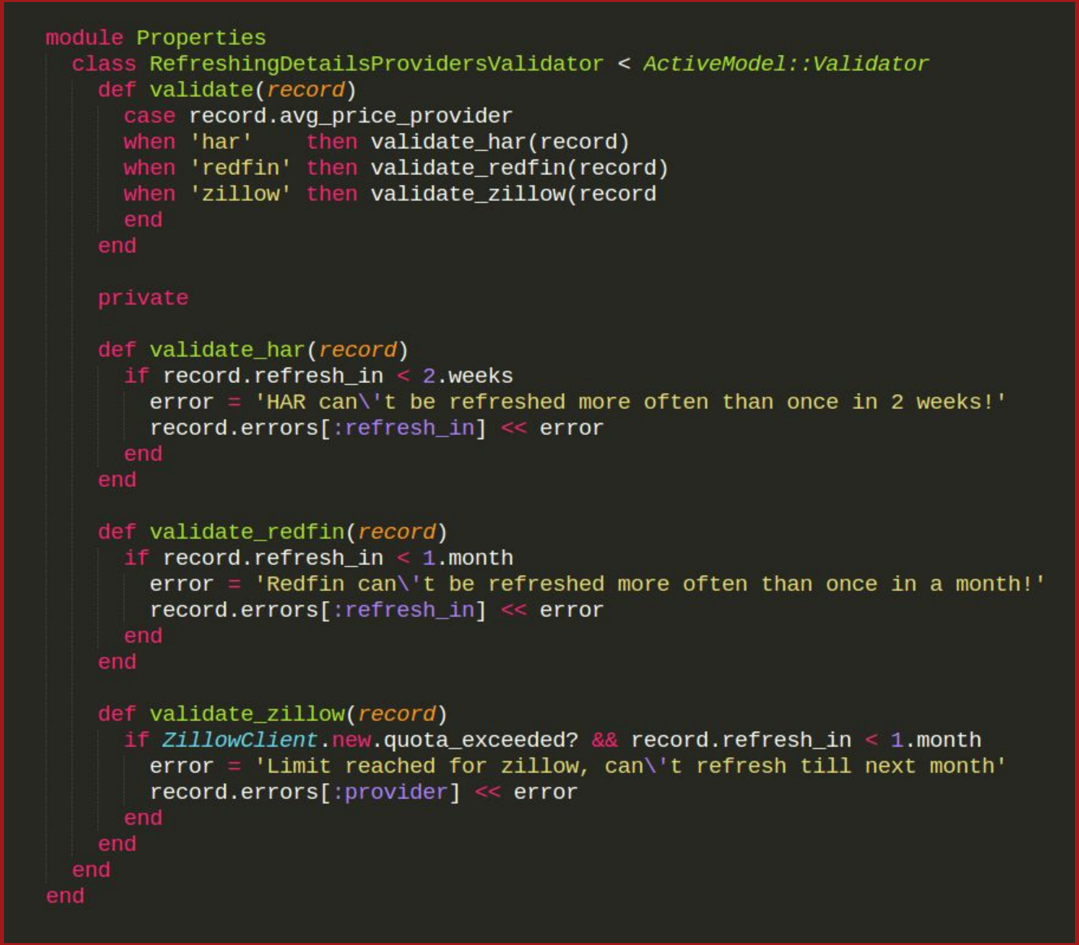


```ruby
module Properties
  class UpdateForm
    include ActiveModel::Model

    VALID_REFRESH_PERIODS = [
      1.hour, 12.hours, 1.day, 1.week, 2.weeks, 1.month
    ].map!(&:to_i).freeze

    VALID_PROVIDERS = %i(zillow redfin har).freeze

    ATTRIBUTES = %i(property address zip last_price avg_price_provider).freeze

    attr_accessor(*ATTRIBUTES)

    validates :address, :zip, :last_price, :avg_price_provider, presence: true
    validates :refresh_in, inclusion: { in: VALID_REFRESH_PERIODS }, allow_nil: true

    validates :avg_price_provider, inclusion: { in: VALID_PROVIDERS }, allow_nil: true
    validate :refresh_in_allowed_by_provider

    def refresh_in_allowed_by_provider
      case avg_price_provider
      when 'har'
        if refresh_in < 2.weeks
          self.errors[:refresh_in] << 'HAR can\'t be refreshed more often than once in 2 weeks!'
        end
      when 'redfin'
        if refresh_in < 1.month
          self.errors[:refresh_in] << 'Redfin can\'t be refreshed more often than once in a month!'
        end
      when 'zillow'
        if ZillowClient.new.quota_exceeded? && refresh_in < 1.month
          self.errors[:provider] << 'Limit reached for zillow, can\'t refresh till next month'
        end
      end
    end
  end
end
```

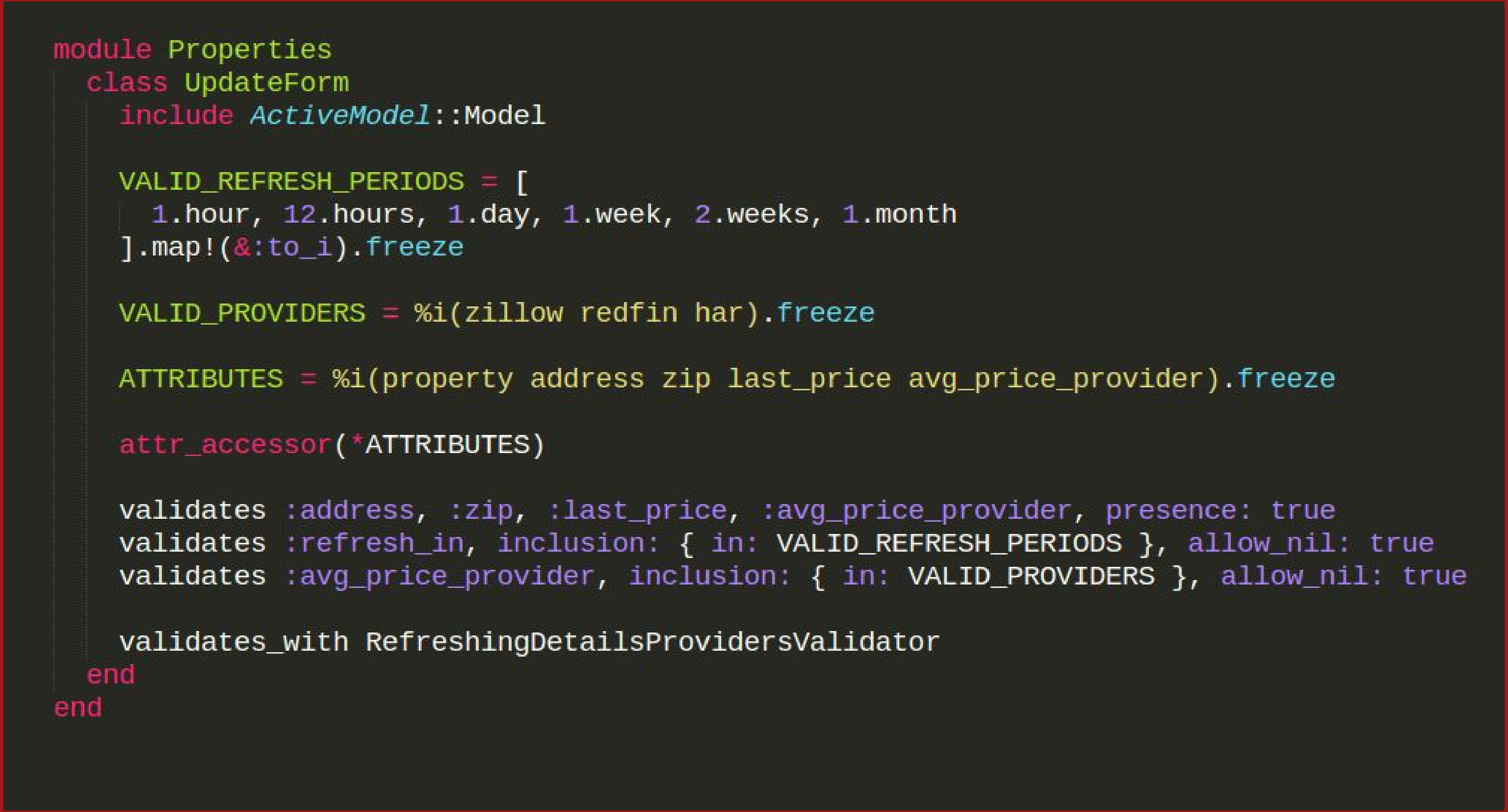
When form object carries
a little bit too much validation…

Object Oriented Rails: Validators

ActiveModel::Validator example



```ruby
module Properties
  class RefreshingDetailsProvidersValidator < ActiveModel::Validator
    def validate(record)
      case record.avg_price_provider
      when 'har'    then validate_har(record)
      when 'redfin' then validate_redfin(record)
      when 'zillow' then validate_zillow(record
      end
    end

    private

    def validate_har(record)
      if record.refresh_in < 2.weeks
        error = 'HAR can\'t be refreshed more often than once in 2 weeks!'
        record.errors[:refresh_in] << error
      end
    end

    def validate_redfin(record)
      if record.refresh_in < 1.month
        error = 'Redfin can\'t be refreshed more often than once in a month!'
        record.errors[:refresh_in] << error
      end
    end

    def validate_zillow(record)
      if ZillowClient.new.quota_exceeded? && record.refresh_in < 1.month
        error = 'Limit reached for zillow, can\'t refresh till next month'
        record.errors[:provider] << error
      end
    end
  end
end
```



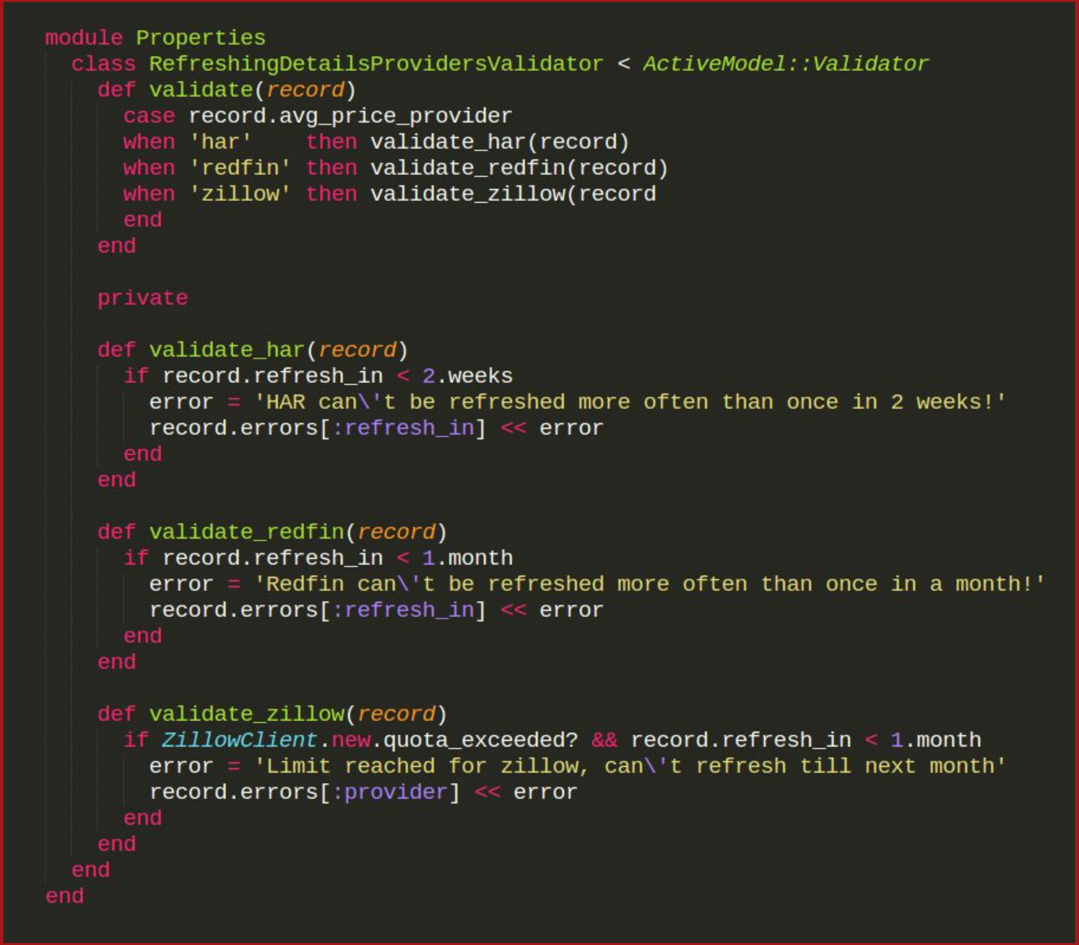
```ruby
module Properties
  class UpdateForm
    include ActiveModel::Model

    VALID_REFRESH_PERIODS = [
      1.hour, 12.hours, 1.day, 1.week, 2.weeks, 1.month
    ].map!(&:to_i).freeze

    VALID_PROVIDERS = %i(zillow redfin har).freeze

    ATTRIBUTES = %i(property address zip last_price avg_price_provider).freeze

    attr_accessor(*ATTRIBUTES)

    validates :address, :zip, :last_price, :avg_price_provider, presence: true
    validates :refresh_in, inclusion: { in: VALID_REFRESH_PERIODS }, allow_nil: true
    validates :avg_price_provider, inclusion: { in: VALID_PROVIDERS }, allow_nil: true

    validates_with RefreshingDetailsProvidersValidator
  end
end
```

Form object using Validator

Q & A time!

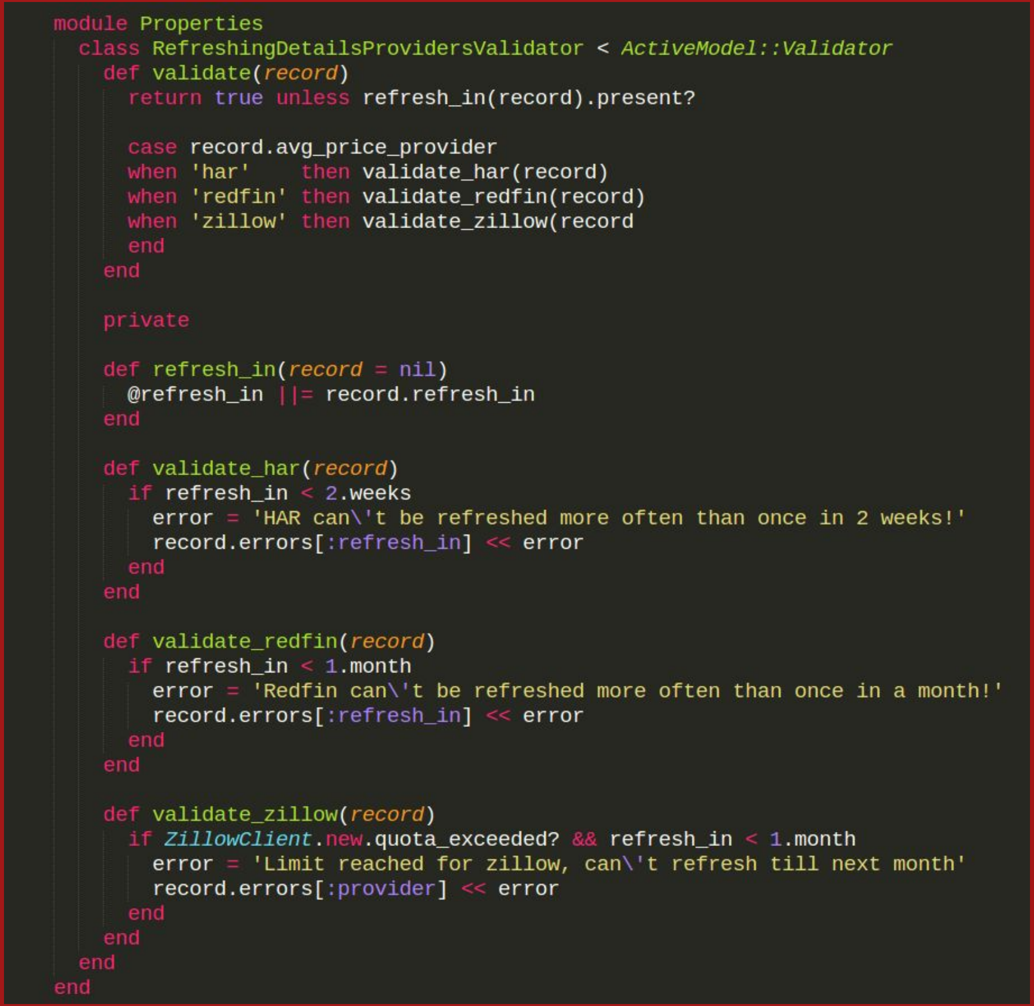
✉ ktopolski.it@gmail.com

Bonus round: ActiveModel::Validator trap



```ruby
module Properties
  class RefreshingDetailsProvidersValidator < ActiveModel::Validator
    def validate(record)
      case record.avg_price_provider
      when 'har'     then validate_har(record)
      when 'redfin'  then validate_redfin(record)
      when 'zillow'  then validate_zillow(record
      end
    end

    private

    def validate_har(record)
      if record.refresh_in < 2.weeks
        error = 'HAR can\'t be refreshed more often than once in 2 weeks!'
        record.errors[:refresh_in] << error
      end
    end

    def validate_redfin(record)
      if record.refresh_in < 1.month
        error = 'Redfin can\'t be refreshed more often than once in a month!'
        record.errors[:refresh_in] << error
      end
    end

    def validate_zillow(record)
      if ZillowClient.new.quota_exceeded? && record.refresh_in < 1.month
        error = 'Limit reached for zillow, can\'t refresh till next month'
        record.errors[:provider] << error
      end
    end
  end
end
```

Why can’t we just set up @refresh_in instance var here?



```ruby
module Properties
  class RefreshingDetailsProvidersValidator < ActiveModel::Validator
    def validate(record)
      return true unless refresh_in(record).present?

      case record.avg_price_provider
      when 'har'     then validate_har(record)
      when 'redfin'  then validate_redfin(record)
      when 'zillow'  then validate_zillow(record
      end
    end

    private

    def refresh_in(record = nil)
      @refresh_in ||= record.refresh_in
    end

    def validate_har(record)
      if refresh_in < 2.weeks
        error = 'HAR can\'t be refreshed more often than once in 2 weeks!'
        record.errors[:refresh_in] << error
      end
    end

    def validate_redfin(record)
      if refresh_in < 1.month
        error = 'Redfin can\'t be refreshed more often than once in a month!'
        record.errors[:refresh_in] << error
      end
    end

    def validate_zillow(record)
      if ZillowClient.new.quota_exceeded? && refresh_in < 1.month
        error = 'Limit reached for zillow, can\'t refresh till next month'
        record.errors[:provider] << error
      end
    end
  end
end
```

Looks more DRY right?

IT'S A TRAP

Rails initializes validators used with **validates_with** only **ONCE**!

At the first autoload in your application.

@refresh_in will always stay as the value passed in **first request after application boots**

Lessons to learn:

- DRY isn’t always better
- Do not trust anything that you don’t explicit initialize
- Write integration tests ;)

Thank you once again!

✉ ktopolski.it@gmail.com