Value Objects in Ruby (and Rails)

Vlado Cingel

I love my Wife, Kids and Ruby

Value Objects?

In computer science,
a value object is a small object that represents a simple entity
whose equality is not based on identity

Wikipedia

A small simple object, like money or a date range, whose equality isn't based on identity.

Martin Fowler
https://martinfowler.com/eaaCatalog/valueObject.html

Two value objects are equal when they have the same value, not necessarily being the same object.

Wikipedia

Identity / Value

```ruby
date1 = Date.new(2022, 9, 13)
date2 = Date.new(2022, 9, 13)

puts date1.object_id # => 60
puts date2.object_id # => 80
puts date1.equal?(date2) # => false
puts date1.eql?(date2) # => true
puts date1 == date2 # => true
```

Value Objects are simple objects whose equality is dependent on their value rather than an identity.

Value Objects in Ruby

- Date
- Range
- Pathname
- URI
- String
- …

- Symbol
- Integer
- TrueClass
- FalseClass
- NilClass
- ?

Ruby Primitives

```ruby
puts :ruby.object_id # => 707228
puts :ruby.object_id # => 707228
puts :ruby.equal?(:ruby) # => true

puts nil.object_id # => 8
puts nil.object_id # => 8
```

Domain specific Value Objects

```ruby
# create_table :products do |t|
#   t.integer :price_in_cents
#   ...
# end

class Product < ApplicationRecord
  VAT_PERCENTAGE = 25

  def price_in_eur
    price_in_cents / 100.0
  end

  def price_with_vat
    price_in_cents + (price_in_cents * VAT_PERCENTAGE / 100.0).round(2)
  end

 def price_with_vat_in_eur
   price_with_vat / 100
 end
end
```

```ruby
module ProductsHelper
  HRK_EXCHANGE_RATE = 7.53450

  def display_product_price_in_hrk(product)
    hrk = (product.price_in_eur * HRK_EXCHANGE_RATE).round(2)
    number_to_currency(price_in_hrk, currency: "HRK")
  end

  def display_product_price_with_vat_in_hrk(product)
    hrk = (product.price_with_vat_in_eur * HRK_EXCHANGE_RATE).round(2)
    number_to_currency(hrk, currency: "HRK")
  end

  def display_total_price_in_eur(products)
    number_to_currency(products.sum { |product| product.price_in_eur })
  end

  def display_total_price_with_vat_eur(products)
    number_to_currency(products.sum { |product| product.price_with_vat_in_eur })
  end
end
```

```ruby
class Price
  attr_reader :cents

  def initialize(cents)
    @cents = cents
  end

  def eur
    cents / 100.0
  end
end
```

```ruby
class Product < ApplicationRecord
  def price
    Price.new(price_in_cents)
  end

  def price=(p)
    self.price_in_cents = p.cents
  end
end

hundred_euros = Price.new(10_000)
p = Product.new(price: hundred_euros)

puts p.price_in_cents # => 10000
puts p.price
# => #<Price:0x00007fcdcf85a940
@cents=10000>

puts p.price.cents # => 10000
puts p.price.eur # => 100.00
```

```ruby
class Price
  HRK_EXCHANGE_RATE = 7.53450

  attr_reader :cents

  def initialize(cents)
    @cents = cents
  end

  def eur
   cents / 100.0
  end

  def hrk
    (eur * HRK_EXCHANGE_RATE).round(2)
  end
end

price = Price.new(10_000)
puts price.eur # => 100.00
puts price.hrk # => 753.45
```

```ruby
class Price
  HRK_EXCHANGE_RATE = 7.53450
  VAT_PERCENTAGE = 25

  attr_reader :cents

  def initialize(cents)
    @cents = cents
  end

  def eur
   cents / 100.0
  end

  def hrk
    (eur * HRK_EXCHANGE_RATE).round(2)
  end

  def with_vat
    cents_with_vat = cents + (cents * VAT_PERCENTAGE / 100.0).round(2)
    Price.new(cents_with_vat)
  end
end

price = Price.new(10_000)
puts price.eur # => 100.00
puts price.hrk # => 753.45
puts price.with_vat.eur # => 125.00
puts price.with_vat.hrk # => 941.81
```

```ruby
class Price
  HRK_EXCHANGE_RATE = 7.53450
  VAT_PERCENTAGE = 25

  attr_reader :cents

  def initialize(cents)
    @cents = cents
  end

  def eur
   cents / 100.0
  end

  def hrk
    (eur * HRK_EXCHANGE_RATE).round(2)
  end

  def with_vat
    cents_with_vat = cents + (cents * VAT / 100.0).round(2)
    Price.new(cents_with_vat)
  end

  def +(other_price)
    Price.new(cents + other_price.cents)
  end
end

hundred_euros = Price.new(10_000)
thousand_euros = Price.new(100_000)
total_price = hundred_euros + thousand_euros
p total_price # => #<Price:0x00007fcdcf85a940 @cents=110000>
puts total_price.eur # => 1100.00
```

Is Price a Value object?

We are not checking for equality

```ruby
class Price
  HRK_EXCHANGE_RATE = 7.53450
  VAT_PERCENTAGE = 25

  attr_reader :cents

  def initialize(cents)
    @cents = cents
  end


  …

  def ==(other_price)
    cents == other_price.cents
  end
end

hundred_euros = Price.new(10_000)
thousand_euros = Price.new(100_000)

hundred_euros == thousand_euros # => false
hundred_euros == Price.new(10_000) # => true
```

```
class Price
  include Comparable

    …

  def <=>(other_price)
    cents <=> other_price.cents
  end
end

hundred_euros = Price.new(10_000)
seven_hundred_euros = Price.new(70_000)
thousand_euros = Price.new(100_000)

hundred_euros == thousand_euros # => false
hundred_euros == Price.new(10_000) # => true

hundred_euros > seven_hundred_euros # => false
seven_hundred_euros < thousand_euros # => true

seven_hundred_euros.between?(hundred_euros, thousand_euros) # => true

[seven_hundred_euros, thousand_euros, hundred_euros].sort
# => [#<Price:0x00007fcdcf85a940 @cents=10000>, #<Price:0x00007fcdcf85a941 @cents=70000>,
#<Price:0x00007fcdcf85a942 @cents=100000>]
```

```ruby
class Product < ApplicationRecord
  def price
    Price.new(price_in_cents)
  end

  def price=(p)
    self.price_in_cents = p.cents
  end
end
```

```ruby
hundred_euros = Price.new(10_000)
p1 = Product.new(price: hundred_euros)

puts p1.price.eur # => 100.00
puts p1.price.hrk # => 753.45
puts p1.price.with_vat.eur # => 125.00
puts p1.price.with_vat.hrk # => 941.81

thousand_euros = Price.new(100_000)
p2 = Product.new(price: thousand_euros)

puts p2.price > p1.price # => true

total_price = p1.price + p2.price
p total_price # => #<Price:0x00007fcdcf85a940 @cents=110000>

puts total_price.with_vat.eur # => 1375.00
```

ActiveRecord models should not contain logic that does not directly correspond to reading/writing to the database

- Separation of concerns
- Combines behaviour with the data
- Removes duplication
- Improves code organisation (Easy to group operation on particular data in a single place)
- Leads to drastic simplification of a system

What is wrong here?

```ruby
class Money
  attr_accessor :currency, :amount

  def initialize(amount, currency)
    @amount = amount
    @currency = currency
  end
end
```

```ruby
eur = Money.new(10, "EUR")
p eur
# => #<Money:0x00007fd89f84ecc8 @amount=10, @currency="EUR">

eur.amount = 20
p eur
# => #<Money:0x00007fd89f84ecc8 @amount=20, @currency="EUR">
```

Value objects should be immutable!

If two value objects are created equal they should remain equal

Client/user should not be able to put the value object in an invalid state or introduce buggy behaviour after instantiation.

Immutable Value object

```ruby
class Money
  # Remove setters (replace attr_accessor with attr_reader)
  attr_reader :amount, :currency

  def initialize(amount, currency)
    @amount = amount
    @currency = currency
  end

  # If you really need a setter initialize a new value object instead of modifying the current one
  def with_amount(new_amount)
    Money.new(new_amount, currency)
  end
end

eur = Money.new(10, "EUR")
p eur # => #<Money:0x00007fd89f84ecc8 @amount=10, @currency="EUR">

other_eur = eur.with_amount(20)
p other_eur #<Money:0x00007fcb55872558 @amount=20, @currency="EUR">
```

Value Object candidates

- Amount of money (number and currency)
- Date range (start and end date)
- Geolocation (latitude and longitude)
- 2D coordinate (x and y)
- Distance (value and unit)
- Weight (value and unit)
- Temperature (degrees and unit)
- …

How to identify Value Objects

Attribute(s) with behaviour

```ruby
class Temperature
  OPTIMAL = 22

  def initialize(degrees)
    @degrees = degrees
  end

  def cold?
    @degrees < OPTIMAL
  end

  def hot?
    @degrees > OPTIMAL
  end

  def optimal?
    @degrees == OPTIMAL
  end
end

t1 = Temperature.new(20)
t2 = Temperature.new(22)
puts t1.optimal? # => false
puts t1.hot? # => false
puts t1.cold? # => true
puts t2.optimal? # => true
```

Inseparable attributes

```ruby
class Money
  def initialize(amount, currency)
    @amount = amount
    @currency = currency
  end

  def to_s
    “#{@amount} #{@currency}”
  end
end

eur = Money.new(10, "EUR")
puts eur # => "10 EUR"
```

Passing arguments together all the time

```ruby
class DateRange
  attr_reader :start_date, :end_date

  def initialize(start_date, end_date)
    @start_date = start_date
    @end_date = end_date
  end

  def include_date?(date)
    date >= start_date && date <= end_date
  end

  def include_date_range?(date_range)
    start_date <= date_range.start_date && end_date >= date_range.end_date
  end

  def overlap_date_range?(date_range)
    start_date <= date_range.end_date && end_date >= date_range.start_date
  end
end
```

Passing arguments together all the time

```ruby
class Address
  attr_reader :city, :zip, :street, :house_no

  def initialize(city, zip, street, house_no)
    @city, @zip, @street, @house_no = city, zip, street, house_no
  end

  def same_city?(other_address)
    city == other_address.city
  end

  def ==(other_address)
    city == other_address.city && zip == other_address.zip &&
     street == other_address.street && house == other_address.house_no
  end
end
```

Data.define(...) added to Ruby 3.2

```ruby
Price = Data.define(:amount, :currency)

Price = Data.define(:amount, :currency) do
  def with_vat
    Price.new(amount * 1.25, currency)
  end
end

hundred_euros = Price.new(100, "EUR")
thousand_euros = Price.new(1000, "EUR")
```

Where to put your Value Objects?

app/domain/**/*.rb
app/domain/price.rb

app/models/**/*.rb
app/models/product/price.rb
...

https://stanko.io/keep-it-boring-dont-surprise-me-52opnwWR6CBh

Libraries

- dry-struct https://dry-rb.org/gems/dry-struct
- Values https://github.com/tcrayford/Values
- Money https://github.com/RubyMoney/money
- Weight https://github.com/shemerey/weight
- …

The End

Questions