SuperGood

Undervalued

The Most Useful Design Pattern

Jared Norman	jardo.dev	VanRuby June 2024

I ❤️ Ruby

I ❤️ Ruby *and Rails*

The Anti-Java

But...

The Factory Method Pattern

Goals

- Factory Methods/Alternate Constructors
- Value and Data Objects
- How to use them together
- When to use them together

Code slides incoming

Generating a product feed

Database models → XML Doc

```ruby
Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
  xml.rss(base_xml_params) do
    xml.channel do
      xml.link(rel: "alternate", type: "text/html", href: @host)
      xml.title("DMC Atom Feed")
      xml.updated(Time.now.utc.iso8601)
      xml.author do
        xml.name("DMC")
      end
      Product.available.each do |product|
        xml.entry do
          xml["g"].id(product.id)
          xml["g"].title(product.name)
          xml["g"].description(product.description)
          xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
          xml["g"].link(Rails.application.routes.url_helpers.product_url(product, host: @host))
          xml["g"].image_link(product.images.cover_image&.attachment_url || "")
          xml["g"].availability(if product.in_stock? then "In Stock" else "Out of Stock" end)
          xml["g"].ean_barcode(product.ean_barcode)
        end
      end
    end
  end
end.to_xml
```

```ruby
Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
  xml.rss(base_xml_params) do
    xml.channel do
      xml.link(rel: "alternate", type: "text/html", href: @host)
      xml.title("DMC Atom Feed")
      xml.updated(Time.now.utc.iso8601)
      xml.author do
        xml.name("DMC")
      end
      Product.available.each do |product|
        xml.entry do
          xml["g"].id(product.id)
          xml["g"].title(product.name)
          xml["g"].description(product.description)
          xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
          xml["g"].link(Rails.application.routes.url_helpers.product_url(product, host: @host))
          xml["g"].image_link(product.images.cover_image&.attachment_url || "")
          xml["g"].availability(if product.in_stock? then "In Stock" else "Out of Stock" end)
          xml["g"].ean_barcode(product.ean_barcode)
        end
      end
    end
  end
end.to_xml
```

```ruby
Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
  xml.rss(base_xml_params) do
    xml.channel do
      xml.link(rel: "alternate", type: "text/html", href: @host)
      xml.title("DMC Atom Feed")
      xml.updated(Time.now.utc.iso8601)
      xml.author do
        xml.name("DMC")
      end
      Product.available.each do |product|
        xml.entry do
          xml["g"].id(product.id)
          xml["g"].title(product.name)
          xml["g"].description(product.description)
          xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
          xml["g"].link(Rails.application.routes.url_helpers.product_url(product, host: @host))
          xml["g"].image_link(product.images.cover_image&.attachment_url || "")
          xml["g"].availability(if product.in_stock? then "In Stock" else "Out of Stock" end)
          xml["g"].ean_barcode(product.ean_barcode)
        end
      end
    end
  end
end.to_xml
```

```ruby
Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
  xml.rss(base_xml_params) do
    xml.channel do
      xml.link(rel: "alternate", type: "text/html", href: @host)
      xml.title("DMC Atom Feed")
      xml.updated(Time.now.utc.iso8601)
      xml.author do
        xml.name("DMC")
      end
      Product.available.each do |product|
        xml.entry do
          xml["g"].id(product.id)
          xml["g"].title(product.name)
          xml["g"].description(product.description)
          xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
          xml["g"].link(Rails.application.routes.url_helpers.product_url(product, host: @host))
          xml["g"].image_link(product.images.cover_image&.attachment_url || "")
          xml["g"].availability(if product.in_stock? then "In Stock" else "Out of Stock" end)
          xml["g"].ean_barcode(product.ean_barcode)
        end
      end
    end
  end
end.to_xml
```

```ruby
Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
  xml.rss(base_xml_params) do
    xml.channel do






      Product.available.each do |product|
        xml.entry do
          xml["g"].id(product.id)
          xml["g"].title(product.name)
          xml["g"].description(product.description)
          xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
          xml["g"].link(Rails.application.routes.url_helpers.product_url(product, host: @host))
          xml["g"].image_link(product.images.cover_image&.attachment_url || "")
          xml["g"].availability(if product.in_stock? then "In Stock" else "Out of Stock" end)
          xml["g"].ean_barcode(product.ean_barcode)
        end
      end
    end
  end
end.to_xml
```

```
Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
  xml.rss(base_xml_params) do
    xml.channel do
      Product.available.each do |product|
        xml.entry do
          xml["g"].id(product.id)
          xml["g"].title(product.name)
          xml["g"].description(product.description)
          xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
          xml["g"].link(Rails.application.routes.url_helpers.product_url(product,
host: @host))
          xml["g"].image_link(product.images.cover_image&.attachment_url || "")
          xml["g"].availability(if product.in_stock? then "In Stock" else "Out of
Stock" end)
          xml["g"].ean_barcode(product.ean_barcode)
        end
      end
    end
  end
end.to_xml
```

```ruby
Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
  xml.rss(base_xml_params) do
    xml.channel do
      Product.available.each do |product|
        xml.entry do
          xml["g"].id(product.id)
          xml["g"].title(product.name)
          xml["g"].description(product.description)
          xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
          xml["g"].link(Rails.application.routes.url_helpers.product_url(product,
host: @host))
          xml["g"].image_link(product.images.cover_image&.attachment_url || "")
          xml["g"].availability(if product.in_stock? then "In Stock" else "Out of
Stock" end)
          xml["g"].ean_barcode(product.ean_barcode)
        end
      end
    end
  end
end.to_xml
```

```
Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
  xml.rss(base_xml_params) do
    xml.channel do
      Product.available.each do |product|
        xml.entry do
          xml["g"].id(product.id)
          xml["g"].title(product.name)
          xml["g"].description(product.description)
          xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
host
          xml["g"].availability(if product.in_stock? then "In Stock" else "Out of
Stock" end)
          xml["g"].ean_barcode(product.ean_barcode)
        end
      end
    end
  end
end.to_xml
```

```
Product.available.each do |product|
Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
  xml.rss(base_xml_params) do
    xml.channel do
      Product.available.each do |product|
        xml.entry do
          xml["g"].id(product.id)
          xml["g"].title(product.name)
          xml["g"].description(product.description)
          xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
          xml["g"].link(Rails.application.routes.url_helpers.product_url(product,
host: @host))
          xml["g"].image_link(product.images.cover_image&.attachment_url || "")
          xml["g"].availability(if product.in_stock? then "In Stock" else "Out of
Stock" end)
          xml["g"].ean_barcode(product.ean_barcode)
        end
      end
    end
  end
end.to_xml
```

```ruby
Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
  xml.rss(base_xml_params) do
    xml.channel do
      Product.available.each do |product|
        xml.entry do
          xml["g"].id(product.id)
          xml["g"].title(product.name)
          xml["g"].description(product.description)
          xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
          xml["g"].link(Rails.application.routes.url_helpers.product_url(product,
host: @host))
          xml["g"].image_link(product.images.cover_image&.attachment_url || "")
          xml["g"].availability(if product.in_stock? then "In Stock" else "Out of
Stock" end)
          xml["g"].ean_barcode(product.ean_barcode)
        end
      end
    end
  end
end.to_xml
```

```ruby
Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
  xml.rss(base_xml_params) do
    xml.channel do
      Product.available.each do |product|
        xml.entry do
          xml["g"].id(product.id)
          xml["g"].title(product.name)
          xml["g"].description(product.description)
          xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
          xml["g"].link(Rails.application.routes.url_helpers.product_url(product,
host: @host))
          xml["g"].image_link(product.images.cover_image&.attachment_url || "")
          xml["g"].availability(if product.in_stock? then "In Stock" else "Out of
Stock" end)
          xml["g"].ean_barcode(product.ean_barcode)
        end
      end
    end
  end
end.to_xml
```

Coupling & Cohesion

Coupling & Cohesion

Coupling & Cohesion

"A class should have only one reason to change"

-Robert C. Martin

What does this code know about?

— The Nokogiri API

— How to load products/what products to load

— The structure of the XML feed

— Product data mapping

— Product data formatting

```ruby
Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
  xml.rss(base_xml_params) do
    xml.channel do
      Product.available.each do |product|
        xml.entry do
          xml["g"].id(product.id)
          xml["g"].title(product.name)
          xml["g"].description(product.description)
          xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
          xml["g"].link(Rails.application.routes.url_helpers.product_url(product,
host: @host))
          xml["g"].image_link(product.images.cover_image&.attachment_url || "")
          xml["g"].availability(if product.in_stock? then "In Stock" else "Out of
Stock" end)
          xml["g"].ean_barcode(product.ean_barcode)
        end
      end
    end
  end
end.to_xml
```

```
Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
  xml.rss(base_xml_params) do
    xml.channel do
      Product.available.each do |product|
        xml.entry do
          xml["g"].id(product.id)
          xml["g"].title(product.name)
          xml["g"].description(product.description)
          xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
          xml["g"].link(Rails.application.routes.url_helpers.product_url(product,
host: @host))
          xml["g"].image_link(product.images.cover_image&.attachment_url || "")
          xml["g"].availability(if product.in_stock? then "In Stock" else "Out of
Stock" end)
          xml["g"].ean_barcode(product.ean_barcode)
        end
      end
    end
  end
end.to_xml
```

```ruby
products = Product.available

Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
  xml.rss(base_xml_params) do
    xml.channel do
      products.each do |product|
        xml.entry do
          xml["g"].id(product.id)
          xml["g"].title(product.name)
          xml["g"].description(product.description)
          xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
          xml["g"].link(Rails.application.routes.url_helpers.product_url(product, host:
@host))
          xml["g"].image_link(product.images.cover_image&.attachment_url || "")
          xml["g"].availability(if product.in_stock? then "In Stock" else "Out of Stock" end)
          xml["g"].ean_barcode(product.ean_barcode)
        end
      end
    end
  end
end.to_xml
```

```ruby
products = Product.available

Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
  xml.rss(base_xml_params) do
    xml.channel do
      products.each do |product|
        xml.entry do
          xml["g"].id(product.id)
          xml["g"].title(product.name)
          xml["g"].description(product.description)
          xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
          xml["g"].link(Rails.application.routes.url_helpers.product_url(product, host:
@host))
          xml["g"].image_link(product.images.cover_image&.attachment_url || "")
          xml["g"].availability(if product.in_stock? then "In Stock" else "Out of Stock" end)
          xml["g"].ean_barcode(product.ean_barcode)
        end
      end
    end
  end
end.to_xml
```

```ruby
class GoogleProductFeed
  def initialize(products)
    @products = products
  end

  def to_xml
    Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
      xml.rss(base_xml_params) do
        xml.channel do
          @products.each do |product|
            xml.entry do
              xml["g"].id(product.id)
              xml["g"].title(product.name)
              xml["g"].description(product.description)
              xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
              xml["g"].link(Rails.application.routes.url_helpers.product_url(product, host: @host))
              xml["g"].image_link(product.images.cover_image&.attachment_url || "")
              xml["g"].availability(if product.in_stock? then "In Stock" else "Out of Stock" end)
              xml["g"].ean_barcode(product.ean_barcode)
            end
          end
        end
      end
    end.to_xml
  end
end
```

```ruby
products = Product.available

Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
  xml.rss(base_xml_params) do
    xml.channel do
      products.each do |product|
        xml.entry do
          xml["g"].id(product.id)
          xml["g"].title(product.name)
          xml["g"].description(product.description)
          xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
          xml["g"].link(Rails.application.routes.url_helpers.product_url(product, host:
@host))
          xml["g"].image_link(product.images.cover_image&.attachment_url || "")
          xml["g"].availability(if product.in_stock? then "In Stock" else "Out of Stock" end)
          xml["g"].ean_barcode(product.ean_barcode)
        end
      end
    end
  end
end.to_xml
```

```ruby
products = Product.available

GoogleProductFeed.new(products).to_xml
```

```ruby
products = Product.available

GoogleProductFeed.new(products).to_xml
```

```ruby
class GoogleProductFeed
  def initialize(products)
    @products = products
  end

  def to_xml
    Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
      xml.rss(base_xml_params) do
        xml.channel do
          @products.each do |product|
            xml.entry do
              xml["g"].id(product.id)
              xml["g"].title(product.name)
              xml["g"].description(product.description)
              xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
              xml["g"].link(Rails.application.routes.url_helpers.product_url(product, host: @host))
              xml["g"].image_link(product.images.cover_image&.attachment_url || "")
              xml["g"].availability(if product.in_stock? then "In Stock" else "Out of Stock" end)
              xml["g"].ean_barcode(product.ean_barcode)
            end
          end
        end
      end
    end.to_xml
  end
end
```

```ruby
class GoogleProductFeed
  def initialize(products)
    @products = products
  end

  def to_xml
    Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
      xml.rss(base_xml_params) do
        xml.channel do
          @products.each do |product|
            xml.entry do
              xml["g"].id(product.id)
              xml["g"].title(product.name)
              xml["g"].description(product.description)
              xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
              xml["g"].link(Rails.application.routes.url_helpers.product_url(product, host: @host))
              xml["g"].image_link(product.images.cover_image&.attachment_url || "")
              xml["g"].availability(if product.in_stock? then "In Stock" else "Out of Stock" end)
              xml["g"].ean_barcode(product.ean_barcode)
            end
          end
        end
      end
    end.to_xml
  end
end
```

```ruby
class GoogleProductFeed
  def initialize(google_products)
    @google_products = google_products
  end

  def to_xml
    Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
      xml.rss(base_xml_params) do
        xml.channel do
          @google_products.each do |product|
            xml.entry do
              xml["g"].id(product.id)
              xml["g"].title(product.name)
              xml["g"].description(product.description)
              xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
              xml["g"].link(Rails.application.routes.url_helpers.product_url(product, host: @host))
              xml["g"].image_link(product.images.cover_image&.attachment_url || "")
              xml["g"].availability(if product.in_stock? then "In Stock" else "Out of Stock" end)
              xml["g"].ean_barcode(product.ean_barcode)
            end
          end
        end
      end
    end.to_xml
  end
end
```

```ruby
class GoogleProductFeed
  def initialize(google_products)
    @google_products = google_products
  end

  def to_xml
    Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
      xml.rss(base_xml_params) do
        xml.channel do
          @google_products.each do |product|
            xml.entry do
              xml["g"].id(product.id)
              xml["g"].title(product.name)
              xml["g"].description(product.description)
              xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
              xml["g"].link(Rails.application.routes.url_helpers.product_url(product, host: @host))
              xml["g"].image_link(product.images.cover_image&.attachment_url || "")
              xml["g"].availability(if product.in_stock? then "In Stock" else "Out of Stock" end)
              xml["g"].ean_barcode(product.ean_barcode)
            end
          end
        end
      end
    end.to_xml
  end
end
```

```ruby
class GoogleProductFeed
  def initialize(google_products)
    @google_products = google_products
  end

  def to_xml
    Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
      xml.rss(base_xml_params) do
        xml.channel do
          @google_products.each do |product|
            xml.entry do
              xml["g"].id(product.id)
              xml["g"].title(product.title)
              xml["g"].description(product.description)
              xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
              xml["g"].link(product.link)
              xml["g"].image_link(product.image_link)
              xml["g"].availability(if product.available then "In Stock" else "Out of Stock" end)
              xml["g"].ean_barcode(product.ean_barcode)
            end
          end
        end
      end
    end.to_xml
  end
end
```

```ruby
class GoogleProductFeed
  def initialize(google_products)
    @google_products = google_products
  end

  def to_xml
    Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
      xml.rss(base_xml_params) do
        xml.channel do
          @google_products.each do |product|
            xml.entry do
              xml["g"].id(product.id)
              xml["g"].title(product.title)
              xml["g"].description(product.description)
              xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
              xml["g"].link(product.link)
              xml["g"].image_link(product.image_link)
              xml["g"].availability(if product.available then "In Stock" else "Out of Stock" end)
              xml["g"].ean_barcode(product.ean_barcode)
            end
          end
        end
      end
    end.to_xml
  end
end
```

```ruby
xml["g"].title(product.title)
```

```
class GoogleProductFeed
  def initialize(google_products)
    @google_products = google_products
  end

  def to_
    Nokog
      xml
        x
          @google_products.each do |product|
            xml.entry do
              xml["g"].id(product.id)
              xml["g"].title(product.title)
              xml["g"].description(product.description)
              xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
              xml["g"].link(product.link)
              xml["g"].image_link(product.image_link)
              xml["g"].availability(if product.available then "In Stock" else "Out of Stock" end)
              xml["g"].ean_barcode(product.ean_barcode)
            end
          end
        end
      end
    end.to_xml
  end
end
```

```
xml["g"].link(product.link)
xml["g"].image_link(product.image_link)
```

```
xml["g"].availability(if product.available
then "In Stock" else "Out of Stock" end)
```

```
class GoogleProductFeed
  def initialize(google_products)
    @goo
  end

  def to
    Noko
      xm
        xml.channel do
          @google_products.each do |product|
            xml.entry do
              xml["g"].id(product.id)
              xml["g"].title(product.title)
              xml["g"].description(product.description)
              xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
              xml["g"].link(product.link)
              xml["g"].image_link(product.image_link)
              xml["g"].availability(if product.available then "In Stock" else "Out of Stock" end)
              xml["g"].ean_barcode(product.ean_barcode)
            end
          end
        end
      end
    end.to_xml
  end
end
```

```ruby
products = Product.available

GoogleProductFeed.new(products).to_xml
```

```ruby
products = Product.available

GoogleProductFeed.new(products).to_xml
```

```ruby
google_products = Product.available.map { |product|
  GoogleProduct.from_product(product)
}

GoogleProductFeed.new(google_products).to_xml
```

```ruby
GoogleProduct = Data.define(
  :id, :name, :description, :price, :link,
  :image_link, :available, :ean_barcode
) do
  def self.from_product(product)
    new(
      id: product.id,
      title: product.name,
      description: product.description,
      price: Money.from_amount(product.price.amount, product.price.currency),
      link: Rails.application.routes.url_helpers.product_url(product, host:
@host),
      image_link: product.images.cover_image&.attachment_url || "",
      available: product.in_stock?,
      ean_barcode: product.ean_barcode
    )
  end
end
```

```ruby
GoogleProduct = Data.define(
  :id, :name, :description, :price, :link,
  :image_link, :available, :ean_barcode
) do
  def self.from_product(product)
    new(
      id: product.id,
      title: product.name,
      description: product.description,
      price: Money.from_amount(product.price.amount, product.price.currency),
      link: Rails.application.routes.url_helpers.product_url(product, host:
@host),
      image_link: product.images.cover_image&.attachment_url || "",
      available: product.in_stock?,
      ean_barcode: product.ean_barcode
    )
  end
end
```

```ruby
GoogleProduct = Data.define(
  :id, :name, :description, :price, :link,
  :image_link, :available, :ean_barcode
) do
  def self.from_product(product)
    new(
      id: product.id,
      title: product.name,
      description: product.description,
      price: Money.from_amount(product.price.amount, product.price.currency),
      link: Rails.application.routes.url_helpers.product_url(product, host:
@host),
      image_link: product.images.cover_image&.attachment_url || "",
      available: product.in_stock?,
      ean_barcode: product.ean_barcode
    )
  end
end
```

Factory Method

```
GoogleProduct = Data.define(
  :id, :name, :description, :price, :link,
  :image_link, :available, :ean_barcode
) do
  def self.from_product(product)
    new(
      id: product.id,
      title: product.name,
      description: product.description,
      price: Money.from_amount(product.price.amount, product.price.currency),
      link: Rails.application.routes.url_helpers.product_url(product, host:
@host),
      image_link: product.images.cover_image&.attachment_url || "",
      available: product.in_stock?,
      ean_barcode: product.ean_barcode
    )
  end
end
```

```ruby
GoogleProduct = Data.define(
  :id, :name, :description, :price, :link,
  :image_link, :available, :ean_barcode
) do
  def self.from_product(product)
    new(
      id: product.id,
      title: product.name,
      description: product.description,
      price: Money.from_amount(product.price.amount, product.price.currency),
      link: Rails.application.routes.url_helpers.product_url(product, host: 
@host),
      image_link: product.images.cover_image&.attachment_url || "",
      available: product.in_stock?,
      ean_barcode: product.ean_barcode
    )
  end
end
```

Data Objects

Data Objects `!= Data.define`

Config Objects

```ruby
MyGem.configure do |config|
  config.some_preference = true
  config.foo = :bar
end
```

Bundles of data

```ruby
GoogleProduct = Data.define(
  :id, :name, :description, :price, :link,
  :image_link, :available, :ean_barcode
) do
  def self.from_product(product)
    new(
      id: product.id,
      title: product.name,
      description: product.description,
      price: Money.from_amount(product.price.amount, product.price.currency),
      link: Rails.application.routes.url_helpers.product_url(product, host:
@host),
      image_link: product.images.cover_image&.attachment_url || "",
      available: product.in_stock?,
      ean_barcode: product.ean_barcode
    )
  end
end
```

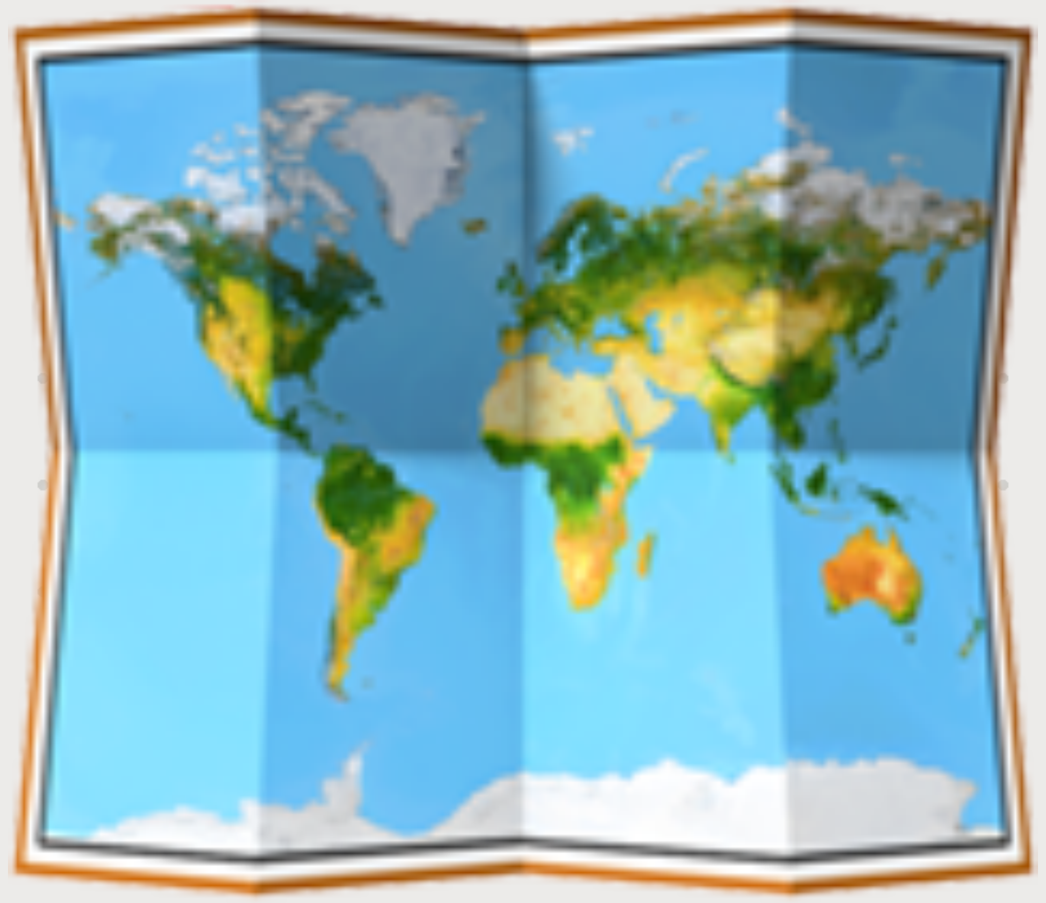
```ruby
GoogleProduct = Data.define(
  :id, :name, :description, :price, :link,
  :image_link, :available, :ean_barcode
) do
  def self.from_product(product)
    new(
      id: product.id,
      title: product.name,
      description: product.description,
      price: Money.from_amount(product.price.amount, product.price.currency),
      link: Rails.application.routes.url_helpers.product_url(product, host:
@host),
      image_link: product.images.cover_image&.attachment_url || "",
      available: product.in_stock?,
      ean_barcode: product.ean_barcode
    )
  end
end
```

Value Objects

JUL
17

❤️ 💚 💙

↔️



Entities and Values

Entities and Values

"Entities [...] are not fundamentally defined by their properties, but rather by [...] identity."

-Eric Evans

Entities and Values

```
Money.new(10, "CAD") != Money.new(10, "USD")
Money.new(100, "JPY") != Money.new(99, "JPY")
Money.new(100, "GBP") == Money.new(100, "GBP")
Money.new(10, "CAD") != Money.new(10, "USD")
Money.new(100, "JPY") != Money.new(99, "JPY")
Money.new(100, "GBP") == Money.new(100, "GBP")
```

```ruby
Money.new(10, "CAD") != Money.new(10, "USD")
Money.new(100, "JPY") != Money.new(99, "JPY")
Money.new(100, "GBP") == Money.new(100, "GBP")
```

```ruby
20.object_id == 20.object_id
#=> true

(2**64).object_id == (2**64).object_id
#=> false

2**64 == 2**64
#=> true
```

Immutability

What about strings?

Implementing Value/Data Objects

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

```ruby
Money.new(10, "CAD") == Money.new(10, "USD")
#=> false

Money.new(100, "JPY") == Money.new(99, "JPY")
#=> false

Money.new(100, "GBP") == Money.new(100, "GBP")
#=> true
```

```ruby
Money.new(amount: 100, currency:"GBP") ==
  Money.new(100, "GBP")
#=> true
```

```
hundo = Money.new(amount: 100, currency:"GBP")
hundo.amount = 200
NoMethodError: undefined method `amount=' for an instance of
Money
```

```ruby
Money = Data.define(:amount, :currency) do
  def +(other)
    unless other.is_a?(Money)
      raise TypeError, "Unsupported argument type: 
#{other.class.name}"
    end

    if other.currency != currency
      raise ArgumentError, "Can only add Money values with the 
same currency"
    end

    Money.new(amount + other.amount, currency)
  end
end
```

```ruby
Money = Data.define(:amount, :currency) do
  def +(other)
    unless other.is_a?(Money)
      raise TypeError, "Unsupported argument type: #{other.class.name}"
    end

    if other.currency != currency
      raise ArgumentError, "Can only add Money values with the same currency"
    end

    Money.new(amount + other.amount, currency)
  end
end
```

```ruby
Money = Data.define(:amount, :currency) do
  def +(other)
    unless other.is_a?(Money)
      raise TypeError, "Unsupported argument type: #{other.class.name}"
    end

    if other.currency != currency
      raise ArgumentError, "Can only add Money values with the same currency"
    end

    Money.new(amount + other.amount, currency)
  end
end
```

```ruby
Money = Data.define(:amount, :currency) do
  def +(other)
    unless other.is_a?(Money)
      raise TypeError, "Unsupported argument type: 
#{other.class.name}"
    end

    if other.currency != currency
      raise ArgumentError, "Can only add Money values with the 
same currency"
    end

    Money.new(amount + other.amount, currency)
  end
end
```

```ruby
Config = Struct.new(
  :feature_a_enabled,
  :feature_b_enabled
)
```

```ruby
Config = Struct.new(
  :feature_a_enabled,
  :feature_b_enabled,
  keyword_init: true
)
```

```ruby
config = Config.new(
  feature_a_enabled: true,
  feature_b_enabled: false
)

config.feature_a_enabled = false
# No error
```

```ruby
Money = Data.define(
  :amount,
  :currency
) do
  def self.from_price(price)
    new(
      amount: price.amount,
      currency: price.currency
    )
  end
end
```

```ruby
Config = Struct.new(
  :feature_a_enabled,
  :feature_b_enabled
) do
  def self.from_user(user)
    new(
      user.membership_active?,
      user.tier == :premium
    )
  end
end
```

```ruby
Money = Data.define(
  :amount,
  :currency
) do
  def self.from_price(price)
    new(
      amount: price.amount,
      currency: price.currency
    )
  end
end
```

```ruby
Config = Struct.new(
  :feature_a_enabled,
  :feature_b_enabled
) do
  def self.from_user(user)
    new(
      user.membership_active?,
      user.tier == :premium
    )
  end
end
```

```ruby
Money = Data.define(
  :amount,
  :currency
) do
  def self.from_price(price)
    new(
      amount: price.amount,
      currency: price.currency
    )
  end
end
```

```ruby
Config = Struct.new(
  :feature_a_enabled,
  :feature_b_enabled
) do
  def self.from_user(user)
    new(
      user.membership_active?,
      user.tier == :premium
    )
  end
end
```

Summary

— Data.define for value object

— Data.define for immutable data objects

— Struct.new for mutable data objects

— Never use OpenStruct

Back to the example

```ruby
google_products = Product.available.map { |product|
  GoogleProduct.from_product(product)
}

GoogleProductFeed.new(google_products).to_xml
```

```ruby
GoogleProduct = Data.define(
  :id, :name, :description, :price, :link,
  :image_link, :available, :ean_barcode
) do
  def self.from_product(product)
    new(
      id: product.id,
      title: product.name,
      description: product.description,
      price: Money.from_amount(product.price.amount, product.price.currency),
      link: Rails.application.routes.url_helpers.product_url(product, host:
@host),
      image_link: product.images.cover_image&.attachment_url || "",
      available: product.in_stock?,
      ean_barcode: product.ean_barcode
    )
  end
end
```

```ruby
class GoogleProductFeed
  def initialize(google_products)
    @google_products = google_products
  end

  def to_xml
    Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
      xml.rss(base_xml_params) do
        xml.channel do
          @google_products.each do |product|
            xml.entry do
              xml["g"].id(product.id)
              xml["g"].title(product.title)
              xml["g"].description(product.description)
              xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
              xml["g"].link(product.link)
              xml["g"].image_link(product.image_link)
              xml["g"].availability(if product.available then "In Stock" else "Out of Stock" end)
              xml["g"].ean_barcode(product.ean_barcode)
            end
          end
        end
      end
    end.to_xml
  end
end
```

What about handling change?

```ruby
google_products = Product.available.map { |product|
  GoogleProduct.from_product(product)
}

GoogleProductFeed.new(google_products).to_xml
```

```ruby
GoogleProduct = Data.define(
  :id, :name, :description, :price, :link,
  :image_link, :available, :ean_barcode
) do
  def self.from_product(product)
    new(
      id: product.id,
      title: product.name,
      description: product.description,
      price: Money.from_amount(product.price.amount, product.price.currency),
      link: Rails.application.routes.url_helpers.product_url(product, host:
@host),
      image_link: product.images.cover_image&.attachment_url || "",
      available: product.in_stock?,
      ean_barcode: product.ean_barcode
    )
  end
end
```

```ruby
class GoogleProductFeed
  def initialize(google_products)
    @google_products = google_products
  end

  def to_xml
    Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
      xml.rss(base_xml_params) do
        xml.channel do
          @google_products.each do |product|
            xml.entry do
              xml["g"].id(product.id)
              xml["g"].title(product.title)
              xml["g"].description(product.description)
              xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
              xml["g"].link(product.link)
              xml["g"].image_link(product.image_link)
              xml["g"].availability(if product.available then "In Stock" else "Out of Stock" end)
              xml["g"].ean_barcode(product.ean_barcode)
            end
          end
        end
      end
    end.to_xml
  end
end
```

What about our tests?

```ruby
google_products = Product.available.map { |product|
  GoogleProduct.from_product(product)
}

GoogleProductFeed.new(google_products).to_xml
```

```ruby
GoogleProduct = Data.define(
  :id, :name, :description, :price, :link,
  :image_link, :available, :ean_barcode
) do
  def self.from_product(product)
    new(
      id: product.id,
      title: product.name,
      description: product.description,
      price: Money.from_amount(product.price.amount, product.price.currency),
      link: Rails.application.routes.url_helpers.product_url(product, host:
@host),
      image_link: product.images.cover_image&.attachment_url || "",
      available: product.in_stock?,
      ean_barcode: product.ean_barcode
    )
  end
end
```

```ruby
class GoogleProductFeed
  def initialize(google_products)
    @google_products = google_products
  end

  def to_xml
    Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
      xml.rss(base_xml_params) do
        xml.channel do
          @google_products.each do |product|
            xml.entry do
              xml["g"].id(product.id)
              xml["g"].title(product.title)
              xml["g"].description(product.description)
              xml["g"].price("%.2f %s" % [product.price.amount, product.price.currency])
              xml["g"].link(product.link)
              xml["g"].image_link(product.image_link)
              xml["g"].availability(if product.available then "In Stock" else "Out of Stock" end)
              xml["g"].ean_barcode(product.ean_barcode)
            end
          end
        end
      end
    end.to_xml
  end
end
```

Tips for factory methods

Boundaries

Multiple factory methods

One to many, many to one

Values highlight what matters to the consumer

Facades

```ruby
class GoogleProduct
  def initialize(product)
    @product = product
  end

  delegate :id, :description, to: :@product

  def title
    @product.name
  end

  def price
    "%.2f %s" % [@product.price.amount, @product.price.currency]
  end

  def link
    Rails.application.routes.url_helpers.product_url(@product)
  end

  # ...
end
```

In conclusion

- Factory methods are alternate constructors
- Use them to draw boundaries
- Value objects represent domain concepts
- Data objects represent bundles of data
- Both can be initialized independently of their factory methods
- Look for data transformation

References

— Data Proposal: https://bugs.ruby-lang.org/issues/16122

— Data Docs: https://docs.ruby-lang.org/en/3.2/Data.html

— Struct Docs: https://docs.ruby-lang.org/en/3.2/Struct.html

— Ruby Money: https://github.com/RubyMoney/money

Mastodon:
@jared@supergood.social

Twitter:
@jardonamron

Podcast:
https://deadcode.website

Web:
https://jardo.dev

Super Good:
https://supergood.software

Mastodon:
@jared@supergood.social

Twitter:
@jardonamron

Podcast:
https://deadcode.website

Web:
https://jardo.dev

Super Good:
https://supergood.software

Thanks!