

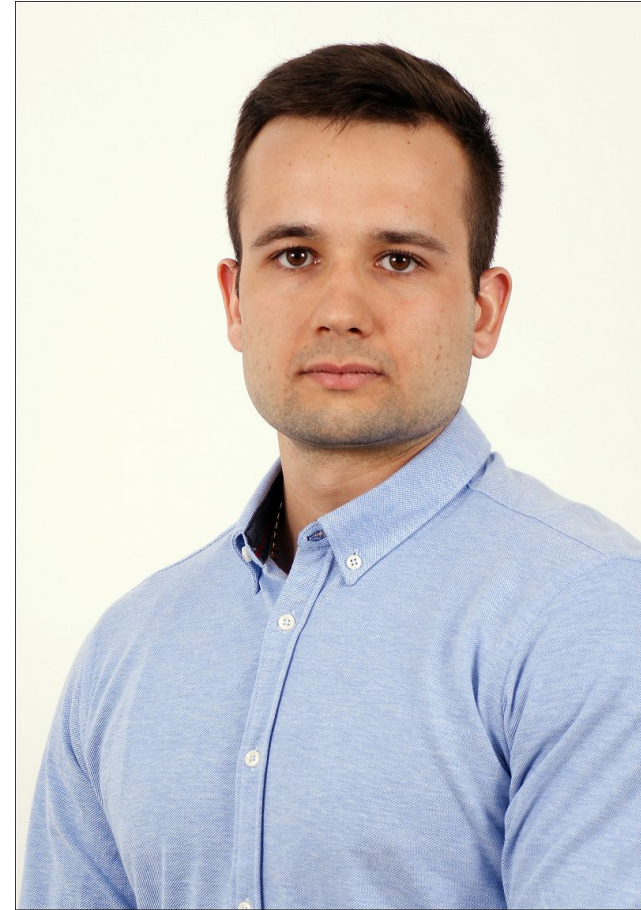
Generowanie CSV z wykorzystaniem wzorca fabrykacji

Michał Zynek

Kraków Ruby User Group, 15 październik 2019

Trochę o mnie

- ✓ Pracuję jako programista Ruby on Rails od 3 lat
- ✓ Poprzednio pracowałem w banku przy aplikacjach wspierających biznes (procesy i pracowników)
- ✓ Wcześniej interesowałem się programowaniem m. in. na studiach
- ✓ Jestem z wykształcenia fizykiem, a studiowałem na Politechnice Wrocławskiej



Dlaczego raporty? I dlaczego CSV?



Bardzo popularne
rozwiązanie w wielu
systemach



Służy również jako
system wymiany
danych



Dobre narzędzie
biznesowe dla osób
nietechnicznych



Samo zagadnienie
dotyczy też innych
rodzajów raportów

Refaktoryzacja - szybkie case study

- ✓ Duży dług technologiczny w projekcie i legacy code
- ✓ Raporty - jedna z kluczowych funkcjonalności dla klienta
- ✓ Generowanie CSV odbywało się za pomocą workerów
- ✓ Każdy z nich ma powtarzający się algorytm
- ✓ Problem: zmiana w algorytmie generującym raport
- ✓ Problem: Utrudniona refaktoryzacja kodu w celu przyspieszenia generowania raportów

```
class SomeCSVWorker
  include Sidekiq::Worker
  sidekiq_options queue: 'package_import'

  def perform(id)
    # previously implemented report saving in database
    export = Export.find(id)
    require 'csv'

    csv_file = ''

    # some logic for CSV data generation

    # appending headers
    csv_file << CSV.generate_line(%w[ID Some Attributes])
    #
    SomeResource.where('created_at >= ?').each do |res|
      csv_file << CSV.generate_line([res.id, res.attribute, res.other_attribute])
    end

    # temp file creation
    file_name = Rails.root.join('tmp', "resource-timestamp.csv")

    File.open(file_name, 'wb') do |file|
      file.puts csv_file
    end

    # AWS upload
    s3 = AWS::S3.new

    key = File.basename(file_name)

    file = s3.buckets['bucket-name'].objects["csvs/#{key}"].write(file: file_name)
    file.acl = :public_read

    # upload key in database
    export.update(file_path: key)
  end
end
```

Pierwsze kroki - service object

- ✓ Pierwszym krokiem refaktoryzacji było przeniesienie ogólnego algorytmu generowania csv do logiki serwisu
- ✓ Większość kroków jest powtarzalna i taka sama dla wszystkich raportów
- ✓ Jak obsłużyć generowanie danych (generate_rows), które jest inne dla każdego z raportów?

```
class CSVExportService
  def initialize; end

  def perform
    export = Export.create(key: key, status: :pending)
    csv_data = generate_rows
    file = write_to_file(csv_data)
    key = upload_to_s3(file)
    Export.update(path: key, status: :completed)
  end

  def write_to_file(data); end

  def generate_rows; end

  def upload_to_s3(file); end
end
```

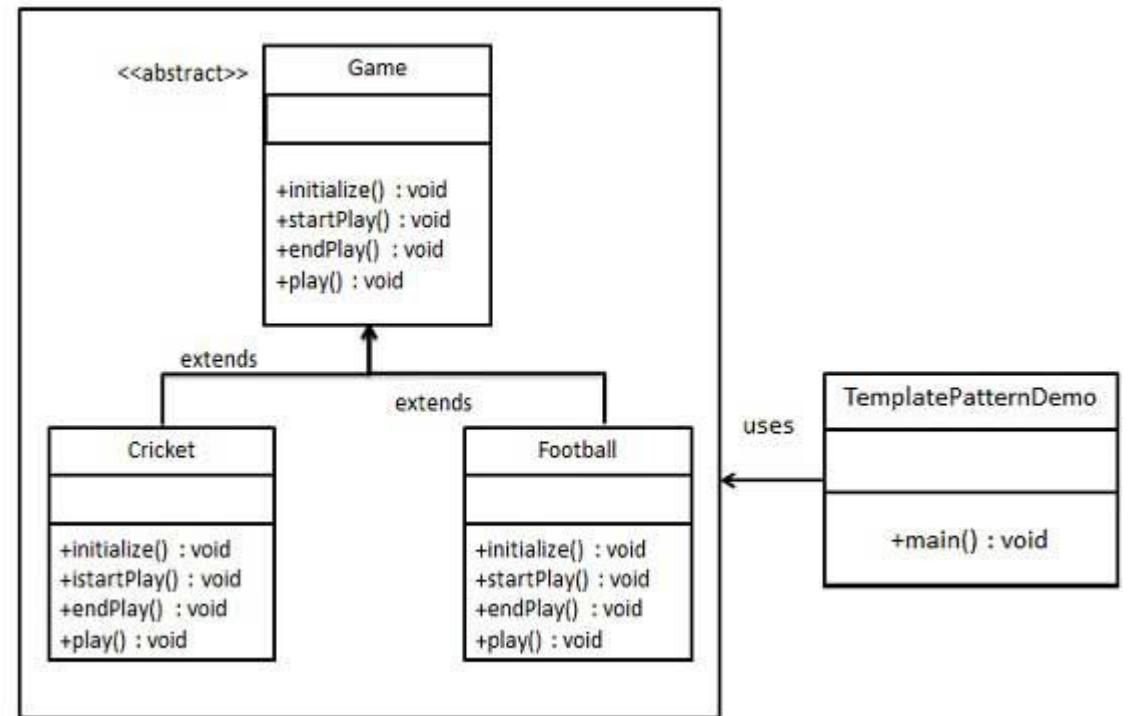
O wzorcach słów kilka

Jakie narzędzia mogą nam się przydać?



Metoda szablonowa

- ✓ Metoda behawioralna
- ✓ Pozwala na hermetyzację wydzielonej części algorytmu
- ✓ Poszczególne klasy są odpowiedzialne za implementację kroków algorytmu



Metoda szablonowa

```
class TemplateCsvService
  attr_reader :export

  def initialize
    @export = Export.new(key: @key, status: :pending)
  end


  def generate_report
    @export.create
    rows = generate_rows
    file = create_csv(headers: headers, rows: rows)
    key = upload_to_bucket(file: file)
    @export.update(status: :completed, path: key)
  end

  private

  def generate_rows
    raise NotImplementedError
  end


  def create_csv(headers:, rows:); end

  def upload_to_bucket(file:); end
end
```



```
class SomeDataGenerator < TemplateCsvService
  private

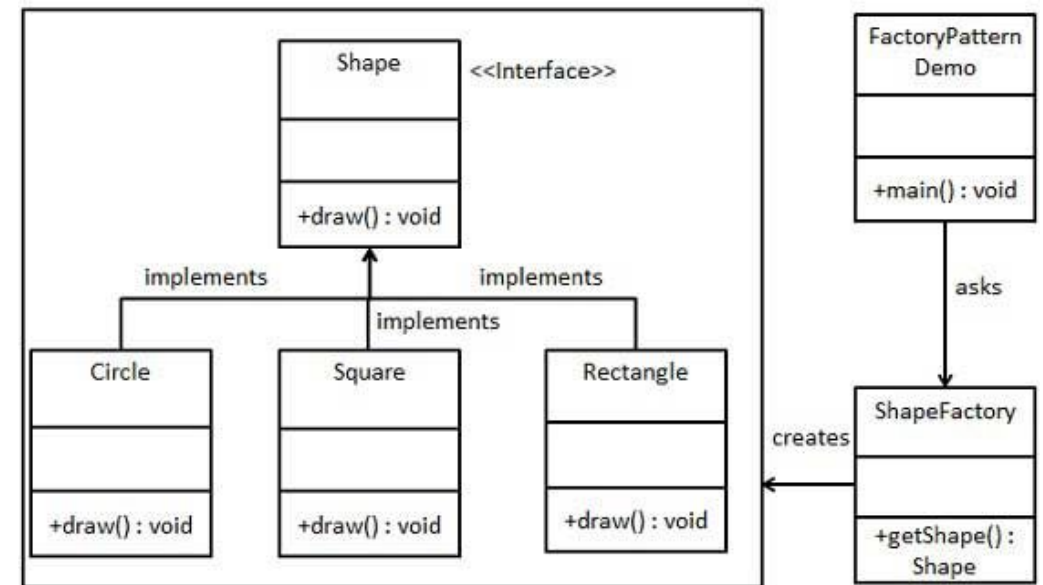
  def generate_rows
    # return ['Something', 'Different']
  end
end
```



```
:065 > exporter = SomeDataGenerator.new
SomeDataGenerator:0x00007fffe353cda8>
:066 > exporter.generate_report
```


Faktoryzacja

- ✓ Wzorzec kreacyjny (!)
- ✓ Pozwala na tworzenie obiektów różnych klas bez ujawniania logiki tworzenia
- ✓ Możemy odwoływać się do tej samej metody (czyli np. `generate_rows`) pomimo różnej implementacji



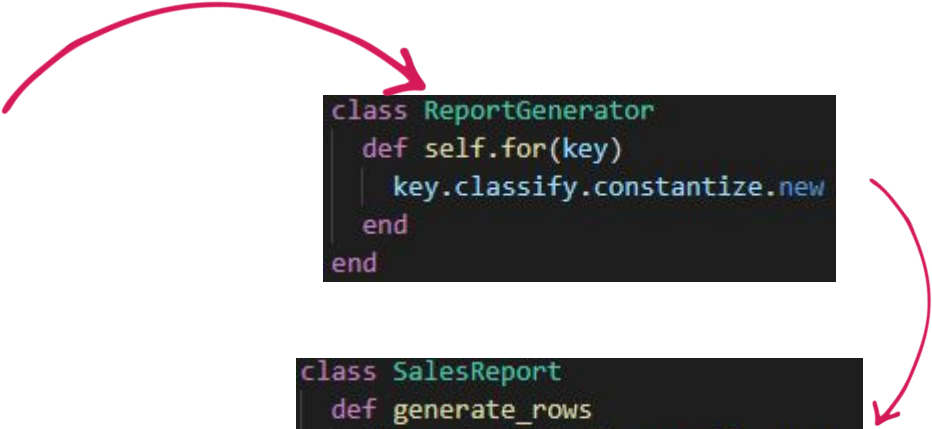
Faktoryzacja

```
class FactoryCsvService
  def initialize(key)
    @key = key
    @export = Export.new(key: @key, status: :pending)
  end

  def generate_report
    @export.create
    rows = ReportGenerator.for(@key).generate_rows
    file = create_csv(rows: rows)
    key = upload_to_bucket(file: file)
    @export.update(status: :completed, path: key)
  end


  def create_csv(rows:); end

  def upload_to_bucket(file:); end
end
```



```
class ReportGenerator
  def self.for(key)
    key.classify.constantize.new
  end
end
```

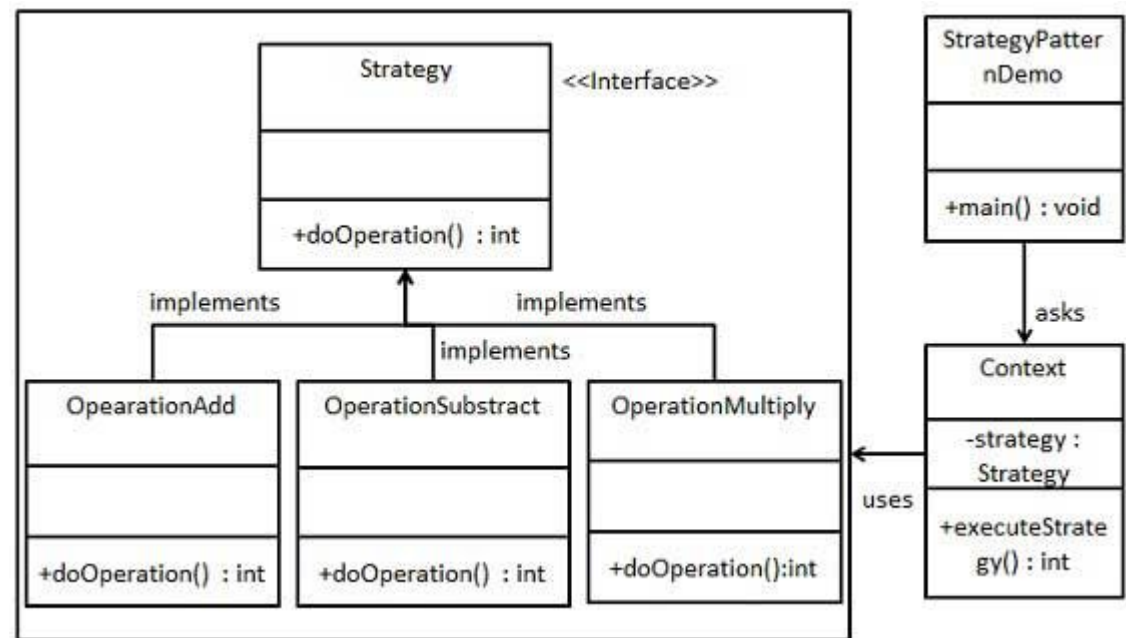
```
class SalesReport
  def generate_rows
    ["Test", "of", "Factory"].to_csv
  end
end
```



```
from bin/rails@0.11.0 in `main`
2.3.7 :025 > ReportGenerator.for('sales_report')
=> #<SalesReport:0x00007ffff83488c0>
2.3.7 :026 > ReportGenerator.for('sales_report').generate_rows
=> "Test,of,Factory\n"
```

Wzorzec strategii

- ✓ Metoda behawioralna (możemy kojarzyć jako Policy pattern)
- ✓ Bazuje na kompozycji, nie na dziedziczeniu - tworzy obiekty reprezentujące różne strategie i obiekt kontekstowy
- ✓ Pozwala wydzielić do klas strategicznych kroki algorytmu, którego zachowanie różni się w zależności od obiektu strategii.



Wzorzec strategii

```
class DummyCsvContext
  attr_accessor :report_strategy

  def initialize(report_strategy)
    @strategy = report_strategy
  end

  def generate_report
    @export.create
    rows = @strategy.generate_rows
    file = create_csv(rows: rows)
    key = upload_to_bucket(file: file)
    @export.update(status: :completed, path: key)
  end

  def create_csv(rows:); end

  def upload_to_bucket(file:); end
end
```



```
class SalesReportStrategy
  def self.generate_rows
    # returns valid set of rows
    headers << query.map(&:csv_attrs).to_csv
  end

  private

  def headers
    # returns ['ID', 'Total Price', 'Client Fullname', 'Products']
  end
end
```

Fabrykacja, czy strategia?



Strategia dotyczy zachowania. Fabrykacja dotyczy tworzenia

Random stack overflow user

Dlaczego fabrykacja?



Chcemy tworzyć obiekt
na podstawie klucza i
uniknąć dziedziczenia



Konieczność
niestandardowej
implementacji dla
poszczególnych klas



Pomocne rozwiązanie w
architekturze docelowej
(dla innych akcji niż
eksport)


Faktoryzacja - efekt końcowy

```
require 'csv'

class CSVService
  include CSVService::CSVErrors

  def initialize(key, action, filters = {})
    @action = action
    @key = key
    @filters = filters
    @bucket = ENV['S3_BUCKET_NAME']
    @export = Export.create(status: :pending, key: @key)
  end

  def perform
    # this eventually goes to mediator pattern to handle
    # multiple actions
    case @action
    when :export then export
    else raise UndefinedActionError
    end
  end
end
```



```
def export
  raise NoKeyProvidedError if @key.blank?

  rows = ReportGenerator.for(@key).generate_rows
  file_name = generate_file_name
  created_file = save_file(path: file_name, data: rows)
  upload_to_s3(file: created_file)
  @export.update(status: :completed, file_path: file_name)
end

def generate_file_name
  "#{@key}_report_#{DateTime.current.to_i}.csv"
end

def save_file(path:, data:)
  File.open(path, 'wb') do |file|
    file.puts data
  end
end

def upload_to_s3(file:)
  client = AWS::S3.new
  filename = File.basename(file)
  s3.buckets[@bucket].objects["generated_reports/#{@key}/#{filename}"].write(file: file)
end
end
```


Faktoryzacja - przykładowy fabrykat

```
class ReportGenerator
  def self.for(key)
    key.classify.constantize.new
  end
end
```

```
class SalesReport
  def headers
    ["Order ID", "First Name", "Last Name", "Phone", "Total Price", "Product S/Q"]
  end

  def generate_rows
    Order.for_order_report.to_csv
  end
end
```


Podsumowanie i przyszłe zastosowania

- ✓ Zastosowanie fabrykacji pozwoliło na refaktoryzację starego kodu i łatwiejsze okrycie go testami
- ✓ Nowe raporty mogą być generowane w prosty sposób - za pomocą tworzenia następnych klas
- ✓ Zyskaliśmy możliwość dopisania nowych funkcjonalności - np. Generowania raportów z szablonu
- ✓ Doświadczenie na przyszłość - oprócz refaktoryzacji, mamy kolejne narzędzia i pomysły - wzorzec strategii i metodę szablonu

Dziękuję za uwagę!

Źródła użyte w prezentacji

1. <http://rubyblog.pro/2016/10/factory-method-pattern-in-ruby>
2. <http://rubyblog.pro/2016/10/ruby-strategy-pattern>
3. <http://rubyblog.pro/2016/10/inheritance-and-template-pattern-simple-example>
4. Refactoring Guru - Factory, Template and Strategy Pattern (dostępne na stronie)
5. <https://medium.com/@dljerome/design-patterns-in-ruby-factory-method-e4e4cd995254>
6. <https://www.sitepoint.com/solving-design-anti-patterns-in-ruby-fix-the-factory/>
7. <http://www.chrisrolle.com/en/blog/the-factory-pattern>
8. <https://www.bigcommerce.com/ecommerce-answers/what-csv-file-and-what-does-it-mean-my-ecommerce-business/>
9. <https://imgflip.com/memegenerator/>
10. https://www.tutorialspoint.com/design_pattern/factory_pattern.htm
11. https://www.tutorialspoint.com/design_pattern/strategy_pattern.htm
12. https://www.tutorialspoint.com/design_pattern/template_pattern.htm