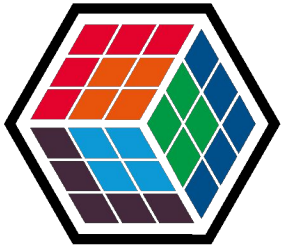
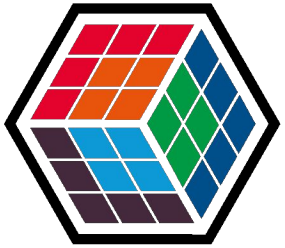


THE DEVELOPER'S CONFERENCE

Trilha – Ruby

Kamila Santos Oliveira
Software Developer



THE DEVELOPER'S CONFERENCE

Vou ter que refatorar. E agora?

Técnicas de refatoração em ruby

Kamila Santos Oliveira

21 anos,

Dev na Cognizant,

~ 3 anos na área,

Graduanda em ciência da computação

THE DEVELOPER'S CONFERENCE

Agenda

THE DEVELOPER'S CONFERENCE

- O que é refatoração
- Refatoração no TDD
- OOP
- S.O.L.I.D
- DRY
- Renomear método
- Extrair método
- Mover método

Agenda

THE DEVELOPER'S CONFERENCE

- Mover campo
- Extrair classe
- Factory
- Template Method
- Strategy
- Adapter

THE
DEVELOPER'S
CONFERENCE

O que é refatoração?

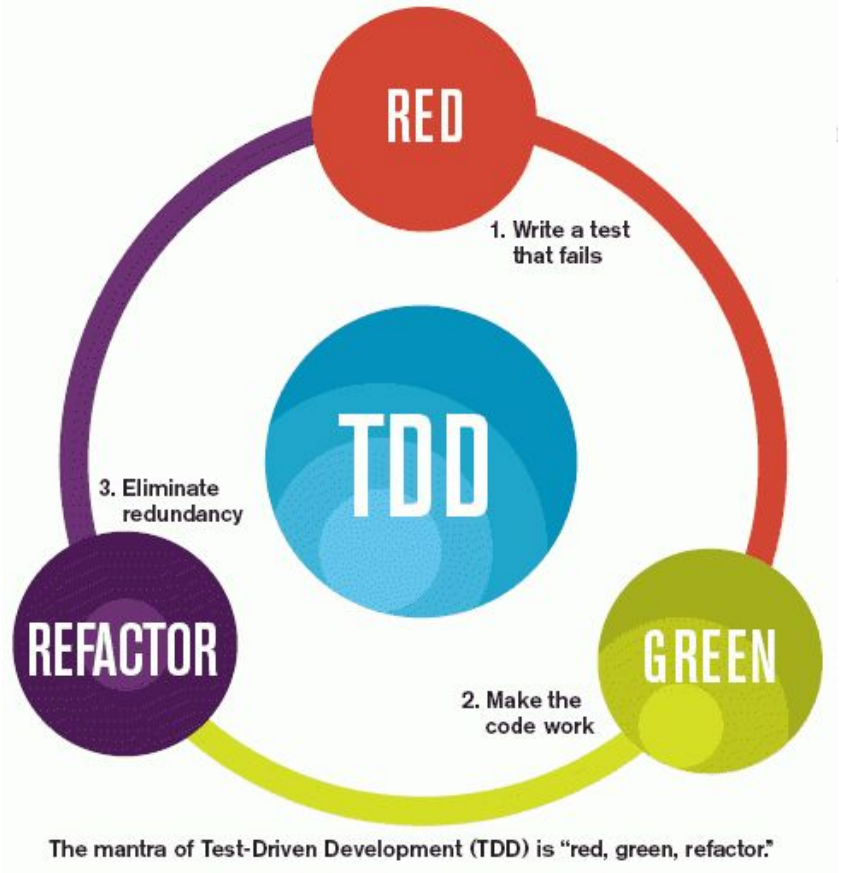
THE DEVELOPER'S CONFERENCE

Refatoração

Refatoração se trata do processo de alterar um sistema de software de uma forma que ela não tenha seu comportamento externo alterado, mas tenha a sua estrutura interna melhorada.

THE
DEVELOPER'S
CONFERENCE

Refatoração no TDD



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

THE
DEVELOPER'S
CONFERENCE

OOP

THE DEVELOPER'S CONFERENCE

OOP

Abstração: Objetos abstrai do do mundo real com uma identidade, propriedades e ações.

Encapsulamento: esconder objetos do resto da aplicação.

Herança: herda da classe pai e modifica alguns comportamentos.

Polimorfismo: alteração do funcionamento interno

THE
DEVELOPER'S
CONFERENCE

S.O.L.I.D

THE DEVELOPER'S CONFERENCE

S : Single responsibility principle - Prinípio da responsabilidade única

Uma classe deve ter uma e somente uma responsabilidade, se tiver mais de uma devemos refatorar.

THE DEVELOPER'S CONFERENCE

SOLID

O : Open/closed principle - Princípio do Aberto/Fechado

Devemos ser capazes de estender um comportamento de determinada classe sem precisar modificá-lo, pode ter seu comportamento alterado com facilidade se necessário porém através herança,interface....

THE DEVELOPER'S CONFERENCE

L : Liskov substitution principle Princípio da substituição de Liskov

As classes derivadas devem poder ser substituíveis pelas classes bases

SOLID

THE DEVELOPER'S CONFERENCE

I : Interface segregation principle - Princípio da segregação de interfaces

Melhor ter várias interfaces específicas do que um interface geral, crie interfaces granulares para cada “cliente”

THE DEVELOPER'S CONFERENCE

SOLID

D: Dependency inversion principle - Princípio da inversão de dependência

Dependa das abstrações, não das implementações, as abstrações tem menores alterações e facilitam a implementação.

THE
DEVELOPER'S
CONFERENCE

DRY

THE DEVELOPER'S CONFERENCE

Don't Repeat YourSelf

Cada pedaço de código deve ter uma única representação, sem ambiguidades no resto do sistema, não teremos que alterar em várias partes uma responsabilidade que está espalhada pelo sistema,.

DRY

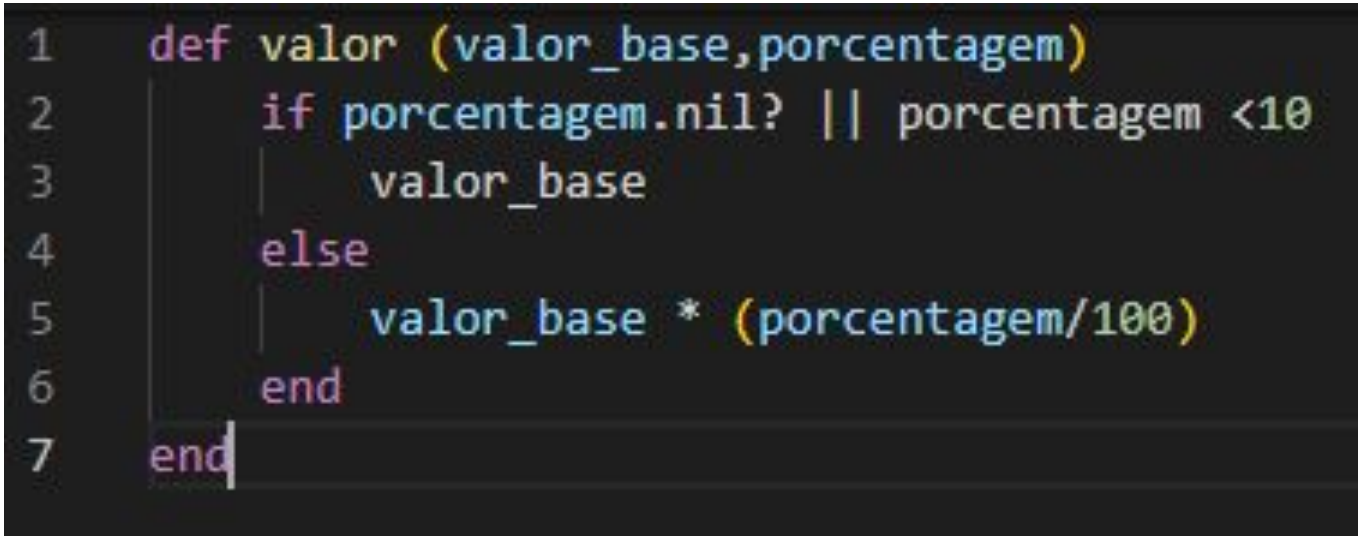
THE
DEVELOPER'S
CONFERENCE

Renomear método

THE DEVELOPER'S CONFERENCE

Para deixar o código mais expressivo, vamos renomear este método de modo que fique mais fácil de identificar qual é o seu propósito:

Renomear método


1 def valor (valor_base,porcentagem)
2 if porcentagem.nil? || porcentagem <10
3 valor_base
4 else
5 valor_base * (porcentagem/100)
6 end
7 end

THE
DEVELOPER'S
CONFERENCE

```ruby
def valor_final(valor_base,porcentagem)
    if porcentagem.nil?||
        porcentagem <10
        valor_base
    else
        valor_base *(porcentagem/100)
    end
end
```

THE
DEVELOPER'S
CONFERENCE

Extrair método

THE DEVELOPER'S CONFERENCE

Esta técnica pode ser utilizada quando precisamos quebrar um método que possui mais de uma responsabilidade.

Extrair método

```ruby
def inativar_clientes
    inativar clientes =
    Clientes.all.select do |cliente|
        cliente.ultima_compra > 1.month.ago && cliente.ativo?
    end
    clientes_para_inativar.each(&:deactivate)
    AvisarClientePorWpp.inativados(inativar_clientes)
end
```

THE DEVELOPER'S CONFERENCE

```ruby
def clientes_para_inativar
    Clientes.all.select do |cliente|
        cliente.ultima_compra >1.month.ago && cliente.ativo?
    end
```

THE
DEVELOPER'S
CONFERENCE

```ruby
def inativar_clientes
    clientes =clientes_para_inativar
    clientes.each(&:deactivate)
    AvisarClientePorWpp.inativados(clientes)
end
```

THE
DEVELOPER'S
CONFERENCE

Mover método

THE DEVELOPER'S CONFERENCE

Esta técnica pode ser utilizada quando se tem um método que usa mais informações de uma classe externa do que da sua própria classe.

Essa alteração reduz a complexidade do nosso código, pois ele vai acessar as informações da nova classe de forma direta

Mover método

```ruby
def inativar_clientes
    clientes= clientes_para_inativar
    clientes.each(&:deactivate)
    AvisarClientePorWpp.inativados(clientes)
end
def clientes_para_inativar
    Clientes.all.select do |cliente|
        cliente.ultima_compra > 1.month.ago && cliente.ativo?
    end
end
```

THE DEVELOPER'S CONFERENCE

Para isso, devemos criar a classe Clientes e mover o método clientes_para_inativar para dentro dela, transferindo sua responsabilidade.

Mover método

```ruby
class Clientes
    def self.clientes_para_inativar
        Clientes.all.select do |cliente|
            cliente.ultima_compra > 1.month.ago && cliente.ativo?
        end
    end
end
```

Depois desta separação, a classe InativarClientesWorker ficaria assim:

THE DEVELOPER'S CONFERENCE

Mover método

```ruby
class InativarClientesWorker
    def inativar_clientes
        clientes = clientes_para_inativar
        clientes.each(&:deactivate)
        AvisarClientePorWpp.inativados(clientes)
    end
end
```

THE
DEVELOPER'S
CONFERENCE

Mover campo

é necessário quando temos um campo (atributo) que é mais usado em uma classe externa do que na sua própria classe.

THE DEVELOPER'S CONFERENCE

Mover campo

THE DEVELOPER'S CONFERENCE

Seu maior benefício é que, ao ser criado na nova classe, temos a garantia de que ele ficará protegido de modificações externas ou criamos um atributo de leitura e – se preciso – de escrita na classe de destino, e alteramos todos os locais em que ele é referenciado.

Mover campo

```ruby
class ContaBancaria
    attr_accessor : taxa_juros
    def juros_por_dias(numero_de_dias, dias)
        @taxa_juros * numero_de_dias * dias / 365;
    end
end
```

THE DEVELOPER'S CONFERENCE

```ruby
class ContaCorrente
    attr_accessor : taxa_juros
end

class ContaBancaria
    def juros_por_dias(numero_de_dias, dias)
        @contacorrente.taxa_juros * numero_de_dias * dias / 365;
    end
end
```

THE
DEVELOPER'S
CONFERENCE

Extrair classe

THE DEVELOPER'S CONFERENCE

É a união do Extrair método. É utilizada quando uma classe tem mais de uma responsabilidade. Então, para a construção dessa nova classe, utilizamos o Mover Método e Mover Campo.

Extrair classe

THE DEVELOPER'S CONFERENCE

Vamos separar a parte de baixar o log de vendas da parte que salva esse log no banco de dados. Primeiro vamos criar uma nova classe para essa responsabilidade e passar os respectivos métodos para ela.

Extrair classe

```ruby
class BaixarLogVendaWorker
    attr_reader :host, :porta, :usuario, :senha
    def self.requisita_servidor(arquivo)
        Net::FTP.open(@host) do |ftp|
            ftp.login(@usuario,@senha)
            salvar_no_banco(ftp.gettextfile(arquivo)
        end
    end

    def salvar_no_banco(arquivo)
        LogVendas.ler_arquivo(arquivo)
    end
end
```

```ruby
class BaixarArquivoServidorFtp
    attr_reader :host, :porta, :usuario, :senha
    def self.requisita_servidor(arquivo)
        Net::FTP.open (@host) do |ftp|
            ftp.login(@usuario,@senha)
            ftp.gettextfile(arquivo)
        end
    end
end
```

THE DEVELOPER'S CONFERENCE

```
1   class BaixarLogVendaWorker
2       def self requisitar_servidor(arquivo)
3           arquivo_de_texto =BaixarArquivoServidorFtp.requisitar_servidor(arquivo)
4           salvar_no_banco(arquivo_de_texto)
5       end
6
7       def salvar_no_banco(arquivo)
8           LogVendas.ler_arquivo(arquivo)
9       end
10  end
```

THE DEVELOPER'S CONFERENCE

Com as responsabilidades separadas e com os métodos simplificados, podemos ver nomes mais claros para as nossas classes.

Extrair classe

THE DEVELOPER'S CONFERENCE

A classe BaixarArquivoServidorFtp é bem genérica para ser usada em outros locais – pode permanecer com o mesmo nome. Já a BaixarLogVendaWorker pode ser renomeada para SalvarRegistroVendasWorker.

Extrair classe

```ruby
class SalvarRegistroVendasWorker
    def salvar_no_banco()
        arquivo_de_texto = BaixarArquivoServidorFto.requisitar_servidor(arquivo)
        LogVendas.ler_arquivo (arquivo)
    end
end

class BaixarArquivoServidorFtp
    attr_reader :host :port :usuario :senha
    def self.requisitar_servidor(arquivo)
        Net:FTP.open(@host) do |ftp|
            ftp.login(usuarios: @usuario,senhas:@senha )
        end
    end
end
```

THE
DEVELOPER'S
CONFERENCE

Padrões de projeto

THE DEVELOPER'S CONFERENCE

Soluções/modelos generalistas para questões recorrentes no desenvolvimento de software

Padrões de projeto

THE
DEVELOPER'S
CONFERENCE

Factory

THE DEVELOPER'S CONFERENCE

Tem como objetivo extrair criação de objetos para métodos e classes específicos.
Os objetos criados são chamados de produtos e as classes que os criam são as fábricas.

Padrões de projeto

THE DEVELOPER'S CONFERENCE

Simple Factory

THE DEVELOPER'S CONFERENCE

Quando temos apenas um tipo de produto que precisa ser criado e só existe uma forma de fazer isso.
Neste caso, a solução é criar uma classe para extrair o comportamento da criação destes objetos.

Padrões de projeto

THE
DEVELOPER'S
CONFERENCE

Factory Method

THE DEVELOPER'S CONFERENCE

Ter um mesmo tipo de produto, mas com várias fábricas, cada uma com seu funcionamento próprio, Dessa forma, além das configurações passadas, cada uma das fábricas terá uma lógica específica que afeta o produto final.

Padrões de projeto

THE
DEVELOPER'S
CONFERENCE

Template Method

THE DEVELOPER'S CONFERENCE

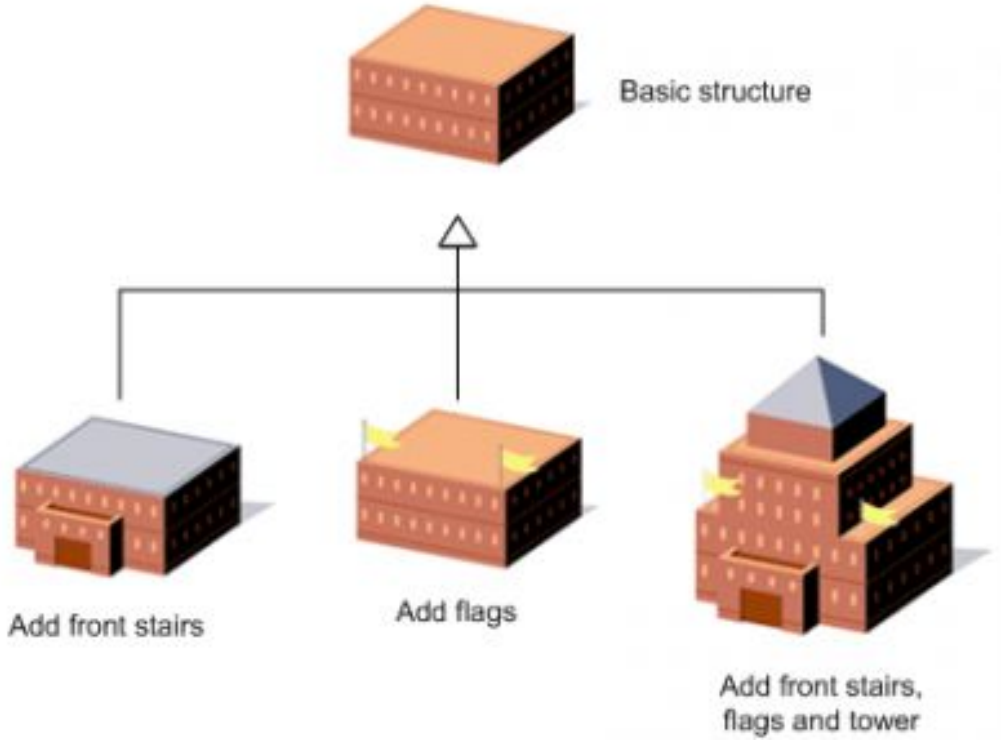
É um padrão comportamental, que tem como objetivo simplificar as responsabilidades dos objetos

Padrões de projeto

THE DEVELOPER'S CONFERENCE

Contexto: podemos separar um comportamento base , que é comum aos demais algoritmos do sistema, criando um template com pontos de extensão

Padrões de projeto


Basic structure
Add front stairs
Add flags
Add front stairs, flags and tower

A base define os métodos que serão chamados, a ordem de execução e o que será retornado por eles, tudo isso numa mesma classe que será utilizada pelas outras para executar esses algoritmos

THE DEVELOPER'S CONFERENCE

Padrões de projeto

Cada um dos demais algoritmos específicos herda essa classe base e sobrescreve os métodos necessários para implementar essa própria lógica.

THE DEVELOPER'S CONFERENCE

Padrões de projeto

Estes métodos que são sobrescritos são denominados métodos gancho, como a base permanece inalterada, o código de utilização permanece o mesmo em qualquer utilização, trazendo maior flexibilidade para a aplicação.

THE DEVELOPER'S CONFERENCE

Padrões de projeto

THE DEVELOPER'S CONFERENCE

Strategy

THE DEVELOPER'S CONFERENCE

Semelhante ao anterior, é um padrão de comportamento, visa resolver problemas de distribuição de responsabilidades. Devemos ter de forma clara como separar essas responsabilidades.

Padrões de projeto

Devemos encapsular cada um deles pelas estratégias de modo que possamos realizar a troca entre elas facilmente.

THE DEVELOPER'S CONFERENCE

Padrões de projeto

THE
DEVELOPER'S
CONFERENCE

Adapter

THE DEVELOPER'S CONFERENCE

É recomendável de ser utilizado quando temos duas classes com interfaces diferentes, porém que precisam trabalhar em conjunto.

Padrões de projeto

THE DEVELOPER'S CONFERENCE

Devemos isolar as responsabilidades de uma interface para outra do restante da lógica do nosso negócio.

Padrões de projeto

A utilização do Adapter faz define o que é esperado pela classe cliente e o que esse adaptador precisa definir. A criação de novos adaptadores não requer criação de novos clientes, mas precisamos que os dados sejam retornados da maneira esperada.

THE DEVELOPER'S CONFERENCE

Padrões de projeto

Referências:

THE DEVELOPER'S CONFERENCE

https://www.casadocodigo.com.br/products/livro-refatoracao-ruby

https://brizeno.wordpress.com/category/refatoracao/

https://www.amazon.com.br/Refatora%C3%A7%C3%A3o-Aperfei%C3%A7oando-Projeto-C%C3%B3digo-Existente-ebook/dp/B019IZK89A

https://refactoring.com/

https://www.devmedia.com.br/refatoracoes-em-ruby-move-method-e-move-field/37446

https://refactoring.com/catalog/

http://www.desenvolvimentoagil.com.br/xp/praticas/refatoracao

https://medium.com/@sawomirkowalski/design-patterns-template-method-45888a2b84bc

Referências:

THE DEVELOPER'S CONFERENCE

https://refactoring.guru/

http://blog.sciensa.com/tdd-test-driven-development-guia-rapido/

https://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264

https://medium.com/thiago-aragao/solid-princ%C3%ADpios-da-programa%C3%A7%C3%A3o-orientada-a-objetos-ba7e31d8fb25

https://www.devmedia.com.br/reutilizacao-de-codigo-com-base-no-dry/33323

https://www.amazon.com.br/Pragmatic-Programmer-Journeyman-Master/dp/020161622X

http://www.macoratti.net/16/04/net_dry1.htm

https://medium.com/@tbaragao/solid-s-r-p-single-responsibility-principle-2760ff4a7edc

Referências:

THE DEVELOPER'S CONFERENCE

https://medium.com/@tbaragao/solid-ocp-open-closed-principle-600be0382244

https://medium.com/@tbaragao/solid-l-s-p-liskov-substitution-principle-3a31c3a7b49e

https://medium.com/@tbaragao/solid-i-s-p-interface-segregation-principle-c0b25d7dccf9

https://medium.com/@tbaragao/solid-d-i-p-dependency-inversion-principle-e87527f8d0be

https://imasters.com.br/desenvolvimento/refatoracao-em-ruby

Obrigada <3

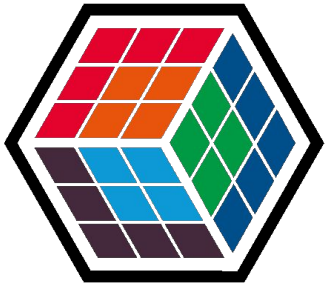
THE DEVELOPER'S CONFERENCE

@kamilah_santos

in/kamila-santos-oliveira/

@kamila_code

kamilahsantos



THE DEVELOPER'S CONFERENCE