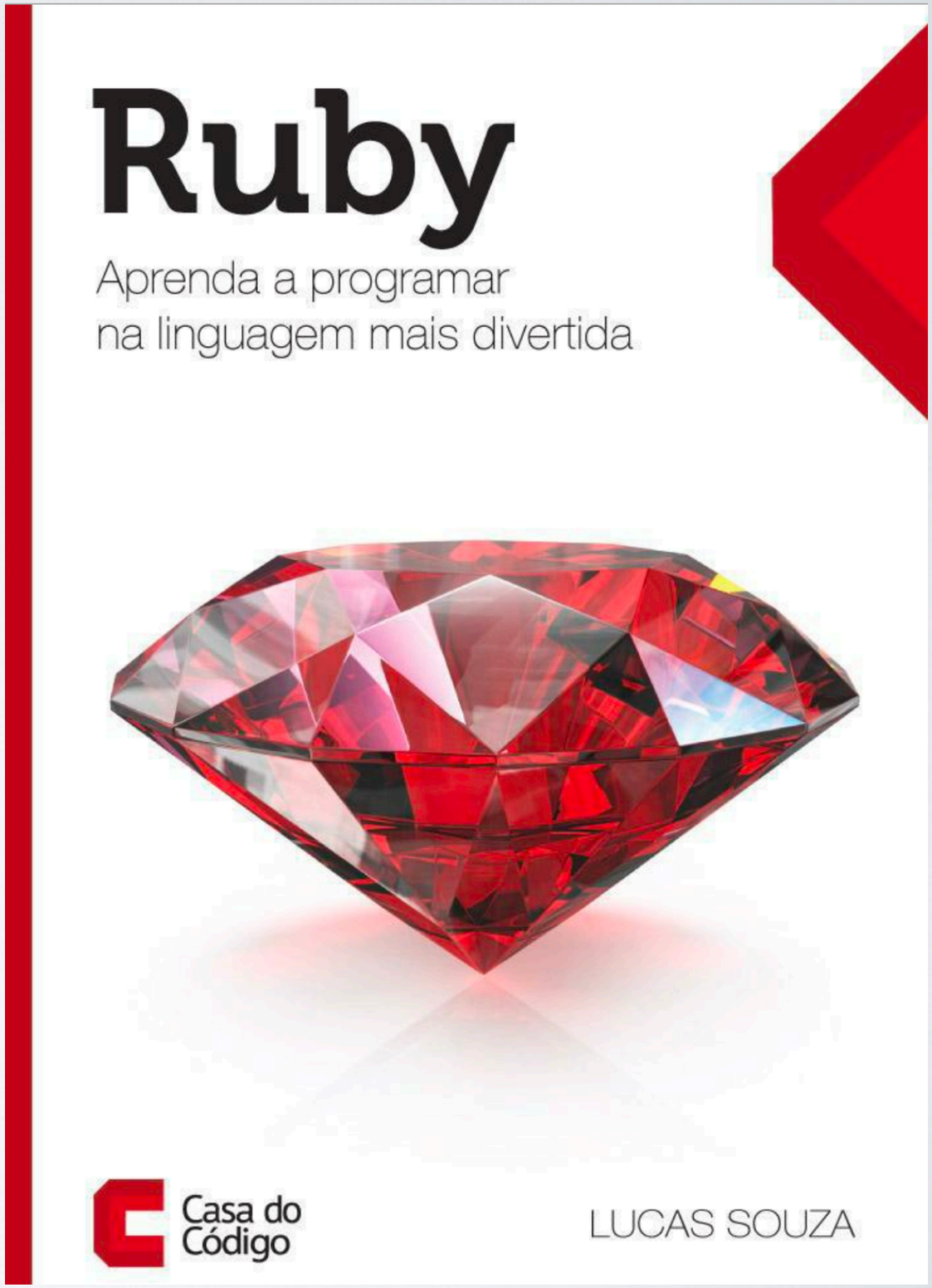
HERANÇA E MÓDULOS

Rafael Vieira Coelho

(rafaelvc2@gmail.com)



MÉTODO P

Nos capítulos anteriores utilizamos bastante o método `p` para imprimir dados na saída padrão. Aprendemos que o método `p` sempre invoca um método chamado `inspect` no objeto que pedimos para ser impresso. O fato é que nós nunca implementamos esse método em nenhuma das classes que criamos até agora.

Quando invocamos qualquer método em um objeto, na verdade estamos enviando uma mensagem para ele e solicitando que seja executado algum comportamento. A linguagem Ruby procura qual método deve ser executado e quando o encontra, executa o mesmo. Por exemplo, quando invocamos o método `inspect` em uma instância de `Livro`, o interpretador Ruby irá procurar o método na instância que representa e possui todos os métodos definidos na classe `Livro`, caso não encontre, o interpretador tenta encontrá-lo em alguma **super classe** da classe `Livro`:

```
1 p Livro.superclass # => Object
```

EXEMPLO DE HERANÇA

- **Dog** (subclasse) é filho de **Animal** (superclasse)
- Métodos, constantes e atributos são herdados.

```ruby
class Animal
  def say_hello
    'Meep!'
  end

  def eat
    'Yumm!'
  end
end

class Dog < Animal
  def say_hello
    'Woof!'
  end
end

spot = Dog.new
spot.say_hello # 'Woof!'
spot.eat       # 'Yumm!'
```

O método `methods` retorna todos os métodos disponíveis para os objetos que são criados a partir da classe ou de **subclasses**. Podemos por exemplo, listar os métodos da classe `Object` e assim concluirmos que ela disponibiliza o método `inspect` para os objetos do tipo `Livro`, já que `Livro` herda de `Object`.

```
19  p Biblioteca.superclass.methods
```

```
[:yaml_tag, :allocate, :superclass, :new, :<=>, :<=, :>=, :==, :===,
:included_modules, :include?, :name, :ancestors, :attr, :attr_reader,
:attr_writer, :attr_accessor, :instance_methods, :public_instance_methods,
:protected_instance_methods, :private_instance_methods, :constants,
:const_get, :const_set, :const_defined?, :class_variables,
:remove_class_variable, :class_variable_get, :class_variable_set,
:class_variable_defined?, :freeze, :inspect, :private_constant,
:public_constant, :const_missing, :deprecate_constant, :include,
:singleton_class?, :prepend, :module_exec, :module_eval, :class_eval,
:remove_method, :<, :>, :undef_method, :class_exec, :method_defined?,
:alias_method, :to_s, :private_class_method, :public_method_defined?,
:private_method_defined?, :protected_method_defined?, :public_class_method,
:autoload?, :instance_method, :public_instance_method, :define_method,
:autoload, :to_yaml, :instance_variable_defined?, :remove_instance_variable,
:instance_of?, :kind_of?, :is_a?, :tap, :instance_variable_get,
:instance_variable_set, :protected_methods, :instance_variables,
:private_methods, :method, :public_method, :public_send, :singleton_method,
:define_singleton_method, :extend, :to_enum, :enum_for, :=~, :!~, :eql?,
:respond_to?, :object_id, :send, :display, :class, :nil?, :hash, :dup,
:singleton_class, :clone, :then, :itself, :yield_self, :untaint, :taint,
:tainted?, :trust, :untrust, :untrusted?, :singleton_methods, :frozen?,
```

CLASSE MÃE

```ruby
class Midia

    attr_accessor :valor
    attr_reader:titulo

    def initialize(valor=0.0, titulo='')
        @valor = valor
        @titulo = titulo
    end

end

```

CLASSE FILHA: DVD

```ruby
require_relative 'midia'

class DVD < Midia

    def initialize(categoria, valor=0.0, titulo='')
        super(valor, titulo)
        @categoria = categoria
    end

    def to_s
        puts "Título: #{@titulo}"
        puts "Categoria: #{@categoria}"
        puts "Valor: #{@valor}"
    end

    def ==(other)
        self.class === other and
        other.categoria == @categoria and
        other.titulo == @titulo
    end

    alias eql? ==

    alias equal? ==

    def hash
        @categoria.hash ^ @titulo.hash # XOR
    end
end
```

CLASSE FILHA: LIVRO

```ruby
require_relative 'midia'

class Livro < Midia

    attr_accessor :numero_paginas,:categoria,:autor

    def initialize(autor, numero_paginas, categoria, valor=0.0, titulo='')
        super(valor, titulo)
        @autor = autor
        @numero_paginas = numero_paginas
        @categoria = categoria
    end

    def to_s
        puts "Título: #{@titulo}"
        puts "Autor: #{@autor}"
        puts "Páginas: #{@numero_paginas}"
        puts "Categoria: #{@categoria}"
        puts "Valor: #{@valor}"
    end
```

CRIANDO UM MÓDULO: SISTEMA.RB

- Podemos agrupar o código de todas as nossas classes em apenas um módulo para facilitar a manutenção do código.

```
  1  module Vendas
  2
  3      require 'yaml'
  4
  5      class BancoDeArquivos ···
 20      end
 21
 22      class Midia ···
 36      end
 37
 38
 39      class DVD < Midia ···
 65      end
 66
 67      class Livro < Midia ···
108      end
109
110      class DVD < Midia ···
136      end
137
138      class Biblioteca ···
166      end
167
168      class Relatorio ···
184      end
185
186
187  end
```

USANDO O MÓDULO VENDAS

```ruby
require_relative 'SISTEMA'

liv1 = Vendas::Livro.new("Stephen King", "123454", "Suspense", 198.89, "A Torre Negra")
liv2 = Vendas::Livro.new "André Agassi", "452565", "Biografia", 53.21, "Minha Vida"
liv3 = Vendas::Livro.new("Rafael Coelho", "1111", "Suspense", 12.90, "Sra. Fantini")
liv4 = Vendas::Livro.new "Stephen King", "1232", "Suspense", 135.99, "It"

biblioteca = Vendas::Biblioteca.new
biblioteca.adiciona(liv1, 247)
biblioteca.adiciona(liv2, 248)
biblioteca.adiciona(liv3, 249)
biblioteca.adiciona(liv4, 250)


relatorio = Vendas::Relatorio.new(biblioteca)
puts 'Títulos: ', relatorio.obtem_titulos
```