Linguagem Ruby: Testes com Minitest

PUC Minas
Poços de Caldas

Luiz Alberto Ferreira Gomes

Curso de Ciência da Computação

25 de setembro de 2020

Ruby

OO

- OO possibilita identificar "coisas"que serão tratadas pelo programa
- Classes são descrições dessas "coisas" e container de métodos
- Objetos são instâncias dessas classes
- Objetos contêm variáveis de instância (estado)

Variáveis de Instância

- Iniciam com: @
  - exemplo: @nome
- Não há necessidade de declará-las
- Disponível para todas as instâncias dos métodos da classe

Criação de Objetos (1)

- Classes são fábricas
  - new cria uma instância da classe e invoca o método initialize
  - O estado do objeto deve ser inicializado no método initialize (construtor)

Criação de Objetos (2)

Listing 1: classes.rb

```ruby
class Pessoa
  def initialize (nome, idade)
    @nome = nome
    @idade = idade
  end
  def get_info
    "Nome: #{@nome}, age: #{@idade}"
  end
end

pessoa1 = Pessoa.new("Jose", 14)
p pessoa1.instance_variables # [:@nome, :@idade]
puts pessoa1.get_info # => Nome: Jose, idade: 14
```

Acesso a Variáveis de Instância (1)

- Variáveis de instância são privadas
- Métodos são públicos por padrão
- Getters/setters para acessar variáveis de instância são necessários

Acesso a Variáveis de Instância (2)

Listing 2: instance_vars.rb

```ruby
class Pessoa
  def initialize (nome, idade)
    @nome = nome
    @idade = idade
  end
  def nome
    @nome
  end
  def nome= (novo_nome)
    @nome = novo_nome
  end
end
pessoa1 = Pessoa.new("Jose", 14)
puts pessoa1.nome  # Jose
pessoa1.nome = "Maria"
puts pessoa1.nome  # Maria
```

Acesso a Variáveis de Instância (3)

```
17 # puts pessoa1.idade # undefined method ‘idade’ for #<Pessoa:
```

- Muitas vezes as lógicas dos getters/setters são muito simples
- Existe uma maneira mais fácil de definir esses métodos em Ruby
  - attr_accessor - getter e setter
  - attr_reader - somente getter
  - attr_writer - somente setter

Acesso a Variáveis de Instância (4)

Listing 3: attr_accessor.rb

```ruby
class Pessoa
  attr_accessor :nome, :idade # getters and setters for nome and id
end

pessoa1 = Pessoa.new
p pessoa1.nome # => nil
pessoa1.nome = "Maria"
pessoa1.idade = 15
puts pessoa1.nome # => Mike
puts pessoa1.idade # => 15
pessoa1.idade = "quinze"
puts pessoa1.idade # => fifteen
```

- Dois problemas com o exemplo acima:
  - Pessoa se encontra em um estado não inicializado na criação

Acesso a Variáveis de Instância (5)

- Algumas vezes é necessário controlar, por exemplo, a idade atribuída

- Solução: use o construtor de forma mais inteligente utilizando o comando self

Listing 4: self.rb

```
1  class Pessoa
2    attr_reader :idade
3    attr_accessor :nome
4
5    def initialize (nome, idade) # CONSTRUCTOR
6      @nome = nome
7      self.idade = idade # call the idade= method
8      puts idade
9    end
```

Acesso a Variáveis de Instância (6)

```
10    def idade= (nova_idade)
11      @idade ||= 5
12      @idade = nova_idade unless nova_idade > 120
13    end
14  end
15
16  pessoa1 = Pessoa.new("Kim", 13) # => 13
17  puts "Minha idade e #{pessoa1.idade}" # => Minha idade e 13
18  pessoa1.idade = 130 # Tenta mudar a idade
19  puts pessoa1.idade # => 13 o setter ãno permite
```

Métodos e Variáveis de Classe (1)

- Use self para definir métodos de classe
- Variáveis de classe começam com @@

Listing 5: class_methods_and_variables.rb

```
1  class Usuario
2    attr_accessor :nome, :email
3    def initialize(name, email)
4      @name   = name
5      @email  = email
6      @@quantidade ||= 0; @@quantidade += 1
7    end
8
9    def self.conta_de_usuario
10      puts "Acessos: #{@@quantidade}"
11   end
```

Métodos e Variáveis de Classe (2)

```
12  end
13
14  usuario = Usuario.new("pedro", "pedro@hotmail.com")
15  usuario = Usuario.new("maria", "maria@uol.com.br")
16
17  Usuario.conta_de_usuario
```

Herança de Classes (1)

Listing 6: inheritance.rb

```
1  class Cao # implicitamente herda de Object
2    def to_s
3      "Cao"
4    end
5    def late
6      "late alto"
7    end
8  end
9  class CaoPequeno < Cao
10   def late # Override
11     "late baixo"
12   end
13 end
14 cao = Cao.new
15 cao_pequeno = CaoPequeno.new
```

Herança de Classes (2)

```
16 puts "#{cao}1 #{cao.late}"
17 puts "#{cao_pequeno}2 #{cao_pequeno.late}"
```

Visibilidade de Métodos em Ruby

1. Todos os atributos são privado por padrão e Todos os métodos são públicos por padrão.
2. private e protected podem ser utilizados para mudar a visibilidade padrão de métodos

Hora de Colocar as Mãos na Massa (1)

1. Elabore na linguagem Ruby os códigos para os seguintes requisitos:
   - 1.1 Escreva a classe **Sobremesa** com *getters* e *setters* para os atributos `nome` e `calorias`. O construtor dessa classe deverá receber como parâmetros `nome` e `calorias`.
   - 1.2 Defina as operações de instância `ehSaudavel`, que retorna `true` se e somente se a sobremesa tem menos de 200 calorias, e `ehDeliciosa`, que retorna `true` para todas as sobremesas.

Hora de Colocar as Mãos na Massa (2)

1.3 Crie a classe **GeleiaEmCompota** que herdará da classe **Sobremesa**. O seu construtor deverá aceitar um único argumento denominado `sabor`; a sua quantidade padrão de `calorias` é 5 e seu `nome` deverá ser precedido de "Geléia em Compota de ", por exemplo, "Geléia em Compota de Morango".

1.4 Inclua um *getter* and *setter* para o atributo `sabor`.

1.5 Modifique a operação `ehDeliciosa` para retornar **false** se o sabor é `alcaçuz` e true para todos os outros sabores. O comportamento dessa operação para sobremesas que não são geléias em compotas não devem ser alterados.

2. Refatore o jogo de adivinhação para utilizar os mecanismos da orientação a objetos.

Recapitulando

- Objetos são criados com new
- Utilize o attr_ para criar getters/setters
- Não se esqueça do self quando necessário
- Variáveis de classe são definidas com @@