Curso PT 2/2015

Ruby - Aula 2

Orientação a Objetos

O que é POO?

- Paradigma de programação cujas principais preocupações são o reuso de código, encapsulamento e modularidade
- Objetos são abstrações de entidades do mundo real
- Indicado para aplicações onde é possível definir tais objetos (domínio bem definido)
- Utilizado pelo Rails em seus componentes

Classes

O principal conceito que precisamos entender em OO é o conceito de classe. O que é uma classe?

- Uma classe é um "molde" ou uma "forma" de uma entidade, a partir da qual podem ser criados objetos
- Na definição de uma classe, são especificados métodos e atributos
- Métodos são semelhantes à funções, e descrevem comportamentos (ações) que aquela classe pode realizar
- Atributos são semelhantes à variáveis, e descrevem estados (caracteristicas) pertencentes a um objeto daquela classe

Classes

```ruby
class Pessoa

    def initialize(nome) #método construtor!!
        @nome = nome  # variáveis com
    end               # @ denotam atributos da classe

    def mostrar_nome
        puts "Olá, meu nome é #{@nome}"
    end
end
```

```ruby
require_relative 'pessoa' #inclui o arquivo pessoa.rb
                          # onde esta escrita a classe

joao = Pessoa.new("João") #cria uma nova pessoa,
                          # passando o nome João
                          # para o método initialize

teste.mostrar_nome        # chama o método mostrar_nome
```

- O método *initialize* é chamado de construtor, e sempre é chamado quando o objeto é instanciado (criado).
- Objetos são instanciados colocando o nome da classe seguido do método *new*
- Um objeto de uma classe pode fazer uso dos métodos e atributos daquela classe

Classes

Classes podem possuir 2 tipos de atributos, de instância ou de classe.

- Atributos de instância se referem a um objeto (instância) de uma classe, portanto cada objeto utiliza seus próprios atributos de instância.

- São usados para representar informações que variam entre as instâncias (número da conta, saldo, etc).

- Atributos de classe se referem à classe como um todo, portanto todas as instâncias (objetos) dessa classe utilizam o mesmo atributo de classe.

- São usados para representar informações constantes entre as instâncias (imposto sobre transação, etc)

Atributos

```ruby
class Conta

    @@imposto = 0.15 ## atributo de classe

    def initialize(numero)
        @numero = numero # atributo de instancia
        @saldo = 0       #atributo de instancia
    end

    def depositar(valor)
        @saldo += valor - (valor  * @@imposto)
    end
```

```ruby
require_relative 'conta'

conta1 = Conta.new(1234)  #cria uma nova conta,
                          # passando o número 1234
                          # para o método initialize
puts "Saldo: #{conta1.mostrar_saldo}"
conta1.depositar(100)   #mostra o saldo antes e depois
puts "Saldo: #{conta1.mostrar_saldo}"
puts "Imposto #{conta1.mostrar_imposto}"

conta2 = Conta.new(4567)
puts "Saldo: #{conta2.mostrar_saldo}"
puts "Imposto: #{conta2.mostrar_imposto}"
```

- Atributos de classe são precedidos de @@
- Atributos de instância são precedidos de @

```
→  PT  ruby main.rb
Saldo: 0
Saldo: 85.0
Imposto 0.15
Saldo: 0
Imposto: 0.15
→  PT
```

Herança

- Recurso da POO que possibilita reuso de código
- Permite que classes "herdem" comportamentos de outras classes, incorporando seus atributos e métodos
- Classes que herdam outras classes são chamadas de subclasses ou classes filhas
- Classes que são herdadas são chamadas de superclasses ou classes mãe

Herança

```ruby
require_relative 'conta'

class ContaPoupanca < Conta

    @@taxa_poupanca = 0.02

    def render
        @saldo += @saldo * @@taxa_poupanca
    end

end
```

```ruby
require_relative 'conta'
require_relative 'conta_poupanca'


conta_p = ContaPoupanca.new(1234)

puts "Saldo: #{conta_p.mostrar_saldo}"
conta_p.depositar(100)
puts "Saldo: #{conta_p.mostrar_saldo}"
conta_p.render
puts "Saldo: #{conta_p.mostrar_saldo}"
```

- Repare que não definimos o atributo saldo ou os métodos depositar e mostrar_saldo na classe ContaPoupanca
- Essas informações estão sendo herdadas da superclasse Conta

```
→  PT  ruby main.rb
Saldo: 0
Saldo: 85.0
Saldo: 86.7
```

Encapsulamento

- Podemos desejar que algumas informações de nossas classes fiquem "escondidas" em relação a outras classes, de forma que possamos escolher que informações desejamos expor

- Para suprir essa necessidade, a POO traz um recurso conhecido como encapsulamento, possibilitando modificarmos o acesso de certos atributos ou métodos

- Uma convenção adotada no mundo OO é a de escondermos os atributos e apenas acessá-los por meio de métodos

- Usamos a palavra reservada *public* para definirmos trechos de código que podem ser acessados por qualquer um, e *private* para trechos que serão acessados somente por objetos da própria classe

Acesso a Atributos

```ruby
require_relative 'conta'

conta = Conta.new(1324)

puts conta.saldo # ao tentarmos acessar o atributo saldo,
                 # recebemos um erro
```

```
→  PT  ruby main.rb
main.rb:5:in `<main>': undefined method `saldo'
ro=1324, @saldo=0> (NoMethodError)
```

```ruby
    def saldo
        return @saldo
    end

    def saldo=(novo_saldo)
        @saldo = novo_saldo
    end
```

```ruby
conta = Conta.new(1324)
puts conta.saldo
conta.saldo= 10
puts conta.saldo
```

```
→  PT  ruby main.rb
0
10
```

- Por padrão, o Ruby não aceita o acesso direto a atributos de um objeto
- Precisamos definir métodos para lermos e alterarmos o valor de um atributo
- Esses métodos são chamados de get (leitura) e set (alteração)
- Padrão do Ruby é o método get ter o próprio nome do atributo, e o método set o nome do atributo seguido de um =

Acesso a Atributos

- Para nos poupar do trabalho repetitivo de escrever esses métodos de acesso, o Ruby traz ferramentas que criam esse código caso desejarmos.

- Utilizamos o *attr_reader* quando queremos criar um método de leitura (get) para determinado atributo

- Utilizamos o *attr_writer* quando queremos criar um método de alteração (set) para determinado atributo

- O caso comum é queremos tanto leitura quanto alteração, e para isso usamos o *attr_accessor,* que cria tanto o get quanto o set

Acesso a Atributos

```ruby
class Conta

    attr_reader :imposto
    attr_reader :saldo
    attr_accessor :numero


    @@imposto = 0.15

    def initialize(numero)
        @numero = numero
        @saldo = 0
    end

    def depositar(valor)
        @saldo += valor - (valor  * @@imposto)
    end


end
```

- Definimos um *attr_reader* para o imposto e o saldo, porque não queremos permitir que um objeto externo altere esses dados

- Definimos um *attr_accessor* para o número da conta pois queremos que tanto a alteração quanto a leitura seja realizável por objetos externos

- Essa ferramenta traz o mesmo resultado do que quando definimos os métodos por conta própria, porém torna o código mais simples e legível

EXERCISES

Para cada um dos exercícios, criar as classes em arquivos separados. Não se esquecer do require_relative 'nome_do_arquivo' para incluir as classes onde quiserem utilizá-las.

1 - Crie uma classe Animal que possua receba em seu método *initialize* o nome e a raça e atribua esses valores à variáveis de instância.

2 - Crie uma classe Cachorro que herde de Animal e acrescente o método *latir* a essa classe. Esse métdo deve imprimir a string "Auau!" na tela.

3 - Adicione o acesso aos atributos nome e raça por meio do *attr_accessor*.

4 - DESAFIO (DE NOVO? SIM!)

DESAFIO

- Crie uma classe para representar uma conta bancária. A classe deverá receber um nome e um saldo inicial no seu *initialize*, e atribuir os valores à variáveis de instância. Esses 2 atributos poderão apenas ser lidos por objetos externos. A classe também deverá possuir um atributo *senha,* que será recebido como parâmetro em todas as suas operações. Esse atributo deve ser definido no *initialize* e não deve ser acessível externamente.

- A classe deverá possuir os seguintes métodos:
  - Depositar (senha, valor): adiciona o valor ao saldo caso a senha informada seja a correta.
  - Sacar(senha, valor): retira o valor do saldo caso a senha informada seja a correta e o valor não for maior do que o saldo atual.
  - mostrar_saldo(senha): imprime na tela o saldo da conta caso a senha informada seja a correta

  É necessário adicionar mensagens de feedback ao usuário em todas as operações.