# **Programação Orientada a Objetos II**

#### Assunto da aula

Nesta segunda parte da aula sobre Programação Orientada a Objetos será abordado como instanciar classes localizadas em outro arquivo de um projeto, os diferentes tipos de variáveis da linguagem Ruby, como inserir e recuperar atributos e como utilizar um método construtor.

#### Conteúdo

* Require
* Escopo de variáveis
* Atributos
* Construtores

#### Require

Na **Aula 5** deste curso, você aprendeu a utilizar require ‘gem\_name’ para carregar arquivos de uma gem e assim poder escrever códigos com funcionalidades desta biblioteca.

Com ele também é possível carregar arquivos ruby que foram escritos por você. Para exemplificar como isso acontece você criará um projeto onde alguns arquivos conterão apenas uma classe.

**Projeto**

1- Crie a pasta do projeto **Animal** executando

|  |  |
| --- | --- |
| 1 | mkdir animal |

2- Dentro da pasta animal crie um arquivo chamado **animal.rb** com o código:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | class Animal   def pular     puts 'Toing! tóim! bóim! póim!'   end     def dormir     puts 'ZzZzzz!'   end  end |

3- Agora crie um arquivo com o nome **app.rb**e adicione a ele o código:

|  |  |
| --- | --- |
| 1  2  3  4  5 | require './animal.rb'    animal = Animal.new    animal.pular |

Perceba que no **require**, você especificou o arquivo que será procurado partindo do caminho relativo a **app.rb**.

Quando você não está trabalhando com gems e quer carregar um arquivo a partir do caminho onde o código será executado, utilize o **require\_relative**.

4- Substitua o código de **app.rb**por

|  |  |
| --- | --- |
| 1  2  3  4 | require\_relative 'animal'    animal = Animal.new  animal.pular |

Se o arquivo **animal.rb**estivesse dentro de uma pasta chamada example, o caminho ficaria ‘example/animal’.Como só existe um arquivo chamado **animal**não é preciso especificar a sua extensão.

5- Agora, crie um arquivo chamado **cachorro.rb** com o código:

|  |  |
| --- | --- |
| 1  2  3  4  5 | class Cachorro < Animal   def latir     puts 'Au Au'   end  end |

A classe **Cachorro** recebe como herança a classe **Animal**.

Porque o **require\_relative** de **animal.rb**não está ai?

A resposta é simples:

Você irá inicializar a classe **Cachorro** dentro de **app.rb** e esse arquivo já faz um require\_relative de **animal.rb**!

6- Para incluir a inicialização e execução dos métodos da classe **Cachorro**substitua o arquivo de **app.rb** por:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | require\_relative 'animal'  require\_relative 'cachorro'    puts '--Animal--'  animal = Animal.new  animal.pular    puts '--Cachorro--'  cachorro = Cachorro.new  cachorro.pular  cachorro.latir |

7- Execute o programa e veja o resultado

|  |  |
| --- | --- |
| 1 | ruby app.rb |

#### Escopo das variáveis

Agora que você sabe dividir um programa em vários arquivos é interessante aprender sobre os diferentes tipos de escopos de variáveis presentes no Ruby. Com este conhecimento você terá a capacidade de escolher qual utilizar dependendo da situação.

As variáveis se dividem em 04 tipos:

* **Variável Local**
* **Variável Global**
* **Variável de Classe**
* **Variável de Instância**

A seguir, veja características de cada uma:

### Variável Local

É declarada com a primeira letra de seu nome sendo uma letra minúscula ou sublinhado.

Pode ser **acessada apenas onde foi criada**. Por exemplo, se você definir uma variável local dentro de de uma classe ela estará disponível apenas dentro desta classe, se a definiu dentro de um método conseguirá acessá-la apenas dentro deste método e assim por diante.

Exemplo:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | class Bar   def foo     # Pode ser definida como local ou \_local     local = 'local é acessada apenas dentro deste metodo'     print local   end  end    bar = Bar.new  bar.foo |

### Variável Global

Declarada com o prefixo $.

Pode ser **acessada em qualquer lugar do programa**.

Seu uso é**FORTEMENTE DESENCORAJADO** pois além de ser visível em qualquer lugar do código, também pode ser alterada em inúmeros locais ocasionando dificuldades no rastreamento de bugs.

Exemplo:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | class Bar   def foo     $global = 0     puts $global   end  end    class Baz   def qux     $global += 1     puts $global   end  end    bar = Bar.new  baz = Baz.new  bar.foo  baz.qux  baz.qux  puts $global |

### Variável de Classe

É declarada com o prefixo @@.

Pode ser acessada em qualquer lugar da classe onde foi declarada e

seu valor é **compartilhado** entre todas as**instâncias de sua classe**. Também Exemplo:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | class User   @@user\_count = 0   def add(name)     puts "User #{name} adicionado"     @@user\_count += 1     puts @@user\_count   end  end    first\_user = User.new  first\_user.add('João')    second\_user = User.new  second\_user.add('Mario') |

### Variável de Instância

Seu nome começa com o símbolo @.

Semelhante a variável de classe, tendo como única diferença o valor que **não é compartilhado** entre todas as**instâncias de sua classe**.

Exemplo:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | class User   def add(name)     @name = name     puts "User adicionado"     hello   end     def hello     puts "Seja bem vindo, #{@name}!"   end  end    user = User.new  user.add('João') |

#### Atributos

### O que são atributos?

Como você já sabe objetos possuem informações e comportamentos.

Na aula passada você viu a primeira parte deste conteúdo utilizando métodos para representar comportamentos. Agora é hora de  aprender o restante adicionando e recuperando **informações** de um objeto.

**Adicionando e Recuperando Informações**

1- Crie um arquivo chamado **atributos.rb**com o código

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | class Dog   def name     @name   end     def name= name     @name = name   end  End    dog = Dog.new    dog.name = 'Marlon'    puts dog.name |

O segundo recebe um valor e o atribui a variável @name.O primeiro método da classe **Dog**retorna o valor da variável de instância @name. Se a variável ainda não estiver definida, o resultado será nil.

Podemos dizer que o primeiro é para recuperar e o segundo para adicionar/alterar uma informação.

Declarar os métodos de um atributo pode ser vantajoso caso queira fazer algo além de definir o valor da variável de instância. De outra forma, existe uma maneira mais fácil de realizar esta operação.

2- Substitua o código de **atributos.rb** por

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | class Dog   attr\_accessor :name, :age  end      dog = Dog.new    dog.name = 'Marlon'  puts dog.name    dog.age = '1 ano'  puts dog.age |

O ruby disponibiliza um método chamado attr\_accessor que cria os métodos var e var= para todos atributos declarados.

#### Contrutores

Outra questão importante é que toda vez que a instância de uma classe é criada, o Ruby procura por um método chamado initialize. Você pode criar este método para especificar valores padrões durante a construção da classe.

1- Crie um arquivo chamado **construtor.rb**com o seguinte código

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | class Person   def initialize(name, age)     @name = name     @age = age   end     def check     puts "Instância da classe iniciada com os valores:"     puts "Name = #{@name}"     puts "Idade = #{@age}"   end  end    person = Person.new('João', 12)  person.check |

O número de parâmetros utilizados no método initialize é opcional.

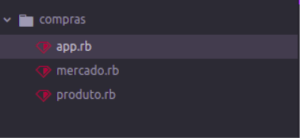
2- É possível criar a instância com valores padrões do objeto e executar o método check em apenas uma instrução.

Substitua as duas últimas linhas do programa por:

|  |  |
| --- | --- |
| 1 | Person.new('João', 12).check |

#### Missões especiais

Para exercitar o conhecimento adquirido nesta aula,  crie um projeto chamado Compras com a seguinte estrutura:

[](https://i1.wp.com/onebitcode.com/wp-content/uploads/2018/07/Captura-de-tela-de-2018-08-31-14-22-06.png?ssl=1)

Neste projeto você simulará o ato de escolher e comprar um produto em um mercado.

**Instruções do projeto:**

1- No arquivo **produto.rb**, crie uma classe chamada **Produto** com os atributos: **nome** e **preço**.

2- No arquivo **mercado.rb**crie uma classe chamada **Mercado** que ao ser inicializada recebe como atributo um objeto da classe Produto.

3 – Dentro da classe, crie um método chamado **comprar**que imprime a seguinte frase **“Você comprou o produto #{@produto.nome} no valor de #{@produto.preco}”**

4- No arquivo **app.rb**crie uma instância da classe **Produto** e adicione valores aos atributos **nome** e **preco**. Depois, inicie uma instância da classe **Mercado**passando um objeto produto como atributo e para finalizar execute o método comprar.