



Curso Completo

Aula 019

Brincando com Ruby

Parte 8/8

Módulos

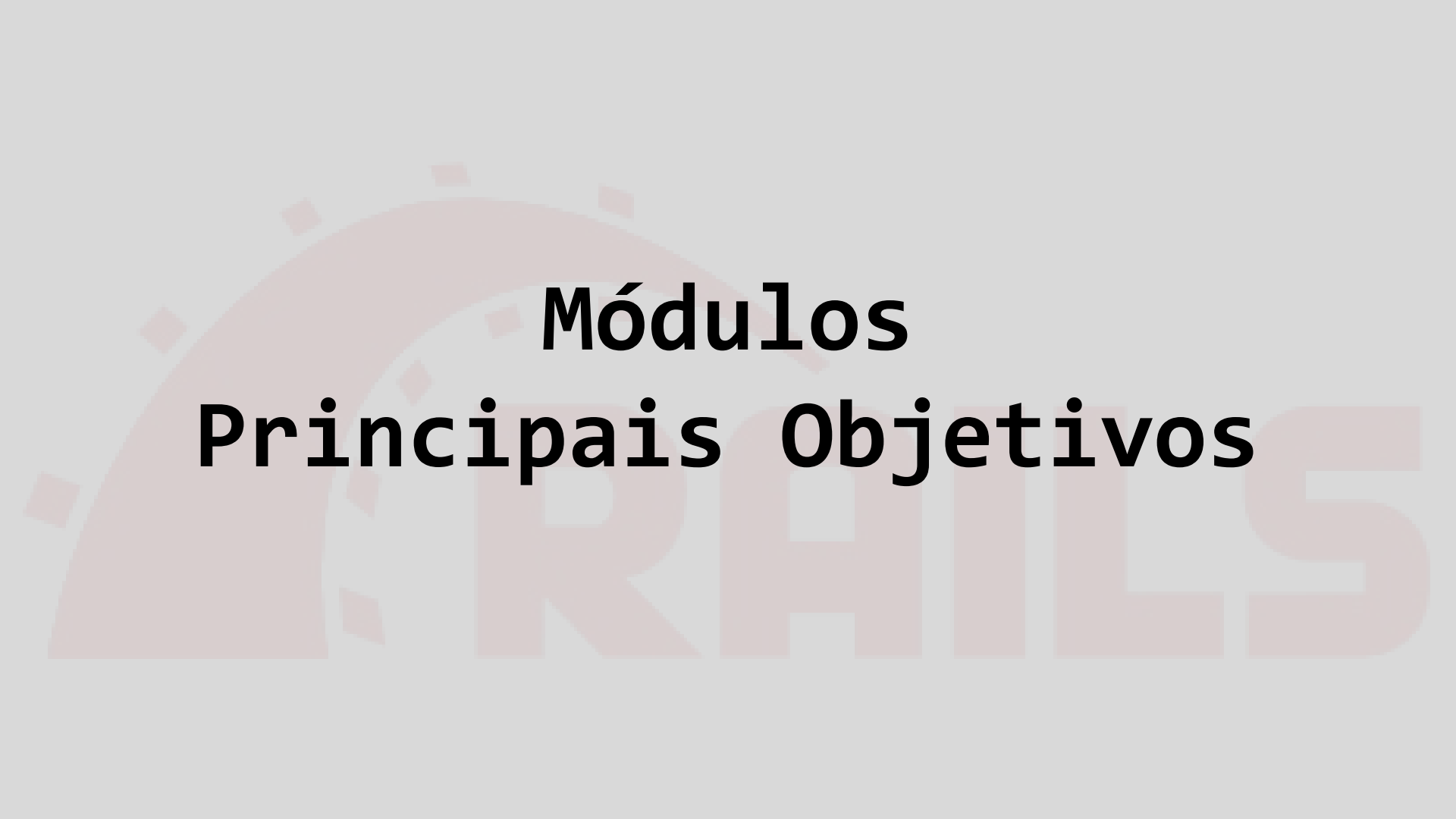
Módulos Ruby são similares a classes em relação ao fato de que também armazenam uma coleção de métodos, constantes e outras definições de módulos e classes.

Módulos

Entretanto, diferente das classes, você não pode criar objetos baseados em módulos nem pode criar módulos que herdam desse módulo; ao invés disso, você especifica qual funcionalidade de um módulo específico você deseja adicionar a uma classe ou a um objeto específico.

Módulos

Módulos permanecem sozinhos; não há hierarquia de módulos ou herança. Módulos são um bom lugar para armazenar constantes em um local centralizado.



Módulos

Principais Objetivos

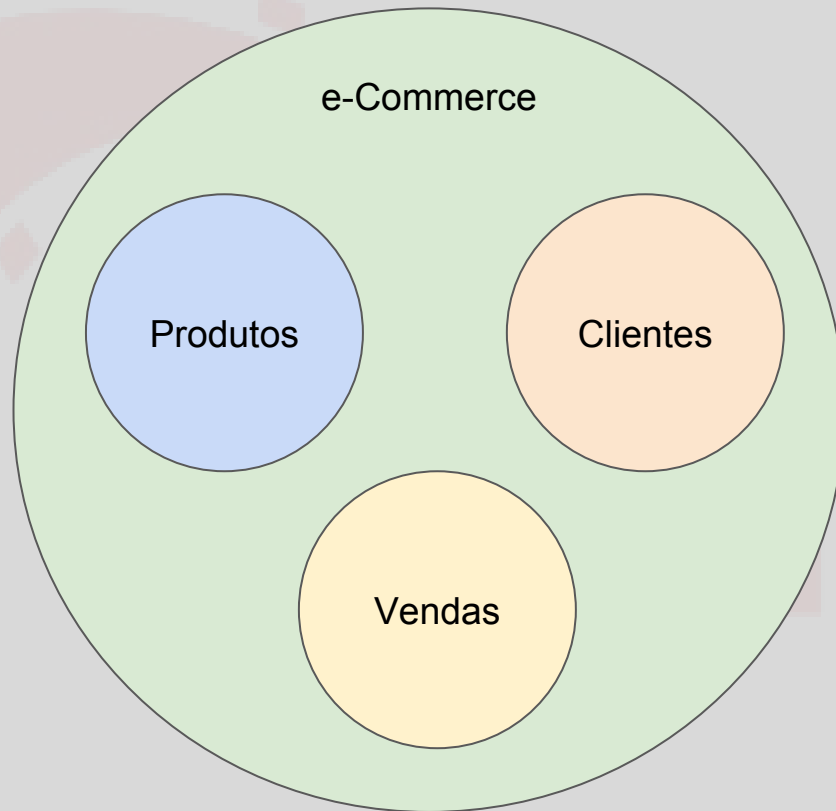
Módulos

Primeiro eles agem como **namespace**, permitindo que você defina métodos cujos nomes não irão colidir com aqueles definidos em outras partes de um programa.

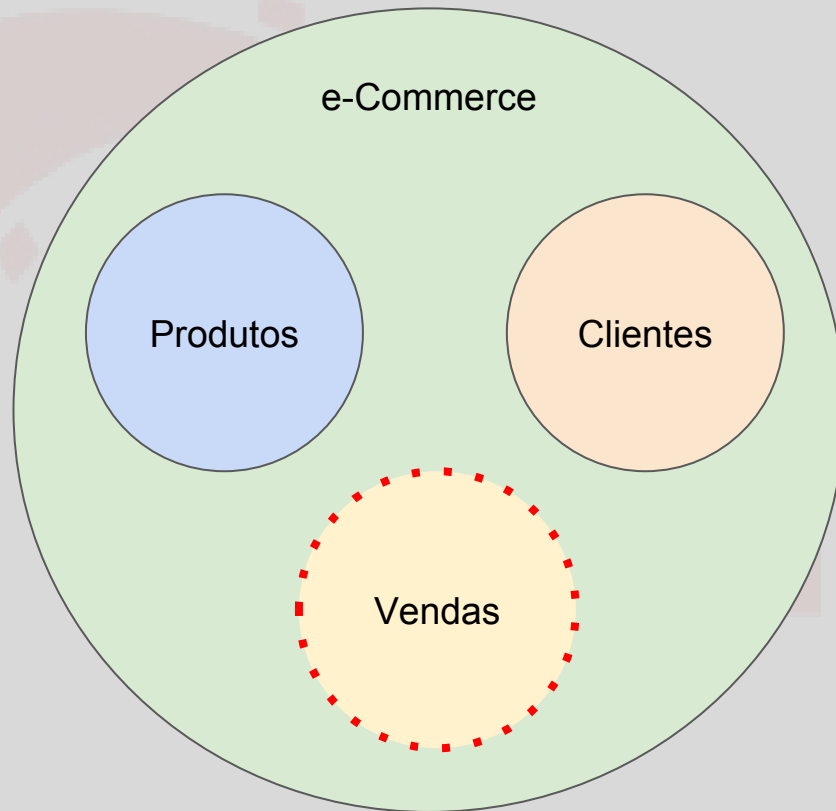
Módulos/Mixins

Em segundo lugar, permitem que você compartilhe funcionalidade entre classes – se uma classe “mistura” (mixes in) um módulo (isto é, o inclui), todos os métodos de instância do módulo se tornam disponíveis como se tivessem sido definidos na classe.

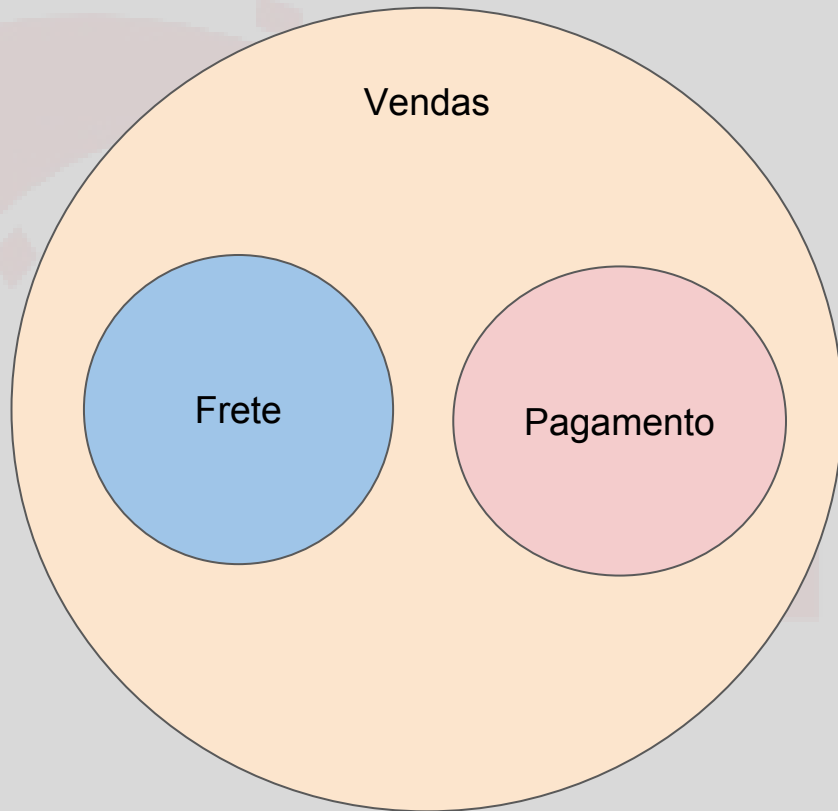
Módulos



Módulos



Módulos



Módulos

```
module Pagamento
  SIMBOLO_MOEDA = "R$"

  def pagar(valor_final)
    puts "Deseja pagar com Cartão? (S/N) "
    opcao = gets.chomp

    if opcao == "S"
      puts "Pagando com cartão..."
    else
      puts "Pagando com dinheiro..."
    end
  end
end
end
```

Módulos

```
module Frete
  TABELA_FRETE = {"BA" => 1.95, "SP" => 3.87, "PE" => 2.56}

  def imprimir_tabela_frete
    puts "--- Tabela de Frete ---"

    TABELA_FRETE.each do |k,v|
      puts "#{k} - #{v}"
    end

    puts "-----"
  end

  def calcular_valor_final(valor_produto, uf)
    valor_produto * TABELA_FRETE[uf]
  end
end
```

Módulos

```
require_relative "pagamento"  
require_relative "frete"
```

```
class Venda  
  include Pagamento  
  include Frete
```

```
  ...  
end
```

Módulos

```
PRODUTOS = {"PS3" => 900.00, "PS4" => 1600.00 }
```

```
def imprimir_produtos
```

```
  puts "--- Produtos ---"
```

```
  PRODUTOS.each do |k,v|
```

```
    puts "#{k} - #{SIMBOLO_MOEDA} #{v}"
```

```
  end
```

```
  puts "-----"
```

```
end
```

Módulos

```
def vender
  puts "Olá! Seja bem-vindo!"
  puts "O que deseja comprar?"

  imprimir_produtos

  puts "> Digite o nome do produto..."
  produto = gets.chomp

  ...
end
```


Módulos

```
def vender
  ...

  puts "Para onde deseja enviar?"

  imprimir_tabela_frete

  puts "> Digite o estado..."
  uf = gets.chomp

  ...
end
```

Módulos

```
def vender
  ...

  puts "Calculando..."
  frete = calcular_valor_final(PRODUTOS[produto], uf)

  puts "Você deve pagar #{SIMBOLO_MOEDA} #{valor_final} do
produto + frete."

  puts "Deseja pagar? (S/N) "
  opcao = gets.chomp

  if opcao == "S"
    pagar(valor_final)
  else
    puts "Ok! Fica para a próxima! :("
  end
end
```

Gran Finale!

```
module Pagamento
  ...
  class Pagseguro
    def initialize
      puts "Usando Pagseguro..."
    end
  end
end
```

Módulos

```
def vender
  ...

  puts "Calculando..."
  valor_final = calcular_valor_final(PRODUTOS[produto], uf)

  puts "Você deve pagar #{SIMBOLO_MOEDA} #{valor_final} do
produto + frete."

  puts "Deseja pagar? (S/N) "
  opcao = gets.chomp

  if opcao == "S"
    pagseguro = Pagseguro.new
    pagar(valor_final)
  else
    puts "Ok! Fica para a próxima! :("
  end
end
```



Learn to Program

<http://www.jmonteiro.com/aprendaaprogramar/>