# **Programação Orientada a Objetos I**

#### Assunto da aula

Como o tema deste episódio é extenso, o conteúdo será dividido em duas partes. Nesta primeira você aprenderá o que é Programação Orientada a Objetos e como utilizá-la na linguagem Ruby.

#### Conteúdo

* POO
* Objeto
* Classe
* Classes no Ruby

#### POO

POO (Programação Orientada a Objetos) é um  Paradigma de programação criado para lidar com softwares grandes e complexos. É um conceito seguido não só pelo Ruby, mas também por várias outras linguagens de programação como Java, Python, C++, etc.

Sua estrutura é definida por 04 pilares: **Abstração**, **Encapsulamento**, **Herança** e **Poliformismo**.

#### Pilares

A seguir, veja o conceito de cada pilar da Programação Orientada a Objetos:

**Abstração**

Ação de abstrair uma entidade do mundo real e transformá-la em uma classe

**Encapsulamento**

Ato de dividir um programa em diversas partes tornando-o flexível, fácil de modificar e incluir novas funcionalidades.

**Herança**

Habilidade de criar uma Classe com características de outra existente. A herança prove o reuso e reaproveitamento de código.

**Poliformismo**

Capacidade de utilizar um método de diferentes formas para diferentes Objetos.

#### Objeto

### O que é?

Por definição, objeto é toda coisa material que pode ser percebida pelos sentidos, como por exemplo um carro, livro, cachorro, caneta, avião e etc.

Na programação um objeto é a representação de um objeto do mundo real.

Todos estes objetos apresentam duas característica em comum: informações e comportamentos.

Ex:  Cachorro

**Informações**

* + **Raça**: Pastor Alemão
  + **Peso**: 35 Quilos
  + **Idade:**3 Anos

**Comportamento**

* **Late**
* **Come**
* **Corre**

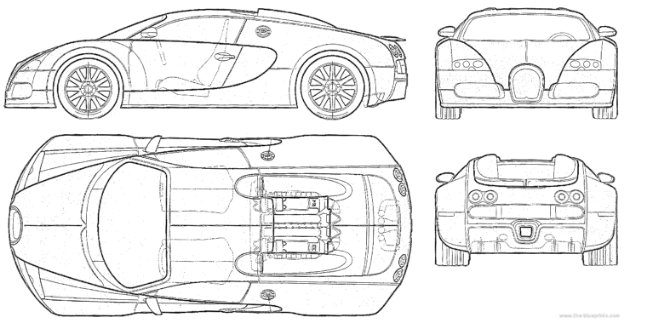
Em POO chamamos as informações de **attributes**e os comportamentos de **methods**.

#### Classe

### O que é?

Tendo como exemplo o objeto carro, pense em uma classe como a planta deste carro. Com ela é possível construir vários carros.

Uma classe é como a planta de um objeto.

[](https://i2.wp.com/onebitcode.com/wp-content/uploads/2018/07/img1-aula6.png?ssl=1)

### Classes no Ruby

Uma classe é definida pela palavra **class** seguida de seu **Nome,** e finalizada pela palavra **end.**

O nome de uma classe deve sempre começar com letra maiúscula. Para nomes compostos utilize o padrão **CamelCase**.

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | class Computer   def turn\_on     'turn on the computer'   end     def shutdown     'shutdown the computer'   end  end |

Você criou a classe do objeto **Computer** com os métodos **turn\_on** e **shutdown**.

2- Inicialize um novo objeto **Computer** e depois execute seu comportamento shutdown adicionando o seguinte código ao final do arquivo **computer.rb**.

|  |  |
| --- | --- |
| 1  2 | computer = Computer.new  computer.shutdown |

Neste programa você utilizou dois pilares da programação orientada a objetos:

**Abstração** -> representando o objeto Computer em uma classe.

**Encapsulamento**-> dividindo o projeto em pequenas partes.

3- Execute o programa

|  |  |
| --- | --- |
| 1 | ruby computer.rb |

4- Faça o teste novamente trocando a instrução **computer.shutdown** por **computer.turn\_on.**

#### Herança

Para herdar características de outra classe, adicione na frente do nome de uma classe filha o símbolo de menor e  depois o nome da classe pai.

Ex:

|  |  |
| --- | --- |
| 1  2 | class ClasseFilha < ClassePai  end |

1- Crie um programa chamado animal.rb com o seguinte código

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | class Animal   def pular     puts 'Toing! tóim! bóim! póim!'   end     def dormir     puts 'ZzZzzz!'   end  end    class Cachorro < Animal   def latir     puts 'Au Au'   end  end    cachorro = Cachorro.new  cachorro.pular  cachorro.dormir  cachorro.latir |

Cachorro é um objeto que possui todos os comportamentos existentes na classe animal (pular e respirar), então ele herda esta classe.

Sendo assim, é possível executar os métodos pular e respirar através de um objeto Cachorro.

#### Poliformismo – Exemplo modificado

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | class Instrumento    def escrever      puts 'Escrevendo'    end  end    class Lapis < Instrumento    def escrever      puts 'Escrevendo à Lápis'    end  end    class Caneta < Instrumento    def escrever      puts 'Escrevendo à Caneta'    end  end      instrumentos = [Lapis.new, Caneta.new]  # Chamamos o método escrever pra qualquer instrumento  Instrumentos.each do |instrumento|    Instrumento.escrever  end |

O comportamento do método escrever depende do Objeto que o invoca.

#### Missões especiais

Agora que você sabe o que é Programação Orientada a Objetos, crie um programa seguindo este paradigma com:

### Classes

* **Esportista**
  + Métodos
    - **competir** -> Imprime a mensagem “Participando de uma competição”

* **JogadorDeFutebol**
  + Métodos
    - **correr** -> Imprime a mensagem “Correndo atrás da bola”

* **Maratonista**
  + Métodos
    - **correr-**> Imprime a mensagem “Percorrendo o circuito”

As classes **JogadorDeFutebol** e **Maratonista** devem herdar os comportamentos da classe **Esportista**.

No final do programa execute os métodos **competir** e **correr** em objetos do tipo **JogadorDeFutebol** e **Maratonista.**