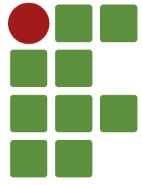
Introdução a Orientação a Objetos

Programação Estruturada e Orientada a Objetos



INSTITUTO
FEDERAL
Rio Grande do Norte

Campus
Ceará-Mirim

Prof. Me. Leo Moreira Silva

2 de março de 2020

Agenda

Introdução

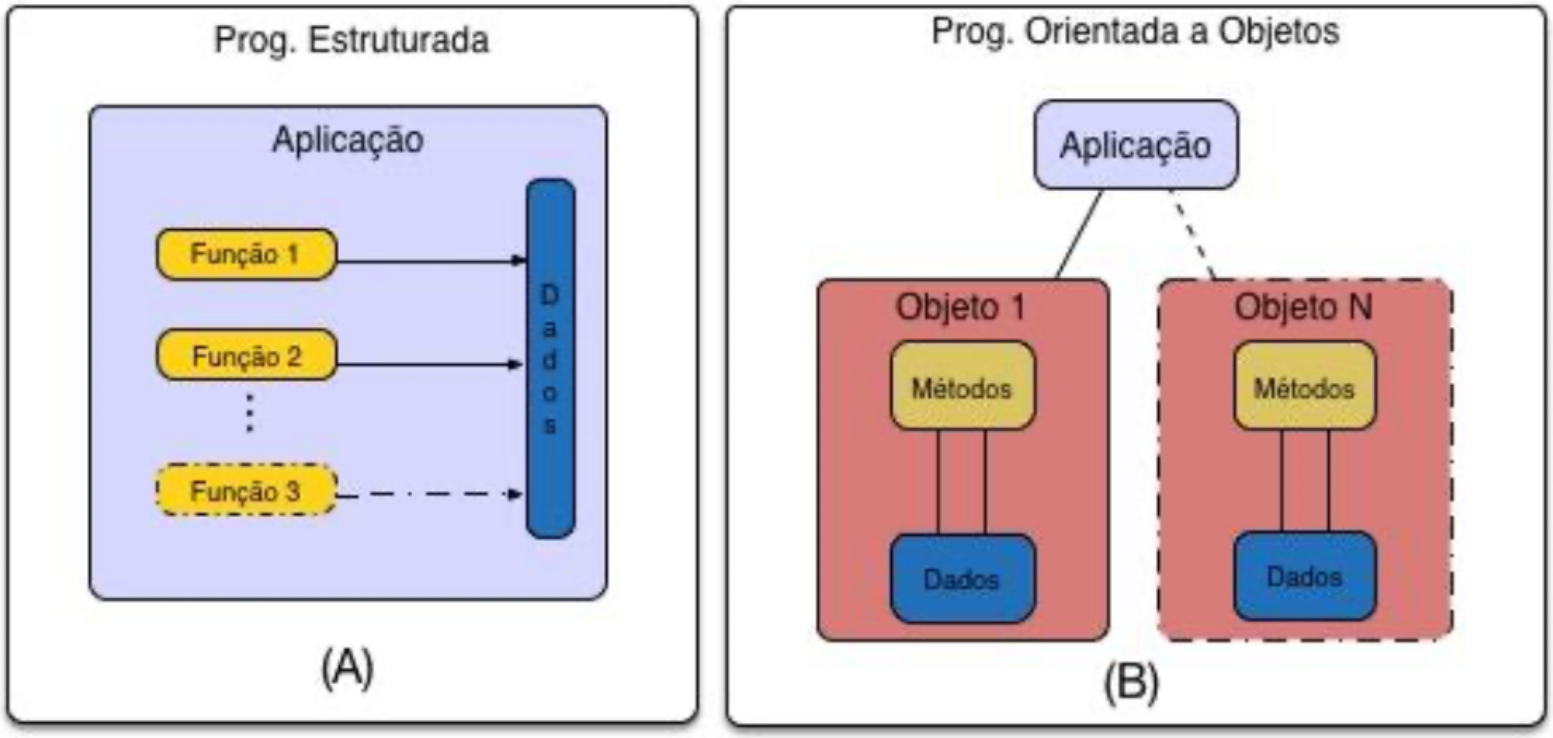
Paradigma da Orientação a Objetos

Fundamentos da Orientação a Objetos

Conceitos da Orientação a Objetos

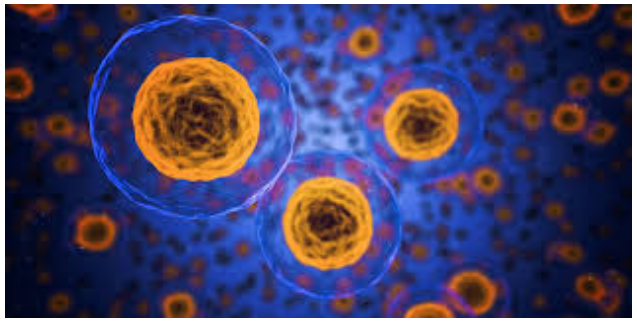
Exercícios

- Programas eram lineares e com poucos módulos (Programação Estruturada)
- Aumento da complexidade dos sistemas e difícil **reusabilidade** de componentes;
- Criou-se um novo paradigma para análise e desenvolvimento de sistemas: a **Programação Orientada a Objetos**.


Prog. Estruturada
Aplicação
Função 1
Função 2
Função 3
Dados
(A)
Prog. Orientada a Objetos
Aplicação
Objeto 1
Métodos
Dados
Objeto N
Métodos
Dados
(B)

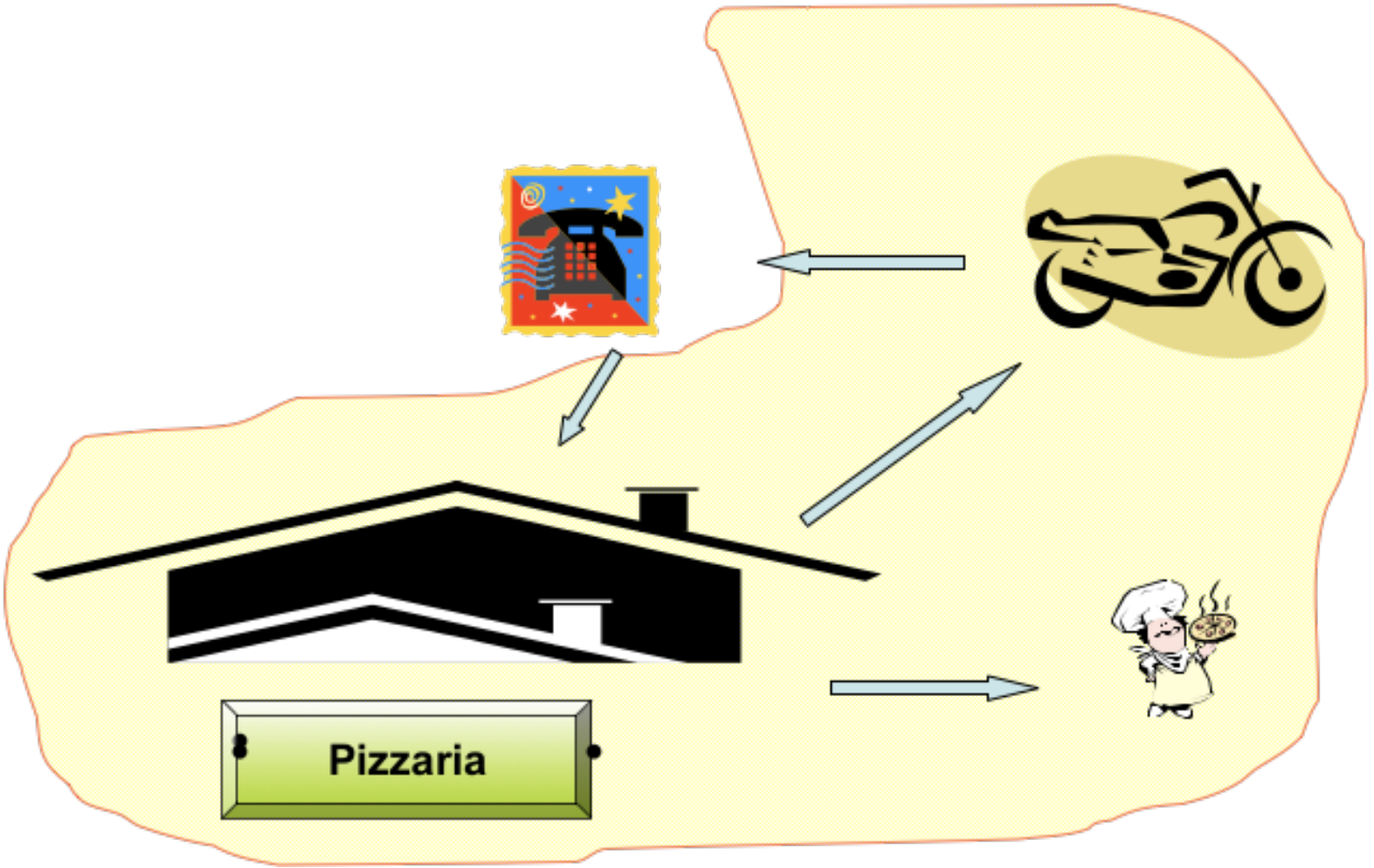
- O paradigma OO surgiu no fim dos anos 60
- Alan Kay, um dos pais desse paradigma, formulou a chamada analogia biológica:
  - ”Como seria um sistema de software que funcionasse como um ser vivo?”

- Cada célula interagiria com outras células através do envio de mensagens para realizar um objetivo comum
- Adicionalmente, cada célula se comportaria como uma unidade autônoma
- De uma forma mais geral, Alan Kay pensou em como construir um sistema de software a partir de **agentes autônomos** que **interagem entre si**.



- Através de sua analogia biológica, Alan Kay definiu os fundamentos da orientação a objetos:
  1. Qualquer coisa é um objeto
  2. Objetos realizam tarefas através da requisição de serviços a outros objetos
  3. Cada objeto pertence a uma determinada classe. Uma classe agrupa objetos similares
  4. A classe é um repositório para comportamento associado ao objeto
  5. Classes são organizadas em hierarquias

SSOO: uma analogia



- Conceitos:
  - Objeto
  - Classe
  - Instância

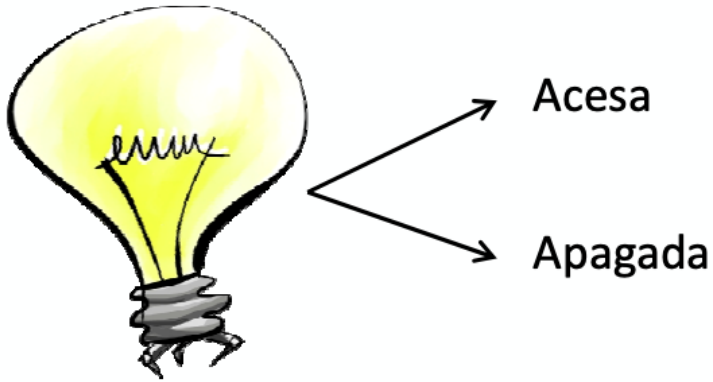
Objeto

- Entidade concretas ou abstratas
- Tem características e podem executar ações;
- ”Um objeto representa um item identificável, uma unidade ou entidade, individual, seja real ou abstrato, com uma regra bem definida”:

Objeto = dados + operações

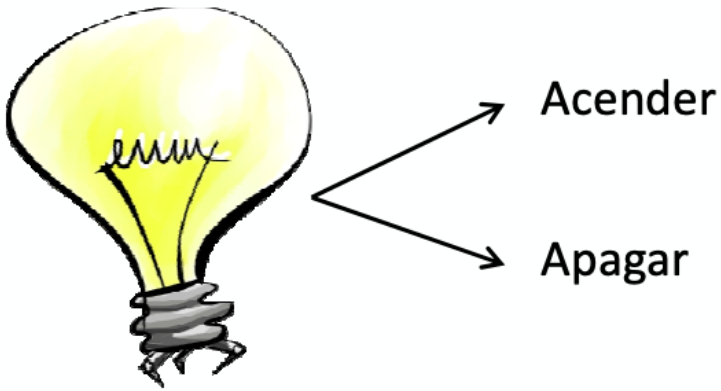
- Possuem:
  - Estado
  - Comportamento
  - Identidade

- Estado:
  - Define os estados possíveis que um objeto pode assumir
  - São os valores dos **atributos** (propriedades)
- Exemplo:



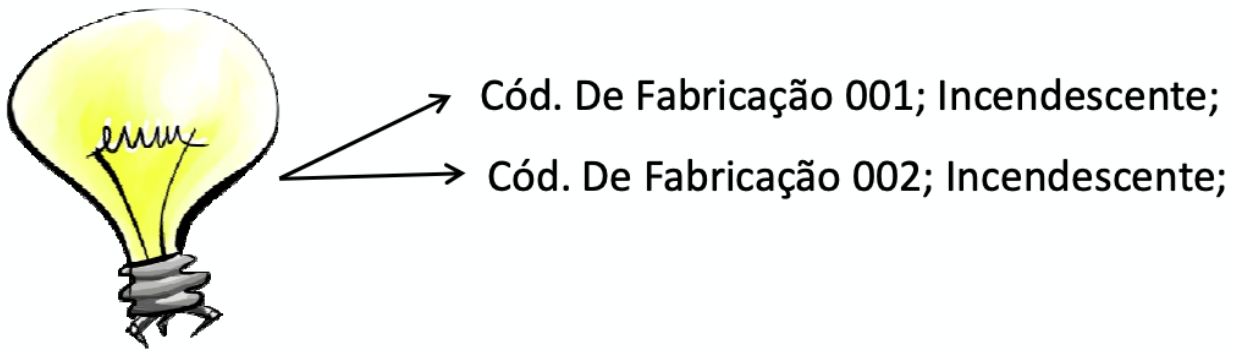
Objeto

- Comportamento:
  - São as funções que podem ser executadas por um determinado objeto
  - Corresponde aos **métodos**
  - O que se pode fazer com o objeto
- Exemplo:



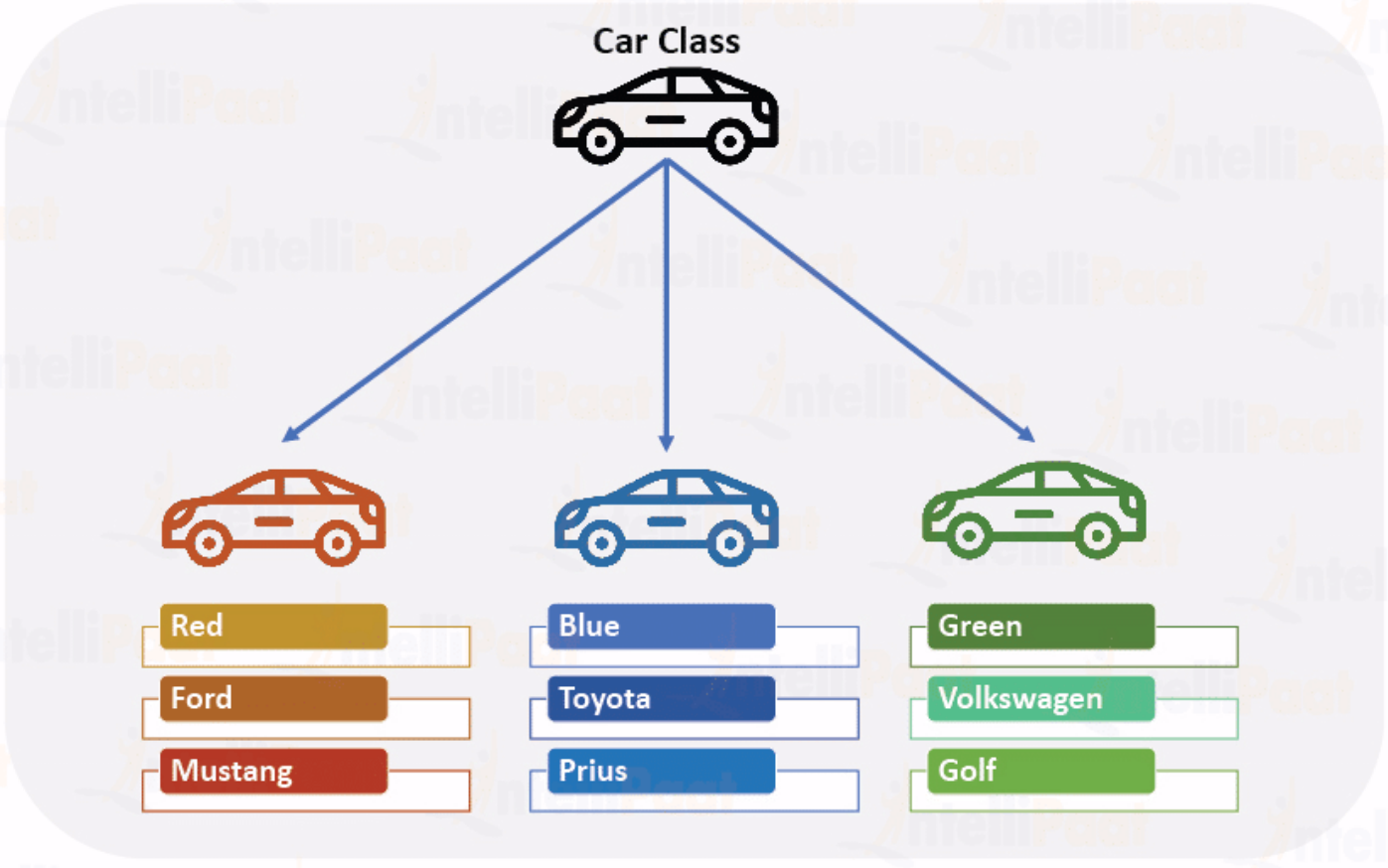
Objeto

- Identidade:
  - Um objeto é único, mesmo que o seu estado seja idêntico ao de outro.
- Exemplo:

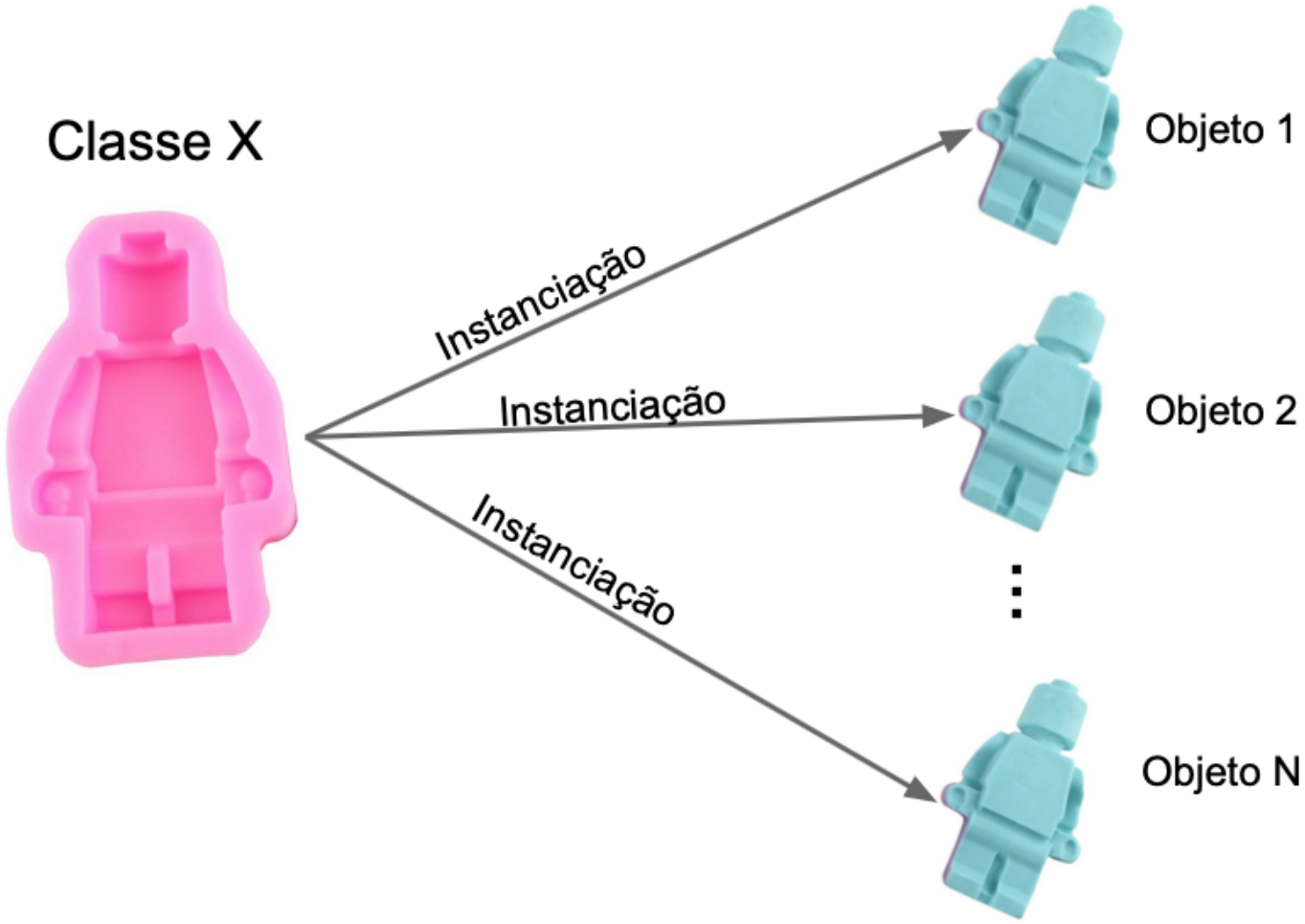


- Modelo ou esquema a partir do qual os objetos são criados (instanciados)
- Modelam os objetos, definindo:
  - Tipos de dados que o objeto armazena, ou seja, os estados possíveis que ele pode assumir (**atributos**)
  - Tipos de operações que podem ser executadas pelo objeto, ou seja, o seu comportamento (**métodos**)
- Abstração de objetos de características semelhantes
- É a **essência** do objeto

- Objetos são instâncias de classes:



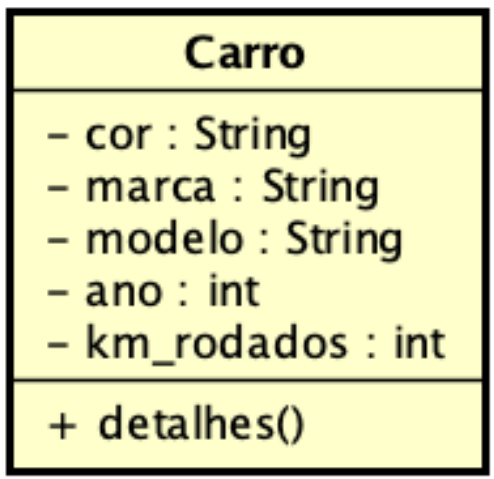
- Objetos são instâncias de classes:



Importante

Todo código Orientado a Objeto está dentro de uma classe!

- Classe:


Carro
- cor : String
- marca : String
- modelo : String
- ano : int
- km_rodados : int
+ detalhes()

- Definindo classe em Python:

```
class NomeClasse:
    atributo_1 = valor_1
    atributo_2 = valor_2
    atributo_3 = valor_3
def metodo_1(self):
    #faz algo aqui
```

- Uma nova instância pode ser criada a partir da chamada:

```
variavel = NomeClasse()
```

- 'variavel' irá armazenar a instância criada

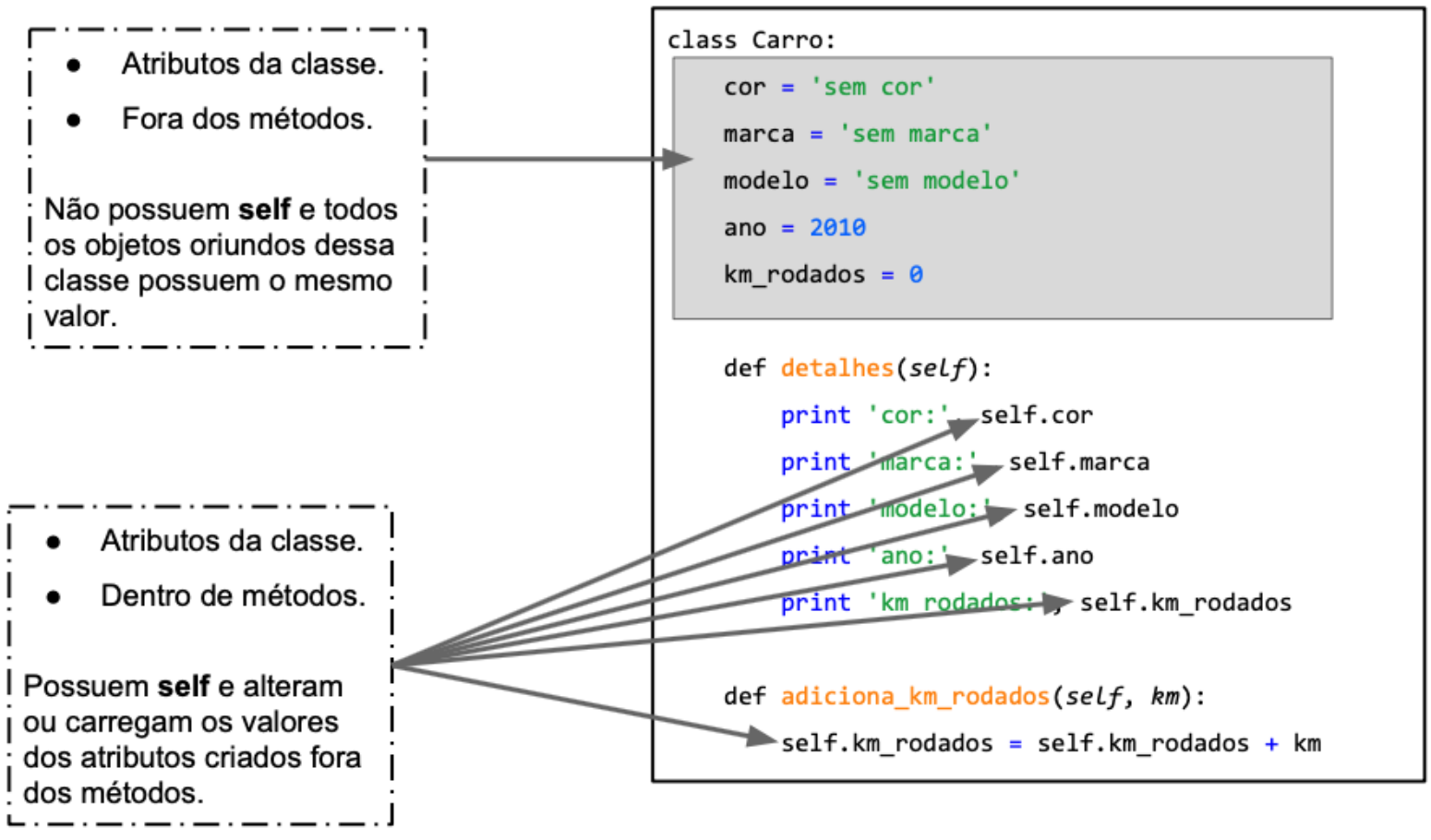
- `self` é uma variável que referencia um determinado objeto da classe
- Todo método de uma classe recebe `self` como primeiro parâmetro
- Tal parâmetro indica qual objeto está executando aquele método
- `self` deve preceder um atributo da classe dentro de métodos

```
class Carro:
    cor = 'sem cor'
    marca = 'sem marca'
    modelo = 'sem modelo'
    ano = 2010
    km_rodados = 0

    def detalhes(self):
        print 'cor:', self.cor
        print 'marca:', self.marca
        print 'modelo:', self.modelo
        print 'ano:', self.ano
        print 'km rodados:', self.km_rodados
```

```
>>>car_1 = Carro() #Instancia o objeto da classe Carro na variável 'car_1'
>>>car_1.cor = 'Vermelho'
>>>car_1.marca = 'Honda'
>>>car_1.modelo = 'HR-V'
>>>car_1.ano = 2016
>>>car_1.detalhes() #Chama o método 'detalhes' implementado na classe Carro

cor: Vermelho
marca: Honda
modelo: HR-V
ano: 2016
km_rodados: 0
```

Métodos recebem **self** como primeiro parâmetro.

```
class Carro:
    cor = 'sem cor'
    marca = 'sem marca'
    modelo = 'sem modelo'
    ano = 2010
    km_rodados = 0

    def detalhes(self):
        print 'cor:', self.cor
        print 'marca:', self.marca
        print 'modelo:', self.modelo
        print 'ano:', self.ano
        print 'km rodados:', self.km_rodados

    def adiciona_km_rodados(self, km):
        self.km_rodados = self.km_rodados + km
```


Atributos da classe.
Fora dos métodos.
Não possuem self e todos os objetos oriundos dessa classe possuem o mesmo valor.
Atributos da classe.
Dentro de métodos.
Possuem self e alteram ou carregam os valores dos atributos criados fora dos métodos.
class Carro:
cor = 'sem cor'
marca = 'sem marca'
modelo = 'sem modelo'
ano = 2010
km_rodados = 0
def detalhes(self):
print 'cor:', self.cor
print 'marca:', self.marca
print 'modelo:', self.modelo
print 'ano:', self.ano
print 'km rodados:', self.km_rodados
def adiciona_km_rodados(self, km):
self.km_rodados = self.km_rodados + km

- ➔ Variável local do método **detalhes**;
- ➔ Não possui **self.**;
- ➔ Só existe durante a execução do método **detalhes**;

```
class Carro:
    cor = 'sem cor'
    marca = 'sem marca'
    modelo = 'sem modelo'
    ano = 2010
    km_rodados = 0

    def detalhes(self):
        print 'cor:', self.cor
        print 'marca:', self.marca
        print 'modelo:', self.modelo
        print 'ano:', self.ano
        print 'km rodados:', self.km_rodados
        passageiro = True

    def adiciona_km_rodados(self, km):
        self.km_rodados = self.km_rodados + km
```

Dúvidas?

Exercício 1

1. Copiar o código da classe Carro
2. Criar um novo objeto, com valores diferentes do slide
3. Imprimir os dados
4. Criar e implementar um novo método chamado `diminuir_km_rodados()`
5. Testar o método criado

Exercício 1 (Cont.)

1. Implementar os métodos abaixo
   1.1 ligarMotor
   1.2 desligarMotor
2. Criar atributo para:
   2.1 estado do motor (ligado/desligado)

Exercício 2

1. Implementar e testar a seguinte classe: