

ORIENTAÇÃO A OBJETOS

A palavra **classe** vem da biologia. Todos os seres vivos de uma mesma classe biológica possuem **atributos** e **comportamentos** em comum, mas não são iguais, podem variar nos valores desses atributos e como realizam esses comportamentos.

Paradigma de Orientação a Objetos

Classe: Modelo que define a forma de um objeto. É um conjunto de planos que especifica como criar um objeto.

O molde é sempre o mesmo, porém os objetos por ele gerados podem ter características variadas, respeitando a estrutura do molde, ou seja da classe, que possui:

Atributos: Características, propriedades

Métodos: Comportamentos, ações

Estado: Características e seus valores

Objetos: São instâncias de uma classe.

	Classe	Classe	Classe
	Funcionário	Jogador	Celular
Atributos	nome idade setor salario rg cpf	nome idade time posição qtd_titulos país	marca número_linha operadora imei cor modelo
Métodos	trabalhar() receberSalario() receberAumento()	cobrarFalta() chutar() cometerFalta() correr()	tocar() enviarMsg() conectarInternet() fazerLigação()

Objeto
(Instância)

**f1 = novo
Funcionario();**

**jogador1 = novo
Jogador();**

**cel1 = novo
Celular();**

Estado

f1.nome = "Lívia";
f1.idade = 28;
f1.setor = "Financeiro";
f1.salario = 2500.00;
f1.rg = 2233344455;
f1.cpf = 44342255443;

jogador1.nome = "Cristiano";
jogador1.idade = 34;
jogador1.time = "Juventus";
jogador1.posição = "Atacante";
jogador1.qtd_títulos = 5;
jogador1.pais = "Portugal";

cel1.marca = "Samsung";
cel1.num_linha = 00000;
cel1.operadora = "Vivo";
cel1.imei = 04515262133;
cel1.cor = "Preto";
cel1.modelo = Galaxy;

Paradigma de Orientação a Objetos - Abstração

Trabalhar um objeto dentro da programação se preocupando somente com as principais propriedades, os aspectos relevantes do problema em questão.

Observar comportamentos e estruturas do dia a dia, e transformá-los em uma linguagem computacional.

Paradigma de Orientação a Objetos - Classe



Cachorro
nome: Caractere raça: Caractere cor: Caractere idade: Inteiro peso: Real vacinado: Logico
latir(): void comer(): void vacinar(): void



Nome da
Classe



Atributos e
seus tipos



Métodos

Paradigma de Orientação a Objetos - Definindo um Objeto



Cachorro
nome: Caractere raça: Caractere cor: Caractere idade: Inteiro peso: Real vacinado: Logico
latir(): void comer(): void vacinar(): void

Aqui temos dois objetos da classe Cachorro

**c1 = novo
Cachorro()**

c1.nome: Bidu
c1.raça: Golden Retriever
c1.cor: amarelo
c1.idade: 4
c1.peso: 9.300
c1.vacinado: true

latir()
comer()
vacinar()

**c2 = novo
Cachorro()**

c2.nome: Mozart
c2.raça: São Bernardo
c2.cor: branco e amarelo
c2.idade: 6
c2.peso: 34.500
c2.vacinado: false

latir()
comer()
vacinar()

Paradigma Orientado a Objetos

- Se outros programadores usam nossa classe, é preciso garantir que erros não ocorram por mau uso.
- Controlar o acesso aos atributos e métodos de uma classe é uma forma eficiente de proteger os dados manipulados dentro da classe, além de determinar onde esta classe poderá ser manipulada.
- Encapsulamento consiste em separar o programa em partes, o mais isolado possível. A idéia é tornar o software mais flexível, e fácil de modificar.

Paradigma Orientado a Objetos

- **Encapsulamento:** Mecanismo de programação que vincula o código e os dados que ele trata, mantendo os dois seguros de interferências externas.
- **Modificadores de Acesso:** Determinam a visibilidade de acesso a classes, métodos e atributos, que podem ser públicos, privados ou protegidos.
- **Métodos Acessores:** Regulam o acesso a dados internos. Geralmente são chamados de métodos Get (recupera um valor) e Set (insere um valor)

Modificadores de Acesso

+	Public (público)	Tudo que é declarado como público é acessível por qualquer classe
-	Private (privado)	Tudo que é declarado como <i>private</i> é acessível somente pela classe que os declara
#	Protected (protegido)	Tudo que é declarado como protected é acessíveis pela classe que os declara e suas subclasses (Herança)



Funcionário

- + nome: caractere
 - + idade: inteiro
 - + setor: caractere
 - salario: real
 - rg: caractere
 - cpf: caractere
 - efetivo: logico
-
- + mostrarFuncionario(): void
 - + demitirFuncionario(d: logico): void
 - + aumentarSalario(sal: real): void



Métodos Get e Set

- O encapsulamento "protege" atributos e métodos dentro de uma classe. Para acessar atributos e métodos privados, precisamos de métodos que impedem o acesso direto a um atributo de visibilidade privada.
 - **Método Acessor (GET):** Pega uma determinada informação, mas não dá acesso direto a um atributo.
 - **Método Modificador (SET):** Modifica uma determinada informação, mas não dá acesso direto a um atributo.
- Os métodos GET e SET sempre serão de visibilidade pública, sendo necessário método GET e um método SET para cada atributo. No entanto, eles não são obrigatórios e devem ser feitos para todos os atributos quando for utilizado.

Modificadores de Acesso

+ public
- private
protected

Funcionário
+ nome: caractere + idade: inteiro + setor: caractere - salario: real - rg: caractere - cpf: caractere - efetivo: logico
+ mostrarFuncionario(): void + demitirFuncionario(d: logico): void + aumentarSalario(sal: real): void

Métodos
Acessores

Método **Get**
pegar uma informação

Método **Set**
inserir uma informação

Métodos Get e Set

```
publico classe Funcionario  
    privado salario: Real  
fimClasse
```

f1 = novo Funcionario

**f1.salario = 2789.95
Escreva (f1.salario)**



nome é um atributo *privado*.
Ele não pode ser acessado
diretamente, não sendo possível
atribuir valor e nem ler seu valor.
Portanto, esse código vai dar erro!

```
Publico classe Funcionario  
    privado salario: Real
```

```
publico Método getSalario{  
    retorne salario  
fimMetodo
```

```
publico Método setSalario(s:real){  
    salario = s  
fimMetodo
```

```
fimClasse
```

f1 = novo Funcionario


**f1.setSalario = 2789.95
Escreva (f1.getSalario)**

Agora será possível chamar os
métodos Get e Set para acessar
o atributo privado, para atribuir
(setar) ou ler seu valor.

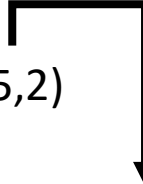
Parâmetros e Argumentos

Parâmetros são valores passados ao método. Ao chamar o método, são passados **argumentos** (valores) eles serão recebidos pelo método que vai utilizar esse números na execução. Os argumentos devem ser do mesmo tipo (inteiro, real, caractere) dos parâmetros do método e na mesma quantidade para não haver erro.

O método receberá dois parâmetros
do tipo inteiro n1 será 5 e n2 será 2



```
Metodo soma(inteiro n1, inteiro n2){  
    inteiro soma = n1 + n2  
    Escreva ("soma")  
fimMetodo
```



```
s1.soma(5,2)
```

5 e 2 serão argumentos para o
método soma, que os receberá
como parâmetro e usará os valores

Referências Bibliográficas

- <https://www.profissionaisti.com.br/2010/10/paradigmas-de-programacao/>
- <https://imasters.com.br/devsecops/paradigmas-de-programacao-sao-importantes>
- <https://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264>
- <https://www.devmedia.com.br/conceitos-da-linguagem-java/5341>
- <https://www.oficinadanet.com.br/post/14614-programacao-orientada-a-objetos>
- <https://www.ramon.pro.br/o-paradigma-orientado-a-objetos/>
- <http://www.dca.fee.unicamp.br/cursos/PooJava/desenvolvimento/umlclass.html>
- <https://pt.slideshare.net/profDanielBrandao/encapsulamento-em-orientao-a-objetos>
- <https://www.cursoemvideo.com/course/curso-de-poo-php/>
- <https://www.cursoemvideo.com/course/curso-de-poo-java/>
- <https://www.caelum.com.br/download/caelum-java-objetos-fj11.pdf>
- <https://www.caelum.com.br/download/caelum-csharp-dotnet-fn13.pdf>

**Material desenvolvido pelo
Prof. Rafael da Silva Polato**

rafael.polato@etec.sp.gov.br