

# Aula 06

## Programação Orientada a Objetos

emerson@paduan.pro.br

# Herança



receber dos antepassados

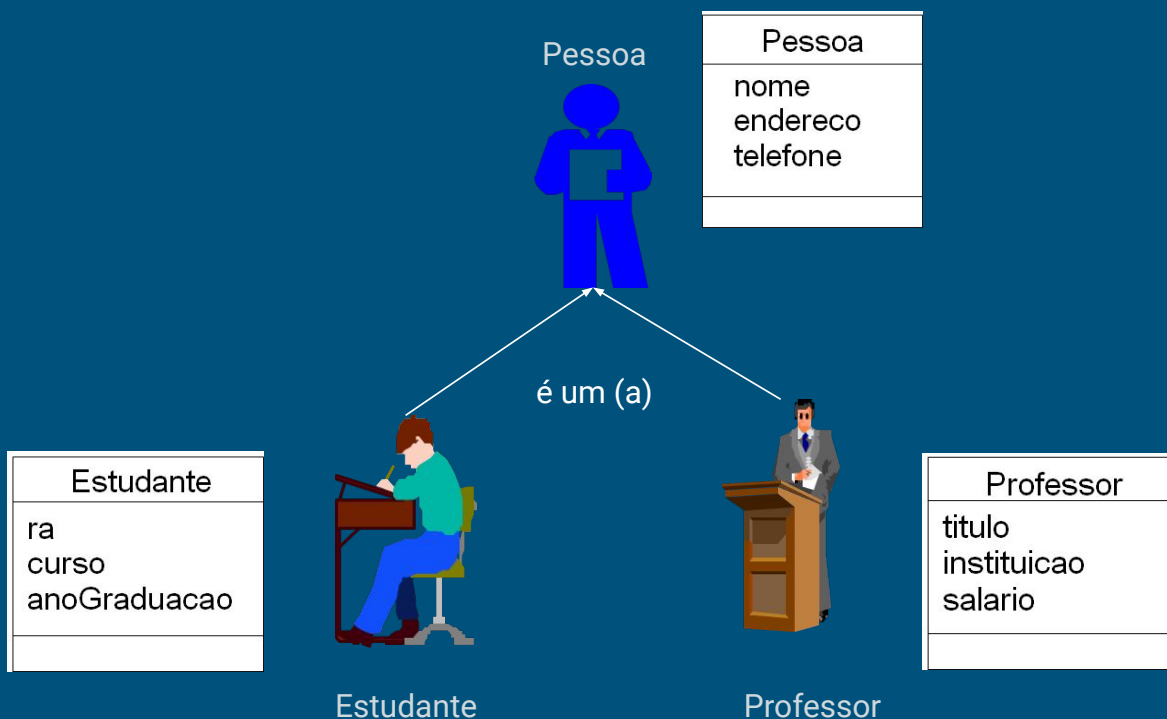
emerson@paduan.pro.br

# Herança

Herança é um mecanismo que permite que características comuns a diversas classes sejam derivadas de uma classe base, ou superclasse.

A herança é uma forma de reutilização de software em que novas classes são criadas a partir das classes existentes, herdando seus atributos e métodos e adicionando novos recursos que as novas classes exigem.

emerson@paduan.pro.br



emerson@paduan.pro.br

# Em Java

```
public class SuperClass {  
    //corpo da superclasse...  
}
```

**extends** - indica que a criação de uma nova classe que **herda** de uma classe existente

```
public class SubClass extends SuperClass {  
    //corpo da subclasse...  
}
```

emerson@paduan.pro.br

## super

Palavra-chave super refere-se a uma superclasse.

Pode indica a chamada ao construtor da superclasse ou ser utilizada para invocar métodos da superclasse dentro da subclasse.

emerson@paduan.pro.br

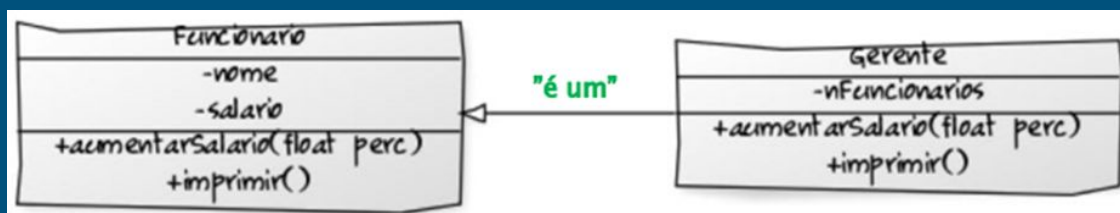
# Let's Code

Vamos criar classes para testar esses conceitos.



emerson@paduan.pro.br

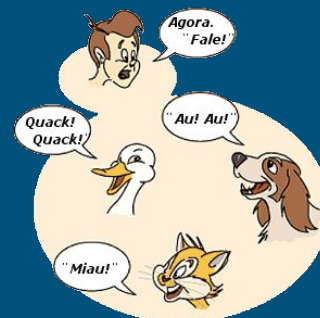
## Exemplo



Sabe-se que o gerente recebe um bônus adicional de 20% além do aumento dos demais funcionários. Os demais funcionários recebem o aumento de acordo com o percentual informado.

emerson@paduan.pro.br

# Polimorfismo



Há muitas formas de "falar"

emerson@paduan.pro.br

## Polimorfismo

Existem dois tipos de polimorfismo: sobrecarga (overload) e sobreposição (override).

### Sobrecarga de métodos (Overload):

Consiste em criar variações de um mesmo método, ou seja, a criação de dois ou mais métodos com nomes iguais em uma classe.

### Sobreposição de métodos (Override):

Um método definido em uma *subclasse* com o mesmo nome e mesma lista de parâmetros que um método em uma de suas classes antecessoras **oculta** o método da classe ancestral a partir da subclasse.

emerson@paduan.pro.br

# Sobrecarga ( Overload )



O mesmo método, múltiplas  
"funções"

emerson@paduan.pro.br

```
public class Pessoa {  
    String nome;  
    float salario;  
  
    public Pessoa() { //construtor default  
        this.nome = "Não cadastrado";  
        this.salario = 0.0;  
    }  
    public Pessoa(String nome, float salario) {  
        this.nome = nome;  
        this.salario = salario;  
    }  
}
```

emerson@paduan.pro.br

# Sobreposição (Override)

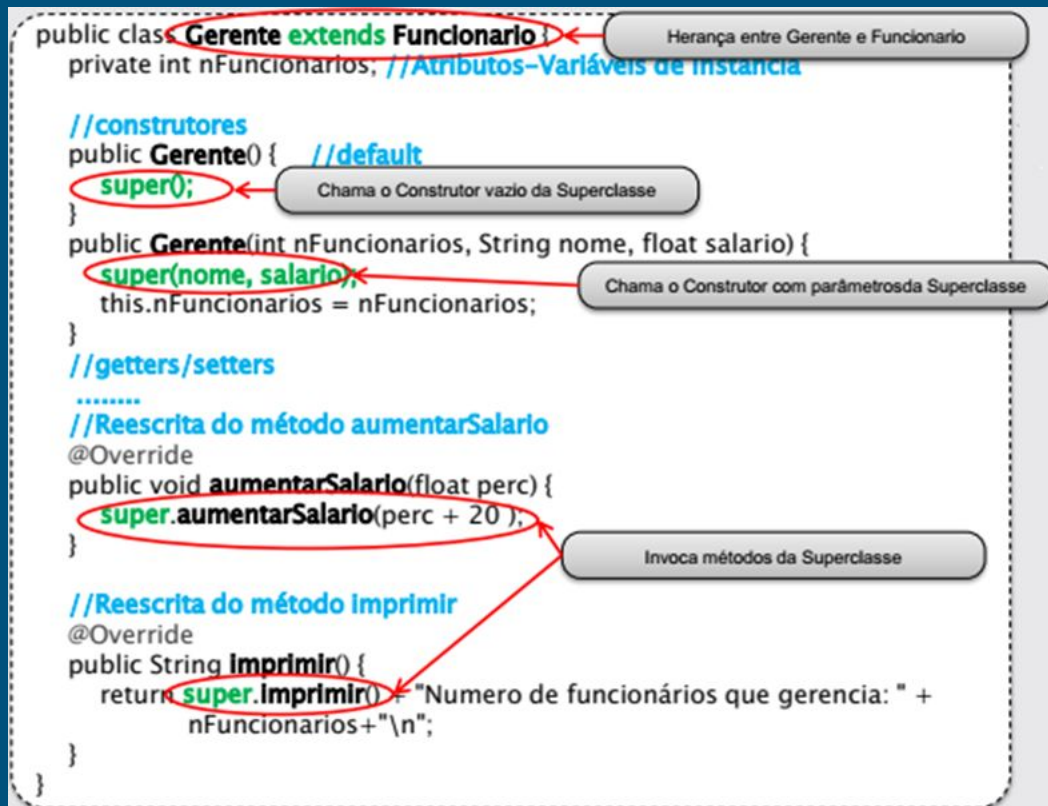


O mesmo método, múltiplas  
"tarefas"

emerson@paduan.pro.br

```
public class Funcionario {  
    //Atributos – Variáveis de Instância  
    private String nome;  
    private float salario;  
  
    //Construtor  
    public Funcionario() { } //default  
    //sobrecarregado  
    public Funcionario(String nome, float salario) {  
        this.nome = nome;  
        this.salario = salario;  
    }  
  
    //getters/setters  
    .....  
  
    //Métodos da classe  
    public void aumentarSalario(float perc){  
        this.salario += this.salario * perc/100.0;  
    }  
  
    public String Imprimir() {  
        return "Funcionario: " + nome +  
            "\nSalário: R$ " + String.format("%.2f \n", salario);  
    }  
}
```

emerson@paduan.pro.br



emerson@paduan.pro.br

# Observe

## Reescrita do método aumentarSalario()

- Para funcionar diferente para gerentes e funcionários comuns (gerente recebe um bônus adicional de 20%).
- Esse método não tem acesso direto às variáveis de instância privadas da superclasse, ou seja, esse método não pode alterar diretamente a variável de instância *salario*, embora cada objeto Gerente tenha uma variável de instância *salario*.
- Modificador de acesso **protected** – dá acesso direto aos atributos da superclasse pela subclasse!

emerson@paduan.pro.br



```

public class AppFuncionarioGerente {
    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);

        Funcionario func = new Funcionario("Jose da Silva", 1000.0f);
        Gerente ger = new Gerente(45, "Joao Medeiros", 5000.0f);

        //calcular 10% de aumento de salário para os funcionarios
        func.aumentarSalario(10);
        ger.aumentarSalario(10);

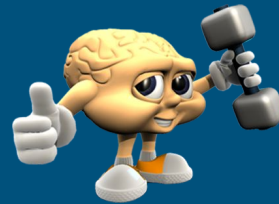
        System.out.println("===== DADOS DO FUNCIONÁRIO =====");
        System.out.println(func.imprimir());
        System.out.println("===== DADOS DO FUNCIONÁRIO =====");
        System.out.println(ger.imprimir());

    }
}

```

emerson@paduan.pro.br

## Exercício 6-1



Crie uma classe Placar, que representa o placar de um jogo de futebol. Discuta quais os parâmetros devem ser adicionados à classe (e os tipos de dados), e faça 3 construtores:

- 1 construtor padrão (default, sem parâmetros);
- 1 construtor que defina apenas quais são os times do jogo (mantendo o placar em 0 a 0);
- 1 construtor que defina os times e o placar do jogo;

Crie no main três objetos do tipo placar, um testar para cada tipo de construtor, e exiba os dados do placar no formato: *Time1 Gols X Gols Time2*.

Exemplo: São Paulo 3 X 0 Corinthians 🤪

emerson@paduan.pro.br

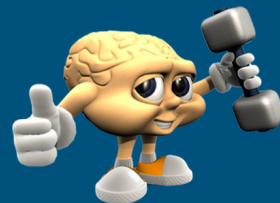
# Classe abstrata

As classes abstratas não permitem instanciar objetos. São classes feitas especialmente para serem modelos para suas classes derivadas.

Métodos abstratos presentes na classe abstrata, obriga a classe filha a definir tais métodos, pois, caso contrário, a classe filha também se tornará abstrata.

emerson@paduan.pro.br

## Exercício 6-2



1. Criar uma classe Conta, que possua um número, um saldo e os métodos para obter dados da conta, depositar e sacar.
2. Crie as subclasses da classe Conta: ContaCorrente, ContaEspecial e ContaPoupanca. A ContaCorrente permite fazer saques somente se houver saldo suficiente. A ContaEspecial possui um limite que permite fazer saques se o saldo mais o limite da conta cobrir o valor pretendido de saque. A ContaPoupança faz saque se houver saldo, mas há uma taxa por operação. Além disso, a ContaCorrente deve reescrever o método deposita, com o objetivo de retirar uma taxa bancária de dez centavos de cada depósito.
3. Crie uma classe AppContas com o método main contendo um menu com opções para realizar operações nas contas.

emerson@paduan.pro.br

# Upcast

É uma conversão na qual subclasses são promovidas a superclasses.

Característica: A conversão é implícita! A promoção é realizada automaticamente!

Exemplo: Gerente "é um" Funcionario

```
Funcionario func = new Gerente(); //upcast
```

emerson@paduan.pro.br

# Downcast

É a operação inversa, superclasses são convertidas em subclasses.

Característica: A conversão é explícita! Tem que indicar o TIPO!

Exemplo:

```
Funcionario func = new Gerente(); //upcast
```

```
Gerente ger = (Gerente) func; //downcast
```

emerson@paduan.pro.br

# Static

---

Aplicado a atributos: um único atributo para todas as instâncias (objetos) da classe

Aplicado a métodos: Pode-se utilizar os métodos sem que seja necessário instanciar um objeto.