Aula 06

Programação Orientada a Objetos

emerson@paduan.pro.br

Antes de iniciar...

Dúvidas ???

Associação entre classes

emerson@paduan.pro.br

Associação

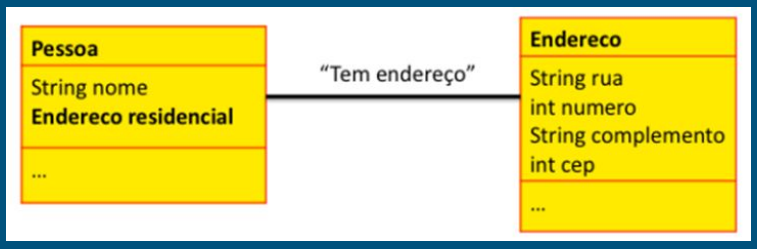
Um sistema é composto por várias Classes.

• As classes se conectam para poderem se comunicar por troca de mensagens (chamadas de métodos)

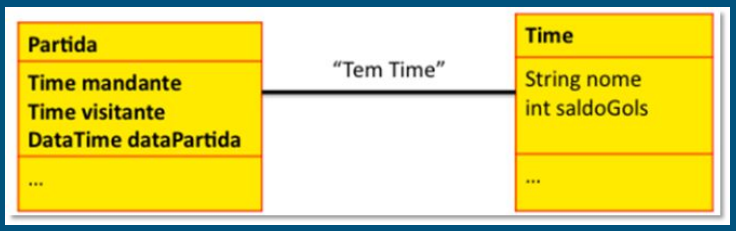
• Quando um ou mais atributos de uma Classe é uma *referência* para outra Classe temos uma **associação**.

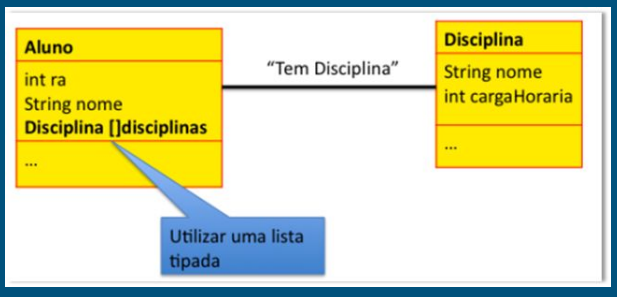
emerson@paduan.pro.br

Exemplo



Exemplo 1: N





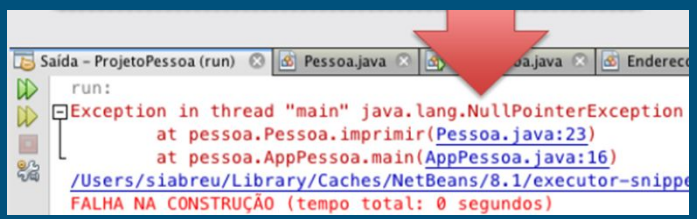
Em Java

```
public class Pessoa{
   //atributos
   private String nome;
   private int idade;
   private char sexo;
   private Endereco end;
   ....

   public String imprimir(){
      return "Nome: " + nome +
             "\nIdade: " + idade +
             "\nSexo: " + sexo +
             "Endereço: " + end.imprimir();
   }
}
```

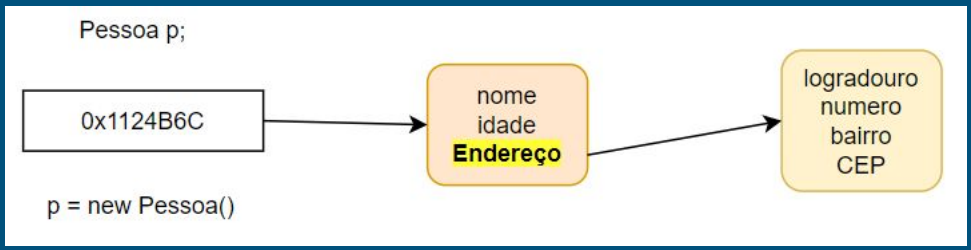
```
public class Endereco{
   //atributos
   private String logradouro;
   private String complemento;
   private int numero;
   private String cep;
   ....

   public String imprimir(){
      return "Logradouro: " + logradouro +
             "\nComplemento: " + complemento +
             "\nNúmero: " + numero +
             "CEP: " + cep;
   }
}
```


Atenção!!!

```
public class AppPessoa {
   public static void main(String[] args) {
      Pessoa objPessoa = new Pessoa();

      System.out.println(objPessoa.imprimir());
   }
}
```



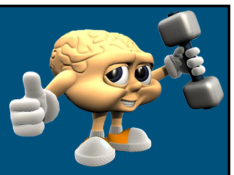
Criar objeto no construtor



Resolvendo

```
public class Pessoa{
    //atributos
    private String nome;
    private int idade;
    private char sexo;
    private Endereco end;

    //construtor default
    public Pessoa() {
        this.end = new Endereco();
    }

    public String imprimir(){
        return "Nome: " + nome +
               "\nIdade: " + idade +
               "\nSexo: " + sexo +
               "Endereço: " + end.imprimir();
    }
}
```

Exercício 6-1

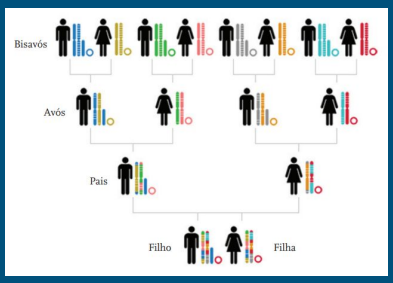


1. Criar um relacionamento de Associação entre as classes Animal e Proprietário:
   Animal "Tem UM" *Proprietario*

2. Criar a classe *Animal* com os seguintes atributos: nome, raça, ano de nascimento e proprietário
3. Criar a Classe *Proprietario* com os seguintes atributos: nome e telefone
4. Criar uma classe main

emerson@paduan.pro.br

Herança



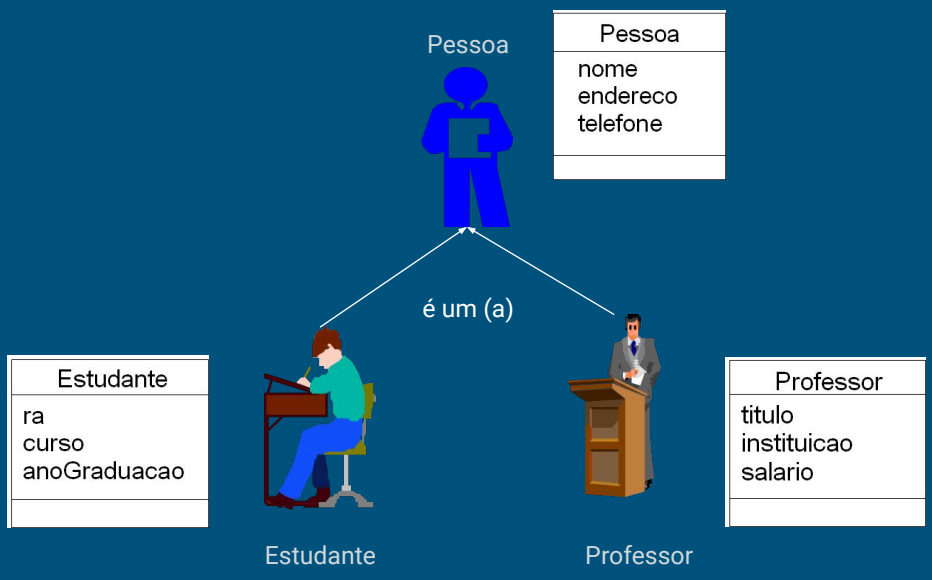
receber dos antepassados

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.



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.

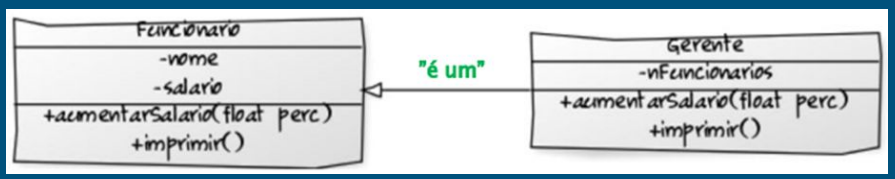
Pode indica a chamada ao construtor da superclasse ou ser utilizada para invocar métodos da superclasse dentro da subclasse.

emerson@paduan.pro.br

Let's Code

Vamos criar classes para testar esses conceitos.

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.

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)

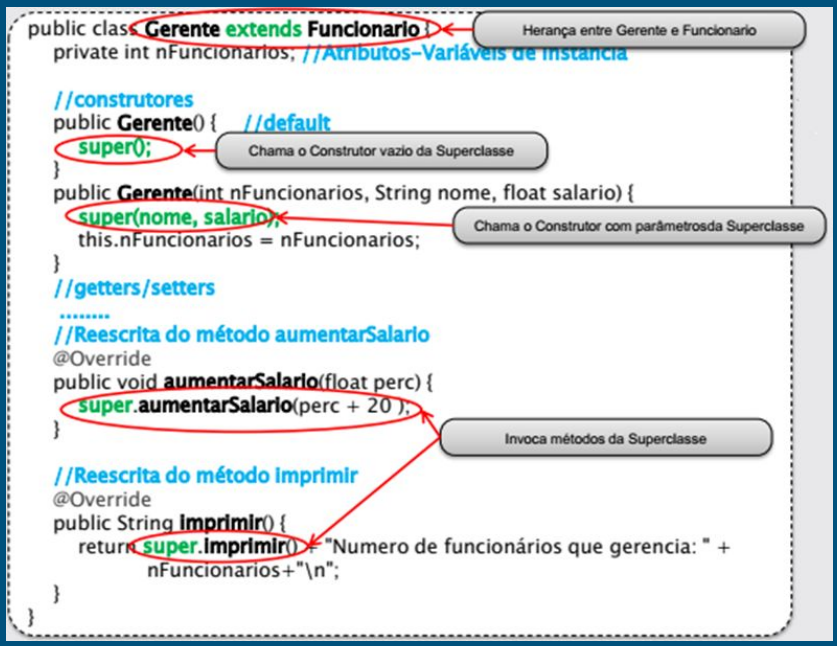
O mesmo método, múltiplas "funções"


```
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;
    }
}
```

Sobreposição (Override)

O mesmo método, múltiplas "tarefas"


```
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);
    }
}
```



Observe

Reescrita do método aumentarSalario()

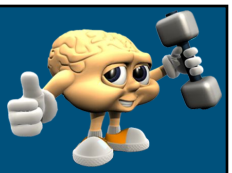
– Para funcionar diferente para gerentes e funcionários comuns (gerente recebe um bônus adicional de 20%).

– Esse método não tem acesso direto às variáveis de instância privados 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!

```
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());

    }
}
```


Exercício 6-2



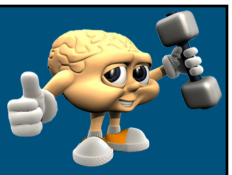
Crie uma classe Placar, que representa o placar de um jogo de futebol. Discuta quais atributos devem ser adicionados à classe (e os tipos de dados), e faça 3 construtores:

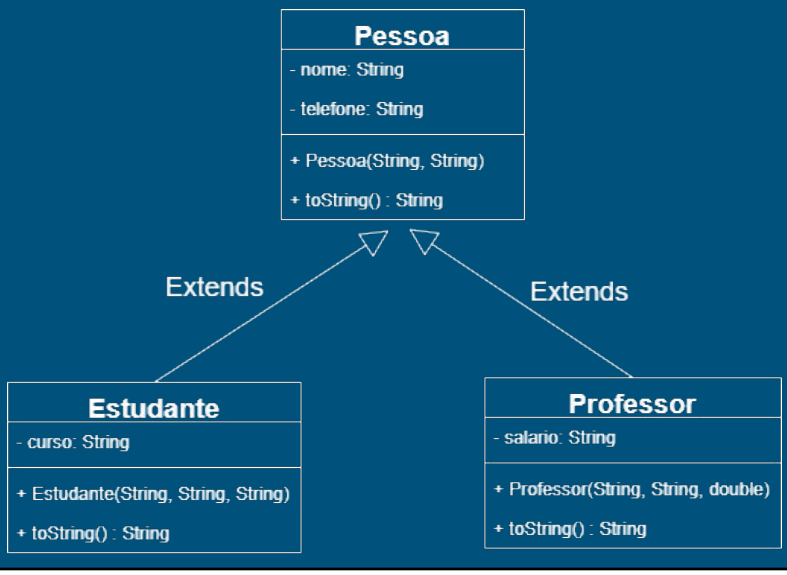
– 1 construtor padrão (default, sem parâmetros);
– 1 construtor que defina apenas quais são os times do jogo (mantendo o placar em 0 a 0);
– 1 construtor que defina os times e o placar do jogo;

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 1 X 0 Corinthians 😜

Exercício 6-3



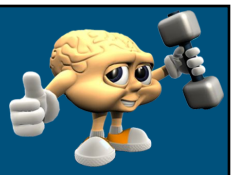


Classe abstrata

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.

Exercício 6-4



1. Criar uma classe Conta, que possua um número, e um saldo, e os métodos para obter dados da conta, realizar depósito e saque.
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.

Static

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

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

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
```


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
```