# Linguagens de Programação

Java Orientado a Objetos

Prof. Waldeck Lindoso Jr.

# Construtores

- Em Java, o construtor é definido como um método cujo nome deve ser o mesmo nome da classe e sem indicação do tipo de retorno -- nem mesmo **void**. O construtor é unicamente invocado no momento da criação do objeto através do operador **new**.
- O construtor pode receber argumentos, como qualquer método.

```
11      Carro() {}
12
13      Carro(String marca, String modelo, int numPassageiros,
14              double capCombustivel, double consumoCombustivel){
15          this.marca = marca;
16          this.modelo = modelo;
17          this.capCombustivel = capCombustivel;
18          this.consumoCombustivel = consumoCombustivel;
19      }
```

# Palavra-chave *this*

- Praticamente a palavra-chave **this** é uma referência ao objeto atual (ele mesmo). isso pode ser usado dentro de algum método ou construtor.
- Esta é uma palavra-chave em Java, que pode ser usada dentro de um método ou do construtor da classe. Ou ainda, quando queremos deixar claro que o método ou atributo que estamos querendo chamar, é daquela própria classe.

```
23    void exibirAutonomia() {
24        System.out.println("A autonomia do carro é: " + this.capCombustivel * this.consumoCombustivel + " km");
25    }
26
27    double obterAutonomia() {
28        return this.capCombustivel * this.consumoCombustivel;
29    }
```

# Métodos **GET** e **SET** (Encapsulamento)

- Conceitua-se encapsulamento como sendo o processo utilizado para proteger os campos e operações de uma classe (atributos e métodos), permitindo que apenas os membros públicos - em Java métodos Get / Set - sejam acessados pelos usuários de determinada classe.

```
5     private String marca;
6     private String modelo;
7     private int numPassageiros;  // número de passageiros
8     private double capCombustivel; // capacidade do tanque de combustível
9     private double consumoCombustivel;  // consumo de combustível por km
```

# Métodos **GET** e **SET** (Encapsulamento)

- Os métodos **GET** e **SET** são técnicas padronizadas para gerenciamento sobre o acesso dos atributos. Nesses métodos determinamos quando será alterado um atributo e o acesso ao mesmo, tornando o controle e modificações mais práticas e limpas, sem contudo precisar alterar assinatura do método usado para acesso ao atributo.

```
23    public String getMarca() {
24        return marca;
25    }
26
27    public void setMarca(String marca) {
28        this.marca = marca;
29    }
30
```

# Sobrecarga de **métodos** (**overload**)

- A sobrecarga de métodos (**overload**) é um conceito do polimorfismo que consiste basicamente em criar variações de um mesmo método, ou seja, a criação de dois ou mais métodos com nomes totalmente iguais em uma classe.

```
5    public int soma(int num1, int num2) {
6        return num1 + num2;
7    }
8
9    public int soma(int num1, int num2, int num3) {
10       return num1 + num2 + num3;
11   }
12
13   public double soma(double num1, double num2) {
14       return num1 + num2;
15   }
```

# Sobrecarga de **construtores** (**overload**)

```
11    Carro() {
12        System.out.println("Classe Carro foi instanciada!");
13    }
14
15    Carro(String marca, String modelo, int numPassageiros,
16            double capCombustivel, double consumoCombustivel){
17        this.marca = marca;
18        this.modelo = modelo;
19        this.capCombustivel = capCombustivel;
20        this.consumoCombustivel = consumoCombustivel;
21        System.out.println("Classe Carro foi instanciada com todos os atributos!");
22    }
23
24    Carro(String marca, String modelo, int numPassageiros){
25        this.marca = marca;
26        this.modelo = modelo;
27        System.out.println("Classe Carro foi instanciada com 2 dos atributos!");
28    }
```

# Variáveis e Métodos estáticos (**static**)

- Quando declaramos um método ou uma variável em uma classe, por default, o mesmo será acessado a partir do objeto, ou seja, para utilizarmos este método ou variável teremos que instanciar um objeto desta classe.

```
5    public static int soma(int num1, int num2) {
6        return num1 + num2;
7    }
8    public static double soma(double num1, double num2) {
9        return num1 + num2;
10   }
11   public static int soma(int num1, int num2, int num3) {
12       return num1 + num2 + num3;
13   }
```

# Variáveis e Métodos estáticos (**static**)

- Métodos e variáveis estáticas são elementos que pertencem à classe e não ao objeto, dessa forma quando os declaramos temos que usá-los a partir da classe.

```
6    public static void main(String[] args) {
7        System.out.println(MinhaCalculadoraEstatica.soma(1, 2));
8        System.out.println(MinhaCalculadoraEstatica.soma(1.1, 2.2));
9        System.out.println(MinhaCalculadoraEstatica.soma(1, 2, 3));
10       int[] vetInt = {1,2,3,4,5};
11       System.out.println(MinhaCalculadoraEstatica.soma(vetInt));
12   }
```

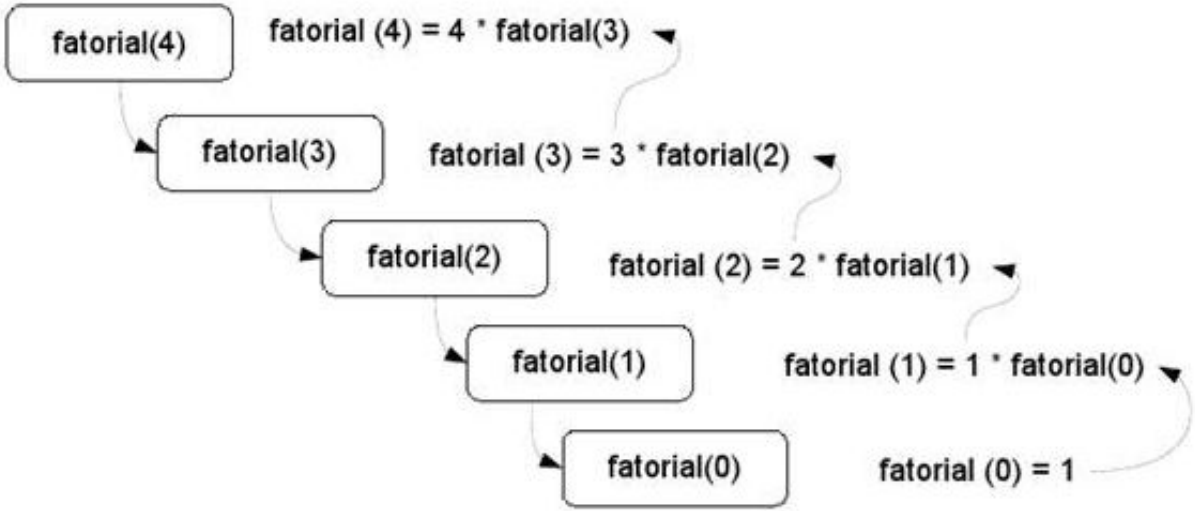
# Recursividade

- Método que chama a si mesmo
- Precisa de um ponto de parada

```
18      static int fatorial(int num) {
19          if (num == 0) return 1; // ponto de parada definido
20          return num * fatorial(num-1); // chamada para ele mesmo
21      }
```

# Recursividade

- 4! = 4 * 3 * 2 * 1 = 24



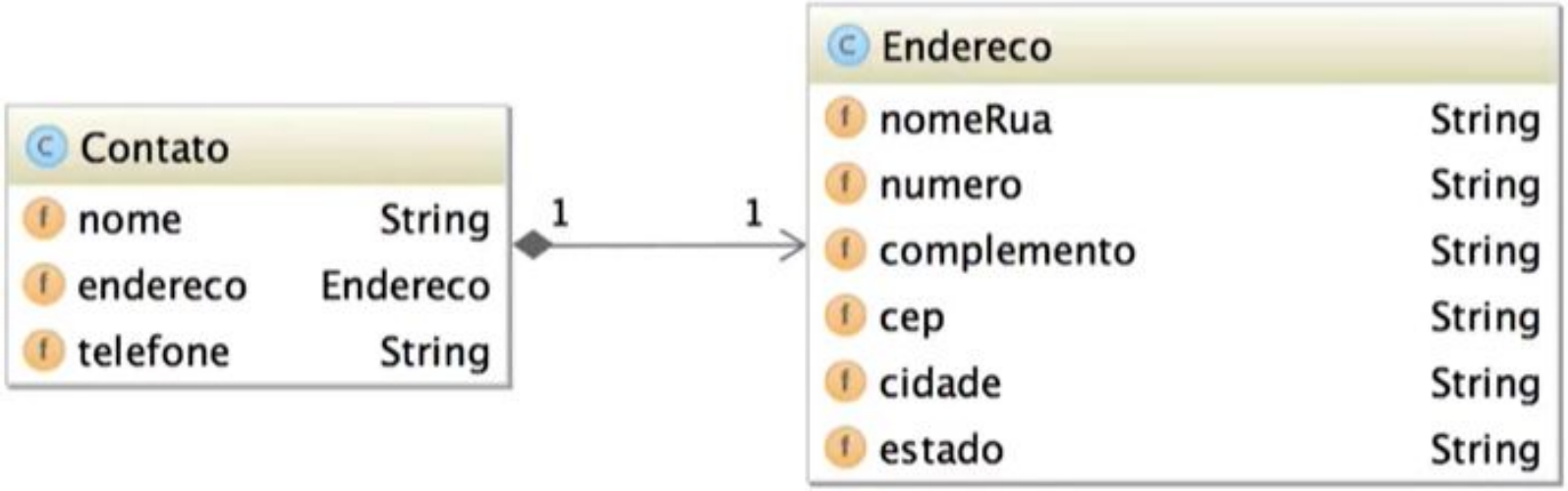
# Recursividade (Fibonacci)

- Escreva um método recursivo e estático que calcule e retorne o N-ésimo termo da sequência de fibonacci.

```
24    static int fibonacci(int num) {
25        if (num < 2) return 1;
26        return fibonacci(num - 1) + fibonacci(num - 2);
27    }
```

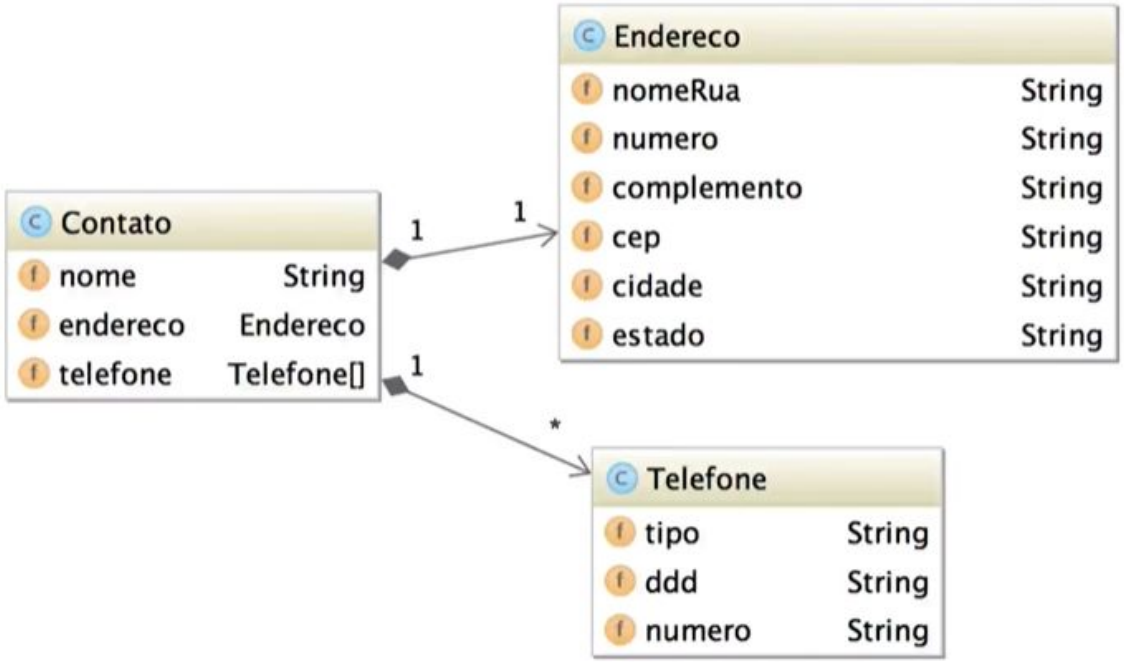
# Relacionamento (tem 1)

- **1** contato **tem 1** endereço.



# Relacionamento (tem 1 e tem muitos)

- **1** contato **tem muitos** telefones.

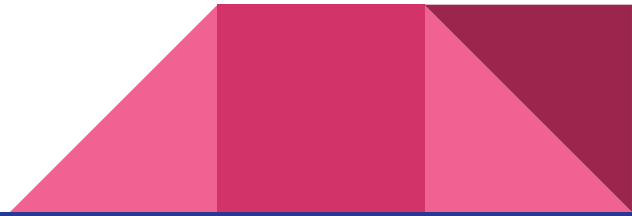


# Herança (**extends**)

- A herança é um mecanismo da Orientação a Objeto que permite criar novas classes a partir de classes já existentes, aproveitando-se das características existentes na classe a ser estendida.

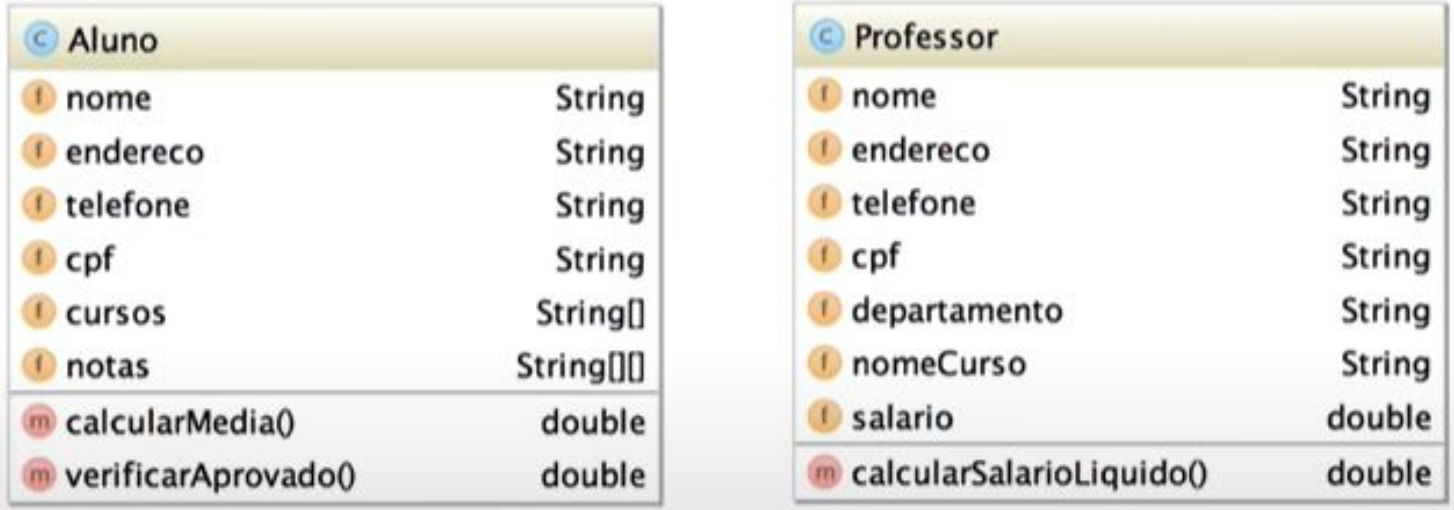
# Herança (**extends**)

- A linguagem Java permite o uso de herança simples, mas não permite a implementação de herança múltipla.



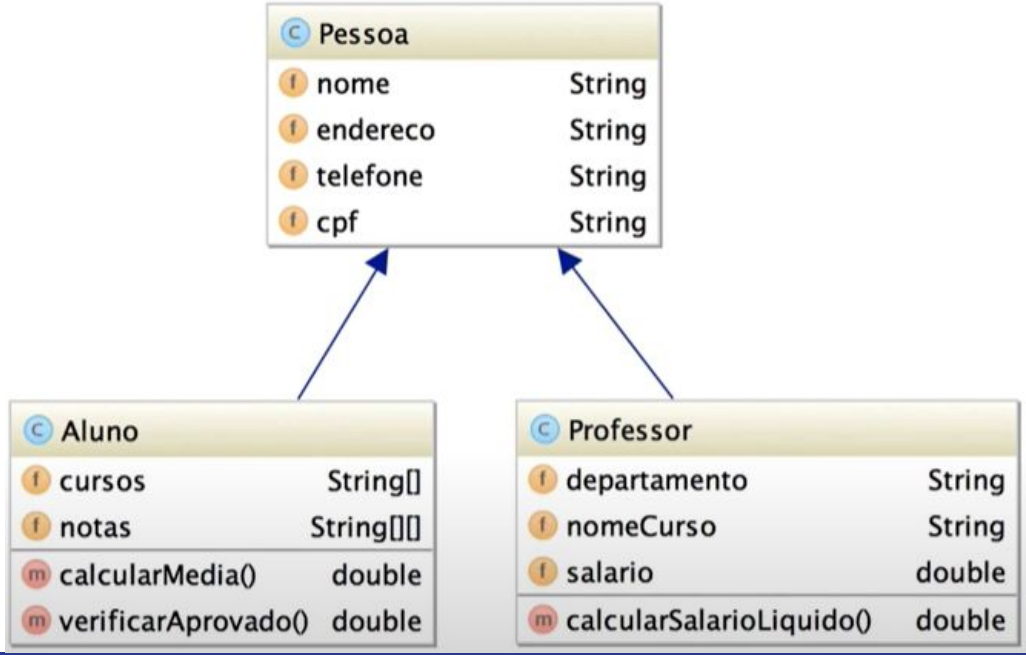
# Herança (**extends**)

- Note que tem alguns atributos que se repetem



# Herança (**extends**)

- Então criaremos outra classe “pessoa” e herdaremos esses atributos dela.



# HANDS ON

{ trust_ful } >>>

# Obrigado!