



SÃO  
PAULO  
TECH  
SCHOOL

# **Estrutura de Dados**

## **Interface & Polimorfismo**

**Prof<sup>a</sup> Célia Taniwaki**

**Prof<sup>a</sup> Giuliana Miniguiti**

**celia.taniwaki@sptech.school**

**giuliana.franca@sptech.school**

# Polimorfismo

Significa muitas (**poli**) formas (**morfo**):

Em Programação Orientada a Objetos, refere-se ao fato de um mesmo método produzir resultados diferentes, pode acontecer em 2 casos



## Métodos sobrecarregados:

Mesmo método com lista de parâmetros diferentes (assinaturas diferentes).

- O Java sabe qual método executar pelo número de parâmetros passados.
- **Ex.:** método média que recebe 2 valores e devolve a média aritmética dos 2 valores e método média que recebe 3 valores e devolve a média aritmética dos 3 valores

# Métodos sobrescritos


Método existente na classe mãe, reescrito na classe filha (mesma assinatura).

- O Java sabe qual método executar devido ao nome do objeto associado ao método chamado.
- **Ex.:** método calcSalario das classes Vendedor e Horista



## // Métodos Sobrecargados

```
class Media {  
    public static double media (double a, double b){  
        return (a + b) / 2;  
    }  
    public static double media (double a, double b, double c){  
        return (a + b + c) / 3;  
    }  
    public static void main(String args[]){  
        System.out.println (media (5, 6));  
        System.out.println (media (5, 6, 7));  
    }  
}
```



## // Método Sobrescritos ou Reescritos

```
public void exhibeTotalSalario(){  
    double total = 0.0;  
    for (Funcionario f: lista){  
        total += f.calcSalario();  
        System.out.println(  
            "O total gasto em salário é " + total);  
    }  
}
```

A person's hands are shown typing on a laptop keyboard. The laptop screen displays a code editor with syntax-highlighted code. The background is dark and out of focus, showing a desk and a smartphone.

# // Interfaces




# Implementação

Declaração de interface semelhante à declaração de uma classe, porém utiliza-se **interface** no lugar de **class**

---

A classe que implementa os métodos especificados numa interface deve ter em sua declaração a cláusula **implements**



```
public interface NomeInterface {  
    // assinaturas dos métodos  
}
```

```
public class NomeClasse implements  
NomeInterface {  
    // implementação dos métodos  
    // definidos na interface  
}
```

# Interface & Polimorfismo

O nome da interface pode ser utilizado como tipo de um vetor (ou ArrayList) ou como tipo de um parâmetro passado a um método

Ex.: interface chamada Tributavel

- Pode-se declarar um List:

- `List<Tributavel> lista = new ArrayList<>();`

- Cada elemento dessa lista pode ser um objeto de qualquer classe que implementa Tributavel

- Pode-se ter um método que recebe obj Tributavel

- `public void adicionaTributavel(Tributavel obj)`

- Esse método aceita como parâmetro um objeto de qualquer classe que implementa Tributavel

# Interface & Polimorfismo

Dessa forma, todos os objetos que estão no List de tipo Tributavel implementam os métodos definidos na interface Tributavel, portanto, é possível chamar obj.metodo( )

Ex: método que calcula total de tributos:

```
public double calculaTotalTributos( ) {  
    double total = 0.0;  
    for (Tributavel t : lista){  
        total = total + t.getValorTributo();  
    }  
    return total;  
}
```

## Interface

- Somente métodos abstratos
- Não depende de Herança
- Uma classe pode implementar várias interfaces
- Não pode ter construtores
- Não pode ter atributos/variáveis de instância (somente variáveis constantes)

## Classe Abstrata

- Métodos abstratos e Concretos
- Depende de Herança
- Uma classe pode implementar somente uma classe abstrata
- Pode ter construtores
- Pode ter atributos

**Agradeço**  
a sua atenção!

**Célia Taniwaki**

celia.taniwaki@sptech.school

**Giuliana Miniguiti**

giuliana.franca@sptech.school

**SÃO  
PAULO  
TECH  
SCHOOL**