

# Aula 6

# Resumindo - Polimorfismo

O que guarda uma variável do tipo Funcionario ?

- Uma referência para um Funcionario , nunca o objeto em si.

Na herança, vimos que todo Gerente é um Funcionario;

- Podemos nos referir a um Gerente como sendo um Funcionario.
- Se alguém precisa falar com um Funcionario do banco, pode falar com um Gerente? Porque?
- Gerente é um Funcionario. Essa é a semântica da herança.

# Polimorfismo

Como representamos isso no código?

```
Gerente gerente = new Gerente();  
Funcionario funcionario = gerente;  
funcionario.setSalario(5000.0);
```

OU

```
Funcionario funcionario = new Gerente();  
funcionario.setSalario(5000.0);
```

# Polimorfismo

Polimorfismo é a capacidade de um objeto poder ser referenciado de várias formas:

- Cuidado, polimorfismo não quer dizer que o objeto se transforma;
- Um objeto nasce de um tipo e morre daquele tipo;
- O que muda é a maneira como referenciar o objeto.

## Associação com o mundo real

- Pensem no polimorfismo como uma procuração para que seu parente possa resolver alguma coisa para você.
- Ou então como uma permissão para acesso a uma área restrita que apenas funcionários pudessem acessar;

# Polimorfismo

Até aqui tudo bem, mas e se eu implementar assim:

```
Funcionario funcionario = new Gerente();  
funcionario.setSalario(5000.0);  
funcionario.getBonificacao();
```

- Qual é o retorno desse método? 500 ou 750?
- A invocação de método sempre vai ser decidida em tempo de execução;
- O objeto será procurado na memória para decidir qual método será chamado;
- Será identificado o objeto de verdade, e não o que usamos para referenciá-lo.
- Apesar de estarmos nos referenciando a esse Gerente como sendo um Funcionario , o método executado é o do Gerente . O retorno é 750.

# Polimorfismo

Parece estranho criar um gerente e referenciá-lo como apenas um funcionário.

Por que faríamos isso?

Na verdade, a situação que costuma aparecer é a que temos um método que recebe um argumento do tipo Funcionario :

```
public class FolhaPagamento {  
  
    public void calculaFolhaPagamento(Funcionario funcionario) {  
        return funcionario.getSalario() + funcionario.getBonificacao();  
    }  
  
}
```

# Polimorfismo

E, na minha aplicação (ou no main , se for apenas para testes):

```
FolhaPagamento folha = new FolhaPagamento();
```

```
Funcionario funcionario1 = new Gerente();  
funcionario1.setSalario(5000.0);  
folha.calcularFolhaPagamento(funcionario1);
```

```
Funcionario funcionario2 = new Diretor();  
funcionario2.setSalario(1000.0);  
folha.calcularFolhaPagamento(funcionario2);
```

# Polimorfismo

Se criarmos uma classe Secretaria, filha de Funcionario , precisaremos mudar a classe de FolhaPagamento?

- Não;
- Basta a classe Secretaria reescrever os métodos que lhe parecerem necessários;
- É exatamente esse o poder do polimorfismo, juntamente com a reescrita de método:
  - Diminuir o acoplamento entre as classes,
  - Evitar que novos códigos provoquem modificações em vários lugares.



# Matéria nova

O que iremos aprender:

- Classes abstratas;
- Interfaces;
- Tratamento de Erros;

# Classes Abstratas

- São classes feitas especialmente para servirem de modelo para suas classes derivadas
- Estas classes **não** permitem realizar qualquer tipo de instância.
- As classes derivadas deverão **sobrescrever os métodos** para realizar a implementação dos mesmos.
- As classes derivadas das classes abstratas são conhecidas como classes concretas.
- Caso um ou mais métodos abstratos estejam presentes nessa classe abstrata, a classe filha será, então, forçada a definir tais métodos, pois, caso contrário, a classe filha também se tornará abstrata.

# Classes Abstratas

## Definição de uma classe abstrata

```
public abstract class Conta {  
    private double saldo;  
    public void setSaldo(double saldo) {  
        this.saldo = saldo;  
    }  
    public double getSaldo() {  
        return this.saldo;  
    }  
    public abstract void imprimeExtrato();  
}
```

# Classes Abstratas

Implementação de uma classe derivada de classe abstrata.

```
public class ContaPoupanca extends Conta {  
    @Override  
    public void imprimeExtrato() {  
        System.out.println("### Extrato da Conta ###");  
        SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/aaaa HH:mm:ss");  
        Date date = new Date();  
        System.out.println("Saldo: " + this.getSaldo());  
        System.out.println("Data: " + sdf.format(date));  
    }  
}
```

# Interfaces

- Interfaces são padrões definidos através de contratos ou especificações.
- Um contrato define um determinado conjunto de métodos que serão implementados nas classes que assinarem esse contrato.
- Uma interface é 100% abstrata, ou seja, os seus métodos são definidos como abstract, e as variáveis por padrão são sempre constantes (**static final**).
- Uma interface é definida através da palavra reservada “**interface**”.
- Para uma classe implementar uma interface é usada a palavra “**implements**”;

# Interfaces

- A linguagem Java não tem herança múltipla e as interfaces ajudam nessa questão
- Uma classe pode ser herdada apenas uma vez, mas pode implementar inúmeras interfaces.
- As classes que forem implementar uma interface terão de adicionar todos os métodos da interface ou se transformar em uma classe abstrata.

# Interfaces

Definição de uma interface:

```
public interface Conta{  
    void depositar(double valor);  
    void sacar(double valor);  
    double getSaldo();  
}
```

# Interfaces

## Definição de uma interface:

```
public class ContaPoupanca implements Conta
{
    private double saldo;
    @Override
    public void deposita(double valor) {
        this.saldo += valor;
    }
    @Override
    public double getSaldo() {
        return this.saldo;
    }
}
```

```
@Override
public void sacar(double valor) {
    this.saldo -= valor;
}
```



# Interfaces X Classes Abstratas

Tabela comparativa entre classes abstratas e interfaces:

Característica	Interface	Classe Abstrata
Herança múltipla	Uma classe pode implementar diversas interfaces	Uma classe pode herdar somente uma classe
Implementação Padrão	Uma interface não pode conter qualquer tipo de código, muito menos código padrão.	Uma classe abstrata pode fornecer código completo, código padrão ou ter apenas a declaração de seu esqueleto para ser posteriormente sobrescrita.
Constantes	Suporte somente constantes do tipo estática.	Pode conter constantes estáticas e de instância.
Clareza	Todas as declarações de constantes em uma interface são presumidamente públicas ou estáticas.	Você pode por código compartilhado em uma classe abstrata. Você pode usar código para computar o valor inicial de suas constantes e variáveis de instância ou estáticas.
Funcionalidades Adicionais	Se você incluir um novo método em uma interface você precisa ajustar todas as implementações da interface.	Se você incluir um novo método em uma classe abstrata você tem a opção de fornecer uma implementação padrão para ele.