

# Linguagem de Programação Orientada a Objetos I

Classes Abstratas e Interfaces

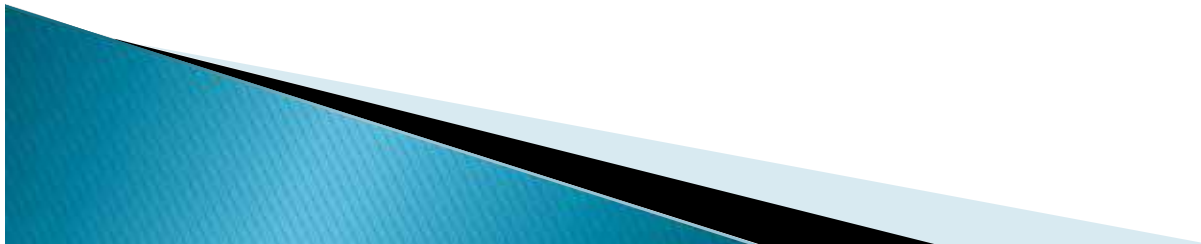
Prof. Tales Bitelo Viegas

<https://fb.com/ProfessorTalesViegas>



# Introdução

- ▶ A boa prática de programação diz que o código comum a diversas classes deve ser colocado em uma classe-base (Template pattern)
- ▶ Entretanto, as vezes verificamos que esta classe-base não deveria permitir instanciação direta



# Introdução

- ▶ Exemplo:
- ▶ Aluno e Professor
- ▶ Criação da classe Pessoa



# Introdução

## ▶ Classes Abstratas

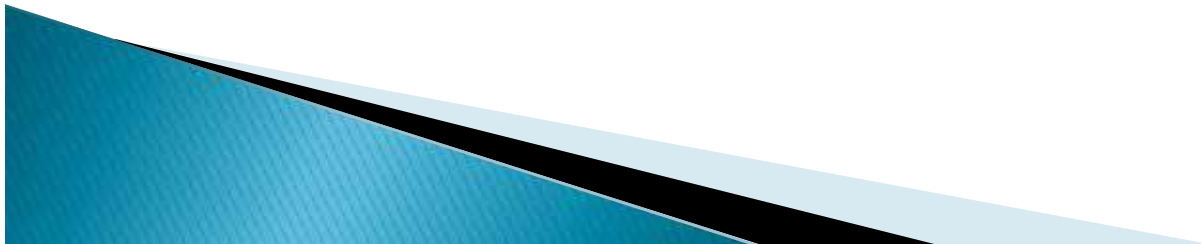
- Representam um conceito abstrato (modelo)
- Não pode ser instanciada (gerar um objeto)
- Servem apenas para permitir a derivação de novas classes
- Identificamos uma classe como abstrata pelo palavra reservada **abstract**
- Exemplos:
  - `public abstract class nomeClasse { ... }`



# Introdução

## ▶ Podem conter:

- Atributos de instância
- Atributos de classe (static)
- Construtores
  - Chamados através de super pelas classes filhas
- Métodos
  - Abstratos
  - Concretos



# Métodos Concretos

- ▶ São métodos que possuem implementação
- ▶ Todos os métodos que criamos até agora na disciplina são métodos concretos



# Métodos Abstratos

- ▶ São métodos declarados nas classes abstratas que não possuem implementação, apenas a definição de sua interface.
- ▶ Identificados pela palavra reservada **abstract** na declaração
- ▶ Servem para definir um comportamento para as classes filhas
- ▶ Importante:
  - Não podem ser declarados como *private*
  - Não podem ser declarados como *final*
  - Não podem ser declarados como *static*



# Classe Abstrata

## ▶ Exemplo

```
public abstract class Pessoa{  
    protected String nome;  
    protected int anoNascimento;
```



Atributos

```
    public Pessoa(){};
```

```
    public Pessoa(String nome, int anoNascimento){  
        this.nome = nome;  
        this.anoNascimento = anoNascimento;  
    }
```



Construtores

...





# Classe Abstrata

## ▶ Exemplo

...

```
public void setNome(String nome){  
    this.nome = nome;  
}  
public String getNome(String nome){  
    return this.nome;  
}  
public abstract void exibirDados();  
public abstract int calcularIdade();  
}
```



Métodos  
Concretos



Métodos  
Abstratos



# Classe que Implementa

```
public class Funcionario extends Pessoa{

    public Funcionario(String nome, int anoNascimento){
        super(nome, anoNascimento);
    }

    public void exibirDados(){
        System.out.println("Nome: " + this.nome);
        System.out.println("Ano de Nascimento: " + this.anoNascimento);
    }

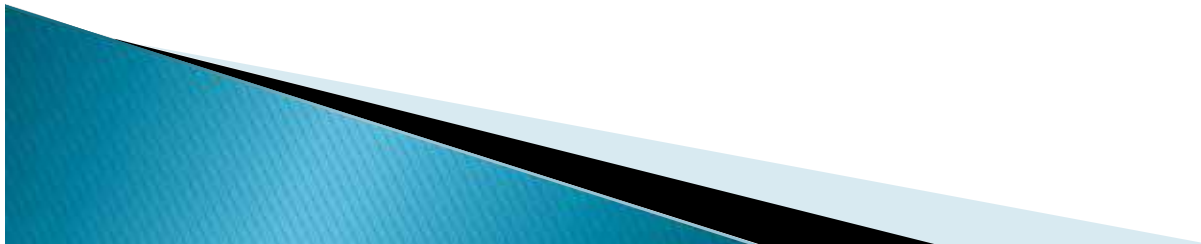
    public int calcularIdade(){
        int anoAtual = Calendar.getInstance().get(Calendar.YEAR);
        return anoAtual - this.anoNascimento;
    }

    ...
}
```



# Interfaces

- ▶ São classes abstratas “puras”
- ▶ São “contratos” que definem o que a classe poderá fazer, mas não dizem nada sobre a maneira como será feito
- ▶ Implementação fica a cargo das classes que utilizam a interface
- ▶ Assim como a abstrata não pode ser instanciada



# Interfaces – Aplicações

- ▶ Definir um comportamento que pode ser implementado por qualquer classe, independentemente da sua hierarquia de classes.
- ▶ Captura de similaridades entre classes não relacionadas, sem ser obrigado a criar artificialmente uma relação de herança entre elas.
- ▶ Declaração de métodos que uma ou mais classes esperam implementar



# Interfaces

## ▶ Características

- Diferença entre classes abstratas e interfaces: é que uma classe filha pode herdar de apenas uma única classe (abstrata ou não) enquanto qualquer classe pode implementar várias interfaces simultaneamente
- Mecanismo “simplificado” de implementação de herança múltipla em Java
- Interfaces podem ser úteis para implementação de bibliotecas de constantes (pois todos os atributos são static e final)



# Interfaces – Criação

- ▶ Declaradas através da palavra-chave *interface* e o modificador de acesso *public*
- ▶ Não tem construtor
- ▶ Não tem variáveis de instância (a não ser constantes estáticas)
- ▶ Todos os métodos são abstratos e públicos (a classe que implementa a interface deve implementar todos os métodos definidos na interface)
- ▶ Não pode ter métodos estáticos



# Interfaces

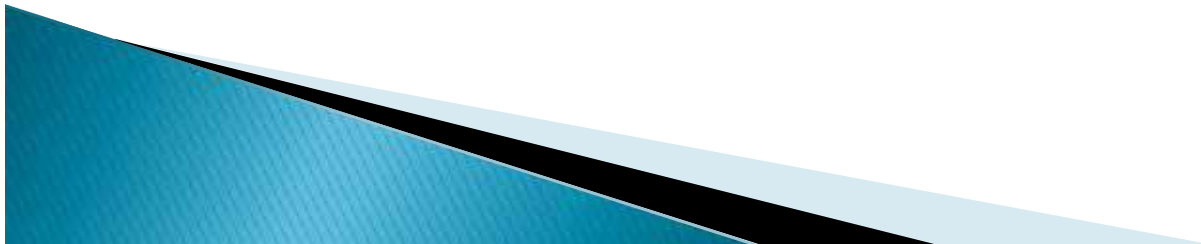
- ▶ Uma classe que implementar a interface deve implementar todos os métodos definidos na interface
- ▶ Uma classe pode implementar interfaces múltiplas
  - Ex: class Tile extends Rectangle implements Cloneable, Comparable
- ▶ Uma interface pode estender outras interfaces
- ▶ Uma interface não pode implementar outras interfaces



# Interfaces – Exemplos

## ► Interface *Comparable*

- A interface *Comparable*:
  - Definida no pacote `java.lang`
  - Serve para comparar objetos
- Geralmente serve na ordenação de elementos
- A classe que implementar *Comparable* é obrigada a fornecer a implementação para o método abstrato:
  - `int compareTo(Object o)`






# Interfaces – Exemplos

```
public interface Pessoa {  
    public static final int max = 10;  
  
    public abstract float calculaSalario();  
}
```



# Exemplo com *Comparable*

```
public class Carro implements Comparable {  
    private String modelo;  
    private int placa;  
    ...  
    public String toString(){  
        return (this.modelo + " " + this.placa);  
    }  
    public int compareTo(Object o){  
        Carro c = (Carro) o;  
        if(this.placa < c.getPlaca())  
            return -1;  
        else {  
            if (this.placa > c.getPlaca() )  
                return 1;  
            else  
                return 0;  
        }  
    }  
}
```



# Exemplo com *Comparable*

## ► Funcionamento do Exemplo:

- retorna negativo se caso a placa atual seja menor que a placa de comparação  
`if(this.placa < c.getPlaca())`  
`return -1;`
- retorna positivo se caso a placa atual seja maior que a placa de comparação  
`if(this.placa > c.getPlaca())`  
`return 1;`
- retorna zero se caso a placa atual seja igual que a placa de comparação  
`if(this.placa == c.getPlaca())`  
`return 0;`



# Exemplo com *Comparable*

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
        ArrayList<Carro> vectorCarros = new ArrayList<Carro>();
        vectorCarros.add(new Carro("Corsa", 10));
        vectorCarros.add(new Carro("Chevete", 1));
        vectorCarros.add(new Carro("Kombi", 4));
        // imprimir antes da ordenacao
        System.out.println(vectorCarros);
        // Ordenar
        Collections.sort(vectorCarros);
        // imprimir apos a ordenacao
        System.out.println(vectorCarros);
    }
}
```

