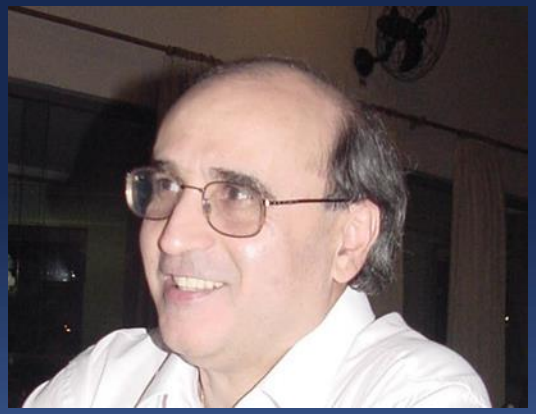
# Programação Orientada a Objetos

# Unidade 4 – Interfaces



Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecidovfreitas@gmail.com

# Bibliografia


- Beginning Java 2 – Ivor Horton – 1999 WROX
- Java2 – The Complete Reference – 7th Edition – Herbert Schildt – Oracle Press
- Core Java Fundamentals – Horstmann / Cornell – PTR- Volumes 1 e 2 – 8th Edition
- Inside the Java 2 – Virtual Machine Venners – McGrawHill
- Understanding Object-Oriented Programming with JAVA – Timothy Budd – Addison Wesley
- Head First Java, 2nd Edition by Kathy Sierra and Bert Bates
- Effective Java, 2nd Edition by Joshua Bloch
- Thinking in Java (4th Edition) by Bruce **Eckel**
- Java How to Program - 9th Edition by Paul Deitel and Harvey Deitel

# O que são Interfaces em Java?

# Interfaces são conceitos relacionados à Classes Abstratas ! ! !

# Classes Abstratas

- São classes que possuem pelo menos 1 método abstrato;
- Um método abstrato contém apenas a assinatura (signature) ou a declaração da função, não contém portanto código;
- Métodos abstratos são usados para servirem de base para outros métodos (overriding);
- Com classes abstratas, **não** se pode criar objetos diretamente a partir delas;

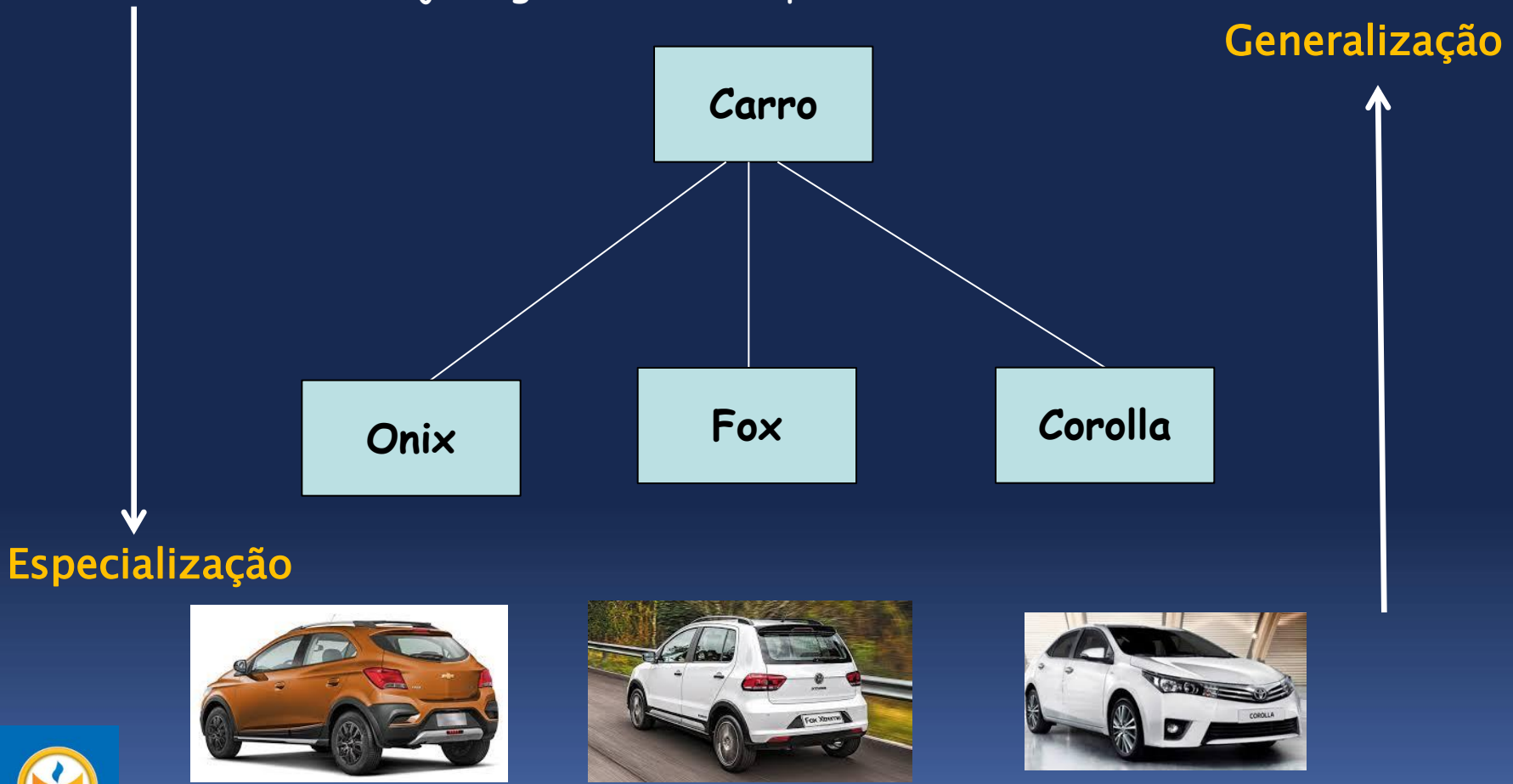
Mas, se não se pode criar objetos a partir de classes abstratas, então para que elas servem ?

# Classes Abstratas

- Classes Abstratas funcionam como modelos para classes de níveis inferiores;
- Auxiliam na organização do Projeto Orientado a Objetos;
- São utilizadas par se representar grupos de objetos que têm características comuns, mas que, em alguns detalhes específicos, possuem algumas diferenças;
- Classes abstratas estão diretamente relacionadas ao conceito de Polimorfismo em Java.

# Classes Abstratas

- Se formos à uma Loja de Veículos para comprar um veículo, não compramos um **carro**, mas sim compramos um **Fox**, um **Onix**, um Corolla, ou seja, algum modelo específico.



# Classes Abstratas

- Um dono de uma empresa não contrata um funcionário. Ele contrata uma secretária, um administrador, um engenheiro, ou alguém com alguma profissão em específico.

# Classes Abstratas

- Na verdade, em nosso mundo real, só existem objetos;
- Classes abstratas, na verdade, são **abstrações** que usamos para representar **agrupamentos** de objetos, ou para **classificá-los**;
- Assim, classes abstratas são **superclasses** que podem ser utilizadas como base (modelo) para se definir outras classes;
- Classes abstratas podem possuir métodos concretos (com implementação) ou métodos abstratos (sem implementação);
- possuem apenas assinatura dos métodos por uma razão bem simples: os métodos irão se comportar de forma diferente nas subclasses.

# Métodos Abstratos

- Métodos que **não** possuem código de implementação;
- São apenas declarados (**assinaturas**) e definidos com a keyword **abstract**;
- Classes abstratas possuem ao menos 1 método abstrato;
- Métodos abstratos possuem apenas assinatura dos métodos por uma razão bem simples: os métodos irão se comportar de forma diferente nas subclasses.

# Como se definem Classes Abstratas em Java ?

# Classes Abstratas

- São definidas com a keyword **abstract**.

# Classes Abstratas - Exemplo

```
abstract class Animal {
    int distanciaPercorrida = 0;

    public abstract void fazerBarulho();

    public void andar() {
        distanciaPercorrida++;
    }

    public void treinar() {
        andar();
        fazerBarulho();
    }
}
```

# Classes Abstratas - Exemplo

```
class Cachorro extends Animal {
    public void fazerBarulho() {
        System.out.println("Au-au!");
    }
}
```

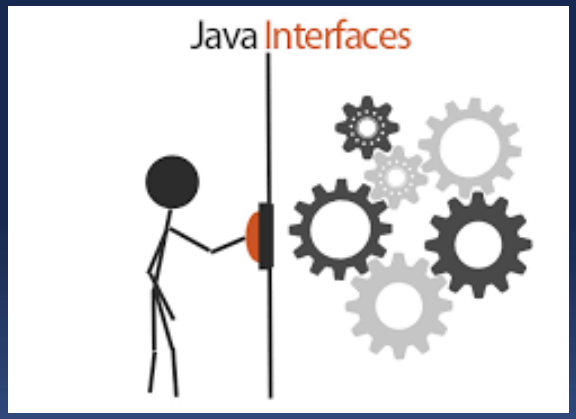
# Classes Abstratas - Exemplo

```
class Gato extends Animal {
    public void fazerBarulho() {
        System.out.println("Miau!");
    }
}
```

# Classes Abstratas - Exemplo

```
class Main {
    public static void main(String[] args) {
        Cachorro cao = new Cachorro();
        Gato gato = new Gato();
        cao.treinar();
        gato.treinar();
    }
}
```


# Interfaces



# Interfaces

- Interface abstrai o conceito de classe abstrata ao extremo;
- Imagine que você irá desenvolver uma aplicação no qual **duas equipes** irão desenvolver o software de forma **simultânea**;
- Cada equipe irá desenvolver seus códigos de forma **independente**.

Brasil

Contrato

India

# Interfaces

- No entanto, deverá haver um "**contrato**" entre as equipes de tal modo que haja interação entre os códigos;
- Este **contrato** é conhecido por **interface**.

# Interfaces

- **Interface** é uma forma de descrever o quê as classes devem fazer, **sem** especificar como elas devem fazê-lo;
- **Interfaces** empregam o conceito de **classe abstrata** ao **extremo**;
- É como se definisse uma classe abstrata no qual **todos** os **métodos** também sejam **abstratos**;
- Em **Java**, uma **interface** não é uma classe, mas um conjunto de **requisitos**, os quais devem ser implementados por alguma classe que **aceite** o **contrato**.

# Interface

- Em Java, uma interface é uma definição de tipo, semelhante à classe, que pode conter apenas **constantes** e assinatura de métodos (**protótipos**);
- Numa interface não há corpo de definição de **método**.
- **Não** podem ser instanciadas, podem somente serem implementadas por classes ou ainda estendidas em outras interfaces.

# Interface

- Todos os métodos de uma interface são automaticamente **public**;
- Por esta razão **não** há necessidade de incluir a keyword **public** quando estivermos declarando um método em uma interface;
- Tendo em vista que interfaces **não** são classes, **nunca** se pode usar o operador **new** para instanciar uma interface;
- Ou seja, **nunca** se instancia um objeto a partir de uma interface.

# Como se define interface em Java ?

# Exemplo – Interface

```
public interface XPTO {
      int func ( String a ); //contrato
}
```

- Isto significa que para qualquer classe que implementa a interface **XPTO** é requerido que se tenha a implementação do método **func** e este método deve ter um parâmetro **String** e retornar um **inteiro**.

# Interfaces

- Uma **interface** é essencialmente uma **coleção** de **constantes** e **métodos abstratos**;
- Para se fazer uso de uma **interface**, deve-se **implementar** a interface na classe;
- Ou seja, deve-se definir a classe que implementa a interface e escrever o código para cada método declarado na **interface**.

# Interfaces

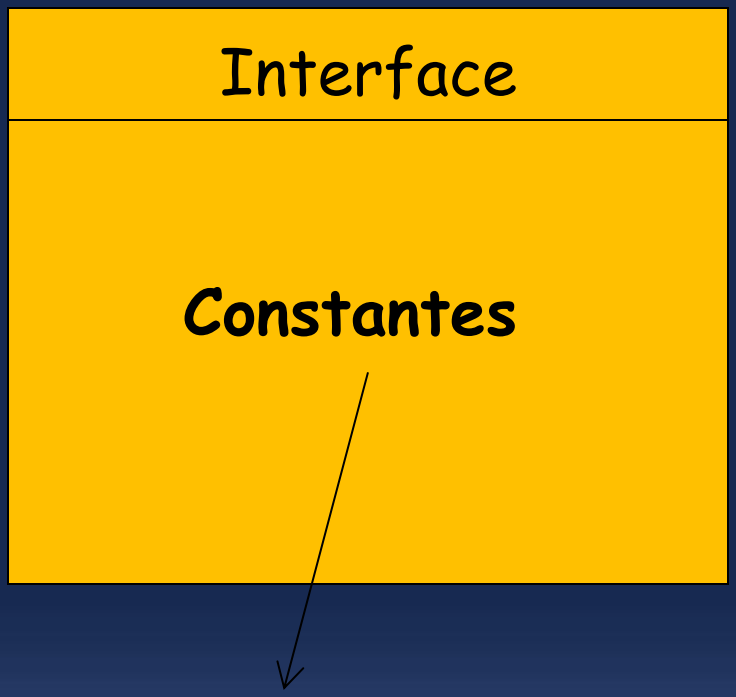
- Quando uma classe implementa uma **interface**, quaisquer constantes que foram definidas na interface são diretamente disponíveis na classe, como se fossem **herdados** de uma classe base.

# O que pode conter uma interface ?

# O que pode conter uma interface ?

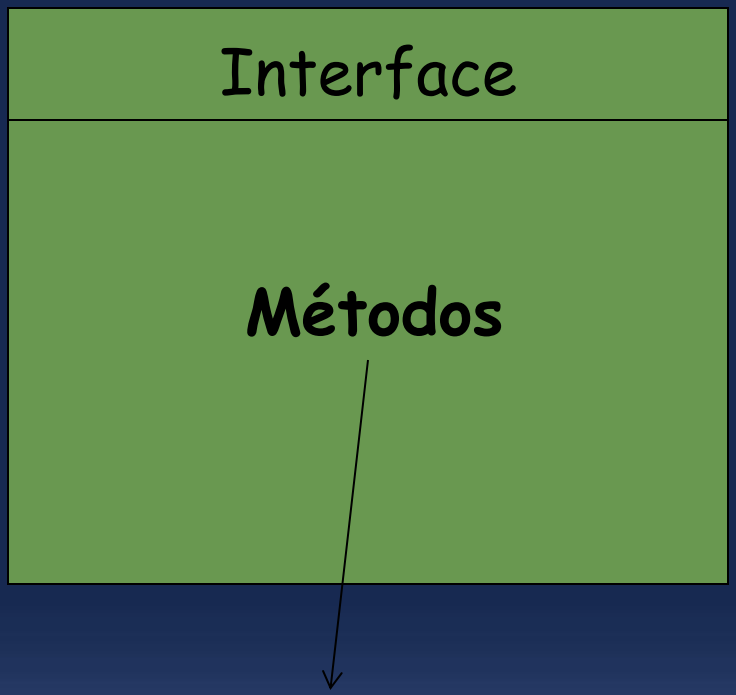
- Uma interface pode conter **constantes**, **métodos abstratos** ou **ambos**.

# Interface com constantes




São sempre **public**, **static** e **final** por default;

# Interface com métodos




São sempre **public** e **abstract** por defaul;

# Interface com métodos e constantes


São sempre **public** e **abstract** por **default**;

São sempre **public**, **static** e **final** por default;

# Como a interface trabalha ?

- Uma **interface** é definida como uma **classe**;
- Mas usa a **keyword** **interface** ao invés de **class**;
- O único atributo de acesso permitido em uma interface é **public**;
- Isto faz a interface acessível **fora** do package que a contém;
- Caso se omita o atributo de acesso public, a interface somente será acessível (visível) no **package** que a contém.

# Exemplo

```
public interface Shape {

        //implicitly public, static and final
        public String LABLE="Shape";

        //interface methods are implicitly abstract and public
        void draw();

        double getArea();
}
```

# Exemplo

```
public class Circle implements Shape {

        private double radius;

        public Circle(double r){
                this.radius = r;
        }

        @Override
        public void draw() {
                System.out.println("Drawing Circle");
        }

        @Override
        public double getArea(){
                return Math.PI*this.radius*this.radius;
        }

        public double getRadius(){
                return this.radius;
        }
}
```