

Avançando em Java com **Polimorfismo (Interfaces)**

Prof.: Hugo Barros



Interfaces

- Como já foi dito, java não permite que uma classe herde recurso de mais de uma classe.
- Herança múltipla é um recurso muito poderoso, o não uso dessa funcionalidade limita muito várias linguagens
- E o java ficou de fora dessa ?



Interfaces

- Interface é uma estrutura que representa uma classe abstrata "pura" em Java
 - Não tem construtor
 - Não tem atributos
 - Todos os métodos são abstratos
 - Não é declarada como class mas sim com interface.
- Interfaces java servem para fornecer **polimorfismo sem herança**
 - Uma classe pode “herdar” a interface de várias interfaces java.
 - Interfaces, portanto, oferecem um tipo de herança múltipla
- Desta forma, além de estender alguma superclasse, a classe em desenvolvimento pode implementar várias interfaces.



Interfaces

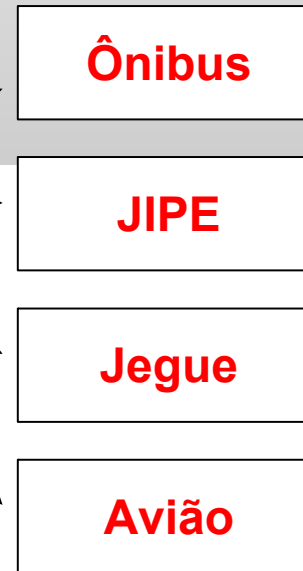
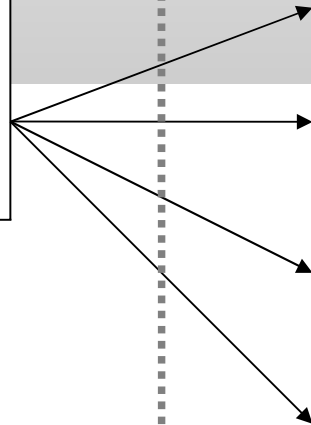
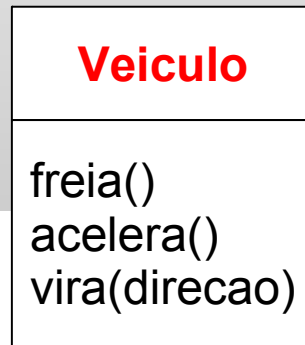
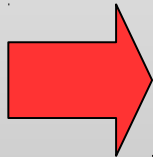
- Como funciona ?
 - Um objeto que faz papel de interface serve de **intermediário** fixo entre o programa principal e os objetos que irão executar as mensagens recebidas.
 - O programa principal não precisa saber da existência dos outros objetos
 - Objetos podem ser **substituídos** sem que programas que usam a interface sejam afetados.



Interfaces - Objetos Substituíveis

O usuário do objeto exerga somente esta interface.

freia()
acelera()
vira(l)



Uma interface, múltiplas implementações !

Subclasses de veiculo !
(Herdam todos os métodos)

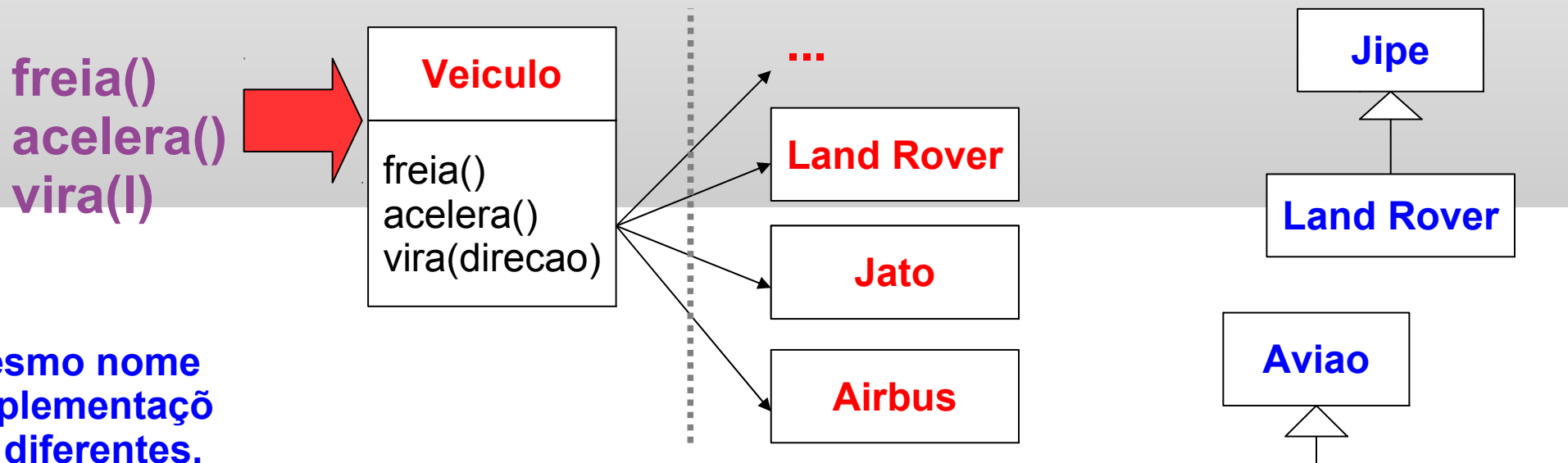
Por exemplo: objeto do tipo **Manobrista** sabe usar comandos básicos para controlar **Veiculo** (não interessa a ele saber como cada **Veiculo** diferente vai acelerar, frear ou mudar de direção). Se outro objeto tiver a **mesma interface**, **Manobrista** saberá usá-lo

Usuário de veiculo ignora a existência desses objetos substituíveis.



Classes Extensíveis

- Novos Objetos podem ser criados por classes que não previam sua existência.
 - Garantia que métodos da interface existem nas classes novas.



Mesmo nome
implementaçõ
es diferentes.

```
Veiculo v1 = new Aviao();  
Veiculo v2 = new Airbus();  
Veiculo v3 = new LandRover();  
  
v1.acelera(); //acelera Aviao  
v2.acelera(); //acelera Airbus  
v3.acelera(); //acelera LandRover
```



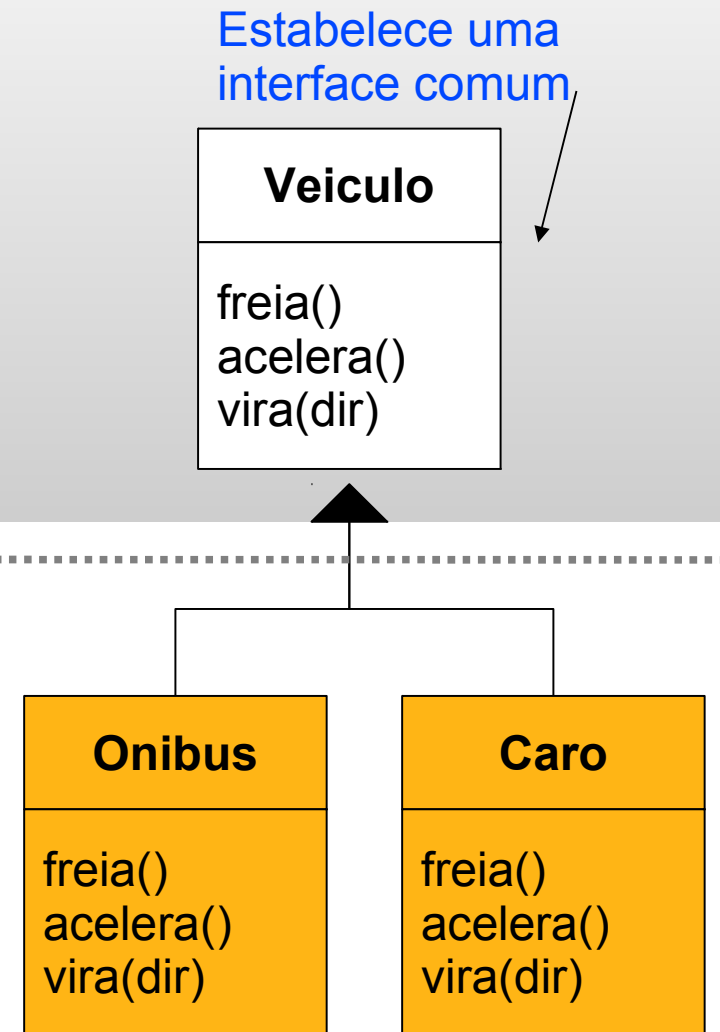
Interfaces Vs. Implementação

- Polimorfismo permite **separar a interface da implementação**
- A classe base define a interface comum
 - Não precisa dizer **como** isso vai ser feito

Não diz: eu sei como frear um carro ou um Ônibus

- Diz apenas que os métodos existem, que eles retornam determinados tipos de dados e requerem certos parâmetros

Diz: Veiculo pode acelerar, frear e virar para uma direção, mas a direção deve ser fornecida

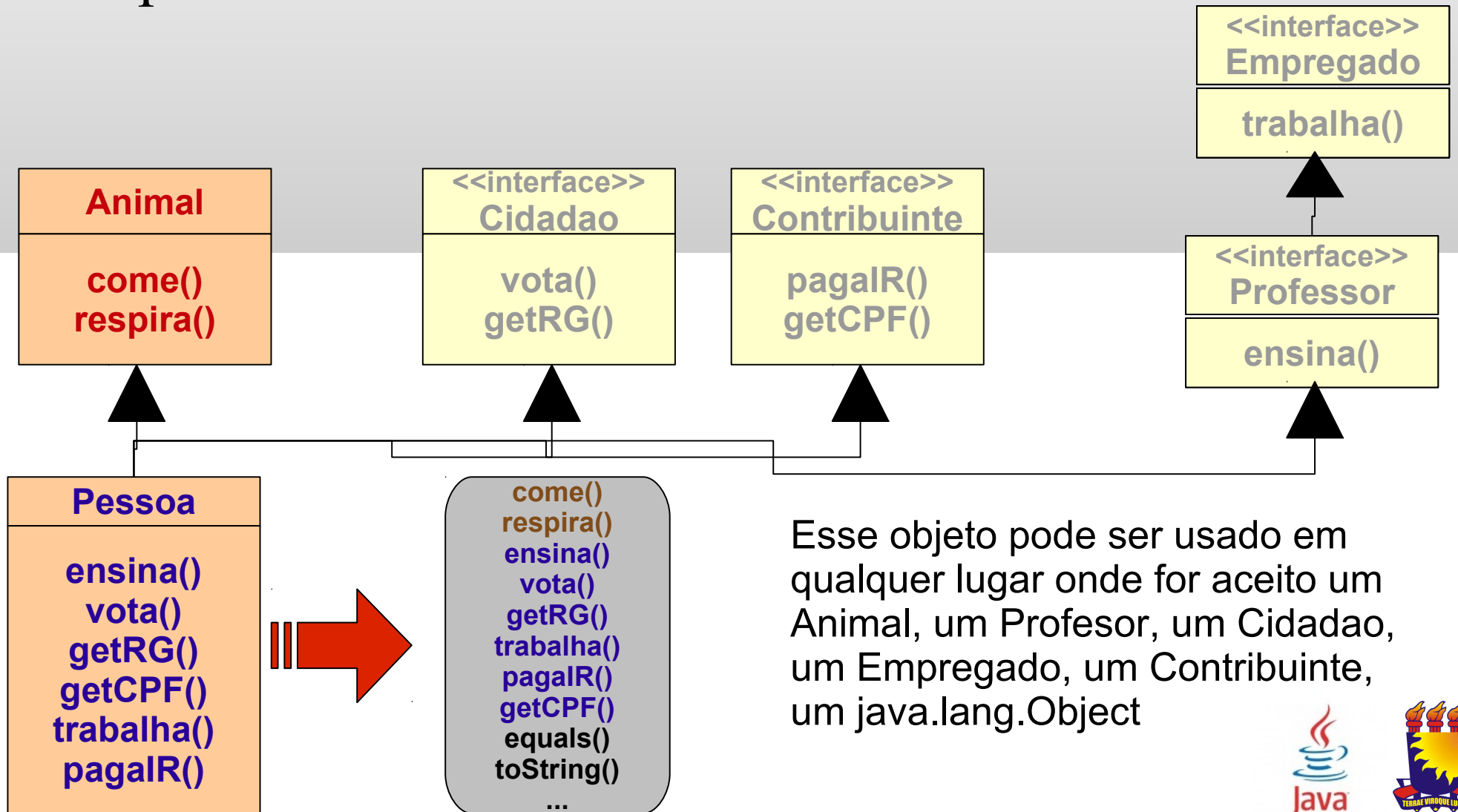


Implementações
da Interface
(dizem como fazer)



Herança Múltipla

- A classe resultante combina todas as interfaces, mas só possui uma “**extensão**”: a da classe mãe.



Exemplo

```
interface Empregado{  
    void trabalha();  
}
```

```
interface Cidadao{  
    void vota();  
    Int getRg();  
}
```

```
interface Professor  
extends Empregado{  
    void ensina();  
}
```

```
interface Contribuinte{  
    boolean pagaIR();  
    long getCPF();  
}
```

- Todos os métodos são implicitamente:
 - `public`
 - `abstract`
- Interface pode ser declarada `public`.



Exemplo

```
public class Pessoa
    extends Animal
    implements Professor, Cidadao, Contribuinte{

    public void ensina() { /*ensinar*/ }
    public void vota() { /*votar*/ }
    public int getRG() {return 12345;}
    public void trabalha() { /*trabalhar*/ }
    public boolean pagaIR() {return false;}
    public long getCPF() {return 1234567890;}
}
```

- A palavra **implements** declara interfaces implementadas
 - Exige que **cada um dos métodos** de cada interface sejam de fato implementados
 - Se alguma implementação estiver faltando, classe só compila se for declarada **abstract**.



Conclusão

- Use interfaces sempre que possível
 - Seu código será mais **reutilizável!**
- Cuidado!



- Não é possível acrescentar métodos a uma interface depois que ela já estiver em uso (as classes que a implementam **não compilarão mais !**)
- Quando a evolução for mais importante que a flexibilidade oferecida pelas interfaces, deve-se usar classes abstratas.

Atividade

Acrescente 3(três) interfaces ao conjunto de classes desenvolvidas na aula anterior e modifique ou duplique agora o método main de forma que agora utilize-as.

