

PARADIGMAS DE PROGRAMAÇÃO

2023/2024

P.PORTO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

Aula 09

1. Palavras Reservadas
2. Herança
3. Interfaces
4. Palavras Reservadas Usadas
5. Links Úteis



Palavras Reservadas

<code>abstract</code>	<code>continue</code>	<code>for</code>	<code>new</code>	<code>switch</code>
<code>assert***</code>	<code>default</code>	<code>goto*</code>	<code>package</code>	<code>synchronized</code>
<code>boolean</code>	<code>do</code>	<code>if</code>	<code>private</code>	<code>this</code>
<code>break</code>	<code>double</code>	<code>implements</code>	<code>protected</code>	<code>throw</code>
<code>byte</code>	<code>else</code>	<code>import</code>	<code>public</code>	<code>throws</code>
<code>case</code>	<code>enum****</code>	<code>instanceof</code>	<code>return</code>	<code>transient</code>
<code>catch</code>	<code>extends</code>	<code>int</code>	<code>short</code>	<code>try</code>
<code>char</code>	<code>final</code>	<code>interface</code>	<code>static</code>	<code>void</code>
<code>class</code>	<code>finally</code>	<code>long</code>	<code>strictfp**</code>	<code>volatile</code>
<code>const*</code>	<code>float</code>	<code>native</code>	<code>super</code>	<code>while</code>

* not used

** added in 1.2

*** added in 1.4

**** added in 5.0

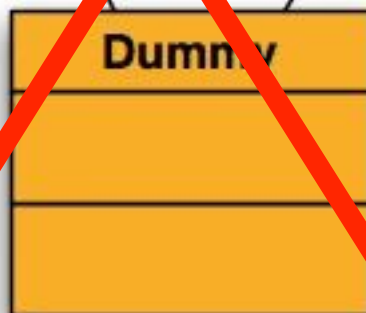
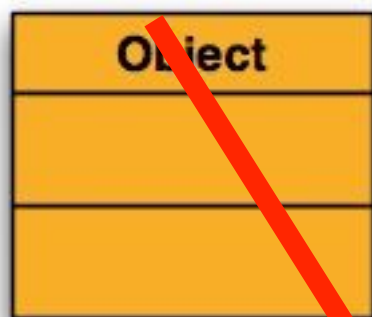


Herança

▪ Conceito

- Java **não permite** herança múltipla

Porquê?



```
public class Dummy  
(extends Object, String){  
    ...  
}
```

Em Java **não** é possível herança múltipla!



- Supondo que se quer modelar uma **criança** a qual herda (comportamentos) do **pai** e da **mãe**

Como o fazer em Java?



▪ É do conhecimento geral que:

➔ Saber **pregar um prego**
e **fritar um ovo...**

...dá muito jeito!



- É também do conhecimento geral que por norma:

- ➔ O comportamento "**martelar**" é herdado do pai

- ➔ O comportamento "**fritar**" é herdado da mãe



Pseudo-Solução

```
class Pai {  
    DedoPartido martelar(int pregos) { ... }  
}  
  
class Mae {  
    Omolete fritar(int ovos) { ... }  
}  
  
class Crianca extends Pai, Mae { ... }
```

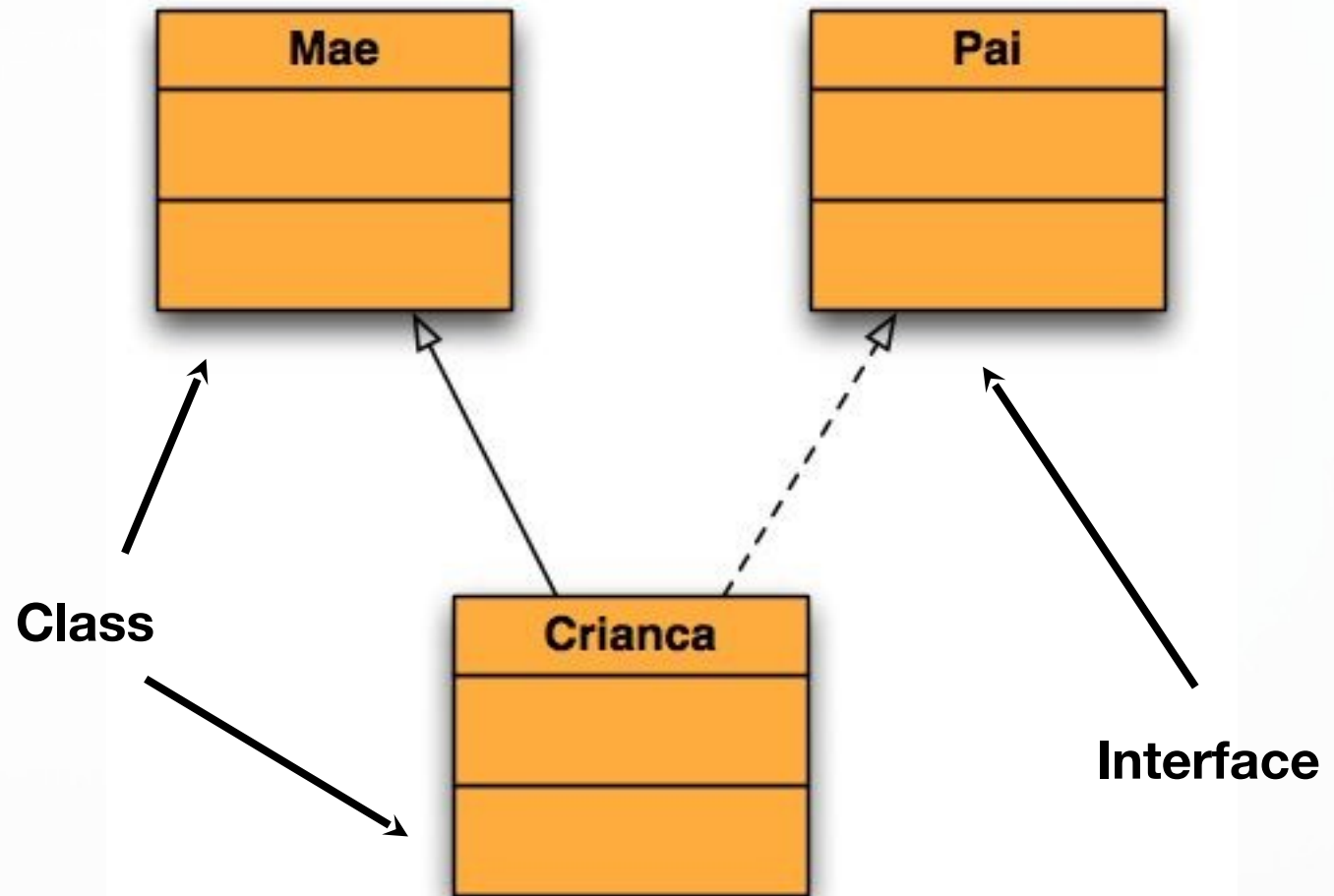



Solução

- Recorrer a ***interfaces***!
 - Não são interfaces gráficas
- Uma ***interface*** *Java* é uma entidade próxima de uma classe 100% abstracta

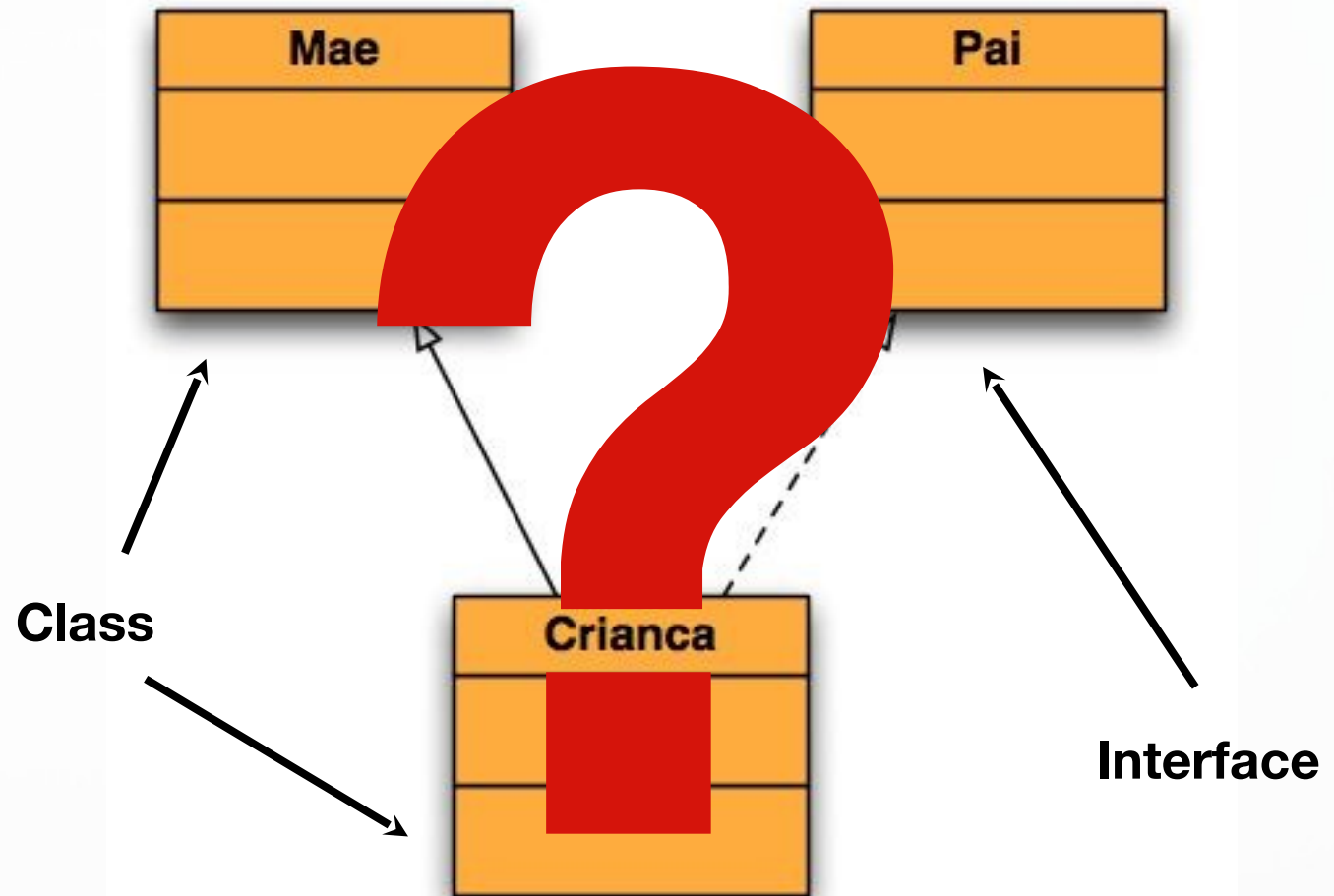


❖ Possível Solução



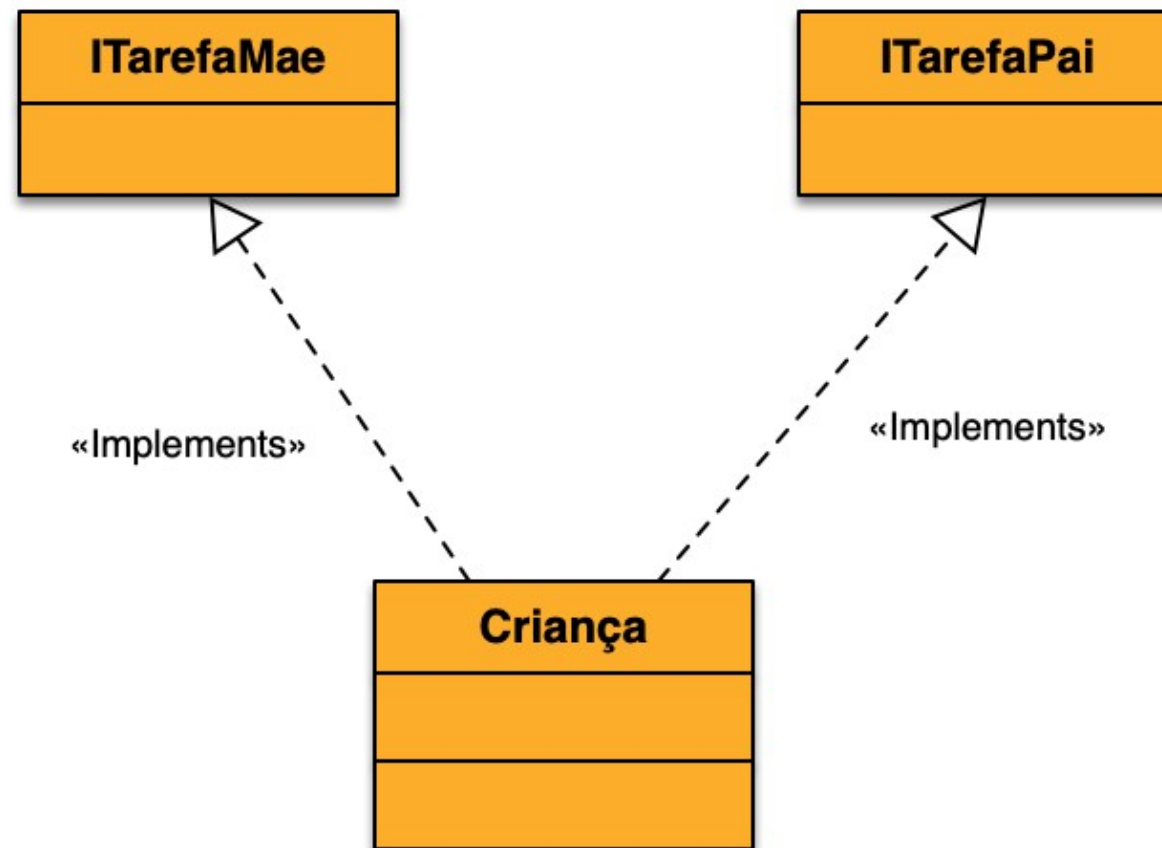


❖ Possível Solução





❖ Solução





Interfaces

▪ Conceito

- Uma interface é um **contrato!**



❖ **Vantagens em recorrer a interfaces:**

- Quem implementa a interface fá-lo de acordo com o que foi contratado
- Quem usa a interface fá-lo de acordo com o que contratou



❖ **Vantagens em recorrer a interfaces:**

- Separa a implementação do código cliente escondendo os detalhes de implementação
- A implementação pode ser refeita sem que o código cliente se aperceba



- Exemplo de como definir uma interface:

```
public interface HelloWorld {  
    void sayHelloTo(String name);  
}
```




```
public interface HelloWorld {  
  
    void sayHelloTo(String name);  
  
}
```

- ❖ Um método de uma interface é `public abstract`, mais uma vez... não é obrigatório explicitá-lo!
- ❖ A assinatura de cada método acaba com ponto e vírgula!



- Exemplo de como definir uma interface:

```
public class Test implements HelloWorld {  
  
    public void sayHelloTo(String name) {  
        System.out.println("Hello" + name + "!");  
    }  
  
}
```



```
public class Test implements HelloWorld {  
  
    public void sayHelloTo(String name) {  
        System.out.println("Hello" + name + "!");  
    }  
  
}
```

- ❖ A class Test tem que implementar o método sayHelloTo, caso contrário terá de ser **abstracta**!



- ❖ Por norma, recorrendo a classes abstractas e a interfaces para especificar e prototipar um módulo de software é vantajoso.
- ❖ O módulo torna-se, assim, mais flexível, ou seja, mais polimórfico!



- ❖ As interfaces são um mecanismo que as linguagens OO (e.g., *Java*, *C#*, ...) disponibilizam, contudo são (**infelizmente**) pouco usada pelos desenvolvedores de *software*...
- ❖ **Vamos mudar isto!**



Palavras Reservadas Usadas

<code>abstract</code>	<code>continue</code>	<code>for</code>	<code>new</code>	<code>switch</code>
<code>assert***</code>	<code>default</code>	<code>goto*</code>	<code>package</code>	<code>synchronized</code>
<code>boolean</code>	<code>do</code>	<code>if</code>	<code>private</code>	<code>this</code>
<code>break</code>	<code>double</code>	<code>implements</code>	<code>protected</code>	<code>throw</code>
<code>byte</code>	<code>else</code>	<code>import</code>	<code>public</code>	<code>throws</code>
<code>case</code>	<code>enum****</code>	<code>instanceof</code>	<code>return</code>	<code>transient</code>
<code>catch</code>	<code>extends</code>	<code>int</code>	<code>short</code>	<code>try</code>
<code>char</code>	<code>final</code>	<code>interface</code>	<code>static</code>	<code>void</code>
<code>class</code>	<code>finally</code>	<code>long</code>	<code>strictfp**</code>	<code>volatile</code>
<code>const*</code>	<code>float</code>	<code>native</code>	<code>super</code>	<code>while</code>

* not used

** added in 1.2

*** added in 1.4

**** added in 5.0



Links Úteis

- ❖ <http://docs.oracle.com/javase/tutorial/java/concepts/inheritance.html>
- ❖ <http://docs.oracle.com/javase/tutorial/java/landl/abstract.html>
- ❖ <http://docs.oracle.com/javase/tutorial/java/landl/final.html>