

Programação Orientada a Objetos

PROFESSOR ISAAC

Interfaces



Interfaces

- **Interface** em Java é o mecanismo pelo qual o programador pode definir um conjunto de operações sem se preocupar com a sua implementação
- A interface indica um modelo de comportamento para outras classes
- **Uma interface define um conjunto de métodos que será implementado por uma classe.**
- As interfaces são semelhantes às classes abstratas com métodos abstratos
- A sintaxe da interface é semelhante a classe abstrata, mas utiliza a palavra-chave **interface** no lugar de ***abstract class***.

Interfaces

- ❑ A classe que implementa a interface utiliza a palavra-chave ***implements*** no lugar de ***extends***.
- ❑ A interface **obriga** um determinado grupo de classes a ter métodos ou propriedades em comum
- ❑ Pode-se dizer, a grosso modo, que uma interface é um contrato que quando assumido por uma classe deve ser **implementado**.

Interfaces

- ❑ Costuma-se dizer que uma interface permite estabelecer um “contrato” entre as classes; funciona de maneira bastante similar a classes abstratas, porém não permite implementação de nenhum método.

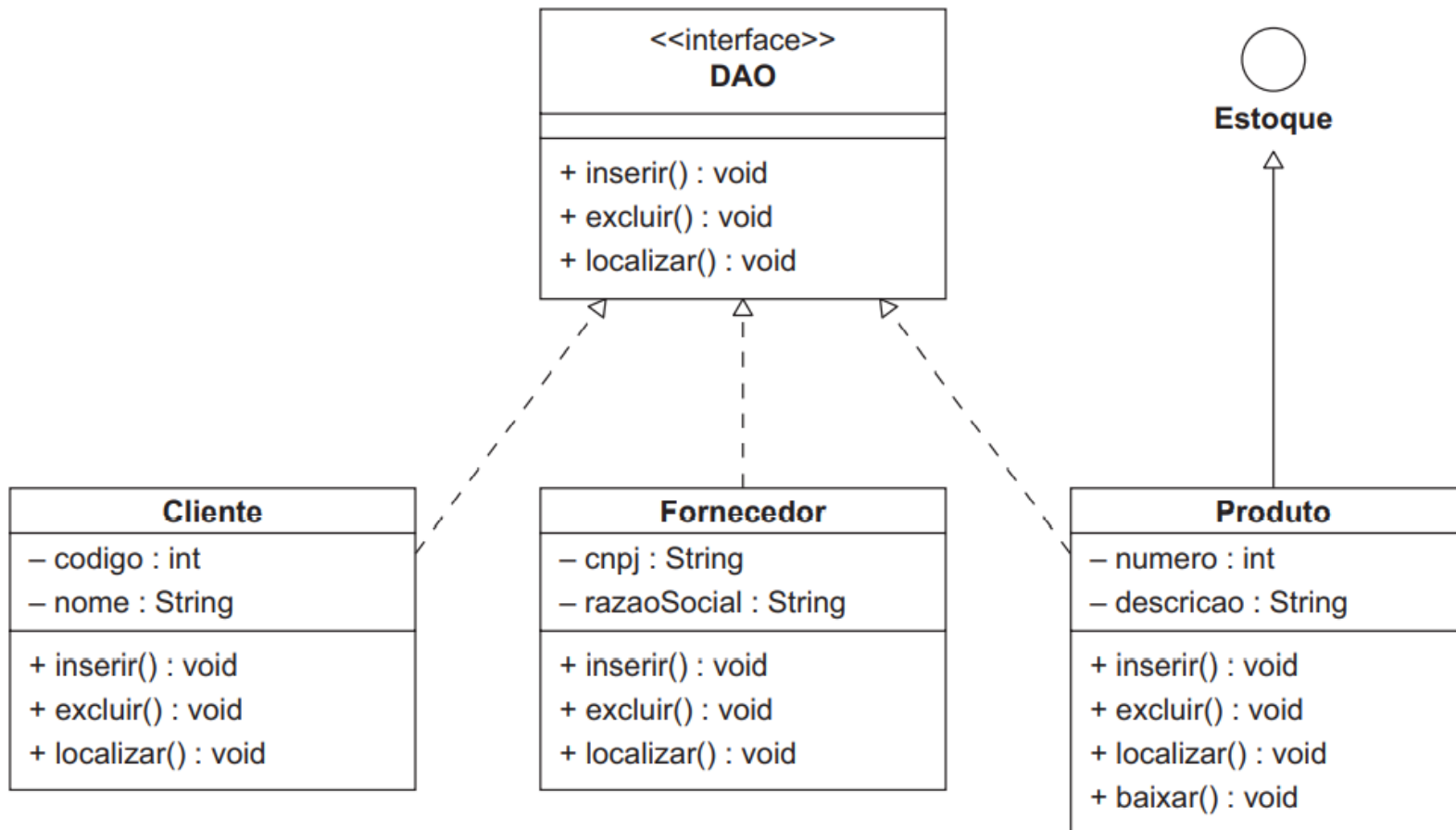
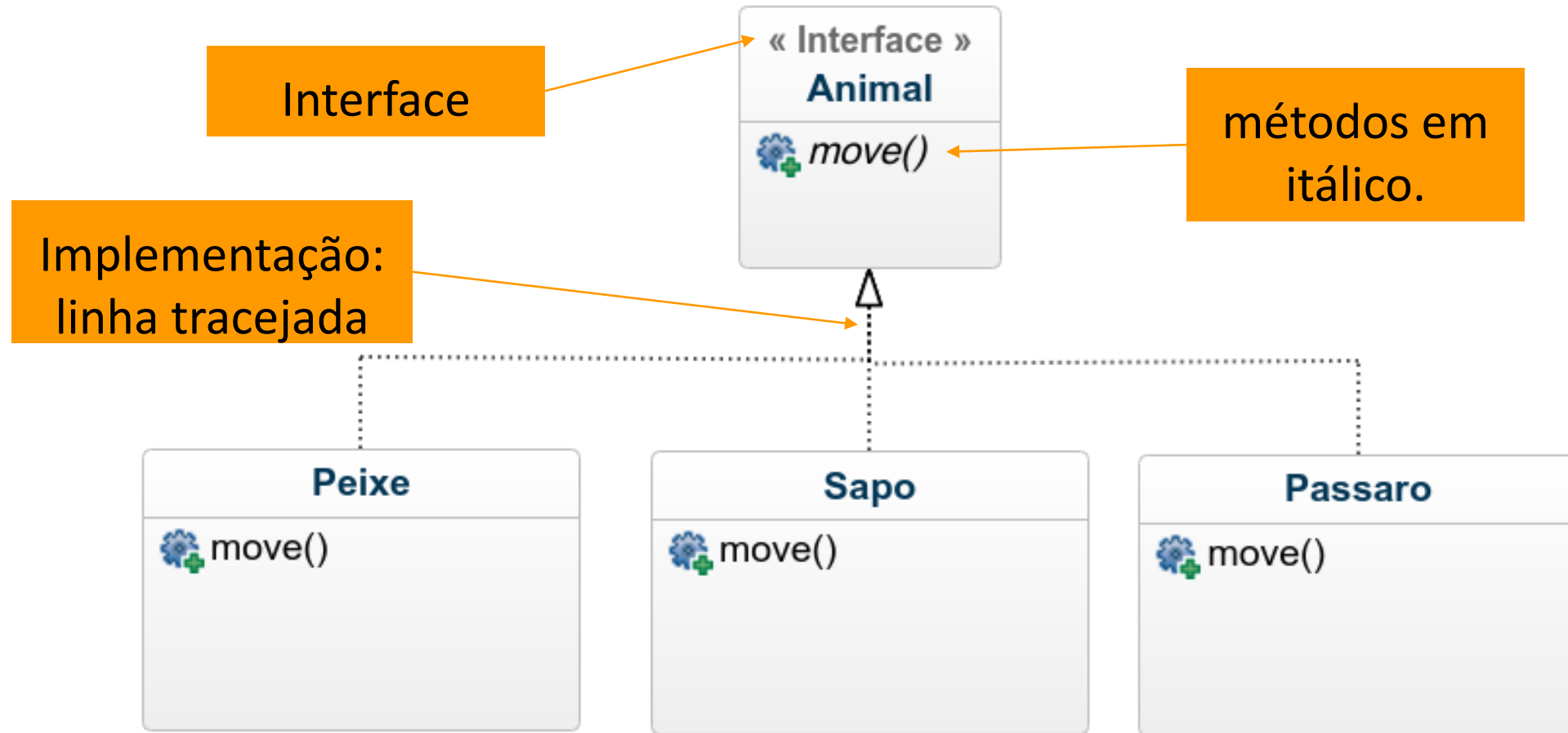


Figura 7.11 – As interfaces DAO e Estoque.

UML - Diagrama de Classes

Interface



Interfaces

- No nosso exemplo, **Animal** poderia ser uma **interface**:

```
1 interface Animal {
2     public void move();
3 }
4
5 class Peixe implements Animal {
6     public void move() {
7         System.out.println( "Nada" );
8     }
9 }
10
11 class Sapo implements Animal {
12     public void move() {
13         System.out.println( "Pula" );
14     }
15 }
16
17 class Passaro implements Animal {
18     public void move() {
19         System.out.println( "Voa" );
20     }
21 }
```

```
1 public class MainAnimalPoli {
2     public static void main( String[] args ) {
3         Animal peixe = new Peixe();
4         Animal sapo = new Sapo();
5         Animal passaro = new Passaro();
6
7         peixe.move();
8         sapo.move();
9         passaro.move();
10    }
11 }
```

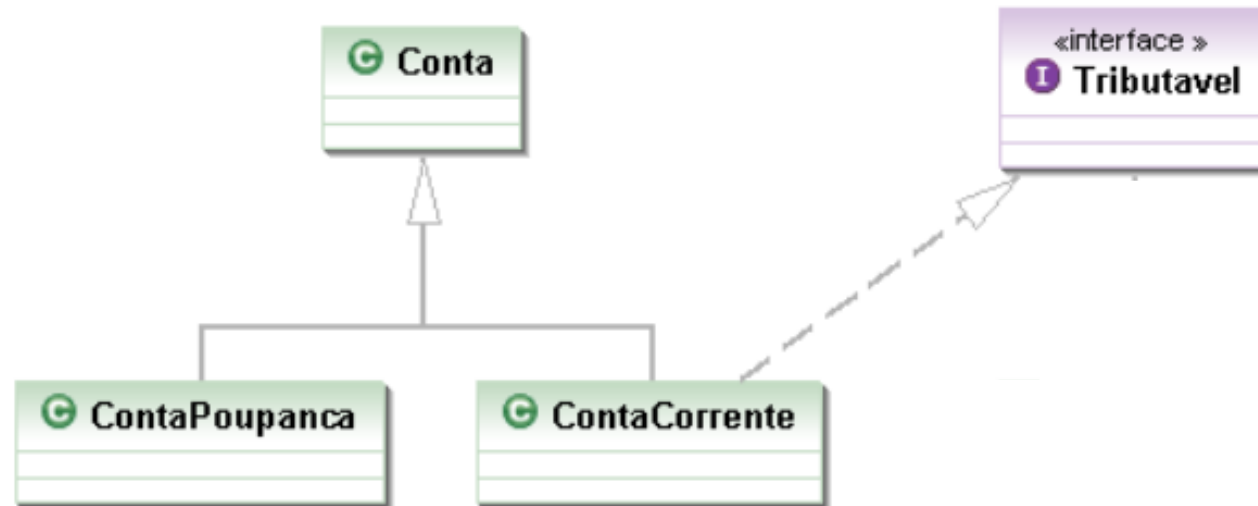
Combinando Interface com Herança

- Uma **classe** pode **herdar** comportamentos de outra classe enquanto também implementa uma **interface**!

```
public class NomeDaClasse extends Superclasse implements Interface{  
    // corpo da classe  
}
```


Combinando Interface com Herança

- Exemplo: Em um banco precisamos tributar alguns bens e outros não: ContaPoupanca não é tributável, já ContaCorrente precisa pagar 1% do saldo em tributos. Para isso, vamos criar a interface *Tributavel*



Combinando Interface com Herança

- Exemplo:

"todos que quiserem ser tributável precisam saber retornar o valor do imposto, devolvendo um double".

```
public interface Tributavel {  
    public double getValorImposto();  
}
```

```
public class Conta {  
    private int numero;  
    private String dono;  
    private double saldo;  
    private double limite;  
  
    public double getSaldo(){  
        return saldo;  
    }  
}
```

```
public class ContaCorrente extends Conta implements Tributavel {  
    public double getValorImposto() {  
        return this.getSaldo() * 0.01;  
    }  
}
```

Por que usar interface?

- Interfaces são muito utilizadas como uma **forma de obrigar** o programador a seguir o padrão do projeto
- O programador é obrigado a implementar os métodos da interface em sua classe, sempre seguindo o padrão
- Interfaces têm a vantagem de não acoplar as classes; ao contrário da herança que traz muito acoplamento (o que pode resultar em quebra do encapsulamento)

Diferenças entre Interface e Classe Abstrata

Métodos

- **Classe abstrata:** podem existir métodos abstratos e não abstratos, públicos, protegidos e privados.
- **Interface:** os métodos são implicitamente abstratos e somente públicos.

Variáveis

- **Classe abstrata:** podem ser definidas variáveis de instância e variáveis de leitura (*final*).
- **Interface:** as variáveis são sempre *public static final*

Quando usar interfaces ou classes abstratas?

Classes Abstratas:

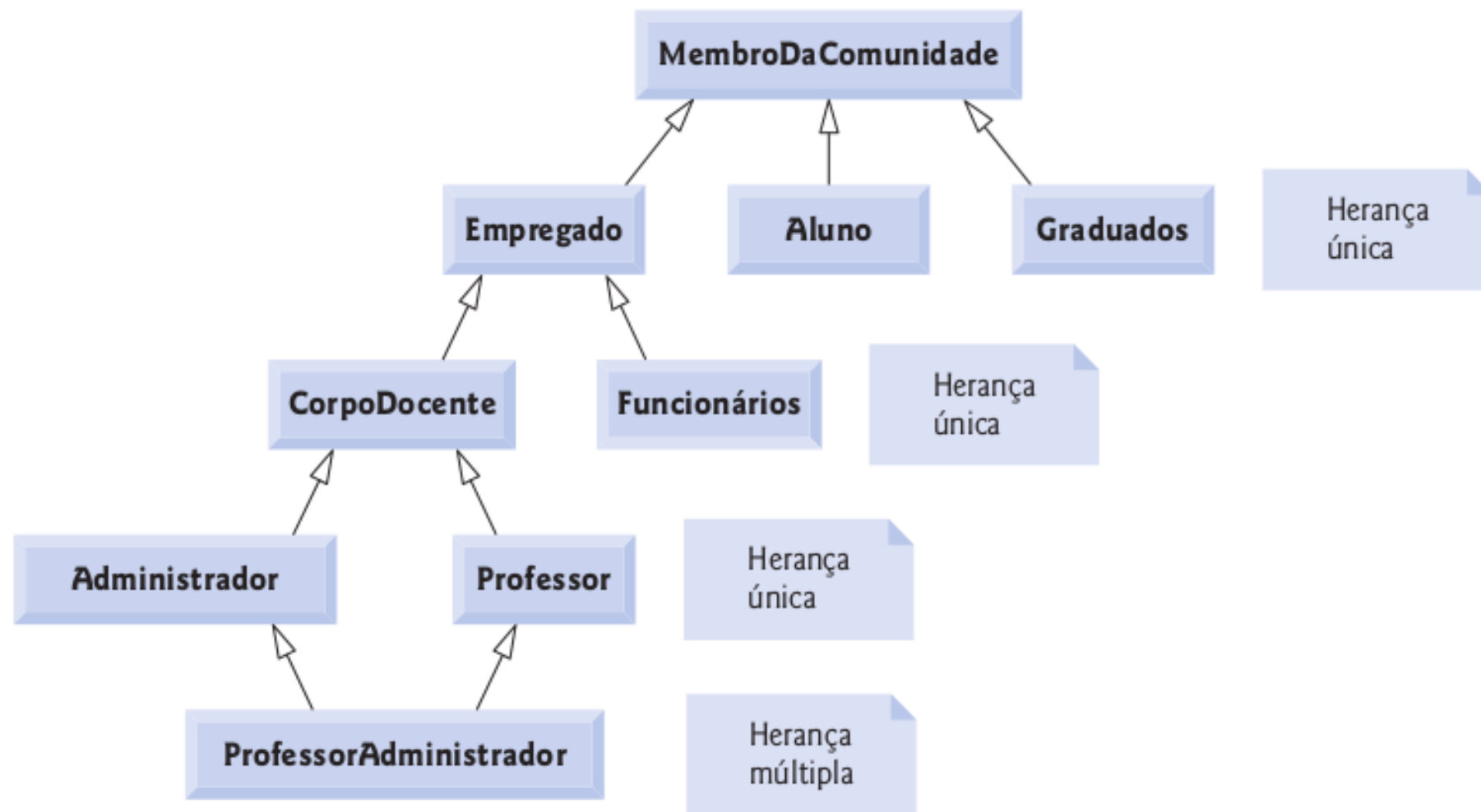
- Você deseja compartilhar código entre várias classes **estritamente relacionadas**.
- Você espera que as classes que estendem sua classe abstrata tenham **muitos métodos ou campos comuns**, ou exigem modificadores de **acesso que não sejam públicos** (como protegidos e privados).
- Você deseja declarar **campos não estáticos ou não finais**. Isso permite que você defina métodos que podem acessar e modificar o estado do objeto ao qual eles pertencem.

Quando usar interfaces ou classes abstratas?

Interfaces:

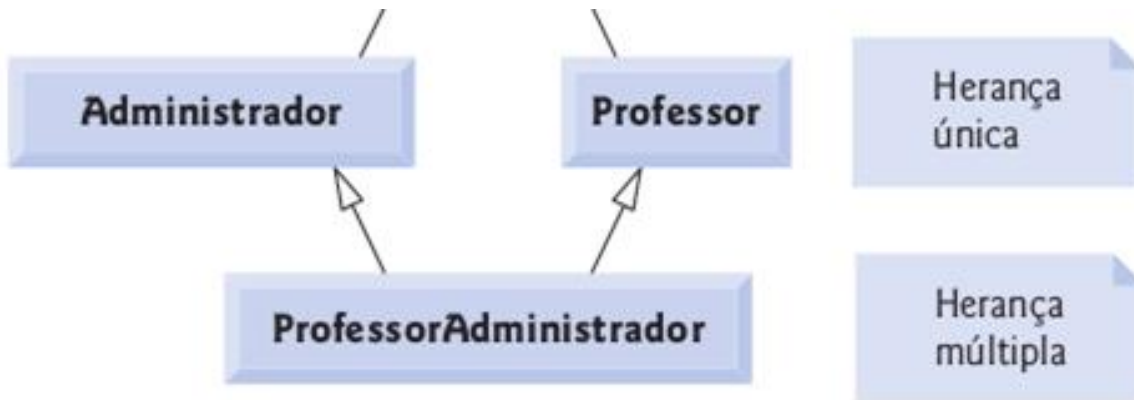
- Você espera que classes não relacionadas implementem sua interface
- Você deseja especificar o comportamento de um determinado tipo de dados, mas não se preocupa com quem implementa seu comportamento
- Você deseja aproveitar a *herança múltipla?!*

Herança Múltipla



Herança Múltipla - Problema

- A herança múltipla pode ser complexa e propensa a erros;
- Pode causar **ambiguidade!**
 - Membros das superclasses podem ter o mesmo nome!



- Tanto a classe *Administrador* quanto a *Professor* podem ter o método *getNome()*, por exemplo.
- A classe *ProfessorAdministrador* vai herdar qual dos dois métodos *getNome()* ?!

Interface

Herança Múltipla no Java?!

```
1 interface Andar
2 {
3     default void andar(){
4         System.out.println("Andando!!");
5     }
6 }
7
8 interface Engatinhar
9 {
10     default void engatinhar(){
11         System.out.println("Engatinhando!!");
12     }
13 }
14
15 class Animal implements Andar, Engatinhar
16 {
17     public static void main(String[] args)
18     {
19         Animal self = new Animal();
20
21         self.andar();
22         self.engatinhar();
23     }
24 }
```

- A partir do Java 8, as interfaces também podem ter comportamentos.
- Isto é possível graças a palavra-chave **default**, utilizada no método
- Então, se uma classe implementar duas interfaces e ambas definirem métodos padrão (*default*), essa nova classe está, essencialmente, herdando comportamentos.

Mas, com interface, os problemas da herança múltipla continuam existindo?

```
1 interface Andar
2 {
3     default void mover(){
4         System.out.println("Andando!!");
5     }
6 }
7
8 interface Engatinhar
9 {
10     default void mover(){
11         System.out.println("Engatinhando!!");
12     }
13 }
14
15 class Animal implements Andar, Engatinhar
16 {
17     public static void main(String[] args)
18     {
19         Animal self = new Animal();
20
21         self.mover();
22     }
23 }
```


```
multiple_inher_problem.java:15: error: class Animal inherits unrelated defaults
for mover() from types Andar and Engatinhar
class Animal implements Andar, Engatinhar
^
1 error
```

Interface

Herança Múltipla no Java?!

Para resolver o conflito, a classe deve decidir qual método *mover()* deseja invocar e então chamar usando a **referência da interface** e a palavra-chave **super**.

```
1 interface Andar
2 {
3     default void mover(){
4         System.out.println("Andando!!");
5     }
6 }
7
8 interface Engatinhar
9 {
10    default void mover(){
11        System.out.println("Engatinhando!!");
12    }
13 }
14
15 class Animal implements Andar, Engatinhar
16 {
17     public void mover(){
18         Andar.super.mover();
19     }
20 }
21
22
23 class Teste{
24     public static void main(String[] args)
25     {
26         Animal a = new Animal();
27         a.mover();
28     }
29 }
```



Exercício 1

Implemente o Diagrama de Classes ao lado. Crie também uma classe para testar o funcionamento de objetos da classe Morcego.

