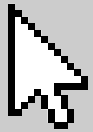
PUC Minas

Programação Modular



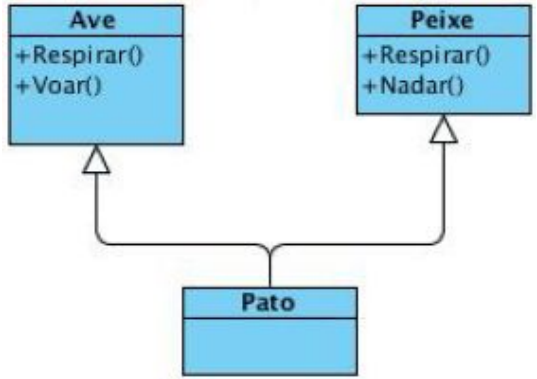
PUC Minas
Bacharelado em Engenharia de Software
Prof. Daniel Kansaon
slides adaptados do Prof. Danilo Boechat

Herança Múltipla

Herança Múltipla

Capacidade de uma classe herdar de duas ou mais superclasses

Aplicação: combinação das características de várias classes na definição de uma nova classe



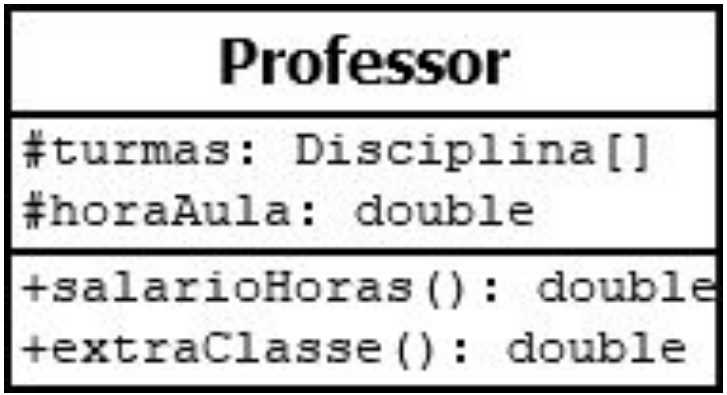
Herança Múltipla - Exemplos

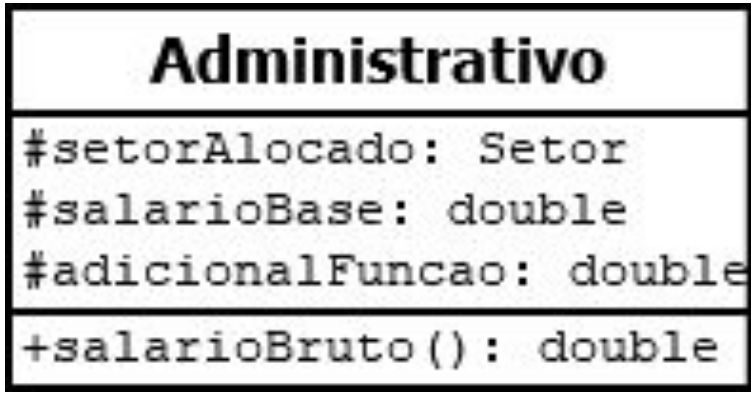
- Um smartphone é um eletrônico e um recarregável
- Um drone militar é um veículo aéreo não tripulado e uma arma de guerra
- Um líder de projeto é um programador e um gerente
- Um carro híbrido é um veículo a combustão e um veículo elétrico
- Um professor pesquisador é um professor e um cientista

Herança Múltipla - Exemplo

Um **professor** ministra aulas para diversas turmas e recebe um salário baseado na sua carga horária

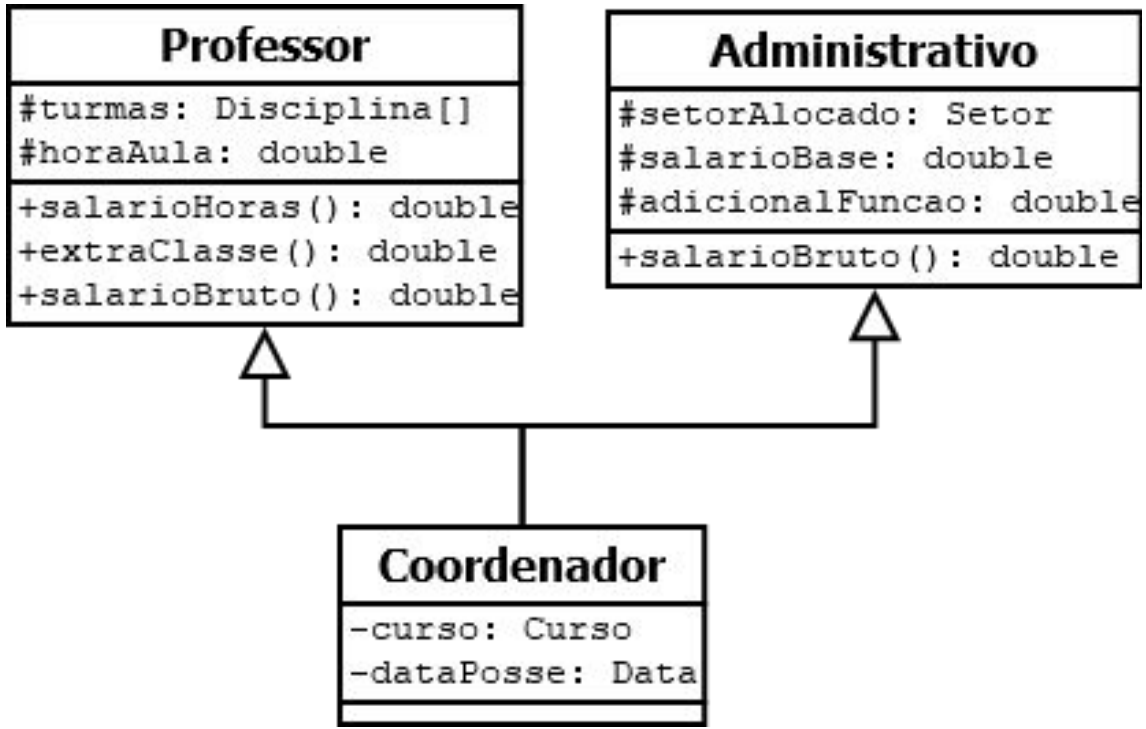
Um **funcionário administrativo** tem um salário base fixo e pode receber adicionais por função





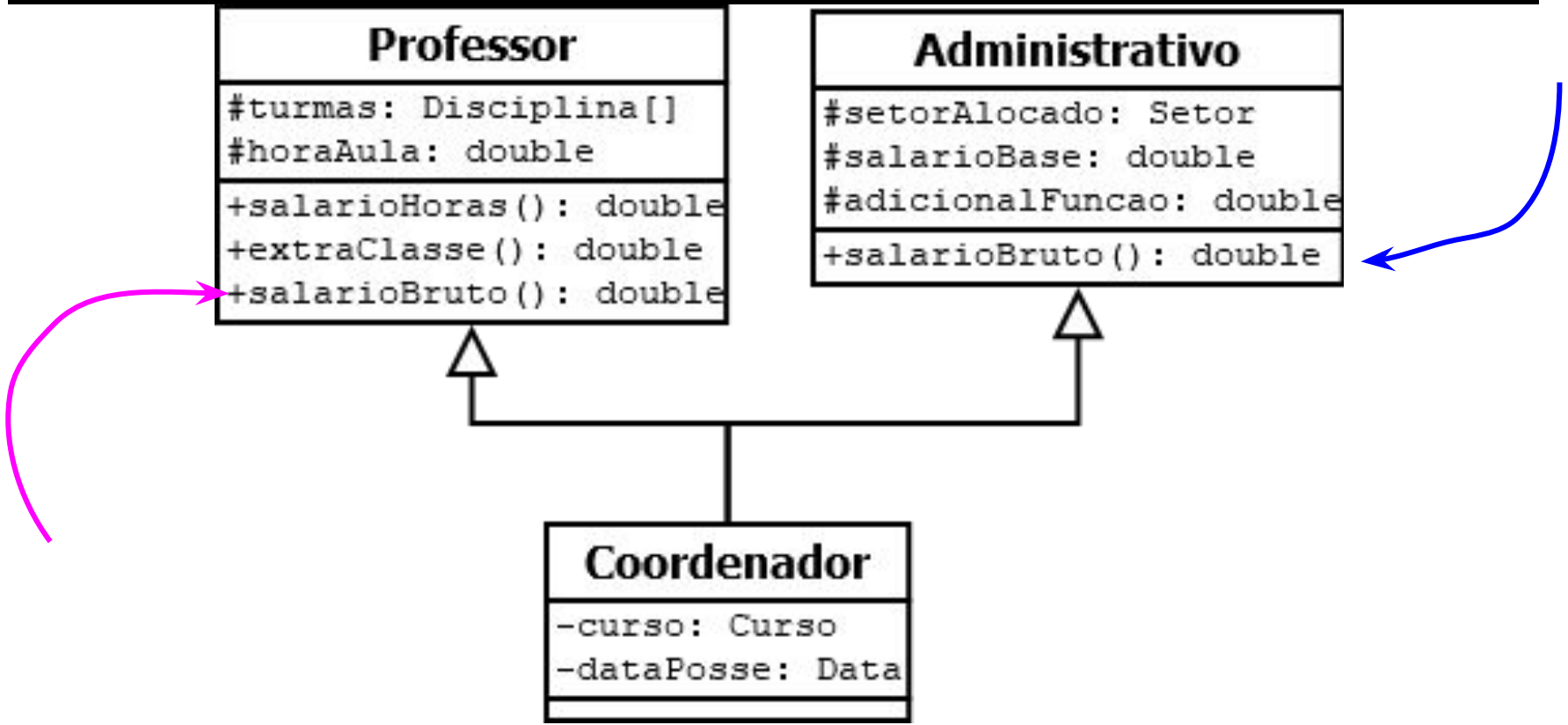
Herança Múltipla - Exemplo

Um **Coordenador** de curso dá aulas e tem funções administrativas

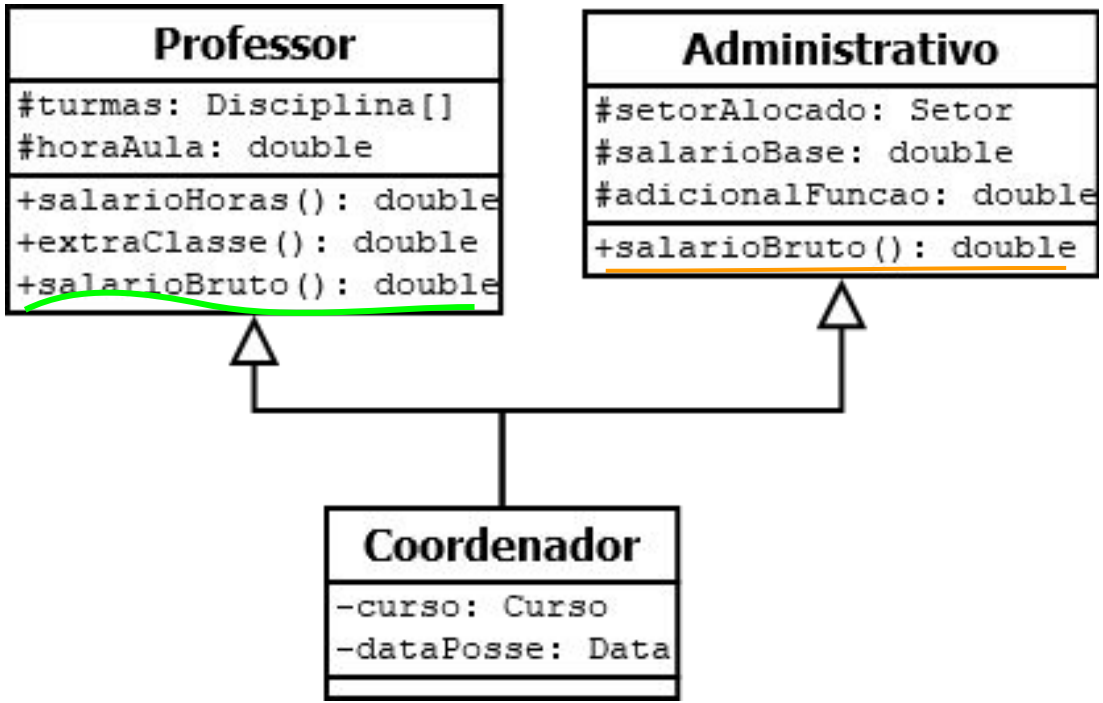


Qual o problema?

Herança Múltipla - Ambiguidades



Herança Múltipla - Ambiguidades



Se um objeto da classe **Coordenador** chamar

salarioBruto()

qual dos métodos é executado?

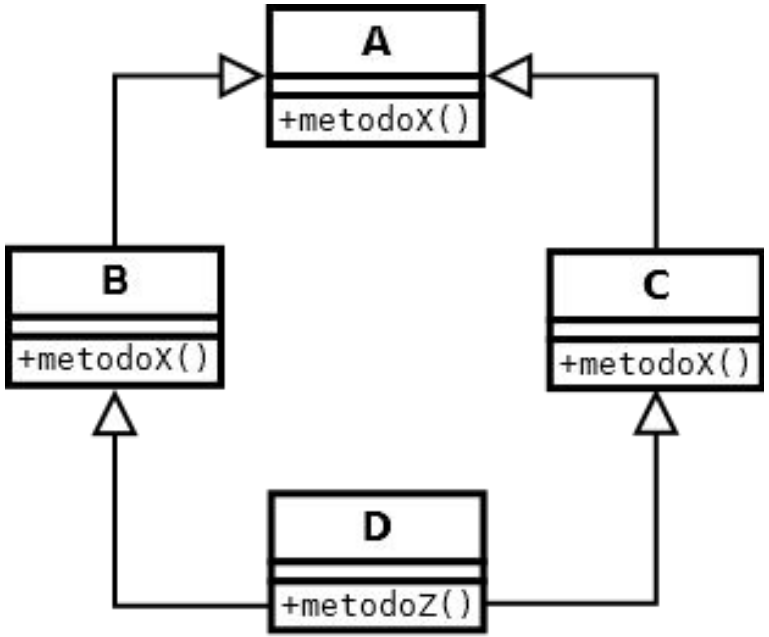
Herança Múltipla - Ambiguidades

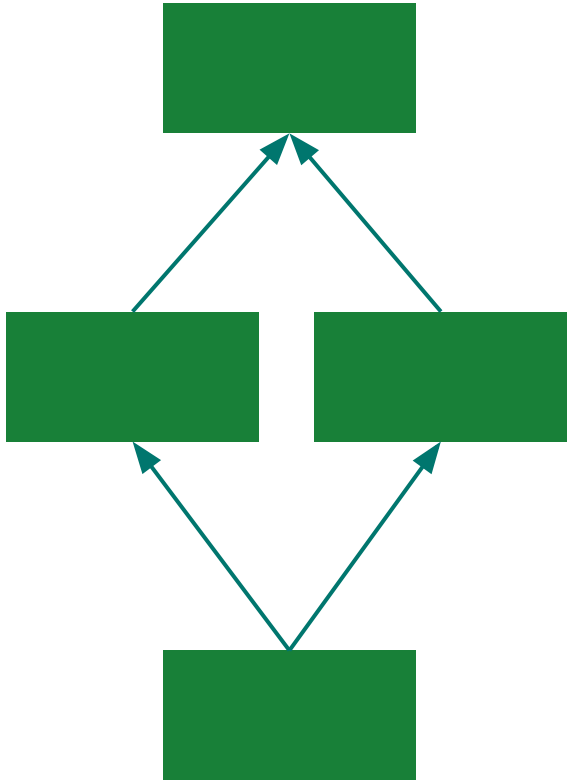
Propriedades de mesmo nome:

- Professor tem matrícula
- Administrativo tem matrícula

Problema do Diamante

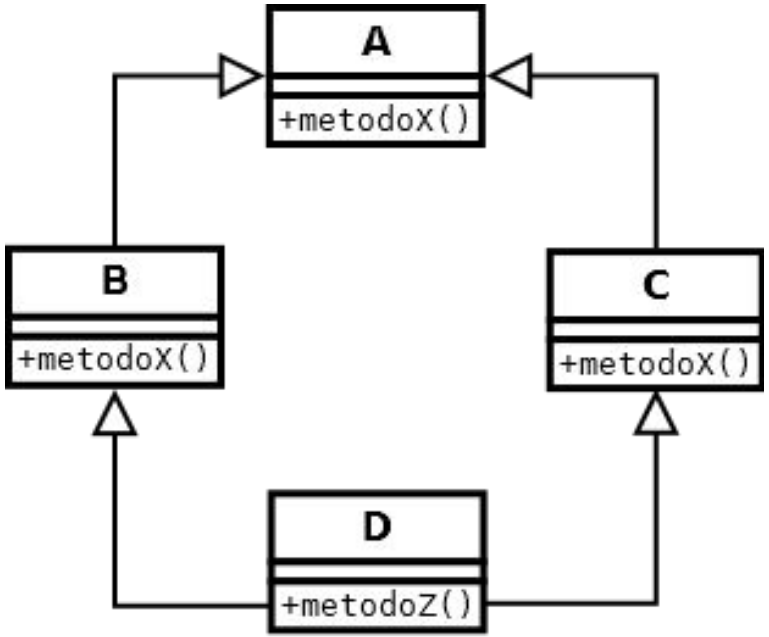
E se...





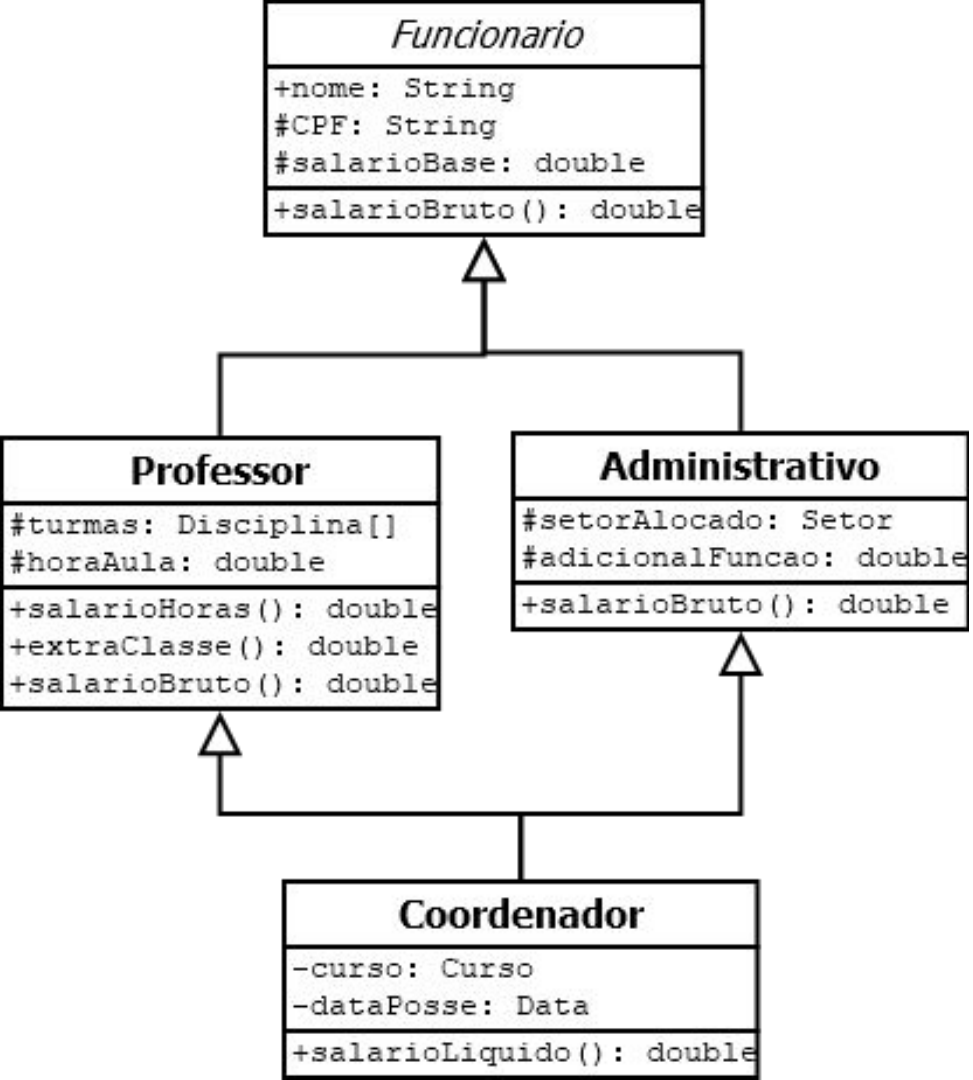
Problema do Diamante

E se...




```
D myD = new D();
myD.metodoZ();
```

E se o *metodoZ()* usa *metodoX()*?




```
public double salarioLiquido(){

    return super.salarioBruto()
                * 0.775;
}
```

Herança Múltipla

Há soluções, mas ainda assim a herança múltipla é, em geral, considerada uma **solução pobre** para um projeto

Ainda assim, **é permitida** em algumas linguagens importantes: C++, Perl, Python

Java e **C#** não permitem herança múltipla

Herança Múltipla

- Então, o que utilizamos para contornar o problema?

Interfaces


Interface

O que são?

São estruturas que definem um conjunto de métodos que uma classe deve implementar

Especificam um contrato, ou seja, as classes que implementam uma interface são obrigadas a fornecer uma implementação para todos os métodos declarados

Interface

O que são?

Se uma classe possuir apenas métodos abstratos, podemos criá-la como uma interface

A interface tem funcionalidade similar à de classes abstratas

Interface

Como criar e utilizar?

Criar uma interface:

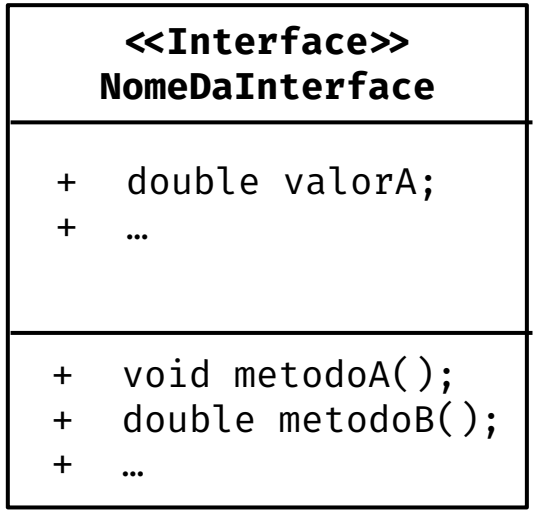
public interface *NomeInterface*{}

Utilizar interface

public class ***NomeClasse* implements *NomeInterface***{}

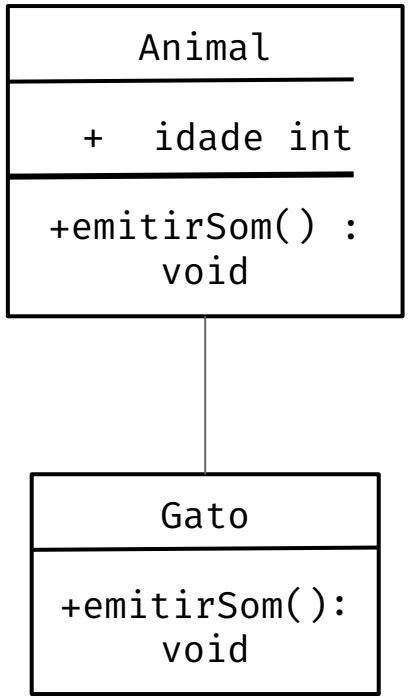
Interface

Como representar na UML?



Polimorfismo

Como representar polimorfismo na UML?



Apenas repetir o método na subclasse mostra que ele foi sobrescrito

Interface

Características

- Interfaces também não podem ser instanciadas
- Todos os métodos na interface são implicitamente **abstract** e **public**
- Campos serão implicitamente static e final (devem ser inicializados na declaração)
- Não possuem construtores

Interface

Características

A diferença essencial entre classes abstratas e interfaces é que uma classe herdeira somente pode herdar de uma única classe (abstrata ou não), enquanto que qualquer classe pode implementar várias interfaces simultaneamente

EXEMPLO

Um drone militar é um veículo aéreo não tripulado e uma arma de guerra

Como implementar?

Exemplo

```
interface VeiculoAereo{
    void voar();
}

interface ArmaGuerra{
    void atacar(double latitude, double longitude);
}
```

interfaces

Exemplo

```
class DroneMilitar implements VeiculoAereo, ArmaGuerra{

    public void voar(){
        //implementação
    }

    public void atacar(double latitude, double longitude){
        //implementação
    }
}
```

Exemplo

```
public static void main(String[] args) {
    DroneMilitar drone = new DroneMilitar();
    drone.voar();
    drone.atacar(-23.5270, -46.6784); //estádio do palmeiras
}
```

Por que não criar uma classe concreta com os métodos ao invés de interface❓

INTERFACE

Interface permite herança múltipla

Interface foca no "o que faz", não "como faz”

Interfaces são ideais para **Polimorfismo**

EXEMPLO 2

Imagine que você foi contratado para desenvolver um sistema de apoio militar automatizado. Esse sistema será responsável por controlar armas de guerra inteligentes, como **mísseis terrestres** e **drones militares** autônomos.

Como implementar?

Exemplo 2

```
interface VeiculoAereo{
    void voar();
}

interface ArmaGuerra{
    void atacar(double latitude, double longitude);
}
```

interfaces

Exemplo 2

```
class DroneMilitar implements VeiculoAereo, ArmaGuerra{

    public void voar(){
        //implementação
    }

    public void atacar(double latitude, double longitude){
        //implementação
    }
}
```

```
class MissilTerrestre implements ArmaGuerra {

    public void atacar(double latitude, double longitude) {
        //implementacao
    }
}
```

Exemplo 2

```
public static void main(String[] args) {
    ArrayList<MissilTerrestre> misseis = new ArrayList<>();
    ArrayList<DroneMilitar> drones = new ArrayList<>();

    misseis.add(new MissilTerrestre());
    misseis.add(new MissilTerrestre());
    drones.add(new DroneMilitar());

    for (MissilTerrestre missel: misseis){
        missel.atacar(-23.5270, -46.6784);
    }

    for (DroneMilitar drone: drones){
        drone.atacar(-23.5270, -46.6784);
    }
}
```

melhor implementação??

Exemplo 2

```
public static void main(String[] args) {
    ArrayList<ArmaGuerra> arsenal = new ArrayList<>();

    arsenal.add(new MissilTerrestre());
    arsenal.add(new MissilTerrestre());
    arsenal.add(new DroneMilitar());
    // adiciono quantas armas eu quiser...

    for (ArmaGuerra arma: arsenal){
        arma.atacar(-23.5270, -46.6784); //estádio do palmeiras
    }
}
```

Polimorfismo (:

EXEMPLO 2

- Polimorfismo: *ArmaDeGuerra arma = new DroneMilitar()*
- Modularidade: Você pode trocar **DroneMilitar** por qualquer classe que implemente a interface
- Extensibilidade: Fácil adicionar novos tipos de armas (ex: Tanque, Navio) sem mudar o código existente

Interfaces

==

Herança Múltipla

Interface

Bibliotecas de constantes

⇒ Interfaces que somente contém campos. Classes que a implementam, terão acesso a estes campos

```
public interface IConstantes {

    double PI = 3.141592;
    double MAX = 10;
    double MIN = 3;

}
```

Interface

Bibliotecas de constantes

```
public class CalculosMatematicos implements IConstantes{

    private int valorA;
    private int valorB;

    public CalculosMatematicos(int valorA
        this.valorA = valorA;
        this.valorB = valorb;
    }

    public double somaPi(){
        return valorA + valorB + PI;
    }

}
```

```
public interface IConstantes {

    double PI = 3.141592;
    double MAX = 10;
    double MIN = 3;

}
```

Interface

Métodos Estáticos

⇒ A partir do Java 8, é possível criar métodos estáticos nas interfaces

Ex:

```
public interface IConstantes {

    double PI = 3.141592;
    double MAX = 10;
    double MIN = 3;

    public static double areaCirculo(double raio){
        return PI * Math.pow(a: raio, b: 2);
    }

}
```

Interface

Métodos Estáticos

⇒ Não fazem parte da API da Interface

=> Ou seja, não é possível acessar tais métodos a partir de objetos de classes que implementam tal interface

Exemplo

Exemplo

```java
public interface IConstantes {

    double PI = 3.141592;
    double MAX = 10;
    double MIN = 3;

    public static double areaCirculo(double raio){
        return PI * Math.pow(a: raio, b: 2);
    }
}
```

```java
public class CalculosMatematicos implements IConstantes{

    private int valorA;
    private int valorB;

    public CalculosMatematicos(int valorA, int valorb) {
        this.valorA = valorA;
        this.valorB = valorb;
    }

    public double somaPi(){
        return valorA + valorB + PI;
    }

}
```

Não podemos chamar o método a partir de um objeto de uma classe que implementa a interface

```java
CalculosMatematicos cm;
cm = new CalculosMatematicos(valorA: 10, valorb: 5);
cm.somaPi();

double area = cm.areaCirculo(13);
```

Interface

Outro Exemplo

Considere que temos diferentes tipos de veículos:

- Carro, Moto, Bicicleta

E queremos implementar algumas funcionalidades a estes veículos, tais como acelerar e freiar

Interface

Outro Exemplo

Ainda, suponha que apenas veículos que possuem motorização movidos à combustíveis (gasolina, etanol, GNV, etc.) são os veículos que de fato possam acelerar

Solução 1 - Classe Abstrata

```java
public abstract class Veiculo {

    private int codigo;
    private String descrição;

    public abstract void acelerar();
    public abstract void freiar();
}
```

```java
public class Moto extends Veiculo{

    @Override
    public void acelerar() {
        // Acelera o motor da moto
    }

    @Override
    public void freiar() {
        // Freia o motor da moto
    }
}
```

```java
public class Carro extends Veiculo{

    @Override
    public void acelerar() {
        // Código pra acelerar o motor do carro
    }

    @Override
    public void freiar() {
        //código para frear o motor do carro
    }

}
```

```java
public class Bicicleta extends Veiculo{

    @Override
    public void acelerar() {
        // Não tem motor, logo não implementa este método
        // Lançar exceção
    }

    @Override
    public void freiar() {
        // Freia a roda da bicicleta
    }
}
```

Solução 1 - Classe Abstrata

E se quisermos incluir um novo veículo não motorizado, tal como Patinete ou Skate?

⇒ métodos “sem utilização”

Solução?

Solução 2 - Interface

```java
public interface Aceleravel {

    public abstract void acelera();
}
```

```java
public abstract class Veiculo {

    private int codigo;
    private String descrição;

    public abstract void freiar();
}
```

```java
public class Carro extends Veiculo implements Aceleravel{

    @Override
    public void freiar() {
        //código para frear o motor do carro
    }

    @Override
    public void acelera() {
        //Code para acelerar o carro
    }
}
```

```java
public class Moto extends Veiculo implements Aceleravel{

    @Override
    public void freiar() {
        // Freia o motor da moto
    }

    @Override
    public void acelera() {
        // Code pra acelerar moto
    }
}
```

Solução 2 - Interface

```java
public abstract class Veiculo {

    private int codigo;
    private String descrição;

    public abstract void freiar();
}
```

Apenas estende a classe veículo

```java
public class Bicicleta extends Veiculo{

    @Override
    public void freiar() {
        // Freia a roda da bicicleta
    }
}
```

Não é obrigada a implementar métodos desnecessários

Solução 2 - Interface

Ajuda na extensibilidade e legibilidade do código

Para Casa

Pense e liste as principais diferenças entre interface e classe abstrata em Java.

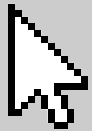
Código da Aula de Hoje

https://github.com/daniloboechat/aulaInterfaces

Dúvidas?

PUC Minas

Obrigado!



PUC Minas
Bacharelado em Engenharia de Software
Prof. Daniel Kansaon
slides adaptados do Prof. Danilo Boechat