# POO Classes abstratas, métodos abstratos e interfaces

Prof. Alcides Calsavara
PUCPR

# Conceitos

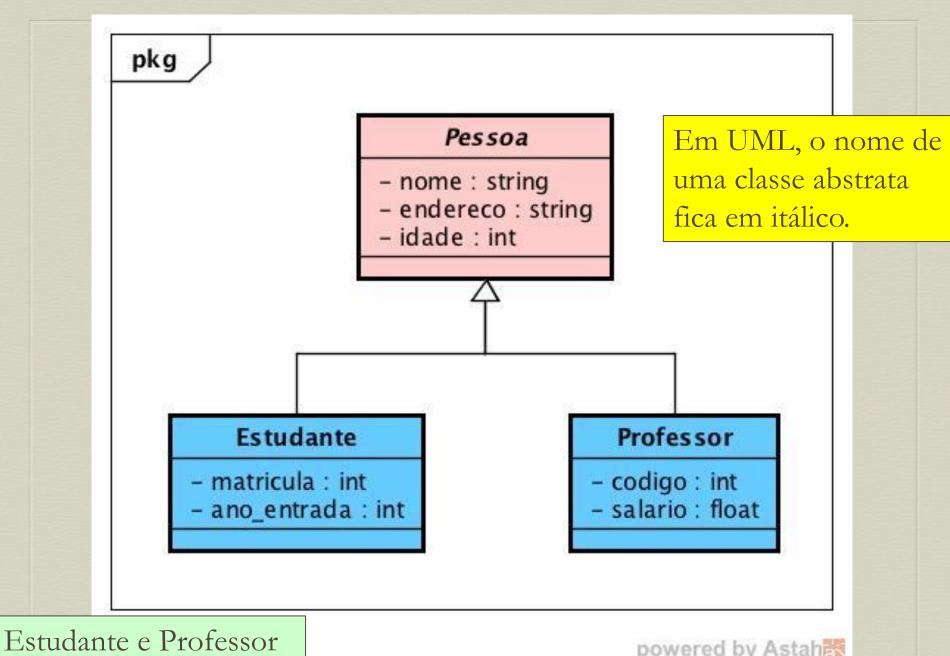
- 1. Classe abstrata
- 2. Método abstrato
- 3. Interface
  - Merança entre interfaces

## Classe Abstrata

É uma classe que **não** pode ser diretamente instanciada.

Sintaxe em Java:

abstract class nome\_da\_classe { ... }



são classes concretas.

powered by Astah

```
abstract class Pessoa { ... }
```

```
class Estudante extends Pessoa { ... }
```

```
class Professor extends Pessoa { ... }
```

```
Pessoa p = new Pessoa();
```

Estudante e = new Estudante();

Pessoa p = new Professor();

```
abstract class Pessoa { ... }
```

```
class Estudante extends Pessoa { ... }
```

```
class Professor extends Pessoa { ... }
```

```
Pessoa p = new Pessoa(); INVÁLIDO
```

Estudante e = new Estudante();

```
Pessoa p = new Professor();
```

```
abstract class Pessoa { ... }
```

```
class Estudante extends Pessoa { ... }
```

```
class Professor extends Pessoa { ... }
```

```
Pessoa p = new Pessoa(); INVÁLIDO
```

```
Estudante e = new Estudante(); VÁLIDO
```

Pessoa p = new Professor();

```
abstract class Pessoa { ... }
```

```
class Estudante extends Pessoa { ... }
```

```
class Professor extends Pessoa { ... }
```

```
Pessoa p = new Pessoa(); INVÁLIDO
```

```
Estudante e = new Estudante(); VÁLIDO
```

Pessoa p = new Professor(); VÁLIDO

#### Estudante e = new Estudante();

- e instanceof Estudante
- e instanceof Pessoa

```
Pessoa p = new Professor();
```

- p instanceof Professor
- p instanceof Pessoa

```
Estudante e = new Estudante();
```

TRUE

e instanceof Pessoa

Pessoa p = new Professor();

p instanceof Professor

p instanceof Pessoa

```
Estudante e = new Estudante();
```

TRUE

e instanceof Pessoa TRUE

Pessoa p = new Professor();

p instanceof Professor

p instanceof Pessoa

```
Estudante e = new Estudante();
```

TRUE

e instanceof Pessoa TRUE

Pessoa p = new Professor();

p instanceof Professor

TRUE

p instanceof Pessoa

```
Estudante e = new Estudante();
```

TRUE

e instanceof Pessoa

TRUE

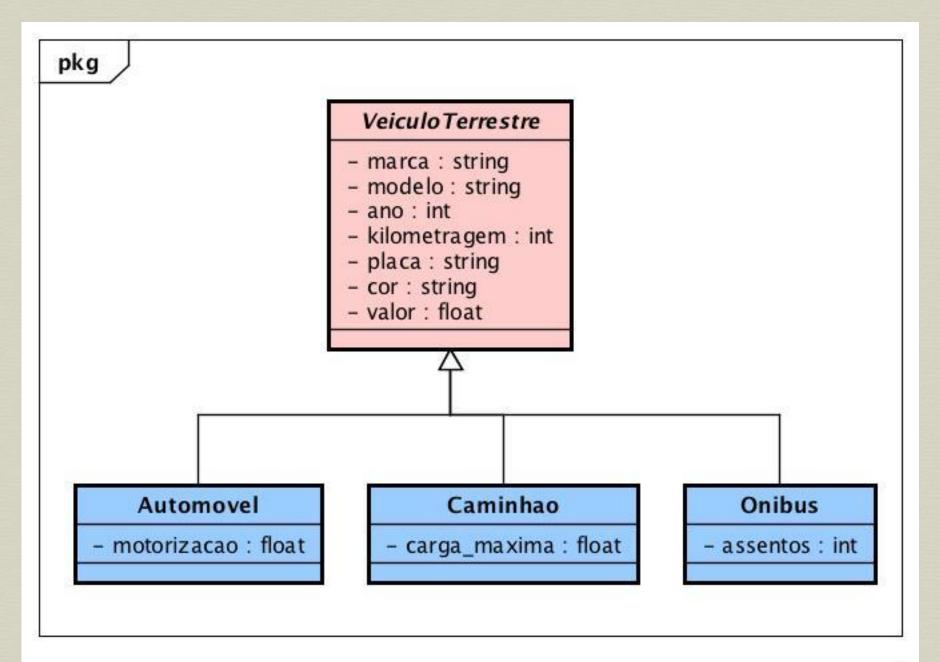
Pessoa p = new Professor();

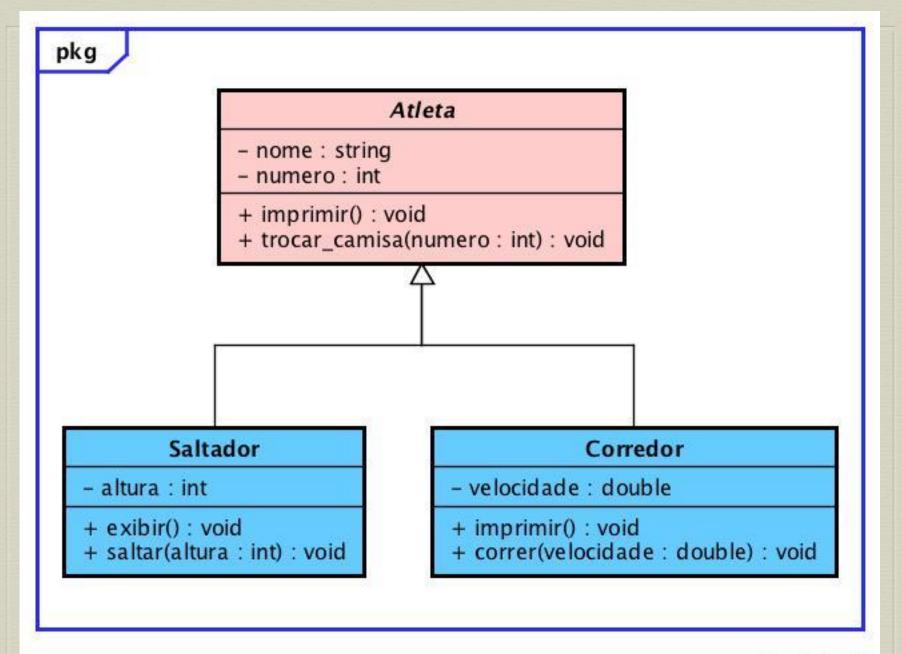
p instanceof Professor TRUE

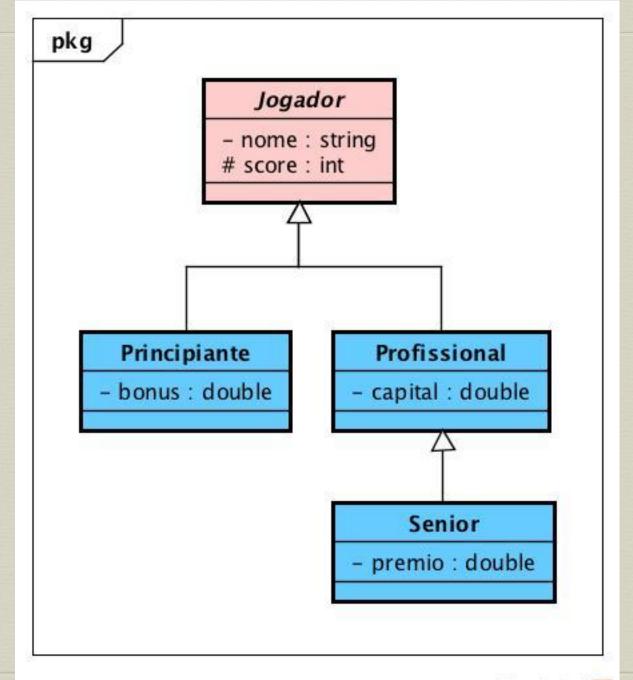
p instanceof Pessoa

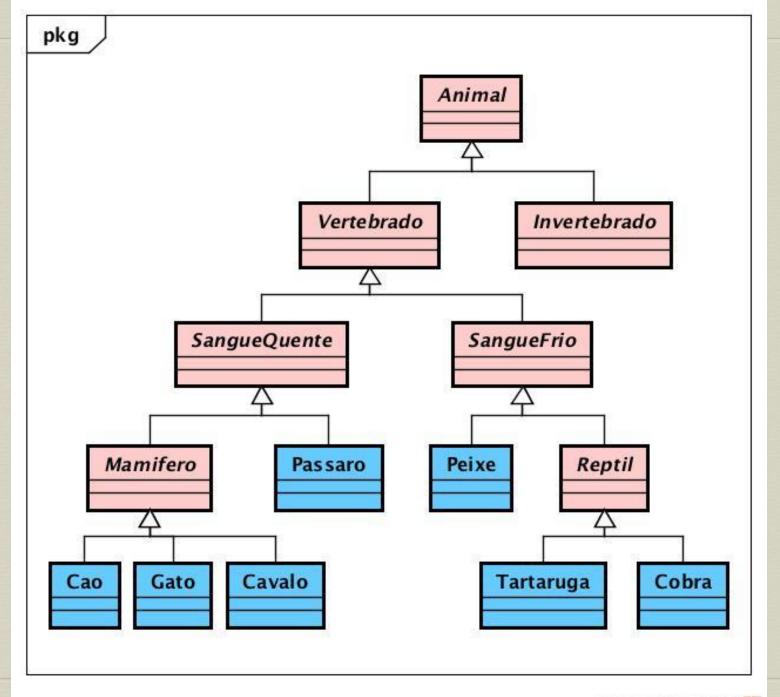
TRUE

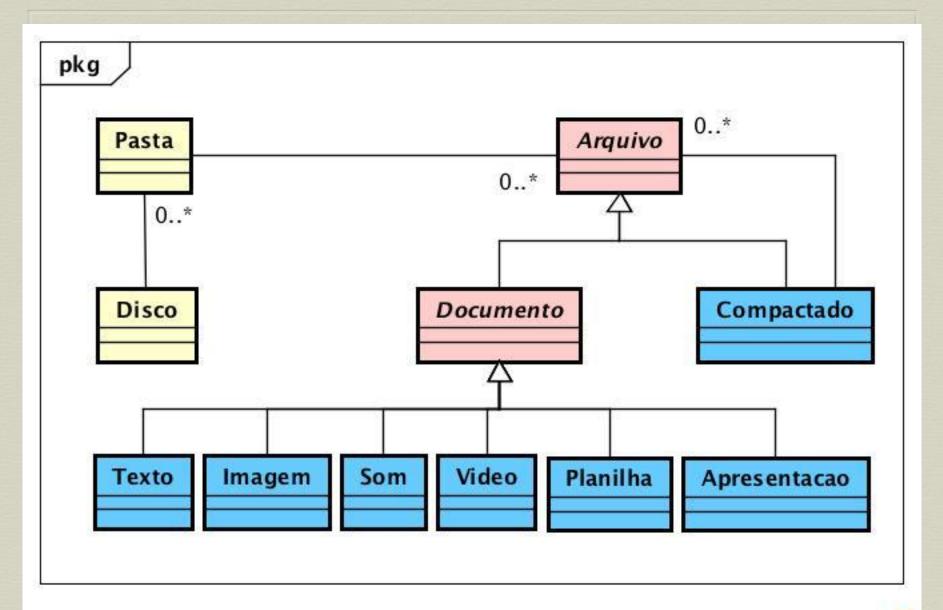
# Revisando os exemplos anteriores



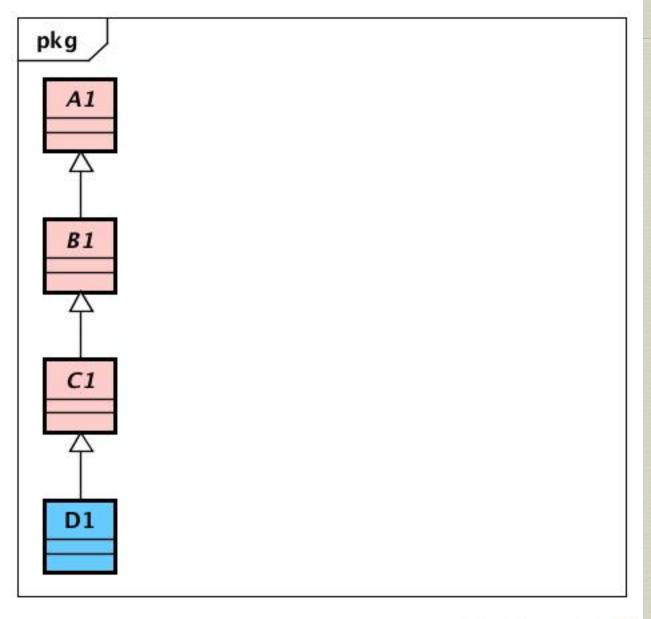


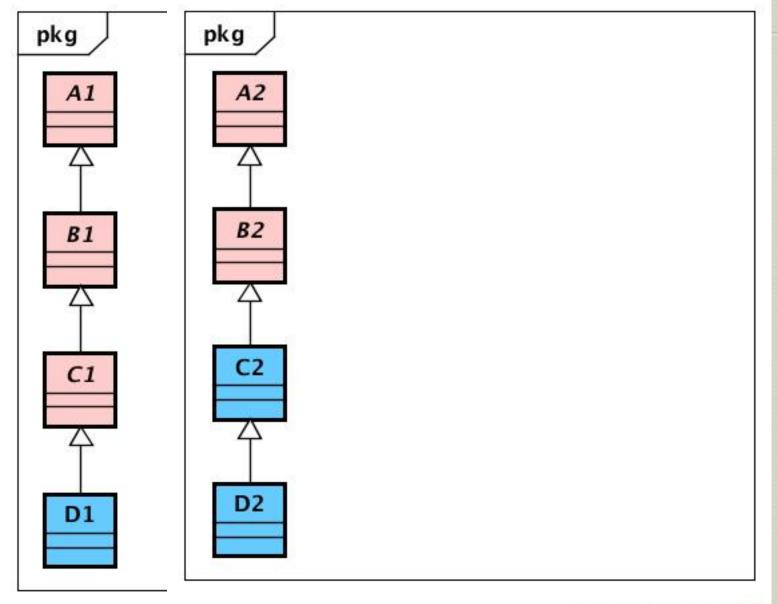




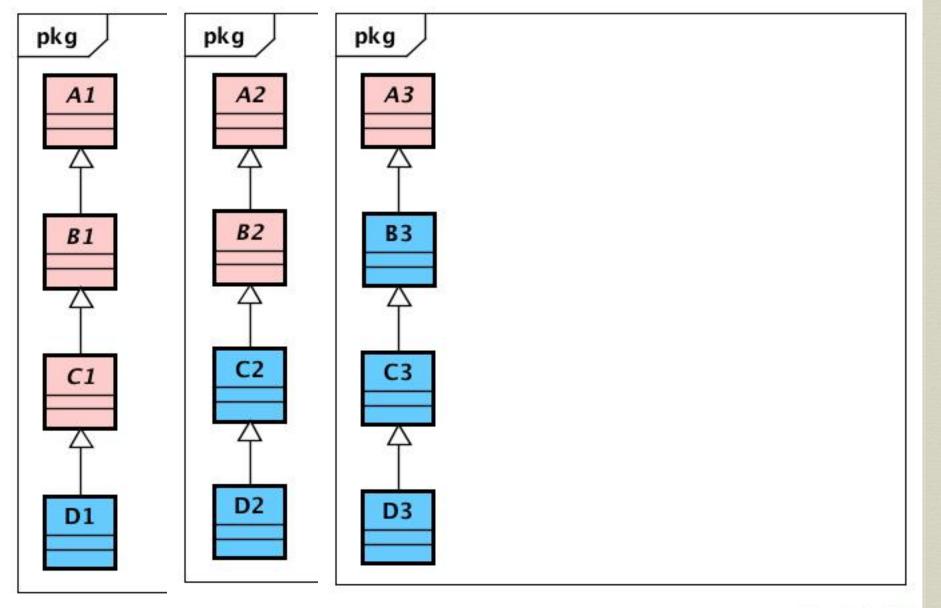


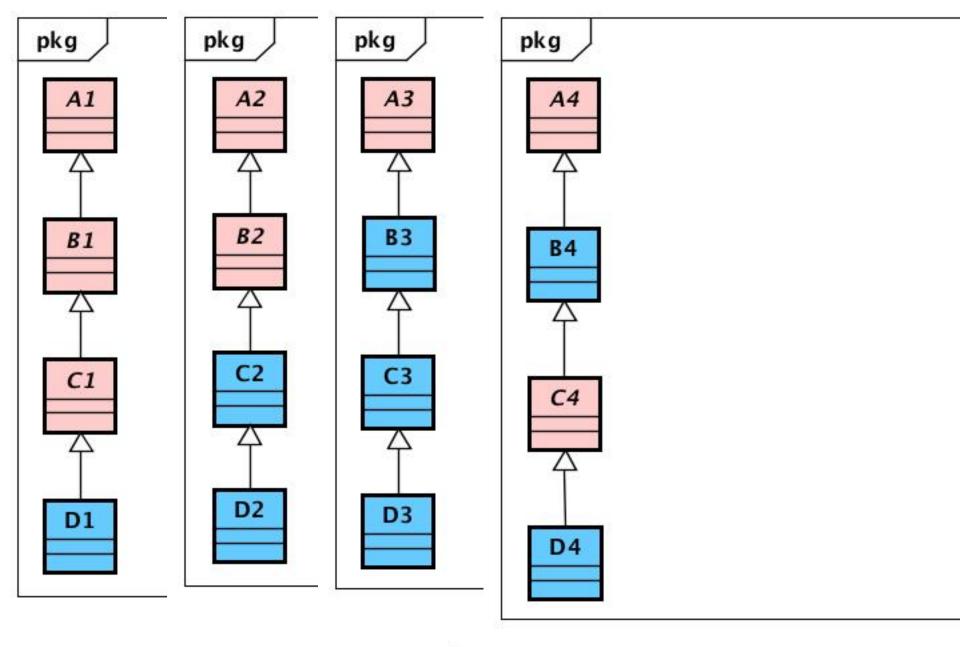
Classes abstratas podem aparecer em qualquer nível da hierarquia de classes.

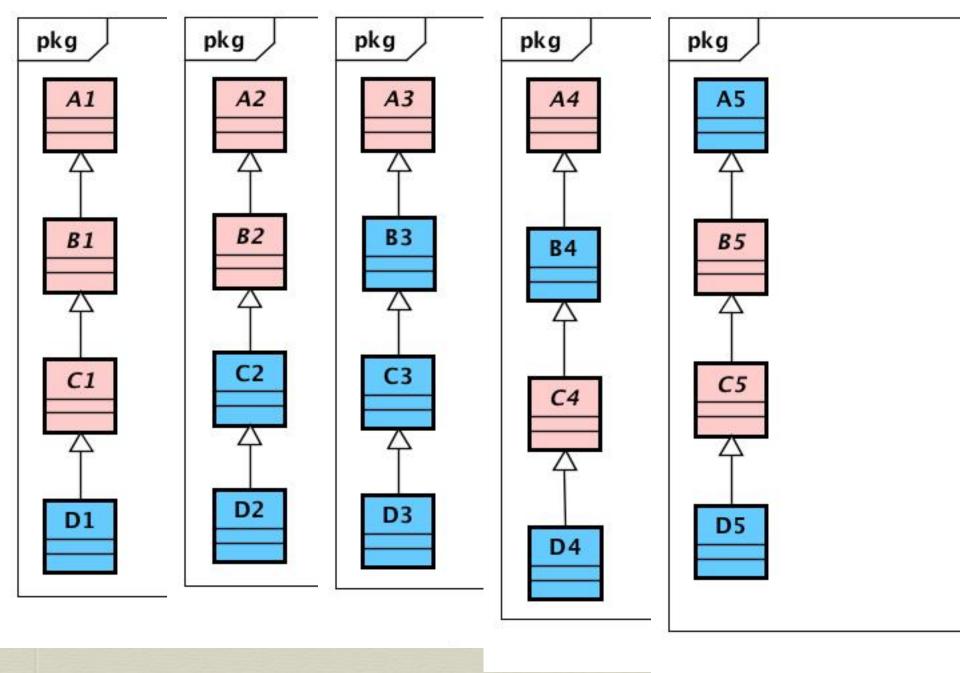


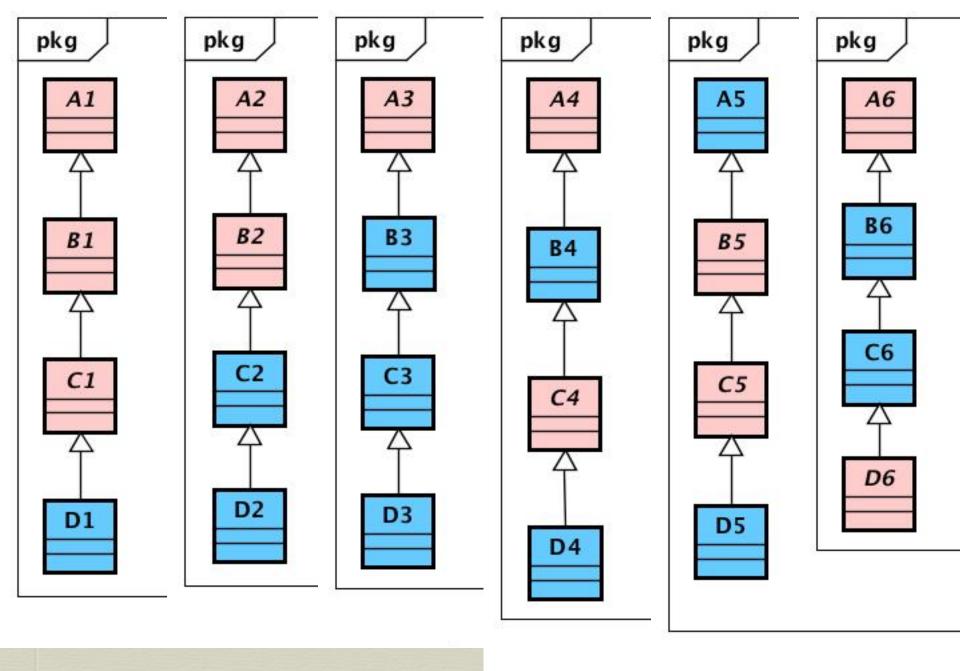


powered by Astah





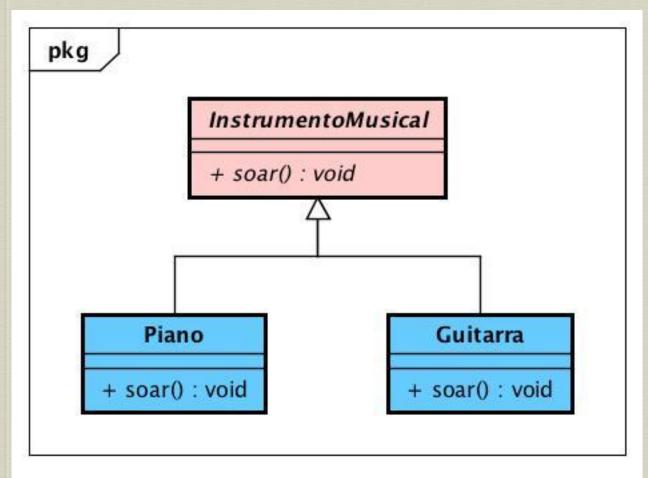




# Métodos Abstratos

### Método Abstrato

- 🗭 É um método que não possui implementação.
- Permitido somente em classe abstrata.
- Deve ser sobrecarregado e implementado em alguma subclasse (abstrata ou concreta).
- Mão pode ter visibilidade private.
- Uma classe concreta não pode ter qualquer pendência de implementação de método abstrato herdado.
- Pode ser chamado de maneira abstrata!



Em UML, um método abstrato fica em itálico.

powered by Astah

O método **soar** é abstrato na classe InstrumentoMusical e é sobrecarregado e implementado nas subclasses Piano e Guitarra.

```
abstract class InstrumentoMusical {
   abstract public void soar();
}
```

```
class Piano extends InstrumentoMusical {
    public void soar() {
        // emite som de piano
    }
}
```

```
class Guitarra extends InstrumentoMusical {
    public void soar() {
        // emite som de guitarra
    }
}
```

```
Piano p = new Piano();
Guitarra g = new Guitarra();
p.soar();
g.soar();
InstrumentoMusical x = p;
x.soar();
x = g;
x.soar();
```

```
Piano p = new Piano();
Guitarra g = new Guitarra();
p.soar(); // soar de Piano
g.soar();
InstrumentoMusical x = p;
x.soar();
x = q;
x.soar();
```

```
Piano p = new Piano();
Guitarra g = new Guitarra();
p.soar(); // soar de Piano
g.soar(); // soar de Guitarra
InstrumentoMusical x = p;
x.soar();
x = g;
x.soar();
```

```
Piano p = new Piano();
Guitarra g = new Guitarra();
p.soar(); // soar de Piano
g.soar(); // soar de Guitarra
InstrumentoMusical x = p;
x.soar(); // soar de Piano
x = q;
x.soar();
```

```
Piano p = new Piano();
Guitarra g = new Guitarra();
p.soar(); // soar de Piano
g.soar(); // soar de Guitarra
InstrumentoMusical x = p;
x.soar(); // soar de Piano
x = q;
x.soar(); // soar de Guitarra
```

```
Piano p = new Piano();
Guitarra g = new Guitarra();
p.soar(); // soar de Piano
g.soar(); // soar de Guitarra
InstrumentoMusical x = p;
x.soar(); // soar de Piano
x = q;
x.soar(); // soar de Guitarra
```

Chamada de método abstrato a partir de uma referência de classe abstrata

```
ArrayList<InstrumentoMusical> instrumentos;
instrumentos = new ArrayList<InstrumentoMusical> ();
instrumentos.add(new Piano());
instrumentos.add(new Guitarra());
instrumentos.add(new Guitarra());
for (InstrumentoMusical i: instrumentos)
    i.soar();
```

Chamada de método abstrato a partir de uma coleção de referências de classe abstrata

```
void play(InstrumentoMusical i) {
   i.soar();
}
```

Chamada de método abstrato a partir de uma referência de classe abstrata

```
Piano p = new Piano();
Guitarra g = new Guitarra();
play(p);
play(g);
```

```
void play(InstrumentoMusical i) {
   i.soar();
}
```

Chamada de método abstrato a partir de uma referência de classe abstrata

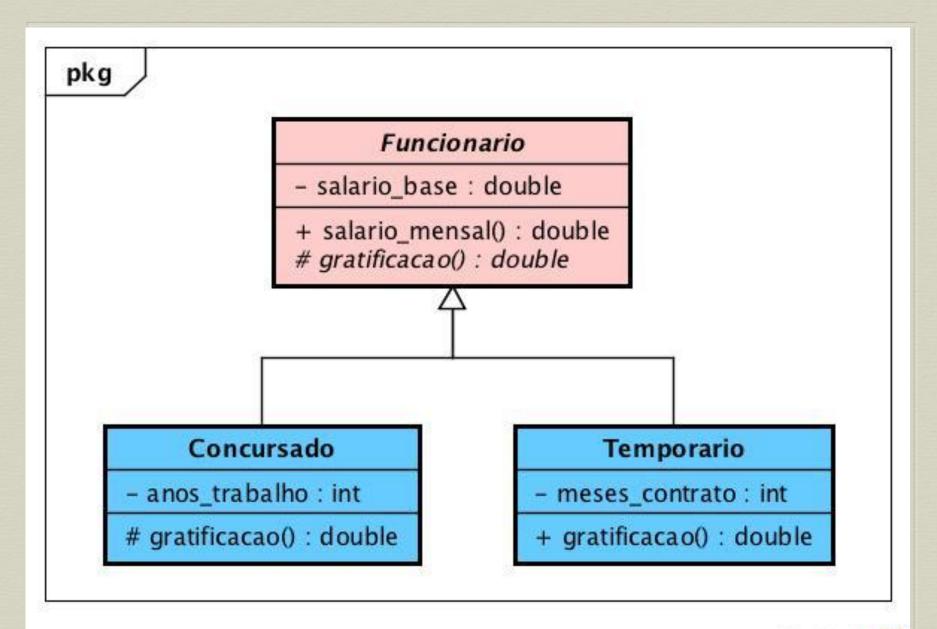
```
Piano p = new Piano();
Guitarra g = new Guitarra();
play(p); // executa soar de Piano
play(g);
```

```
void play(InstrumentoMusical i) {
   i.soar();
}
```

Chamada de método abstrato a partir de uma referência de classe abstrata

```
Piano p = new Piano();
Guitarra g = new Guitarra();
play(p); // executa soar de Piano
play(g); // executa soar de Guitarra
```

## Chamada de método abstrato interna à hierarquia de classes



```
abstract class Funcionario {
   private double salario base;
    public Funcionario(double salario base) {
        this.salario base = salario base;
    public double salario mensal() {
        double total = salario base + gratificacao();
        return total;
    abstract protected double gratificacao();
```

Chamada de método abstrato por outro método da mesma classe (poderia ser de uma subclasse)

```
class Concursado extends Funcionario {
   private int anos trabalho;
    public Concursado (double salario base,
                      int anos trabalho) {
        super(salario base);
        this.anos trabalho = anos trabalho;
    protected double gratificacao() {
        return (anos trabalho * 100);
```

Classe concreta sobrecarrega e implementa o método abstrato herdado

```
class Temporario extends Funcionario {
   private int meses contrato;
    public Temporario (double salario base,
                      int meses contrato) {
        super(salario base);
        this.meses contrato = meses contrato;
    public double gratificacao() {
        return (meses contrato * 10);
```

Classe concreta sobrecarrega e implementa o método abstrato herdado

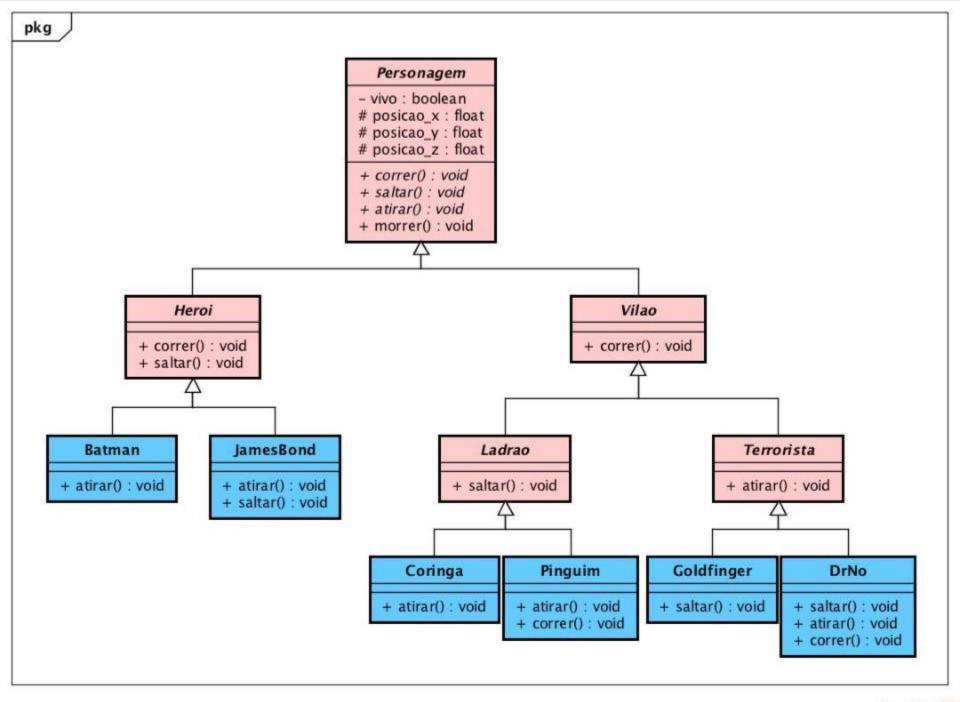
```
Concursado joao = new Concursado(4000, 5);
Temporario jose = new Temporario( 2000, 12);
System.out.println( joao.salario_mensal() );
System.out.println( jose.salario_mensal() );
```

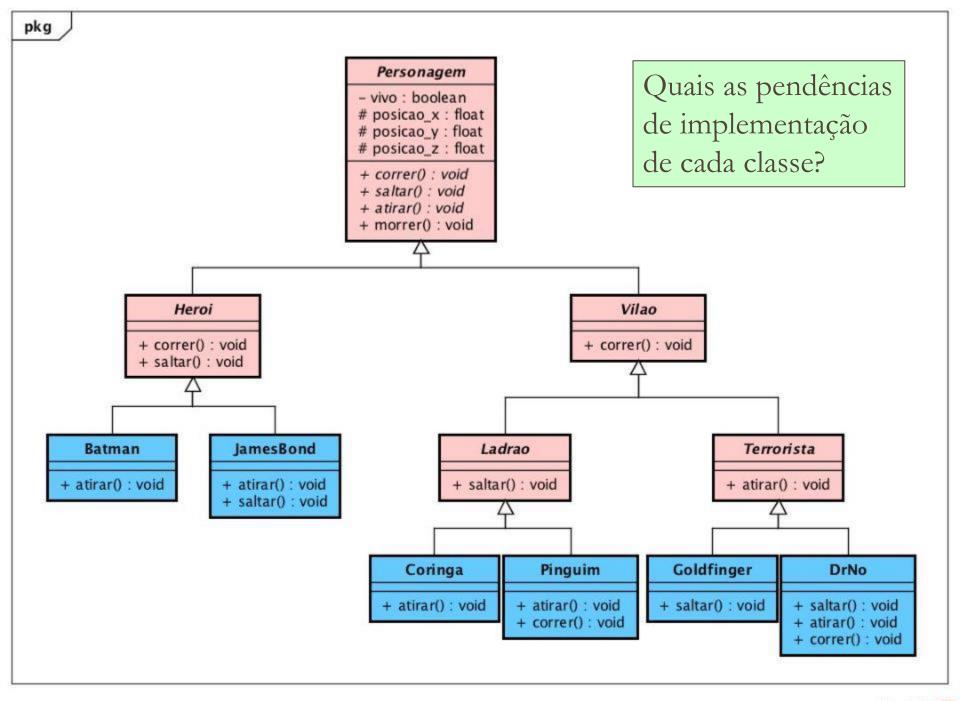
Quais os valores impressos?

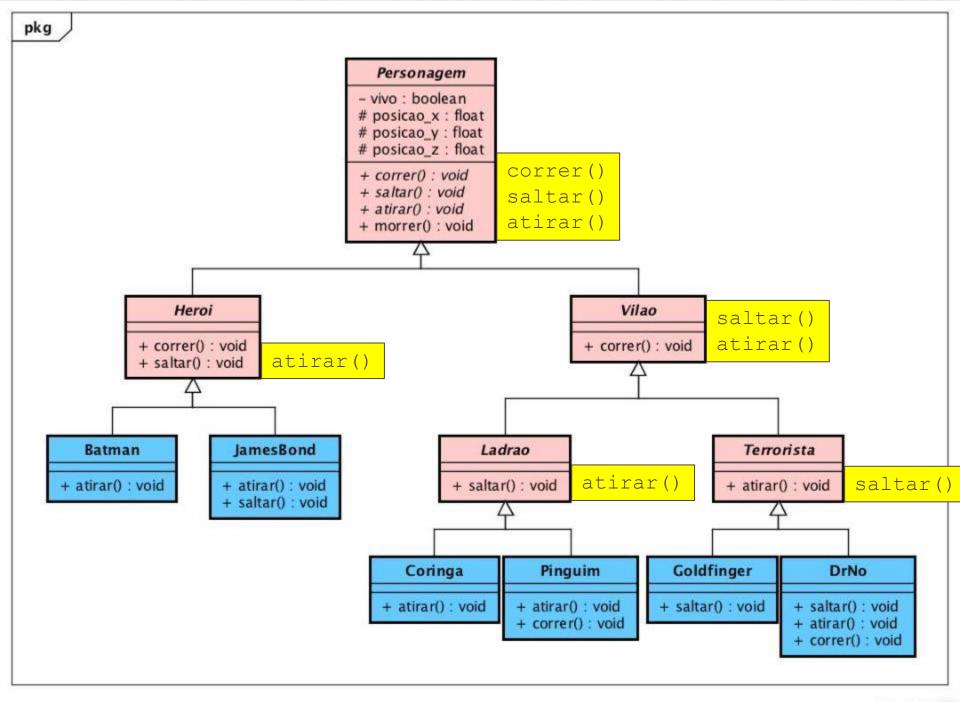
```
Concursado joao = new Concursado(4000, 5);
Temporario jose = new Temporario( 2000, 12);
System.out.println( joao.salario_mensal() );
System.out.println( jose.salario_mensal() );
```

Quais os valores impressos?

4500.00 2120.00 Uma classe concreta não pode ter qualquer pendência de implementação de método abstrato herdado.



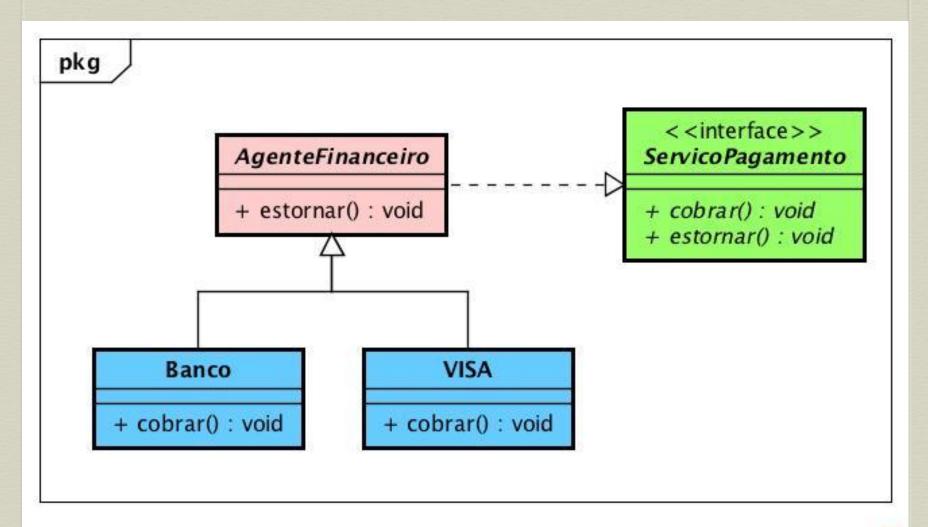




## Interface

## Interface

- É equivalente a uma classe abstrata composta somente de métodos abstratos públicos.
- Também pode definir constantes.
- Uma classe pode implementar uma ou mais interfaces.
- Uma classe que implementa certa interface deve implementar todos os métodos (abstratos) definidos ou herdados pela interface. Caso não implemente algum desses métodos, esse passa a ser considerado como abstrato da classe.
- Pode haver relacionamento de herança (múltipla, inclusive) entre interfaces.



```
abstract class AgenteFinanceiro implements ServicoPagamento {

public void estornar(Lojista lojista,

Consumidor consumidor,

double valor) {

lojista.debitar(valor);

consumidor.depositar(valor);

}

}
```

```
class Banco extends AgenteFinanceiro {
   private String nome;
   private double taxa;
   public Banco(String nome, double taxa) {
        this.nome = nome;
        this.taxa = taxa;
   public void cobrar (Lojista lojista,
                       Consumidor consumidor,
                       double valor) {
        consumidor.debitar(valor);
        double tarifa = valor * taxa;
        lojista.depositar(valor - tarifa);
```

```
class VISA extends AgenteFinanceiro {
   public void cobrar (Lojista lojista,
                       Consumidor consumidor,
                       double valor) {
        consumidor.debitar(valor);
        consumidor.bonificar((int)(valor/100));
        double taxa = 0.05;
        if (lojista.categoria() == 'A')
           taxa = 0.02;
        else if (lojista.categoria() == 'B')
            taxa = 0.07;
        double tarifa = valor * taxa;
        lojista.depositar(valor - tarifa);
```

```
abstract class Correntista {
   private double saldo;
   public Correntista(double saldo) {
        this.saldo = saldo;
   public void debitar(double valor) {
        saldo = saldo - valor;
   public void depositar(double valor) {
        saldo = saldo + valor;
```

```
class Consumidor extends Correntista {
   private int pontos;
   public Consumidor(double saldo) {
      super(saldo);
      pontos = 0;
   }
   public void bonificar(int pontos) {
      this.pontos = this.pontos + pontos;
   }
}
```

```
class Lojista extends Correntista {
    private char categoria;
    public Lojista(double saldo, char categoria) {
        super(saldo);
        this.categoria = categoria;
    }
    public char categoria() {
        return categoria;
    }
}
```

```
Banco itau = new Banco("Itau", 0.02);
VISA visa = new VISA();
Lojista americanas = new Lojista(10000, 'C');
Lojista amazon = new Lojista(20000, 'A');
Consumidor carmem = new Consumidor (500.00);
Consumidor beatriz = new Consumidor (800.00);
ServicoPagamento servico;
servico = itau;
servico.cobrar(americanas, carmem, 200.00);
servico.estornar(amazon, beatriz, 100.00);
servico = visa;
servico.cobrar(amazon, carmem, 300.00);
servico.estornar(amazon, carmem, 50.00);
```

## Hiearquia de Interfaces e Hierarquia de Classes

