POO – Módulo 7

Classes abstratas, métodos abstratos e interfaces

Módulo ministrado pelo

Prof. Edson Emílio Scalabrin

PUCPR

Material produzido pelo
prof. Alcides Calsavara (BCC/PUCPR)

Conceitos

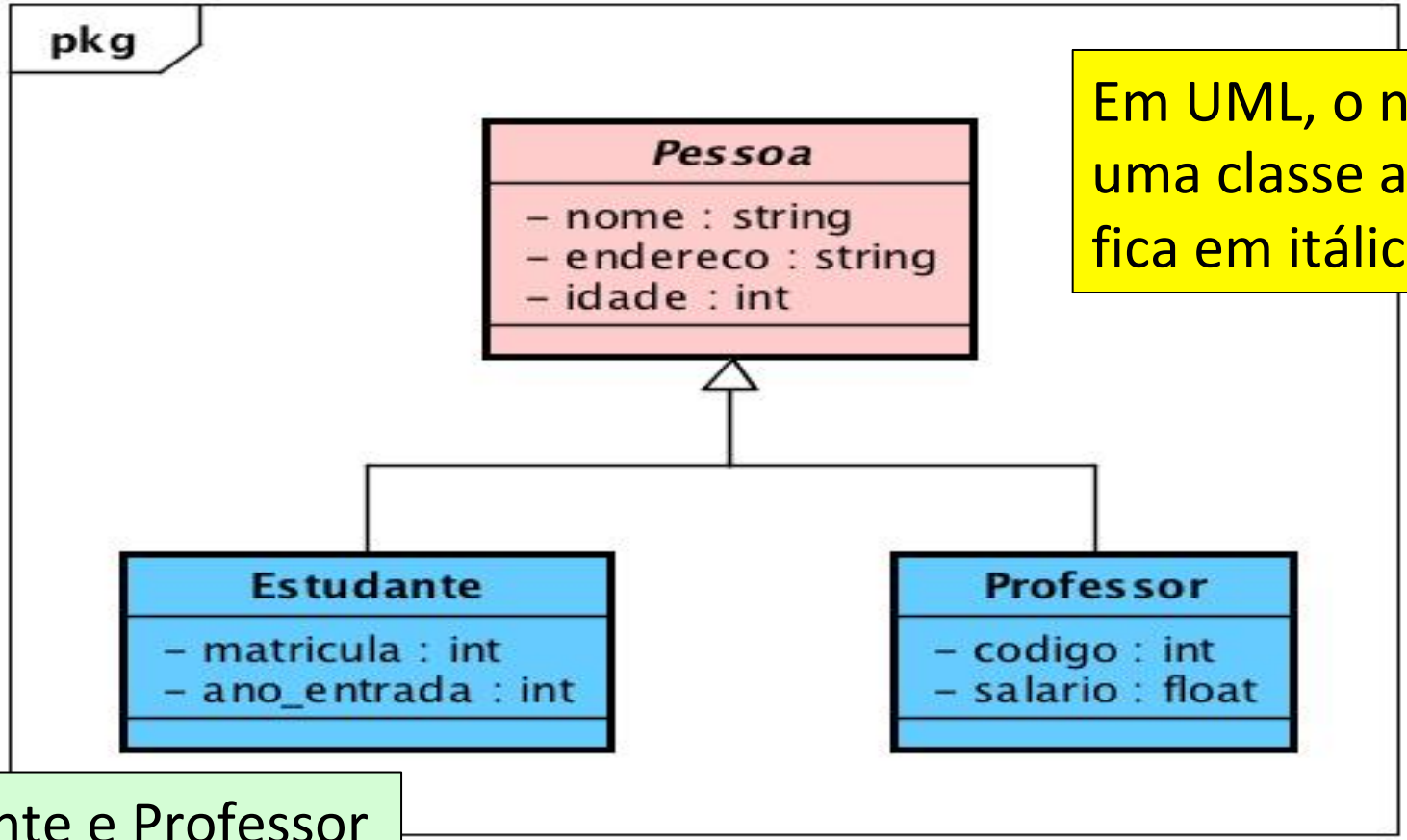
1. **Classe abstrata**
2. **Método abstrato**
3. **Interface**
   - **Herança entre interfaces**

Classe Abstrata

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

Sintaxe em Java:

abstract **class** *nome_da_classe* { … }



Em UML, o nome de uma classe abstrata fica em itálico.

Estudante e Professor são **classes concretas**.

```
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();
```

```
e instanceof Estudante
```

TRUE

```
e instanceof Pessoa
```

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

```
p instanceof Professor
```

```
p instanceof Pessoa
```

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

```
e instanceof Estudante
```

TRUE

```
e instanceof Pessoa
```

TRUE

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

```
p instanceof Professor
```

```
p instanceof Pessoa
```

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

```
e instanceof Estudante
```
TRUE

```
e instanceof Pessoa
```
TRUE

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

```
p instanceof Professor
```
TRUE

```
p instanceof Pessoa
```

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

```
e instanceof Estudante
```
TRUE

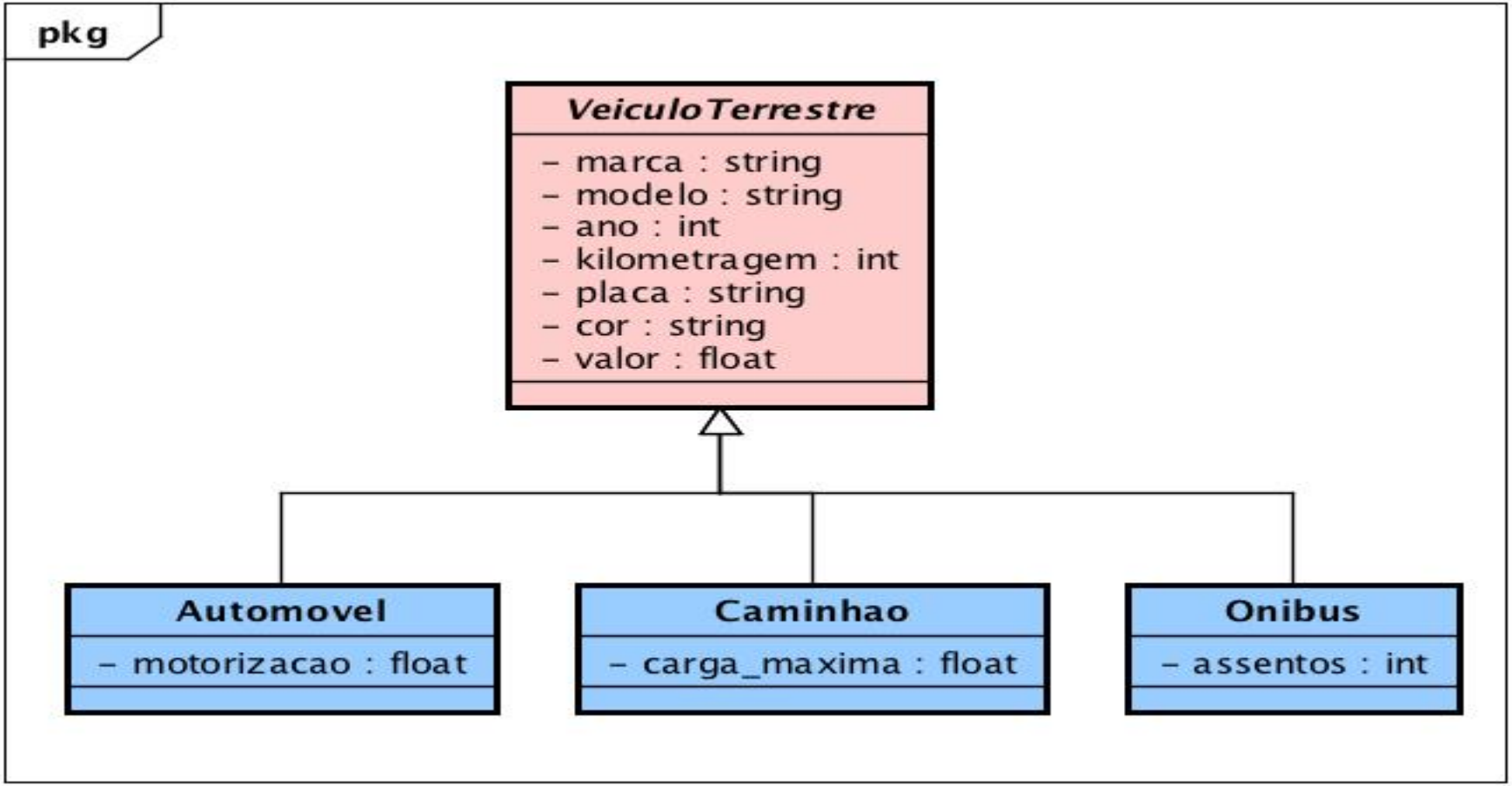
```
e instanceof Pessoa
```
TRUE

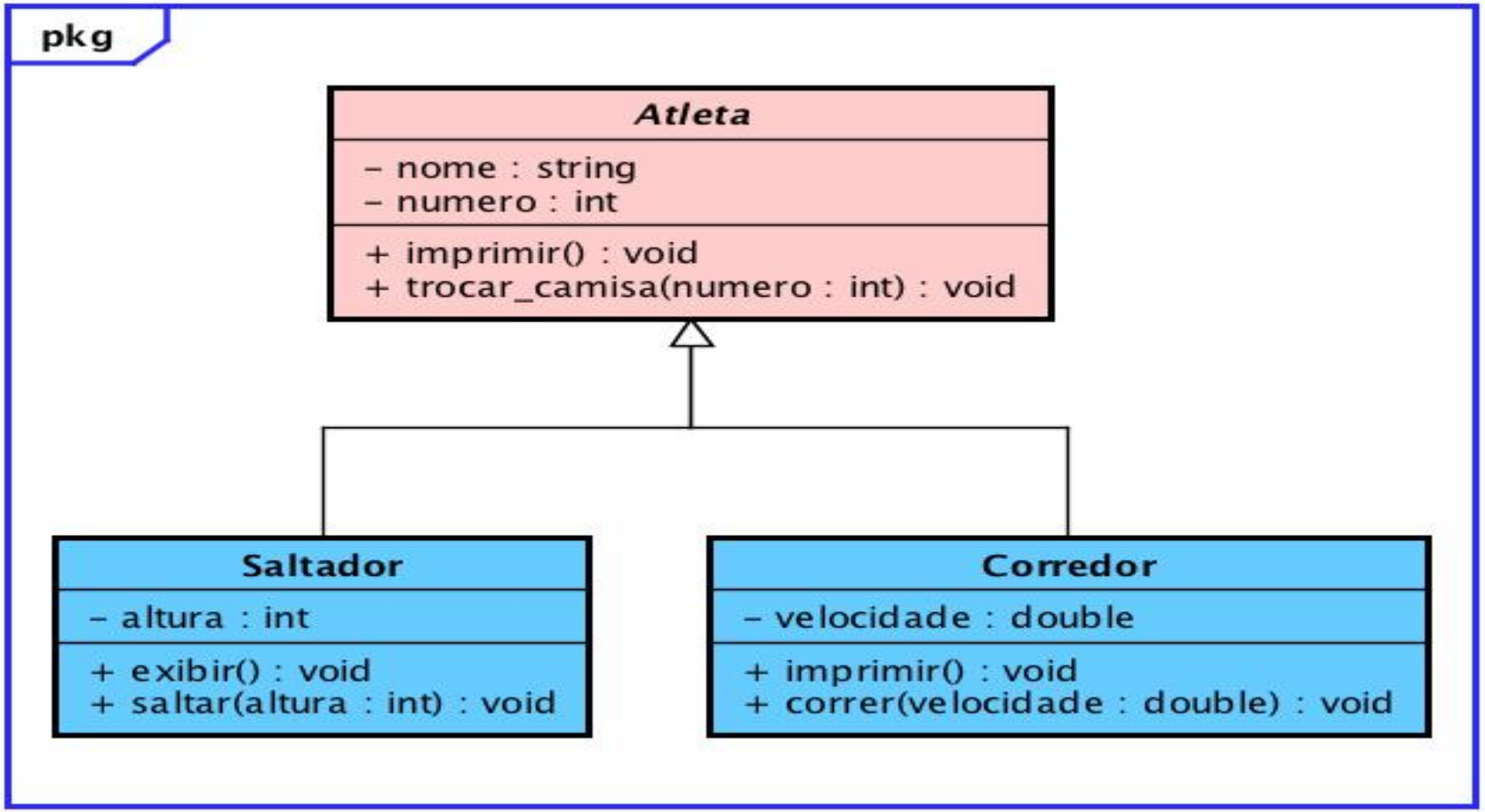
```
Pessoa p = new Professor();
```

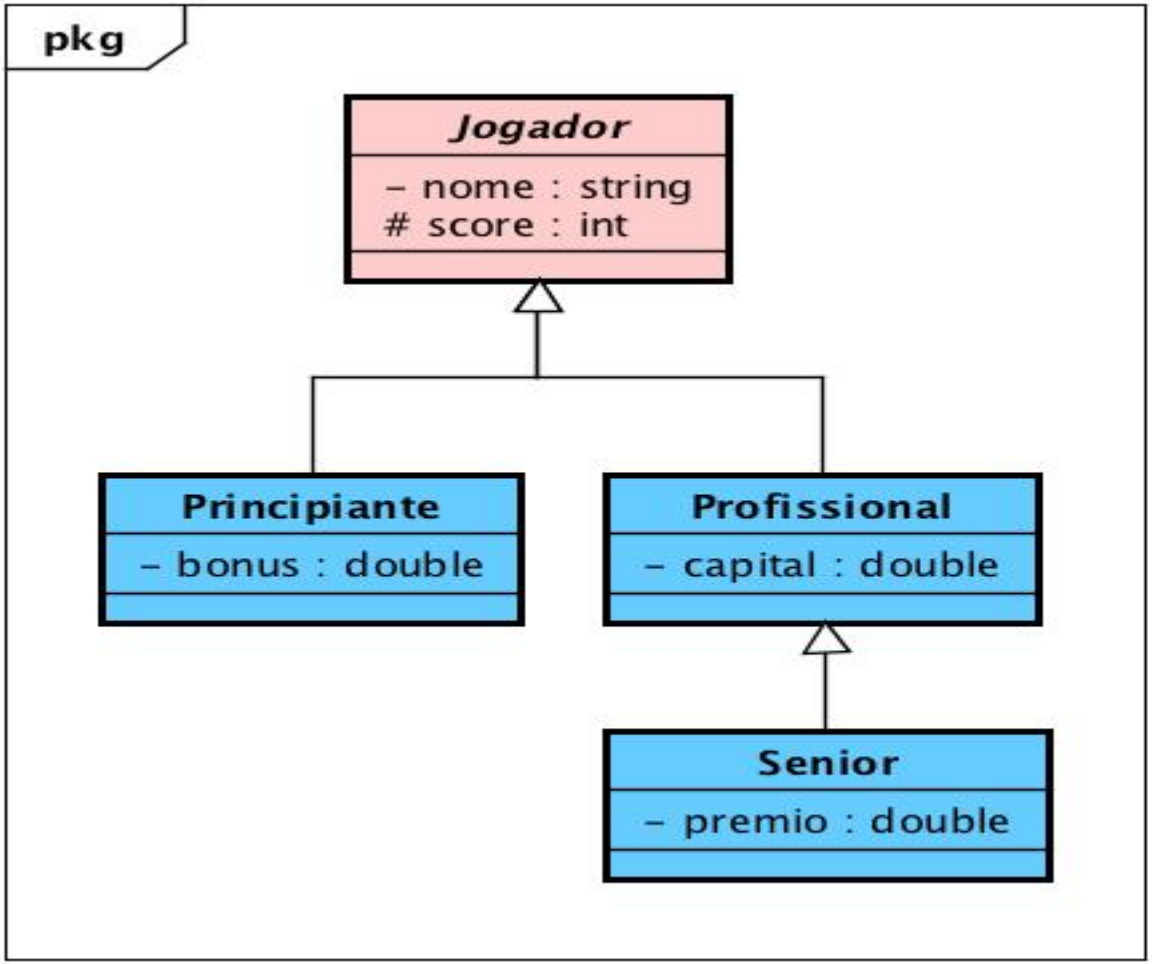
```
p instanceof Professor
```
TRUE

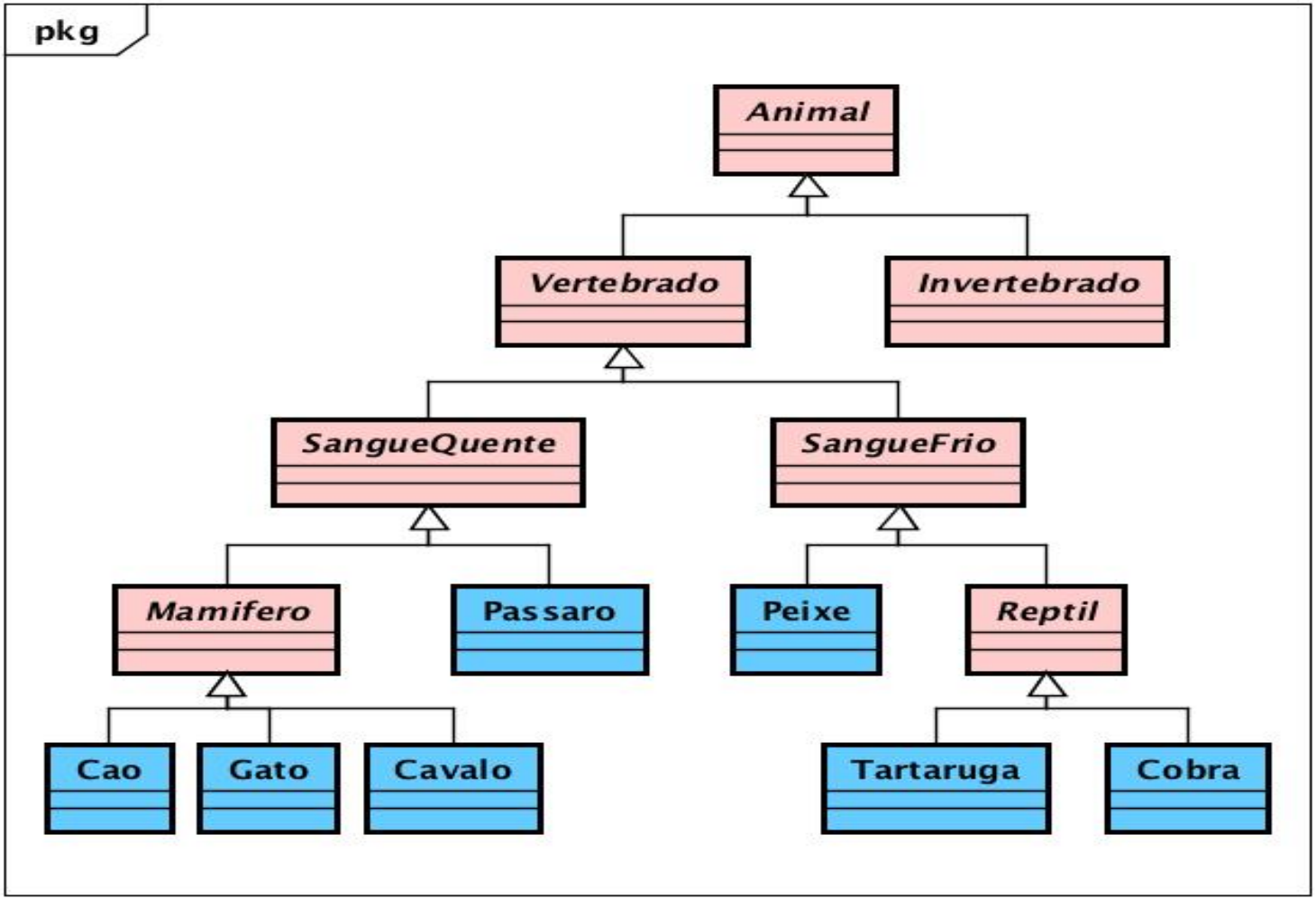
```
p instanceof Pessoa
```
TRUE

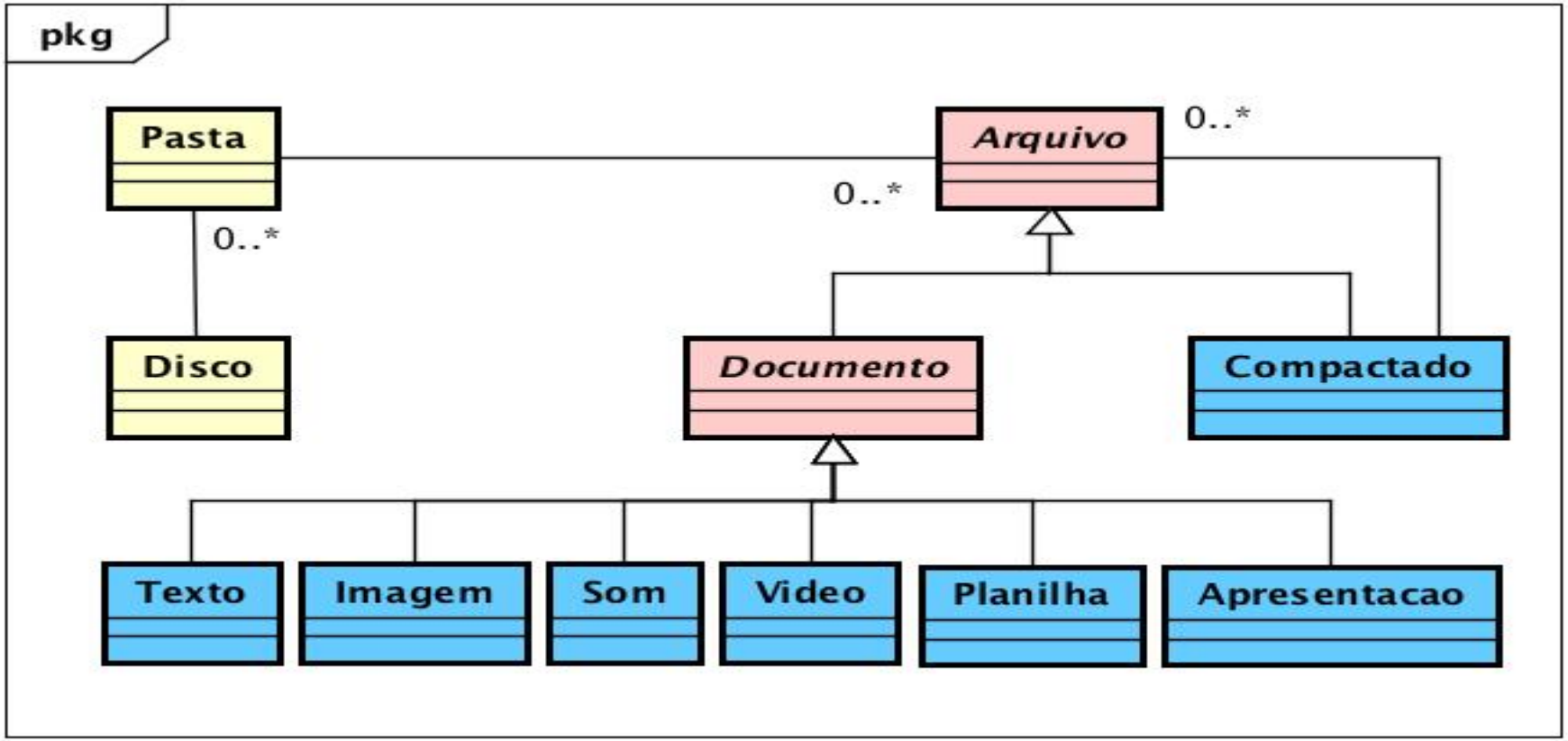
REVISANDO OS EXEMPLOS ANTERIORES



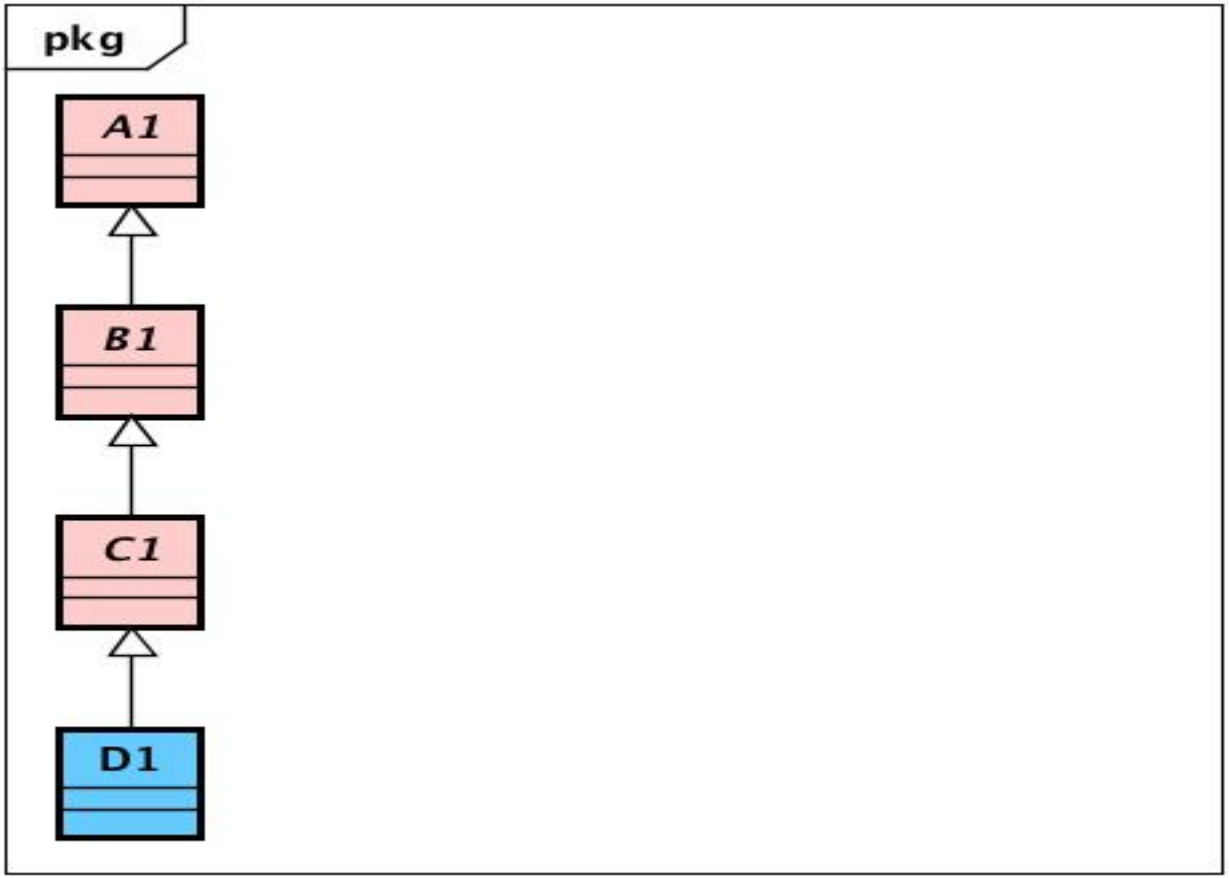


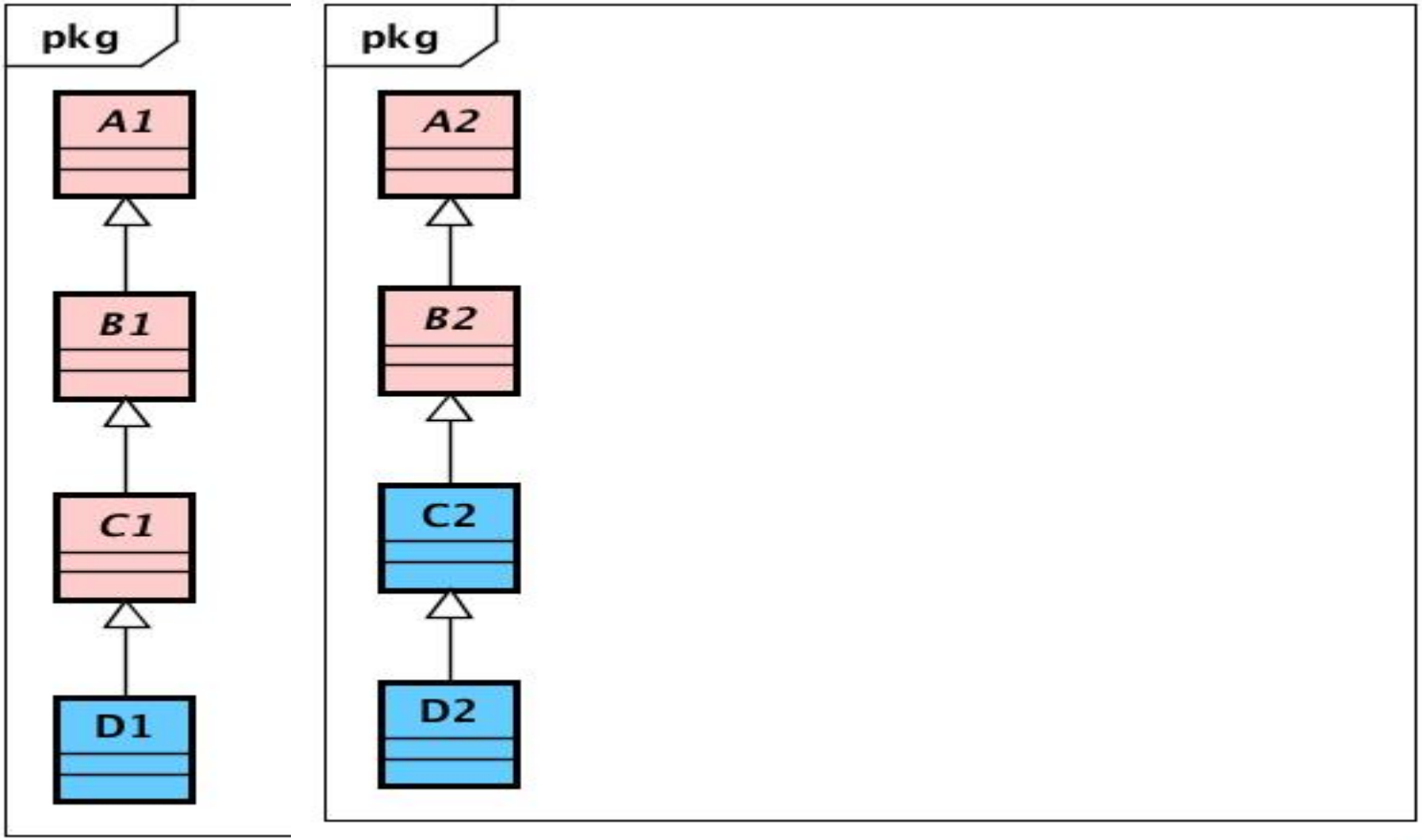


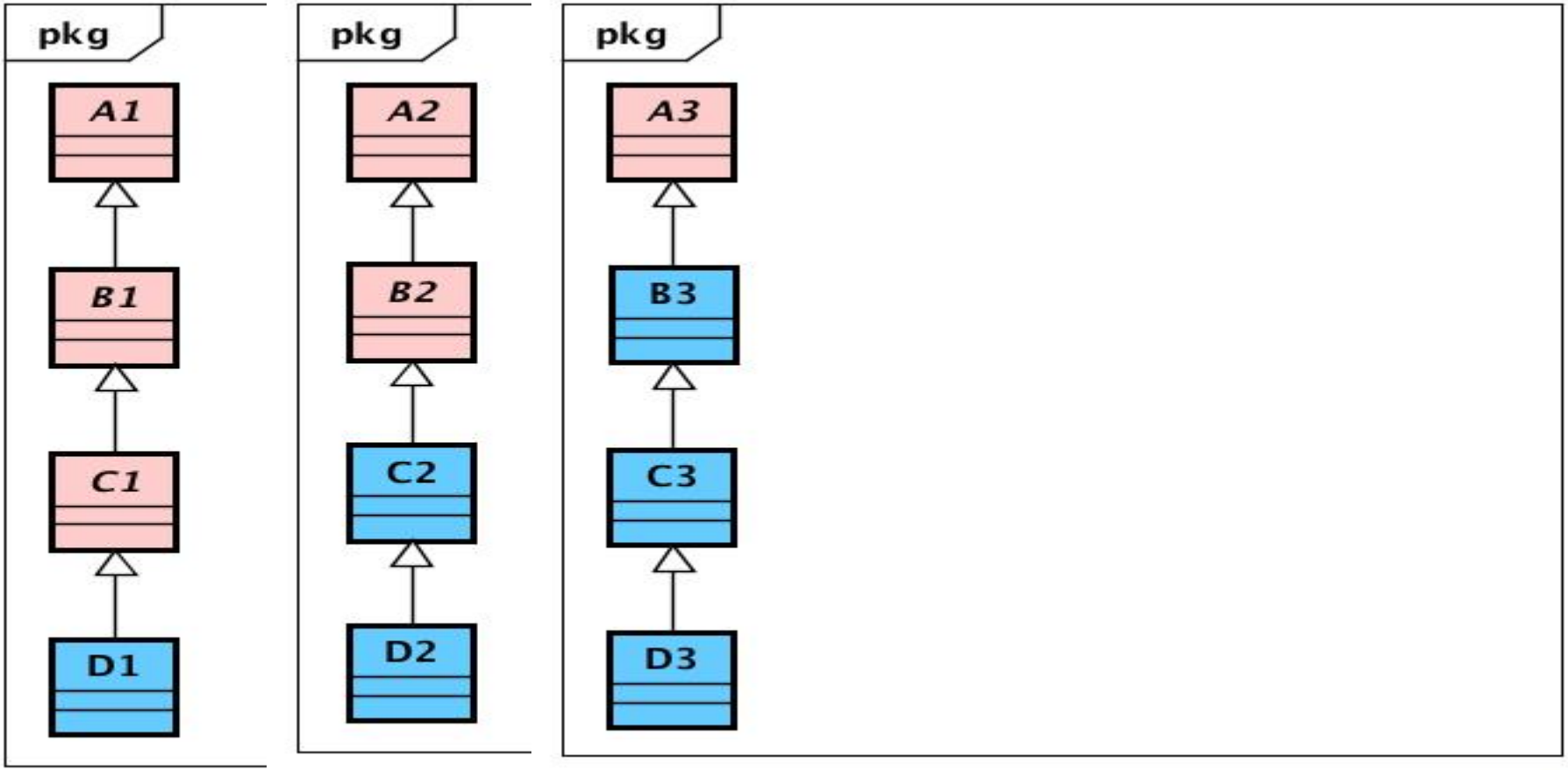


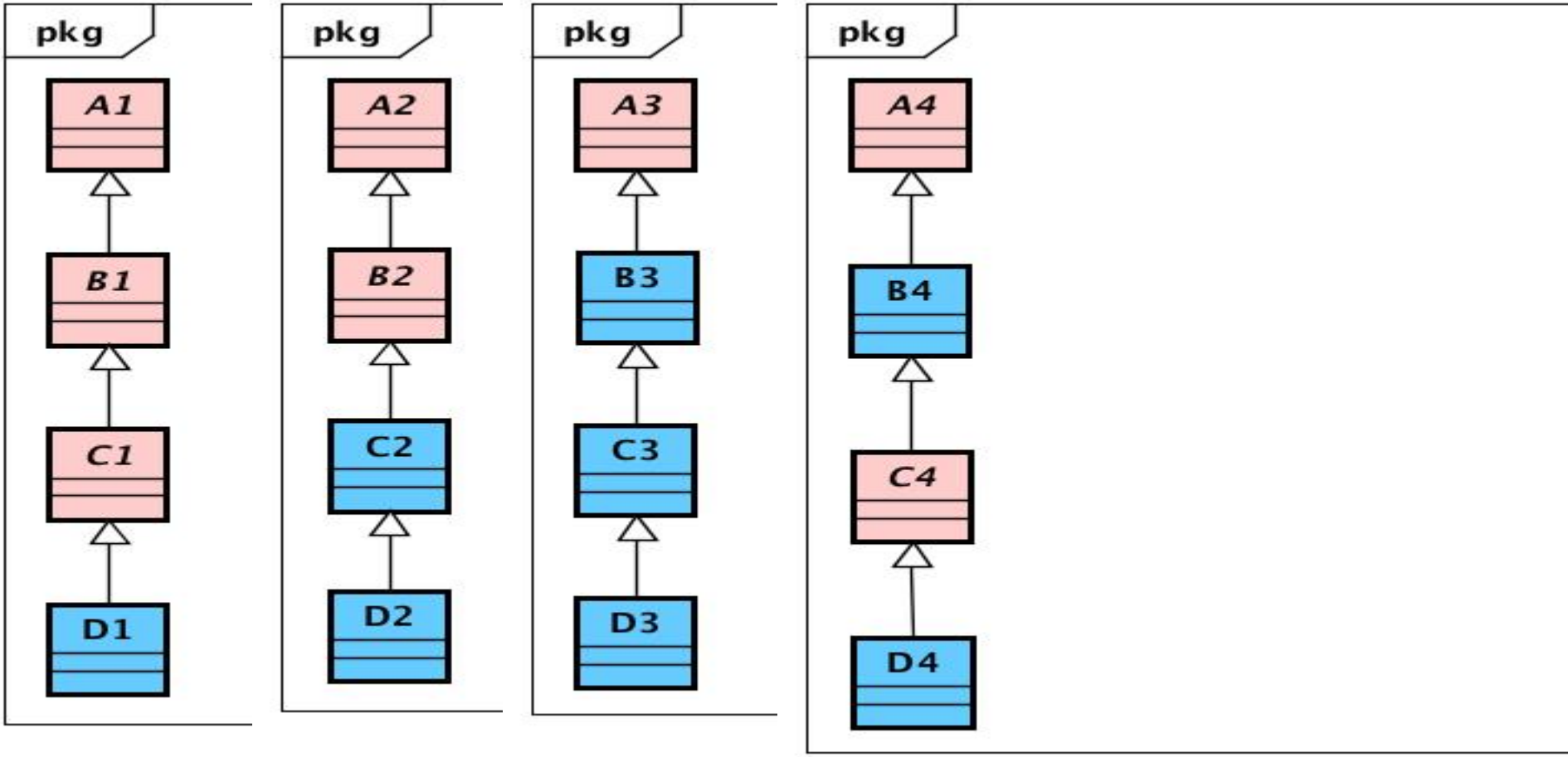


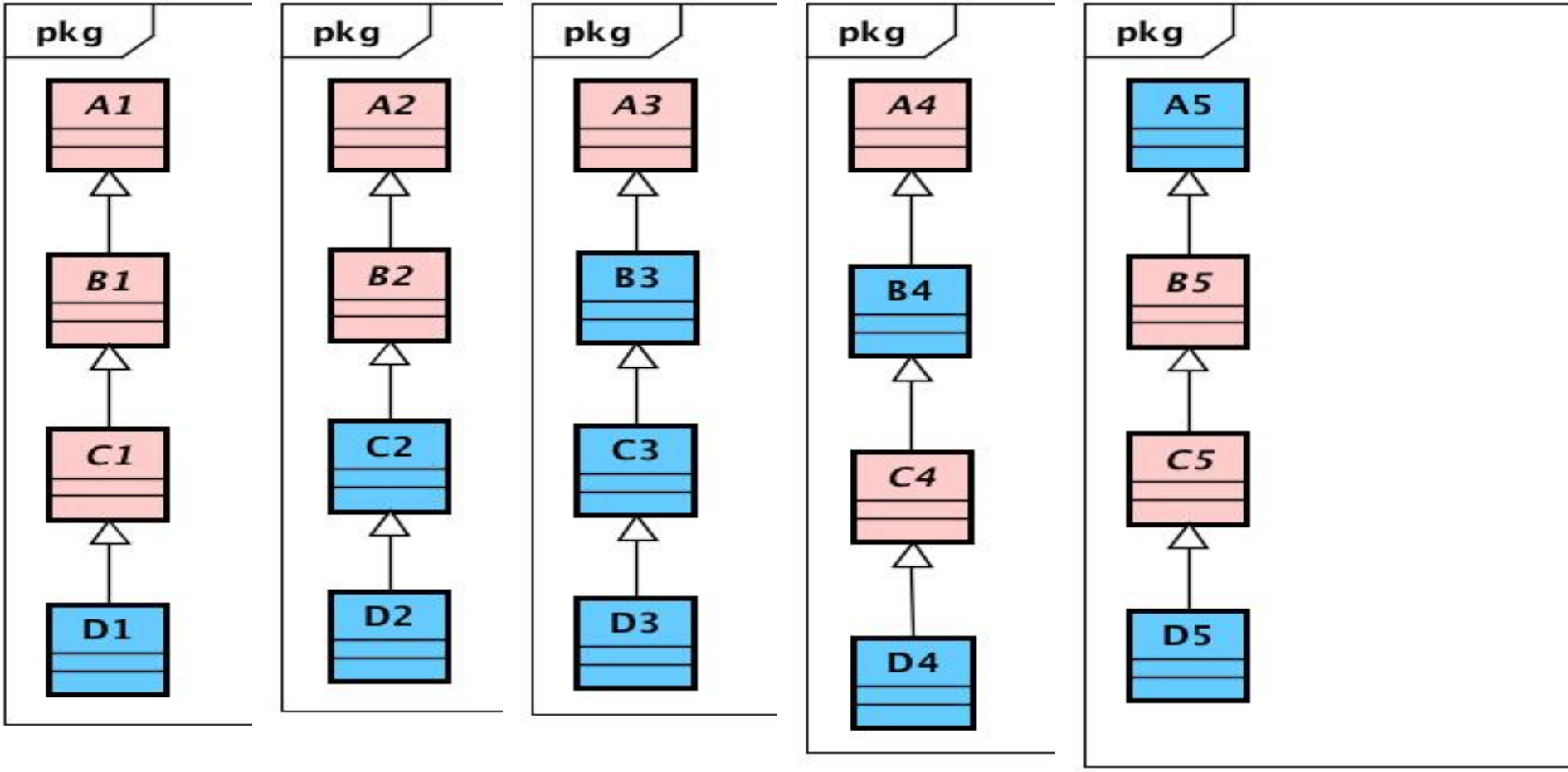
CLASSES ABSTRATAS PODEM APARECER EM QUALQUER NÍVEL DA HIERARQUIA DE CLASSES.

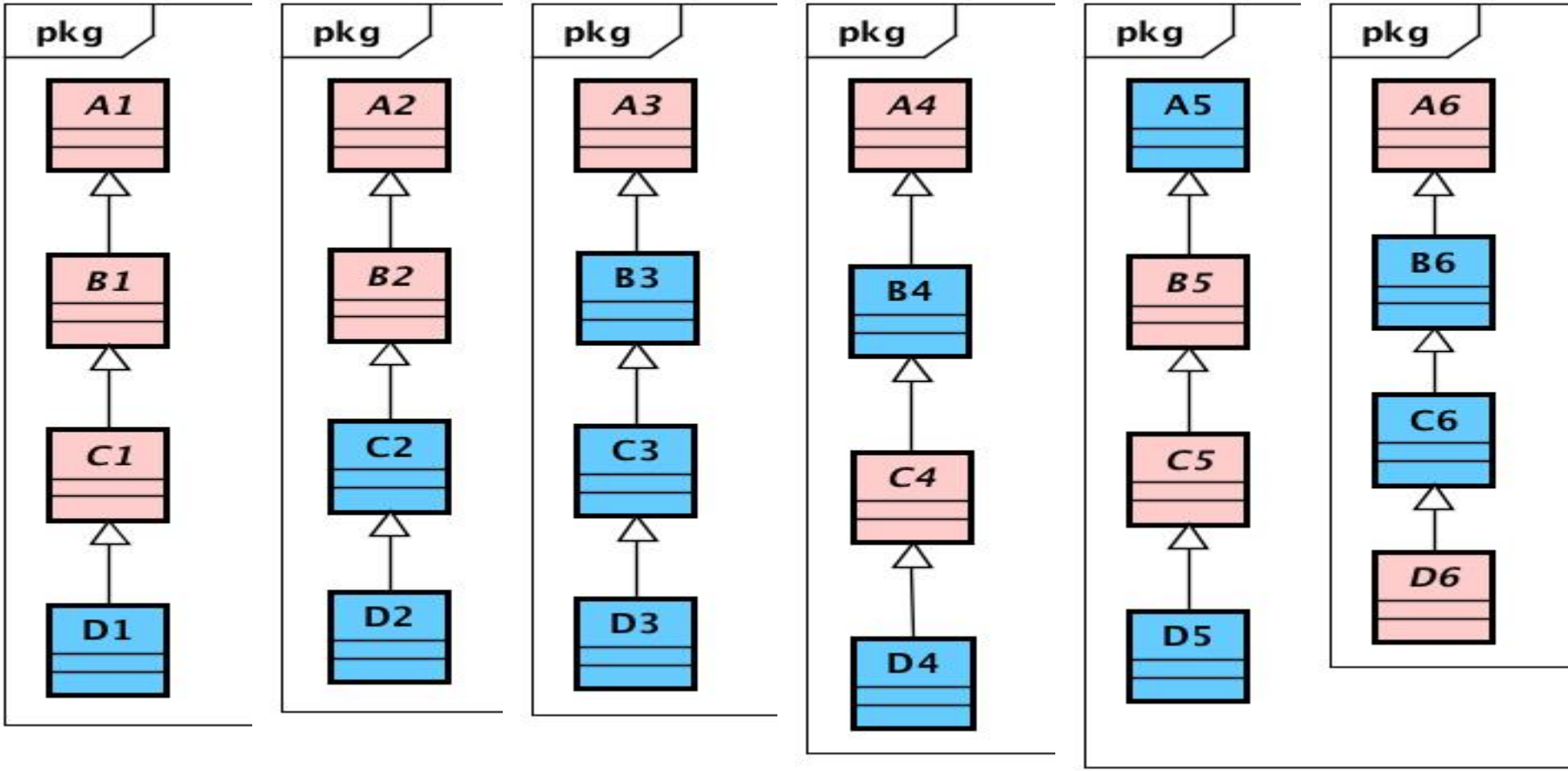








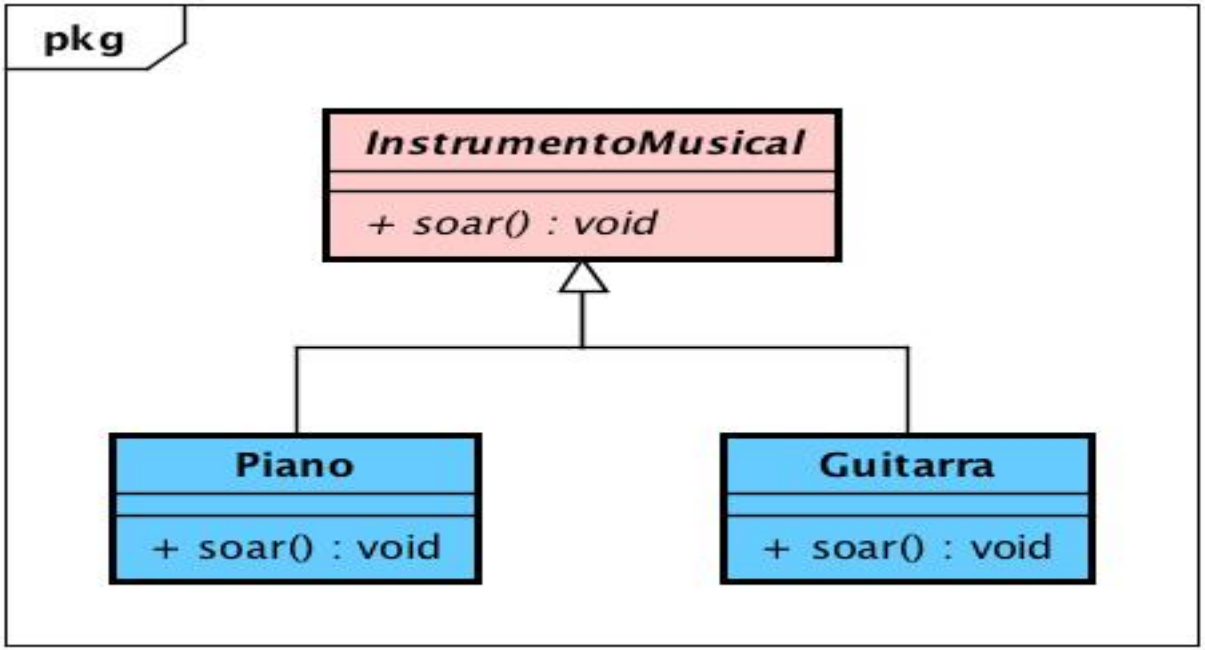

pkg
A1
B1
C1
D1
pkg
A2
B2
C2
D2
pkg
A3
B3
C3
D3
pkg
A4
B4
C4
D4
pkg
A5
B5
C5
D5


pkg
A1
B1
C1
D1
pkg
A2
B2
C2
D2
pkg
A3
B3
C3
D3
pkg
A4
B4
C4
D4
pkg
A5
B5
C5
D5
pkg
A6
B6
C6
D6

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).
- **Não** 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.

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 = 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();

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 = 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 = g;
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 = g;
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

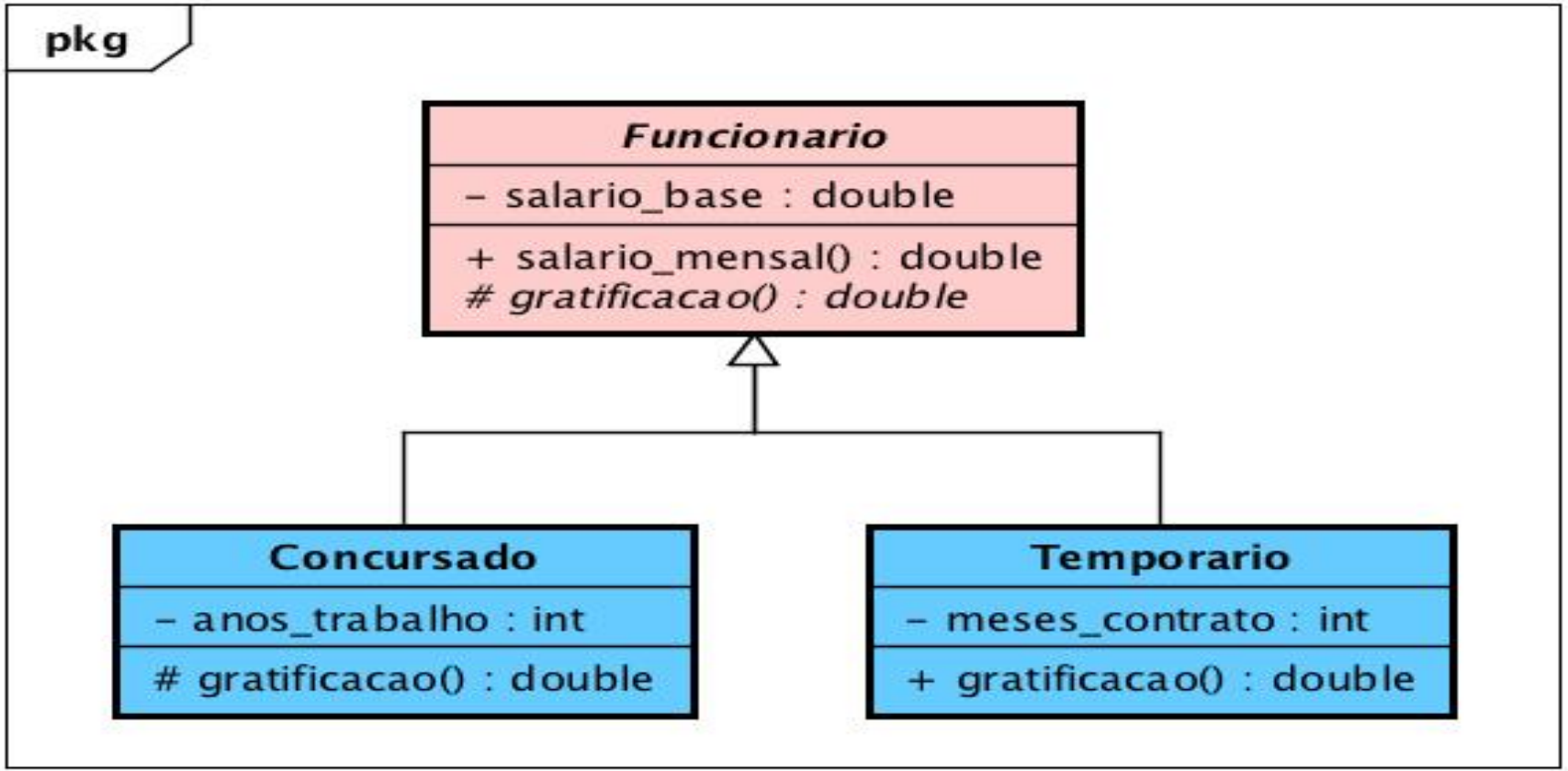
```
Piano p = new Piano();
Guitarra g = new Guitarra();

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() );
```

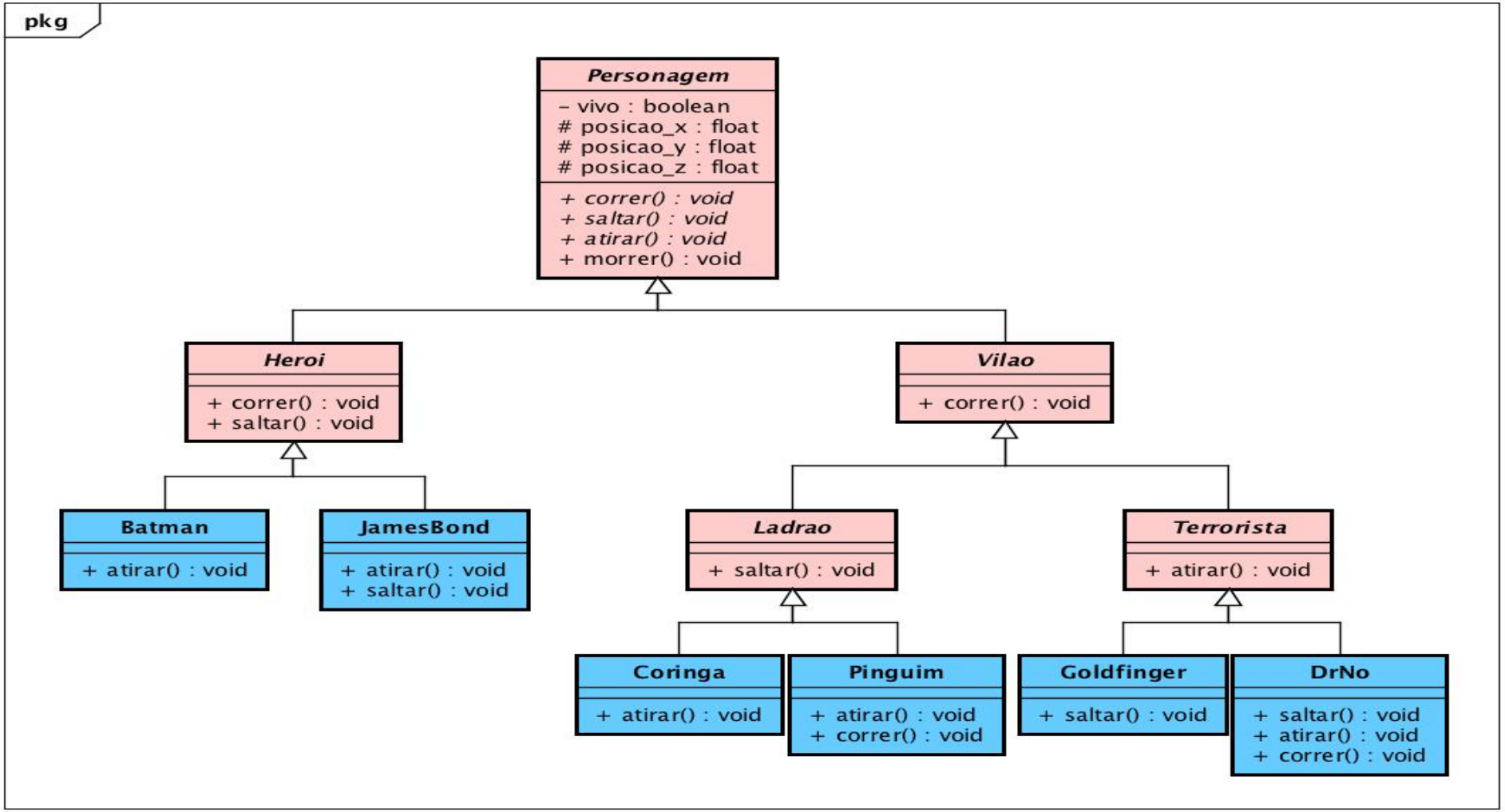
Quais os valores impressos?

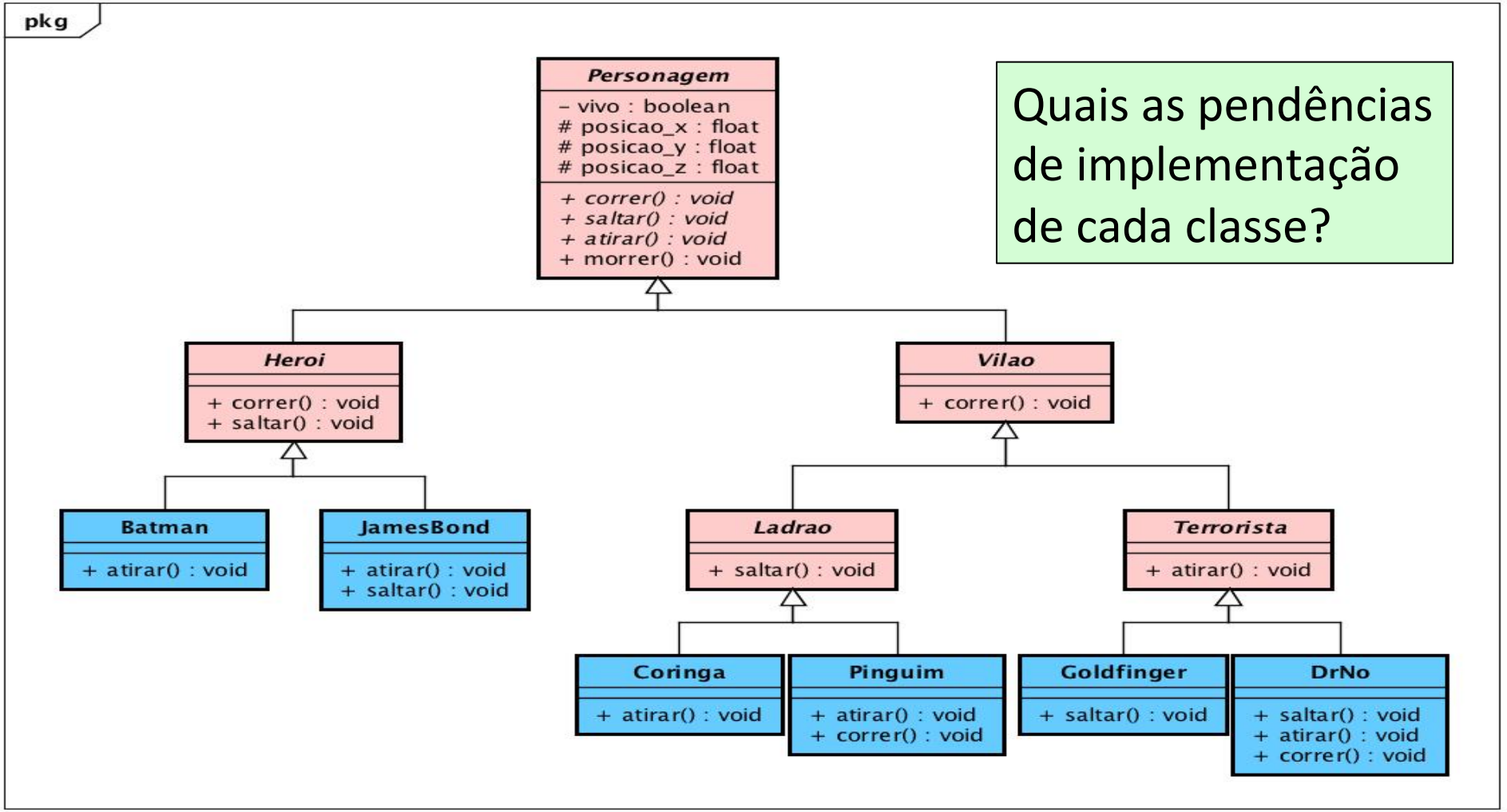
```
Concursado joao = new Concursado(4000, 5);

Temporario jose = new Temporario( 2000, 12);

System.out.println( joao.salario_mensal() );

System.out.println( jose.salario_mensal() );
```

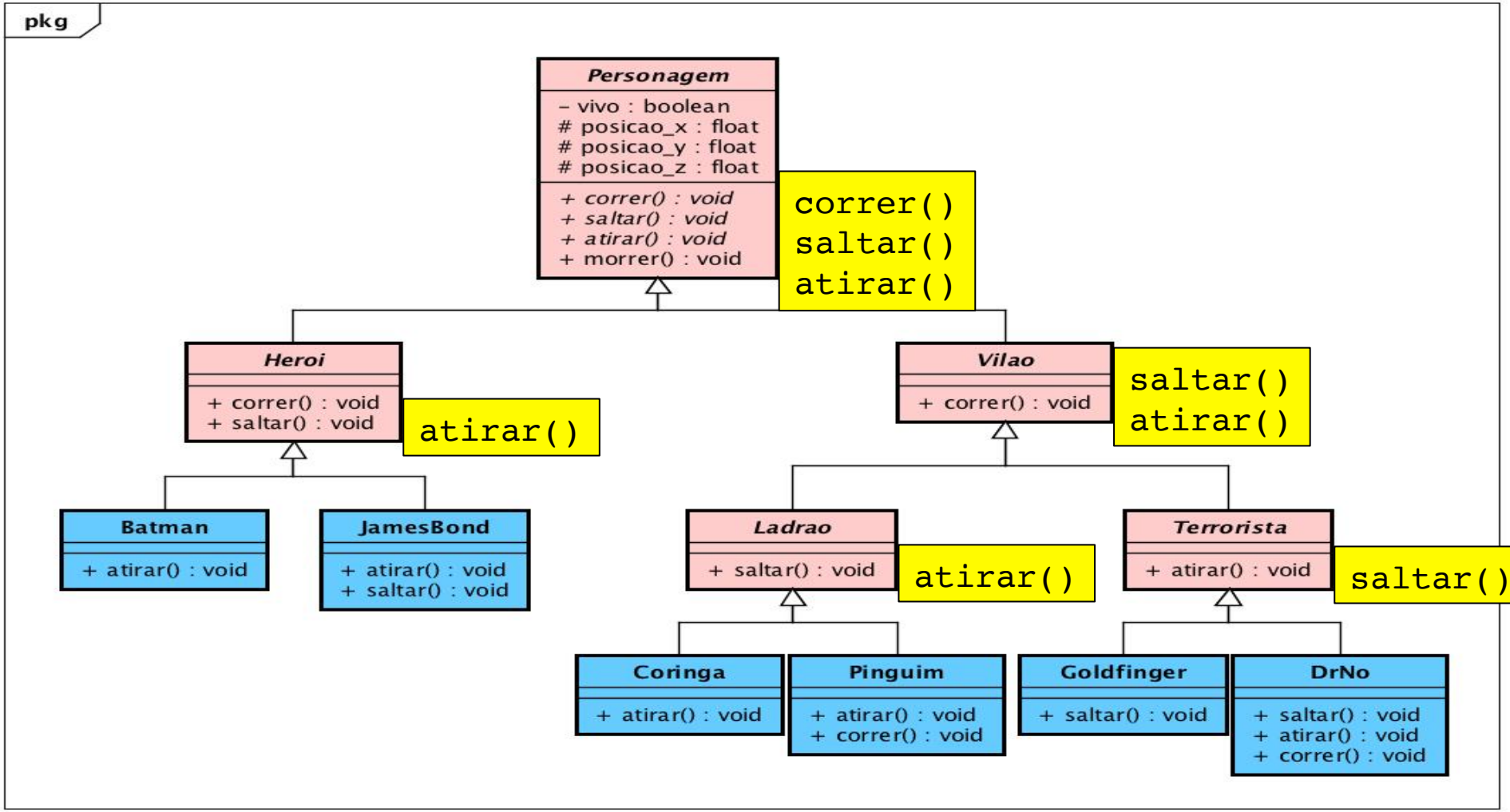
Quais os valores impressos?

```
4500.00
2120.00
```

UMA CLASSE CONCRETA NÃO PODE TER QUALQUER PENDÊNCIA DE IMPLEMENTAÇÃO DE MÉTODO ABSTRATO HERDADO.



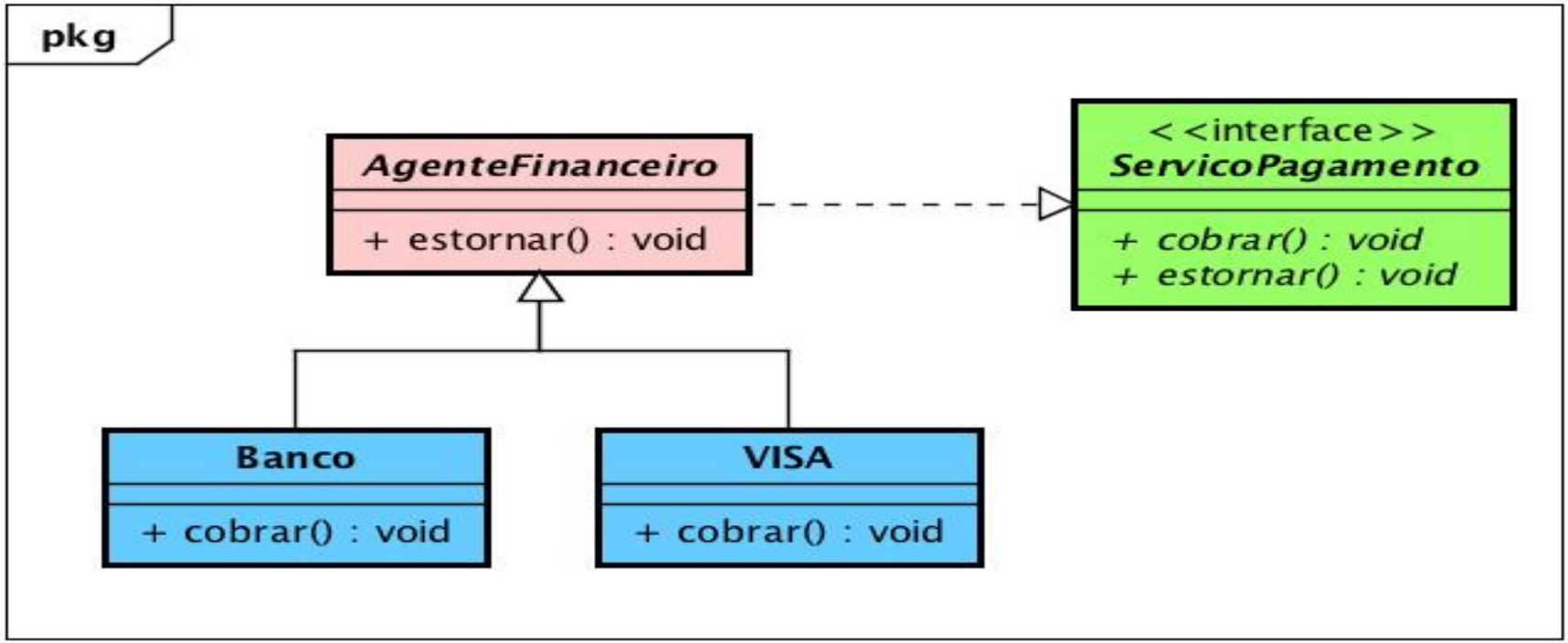




INTERFACE

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.



```
interface ServicoPagamento {
    void cobrar(Lojista lojista,
                Consumidor consumidor,
                double valor);
    void estornar(Lojista lojista,
                  Consumidor consumidor,
                  double valor);
}
```

```
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);
    }
}
```

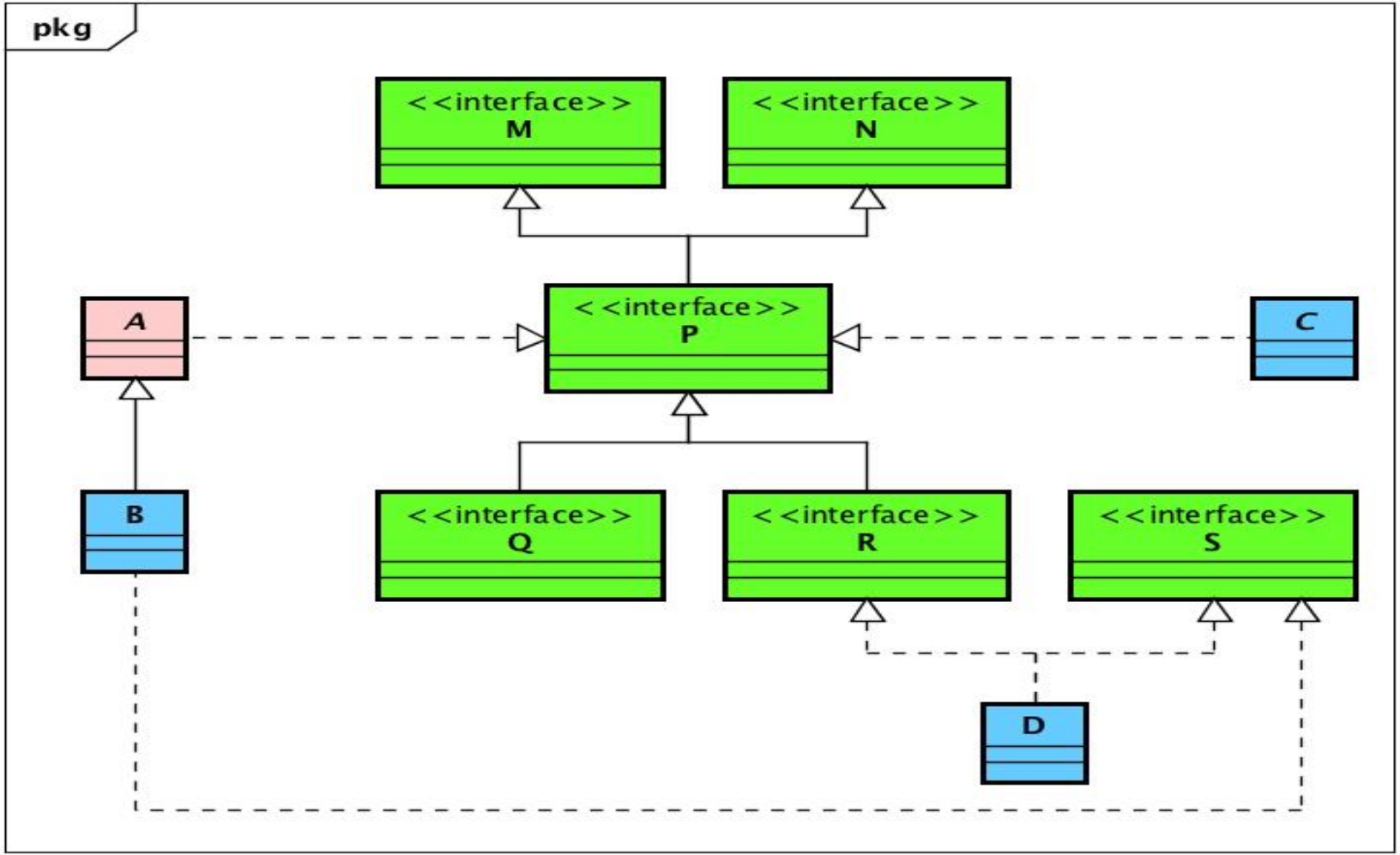
```
abstract class Correntista {
    private double saldo;
    public Correntista(double saldo) {this.saldo = saldo;}
    public void debitar(double valor) {saldo = saldo – valor;}
    public void depositar(double valor) {saldo = saldo + valor;}
}
```

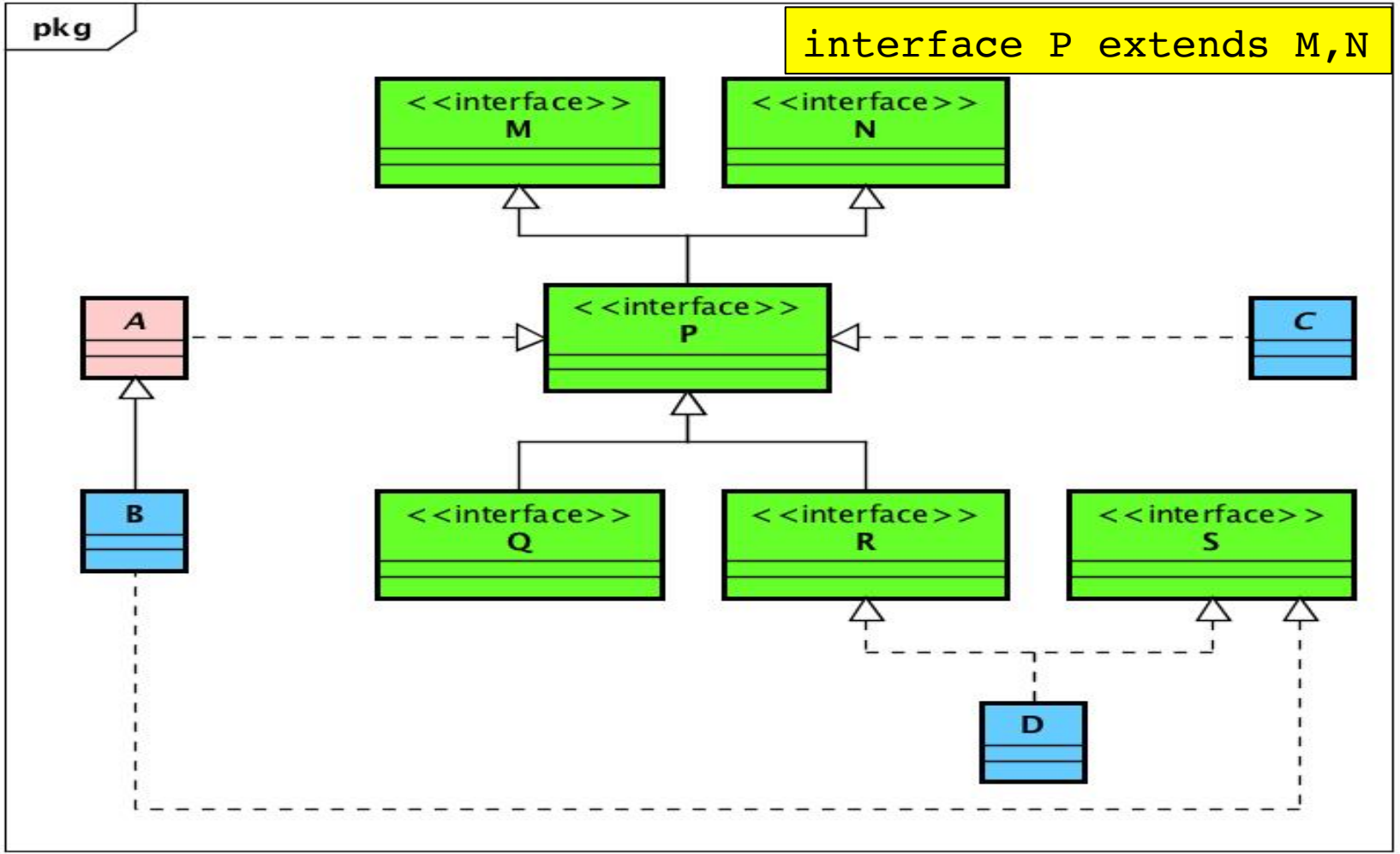
```
class Consumidor extends Correntista {
   private int pontos;
   public Consumidor(double saldo) {
      super(saldo);
      pontos = 0;
   }
   public void bonificar(int pontos) {
      this.pontos = this.pontos + pontos;
   }
}
```

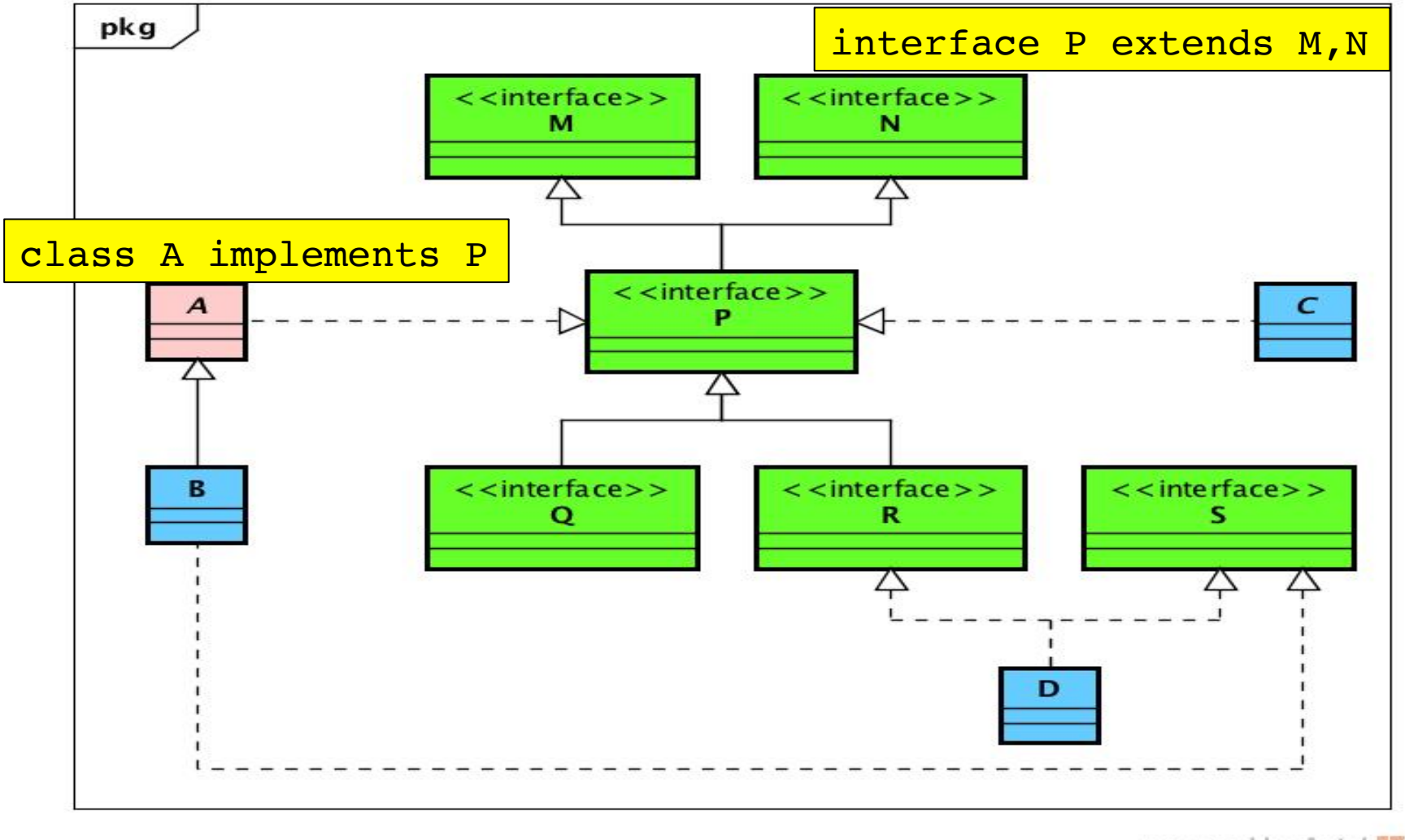
```
class Lojista extends Correntista {
   private char categoria;
   public Lojista(double saldo, char categoria) {
      super(saldo);
      this.categoria = categoria;
   }
   public char categoria() {
      return categoria;
   }
}
```

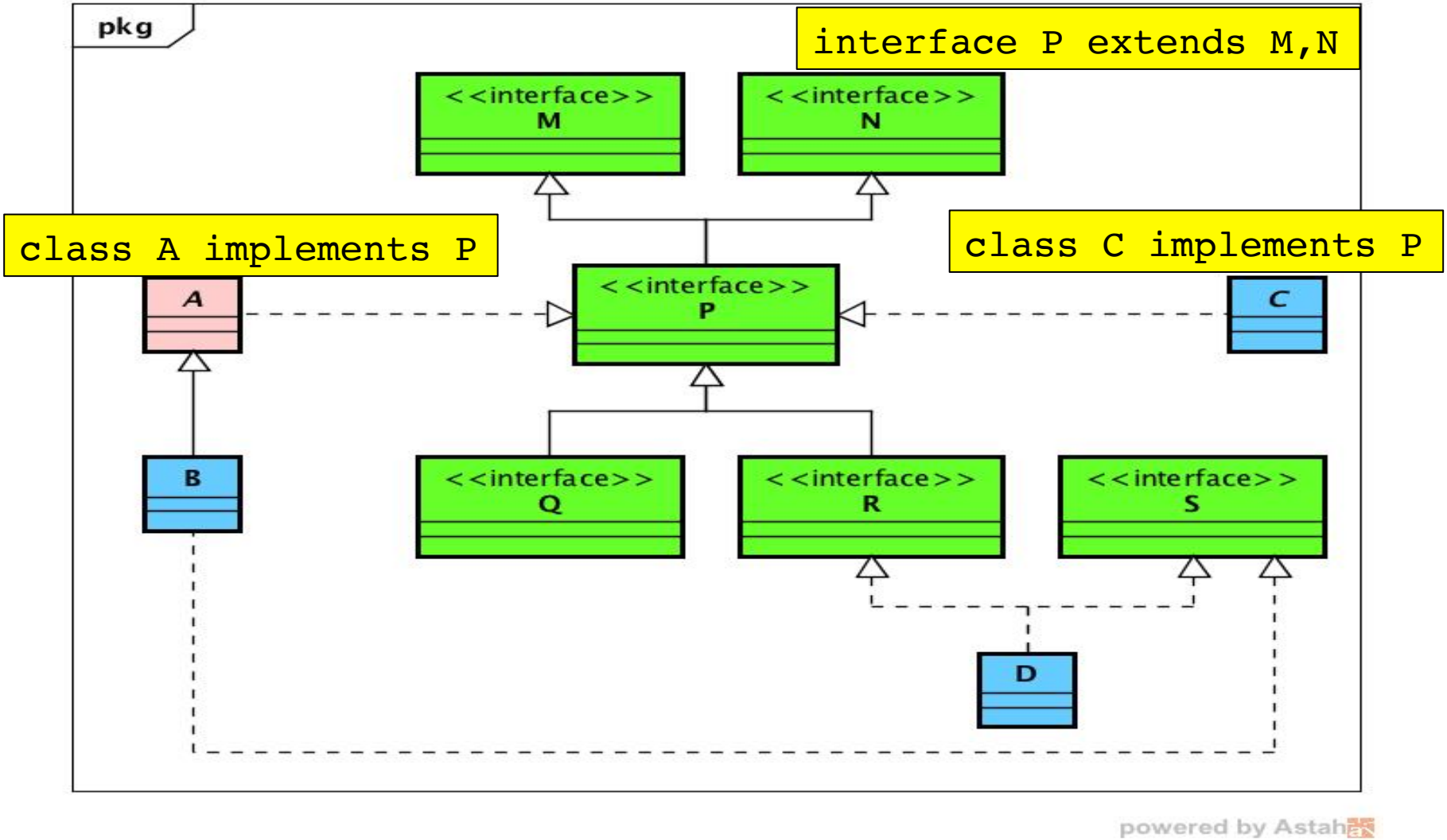
```
Banco itau = new Banco("Itau", 0.02);
VISA visa = new VISA();

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








pkg
interface P extends M,N
<<interface>>
M
<<interface>>
N
class A implements P
class C implements P
A
<<interface>>
P
C
B
<<interface>>
Q
<<interface>>
R
<<interface>>
S
D
powered by Astah