POO

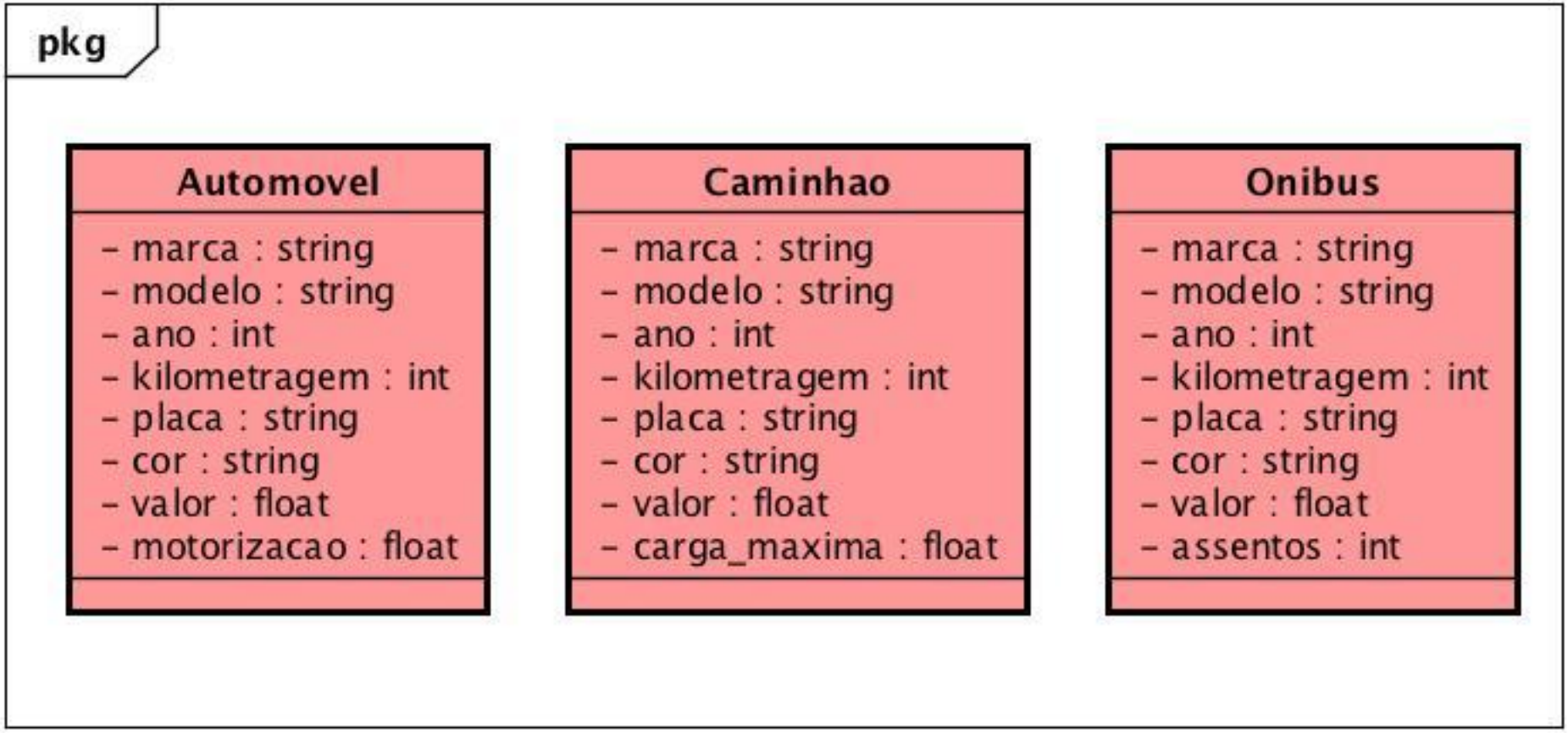
Hierarquia de classes

Prof. Alcides Calsavara
PUCPR

Conceitos

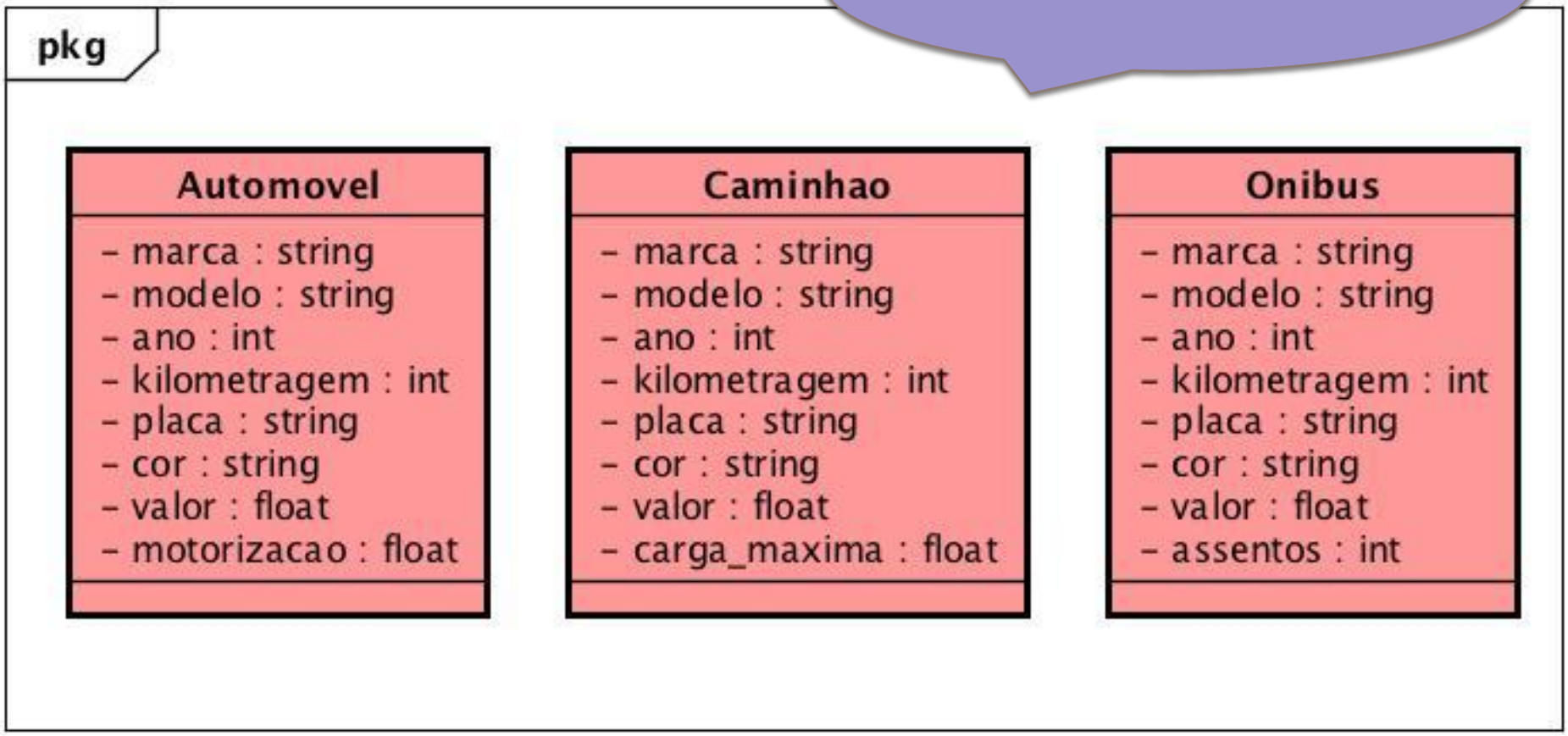
1. Superclasse
2. Subclasse
3. Generalização
4. Especialização
5. Herança
6. Encadeamento de construtores
7. Sobrecarga de método
8. Visibilidade protected

Modelo de Classes - UML



Modelo de Classes - UML

REDUNDÂNCIA



Implementação - Java

```
class Automovel {

    private String marca;
    private String modelo;
    private int ano;
    private int kilometragem;
    private String placa;
    private String cor;
    private float valor;
    private float motorizacao;

    // métodos …
}
```

Implementação - Java

```
class Caminhao {

    private String marca;
    private String modelo;
    private int ano;
    private int kilometragem;
    private String placa;
    private String cor;
    private float valor;
    private float carga_maxima;

    // métodos …
}
```

Implementação - Java

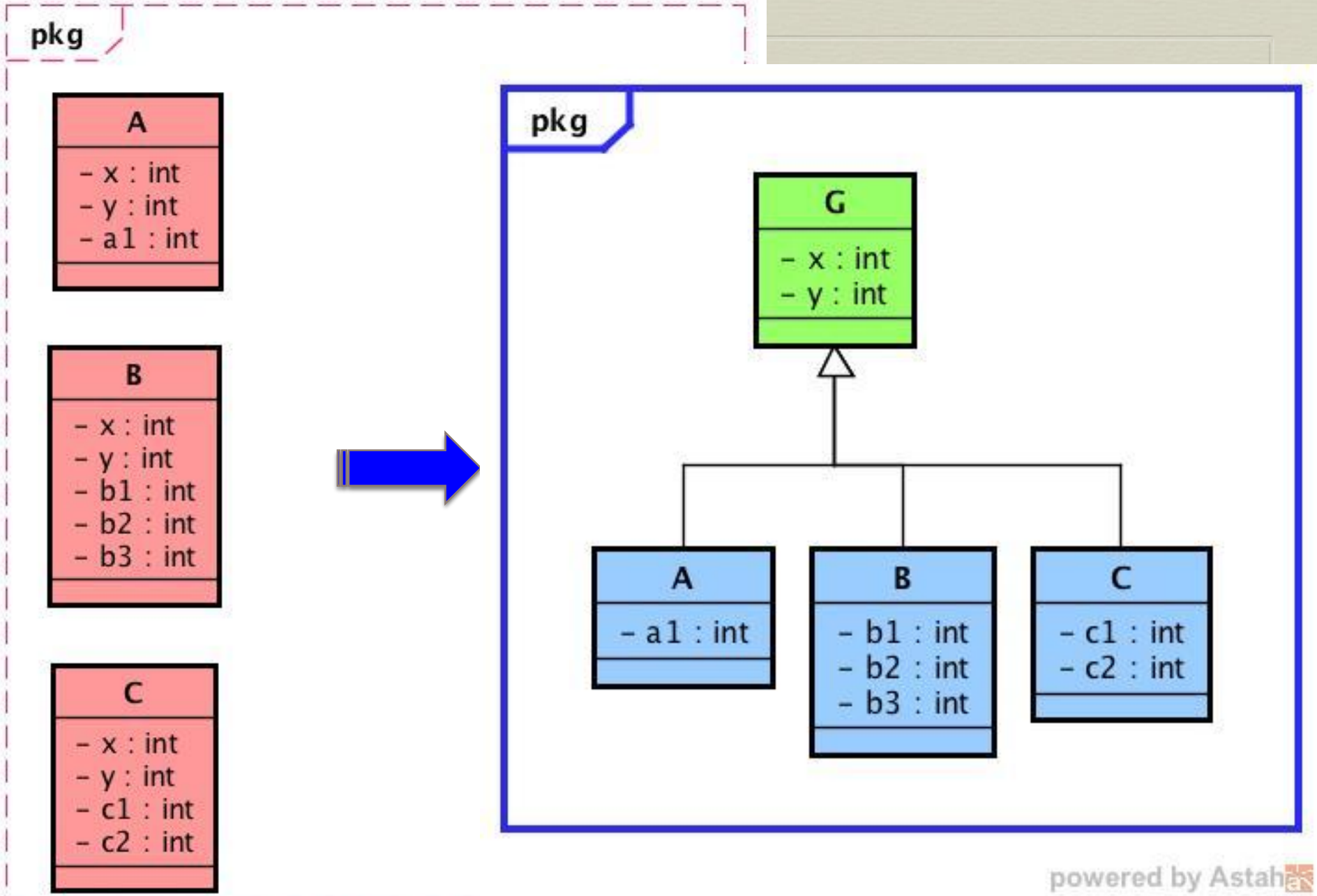
```
class Onibus {

    private String marca;
    private String modelo;
    private int ano;
    private int kilometragem;
    private String placa;
    private String cor;
    private float valor;
    private int assentos;

    // métodos …
}
```

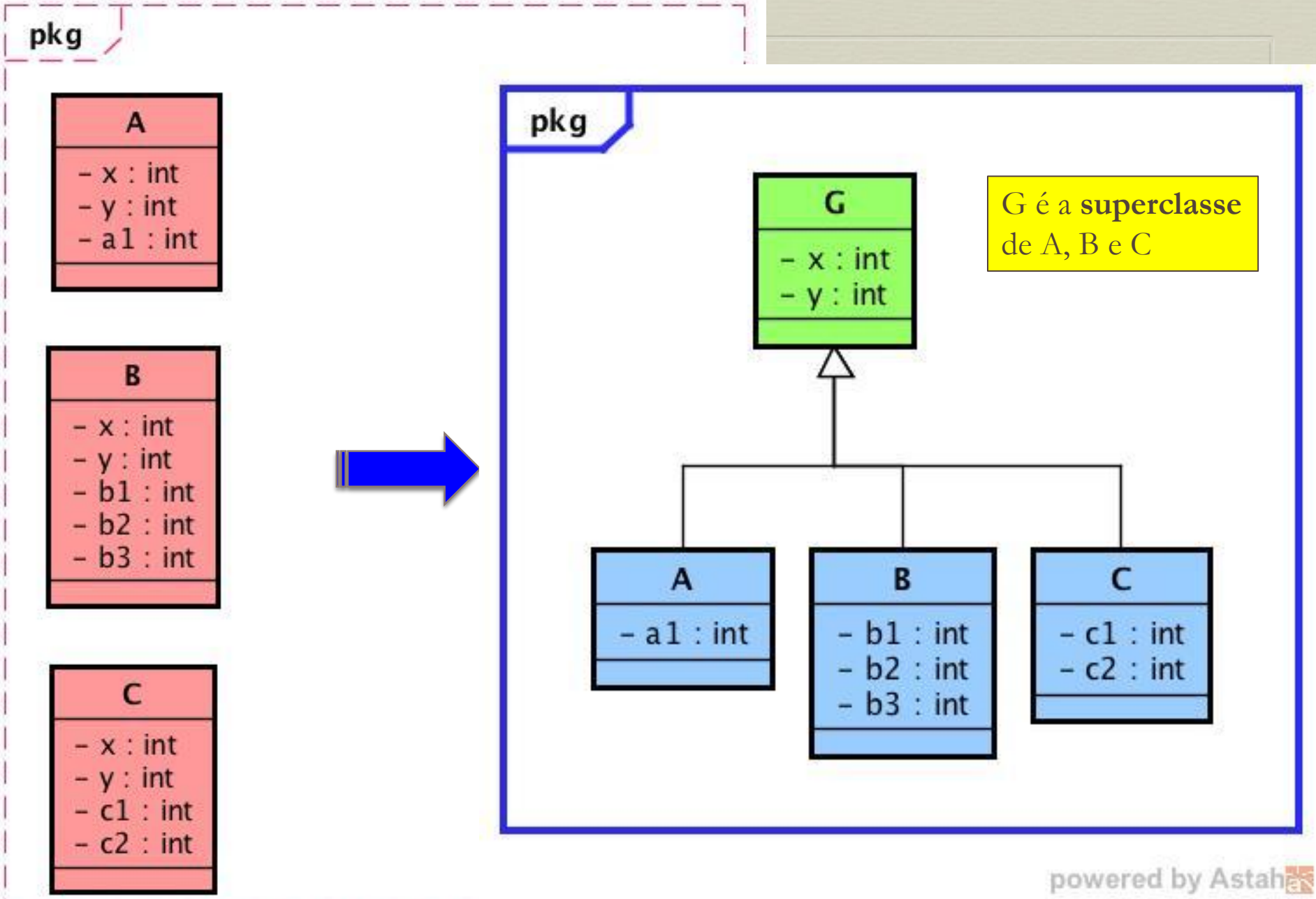
Efeitos nocivos da redundância

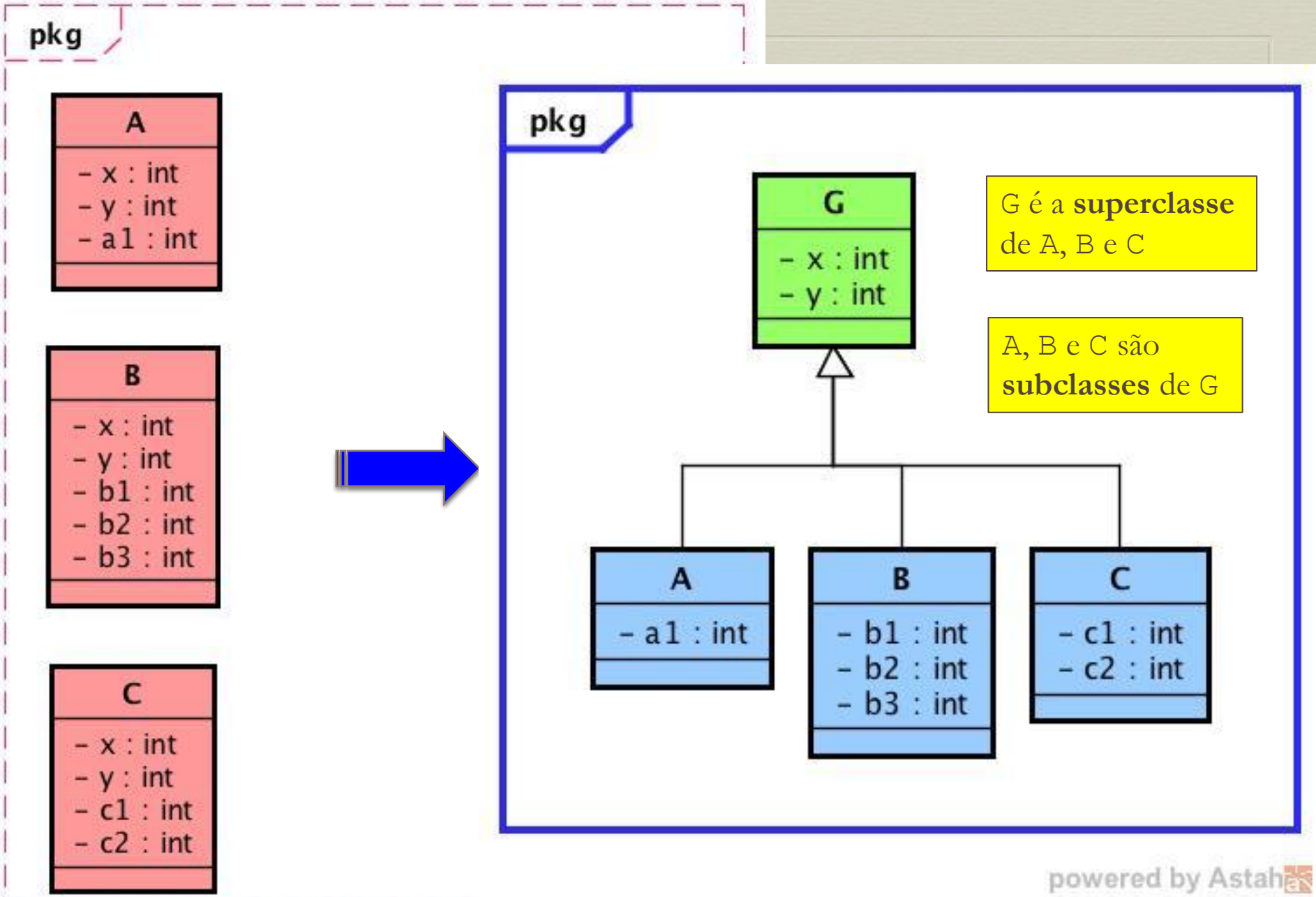
1. **Rápido crescimento do volume de código**
   - A criação de uma nova classe implica em replicar parte significativa do código.
   - Por exemplo, a classe `Motocicleta` replicaria quase todos os atributos da classe `Automovel`.
2. **Manutenção complexa**
   - Qualquer modificação de um membro comum entre as classes implica em atualizar o código de cada classe.
   - Por exemplo, a alteração do tipo do atributo `valor` de `float` para `double` exigiria a atualização do código das três classes: `Automovel`, `Caminhao` e `Onibus`.

Eliminação da redundância

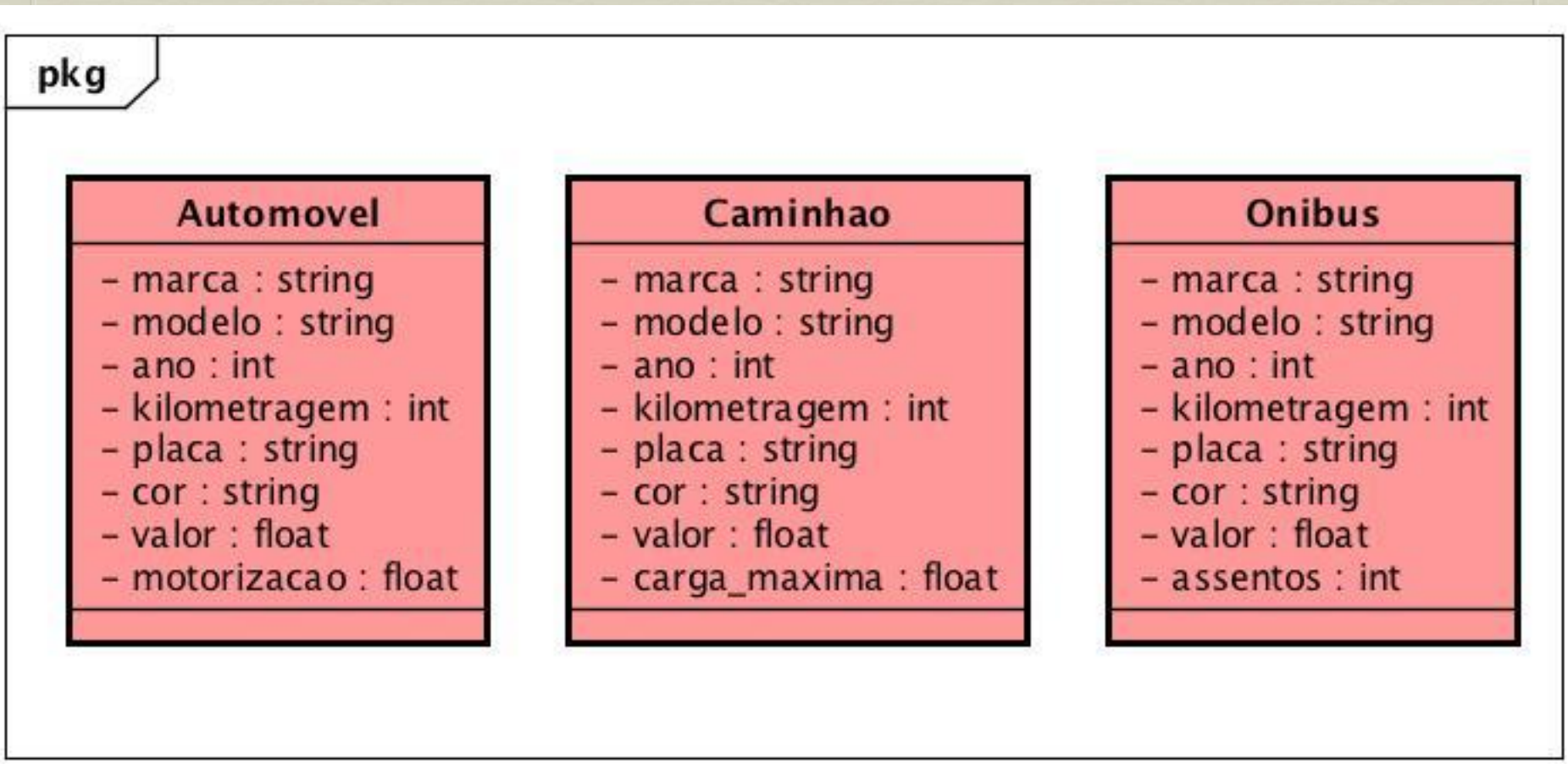
1. Criar uma nova classe contendo os membros comuns das classes existentes.
2. Remover os membros comuns das classes existentes.
3. Estabelecer um relacionamento de hierarquia entre cada classe existente e a nova classe criada:
   - Cada classe existente torna-se uma **subclasse** da nova classe
   - A nova classe torna-se a **superclasse** de cada classe existente



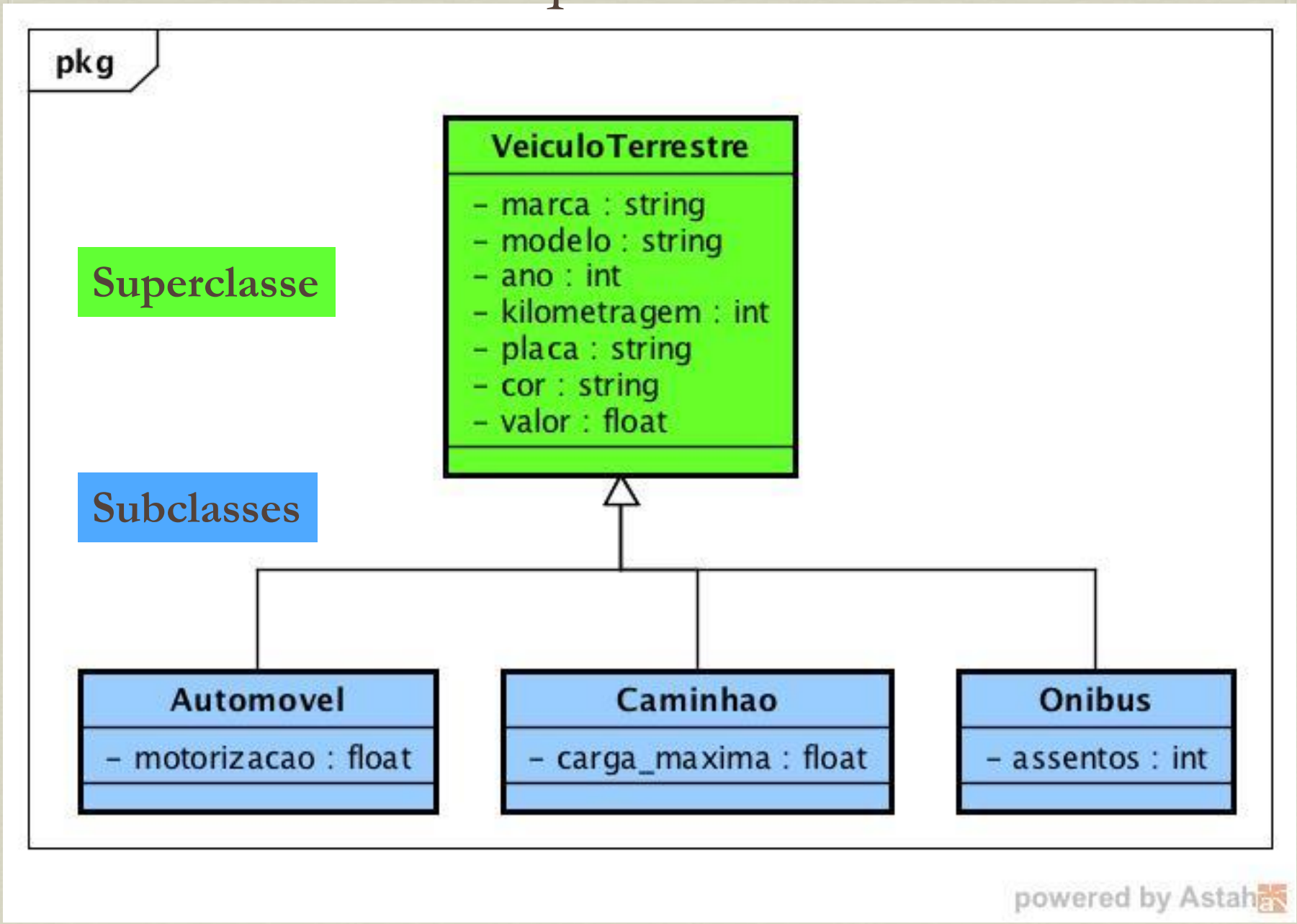




Eliminando redundância ...



Hierarquia de classes



Implementação - Java

```
class VeiculoTerrestre {

    private String marca;
    private String modelo;
    private int ano;
    private int kilometragem;
    private String placa;
    private String cor;
    private float valor;

    // métodos
}
```

Implementação - Java

```
class Automovel extends VeiculoTerrestre {

    private float motorizacao;

    // métodos
}
```

O termo **extends** é usado para estabelecer o relacionamento de hierarquia entre duas classes.

class *subclasse* **extends** *superclasse*

Implementação - Java

```
class Caminhao extends VeiculoTerrestre {

    private float carga_maxima;

    // métodos
}
```

Implementação - Java

```
class Onibus extends VeiculoTerrestre {

    private int assentos;

    // métodos
}
```

Terminologia e efeitos

```
class Y extends X
```

Terminologia e efeitos

```
class Y extends X
```

- X é **superclasse** de Y
- Y é **subclasse** de X

Terminologia e efeitos

class Y **extends** X

- X é **superclasse** de Y
- Y é **subclasse** de X

- X é a **classe base** de Y
- Y é uma **classe derivada** de X

Terminologia e efeitos

class Y **extends** X

- X é **superclasse** de Y
- Y é **subclasse** de X

- X é a **classe base** de Y
- Y é uma **classe derivada** de X

- Y é uma **extensão** de X
- Y **estende** X

Terminologia e efeitos

class Y **extends** X

- X é **superclasse** de Y
- Y é **subclasse** de X

- X é a **classe base** de Y
- Y é uma **classe derivada** de X

- Y é uma **extensão** de X
- Y **estende** X

- X é uma **generalização** de Y
- Y é uma **especialização** de X

Terminologia e efeitos

class Y **extends** X

- X é **superclasse** de Y
- Y é **subclasse** de X

- X é a **classe base** de Y
- Y é uma **classe derivada** de X

- Y é uma **extensão** de X
- Y **estende** X

- X é uma **generalização** de Y
- Y é uma **especialização** de X

- X e Y possuem um relacionamento de **herança**
- Y **herda** os membros (atributos e métodos) de X

Terminologia e efeitos

class Y **extends** X

- X é **superclasse** de Y
- Y é **subclasse** de X

- X é a **classe base** de Y
- Y é uma **classe derivada** de X

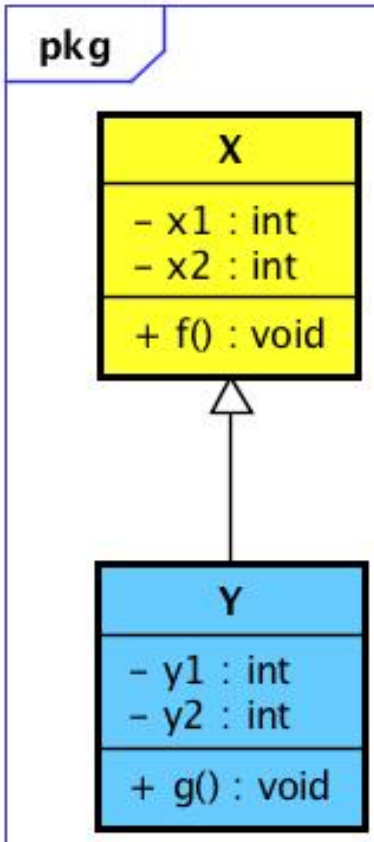
- Y é uma **extensão** de X
- Y **estende** X

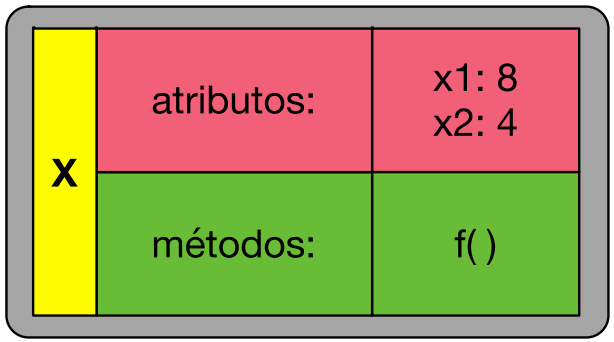
- X é uma **generalização** de Y
- Y é uma **especialização** de X

- X e Y possuem um relacionamento de **herança**
- Y **herda** os membros (atributos e métodos) de X

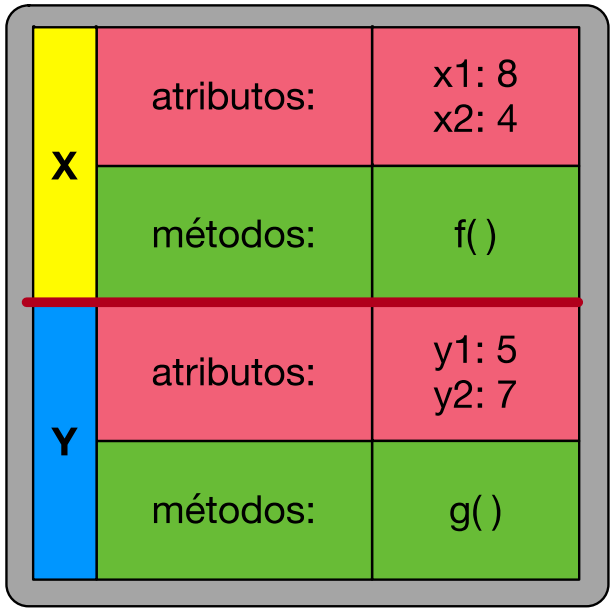
- Uma instância de Y é uma instância de X
- Um objeto da classe Y possui todos os membros definidos por Y e X

Estrutura de um objeto





Uma instância da classe X



Uma instância da classe Y

Encadeamento de construtores

```
class X {
    private int x1;
    private int x2;
    public X(int x1, int x2) {
        this.x1 = x1;
        this.x2 = x2;
    }
}
```

```
class Y extends X {
    private int y1;
    private int y2;
    public Y(int x1, int x2, int y1, int y2) {
        super(x1, x2); // chama construtor da superclasse
        this.y1 = y1;
        this.y2 = y2;
    }
}
```

Herança de método

```
class A {
    public void r( ) { System.out.println("@"); }
    public void s( ) { System.out.println("#");}
}
```

```
class B extends A {
    public void f( ) { r(); s(); }
}
```

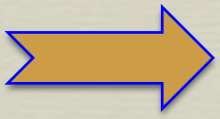
```
A a = new A();
B b = new B();
a.r();
a.s();
b.r();
b.s();
b.f();
```

Herança de método

```
class A {
    public void r( ) { System.out.println("@"); }
    public void s( ) { System.out.println("#");}
}
```

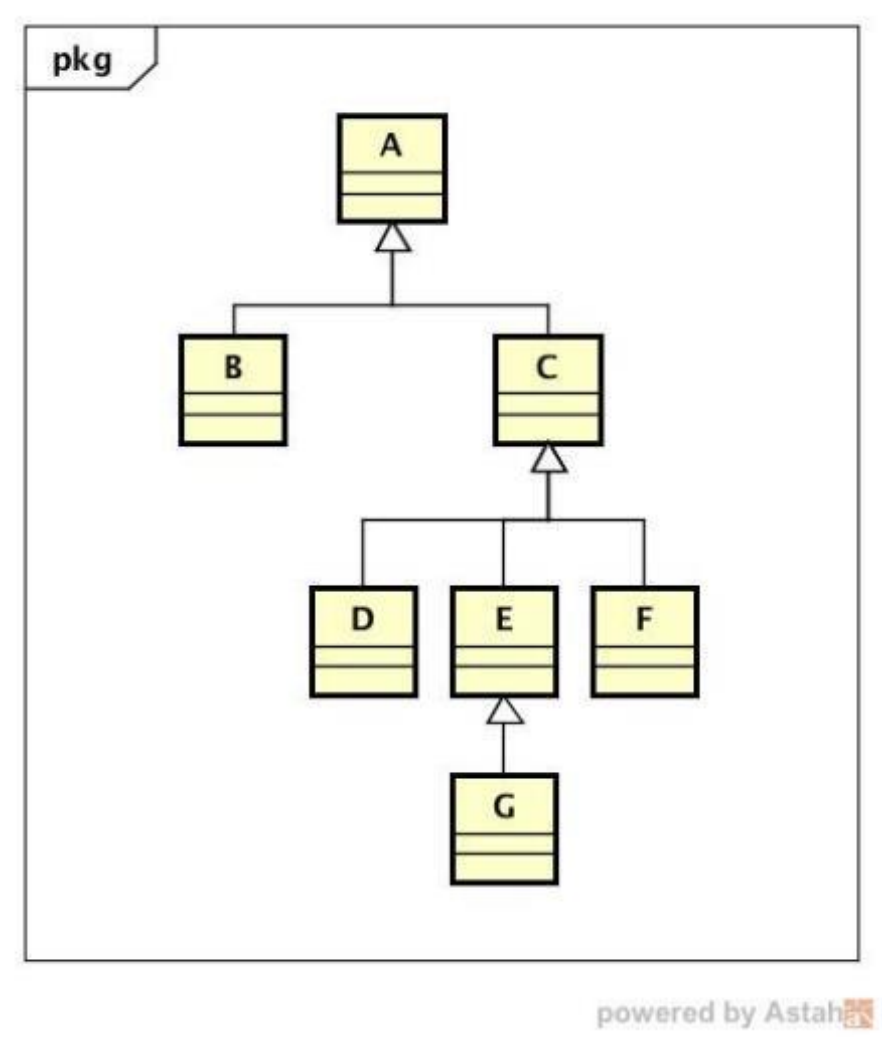
```
class B extends A {
    public void f( ) { r(); s(); }
}
```

```
A a = new A();
B b = new B();
a.r();
a.s();
b.r();
b.s();
b.f();
```



```
@
#
@
#
@
#
```

Hierarquia de classes



Pode haver qualquer quantidade de níveis.

G é subclasse de E, que é subclasse de C, que é subclasse de A.

A é superclasse de C, que é superclasse de E, que é superclasse de G.

A é superclasse direta de B e C e é superclasse indireta de D, E, F e G.

G é subclasse direta de E e é subclasse indireta de C e A.

A é a **raiz** da hierarquia.

Em Java, a raiz de toda hierarquia é a classe `Object`.

Visibilidade protected

- Um atributo ou um método de uma classe C qualificado como **protected** pode ser acessado por:
  1. qualquer subclasse (direta ou indireta) de C, independentemente de **pacote**
  2. qualquer classe que esteja no mesmo **pacote** de C, independentemente de relacionamento de hierarquia.

```
package P1;

public class A {
    protected int x;
    protected static int y;
    protected void f() { x = 10; }
    void g() { y = 20; }
}
```

```
package P1;

public class X {
    public void m() {
        A.y = 30;
    }
}
```

```
package P1;

public class B extends A {
    public void h() { f(); g();}
    public void t() { x = 4; y = 6; }
}
```

```
package P2;

import P1.A;

public class C extends A {
    public void h() { f(); x = 7; y = 8; }
}
```

Sobrescrita de método

```
class A {
    public void r( ) { System.out.println("@"); }
    public void s( ) { System.out.println("#");}
}
```

```
class B extends A {
    public void r( ) { System.out.println("%");}
    public void s( ) { super.s(); System.out.println("$");}
}
```

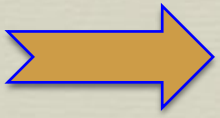
```
A a = new A();
B b = new B();
a.r();
a.s();
b.r();
b.s();
```

Sobrescrita de método

```
class A {
    public void r( ) { System.out.println("@"); }
    public void s( ) { System.out.println("#");}
}
```

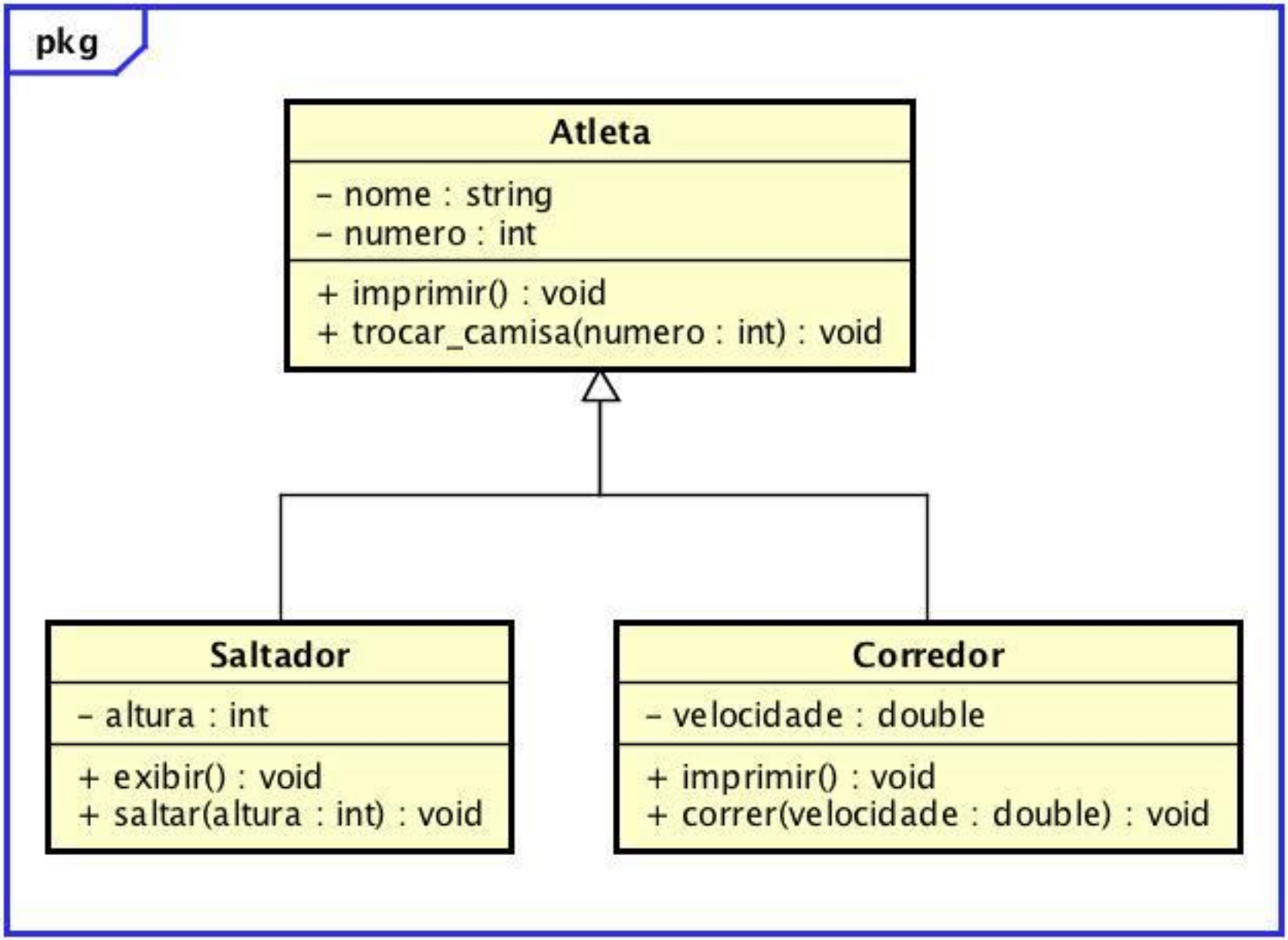
```
class B extends A {
    public void r( ) { System.out.println("%");}
    public void s( ) { super.s(); System.out.println("$");}
}
```

```
A a = new A();
B b = new B();
a.r();
a.s();
b.r();
b.s();
```



@
#
%
#
$

Exemplo Completo



```
class Atleta {
    private String nome;
    private int numero;
    public Atleta(String nome, int numero) {
        this.nome = nome;
        this.numero = numero;
    }
    public void imprimir() {
        System.out.println(nome);
        System.out.println(numero);
    }
    public void trocar_camisa(int numero) {
        this.numero = numero;
    }
}
```

```
class Saltador extends Atleta {
    private int altura;
    public Saltador(String nome, int numero, int altura) {
        super(nome, numero); // chama construtor da superclasse
        this.altura = altura;
    }
    public void exibir() {
        imprimir(); // chama método da superclasse
        System.out.println(altura);
    }
    public void saltar(int altura) {
        this.altura = altura;
    }
}
```

```
class Corredor extends Atleta {
    private double velocidade;
    public Corredor(String nome, int numero, double velocidade) {
        super(nome, numero); // chama construtor da superclasse
        this.velocidade = velocidade;
    }
    public void imprimir() {
        super.imprimir(); // chama método da superclasse
        System.out.println(velocidade);
    }
    public void correr(double velocidade) {
        this.velocidade = velocidade;
    }
}
```

```
class Competicao {
    public static void main(String[] args) {
        Atleta falcao = new Atleta("Paulo Roberto Falcao", 5);
        Saltador sotomayor =
                    new Saltador( "Javier Sotomayor", 76, 245);
        Corredor bolt = new Corredor("Usain Bolt", 709, 37.58);

        falcao.imprimir();
        sotomayor.imprimir();
        sotomayor.exibir();
        bolt.imprimir();

        falcao.trocar_camisa(10);
        sotomayor.trocar_camisa(31);
        bolt.trocar_camisa(2163);
        sotomayor.saltar(233);
        bolt.correr(36.92);

        falcao.imprimir();
        sotomayor.exibir();
        bolt.imprimir();
    }
}
```

Paulo Roberto Falcao
5
Javier Sotomayor
76
Javier Sotomayor
76
245
Usain Bolt
709
37.58
Paulo Roberto Falcao
10
Javier Sotomayor
31
233
Usain Bolt
2163
36.92