

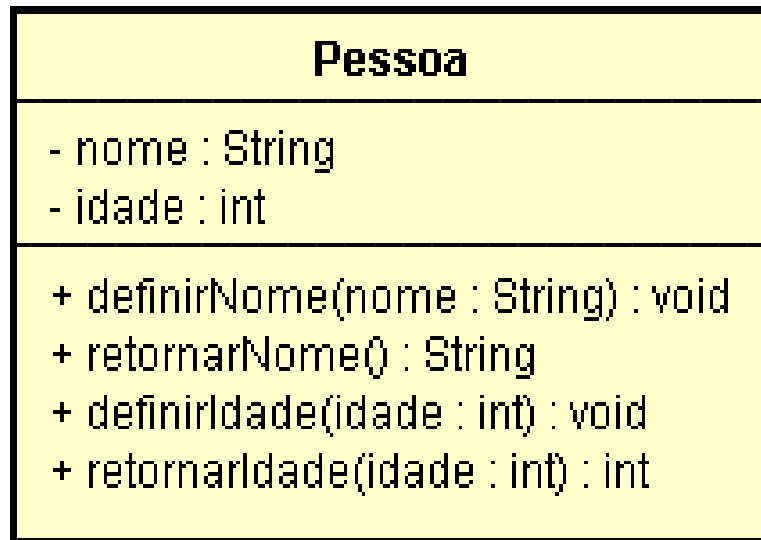
# Aula 3

# A LINGUAGEM JAVA

- ▶ Controlando o acesso aos membros da classe

Especificador	classe	subclasse	pacote	todos
<hr/>				
<b>private</b>	<b>X</b>			
<b>package</b>	<b>X</b>		<b>X</b>	
<b>protected</b>	<b>X</b>	<b>X</b>	<b>X</b>	
<b>public</b>	<b>X</b>	<b>X</b>	<b>X</b>	<b>X</b>

# Orientação a Objetos – Encapsulamento dos Dados



Proteger os atributos

Permitir acesso aos atributos através dos métodos

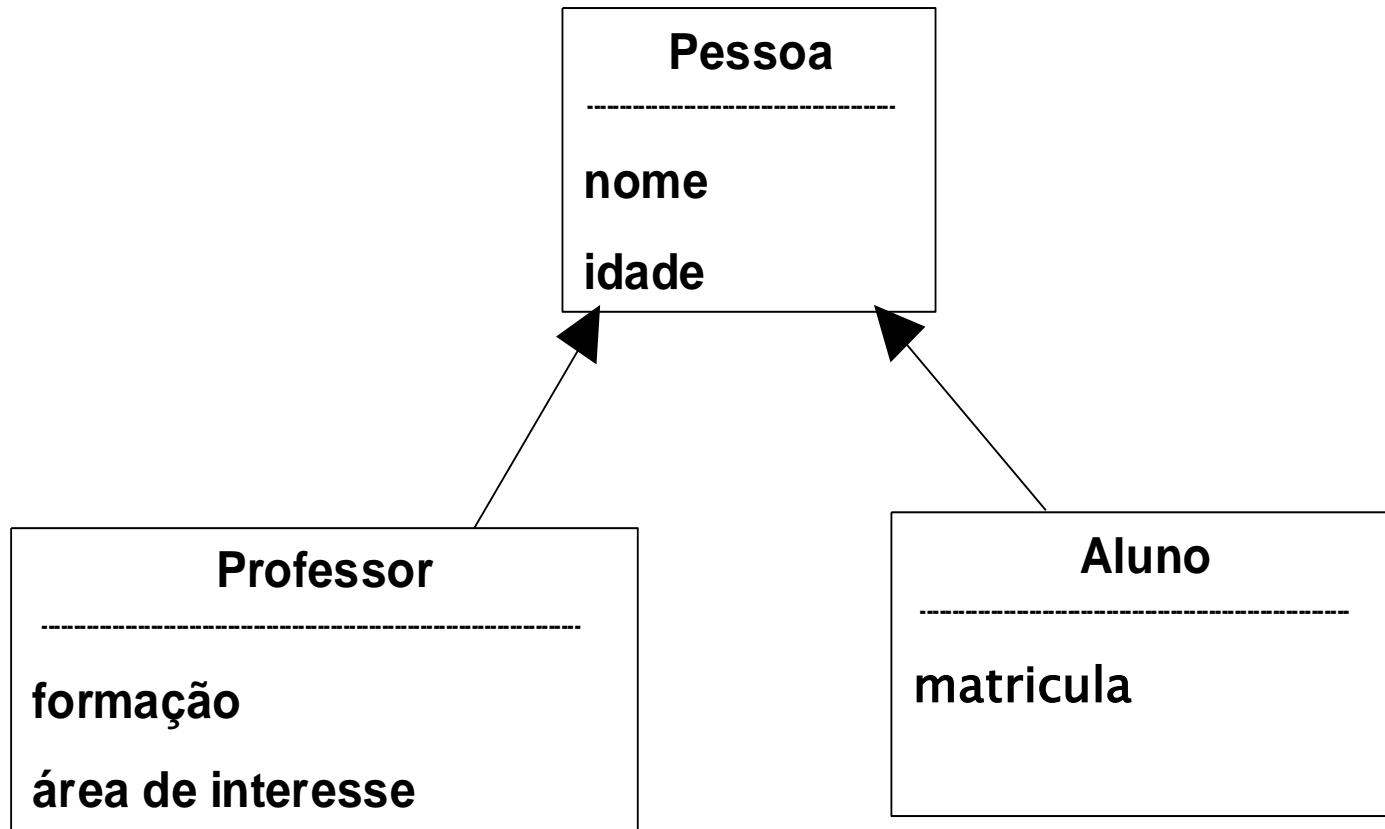
# Modificadores de Acesso

- Determinam se atributos e métodos poderão ser acessados por outras classes
  - public (público)
  - private (privado)
  - protected (protegido)
  - modificador não explícito (package)

# Orientação a Objetos – Herança

- ◉ Permite a uma classe herdar o estado (atributos) e o comportamento (métodos) de outra classe
- Entre diferentes classes podem existir diversas semelhanças
  - Duas ou mais classes poderão compartilhar os mesmos atributos e/ou os mesmos métodos
    - Superclasse
    - Subclasse
    - Ancestral
    - Descendente

# Orientação a Objetos – Herança

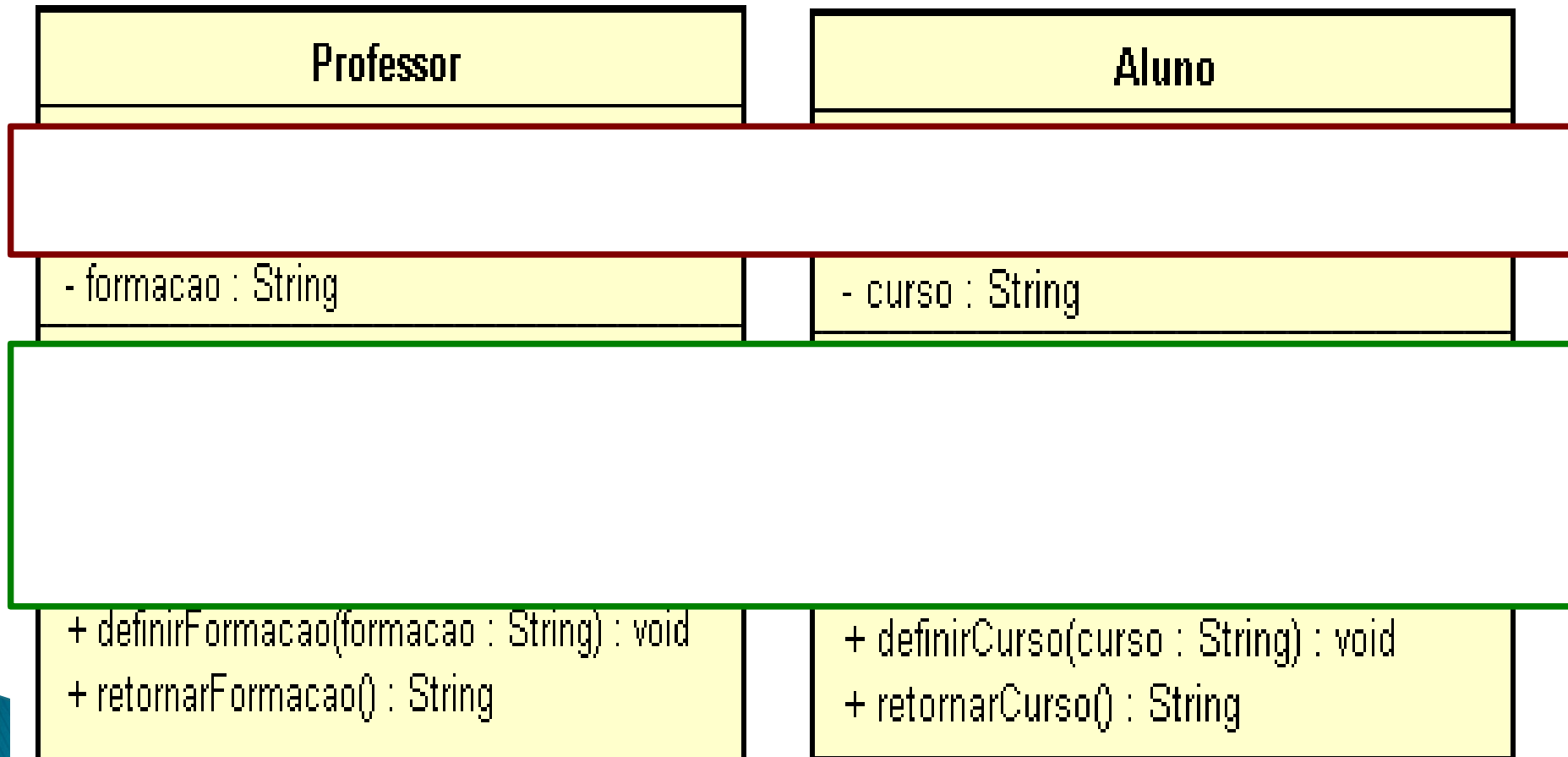


# Orientação a Objetos – Herança

Professor
<ul style="list-style-type: none"><li>- nome : String</li><li>- idade : int</li><li>- formacao : String</li></ul>
<ul style="list-style-type: none"><li>+ definirNome(nome : String) : void</li><li>+ retornarNome() : String</li><li>+ definirIdade(idade : int) : void</li><li>+ retornarIdade() : int</li><li>+ definirFormacao(formacao : String) : void</li><li>+ retornarFormacao() : String</li></ul>

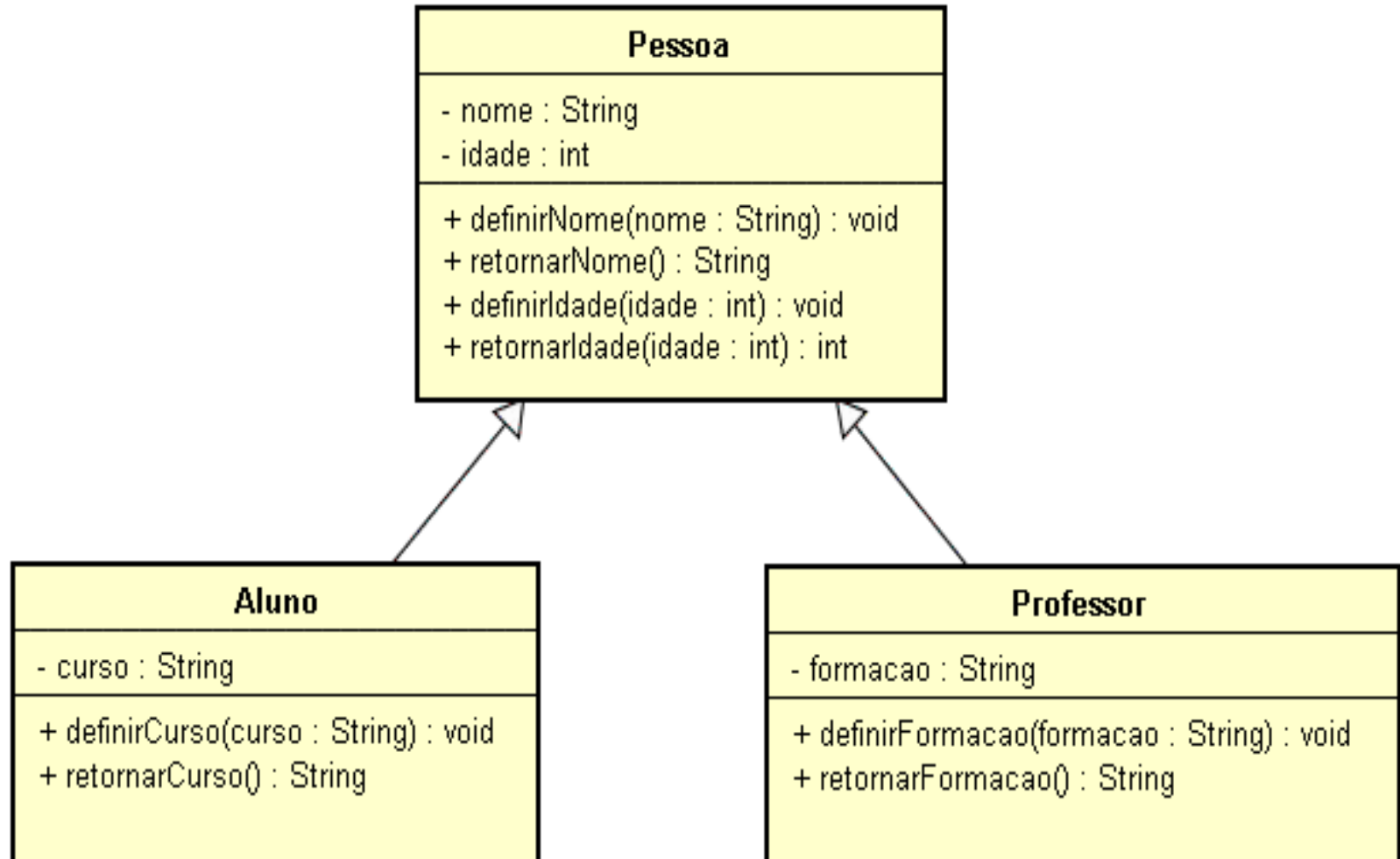
Aluno
<ul style="list-style-type: none"><li>- nome : String</li><li>- idade : int</li><li>- curso : String</li></ul>
<ul style="list-style-type: none"><li>+ definirNome(nome : String) : void</li><li>+ retornarNome() : String</li><li>+ definirIdade(idade : int) : void</li><li>+ retornarIdade() : int</li><li>+ definirCurso(curso : String) : void</li><li>+ retornarCurso() : String</li></ul>

# Orientação a Objetos – Herança





# Orientação a Objetos – Herança



# Orientação a Objetos – Herança

```
// SuperClass.java
```

```
public class SuperClass  
{  
    // Atributos e métodos  
}
```

```
// SubClass1.java
```

```
public class SubClass1 extends SuperClass  
{  
    // Atributos e métodos  
}
```

```
// SubClass2.java
```

```
class SubClass2 extends SuperClass {  
    ...  
}
```

# Orientação a Objetos – Herança

```
class Pessoa {  
    String    nome;  
    int       idade;  
  
    void definirNome(String valor) {  
        nome = valor;  
    }  
  
    String retornarNome() {  
        return nome;  
    }  
  
    void definirIdade(int valor) {  
        idade = valor;  
    }  
  
    int retornarIdade() {  
        return idade;  
    }  
}
```

```
class Aluno extends Pessoa {  
    String    curso;  
  
    void definirCurso(String valor) {  
        curso = valor;  
    }  
  
    String retornarCurso() {  
        return curso;  
    }  
}
```

# Orientação a Objetos – Herança

Em outra classe, no método main faça o que está abaixo

```
Aluno aluno1 = new Aluno();  
aluno1.definirNome("João");  
aluno1.definirIdade(25);  
aluno1.definirCurso("Sistemas de  
Informação");  
System.out.println(  
    aluno1.retornarNome());
```

João  
25  
Sistemas de Informação

```
Aluno aluno2 = new Aluno();  
aluno2.definirNome("Maria");  
aluno2.definirIdade(20);  
aluno2.definirCurso("Sistemas de  
Informação");  
System.out.println(  
    aluno2.retornarNome());
```

Maria  
20  
Sistemas de Informação

# Orientação a Objetos – Herança

- Classes Abstratas X Classes Concretas
  - Uma *classe abstrata* é uma classe que *não tem instâncias*, mas cujas classes descendentes podem ter instâncias.
  - Uma *classe concreta* é uma *classe que pode ser instanciada*.
- Classes Abstratas X Interfaces
  - *A classe abstrata pode possuir métodos não abstratos.*
  - Uma *interface apenas propõe os métodos que devem ser implementados* pelas classes que desejarem implementar a interface.

# Orientação a Objetos – Herança

```
public abstract class Empregado {  
    String nome;  
    double salario;  
    public Empregado (String nome, double salario) {  
        this.nome = nome;  
        this.salario = salario;  
    }  
}
```

```
public class Vendedor extends Empregado{  
    public void aumentaSalario (double percentualDeAumento){  
        this.salario = this.salario * (1+(percentualDeAumento/100));  
    }  
}
```

```
public class Gerente extends Empregado {  
    public void aumentaSalario(double percentual) {  
        this.salario = this.salario * (1+(2 * percentual/100));  
    }  
}
```

# Orientação a Objetos – Herança

```
public interface Assalariado {  
    public float salario = 0;  
    public float getSalario();  
}  
  
public class Empregado implements Assalariado {  
    private String nome;  
  
    public float getSalario() {  
        return this.salario;  
    }  
  
    public String getNome() {  
        return this.nome;  
    }  
}
```

# Orientação a Objetos – Herança

- ▶ Atributos declarados em interfaces são públicos e estáticos
- ▶ É possível sobrescrever o atributo para que ele não seja estático, ex:

```
public class Empregado implements Assalariado {  
    private String nome;  
    public float salario; ←
```



# Contrutores

```
class Pessoa {  
    private String nome;  
    int idade;  
  
    public Pessoa (String nome, int  
idade) {  
        this.nome = nome;  
        this.idade = idade;  
    }  
  
    public Pessoa () {  
    }  
  
    public void setNome(String nome){  
        this.nome = nome;  
    }  
  
    public String getNome(){  
        return this.nome;  
    }  
}
```

```
class CriarPessoa{  
  
    public static void main(String[] args){  
        Pessoa p = new Pessoa("Maria",  
20);  
        System.out.println(p.getNome());  
        System.out.println(p.idade);  
  
        p = new Pessoa();  
        p.setNome("João Silva");  
        p.idade = 10;  
        System.out.println(p.getNome());  
        System.out.println(p.idade);  
    }  
}
```

# Contrutores

- Devem ter o *mesmo nome da classe* que o inicializam
- Podem ter parâmetros
- Não tem retorno
- Se não é declarado nenhum construtor
  - A linguagem provê um construtor padrão sem argumentos

# Contrutores

Pacotes ou  
classes  
importados

```
import java.util.*;
```

Classe

```
public class AloMundo {
```

```
    private String mensagem = " ";
```

```
    public AloMundo () {  
        Date data = new Date();  
        mensagem = "Alô, Mundo" + data.toString();  
    }
```

```
    public void mostrarMensagem () {  
        System.out.println( mensagem );  
    }
```

```
}
```

Variáveis

Construtores

Métodos

# Pacotes

## ► Diretorio

- As Classes podem ser agrupadas em *packages*
- Um pacote é o mesmo que um diretório
- Você deve declarar o diretório quando precisa utilizar elementos da classes em pacotes diferentes

# Pacotes

- ▶ Importar a classe JOptionPane do swing

```
import javax.swing.JOptionPane;
class ImportTest {
    public static void main(String[] args) {
        JOptionPane.showMessageDialog(null, "Hi");
    }
}
```

# Pacotes

- ▶ Importar todas as classes do pacote swing:

```
import javax.swing.*;
```

```
class ImportTest {  
    public static void main(String[] args) {  
        String x = JOptionPane.showInputDialog("Qual o seu nome?");  
        JOptionPane.showMessageDialog(null, "Oi " + x);  
    }  
}
```

# Pacotes

**Exemplo: Crie o pacote calculo e a classe abaixo dentro dele.**

```
package calculo;
```

```
public class Calcular{
```

```
    public int somar(int x, int y){  
        return x+y;
```

```
    }
```

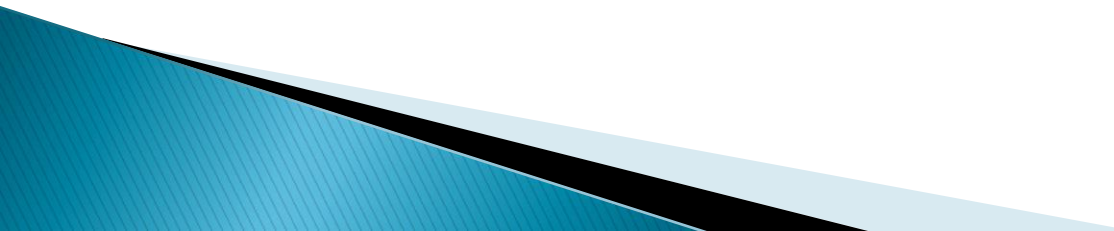
```
}
```



# Pacotes

**Exemplo (continuação) – Crie outra classe fora do pacote como a seguir:**

```
class Teste2{  
  
    public static void main(String[] args){  
        Calcular c1 = new Calcular();  
        System.out.println(c1.somar(5, 7));  
    }  
}
```





# Pacotes

**Exemplo (continuação) – Insira o Import como no exemplo abaixo:**

```
import calculo.Calcular;
```

```
class Teste2{
```

```
    public static void main(String[] args){
```

```
        Calculo c1 = new Calculo();
```

```
        System.out.println(c1.somar(5, 7));
```

```
    }
```

```
}
```

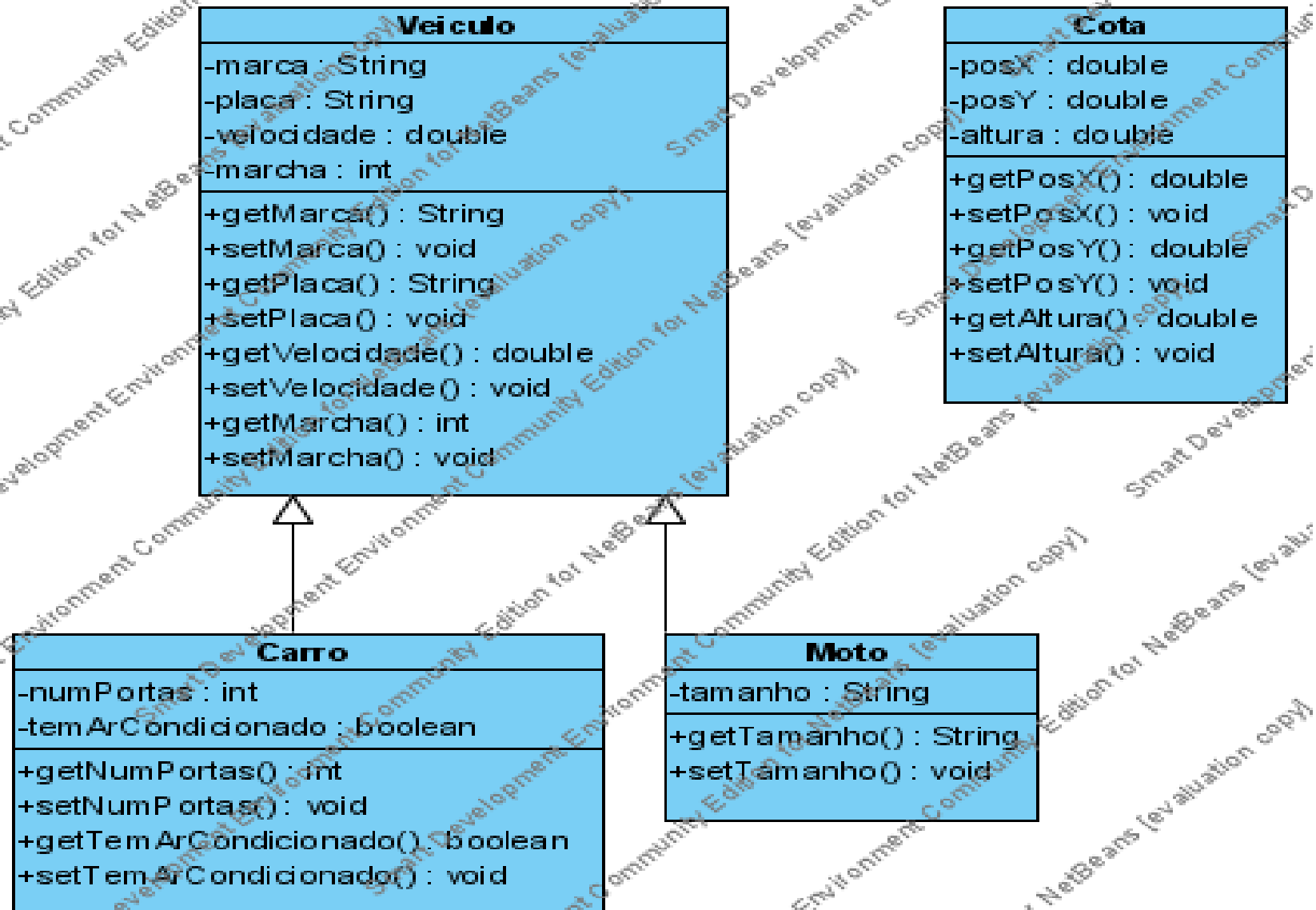


# Exercício

- ▶ Implemente os diagramas de classes a seguir
- ▶ Divida as classes em pacotes diferentes

# Exercícios

- Exercício1: Escreva um programa que implemente as classes abaixo conforme o diagrama de classes em UML



# Exercícios

- Exercício2: Implemente o exemplo abaixo

