

07.Pacotes e Interfaces (Polimorfismo)

Prof. Alexandre Krohn



Roteiro











- Pacotes
 - Definição e Utilização
 - Encapsulamento
- Interfaces
- Polimorfismo
- Exercícios



Pacotes

- Pacotes são agrupadores lógicos de classes que criamos, ou fazem parte de bibliotecas que utilizamos
- Sua função é organizar o código-fonte e simplificar a compreensão do sistema.

Pacotes : Visualização no Eclipse IDE

- ▼  ExemploFormasGeometricas
 - ▶  JRE System Library [JavaSE-13]
 - ▼  src
 - ▼  (default package)
 - ▶  Circulo.java
 - ▶  FormaGeometrica.java
 - ▶  Quadrado.java
 - ▶  Retangulo.java
 - ▶  TestaFormas.java
 - ▶  Triangulo.java

Projeto que **não** usa pacotes

Pacotes : Visualização no Eclipse IDE

- ▼ ExemploFormasGeometricas
 - ▶ JRE System Library [JavaSE-13]
 - ▼ src
 - ▼ (default package)
 - ▶ Circulo.java
 - ▶ FormaGeometrica.java
 - ▶ Quadrado.java
 - ▶ Retangulo.java
 - ▶ TestaFormas.java
 - ▶ Triangulo.java

Projeto que **não** usa pacotes

Note que todas as classes ficam juntas, dentro do **default package**

Pacotes : Visualização no Eclipse IDE

- ▼ ExemploFormasGeometricas
 - ▶ JRE System Library [JavaSE-13]
 - ▼ src
 - ▼ (default package)
 - ▶ Circulo.java
 - ▶ FormaGeometrica.java
 - ▶ Quadrado.java
 - ▶ Retangulo.java
 - ▶ TestaFormas.java
 - ▶ Triangulo.java

Projeto que **não** usa pacotes

Note que todas as classes ficam juntas, dentro do **default package**












default package = sem pacotes = desorganizado!



Pacotes : Visualização no Eclipse IDE

Projeto **com** pacotes

Pacotes organizam as classes por **similaridade funcional**

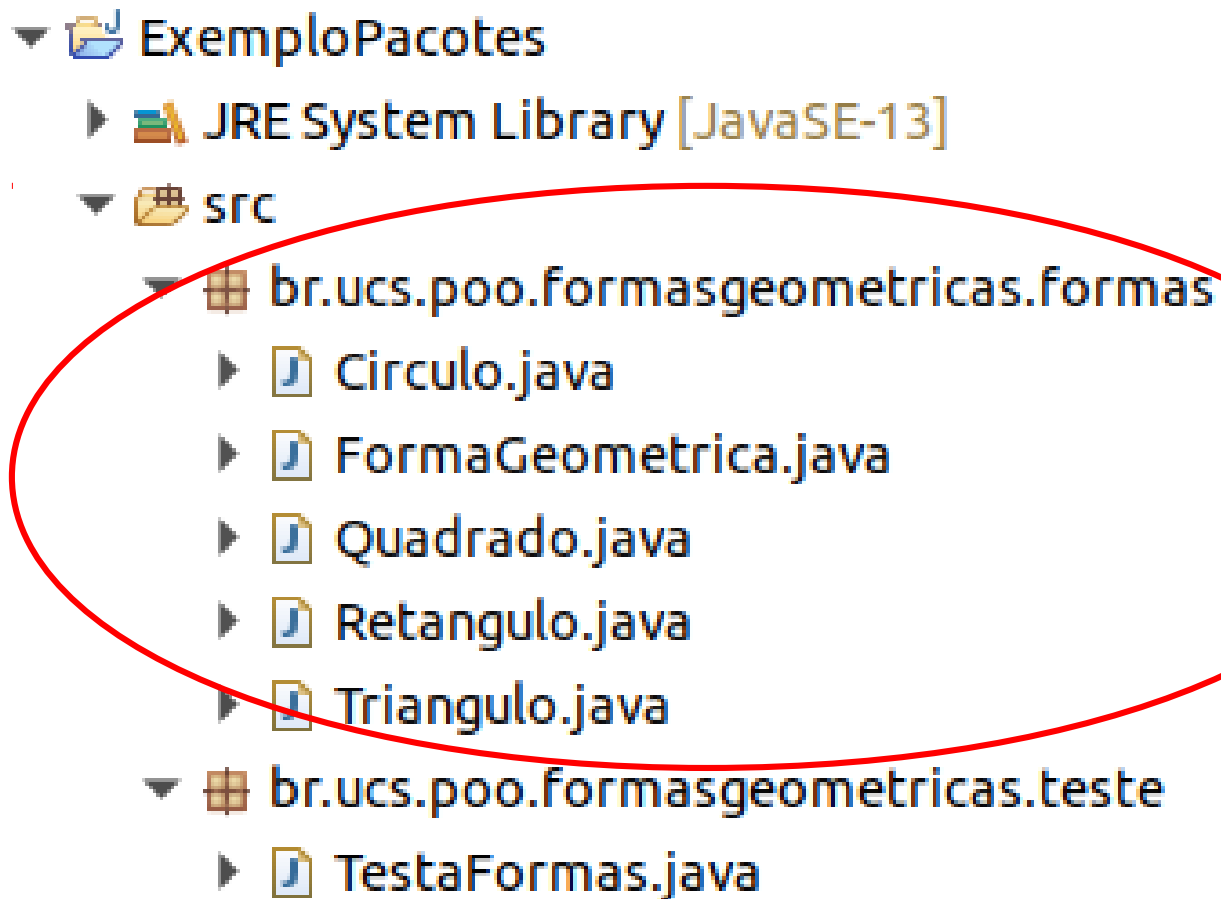
- ▼  ExemploPacotes
 - ▶  JRE System Library [JavaSE-13]
 - ▼  src
 - ▼  br.ucs.poo.formasgeometricas.formas
 - ▶  Circulo.java
 - ▶  FormaGeometrica.java
 - ▶  Quadrado.java
 - ▶  Retangulo.java
 - ▶  Triangulo.java
 - ▼  br.ucs.poo.formasgeometricas.teste
 - ▶  TestaFormas.java



Pacotes : Visualização no Eclipse IDE

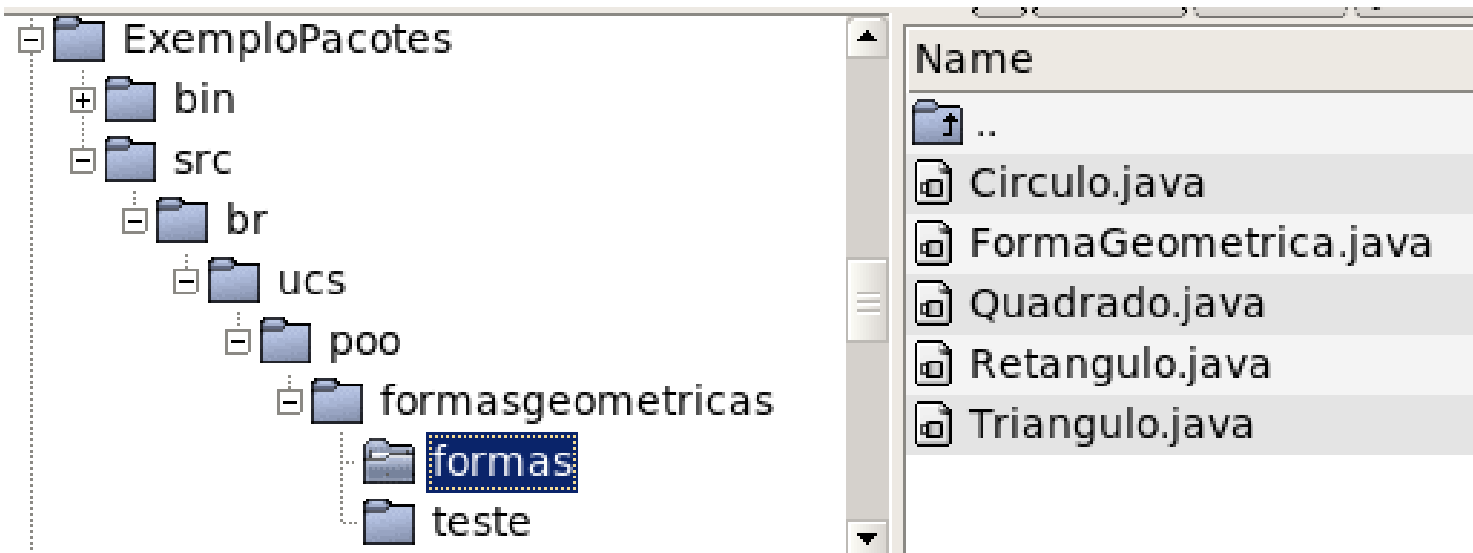
Projeto **com** pacotes

Pacotes organizam as classes por **similaridade funcional**

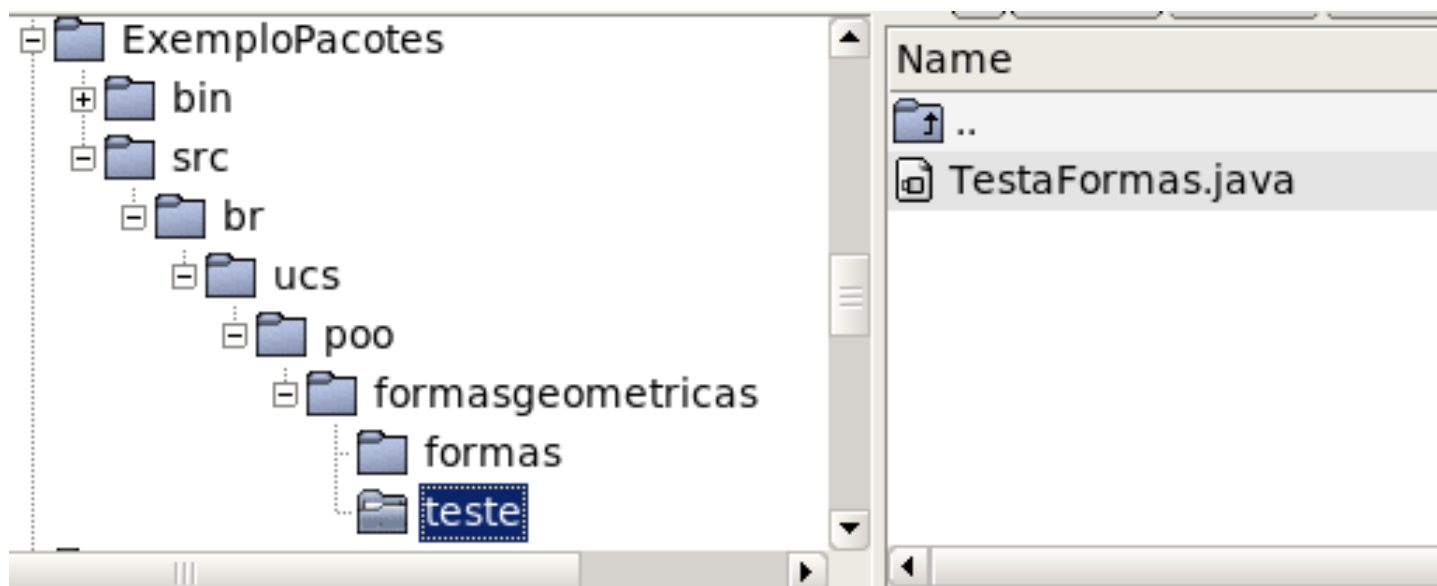


Neste pacote só haverão classes que representam formas geométricas

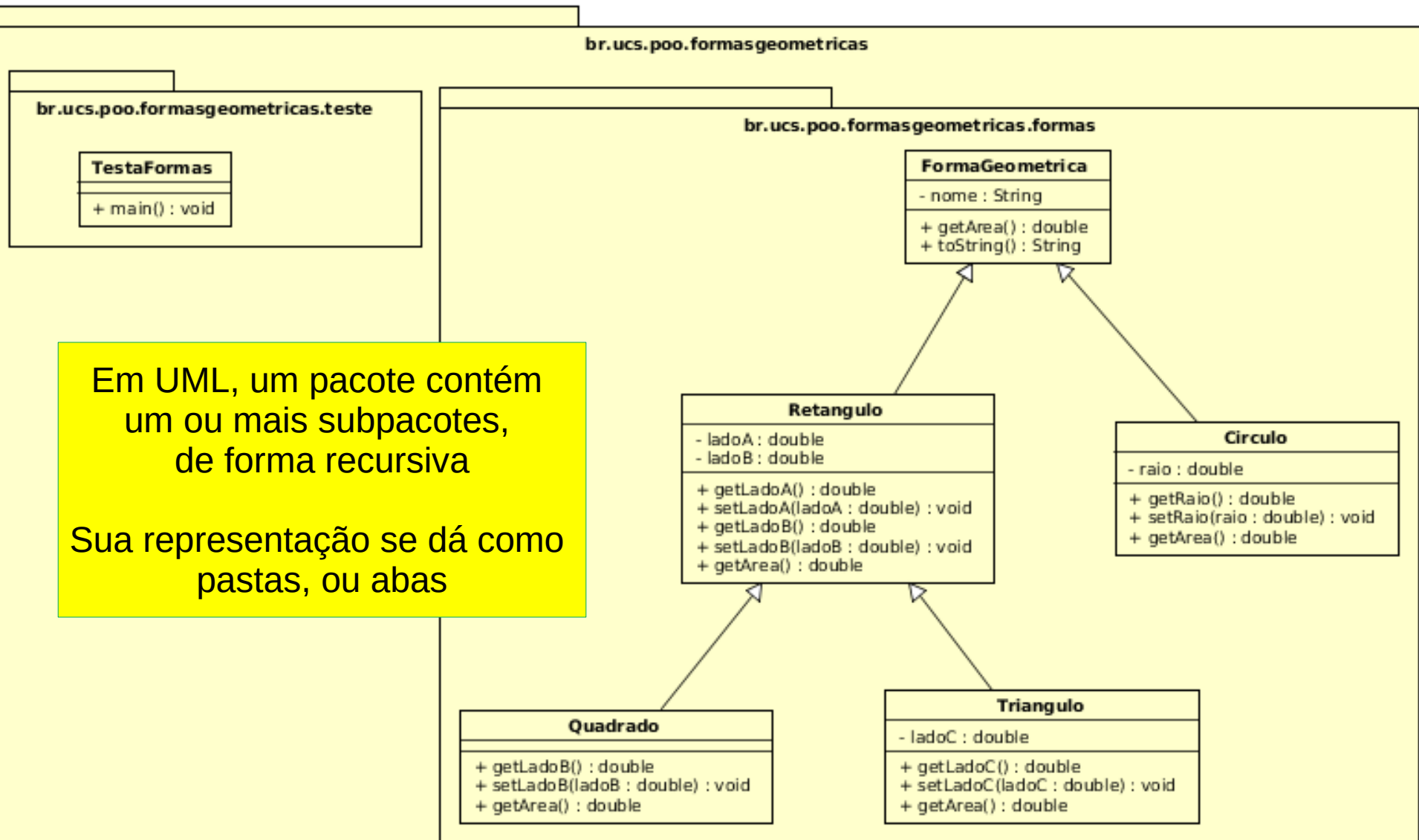
Pacotes : Diretórios



No sistema operacional, os pacotes são armazenados em **árvores de diretórios**, cada uma com suas respectivas classes.



Pacotes : Representação UML





Pacotes : Utilização

- No Eclipse, primeiro criamos os pacotes
- Seus nomes são todos em letras minúsculas, separados por pontos
- Cada ponto virará uma subpasta

Padrão de Nomenclatura dos Pacotes

- O padrão da Sun (agora Oracle) para dar nome aos pacotes é relativo ao nome da empresa que desenvolveu a classe:
 - `br.com.nomedaempresa.nomedoprojeto.subpacote`
 - `br.com.nomedaempresa.nomedoprojeto.subpacote2`
 - `br.com.nomedaempresa.nomedoprojeto.subpacote2.subpacote3`
- Os pacotes só possuem letras minúsculas, não importa quantas palavras estejam contidas nele. Esse padrão existe para evitar ao máximo o conflito de pacotes de empresas diferentes.
- As classes do pacote padrão de bibliotecas não seguem essa nomenclatura, que foi dada para bibliotecas de terceiros.

Pacotes : Utilização

- Depois, criamos as classes. Sua primeira linha declara à qual pacote a classe pertence:

```
package br.ucs.poo.formasgeometricas.formas;  
  
public class Quadrado extends Retangulo {  
  
    public Quadrado() {  
        super("Quadrado");  
    }  
  
    public Quadrado(String nome) {  
        super(nome);  
    }  
}
```

Pacotes : Utilização

- Depois, criamos as classes. Sua primeira linha declara à qual pacote a classe pertence:

```
package br.ucs.poo.formasgeometricas.formas;
```

```
public class Quadrado extends Retangulo {
```

```
    public Quadrado() {  
        super("Quadrado");  
    }
```

```
    public Quadrado(String nome) {  
        super(nome);  
    }
```

Declaração do pacote



Pacotes : Utilização

- As classes que pertencem ao mesmo pacote “enxergam-se” entre si, e não necessitam importar umas às outras para compilar.



Pacotes : Utilização

- Já as classes que usam classes de um pacote diferente do seu precisam importar as classes que referenciam

Pacotes : Utilização

```
package br.ucs.poo.formasgeometricas.teste;
```

```
import br.ucs.poo.formasgeometricas.formas.Circulo;  
import br.ucs.poo.formasgeometricas.formas.FormaGeometrica;  
import br.ucs.poo.formasgeometricas.formas.Quadrado;  
import br.ucs.poo.formasgeometricas.formas.Retangulo;  
import br.ucs.poo.formasgeometricas.formas.Triangulo;
```

```
public class TestaFormas {
```

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

```
        FormaGeometrica f = new FormaGeometrica();
```

```
        System.out.println(f);
```

Importações

Pacotes : Utilização

```
package br.ucs.poo.formasgeometricas.teste;
```

```
import br.ucs.poo.formasgeometricas.formas.Circulo;  
import br.ucs.poo.formasgeometricas.formas.FormaGeometrica;  
import br.ucs.poo.formasgeometricas.formas.Quadrado;  
import br.ucs.poo.formasgeometricas.formas.Retangulo;  
import br.ucs.poo.formasgeometricas.formas.Triangulo;
```

```
public class TestaFormas {
```

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

```
        FormaGeometrica f = new FormaGeometrica();
```

```
        System.out.println(f);
```

Importações

O Eclipse realiza as importações automaticamente.

Digite:

CTRL + SHIFT + O



Roteiro

- Pacotes
 - Definição e Utilização
 - Encapsulamento
- Interfaces
- Polimorfismo
- Exercícios



Pacotes : Encapsulamento

- Quando você não define um modificador (**public**, **private** ou **protected**) para uma classe, método ou atributo, o mesmo é visível para todas as classes dentro do mesmo pacote.
- **Use isso com cuidado!**



Roteiro

- Pacotes
 - Definição e Utilização
 - Encapsulamento
- Interfaces
- Polimorfismo
- Exercícios



Interfaces

- O conceito de interface é usado em praticamente toda a computação, até mesmo na engenharia do ramo programação em baixo nível (interface de comunicação entre hardware e o kernel do Sistema Operacional).



Interfaces

- Podemos definir **interface** como uma face, um plano, uma divisa que faz a comunicação entre dois meios diferentes.
- Por exemplo, quando trabalhamos com eletrônica, basicamente é tudo estado alto ou estado baixo, que são nada mais que níveis de voltagem ou corrente elétrica. Lá se usa registros, CI (circuitos integrados), portas lógicas, equipamentos de elétrica e eletrônica.



Interfaces

- Em programação O.O., interface é o meio que uma classe usa para expôr seus métodos para outras
- Diz-se que a interface de uma classe é **tudo aquilo que podemos “usar”, ou “chamar” dela**



Interfaces Java

- Java possui um tipo especial de artefato chamado **interface** que serve para isso: Expôr as funcionalidades de uma classe
- Na prática, uma classe que implementa uma interface é obrigada a implementar todos os métodos definidos na interface.
- Chamamos essa obrigação de **contrato**.



Exemplo : Interface Shape

- Vamos contruir um software para realizar algumas operações clássicas da geometria.
- Cálculos de **Áreas e Volumes**

Exemplo : Interface Shape

```
package br.ucs.poo.exemplointerfaces.shapes;  
  
public interface Shape {  
  
    double area();  
    double volume();  
    String getName();  
  
}
```

Exemplo : Interface Shape

```
package br.ucs.poo.exemplointerfaces.shapes;
```


```
public interface Shape {
```

```
    double area();
```

```
    double volume();
```

```
    String getName();
```

```
}
```



Note que nossa interface
também pertence
a um pacote

Exemplo : Interface Shape

```
package br.ucs.poo.exemplointerfaces.shapes;
```

```
public interface Shape {
```

```
    double area();  
    double volume();  
    String getName();
```

```
}
```

Os métodos que fazem parte de nosso **contrato** estão declarados na interface.

Todas as classes que implementarem a interface shape precisarão implementar todos os métodos declarados nela.

Exemplo : Interface Shape

```
package br.ucs.poo.exemplointerfaces.shapes;
```

```
public interface Shape {
```

```
    double area();
```

```
    double volume();
```

```
    String getName();
```

```
}
```

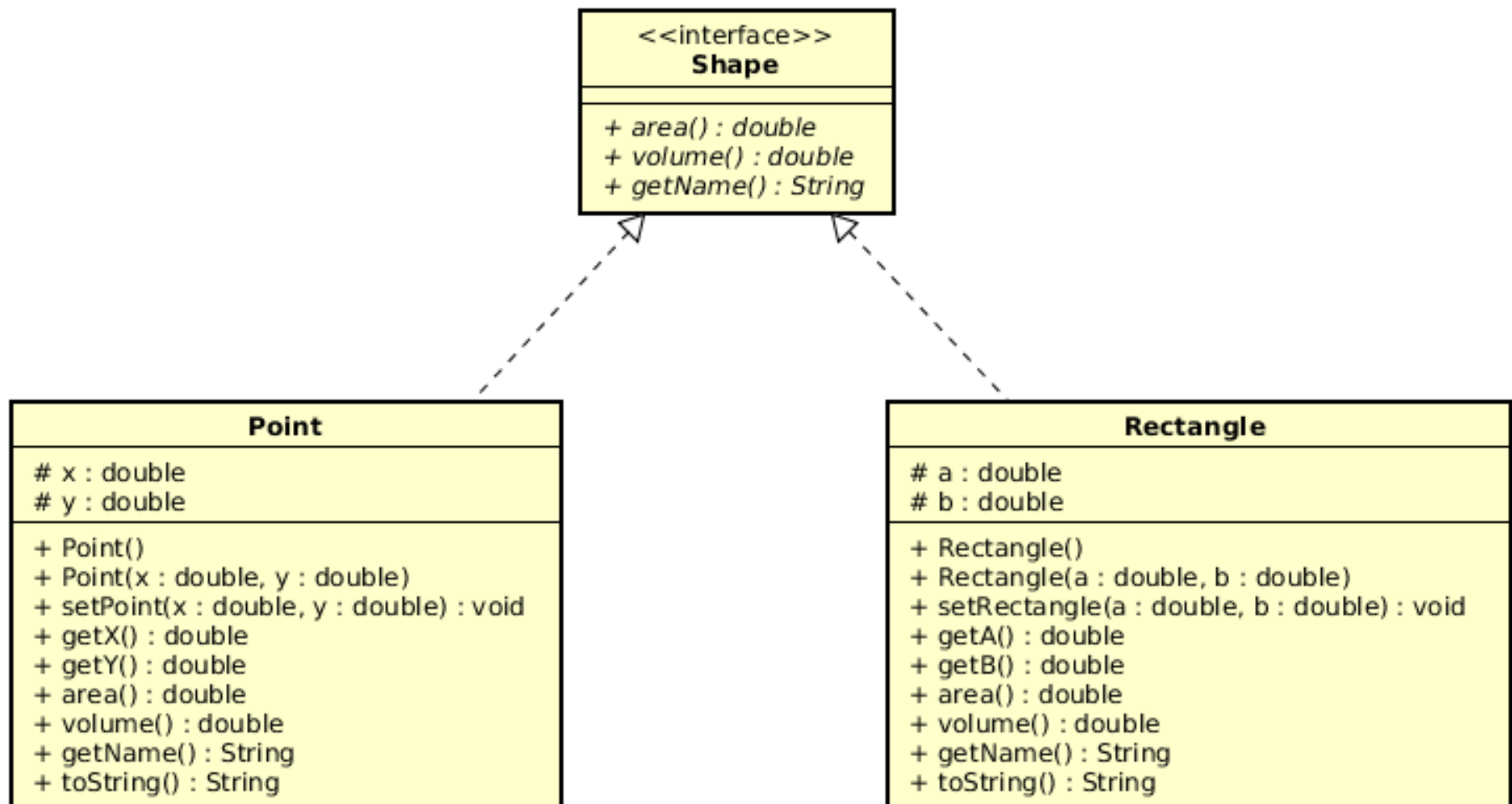
**Os métodos das interfaces
são sempre públicos,**
portanto não é necessário
utilizar modificadores neles



Exemplo : Interface Shape

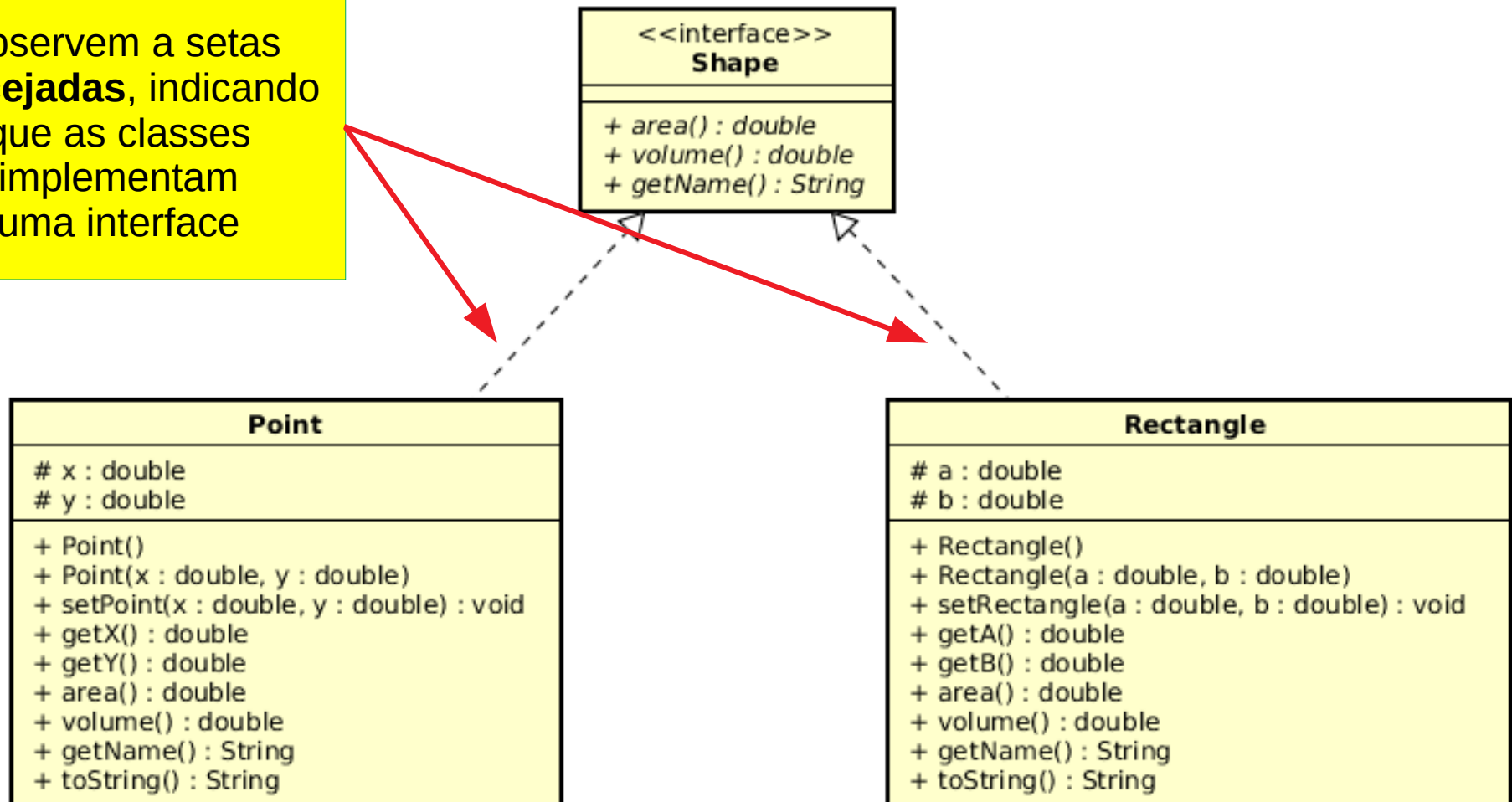
- A partir da interface Shape vamos definir duas formas geométricas :
Point e Rectangle

Exemplo : Interface Shape

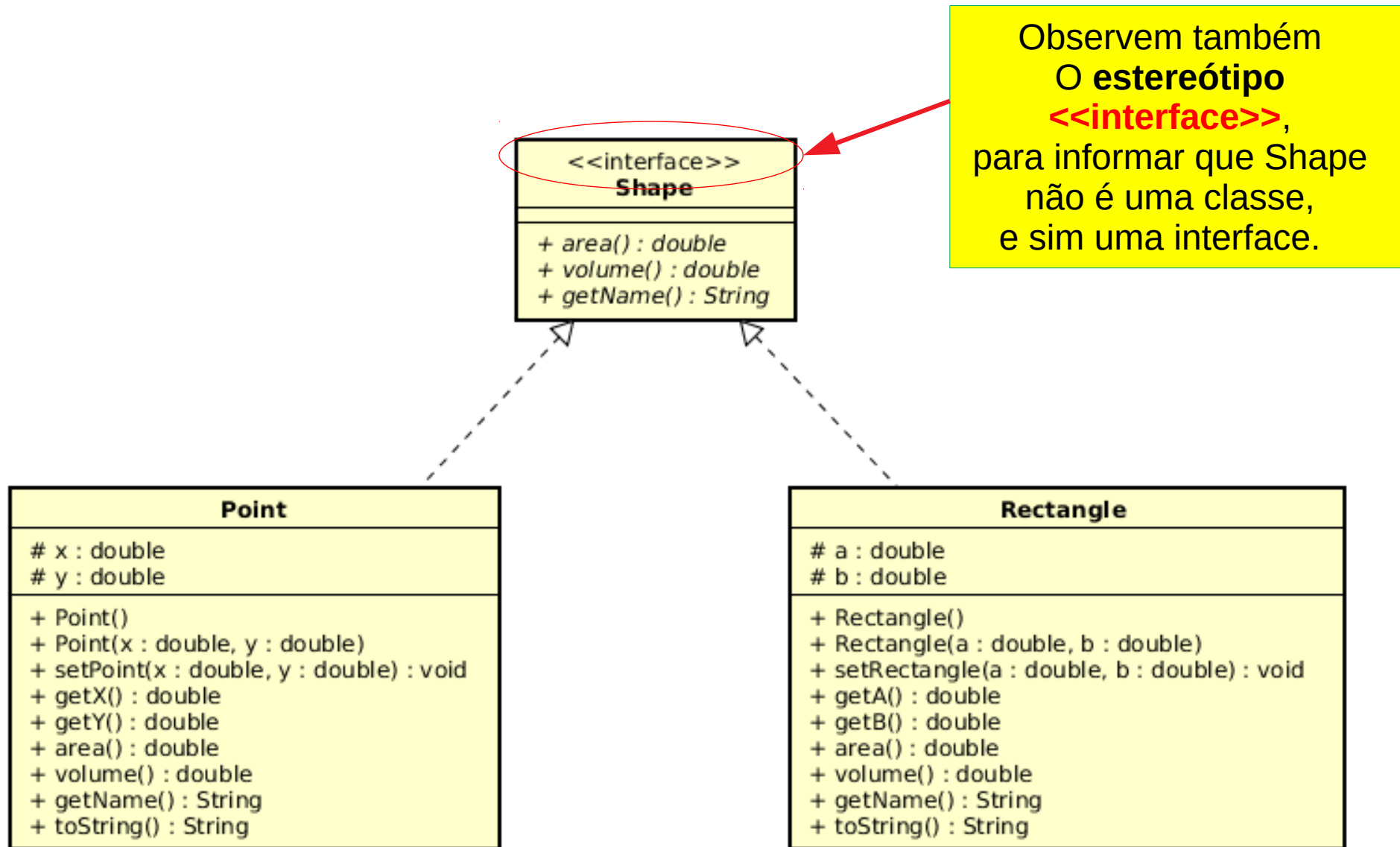


Exemplo : Interface Shape

Observem as setas tracejadas, indicando que as classes implementam uma interface



Exemplo : Interface Shape



Interface Java : Implementação

- Usa-se a palavra-chave `implements` para informar que uma classe implementa uma interface.

```
public class Point implements Shape {
```

```
public class Rectangle implements Shape {
```

Classe **Point**

```
package br.ucs.poo.exemplointerfaces.shapes;
```

```
public class Point implements Shape {
```

```
    protected double x, y;
```

```
    public Point() {  
        setPoint(0,0);  
    }
```

```
    public Point(double x, double y) {  
        setPoint(x,y);  
    }
```

```
    public void setPoint(double x, double y) {  
        this.x = x;  
        this.y = y;  
    }
```

```
    public double getX() {  
        return x;  
    }
```

```
    public double getY() {  
        return y;  
    }
```

```
    @Override  
    public double area() {  
        return 0;  
    }
```

```
    @Override  
    public double volume() {  
        return 0;  
    }
```

```
    @Override  
    public String getName() {  
        return "Point";  
    }
```

```
    @Override  
    public String toString() {  
        return "[" + x + "," + y + "];"  
    }
```

```
}
```

Classe **Point**

```
package br.ucs.poo.exemplointerfaces.shapes;
```

```
public class Point implements Shape {
```

```
    protected double x, y;
```

```
    public Point() {  
        setPoint(0,0);  
    }
```

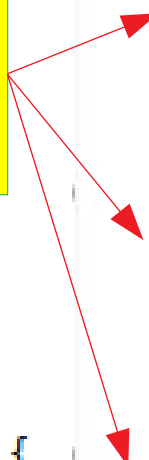
```
    public Point(double x, double y) {  
        setPoint(x,y);  
    }
```

```
    public void setPoint(double x, double y) {  
        this.x = x;  
        this.y = y;  
    }
```

```
    public double getX() {  
        return x;  
    }
```

```
    public double getY() {  
        return y;  
    }
```

Aqui estão
os métodos
obrigatórios



```
    @Override  
    public double area() {  
        return 0;  
    }
```

```
    @Override  
    public double volume() {  
        return 0;  
    }
```

```
    @Override  
    public String getName() {  
        return "Point";  
    }
```

```
    @Override  
    public String toString() {  
        return "[" + x + "," + y + "];"  
    }
```

```
}
```

Classe **Rectangle**

```
package br.ucs.poo.exemplointerfaces.shapes;
```

```
public class Rectangle implements Shape {
```

```
    protected double a, b;
```

```
    public Rectangle() {  
        setRectangle(0, 0);  
    }
```

```
    public Rectangle(double a, double b) {  
        setRectangle(a, b);  
    }
```

```
    public void setRectangle(double a, double b) {  
        this.a = a > 0 ? a : 0;  
        this.b = b > 0 ? b : 0;  
    }
```

```
    public double getA() {  
        return a;  
    }
```

```
    public double getB() {  
        return b;  
    }
```

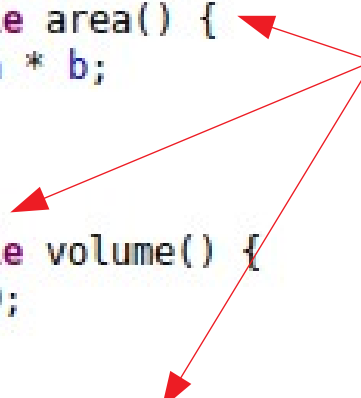
```
@Override  
public double area() {  
    return a * b;  
}
```

```
@Override  
public double volume() {  
    return 0;  
}
```

```
@Override  
public String getName() {  
    return "Rectangle";  
}
```

```
public String toString() {  
    return "[ a = " + a + ", b = " + b + " ]";  
}
```

Aqui estão
os métodos
obrigatórios

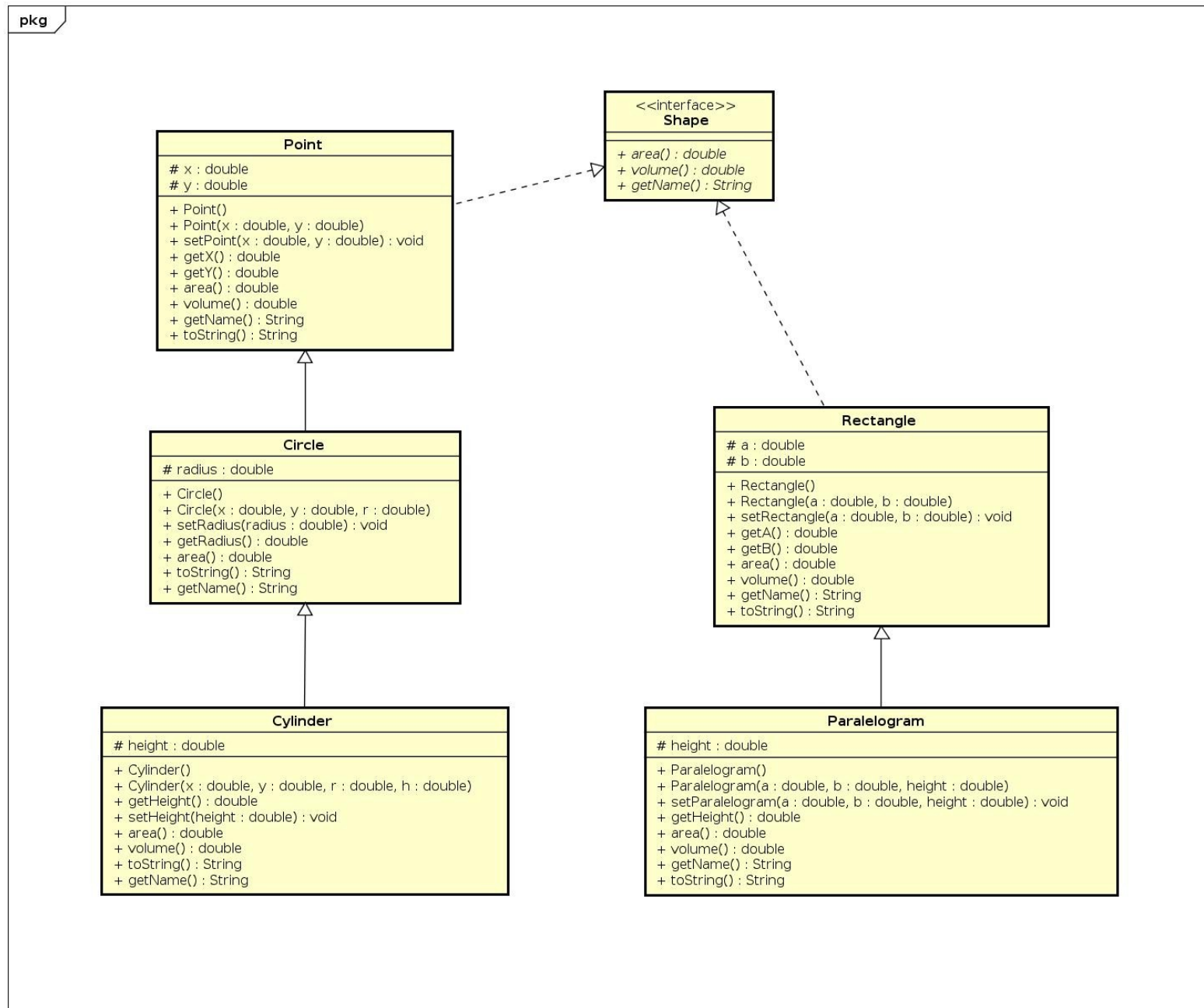




Herança e implementação

- Pode-se combinar herança (*extends*) e implementação (*implements*).

Herança e implementação



Múltiplas interfaces

- Também pode-se implementar múltiplas interfaces

```
package br.ucs.poo.exemplointerfaces.touch;  
  
public interface Touchable {  
    boolean isTouched();  
}  
  
public class Rectangle implements Shape, Touchable {
```

Múltiplas interfaces

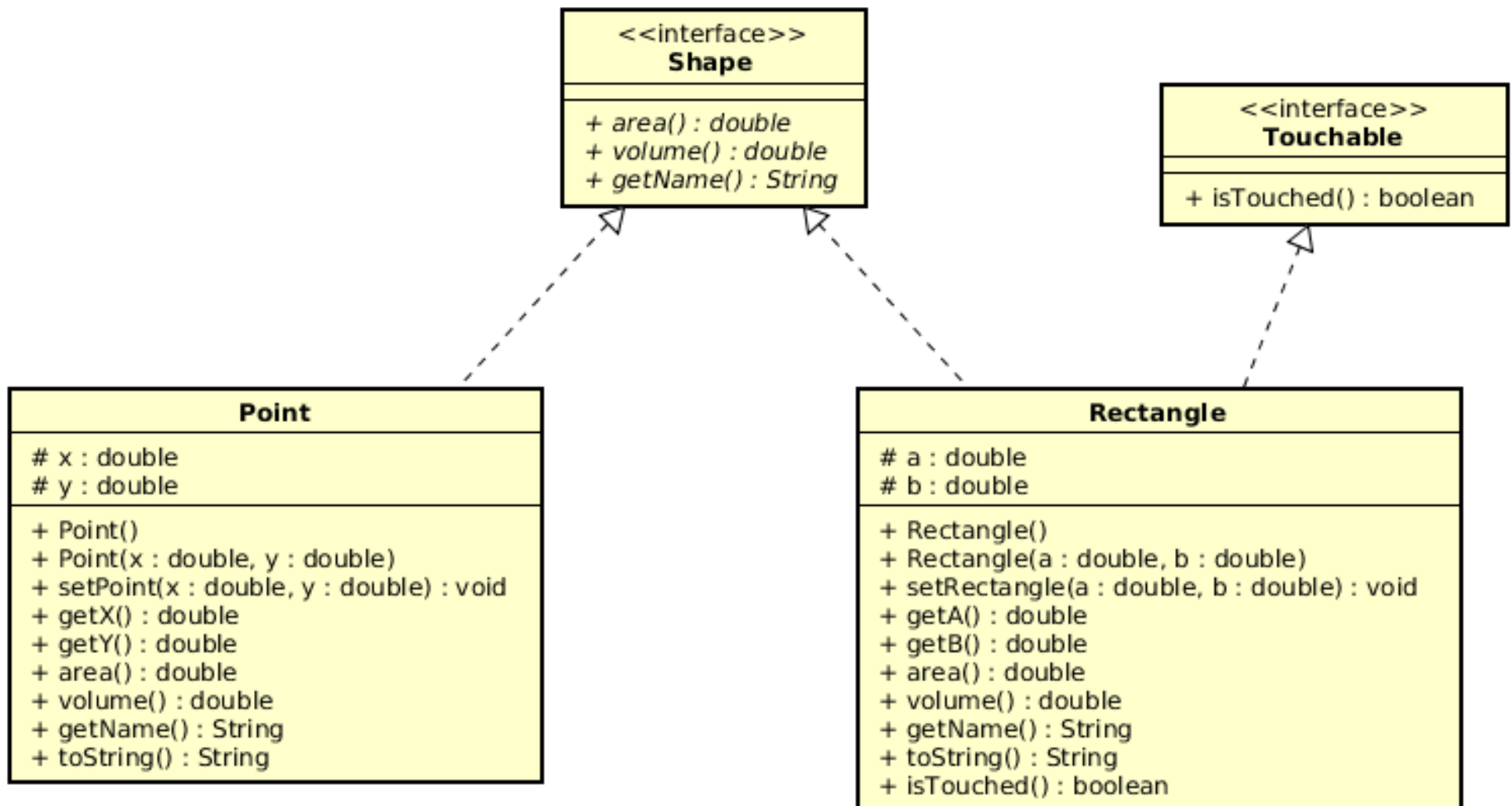
- Também pode-se implementar múltiplas interfaces

```
package br.ucs.poo.exemplointerfaces.touch;  
  
public interface Touchable {  
    boolean isTouched();  
}
```

Separa-se as interfaces
Implementas usando
vírgulas

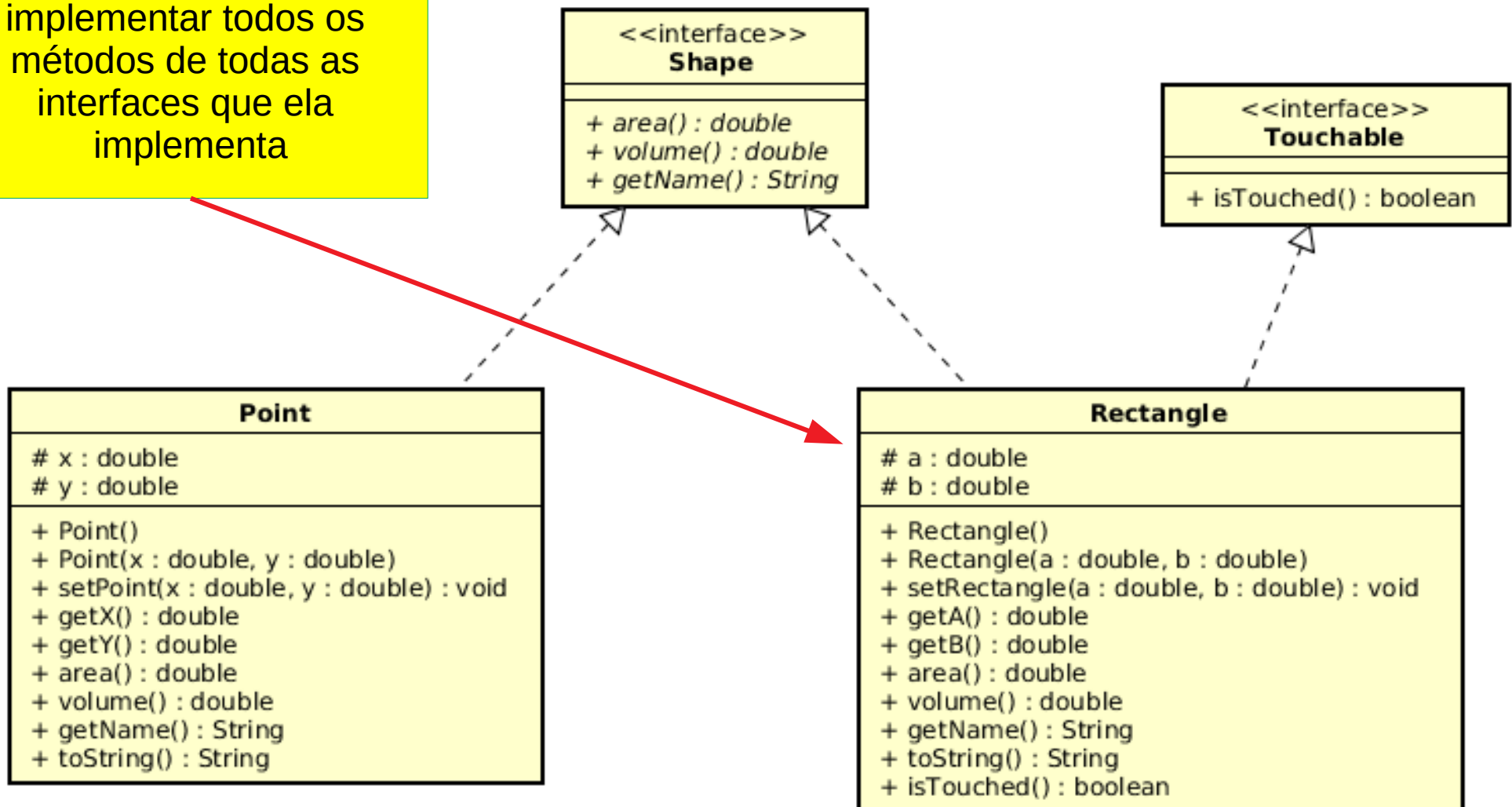
```
public class Rectangle implements Shape, Touchable {
```

Múltiplas interfaces



Múltiplas interfaces

A classe Rectangle deve implementar todos os métodos de todas as interfaces que ela implementa





Roteiro

- Pacotes
 - Definição e Utilização
 - Encapsulamento
- Interfaces
- Polimorfismo
- Exercícios



Interface Java : Polimorfismo

- De forma similar a da Herança, também posso usar a expressão “*é um*”
- No nosso exemplo,
 - Ponto *é uma* Forma
 - Retângulo *é uma* Forma



Polimorfismo

- No polimorfismo, pode-se invocar operações de uma classe sem saber exatamente qual é a classe que se está chamando, desde que exista a relação “**é um**”



Polimorfismo

```
Shape forma1 = new Point(7, 11);  
Shape forma4 = new Rectangle(3, 4);
```


Polimorfismo

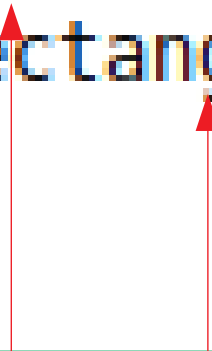
```
Shape forma1 = new Point(7, 11);  
Shape forma4 = new Rectangle(3, 4);
```



Declara-se as variáveis pela sua
abstração, no caso aqui,
pela **interface** Shape

Polimorfismo

```
Shape forma1 = new Point(7, 11);  
Shape forma4 = new Rectangle(3, 4);
```



E as instanciamos pela sua
Concretização, no caso as **classes**
Point e Rectangle

Polimorfismo

- Quando invoco os métodos, não sei qual a classe atenderá:

```
System.out.println(forma1.getName() + " : " + forma1);  
System.out.println("Area = " + forma1.area());  
System.out.println("Volume = " + forma1.volume() + "\n");  
  
System.out.println(forma4.getName() + " : " + forma4);  
System.out.println("Area = " + forma4.area());  
System.out.println("Volume = " + forma4.volume() + "\n");
```



Polimorfismo e Vetores

- O polimorfismo fica mais evidente quando começamos a colocar os objetos em estruturas de dados como vetores e listas.
- Isso será o assunto da próxima aula.



Dúvidas?





Atividades

- Execute as atividades presentes no documento

07.Lista.de.Exercícios.POO.pdf

Próximos passos



- Vetores