

# Interfaces

---

# Interfaces

---

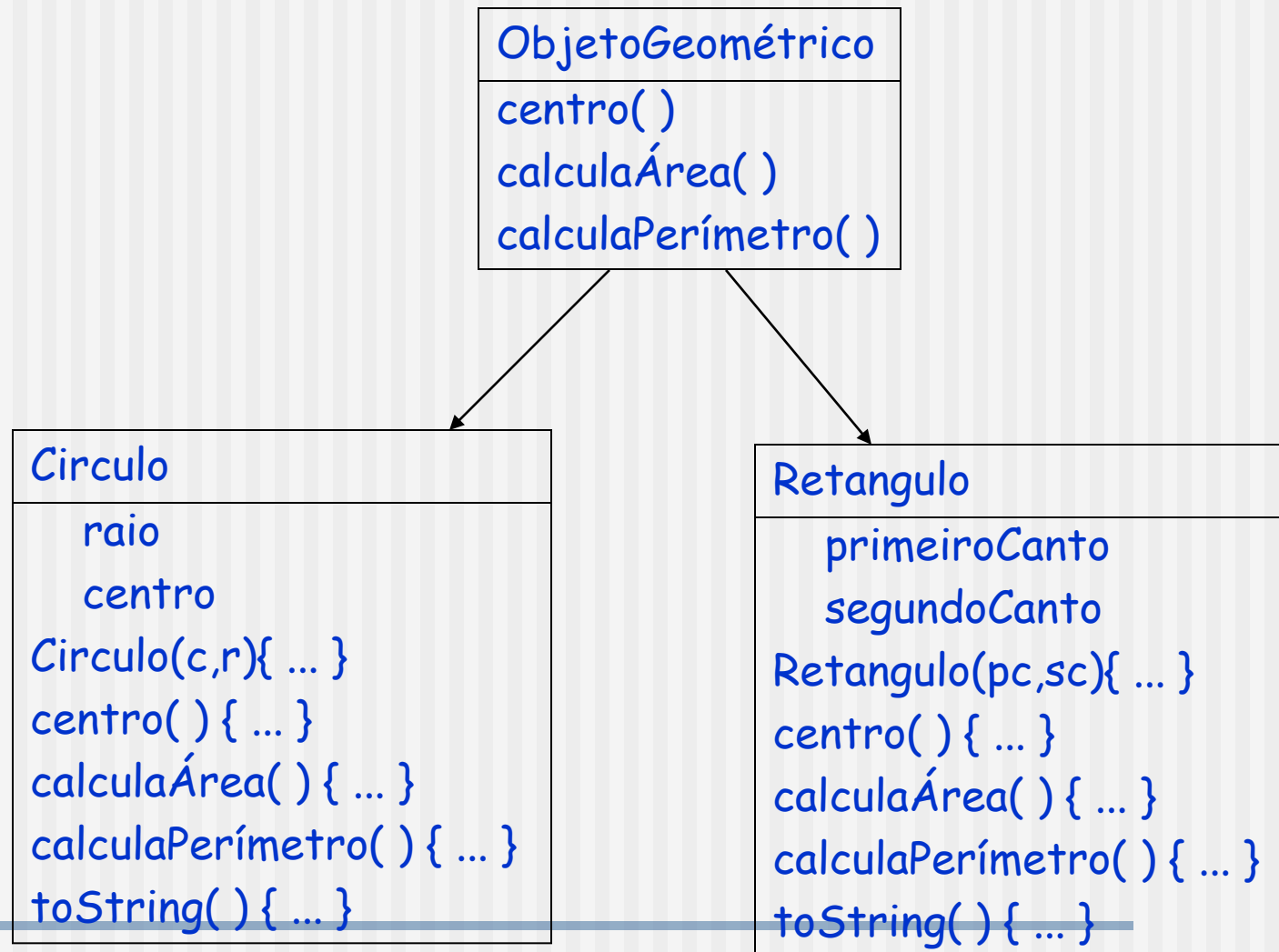
- Classes abstratas que possuem apenas métodos abstratos

Na Interface:

- Métodos são implicitamente  
abstract e public
- Campos são implicitamente  
static e final
- Não possuem construtores

assim como as  
classes abstratas,  
as interfaces não  
podem ser  
instanciadas.

# Interfaces



# Interfaces

```
interface ObjetoGeometrico
```

```
{
```

```
    Ponto2D centro();
```

```
    double calculaÁrea();
```

```
    double calculaPerímetro();
```

```
} // fim da interface ObjetoGeometrico
```

declaração diferente de classes

métodos sem modificadores `abstract`  
e `public`

É possível fazer o uso de outras  
classes na assinatura dos métodos.

ObjetoGeométrico.java

# Interfaces

```
class Circulo implements ObjetoGeometrico {
    private Ponto2D centro;
    private double raio;
    Circulo(Ponto2D centro, double raio) {
        this.centro = centro; this.raio = raio;
    }
    public Ponto2D centro() {
        return centro;
    }
    public double calculaÁrea() {
        return Math.PI*raio*raio;
    }
    public double calculaPerímetro() {
        return 2.0*Math.PI*raio;
    }
    public String toString() {
        return
            "Círculo com centro em
            "+centro+" e raio "+raio;
    }
} // fim da classe Circulo
```

Cláusula de herança

Todos os métodos da interface são implementados.

Devem ser sobrescritos com modificador public.

Aula10: private, protected ou modificador ausente tornariam o acesso mais restritivo.

Aula10: modificador static também não é permitido

Circulo.java

# Interfaces

```
class Retangulo implements ObjetoGeometrico {
private Ponto2D primeiroCanto,segundoCanto;
Retangulo(Ponto2D pc,Ponto2D sc) {
    primeiroCanto = pc; segundoCanto = sc; }
public Ponto2D centro() {
    double coordX = (primeiroCanto.getX()+segundoCanto.getX())/2.;
    double coordY = (primeiroCanto.getY()+segundoCanto.getY())/2.;
    return new Ponto2D(coordX,coordY); }
public double calculaÁrea() {
    ... }
public double calculaPerímetro() {
    .... }
public String toString() {
    return "Retângulo com cantos "+primeiroCanto+" e "+segundoCanto; }
} // fim da classe Retangulo
```

Métodos da interface  
implementados de forma  
diferente da classe Circulo

Retangulo.java

# Interfaces

```
class DemoObjetosGeometricos {
public static void main(String[] argumentos) {
    Circulo c1 = new Circulo(new Ponto2D(0,0),100);
    Retangulo r1 = new Retangulo(new Ponto2D(-2,2),
                                new Ponto2D(2,2));
    imprimeTodosOsDados(c1);
    imprimeTodosOsDados(r1);
}
private static void imprimeTodosOsDados(ObjetoGeometrico og) {
    System.out.println(og);
    System.out.println("Perímetro:"+og.calculaPerímetro());
    System.out.println("Área:"+og.calculaÁrea());
    System.out.println();
}
}
```

A interface como parâmetro  
ou retorno de método:  
instância de subclasse deverá  
ser passada.

DemoObjetosGeometricos.java

# Interfaces

```
class DemoObjetosGeometricosEPolimorfismo {
public static void main(String[] argumentos) {
    ObjetoGeometrico o1,o2;
    o1 = new Circulo(new Ponto2D(0,0),20);
    o2 = new Retangulo(new Ponto2D(-1,-1),
        new Ponto2D(1,1));
    System.out.println("o1 é um Círculo ? "+
        (o1 instanceof Circulo));
    System.out.println("o1 é um Retângulo ? "+
        (o1 instanceof Retangulo));
    System.out.println("o1 é um ObjetoGeometrico ? "+
        (o1 instanceof ObjetoGeometrico));
    ....
}
} // fim da classe DemoObjetosGeometricosEPolimorfismo
```

Referências à interface ....  
.... apontando para instâncias  
das subclasses.  
(Polimorfismo: aula10).

Verifique que o1 é  
círculo e também é  
objeto geométrico

DemoObjetosGeometricosEPolimorfismo.java



# Interfaces e Herança Múltipla

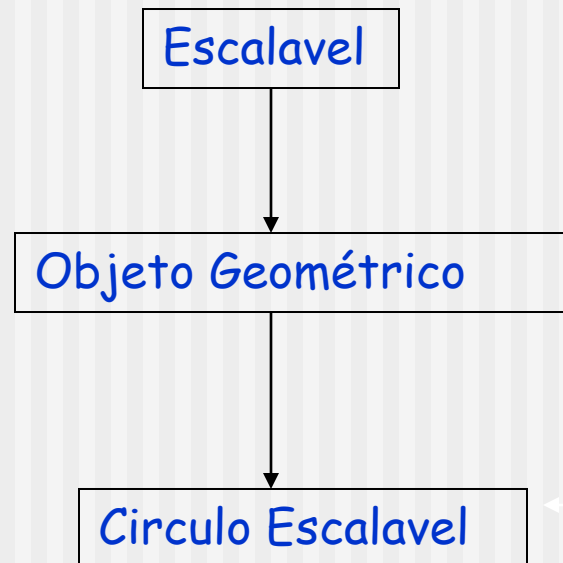
---

- Modelar Objetos Geométricos
  - Modelar Objetos escaláveis

Nem todo objeto geométrico deve ser escalável !!!

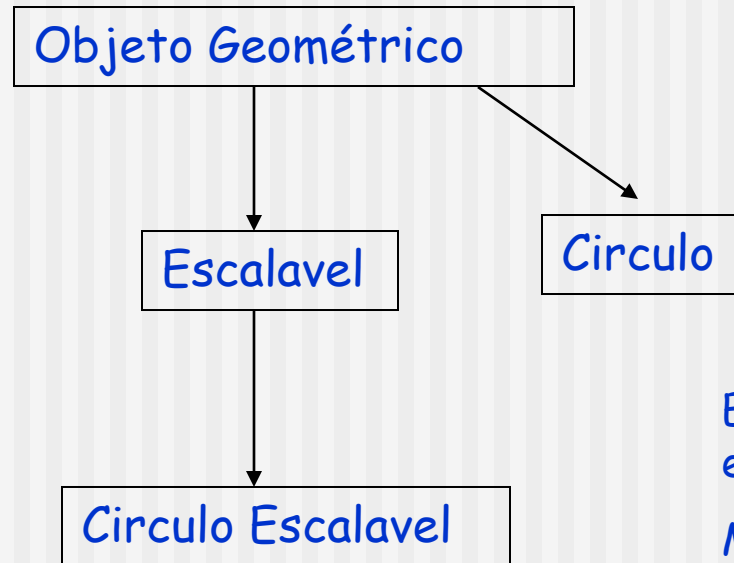
Nem todo objeto escalável deve ser geométrico !!!

# Interfaces e Herança Múltipla



Toda subclasse neste nível  
é necessariamente Objeto  
Geométrico e escalável !!!  
Como fazer um objeto  
geométrico que não seja  
escalável ???

# Interfaces e Herança Múltipla



Este círculo não é escalável !!!

Mas e se for necessário definir outras classes que não sejam objetos geométricos e que sejam escaláveis????

# Interfaces e Herança Múltipla

---

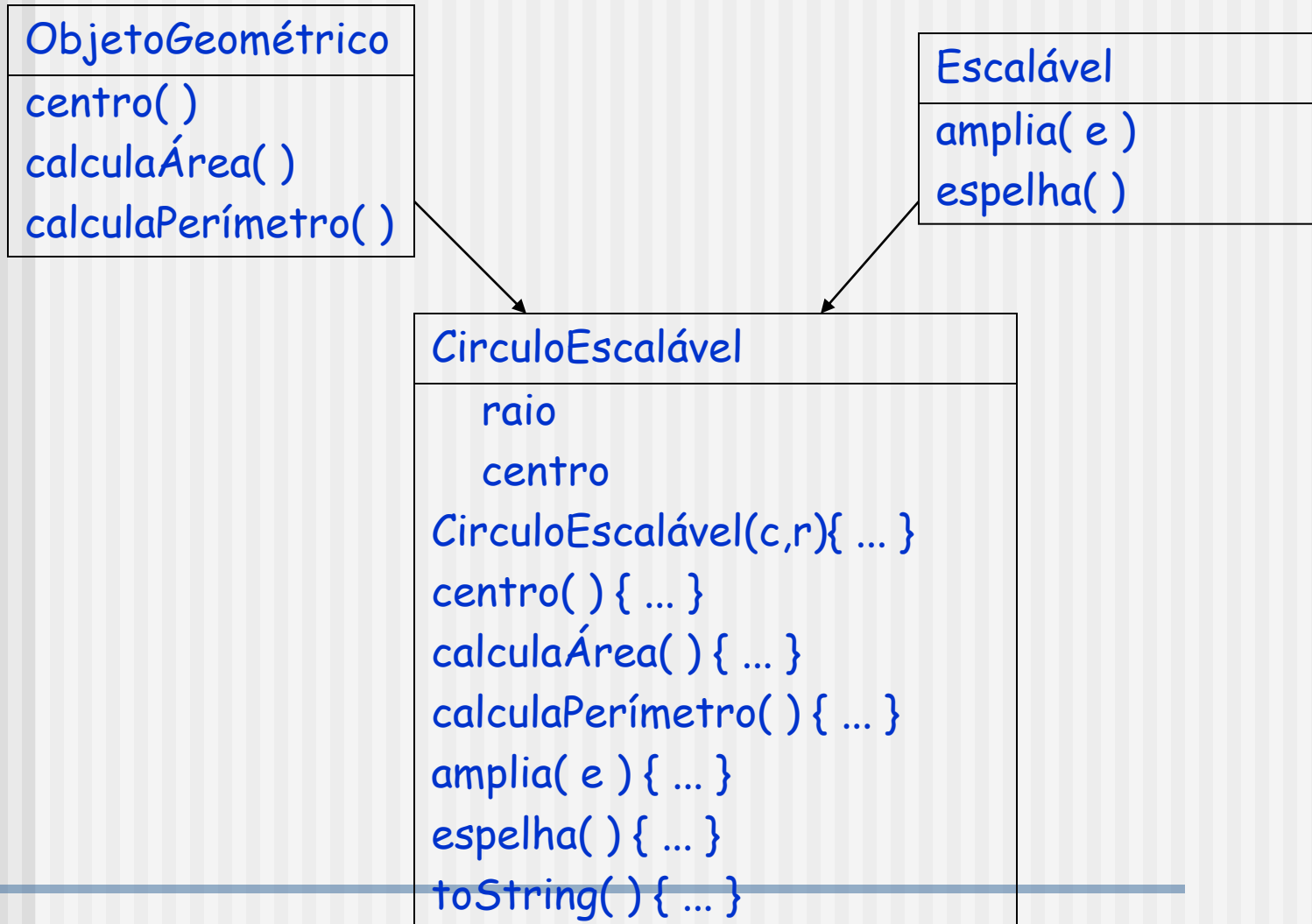
- Objetos Geométricos
- Objetos Escaláveis

*Características  
independentes !!!*

Nem todo objeto geométrico é escalável !!!

Nem todo objeto escalável é geométrico !!!

# Interfaces: Herança Múltipla "controlada"



# Interfaces

---


```
interface Escalavel  
{  
void amplia(double escala);  
void espelha();  
  
} // fim da interface Escalavel
```

Escalavel.java

# Interfaces

```
class CirculoEscalavel implements ObjetoGeometrico, Escalavel {  
    private Ponto2D centro;  
    private double raio;  
    CirculoEscalavel(Ponto2D centro, double raio) {  
        this.centro = centro;  
        this.raio = raio;    }  
    public Ponto2D centro() {  
        return centro;    }  
    public double calculaÁrea() {  
        return Math.PI*raio*raio;    }  
    public double calculaPerímetro() {  
        return 2.0*Math.PI*raio;    }  
    public void amplia(double escala) {  
        raio *= escala;    }  
    public void espelha() {  
        centro = new Ponto2D(-centro.getX(),centro.getY());    }  
    public String toString() {  
        return "Círculo com centro em "+centro+" e raio "+raio;    }  
}
```

cláusula de  
herança múltipla



# Conflitos em Herança Múltipla

---

- Conflitos de métodos:
  - As superclasses possuem métodos com mesma assinatura. Qual deles herdar???
- Conflitos de campos:
  - As superclasses possuem campos com mesmo nome. Qual deles herdar ???



# Interfaces: colisões de nomes

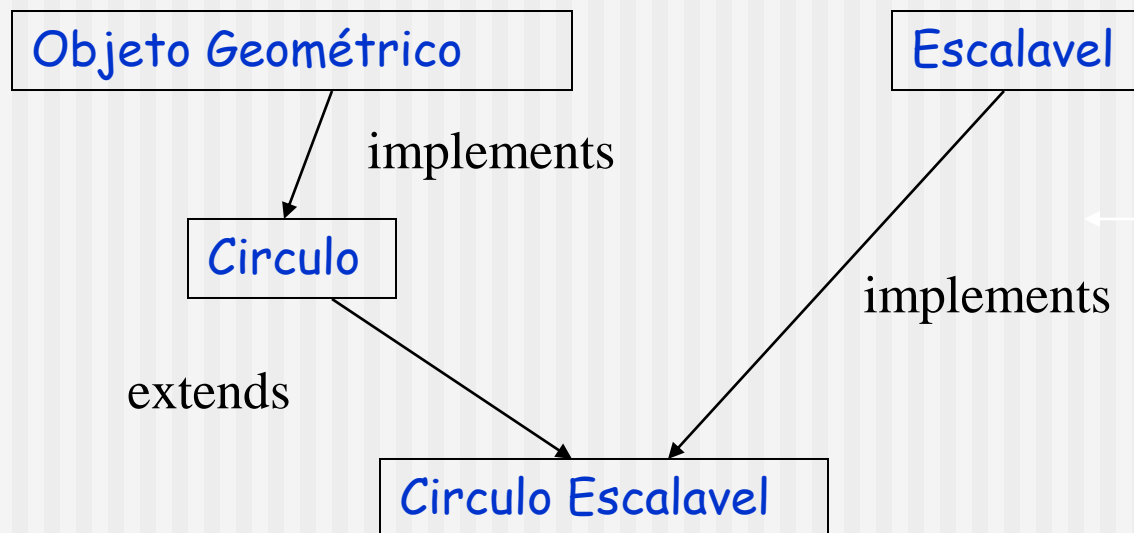
- O que acontece quando uma classe implementa várias interfaces e, por exemplo, dois métodos dessas interfaces têm o mesmo nome?
  - Se não têm a mesma assinatura, considera-se uma sobrecarga
  - Se têm a mesma assinatura e o mesmo tipo de retorno são considerados como um só método (fusão).
  - Se têm a mesma assinatura e o tipo de retorno é diferente, será gerado um erro de compilação
  - Se têm a mesma assinatura e o mesmo tipo de retorno mas lançam exceções de tipos diferentes, são considerados o mesmo método e o `throws` da implementação é a união das duas listas de `throws`

# Interfaces: colisões de nomes

```
public interface X {  
    public void method1(int i);  
    public void method2(int i);  
    public void method3(int i);  
    public void method4(int i) throws Exception1;  
}  
  
public interface Y {  
    public void method1(double i);  
    public void method2(int i);  
    public int method3(int i);  
    public void method4(int i) throws Exception2;  
}  
  
public class MyClass implements X, Y {  
    public void method1(int i) { ... } // redefine method1 de X  
    public void method1(double i) { ... } // redefine method1 de Y  
    public void method2(int i) { ... } // redefine method2 de X e Y  
    public void method4(int i) // redefine method4 de X e Y  
        throws Exception1, Exception2  
        { ... }  
}
```

# Classes abstratas e interfaces

## Exercício:



É possível?

DemoCirculoEscalável continua funcionando?