

Classes Abstratas e Interfaces

MATA 55 - LPOO



Classes Abstratas

- Muitas vezes é preferível definir métodos que classes herdeiras devem implementar
 - sem que seja possível instanciar a classe ancestral.
- Nesses casos, a classe ancestral seria apenas “um guia”:
 - de quais atributos as classes devem herdar.
 - quais métodos as classes herdeiras devem implementar concretamente.



Classes Abstratas

- Não é possível instanciá-la
 - não há concretização (objeto) da mesma.
- Agrupam características e comportamentos que serão herdados por outras classes
- Fornecem padrões de comportamento que serão implementados nas subclasses



Classes Abstratas

- Poderoso Mecanismo de Abstração:
 - Permite a herança do código sem violar a noção de subtipo
 - Devem ser estendidas
 - Diz o que deve ter a subclasse, mas não diz como !
- A classe abstrata:
 - código genérico, livre de particularidades (O que)
- As subclasses:
 - detalhes particulares (como)



Classes Abstratas

Exemplo:

Círculos, Quadrados e Triângulos.

Círculo
x,y raio move(x,y) { ... } mostra() { ... } alteraDiâmetro(d) { ... }

Quadrado
x,y lado move(x,y) { ... } mostra() { ... } alteraLado(d) { ... }

Triângulo
x,y l1,l2,l3 move(x,y) { ... } mostra() { ... } alteraLado1(l) { ... } alteraLado2(l) { ... } alteraLado3(l) { ... }



Classes Abstratas

- Métodos Abstratos

- Só a assinatura, sem corpo
- Precisam ser implementados pelas subclasses (folhas)
- Só podem existir em classes abstratas
- A classe abstrata deve enumerar características genéricas (métodos abstratos) do modelo
- As classes que derivarem de um classe abstrata devem obrigatoriamente implementar todos os métodos abstratos definidos na classe Pai.
 - Para isto, cada subclasse se utiliza de seus detalhes particulares

- Métodos que não são abstratos podem ser usados normalmente pelos objetos das classes que derivam da classe abstrata.



Classes Abstratas

- Sintaxe em Java:

```
public abstract class MinhaClasseAbstrata{...}  
public class MinhaClasseConcreta extends  
    MinhaClasseAbstrata {...}
```

- Classes abstratas permitem que métodos sejam declarados como abstratos.

```
public abstract boolean saque(float valor);  
public abstract float calculaImposto(int opcao);
```

- Classes abstratas também podem conter métodos concretos.

```
public boolean deposito(float valor){...}
```

Classes Abstratas -Exemplo

```
abstract class Figura
{  int x;  // coordenada x
  int y;  // coordenada y
```

Definição da classe abstrata, através da palavra **abstract**.

```
public Figura (int x1, int y1)
{  x = x1;
  y = y1;
}
```

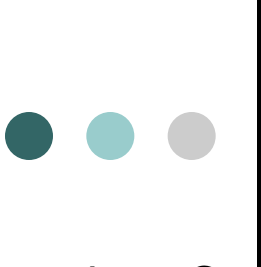
A classe pode ter um construtor, mesmo que não possa instanciar objetos.

```
public abstract void desenha();
public abstract void apaga();
```

Declaração dos métodos abstratos. Esses métodos devem obrigatoriamente ser implementados nas classes derivadas.

```
public void move (int x1, int y1)
{  apaga();
  x = x1;
  y = y1;
  desenha();
}
}
```

Declaração de um método normal que poderá ser utilizado pelos objetos de classes derivadas.



```
class Quadrado extends Figura
{
    public Quadrado(int x1, int y1)
    {
        super(x1, y1);
    }

    public void desenha()
    {
        System.out.println("Desenhando quadrado (" + x + "," + y + ")");
    }

    public void apaga()
    {
        System.out.println("Apagando quadrado (" + x + "," + y + ")");
    }
}
```

Utilização do construtor da classe Pai.

Implementação obrigatória dos métodos definidos na classe abstrata.

Classe para testar o exemplo.

```
class TesteAbstract
{
    public static void main (String args[])
    {
        Quadrado q = new Quadrado(10,10);
        q.desenha();
        q.move(50,50);
        q.apaga();
    }
}
```



Classes Abstratas

○ Considerações Finais

- Classes abstratas pois possuem ao menos um método abstrato.
- Não existem “campos abstratos”.
- Um construtor não pode ser abstrato
 - Seu código é necessário para inicializar corretamente os campos da classe abstrata



Exercício 1

- Re-escrever o exercicio animais

Criar o modelo e implementar em Java uma classe que represente animais. A classe possui os atributos tipo (mamífero, anfíbio, ave), nome, idade. Esta classe deve possuir o seguinte método:

- método para calcular a quantidade de alimento consumido, sabendo-se que a cada ano de vida um mamífero come 2 quilos/dia, uma ave come 100g/dia e um anfíbio 20g/dia.



Exercício 2

- Uma conta pode ser corrente ou poupança. Uma conta possui um nome do titular, agencia e um saldo. Contas corrente possuem uma taxa de débito mensal e um limite. Uma conta poupança possuem um rendimento mensal.
 - Saque, deposito, rendimento mensal



Interfaces

- Fornecem um contrato entre a classe e o mundo externo
- Quando uma classe implementa uma interface, ela está comprometida a fornecer o comportamento publicado pela interface
- Programador sabe o que o método faz, não sabe como..



Interfaces

○ Exemplo:

Interface

```
interface IConta {  
    double saldo();  
    int numConta();  
}
```

```
class ContaCorrente implements IConta {  
    ...  
    double saldo() {  
        // código específico  
    }  
  
    int numConta() {  
        // código específico  
    }  
    ...  
}
```



Interfaces

- Se uma classe é definida por apenas métodos abstratos então é melhor não usar a palavra-chave `abstract`.
- Para estes caso o Java fornece uma técnica chamada de interfaces que pode ser usada para definir um modelo de comportamento.
- As interfaces diferem das classes abstratas no fato de que nenhum método da interface pode ter um corpo.
- As interfaces são estritamente modelos.
- As interfaces não podem ser instanciadas.
- Uma classe pode implementar várias interfaces mas apenas herda (`extends`) de uma única classe.
- As Interfaces são compiladas da mesma forma que uma classe.



Interfaces

- Uma coleção de definição de métodos.
 - pode também declarar constantes.
- Java possui a palavra reservada **interface** para indicar a definição de uma interface.
- Sintaticamente é equivalente a uma classe completamente abstrata.
 - Todos seus métodos são abstratos;
 - Todas as suas variáveis são **public static final**;
- Semanticamente entretanto, as duas diferem, pois uma interface guarda uma idéia de protocolo (compromisso) entre classes



Interfaces

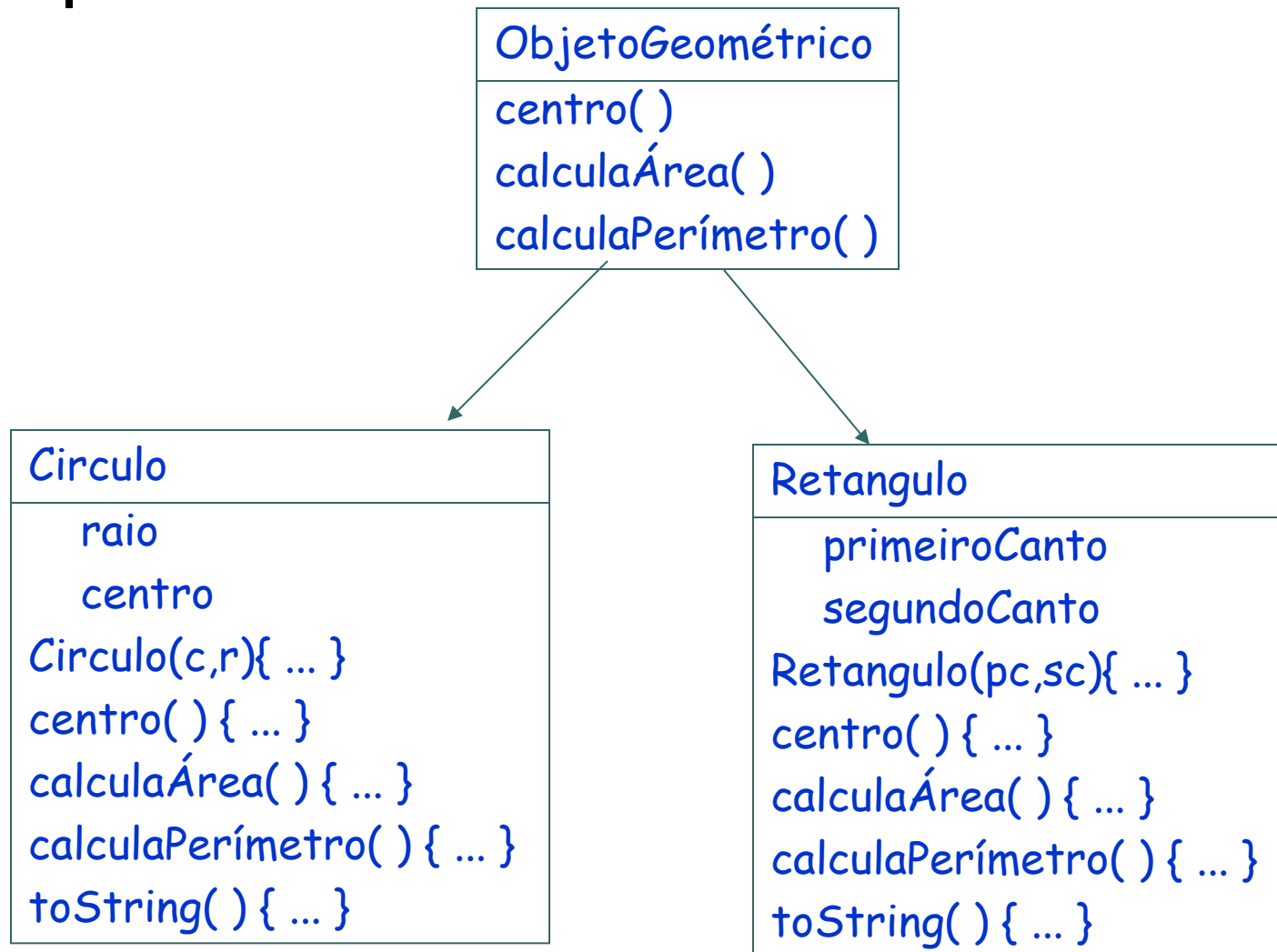
- Resumindo....
- Classes abstratas que possuem apenas métodos abstratos

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.

Exemplo





```
interface ObjetoGeometrico
```

```
{
```

```
    Ponto2D centro();
```

```
    double calculaÁrea();
```

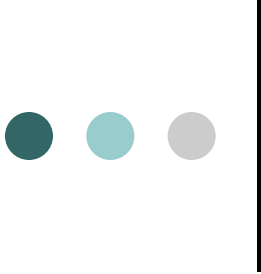
```
    double calculaPerímetro();
```

```
} // 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.



```
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

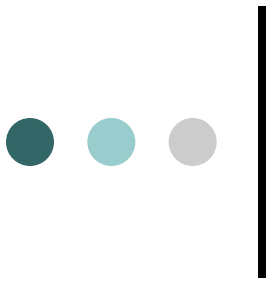
Todos os métodos da interface
são implementados.

Devem ser sobrescritos com
modificador **public**.

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

modificador **static** também não
é permitido

Circulo.java



```
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



```
class DemoObjetosGeometricos
```

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

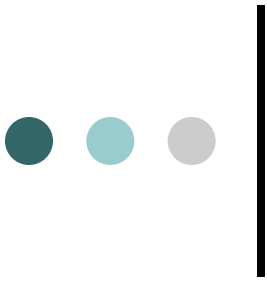
```
        Circulo c1 = new Circulo(new Ponto2D(0,0),100);  
        Circulo c2 = new Circulo(new Ponto2D(-1,-1),1);  
        Circulo c3 = new Circulo(new Ponto2D(10,8),0);  
        Retangulo r1 = new Retangulo(new Ponto2D(-2,-2), new Ponto2D(2,2));  
        Retangulo r2 = new Retangulo(new Ponto2D(-100,-1), new Ponto2D(100,1));  
        Retangulo r3 = new Retangulo(new Ponto2D(0,0), new Ponto2D(0,0));  
        imprimeTodosOsDados(c1);  
        imprimeTodosOsDados(c2);  
        imprimeTodosOsDados(c3);  
        imprimeTodosOsDados(r1);  
        imprimeTodosOsDados(r2);  
        imprimeTodosOsDados(r3);  
    }  
}
```



continuação

```
private static void imprimeTodosOsDados(ObjetoGeometrico og)
{
    System.out.println(og);
    System.out.println("Perímetro:"+og.calculaPerimetro());
    System.out.println("Area:"+og.calculaArea());

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



```
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:).

Verifique que o1 é
círculo e também é
objeto geométrico

DemoObjetosGeometricosEPolimorfismo.java



Classes Abstratas e Interfaces

Classes (abstratas)	Interfaces
Agrupa objetos com implementações compartilhadas	Agrupa objetos com implementações diferentes
Define novas classes através de herança de código	Define novas interfaces através de herança de assinaturas
Só uma classe pode ser supertipo de outra classe	Várias interfaces podem ser supertipo do mesmo tipo



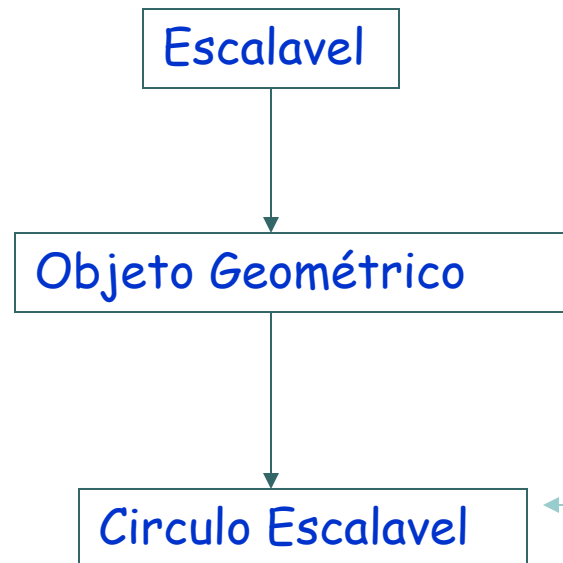
Interfaces e Herança Múltipla

- Modelar Objetos Geométricos
 - Modelar Objetos escaláveis

Nem todo objeto geométrico deve ser escalável !!!

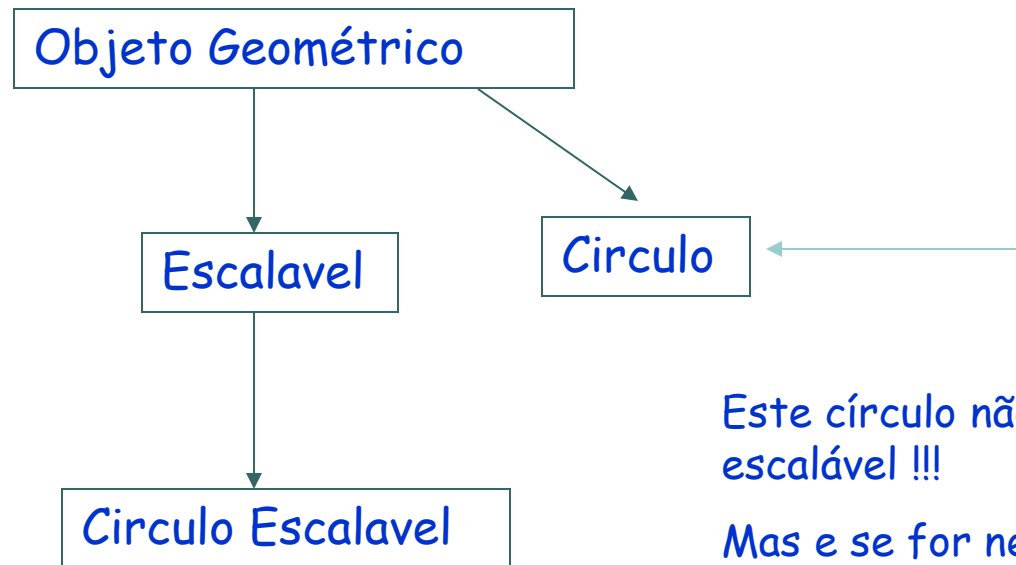
Nem todo objeto escalável deve ser geométrico !!!

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????



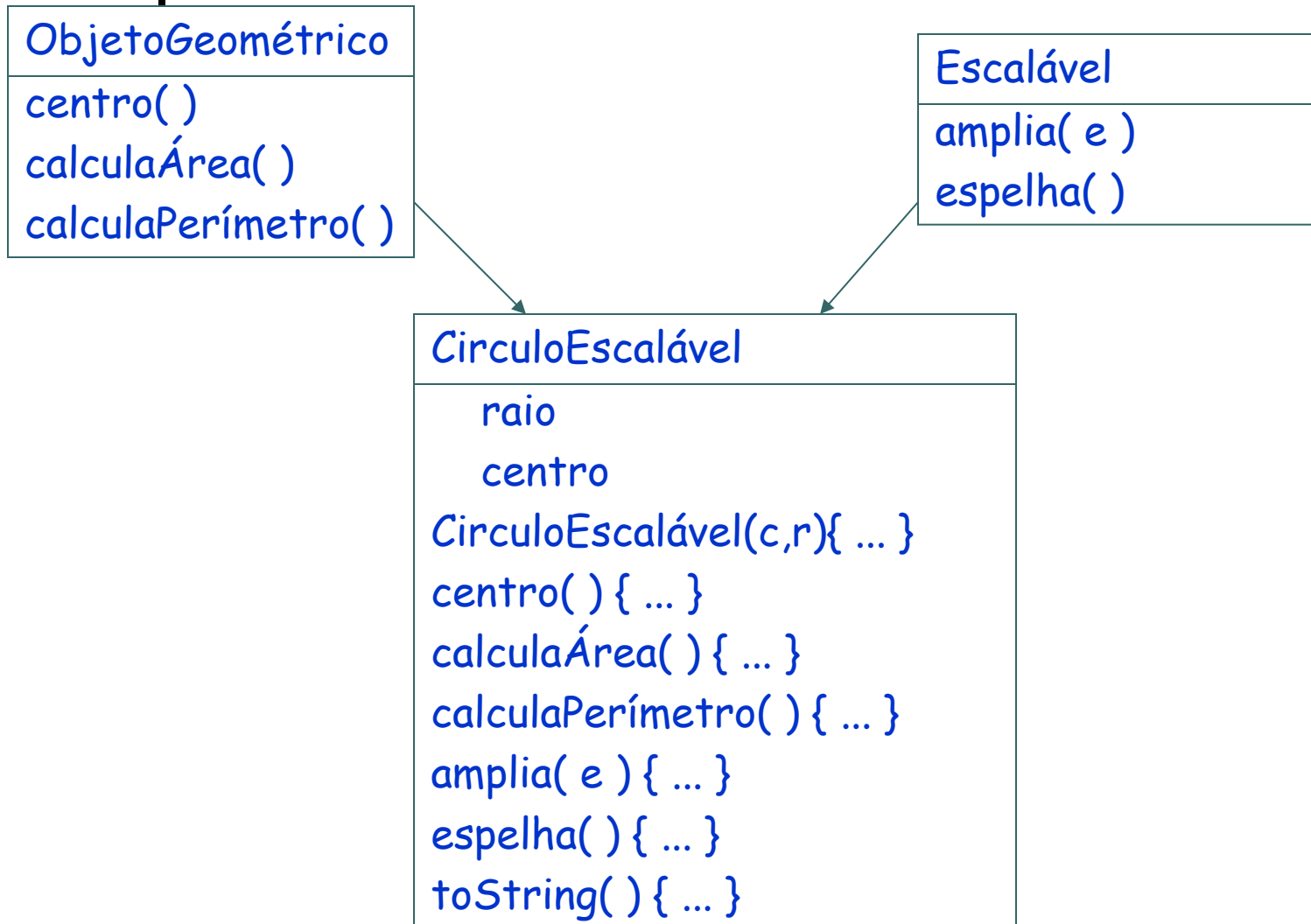
Interfaces e Herança Múltipla

- 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() {
```

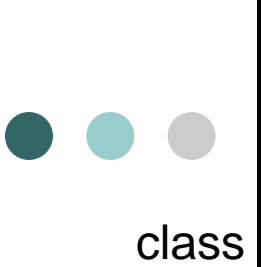
```
        centro = new Ponto2D(-centro.getX(), centro.getY());    }
```

```
    public String toString() {
```

```
        return "Círculo com centro em "+centro+" e raio "+raio;    }
```

```
}
```

cláusula de
herança múltipla



```
class DemoCirculoEscalavel {
    public static void main(String[] argumentos)
    {

        CirculoEscalavel ce = new CirculoEscalavel(newPonto2D(10,10),30);
        System.out.println(ce);
        ce.amplia(3);
        System.out.println(ce);
        ce.espelha();
        System.out.println(ce);
        System.out.println(ce instanceof CirculoEscalavel); // true
        System.out.println(ce instanceof ObjetoGeometrico); // true
        System.out.println(ce instanceof Escalavel);      // true
    }
}
```



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 ???

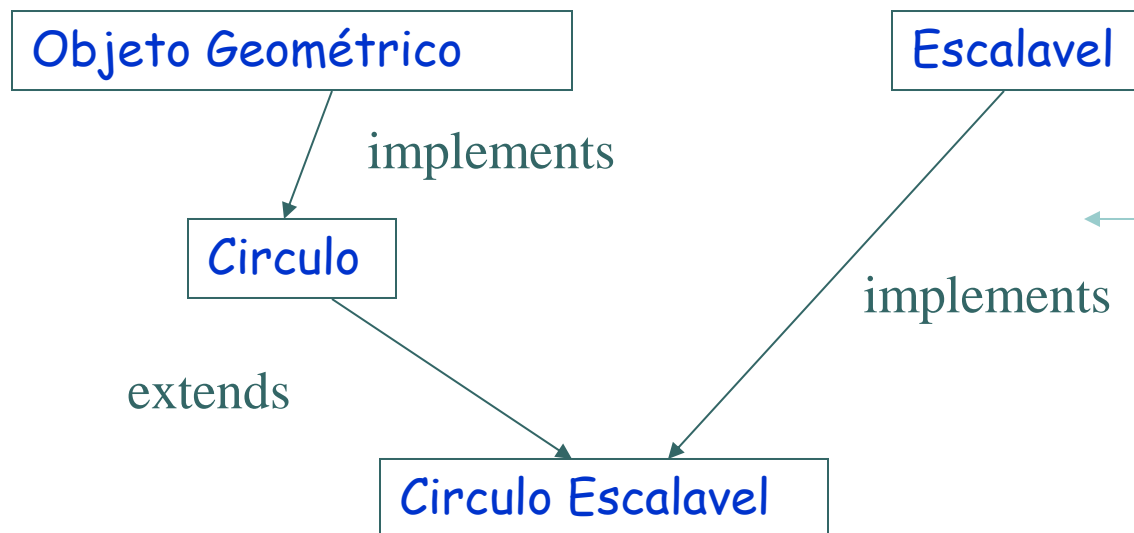


Conflitos em Herança Múltipla

- Solução de C++: herança seletiva
- Solução de Java: interfaces
 - Não há conflito de métodos porque a sobrescrição é obrigatória nas classes herdeiras
 - O compilador detecta conflito de campos e não compila a classe herdeira.

Classes abstratas e interfaces

Exercício:



É possível?

DemoCirculoEscalável continua funcionando?