Programação Orientada por Objectos

Interfaces

Prof. Rui César das Neves, Prof. José Cordeiro

Departamento de Sistemas e Informática
Escola Superior de Tecnologia de Setúbal – Instituto Politécnico de Setúbal
2014/2015

Sumário

- Interface
  - O que é?
  - Para que serve?
  - Palavra reservada "`implements`"
  - Interface como Tipo de Dados
  - Exemplos
  - Herança Simples e Múltipla entre Interfaces
- Interfaces vs Classes Abstratas

Interfaces

- **`interface`** - é **um conjunto de declarações de métodos** que representam um comportamento particular e que qualquer classe pode implementar
  - A definição de uma interface em Java especifica um conjunto de métodos, implicitamente **abstratos e `public`**, e opcionalmente um conjunto de constantes, implicitamente **`public static final`**
- **Para quê?**
  - Recorremos às interfaces para especificar comportamentos comuns a diferentes classes que partilhem esses comportamentos independentemente das hierarquias de herança
  - EX:

```
public interface Movimentavel{
    int MOVIMENTO_MINIMO = 1;       // public static final
    void moverEmX(int dx);          // public
    void moverEmY(int dy);          // public
    void deslocar(int dx, int dy); // public
    Ponto obterPosicao();           // public
}
```

Interfaces

- **`implements`** - palavra reservada do Java usada para especificar que uma determinada classe implementa uma determinada interface
  - **Uma classe que declare implementar uma interface terá de poder executar todos os métodos dessa interface**, seja porque os implementa ou porque herda as suas implementações.

```
public interface Movimentavel{
  int MOVIMENTO_MINIMO = 1;
  void moverEmX(int dx);
  void moverEmY(int dy);
  void deslocar (int dx, int dy);
  Ponto obterPosicao();
}
```

```
public class Retangulo
      extends FormaGeometrica
      implements Movimentavel {
  // […] Restante Membros de Retangulo
  // implementação da interface
  public void moverEmX(int dx) {
    deslocar(dx, 0);setX2 (x2+dx);}
  public void moverEmY(int dy){
    deslocar(0, dy);setY2 (y2+dy);}
  public void deslocar (int dx, int dy) {
        setX(dx);
        setY(dy);}
  public Ponto obterPosicao() {
        return new Ponto (getX(),getY());}
}
```

Interface como Tipo de Dados

- **Tipo de Dados** - Uma especificação de um formato de dados em que se define a gama de valores possíveis e as operações passíveis de efectuar com os mesmos.
  - Uma Interface especifica um tipo de dados
    - i. e declara um conjunto de métodos abstratos a definir nas classes que implementem essa interface.
  - Qualquer classe que implemente essa interface passa a ser compatível com esse tipo de dados
    - i.e. podemos guardar numa variável dessa interface objectos dessas classes
    - EX:

```
package formas;
public class Main {
    public static void main(String[] args) {

        Movimentavel movimentavel = new Retangulo();
        movimentavel.moverEmX(4);

    }
```

Interface como Tipo de Dados

☐ **Polimorfismo:**

Se uma interface é um Tipo de Dados então podemos declarar uma referência como sendo desse Tipo de Dados e atribuir-lhe um objeto de uma Classe que implemente essa interface.

```
public static void main(String[] args) {
  Movimentavel[] movimentaveis = new Movimentavel[4];
  Ponto posicao;

  movimentaveis[0] = new Circulo(1, 1, 2, cor);
  movimentaveis[1] = new Circulo(10, 10, 4, cor);
  movimentaveis[2] = new Quadrado(10, 10, 15, cor);
  movimentaveis[3] = new Quadrado(20, 20, 25, cor);

  for (int i=0; i < movimentaveis.length; i++) {
      movimentaveis[i].deslocar(3,0);
 }
  System.out.println();

  for (int i=0; i < movimentaveis.length; i++) {
     posicao = movimentaveis[i].obterPosicao();
     System.out.println( "X=" + posicao.getX() + " Y=" +
                          posicao.getY());
    }
}
```

Interface como Tipo de Dados

- A partir de um objeto declarado como referencia de uma interface, **só temos acesso aos métodos declarados na interface.**

  - **EX:**

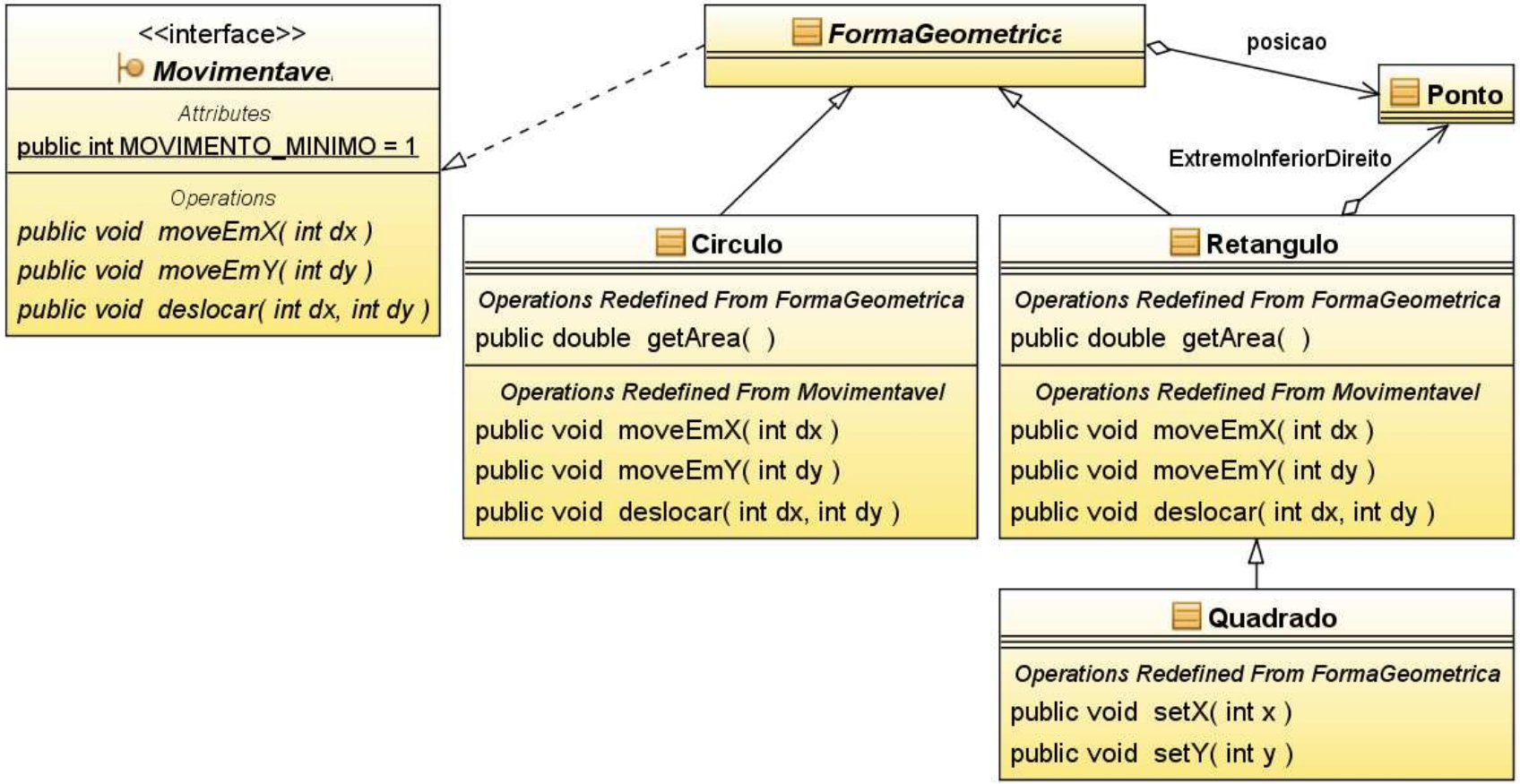
```
package formas;
public class Main {
   public static void main(String[] args) {

      Movimentavel movimentavel = new Retangulo();
      movimentavel.moverEmX(4); // OK! Está declarado na interface
      movimentavel.getX();  // Impossível! Não está declarado na interface
      movimentavel.getX2(); // Impossível! Não está declarado na interface

   }
```

Interfaces - implementações

- **FormaGeometrica implementa a interface Movimentavel**
  - A implementação de Movimentavel não está em FormaGeometrica porque é abstracta. Todas as suas subclasses a implementam



Interfaces – Herança Simples

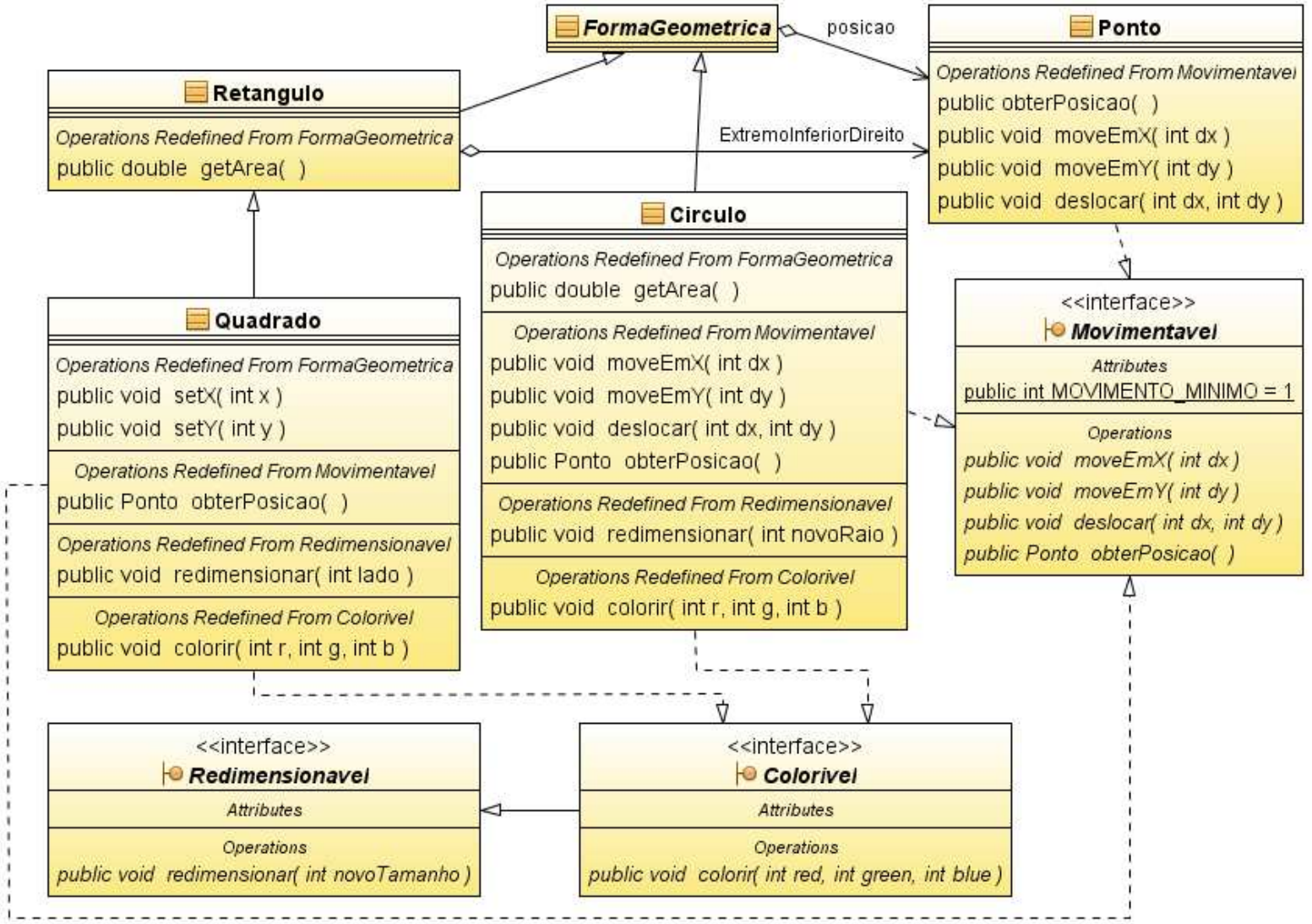
EX:
Vamos interpretar este diagrama em que:

Ponto implementa a interface **Movimentavel.**

Quadrado e Circulo implementam a Interface **Colorivel .**

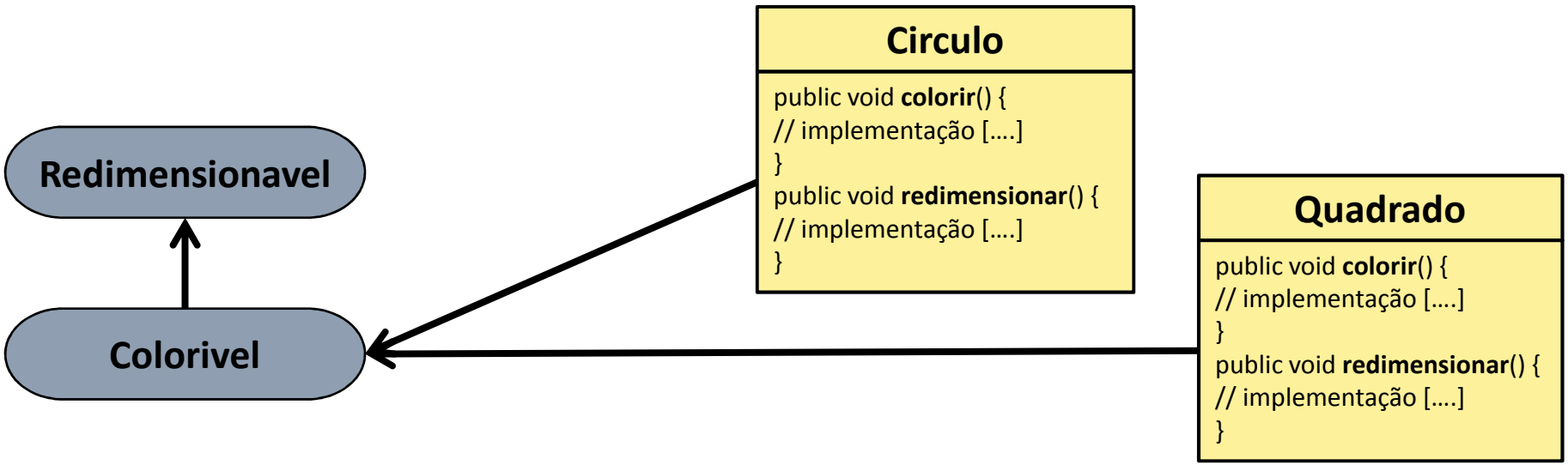
Como **Colorivel herda de Redimensionavel**

Quadrado e Circulo terão de implementar também a interface **Redimensionavel**



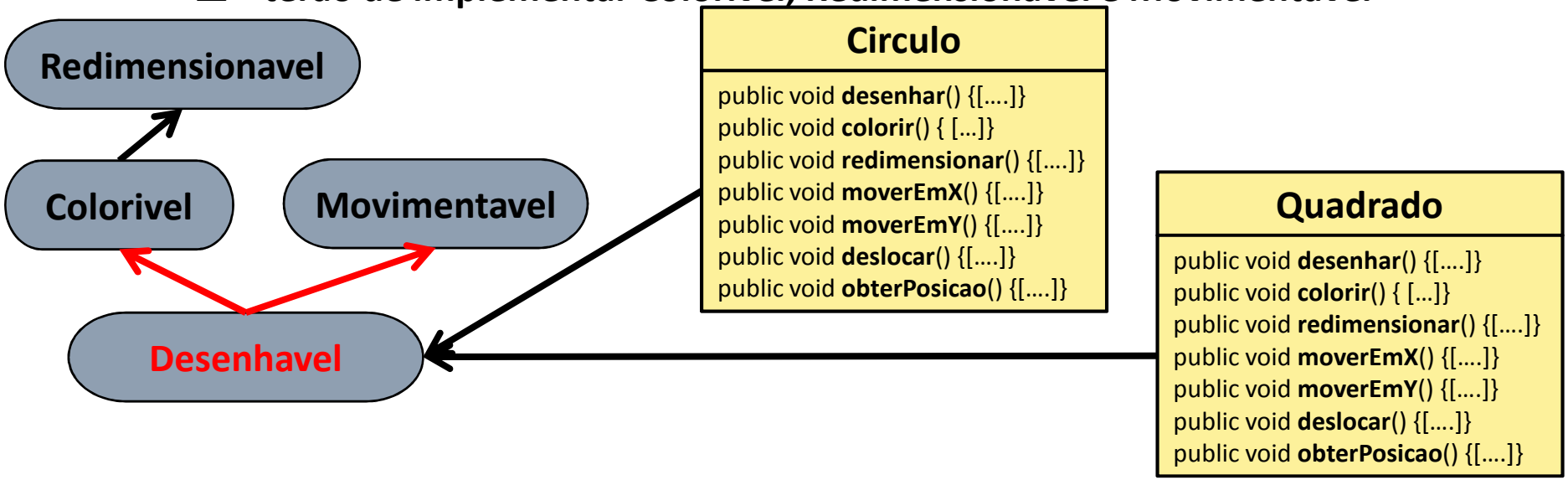
Interfaces – Herança Simples

- **Um Interface pode herdar de uma** ou mais **interfaces**
  - Uma classe que implemente uma "sub-interface" tem obrigatoriamente que implementar também a sua "super-interface".
  - EX: **Circulo e Quadrado implementam Colorivel**
    - **Como Colorivel herda de Redimensionavel**
      (`public interface Colorivel extends Redimensionavel`)
    - **Quadrado e Circulo têm de implementar também Redimensionavel**



Interfaces – Herança Múltipla

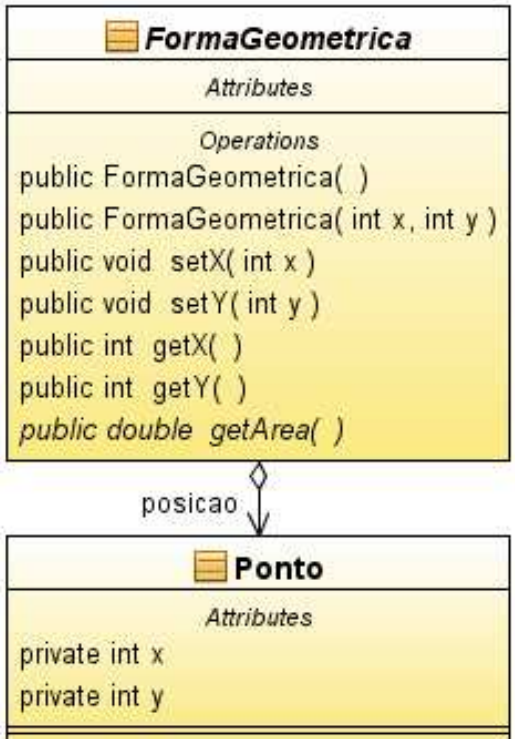
- Uma Interface pode herdar de uma ou mais interfaces
  - Se uma interface pode herdar de várias interfaces, então poderemos ter hierarquias de herança múltipla entre interfaces.
  - **Uma "sub" interface pode ter várias "super" interfaces**
  - EX: **Desenhavel herda de Colorivel e de Movimentavel**
    `(public interface Desenhavel extends Colorivel, Movimentavel)`
    - **Circulo e Quadrado implementam Desenhavel por isso**
    - **terão de implementar Colorivel, Redimensionavel e Movimentavel**



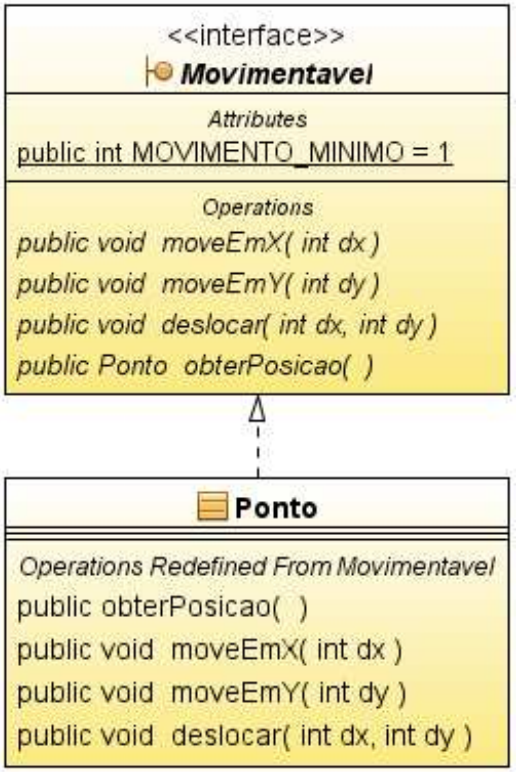
Classes Abstratas versus Interfaces

☐ **As Classes Abstratas e as Interfaces servem diferentes propósitos**

- Em Java as Classes Abstratas modelam entidades abstratas (eventualmente com estado e comportamento) das quais nunca serão instanciados objetos.

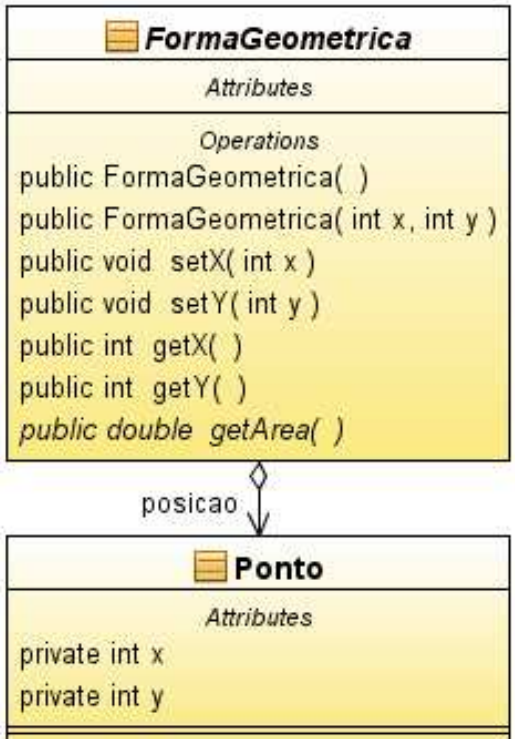


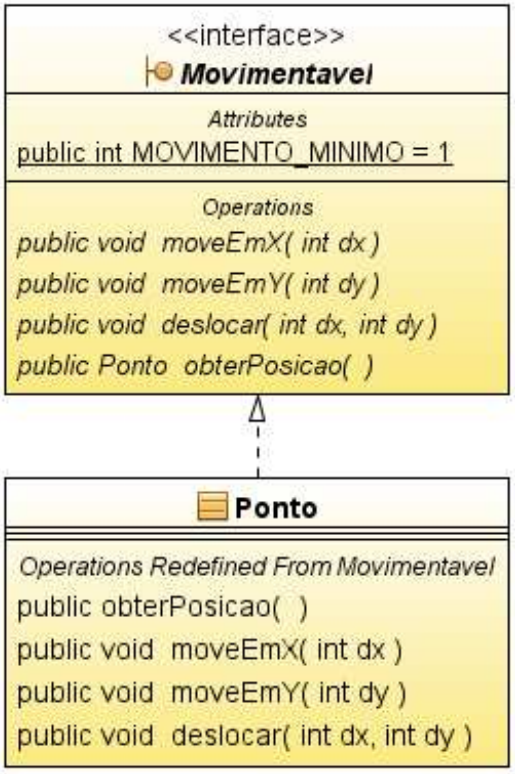
- As Interfaces servem para extrair "funcionalidades" comuns a um grupo de classes que podem ser reutilizadas por outras classes.



Classes Abstratas versus Interfaces

- **Classe Abstrata pode ou não ser 100% abstrata. Interface é sempre 100% abstrata!**
  - Uma Classe Abstrata pode ter **ZERO ou mais** métodos abstratos
    - **i.e. pode declarar zero ou mais métodos sem os implementar, delegando a sua implementação nas suas subclasses**
  - **TODOS** os métodos de uma interface são abstratos
    - **i.e. só declaram os métodos, sendo a sua implementação obrigatória nas Classes que declarem implementá-los**

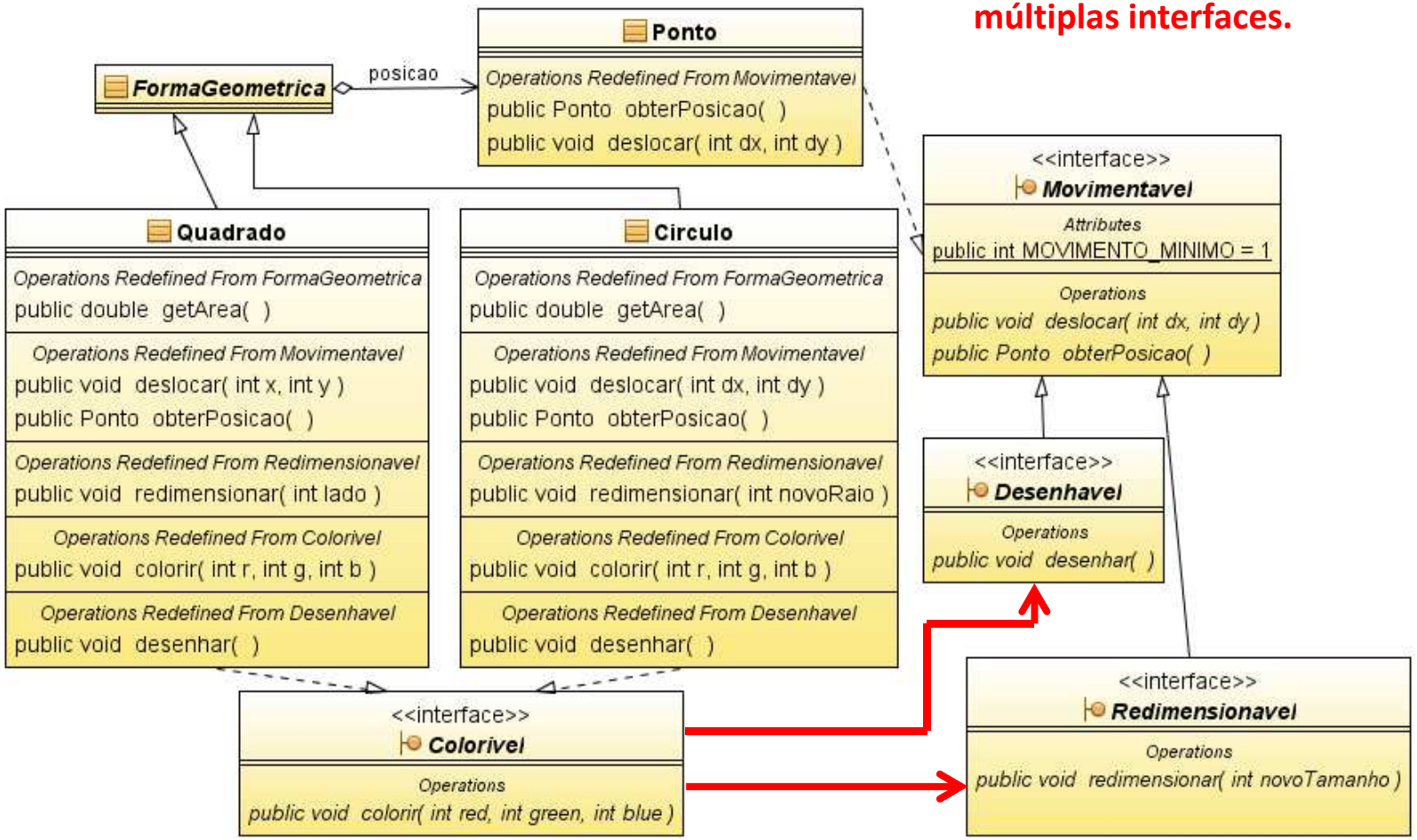




Classes Abstratas versus Interfaces

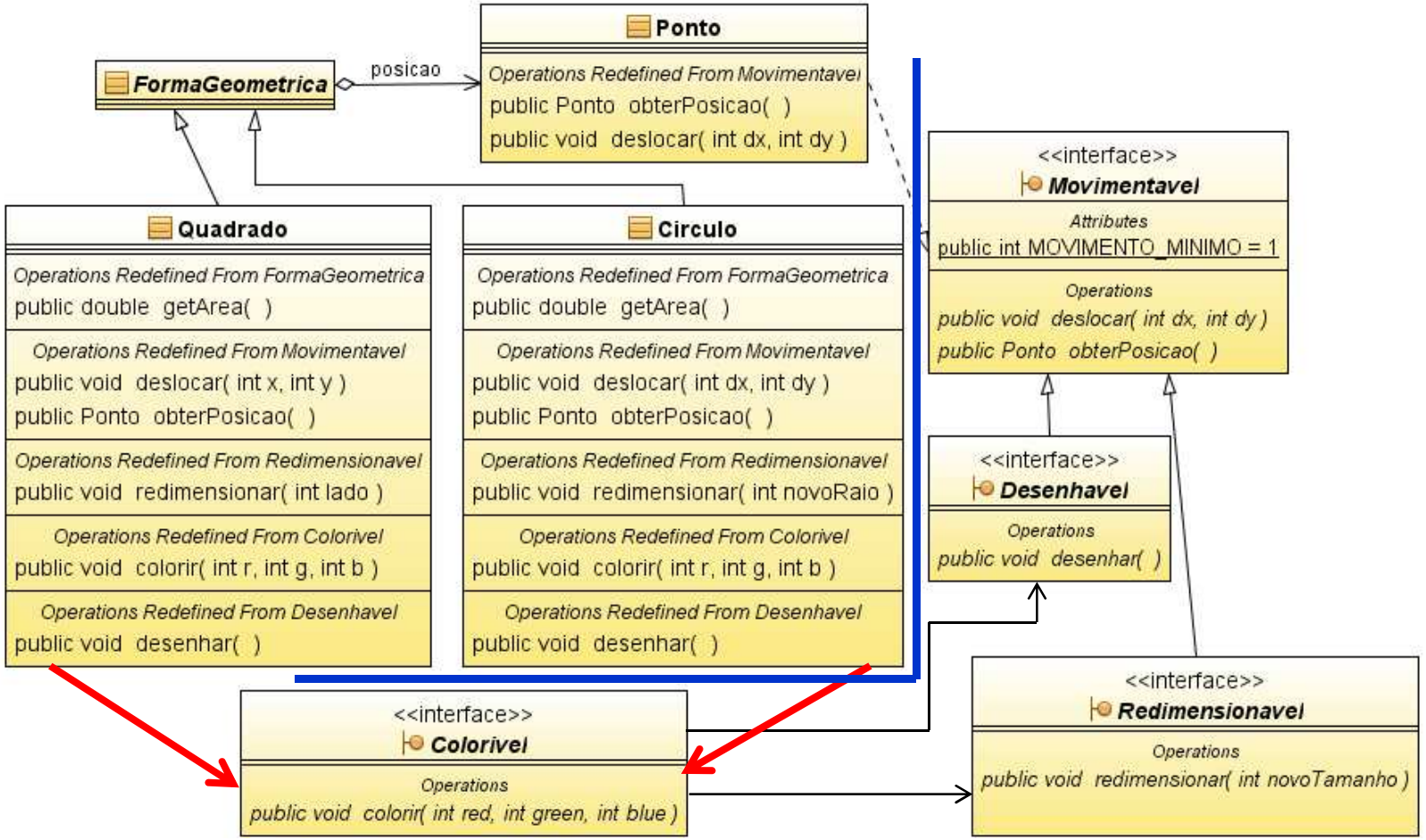
☐ **Entre Classes temos herança simples. Entre Interfaces podemos ter herança múltipla.**

- Uma subclasse herda apenas de uma superclasse
- **Uma interface pode herdar de múltiplas interfaces.**



Classes Abstratas versus Interfaces

- **A hierarquia das interfaces é independente da hierarquia das classes.**
  - Classes que implementam a mesma interface podem, ou não, estar relacionadas por herança de classes.



Resumindo

Interface:

- Conjunto de declarações de métodos que representam um comportamento particular e que qualquer classe pode implementar.
- Serve para especificar um comportamento que terá de ser partilhado por todas as classes que declarem implementá-la.

Interface como Tipo de Dados:

- podemos declarar uma referência como sendo de uma dada Interface e atribuir-lhe um objeto de qualquer das classes que a implementem.

A herança Múltipla entre Interfaces

- Uma Interface pode herdar de várias interfaces.

Interfaces vs Classes Abstratas

- Classes Abstratas e Interfaces coexistem em JAVA em hierarquias com propriedades distintas.
- O poder expressivo e de definição de tipos de dados é assim alargado pela coexistência e simbiose das duas.

Leitura Complementar

- Capítulo 7
  - Páginas 227 a 251