Programação Orientada por Objectos

Herança (continuação)

Prof. Rui César das Neves, Prof. José Cordeiro

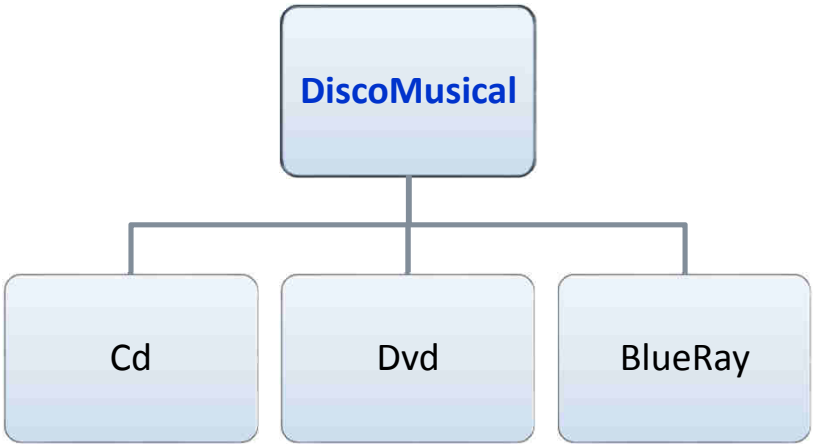
Departamento de Sistemas e Informática
Escola Superior de Tecnologia de Setúbal – Instituto Politécnico de Setúbal
2014/2015

Sumário

- **Herança (is-a)**
  - Mecanismo de Herança (continuação)
    - Hierarquia de Classes
      - Conceitos de superclasse/pai/base e subclasse/filha/derivada
      - Palavra reservada “super” e sua utilização
  - Exemplo de utilização
  - Redefinição de Métodos
  - Generalização versus Especialização
    - Bottom-up versus Top-down
  - A classe Object
    - O método toString() da classe Object

Herança – Hierarquia de Classes

- Ao estabelecermos uma relação de herança entre duas classes estamos também a estabelecer uma hierarquia de classes em que:
  - a classe que herda torna-se uma subclasse direta da classe da qual herda.
  - a classe da qual outra(s) herda(m) diretamente torna-se uma superclasse direta das restantes



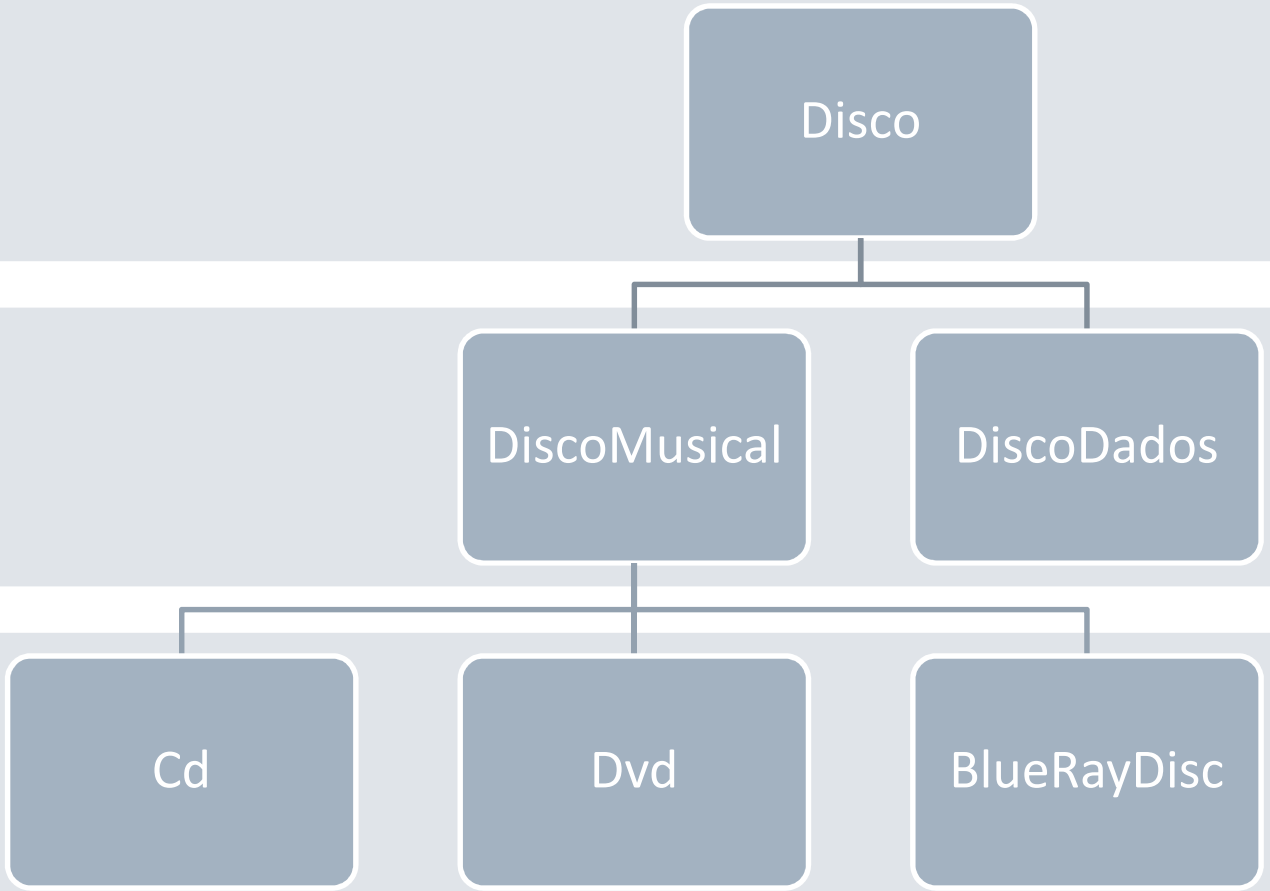
- **DiscoMusical é superclasse direta** de Cd, Dvd e BlueRay.
- **Cd é subclasse direta** de DiscoMusical
- **Dvd é subclasse direta** de DiscoMusical
- **BlueRay é subclasse direta** de DiscoMusical

Herança – Hierarquia de Classes - vocabulário

Disco é **super classe direta (classe base ou classe pai)** de DiscoMusical e DiscoDados, e é super classe indireta de Cd,Dvd e BlueRay

DiscoMusical é **subclasse direta (classe derivada ou classe filha)** de Disco e super classe direta (base ou pai) de Cd, Dvd e BlueRay

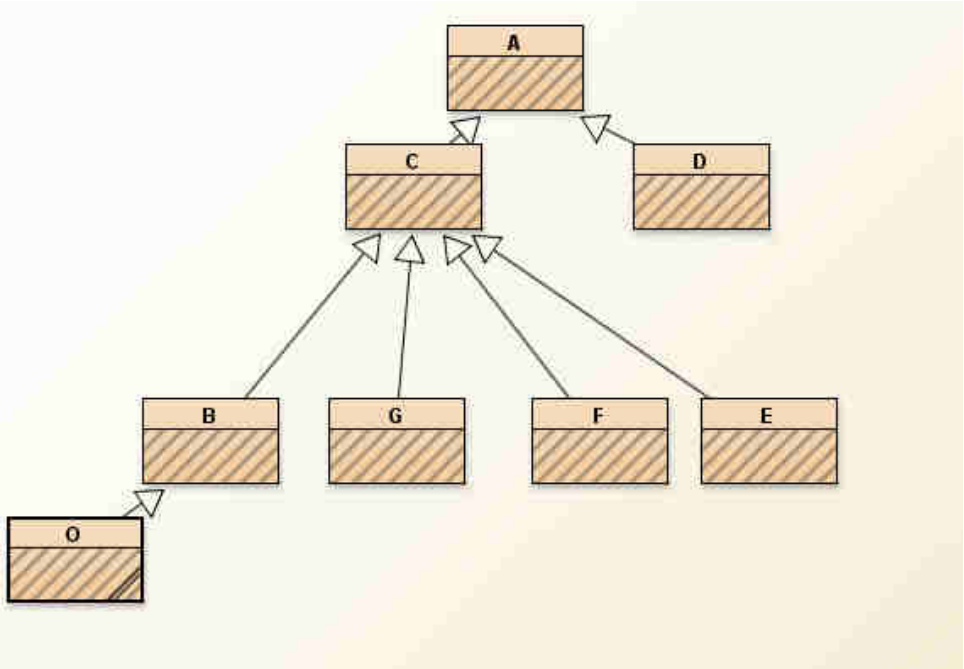
Cd é **subclasse direta (classe derivada ou classe filha)** de DiscoMusical e indirecta de Disco



Herança – Hierarquia de Classes

- Em Java uma classe só pode ter uma única superclasse directa.

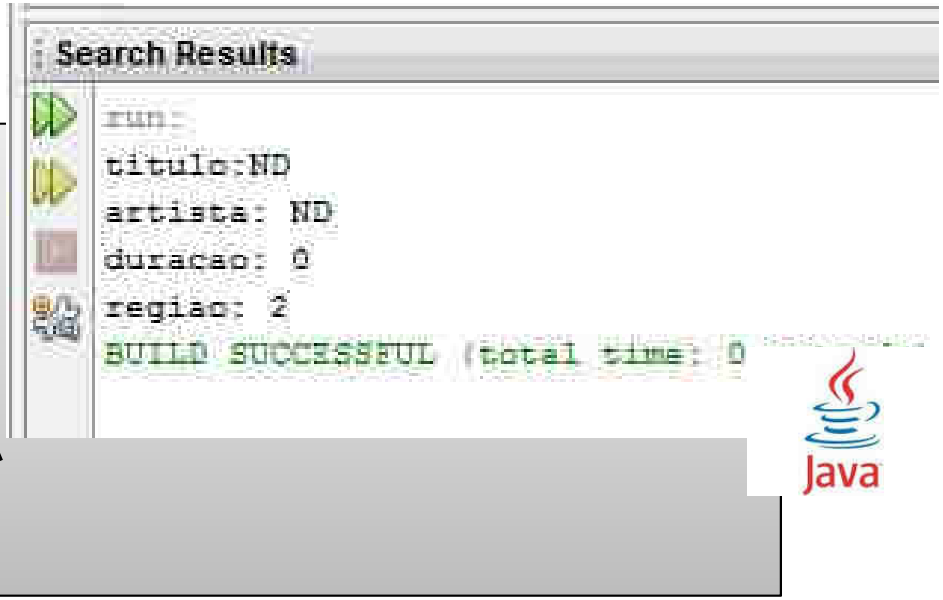
- Em Java uma classe pode ter teoricamente, um número indeterminado de subclasses.



Herança (is-a) – construtores

- A subclasse não herda os contrutores da superclasse.
- No entanto, por omissão, o construtor sem argumentos da superclasse é sempre invocado.
- Ex:

```
public Dvd (int regiao) {
        this.regiao = regiao;
}

No main:

Dvd disco = new Dvd(2);
```

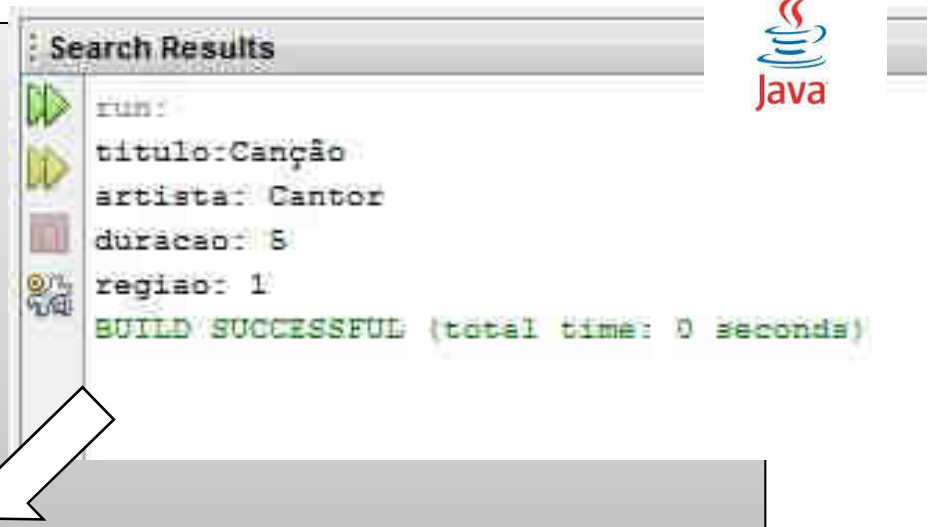


- No entanto, é possivel invocar explicitamente um construtor especifico da **super** classe usando a palavra reservada … **super**

Herança (is-a) – construtores

- **super** – palavra reservada do java usada, entre outras coisas, para invocar construtores da superclasse.
  - O construtor de uma subclasse pode seleccionar **"o"** construtor da super classe que pretende usar recorrendo a super(com a lista de argumentos desse construtor)
  - A chamada a super() tem de ser a primeira linha do construtor
  - EX:

```
public Dvd (String titulo,
                String artista,
                int duracao,
                int regiao) {
        super(artista, titulo, duracao);
        this.regiao = regiao;
}

No main:

Dvd disco = new Dvd ("Canção", "Cantor", 5, 1);
```

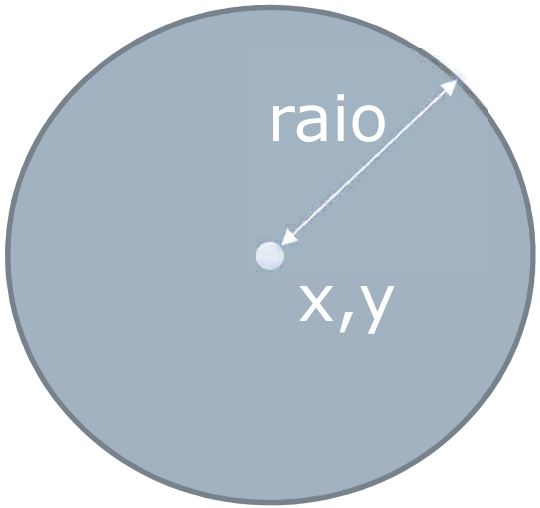


Herança (is-a) – super

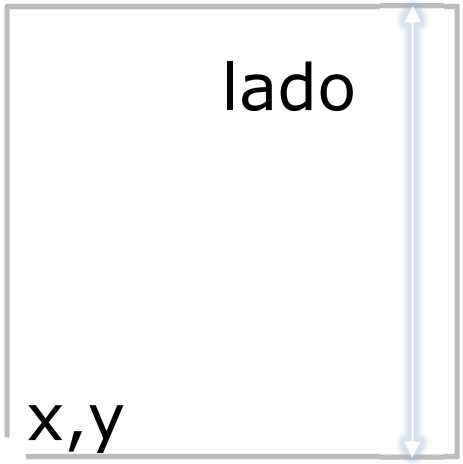
```
public class DiscoMusical {

  private String titulo;
  private String artista;
  private int duracao;

  public DiscoMusical() {
    titulo = artista = "ND";
    duracao = 0;
  }

  public DiscoMusical(String titulo,
      String artista, int duracao) {
    this.titulo = titulo;
    this.artista = artista;
    this.duracao = duracao;
  }

  //inspectores e modificadores…
  //[…]
}
```

```
public class Dvd extends DiscoMusical {

  private int regiao;

  public Dvd (String titulo,
              String artista,
              int duracao,
              int regiao) {
    super(titulo, artista, duracao);
    this.regiao = regiao;
  }

  public int getRegiao() {
    return regiao;
  }

  public void setRegiao(int regiao) {
    this.regiao = regiao;
  }
}
```

Herança - exemplo

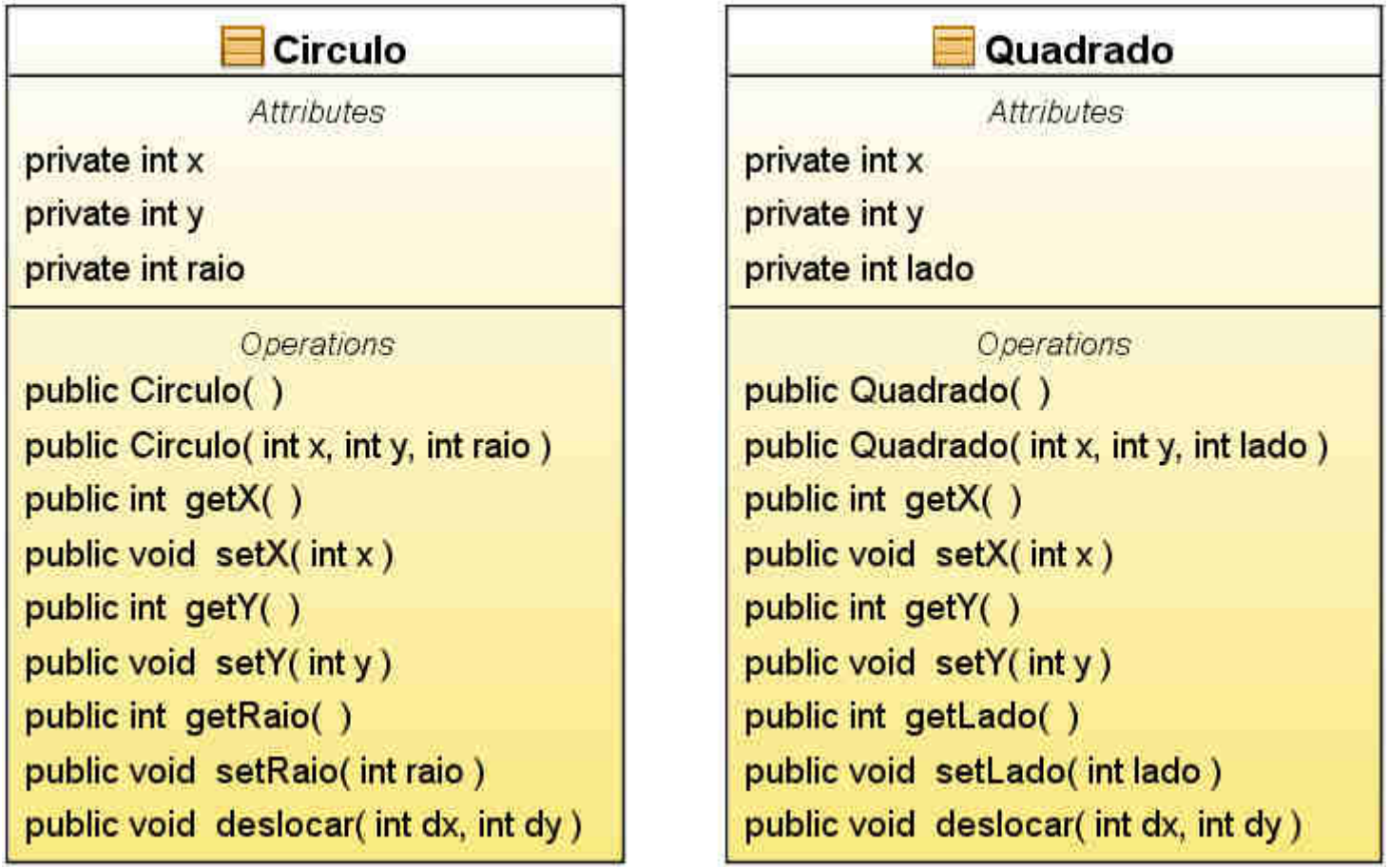


Círculo



Quadrado

Herança - exemplo



Herança - exemplo

```
public class Circulo {
    private int x, y;
    private int raio;

    public Circulo() {
        this( 0, 0, 1);
    }

    public Circulo(int x, int y, int raio) {
        this.x = x;
        this.y = y;
        this.raio = raio;
    }

    public int getRaio() {
        return raio;
    }

    public void setRaio(int raio) {
        this.raio = raio;
    }
```

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

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }

    public void deslocar( int dx, int dy ) {
        x += dx; y += dy;
    }
}
```

Herança - exemplo

```
public class Quadrado {
    private int x, y;
    private int lado;

    public Quadrado() {
        this( 0, 0, 1);
    }

    public Quadrado(int x, int y, int lado) {
        this.x = x;
        this.y = y;
        this.lado = lado;
    }

    public int getLado() {
        return lado;
    }

    public void setLado(int lado) {
        this.lado = lado;
    }
```

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

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }

    public void deslocar( int dx, int dy ) {
        x += dx; y += dy;
    }
}
```

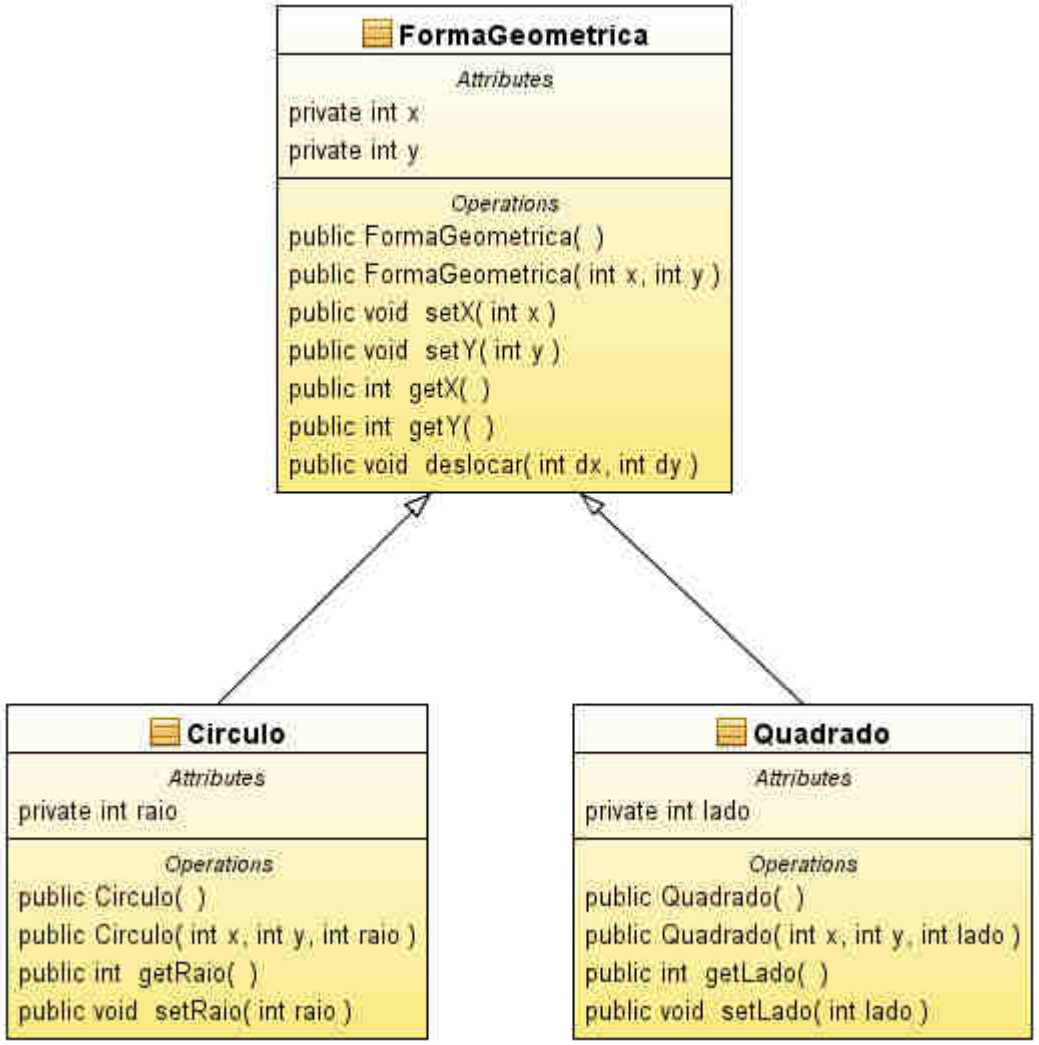
Herança - exemplo

```
public class Programa {

  public static void main(String[] args) {
    Circulo circulo = new Circulo(1, 1, 23);
    Quadrado quadrado = new Quadrado(0, 0, 4);

    System.out.println("Circulo: - Posição (" + circulo.getX() +
                       "," + circulo.getY() +
                       ") - Raio: " + circulo.getRaio() );

    System.out.println("Quadrado: - Posição (" + quadrado.getX() +
                       "," + quadrado.getY() +
                       ") - Lado: " + quadrado.getLado() );

    quadrado.deslocar( 2, 2);

    System.out.println("Quadrado: - Posição (" + quadrado.getX() +
                       "," + quadrado.getY() +
                       ") - Lado: " + quadrado.getLado() );

  }
}
```

Herança - exemplo

- As classes Circulo e Quadrado partilham alguns dos atributos e métodos:
  - 2 atributos (x e y)
  - 5 geters & seters
- É muito estado e muito comportamento repetido …
- vamos Introduzir a classe **FormaGeometrica** como **generalização** de Circulo e Quadrado

Herança - exemplo



Herança - exemplo

```
public class FormaGeometrica {
    private int x, y;

    public FormaGeometrica () {
        this( 0, 0);
    }

    public FormaGeometrica (int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }

    public void deslocar( int dx, int dy ) {
        x += dx; y += dy;
    }
}
```

Herança - exemplo

```
public class Circulo extends FormaGeometrica {
   private int raio;

   public Circulo() {
      this( 0, 0, 1);
   }

   public Circulo(int x, int y, int raio) {
      super(x, y);
      this.raio = raio;
   }

   public int getRaio() {
      return raio;
   }

   public void setRaio(int raio) {
      this.raio = raio;
   }
}
```

- Circulo fica com apenas um atributo – raio - declarado no seu corpo, assim como com os respetivos geters & seters.
- Os atributos x e y são herdados da classe FormaGeométrica juntamente com todos os métodos dessa classe.
- Apesar de Circulo possuir os atributos x e y, não consegue aceder-lhes pois foram declarados como private em FormaGeométrica.

Herança - exemplo

```
public class Quadrado extends FormaGeometrica {
    private int lado;

    public Quadrado() {
        this( 0, 0, 1);
    }

    public Quadrado(int x, int y, int lado) {
        super(x, y);
        this.lado = lado;
    }

    public int getLado() {
        return lado;
    }

    public void setLado(int lado) {
        this.lado = lado;
    }
}
```

- Quadrado fica com apenas um atributo – lado - declarado no seu corpo, assim como com os respetivos geters & seters.
- Os atributos x e y são herdados da classe FormaGeométrica juntamente com todos os métodos dessa classe.
- Apesar de Quadrado possuir os atributos x e y, não consegue aceder-lhes pois foram declarados como private em FormaGeométrica.

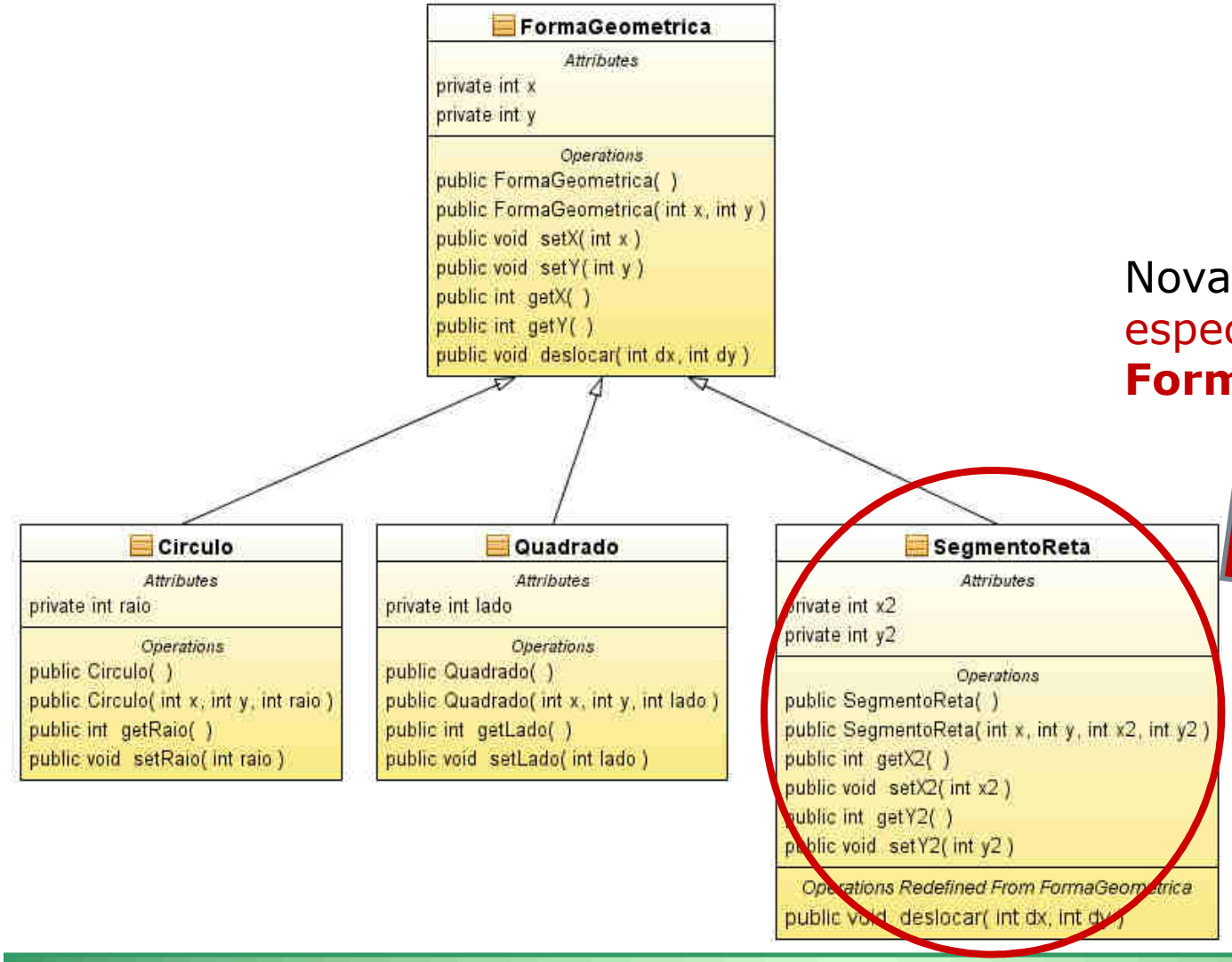
Herança - exemplo

O método main do programa não sofre quaisquer alterações!

```
public class Programa {
  public static void main(String[] args) {
    Circulo circulo = new Circulo(1, 1, 23);
    Quadrado quadrado = new Quadrado(0, 0, 4);
    System.out.println("Circulo: - Posição (" + circulo.getX() + "," + circulo.getY() +
                       ") – Raio: " + circulo.getRaio() );
    System.out.println("Quadrado: - Posição (" + quadrado.getX() + "," + quadrado.getY() +
                       ") – Lado: " + quadrado.getLado() );
    quadrado.deslocar( 2, 2);
    System.out.println("Quadrado: - Posição (" + quadrado.getX() +"," + quadrado.getY() +
                       ") – Lado: " + quadrado.getLado() );
  }
}
```

Em seguida vamos **criar, por especialização**, uma nova classe **SegmentoReta** como subclasse de FormaGeometrica

Herança - exemplo



Nova classe definida por especialização da classe **FormaGeometrica**

Herança - exemplo

```
public class SegmentoReta extends FormaGeometrica {
    private int x2, y2;

    public SegmentoReta() {
        this( 0, 0, 1, 1);
    }

    public SegmentoReta(int x, int y,
            int  x2, int y2) {
        super(x, y);
        this.x2 = x2;
        this.y2 = y2;
    }

    public int getX2() { return x2; }

    public void setX2(int x2) { this.x2 = x2; }

    public int getY2() { return y2;  }

    public void setY2(int y2) { this.y2 = y2;  }
}
```

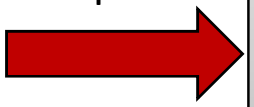
- Para definir o Segmento de reta necessitamos de mais uma coordenada (x2,y2).
- Os atributos x e y são herdados da classe FormaGeométrica juntamente com **todos os métodos** dessa classe.
- O problema é que ao herdar "**todos**" os métodos de FormaGeométrica herda também um cujo comportamento não é adequado a um segmento de reta … **qual?**

Herança – Redefinição de Métodos

- O método:

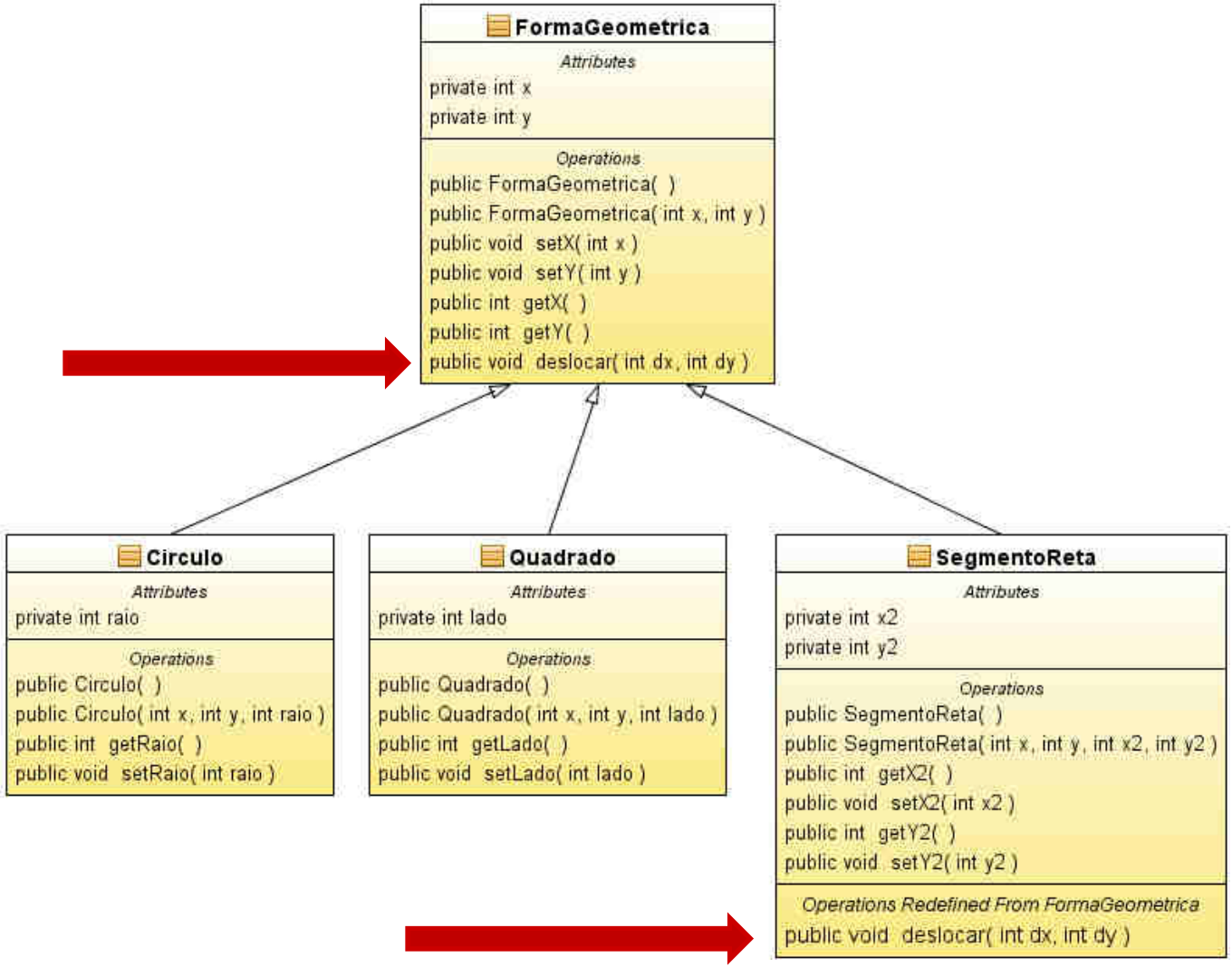
```
public void deslocar( int dx, int dy ) {
    x += dx; y += dy;
}
```

  - Da classe FormaGeometrica **não possui o comportamento desejado** para a classe SegmentoReta.
    - **Mas podemos redefinir métodos** herdados sempre que estes não satisfaçam as nossas necessidades.
  - Quando esse método for invocado, a redefinição terá prioridade sobre o método herdado.
  - Para redefinir um método **basta defini-lo novamente** no corpo da subclasse:



```
public class SegmentoRecta extends FormaGeometrica {
        //[…] código do slide 15

        /* Redefinição do Método */

        @Override
         public void deslocar( int dx, int dy ) {
                super.deslocar(dx, dy);
                x2 += dx; y2 += dy;
        }
}
```

Herança – Redefinição de Métodos



Herança (is-a) – Generalização vs Especialização

- Uma **superclasse** de outras classes é uma **generalização** dessas classes agrupando em sí:
  - a estrutura de dados (atributos) e/ou
  - o comportamento (métodos)
  - comum às suas subclasses.

- Versus …

- Uma **subclasse** de uma dada classe é uma extensão, refinamento ou **especialização** desta, acrescentado-lhe:
  - mais estrutura de dados (atributos) e/ou
  - mais comportamento (métodos)

Herança (is-a) – Generalização vs Especialização

- Mas realmente relevantes no desenho de uma aplicação OO são os **processos de generalização e especialização**

- **Generalização**
  - Quando, no processo desenho de uma aplicação, nos apercebemos que várias classes (Cd, Dvd, etc) partilham um conjunto de atributos e/ou métodos e concluímos que podemos agrupar esses atributos e métodos numa classe genérica dizemos que estamos a usar herança por generalização

- **Especialização**
  - Quando temos já implementada uma classe genérica (DiscoMusical) e pretendemos reutilizar o seu código para criar outras classes que tratarão de casos especificos (CD's, DVD's etc), dizemos que estamos a usar a herança por especialização

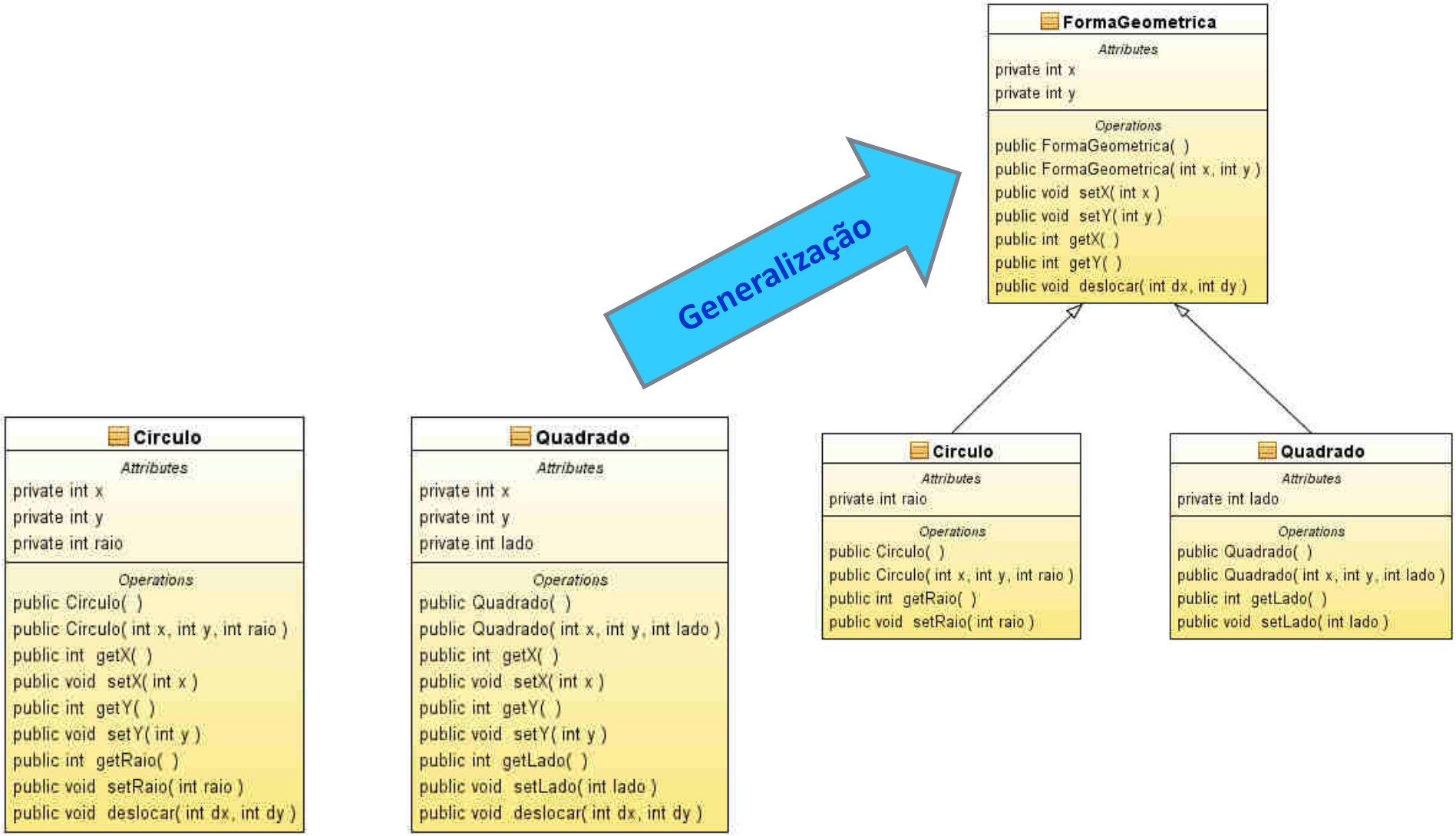
Herança (is-a) – Generalização vs Especialização

- A Herança é utilizada em dois casos distintos:

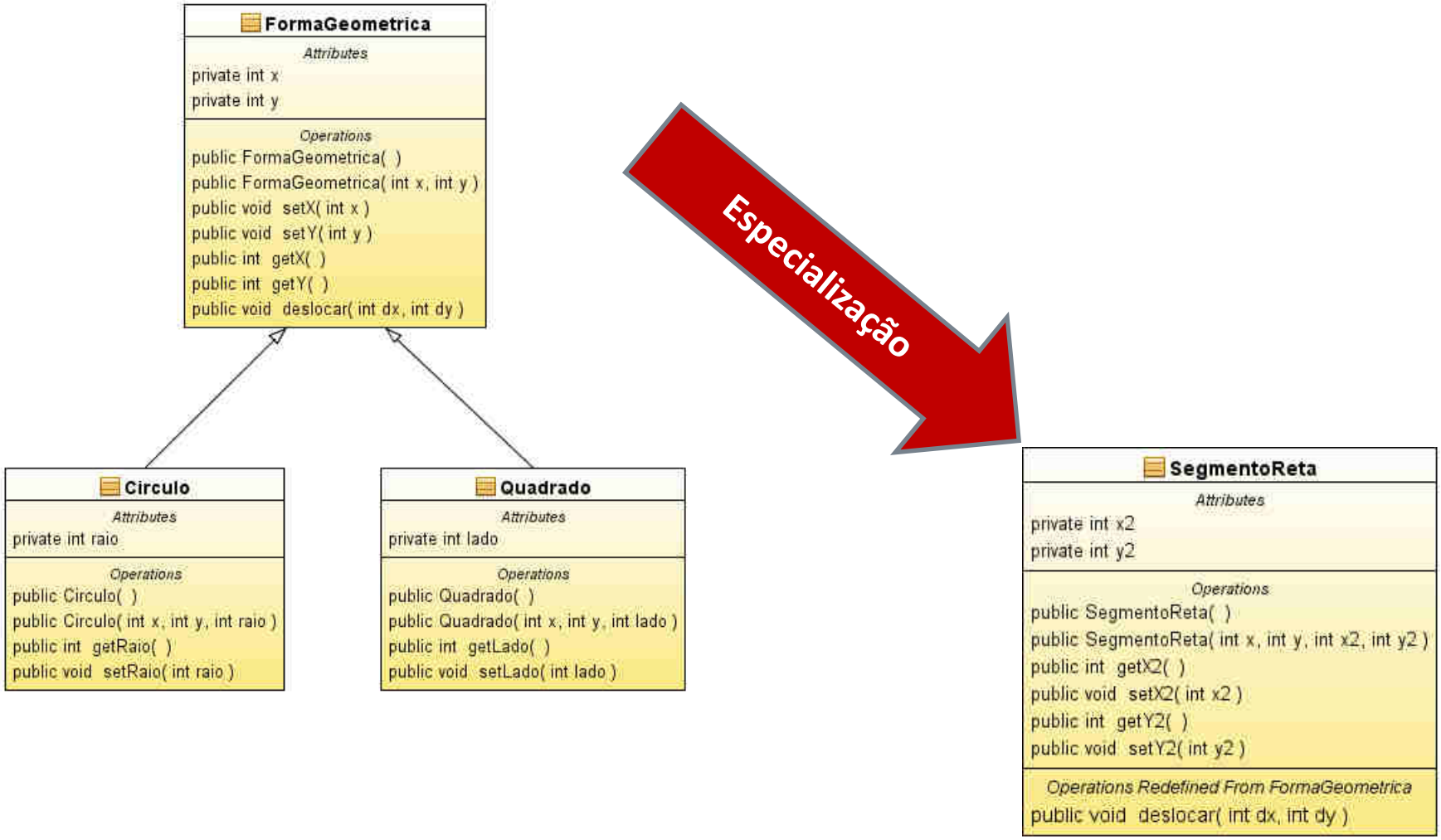
1. Quando se pretende **generalizar** um conceito o conjunto de características comuns a varias classes são agregadas numa **superClasse.**
   Exemplos : dos Cd's e Dvd's e dos Quadrados e Circulos

2. Quando se pretende **especializar** uma entidade nos seus vários subtipos. Definimos as diferentes subclasses que especificam os diferentes subtipos.
   Exemplo : do Segmento de Recta

Herança (is-a) – Generalização vs Especialização



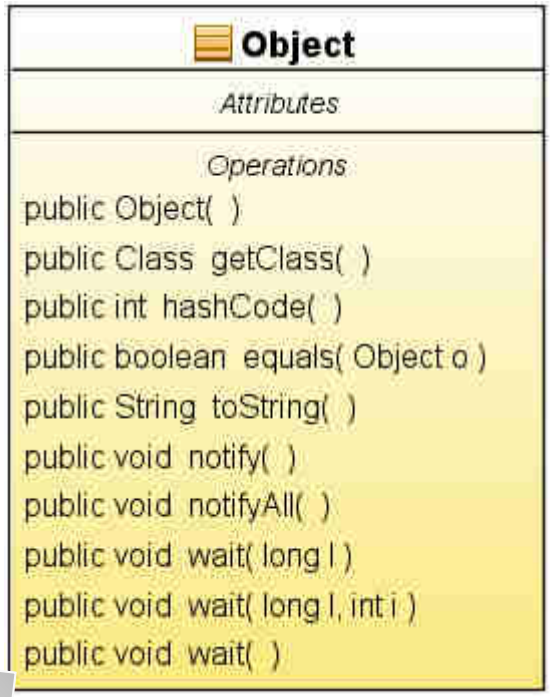
Herança (is-a) – Generalização vs Especialização



Classe Object

- **Object** é uma classe particular do Java, da qual todas as outras são subclasses.

  - Object está no topo da hierarquia de classes do Java
  - Todas as classes são uma especialização de Object.
  - Todas as classes herdam os métodos definidos na classe Object
  - Existe um método, frequentemente utilizado para devolver informação sobre uma instância:



public String toString()

Classe Object - Redefinição toString()

- Uma vez que todas as classes herdam da classe Object …
  - Todas as classes herdam o método toString()
  - No entanto o seu comportamento tem de ser especifico de cada classe uma vez que este método se destina a representar numa string uma objeto dessa classe
  - Assim sendo o método toString() deve ser redefinido em cada classe …
  - EX:

```
public class FormaGeometrica {

    // […] todo o código do slide 15

    @Override
    public String toString() {
        return String.format("Posição(%d,%d)", x, y);
    }
}
```

Classe Object - Redefinição toString()

```
public class Circulo extends FormaGeometrica {

    //[…] Todo o código do slide 11

    @Override
    public String toString() {
        return "Centro: " + super.toString()
                + " Raio: " + raio;
    }
}
```

```
public class Quadrado extends FormaGeometrica {

    //[…]

    @Override
    public String toString() {
        //experimente redefinir este método
        // para a classe Quadrado
    }
}
```

Classe Object - Redefinição toString()

```
public class Programa {

  public static void main(String[] args)
  {
    Circulo circulo1 = new Circulo(1, 1, 23);
    FormaGeometrica forma1 = new FormaGeometrica(2, 5);
    Quadrado quadrado1 = new Quadrado(0, 0, 4);

    System.out.println("Circulo: " + circulo1);

    System.out.println("Forma: " + forma1);

    quadrado1.deslocar( 2, 2);

    System.out.println("Quadrado: " + quadrado1);

  }
}
```

Métodos toString() são invocados automaticamente.

Qual o output?

Classe Object - Redefinição toString()

- Como em Quadrado o método toString() não foi redefinido, mantém o comportamento presente em FormaGeométrica

- Se o método toString() não tivesse sido redefinido em FormaGeometrica, então no caso do Quadrado iria manter o comportamento definido em Object.

- **Qual será…?**

Sumário

- **Herança (is-a)**
  - Mecanismo de Herança
    - Hierarquia de Classes
      - Conceitos de superclasse/pai/base e subclasse/filha/derivada
      - Palavra reservada “super” e sua utilização
  - Exemplo de utilização
  - Redefinição de Métodos
  - Generalização versus Especialização
    - Bottom-up versus Up-Down
  - A classe Object
    - O método toString() da classe Object

Leitura Complementar

- Hierarquia de Classes
  - Capítulo 5
    - pgs 159 a 187