Programação Orientada por Objectos

Introdução à Herança (is-a)

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

- **Strings**
  - Criação, principais métodos e utilização
- **Arrays**
  - Criação e utilização
- **Herança (is-a)**
  - Motivação
    - Classes que são “generalizadas” porque partilham parte do estado e/ou parte do comportamento.
    - Classes que reutilizam uma classe já existente porque a especializam.
  - Mecanismo de Herança
    - Conceito de Herança (relação “is-a”)
      - Para que serve e o que é
        - Palavra reservada “extends”
        - Acesso a atributos privados da superclasse (private) através dos seters & geters

Strings

String: sequência de caracteres

- Mais precisamente sequência de variáveis do tipo char
  - Em Java são objetos e como tal tipos referenciados … mas …
  - … Sendo a sua utilização tão comum, o seu carater de objeto está mascarado no que respeita:
    - à sua instanciação (dispensa o operador new)
    - às operações de atribuição comportam-se como se de tipos por valor se tratassem:
    - EX:

```
String nome = “Bruno Silva”;
String outroNome = nome;
nome = “Luis Alves”
System.out.println(“nome: “ + nome + “outroNome: “ + outroNome);
```

  - nome e outroNome são instâncias (objetos) da classe String

Strings

String: sequência de caracteres

- Instâncias da classe String têm acesso a métodos úteis de manipulação de texto:

  char charAt(int índice)

  int compareTo(String outraString)

  String substring(int índiceInicial, int índiceFinal)

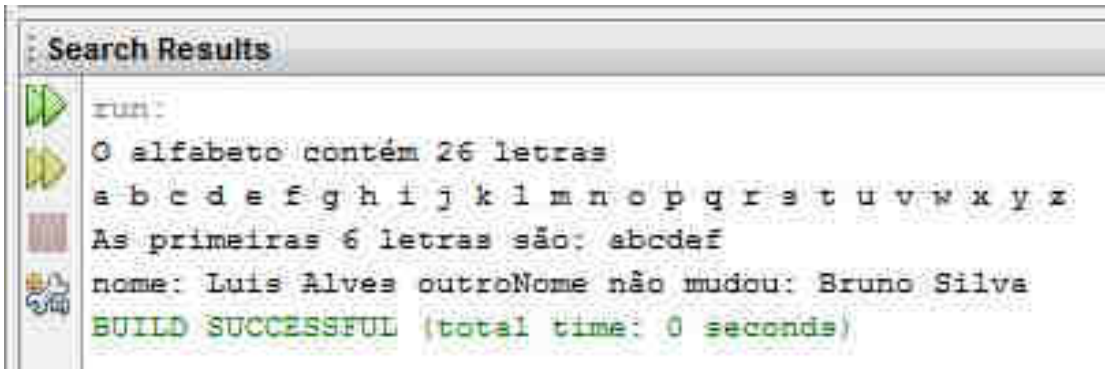
  String concat(String outraString)

  int length()

Strings

Exemplo: uso de métodos da classe String

```
public static void main(String[] args) {
    String alfabeto = "abcdefghijklmnopqrstuvwxyz";
    System.out.println("O alfabeto contém " + alfabeto.length() + " letras");

    for(int i=0; i<alfabeto.length(); i++) {
        System.out.print( alfabeto.charAt(i) + " ");
    }
    System.out.println();

    System.out.println("As primeiras 6 letras são: " + alfabeto.substring(0,6));

    // Strings são objetos e como tal tipos referenciados mas …
    // comportam-se como se de tipos por valor se tratassem!
    String nome = "Bruno Silva";
    String outroNome = nome;
    nome = "Luis Alves";
    System.out.println(
        "nome: " + nome +
        " outroNome não mudou: " +
        outroNome);
}
```



Strings

- Exemplo: concatenação e compareTo
  - existe também o compareToIgnoreCase(String)

```
public static void main(String[] args) {
    String utilizador = "bsilva";
    String dominio = "@est.ips.pt";

    String email = utilizador.concat( dominio );
    String email2 = utilizador + dominio; //com o operador '+'
    System.out.println("O email é: " + email);

    if ( email2.compareTo("bsilva@est.ips.pt") == 0 )
        System.out.println("Strings iguais");
    else
        System.out.println("Strings diferentes");
}
```

Arrays

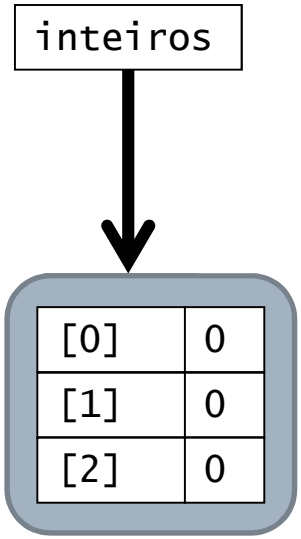
- **Array: sequência de elementos do mesmo tipo**
  - Permitem guardar e manipular sequências de tipos primitivos ou de objetos de um mesmo tipo.
    - Em Java os Arrays são objetos e como tal tipos referenciados …
      - Os elementos dos arrays de tipos primitivos são automaticamente inicializados com o valor por omissão do tipo respetivo: 0 para os tipos numéricos, carácter com o código 0 para os `char` e `false` para o `boolean`

      mas

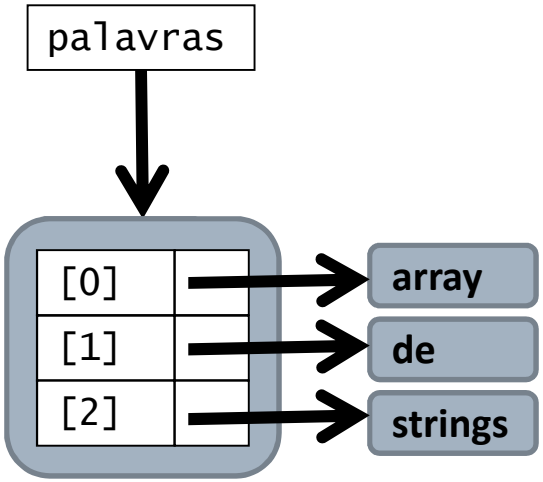
      - Os elementos dos arrays de objetos têm de ser explicitamente inicializados!

Arrays

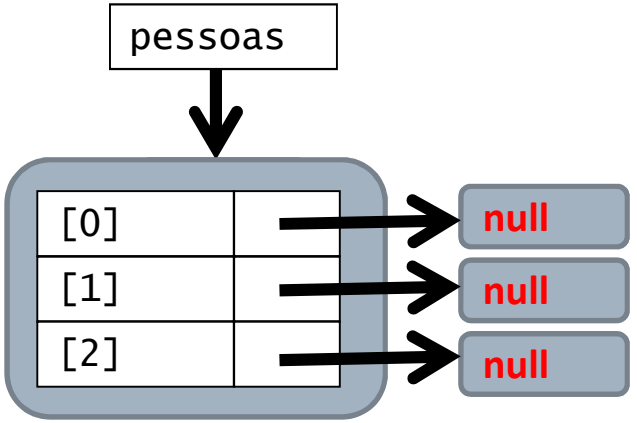
int [ ] inteiros;
inteiros = new int[3];



String [ ] palavras;
palavras = {"array", "de", "strings"};



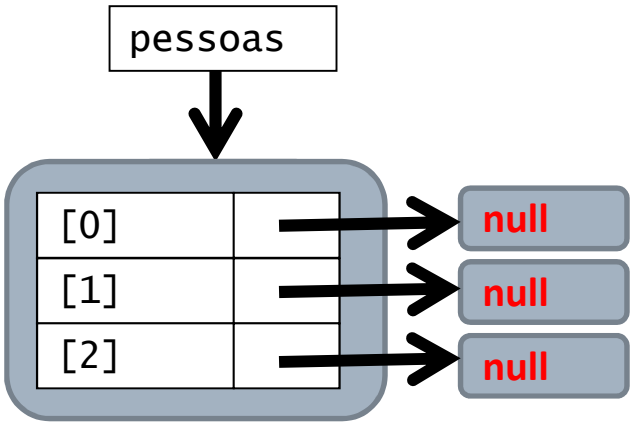
Pessoa [ ] pessoas;
pessoas = new Pessoa [3];



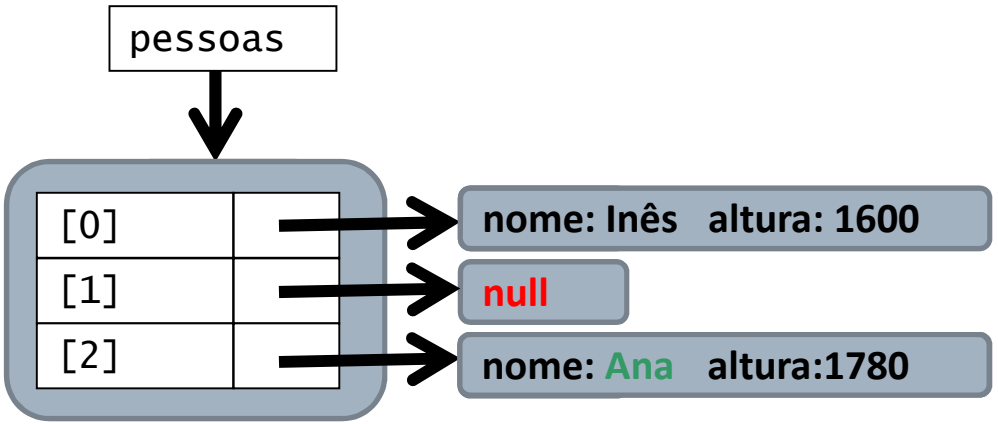
Arrays

- Em arrays de tipos referenciados é necessário garantir que não se acede a elementos não inicializados:

```
Pessoa [ ] pessoas;
pessoas = new Pessoa [3];
pessoas[0].setNome("Inês");  // ERRO!
```




```
Pessoa [ ] pessoas;
pessoas = new Pessoa [3];
pessoas [0] = new Pessoa ( "Inês", 1600);
pessoas [2] = new Pessoa ( "Jorge", 1780 );
pessoas [2].setNome ( "Ana");          // OK!
String nome = pessoas [1].getNome(); // ERRO!
```



Arrays

- Em Java, os **[ ]** podem aparecer tanto como

  - **parte do** tipo no **início** da declaração:
    - int**[ ]** vectorLinha;

  - **parte** da declaração **de uma variável particular**:
    - int x, y, z , coordenada3D**[ ]**;

  - **ou em ambos** como em:
    - int**[ ]** vectorLinha, vetorColuna, pontos2D**[ ]**;
      - Esta declaração equivale a:
      - int vectorLinha[ ], vectorColuna[ ], pontos2D[ ] [ ];

Arrays

- **Declarações de Arrays (sem criação!)**
  - int[ ] arrayDeInteiros;
  - float arrayDeFloats[ ]; //evitar: menos legível que o anterior
  - short[ ][ ] arrayBiDimensionalDeShorts;
  - String arrayBiDimensionalDeStrings[ ] [ ]; //evitar: menos legível que o anterior
  - Pessoa[ ] arrayDePessoas;

- **Declarações e instanciações (criação) de arrays:**
  - int[ ] arrayDeInteiros = new int[10];
  - int[ ] factorial = { 1, 1, 2, 6, 24, 120, 720, 5040 };
  - float[ ][ ] arrayDeFloats[ ] = new float [10][20];
  - short[ ][ ] arrayBiDimensionalDeShorts = new short [10][20];
  - Pessoa[ ] arrayDePessoas = new Pessoa[10] //Não inicializa os elementos!

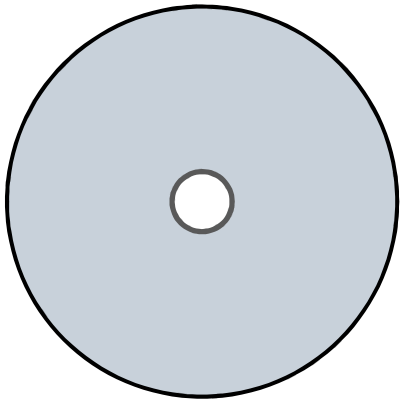
Arrays

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

    // declaração e criação de um array de objetos
    Pessoa[] pessoas = new Pessoa[3];

    // inicialização dos elementos do array
    pessoas[0] = new Pessoa("João");
    pessoas[1] = new Pessoa("Maria");
    pessoas[2] = new Pessoa("Marta");

    // iteração do array: percorrer os elementos sequencialmente
    for(int i=0; i<pessoas.length; i++) {
        System.out.println(pessoas[i].getNome());
    }

    // criação de uma nova referência para o array
    Pessoa[] referenciaParaOArrayPessoas = pessoas; //o que faz?

    System.out.println("A primeira pessoa chama-se " +
        referenciaParaOArrayPessoas[0].getNome() );
}
```

Herança (is-a) – Para quê?

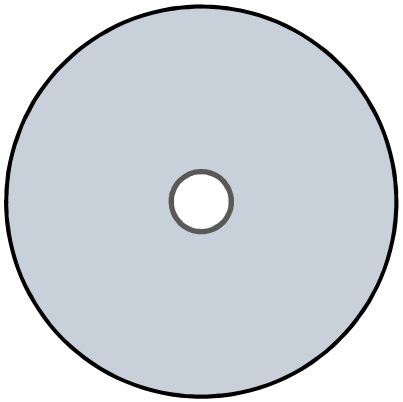
- Motivação
  - Classes que **partilham** apenas uma parte do estado e/ou do comportamento
  - i.e. que **partilham** apenas alguns dos atributos e dos métodos

CD



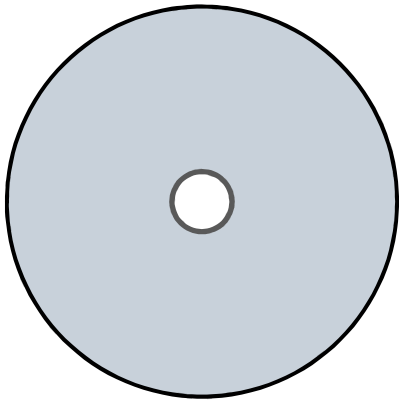
- **Título**
- **Artista**
- **Duração**
- Numero de Faixas

DVD



- **Título**
- **Artista**
- **Duração**
- Região

BlueRay



- **Título**
- **Artista**
- **Duração**
- Codec
- DRM*

*Digital Rights Managment

Herança (is-a) – Para quê?

- Com os conhecimentos que possuimos a modelação deste problema implicaria a criação de 3 classes que teriam vários atributos/métodos em comum.

Classe **Cd**

```
String titulo;
String artista;
int duracao;
int numeroDeFaixas;

// [...]
// inspetores e modificadores comuns
// [...]
void setNumeroDeFaixas(int
    numeroDeFaixas)  {
    this.numeroDeFaixas =
        numeroDeFaixas;
}
int getNumeroDeFaixas()  {
    return numeroDeFaixas;
}
```

Classe **Dvd**

```
String titulo;
String artista;
int duracao;
int regiao;

// [...]
// inspetores e modificadores comuns
// [...]
void setRegiao(String regiao)  {
    this.regiao = regiao;
}
int getRegiao()  { return regiao; }
```

Classe **BlueRay**

```
String titulo;
String artista;
int duracao;
String codec;
String drm;
// [...]
// inspetores e modificadores comuns
// [...]
void setCodec(String codec)  {
    this.codec = codec;
}
void setDrm(String drm)  {
    this.drm = drm;
}
int getCodec()  { return codec; }
String getDrm()  { return drm; }
```

- Mas foi prometido que o paradigma OO iria maximizar a "reutilização de código"!

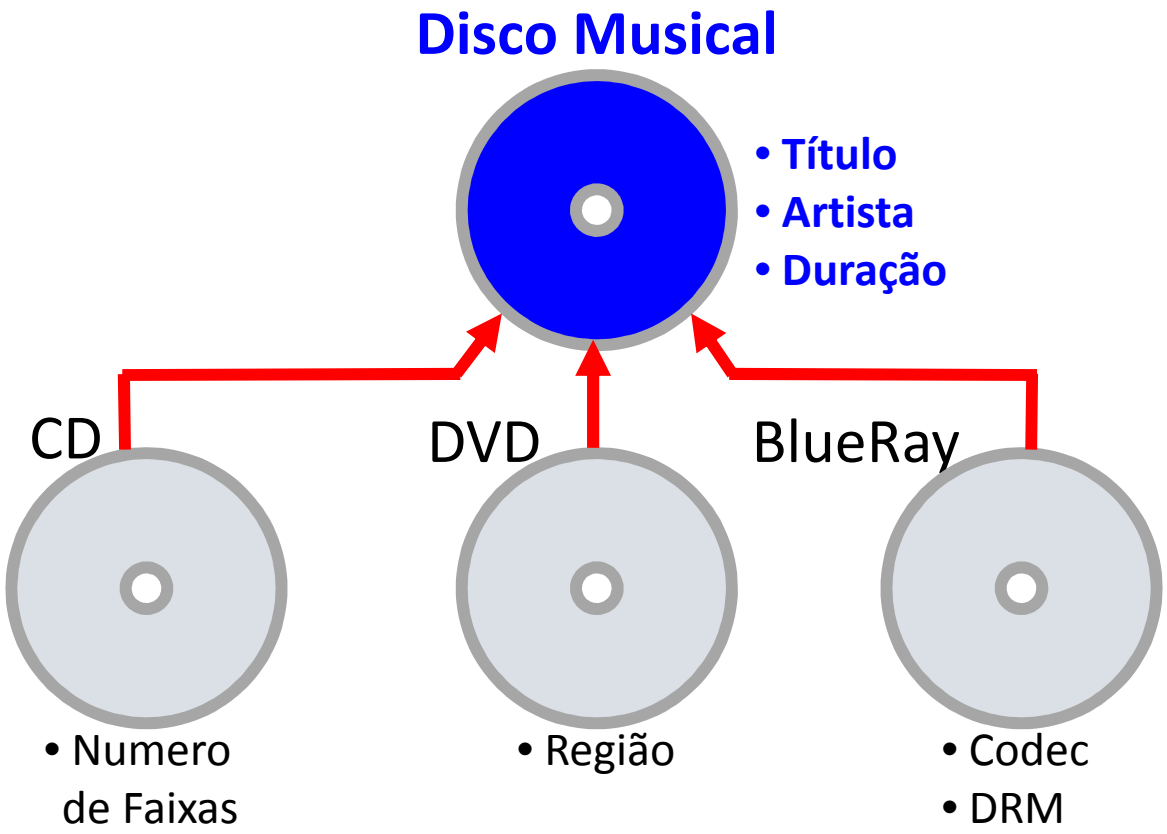
Herança (is-a) – Para quê?

- **Para reutilizar o código de forma Massiva:**
  - Colocar os atributos e métodos comuns (todo o código comum) numa só classe.
  - Deixar nas outras classes só os atributos e métodos especificos dessas classes.
  - E arranjar uma **forma destas classes "reutilizarem" o código** comum!

Herança
CD herda de Disco Musical
CD *is-a* Disco Musical

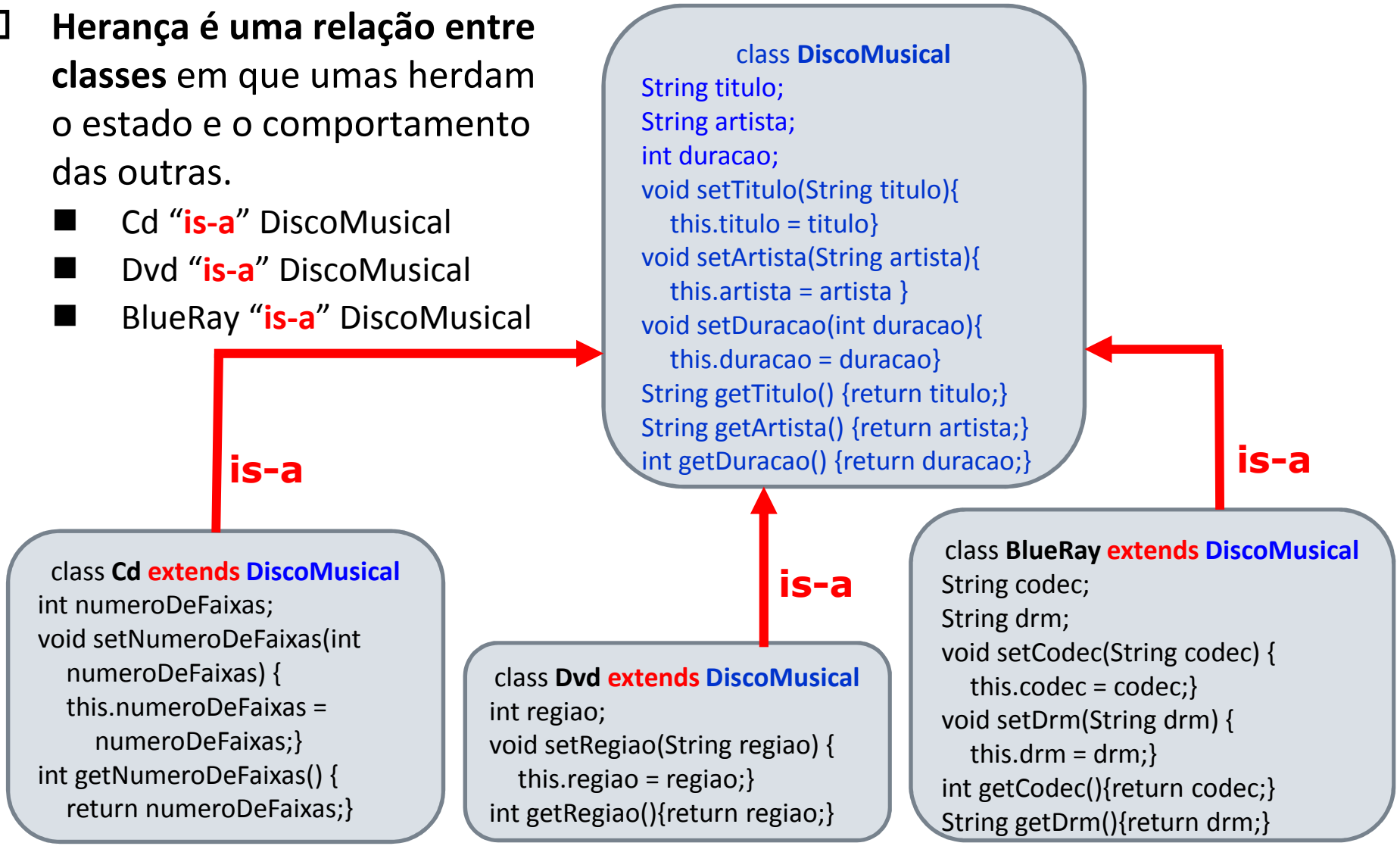
DVD herda de Disco Musical
DVD *is-a* Disco Musical

BlueRay herda de Disco Musical
BlueRay *is-a* Disco Musical



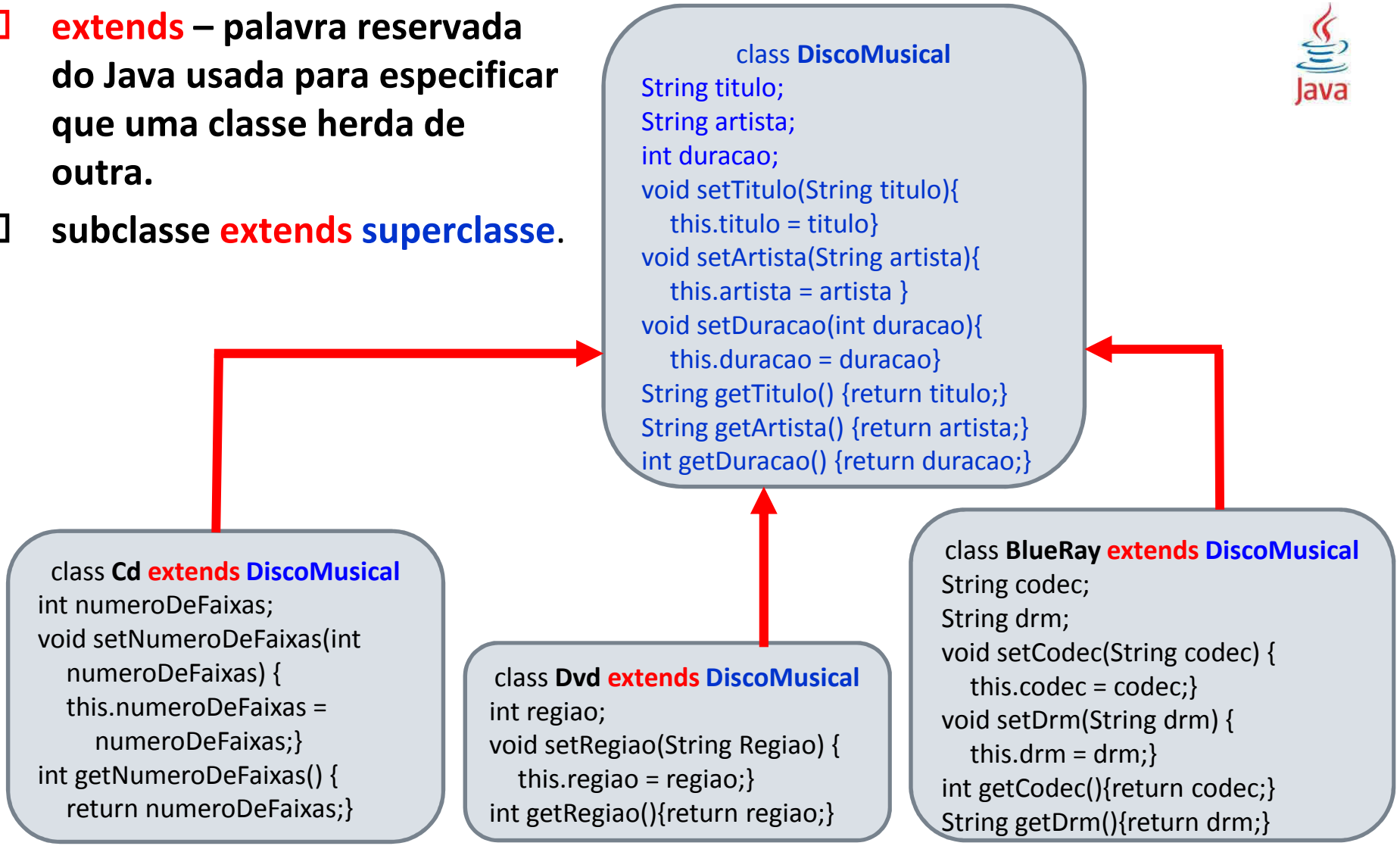
Herança (is-a) – O que é?

- **Herança é uma relação entre classes** em que umas herdam o estado e o comportamento das outras.
  - Cd "**is-a**" DiscoMusical
  - Dvd "**is-a**" DiscoMusical
  - BlueRay "**is-a**" DiscoMusical



Herança (is-a) – extends

- **extends – palavra reservada do Java usada para especificar que uma classe herda de outra.**
- **subclasse extends superclasse.**



Herança (is-a) – extends

```
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) {
    this.setTitulo(titulo);
    this.setDuracao(duracao);
    this.setArtista(artista);
    this.regiao = regiao;
  }

  public int getRegiao() {
    return regiao;
  }

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

Note-se que Dvd não acede diretamente aos atributos de Disco Musical! Porquê?

Herança (is-a) – atributos privados

- Uma classe que herde de outra não deve aceder aos atributos dessa outra classe e estes deverão mesmo ser declarados como private.

- A classe Dvd embora herde os atributos de DiscoMusical, não consegue aceder directamente a esses atributos.
  - Ex: this.titulo = "nada porque nem lá chega!"; // ERRADO!

- Solução:
  - Utilizar os modificadores (são public):
  - Ex:

```
this.setTitulo ( titulo );
this.setDuracao ( duracao );
this.setArtista ( artista );
```

Resumindo

- **Herança (is-a)**
  - Motivação
    - É um poderoso mecanismo de reutilização e simplificação de código … mas não só.
  - Conceito de Herança (relação “is-a”)
    - Uma classe herda atributos e métodos de outra classe
      - Ao fazê-lo dizemos que essa classe é (também) a sua superclasse (um Dvd é também um DiscoMusical)
      - O acesso aos atributos da superclasse faz-se através dos inspetores e modificadores

Leitura Complementar

- Capítulo 5
  - Páginas 159 a 209