

Linguagem de Programação II

IMD0040

Aula 08 – Herança

Herança

- ❑ **Herança** é um mecanismo que, permite que uma classe possa **herdar** o **comportamento** e **características** de outra classe;
- ❑ E, ao mesmo tempo em que **novos comportamentos** e **características** podem ser estabelecidas.
 - ❖ A relação de herança é dada entre classes;
 - ❖ Podemos chamar de **superclasse** e outra de **subclasse**.
- ❑ A vantagem da herança é **agrupar coisas comuns** para poder **reaproveitar o código**.

Introdução

- ❑ O que acontece quando não utilizamos **herança**?

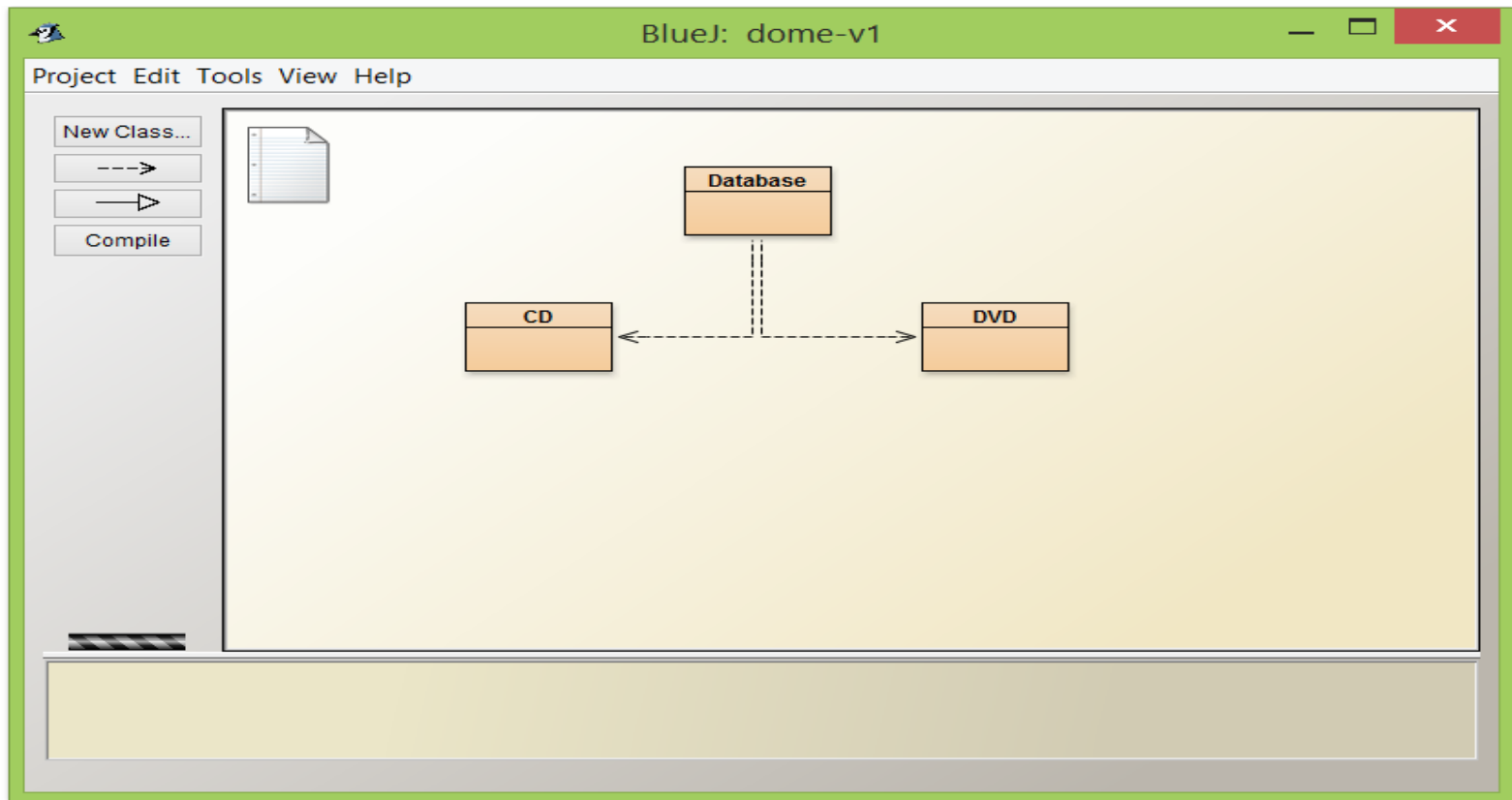
Introdução

- ❑ O que acontece quando não utilizamos **herança**?

Duplicação de código!!!!

Introdução

Projeto dome-v1:



Introdução

□ A partir do projeto dome-v1:

```
public class Database
{
    private ArrayList<CD> cds;
    private ArrayList<DVD> dvds;

    /**
     * Construct an empty Database.
     */
    public Database()
    {
        cds = new ArrayList<CD>();
        dvds = new ArrayList<DVD>();
    }
}
```

```
public void addCD(CD theCD)
{
    cds.add(theCD);
}

/**
 * Add a DVD to the database.
 * @param theDVD The DVD to be added.
 */
public void addDVD(DVD theDVD)
{
    dvds.add(theDVD);
}
```

Introdução

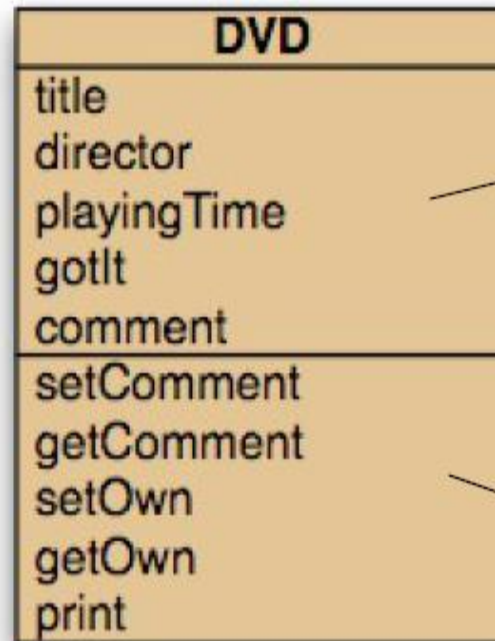
- ❑ A partir do projeto dome-v1:

```
public void list()
{
    // print list of CDs
    for(CD cd : cds) {
        cd.print();
        System.out.println();    // empty line between items
    }

    // print list of DVDs
    for(DVD dvd : dvds) {
        dvd.print();
        System.out.println();    // empty line between items
    }
}
```

Introdução

□ O que podemos **perceber**???

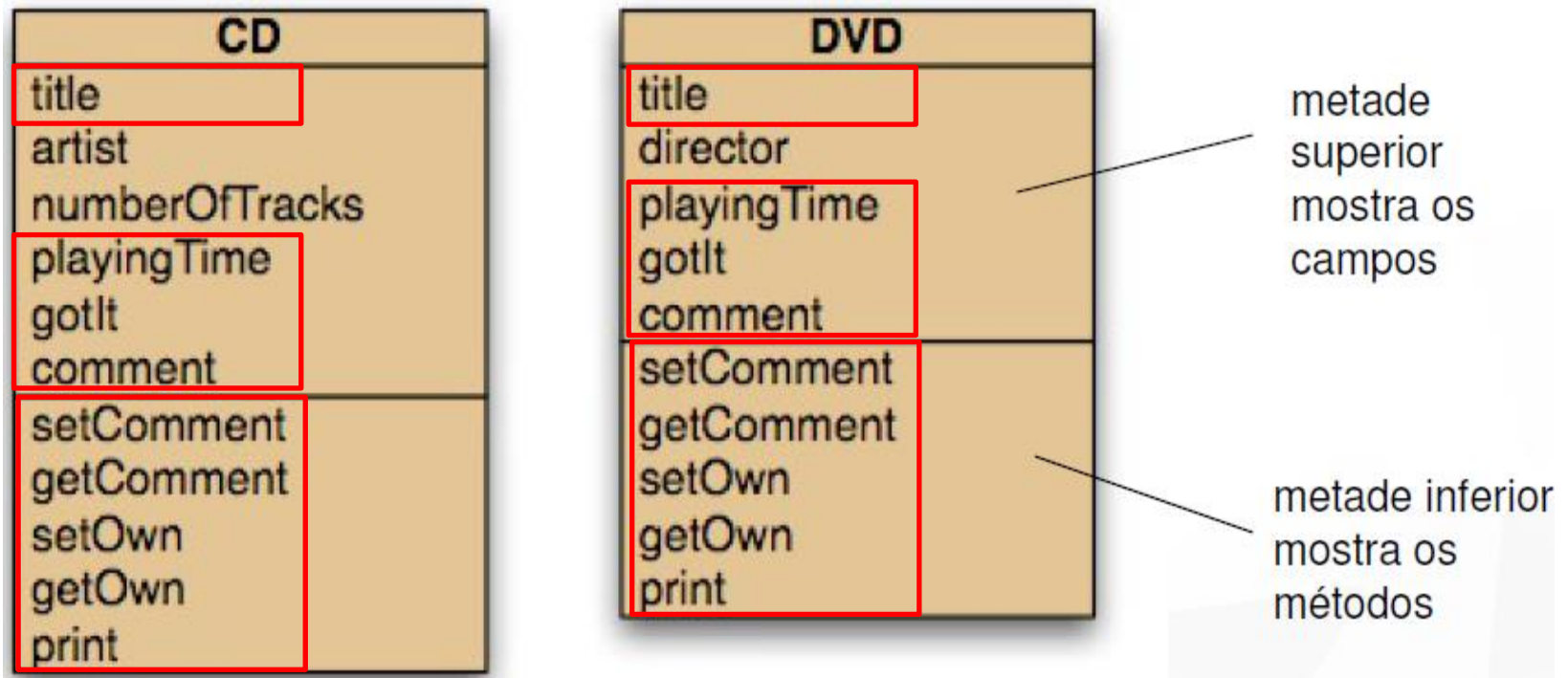


metade superior mostra os campos

metade inferior mostra os métodos

Introdução

- ❑ Vamos analisar o exemplo abaixo:
- ❑ O que podemos **perceber**???



Introdução

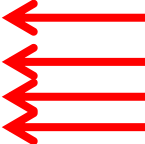
□ Classe CD:

```
8 public class CD {
9     private String title;
10    private String artist;
11    private int numberOfTracks;
12    private int playingTime;
13    private boolean gotIt;
14    private String comment;
15
16    /**
17     * Initialize the CD.
18     * @param theTitle The title of the CD.
19     * @param theArtist The artist of the CD.
20     * @param tracks The number of tracks on the CD.
21     * @param time The playing time of the CD.
22     */
23    public CD(String theTitle, String theArtist, int tracks, int time){
24        title = theTitle;
25        artist = theArtist;
26        numberOfTracks = tracks;
27        playingTime = time;
28        gotIt = false;
29        comment = "";
30    }
```

Introdução

□ Classe CD:

```
8 public class CD {
9     private String title;
10    private String artist;
11    private int numberOfTracks;
12    private int playingTime;
13    private boolean gotIt;
14    private String comment;
15
16    /**
17     * Initialize the CD.
18     * @param theTitle The title of the CD.
19     * @param theArtist The artist of the CD.
20     * @param tracks The number of tracks on the CD.
21     * @param time The playing time of the CD.
22     */
23    public CD(String theTitle, String theArtist, int tracks, int time){
24        title = theTitle;
25        artist = theArtist;
26        numberOfTracks = tracks;
27        playingTime = time;
28        gotIt = false;
29        comment = "";
30    }
```



Introdução

❑ Classe CD:

```
62  /**
63   * Print details about this CD to the text terminal.
64   */
65  public void print() { ←
66      System.out.print("CD: " + title + " (" + playingTime + " mins)");
67      if(gotIt) {
68          System.out.println("*");
69      }
70  else {
71      System.out.println();
72  }
73      System.out.println("    " + artist);
74      System.out.println("    tracks: " + numberOfTracks);
75      System.out.println("    " + comment);
76  }
77 }
```

Introdução

□ Classe DVD:

```
9 public class DVD {
10     private String title;
11     private String director;
12     private int playingTime; // playing time of the main feature
13     private boolean gotIt;
14     private String comment;
15
16     /**
17      * Constructor for objects of class DVD
18      * @param theTitle The title of this DVD.
19      * @param theDirector The director of this DVD.
20      * @param time The running time of the main feature.
21      */
22     public DVD(String theTitle, String theDirector, int time){
23         title = theTitle;
24         director = theDirector;
25         playingTime = time;
26         gotIt = false;
27         comment = "";
28     }
```

Introdução

□ Classe DVD:

```
9 public class DVD {
10     private String title; ←
11     private String director;
12     private int playingTime; ← playing time of the main feature
13     private boolean gotIt;
14     private String comment; ←
15
16     /**
17      * Constructor for objects of class DVD
18      * @param theTitle The title of this DVD.
19      * @param theDirector The director of this DVD.
20      * @param time The running time of the main feature.
21      */
22     public DVD(String theTitle, String theDirector, int time){
23         title = theTitle;
24         director = theDirector;
25         playingTime = time;
26         gotIt = false;
27         comment = "";
28     }
```

Introdução

❑ Classe DVD:

```
60  /**
61   * Print details about this DVD to the text terminal.
62   */
63  public void print() { ←
64      System.out.print("DVD: " + title + " (" + playingTime + " mins)");
65      if(gotIt) {
66          System.out.println("*");
67      }
68      else {
69          System.out.println();
70      }
71      System.out.println("    " + director);
72      System.out.println("    " + comment);
73  }
74 }
```

Introdução

- ❑ Duplicação de código:
 - ❖ Classes CD e DVD muito semelhantes (grande parte é idêntica);
 - ❖ Torna manutenção difícil/mais trabalho;
 - ❖ Introduz risco de **bugs** por meio de manutenção incorreta.

Introdução

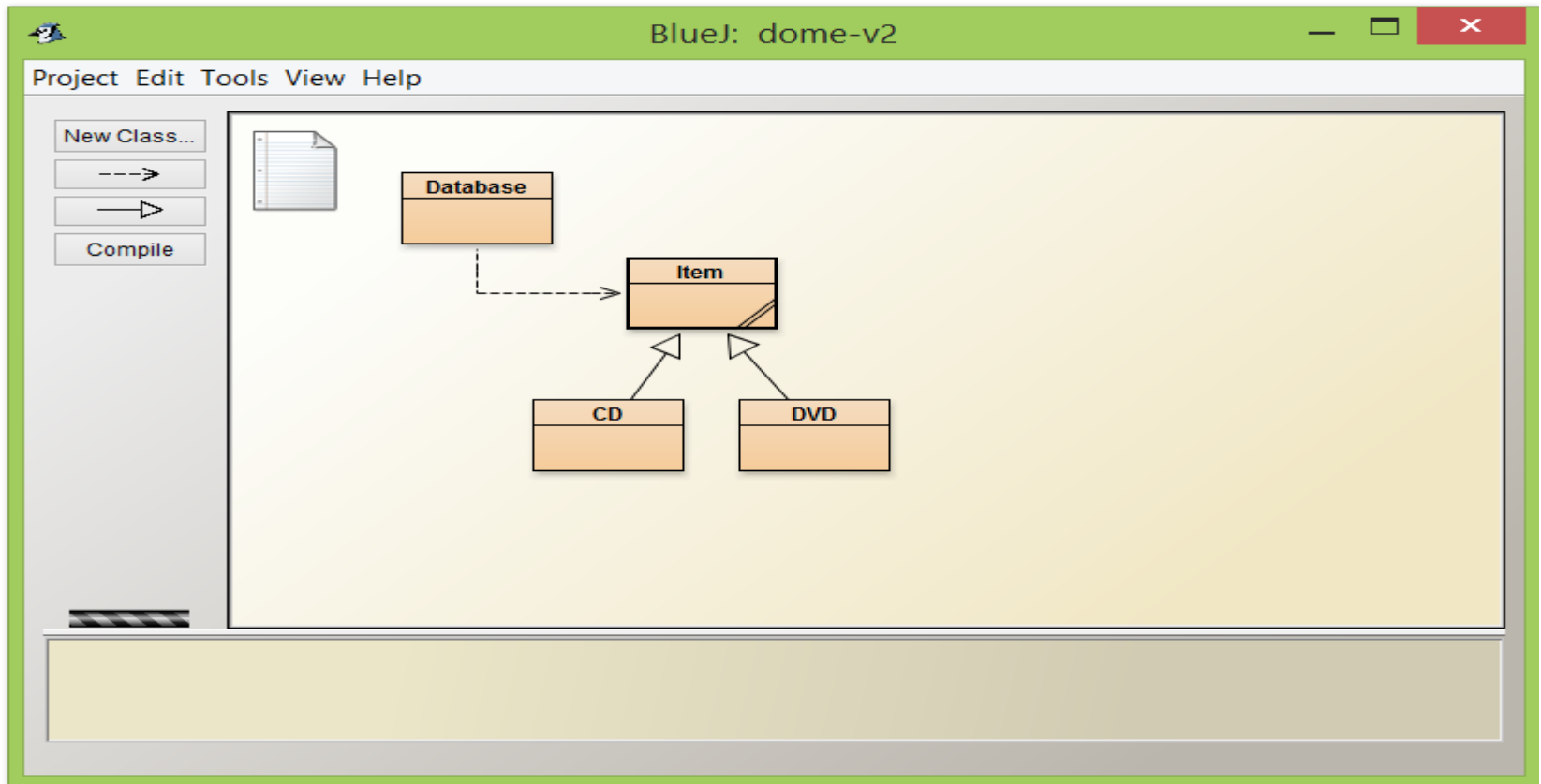
❑ Como resolver o **problema**???



<http://fagnervianna.com.br/3-maneyras-de-usar-o-email-marketing-lucrativo-e-eficiente/homem-pensando-email-marketing/>

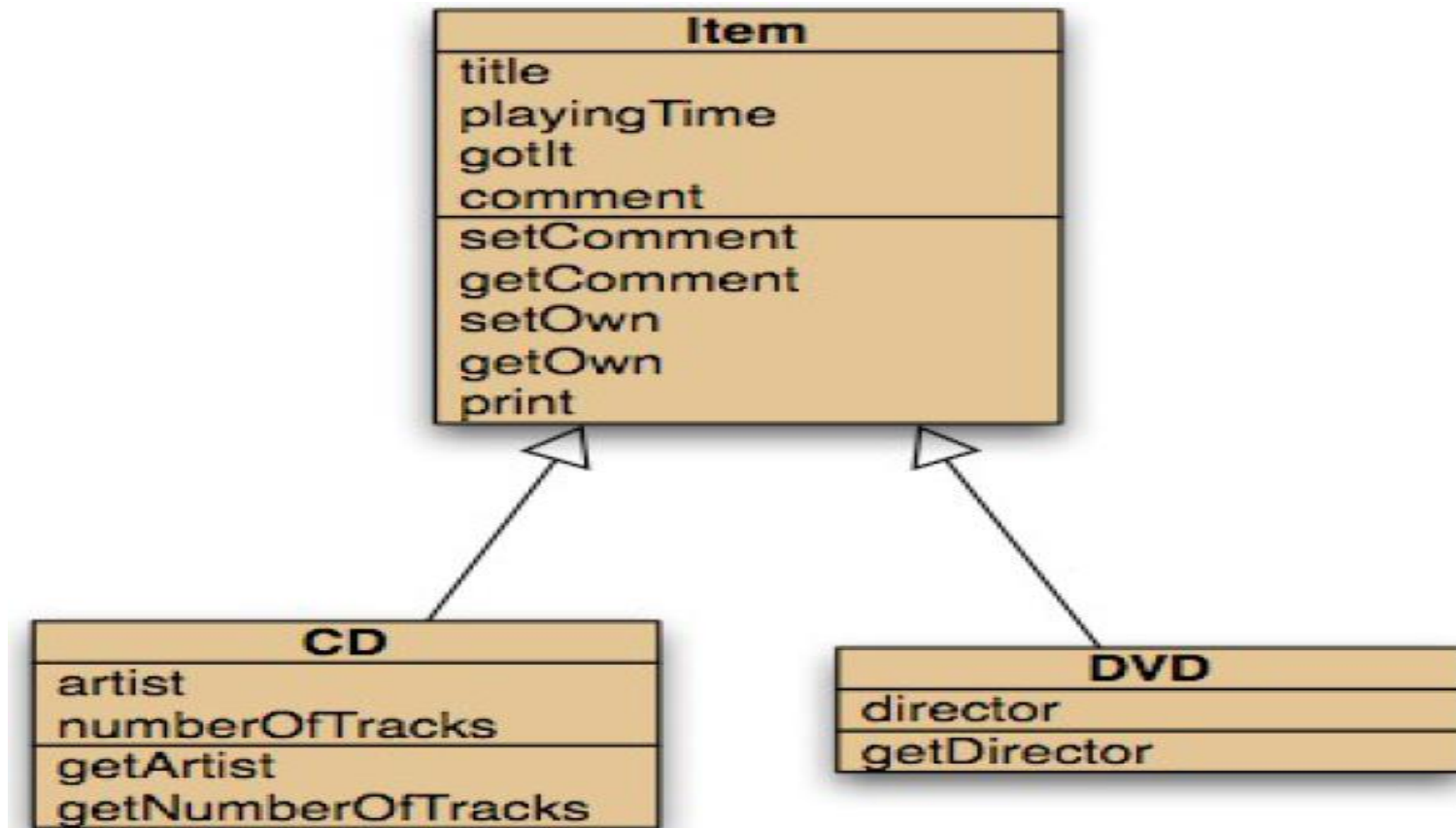
Herança

Usando Herança:



Herança

❑ Usando Herança:

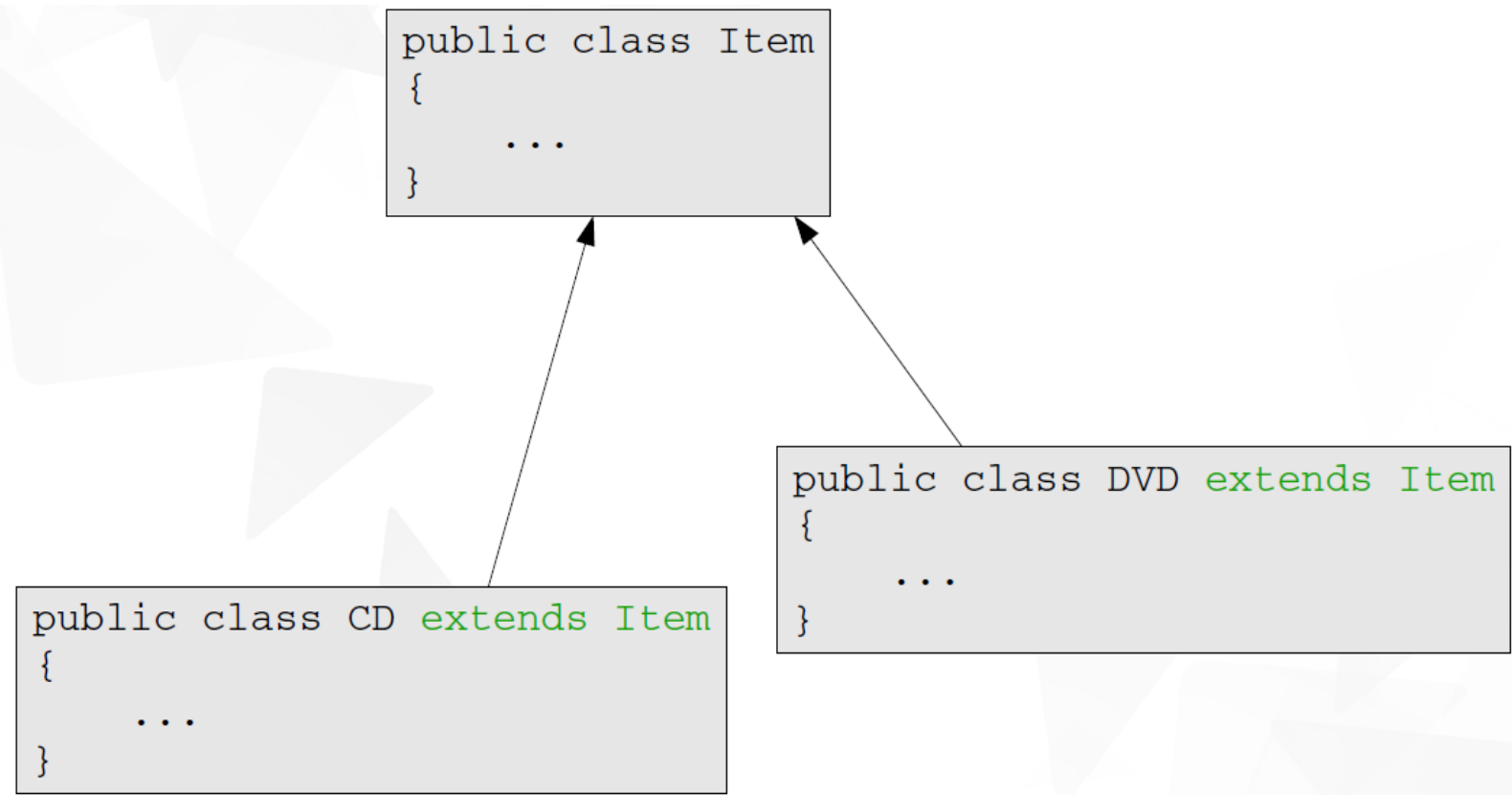


Herança

- ❑ Como funciona:
 - ❖ Define uma **superclasse**: item;
 - ❖ Define **subclasses** para DVD e CD;
 - ❖ A superclasse define **atributos comuns**;
 - ❖ As subclasses herdam os **atributos da superclasse**;
 - ❖ As subclasses adicionam **atributos próprios**.

Herança

□ Herança em Java:



Herança

❑ Programando a Superclasse:

```
public class Item {  
  
    protected String title;  
    protected int playingTime;  
    protected boolean gotIt;  
    protected String comment;  
  
    public Item() {  
        gotIt = false;  
        comment = "";  
        title = "";  
    }  
}
```

Herança

□ Programando as Subclasses:

```
public class CD extends Item{  
  
    private String artist;  
    private int numberOfTracks;  
  
}
```

Herança

□ Programando as Subclasses:

```
public class DVD extends Item{  
    private String director;  
    public String getDirector() {  
        return director;  
    }  
    public void setDirector(String director) {  
        this.director = director;  
    }  
}
```


Herança

❑ Construtores:

```
public class CD extends Item
{
    private String artist;
    private int numberOfTracks;

    public CD(String theTitle, String theArtist,
              int track, int time)
    {
        → super(theTitle, time);
        artist = theArtist;
        numberOfTracks = tracks;
    }

    //métodos omitidos
}
```

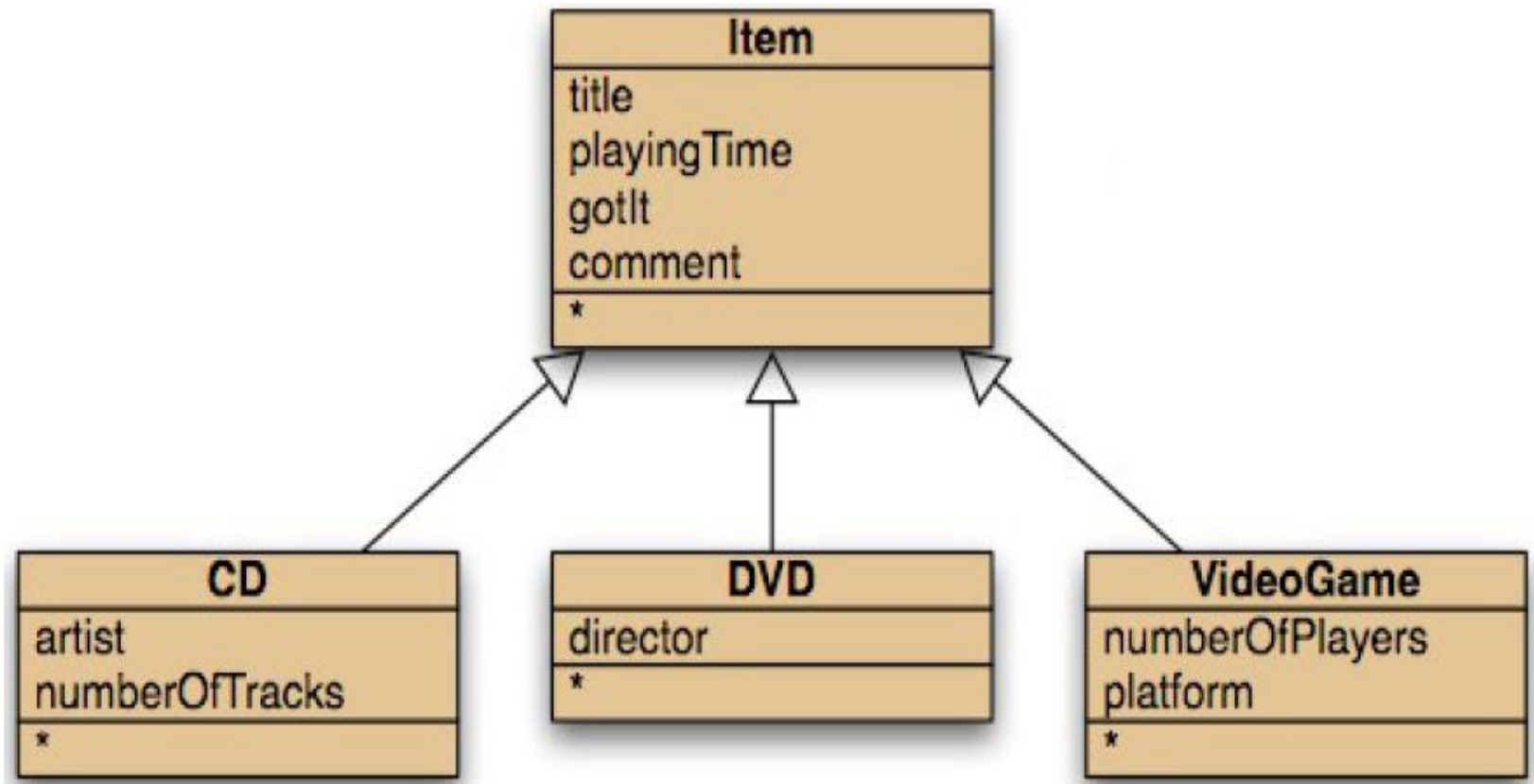
Herança

❑ Construtores:

```
public class DVD extends Item {  
    private String director;  
  
    public DVD(String theTitle, String theDirector,  
               int time) {  
        → super(theTitle, time);  
        director = theDirector;  
    }  
  
    public String getDirector() {  
        return director;  
    }  
}
```

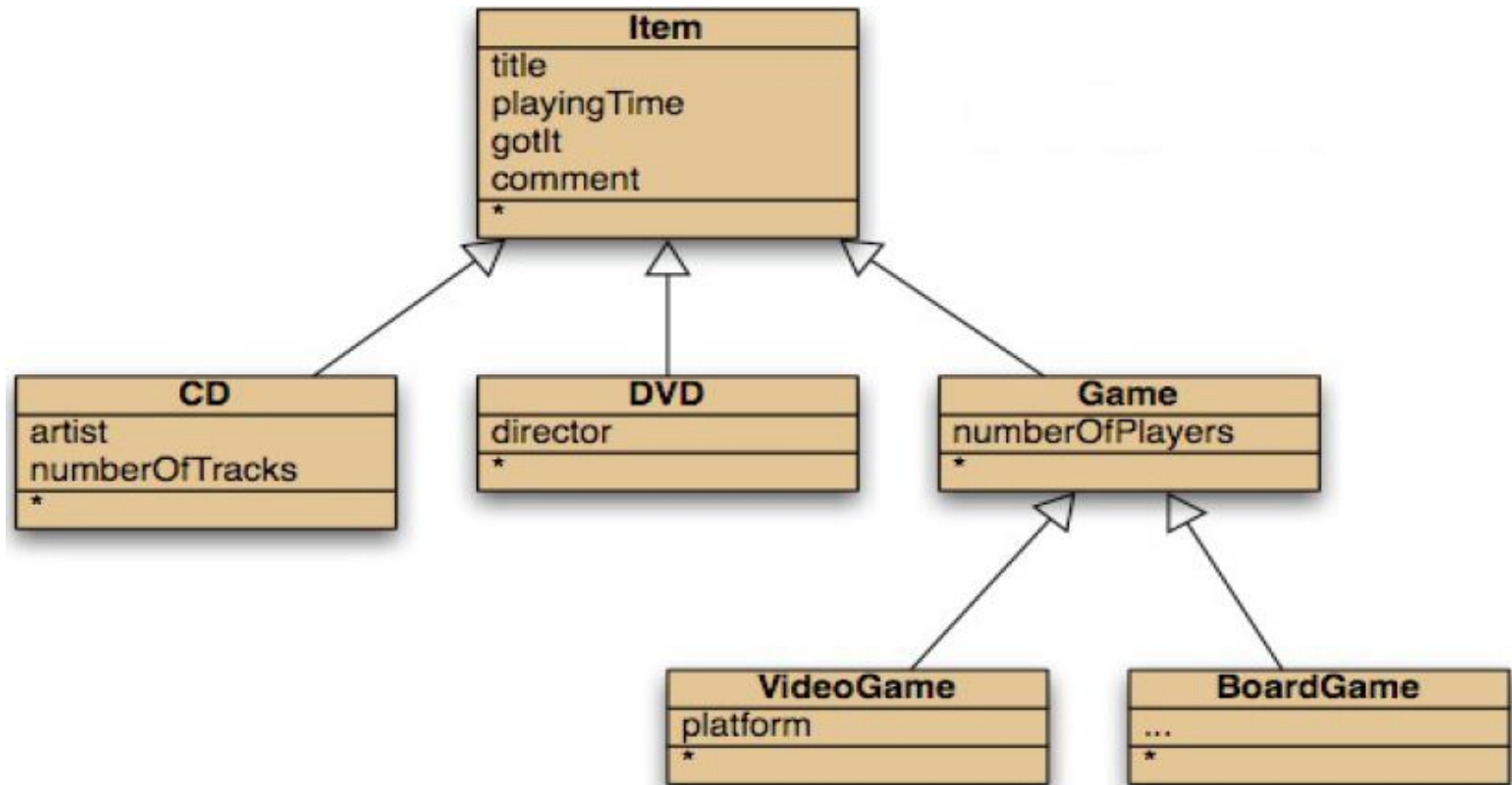
Herança

- Adicionando mais um item:



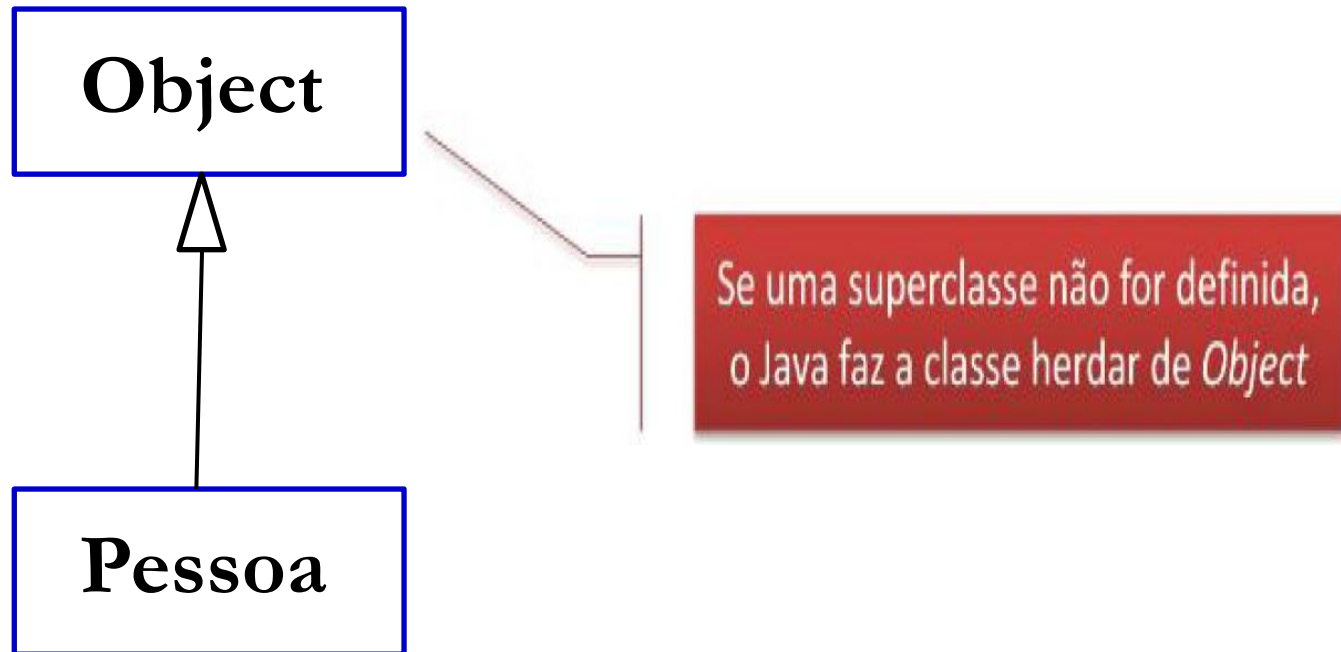
Herança

- Hierarquias mais profundas:



Herança

- Todas as classes **herdam** apenas de uma superclasse:



Herança

- ❑ Sobrescrita de métodos:
 - ❖ Técnica conhecida como **override**;
 - ❖ Quando uma classe herda de outra, ela pode **redefinir métodos** da superclasse, isto é, sobrescrever métodos.
 - Os métodos sobrescritos **substituem** os métodos da superclasse;
 - A **assinatura do método** sobrescrito deve ser a mesma do método original.

Herança

❑ Sobrescrita de métodos:

- ❖ Usando o **super**;
- ❖ O método que foi sobrescrito pode ser acessado pelo método que o sobrescreveu através da palavra-chave **super**.

Item.java


```
public void print(){
    System.out.print("Item: " + title + " (" + playingTime + " mins)");
    if(gotIt) {
        System.out.println("*** possuio ***");
    }
    else {
        System.out.println();
    }
}
```

Herança

❑ Sobrescrita de métodos:

- ❖ Usando o **super**;
- ❖ O método que foi sobrescrito pode ser acessado pelo método que o sobrescreveu através da palavra-chave **super**.

```
/**
 * Print details about this CD to the text terminal.
 */
@Override
public void print() {
    super.print();
    System.out.println("    " + artist);
    System.out.println("    tracks: " + numberOfTracks);
    System.out.println("    " + comment);
}
```




Herança

❑ Sobrescrita de métodos:

- ❖ Usando o **super**;
- ❖ O método que foi sobrescrito pode ser acessado pelo método que o sobrescreveu através da palavra-chave **super**.

```
/**
 * Print details about this DVD to the text terminal.
 */
@Override
public void print() {
    super.print();
    System.out.println("    " + director);
    System.out.println("    " + comment);
}
```



Perguntas ...



Tipo Estático e Tipo Dinâmico

- ❑ O tipo declarado de uma variável é seu **tipo estático**.
- ❑ O tipo do objeto ao qual uma variável se refere é seu **tipo dinâmico**.

Tipo Estático e Tipo Dinâmico

- ❑ O tipo declarado de uma variável é seu **tipo estático**.
- ❑ O tipo do objeto ao qual uma variável se refere é seu **tipo dinâmico**.

Qual o tipo de c?

```
Car c = new Car();
```

Qual o tipo de v?

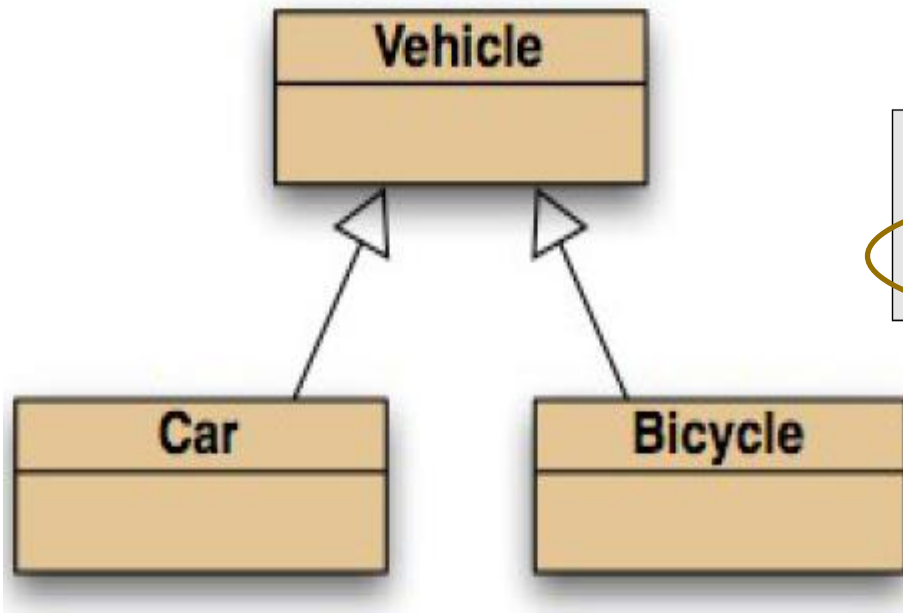
```
Vehicle v = new Car();
```

Tipo Estático e Tipo Dinâmico

- ❑ Uma referência para a **superclasse** pode apontar para um objeto da **subclasse**.
- ❑ Porém:
 - ❖ **O contrário não é verdadeiro!**
- ❑ Exemplo:
 - ❖ Todo **carro** é **veículo**, mas nem todo **veículo** é **carro**.
 - ❖ Pode ser uma **bicicleta**.

Tipo Estático e Tipo Dinâmico

- Uma variável pode conter **objetos** de seu tipo declarado ou de qualquer **subtipo** declarado.



```
Vehicle v1 = new Vehicle();  
Vehicle v2 = new Car();  
Vehicle v3 = new Bicycle();
```

Tipo Estático e Tipo Dinâmico

- ❑ Variáveis podem armazenar **objetos** de mais de um tipo:
 - ❖ Objetos do tipo declarado; ou
 - ❖ Objetos do subtipo do tipo declarado.

```
Vehicle v1 = new Vehicle();  
Vehicle v2 = new Car();  
Vehicle v3 = new Bicycle();
```

Perguntas ...



Conceitos

- ❑ **Não confunda Overloading** ou com **Overriding**.
- ❑ Overloading (Sobrecarga de Método):
 - ❖ Same method name but different parameters.
 - ❖ Overloading happens at compile-time;
 - ❖ **Static binding** is being used for overloaded methods.
- ❑ Overriding (Sobrescrita de Método):
 - ❖ Same method name and parameters (i.e., method signature);
 - ❖ Overriding happens at runtime;
 - ❖ **Dynamic binding or Late binding** is being used for overriding methods.

Obrigado!!!

