Herança e Polimorfismo

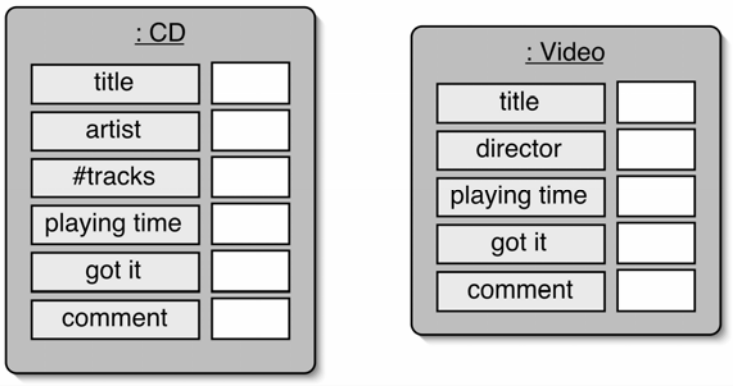
- Parte I -

Mário Meireles Teixeira
mario@deinf.ufma.br

O exemplo DoME

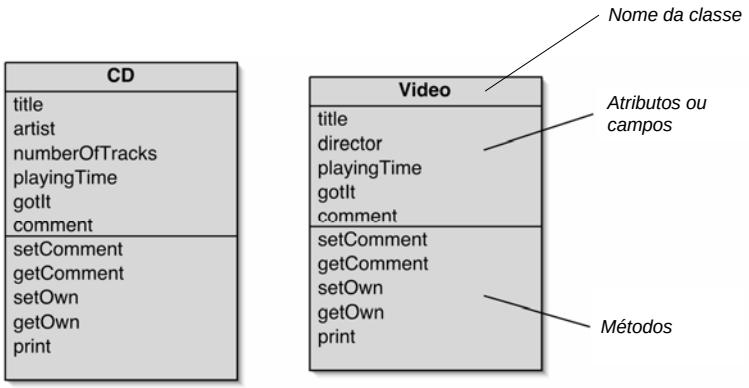
- "Database of Multimedia Entertainment"
- Armazena detalhes sobre CDs e vídeos
  - CD: título, artista, número de faixas, duração, comentário, flag de posse
  - Vídeo: título, diretor, duração, flag de posse, comentário.
- Permite pesquisar informações ou imprimir listas

Objetos DoME



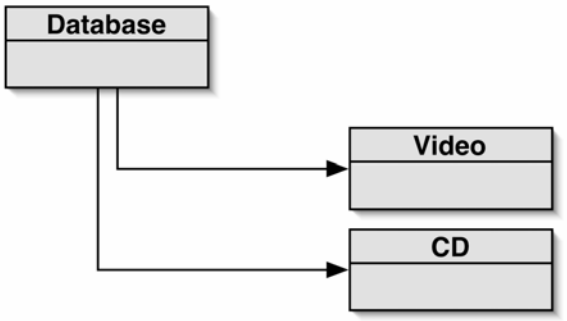
3

Classes DoME



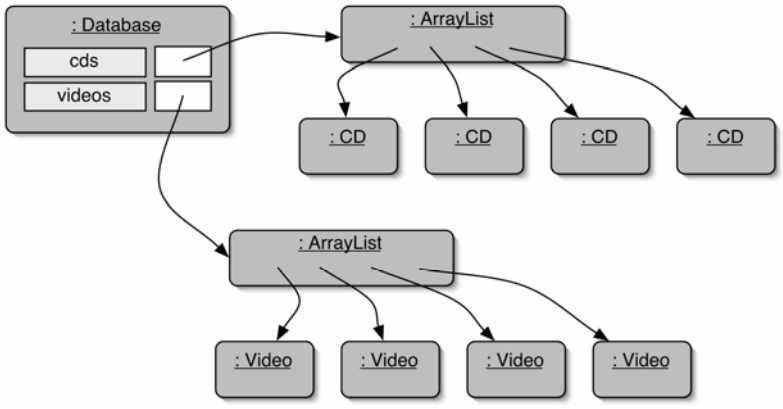
4

Diagrama de classes



5

Modelo de objetos



6

Classe CD

```
public class CD {
   private String title;
   private String artist;
   private String comment;

   public CD(String theTitle, String theArtist)
   {
       title = theTitle;
       artist = theArtist;
       comment = " ";
   }

   public void setComment(String newComment) { ... }

   public String getComment() { ... }

   public void print() { ... }

   ...
}
```

7

Classe Video

```
public class Video {
   private String title;
   private String director;
   private String comment;

   public CD(String theTitle, String theDirector)
   {
       title = theTitle;
       artist = theDirector;
       comment = " ";
   }

   public void setComment(String newComment) { ... }

   public String getComment() { ... }

   public void print() { ... }

   ...
}
```

8

Classe Database (1)

```
public class Database {

   private ArrayList cds;
   private ArrayList videos;

    public Database()  {
        cds = new ArrayList();            // código repetitivo
        videos = new ArrayList();
    }

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

    public void addVideo(Video theVideo)  {
        videos.add(theVideo);
    }
    . . .
}
```

9

Classe Database (2)

```
class Database {
   . . .
   public void list()                    // código repetitivo
   {
        // lista de CDs
        for (Iterator iter = cds.iterator(); iter.hasNext(); ) {
            CD cd = (CD) iter.next();
            cd.print();
            System.out.println();
         }

        // lista de vídeos
        for (Iterator iter = videos.iterator(); iter.hasNext(); ) {
            Video video = (Video) iter.next();
            video.print();
            System.out.println();

         }
   }
}
```

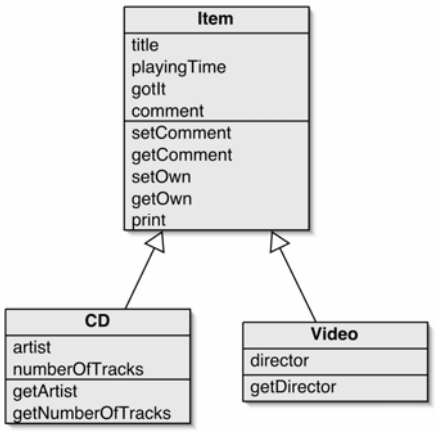
10

Crítica ao DoME

- Duplicação de código:
  - classes `CD` e `Video` bem semelhantes (boa parte é idêntica);
  - torna a manutenção difícil/mais trabalhosa;
  - introduz o perigo de bugs por causa de uma manutenção incorreta
- Duplicação de código também na classe `Database`

11

Utilizando herança



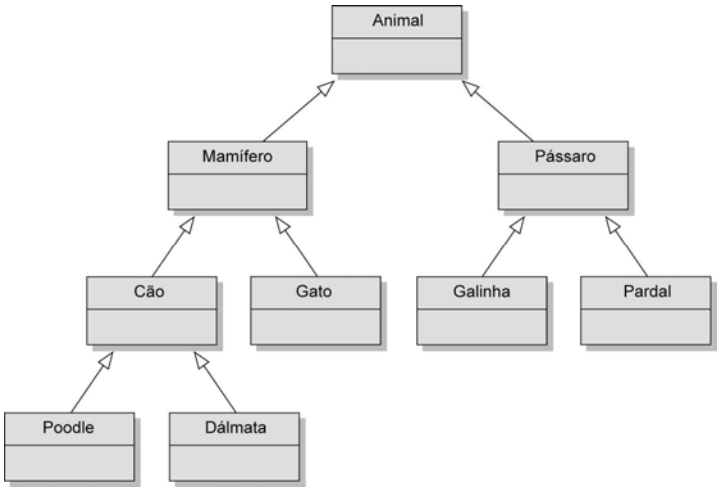
12

Utilizando herança

- Define uma **superclasse**: `Item`
- Define **subclasses** para `Video` e `CD`
- A superclasse define os atributos e métodos comuns
- As subclasses **herdam** os atributos e métodos da superclasse
- As subclasses adicionam seus próprios atributos e métodos

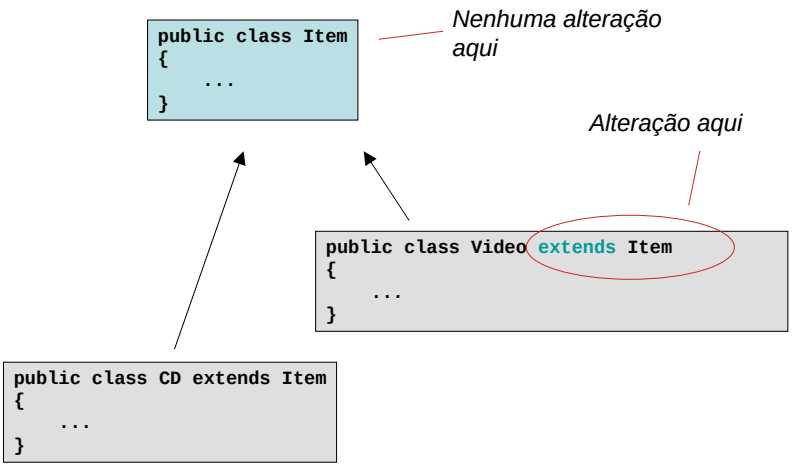
13

Hierarquias de herança



14

Herança em Java



15

Superclasse

```
public class Item
{
    private String title;
    private int playingTime;
    private boolean gotIt;
    private String comment;

    // construtores e métodos omitidos
}
```

16

Subclasses

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

    // construtores e métodos omitidos
}
```

```
public class Video extends Item
{
    private String director;

    // construtores e métodos omitidos
}
```

17

Herança e construtores

```
public class Item
{
    private String title;
    private int playingTime;
    private boolean gotIt;
    private String comment;

    public Item(String theTitle, int time)
    {
        title = theTitle;
        playingTime = time;
        gotIt = false;
        comment = "";
    }
    . . .
}
```

18

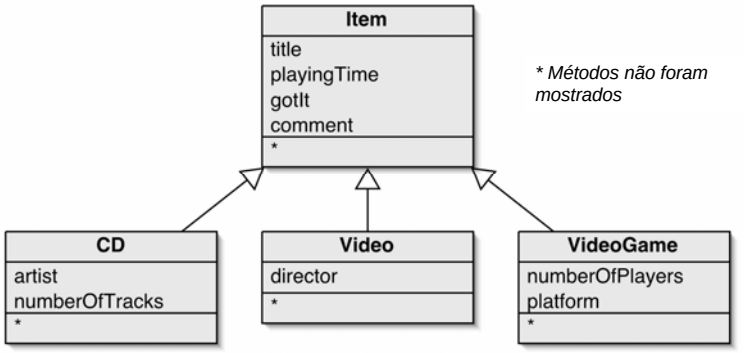
Herança e construtores

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

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

    . . .
}
```


Chamada para o construtor de superclasse

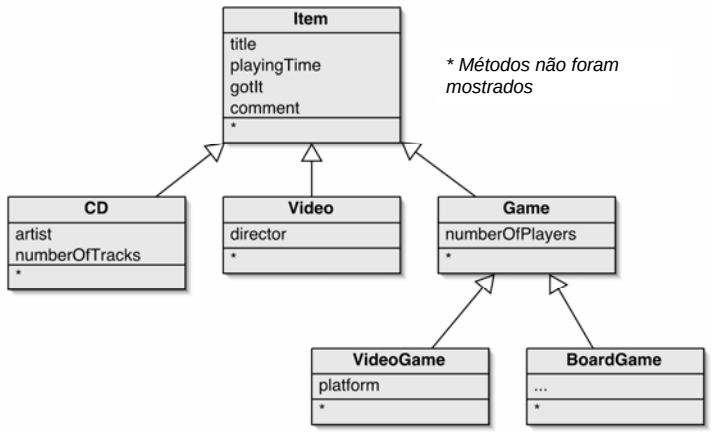
- Construtores de subclasse sempre devem conter uma chamada 'super'
- Deve ser a primeira instrução no construtor da subclasse
- Se nenhuma for escrita, o compilador inserirá uma (sem parâmetros):
  - funciona apenas se a superclasse tiver um construtor sem parâmetros

Adicionando outros tipos de item



21

Hierarquias mais especializadas



22

Vantagens da Herança

Herança (até aqui) ajuda a:

- evitar duplicação do código;
- reutilizar o código;
- simplificar a manutenção; e
- a extensibilidade

23

Nova Classe `Database` (1)

```
public class Database
{
    private ArrayList items;

    public Database()
    {
        items = new ArrayList();        // código genérico
    }

    public void addItem(Item theItem)
    {
        items.add(theItem);
    }
    ...
}
```

24

Nova Classe `Database` (2)

```
/**
 * Imprime uma lista de todos os CDs e vídeos atualmente
 * armazenados na base de dados
*/
public void list()
{
    for ( Iterator iter = items.iterator(); iter.hasNext(); )
    {
        Item item = (Item) iter.next();
        item.print();
        System.out.println();
    }
}
```


Subtipagem

- Inicialmente, tínhamos:

```
public void addCD(CD theCD)
public void addVideo(Video theVideo)
```

- Utilizando herança, temos:

```
public void addItem(Item theItem)
```

- Esse método é chamado com:

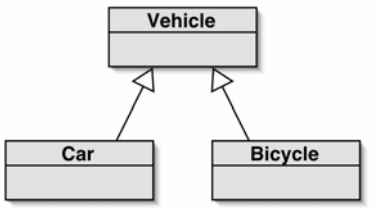
```
Video myVideo = new Video(...);
database.addItem(myVideo);
```

Subclasses e subtipos

- Classes definem tipos; Subclasses definem subtipos
- Objetos das subclasses podem ser utilizados onde os objetos dos supertipos são requeridos → **substituição**
- Uma variável pode conter objetos do seu tipo declarado ou de quaisquer subtipos do seu tipo declarado

27

Subtipagem e atribuição



Objetos da subclasse podem ser atribuídos a variáveis da superclasse

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

```
Car c1 = new Vehicle();
Car c2 = new Bicycle();
```

Inválido!

28

Subtipagem e passagem de parâmetros

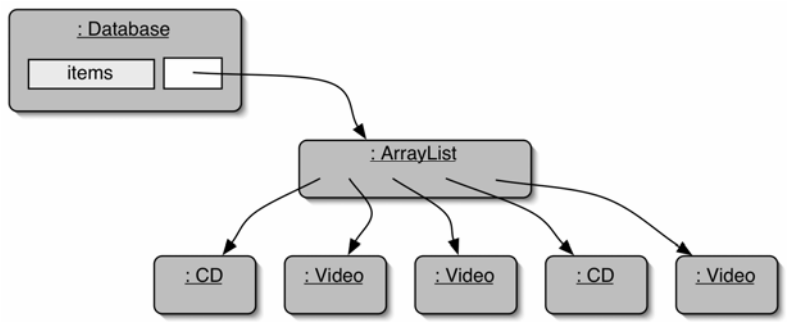
```
public class Database
{
    ...
    public void addItem(Item theItem)
    {
        ...
    }
}
```

```
Video video = new Video(...);
CD cd = new CD(...);

database.addItem(video);
database.addItem(cd);
```

Objetos da subclasse podem ser passados como parâmetros para a superclasse

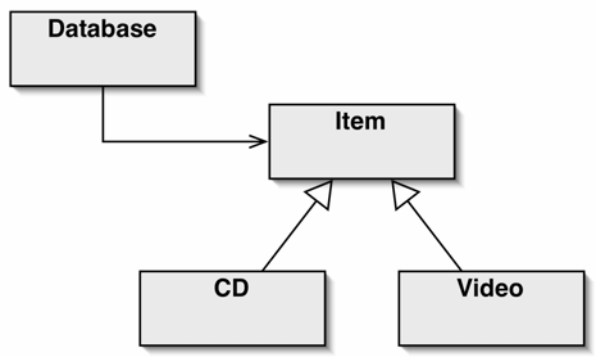
29

Diagrama de objetos



30

Diagrama de classes

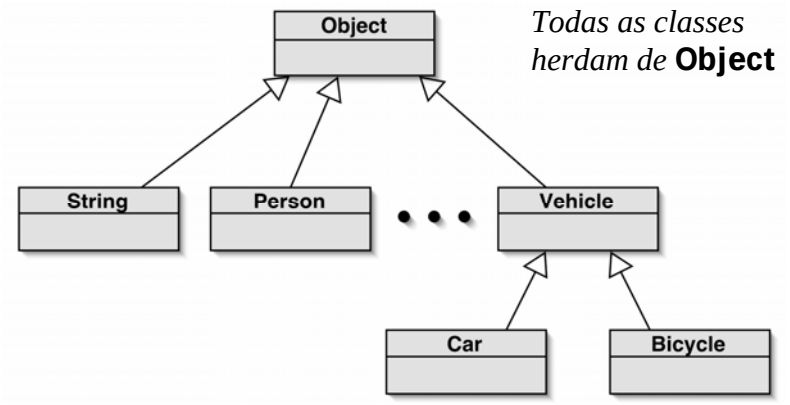


Variáveis polimórficas

- Variáveis que contêm tipos de objeto em Java são **polimórficas**
- Elas podem armazenar objetos do tipo declarado ou dos subtipos do tipo declarado:

```
for ( Iterator iter = items.iterator();
iter.hasNext(); )
     {
          Item item = (Item) iter.next();
          item.print();
          System.out.println();
     }
```

A classe `Object`



Todas as classes herdam de **Object**

33

Coleções polimórficas

- Todas as coleções são polimórficas. Seus elementos são do tipo `Object`:

```
public void add(Object element)

public Object get(int index)
```

- A coleção `ArrayList` herda de `AbstractList` que, por sua vez, herda de `AbstractCollection`

34

Conversão de supertipo para subtipo

- Pode-se atribuir um subtipo ao supertipo, porém não se pode atribuir um supertipo ao subtipo diretamente:

```
String s1 = myList.get(1);    // erro!
```

- Nesse caso, a conversão de tipos é obrigatória:

```
String s1 = (String) myList.get(1);
```

Válido somente se o elemento for realmente uma String

```
Vehicle v;  Car c;  Bicycle b;

c = new Car();
v = c;                  // ok
b = (Bicycle) c;        // erro em tempo de compilação
b = (Bicycle) v;        // erro em tempo de execução
```

35

Classes *wrappers* (empacotadoras)

- Tipos simples (`int`, `char`, `float`) não são objetos, porém, às vezes, eles precisam ser tratados como objetos

Tipo simples	*Classe empacotadora*
`byte`	`Byte`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`
`char`	`Character`
`boolean`	`Boolean`

36

Classes *wrappers*

Empacota o valor de int

```
int i = 18;
Integer iwrap = new Integer(i);

myCollecton.add(iwrap);
...

Integer element = (Integer) myCollection.get(0);
int value = element.intValue()
```

Adiciona o tipo empacotado

Recupera o tipo empacotado

Desempacota

37