Conception objet en Java avec BlueJ
une approche interactive

8. Mieux structurer avec l'héritage

David J. Barnes, Michael Kölling
version française: Patrice Moreaux

Rédigé avec OpenOffice.org

Principaux concepts étudiés

- Héritage
- Sous-typage
- Substitution
- Variables polymorphes

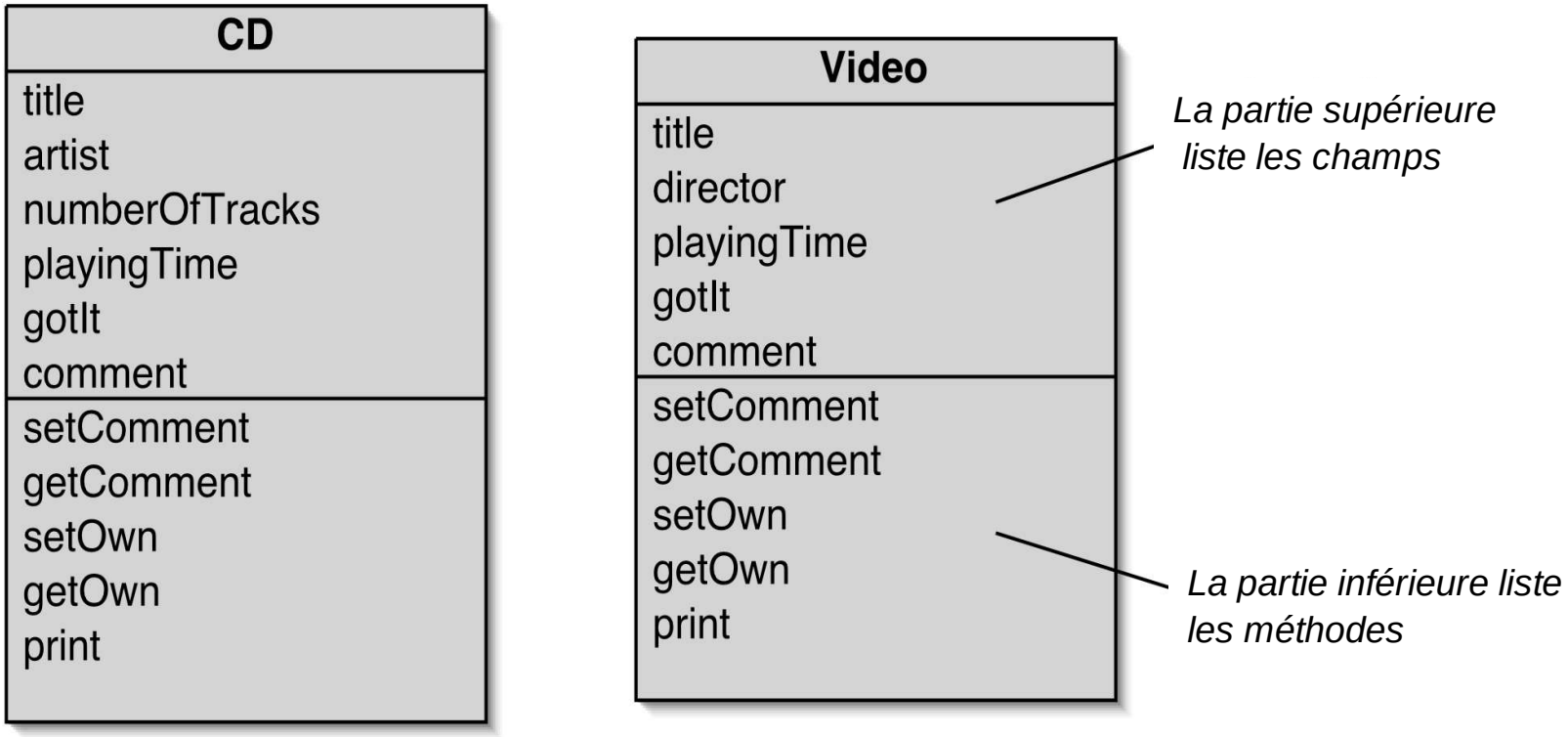
Le projet *DoME*

"Base de données Multimedia de loisirs"

- Stocke les informations sur des CDs et des vidéos
  - CD: titre(title), artiste(artist), nbre de pistes (# tracks), durée (playing time), obtenu(got-it), commentaire(comment)
  - Video: titre, réalisateur(director), durée, obtenu, commentaire
- Permettra de rechercher des informations ou d'afficher des listes

Objets de *DoME*

Classes de *DoME*



Modèle objet de *DoME*

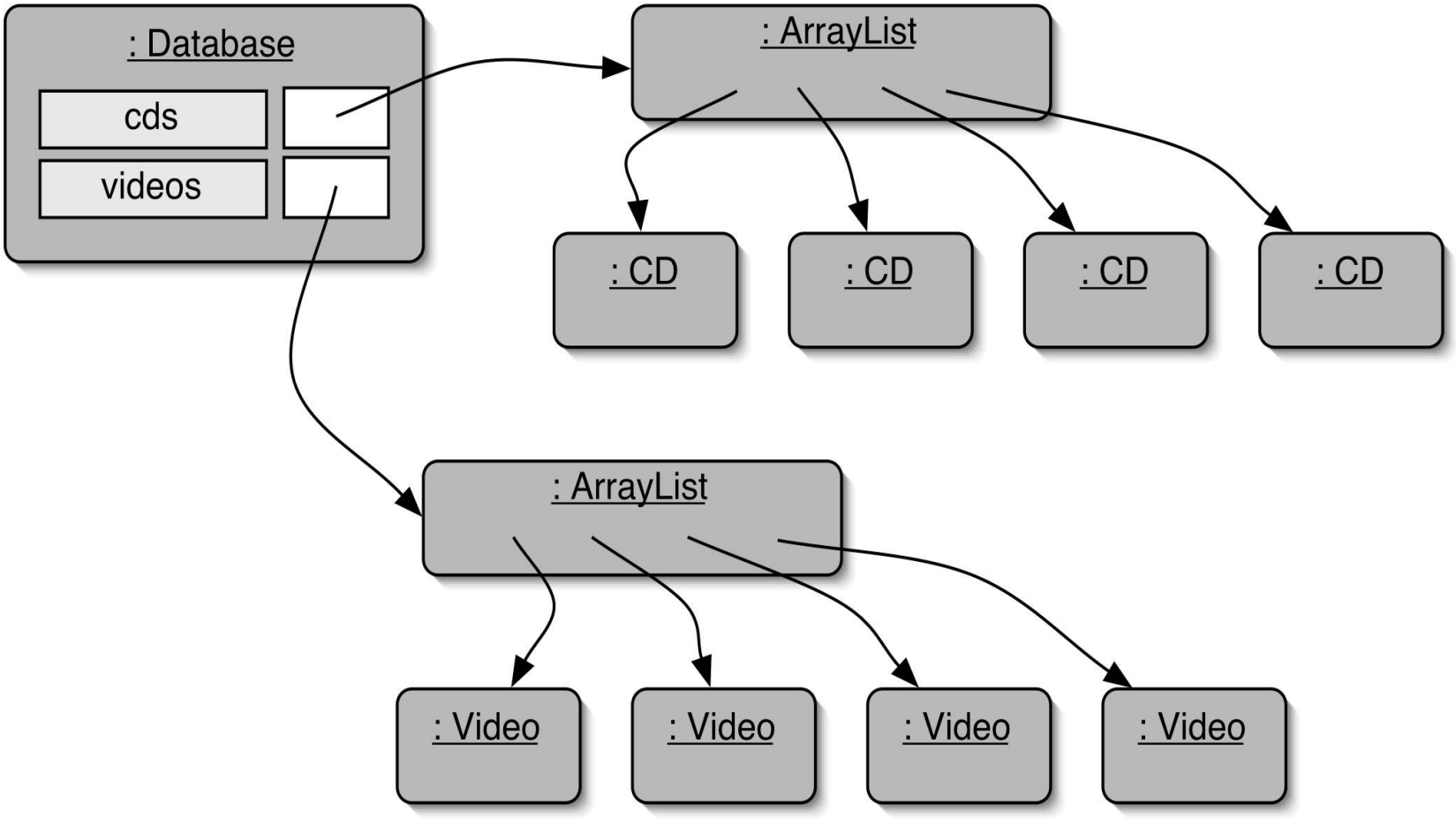
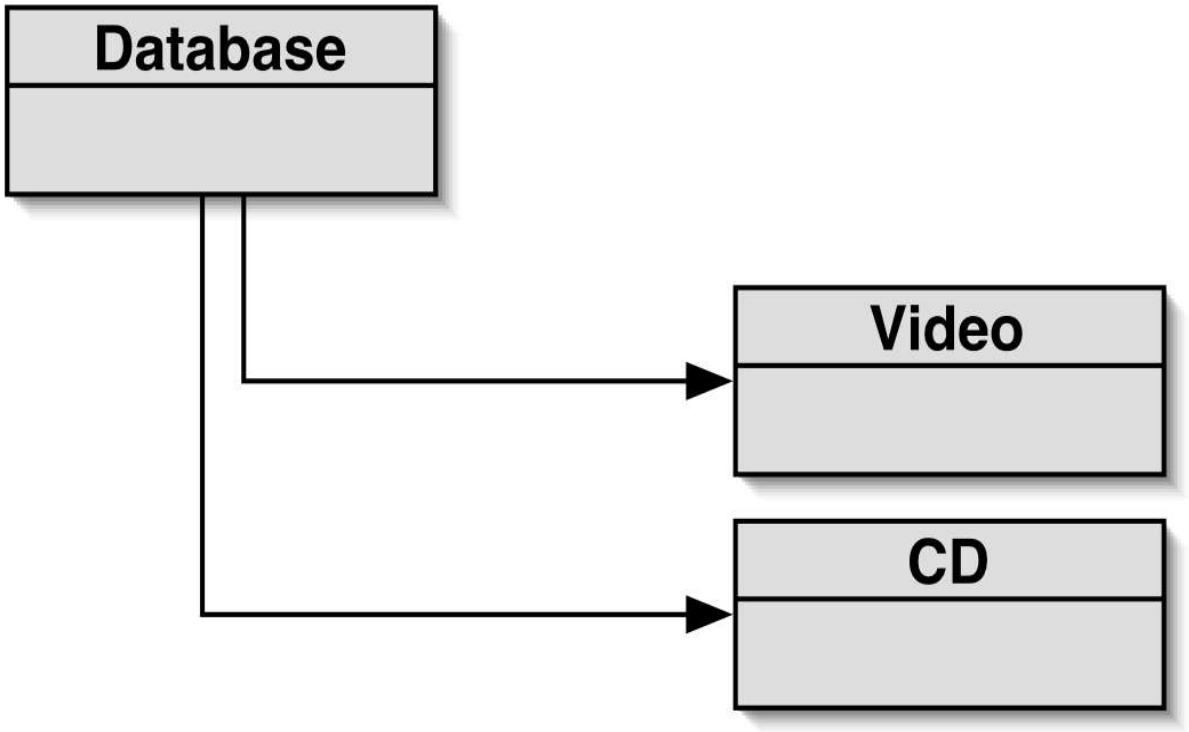


Diagramme de classes



CD: Code source

Commentaires à ajouter!

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

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

    void setComment(String newComment)
    { ... }

    String getComment()
    { ... }

    void print()
    { ... }
```

Video: code source

Commentaires à ajouter!

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

   Video(String theTitle, String
theDirect)
   {
       title = theTitle;
       director = theDirect;
       comment = " ";
   }

   void setComment(String newComment)
   { ... }

   String getComment()
   { ... }

   void print()
   { ... }
   ...
```

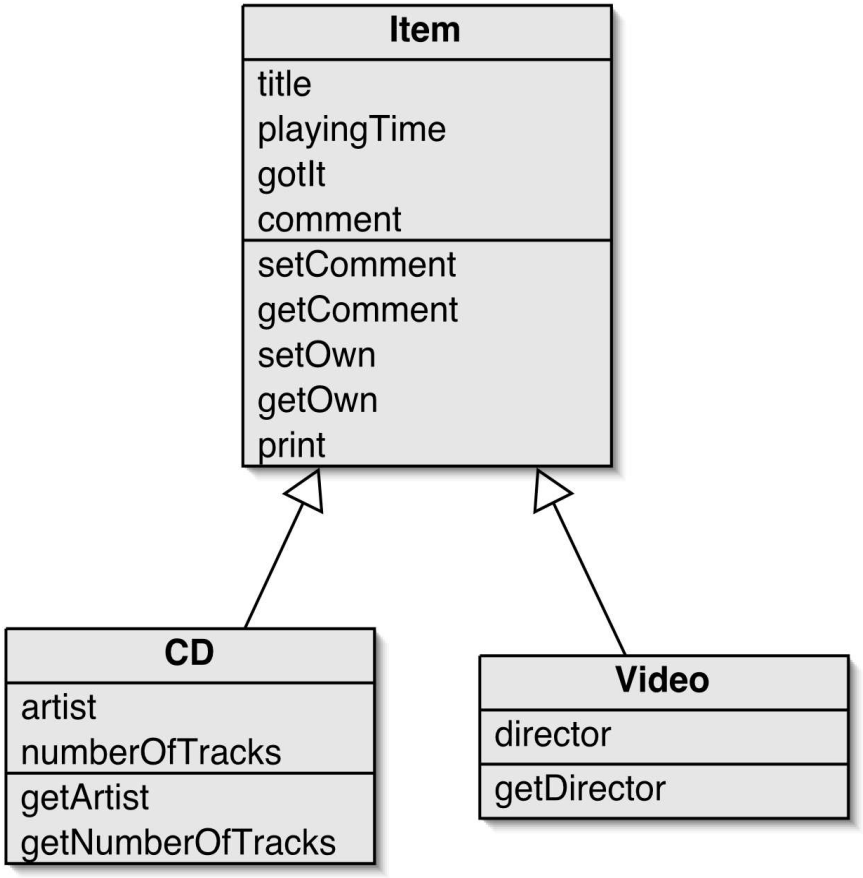
Database: code source

```
class Datab

    private Arra
    private ArrayList videos;
    ...

    public void list()
    {
        for(Iterator iter = cds.iterator(); iter.hasNext(); ) {
            CD cd = (CD)iter.next();
            cd.print();
            System.out.println();   // ligne vide entre items
        }

        for(Iterator iter = videos.iterator(); iter.hasNext(); ) {
            Video video = (Video)iter.next();
            video.print();
            System.out.println();   // ligne vide entre items
        }
    }
}
```

Analyse critique de *DoME*

- Duplication de code
  - les classes CD et Video sont très semblables (plusieurs parties sont même identiques)
  - rend la maintenance laborieuse et plus difficile
  - augmente les risques d'erreurs dues à une maintenance incorrecte
- Duplication de code aussi dans la classe Database

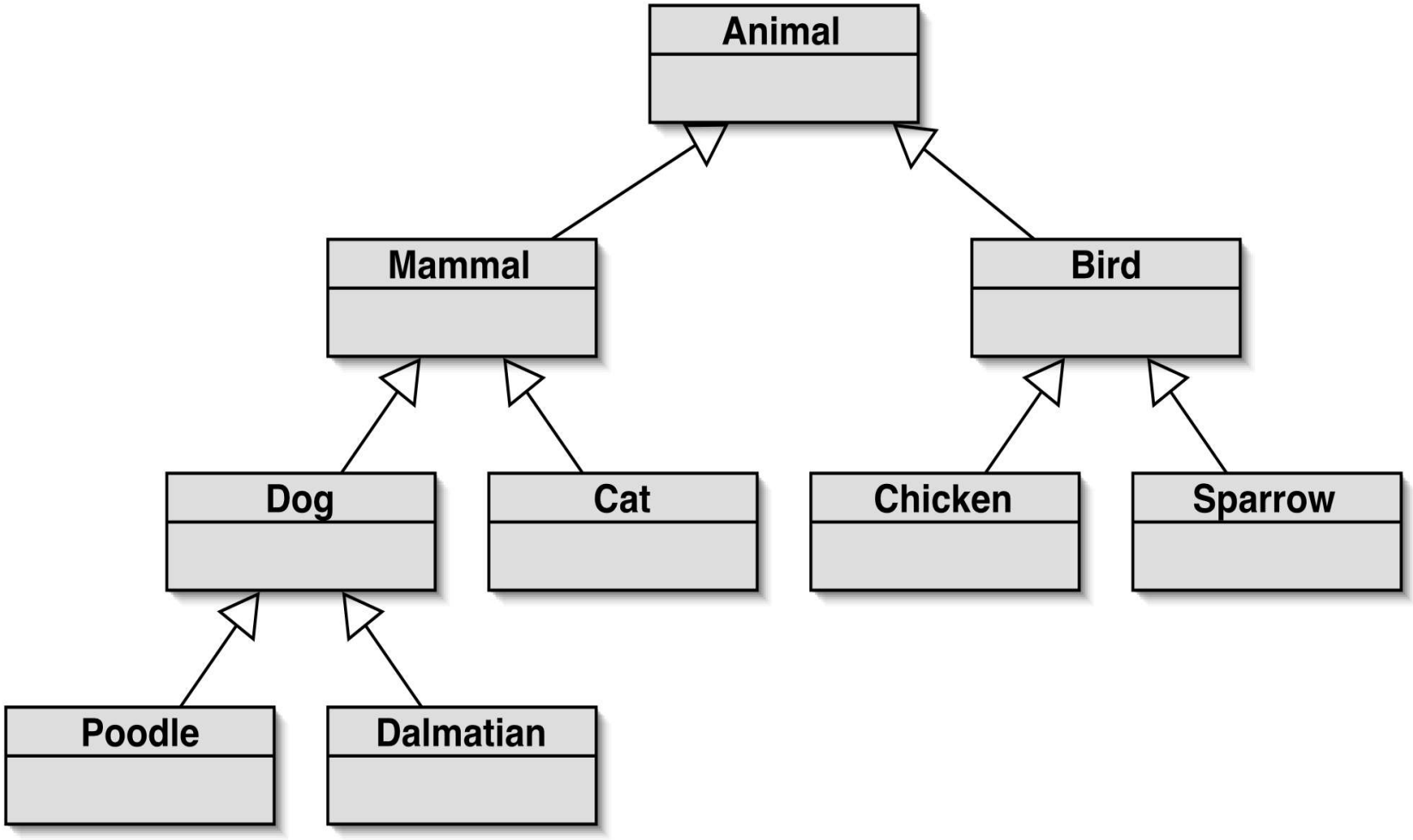
Utiliser l'héritage (1)



Utiliser l'héritage (2)

- Définir une superclasse : Item
- Définir les sous-classes Video et CD
- La superclasse définit les attributs communs
- Les sous-classes héritent des attributs de la superclasse
- Les sous-classes ajoutent leurs propres attributs

Hiérarchies d'héritage



L'héritage en Java

```
public class Item
{
    ...
}
```

Pas de changement ici

changement ici

```
public class Video extends Item
{
    ...
}
```

```
public class CD extends Item
{
    ...
}
```

Superclasse

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

    // constructeurs et méthodes
    // non reproduits.
}
```

Sous-classes

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

    // constructeurs et méthodes
    // non reproduits.
}
```

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

    // constructeurs et méthodes
    // non reproduits.
}
```

Héritage et constructeurs (1)

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

    /**
     * Initialise les champs de l'item.
     */
    public Item(String theTitle, int time)
    {
        title = theTitle;
        playingTime = time;
        gotIt = false;
        comment = "";
    }

    // méthodes non reproduites
}
```

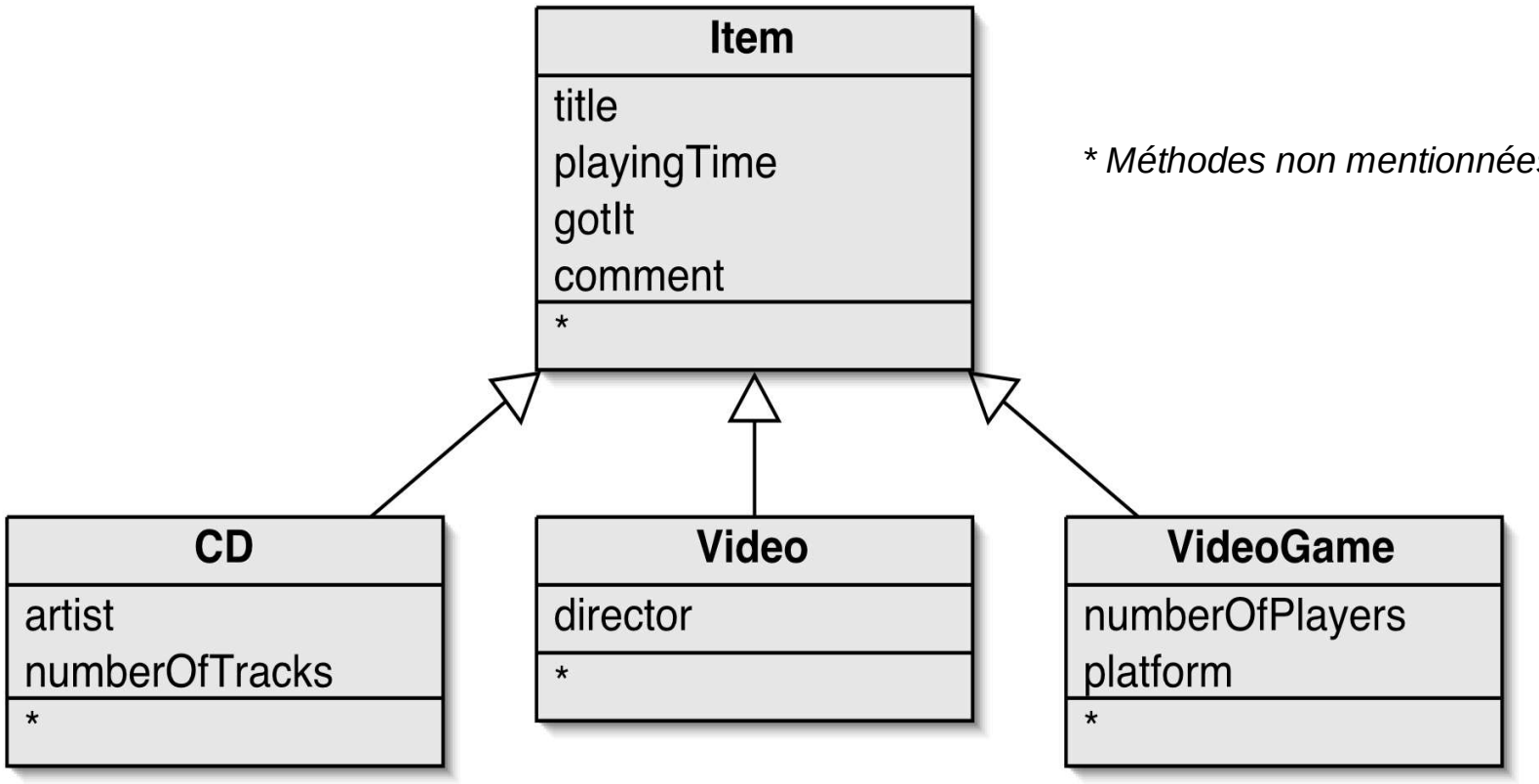
Héritage et constructeurs (2)

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

    /**
     * Constructeur d'objets de la classe CD
     */
    public CD(String theTitle, String theArtist,
              int tracks, int time)
    {
        super(theTitle, time);
        artist = theArtist;
        numberOfTracks = tracks;
    }

    // méthodes non reproduites
}
```

Appel du constructeur de la superclasse

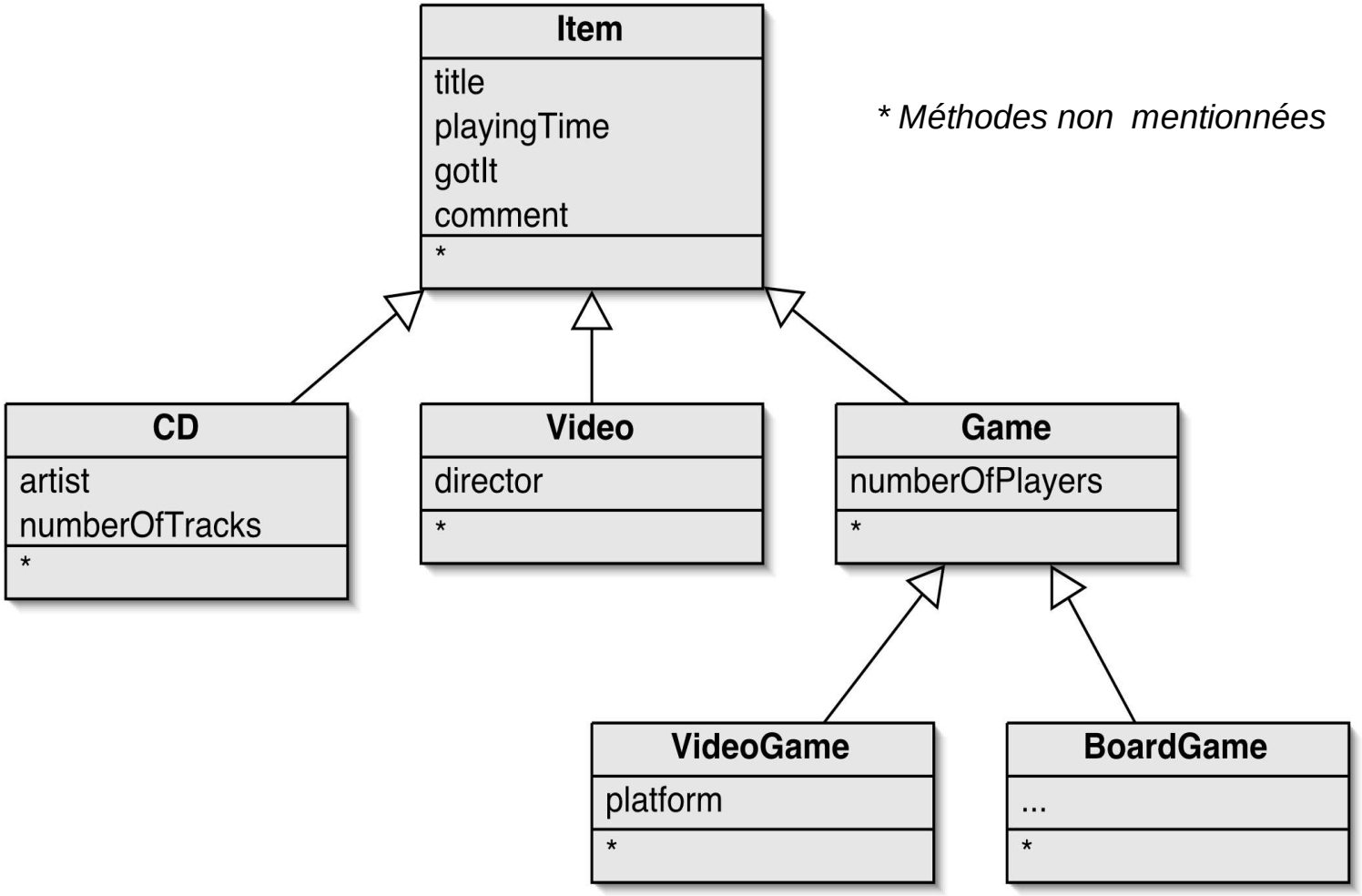
- le constructeur de la sous-classe doit toujours comporter un appel **`super`**.
- S'il n'y en a pas le compilateur en ajoute un (sans paramètre)
  - correct seulement si la superclasse possède un constructeur sans paramètre
- Doit être la première instruction du constructeur de la sous-classe.

Ajouter de nouveaux types d'items



** Méthodes non mentionnées*

Hiérarchies à plusieurs niveaux



Résumé (intermédiaire)

L'héritage permet:

- de supprimer la duplication de code
- de réutiliser du code
- une maintenance plus aisée
- de gérer les extensions

```
public class Database
{
private ArrayList items;

/**
* Construit une base vide.
*/
public Database()
{
items = new ArrayList();
}

/**
* Ajoute un item à la base.
*/
public void addItem(Item theItem)
{
items.add(theItem);
}
...
}
```

Database: nouveau code source (1)

élimine la duplication de code dans le client!

Database: nouveau code source (2)

```
/**
 * Affiche sur le terminal texte, la liste de tous
 * les CDs et vidéos actuellement stockés.
 */
public void list()
{
    for(Iterator iter = items.iterator(); iter.hasNext(); ) {
        Item item = (Item)iter.next();
        item.print();
        System.out.println();   // ligne vide entre items
    }
}
```

Sous-typage

Avant:

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

Maintenant:

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

Nous appelons cette méthode par:

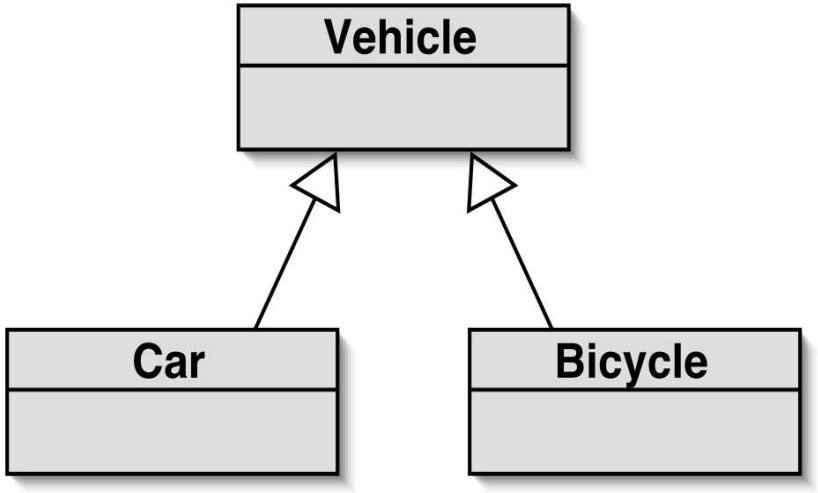
```
Video myVideo = new Video(...);
database.addItem(myVideo);
```

Sous-classes et sous-typage

- Les classes définissent des types.
- Les sous-classes définissent des sous-types.
- Des objets des sous-classes peuvent être utilisés en lieu et place des objets du supertype:

  c'est la substitution

Sous-typage et affectation



Des objets d'une sous-classe peuvent être affectés à une variable de superclasse

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

Sous-typage et passage de paramètre

```
public class Database
{

    public void addItem(Item theItem)
    {
        ...
    }
}

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

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

Des objets d'une sous-classe peuvent être passés comme paramètre de superclasse

Diagramme d'objets

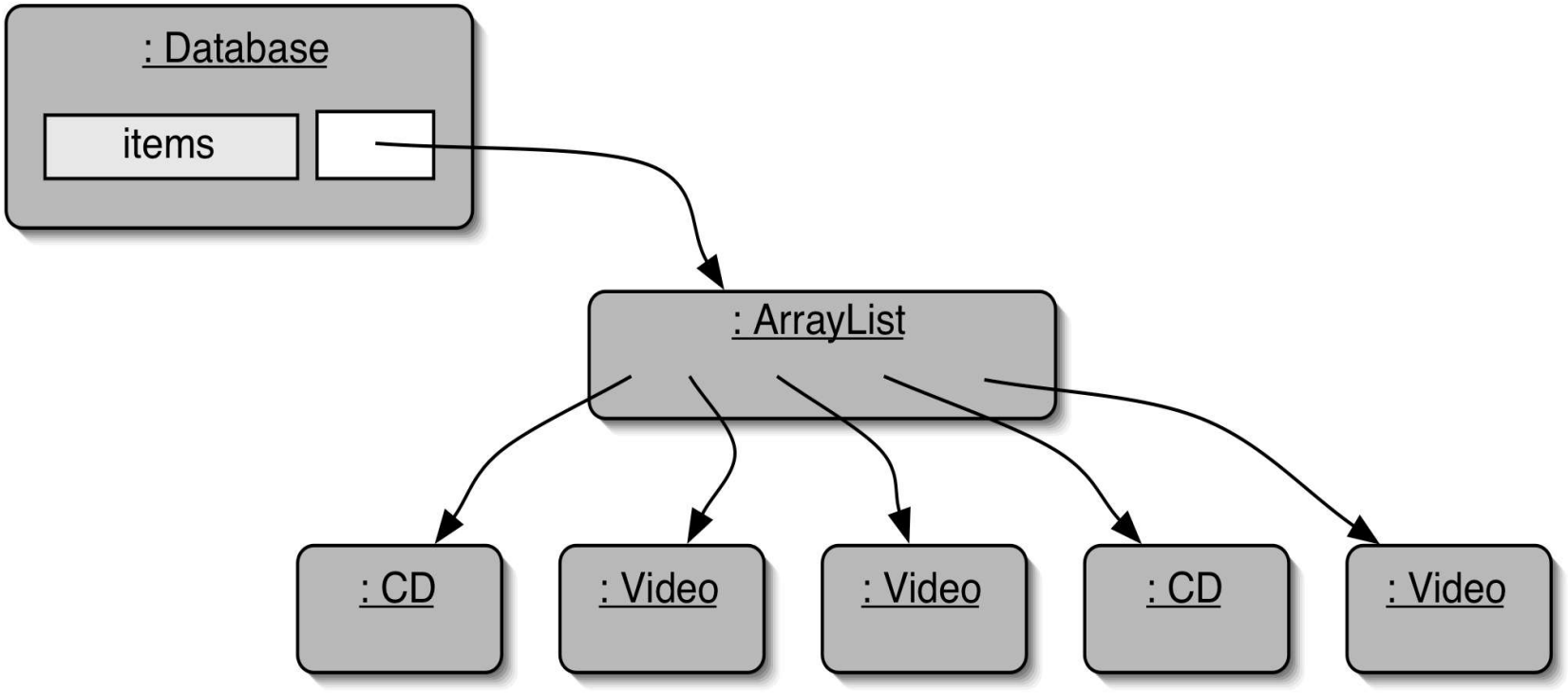
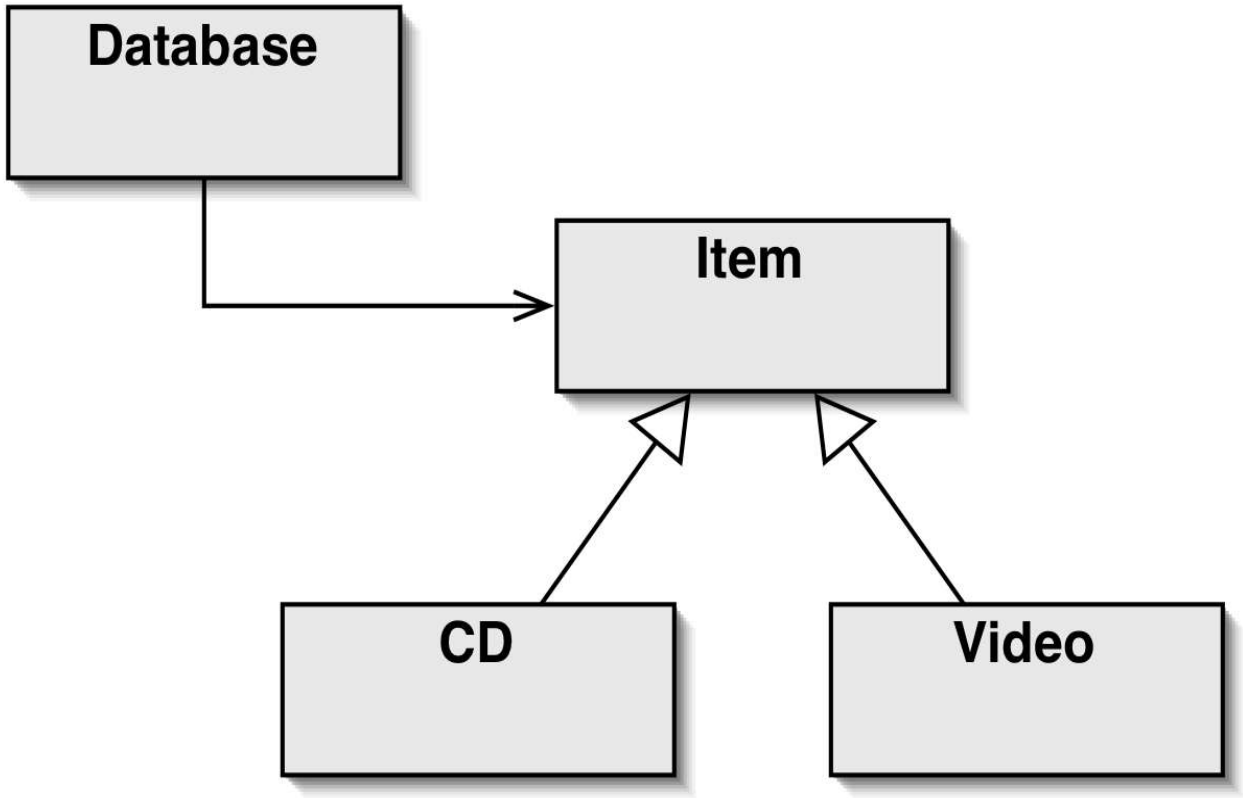


Diagramme de classes



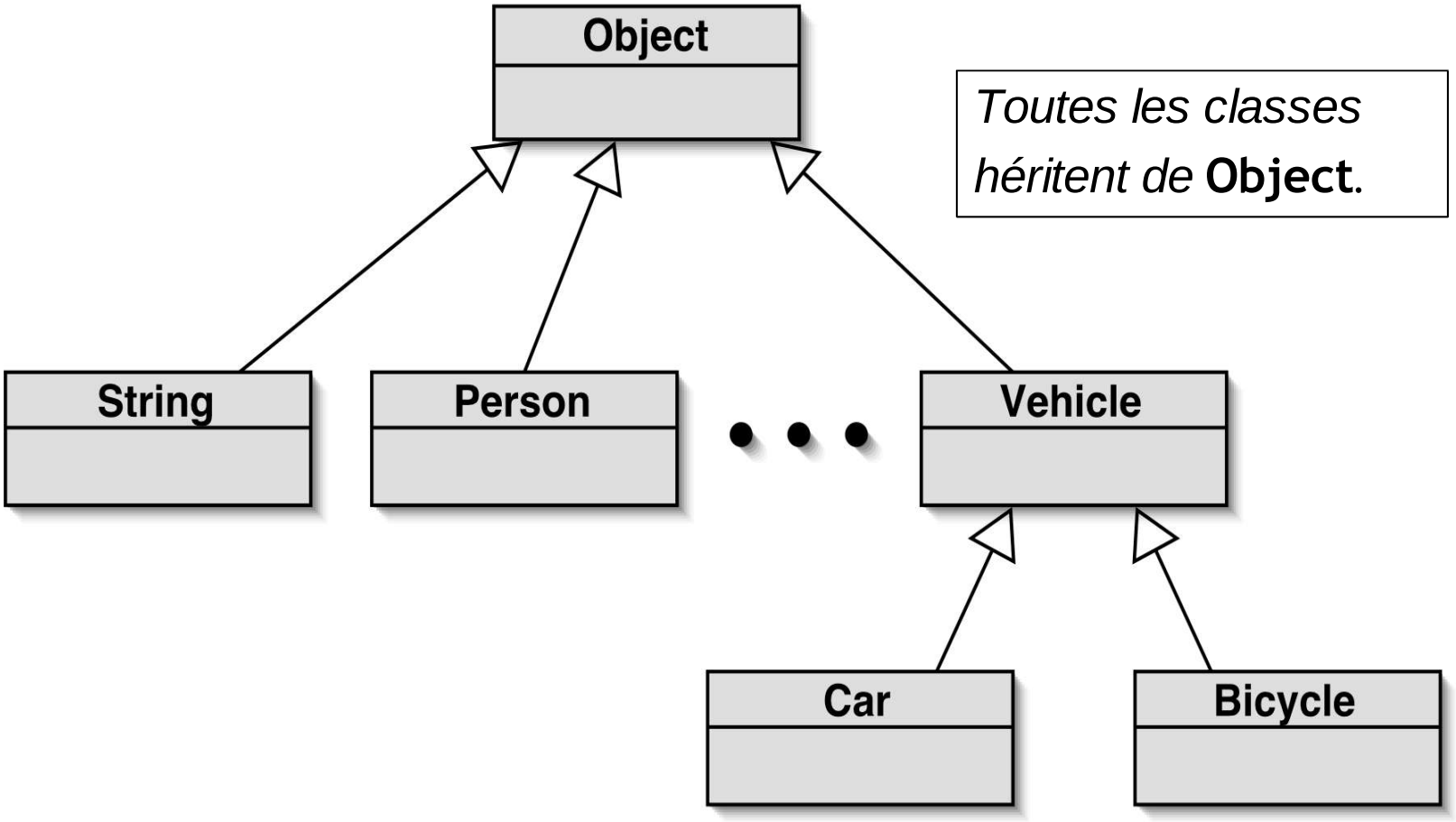
Variables polymorphes

- Les variables objets en Java sont polymorphes:

  elles peuvent contenir des objets de plusieurs types.

- Elles peuvent contenir des objets du type déclaré ou d'un de ses sous-types.

La classe `Object`



Toutes les classes héritent de **Object**.

Collections polymorphes

- Toutes les collections sont polymorphes.
- Les éléments sont de type **`Object`**.

```
public void add(Object element)

public Object get(int index)
```

Retour sur le transtypage

- On peut affecter un objet d'un sous-type à une variable d'un supertype.
- On ne peut pas affecter un objet d'un supertype à une variable d'un sous-type!

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

- Utiliser pour cela le transtypage:

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

(seulement si l'élément est vraiment de type **`String`**!)

Classes enveloppes (1)

- Tous les objets peuvent être stockés dans une collection...
- ...puisque les collections acceptent des éléments de type **`Object`**...
- ...et que toutes les classes sont des sous-types de **`Object`**.
- Parfait! Mais qu'en est-il pour les types simples?

Classes enveloppes (2)

- Les types simples (**`int`**, **`char`**, *etc*) ne sont pas des objets. Il faut les “envelopper" dans des objets!
- Il existe des classes enveloppes pour les types simples:

simple type	*wrapper class*
`Int`	**`Integer`**
`float`	**`Float`**
`Char`	**`Character`**
...	...

Classes enveloppes (3)

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

myCollecton.add(iwrap);
...

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

enveloppe la valeur **int**

ajoute l'enveloppe

extrait l'enveloppe

supprime l'enveloppe

Résumé

- L'héritage permet la définition de classes comme extensions d'autres classes.
- L'héritage
  - élimine la duplication de code
  - Permet la réutilisation de code
  - simplifie le code
  - simplifie la maintenance et les extensions
- Les variables peuvent contenir des objets de sous-types.
- Les sous-types peuvent être utilisés partout où un supertype est attendu (substitution).

Sommaire général