Objects First With Java
A Practical Introduction Using BlueJ

Improving structure with inheritance

Main concepts to be covered

- Inheritance
- Subtyping
- Substitution
- Polymorphic variables

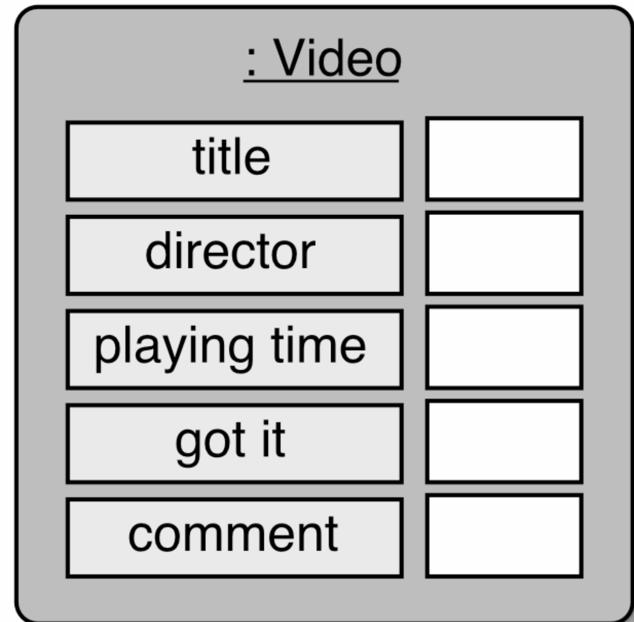
➔key concepts of object orientation!

The DoME example

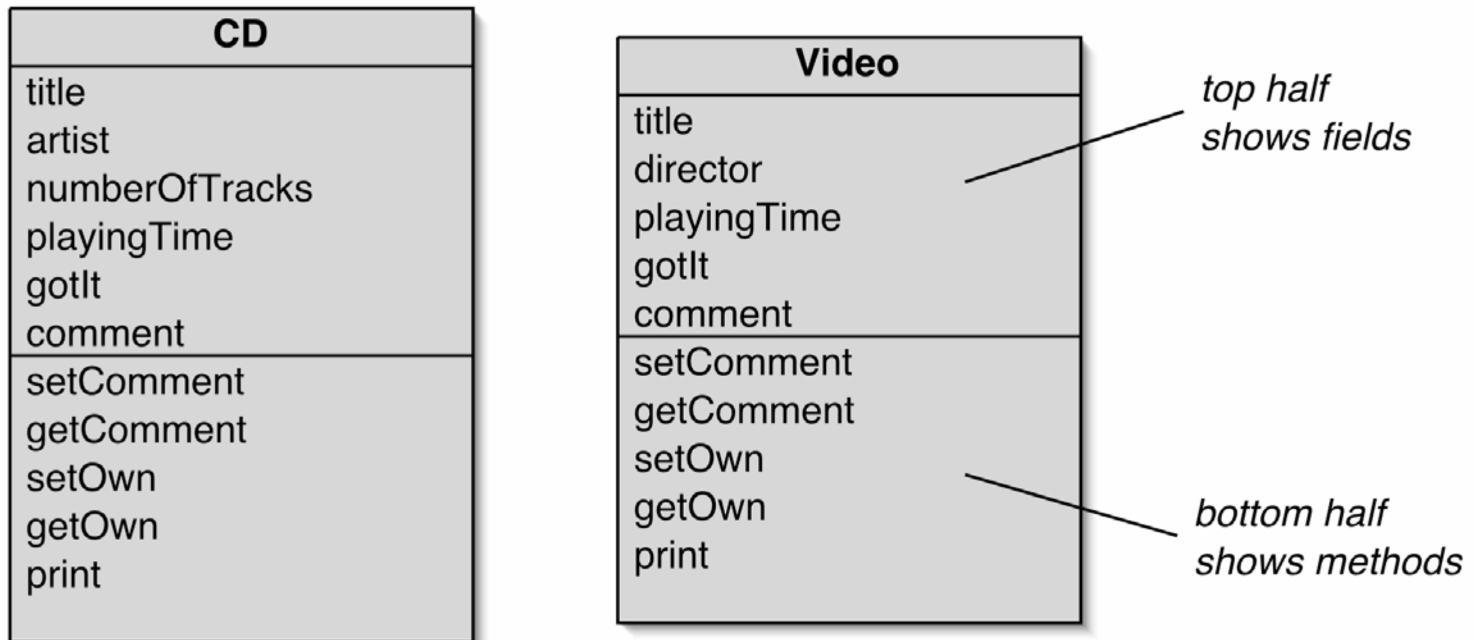
"Database of Multimedia Entertainment"

- stores details about CDs and videos
  - CD: title, artist, # tracks, playing time, got-it, comment
  - Video: title, director, playing time, got-it, comment
- allows (later) to make additions or to search for information or to print lists

DoME objects



DoME classes

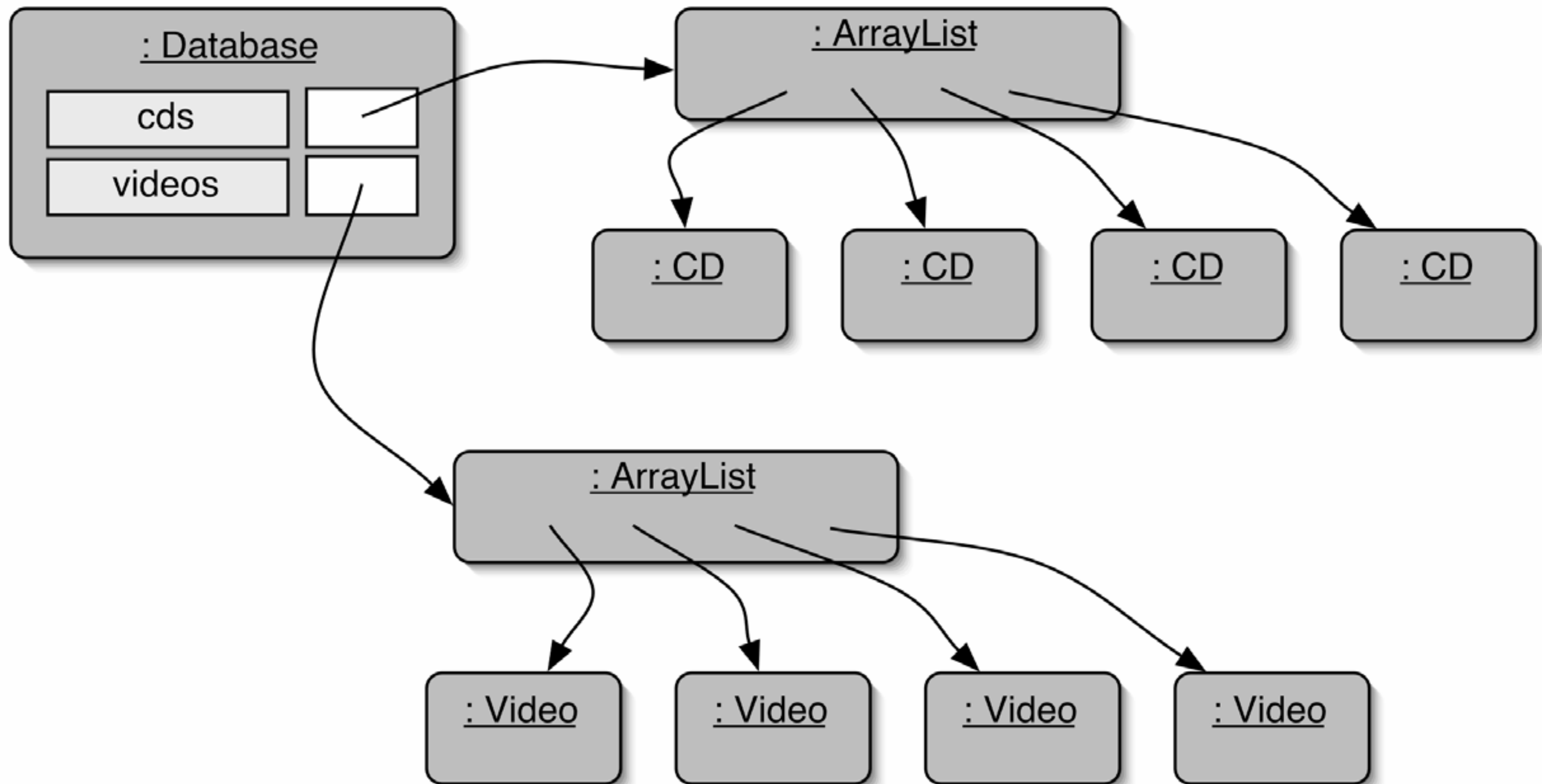


accessor and mutator methods for varying fields (`gotIt, comment`)

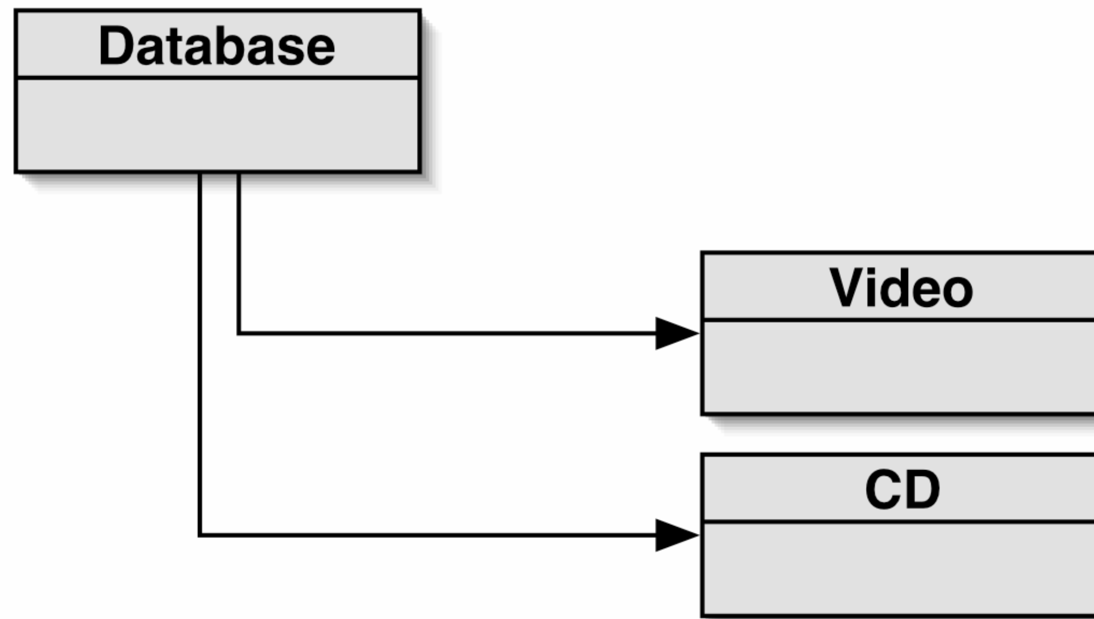
the other fields are set in the constructor

DoME object model

now the database object, holding two collection objects



Class diagram



No classes from the Java standard library are shown!

CD source code

[incomplete (comments!)]

```
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 source code

[incomplete (comments!)]

very similar to CD!!

```
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 source code

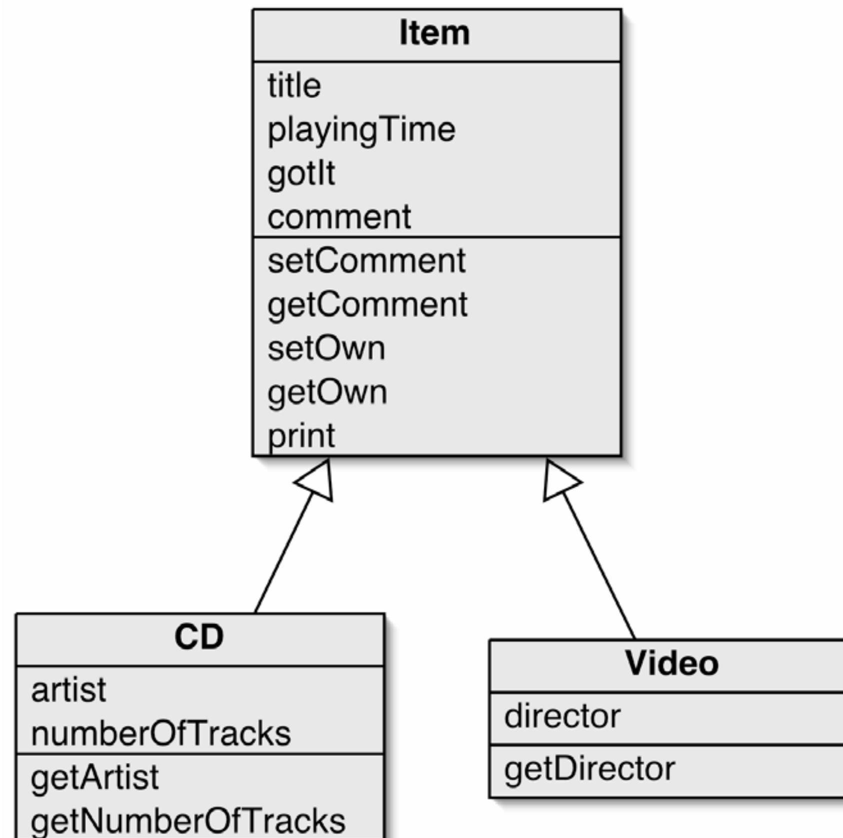
```
class Database {

   private ArrayList cds;
   private ArrayList videos;
   ...

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

      for(Iterator iter = videos.iterator(); iter.hasNext(); ) {
          Video video = (Video)iter.next();
          video.print();
          System.out.println();   // empty line between items
      }
   }
}
```

prints a list of all CDs and videos

Critique of DoME

- code duplication
  - CD and Video classes very similar (large parts are identical)
  - makes maintenance difficult/more work
  - introduces danger of bugs through incorrect maintenance
- code duplication also in Database class
  - Imagine a third media “DVD” – what has to be done?

Using inheritance

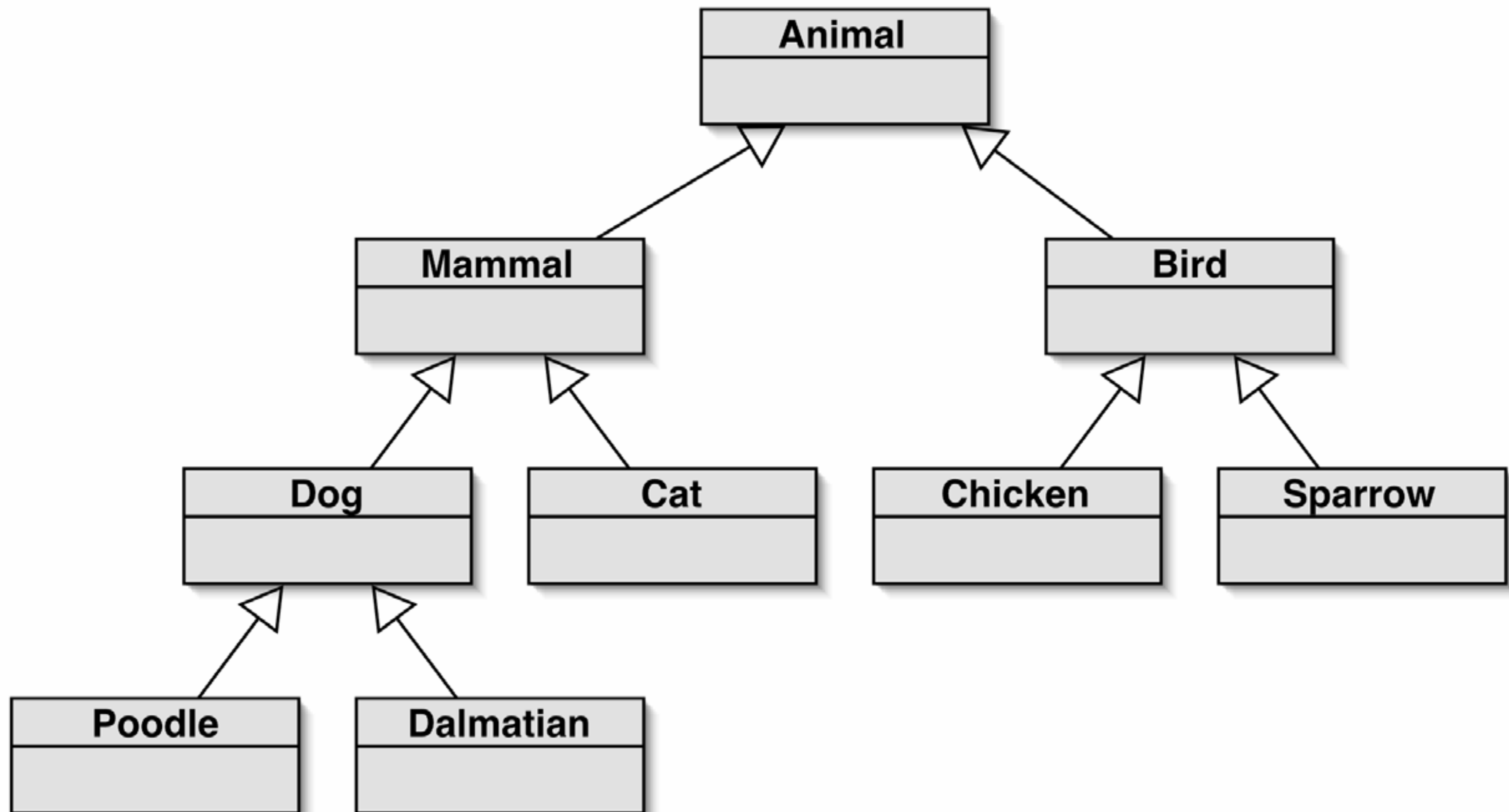


solve the problem of duplication!

Using inheritance

- define one **superclass** : Item
- define **subclasses** for Video and CD
- the superclass defines common attributes
- the subclasses **inherit from** or **extend** the superclass
- the subclasses **inherit** the superclass attributes
- the subclasses add their own attributes

Inheritance hierarchies



Inheritance in Java

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

no change here

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

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

change here

Superclass

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

    // constructors and methods omitted.
}
```

object generation possible, but not intended!

Subclasses

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

    // constructors and methods omitted.
}
```

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

    // constructors and methods omitted.
}
```

Inheritance and constructors

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

    /**
     * Initialise the fields of the item.
     */
    public Item(String theTitle, int time)
    {
        title = theTitle;
        playingTime = time;
        gotIt = false;
        comment = "";
    }

    // methods omitted
}
```

Inheritance and constructors

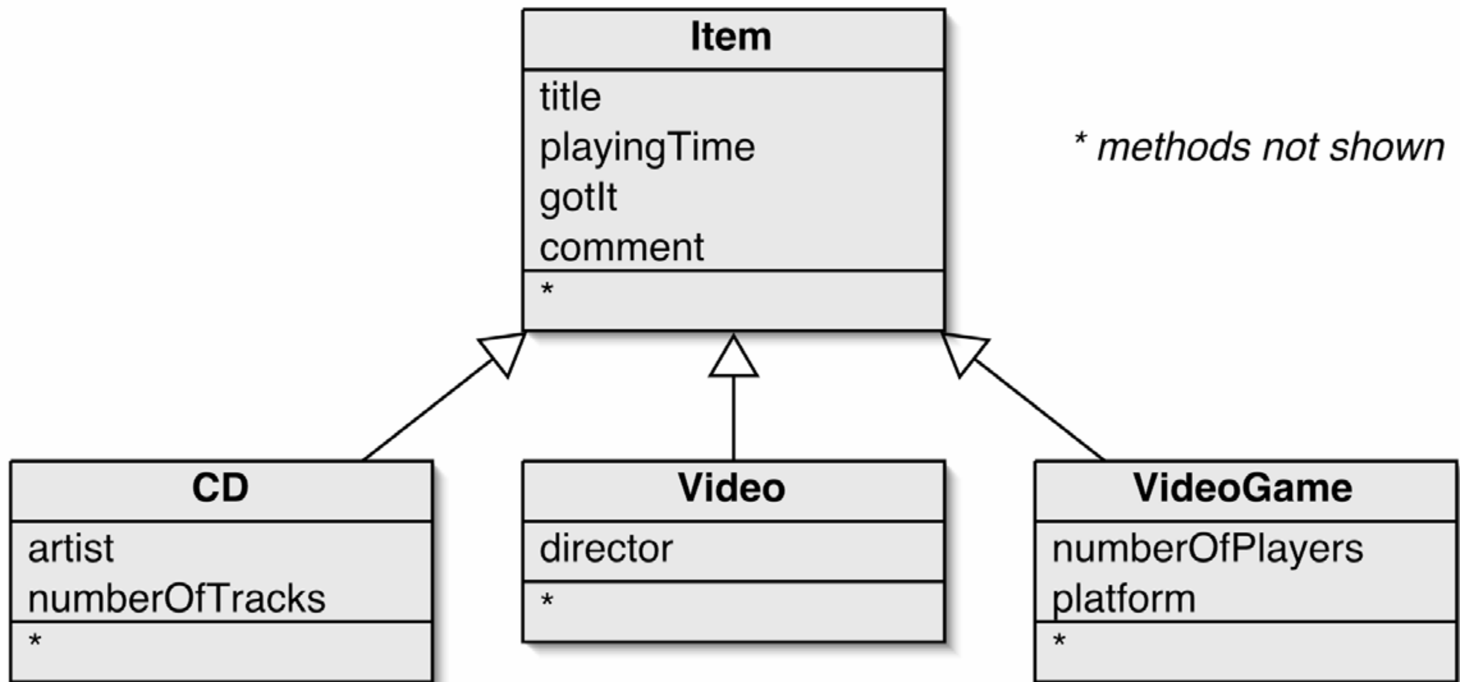
```
public class CD extends Item
{
    private String artist;
    private int numberOfTracks;

    /**
     * Constructor for objects of class CD
     */
    public CD(String theTitle, String theArtist,
              int tracks, int time)
    {
        super(theTitle, time);
        artist = theArtist;
        numberOfTracks = tracks;
    }

    // methods omitted
}
```

privacy also applies between subclasses and their superclass

Superclass constructor call

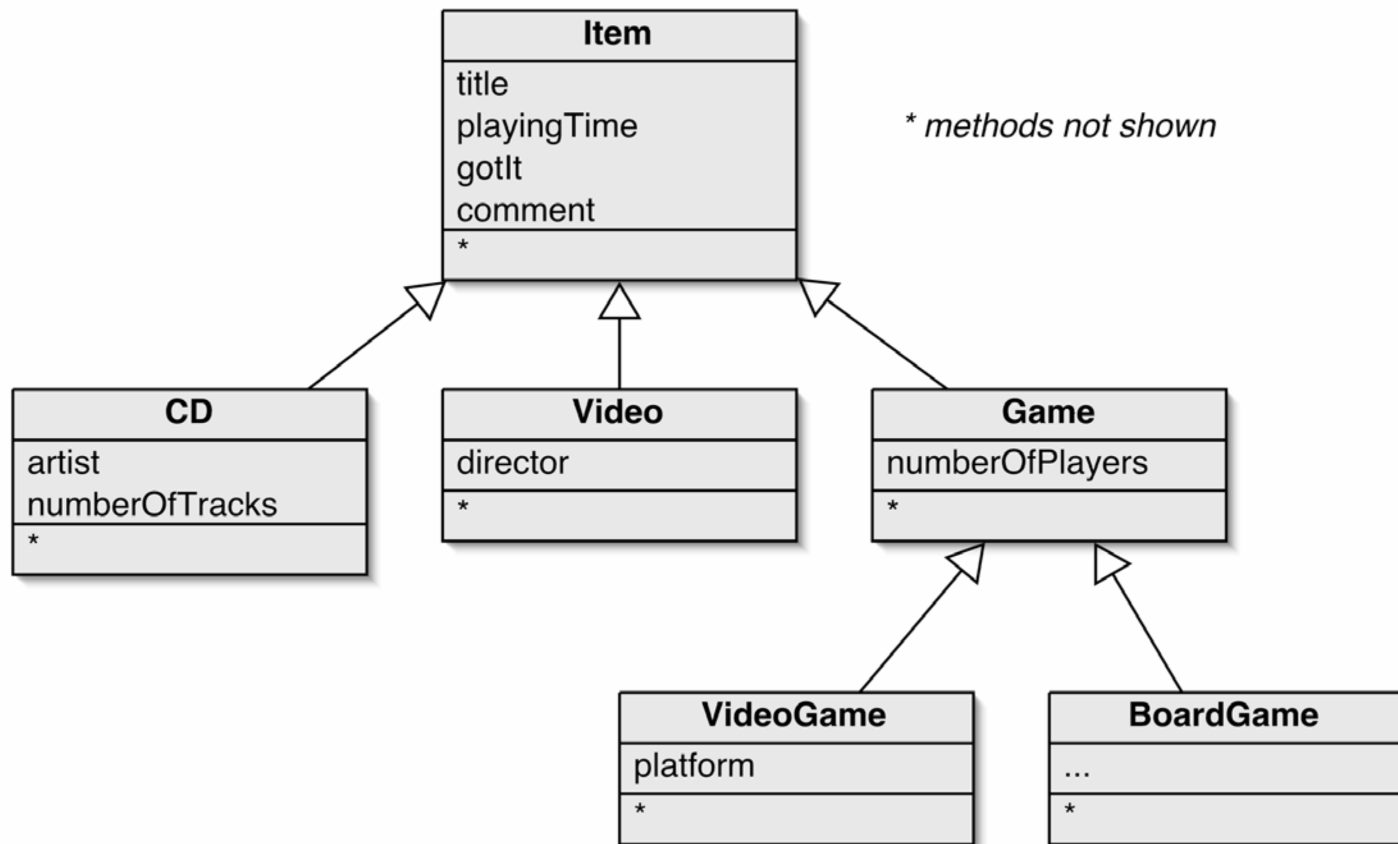
- Subclass constructors must always contain a 'super' call.
- If none is written, the compiler inserts one (without parameters)
  - works only, if the superclass has a constructor without parameters
- Must be the first statement in the subclass constructor.

Adding more item types



example of code reuse!

Deeper hierarchies



Review (so far)

Inheritance (so far) helps with:

- Avoiding code duplication
- Code reuse
- Easier maintenance
- Extendibility

New Database source code

```
public class Database
{
    private ArrayList items;

    /**
     * Construct an empty Database.
     */
    public Database()
    {
        items = new ArrayList();
    }

    /**
     * Add an item to the database.
     */
    public void addItem(Item theItem)
    {
        items.add(theItem);
    }
    ...
}
```

avoids code duplication in client!

New Database source code

```
/**
 * Print a list of all currently stored CDs and
 * videos to the text terminal.
 */
public void list()
{
   for(Iterator iter = items.iterator(); iter.hasNext(); ) {
       Item item = (Item)iter.next();
       item.print();
       System.out.println();   // empty line between items
   }
}
```

Subtyping

First, we had:

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

Now, we have:

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

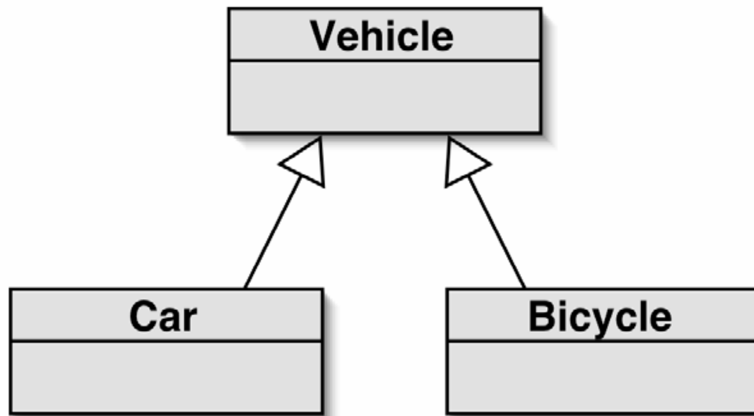
We call this method with:

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

Subclasses and subtyping

- Classes define types.
- Subclasses define subtypes.
- Objects of subclasses can be used where objects of supertypes are required.
  (This is called **substitution**.)

Subtyping and assignment



subclass objects may be assigned to superclass variables – but not the other way round!

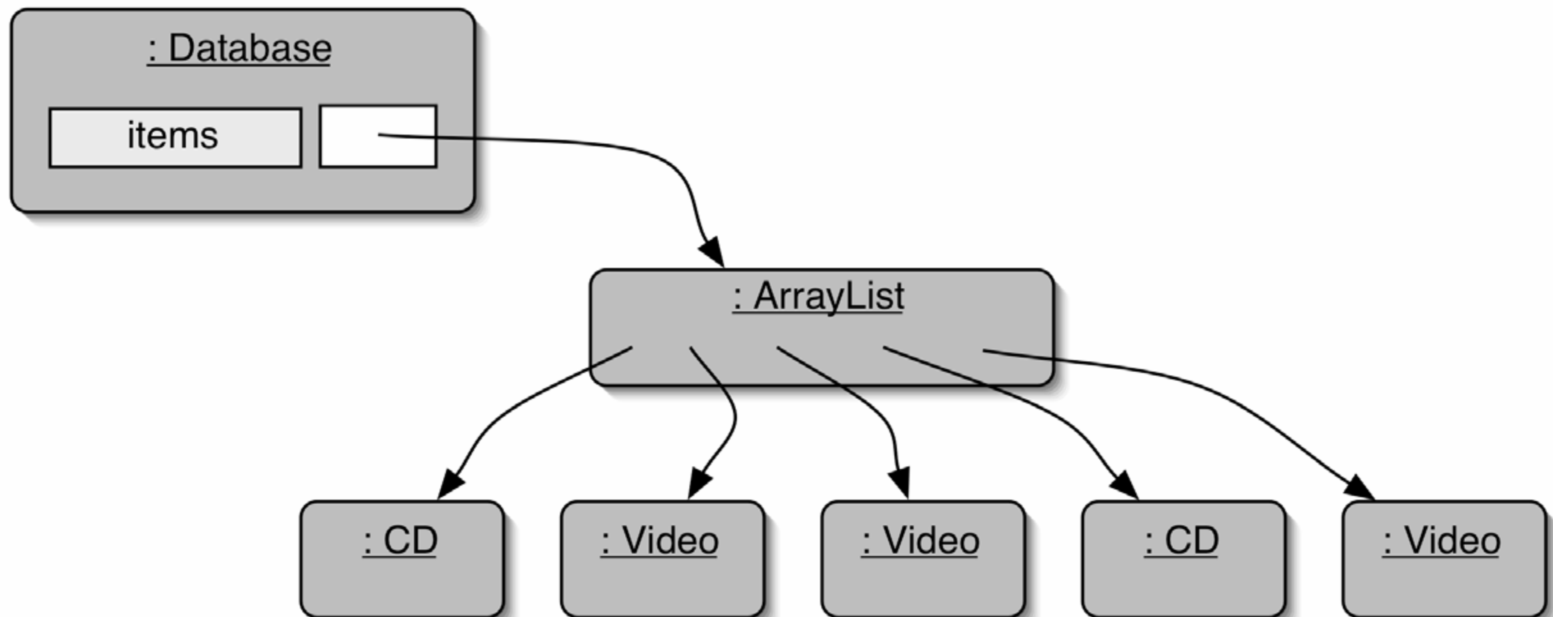
```
Vehicle v1 = new Vehicle();
Vehicle v2 = new Car();
Vehicle v3 = new Bicycle();
```

Subtyping and parameter passing

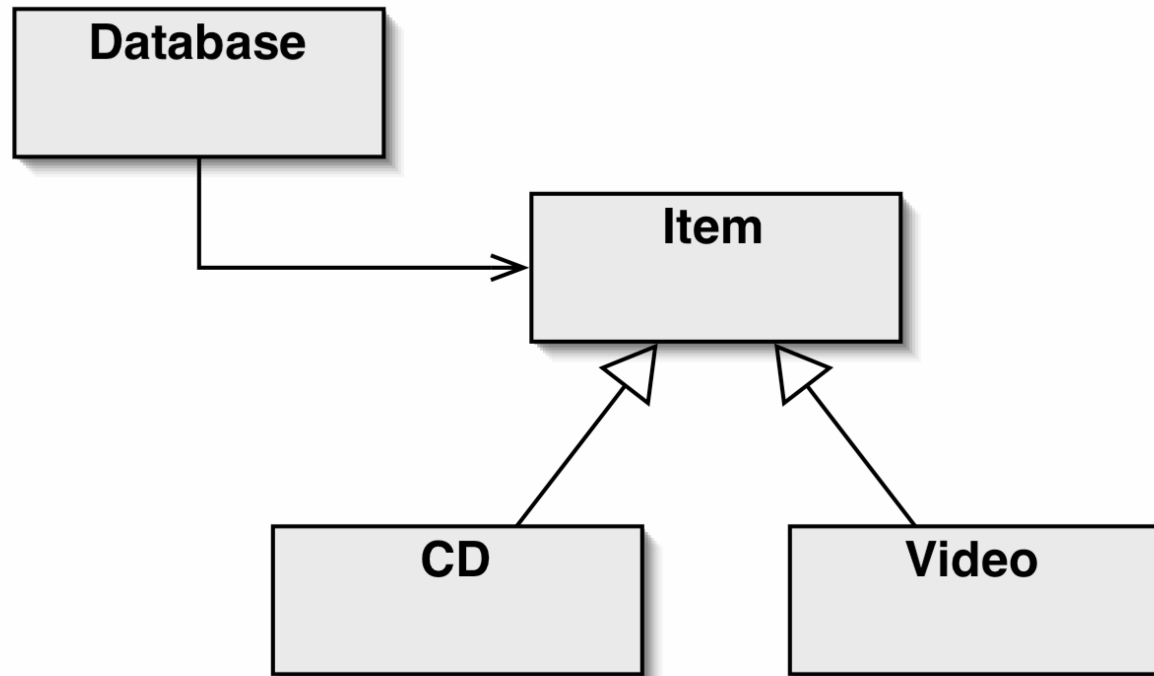
```
public class Database
{

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

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

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

subclass objects may be passed to superclass parameters – but not the other way round!

Object diagram



Class diagram



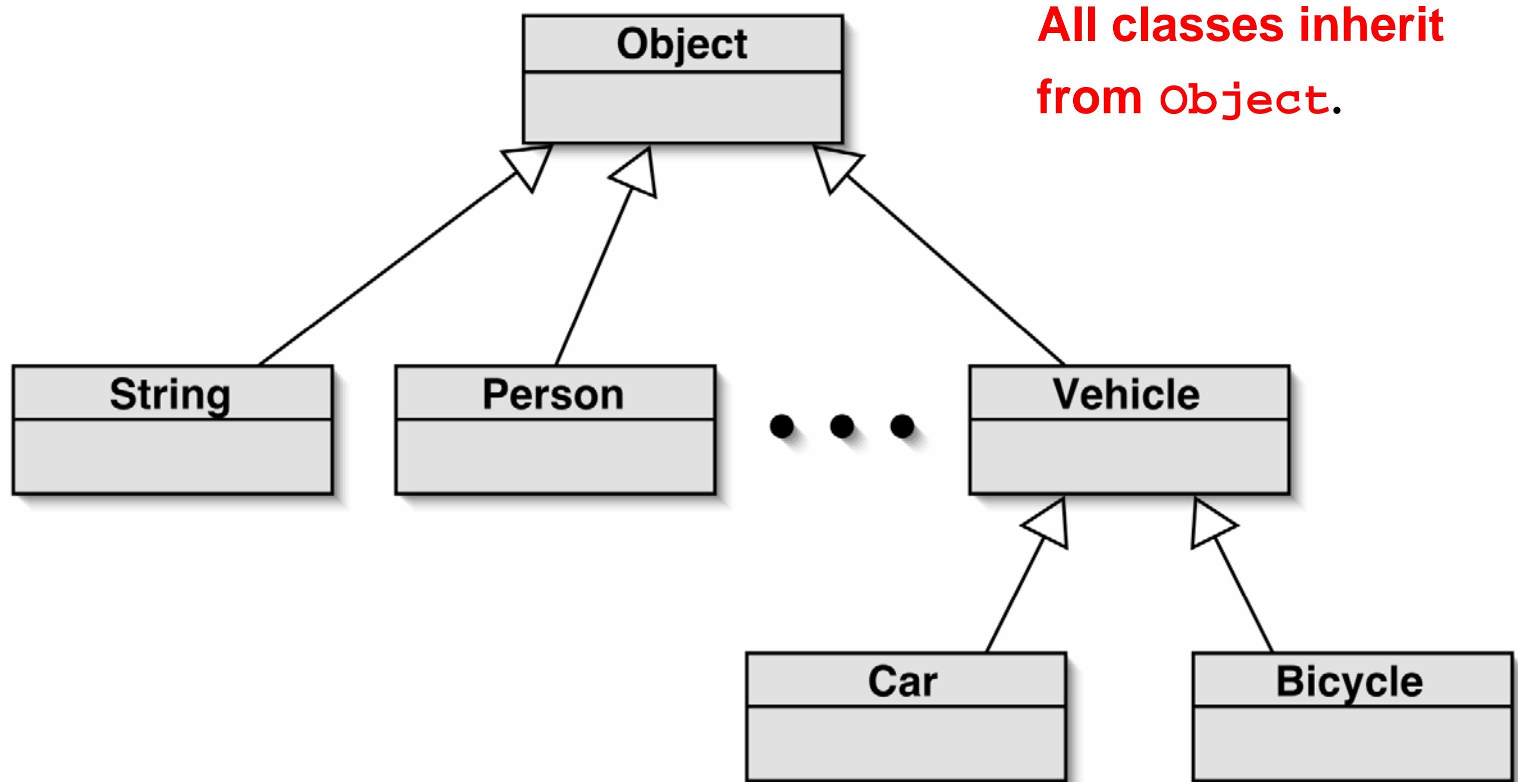
Polymorphic variables

- `Object:` a superclass for all objects
- `Object` variables in Java are **polymorphic**.

  (They can hold objects of more than one type.)

- They can hold objects of the declared type, or of subtypes of the declared type.

The Object class



Polymorphic collections

- All collections are polymorphic.
- The elements are of type `Object`.

```
public void add(Object element)

public Object get(int index)
```

Casting revisited

- Can assign subtype to supertype.
- Cannot assign supertype to subtype!

```
Video v1 = myList.get(1);
```
error!

- Casting fixes this:

```
Video v1 = (Video) myList.get(1);
```

(only if the element really is a Video – otherwise error & stop!)

Wrapper classes

- All objects can be entered into collections...
- ...because collections accept elements of type Object...
- ...and all classes are subtypes of Object.
- Great! But what about simple types?

Wrapper classes

- Simple types (int, char, *etc*) are not objects. They must be wrapped into an object!
- Wrapper classes exist for all simple types:

simple type	*wrapper class*
int	Integer
float	Float
char	Character
...	...

Wrapper classes

wrap the int value

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

myCollecton.add(iwrap);
...

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

add the wrapper

retrieve the wrapper

unwrap

Review

- Inheritance allows the definition of classes as extensions of other classes.
- Inheritance
  - avoids code duplication
  - allows code reuse
  - simplifies the code
  - simplifies maintenance and extending
- Variables can hold subtype objects.
- Subtypes can be used wherever supertype objects are expected (substitution).