

Mejora de la estructura mediante la herencia

Principales conceptos a ser cubiertos

- Herencia
- Subtipado
- Sustitución
- Variables polimórficas

El ejemplo DoME

"Base de datos de Entretenimiento Multimedia" (DoME)

- almacena detalles acerca de CDs y DVDs
 - CD: título, artista, nro. de pistas, duración, lo tengo, comentario
 - DVD: título, director, duración, lo tengo, comentario
- permite (más tarde) buscar información o imprimir listas

Objetos DoME

: CD

title	
artist	
#tracks	
playing time	
got it	
comment	

: DVD

title	
director	
playing time	
got it	
comment	

Las clases de DoME

CD
title artist numberOfTracks playingTime gotIt comment
setComment getComment setOwn getOwn print

DVD
title director playingTime gotIt comment
setComment getComment setOwn getOwn print

top half
shows fields

bottom half
shows methods

Modelo de objetos DoME

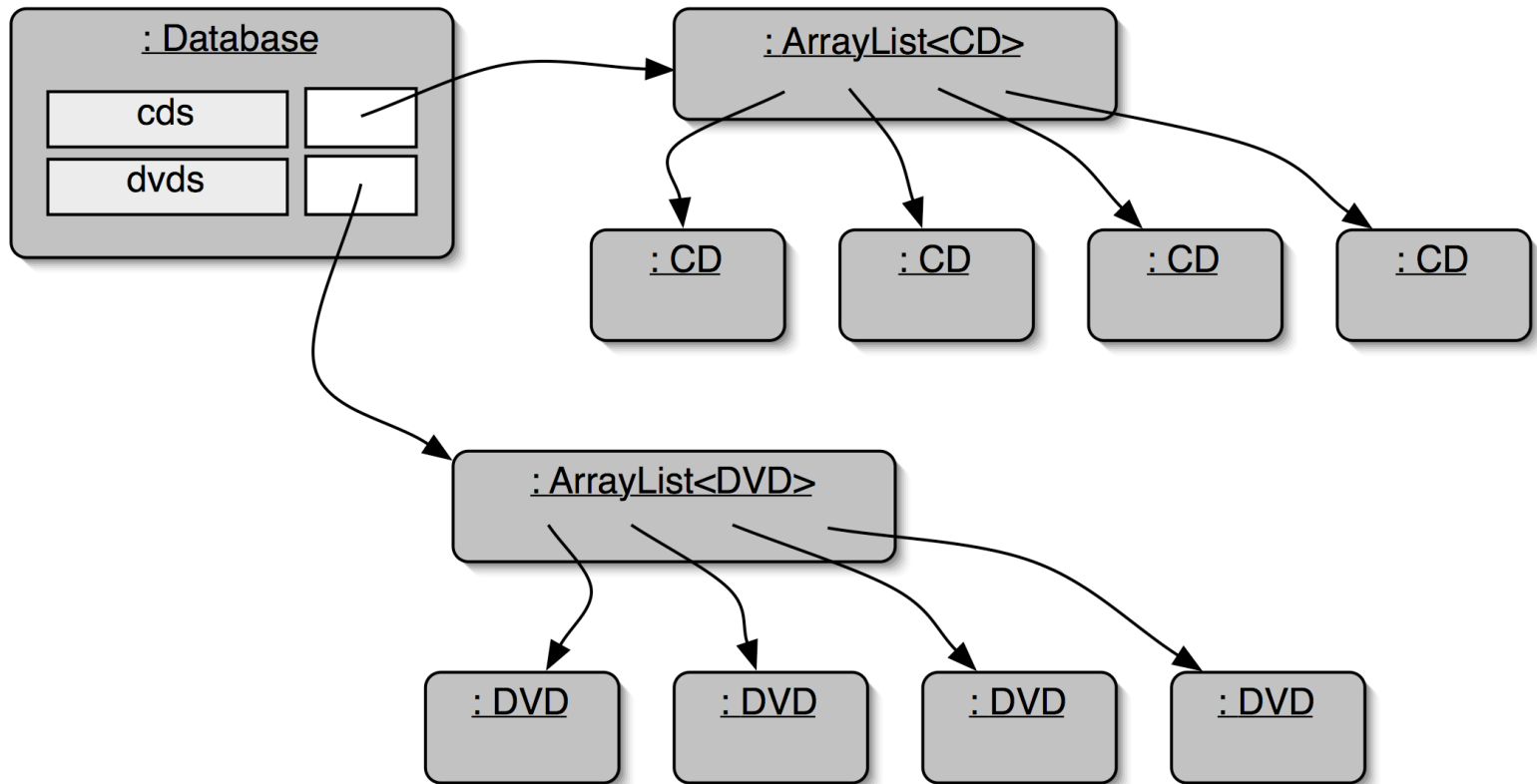
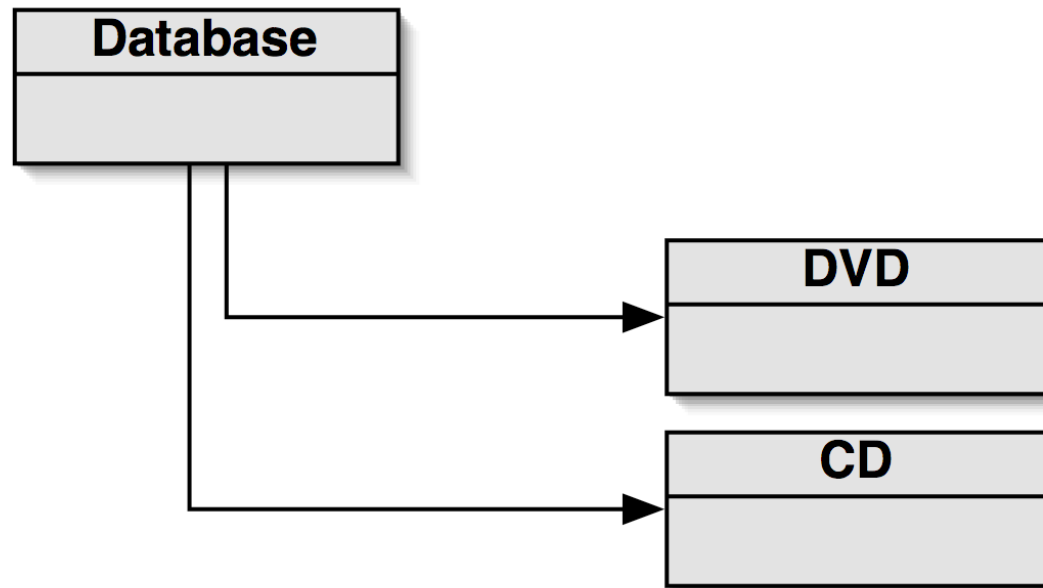


Diagrama de clases



Código fuente CD

[¡comentarios
incompletos!]

```
public class CD
{
    private String titulo;
    private String interprete;
    private String comentario;

    CD(String elTitulo, String elInterprete)
    {
        titulo = elTitulo;
        interprete = elInterprete;
        comentario = " ";
    }

    void setComentario(String comentario)
    { ... }

    String getComentario()
    { ... }

    void imprimir()
    { ... }
    ...
}
```


Código fuente DVD

[¡comentarios
incompletos!]

```
public class DVD
{
    private String titulo;
    private String director;
    private String comentario;

    DVD(String elTitulo, String elDirector)
    {
        titulo = elTitulo;
        director = elDirector;
        comentario = " ";
    }

    void setComentario(String comeentario)
    { ... }

    String getComentario()
    { ... }

    void imprimir()
    { ... }
    ...
}
```

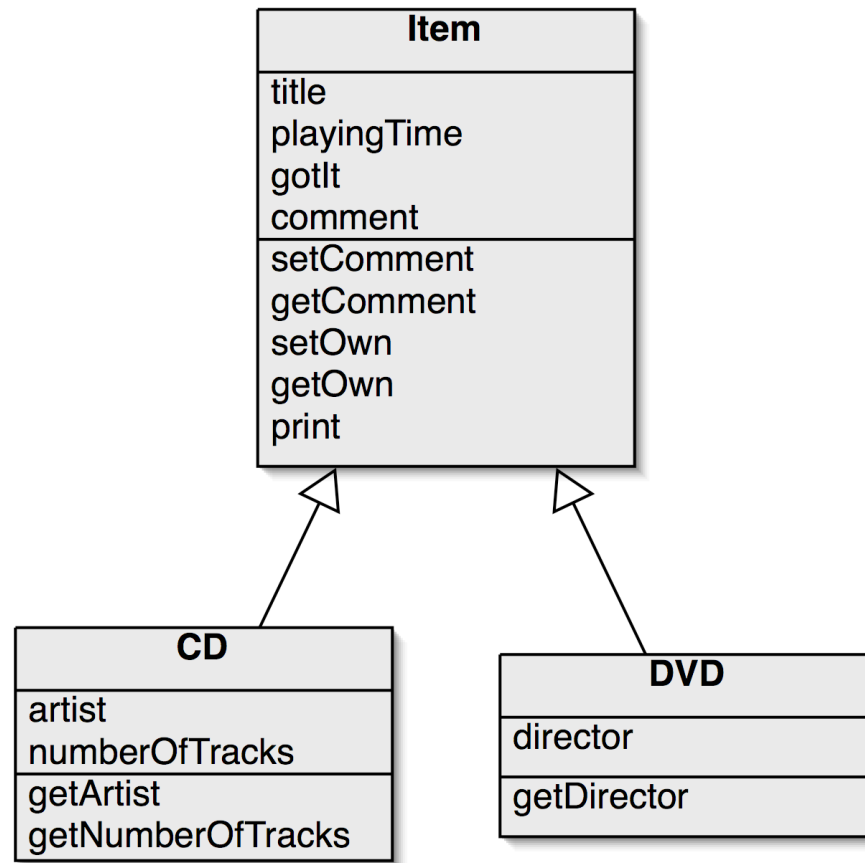
Código fuente de BaseDeDatos

```
class BaseDeDatos {  
  
    private ArrayList<CD> cds;  
    private ArrayList<DVD> dvds;  
    ...  
    public void listar()  
    {  
        for(CD cd : cds) {  
            cd.imprimir();  
            System.out.println(); // línea vacía entre items  
        }  
  
        for(DVD dvd : dvds) {  
            dvd.imprimir();  
            System.out.println(); // línea vacía entre items  
        }  
    }  
}
```

Discusión sobre DoME

- Duplicación de código
 - Las clases CD y DVD son muy similares (en gran parte son idénticas).
 - Hace difícil el mantenimiento y da más trabajo.
 - Introduce peligro de cometer errores a través del mantenimiento incorrecto.
- Duplicación de código incluso en la clase *BaseDeDatos*

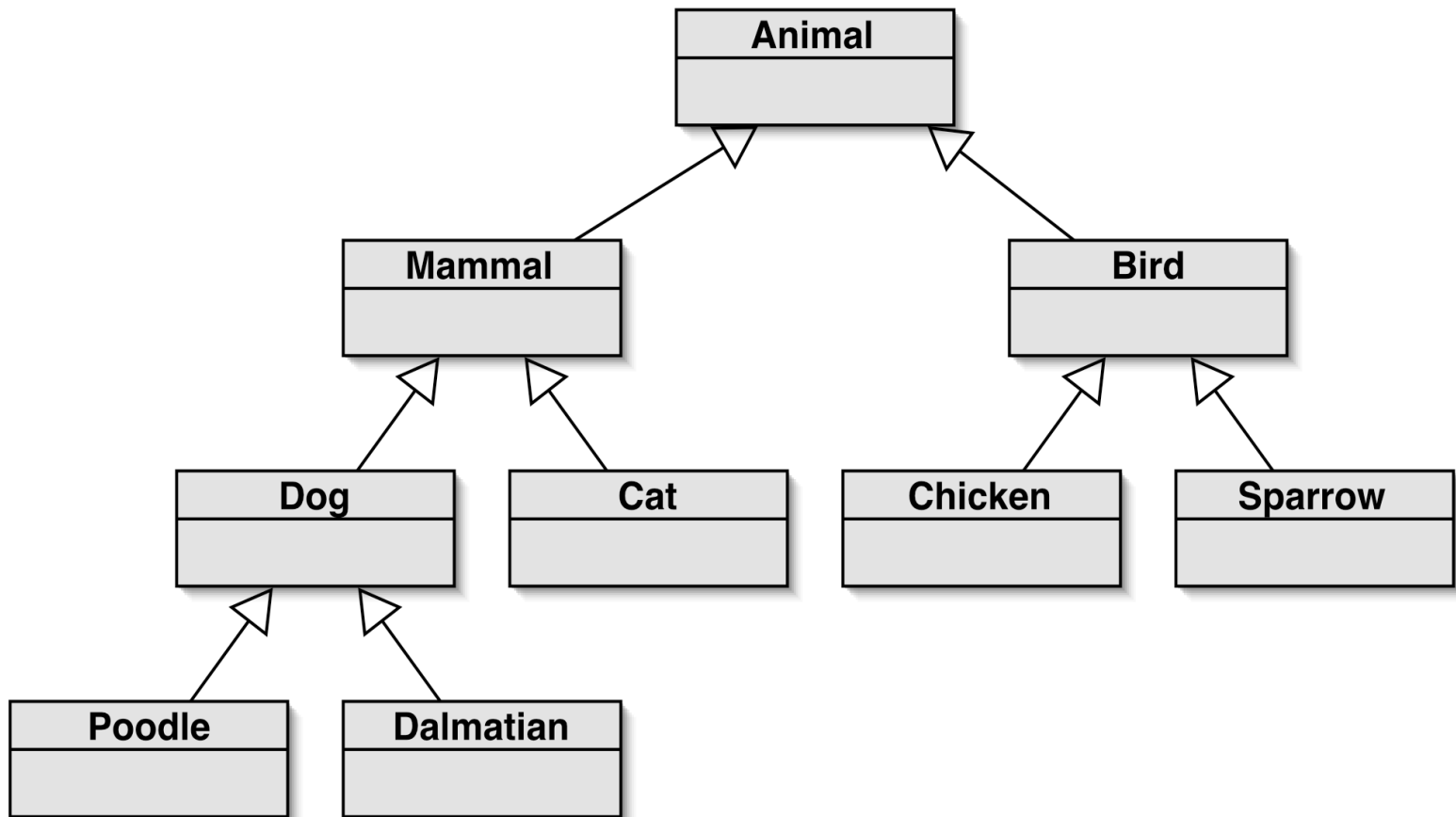
Usar herencia



Usar herencia

- defina una **superclase** : Elemento
- defina **subclases** para DVD y CD
- la superclase define atributos comunes
- las subclases **heredan** los atributos de la superclase
- las subclases agregan sus propios atributos

Jerarquías de herencia



Herencia en Java

aquí no hay cambios

```
public class Elemento
{
    ...
}
```

aquí cambió

```
public class DVD extends Elemento
{
    ...
}
```

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


Superclase

```
public class Elemento
{
    private String titulo;
    private int duracion;
    private boolean loTengo;
    private String comentario;

    // constructores y métodos omitidos.
}
```


Subclases

```
public class CD extends Elemento
{
    private String interprete;
    private int numeroDeTemas;

    // constructores y métodos omitidos.
}
```

```
public class DVD extends Elemento
{
    private String director;

    // constructores y métodos omitidos.
}
```

Herencia y constructores

```
public class Elemento
{
    private String titulo;
    private int duracion;
    private boolean loTengo;
    private String comentario;

    /**
     * Inicializa los campos del elemento.
     */
    public Elemento(String elTitulo, int tiempo)
    {
        titulo      = elTitulo;
        duracion     = tiempo;
        LoTengo      = false;
        comentario   = "";
    }

    // methods omitted
}
```

Herencia y constructores

```
public class CD extends Elemento
{
    private String interprete;
    private int numeroDeTemas;

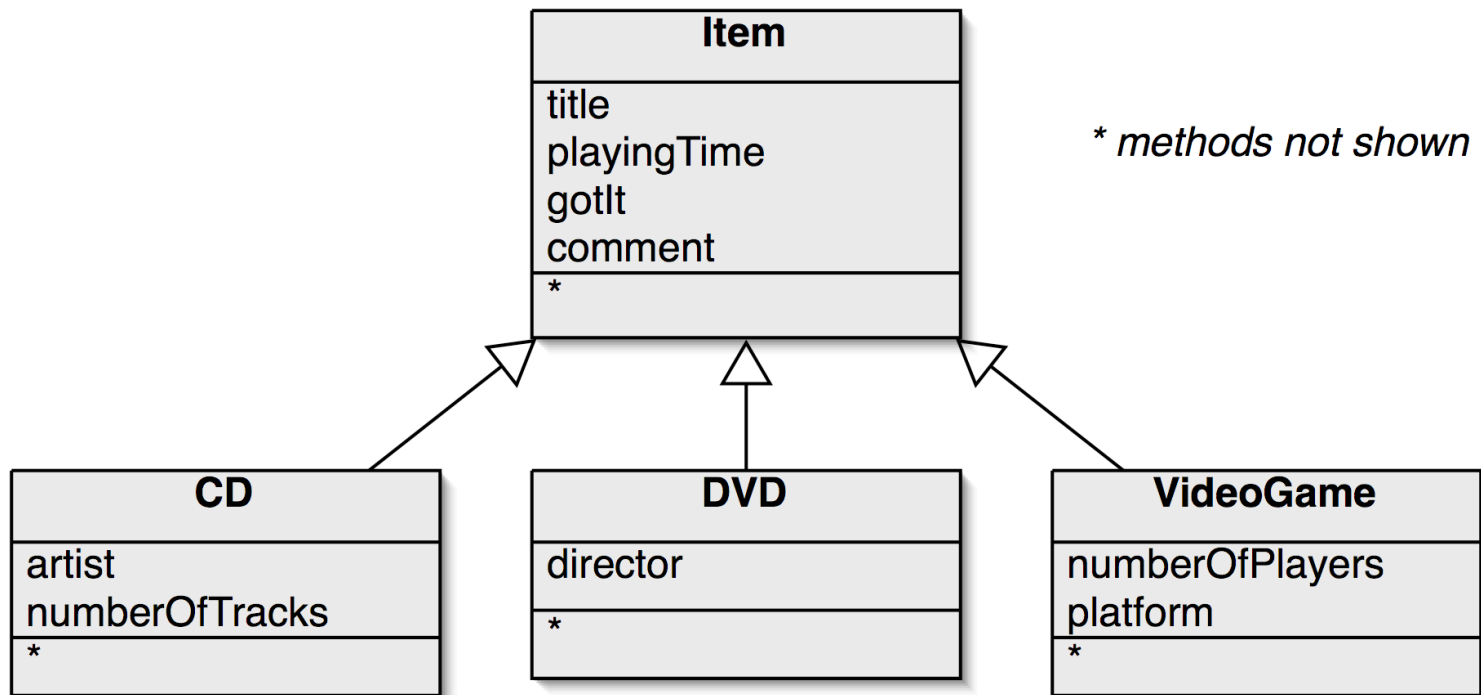
    /**
     * Constructor para objetos de la clase CD
     */
    public CD(String elTitulo,
               String el Interprete,
               int temas, int tiempo)
    {
        super(elTitulo, tiempo);
        interprete = elInterprete;
        numeroDeTemas = temas;
    }

    // métodos omitidos
}
```

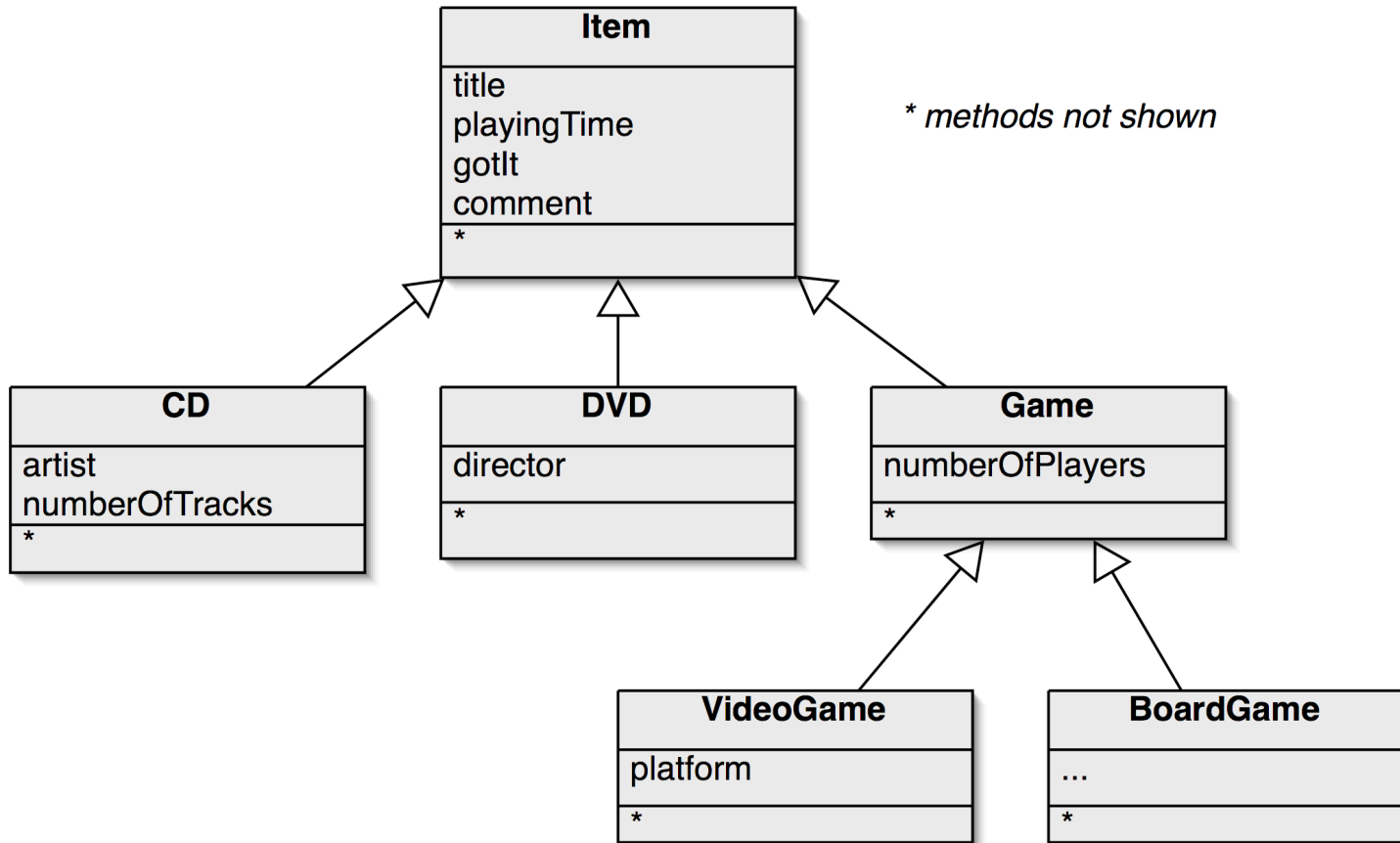
Llamado al constructor de la superclase

- Los constructores de las subclases siempre *deben* contener una llamada a **super**.
- Si no se escribe ninguna, el compilador inserta una (sin parámetros)
 - Funciona sólo si la superclase tiene un constructor sin parámetros
- Debe ser la primera instrucción en el constructor de la subclase.

Agregar otros tipos de Elementos



Jerarquías más profundas



Repaso

La herencia (hasta ahora) ayuda a:

- evitar la duplicación de código
- reusar el código
- facilitar el mantenimiento
- facilitar la extensibilidad

Código fuente de la nueva BaseDeDatos

```
public class BaseDeDatos
{
    private ArrayList<Elemento> elementos;

    /**
     * Construye una BaseDeDatos vacía.
     */
    public BaseDeDatos()
    {
        elementos = new ArrayList<Elemento>();
    }

    /**
     * Agrega un elemento a la BaseDeDatos.
     */
    public void agregarElemento(elemento elElemento)
    {
        elementos.add(elElemento);
    }
    ...
}
```

*Evita la duplicación de
código en el cliente*

Nuevo código fuente de BaseDeDatos

```
/**
 * Imprime, en la terminal de texto, una lista de
 * todos los CD y DVS actualmente almacenados.
 */
public void listar()
{
    for(Elemento elemento : elementos) {
        elemento.imprimir();
        // Imprime una línea en blanco entre elementos
        System.out.println();
    }
}
```

Subtipos

Primero, teníamos:

```
public void agregarCD (CD elCD)
public void agregarDVD (DVD elDVD)
```

Ahora tenemos:

```
public void agregarElemento (Elemento
elElemento)
```

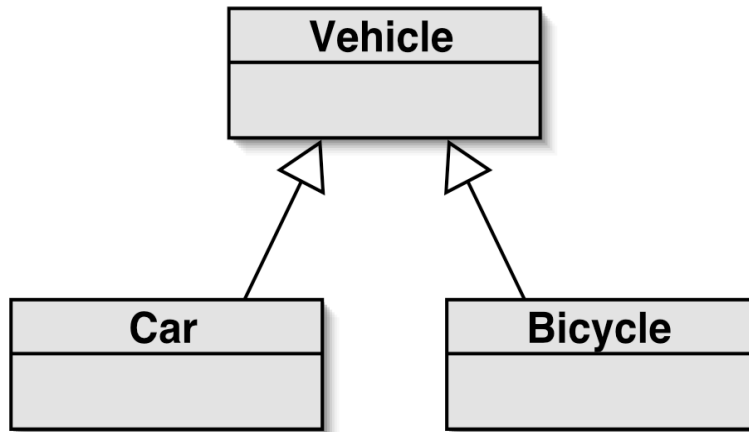
Invocábamos éste método con:

```
DVD miDVD = new DVD (...);
bd.agregarElemento (miDVD);
```

Subclases y subtipos

- Las clases definen tipos.
- Las subclases definen subtipos.
- Los objetos de las subclases pueden usarse donde se requieren los objetos de los supertipos.
(Esto se llama *sustitución* .)

Subtipos y asignación



Los objetos de las subclases se pueden asignar a variables de las superclases

```
Vehiculo v1 = new Vehiculo();  
Vehiculo v2 = new Coche();  
Vehiculo v3 = new Bicicleta();
```

Subtipos y paso de parámetros

```
public class BaseDeDatos
{
    public void agregarElemento(Elemento elElemento)
    {
        ...
    }
}

DVD dvd = new DVD(...);
CD cd = new CD(...);

bd.agregarElemento(dvd);
bd.agregarElemento(cd);
```

Los objetos de una subclase se pueden pasar a los parámetros de su superclase

Diagrama de objetos

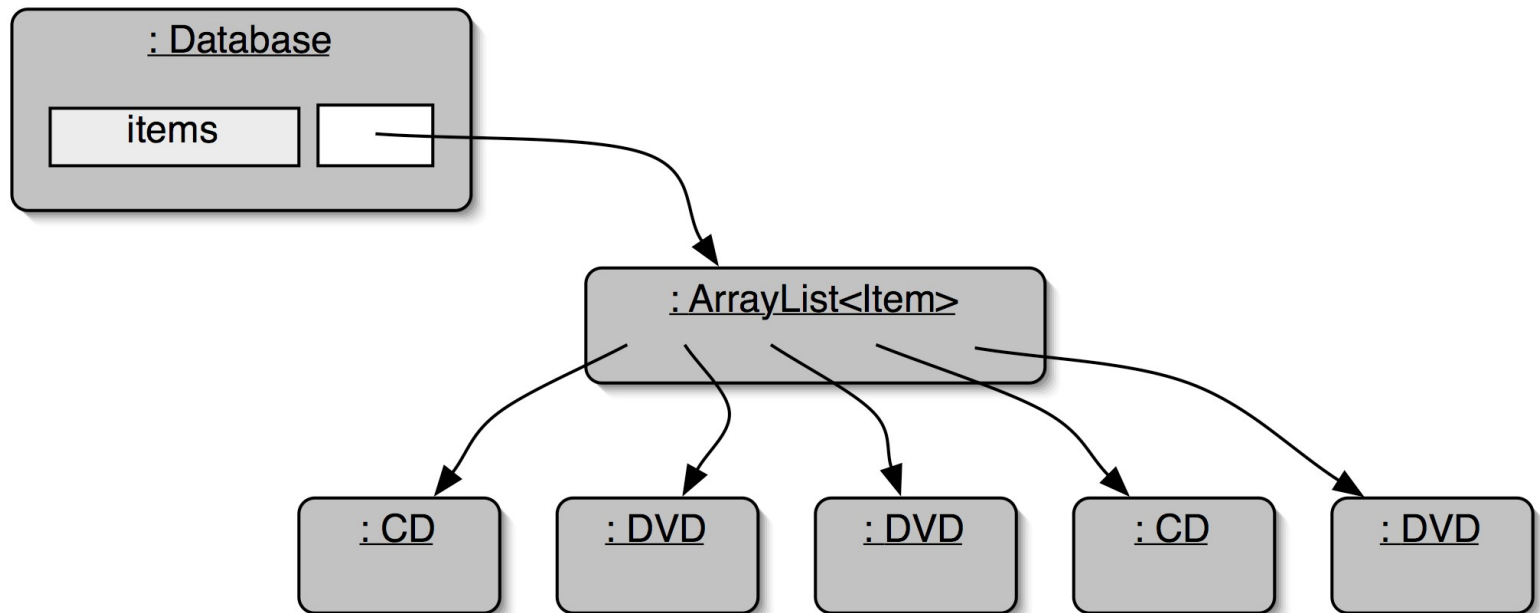
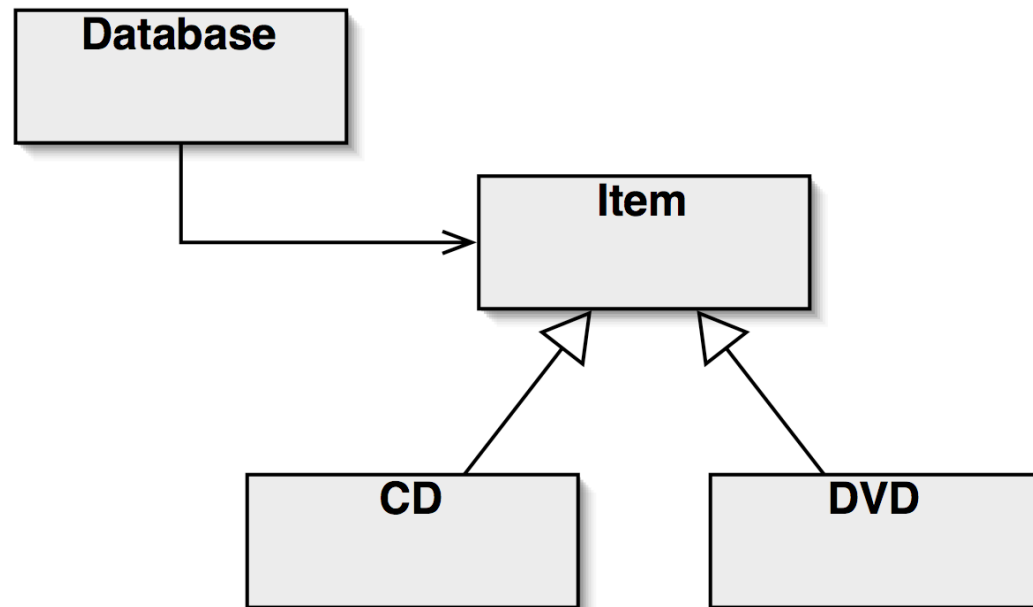


Diagrama de clases



Variables polimórficas

- Las variables de objetos en Java son **polimórficas**.

(Pueden sostener objetos de más de un tipo.)

- Pueden sostener objetos del tipo declarado o de subtipos del tipo declarado.

Enmascaramiento de tipos (*Casting*)

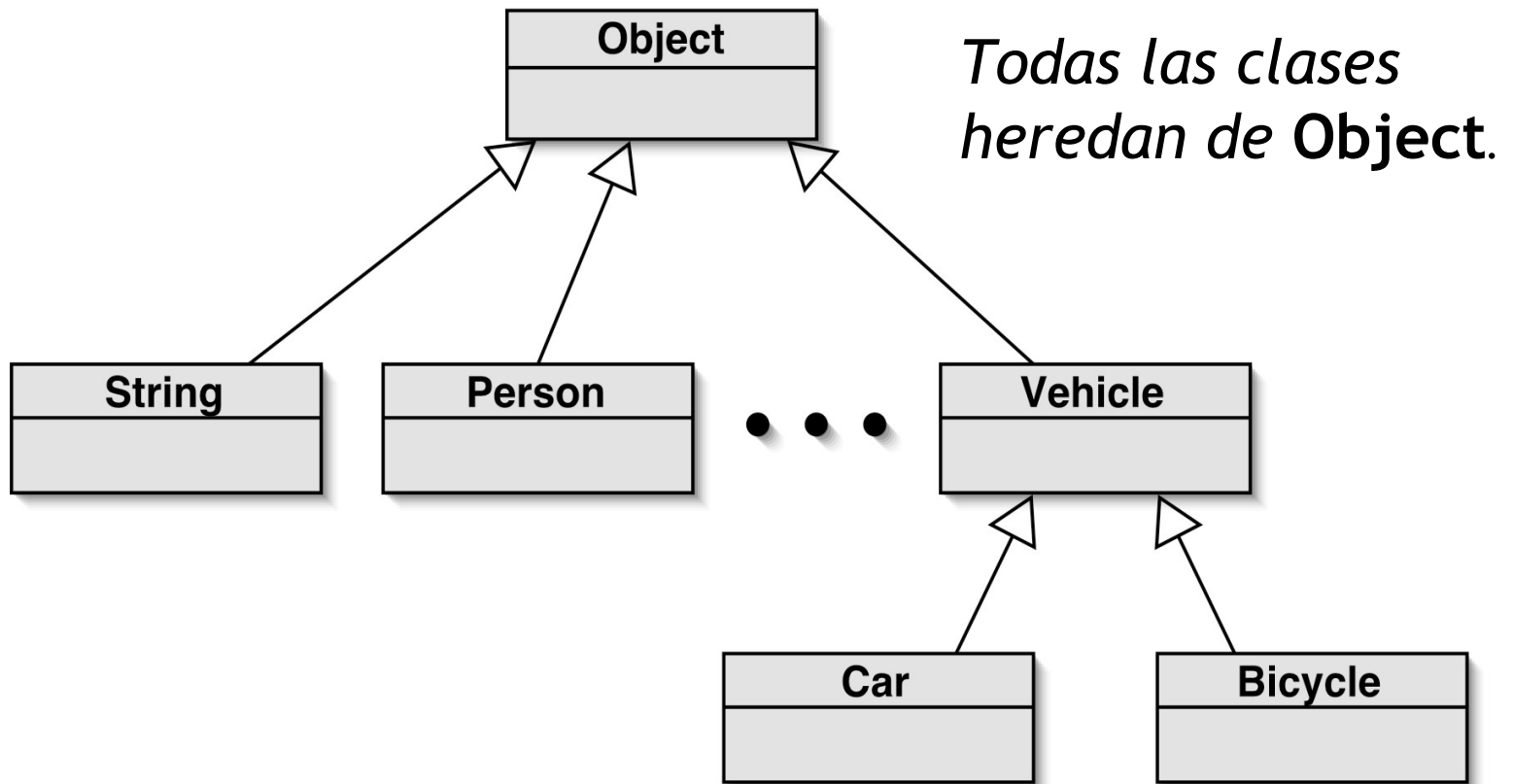
- Permite asignar un subtipo a un supertipo.
- ¡No permite asignar un supertipo a un subtipo!
 Vehiculo v;
 Coche c = new Coche();
 v = c; // correcta;
 c = v; *¡error en tiempo de compilación!*
- El *casting* arregla esto:
 c = (Coche) v;

(¡es correcto solo si el vehículo realmente es un Coche!)

Enmascaramiento (*Casting*)

- Un tipo objeto entre paréntesis.
- Se usa para superar la *pérdida de tipo*.
- El objeto no se cambia en ninguna forma.
- Se realiza una prueba durante la ejecución para asegurar que el objeto es realmente de ese tipo:
 - **ClassCastException** ¡si no lo es!
- Usarlo criteriosamente.

La clase Object



Colecciones polimórficas

- Todas las colecciones son polimórficas.
- Los elementos que almacenan son del tipo **Object**.

```
public void add(Object elemento)
```

```
public Object get(int indice)
```

Colecciones y tipos primitivos

- Todos los objetos pueden agregarse a colecciones ...
- ... dado que las colecciones aceptan elementos del tipo *Object* ...
- ... y todas las clases son subtipos de *Object*.
- ¡Genial! - Pero, ¿qué podemos hacer con los tipos simples?

Clases envoltorio (*wrappers*)

- Los tipos primitivos (*int*, *char*, *etc.*) no son objetos. ¡Deben ser envueltos dentro de un objeto!
- Estas clases existen para los tipos simples:

<i>Tipo simple</i>	<i>Tipo objeto</i>
int	Integer
float	Float
char	Character
...	...

Clases envoltorio

```
int i = 18;  
Integer iwrap = new Integer(i); ← Envuelve el valor  
...  
int value = iwrap.intValue(); ← Lo desenvuelve
```

En la práctica, *autoboxing* y *unboxing* significa que no es usual utilizarlo.

Autoboxing y Unboxing

```
private ArrayList<Integer> listaDeMarcas;
```

```
...
```

```
public void almacenarMarca(int marca)
```

autoboxing

```
{
```

```
    listaDeMarcas.add(marca);
```

```
}
```

unboxing

```
int primeraMarca = listaDeMarcas.remove(0);
```


Repaso

- La herencia permite la definición de clases como extensiones de otras clases.
- Herencia:
 - evita la duplicación de código.
 - permite el reuso del código.
 - simplifica el código.
 - simplifica el mantenimiento y la extensión.
- Las variables pueden sostener objetos de su tipo y de sus subtipos.
- Los subtipos pueden usarse cuando se esperan objetos de un supertipo (sustitución).