Estructura de las Aplicaciones Orientadas a Objetos

Herencia de clases

Programación Orientada a Objetos
Facultad de Informática

Juan Pavón Mestras
Dep. Ingeniería del Software e Inteligencia Artificial
Universidad Complutense Madrid

Basado en el curso Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

Conceptos

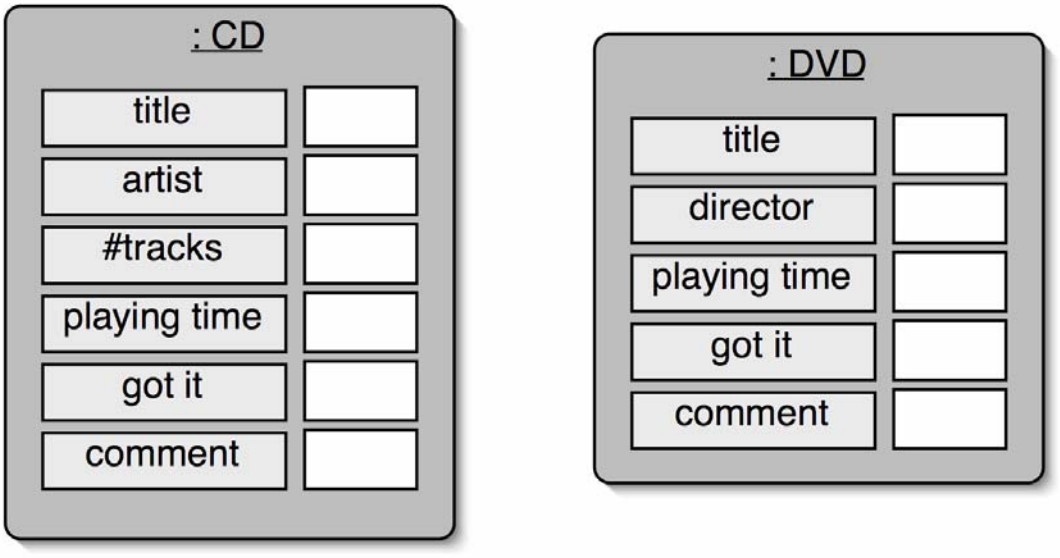
- *Herencia* de clases
- Subtipos
- Sustitución
- Polimorfismo: variables polimórficas

- Construcciones del lenguaje Java:
  - extends
  - super
  - enmascaramiento (casting)
  - la clase Object
  - clases envoltorio (wrappers) de tipos primitivos

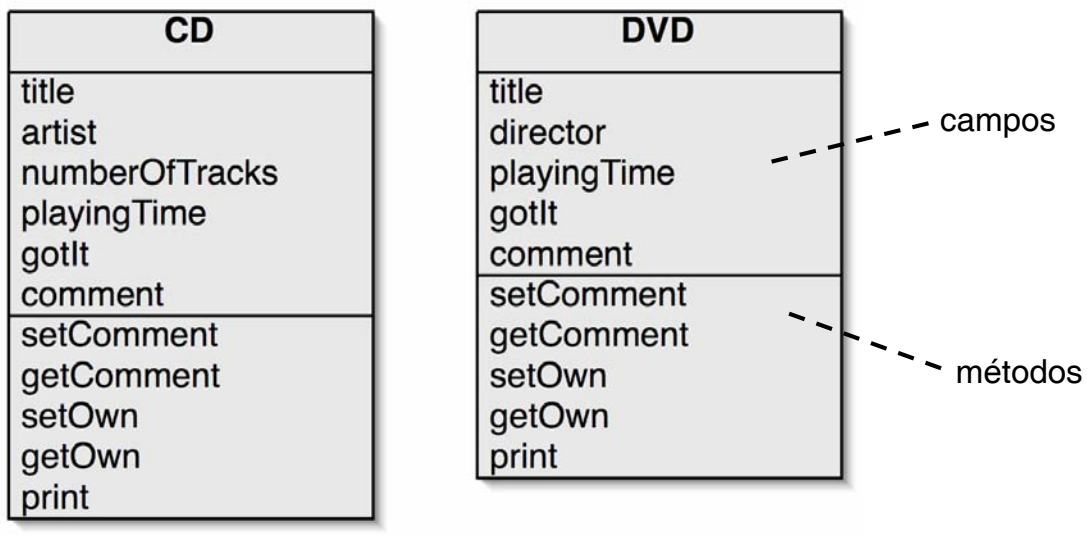
El ejemplo DoME

- DoME: "Database of Multimedia Entertainment"
- Aplicación que permite guardar información sobre discos de música (CD) y películas (DVD)
  - CD: title, artist, # tracks, playing time, got-it, comment
  - DVD: title, director, playing time, got-it, comment
  - Y permite buscar información y sacar listados
- Proyecto en chapter08/dome-v1 y dome-v2

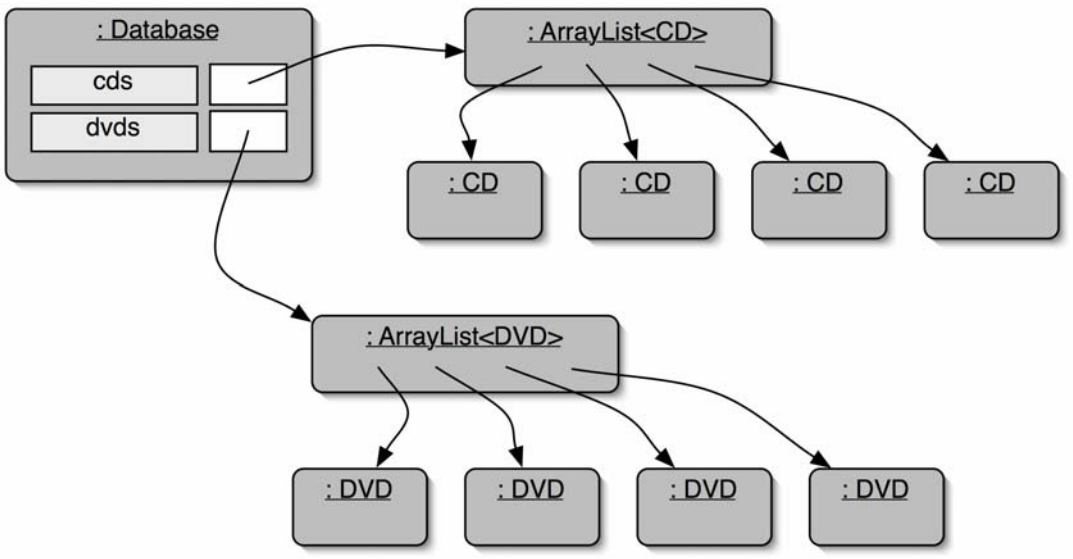
Los objetos de DoME



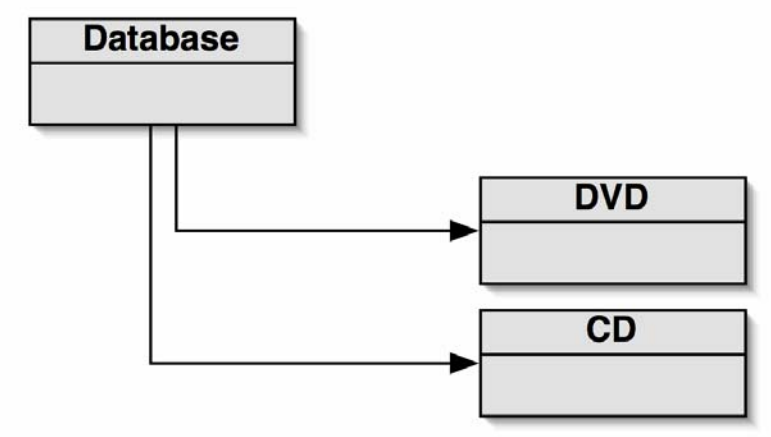
Las clases de DoME



Modelo de objetos de DoME



Modelo de clases de DoME



Código fuente de la clase de los CD

```
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()
   { ... }
   ...
}
```

Código fuente de la clase de los DVD

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

   DVD(String theTitle, String theDirector)
   {
        title = theTitle;
        director = theDirector;
        comment = " ";
   }

   void setComment(String newComment)
   { ... }

   String getComment()
   { ... }

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

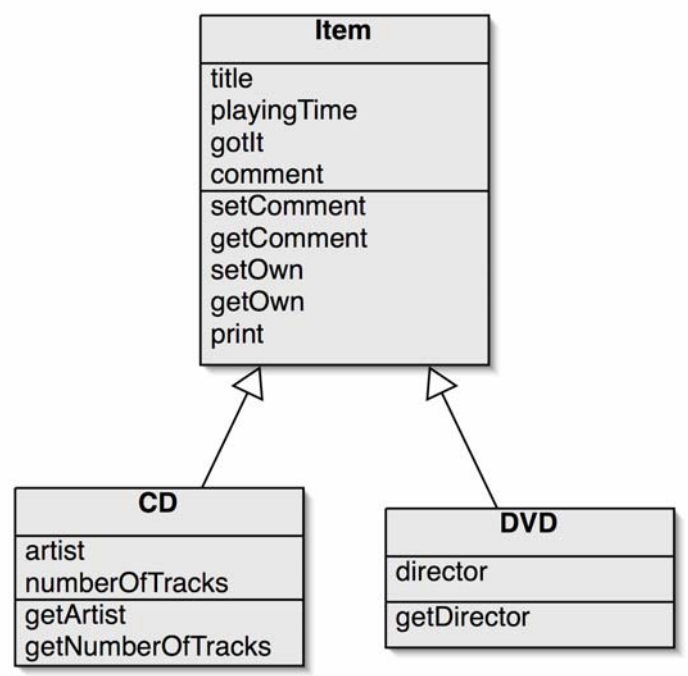

Código fuente de la clase de la base de datos

```
class Database {

   private ArrayList<CD> cds;
   private ArrayList<DVD> dvds;
   ...
   public void list()
   {
      for(CD cd : cds) {
          cd.print();
          System.out.println();  // empty line between items
       }

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

Crítica de la v1 de DoME

- Duplicación de código
  - Las clases CD y DVD son muy parecidas (casi idénticas)
  - Esto hace el mantenimiento difícil y más trabajoso
  - Riesgo de errores con un mantenimiento inadecuado
  - También hay duplicación de código en la clase Database

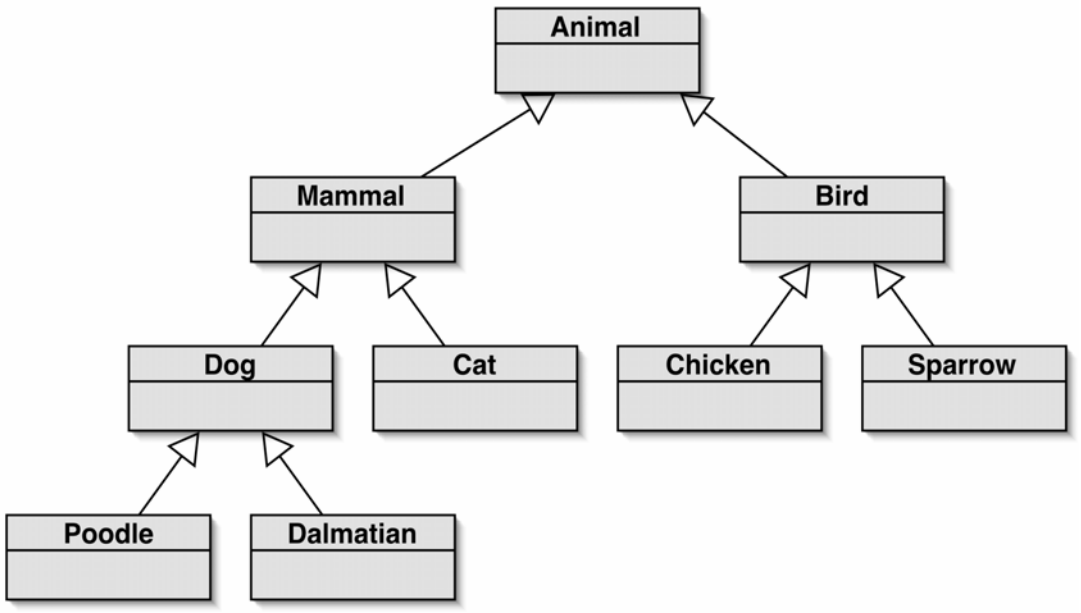
Uso de la herencia



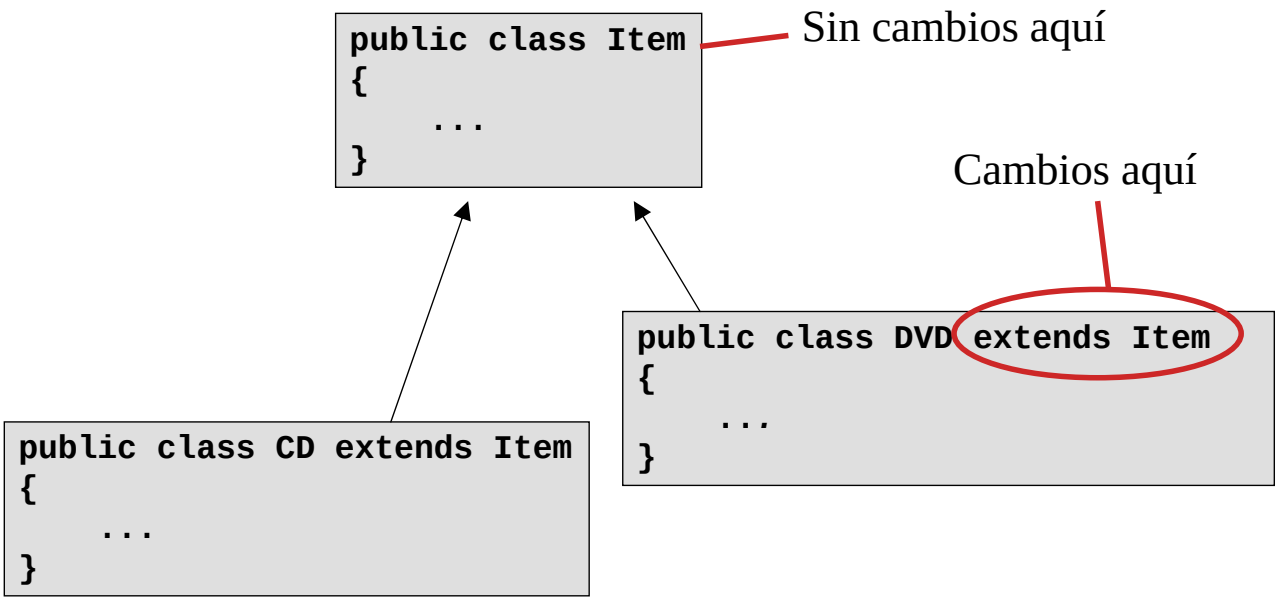
Uso de la herencia

- Se define una **superclase** : Item
- Se definen **subclases** para Video y CD
- La superclase define atributos comunes
- Las subclases **heredan** los atributos de la superclase
- Las subclases pueden tener sus propios atributos

Jerarquías de herencia



La herencia en Java



La superclase

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

    // constructors and methods omitted.
}
```

Las subclases

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

    // constructors and methods omitted.
}
```

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

    // constructors and methods omitted.
}
```


Herencia y constructores

```
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
}
```

Herencia y constructores

```
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
}
```


Llamada al constructor de la superclase

- Los constructores de una subclase siempre deben contener una llamada a un constructor de la superclase
  - Utilizando

    `super(parámetros);`
  - Siempre tiene que ser ***la primera instrucción*** del código de un constructor
- Si no se pone nada, el compilador asume que hay una llamada sin parámetros:

  `super();`
  - Esto implica que la superclase tendría que tener definido un constructor sin parámetros
    - Si sólo tuviera constructores con parámetros, entonces el compilador señalaría el error

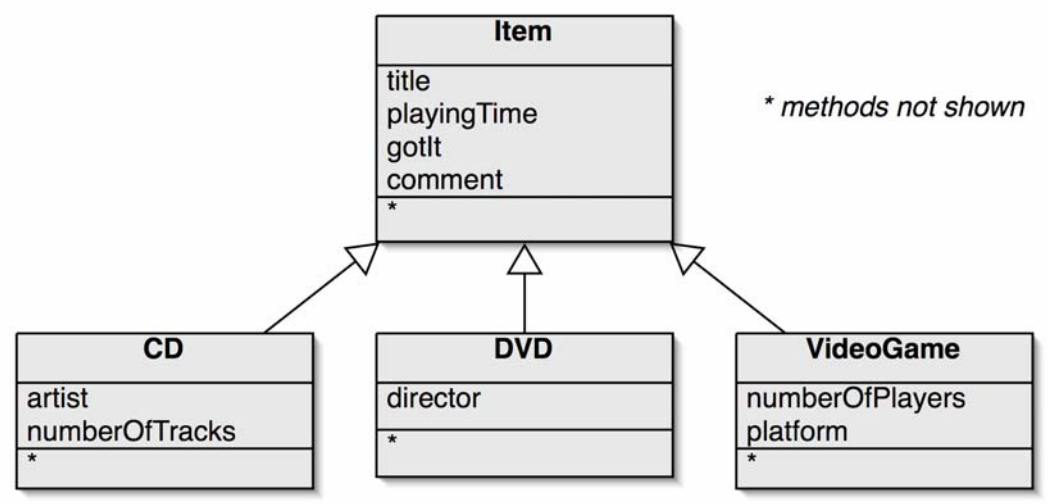
Llamadas a métodos de la superclase

```
// En class Item:

public void toString() {
    return title + "(" + comment + ")";
}
```

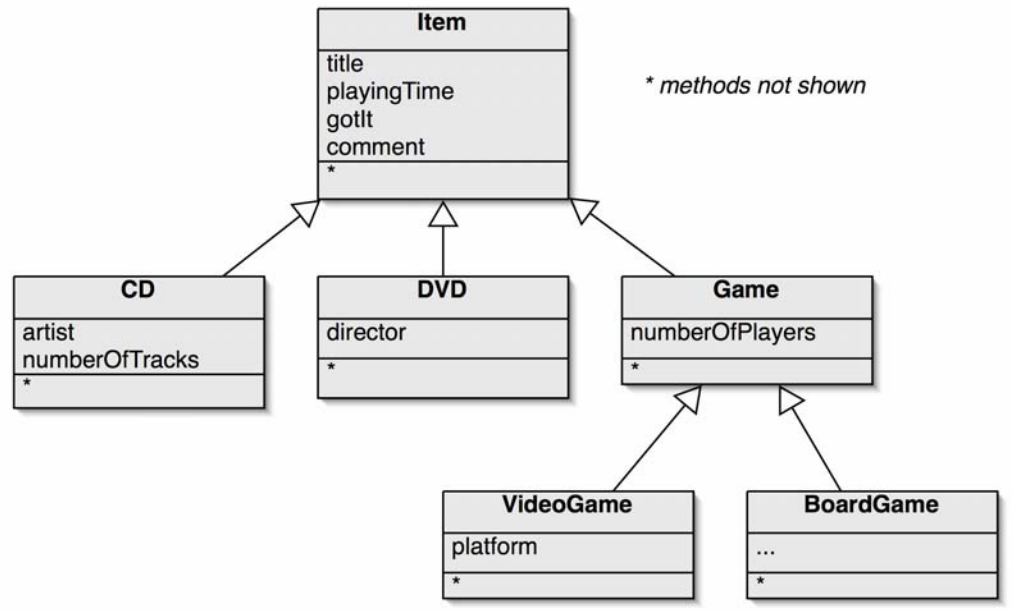
```
// En class CD:

public void toString() {
    return artist + ": " + super.toString();
}
```

```
// o bien:

public void toString() {
    return artist + ": " + getTitle();
}
```


Se pueden añadir nuevos tipos de item



Y definir jerarquías más profundas



En resumen...

- La herencia contribuye a:
  - Evitar duplicación de código
  - Reutilizar código
  - Mejorar el mantenimiento
  - Extensibilidad

Nuevo código de la base de datos

```
public class Database
{
    private ArrayList<Item> items;

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

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


Nuevo código de la base de datos

```
/**
 * Print a list of all currently stored CDs and
 * DVDs to the text terminal.
 */
public void list()
{
   for(Item item : items) {
       item.print();
       // Print an empty line between items
       System.out.println();
   }
}
```

Subtipos

- Al principio se tiene:

```
public void addCD(CD theCD)
public void addVideo(DVD theDVD)
```

- Luego se cambia por:

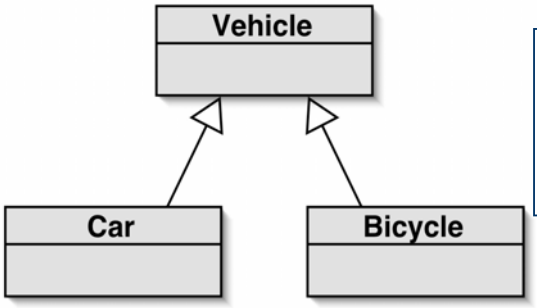
```
public void addItem(Item theItem)
```

- Que se puede llamar como sigue:

```
DVD myDVD = new DVD(...);
database.addItem(myDVD);
```


Subclases y subtipos

- Las clases definen tipos
- Las subclases definen subtipos
- Los objetos de subclases se pueden usar como objetos de los supertipos
  - A esto se le llama **sustitución**
  - Ejemplo: los objetos de las subclases se pueden asignar a variables de la superclase




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

Subtipos y paso de parámetros

- De la misma manera que con la asignación, también se pueden usar subtipos para pasar como parámetros en métodos que tienen definidos parámetros de la superclase

```
public class Database
{

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

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

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


Diagrama de objetos

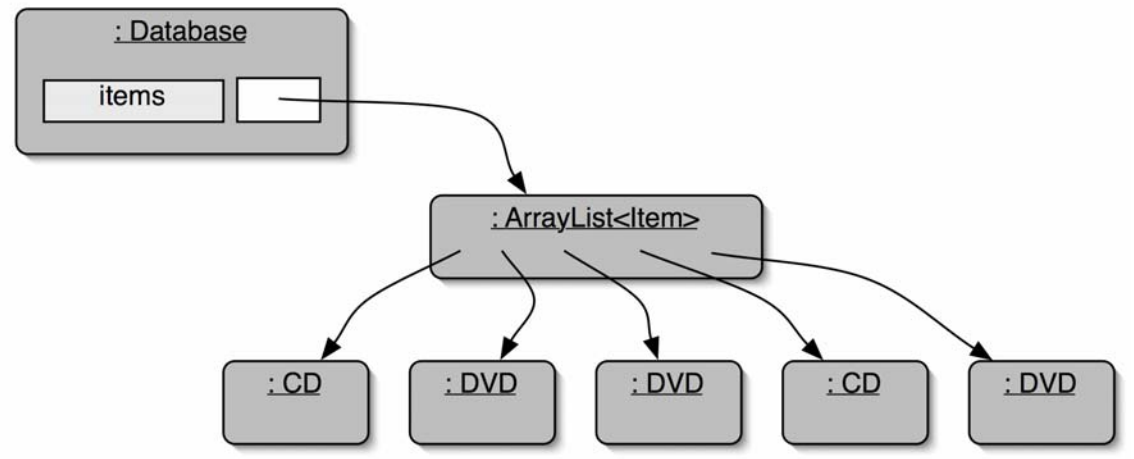
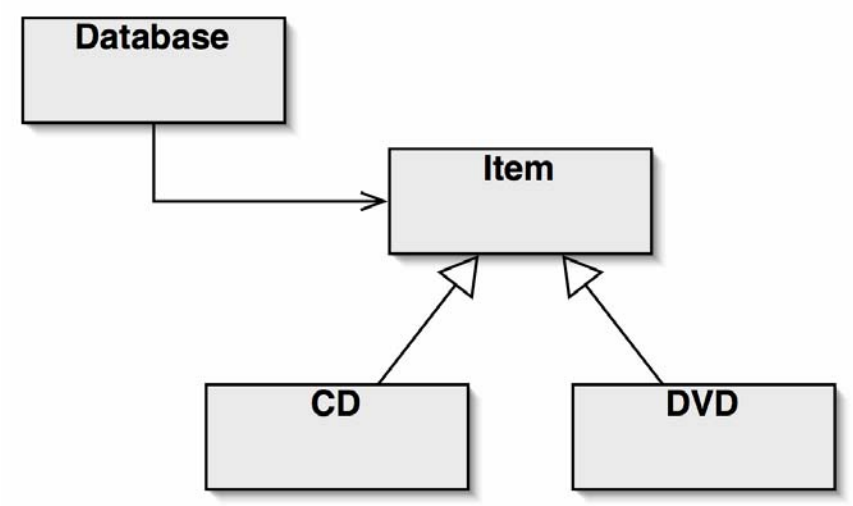
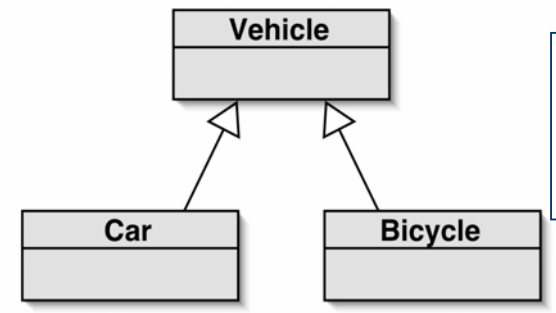


Diagrama de clases



Polimorfismo

- Las variables de referencia a objeto en Java son polimórficas
  - Pueden referenciar objetos de más de un tipo
  - Que son objetos del tipo declarado o sus subtipos




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

Enmascaramiento de tipos (*casting*)

- Se puede asignar un subtipo a un supertipo
- ¡Pero no al revés!

```
Vechicle v;
Car c = new Car();
v = c; // correcto;
c = v;   ¡Error en tiempo de compilación!
```

- Aunque si fuera necesario se puede hacer con la técnica de casting

```
c = (Car) v;
```

  - Pero sólo si v es realmente un Car

Casting

- Se especifica indicando el tipo de objeto entre paréntesis
- El objeto no cambia en nada
  - Simplemente se permite usar la referencia adecuadamente
- En tiempo de ejecución se comprueba que el objeto es realmente de ese tipo
  - **`ClassCastException`** si no lo es
  - En Java esto se puede comprobar

```
if (v instanceof A )  // si el objeto v pertenece a la
                      // clase A o una de sus subclases
```

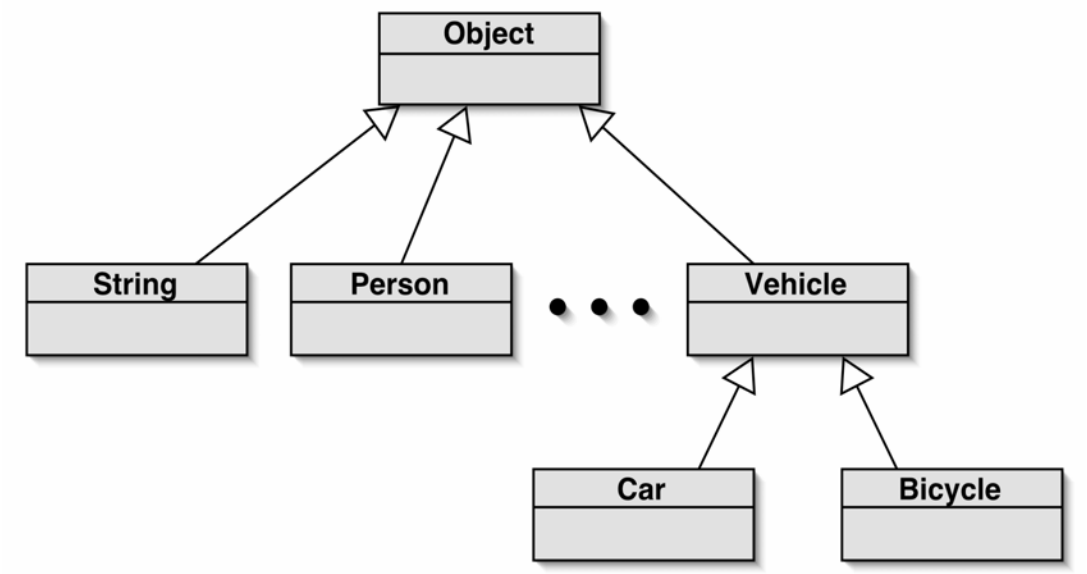
  - Para el ejemplo anterior:

```
if (v instanceof Car )
        c = (Car)v;
```

- *Úsese con moderación*

La clase *Object*

- En Java, todas las clases heredan de la clase *java.lang.Object*



Las colecciones en Java son polimórficas

- Antes de tener tipos genéricos se definieron las colecciones como polimóficas
  - Operaciones del tipo:

```
public void add(Object element)
public Object get(int index)
```

  - permiten trabajar con cualquier tipo de objeto
    - Ya que todas las clases heredan de la clase Object

Clase Object

- public final Class<?> **getClass**()
  - Devuelve la clase del objeto
  - **getName()** sobre el objeto Class devuelve un String con el nombre de la clase
- public int **hashCode**()
  - Devuelve el valor hash code del objeto (identificador único)
- public String **toString**()
  - Devuelve la representación textual como String del objeto
  - Se recomienda que todas las clases redefinan este método
  - Por defecto, devuelve el siguiente texto:

```
getClass().getName() + '@' + Integer.toHexString(hashCode())
```


Clase Object

- public boolean **equals**(Object obj)
  - Comprueba si dos objetos son iguales
- protected Object **clone**() throws CloneNotSupportedException
  - Crea y devuelve una copia del objeto
  - La clase debe implementar la interfaz **Cloneable**
  - x.clone() != x
  - Shallow copy vs. Deep copy

Clase Object – Método finalize()

- **protected void finalize() throws Throwable**
  - Método invocado por el recogedor de basura cuando no hay referencias al objeto y se va a eliminar
  - Sirve para hacer operaciones de limpieza y liberar recursos asociados al objeto
  - El método en la clase Object no realiza ninguna operación

Clases envoltorio (*wrapper*)

- Si se quieren utilizar los tipos primitivos (int, boolean, etc.) donde valga un Object, ¿cómo hacerlo?
- La respuesta es un conjunto de clases envoltorio (*wrappers*) que envuelven la variable

tipo simple	*clase wrapper*
int	Integer
float	Float
char	Character
boolean	Boolean
byte	Byte
…	…

Clases envoltorio (*wrapper*)

```
int i = 18;
Integer iwrap = new Integer(i);
…
int value = iwrap.intValue();
```

Envuelve el valor

Lo desenvuelve

Autoboxing y *unboxing*

- Aunque en ocasiones donde se espera un Object el compilador se encarga de hacer la conversión automática

```
private ArrayList<Integer> markList;
…
public void storeMark(int mark)
{
    markList.add(mark);
}
```

autoboxing

```
int firstMark = markList.remove(0);
```

unboxing