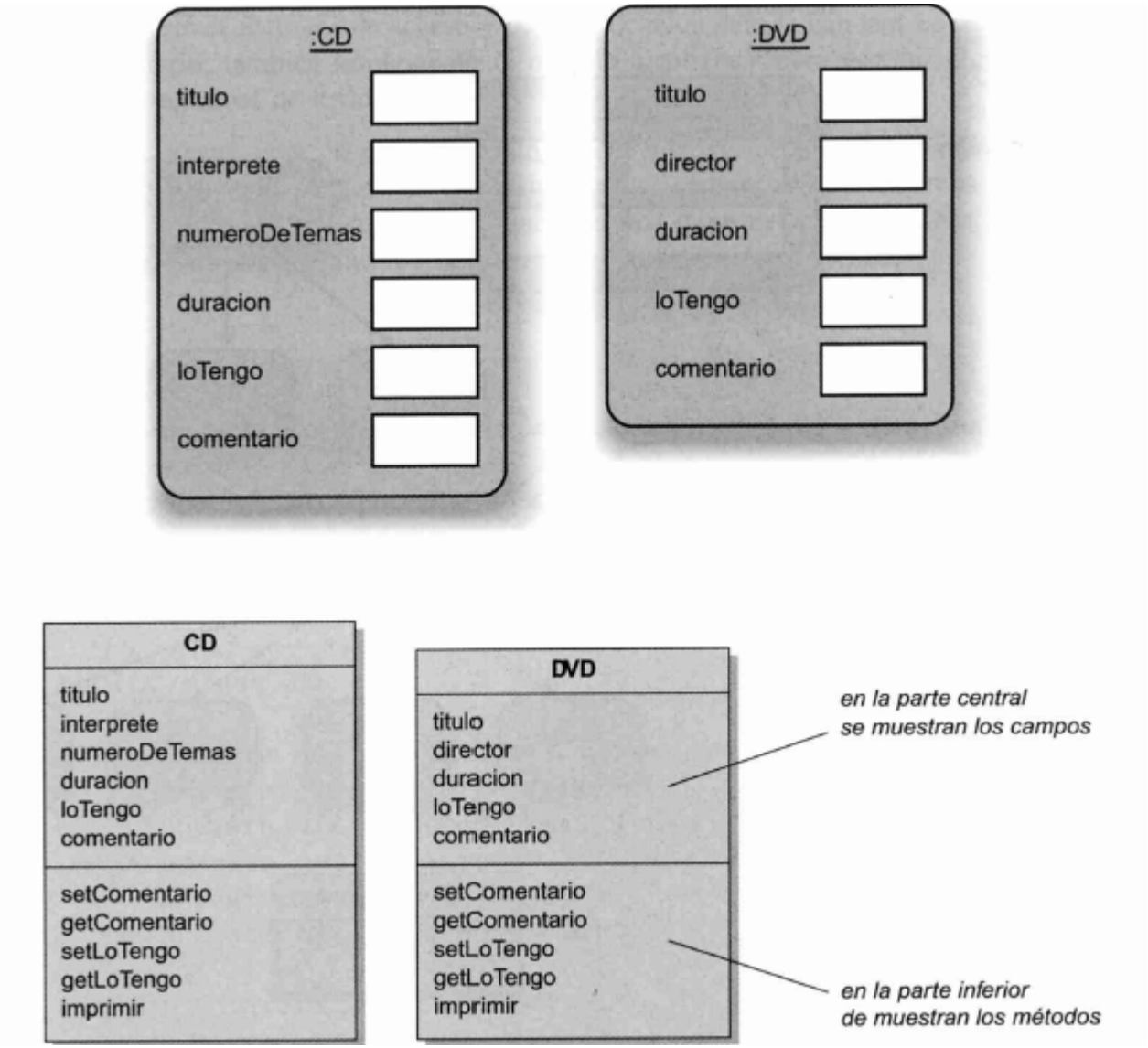
Programación orientada a objetos

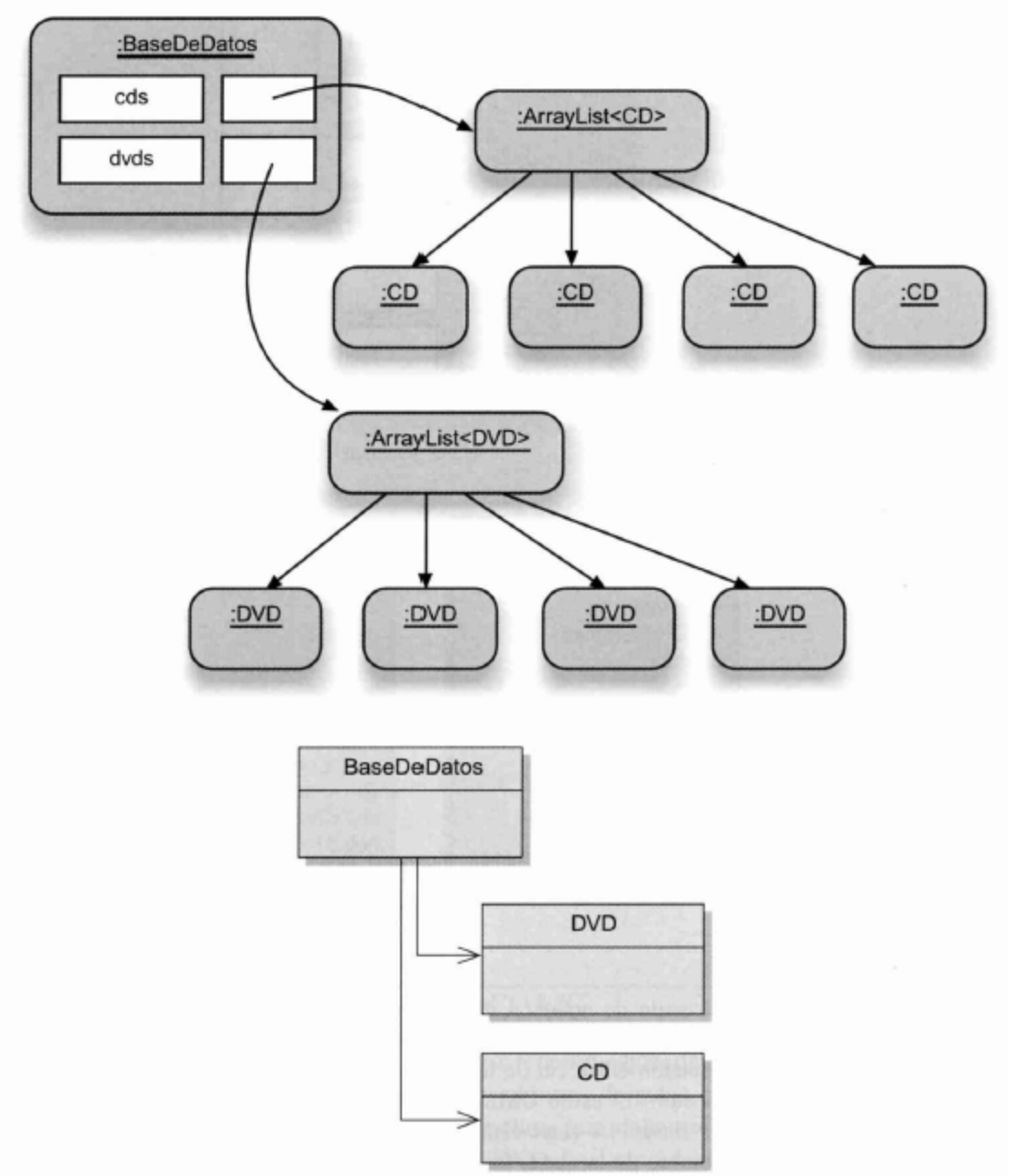
Capítulo 8

Mejora de las estructuras mediante herencia

Tema 8: Extensión de clases: Herencia. Semana 8	
1- El uso de la herencia. 2- Jerarquías de herencia. 3- Herencia en Java. 4- Herencia y derechos de acceso. 5- Herencia e inicialización. 6- Reutilización de código por medio de la herencia. 7- Subtipos a. Subclases y subtipos. b. Subtipos y asignación. c. Subtipo y paso de parámetros. d. Variables polimórficas. e. Enmascaramiento de tipos. 8- La clase Object. 9- Tipos estáticos y dinámicos. 10- Sobreescritura de métodos. 11- Llamada a métodos con la palabra reservada super. 12- Métodos polimórficos 13- Acceso protegido	1- Estudiar el capítulo 8 del libro base para la "Unidad Didáctica II". 2- Realizar los ejercicios correspondientes del libro base. 3- Realizar los ejercicios resueltos en examines de años anteriores en los que se utilice la herencia.

Clases y objetos de DoME



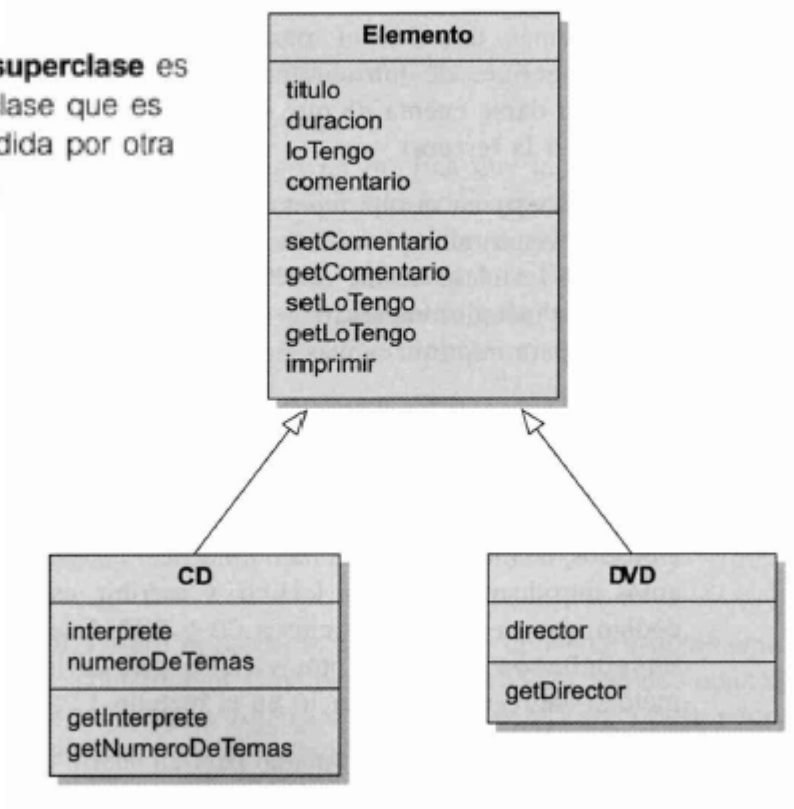

:BaseDeDatos
cds
dvds
:ArrayList<CD>
:CD
:CD
:CD
:CD
:ArrayList<DVD>
:DVD
:DVD
:DVD
:DVD
BaseDeDatos
DVD
CD

Usar herencia

La **herencia** nos permite definir una clase como una extensión de otra.

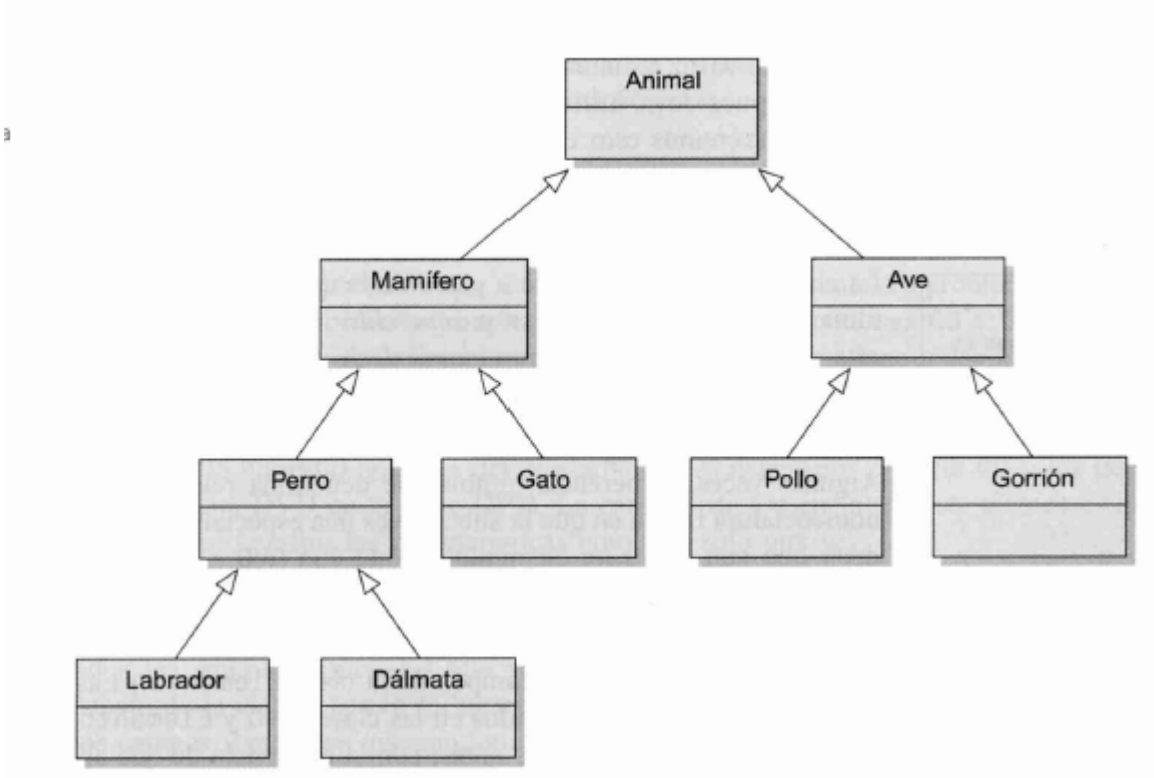
Una **superclase** es una clase que es extendida por otra clase.

Una **subclase** es una clase que extiende a otra clase. Hereda todos los campos y los métodos de la superclase.

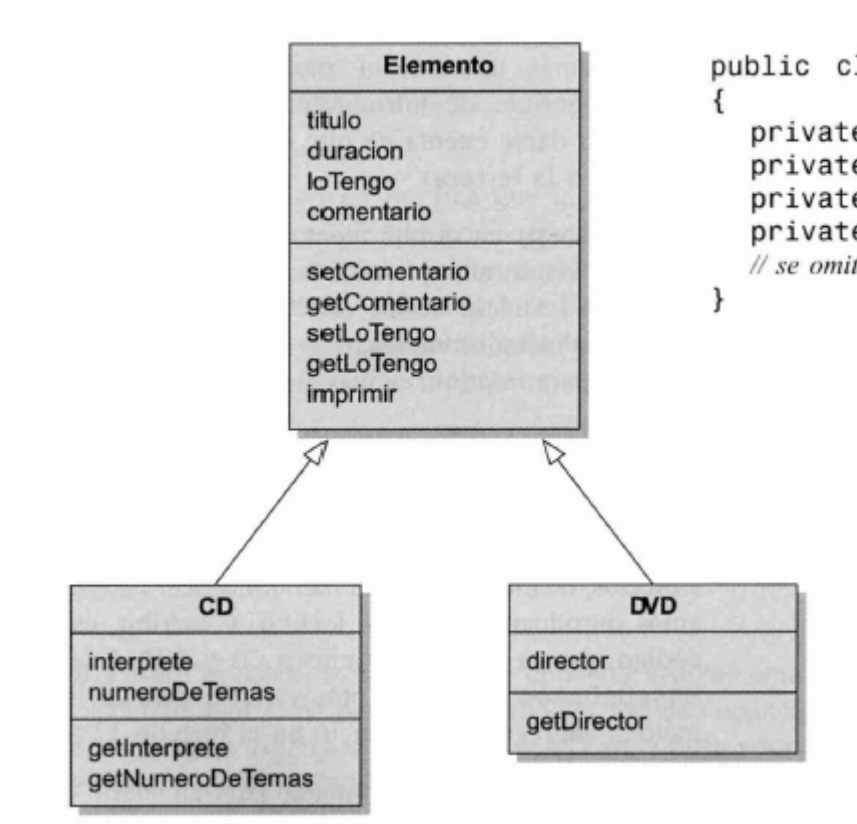


Jerarquía de herencia

Las clases que están vinculadas mediante una relación de herencia forman una **jerarquía de herencia**.



Herencia en Java




```
public class Elemento
{
   private String titulo;
   private int duracion;
   private bolean loTengo;
   private String comentario;
   // se omitieron constructores y métodos
}
```

```
public class CD extends Elemento
{
   private String interprete;
   private int numeroDeTemas;
   // se omitieron constructores y métodos
}
```

```
public class DVD extends Elemento
{
   private String director;
   // se omitieron constructores y métodos
}
```

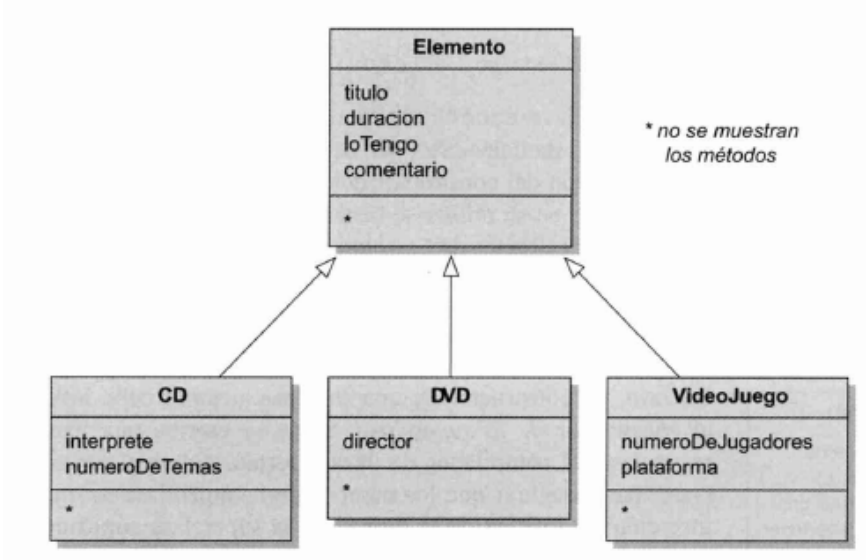
Herencia e iniciación

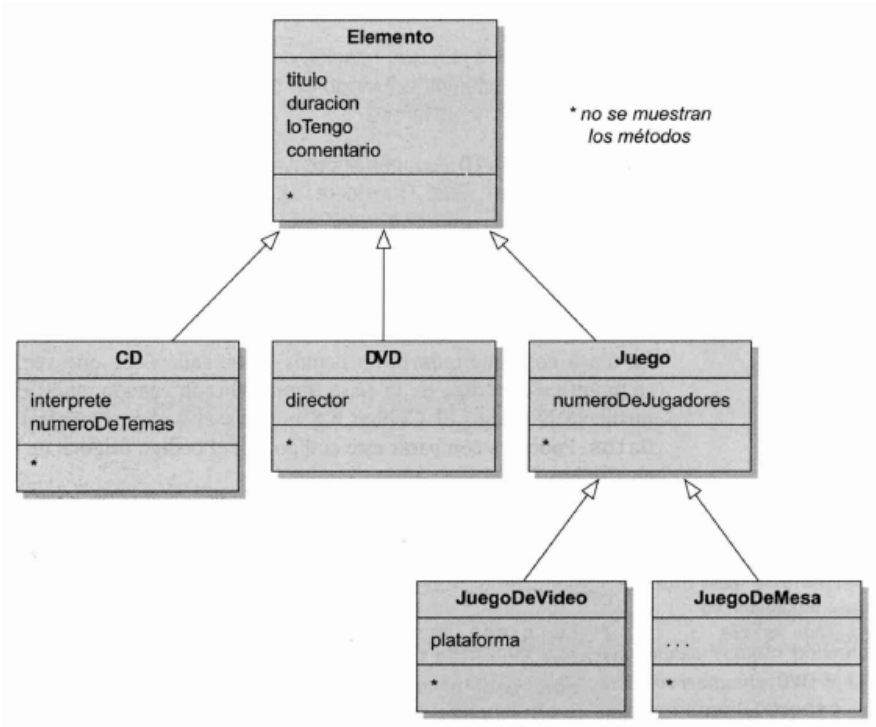
```
public class Elemento
{
      private String titulo;
      private int duracion;
      private boolean loTengo;
      private String comentario;
      /**
       * Inicializa los campos del elemento.
       * @param elTitulo   el título de este elemento.
       * @param tiempo La duración de este elemento.
       */
      public Elemento(String elTitulo, int tiempo)
      {
            titulo = elTitulo;
            duracion = tiempo;
            loTengo = false;
            comentario = "";
      }
      // Se omitieron métodos
}
public class CD extends Elemento
{
      private String interprete;
      private int numeroDeTemas;
      /**
       * Constructor de objetos de la clase CD
       * @param elTitulo El título del CD.
       * @param elInterprete   El intérprete del CD.
       * @param temas   El número de temas del CD.
       * @param tiempo La duración del CD.
       */
      public CD(String elTitulo, String elInterprete, int
temas, int tiempo)
      {
            super(elTitulo, tiempo);
            interprete = elInterprete;
            numeroDeTemas = temas;
      }
      // Se omitieron métodos
}
```

Constructor de superclase. El constructor de una subclase debe tener siempre como primera sentencia una invocación al constructor de su superclase. Si el código no incluye esta llamada, Java intentará insertarla automáticamente.

La herencia nos permite **reutilizar** en un nuevo contexto clases que fueron escritas previamente.

Agregar otras subclases



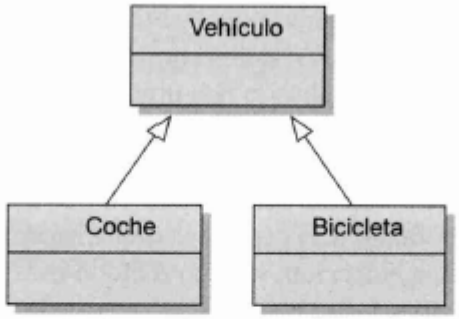


Ventajas de la Herencia

- **Evita la duplicación de código** El uso de la herencia evita la necesidad de escribir copias de código idénticas o muy similares dos veces (o con frecuencia, aún más veces).
- **Se reutiliza código** El código que ya existe puede ser reutilizado. Si ya existe una clase similar a la que necesitamos, a veces podemos crear una subclase a partir de esa clase existente y reutilizar un poco de su código en lugar de implementar todo nuevamente.
- **Facilita el mantenimiento** El mantenimiento de la aplicación se facilita pues la relación entre las clases está claramente expresada. Un cambio en un campo o en un método compartido entre diferentes tipos de subclases se realiza una sola vez.
- **Facilita la extendibilidad** En cierta manera, el uso de la herencia hace mucho más fácil la extensión de una aplicación.

Subclases y subtipos

Subtipo. Por analogía con la jerarquía de clases, los tipos forman una jerarquía de tipos. El tipo que se define mediante la definición de una subclase es un subtipo del tipo de su superclase.



Variables y subtipos. Las variables pueden contener objetos del tipo declarado o de cualquier subtipo del tipo declarado.

Imagine que tenemos una clase `Vehiculo` con dos subclases, `coche` y `Bicicleta` (Figura 8.9). En este caso la regla de tipos admite que las siguientes sentencias son todas legales:

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

Sustitución. Se pueden usar objetos de subtipos en cualquier lugar en el que se espera un objeto de un supertipo. Esto se conoce como sustitución.

Enmascaramiento de tipos

```
Vehiculo v;
  Coche a = new Coche();
  v = a;       // es correcta
  a = v;       // es un error
```

```
a = (Coche) v;   // correcto
```

El operador de enmascaramiento consiste en el nombre de un tipo (en este caso, `Coche`) escrito entre paréntesis, que precede a una variable o a una expresión. Al usar esta operación, el compilador creerá que el objeto es un `Coche` y no informará ningún error.

Ahora considere este fragmento de código en el que `Bicicleta` también es una subclase de `Vehiculo`:

```
Vehiculo v;
  Coche a;
Bicicleta b;
a = new Coche();
  v = a;       // correcta
  b = (Bicicleta) a;      // ¡error en tiempo de compilación!
  b = (Bicicleta) v;      // ¡error en tiempo de ejecución!
```

Las últimas dos asignaciones fallarán. El intento de asignar la variable `a` en la variable `b` (aun estando enmascarada) dará por resultado un error en tiempo de compilación. El compilador se dará cuenta de que `Coche` y `Bicicleta` no constituyen una relación subtipo/supertipo, y por este motivo la variable `a` nunca puede contener un objeto `Bicicleta`; esta asignación no funcionará nunca.

Términos introducidos en este capítulo

herencia, superclase(padre), subclase(hijo), es-un, jerarquía de herencia, clase abstracta, subtipo, sustitución, variable polimórfica, pérdida de tipo, enmascaramiento, autoboxing, clases envoltorio

Resumen de conceptos

- **herencia** La herencia nos permite definir una clase como extensión de otra.
- **superclase** Una superclase es una clase que es extendida por otra clase.
- **subclase** Una subclase es una clase que extiende (deriva de) otra clase. Hereda todos los campos y métodos de su superclase.
- **jerarquía de herencia** Las clases que están vinculadas mediante una relación de herencia forman una jerarquía de herencia.
- **constructores de superclase** El constructor de una subclase siempre debe invocar al constructor de la superclase en su primera sentencia. Si el código fuente no incluye esta llamada, Java intentará insertarla automáticamente.
- **reutilización** La herencia nos permite reutilizar en un nuevo contexto clases escritas previamente.
- **subtipos** Por analogía con la jerarquía de clase, los tipos forman una jerarquía de tipos. El tipo definido mediante una definición de subclase es un subtipo del tipo de su superclase.
- **variables y subtipos** Las variables pueden contener objetos de su tipo declarado o de cualquier subtipo de su tipo declarado.
- **sustitución** Los objetos subtipo pueden usarse cada vez que se espera un supertipo. Esto se conoce como sustitución.
- **Object** Todas las clases que no tienen una superclase explícita tienen a `Object` como su superclase.
- **Autoboxing** El proceso de autoboxing se lleva a cabo automáticamente cuando se usa un valor de un tipo primitivo en un contexto que requiere un tipo objeto.