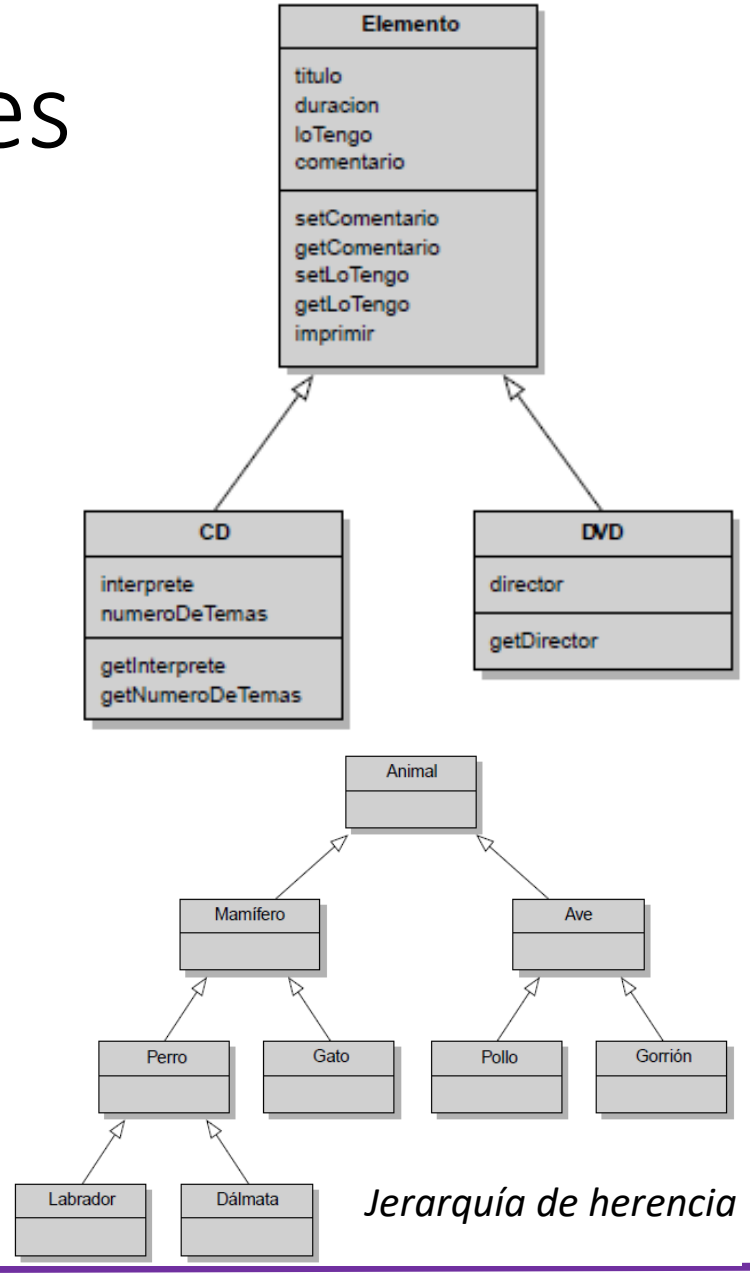
Programación Orientada a Objetos

Capítulo 10: Mejora de la estructura mediante herencia

Nieves Pedreira Souto

HERENCIA: Conceptos importantes

- **Herencia:** Mecanismo que permite crear unas clases como extensión de otras. Es una técnica de abstracción que permite categorizar las clases de objetos bajo cierto criterio y ayuda a especificar las características de estas clases.
  - Para entidades similares, se define una clase que tiene todos los elementos comunes, y después una subclase por cada una de las entidades (**extends**)
- **Superclase**: clase extendida por otras.
- **Subclase**: clase que extiende a otra. Hereda todos los campos y métodos de la superclase
- **Permisos**: una subclase no puede acceder a los miembros privados de su superclase. Una subclase puede invocar a cualquier método público de su superclase como si fuera propio
- **super**: referencia a la 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.
- Todas aquellas clases que no tienen una superclase explícita tienen como su superclase a la clase **Object**.

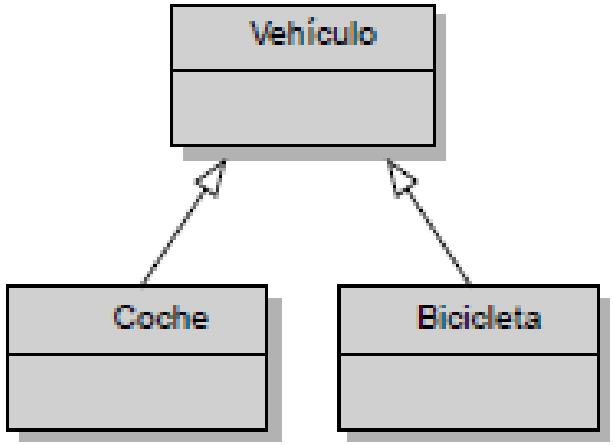


Jerarquía de herencia

HERENCIA: Conceptos importantes

- **Subtipo:** 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 polifórmicas** : variables que contienen objetos. Pueden contener objetos del tipo declarado o de cualquier subtipo del tipo declarado.
- **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**: indicar del subtipo con el tipo entre paréntesis (puede dar error en ejecución):

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



- **Autoboxing:** cuando se usa un valor de un tipo primitivo en un contexto que requiere un tipo objeto.

HERENCIA: Conceptos importantes

- **Tipos primitivos:** los tipos predefinidos, como int, boolean, char. Estos no pertenecen a la clase Object, por lo que no podrían ser utilizados donde se reuieran las características de Object. Por ejemplo, para formar parte de colecciones (arrayList)
- Cada tipo simple o primitivo tiene su correspondiente **clase envoltorio** que representa el mismo tipo pero que, en realidad, es un tipo objeto. Por ejemplo, la clase envoltorio para el tipo simple int es la clase de nombre Integer:

```
int i;
Integer ienvuelto = new Integer(i);
```

- **Autoboxing:** En cualquier lugar en el que se use un valor de un tipo primitivo en un contexto que requiere un tipo objeto, el compilador automáticamente envuelve al valor de tipo primitivo en un objeto con el envoltorio adecuado.
  - Así, se podría crear un arrayList de Integer y añadir elementos tipo int
- La operación inversa, **unboxing**, también se lleva a cabo automáticamente.
- El proceso de **autoboxing** se aplica en cualquier lugar en el que se pase como parámetro un tipo primitivo a un método que espera un tipo envoltorio, y cuando un valor primitivo se almacena en una variable de su correspondiente tipo envoltorio.
- De manera similar, el proceso de **unboxing** se aplica cuando un valor de tipo envoltorio se pasa como parámetro a un método que espera un valor de tipo primitivo, y cuando se almacena en una variable de tipo primitivo.

HERENCIA: Ventajas

- **Evita la duplicación de código** El uso de la herencia evita la necesidad de escribir copias de código idénticas o muy similares.
- **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.