

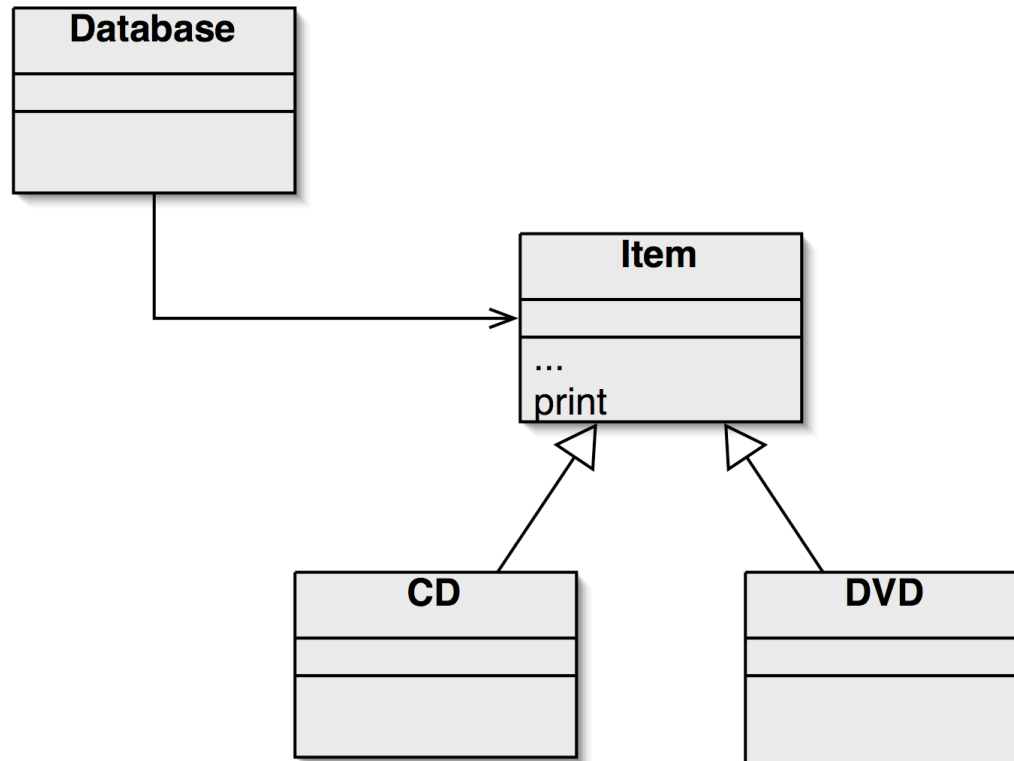
Polimorfismo

Algo más acerca de la herencia

Principales conceptos a ser cubiertos

- Polimorfismo de métodos.
- Tipos estaticos y dinámicos.
- Sobreescribir.
- Búsqueda dinámica de métodos.
- Acceso protegido.

La jerarquía de herencia



Salida conflictiva

¿Qué queremos?

CD: A Swingin' Affair (64 mins)*
Frank Sinatra
tracks: 16
my favourite Sinatra album

DVD: O Brother, Where Art Thou? (106 mins)
Joel & Ethan Coen
The Coen brothers' best movie!

¿Que tenemos?

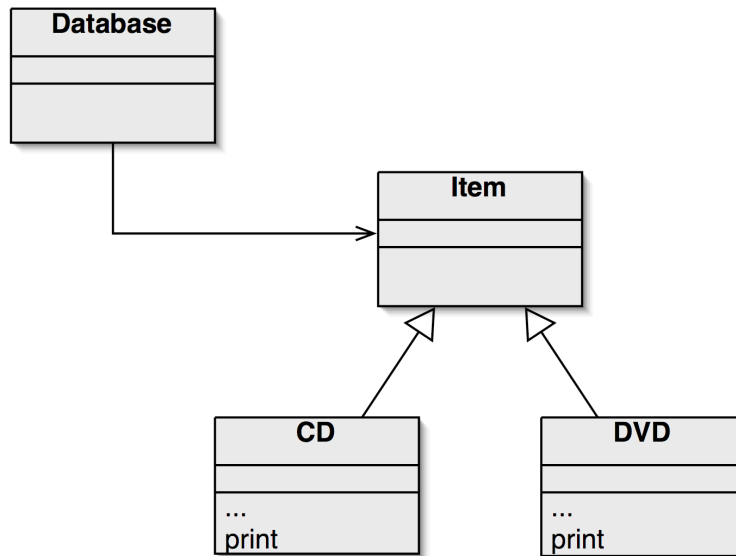
title: A Swingin' Affair (64 mins)*
my favourite Sinatra album

title: O Brother, Where Art Thou? (106 mins)
The Coen brothers' best movie!

El problema

- El método `print` en `Elemento` sólo imprime los campos comunes.
- La herencia es una calle de una sola mano:
 - Una subclase hereda los campos de la superclase.
 - La superclase no sabe nada acerca de los campos de sus subclases.

Intento de resolver el problema



- Colocar la **print** donde tenga acceso a la información que necesita.
- Cada subclase tiene su propia versión.
- Pero los campos del Elemento son private.
- **Database** no puede encontrar un método **print** en **Elemento**.

Tipo estático y tipo dinámico

- Un tipo de jerarquía más compleja requiere conceptos adicionales para describirla.
- Alguna terminología nueva:
 - Tipo estático.
 - Tipo dinámico.
 - Búsqueda/despacho de métodos.

Tipo estático y dinámico

¿Cuál es el tipo de c1?

```
Coche c1 = new Coche();
```

¿Cuál es el tipo de v1?

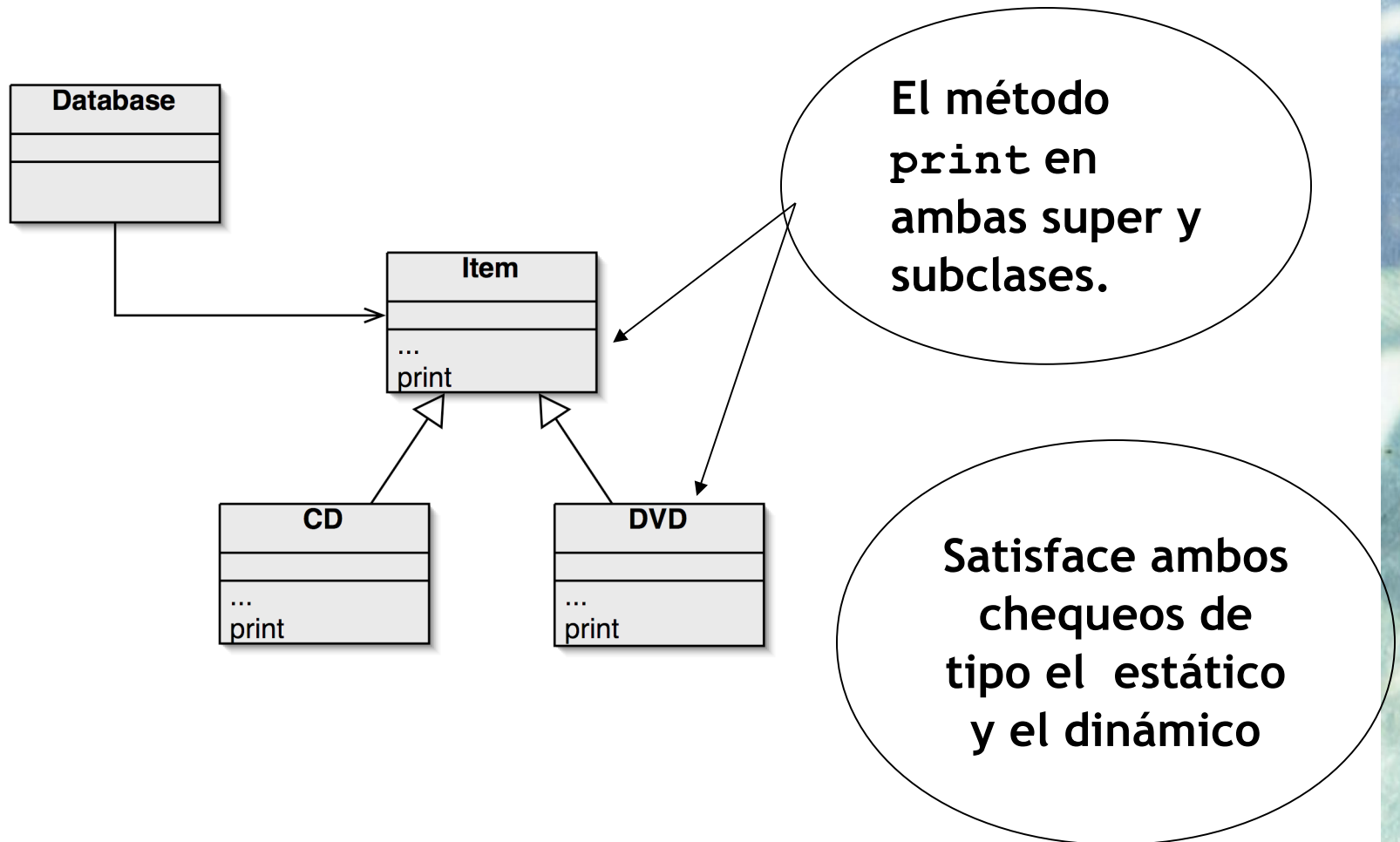
```
Vehiculo v1 = new Coche();
```


Tipo estático y dinámico

- El tipo *declarado* de una variable es su tipo *estático*.
- El tipo del objeto al cual se *refiere* una variable es su tipo *dinámico*.
- El trabajo del compilador es verificar las violaciones de los tipos estáticos.

```
for(Elemento elemento : elementos) {  
    elemento.print(); // Error de Compilación  
}
```

Sobreescritura: la solución



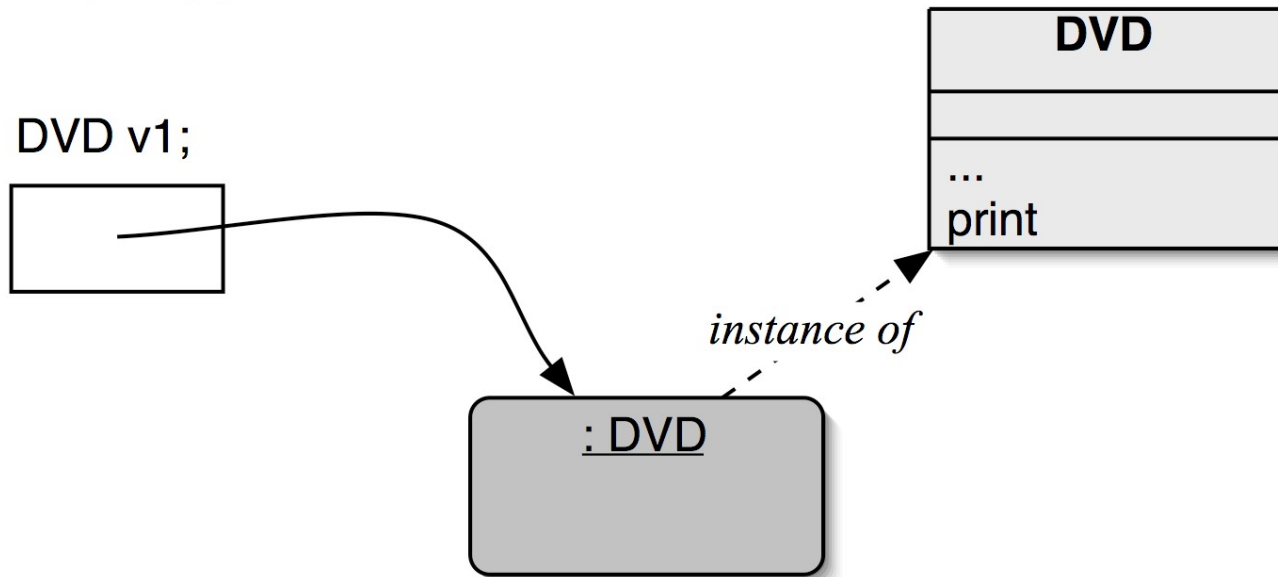
Sobreescritura

- Las superclases y las subclases definen métodos con la misma signatura.
- Cada una accede a los campos de su clase.
- La superclase satisface la verificación estática de tipo.
- El método de la subclase es llamado en tiempo de ejecución - *sobreescribe* la versión de la superclase.
- ¿Qué ocurre con la versión de la superclase?

Búsqueda del método

v1.print();

DVD v1;

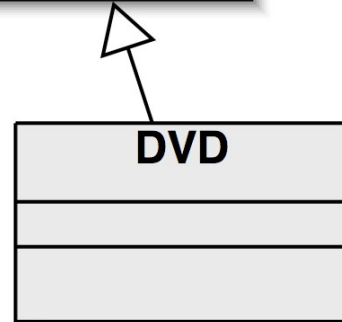
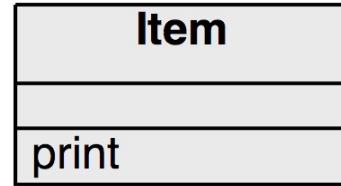
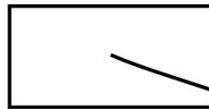


Ni herencia ni polimorfismo.
Se elige el método obvio.

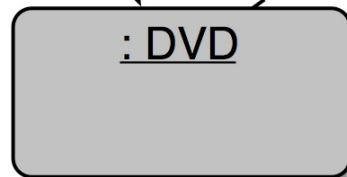
Búsqueda del método

v1.print();

DVD v1;



instance of

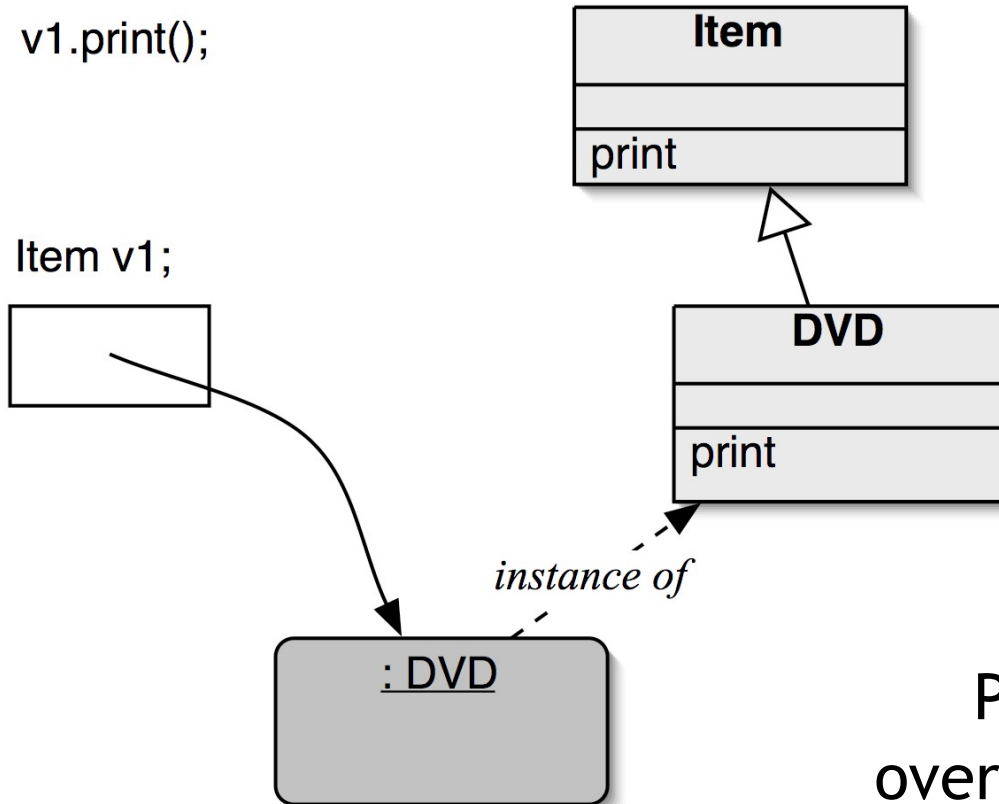


Herencia pero no
sobreescripción. Se asciende
en la jerarquía de herencia,
buscando un ajuste.

Method lookup

`v1.print();`

`Item v1;`



Polymorphism and overriding. The ‘first’ version found is used.

Algoritmo de búsqueda del método a ejecutar

- Se accede a la variable.
- Se encuentra el objeto almacenado en la variable.
- Se encuentra la clase del objeto.
- Se busca un ajuste del método en la clase, si no lo hay, se busca en la superclase.
- Esto se repite hasta que se encuentra un ajuste, o se agota la jerarquía de clases.
- Los métodos que sobreescriben toman precedencia.

Llamada a *super* en métodos

- Los métodos sobrescritos son ocultos ...
- ... pero a menudo todavía queremos ser capaces de llamarlos.
- Un método sobrescrito puede ser llamado desde el método que lo sobreescribe.
 - `super.method(...)`
 - Comparelo con el uso de `super` en los constructores.

Llamada a un método sobrescrito

```
public class CD
{
    ...
    public void imprimir()
    {
        super.imprimir();
        System.out.println("      " + interprete);
        System.out.println("      temas: " +
                           numeroDeTemas);
    }
    ...
}
```

Polimorfismo de métodos

- Se ha discutido el despacho de métodos *polimórficos*.
- Una variable polimórfica puede almacenar objetos de varios tipos.
- Los llamados a métodos son polimórficos.
 - El método real llamado depende del *tipo dinámico* del objeto.

Los métodos de la clase Object

- Los métodos en `Object` son heredados por todas las clases.
- Cualquiera de estos puede ser sobrescrito.
- El método `toString` es comunmente sobrescrito:
 - `public String toString()`
 - Retorna una representación en string del objeto.

Sobreescritura de toString

```
public class Elemento
{
    ...

    public String toString()
    {
        String linea1 = titulo +
                        " (" + duracion + " minutos)";
        if(loTengo) {
            return linea1 + "*\n" + "      " +
                comentario + "\n");
        } else {
            return linea1 + "\n" + "      " +
                comentario + "\n");
        }
    }
    ...
}
```

Sobreescritura de toString

- A menudo se pueden omitir, explícitamente, métodos `print` de una clase:

```
System.out.println(elemento.toString());
```

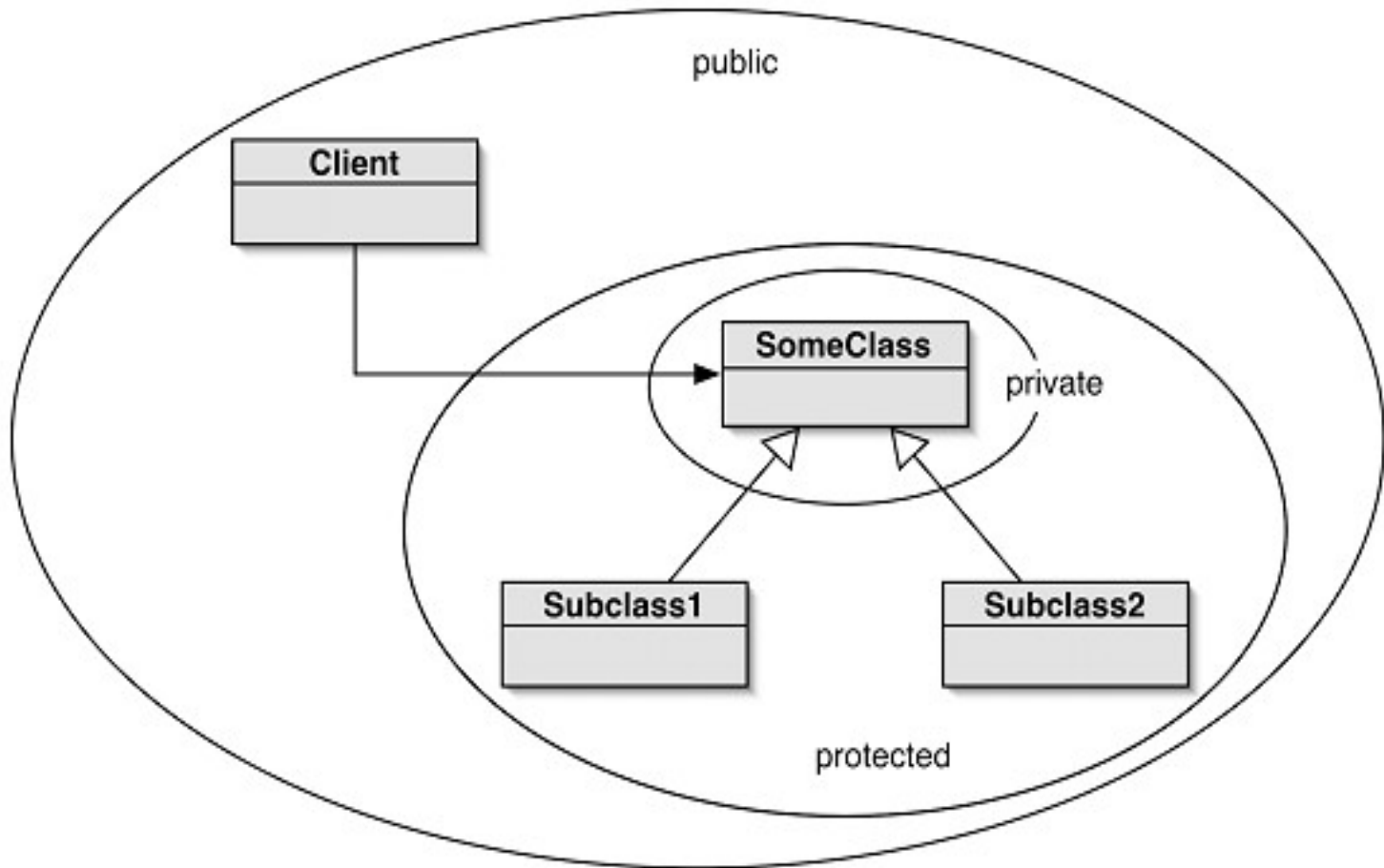
- Los llamados a `println` con solo un objeto resultan automáticamente en una llamada a `toString`:

```
System.out.println(elemento);
```

Acceso protegido

- El acceso privado en la superclase puede muy restrictivo para una subclase.
- La relación de herencia mas cercana es suportada por el acceso *protected* (protegido).
- El acceso protegido es más restrictivo que el acceso público.
- Aún así se recomienda mantener los campos *private*.
 - Defina *protected* setters y getters.

Niveles de acceso



Repaso

- El tipo declarado de una variable es su tipo estático.
 - Los compiladores verifican los tipos estáticos.
- El tipo de un objeto es su tipo dinámico.
 - Los tipos dinámicos son usados en tiempo de ejecución.
- Los métodos pueden sobreescribirse en una subclase.
- La búsqueda del método se inicia con el tipo dinámico.
- El acceso *protected* soporta la herencia.