Escuela de Ingeniería Informática
Universidad de Oviedo

Escuela de Ingeniería Informática

UNIVERSIDAD DE OVIEDO

Unidad 3: Más sobre la herencia

Metodología de la Programación

Curso 2021-2022

© *Candi Luengo Díez , Francisco Ortín Soler y José Manuel Redondo López, Alberto M. Fernandez Álvarez*

Bibliografía

- **Programación orientada a objetos con Java. 6th Edición**
  David Barnes, Michael Kölling.
  Pearson Education. 2017

  Capítulo 11: Más sobre la herencia

Principales conceptos

- Tipo estático y dinámico
- Búsqueda dinámica de métodos. Enlace dinámico (Dynamic Binding)
- Redefinir o sobrescribir métodos (Overriding)
- Métodos polimórficos

Recordando el polimorfismo

- Con el Polimorfismo, las variables objeto tienen más de un tipo.
- ¿Cuál es el tipo del objeto al que apunta la referencia `item` ?

```
public void add(Item item) {
    items.add(item);
}
```

- Puede ser `Item`, o `DVD`, o `CD`, o `VideoGame`…
- El polimorfismo añade **una nueva dimensión** en la relación *es de tipo.*

Tipo estático y tipo dinámico

- El **tipo estático** de una variable objeto (referencia) es su tipo declarado en el código fuente.
  - El tipo estático de `item` es `Item`

```
public void add(Item item) {
        items.add(item);// el tipo estático es Item
}
```

- El **tipo dinámico** es el tipo del objeto que está almacenado en la variable (al que apunta la referencia) en un punto específico de ejecución.
- Si se llama al método anterior así:
  - `ml.add(new CD(…)); // el tipo dinámico de Item es CD`
  - `ml.add(new DVD(…));// el tipo dinámico de Item es DVD`

Tipo estático y tipo dinámico

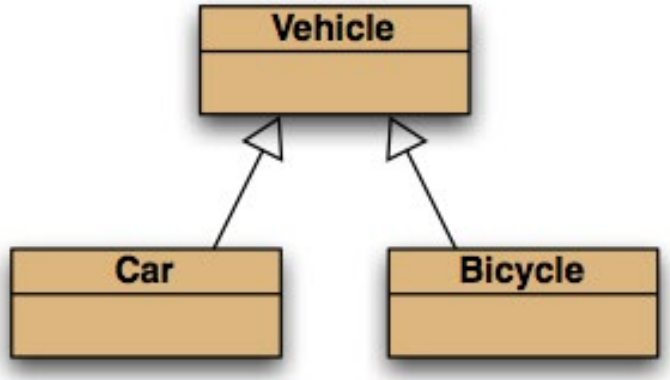
El **compilador** solo hace comprobaciones de tipo estático.

```
Car c1 = new Car();
```

¿Cual es el tipo estático y dinámico de c1?

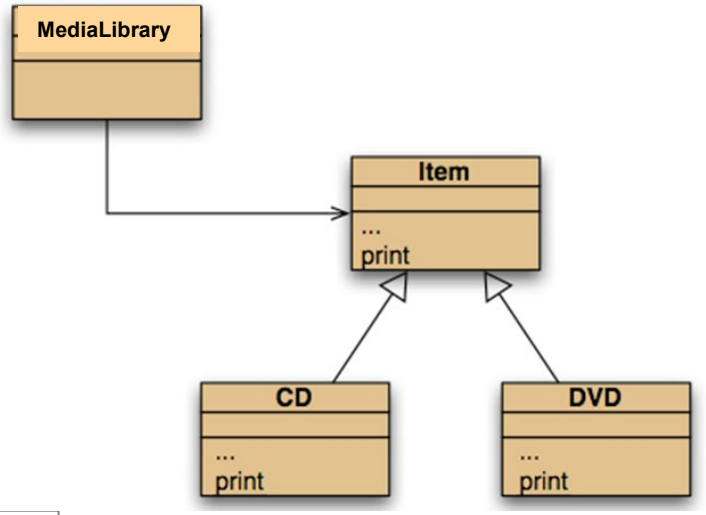
```
Vehicle v1 = new Car();
```

¿Cual es el tipo estático y dinámico de v1?



Volviendo al proyecto DoME

```
public class MediaLibrary {
  public void add(Item item) {
    items.add(item);
  }
```




```
public void list(PrintStream out)
{
    for (Item item: items) {
      item.print(out);
      out.println();
    }
  }
```

¿Qué método print, se ejecutará?

Problema: el método “print”

Creamos 2 objetos con los siguientes datos

```
CD: Título: October
    Duración: 64
    Lotengo: true
    Comentario: Mi álbum favorito de U2
    Interprete: U2
    Número de temas: 16

 DVD: Título: La red social
      Duración: 120 mins
      Lotengo: false
      comentario: “No me gustó demasiado”
      Director: David Fincher
```

Lo que se “puede” mostrar en pantalla si print está en la superclase

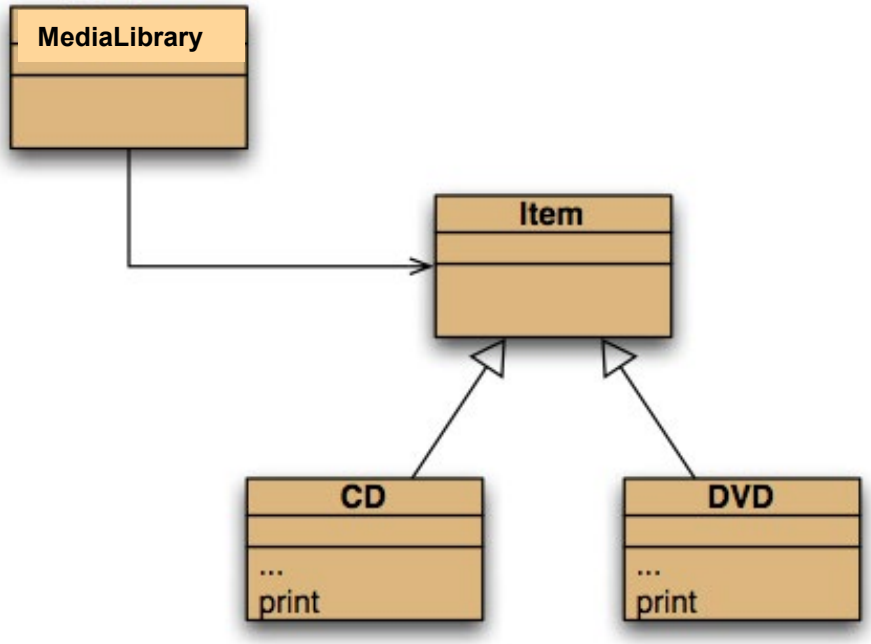
```
    Título: October
    Duración: 64
    Lotengo: true
    Comentario: Mi album favorito de U2

    Título: La red social
    Duración: 120 mins
    Lotengo: false
    comentario: “No me gustó demasiado”
```

Problema: el método “print”

- El método `print` en la clase `Item` solo muestra las propiedades comunes de las clases: `CD` y `DVD`
- La herencia es un camino de un solo sentido:
  - Una subclase **hereda** las **propiedades** de la **superclase**
  - La superclase **no sabe nada** sobre las **propiedades** de sus **subclases**

Primer intento para resolver el problema



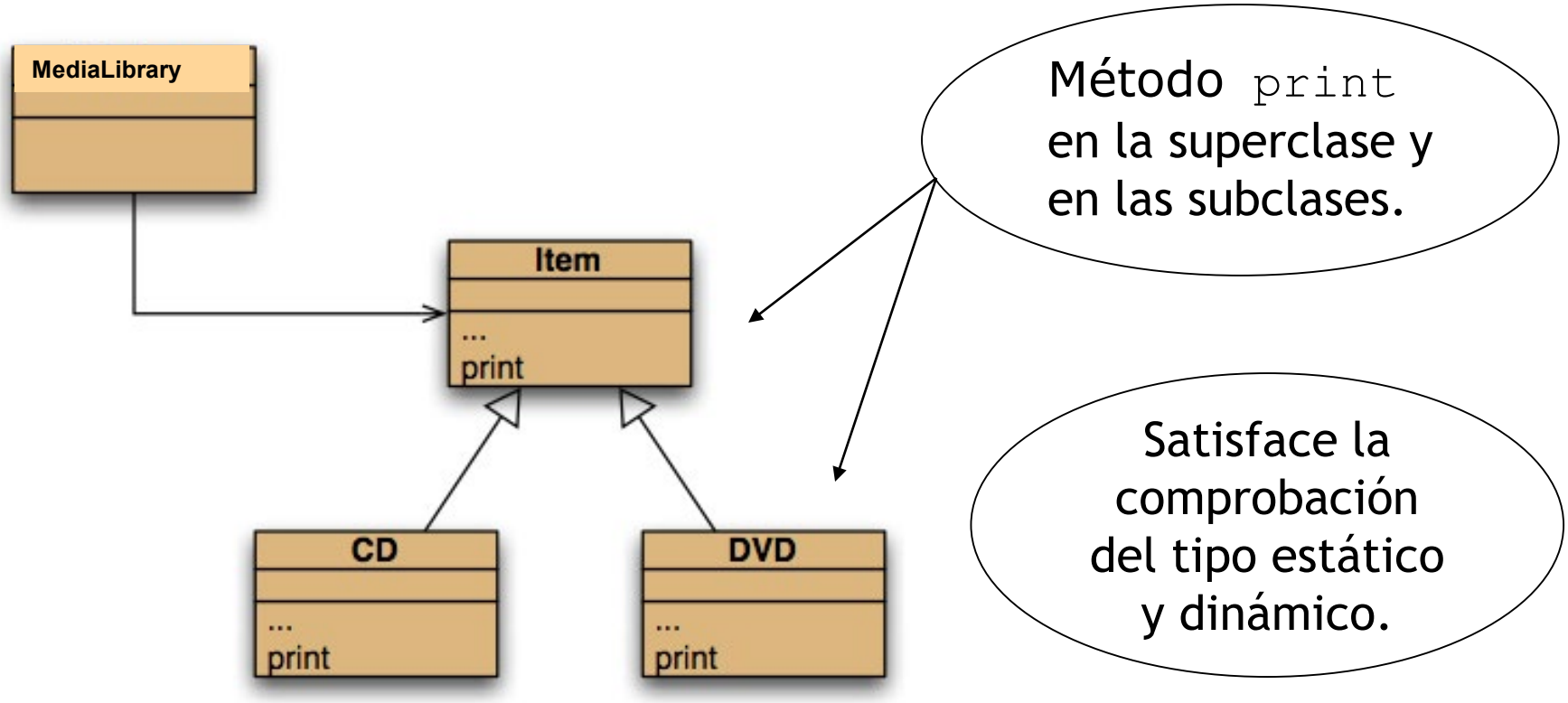
- Situar **`print`** donde se tiene acceso a la información que necesita.
- Cada subclase tiene que tener su propia versión.

Problema

- La clase **`MediaLibrary`** ya no encuentra el método **`print`** en la clase **`Item`**.

```
public void list(PrintStream out){
    for (Item item: items) {
      item.print(out);
      out.println();
    }
}
```

Redefinir el método: la solución



Los objetos de la subclase tienen **dos métodos** con el mismo nombre y la misma signatura (uno heredado y otro propio de la clase). **Redefinición**

¿ Cual de los dos se ejecutará?

¿Que método será llamado?

- El método que se llama es el de su tipo dinámico
  - En vez de llamar a **print** de la clase Item, se busca el de su tipo dinámico y se invoca.
- Este mecanismo se conoce como **Enlace Dinámico** (Dynamic Binding) o **Enlace Tardío** (Late Binding)
- Si queremos evitar el enlace dinámico, el método de la superclase debe tener la palabra `final` (en su signatura)

```
public void list(PrintStream out) {
    for (Item item: items) {
      item.print(out);
      out.println();
    }
  }
```

¿Se invoca a print de CD o a print de DVD?

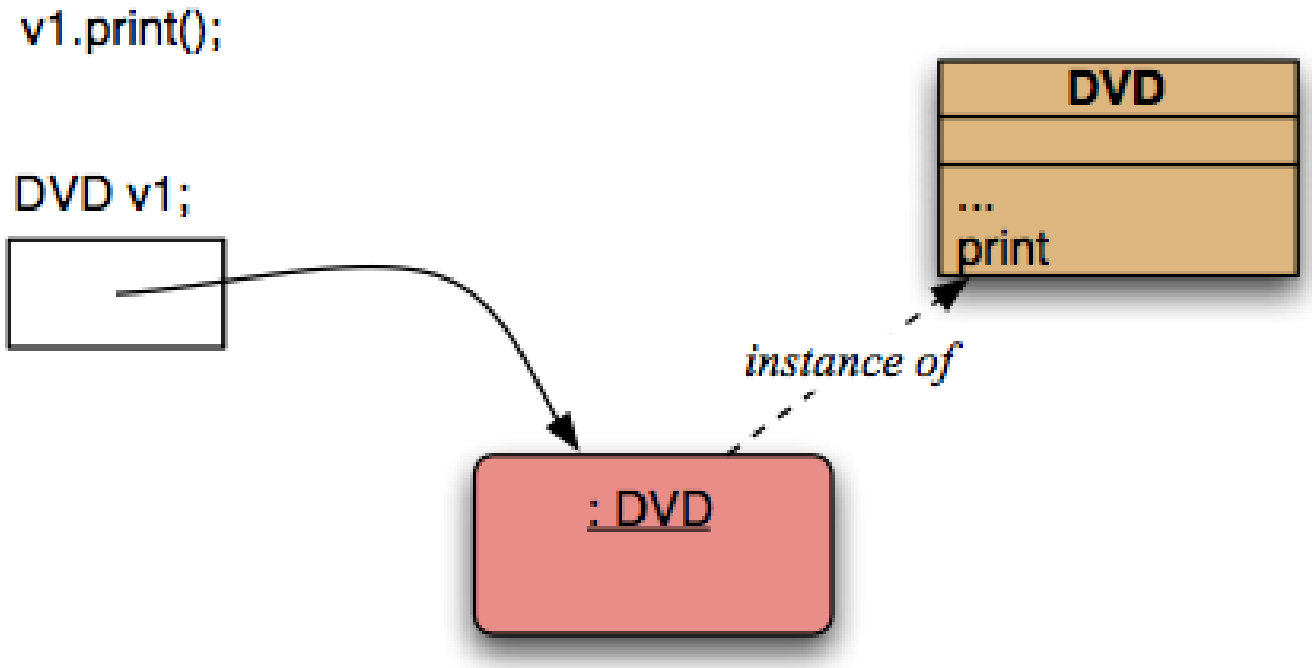
No se conoce en tiempo de compilación.

Es necesario conocer el tipo dinámico para saber que método será llamado y se establece en tiempo de ejecución.

Búsqueda dinámica del método

Cuando no hay herencia.

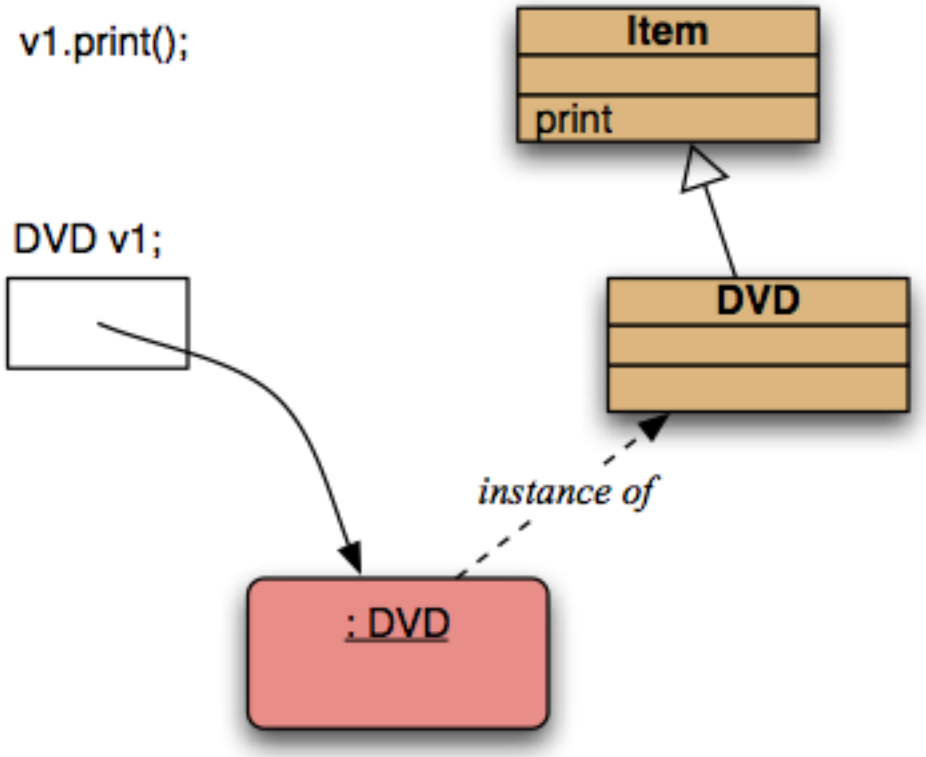
El método seleccionado es obvio.



Búsqueda dinámica del método

Cuando hay herencia pero no redefinición.

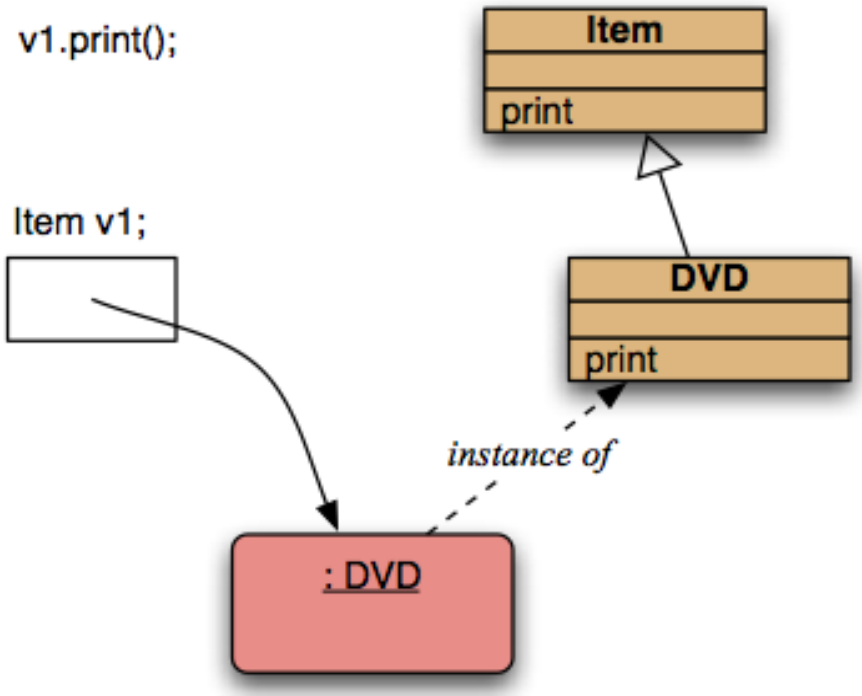
La búsqueda en la jerarquía es ascendente.



Búsqueda dinámica del método

Cuando hay herencia y redefinición con una variable polimórfica.

La 'primera' versión encontrada es la que se ejecuta.



- El método que se invoca es el de su **tipo dinámico** (**Dynamic Type**)
- Este mecanismo se conoce como **enlace dinámico** (**Dynamic Binding**)
- Si se quiere evitar el enlace dinámico, el método de la superclase debe tener en la signatura la palabra `final`

Enlace dinámico (Búsqueda dinámica del método)

- Se tiene acceso a la variable.
- Se encuentra el objeto almacenado en esa variable.
- Se encuentra la clase del objeto.
- Se busca la implementación del método en la clase.
- Si no se encuentra, se busca en la superclase.
- Esto se repite hasta que se encuentra una coincidencia, o se agota la búsqueda en la jerarquía de clases.
- **La redefinición de métodos tienen prioridad**.

Enlace Dinámico

- El método `print` de `DVD` y `CD` **redefine** (**override**) al método `print` de `Item`
  - Se dice que el método print es polimórfico
- La redefinición de métodos (dynamic binding) **requiere que todos tengan la misma signatura**

```
public class Item {
  public void print(PrintStream out) {
    //...
  }
  //...
}
```

```
public class CD extends Item {
  @Override
  public void print(
              PrintStream out) {
    //...
  }
  //...
}
```

```
public class DVD extends Item {
  @Override
  public void print(
              PrintStream out) {
    //...
  }
  //...
}
```

Enlace Dinámico

- Por lo tanto, enlace dinámico y redefinición de métodos es un mecanismo de **especialización**
- El mensaje `print` se ha **especializado** en las clases derivadas
  - Las subclases pueden implementar su propio método print

```
public void list(PrintStream out) {
    for (Item item: items) {
      item.print(out);
      out.println();    }  }
```

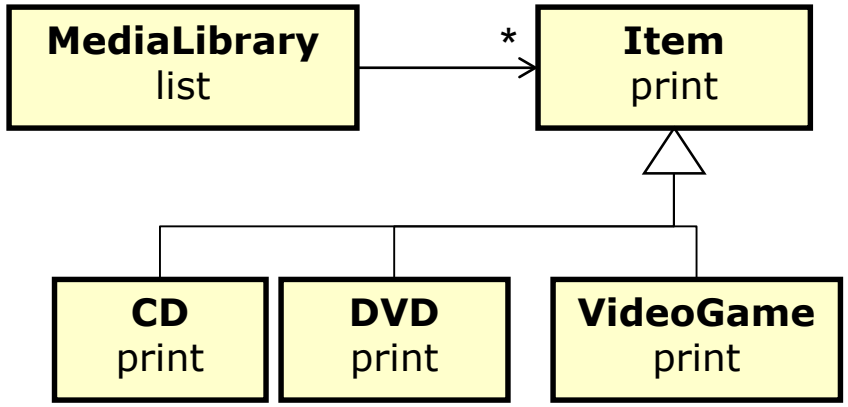
```
public class CD extends Item {
  @Override
  public void print(
            PrintStream out) {
   //...Forma de imprimir del CD
  }
  //...
}
```

```
public class DVD extends Item {
  @Override
  public void print(
            PrintStream out) {
 //...Forma de imprimir del DVD
}
  //...
}
```

Añadiendo un nuevo Item

- ¿Qué sucede si se añade un nuevo item, `VideoGame` y su correspondiente método especializado `print`?
- El método `list` en `MediaLibrary`:
  1. No requiere cambios (homogeneo = generalizado, Polimorfismo)
  2. Llamará al método `print` de la clase `VideoGame` (heterogeneo = especializado, Enlace Dinámico)

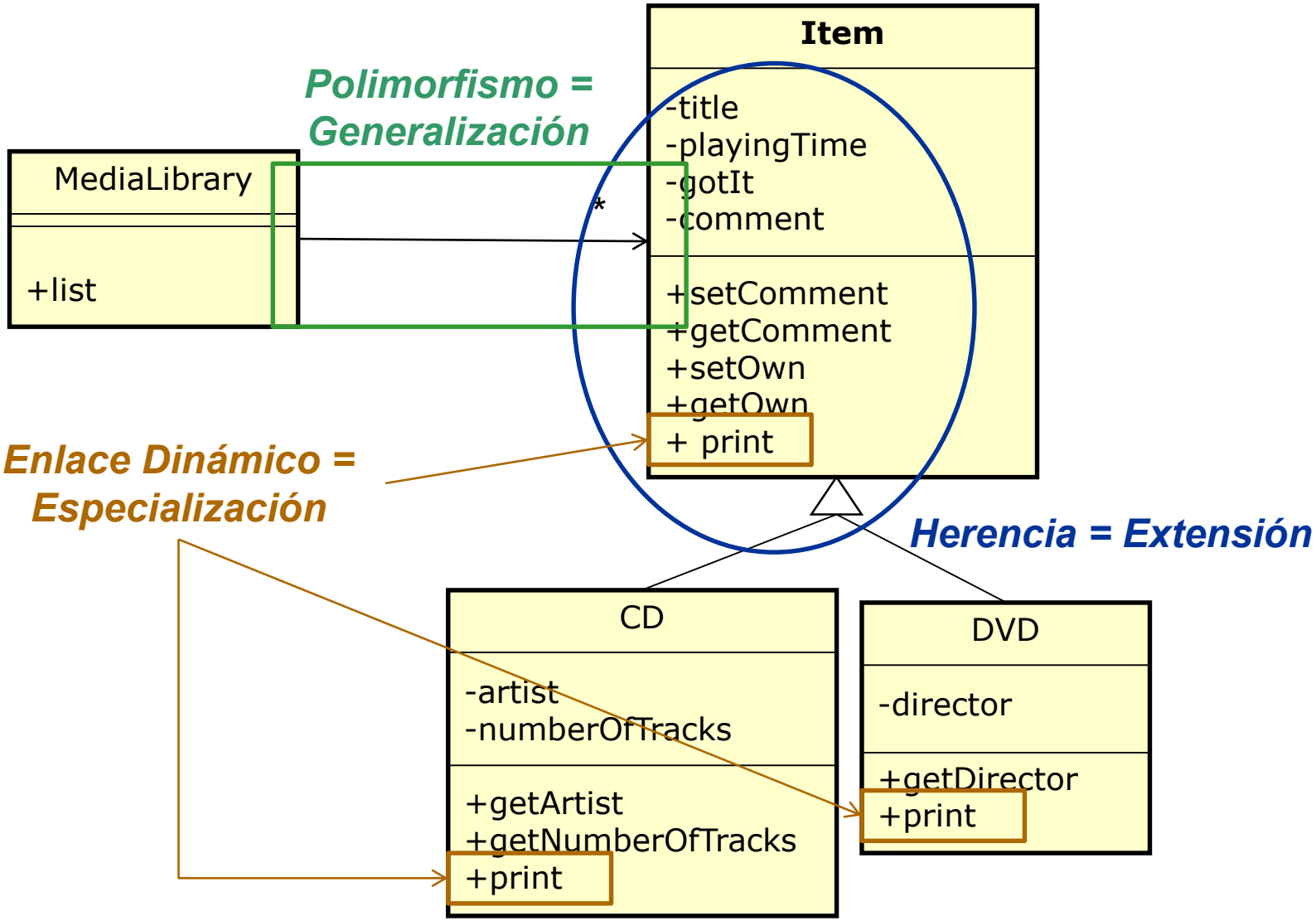
```
public void list(PrintStream out) {
    for (Item item: items) {
      item.print(out);
      out.println();
    }
  }
```



Resumiendo

- Tres conceptos **diferentes**
- En Java, uno depende del otro
  1. Herencia — Extensibilidad
  2. Polimorfismo — Generalización
  3. Enlace Dinámico — Especialización
- Los tres son necesarios para obtener software mantenible.

Resumiendo


Item
-title
-playingTime
-gotIt
-comment
+setComment
+getComment
+setOwn
+getOwn
+ print
Polimorfismo =
Generalización
MediaLibrary
+list
*
Enlace Dinámico =
Especialización
Herencia = Extensión
CD
-artist
-numberOfTracks
+getArtist
+getNumberOfTracks
+print
DVD
-director
+getDirector
+print

Duplicación del código


```
public class Item {
  public void print(PrintStream out) {
    out.print(title + " (" + playingTime + " mins)");
    if(gotIt) out.println("*");
    else out.println();
    out.println(" " + comment);  }  }
```

¿Se puede evitar este código duplicado?

```
public class CD extends Item {
  @Override
  public void print(PrintStream out) {
    out.print(getTitle() + " (" + getPlayingTime() + " mins)");
    if(getOwn()) out.println("*");
    else  out.println();
    out.println(" " + getComment());
    out.println(" The artist: " + artist);
    out.println(" Tracks: " + numberOfTracks);  }  }
```

```
public class DVD extends Item {
  @Override
  public void print(PrintStream out) {
    out.print(getTitle() + " (" + getPlayingTime() + " mins)");
    if(getOwn()) out.println("*");
    else out.println();
    out.println(" " + getComment());
    out.println(" Director: " + director);     }  }
```

Llamada a **super** en los métodos

- **Un método redefinido** (de una superclase) puede ser llamado desde el método que lo redefine (de una subclase).
- Hay que usar la **referencia super** de la forma:

```
super.nombreMetodo(parámetros)
```

  - La lista de parámetros puede estar vacía.
  - La llamada **super** puede estar en cualquier lugar del método (no tiene que estar al principio).
  - No se generará ninguna llamada **super** automática.
  - El método de la subclase sustituye (oculta) la versión de ese mismo método contenido de la superclase.

Duplicación del código


```
public class Item {
  public void print(PrintStream out) {
    out.print(title + " (" + playingTime + " mins)");
    if(gotIt) out.println("*");
    else out.println();
    out.println(" " + comment);  }  }
```

```
public class CD extends Item {
  @Override
  public void print(PrintStream out) {
    super.print(out);
    out.println(" The artist: " + artist);
    out.println(" Tracks: " + numberOfTracks);  }  }
```

```
public class DVD extends Item {
  @Override
  public void print(PrintStream out) {
    super.print(out);
    out.println(" Director: " + director);    }  }
```

Métodos de la clase **`Object`**

- Los métodos de la clase `Object` son heredados por todas las clases.
- Algunos de esos métodos pueden ser redefinidos: `toString`, `equals`, `hahsCode`

```
publica String toString()
public boolean equals(Object obj)
public int hashCode()
```

- El método `toString` devuelve una cadena de caracteres (un string) representando al objeto.
- El método `equals` indica si dos objetos son iguales o no.
- El método `hashCode` devuelve un entero que representa un objeto. Los objetos tienen valores hashCode distintos.

Métodos **equals** y **hashCode**

- Igualdad de referencias con el operador ==
  - Si dos variables hacen referencia al mismo objeto
- Igualdad de contenidos con el método equals
  - Si dos objetos son iguales internamente (por contenido).
- Hay que redefinir el método **equals** heredado de la clase **Object** ya que solo comprueba igualdad de referencias.
- Si se redefine el método **equals** hay que redefinir el método **hashCode**.
  - Dos objetos que sean iguales, según determine una llamada a **equals** deben devolver valores idénticos de **hashCode**

La redefinición de equals y hashCode se detalla en el siguiente capítulo

Método **toString**

- En los métodos `System.out.print` y `System.out.println` si el parámetro de estos métodos no es un objeto `String`, el método invoca automáticamente al método `toString` de dicho objeto.
  - **`System.out.println(item.toString());`**
- Puede ser sustituido por:
  - **`System.out.println(item);`**

Revisión

- El uso del **polimorfismo**, está dirigido a obtener generalización.
- Una vez obtenida la generalización, es necesario que algunos métodos sean especializados.
  - Y serán llamados teniendo en cuenta el tipo dinámico del objeto que recibe el mensaje.
- El **Enlace Dinámico** ofrece esta especialización
- Estos métodos se dice que son polimórficos
- Cada implementación **reemplaza** (overrides) al de su superclase.
  - La signatura del método debe ser exactamente la misma.