

# Curso de Programación para Internet con Java EE (Parte V - Proyectos)

## T02 UML Modeling

© Antonio Santos Ramos 2018

14

## P00. Abstracción



### ■ Abstracción

- Capacidad que permite representar las características esenciales (atributos y propiedades) de un objeto sin preocuparse de las restantes características (no esenciales).




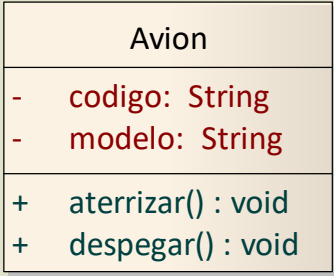
La abstracción se centra en las características esenciales de algún objeto, en relación a la perspectiva del observador.

© Antonio Santos Ramos 2018

15

# P.OO. Metodología



Análisis OO	Diseño OO	Programación OO
		<pre>public class Avion{     private String modelo;     private String codigo;      public void aterrizar(){         ...     }     public void despegar(){         ...     } }</pre>

UML 1.1:	11/97
UML 1.4.2	01/05 (estándar)
UML 1.5:	03/03
UML 2.0:	08/05
UML 2.5:	06/15

UML



## ■ UML (Lenguaje Unificado de Modelado)

<http://www.uml.org/>

Lenguaje gráfico para especificar, visualizar y documentar esquemas de sistemas de software orientado a objetos.

- Controlado por el grupo de administración de objetos (OMG).
- Es el estándar de descripción de esquemas de software.
- Incorpora 14 diagramas: de clase, de secuencia, de estado, de colaboración, de actividad...

<http://holub.com/uml/>  
<http://www.uml-diagrams.org>

<http://docs.kde.org/stable/es/kdesdk/umbrello/uml-basics.html>  
<http://www.tutorialspoint.com/uml/>

- NOTA: Todos los diagramas/gráficos usados en el curso serán UML

# Diagramas UML.

## Diagramas de estructura

Diagrama de clases	Diagrama de objetos	Diagrama de paquetes	Diagrama de despliegue	Diagrama de estructura compuesta	Diagrama de componentes
<ul style="list-style-type: none"> <li>Describe los diferentes tipos de objetos en un sistema y las relaciones existentes entre ellos.</li> <li>Dentro de las clases muestra las propiedades y operaciones, así como las restricciones de las conexiones entre objetos.</li> </ul>	<ul style="list-style-type: none"> <li>Foto de los objetos de un sistema en un momento del tiempo.</li> <li>Obsoletos a partir de UML 2.5</li> </ul>	<ul style="list-style-type: none"> <li>Muestra la estructura y dependencia entre paquetes, los cuales permiten agrupar elementos (no solamente clases) para la descripción de grandes sistemas.</li> </ul>	<ul style="list-style-type: none"> <li>Muestra la relación entre componentes o subsistemas software y el hardware donde se despliega o instala.</li> </ul>	<ul style="list-style-type: none"> <li>Descompone jerárquicamente una clase mostrando su estructura interna.</li> </ul>	<ul style="list-style-type: none"> <li>Muestra la jerarquía y relaciones entre componentes de un sistema software.</li> </ul>

# Diagramas UML.

## Diagramas de Comportamiento

<b>Diagrama de casos de uso</b>					
<ul style="list-style-type: none"> <li>Permite capturar los requerimientos funcionales de un sistema.</li> </ul>					
*					
<b>Diagrama de estado</b>					
<ul style="list-style-type: none"> <li>Permite mostrar el comportamiento de un objeto a lo largo de su vida.</li> </ul>					
<b>Diagrama de actividad</b>					
<ul style="list-style-type: none"> <li>Describe la lógica de un procedimiento, un proceso de negocio o workflow.</li> </ul>					
<b>Diagramas de interacción</b>					
<ul style="list-style-type: none"> <li>Describen cómo los grupos de objetos colaboran para producir un comportamiento. Son los siguientes:</li> </ul>					
<b>Diagrama de secuencia</b>	<b>Diagrama de comunicación</b>	<b>Diagrama de colaboración</b>	<b>Diagrama de interacción</b>	<b>Diagrama de tiempo</b>	
<ul style="list-style-type: none"> <li>Muestra los mensajes que son pasados entre objetos en un escenario.</li> </ul>	<ul style="list-style-type: none"> <li>Muestra las interacciones entre los participantes haciendo énfasis en la secuencia de mensajes.</li> </ul>	<ul style="list-style-type: none"> <li>(Sólo en UML 1.x)</li> <li>Muestra las interacciones organizadas alrededor de los roles.</li> </ul>	<ul style="list-style-type: none"> <li>Se trata de mostrar de forma conjunta diagramas de actividad y diagramas de secuencia.</li> </ul>	<ul style="list-style-type: none"> <li>Pone el foco en las restricciones temporales de un objeto o un conjunto de objetos.</li> </ul>	
*					

# P00. Clases



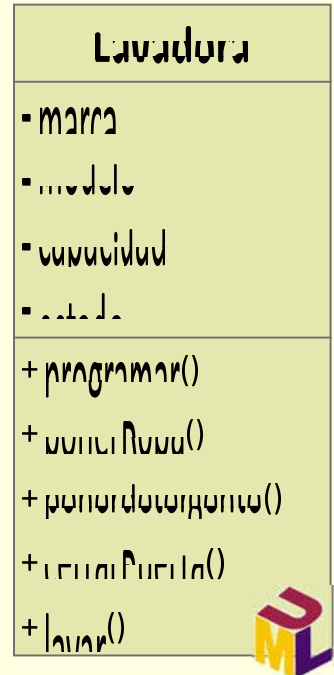
## ■ Clase

- Plantilla o molde para construir objetos.
- Una clase contiene
  - **atributos** (estado): propiedades comunes a los objetos
  - **métodos** (comportamiento): servicios que proporcionan todos los objetos de esa clase para operar con ellos
- En Java se define mediante la palabra **class**.

```
[public] class MyClase {
    // cuerpo de la clase
}
...
MyClase unObjeto;
```

© Antonio Santos Ramos 2018

```
public class Lavadora{
    public String marca;
    public int capacidad;
}
```



20

# P00. Objetos



## ■ Objeto

- Un objeto (o **instancia** de una clase) es una encapsulación abstracta de información, junto con los métodos o procedimientos para manipularla.
- Informalmente es “una clase con valores concretos”

## ■ En programación un objeto contendrá

- Operaciones que definen su comportamiento
- Atributos con valores concretos que definan su estado

lavad1:Lavadora

```
marca: Balay
modelo: BL123
capacidad: 5
estado: centrifugando
```

© Antonio Santos Ramos 2018

21

# Diagrama de clases UML (I/II)

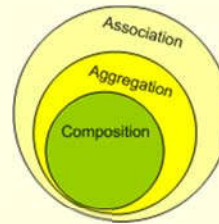


## ■ Diagrama de clases

- Diagrama estructural usado en la parte de diseño que muestra las clases, interfaces y sus relaciones
- Comunica un aspecto de la vista de diseño estática
  - Contiene los elementos esenciales para comprender ese aspecto
  - No debe dejar nociones semánticas “al aire”

## ■ Relaciones UML empleadas entre clases

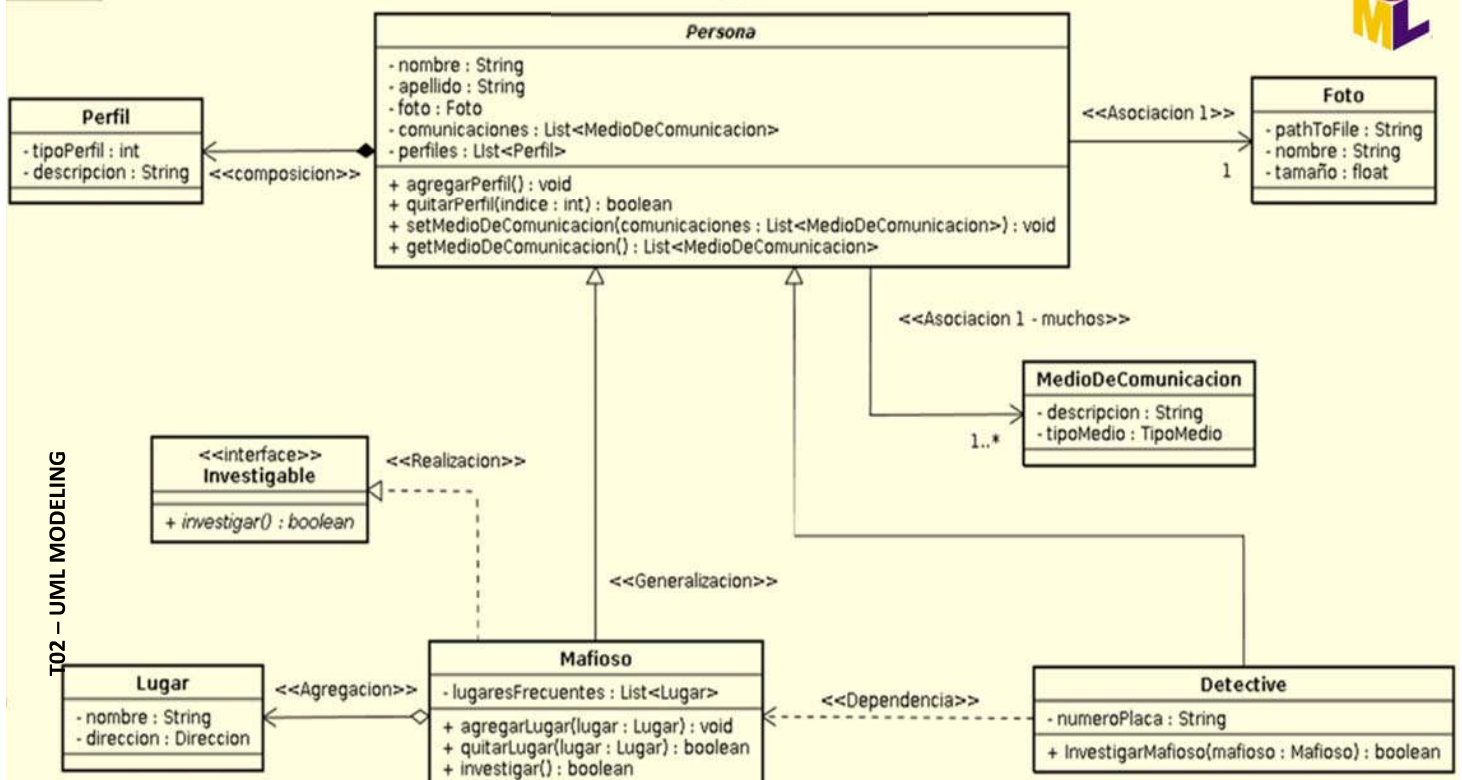
- Dependencia
- Asociación
  - Agregación
  - Composición
- Generalización
- Realización



Además una asociación implica siempre una dependencia

<http://idiotechie.com/uml2-class-diagram-in-java/>

## Diagrama de clases UML (II/II). Ejemplo

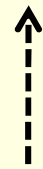




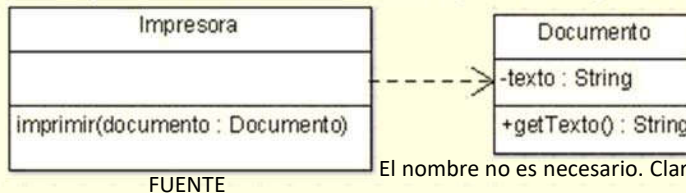


# P00. Relaciones (I/X)

## Dependencia (I/II)



- **Dependencia** (...depende de ...)(...restringido a...) (...usa...)
- Relación semántica temporal entre cliente y proveedor (objetivo)
  - La fuente depende, de alguna forma, del objetivo.
  - Si el objetivo cambia, el cliente (la fuente) se ve afectada



La impresora necesita el documento



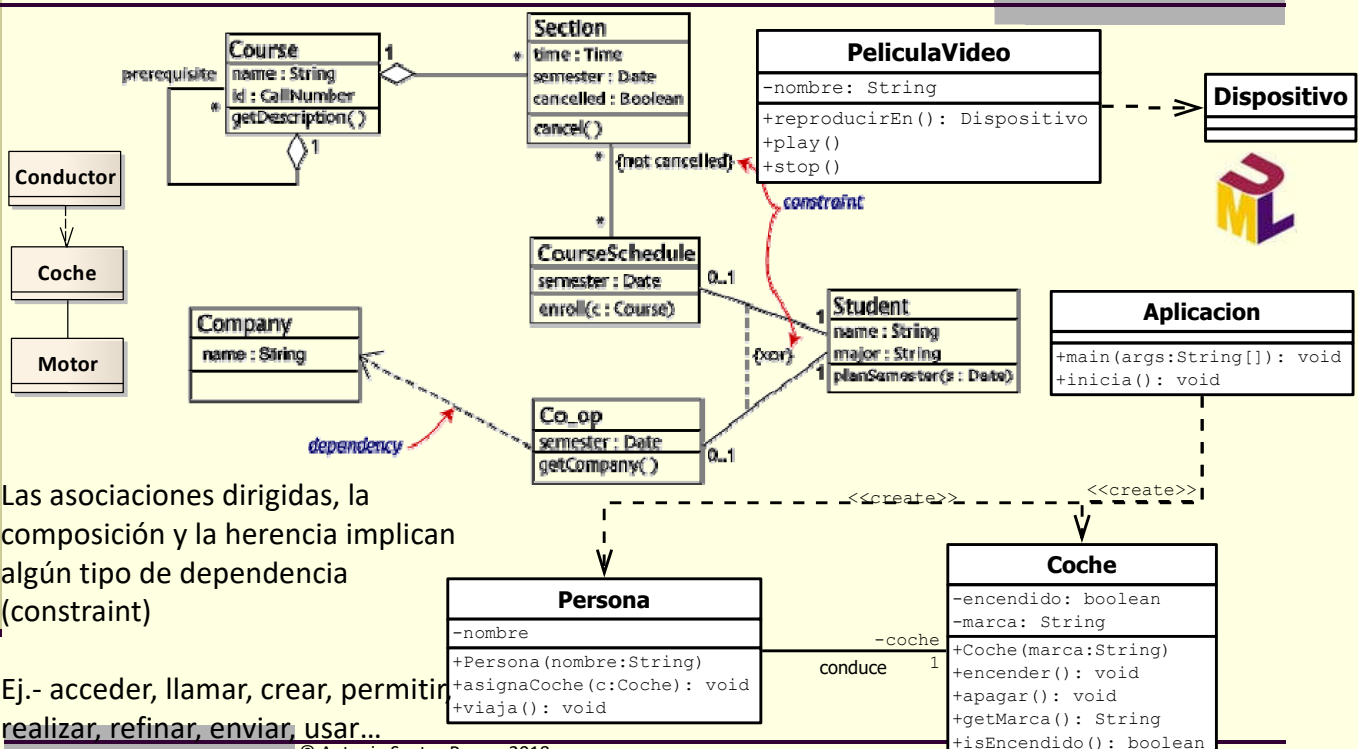
FUENTE

El nombre no es necesario. Claramente "usa"

- La dependencia existe si una clase:
  - Tiene una variable local basada en otra clase
  - Tiene una referencia directa a un objeto
  - Tiene una referencia indirecta a un objeto (ej.- a través de los parámetros de operaciones)
  - Usa una operación de una clase estática

# P00. Relaciones (II/X)

## Dependencia (II/II)



Las asociaciones dirigidas, la composición y la herencia implican algún tipo de dependencia (constraint)

Ej.- acceder, llamar, crear, permitir realizar, refinar, enviar, usar...



# POO. Relaciones (III/X)

## Asociación (I/IV)

### "has...a"

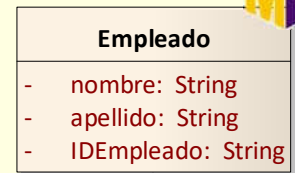


#### ■ Asociación (...conoce...)(...trabaja para...)

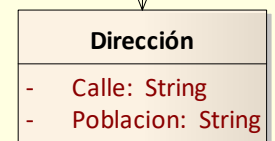
- Relación estructural permanente (roles) que describe conexiones entre objetos.
  - El estado de la clase A contiene la clase B
  - Un objeto accede a atributos y métodos de otro objeto.

```
public class Empleado {
    private String nombre;
    private String apellido;
    private String IDEmpleado;
    private Direccion dirCasa;
}

public class Direccion {
    private String Calle;
    private String Poblacion;
}
```



dirCasa



Significa que el objeto Empleado contiene en su estado un objeto de tipo Direccion

© Antonio Santos Ramos 2018

26

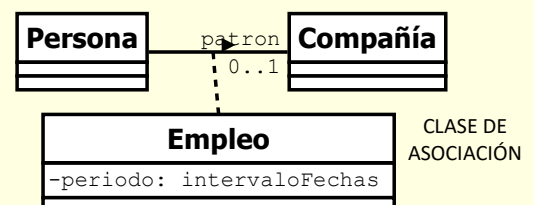
<https://www.codeproject.com/Articles/330447/Understanding-Association-Aggregation-and-Composit>

# POO. Relaciones (IV/X)

## Asociación (II/IV)



- Puede ser "one-way" o bidireccional (no flecha) y tener multiplicidad
- Un atributo también puede ser una clase asociada a otra (en la línea indicas el nombre y un guión)



<http://blog.jotadeveloper.es/tag/multiplicidad/>

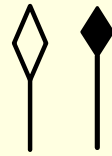
© Antonio Santos Ramos 2018

27



# POO. Relaciones (V/X)

## Asociación (III/IV)



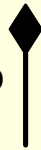
### ■ Agregación (...es accesorio de...) (... usa ...)

- Tipo de asociación que define que un objeto es parte de otro objeto (es un atributo de ese objeto). Comparto info

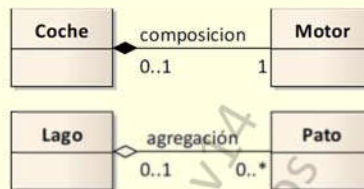


### ■ Composición (...es imprescindible para..) (... pertenece ...)

- Asociación que define que un objeto es parte imprescindible de otro objeto (es decir, es un atributo de ese objeto)
- El Objeto B no tiene sentido sin el objeto A. No lo comparto
- La eliminación de uno supone la eliminación del otro



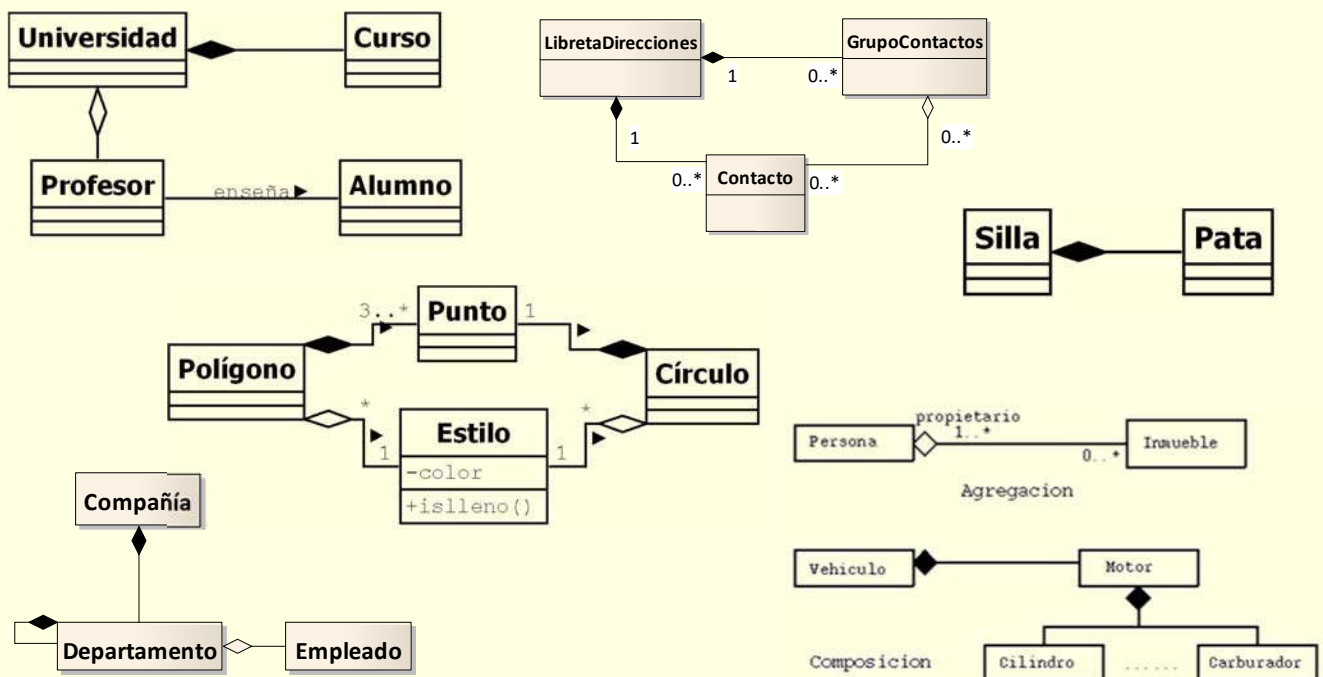
```
public class Coche {
    private Motor m;
}
```



```
public class Lago {
    private Pato[] p;
}
```

# POO. Relaciones (VI/X)

## Asociación (IV/IV)







# POO. Relaciones (VII/X)

## Generalización(I/II)

### “is...a”

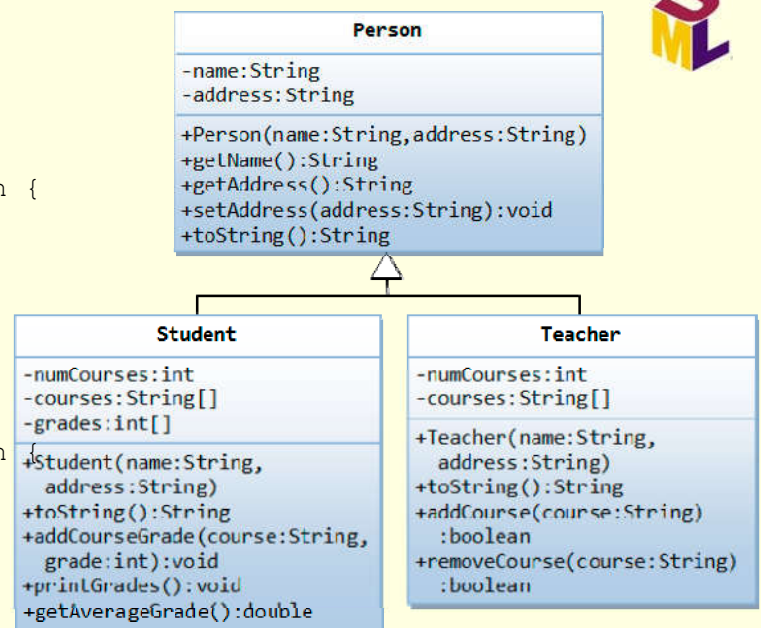


- Definición (alias Herencia) (“es un”)
  - Relación en el cual el hijo (elemento especializado) se basa en la especificación del padre (elemento generalizado)
- ¿Cómo funciona?
  - Las subclases “heredan” atributos y métodos de las superclases (excepto constructores).
    - Se agrupan las partes comunes en una misma clase (clase padre).
    - Se crean otras clases (hijas) con las partes no comunes.
- En Java
  - Heredan de la clase padre mediante la palabra **extends**
  - No existe herencia múltiple (simulada con interfaces)

# POO. Relaciones (VIII/X)

## Generalización(II/II)

```
public class Person {  
    private String name;  
    private String address;  
    ...  
}  
  
public class Student extends Person {  
    private int numCourses;  
    private String[] courses;  
    private int[] grades;  
    ...  
}  
  
public class Teacher extends Person {  
    private int numCourses;  
    private String[] courses;  
    ...  
}
```





# POO. Relaciones (IX/X)

## Realización (I/II)



### Realización

- Una clase especifica un contrato que la otra clase debe cumplir.

### Definiciones

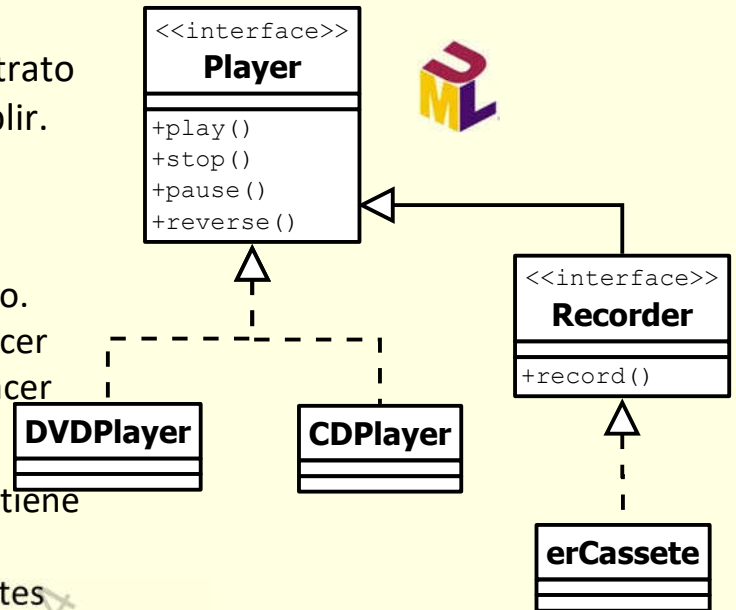
#### Método abstracto

- Método no implementado. Especifica QUÉ se va a hacer pero no CÓMO se va a hacer

#### Interface

- Es una clase que sólo contiene métodos abstractos y sus propiedades son constantes

### Semántica: ...se comporta como...



# POO. Relaciones (X/X)

## Realización (II/II)

### Definición para Java

- Informalmente es una clase con **todos** sus métodos abstractos
- Sólo detalla declaraciones de métodos. No implementa

```
01. import java.awt.Rectangle;
02.
03. public interface Drawable {
04.     int MAX_WIDTH = 1024;
05.
06.     public void draw();
07.     Rectangle getDimensions();
08.     void resize(int w, int h);
09. }
```

```
public class dibujo implements Drawable
```

- Una interface sería similar a un clase sin atributos ¿¿??
- No puede ser instanciada. Sólo puede ser implementada... es decir la clase que la implemente asume su comportamiento