

Módulo profesional entornos de  
desarrollo

| UD 9 –  
Diagrama de  
clases

# contenido

- UML
- DIAGRAMAS DE CLASES
- CLASES, ATRIBUTOS Y MÉTODOS
- RELACIONES ENTRE CLASES (ASOCIACIONES)

# Diagrama de clases

Un **diagrama de clases** es una estructura estática que muestra el conjunto de clases que forman parte de un sistema, junto con las relaciones existentes entre ellas.

Los diagramas de clases son el pilar básico del modelado con **UML**, siendo utilizados tanto para mostrar lo que el sistema puede hacer (análisis), como para mostrar como puede ser construido (diseño).

# UML –Unified Modeling Language-

¿Qué es UML?

UML es ante todo un lenguaje. Un lenguaje proporciona un vocabulario y una reglas para permitir una comunicación. En este caso, este lenguaje se centra en la representación gráfica de un sistema.

# UML –Unified Modeling Language-

Aunque UML está pensado para **modelar sistemas complejos** con gran cantidad de software, el lenguaje es lo suficientemente expresivo como para modelar sistemas que no son informáticos, como flujos de trabajo (workflow) en una empresa, diseño de la estructura de una organización y por supuesto, en el diseño de hardware.

# Modelado de software

El **modelado** es el **análisis y diseño** de aplicaciones software antes de escribir el código.

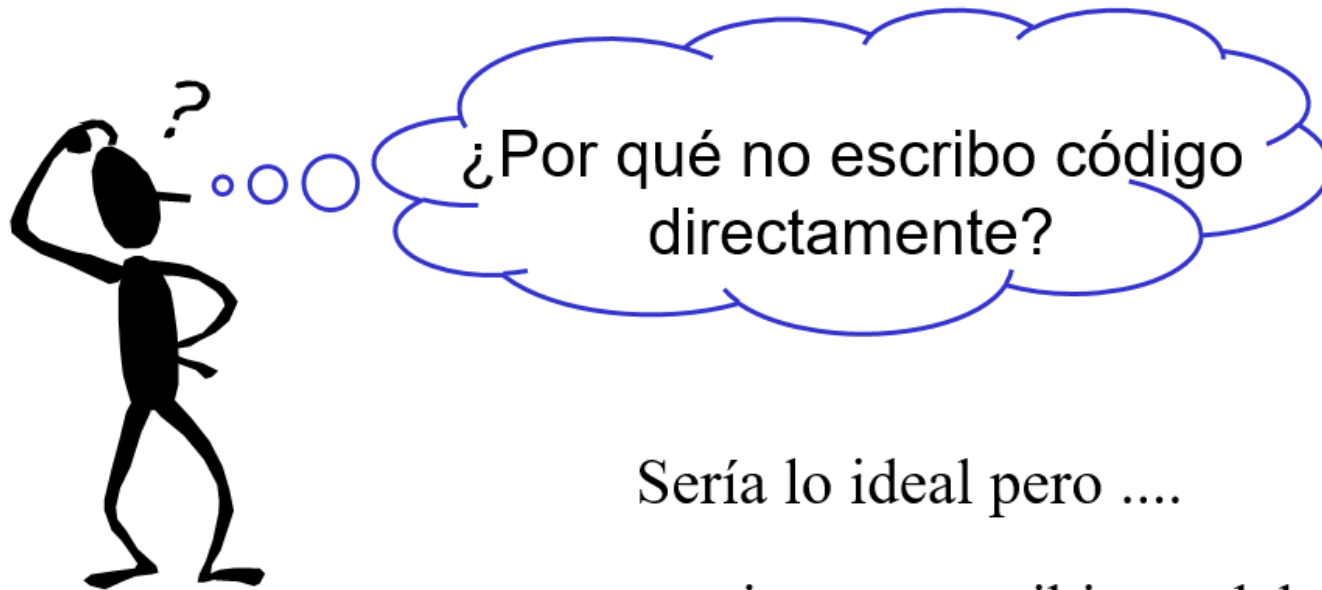
Se crean un conjunto de modelos (“planos del software”) que permiten especificar aspectos del sistema como los requisitos, la estructura y el comportamiento.

# Utilidad del modelado

“Una empresa software con éxito es aquella que **produce de manera consistente software de calidad** que satisface las necesidades de los usuarios”

“**El modelado es la parte esencial** de todas las actividades que conducen a la producción de software de calidad”

# Utilidad del modelado



Sería lo ideal pero ....

.... necesitamos escribir modelos,

aunque la mayoría de desarrolladores  
todavía no practican el modelado



# Utilidad del modelado

Los objetivos de UML son muchos, pero se pueden sintetizar sus funciones:

- **Visualizar:** UML permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.
- **Especificar:** UML permite especificar cuáles son las características de un sistema antes de su construcción.
- **Construir:** A partir de los modelos especificados se pueden construir los sistemas diseñados.
- **Documentar:** Los propios elementos gráficos se utilizan como documentación del sistema desarrollado que pueden servir para su futura revisión.

# ¿Qué es un modelo de software?

*Un modelo es una descripción de un aspecto del sistema, expresada en un lenguaje bien definido.*

# Características de uml

El UML es un lenguaje para construir modelos; no guía al desarrollador en la forma de realizar el análisis y diseño orientados a objetos ni le indica cuál proceso de desarrollo adoptar.

# UML –Unified Modeling Language-

Un **modelo UML** esta compuesto por tres clases de bloques de construcción:

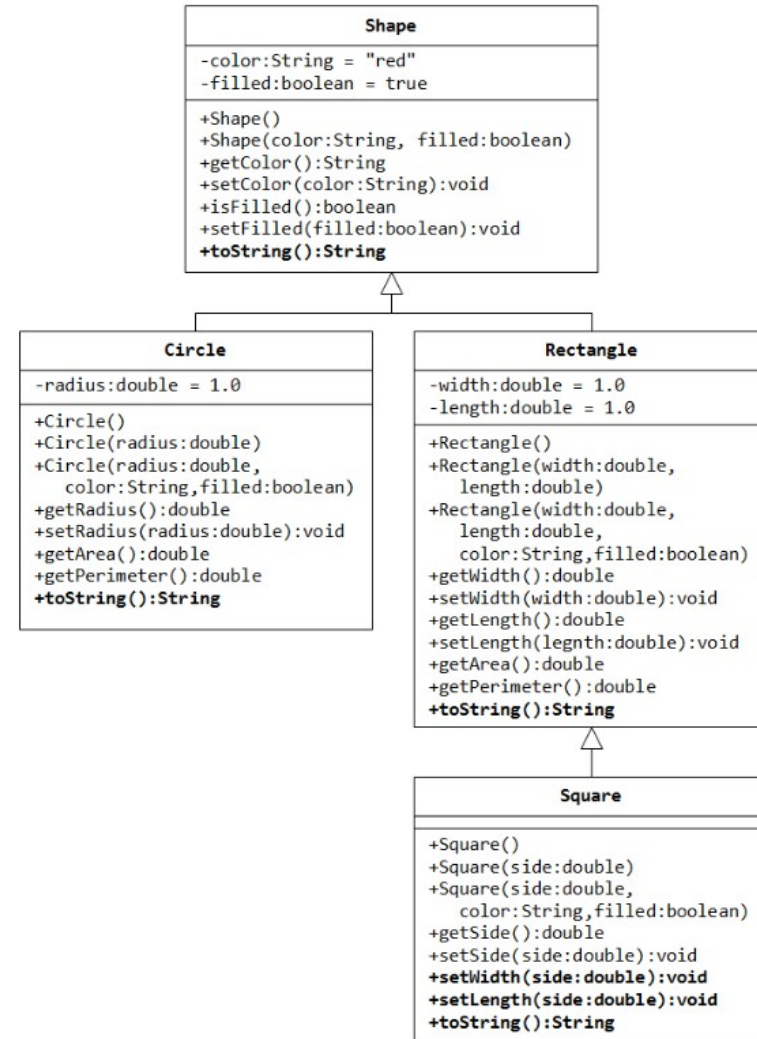
- **Elementos:** Los elementos son abstracciones de cosas reales o ficticias (objetos, acciones, etc.)
- **Relaciones:** relacionan los elementos entre sí.
- **Diagramas:** Son colecciones de elementos con sus relaciones.



**Diagramas no  
son modelos**

# UML –Unified Modeling Language-

Por ejemplo, el **diagrama de clases** muestra un conjunto de clases, interfaces y sus relaciones. Éste es el diagrama más común a la hora de describir el diseño de los sistemas orientados a objetos.



# diagrama

Un **diagrama** es la representación gráfica de un conjunto de elementos con sus relaciones.

En concreto, un diagrama ofrece una **vista del sistema a modelar**. Para poder representar correctamente un sistema, UML ofrece una amplia variedad de diagramas para visualizar el sistema desde varias perspectivas.

# diagrama

Surgen así los diferentes tipos de diagramas, 13 en total en la última versión de UML, la 2.5.1 (diciembre de 2017):

- **Diagramas de estructura**

- Diagrama de clases
- Diagrama de estructuras compuestas
- Diagrama de componentes
- Diagrama de despliegue
- Diagrama de objetos
- Diagrama de paquetes

- **Diagramas de comportamiento**

- Diagrama de casos de uso
- Diagrama de actividad
- Diagramas de interacción
  - Diagrama de secuencia
  - Diagrama de comunicación o colaboración
  - Diagrama de visión global de la interacción
  - Diagrama de tiempo
- Diagrama de maquina de estados

# diagrama

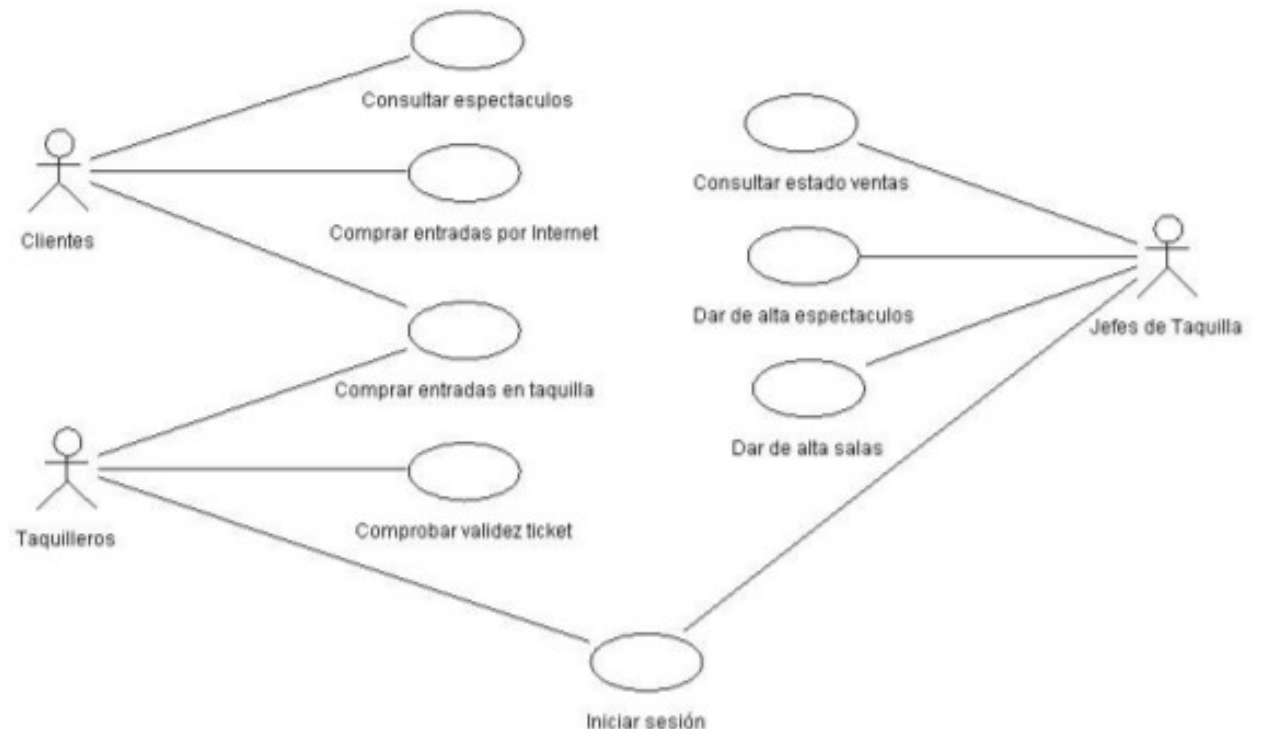
Los diagramas más interesantes (y los más usados) son los de **casos de uso**, **clases** y **secuencia**, por lo que nos centraremos en éstos.



# Diagrama de casos de uso

El **diagrama de casos de uso** representa gráficamente los casos de uso que tiene un sistema. Se define un **caso de uso** como cada interacción supuesta con el sistema a desarrollar, donde se representan los requisitos funcionales. Es decir, se está diciendo lo **que** tiene que hacer un sistema.

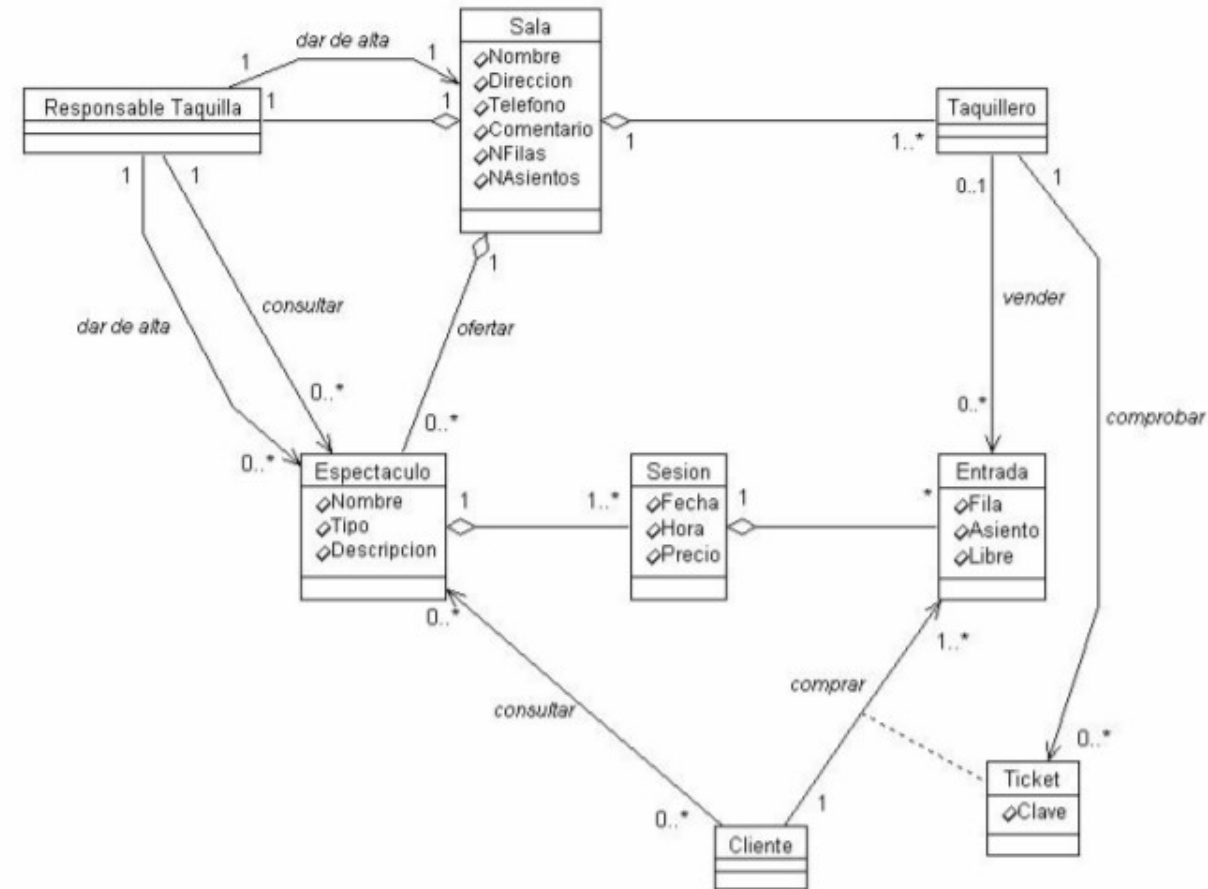
En la figura se muestra un ejemplo de casos de uso, donde se muestran tres actores (los clientes, los taquilleros y los jefes de taquilla) y las operaciones que pueden realizar (sus roles).



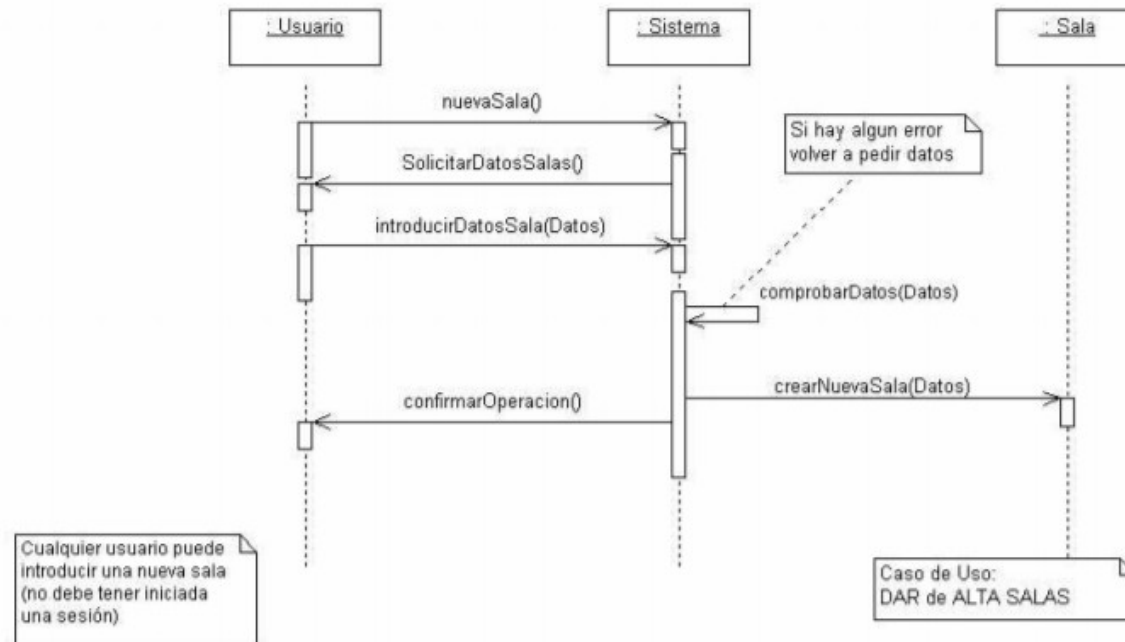
# Diagrama de clases

El **diagrama de clases**, muestra un conjunto de clases, interfaces y sus relaciones. Éste es el diagrama más común a la hora de describir el diseño de los sistemas orientados a objetos.

En la figura muestran las clases, sus atributos y las relaciones de una posible solución al problema de la venta de entradas.



# Diagrama de secuencia



El diagrama de **secuencia** se muestra la interacción de los objetos que componen un sistema de forma temporal. Siguiendo el ejemplo de venta de entradas, la figura 5 muestra la interacción de crear una nueva sala para un espectáculo.

# Diagrama de clases

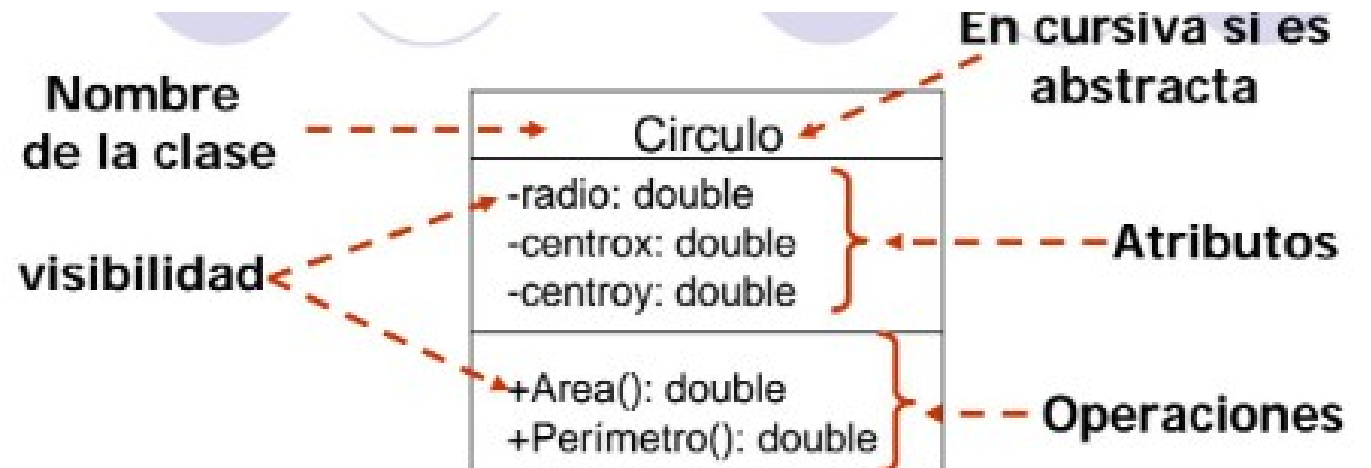
Los **elementos** más comunes de un **diagrama de clases** son:

- Clases: descripciones de objetos con **características**, **comportamiento** comunes.
- Atributos: describen las **características** de los objetos de la clase
- Métodos: describe genéricamente un servicio que puede ser requerido a cualquier objeto de una clase para que muestre un **comportamiento**
- Asociaciones (relaciones): es un término formal para definir las **relaciones entre clases**.

# CLASES, ATRIBUTOS Y MÉTODOS

En los diagramas de clases, las **clases** son los elementos fundamentales del diagrama. Se representan mediante un **rectángulo** con tres divisiones internas.

- El nombre de la clase va arriba.
- Los atributos en medio.
- Los métodos se ubican abajo.

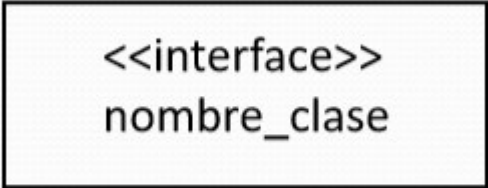


# CLASES, ATRIBUTOS Y MÉTODOS

Las **interfaces** son un tipo especial de clases que se usan para modelar una serie de métodos que pueden ser ofrecidos por diferentes clases

Se representan de la misma manera que las clases, salvo una diferencia:

En el primer recuadro donde aparece el nombre de la clase, se antepone <<interface>>.

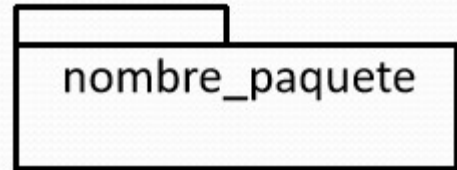


```
classDiagram
    class Interface {
        <<interface>>
        nombre_clase
    }
```

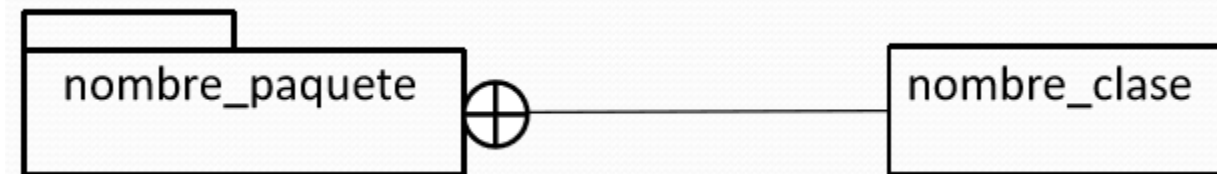
The diagram shows a rectangular box representing a class or interface. Inside the box, the text "<<interface>>" is on the top line, and "nombre\_clase" is on the bottom line. This notation is used in UML to distinguish interfaces from regular classes.

# CLASES, ATRIBUTOS Y MÉTODOS

- Los **paquetes** son un elemento organizador que permiten agrupar las clases. Se pueden modelar las relaciones entre clases o entre paquetes.
- Su representación suele ser el de una carpeta, dentro de la cual se ubican las clases:



- En algunos sistemas se representa del mismo tamaño que las clases y se asocia con ellas con un conector de la siguiente forma



# CLASES, ATRIBUTOS Y MÉTODOS

**Atributos:** Representan las propiedades de los objetos de una clase. Cada atributo de un objeto tiene un valor que pertenece a un dominio de valores determinado.

- La sintaxis de un atributo es:

[ visibilidad ] nombreAtributo [ : tipo ] [ [multiplicidad] ] [ = valorInicial ]

Visibilidad: ámbito en el que el atributo es visible (+: público, -: privado, #: protegido o ~: paquete)
Nombre: identifica el atributo dentro de la clase (no se puede repetir)
Tipo: describe el tipo del valor del atributo (entero, real, ...)
Multiplicidad: describe el número mínimo y máximo de valores posibles de un atributo
Valor inicial: describe el valor que se asigna por defecto a un atributo cuando se instancia un objeto
Alcance: describe si es un atributo de instancia o de clase



# CLASES, ATRIBUTOS Y MÉTODOS

Además:

- Los atributos de clase se escriben subrayados
- Por defecto, la visibilidad es pública y la multiplicidad es [1..1]

Ejemplos:

nombrePersona

origen: Punto

idUnico: Long

+provincia: String =“Zaragoza”

-segundoApellido: Integer [0..1]

#prioridad: Entero =1

# CLASES, ATRIBUTOS Y MÉTODOS

Persona
nombrePersona
primerApellido
segundoApellido
fechaNacimiento

Dirección
calle: String
localidad: String [0..1]
provincia: String [0..1] ="Zaragoza"
CP: String

Libro
+título: String [0..1]
+autor: String [0..*]
+fechaPublicación: date

Producto
#id: Float
+nombre: String
+precio: Float

# CLASES, ATRIBUTOS Y MÉTODOS

## Métodos

[ visibilidad ] nombreOperación [ (listaParámetros) ] [ :tipoRetorno ]

- Por defecto, la visibilidad es pública
- Los métodos de clase se escriben subrayados

Ejemplos:

mover( )

ponerAlarma(t:temperatura)

copiasEnEstantería( ):Entero

+añadirCurso( c:Curso):Booleano

-compactar( )

#comprobarErrores( )

# CLASES, ATRIBUTOS Y MÉTODOS

Figura
origen
mover( ) redimensionar( ) visualizar( )

Ventana
origen:Punto tamaño:Vector
abrir( ) cerrar( ) mover( )

SensorTemperatura
+reiniciar( ) +ponerAlarma(t:temperatura ) +valor( ): temperatura

Transacción
+ejecutar( ) +rollback( ) #prioridad( ) #marcaDeTiempo( )

# CLASES, ATRIBUTOS Y MÉTODOS

- Para indicar que un método o clase son abstractos se escribe en cursiva o se les añade al final la etiqueta `{abstract}`.
- Para indicar que un método, atributo o clase son finales se les añade la etiqueta `{frozen}`.

# instancias

## Instancia de clase (objeto):

describe un objeto de una clase mediante valores de los atributos de la clase. El objeto responde a las operaciones de la clase

## Notación

nombre:clasificador

# instancias

: Persona

Juan : Persona

Persona

nombre: string  
edad: integer

Juan:  
Persona

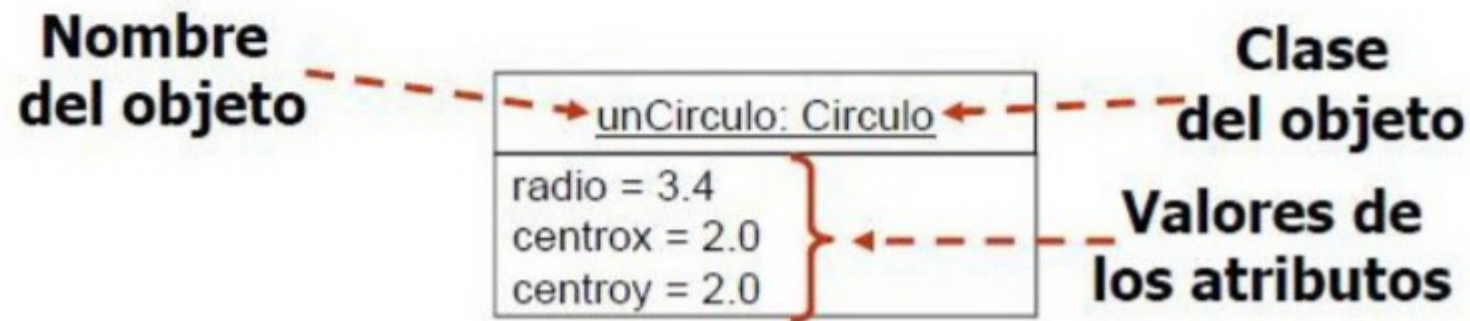
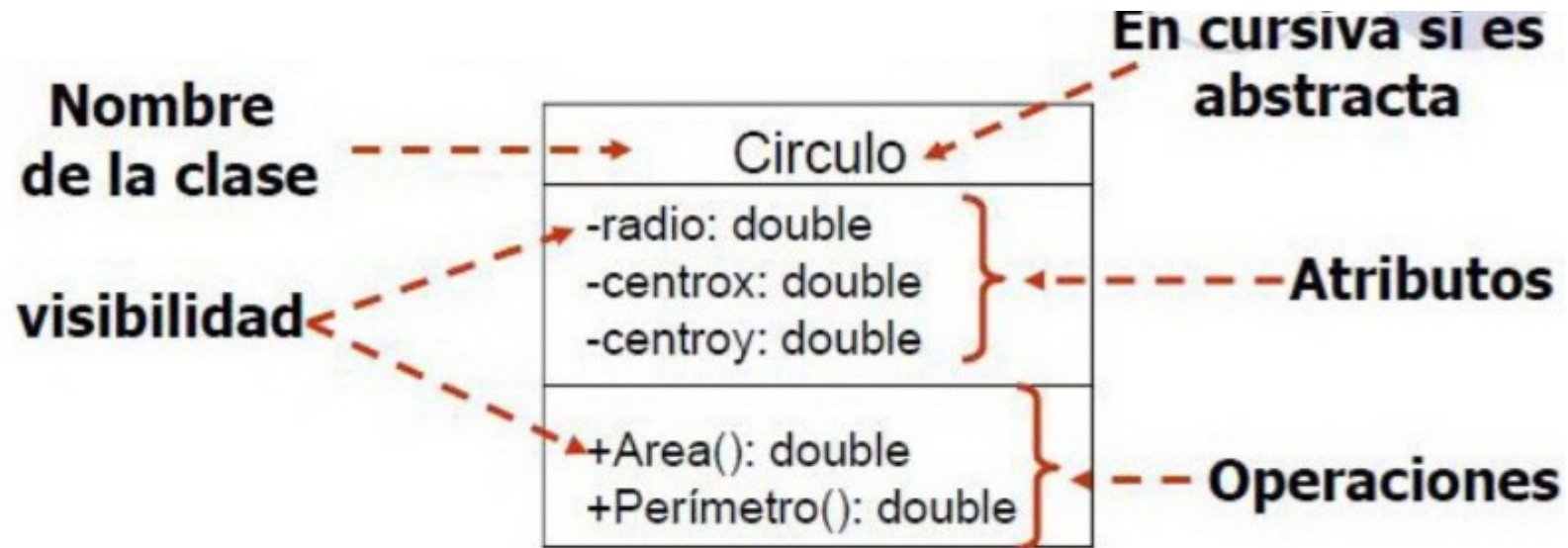
nombre = "Juan"  
edad = 24

Marcos:  
Persona

nombre = "Marcos"  
edad = 52

*Clase con Atributos*

*Objetos con Valores*





# Resumen de notación

Nombre de la clase
+ Atributo público # Atributo protegido - Atributo privado ~Atributo de paquete <u>Atributo de clase</u>
+ Operación pública() # Operación protegida() - Operación privada() ~Operación de paquete <u>Operación de clase()</u>

# relaciones

Las **relaciones** que existen entre las diferentes clases se denominan de forma genérica **asociaciones**.

Nosotros lo vamos a seguir llamando relaciones para no confundir con el tipo de relación que es asociación. Se pueden encontrar diferentes tipos de relaciones entre clases. Estas se pueden clasificar de muchas formas.

- **asociación**
- **agregación**
- **composición**
- **dependencia**
- **generalización**

# relaciones

La **clase A** y la **clase B** están relacionadas sí:

- un objeto de la clase **A** **envía un mensaje a un objeto de la clase B** (es decir, invoca a un método de la clase B)
- un objeto de la clase **A** **crea un objeto de la clase B**
- un objeto de la clase **A** **tiene un atributo cuyos valores son objetos de la clase B** o colecciones de objetos de la clase B
- un objeto de la clase **A** **recibe un mensaje con un objeto de la clase B pasado como argumento** (es decir, se invoca un método de la clase A con un objeto de la clase B como parámetro)

# asociación

**Asociaciones:** Representan las relaciones entre las instancias clases. Existen relaciones de conocimiento entre clases, en las cuales instancias de una clase se relacionan con instancias de otras clases. Permite asociar objetos que instancian clases que colaboran entre sí.

Se representa mediante una **línea continua sin flechas** (en el caso que sea bidireccional).

Las asociaciones pueden tener un **nombre** que describe la naturaleza de la relación



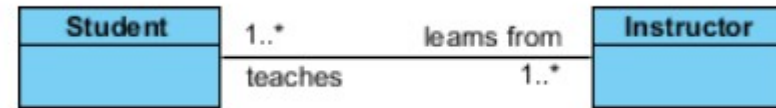
# Elementos de las asociaciones

Las asociaciones pueden tener etiquetas

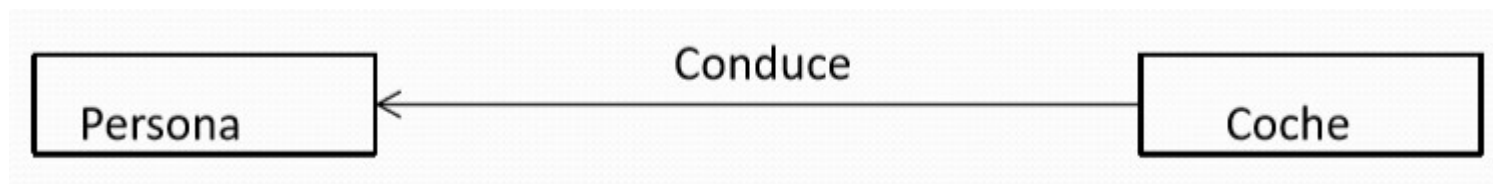
- rol
- navegación
- multiplicidad
- tipo

# Elementos de las asociaciones

- **Rol:** Cada extremo de la asociación puede llevar un nombre. Un rol indica el papel que juega una clase en una asociación.



- **Navegación:** navegación define la posibilidad (o no) de acceder a los objetos de una clase desde otra. Se indica mediante una punta de flecha (unidireccional). Si no se indica nada, la relación se da en ambos sentidos (bidireccional). En las asociaciones unidireccionales una clase está consciente de la otra e interactúa con ella.



Navegación:  
Unidireccional  
Bidireccional  
No especificado.  
No navegable (x)

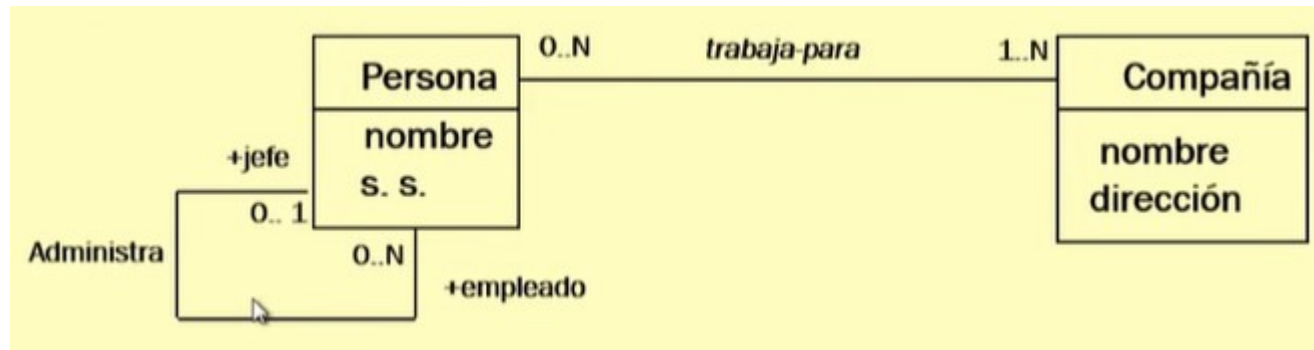
# Elementos de las asociaciones

- **Multiplicidad:** Describe la cardinalidad de la asociación. Es decir cuando objetos de cada clase intervienen en dicha asociación.
- 1: Uno y sólo 1
- 0,1: Cero o uno.
- 0..\*: Cero o muchos.
- 1..\*: 1 o muchos.



# Elementos de las asociaciones

- **Tipo:** Según el número de clases involucradas en la asociación esta puede ser:
  - **Asociación unaria:** Una clase se asocia consigo misma.
  - **Asociación Binaria:** Dos clases que se relacionan
  - **Asociación n-Aria:** Asociación que involucra a más de dos clases

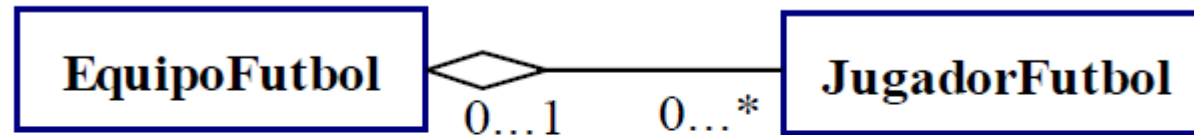




# agregación

**Agregación:** es una asociación que describe una relación entre **un todo y sus partes** de modo que las partes pueden existir por sí mismas

Se denota con un **rombo vacío** en el extremo de la **clase base**.



relación tipo ‘tiene un’

Esto se lee: “Un equipo de fútbol tiene cero o más jugadores de fútbol, y un jugador de fútbol pertenece a un equipo de fútbol o a ninguno”

# Agregación

Se trata de una relación del tipo **todo-parte**. Este tipo de relación implica dos tipos de objetos, el objeto llamado **base** y el objeto que estará **incluido** en el objeto base.

La relación indica que el **objeto base** necesita del **objeto incluido** para poder existir y hacer sus funcionalidades. Si desaparece el objeto base, el o los objetos que se encuentran incluidos en el objeto base no desaparecerán y podrán continuar existiendo con sus funcionalidades propias.

# composición

**Composición:** Asociación de agregación fuerte, en el que la existencia de la parte depende del todo. Es decir si se elimina la clase base, la subclase que lo compone también desaparece.

Se denota con un **rombo relleno** en el extremo de la clase base.



relación tipo 'contiene a'

# composición

Es una relación del tipo todo-parte. Es una relación muy parecida a la relación de asociación de agregación, con la diferencia que hay una **dependencia de existencia** entre el **objeto base** y el objeto (o los objetos) que **está incluido**.

**Es decir, si deja de existir el objeto base, dejará de existir también el o los objetos incluidos.**

# dependencia

- **Dependencia:** Este tipo de relación se representa mediante una flecha discontinua entre dos elementos. El objeto del que sale la flecha se considera un **objeto dependiente**. El objeto al que llega la flecha se considera un **objeto independiente**.

Un cambio en la clase utilizada puede afectar al funcionamiento en la clase utilizadora, pero no al contrario

Es una relación de **conocimiento/uso** entre clases.

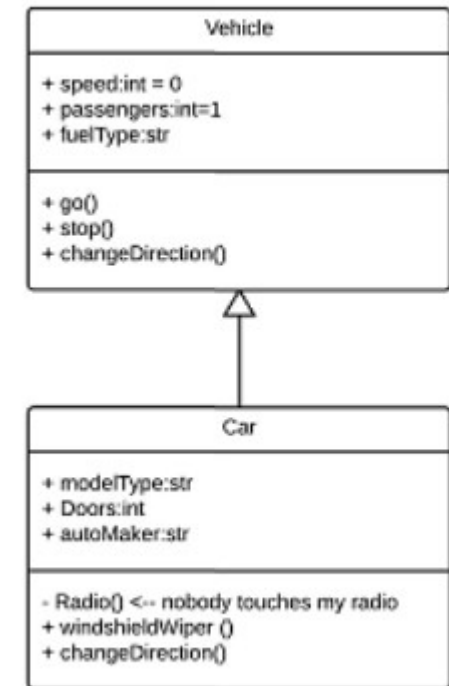


En la practica este tipo de relación se interpreta como que **Impresora** hace uso de **Documento** ya sea instanciándolo directamente, o bien, recibéndolo como parámetro de entrada en uno de sus métodos.

# Especialización/generalización

- **Generalización:** se da entre dos clases donde hay un vínculo que se puede considerar de **herencia**. Una clase es llamada clase *madre* (o superclase). La otra (o las otras) son las llamadas clases *hijas* o subclases, que heredan los atributos y los métodos y comportamiento de la clase *madre*. Este tipo de relación queda especificado mediante **una flecha que sale de la clase hija y que termina en la clase madre**.

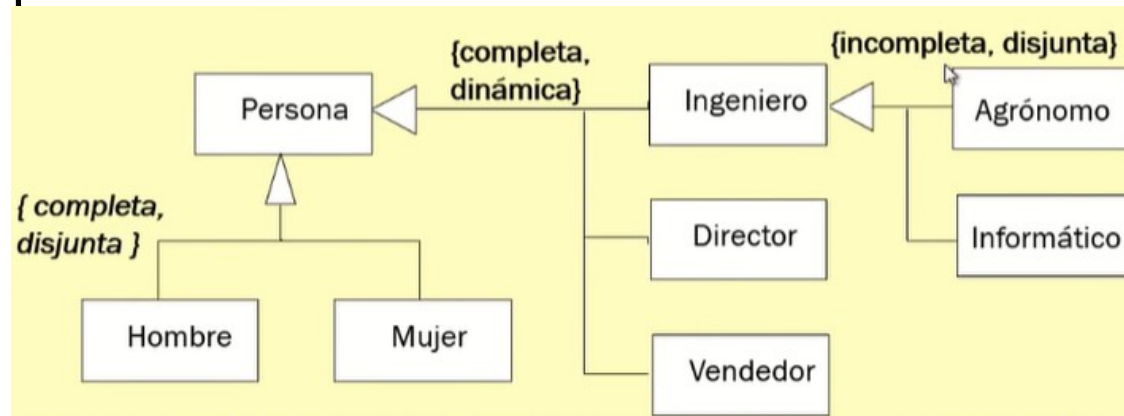
En estas asociaciones no hay multiplicidad ni roles. **Si se trata de una** interfaz **la** línea es discontinua.



# Especialización/generalización

Este tipo de relación tiene dos posibles **propiedades**:

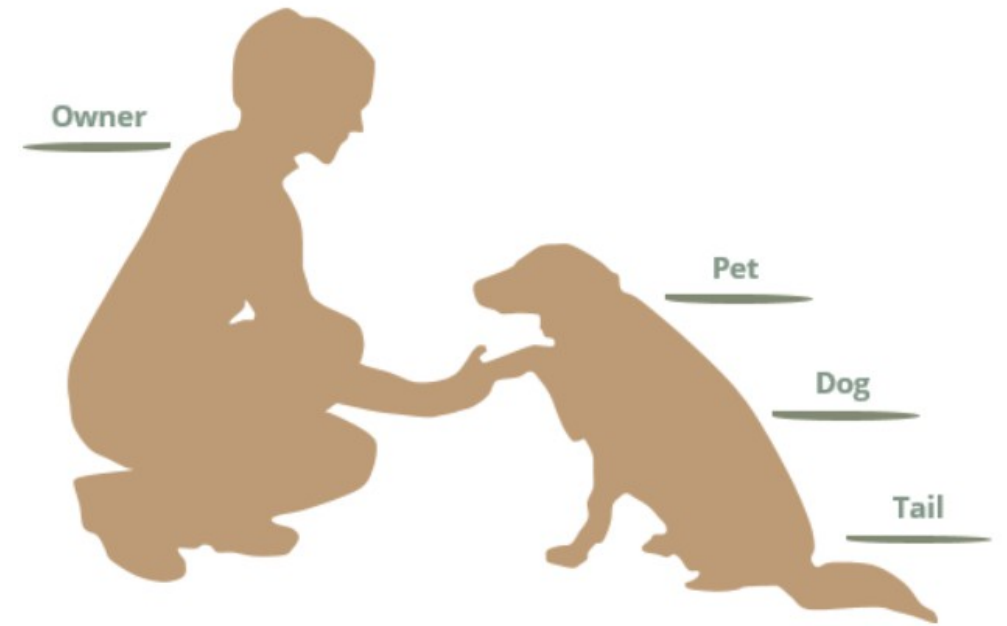
- **Completa/Incompleta:** completa quiere decir que se han especificado todas las subclases posibles.
- **Disjunta/No disjunta:** significa que un objeto de la superclase puede ser solo de un tipo de la subclase. Por ejemplo, una persona puede ser hombre o mujer pero no ambos.



# Asociación-agregación-composición-herencia

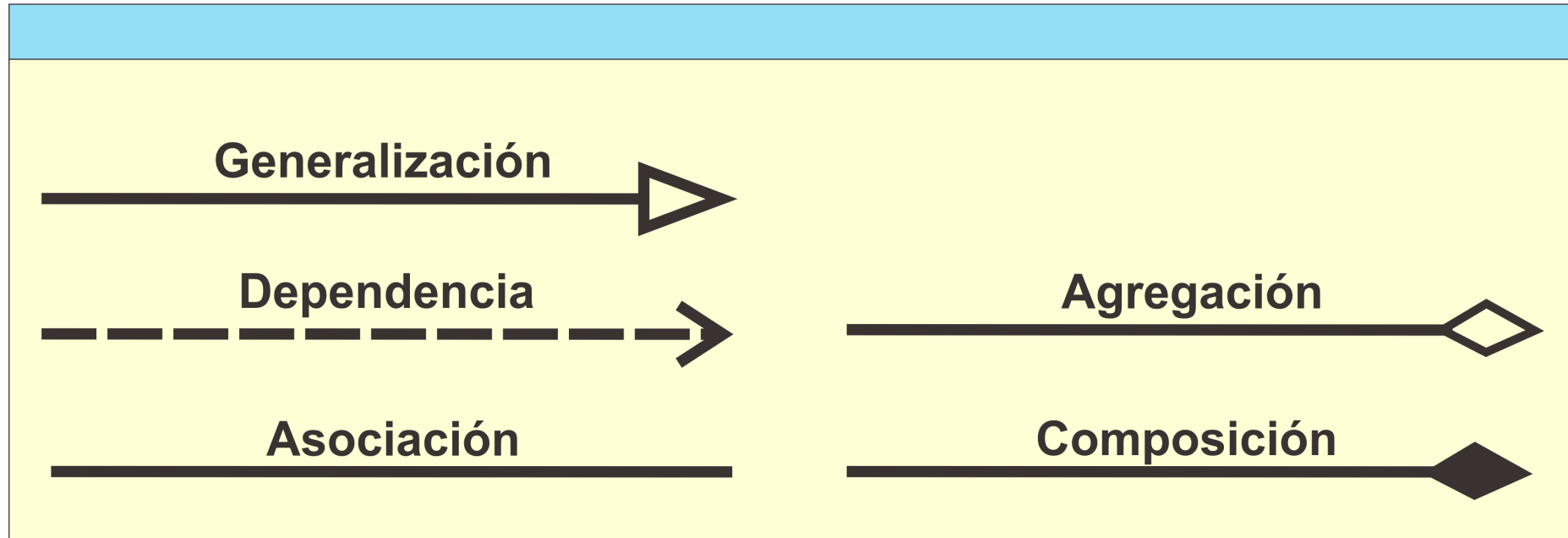
Vemos las siguientes relaciones:

- los **dueños alimentan mascotas, mascotas agradecen a los propietarios** (asociación)
- una **cola es una parte de ambos perros y gatos** (agregación / composición)
- un **gato es un tipo de mascota** (herencia / generalización)

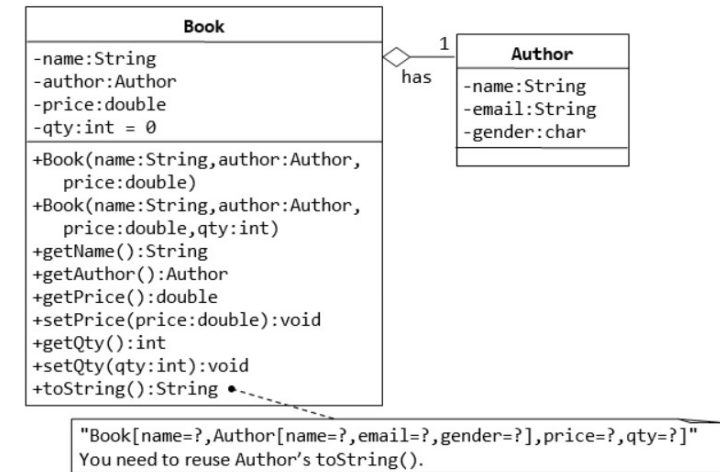




# resumen



# Atributo o asociación



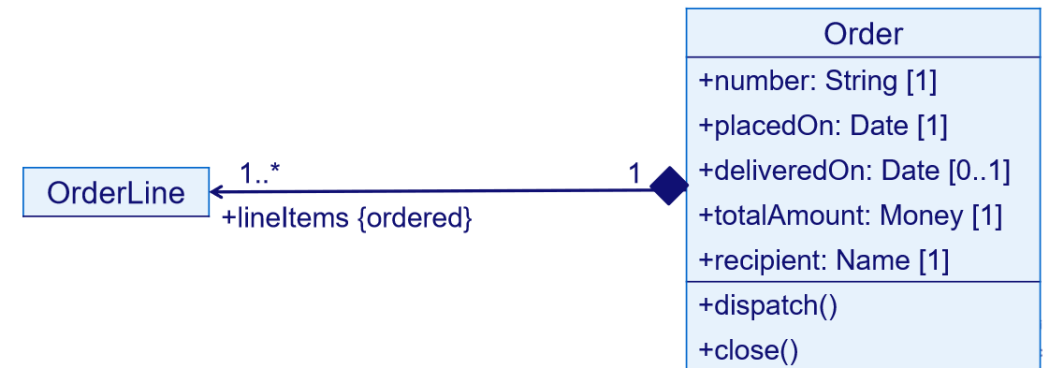
Hay dos formas diferentes en que los **atributos** pueden ser presentados

- atributo
- asociación

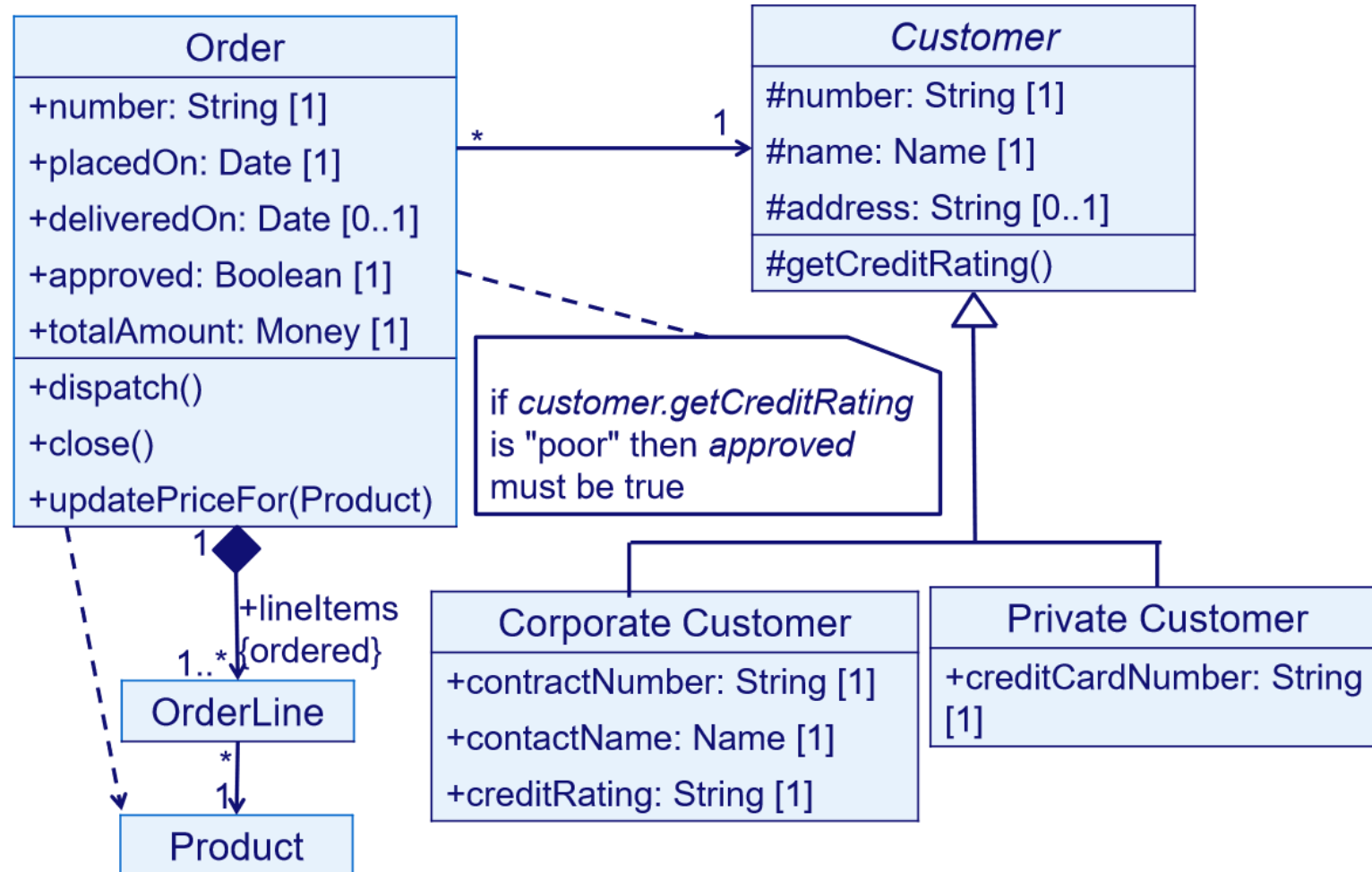


¿Cómo elegir una u otra forma?

- Atributo:
  - no tiene identidad Date, Number, String
  - Poco comportamiento y funcionalidad
- Asociación (ejemplo, una clase separada)
  - La identidad es importante: Cliente, Orden, Alumno, Libro
  - Comportamiento y funcionalidad complejos



# Poniendo todo en uno



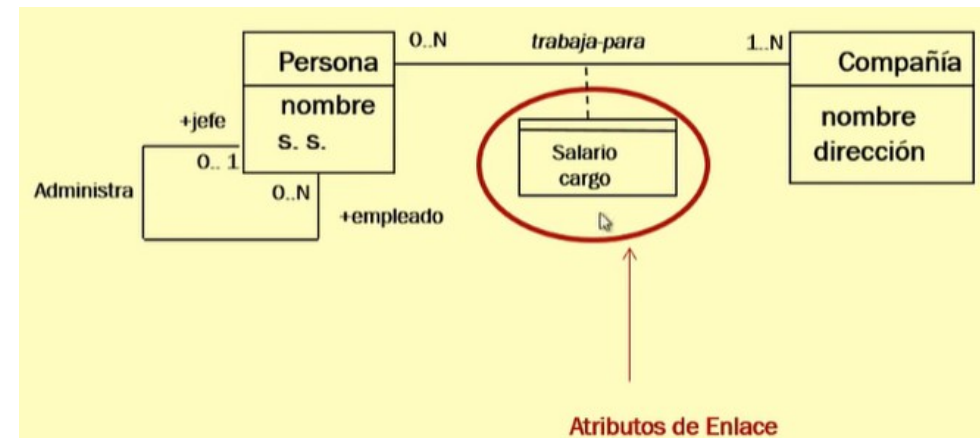
# Clases asociativas

Muchas veces una asociación necesita almacenar información relevante. Las clases asociativas se derivan de una asociación entre dos o más clases. **Una instancia de esta clase está asociada siempre a una y solo una instancia de cada participante de la asociación.** Se recomienda su uso cuándo una **asociación** (binaria, recursiva, N-aria, pero no agregación ni composición) **necesita almacenar atributos propios.**

Asociación con atributos propios.

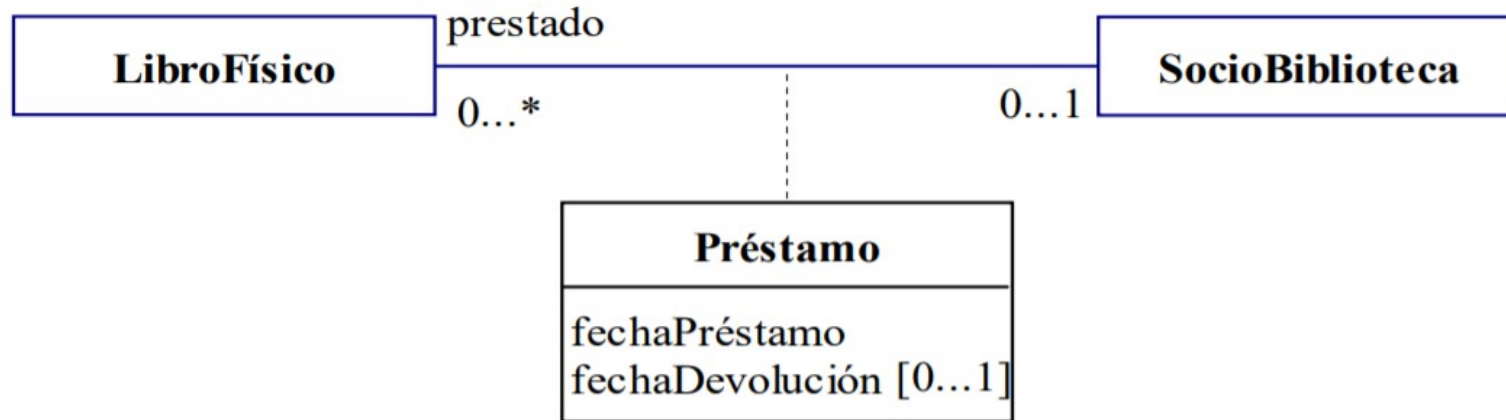


29



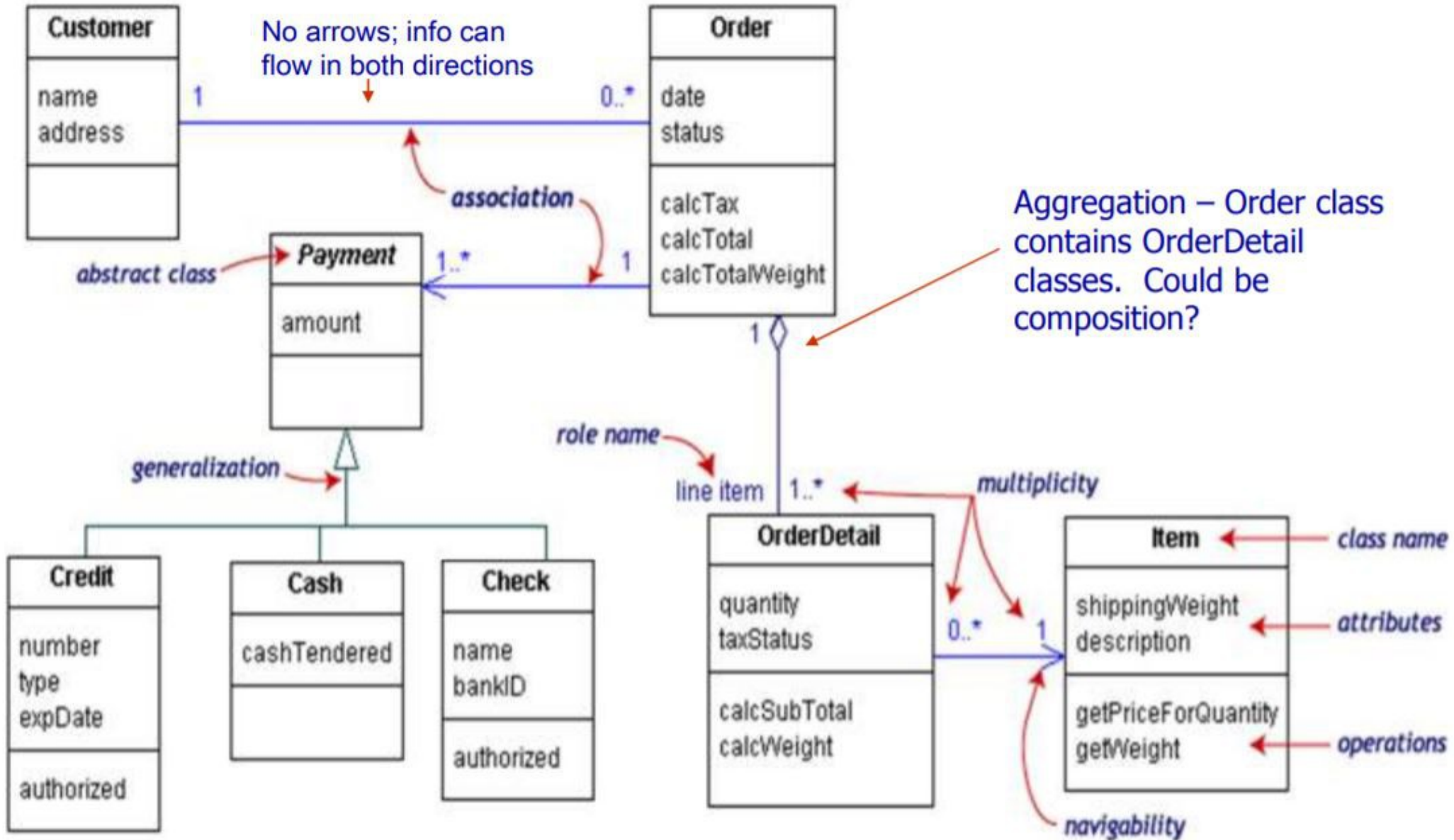
# Clases asociativas

Un préstamo es una asociación entre un único socio y un único libro.



# Estilo. Diagramas de clases

- Los **atributos no deben ser objetos** (utilizar relaciones en tal caso)
- En los diagramas de clases no suelen aparecer (son detalles de implementación y no de diseño):
  - Constructores
  - Métodos de acceso (“get/set”)
  - Métodos de gestión de elementos de una asociación o agregación (por ejemplo, “add/remove”)



# enumeraciones EN DIAGRAMAS DE CLASES





# Herramientas para la elaboración de diagramas uml

- **ArgoUML**
- **Visual Paradigm**
- **Draw io**
- **Dia**

# referencias

- Página de referencia de UML. <http://www.uml.org/>
- Enrique Hernández Orrallo “El lenguaje Unificado de Modelado (UML)”.  
<http://www.disca.upv.es/enheror/pdf/ActaUML.pdf>