Introducción al Lenguaje de Modelado UML

Prof. Roxana Giandini

LIFIA - Facultad de Informática - UNLP

Lenguaje Unificado de Modelado

Modelos en Ingeniería

- Todas las ingenierías han usado y usan modelos científicos (matemáticos, físicos, etc.) para alcanzar mayor confiabilidad con mucho éxito.

- Sin embargo, en las ingenierías “clásicas” la transmisión de las intenciones de diseño pueden tener errores (malos materiales, personas con diferentes talentos que pueden comprender mal, etc.).

Modelos en Software

- Usamos modelos con diferentes propósitos:
- Para entender
- Para comunicar
- Para documentar, bosquejar y planear

Modelos como “Bosquejos”

- Para analizar y predecir

Modelos como “Guías”

- Eventualmente para producir software

Modelos como “Programas”

Modelado

- Ayudan a la comprensión de sistemas complejos
- Indican QUE hará el sistema pero NO COMO lo hará
- Ayuda a la corrección de errores
- Ayuda a la evolución y reuso
- Esencial para la comunicación entre miembros de un equipo

Lenguaje

- En la unificación se adopta un **Lenguaje Gráfico estándar**

- Como toda definición de Lenguaje, posee:

  - **Sintaxis:**

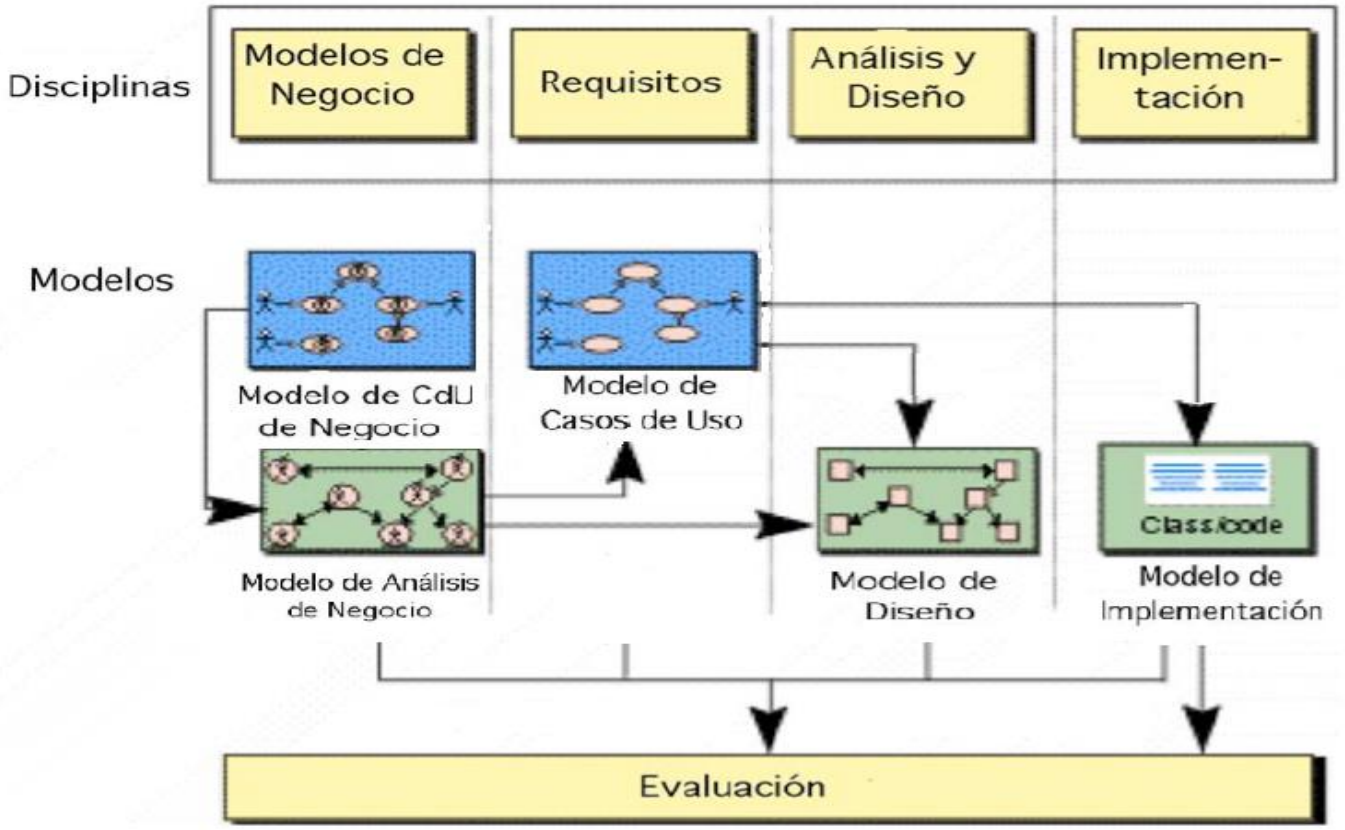
    determina las reglas de construcción de los diagramas de modelado.
    Provee base formal para los diagramas (metamodelo, OCL).

  - **Semántica:**

    otorga la descripción detallada del significado conceptual de cada elemento de modelado.

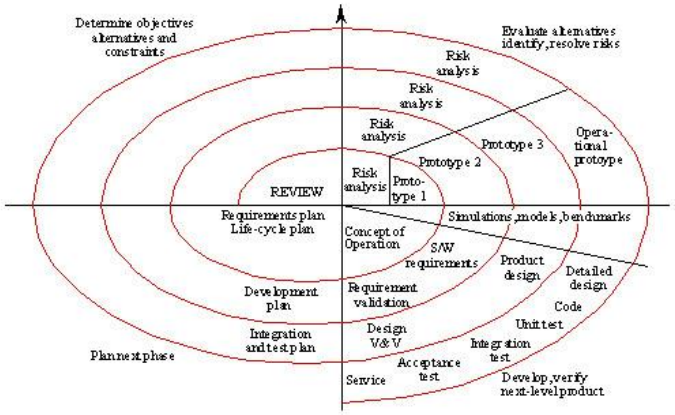
Lenguaje Unificado de Modelado

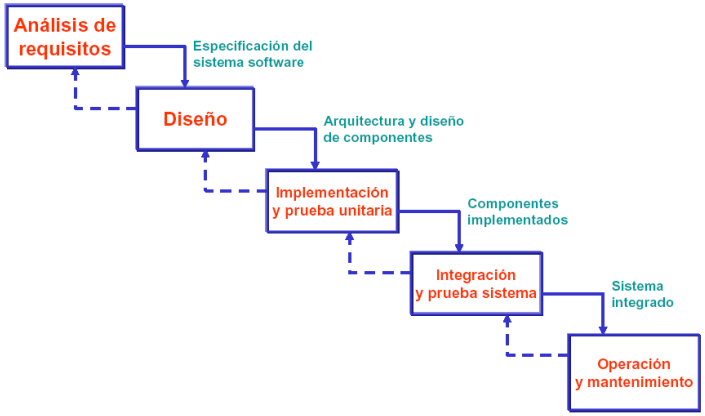
- Útil en las diferentes etapas del ciclo de vida del desarrollo de sistemas.

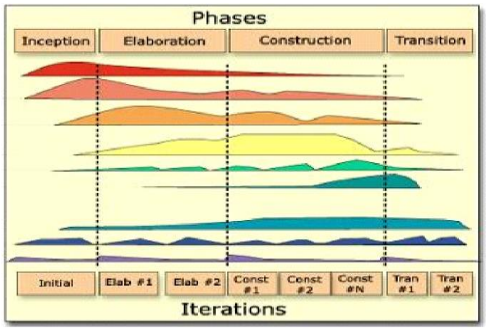


Lenguaje Unificado de Modelado

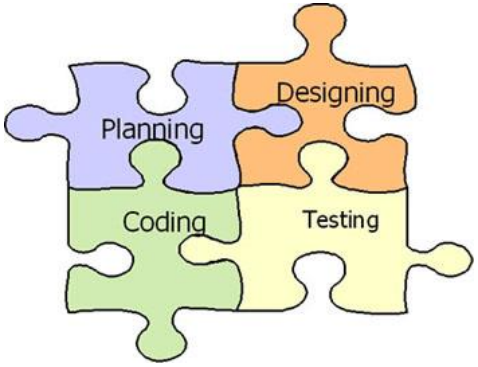
- Independiente del proceso de desarrollo de software.







MSF



Lenguaje Unificado de Modelado

- Independiente del lenguaje de implementación.

C++

python™

Microsoft .net

Ada

Java

Utilidades de UML

UML permite:

- Definir los límites del sistema y sus principales funciones, mediante **Casos de Uso y actores.**
- Ilustrar el funcionamiento de un caso de uso mediante **Diagramas de Interacción**
- Representar la estructura de un sistema mediante **Diagramas de Clases.**
- Modelar el comportamiento de los objetos mediante **Diagramas de Máquinas de Estados.**

Utilidades de UML

UML permite:

- Describir la arquitectura de la implementación física con **Diagramas de Componentes y Despliegue.**
- Extender la funcionalidad de elementos estándar mediante **estereotipos.**
- Proveer **base formal** para los diagramas (metamodelo, OCL).

Diagramas de UML 2.0

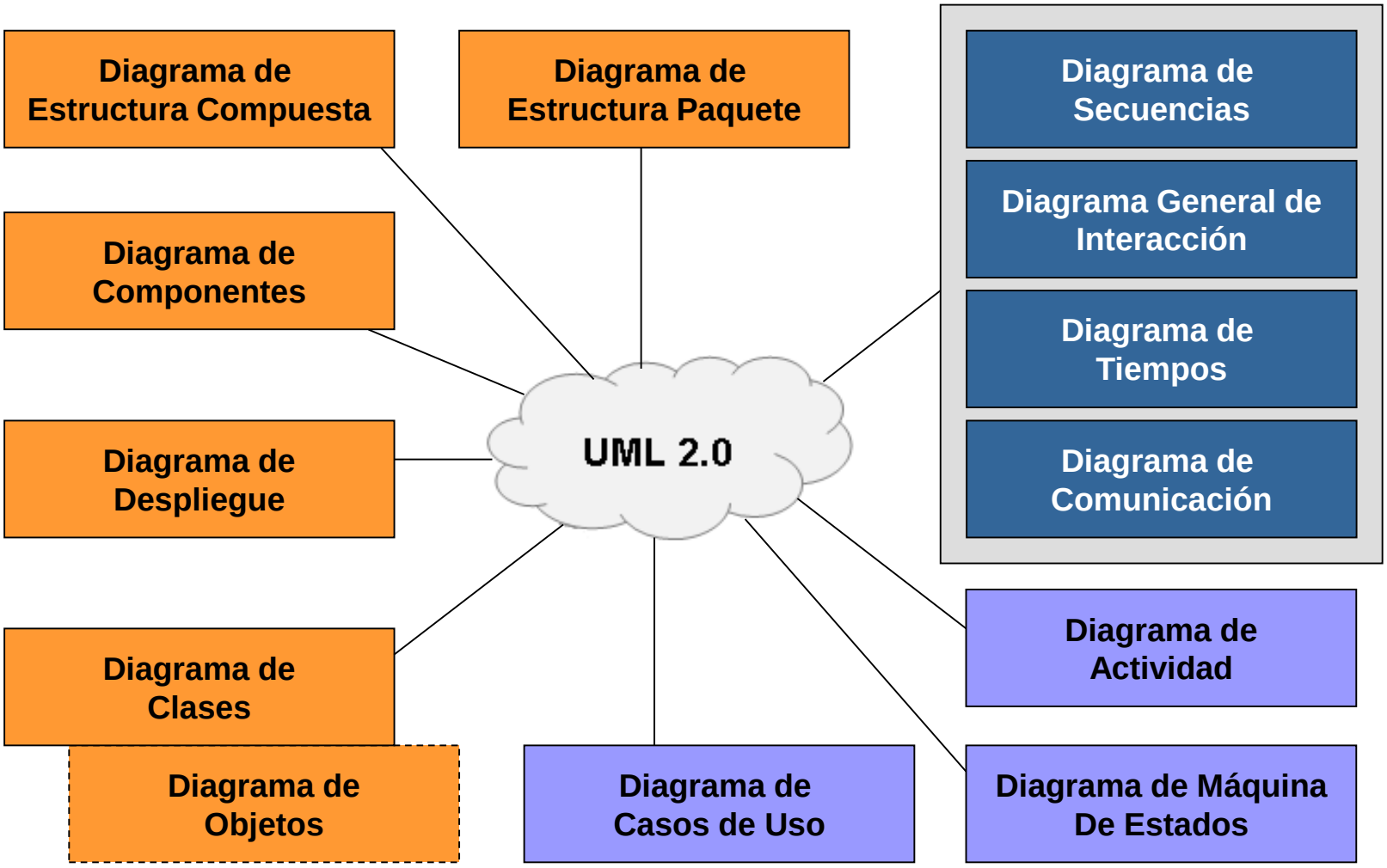


Diagrama de Clases

Diagrama de Clases

Definición:

Un diagrama de clases muestra las clases del sistema y sus relaciones. Describe la vista estática de un sistema.

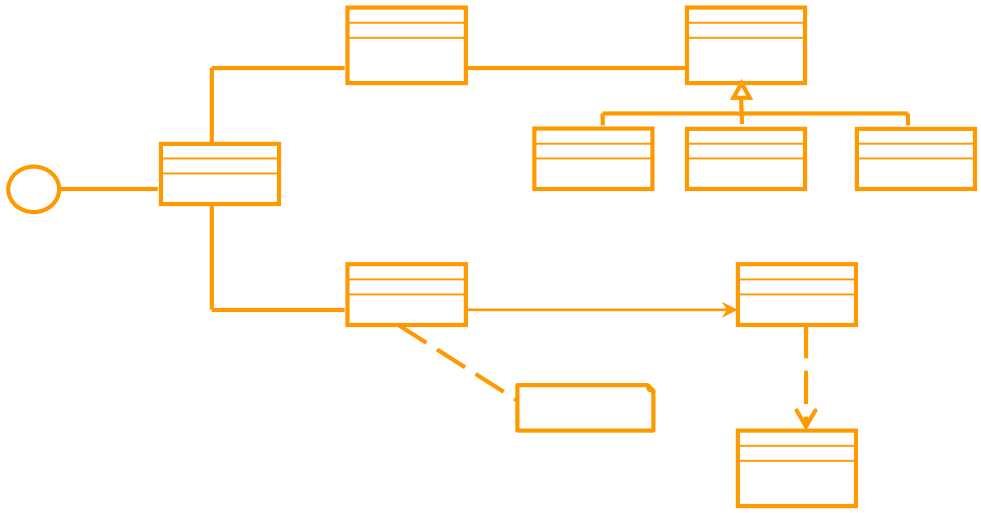
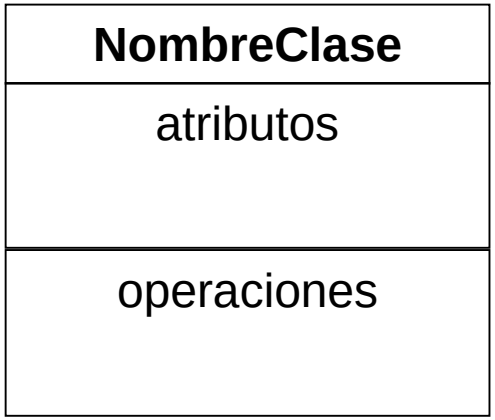


Diagrama de Clases: clase

- **¿Qué es una clase?**

  Una clase es una descripción de un conjunto de *objetos* que comparten los mismos *atributos* (estado interno de los objetos de la clase), *operaciones* (mensajes que responden los objetos), relaciones y semántica.

- **Estructura:**



Clase: atributo

Sintaxis

visibilidad **nombre** *[multiplicidad]:***tipo** *=valor inicial {propiedades}*

- Algunas partes son opcionales.
- Vamos a especificar nombre, tipo y visibilidad para los atributos
- **Visibilidad:** sirve para limitar la visibilidad del atributo a los objetos externos a la clase a la que pertenece.
  - Pública (+) : el atributo puede ser visto y usado fuera de la clase.
  - Privada (-) : el atributo no puede ser accedido por otras clases.
  - Protegida (#): es privado pero visible a las subclases de la clase donde está declarado.

NOTA: Si bien UML lo permite, ***no diseñaremos*** *clases con* ***atributos públicos*** *ya que esto contradice al concepto de encapsulamiento de los objetos.*

Clase: atributo

Sintaxis

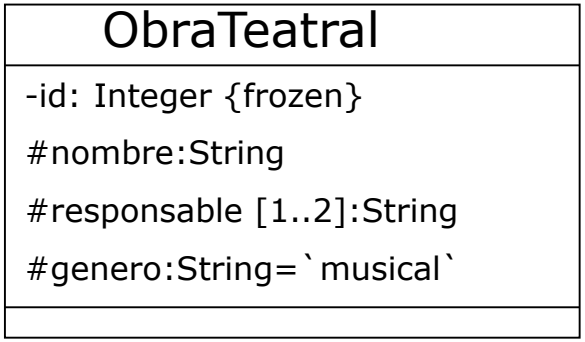
Propiedades:

frozen

readOnly

.....

Ejemplo:



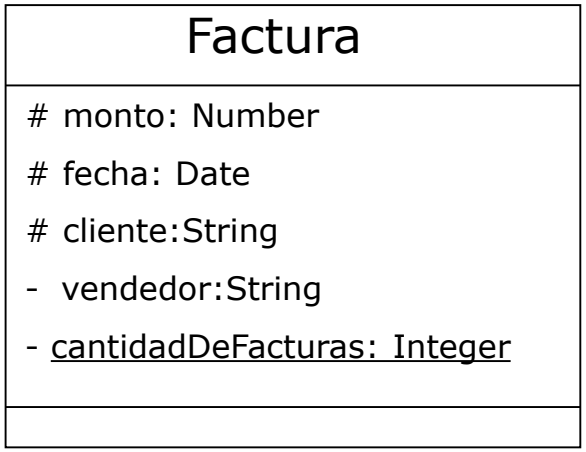
Clase: atributo

Sintaxis

Atributo de Clase:

como variable de clase. Va subrayado y significa que el valor de este atributo es el mismo para todos los objetos de la clase, ya que lo comparten.

Por ejemplo, el atributo cantidadDeFacturas es usado para contabilizar internamente las facturas en un comercio.



Clase: operación

- **Sintaxis**

visibilidad nombre([*lista de parámetros*]) :tipo de retorno {*propiedades*}

  - Algunas partes son opcionales. Vamos a especificar nombre, tipo y visibilidad para las operaciones.
  - Los parámetros se separan por comas.
  - Si la operación no tiene parámetros, igualmente van los () al final del nombre.
  - El tipo de retorno sólo va cuando la operación devuelve un valor. Si no retorna nada, no se especifica nada.

- Declaración de un parámetro:

  nombre :tipo [*multiplicidad*] =valor

Clase: operación

- Si el retorno de la operación es un objeto compuesto (Collection). No poner "Collection“, sino el tipo/Clase del elemento que la compone, con multiplicidad. Ejemplo:

+obtenerOfertasDelDia(): Oferta[*]

- Las operaciones abstractas van en cursiva (si se utiliza una herramienta de modelado) o con la notación «abstract».

+*calcularSueldo*(): Number

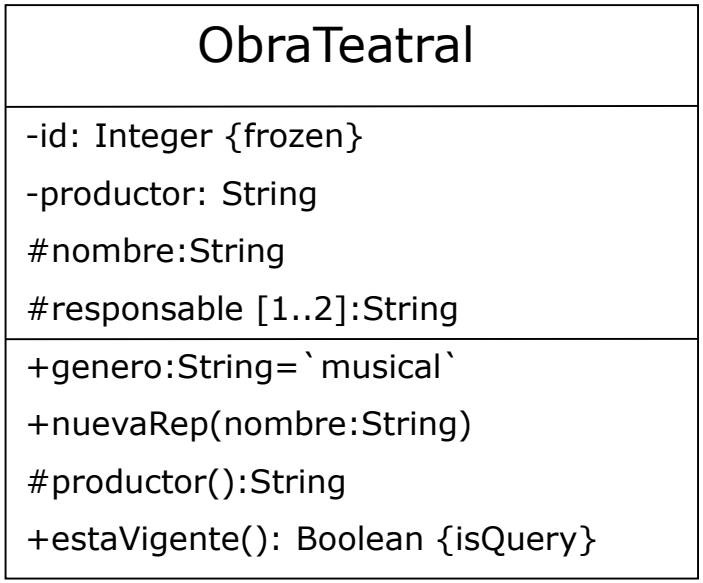
«abstract» +calcularSueldo(): Number

Clase: operación

Propiedades

- isQuery
- --

Ejemplo



Clase: operación

Operaciones de Clase

- Mensajes de Clase, generalmente de creación también llamados constructores
- Van subrayadas, como los atributos de clase

Ejemplo

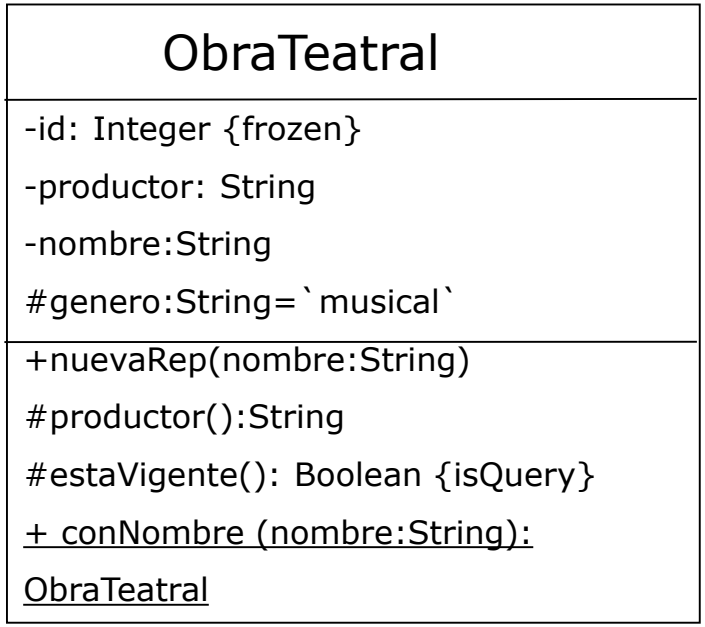


Diagrama de clases: relaciones

¿Qué son las relaciones entre clases?

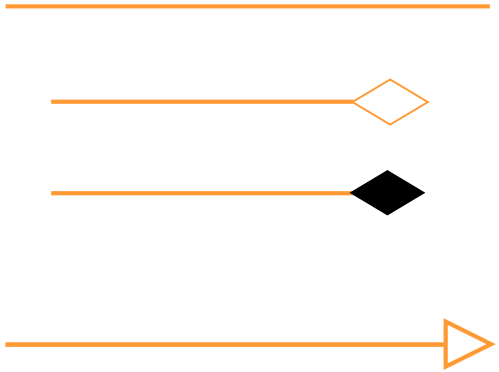
Una relación es una conexión semántica entre objetos. Proveen un camino de comunicación entre ellos.

¿Qué tipos de relaciones usaremos?

- Asociación
- Generalización

Diagrama de clases: relaciones

- **Notación:**
  - Asociación
    - Agregación
    - Composición
  - Generalización



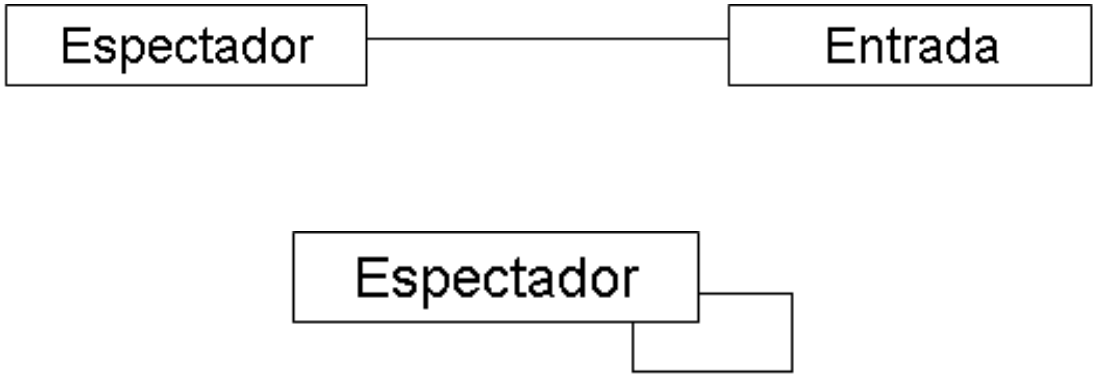
Relaciones: Asociación

¿Qué es una Asociación?

Es una relación estructural que especifica que los objetos de un elemento están conectados con los objetos de otro.

Si no ponemos multiplicidad en los extremos, asume 1.

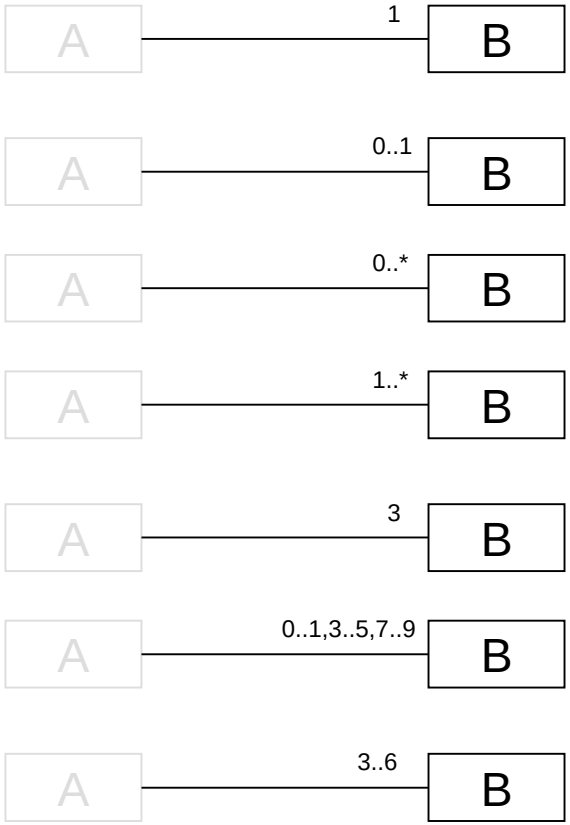
Ejemplo



Asociación: propiedades (cont.)

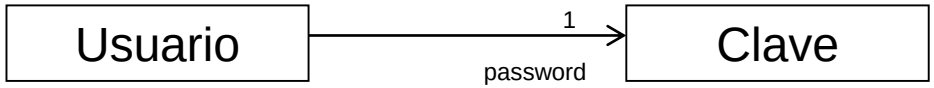
Multiplicidad: indica “cuántos” objetos pueden conectarse a través de una instancia de una asociación.

- **Se puede indicar una multiplicidad de:**
  - **Exactamente Uno:**
  - **Cero o Uno:**
  - **Muchos:**
  - **Uno o Más:**
  - **Número Exacto:**
  - **Lista:**
  - **Rango:**

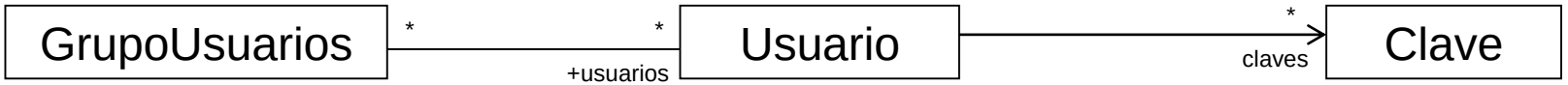


Asociación: propiedades

- **Rol:** son las caras que presentan las clases a las demás.
- **Navegabilidad**: sirve para limitar la navegación a una sola dirección.

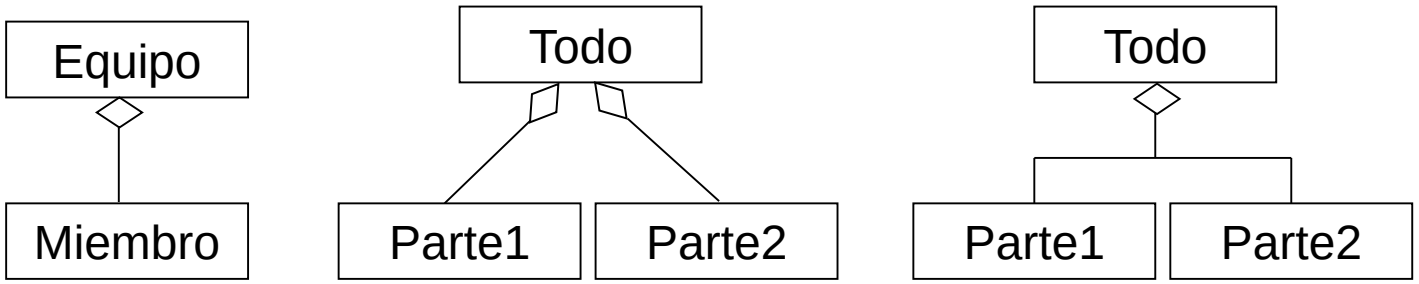


- **Visibilidad:** limita el alcance de los objetos que se relacionan. Similar al alcance de atributos.
  - Pública (+)
  - Protegida (#)
  - Privada (-)

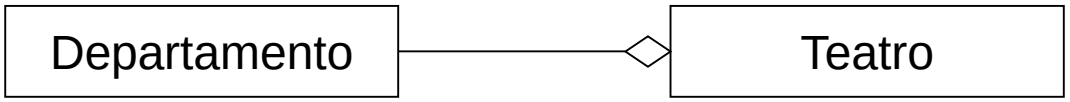


Asociación: propiedades (cont.)

Agregación: es una asociación especial, una relación del tipo "todo/parte" dentro de la cual una o más clases son partes de un todo.



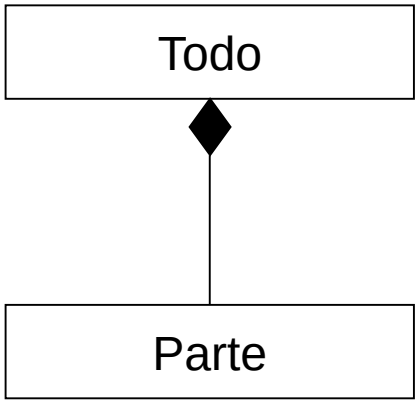
Ejemplo:



Asociación: propiedades (cont.)

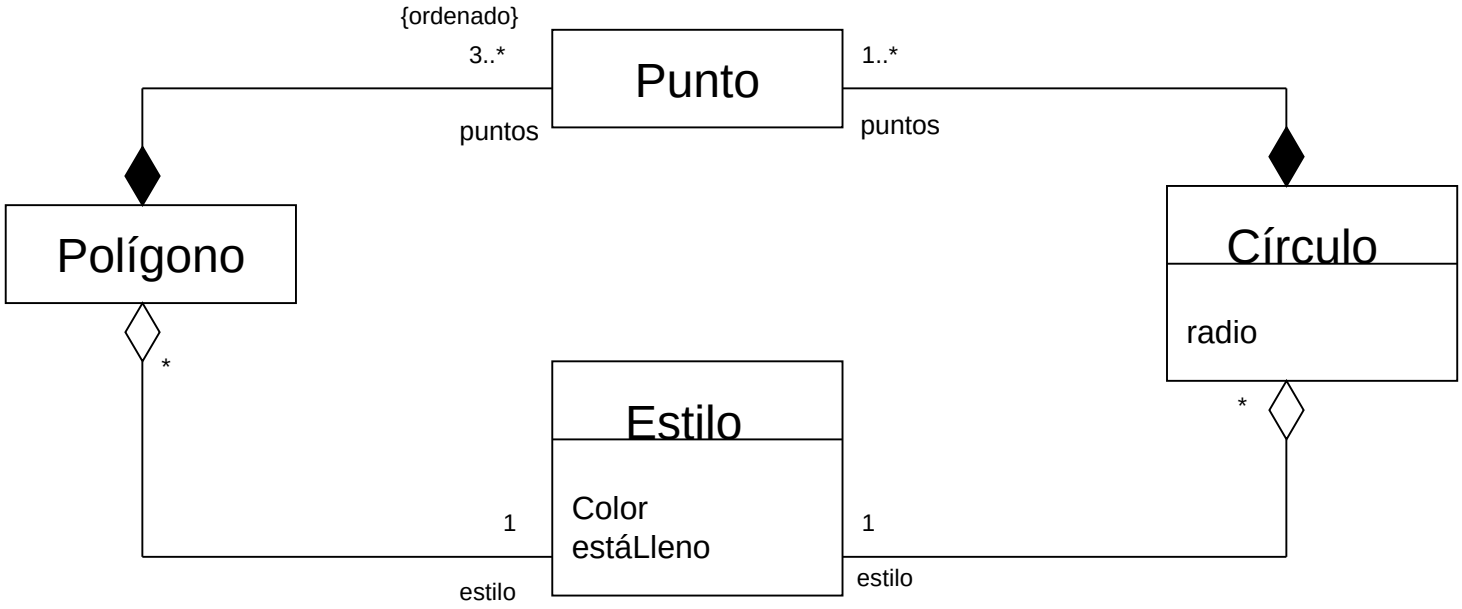
Composición: es una forma de agregación, con fuerte sentido de posesión y tiempo de vida coincidentes de las partes con el conjunto.

- Una parte puede pertenecer solamente a una composición (un *todo*).
- Cuando el todo desaparece, también lo hacen sus partes.



Asociación: propiedades (cont.)

Ejemplos de Agregación y Composición

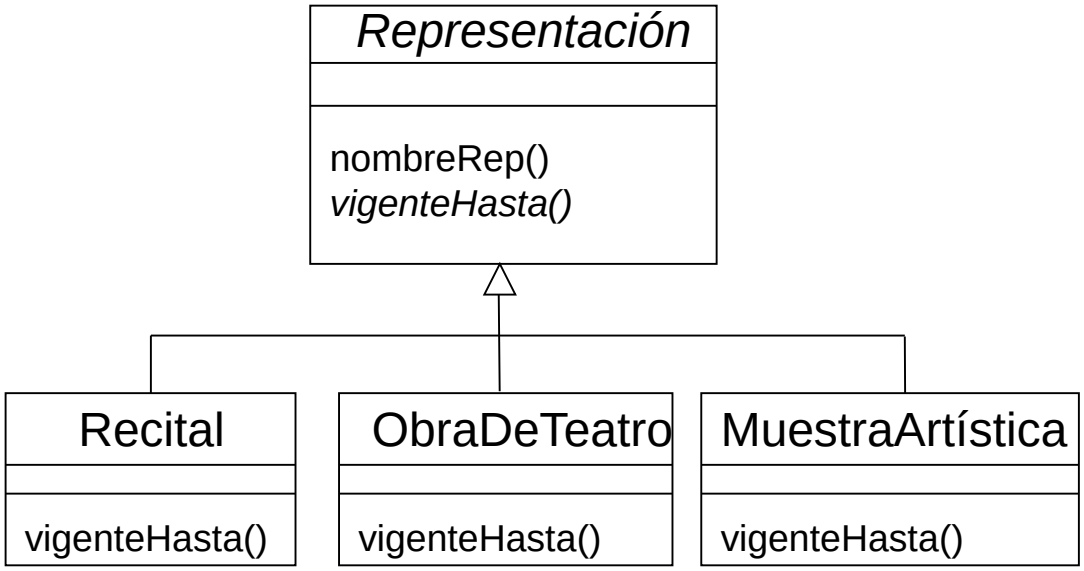


Relaciones: Generalización

¿Qué es una Generalización?

Es una relación de herencia entre un elemento general (llamado superclase) y un caso más específico de ese elemento (llamado subclase).

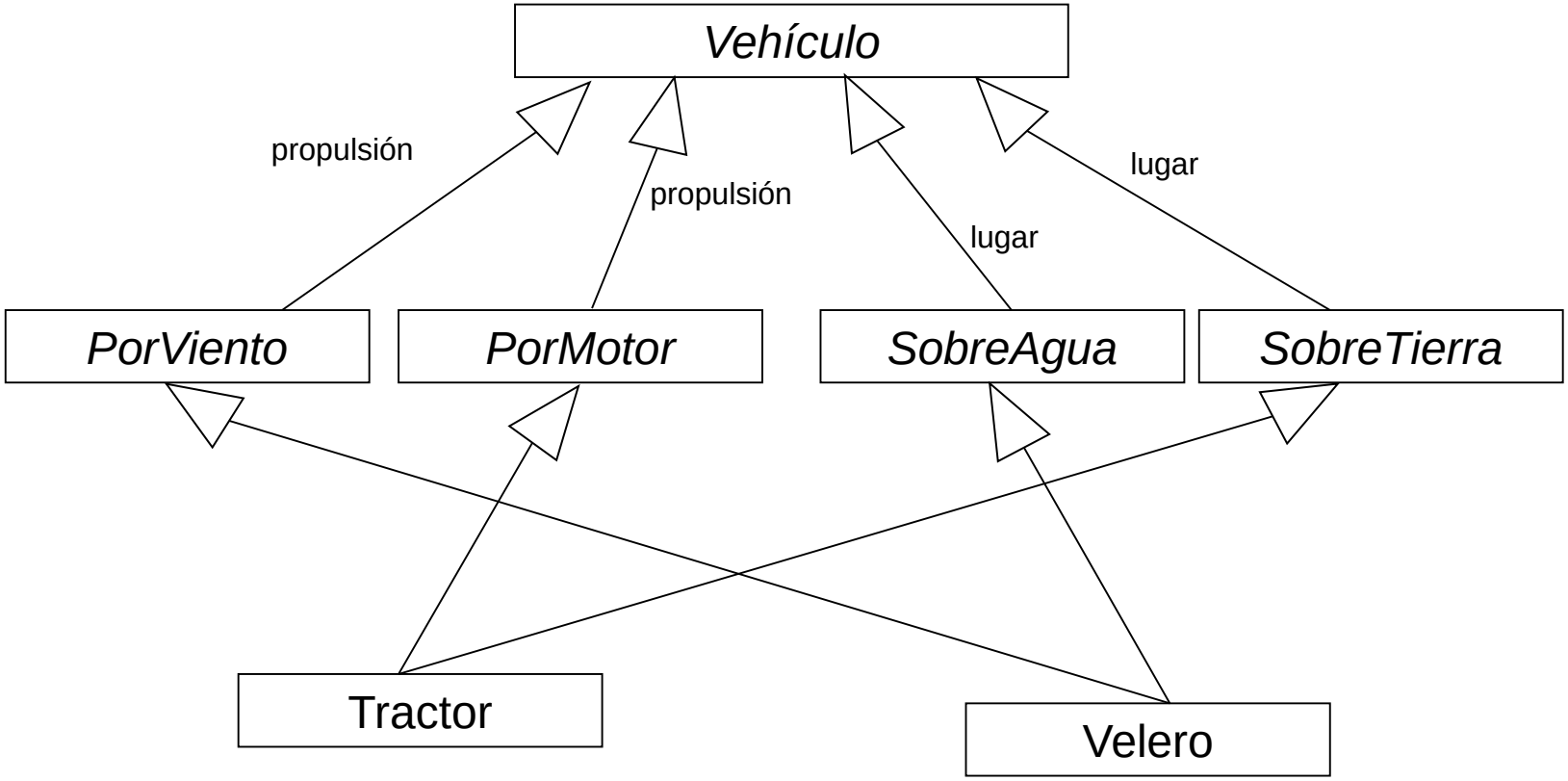
Ejemplo



- En UML puede representarse herencia múltiple
- La clase abstracta (que no puede instanciarse por tener operaciones abstractas) va en cursiva o bien con <<abstract>> acompañando al nombre

Generalización

- **Ejemplo con discriminador**



Elementos de anotación

- **¿Qué es una nota?**

  Es un símbolo para mostrar restricciones y comentarios junto a un elemento o una colección de elementos.

- **Ejemplo**

Con autoasociación

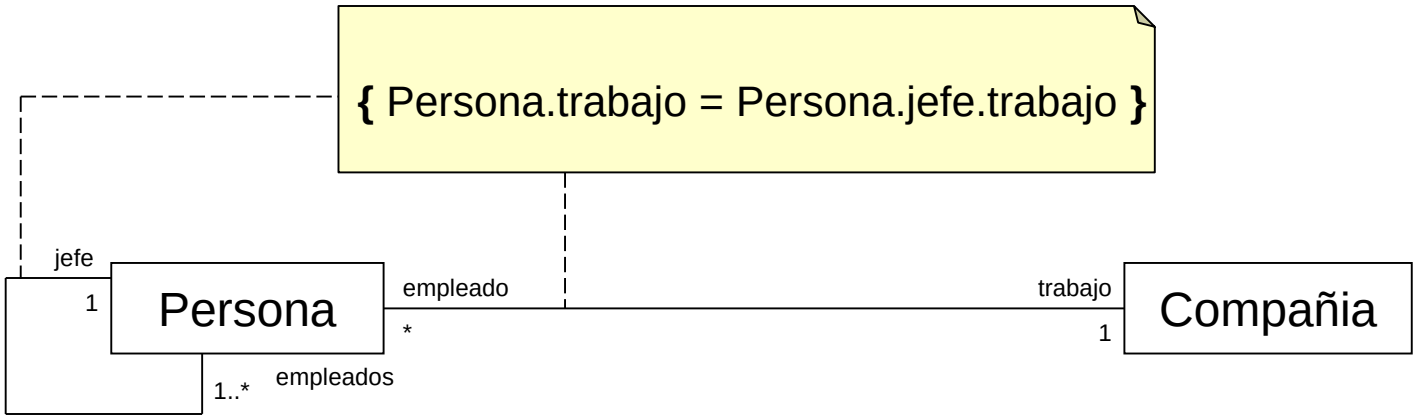


Diagrama de clases – Ejemplo

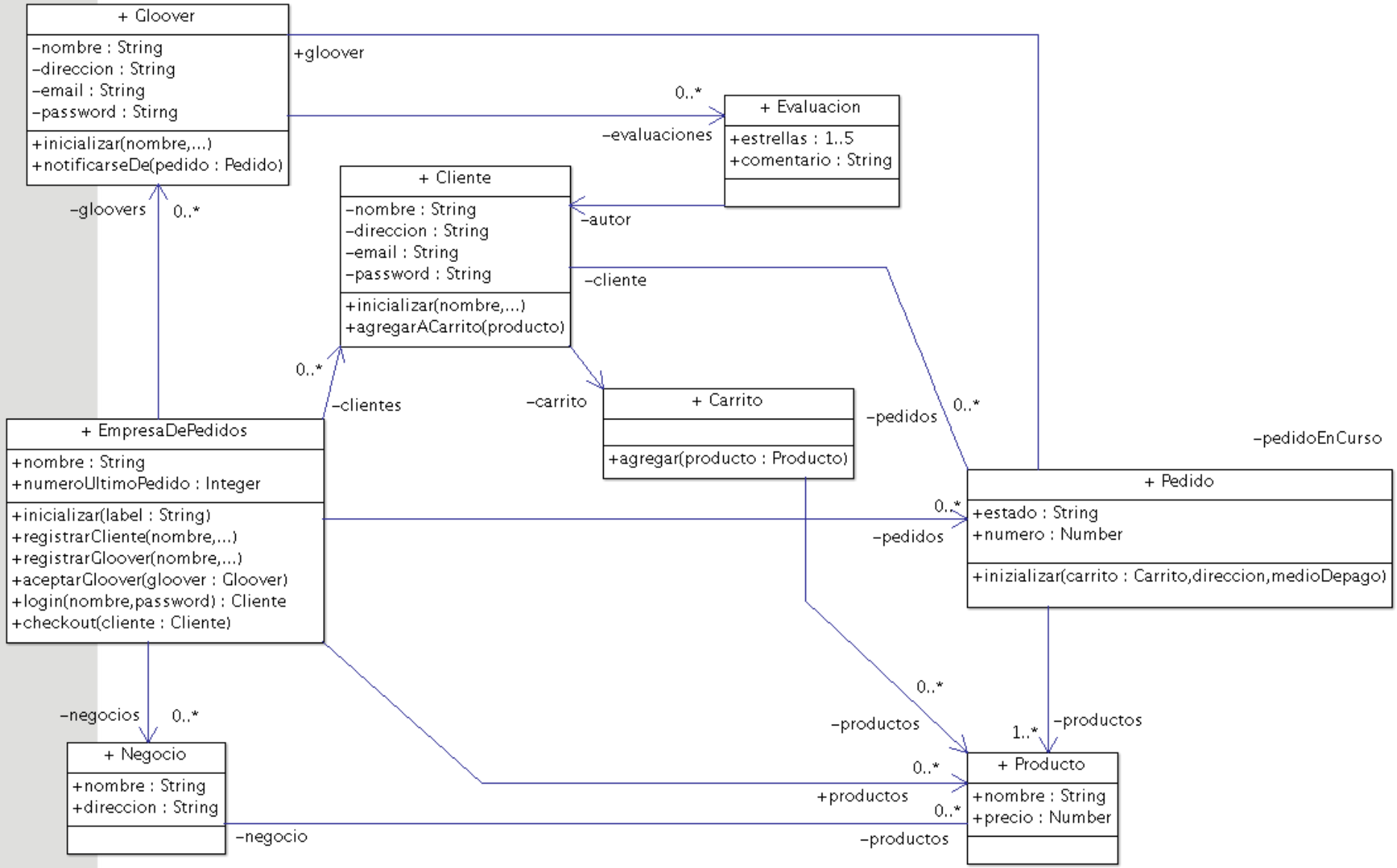


Diagrama de clases – Interfaces

Una clase Interfaz especifica una parte, visible externamente, del comportamiento de otras clases.

En UML se especifica con el estereotipo **<<Interface>>**.

Es abstracta ya que otra/s clase/s implementa sus métodos.

La clase que implementa la interfaz, implementa todos los métodos de la interfaz.

La relación con la clase que la implementa, se representa con línea punteada hacia la clase Interfaz.

Diagrama de clases – Interfaces