3. MODELADO ESTRUCTURAL

Diseño de Software

Grado en Ingeniería Informática

Curso 2024/2025

José Varela Pet - Departamento de Electrónica y Computación

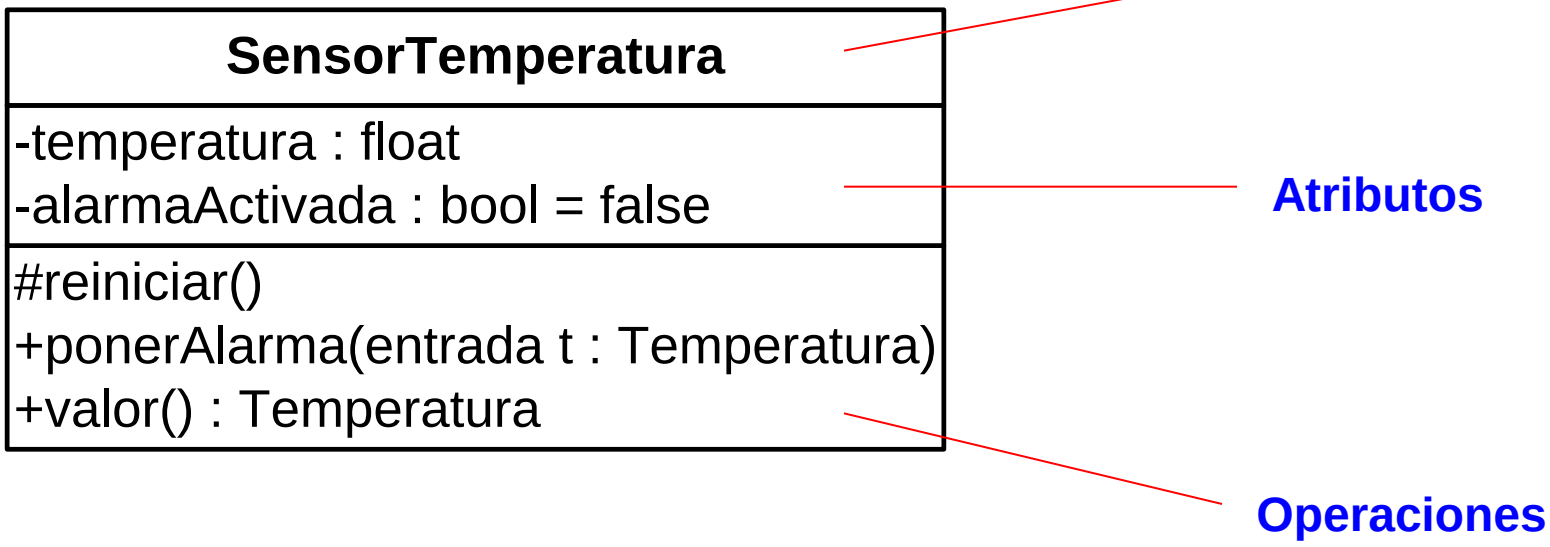
Bibliografía

- **Fowler**. *UML distilled*. 3ª ed. Addison-Wesley, 2004
  - Capítulo 3 (pp. 35–52)
- **Booch, Rumbaugh, Jacobson**. *El Lenguaje Unificado de Modelado*. 2ª ed. Addison-Wesley, 2006
  - Capítulos 4 y 5 (pp. 51–80)
  - Capítulo 8 (pp. 113–124)

CLASES

- Bloques de construcción más importantes del paradigma OO
- **Clase**: descripción de un conjunto de **objetos** que comparten atributos, operaciones, relaciones y semántica
- Son **abstracciones** independientes del lenguaje de programación capaces de implementar una o más **interfaces**

Representación gráfica



Especificación de una clase

- **Nombre**: único dentro del paquete
- **Atributos**: propiedades compartidas por todos los representantes de la clase
  - Determinan *estado* ⇨ expresión nominal
  - Sintaxis: `visibilidad nombre : tipo = valor defecto`
- **Operaciones**: servicios que pueden ser requeridos a cualquier instancia
  - Definen *comportamiento* ⇨ expresión verbal
  - Sintaxis: `visibilidad nombre (parámetros) : retorno`

Responsabilidades y colaboraciones

- **Responsabilidad**: *contrato* para una clase
- Los objetos de una clase tienen los mismos *tipos* de **estado** y de **comportamiento**
- A nivel abstracto, atributos y operaciones son un medio para asumir responsabilidades
- Una clase *bien estructurada* tiene unas pocas responsabilidades
- La mayoría de las clases **colaboran** con otras para ejercer sus responsabilidades

Tarjetas CRC

- **CRC** = Clase-Responsabilidad-Colaborador
- Descritas por Cunningham y Beck en los 80
- No forman parte de UML
- Cada tarjeta consta de:
  - **Nombre** de la clase
  - **Responsabilidades**: frases cortas que resumen lo que deberían hacer sus instancias
  - **Colaboradores**: clases en las que se apoyan para asumir dichas responsabilidades

Tarjetas CRC

Pedido	
Responsabilidades	**Colaboradores**
Comprobar artículos en stock	LineaPedido
Determinar precio	Cliente
Comprobar validez de pago	
Enviar a dirección de reparto	

Tarjetas CRC

- **Ventajas**:
  - Fomentan discusión entre desarrolladores
  - Permiten concebir interacciones entre objetos sin necesidad de diagramas
  - Previenen incorporar clases limitadas a simples *contenedores de datos*
- Gran **riesgo**: generación de largas listas de responsabilidades de bajo nivel

Modelado de clases

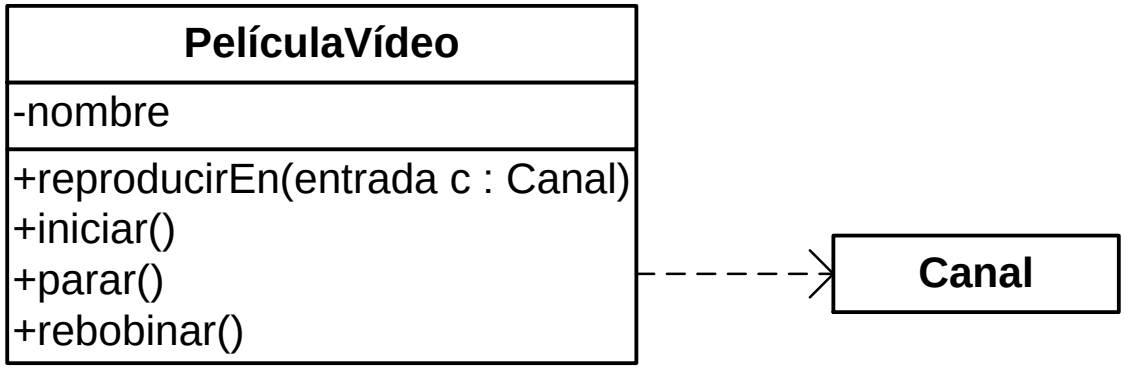
- ☑ Identificar y definir las abstracciones que se utilizan para describir el *dominio*
- ☑ Repartir *adecuadamente* las responsabilidades entre las clases
- ☑ Proporcionar a cada clase los atributos, operaciones y relaciones necesarios para asumir sus responsabilidades
- ☑ Para cada clase, distinguir la interfaz de la implementación

RELACIONES

- La mayoría de las clases no están aisladas, sino que *colaboran*
- Tres tipos de **relaciones**:
  - **Dependencia**: relación de uso
  - **Generalización**: relación padre/hijo (“es un tipo de”)
  - **Asociación**: relación estructural
- Importancia de equilibrar el modelado de relaciones

Dependencia

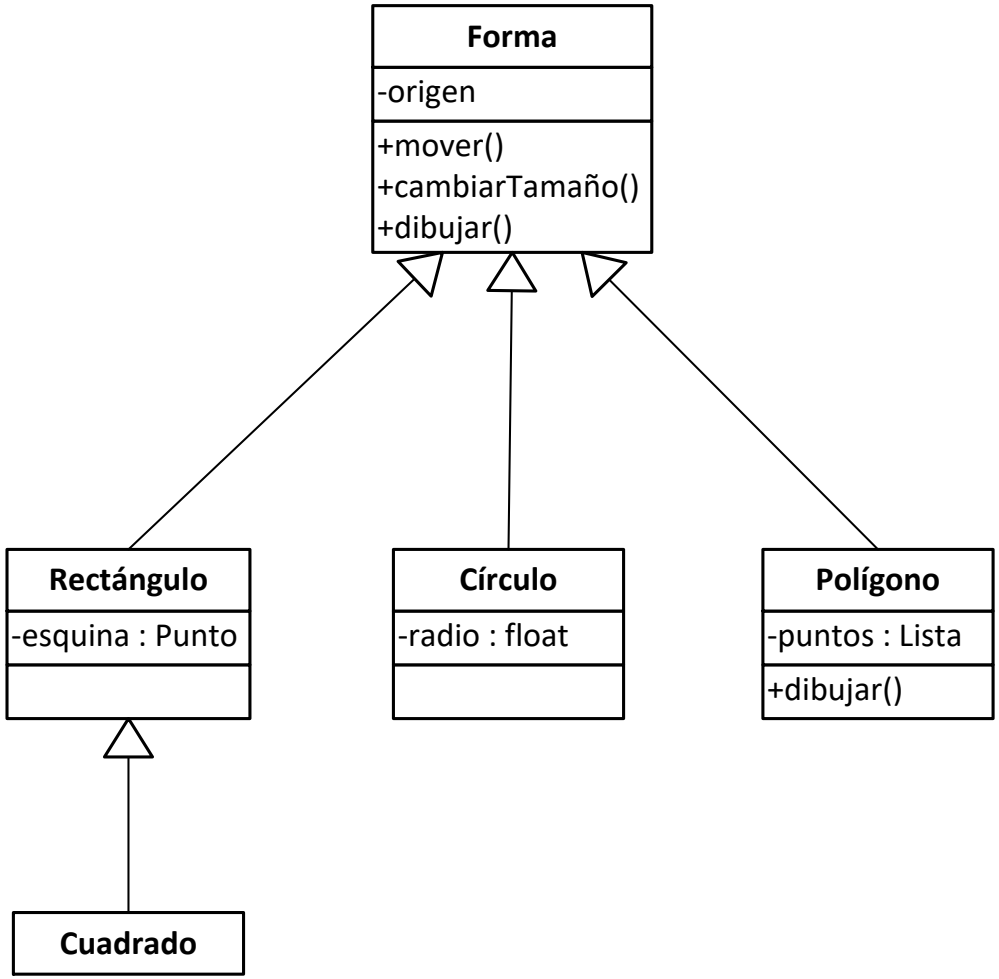
- Declara que los cambios en la especificación de un elemento pueden afectar a la de otro que lo usa
- Se emplea para indicar **utilización**



Generalización

- Relación entre un elemento general (*padre*) y otro específico (*hijo*)
- Rige el principio de **sustitución**
- Las subclases heredan propiedades de las superclases; pueden añadir otras y redefinir operaciones (*polimorfismo*)
- En UML se admite *herencia múltiple*
- **Clase raíz**: clase sin padres
- **Clase hoja**: clase sin hijos

Ejemplo de generalización



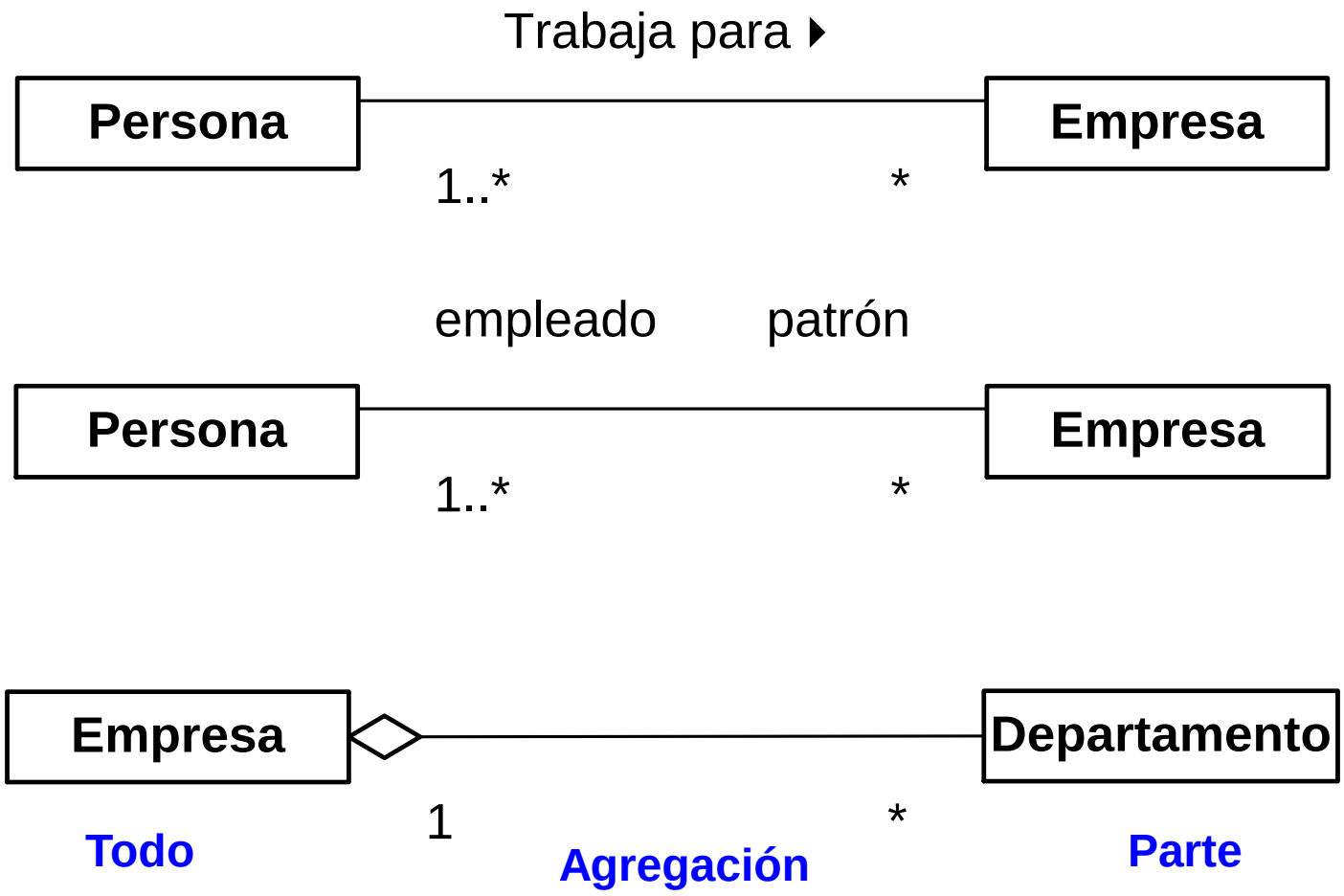
Modelado de herencia simple

- ☑ Identificar responsabilidades, atributos y operaciones comunes a varias clases
- ☑ Elevar estas propiedades a una clase más general (crearla si no existe)
- ☑ Construir jerarquía especificando las generalizaciones necesarias

Asociación

- Relación *estructural* entre objetos
- Vincula clases del mismo nivel (frente a dependencia y generalización)
- Adornos:
  - **Nombre**: etiqueta que describe la relación
  - **Rol**: papel que juega cada clase relacionada
  - **Multiplicidad**: nº de instancias participantes
  - **Agregación**: asociación todo-partes

Ejemplos de asociación



Modelado de asociaciones

- ☑ Establecer asociación entre clases si:
  - Se necesita **navegar** entre sus objetos
  - Sus instancias deben **interactuar**
- ☑ Especificar la **multiplicidad** en los extremos del arco
- ☑ Adornar como **agregación** si una de las clases se percibe como un todo frente a la otra

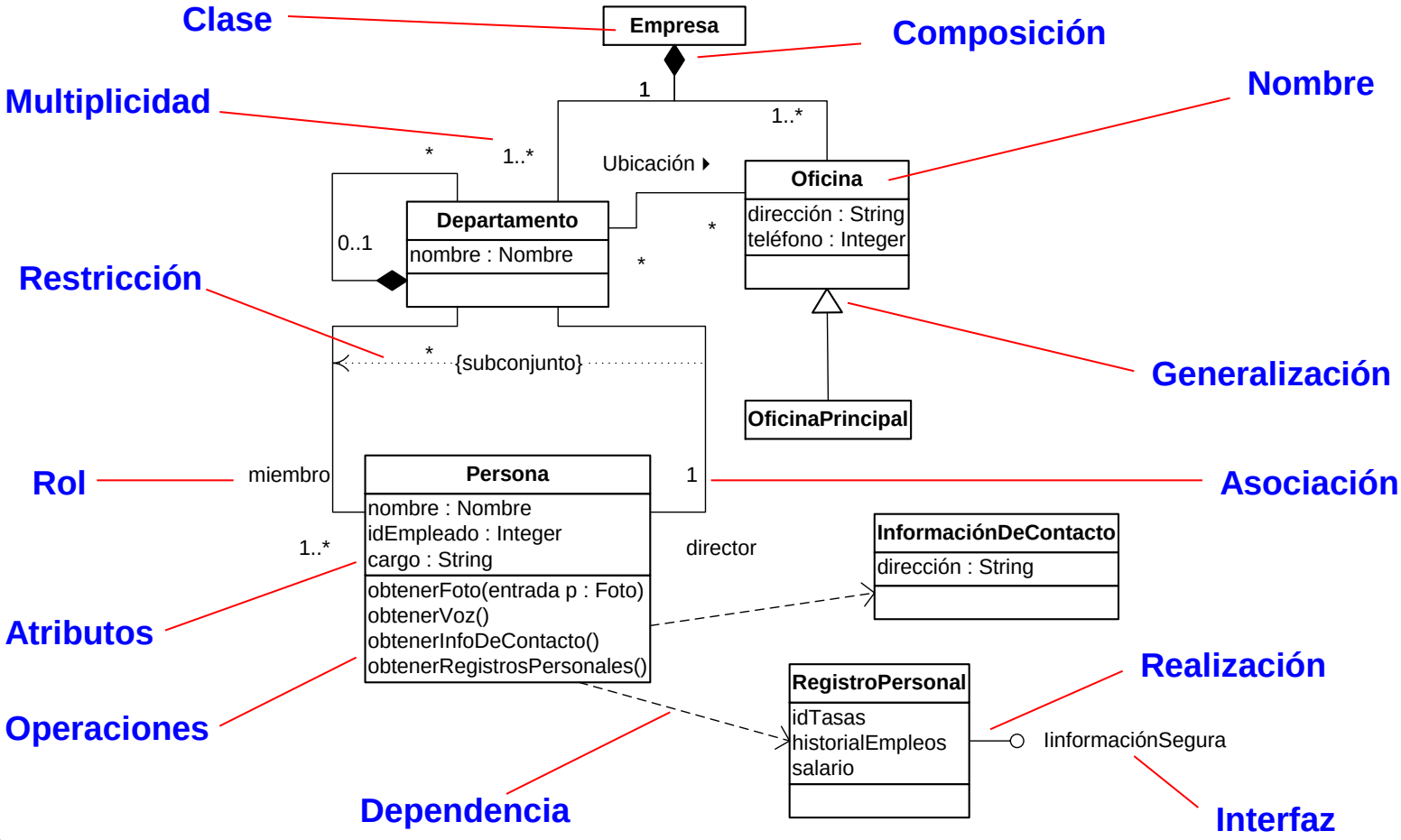
Recomendaciones de uso

- **Dependencia**: relación más débil de las tres
- **Generalización**: relación “es-un-tipo-de”
  - Evitar ciclos
  - Equilibrar jerarquías
- **Asociación**: relación estructural
- Sugerencias para trazar relaciones:
  - Usar consistentemente líneas rectas y oblicuas
  - Evitar cruces de líneas
  - Mostrar sólo las relaciones necesarias

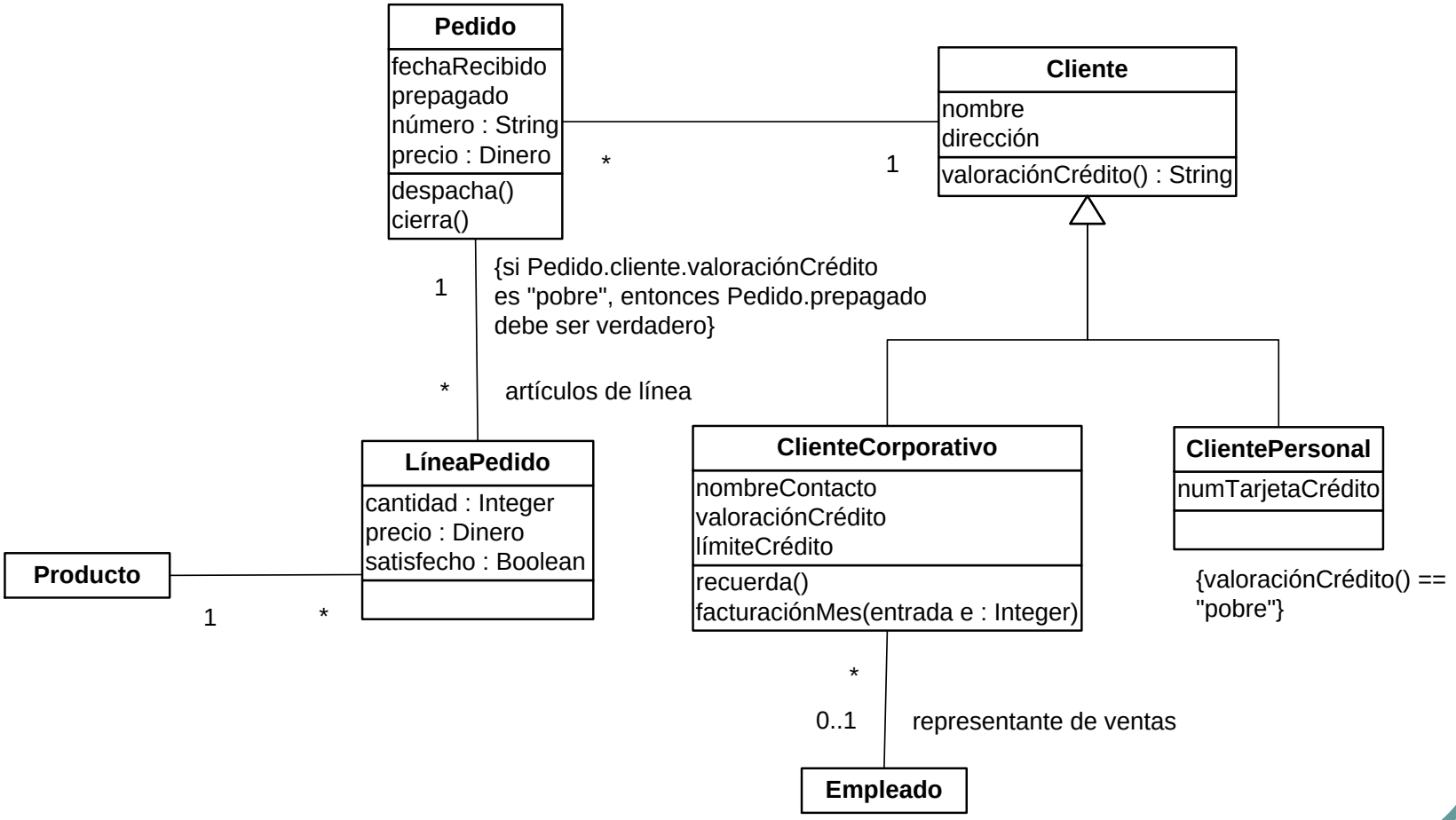
DIAGRAMAS DE CLASES

- Columna vertebral del modelado OO:
  - **Clases**, **interfaces** y **colaboraciones**
  - **Relaciones**: dependencia, generalización, asociación y realización
- Modelan la **vista de diseño**
- Base para diagramas de **paquetes**, **componentes** y **despliegue**
- Facilitan la construcción de sistemas aplicando **ingeniería directa** e **inversa**

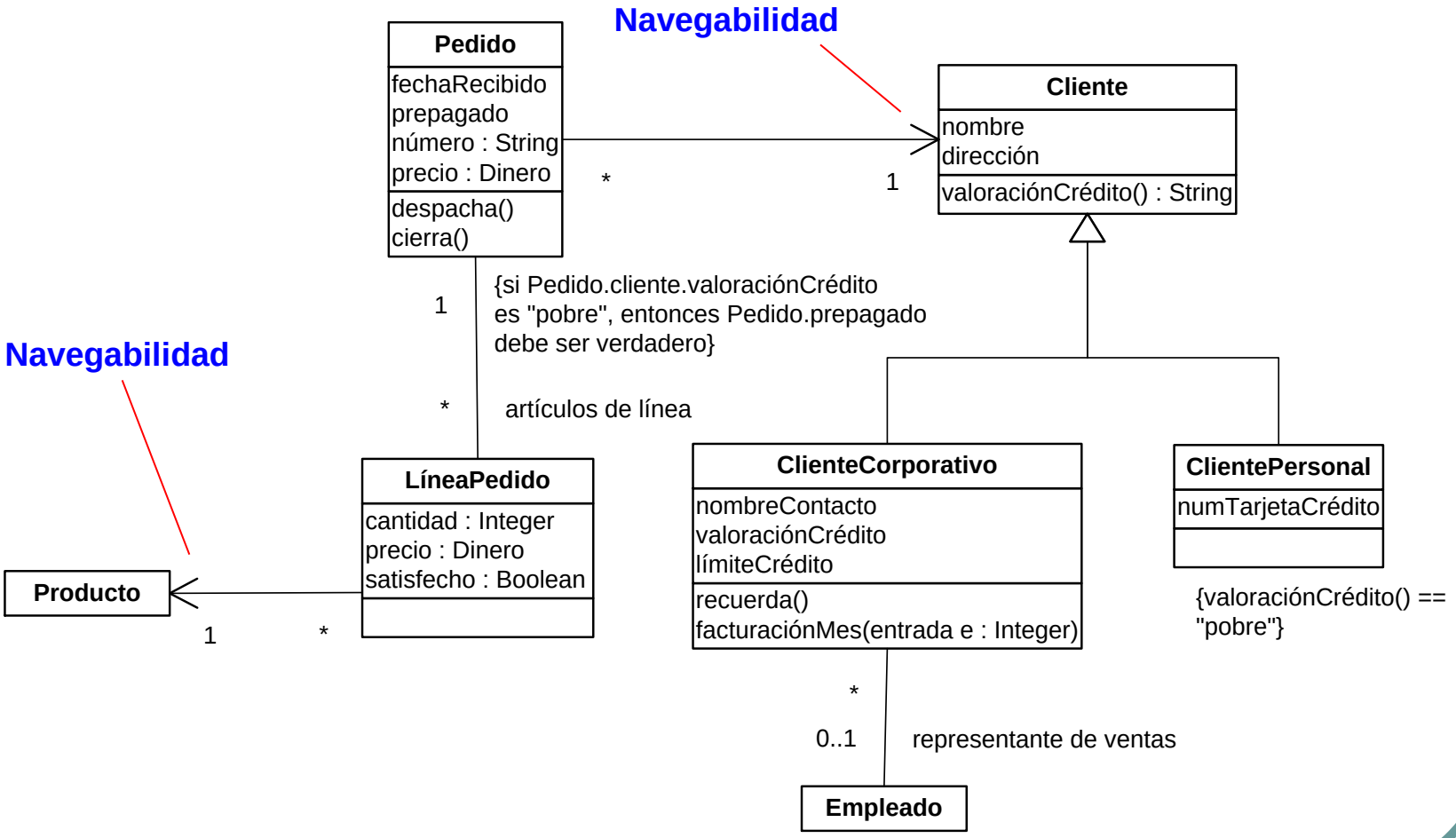
Ejemplo de diagrama de clases



Otro ejemplo



Navegabilidad



Modelado con diagramas de clases

1. **Vocabulario** de un dominio
   - ¿Cuáles son los conceptos relevantes del dominio y cómo se relacionan entre sí?
2. Estructura de **colaboraciones**
   - **Colaboración**: sociedad de elementos que proporcionan un comportamiento cooperativo mayor que la suma de ellos
3. Esquema conceptual de una **BD**
   - Diseño de BD relacionales, OO o híbridas

Ingeniería directa e inversa

- Los modelos se transforman en **código**
- UML no especifica **correspondencia** con ningún **lenguaje de programación**
- **Conexión** con lenguajes OO: Java, C++, C#, Smalltalk, Eiffel, etc.
- **Ingeniería directa**: del modelo al código aplicando reglas de correspondencia
- **Ingeniería inversa**: el proceso contrario

Ingeniería directa e inversa

- **Ingeniería directa** produce pérdida de información. Se necesita:
  1. Identificar las reglas de correspondencia
  2. Restringir ciertas características de UML
  3. Especificar el lenguaje de programación
  4. Usar las herramientas apropiadas
- **Ingeniería inversa** produce cantidad de información, pero no permite recrear exactamente el modelo

Recomendaciones

- No usar notación completa, es mejor comenzar por elementos básicos
- Ajustar perspectiva de los modelos a la etapa del proyecto:
  - Modelos **conceptuales**: análisis
  - Modelos de **especificación**: diseño
  - Modelos de **implementación**
- No construir modelos para todo

NOTAS

- Tipo de *adorno* aislado más importante
- Símbolos gráficos para mostrar **restricciones** o **comentarios** en formato libre
- Añaden al modelo requisitos, observaciones, revisiones y explicaciones

Dentro de una nota se puede escribir sólo texto, o incluir una URL o un enlace a un documento

Modelado de comentarios

- ☑ Enlazar la nota al bloque aludido
- ☑ No es necesario que una nota sea **visible** en todos los lugares donde aparezca el elemento o la relación
- ☑ Mover comentarios largos o complejos a un **documento externo**
- ☑ Mantener sólo las notas **significativas** no deducibles del propio modelo