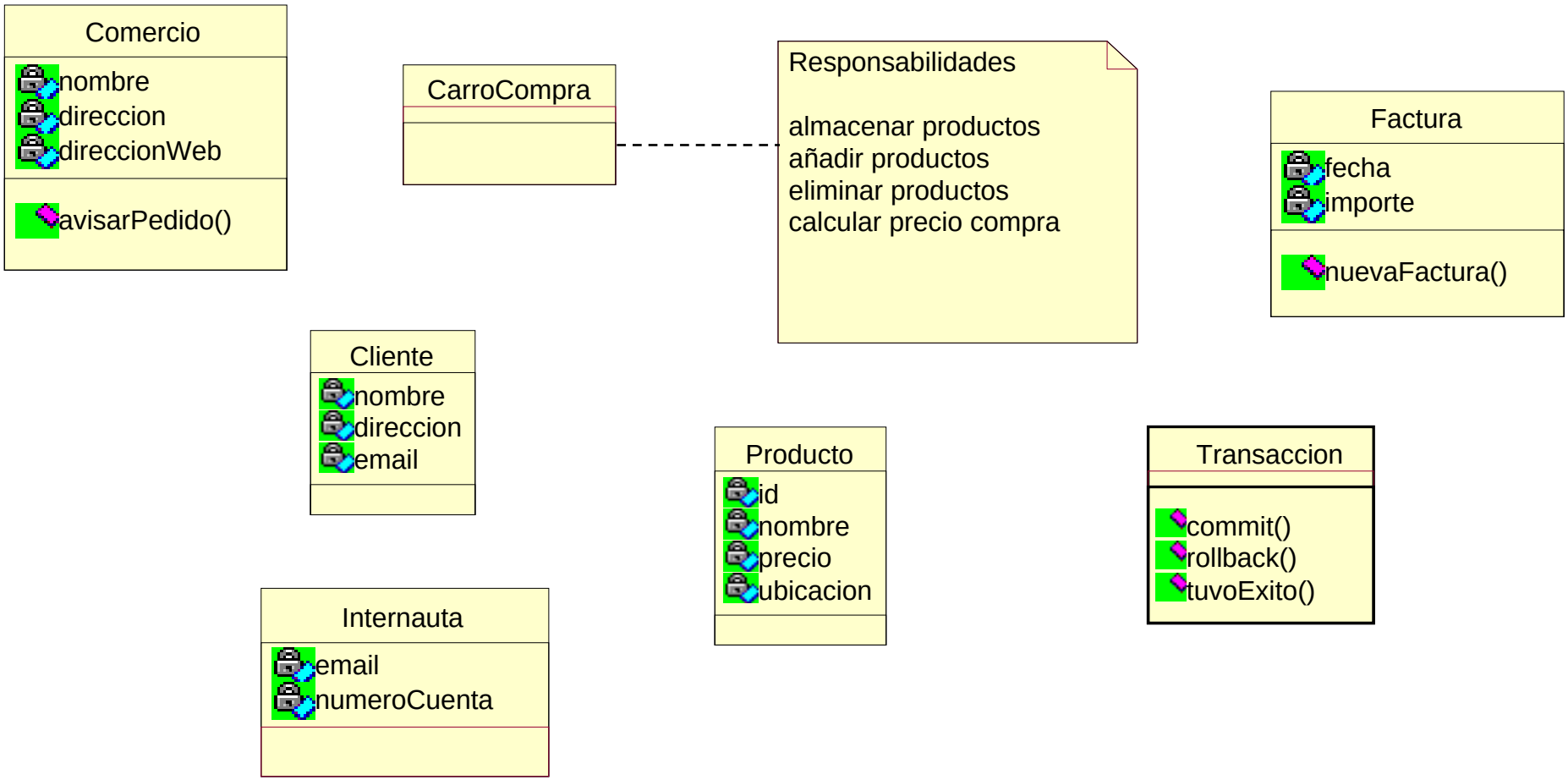
Modelado estructural

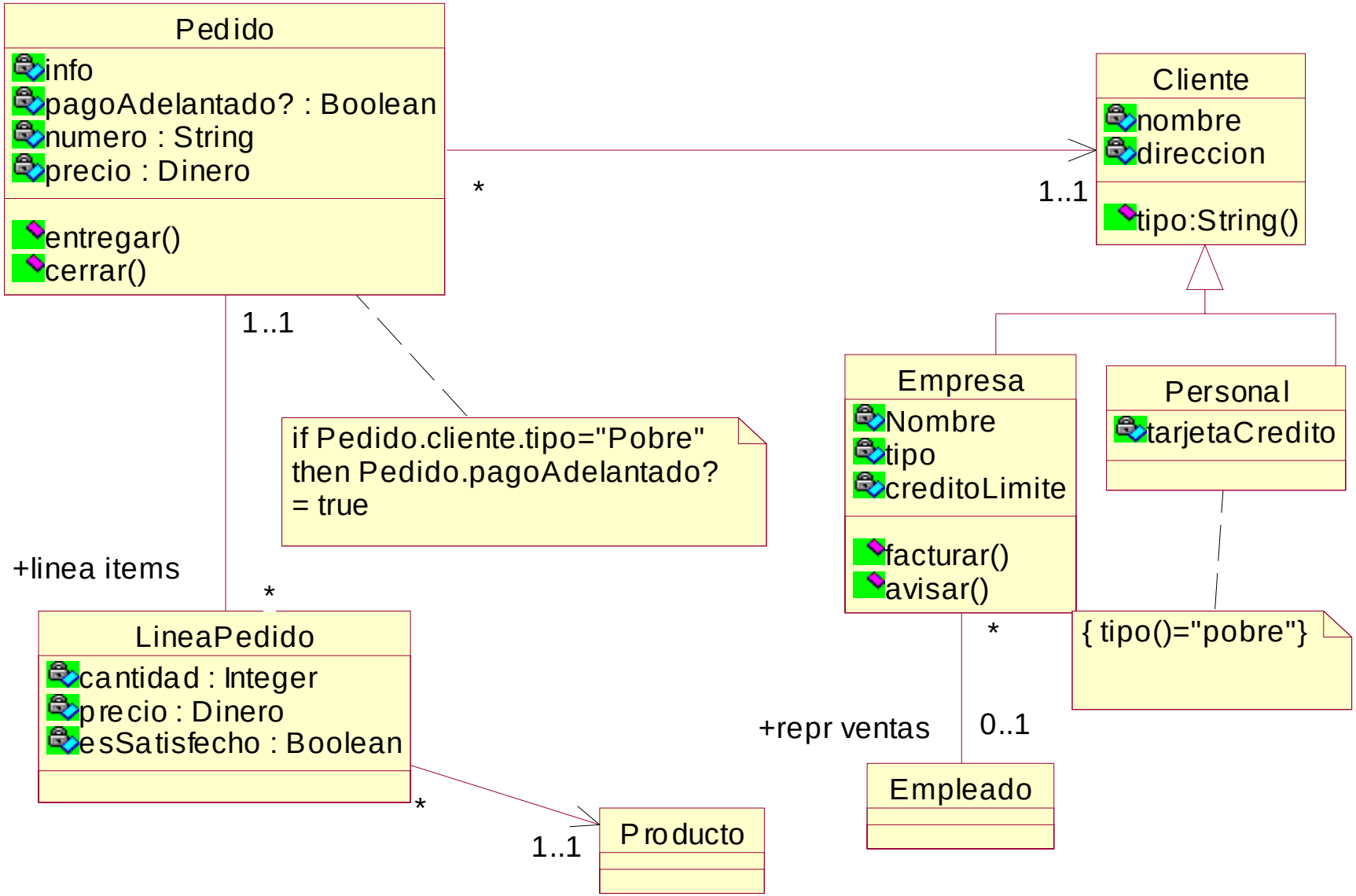
- Se describen los tipos de objetos de un sistema y las **relaciones estáticas** que existen entre ellos.
- Se expresa mediante los **diagramas de clase.**
- Normalmente contienen:
  - Clases
  - Interfaces
  - Relaciones de dependencia, realización, generalización y asociación (agregación, composición)
- También pueden incluir paquetes y colaboraciones.

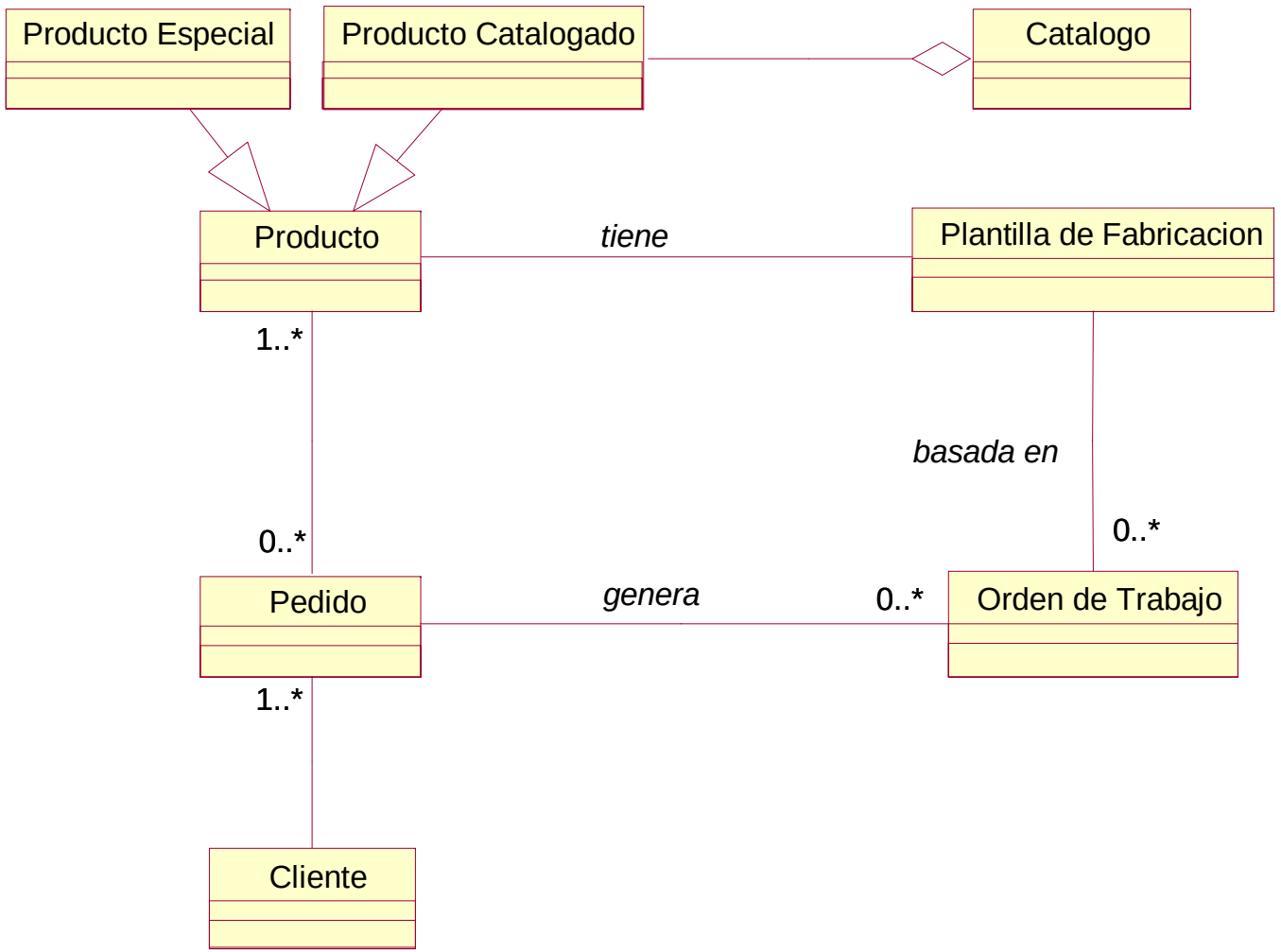
Diagramas de Clase

- Forman parte de la *vista de diseño estática* del sistema.
- Los diagramas de clase se usan para modelar:
  - **vocabulario del sistema**
    - identificar clases para abstracciones relevantes del dominio del problema
  - **colaboraciones (parte estática)**
    - identificar clases e interfaces cuya interacción produce el comportamiento deseado.
  - **esquema lógico de base de datos**

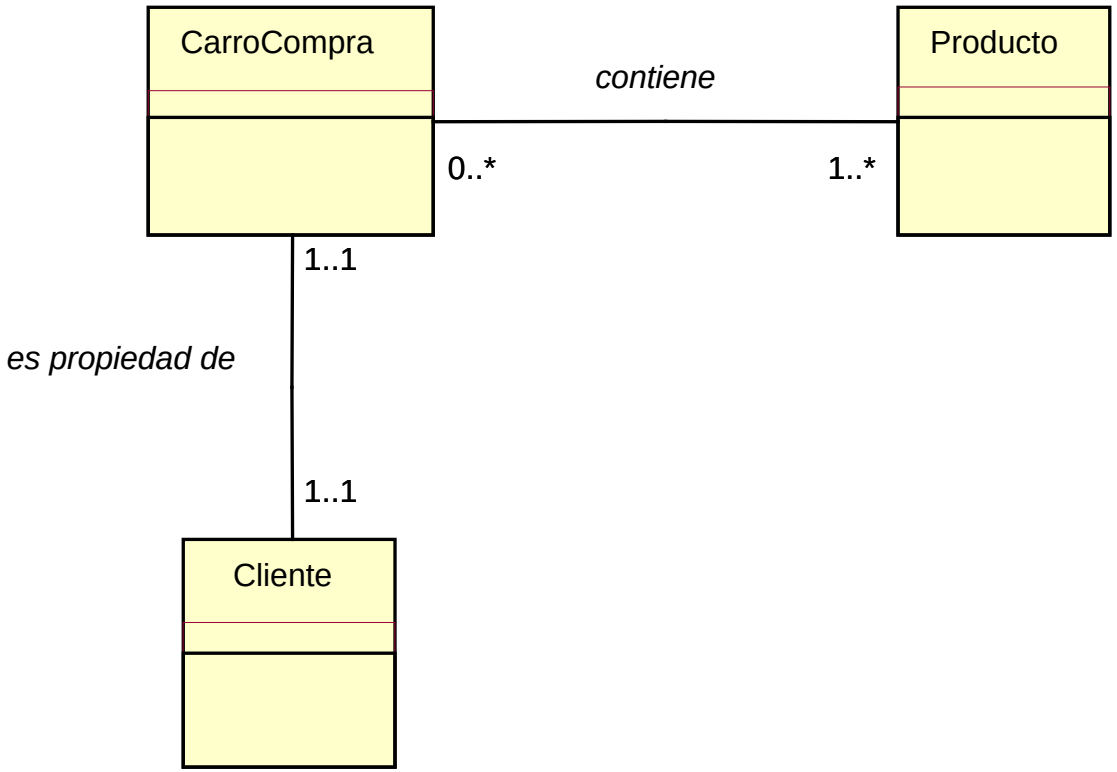
Vocabulario del sistema



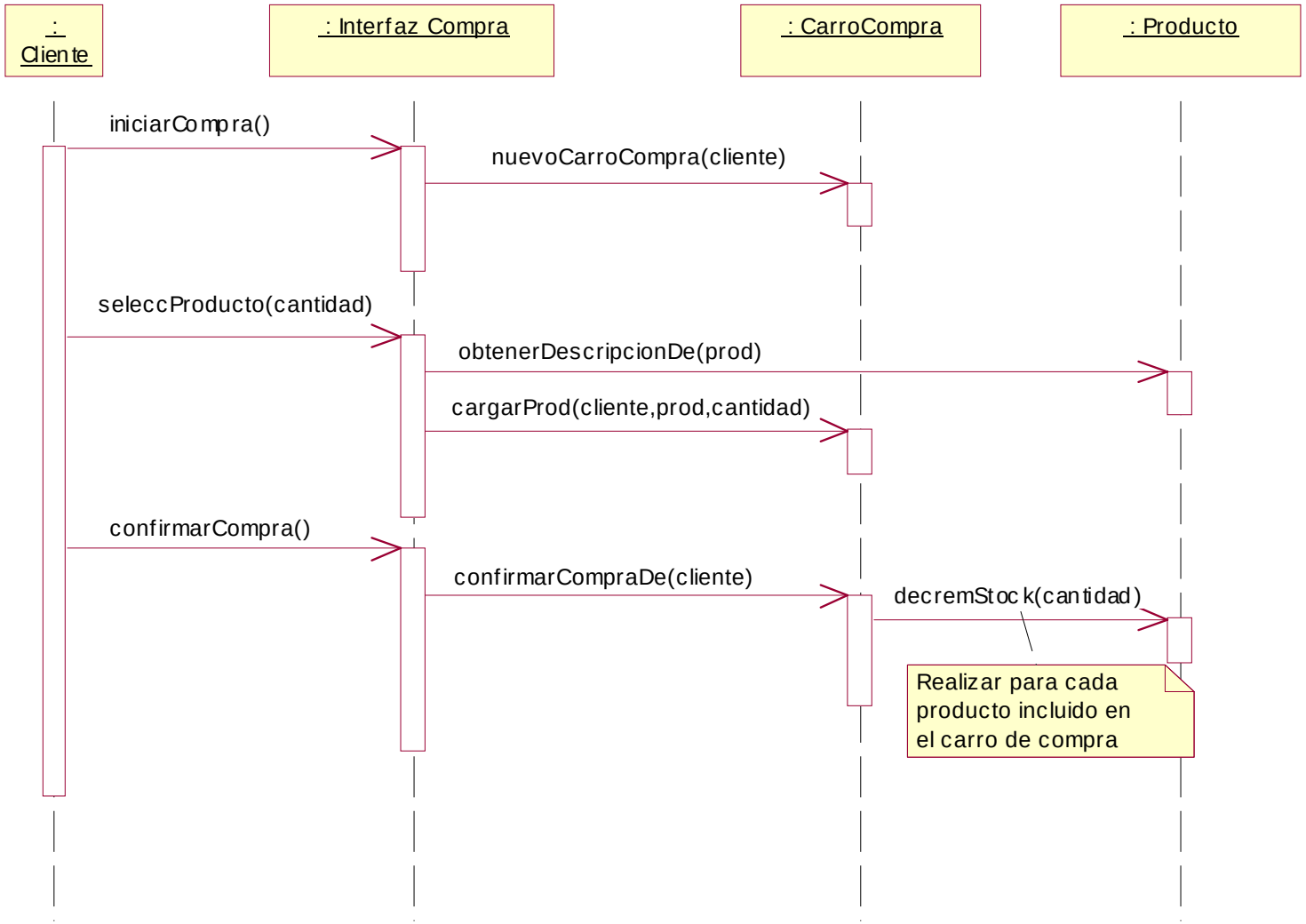

Pedido
info
pagoAdelantado? : Boolean
numero : String
precio : Dinero
entregar()
cerrar()
*
1..1
Cliente
nombre
direccion
tipo:String()
1..1
if Pedido.cliente.tipo="Pobre"
then Pedido.pagoAdelantado?
= true
Empresa
Nombre
tipo
creditoLimite
facturar()
avisar()
Personal
tarjetaCredito
{ tipo()="pobre"}
+linea items
*
LineaPedido
cantidad : Integer
precio : Dinero
esSatisfecho : Boolean
*
1..1
Producto
*
+repr ventas
0..1
Empleado


Producto Especial
Producto Catalogado
Catalogo
Producto
tiene
Plantilla de Fabricacion
1..*
basada en
0..*
0..*
Pedido
genera
0..*
Orden de Trabajo
1..*
Cliente

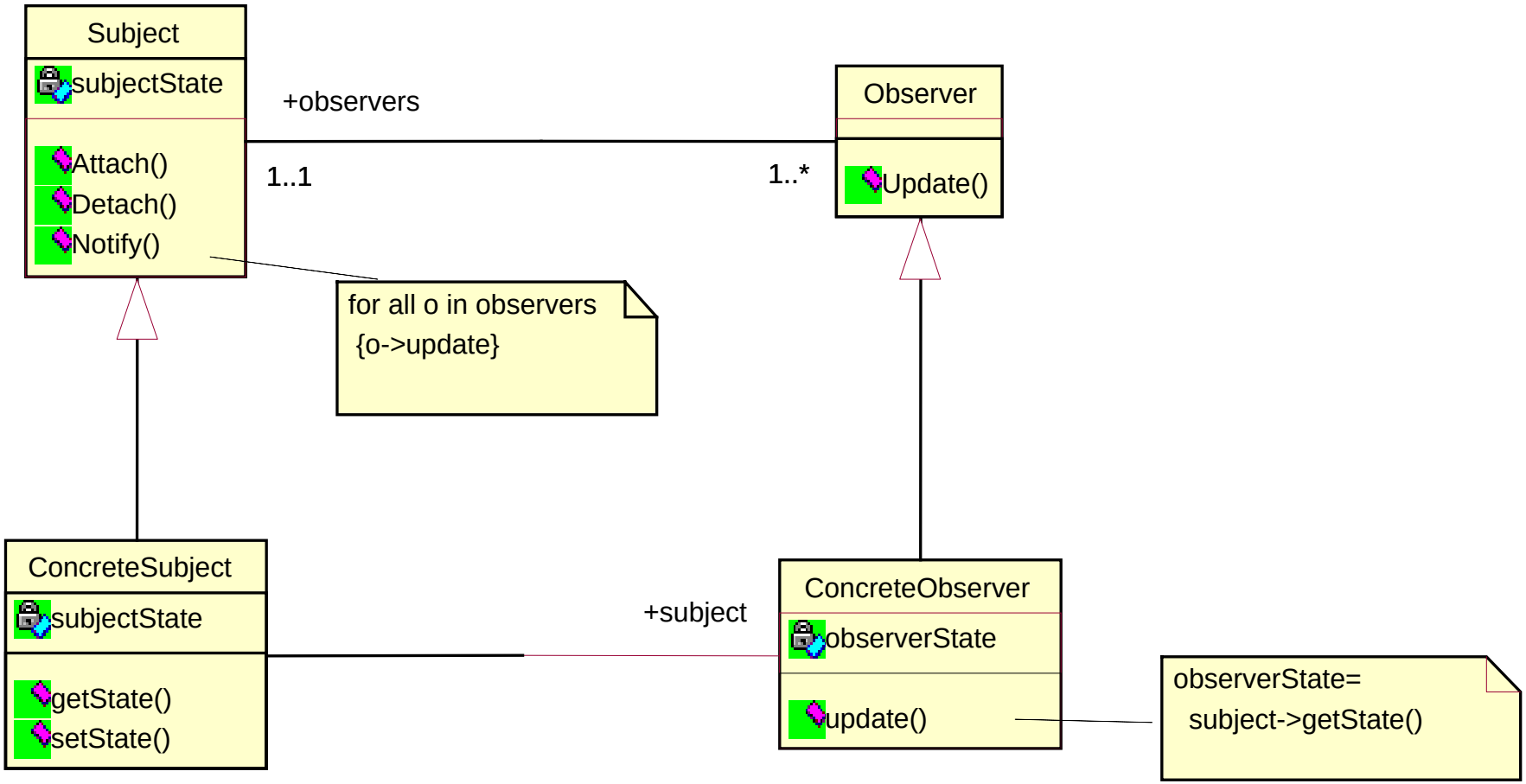
Colaboración (Parte Estática)



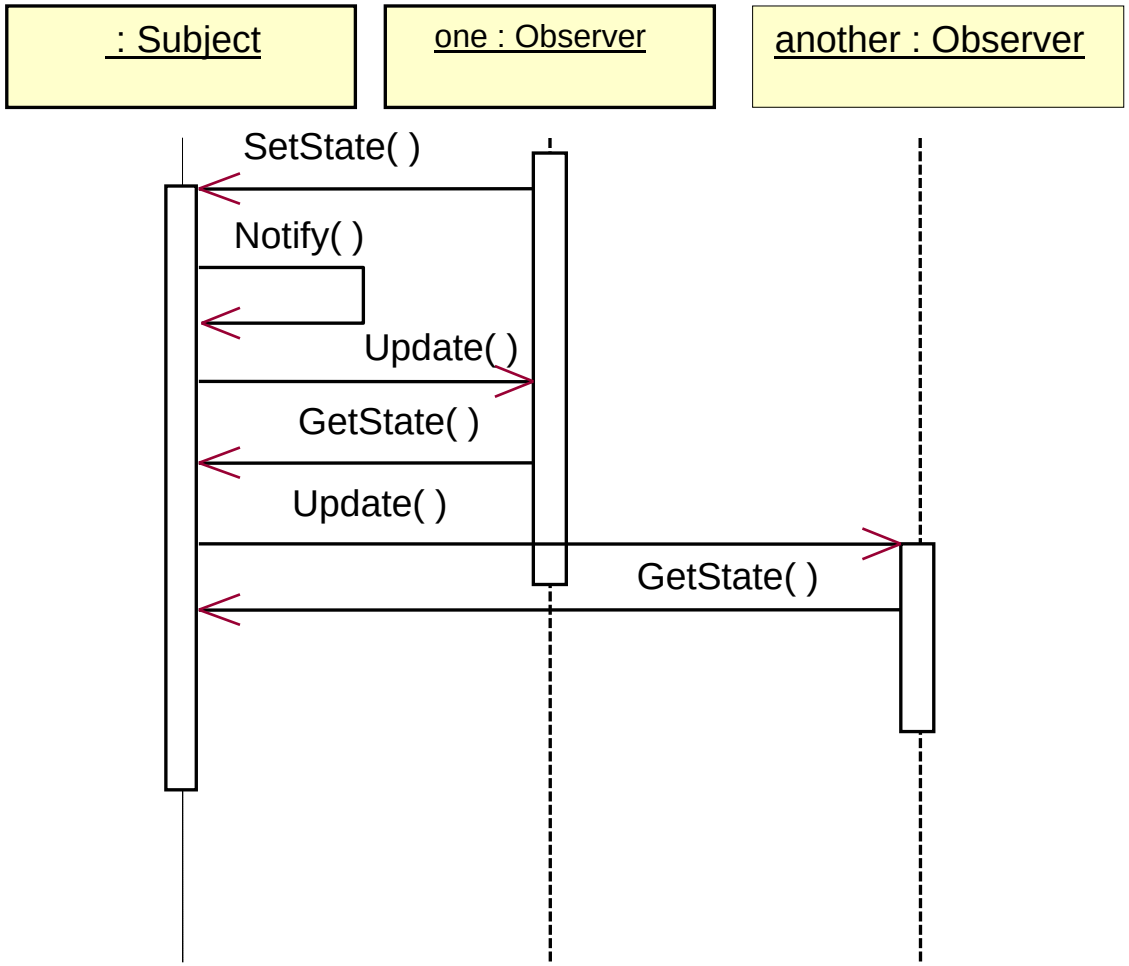
Colaboración (Parte dinámica)



Patrón de diseño (Parte Estática)



Patrón de diseño (Parte dinámica)



Diagramas de Clase

Diferentes perspectivas:

Conceptual

Se representan los conceptos del dominio estudiado.

Especificación

Se representan los tipos que representan una interface la cual puede tener cualquier implementación

Implementación

Se representan las clases tal y como serán implementadas

Clases y asertos

- Posibilidad de incluir en la especificación:
  - Precondiciones
  - Postcondiciones
  - Invariante
- Recomendación: Usar "*Diseño por Contrato*"

Ingeniería directa e inversa

- **Ingeniería directa**
  - Transformar modelos en código en un lenguaje de programación determinado
- **Ingeniería inversa**
  - Obtener un modelo a partir de código.
  - Más difícil ya que hay pérdida de información al pasar de los modelos al código.

Atributos

[visibilidad] nombre [: tipo] [= valor_inicial] [{propiedades}]

visibilidad
- \+ = pública
- # = protegida
- \- = privada

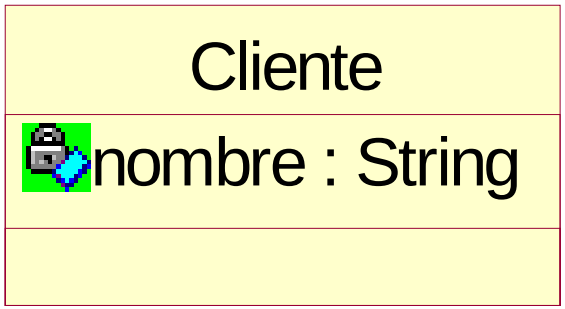
nombre: nombre del atributo

tipo: tipo del atributo

valor_inicial: valor inicial o por defecto

propiedades: {frozen} {addOnly}

Diagramas de Clase: Atributos



- Nivel **Conceptual**: “Los clientes tienen un nombre”
- Nivel de **Especificación**: “El cliente puede almacenar y consultar su nombre”
- Nivel de **Implementación**: “Cliente tiene un campo de tipo string que almacena su nombre y un método que lo devuelve”

Operaciones

[visibilidad] nombre [(lista_parametros)] [: tipo_retorno]
[{propiedades}]

visibilidad
- \+ = pública
- # = protegida
- \- = privada

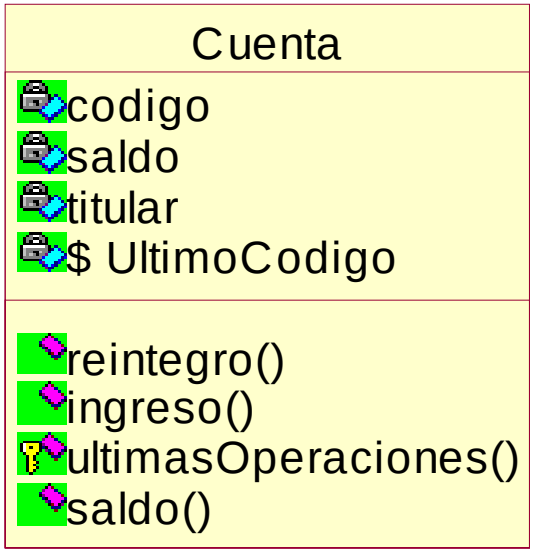
nombre: nombre de la operación

lista_parámetros: lista de parámetros separados por comas

tipo retorno: tipo de valor devuelto por la operación

propiedades: {isQuery}, {sequential}, {concurrent}

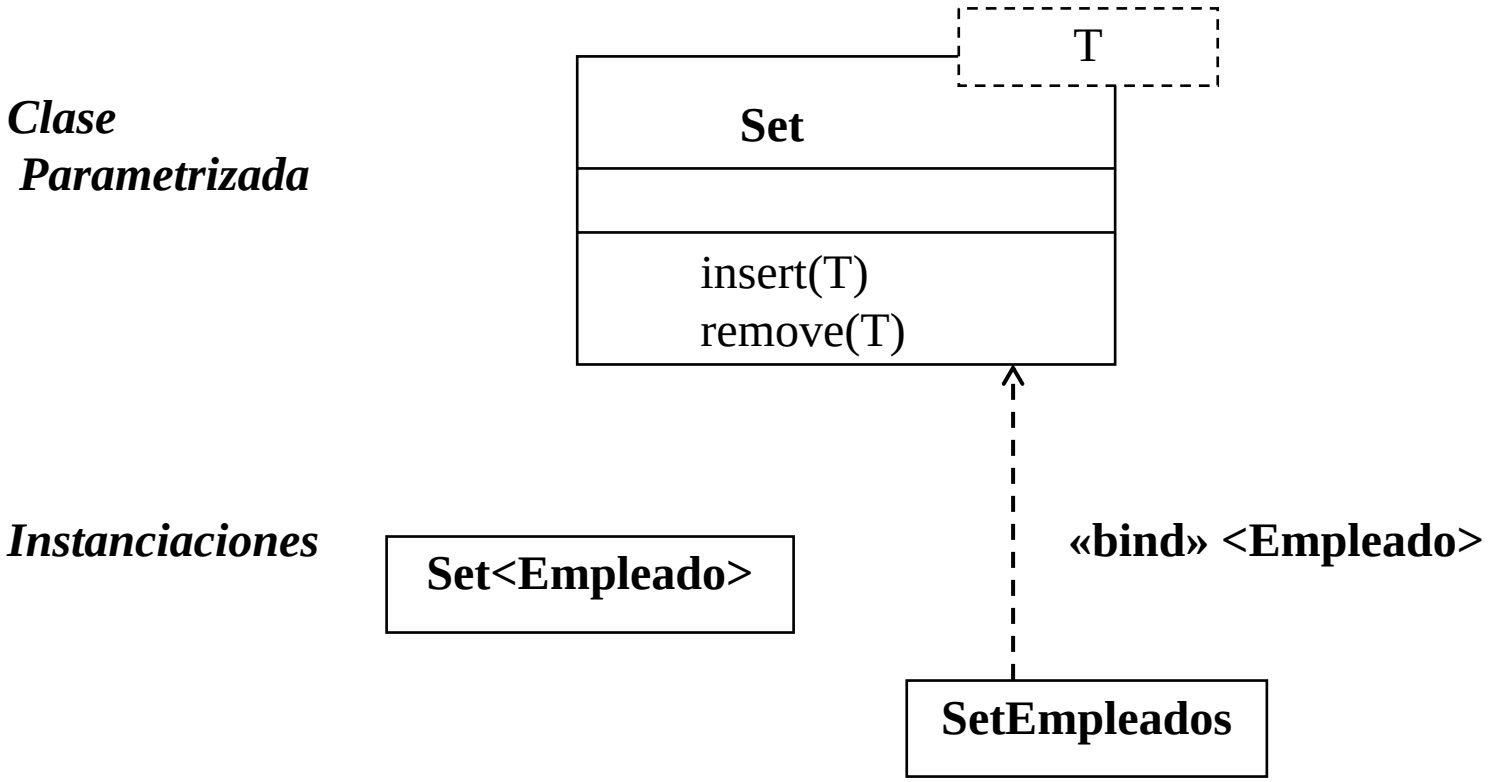
Diagramas de clase: Operaciones



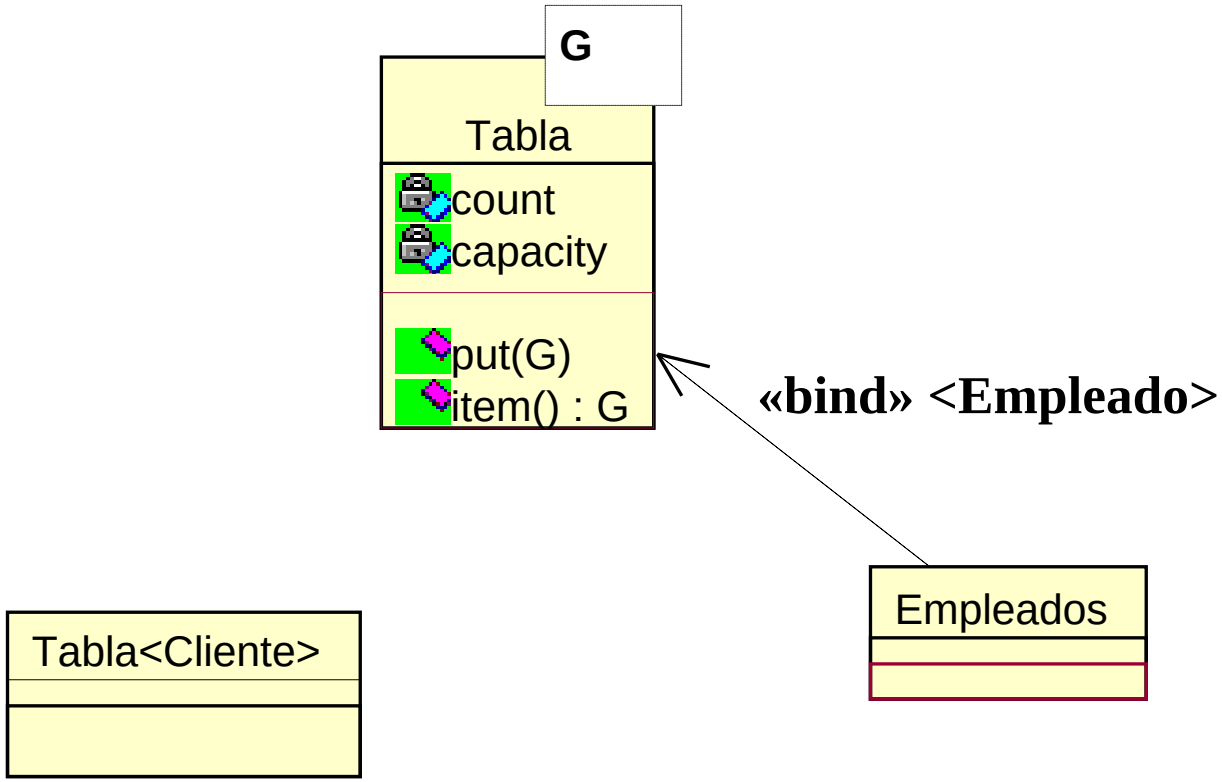
Diagramas de Clase: Operaciones

- Nivel Conceptual
  - Responsabilidades de la clase
  - Tarjetas CRC: Descripción de alto nivel del propósito de la clase
- Nivel Especificación
  - Protocolo de la clase (operaciones públicas)
- Nivel Implementación
  - Conjunto de métodos de la clase

Clases Parametrizadas

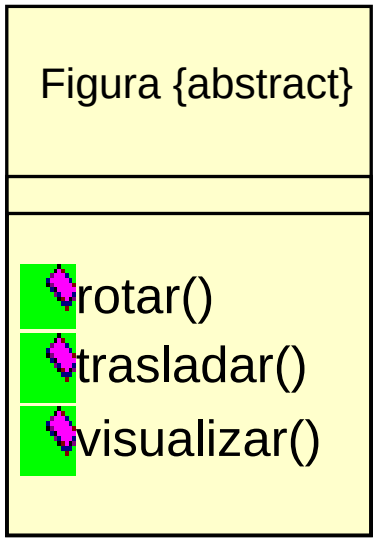


Clases Parametrizadas



Otras propiedades

- Clases diferidas
- Multiplicidad
- Variables y métodos de clase

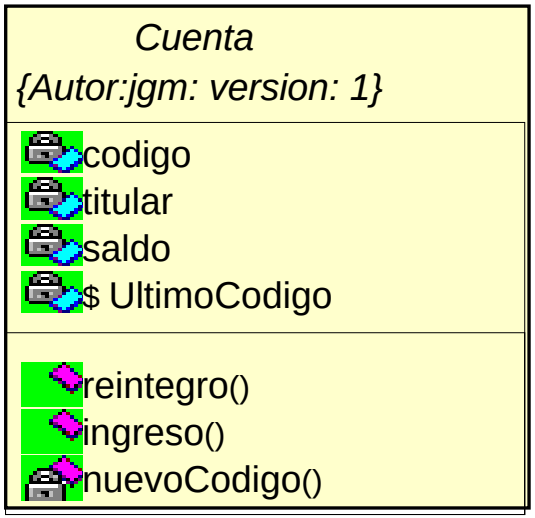


Clases Estereotipadas

<<metaclass>>
MetaclaseCuenta

<<exception>>
FueraRango

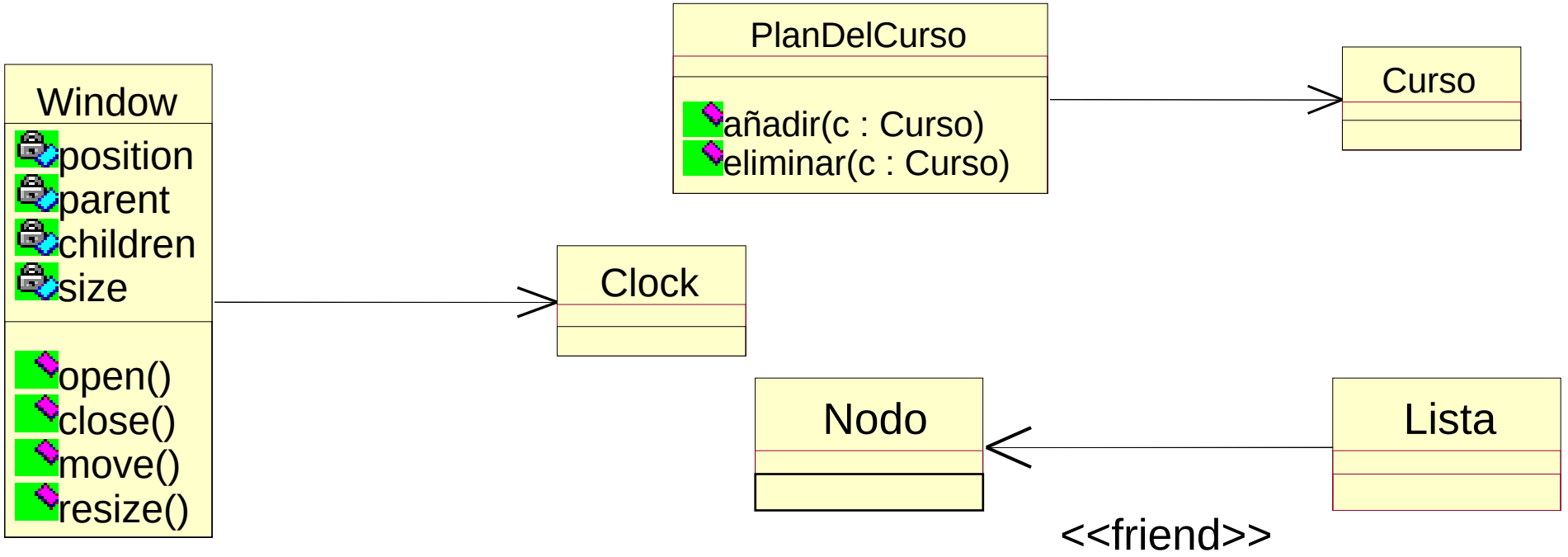
Clases y valores etiquetados



Diagramas de Clases: Relaciones

- **Dependencia**

Un cambio en la especificación de un elemento afecta a otro

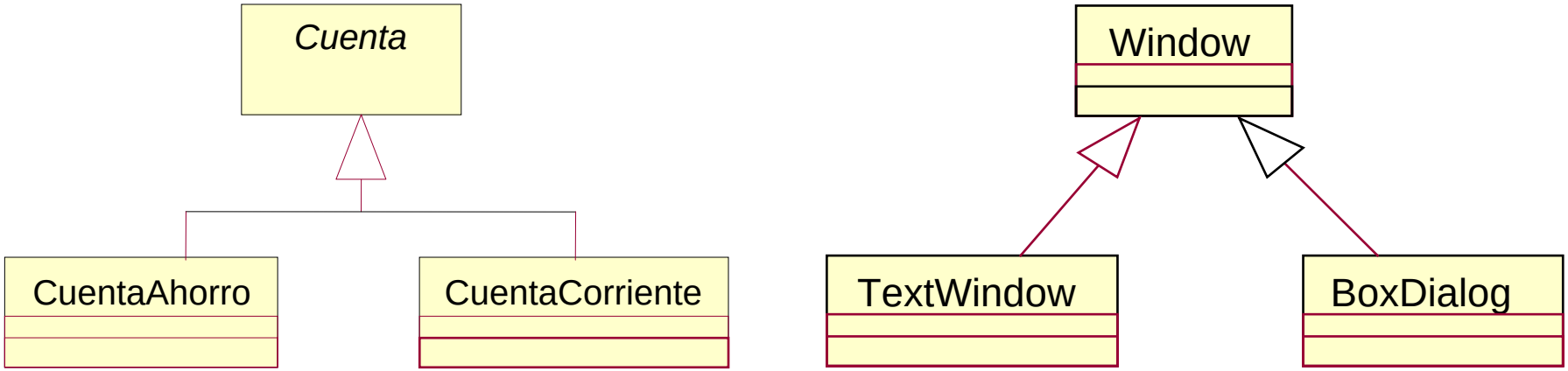


Estereotipos para dependencias

- **bind**: entre una clase genérica y una instanciación
- **friend**: dependencia de clase amiga
- **refine**: relación de refinamiento
- **use**: relación de uso
- **import**: un paquete importa los elementos de otro
- **extend**: para casos de uso
- **include**: para casos de uso

Diagramas de Clases: Relaciones

- **Generalización**
  - *"Es-un-tipo-de"*



Diagramas de Clase: Generalización

- Nivel **Conceptual**
  - "Todas las instancias de *CuentaCorriente* son instancias de *Cuenta*"
- Nivel **Especificación**
  - "La interface de *CuentaCorriente* incluye la interface de *Cuenta*"
  - Principio Sustitución
- Nivel **Implementación**
  - Herencia

Adornos para la generalización

- **implementation** (estereotipo): herencia privada C++
- **complete/incomplete** (restricción):
  - ¿se han especificado todos los descendientes?
- **Disjoint/overlapping** (restricción):
  - clasificación estática vs. clasificación dinámica

Restricciones semánticas entre las subclases

OVERLAPPING

Una nueva clase puede ser subclase de más de una subclase

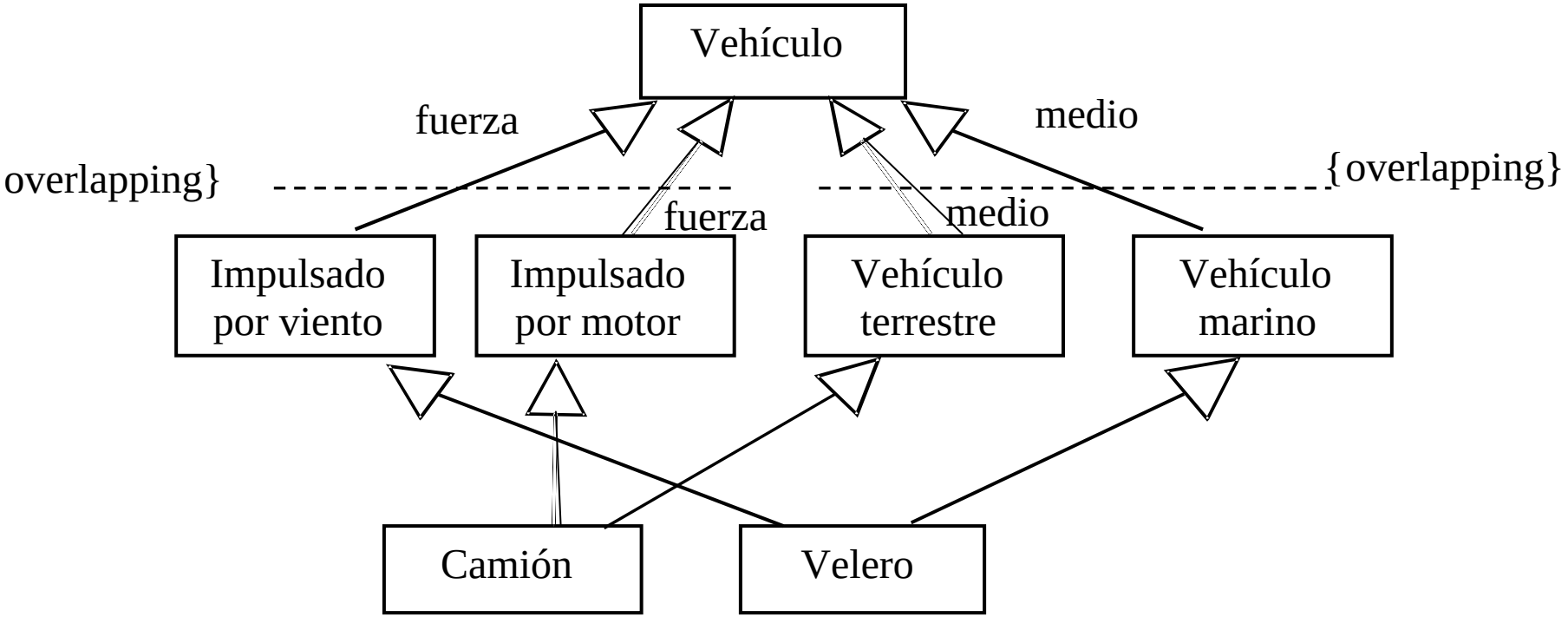
Una instancia puede ser instancia directa o indirecta de dos más subclases

DISJOINT

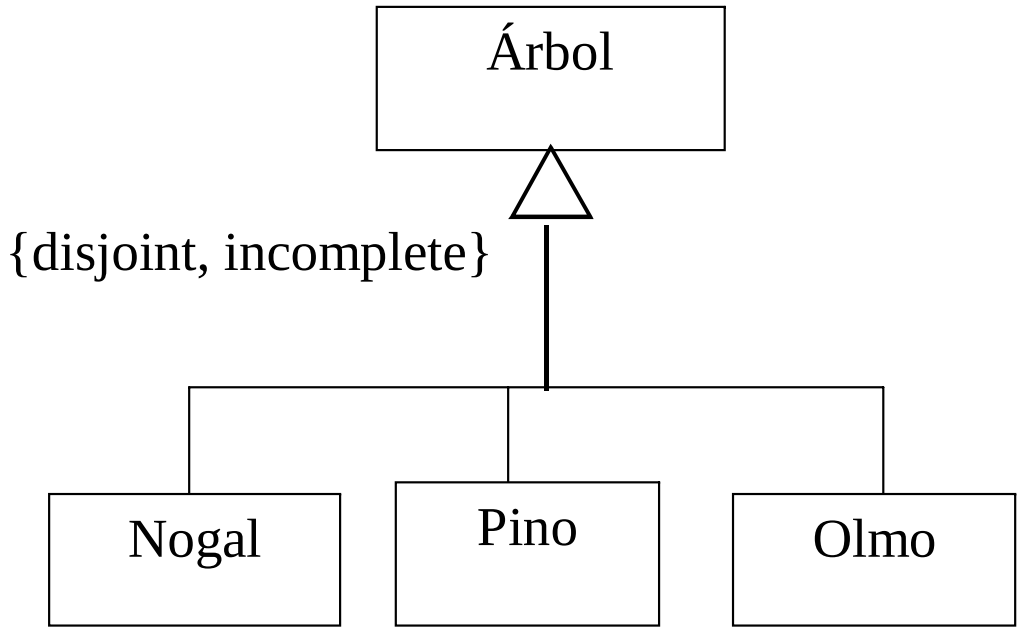
Una nueva clase no puede ser subclase de más de una subclase.

Instancias de una única clase.

Generalización: “overlapping”

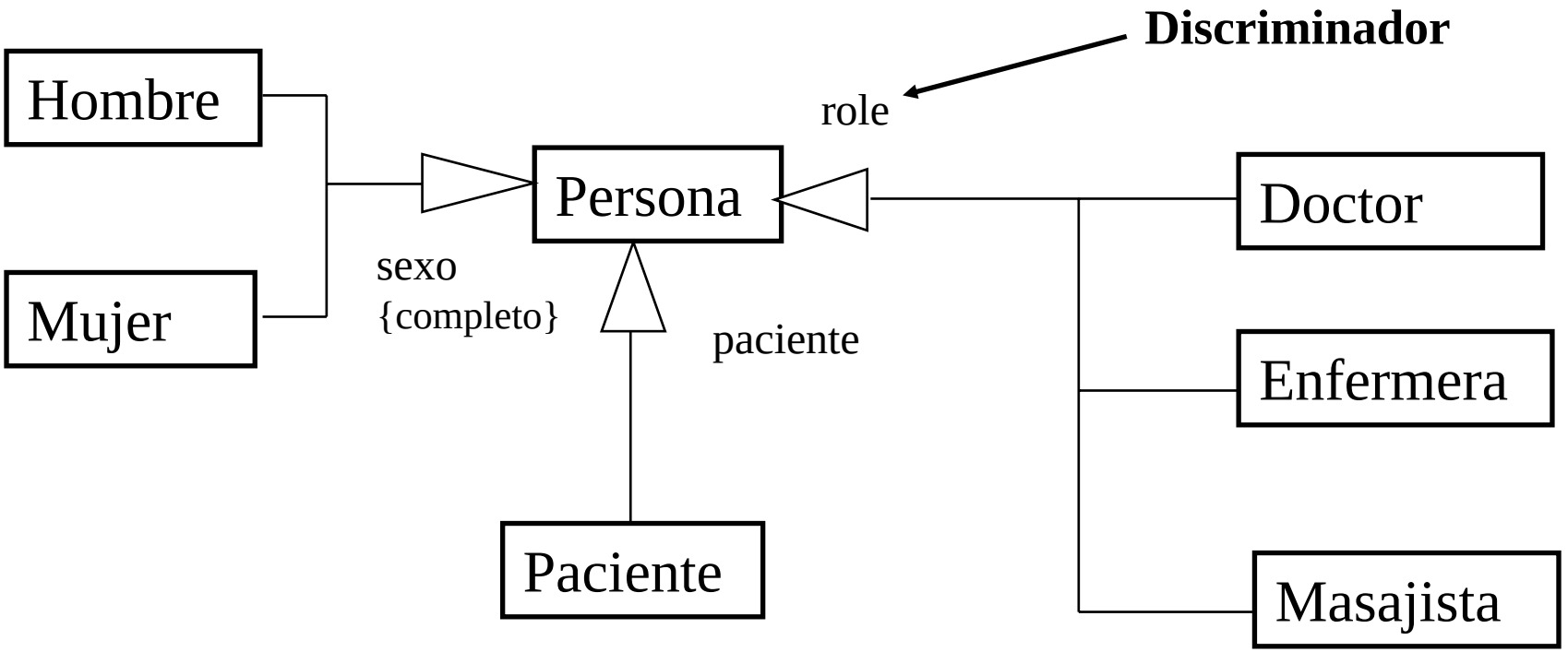


Generalización: “disjoint”



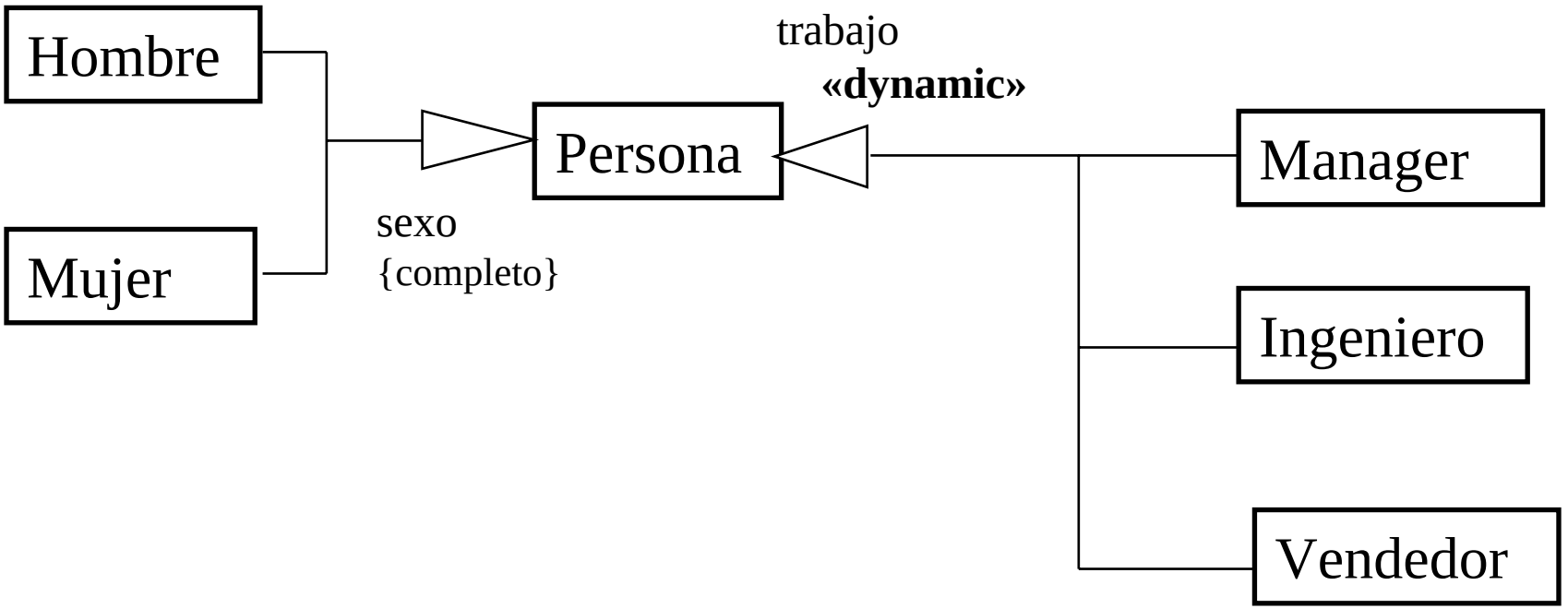
Clasificación Múltiple

- Un objeto puede ser instancia de más de una clase, no necesariamente conectadas por la herencia



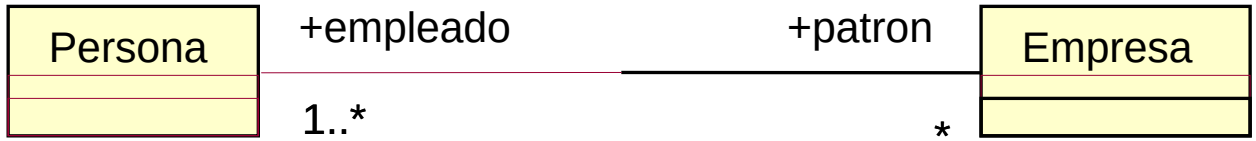
Clasificación Dinámica

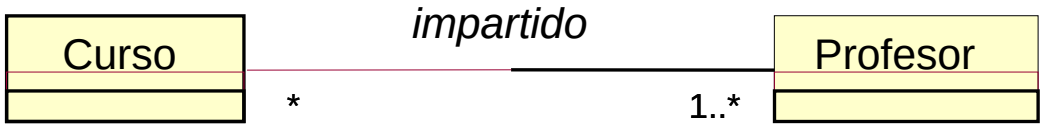
- Un objeto puede cambiar de clase dentro de la jerarquía de subclases.



Diagramas de Clase: Asociación

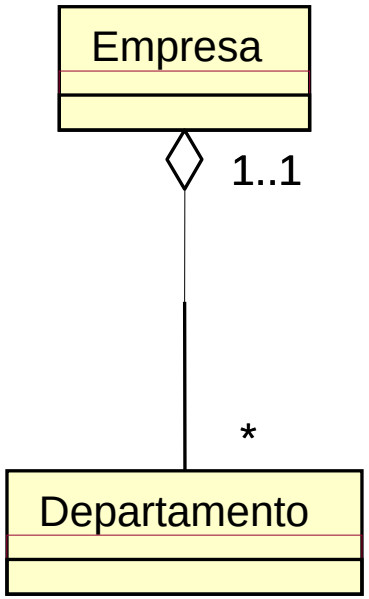
- **Asociación**
  - Relación estructural que especifica que los objetos de un tipo están conectados con los de otro.





Asociaciones

- **Agregación**
  - Caso especial de asociación
  - Relación estructural *parte-de*



Asociaciones

- Nivel **Conceptual**
  - Muestran la relación conceptual entre dos clases.

    "Un cliente tiene varios pedidos"
- Nivel de **Especificación**
  - Representan responsabilidades
  - Detectamos los mensajes del protocolo de una clase con respecto a la otra
- Nivel de **Implementación**
  - Establecer atributos: navegabilidad

Asociaciones

- **Especificación**:

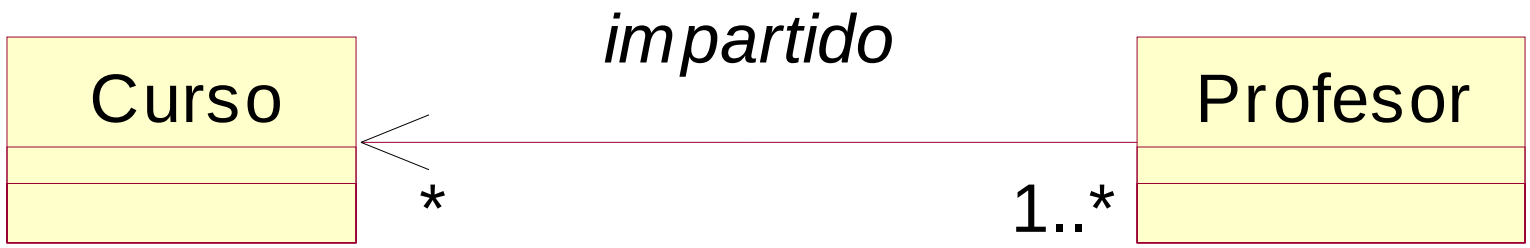
```
class Pedido {
        public  Cliente  getCliente;
        public  Set      getLineaPedido;... }
```

- **Implementación**

```
class Pedido {
        private Cliente    _cliente;
        private Set        _lineasPedido; …}
```

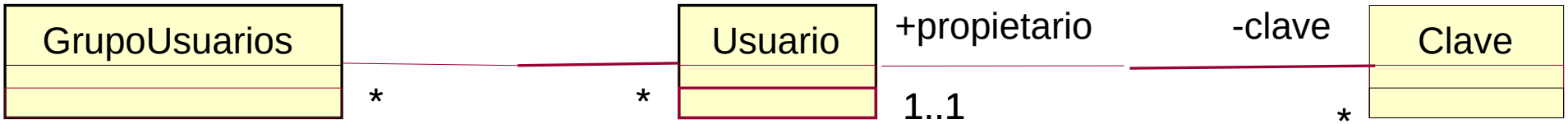
Navegación

- Posibilidad de limitar la navegación a una sola dirección
- Determina si una clase de la asociación tiene “conocimiento” de la otra.
- Nivel de especificación o implementación

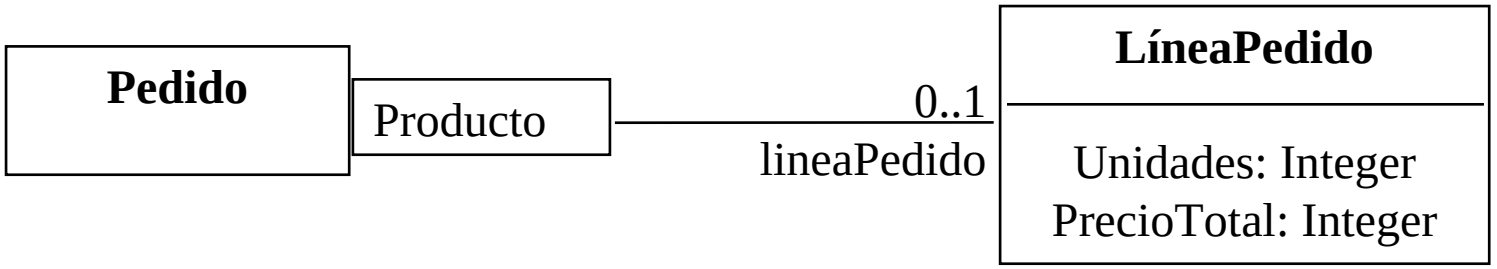


Visibilidad

- Pública: +propietario
- Protegida: #propietario
- Privada: -propietario



Asociaciones calificadas



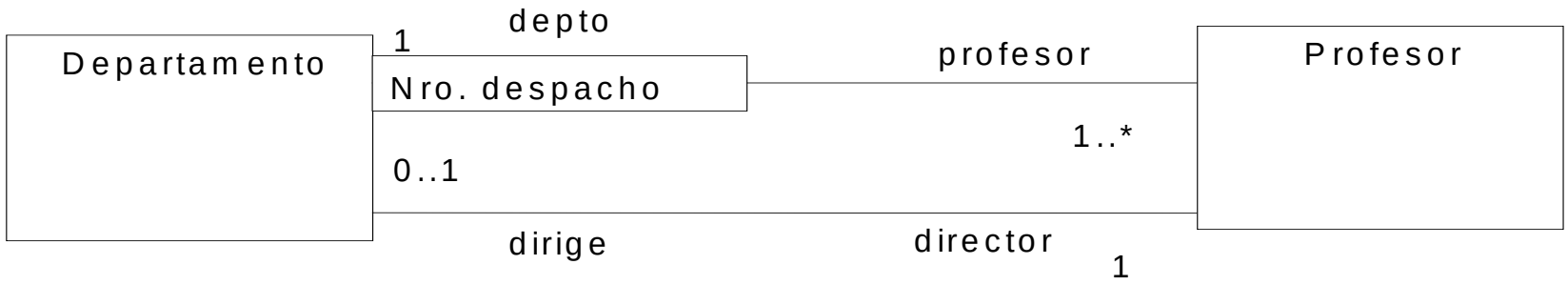
- N. **Conceptual**: “Dentro del mismo pedido no pueden existir dos líneas con el mismo producto”
- N. **Especificación**: “El acceso a *lineaPedido* es indexado por productos”
- N. **Implementación**: “Se usa una tabla para almacenar las líneas de pedido”

Asociaciones calificadas

```
Class Pedido {
    private   Tabla          _lineasPedido;

    public LineaPedido  getLineaPedido(Producto unProducto);
    public void addLineaPedido (Integer cantidad, Producto elProducto);
    …
}
```

Ejemplo


Departamento
1
depto
Nro. despacho
0..1
dirige
profesor
1..*
director
1
Profesor

Agregación

- Dos criterios:
  - Dependencia:

    ¿La existencia de una parte va ligada a la del agregado?
  - Exclusividad:

    ¿Una parte puede pertenecer a más de un agregado?
- Cuatro posibles tipos de agregación

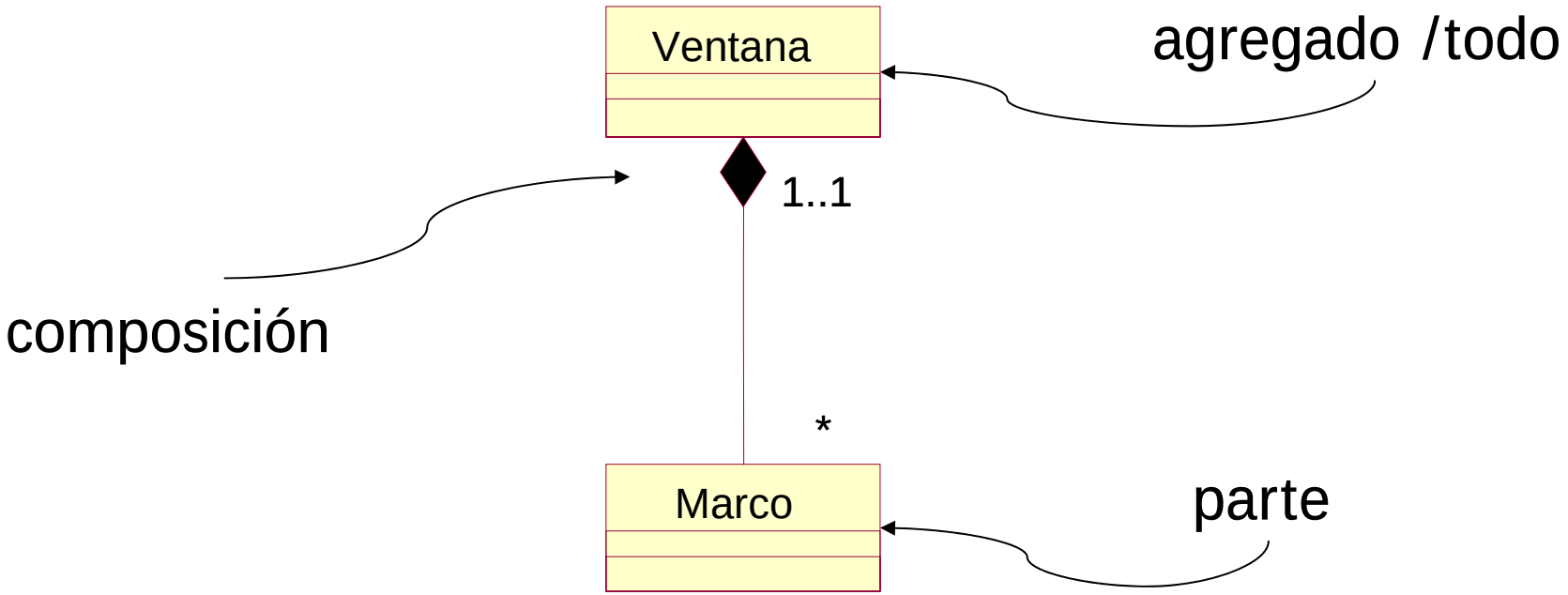
Composición

- Es un caso particular de agregación:

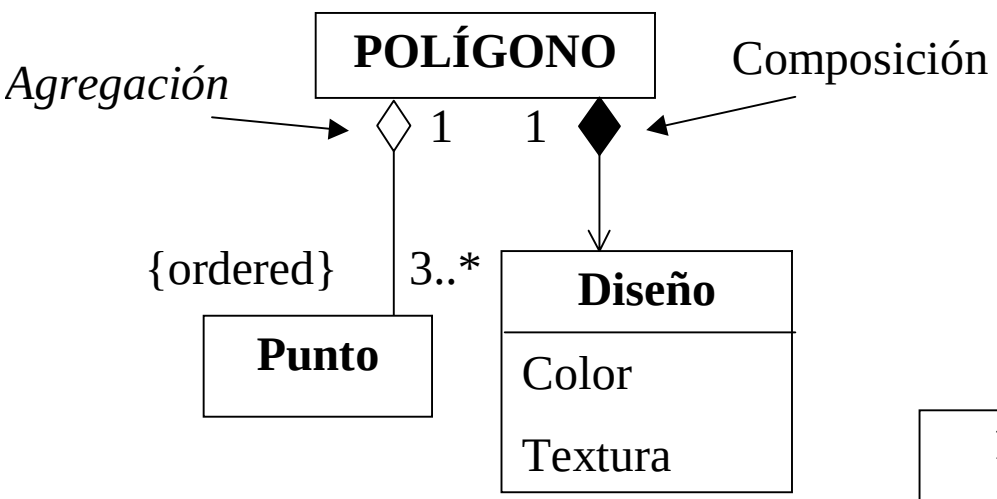
  exclusiva y dependiente

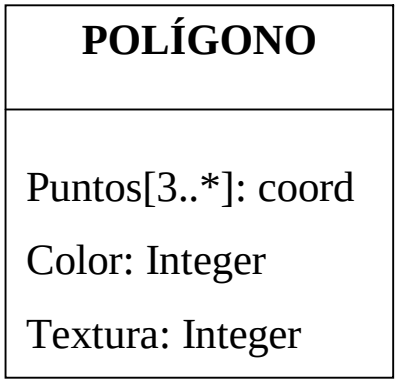
- Las partes pueden crearse después del agregado compuesta al que pertenecen, pero una vez creadas viven y mueren con ella.
- La parte sólo puede formar parte de un agregado.
- El agregado gestiona la creación y destrucción de las partes.
- Las partes se pueden eliminar antes de eliminar el agregado.

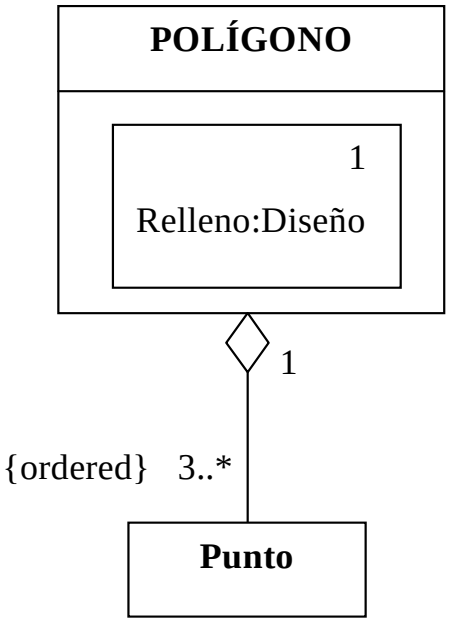
Composición



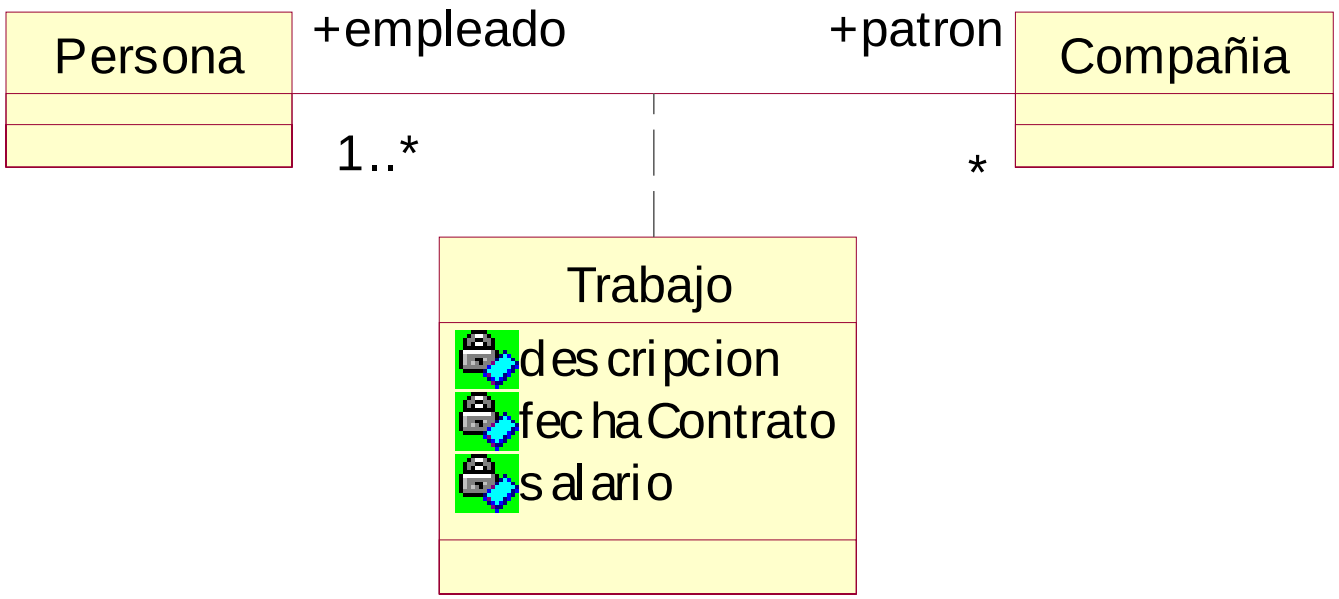
Composición







Clases Asociación



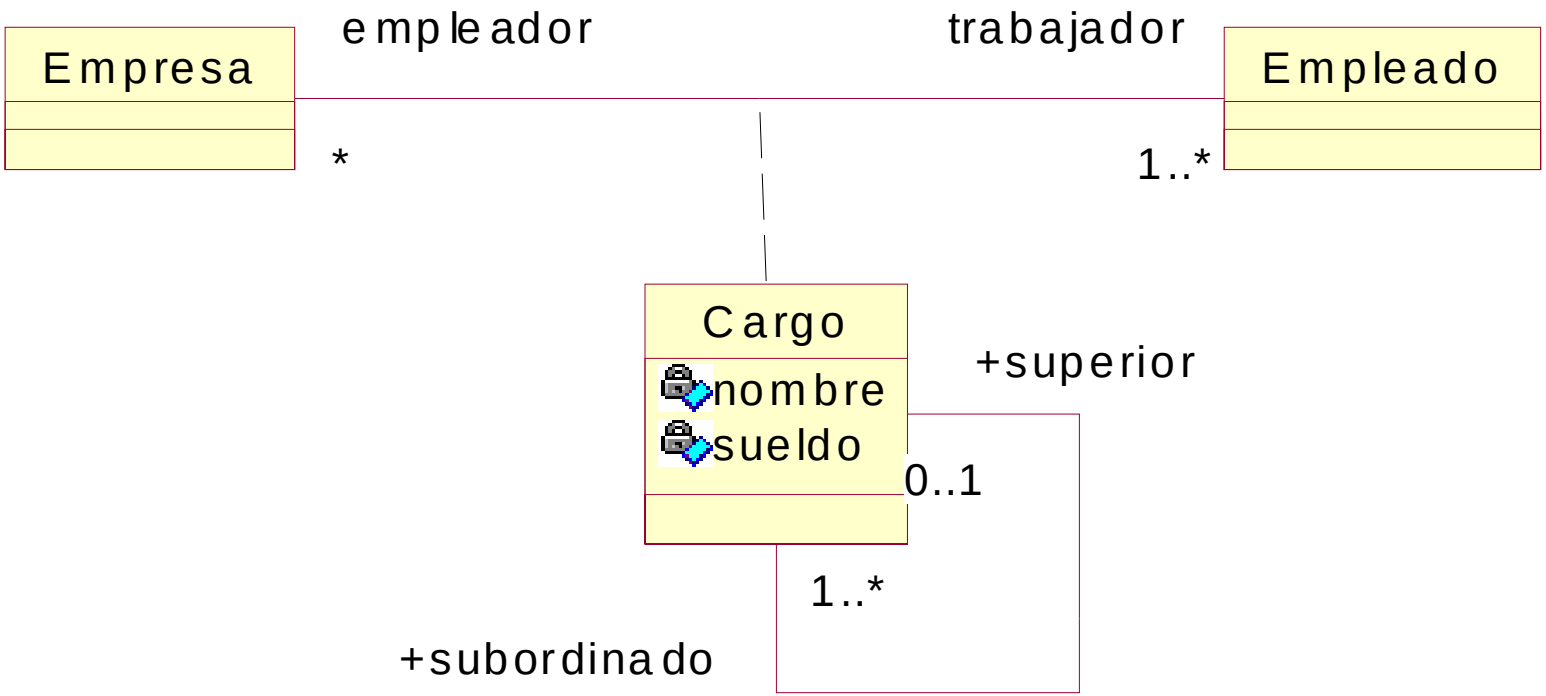
Clases Asociación

- Una clase asociación añade una restricción:

  “Sólo puede existir una instancia de la asociación entre cualquiera par de objetos participantes”

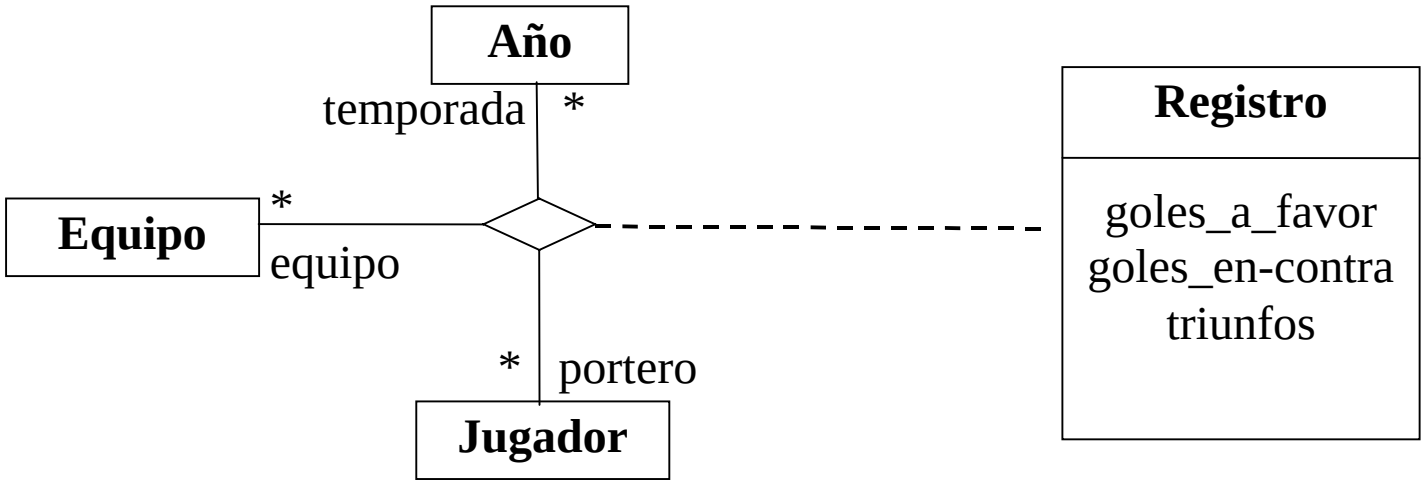
- No podríamos modelar que una persona tiene diferentes contratos para una misma compañía a lo largo del tiempo.

Ejemplo

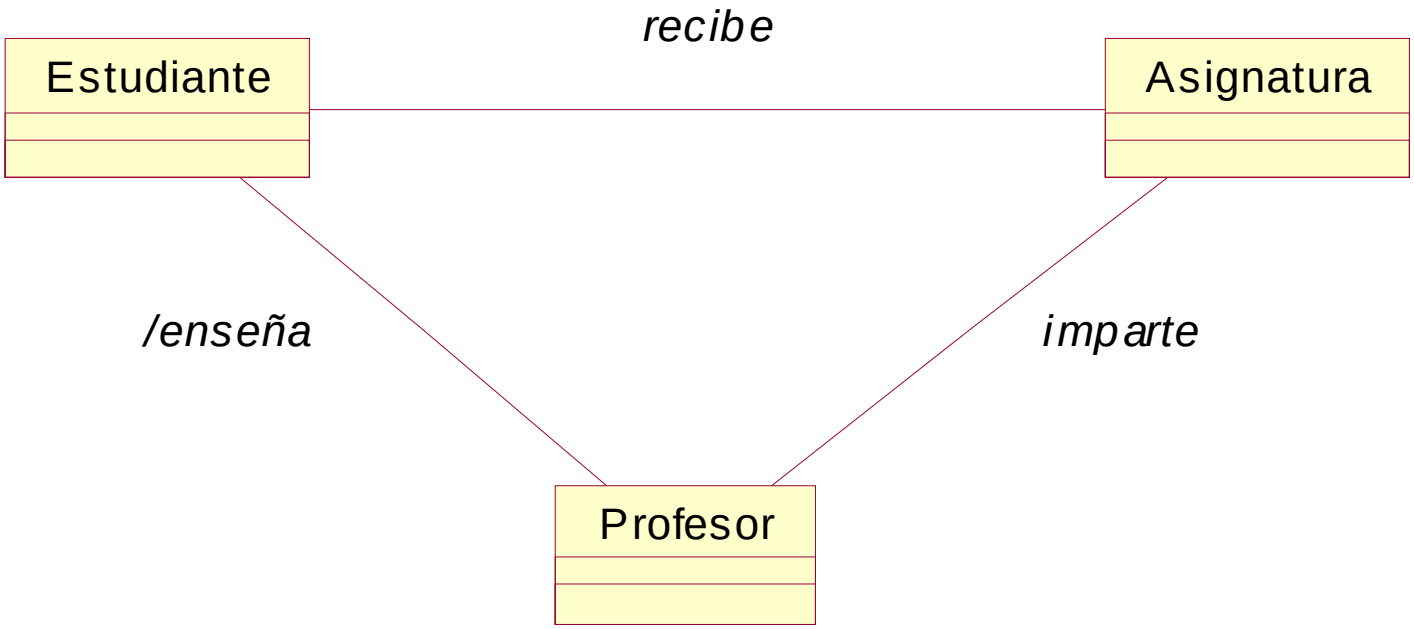

Empresa
empleador
trabajador
Empleado
*
1..*
Cargo
nombre
sueldo
+superior
0..1
1..*
+subordinado

Asociaciones n-arias

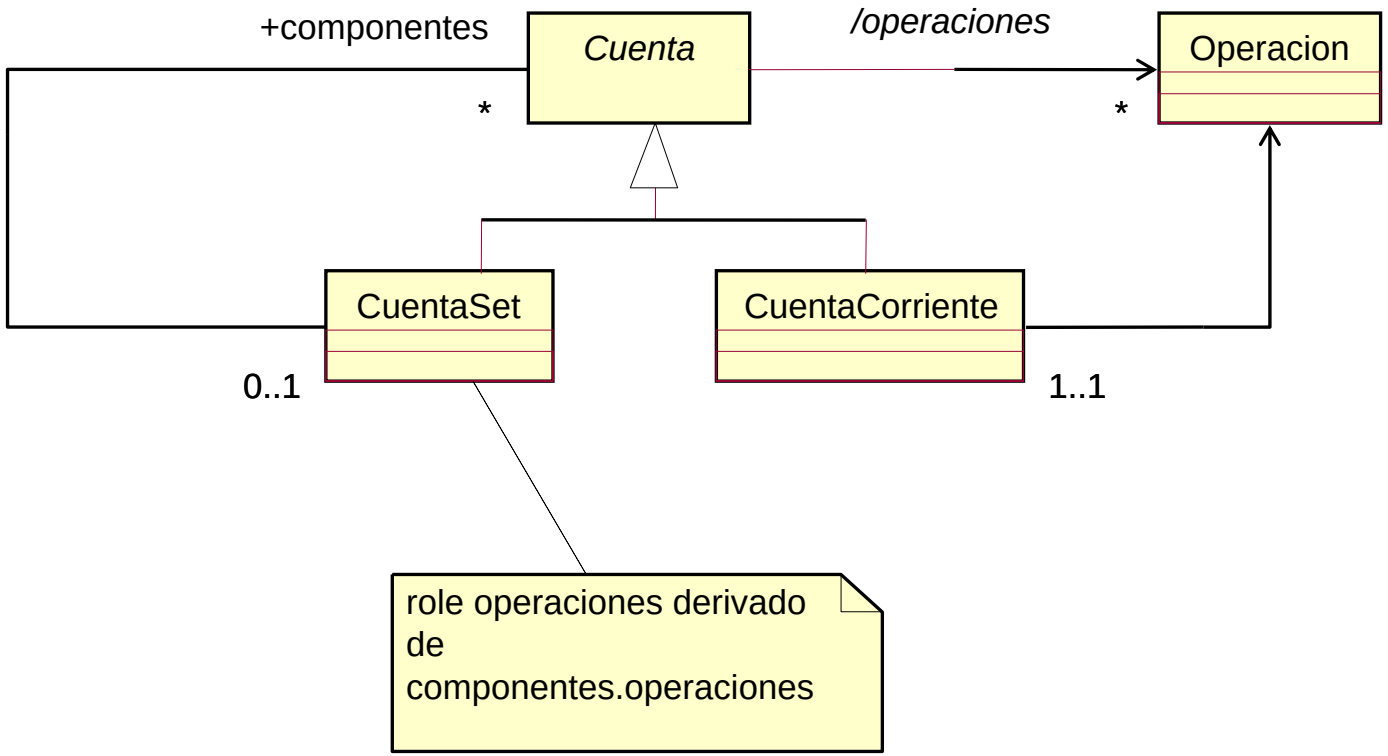
- Asociación entre tres o más clases.
  - Cada instancia de la asociación es una n-tupla de valores de cada una de las respectivas clases .



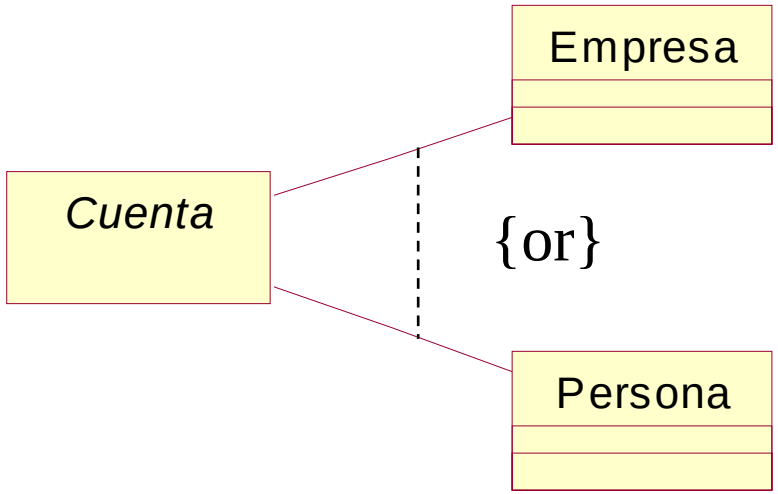
Asociaciones derivadas

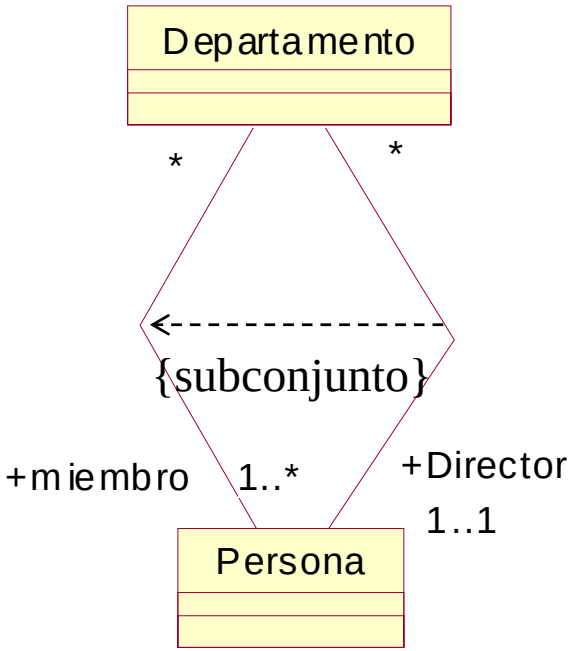


Asociaciones derivadas

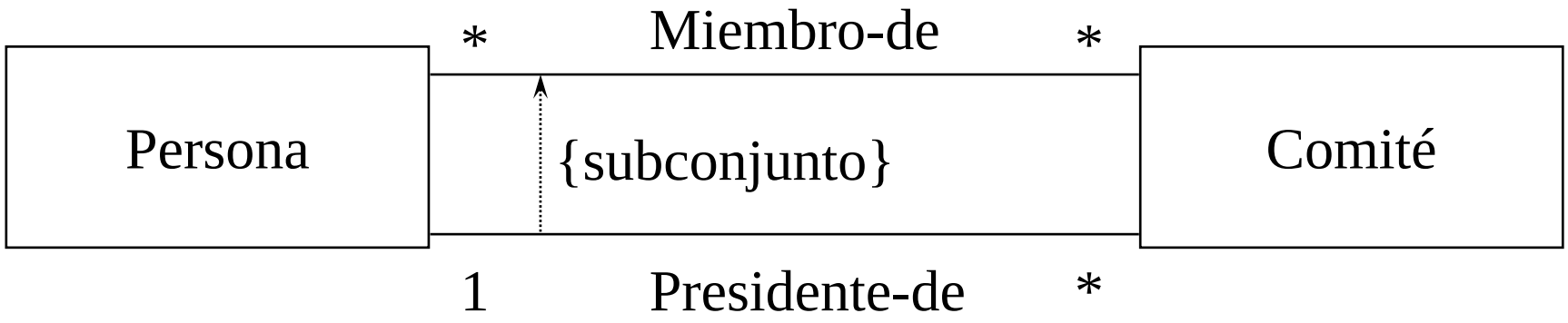


Restricciones para Asociaciones

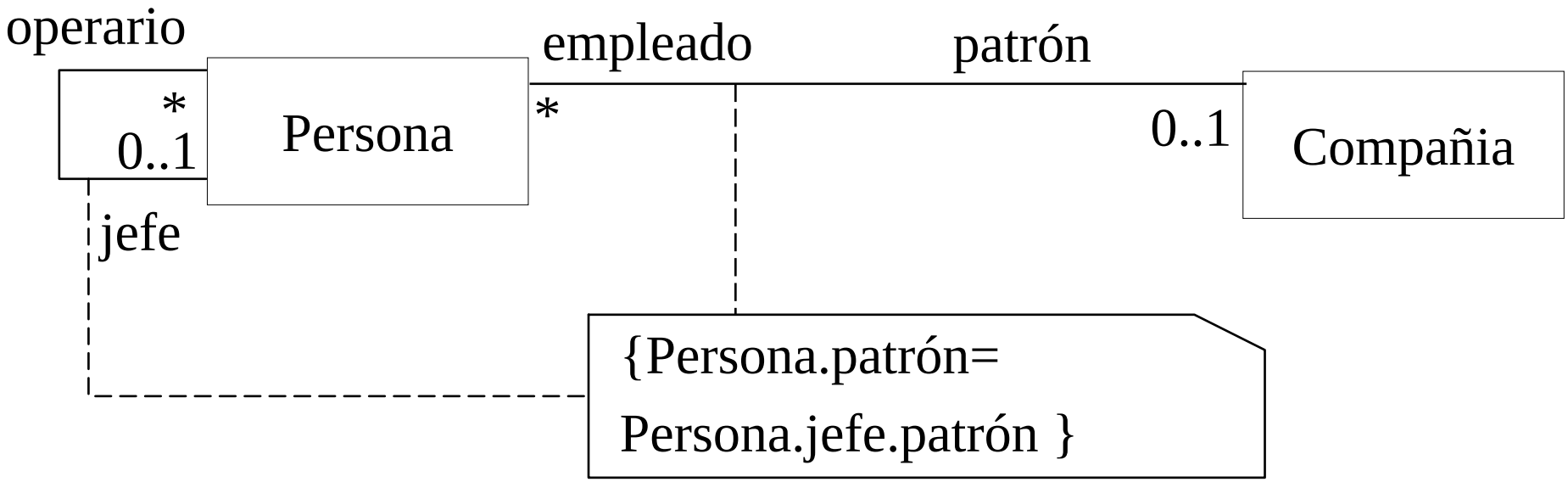




Restricciones para Asociaciones

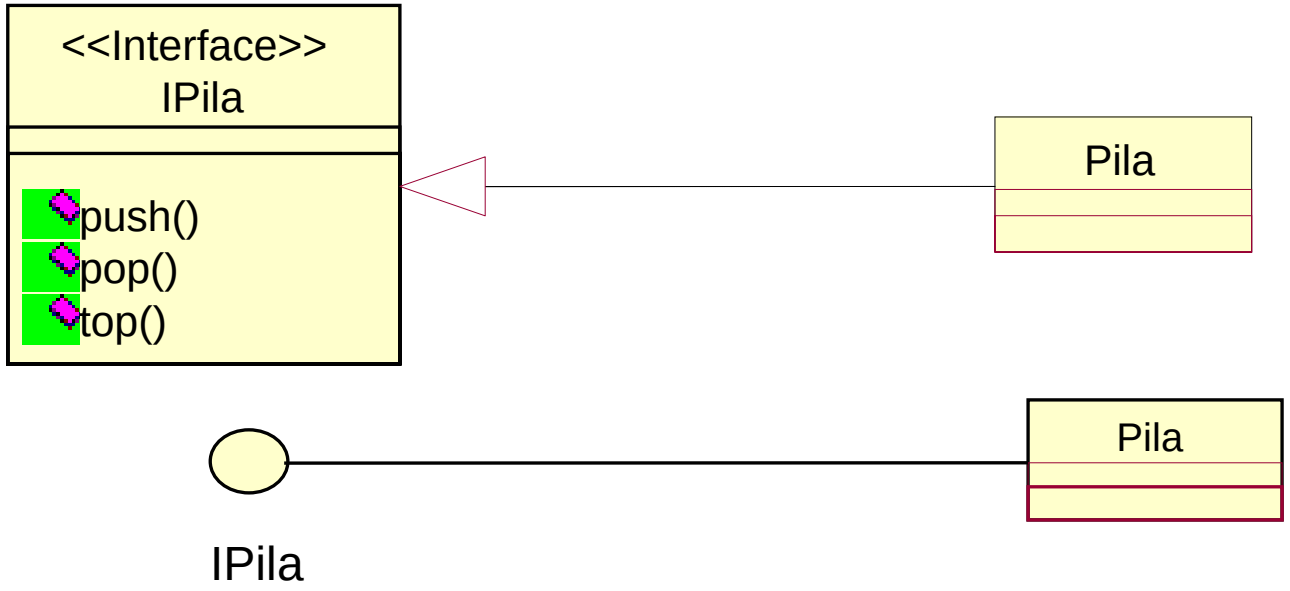


Restricciones para Asociaciones

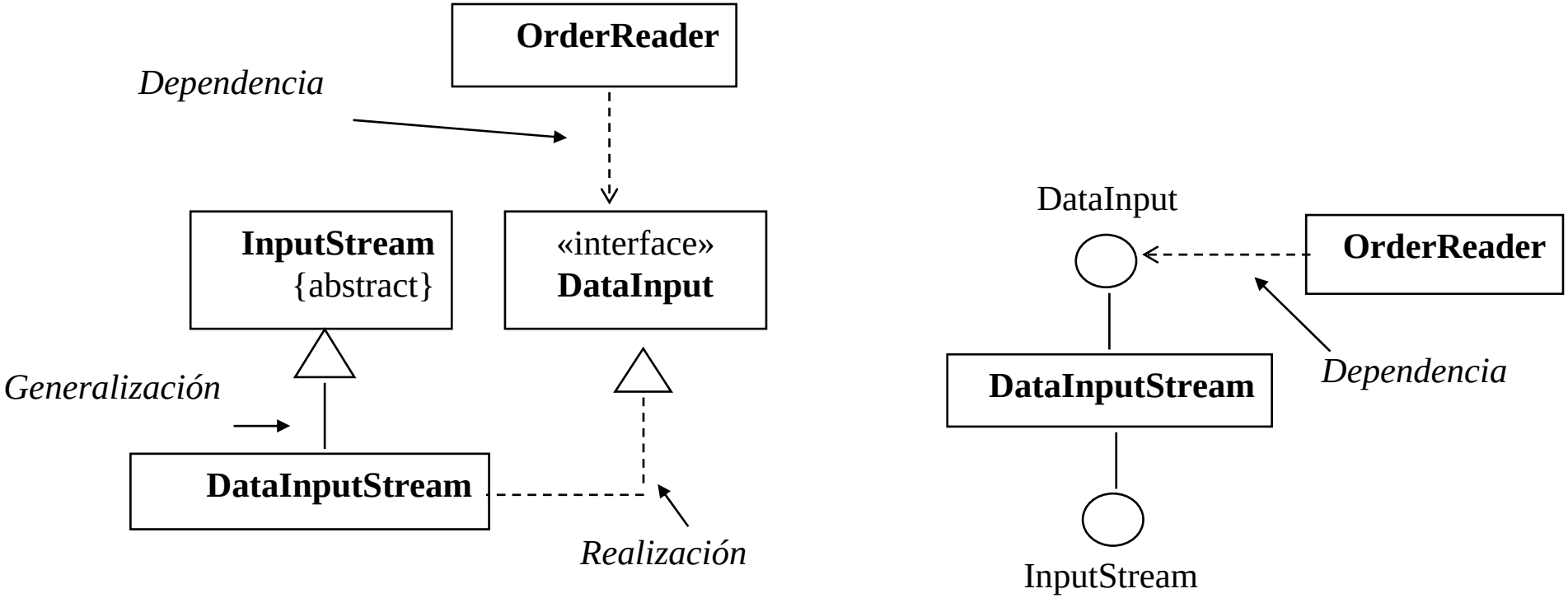


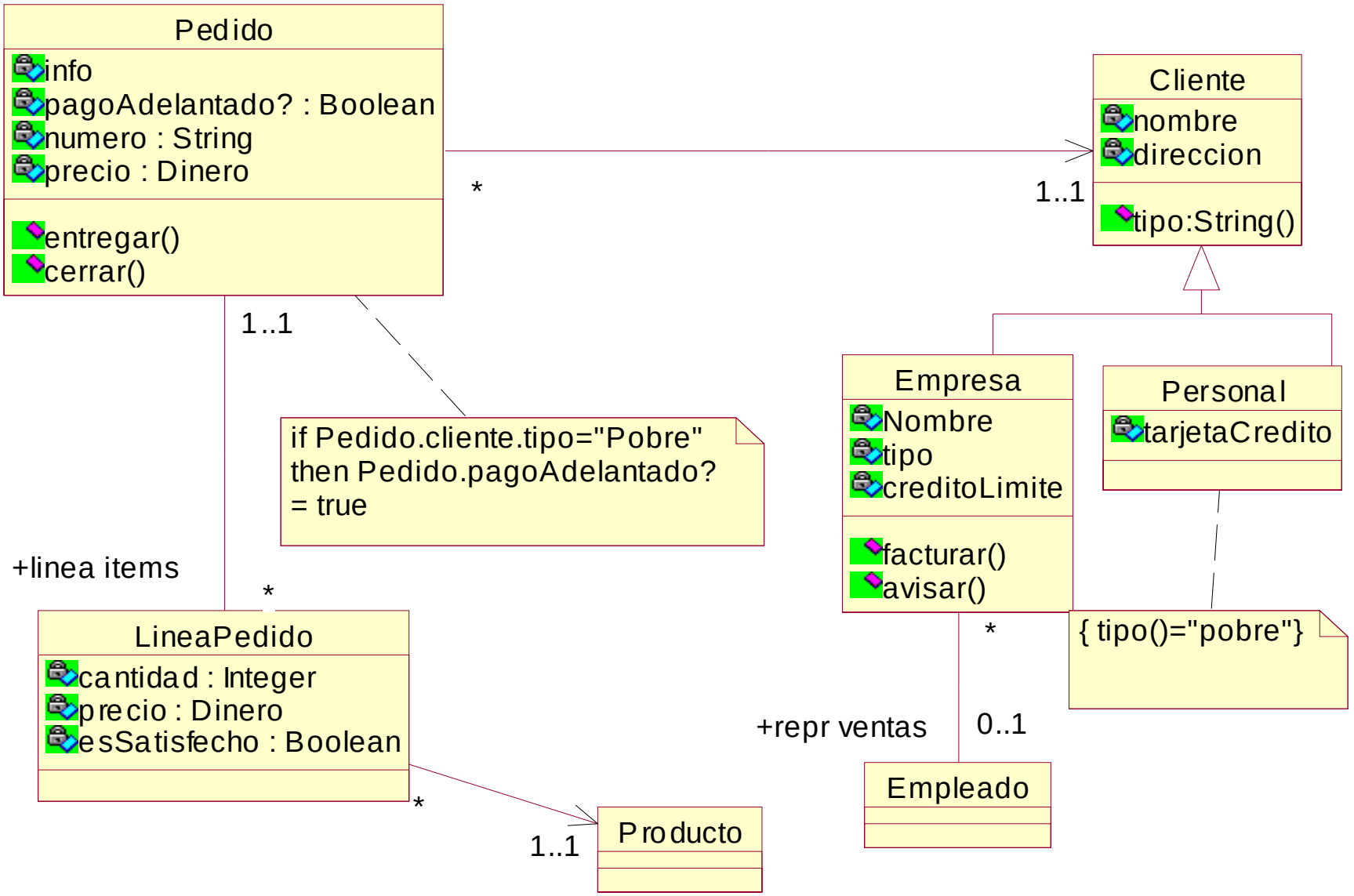
Diagramas de Clase: Realización

Relación entre clasificadores, un clasificador especifica un contrato que otro clasificador garantiza que cumplirá.



Clases Abstractas e Interfaces




Pedido
info
pagoAdelantado? : Boolean
numero : String
precio : Dinero
entregar()
cerrar()
*
1..1
Cliente
nombre
direccion
tipo:String()
1..1
if Pedido.cliente.tipo="Pobre"
then Pedido.pagoAdelantado?
= true
Empresa
Nombre
tipo
creditoLimite
facturar()
avisar()
Personal
tarjetaCredito
{ tipo()="pobre"}
+linea items
*
LineaPedido
cantidad : Integer
precio : Dinero
esSatisfecho : Boolean
*
1..1
Producto
*
+repr ventas
0..1
Empleado