



Tema 3. Modelado de Estructura del Sistema

Modelado del Software

Raquel Martínez España

Escuela Politécnica



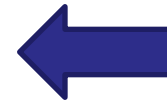
Contenidos

- Elementos estructurales
- Diagrama de clases
- Diagrama de objetos



Contenidos

- **Elementos estructurales**
- Diagrama de clases
- Diagrama de objetos

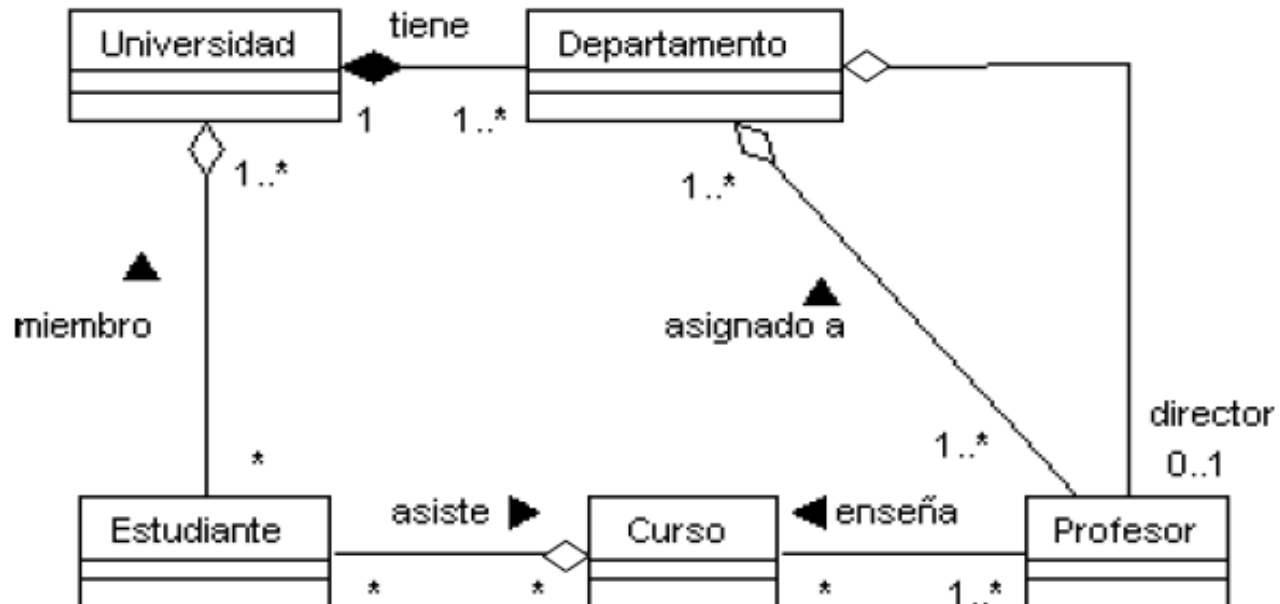


Elementos estructurales

- El **modelado de estructura** requiere el uso de **clases**, **interfaces** y **colaboraciones**, así como las **relaciones** entre ellos.
- El uso del modelado de estructura es común durante el **análisis** y el **diseño**.
 - Explorar conceptos del dominio
 - Analizar requisitos
 - Describir el diseño detallado de un software OO
- El modelado de estructura es la base para otros diagramas:
 - Objetos
 - Componentes y Despliegue

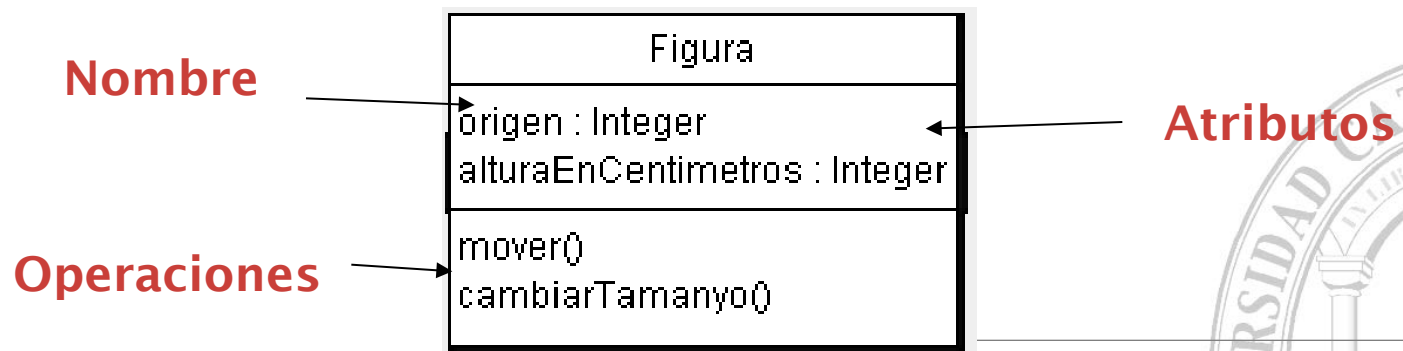
Ejemplo

- Pueden contener:
 - Clases, Interfaces, Relaciones, Notas, Restricciones, Paquetes



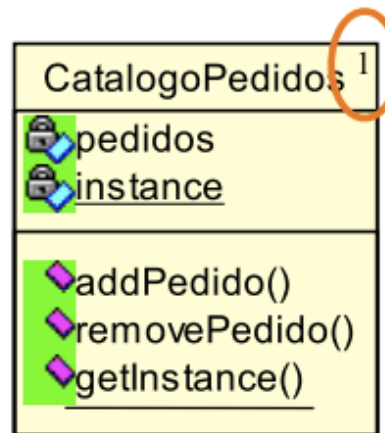
Clases (i)

- Sirven para identificar las “**cosas**” importantes desde una visión particular
 - Ejemplo: software, hardware, conceptos, etc.
- Capturan el vocabulario del sistema que se modela
- Características básicas:



Clases (ii)

- A veces se desea restringir el número de instancias (**multiplicidad** de una clase)
- Indicar la multiplicidad en la parte superior derecha
 - 1 → Clase unitaria
 - K → Pueden existir múltiples instancias



Especificación de atributos

- Como mínimo, indicar el nombre del atributo
- Se puede completar con otra información:
 - **Visibilidad, Tipo, Multiplicidad, Valor inicial, Modificador**
- Ejemplos de atributos:

origen *nombre*

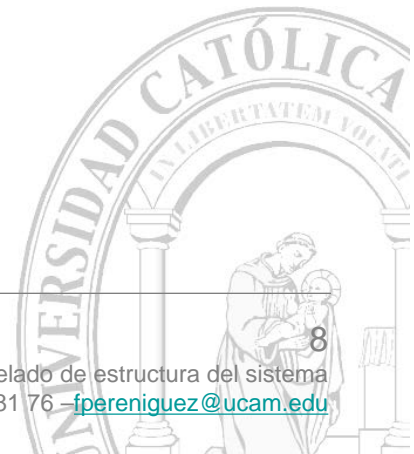
+ origen *visibilidad y nombre*

origen:Punto *nombre y tipo*

nombre: String[0..1] *nombre, tipo, multiplicidad*

origen: Punto = (0,0) *nombre, tipo, valor inicial*

id: Integer {readOnly} *nombre, tipo y modificador*



Visibilidad de atributos (i)

- Sirven para ocultar detalles y mostrar sólo ciertas características de una clase
- **Principio de ocultación de información**
- Niveles de visibilidad:
 - **public (+)**: características visible externamente
 - **protected (#)**: sólo los descendientes pueden usarla
 - **private (-)**: sólo puede ser usada por la clase
 - **package (~)**: sólo las clases declaradas en el mismo paquete pueden usarla



Visibilidad de atributos (ii)

- Ejemplo:

BarraHerramientas
selecciónActual:Herramienta # contadorHerramienta:Integer
+ elegirElemento(i:Integer) + añadirHerramienta(t:Herramienta) + quitarHerramienta(i:Integer) + obtenerHerramienta(): Herramienta # comprobarHuerfanos() - compactar() ~ reconfigurar()

Alcance de atributos

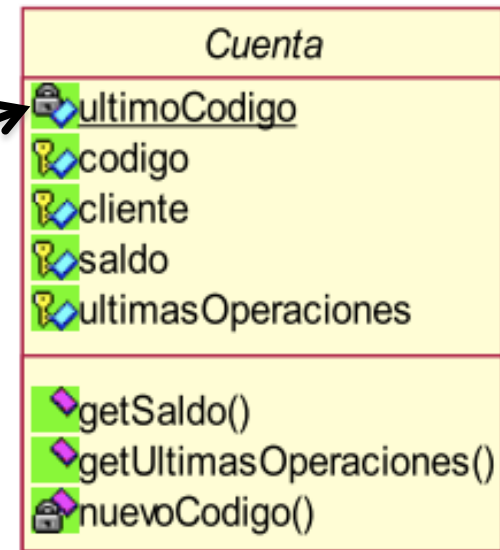
- Una característica de una clase (atributo u operación) puede ser estática o dinámica

- **Estática:**

- Hay **un único valor o instancia** para todos los objetos
- Se muestra subrayando el nombre

- **Dinámica:**

- Cada instancia del clasificador tiene su propio valor o instancia para la características
- **Opción por defecto**



Relación entre código y UML

```
public class ComplexNumber {  
  
    private double r;  
    private double i;  
  
    public ComplexNumber(double r, double i) {  
        this.r = r;  
        this.i = i;  
    }  
  
    public double norm() {  
        return Math.sqrt(r * r + i * i);  
    }  
}
```

El código es
Java ;-)

ComplexNumber
-r : double -i : double
+ComplexNumber(r : double, i : double) +norm() : double

Multiplicidad de atributos

- Indica el número de valores simultáneos que pueden existir para cada instancia de la clase
 - Opción **por defecto: un solo valor**
 - En otro caso, indicar entre corchetes [] el número mínimo y máximo
- Ejemplos:
 - [0...1]
 - [1...3]
 - [1..*]
 - [*]
- Cuando es multivaluado, se pueden incluir restricciones de orden (*ordered*) y unicidad (*unique*)

Especificación de operaciones de clases (i)

- Invocar un **comportamiento** concreto **de una clase**
- Distinguir entre operación y método:
 - **Operación**: servicio abstracto
 - **Método**: implementación de una operación concreta
 - Pueden haber **varios métodos que implementen una operación (polimorfismo)**
- La especificación de una operación debe incluir al menos el nombre. Opcionalmente, se puede incluir:
 - **Visibilidad, nombre, parámetros, tipo devuelto, modificadores**

Especificación de operaciones (ii)

- Ejemplos de operaciones de una clase:

mostrar *nombre*

+ mostrar *visibilidad y nombre*

set (n:Nombre, s:String) *nombre y parámetros*

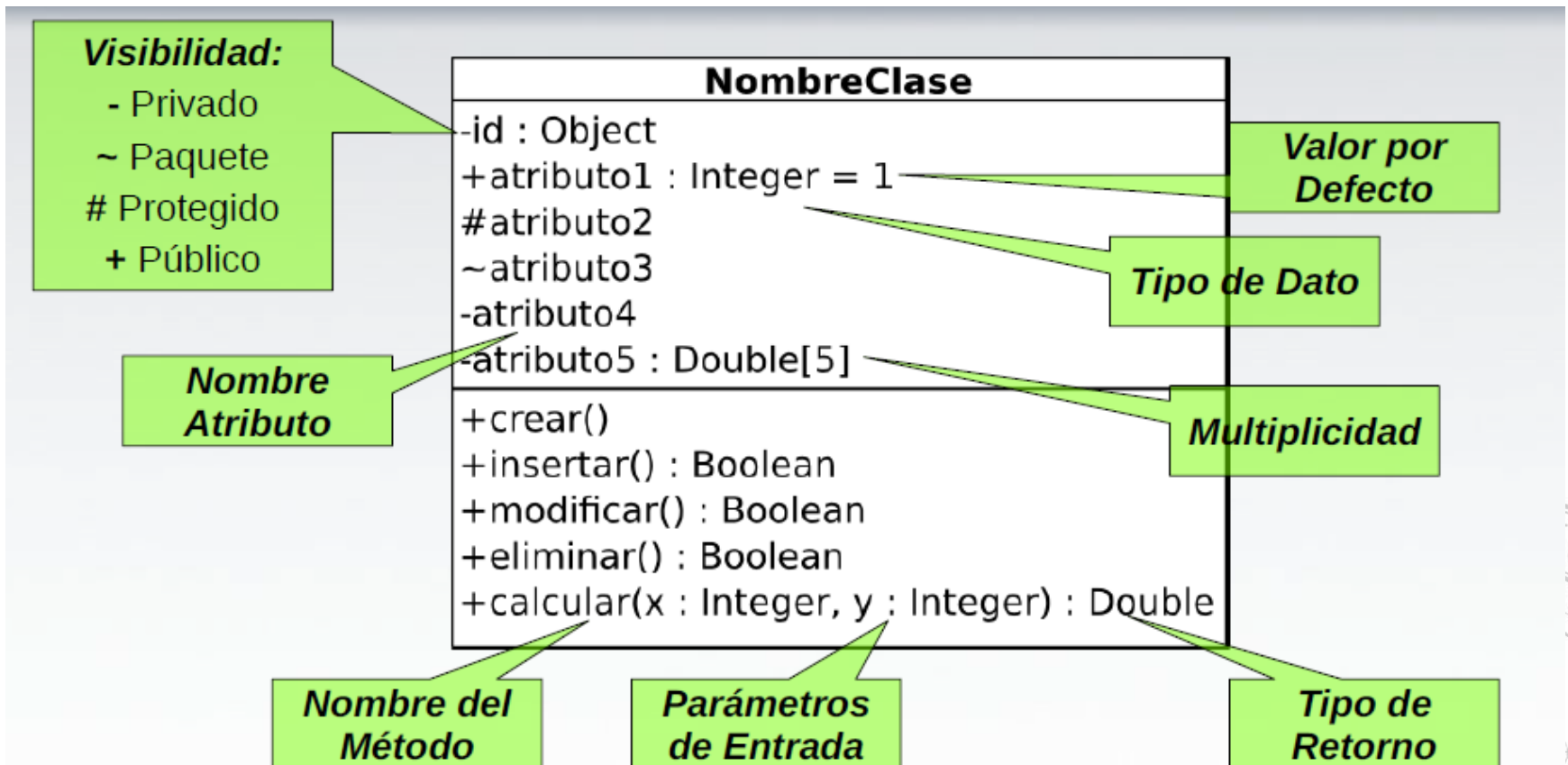
obtenerID(): Integer *nombre y tipo de retorno*

saldo() {ordered} *nombre y modificador*



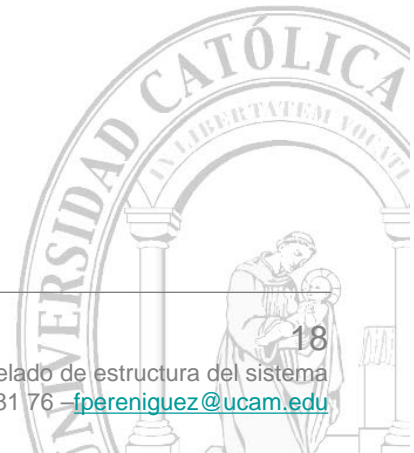
Visibilidad de atributos (ii)

- Ejemplo:



Recomendación de uso

- Las **clases**:
 - La especificación inicial de una clase debe ser básica.
 - Conforme avanza el diseño, se puede refinar la definición de las clases.
 - Elementos avanzados:
 - Visibilidad de atributos y operaciones
 - Alcance de atributos
 - Multiplicidad de clases y atributos
 - Valor inicial y modificabilidad de atributos
 - Tipo de las operaciones

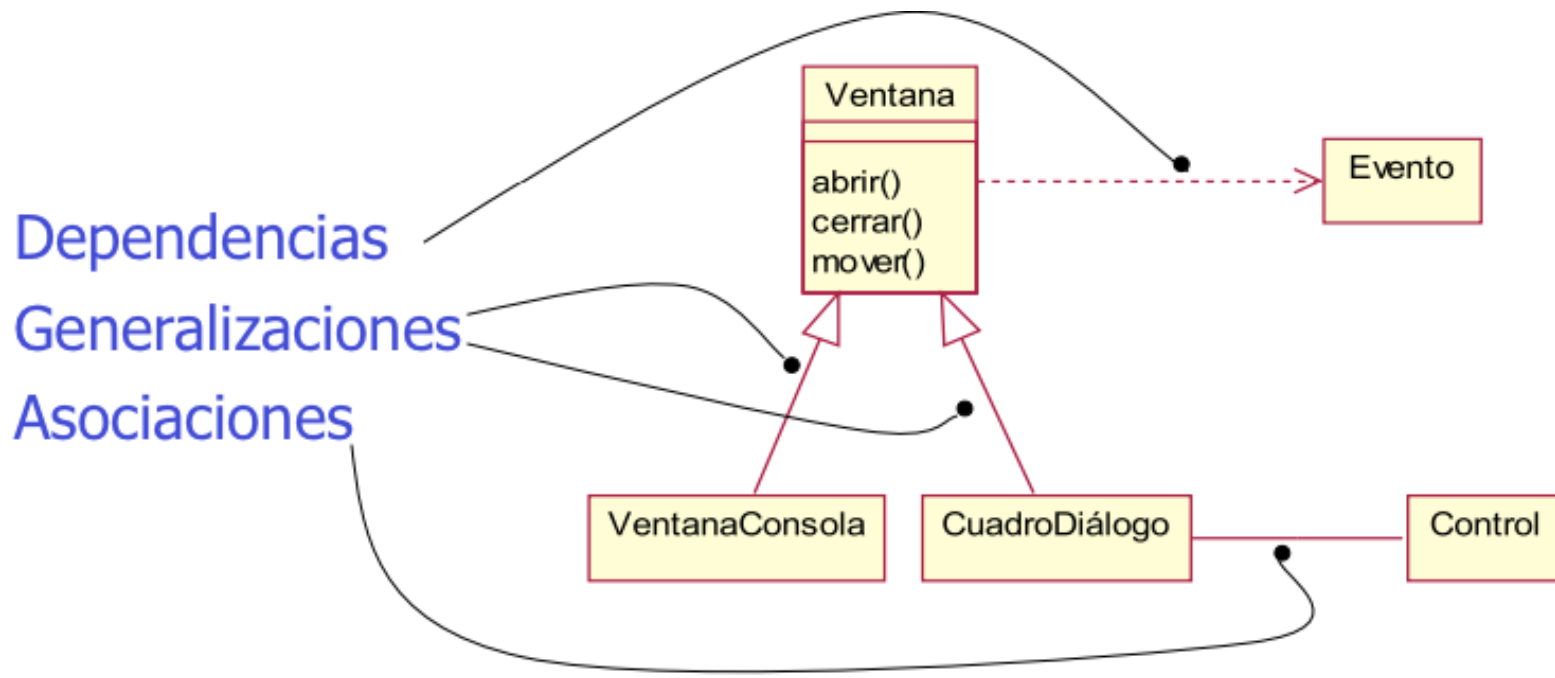


Relaciones entre clases

- Las clases suelen estar relacionadas => **Relaciones**
- Tipos de relaciones vistas:
 - Dependencias: una clase usa otra como parámetro de una operación
 - Generalizaciones: conectan clases generales con sus implementaciones. Denota una relación de **herencia**.
 - Asociaciones: relaciones estructurales entre objetos.
 - Adornos básicos: **nombre, rol, multiplicidad, agregación**
 - Tipos: **Agregación, Composición**
 - Realización: implementación de una interfaz

Ejemplo uso de relaciones

- Uso de los distintos tipos de relaciones:



Relaciones asociación

- **Multiplicidad**

- Define cuantas instancias de una clase A pueden asociarse con una instancia de la clase B.
- Denota una **restricción del dominio**.

- **Navegación**

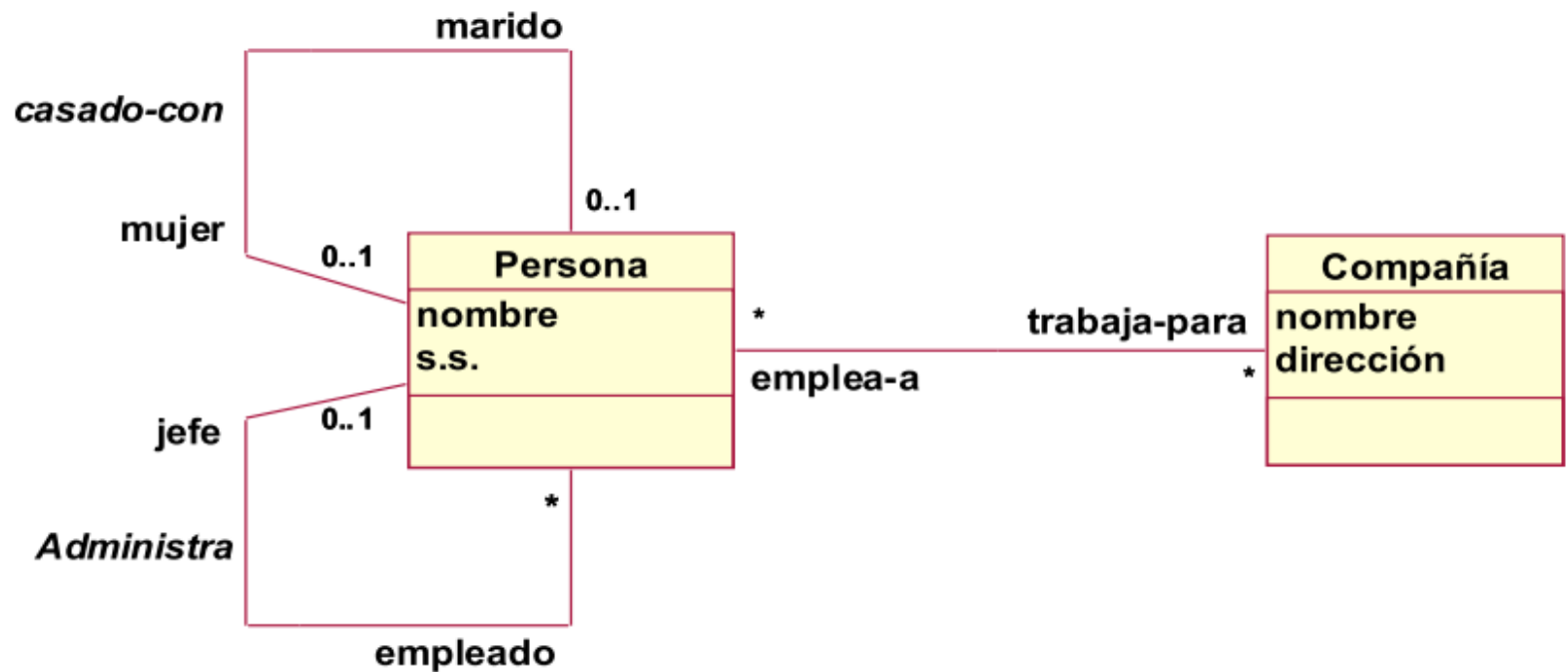
- Por defecto, las asociaciones son bidireccionales
- Si no queremos que sea así, usar flecha

- **Visibilidad**

- Permite controlar si los objetos de una clase pueden ver los de otra.
- Tipos: Pública (+), Privada (-), Protegida (#)

Uso de relaciones de asociación y adornos

- Ejemplo:



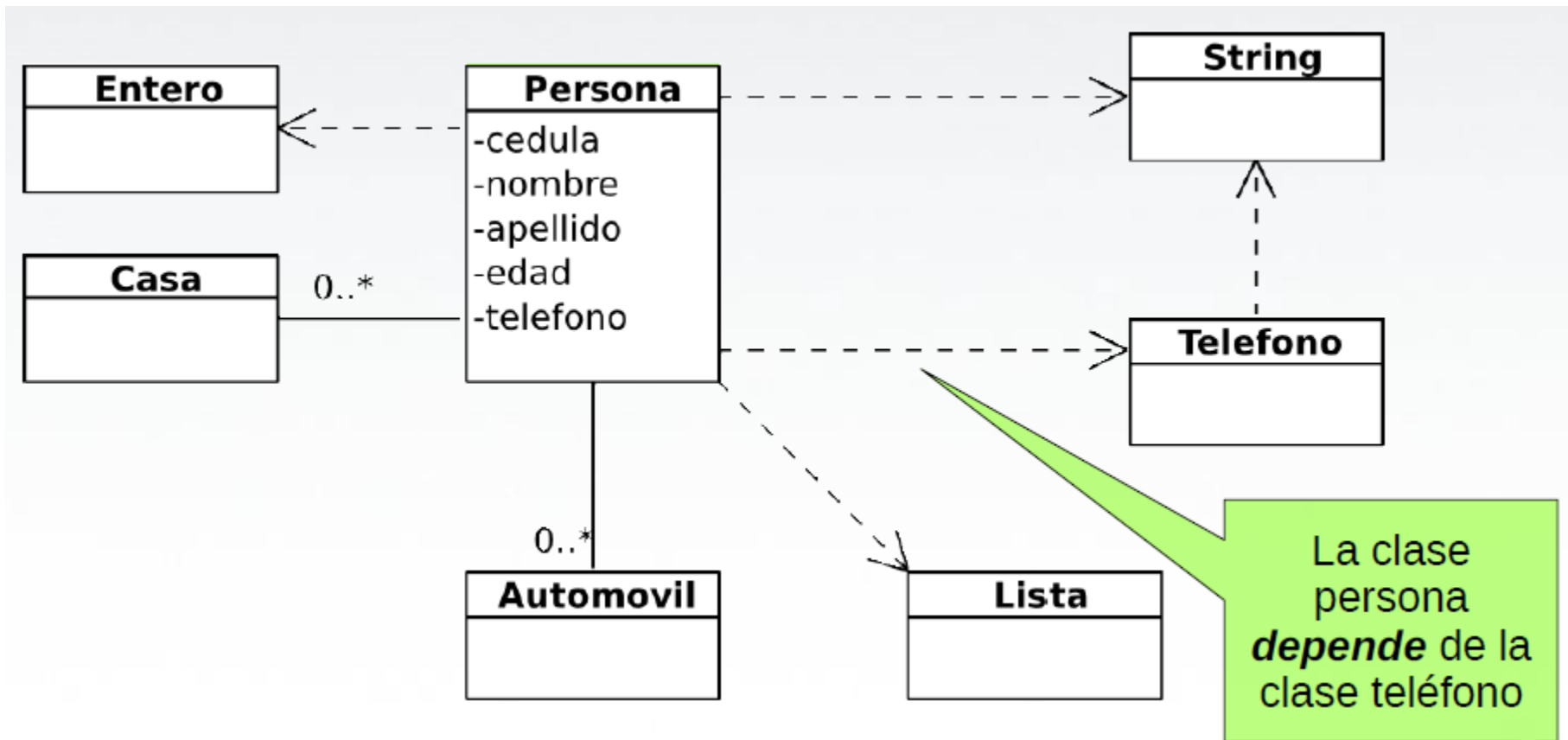
Visibilidad en las asociaciones

- Importante conocer el uso de visibilidad en las asociaciones:

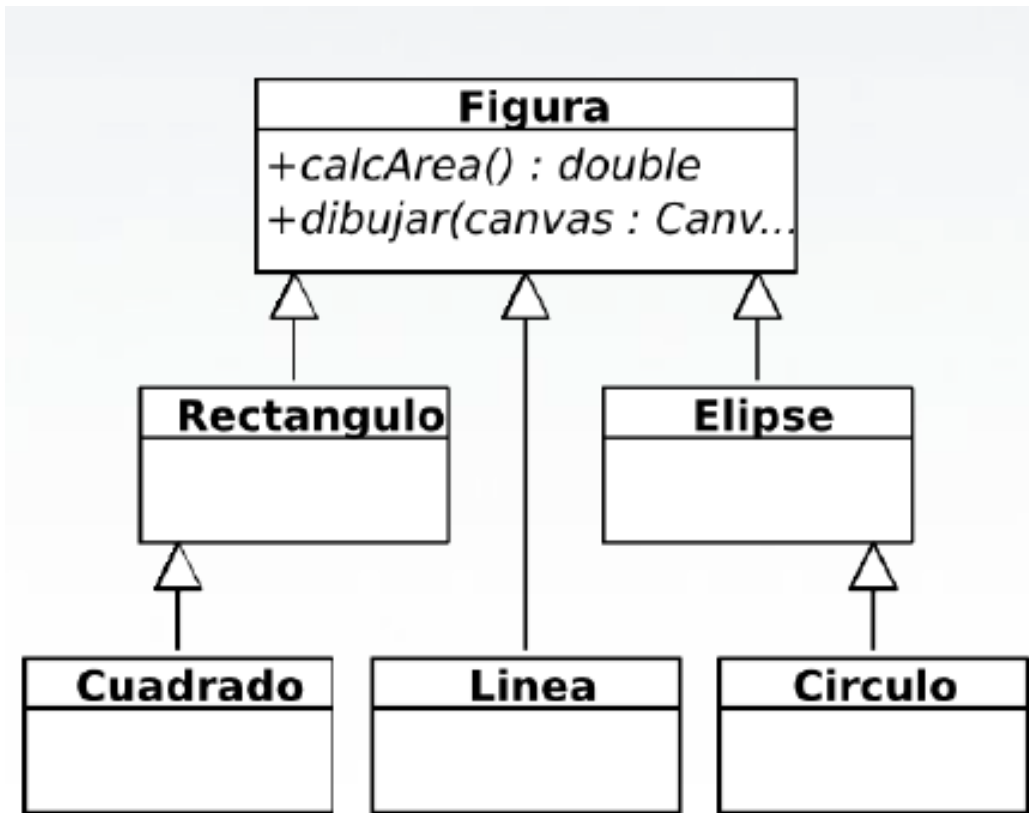


- Dado un objeto “Clave”, es posible conocer su propietario
- Un clave es privada a un objeto “Usuario”
 - Por ejemplo, un objeto “GrupoUsuarios” puede ver sus objetos “Usuario”, pero no acceder a las claves de cada uno.

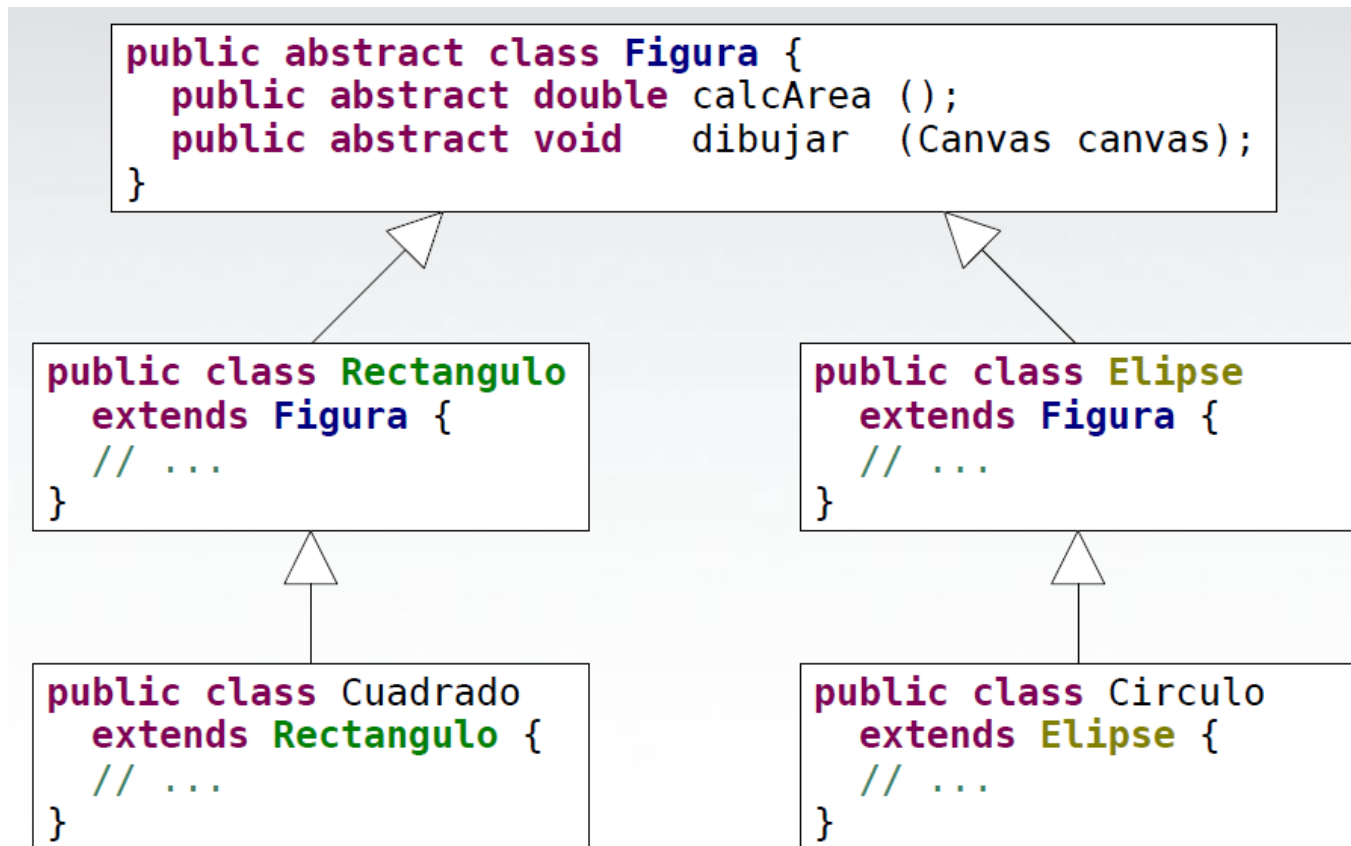
Ejemplo Detallado: Implementación (Dependencias)



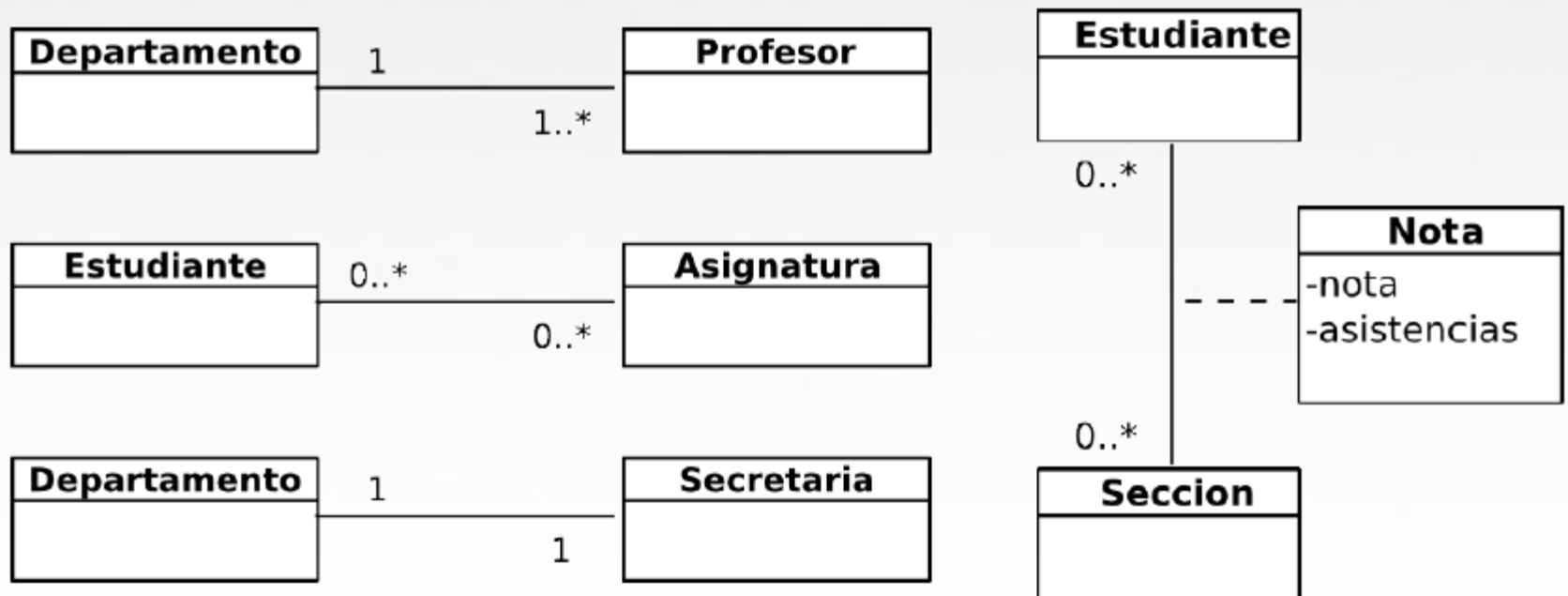
Ejemplo Detallado: Implementación (Herencias)



Ejemplo Detallado: Implementación (Herencias)

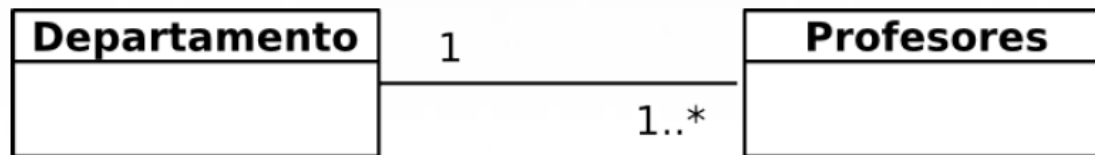
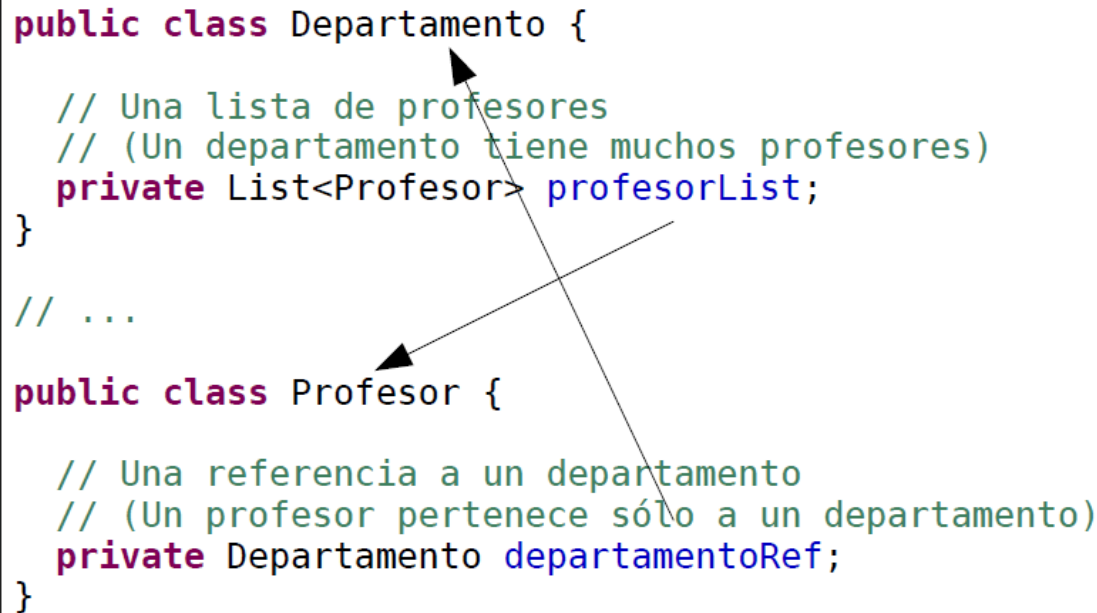


Ejemplo Detallado (Asociaciones)



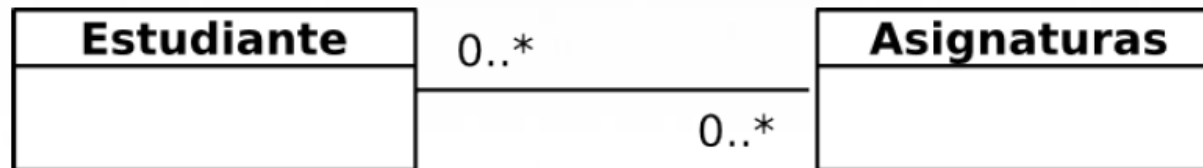
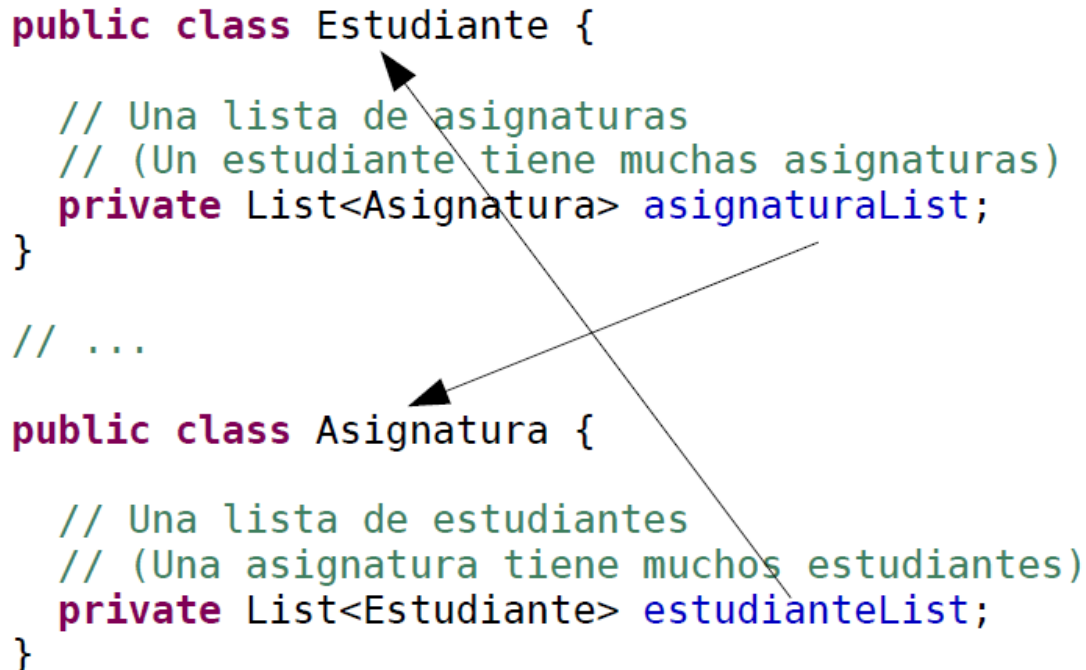
Ejemplo Detallado: Implementación (Asociaciones)

```
public class Departamento {  
    // Una lista de profesores  
    // (Un departamento tiene muchos profesores)  
    private List<Profesor> profesorList;  
}  
  
// ...  
  
public class Profesor {  
    // Una referencia a un departamento  
    // (Un profesor pertenece sólo a un departamento)  
    private Departamento departamentoRef;  
}
```

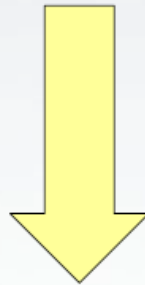
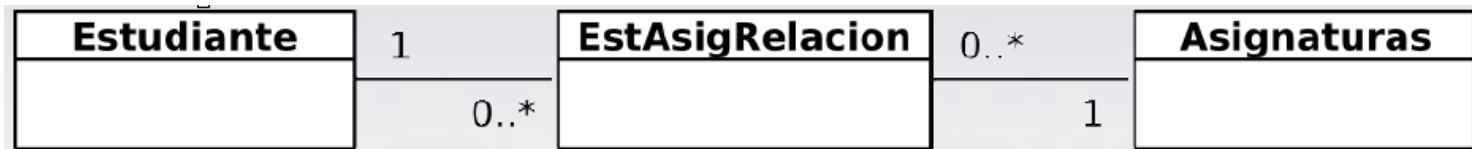


Ejemplo Detallado: Implementación (Asociaciones)

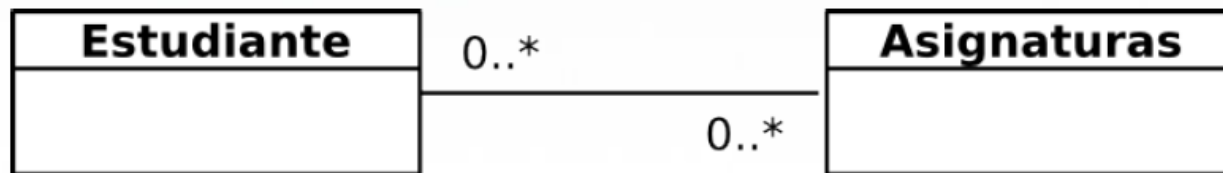
```
public class Estudiante {  
    // Una lista de asignaturas  
    // (Un estudiante tiene muchas asignaturas)  
    private List<Asignatura> asignaturaList;  
}  
  
// ...  
  
public class Asignatura {  
    // Una lista de estudiantes  
    // (Una asignatura tiene muchos estudiantes)  
    private List<Estudiante> estudianteList;  
}
```



Ejemplo Detallado: Implementación (Asociaciones)

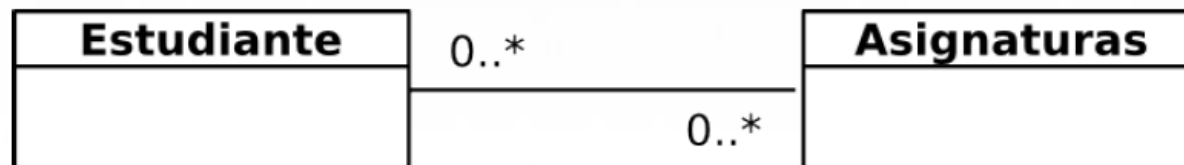
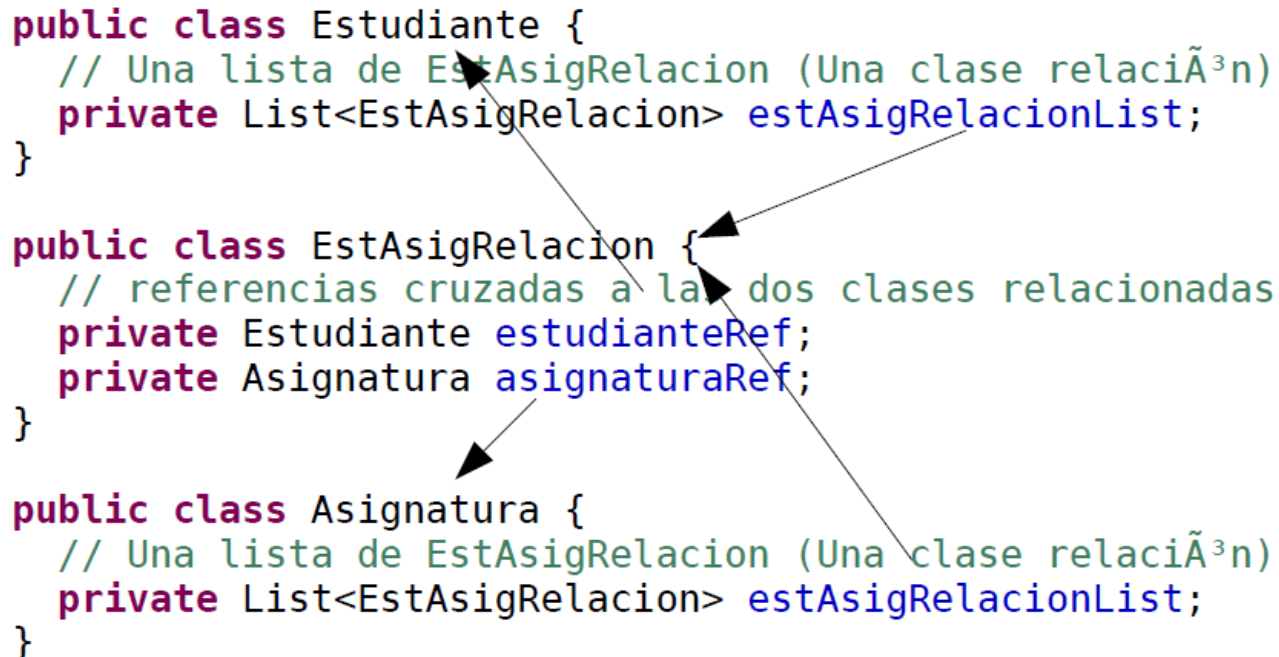


Una relación
muchos a muchos
se puede ver como
dos relaciones uno a
muchos



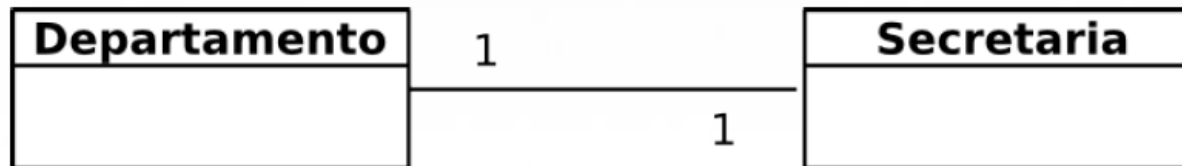
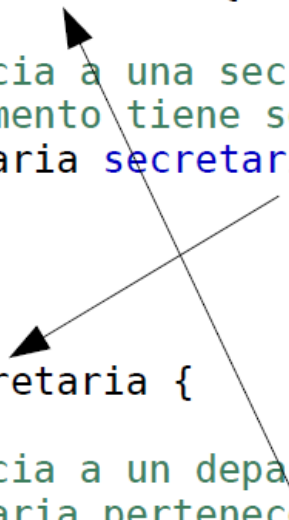
Ejemplo Detallado: Implementación (Asociaciones)

```
public class Estudiante {  
    // Una lista de EstAsigRelacion (Una clase relaciÃ³n)  
    private List<EstAsigRelacion> estAsigRelacionList;  
}  
  
public class EstAsigRelacion {  
    // referencias cruzadas a las dos clases relacionadas  
    private Estudiante estudianteRef;  
    private Asignatura asignaturaRef;  
}  
  
public class Asignatura {  
    // Una lista de EstAsigRelacion (Una clase relaciÃ³n)  
    private List<EstAsigRelacion> estAsigRelacionList;  
}
```



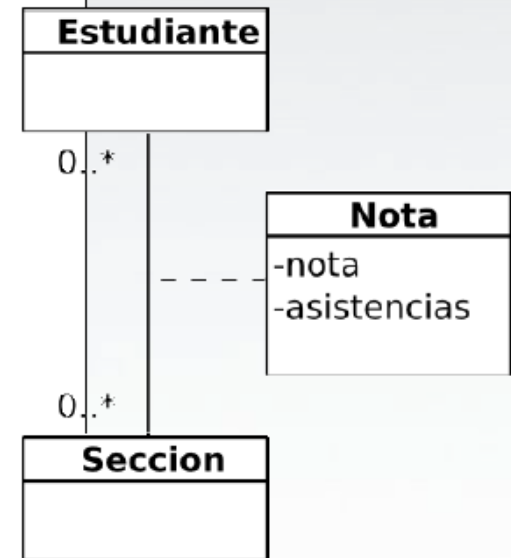
Ejemplo Detallado: Implementación (Asociaciones)

```
public class Departamento {  
    // Una referencia a una secretaria  
    // (Un departamento tiene sólo una secretaria)  
    private Secretaria secretariaRef;  
}  
  
// ...  
  
public class Secretaria {  
    // Una referencia a un departamento  
    // (Una secretaria pertenece sólo a un departamento)  
    private Departamento departamentoRef;  
}
```

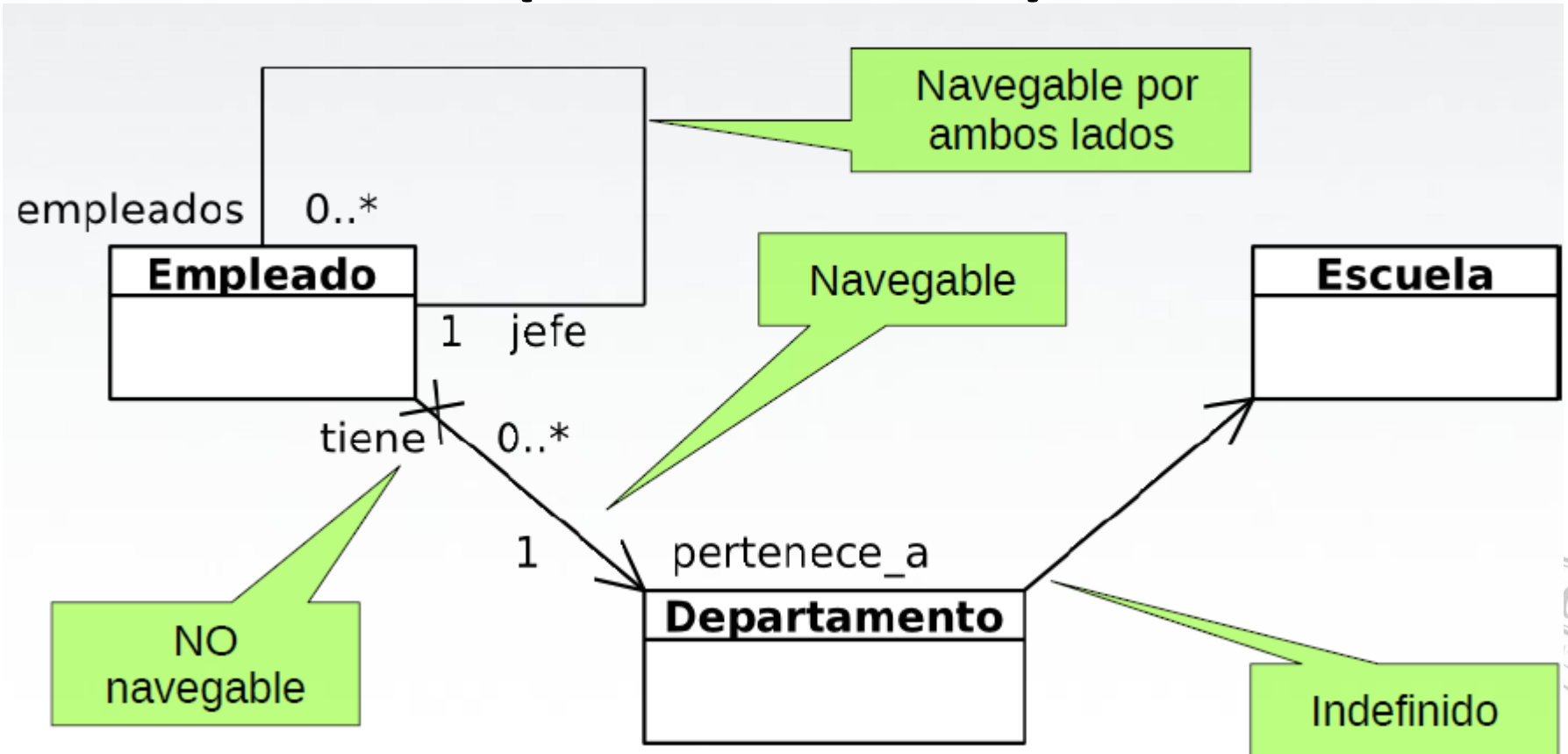


Ejemplo Detallado: Implementación (Asociaciones)

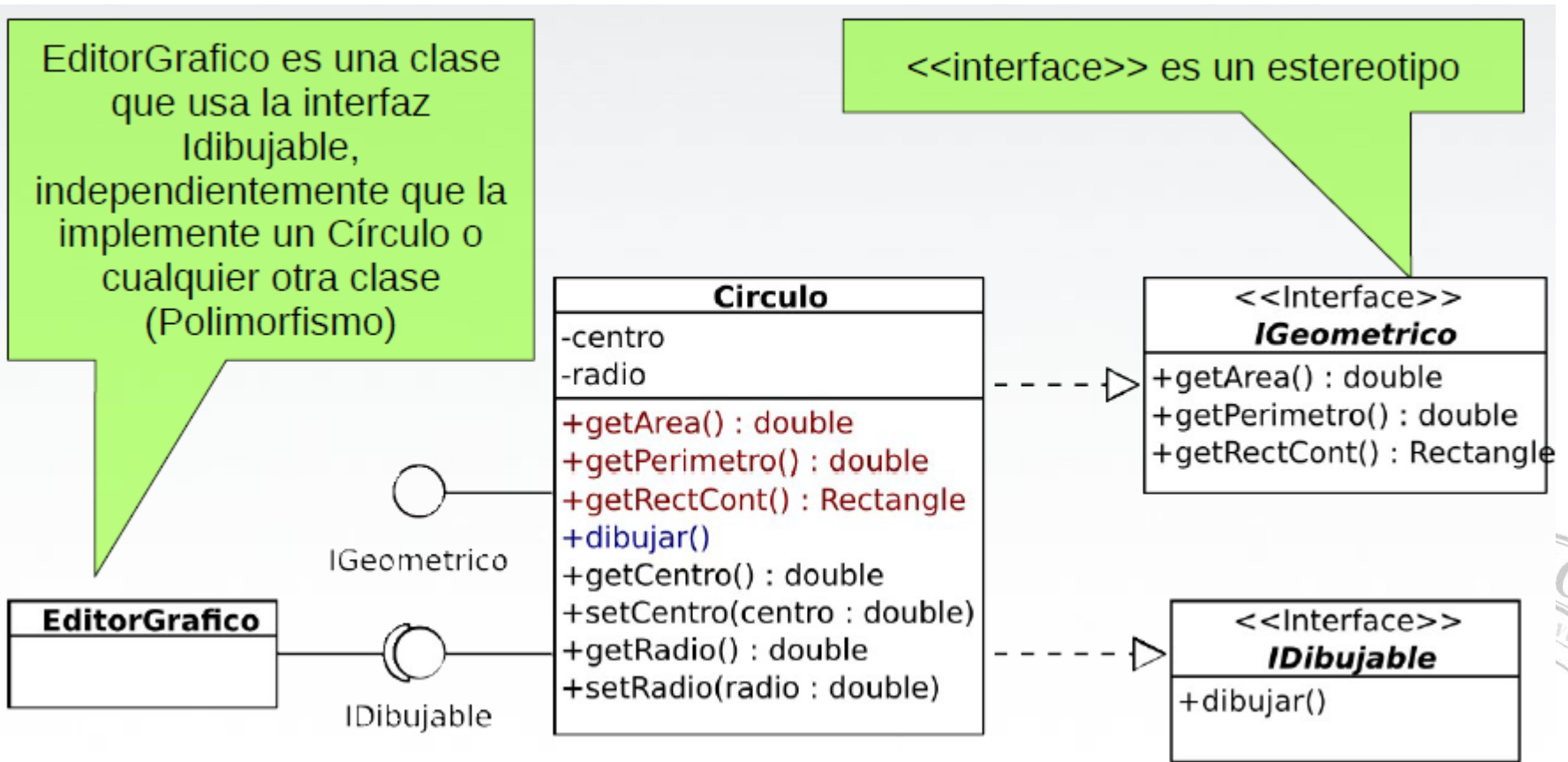
```
public class Estudiante {  
    // Una lista de Nota (Una clase asociación)  
    private List<Nota> notaList;  
}  
  
public class Nota {  
  
    // Datos de la asociación  
    private double nota;  
    private int asistencias  
  
    // referencias cruzadas a  
    // las dos clases relacionadas  
    private Estudiante estudianteRef;  
    private Seccion seccionRef;  
}  
  
public class Seccion {  
    // Una lista de Nota (Una clase asociación)  
    private List<Nota> notaList;  
}
```



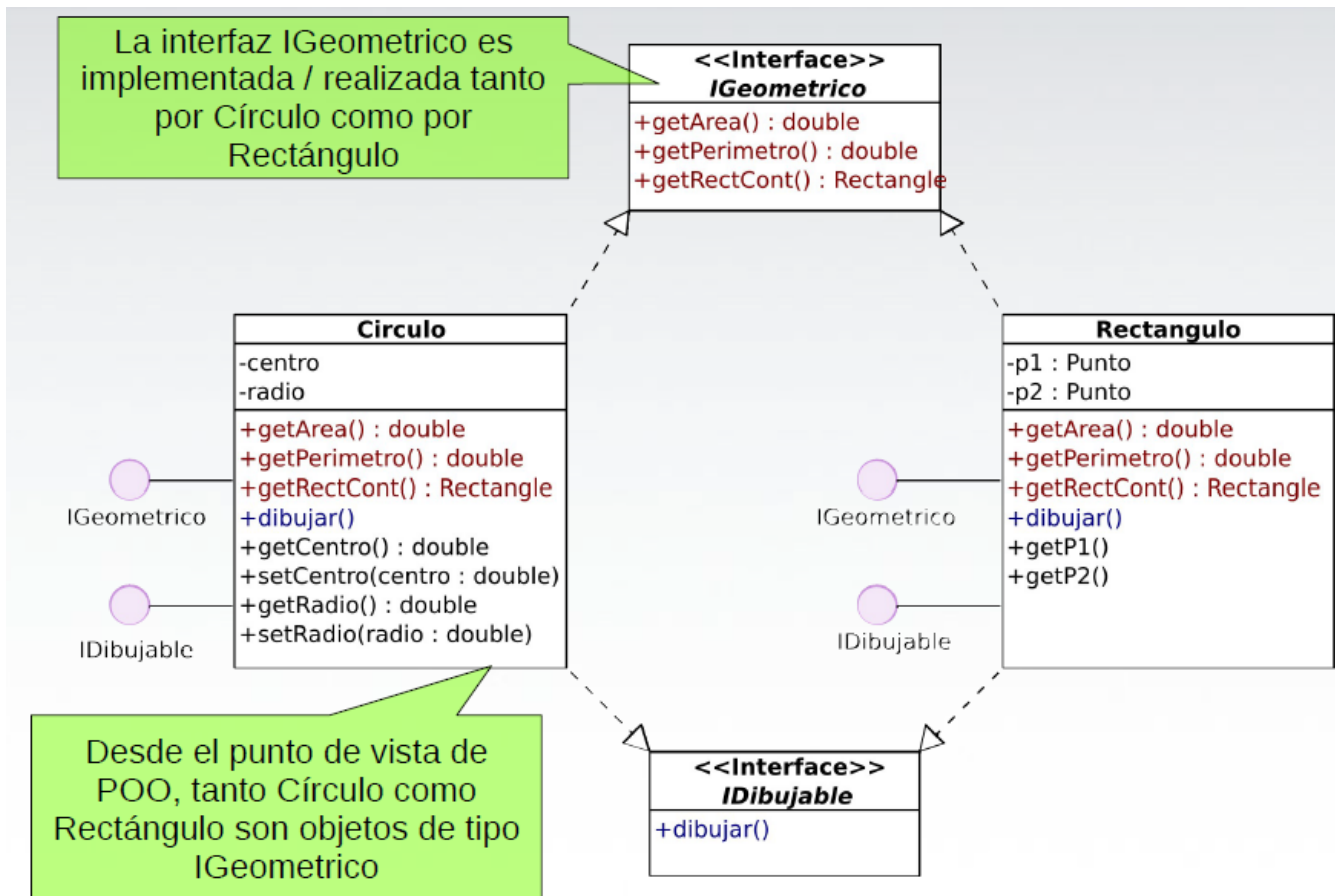
Ejemplo Detallado: Implementación (Asociaciones)



Ejemplo Detallado: Implementación (Interfaces/ Realizaciones)



Ejemplo Detallado: Implementación (Interfaces/ Realizaciones)



Ejemplo Detallado: Implementación (Interfaces/ Realizaciones)

```
import java.awt.Point;
import java.awt.Rectangle;

public class Circulo implements IGeometrico, IDibujable {

    private double centro;
    private double radio;

    public double    getArea()          { /* de IGeometrico */ }
    public double    getPerimetro()     { /* de IGeometrico */ }
    public Rectangle getRectCont()      { /* de IGeometrico */ }

    public void dibujar()                { /* de IDibujable */ }

    public Point  getCentro()           { /* de circulo */ }
    public void  setCentro(...)         { /* de circulo */ }

    public double getRadio()            { /* de circulo */ }
    public void  setRadio(...)          { /* de circulo */ }
}
```

Ejemplo Detallado: Implementación (Interfaces/ Realizaciones)

```
import java.awt.Rectangle;

public interface IGeometrico {

    public double getArea();

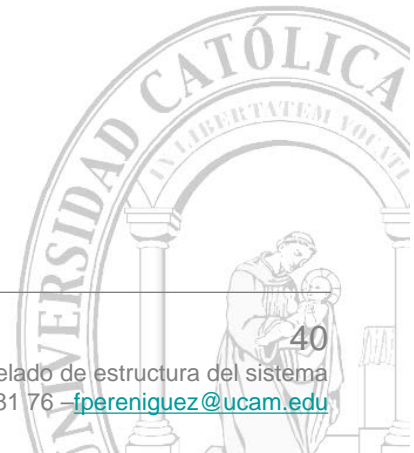
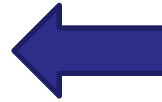
    public double getPerimetro();

    public Rectangle getRectCont();
}

public interface IDibujable {
    public void dibujar();
}
```

Contenidos

- Elementos estructurales
- **Diagrama de clases**
- Diagrama de objetos



Identificar clases (i)

- ¿Cómo encontrar clases?
 - Los **nombres** son candidatos a clases (no a objetos)
- Ejemplo:

“El ascensor cerrará sus puertas antes de subir a la siguiente planta”

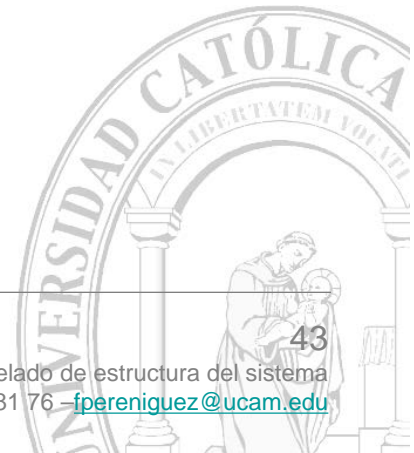
- Posibles clases: **Ascensor, Puerta, Planta**

Identificar clases (ii)

- Evitar clases inútiles
 - ¿Es necesaria la clase “*Puerta*”?
- Saber detectar la necesidad de nuevas clases
 - La noción “*Planta*”, ¿requiere que sea modelada con una clase o vale con usar el tipo *Integer*?
- Incluir la funcionalidad necesaria
 - El ascensor debe soportar las operaciones *subir* y *bajar*.

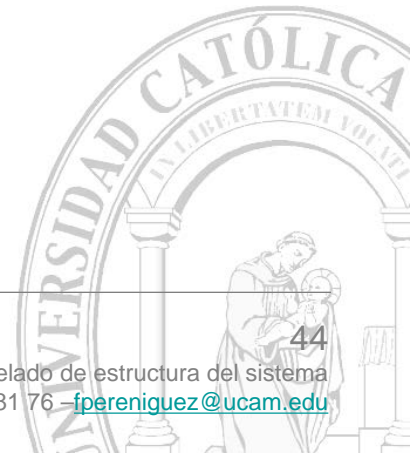
Recomendaciones (ii)

- Buenas prácticas en la identificación de clases:
 - El nombre de una clase es un **nombre** o un **adjetivo**
 - Describe un conjunto de objetos reales en tiempo de ejecución
 - Incluir métodos de consulta y modificación
 - Incluir propiedades abstractas: invariante, precondiciones, postcondiciones, etc.

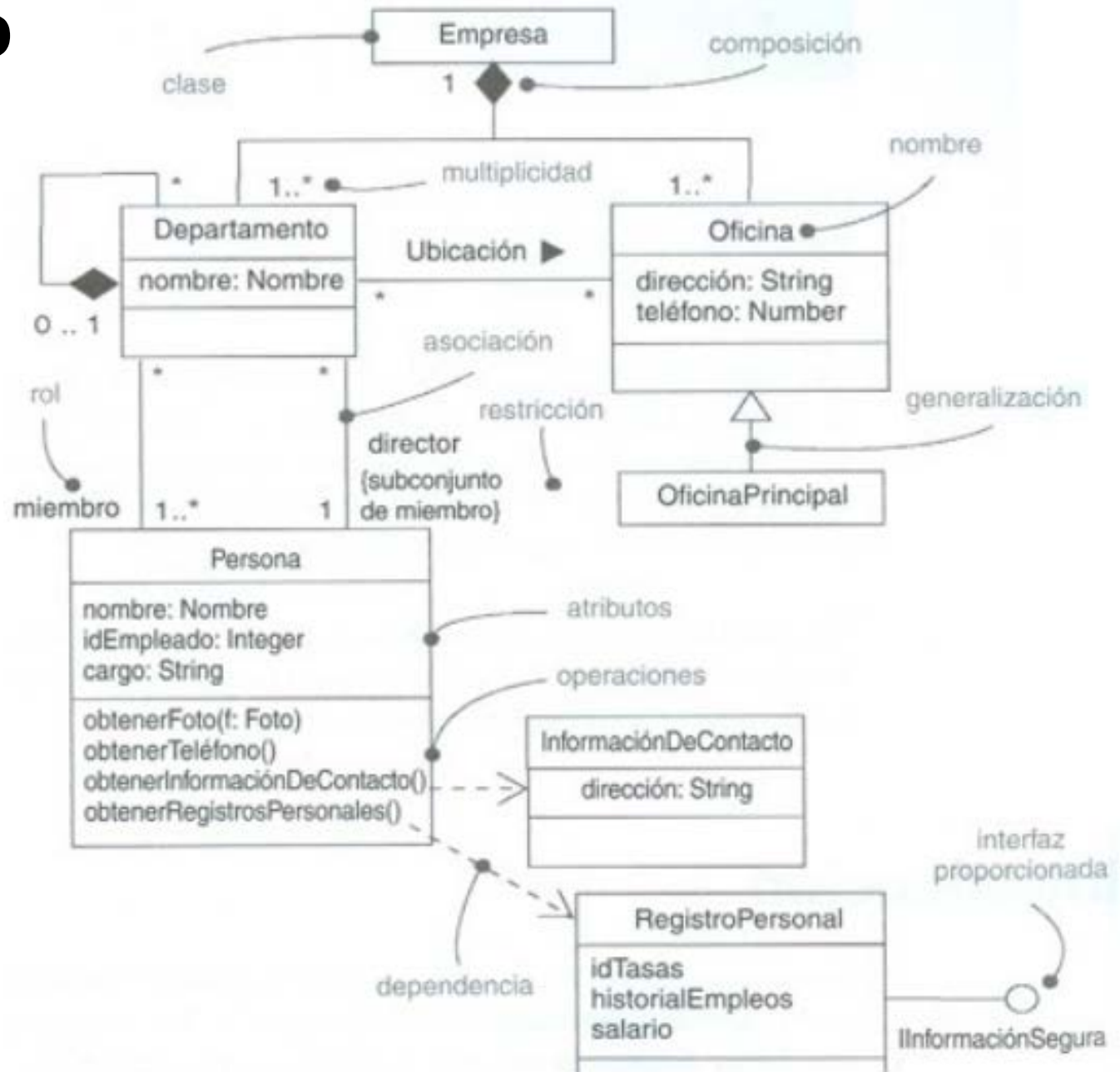


Recomendaciones (i)

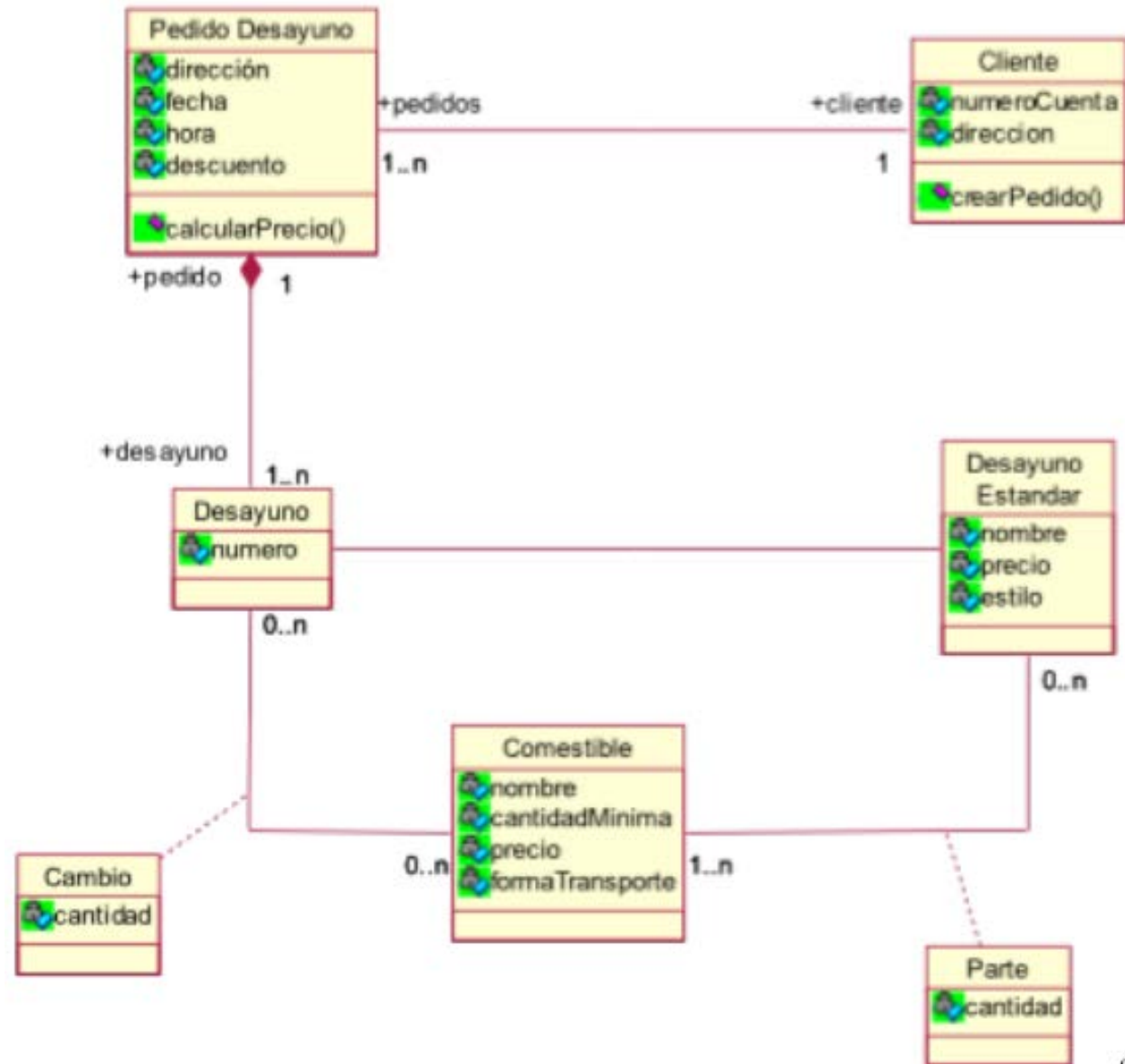
- **Malas prácticas** que se deben evitar durante la identificación de clases:
 - “*Mi clase hace...*”
 - Nombre imperativos (clase Analizar, Imprimir,...)
 - Un solo método público
 - Mezclar dos abstracciones en una misma clase
 - Clases sin métodos de modificación



Ejemplo




Ejemplo



Aspectos a tener en cuenta...

- Un diagrama de clases realizado durante el diseño tendrá más adornos que si se hace durante el análisis.
- Una clase debe corresponderse con una **abstracción del dominio** de la aplicación
- Se deben **mostrar las clases relacionadas** en un diagrama de clases
- Llevar **cuidado con el tipo de relación** que se elige para modelar una relación entre clases
- Incluir todas las **anotaciones** que sean necesarias.

Contenidos

- Elementos estructurales
- Diagrama de clases
- **Diagrama de objetos** 

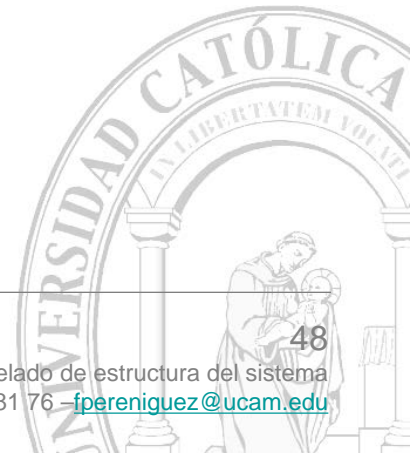
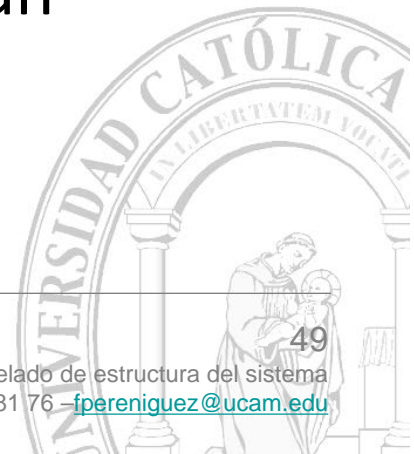


Diagrama de Objetos

- Muestra un conjunto de **objetos** y sus **relaciones**.
- Representan instantáneas estáticas de instancias de elementos recogidos en los diagramas de clases.
- Es como una **foto** del sistema en un **instante** dado.
- Describen la **vista de diseño estática** desde un punto de vista de casos reales.

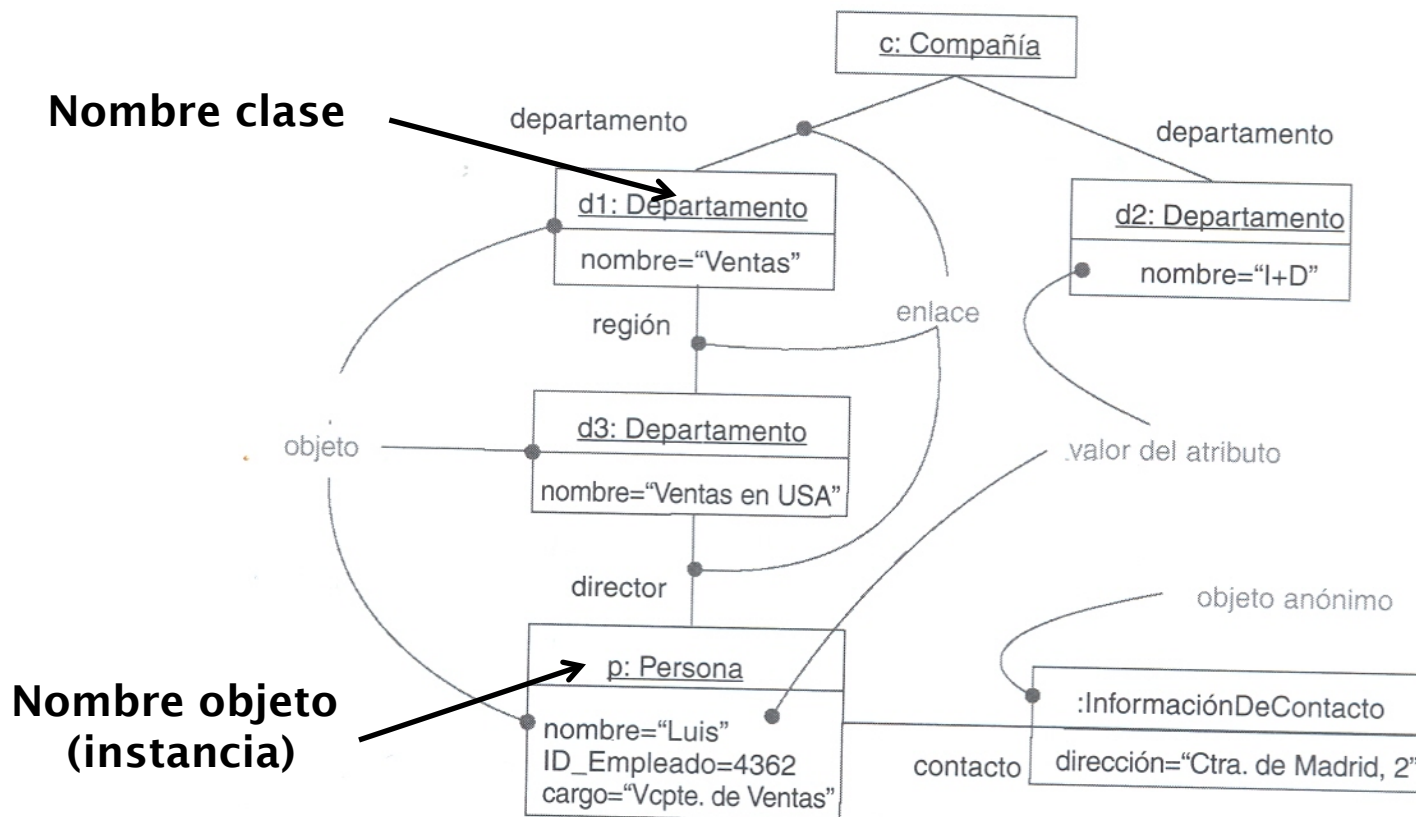


Elemento básico: Objeto

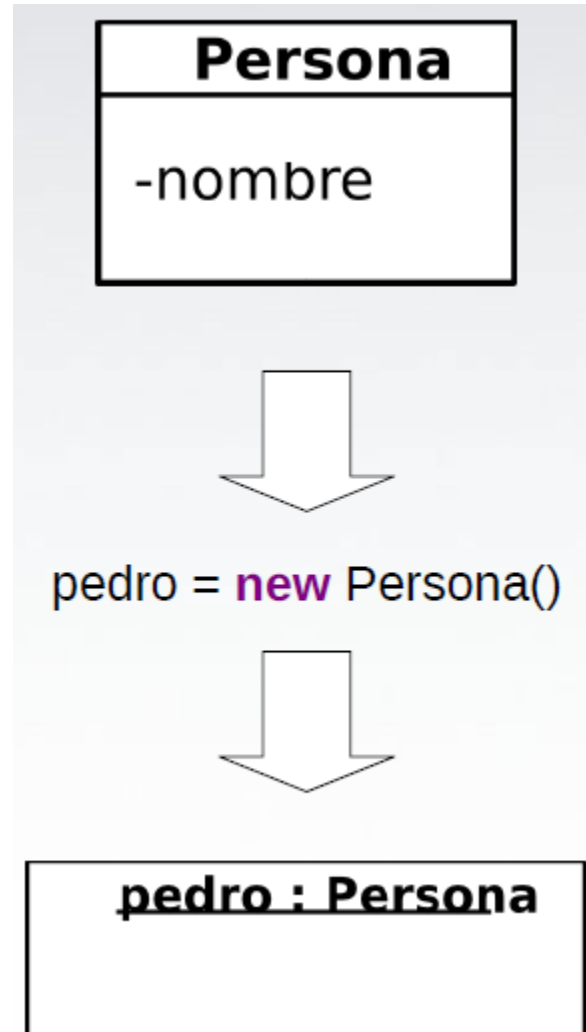
- Un objeto es una **instancia** de una abstracción de tipo clase.
- El estado de un objeto viene determinado por todos los pares **<propiedad>: <valor>**
- Se puede indicar el estado del objeto entre corchetes []



Diagrama de Objetos: Notación



Ejemplo Detallado: OBJETOS



Ejemplo Detallado: OBJETOS

```
public class Persona {  
  
    private String nombre;  
    private char   sexo;  
    private Date   fechaNac;  
    private String profesion;  
  
    public Persona(  
        String nombre, char sexo, Date fechaNac, String profesion) {  
        this.nombre     = nombre;  
        this.sexo       = sexo;  
        this.fechaNac   = fechaNac;  
        this.profesion  = profesion;  
    }  
}
```

Persona
-nombre
-sexo
-fechaNac
-profesion
+Persona(nombre, sexo, fechaNac, profesion)

Existe una diferencia muy importante entre un **Objeto** y una **Clase**

Ejemplo Detallado: OBJETOS

Persona
-nombre
-sexo
-fechaNac
-profesion
+Persona(nombre, sexo, fechaNac, profesion)

```
Persona p1 = new Persona(  
    "Pedro", 'M', new Date(16, 7, 1988), "Actor" );  
  
Persona p2 = new Persona(  
    "Andrea", 'F', new Date(14, 4, 1980), "Ceramista");  
  
Persona p3 = new Persona(  
    "María", 'F', new Date(23, 11, 1960), "Médico" );  
  
Persona p4 = new Persona(  
    "Luis", 'M', new Date(12, 1, 1977), "Ingeniero");
```

Crear Instancias (Instanciar)

Pedro : Persona
sexo = M fechaNac = 16/07/1988 profesion = Actor

Andrea : Persona
sexo = F fechaNac = 14/04/1980 profesion = Ceramista

María : Persona
sexo = F fechaNac = 23/11/1960 profesion = Médico

Luis : Persona
sexo = M fechaNac = 12/01/77 profesion = Ingeniero

Aspectos a tener en cuenta...

- Usados cuando se desea modelar **una vista del sistema en un instante concreto**
- Un diagrama de objetos debe **corresponderse** con un diagrama de clases
- Un diagrama de objetos debe ser **consistente** con el diagrama de clases que instancia
 - o **No pueden aparecer nuevas relaciones**
- Usar anotaciones para realizar las aclaraciones que sean necesarias