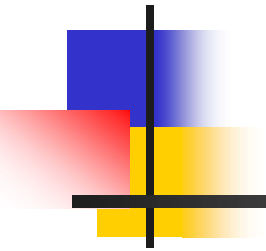


Unidad 1
Diseño Orientado a Objetos

Asignatura: Programación Orientada a Objetos

Licenciatura en Ciencias de la Computación

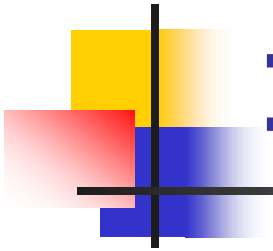
Licenciatura en Sistemas de Información

Tecnicatura en Programación WEB

Departamento de Informática

FCEFN – UNSJ

Año 2025



INTRODUCCION A LA O.O (1)

Objetivos:

Al término de esta unidad se espera:

- Que el estudiante distinga las ventajas de la orientación a objetos, frente a otros tipos de diseño.
- Que el estudiante Interprete los conceptos de encapsulamiento, abstracción, reusabilidad y herencia.
- Que el estudiante diseñe y organice adecuadamente una jerarquía de clases, atendiendo a los distintos tipos de relaciones entre clases.

INTRODUCCION A LA O.O (2)

Programación Imperativa

→ **los algoritmos se describen en base a procesos**

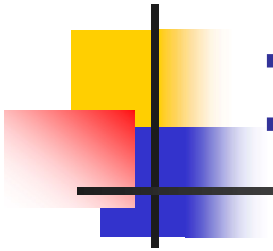
cómo se abre una ventana, cómo se la cierra, cómo se la limpia, etc. Así, los algoritmos se expresan mediante procesos o funciones y éstos como una secuencia de tareas a realizar por la computadora. Por eso este tipo de programación se llama procedimental o imperativa.

Programación Modular

→ Así, la programación modular permitió dividir el programa en módulos autónomos que se pudieran programar, verificar y modificar individualmente

Problema: El uso de la Abstracción

Solución: Ocultamiento de la implementación



INTRODUCCION A LA O.O (3)

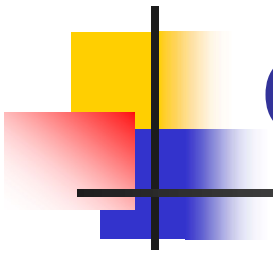
Tipo de Datos Abrstracto:

1. **Un tipo de datos definido por el programador,**

2. **Un conjunto de operaciones abstractas sobre objetos de ese tipo,**

3. **Encapsulamiento de los objetos de ese tipo, de tal manera que el usuario final del tipo no pueda manipular esos objetos excepto a través del uso de las operaciones definidas.**

Pratt (Pág. 351)



Características del Diseño O.O

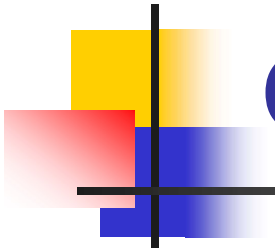
- En la programación imperativa tradicional, los algoritmos se describen en base a procesos.

- La programación orientada a objetos encara la resolución de cada problema desde la óptica del objeto.

- ***El objeto*** combina los datos (**atributos del objeto**) con los procedimientos u operaciones (***métodos***) que actúan sobre dichos datos.

- Los objetos interactúan entre sí enviando ***mensajes***

- Los métodos son, en la práctica, muy similares a los procedimientos de la programación imperativa tradicional y los mensajes se podrían pensar como invocaciones a esos procedimientos.



Características del Diseño O.O

- El programador orientado a objetos puede agrupar características comunes de un conjunto de objetos en ***clases***, a través de un proceso de abstracción.

- Los descendientes de estas clases se construyen por medio del mecanismo de subclasificación, permitiendo que sean ***heredados*** los métodos programados anteriormente y debiendo programar solamente las diferencias.

Características del Diseño O.O

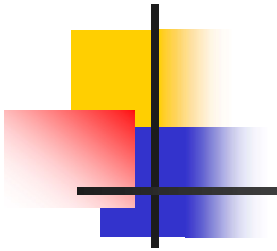
- La Programación Orientada a Objetos (POO) no se puede desligar de todo el paradigma de orientación a objetos.

  Paradigma: Conjunto de prácticas y saberes que definen una disciplina científica durante un período específico (Thomas S. Kuhn)

- El principio fundamental del paradigma de programación orientada a objetos es construir un sistema de software en base a las entidades de un modelo elaborado a partir de un proceso de abstracción y clasificación.

- Para hacer buena POO hay que desarrollar todo el sistema utilizando el paradigma, empezando por un análisis y un diseño orientados a objetos.

- En general, el desarrollo de software implica, desde una visión orientada a objetos, una serie de etapas para hacer: requerimientos, análisis, diseño, implementación y prueba.



ACTIVIDAD 1

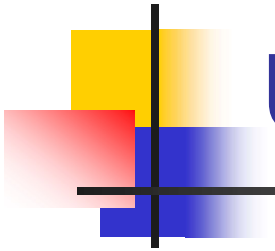
a) Agregue nuevas cuestiones inherentes a los conceptos de abstracción y ocultamiento de información que a su criterio no están presentes en este documento.

[1] Fontela, Carlos. Programación Orientada a Objetos – Técnicas Avanzada de Programación – Pág. 16 a 23

b) Teniendo en cuenta que uno de los objetivos de la orientación a objetos es la calidad del software, defina calidad de software e indique factores para obtener dicha calidad.

c) Cuando se habla de productividad del software, se habla de reducir los costos del software y el tiempo de desarrollo. Investigue sobre cualidades relacionadas a la productividad del software.

[2] Libro Programación Orientada a Objetos UNSur – Pág. 5 a 7



UML (Lenguaje de Modelado Unificado)

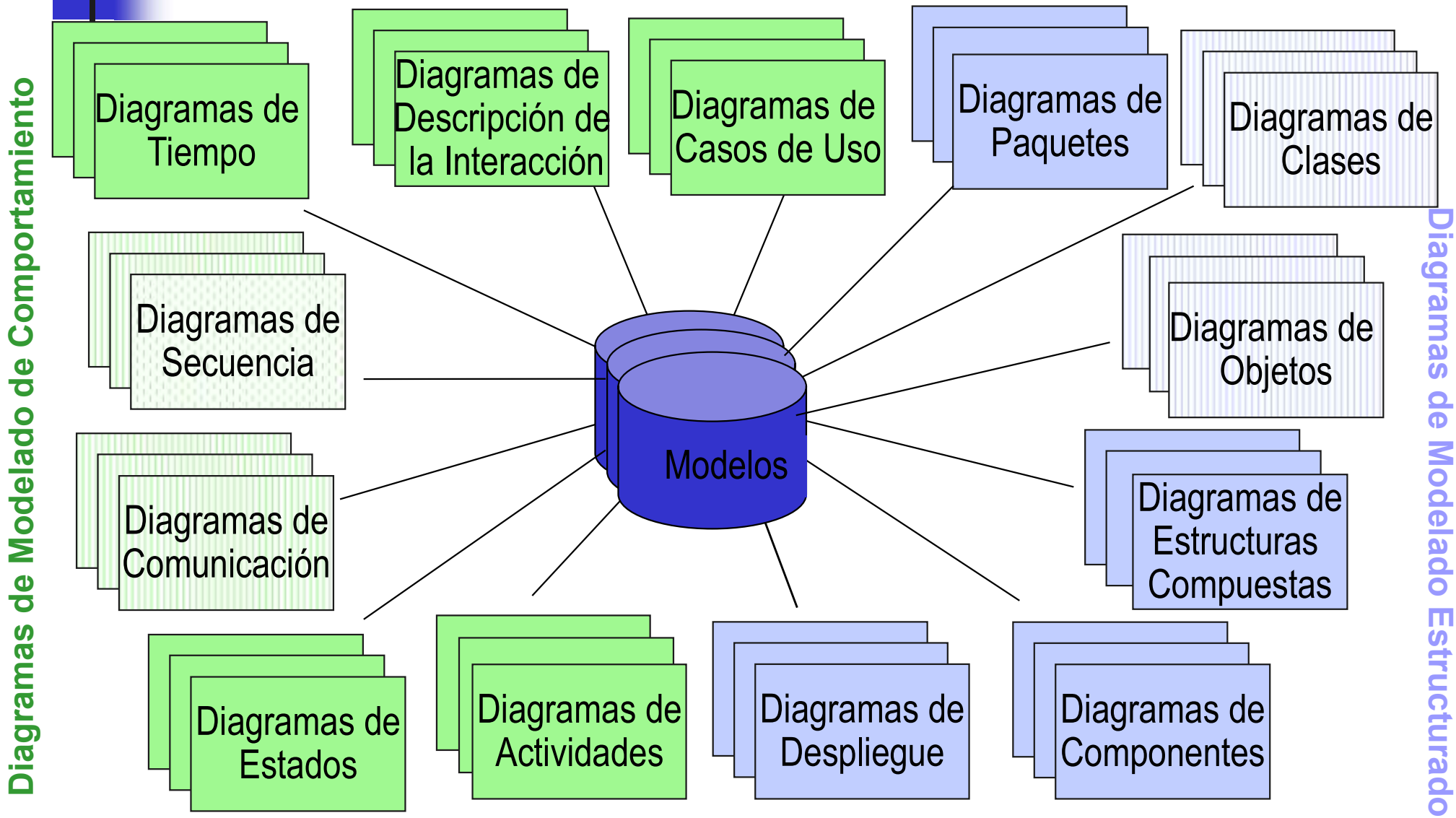
- UML es un lenguaje que permite la visualización, especificación y documentación de sistemas orientados a objetos.

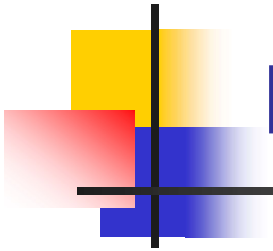
- UML no es una metodología sino una notación, que aglutina distintos enfoques de orientación a objetos.

- UML 2 define trece (13) diagramas para describir distintas perspectivas del sistema

Los diagramas que están con raya verticales son los que usaremos a los largo de esta asignatura

UML (Lenguaje de Modelado Unificado – UML 2)





Elementos Básicos de la O.O

1. Objetos
2. Clases
3. Métodos y Mensajes
4. Herencia

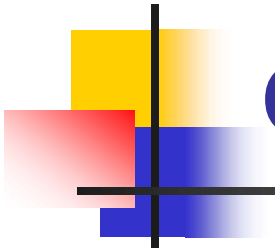
Objetos

identifica

Un **objeto del problema** es una entidad, física o conceptual, caracterizada a través de atributos y comportamiento. El comportamiento queda determinado por un conjunto de servicios que el objeto puede brindar y un conjunto de responsabilidades que debe asumir.

abstrae

Un objeto de software es un modelo, una representación de un objeto del problema.



Objetos

Un objeto es una unidad atómica que encapsula estado y comportamiento.

La encapsulación en un objeto permite una alta cohesión y un bajo acoplamiento.

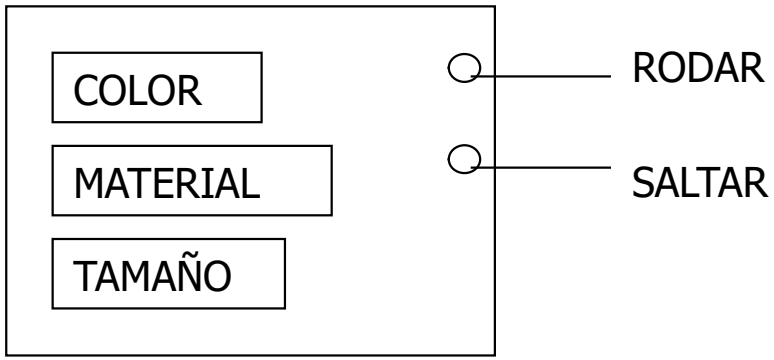
Los objetos son entidades que tienen atributos (datos) y comportamiento particular (procedimientos)

Atributos de un objeto: Los atributos describen la abstracción de características individuales que posee un objeto.

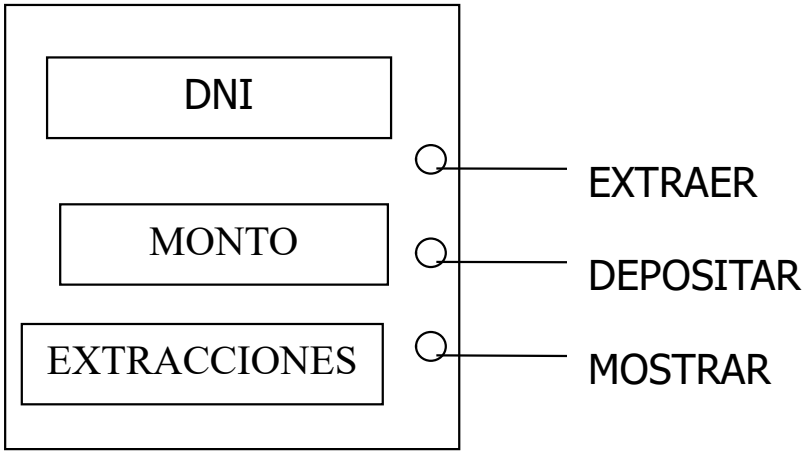
Comportamientos: Los comportamientos de un objeto representan las operaciones que pueden ser realizadas por un objeto.

Objetos

PELOTA

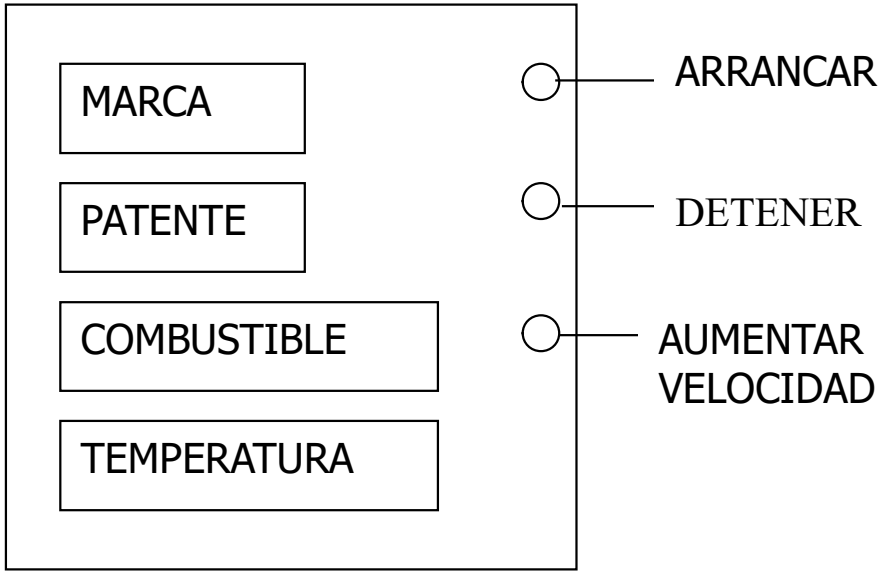


CUENTA



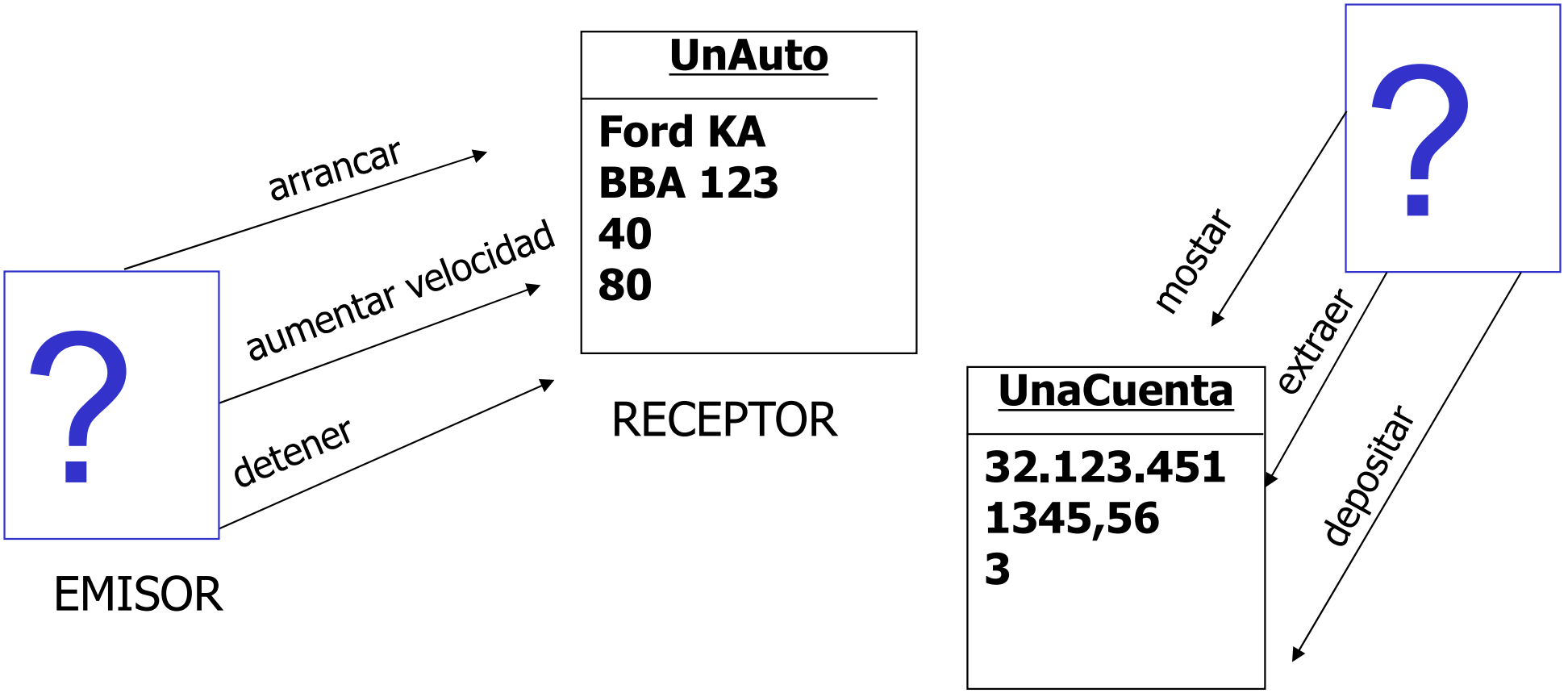
Objeto : pelota
Comportamiento: rodar, saltar
Estado : color : rojo, material : cuero, tamaño : pequeño.

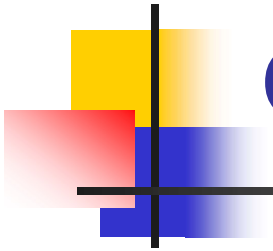
AUTO



Objetos

Objeto = Estado + Comportamiento + Identidad





Objetos

Objeto = Estado + Comportamiento + Identidad

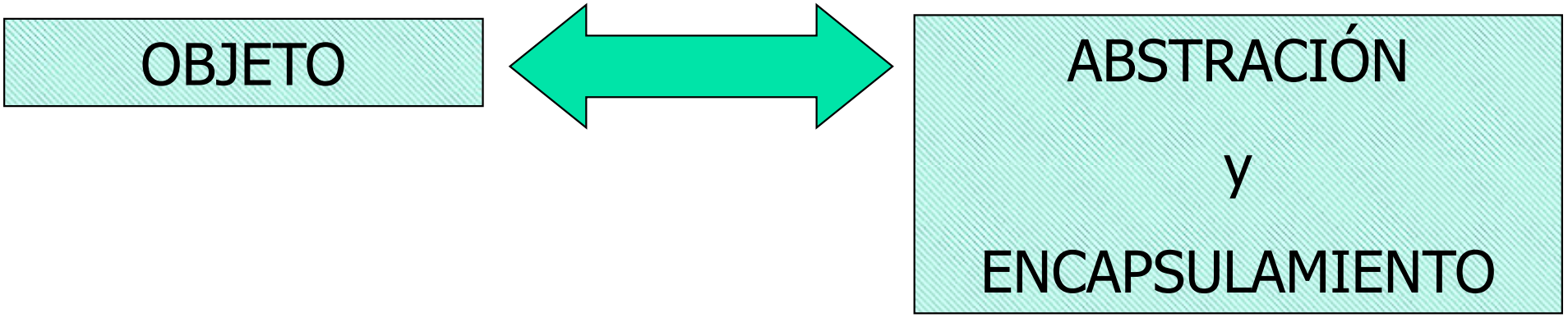
- El estado agrupa los valores instantáneos de todos los atributos de un objeto. El estado evoluciona con el tiempo.

- El comportamiento describe las acciones y reacciones de ese objeto.

- Las acciones u operaciones de un objeto se desencadenan como consecuencia de un estímulo externo, representado en forma de un mensaje enviado por otro objeto**. El estado y el comportamiento están relacionados.**

- La identidad permite distinguir los objetos de forma no ambigua, independientemente de su estado. Esto permite distinguir dos objetos en los que todos los valores de los atributos son idénticos.

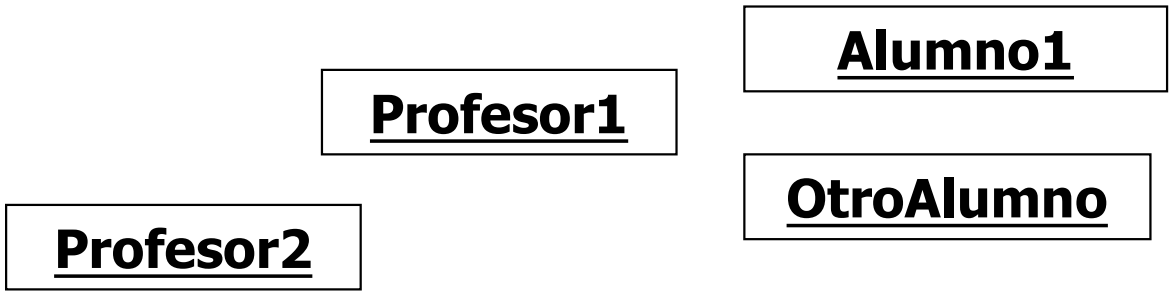
Objetos



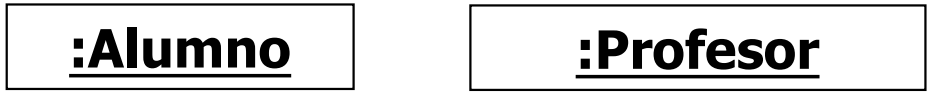
Los objetos informáticos definen una representación abstracta de las entidades de un mundo real o virtual, con el objetivo de controlarlos o simularlos

Objetos

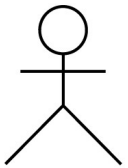
En UML, un objeto se representa bajo la forma de un rectángulo; el nombre del objeto se subraya.



O bien usando un nombre genérico mediante el uso de los ‘:’



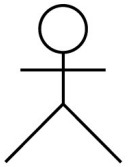
Actores



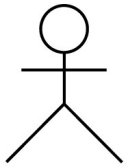
Actor

La interacción de los objetos es iniciada por el usuario. Por lo que, para modelar un sistema es necesario identificar quien o quienes son sus usuarios. A partir de esto, se define el concepto de actor, que es una entidad externa al propio sistema, pero que necesita intercambiar información con él.

Los actores no están restringidos a ser personas físicas, también pueden representar a sistemas externos al sistema que se está modelando.
En el caso de los usuarios que son personas reales, el actor representa la función que la persona realiza.



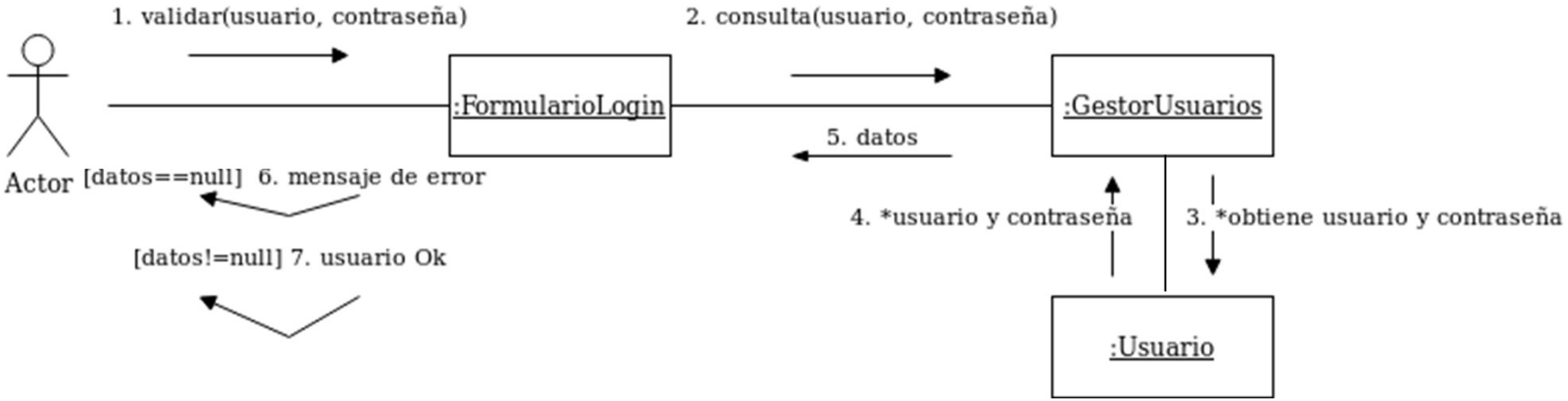
Cajero



Gerente

Objetos – Diagramas de Comunicación

Muestran las **interacciones entre objetos** en la estructura espacial estática, que permite la **colaboración entre objetos**. El tiempo no se representa de manera explícita, por lo tanto los mensajes se numeran para indicar el orden de los envíos.

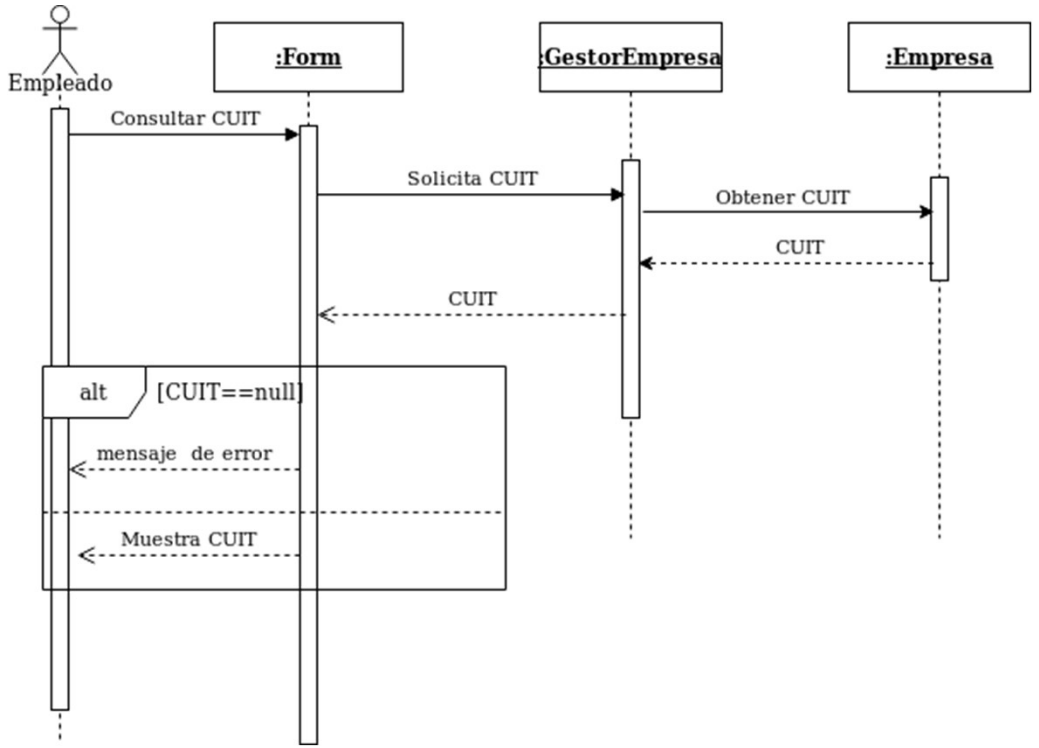


El * precediendo un mensaje, se utiliza para representar iteración
Los [ ], con texto entre ellos, se utilizan para expresar una condición de guarda en un condicional, del tipo if… then...

Objetos – Diagramas de secuencia

Estos diagramas muestran interacciones entre objetos según un punto de vista temporal. Un objeto se representa por un rectángulo y el tiempo de vida se representa por una barra vertical llamada línea de vida de los objetos. El orden de envío de los mensajes está dado por la posición sobre el eje vertical.

Ejemplo: un Empleado, solicita a través de un formulario, el CUIT de una Empresa, el formulario, muestra al empleado, el CUIT o un mensaje de error, si el CUIT es null.



Objetos – Diagramas de secuencia
Fragmentos combinados

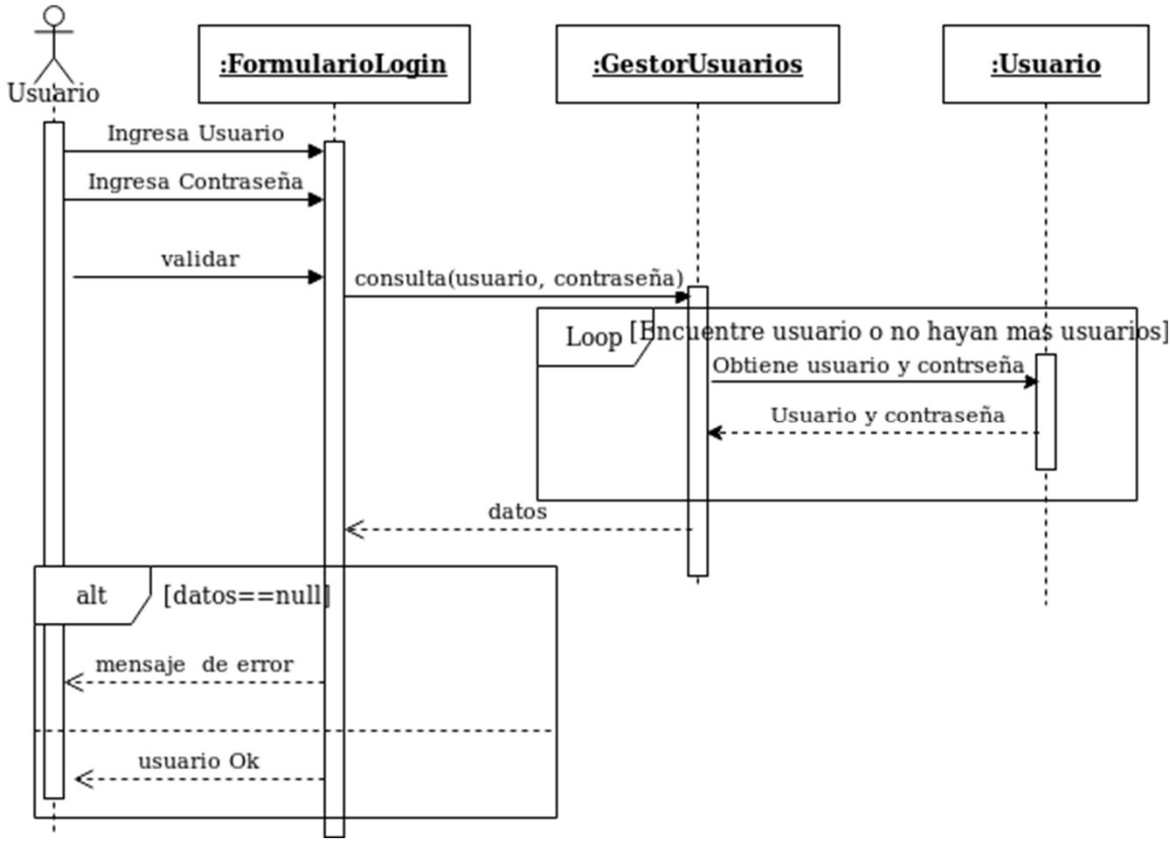
Existen mecanismos que permiten agregar un grado de lógica de procedimientos a los diagramas.
Un fragmento combinado es una o más secuencias de procesos incluidas en un marco y ejecutadas bajo circunstancias específicas.

Fragmento Alternativa (denotado "**alt**"): modela la elección de una interacción de objetos, a través de una condición de guarda, es decir, modela estructuras condicionales del tipo if…then…else.

Fragmento de iteración o bucle (denotado "**loop**"): el fragmento incluye un conjunto de mensajes que se ejecutan múltiples veces, según lo indique la condición de guarda

Objetos – Diagramas de secuencia
Fragmentos Combinados (I)

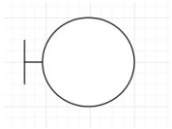
Ejemplo: un Usuario, ingresa nombre de usuario y contraseña, selecciona validar, la aplicación valida el usuario y contraseña ingresado, retornando los un mensaje de error en caso de que el usuario y/o contraseña no sean válidos, o una bandera para indicar el caso contrario. validar el usuario.

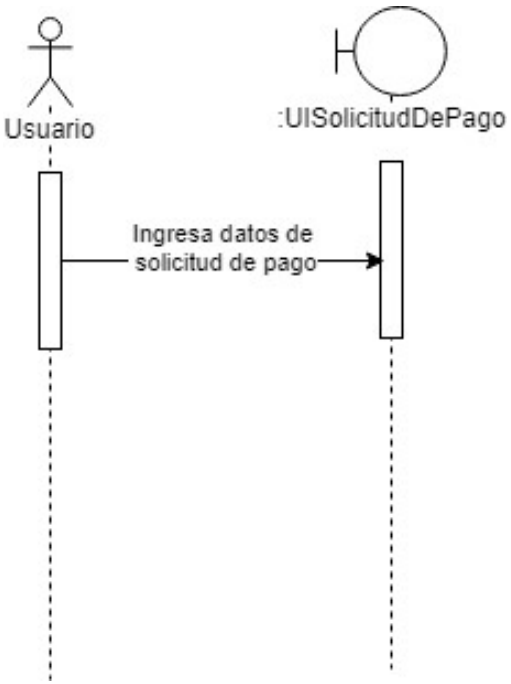


Objetos con estereotipo (I)

Es común que en las aplicaciones (en especial las aplicaciones web) usen otros **objetos que no pertenecen al dominio del problema**, algunos surgen para presentar una interfaz al usuario (actor) de la aplicación, otros para controlar la lógica interna de la ésta.

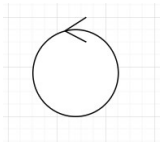
a) Objeto de Interfaz (Boundary): representa un elemento (ejemplo un formulario web) con el cual interactúa el usuario (actor). A menudo representan una abstracción de una ventana, un formulario, una interfaz de comunicación, etc. Ejemplo: el formulario de entrada de usuario y contraseña para ingreso a una cuenta de correo.

Notación 

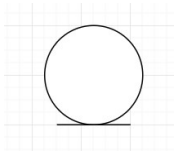


Objetos con estereotipos (II)

b) Objeto de Control (Controllers): se ocupa de organizar y controlar la lógica requerida para alcanzar el objetivo de una determinada funcionalidad. Media entre los objetos de interfaz y los de entidad. Ejemplo, lectura de datos del usuario del sistema de correo electrónico.

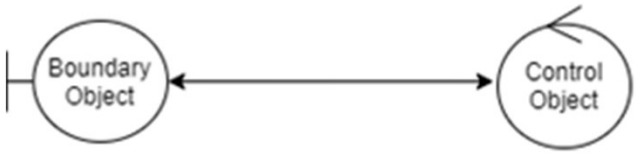
Notación 

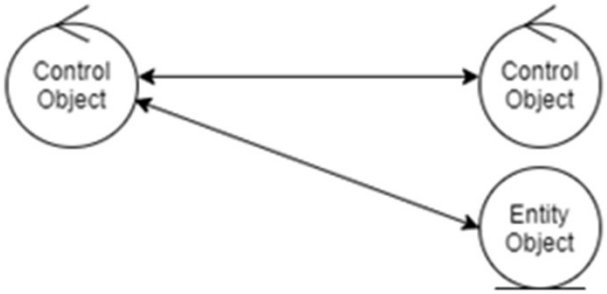
c) Objeto de Entidad (Entity): para cada uno de los objetos (entidades) del dominio requeridos para realizar la funcionalidad. Modelan información asociada a algún fenómeno o concepto, como persona o un objeto.

Notación 

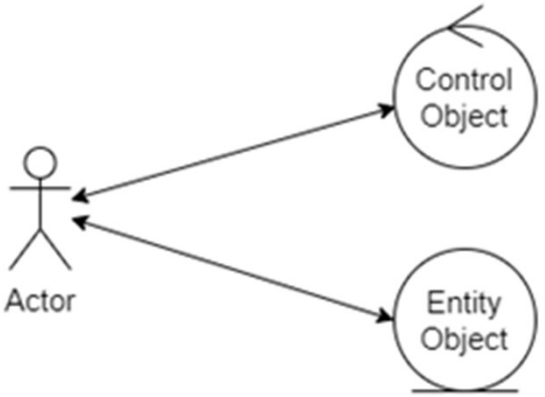
Estereotipos – Relaciones permitidas y no permitidas

Relaciones permitidas



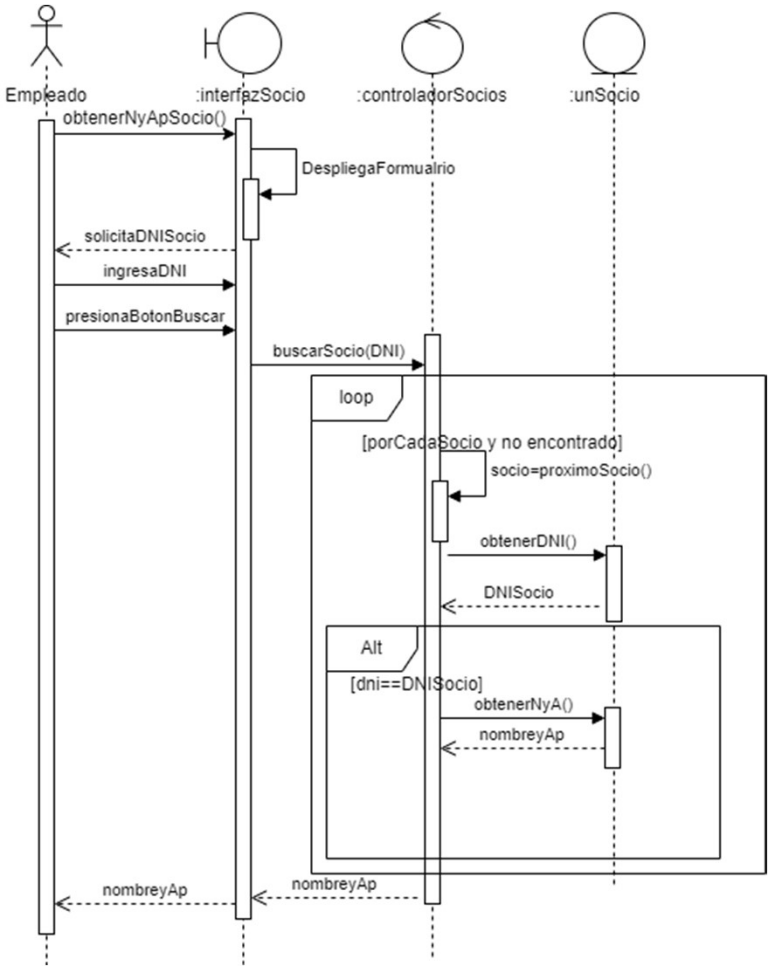


Relaciones NO permitidas



Ejemplo de diagrama de secuencia con estereotipos – Búsqueda de información

Ejemplo: El empleado, selecciona obtener Nombre y Apellido de un socio, el sistema solicita el DNI del socio a buscar, la aplicación devuelve el nombre y apellido del socio al que pertenece el DNI ingresado.

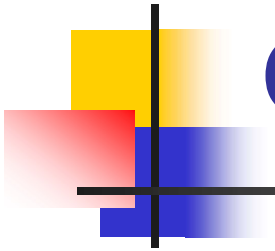


Clases

- Una clase abstrae las características de un conjunto de objetos con comportamientos similares.

- La **encapsulación** de una clase permite la cohesión y presenta distintas ventajas básicas:

✓ Se protegen los datos de accesos indebidos
✓ El acoplamiento entre las clases se disminuye
✓ Favorece la modularidad y el mantenimiento



Clases

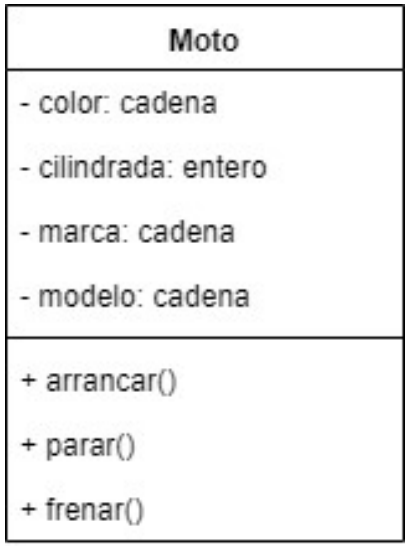
- Una **clase** es una descripción de un conjunto de objetos, ya que consta de comportamientos y atributos que resumen las características comunes del conjunto.

- La posibilidad de definir clases es una de las ventajas de la orientación a objetos; definir clases significa **colocar código reutilizable** en un depósito común en lugar de redefinirlo cada vez que se necesite.

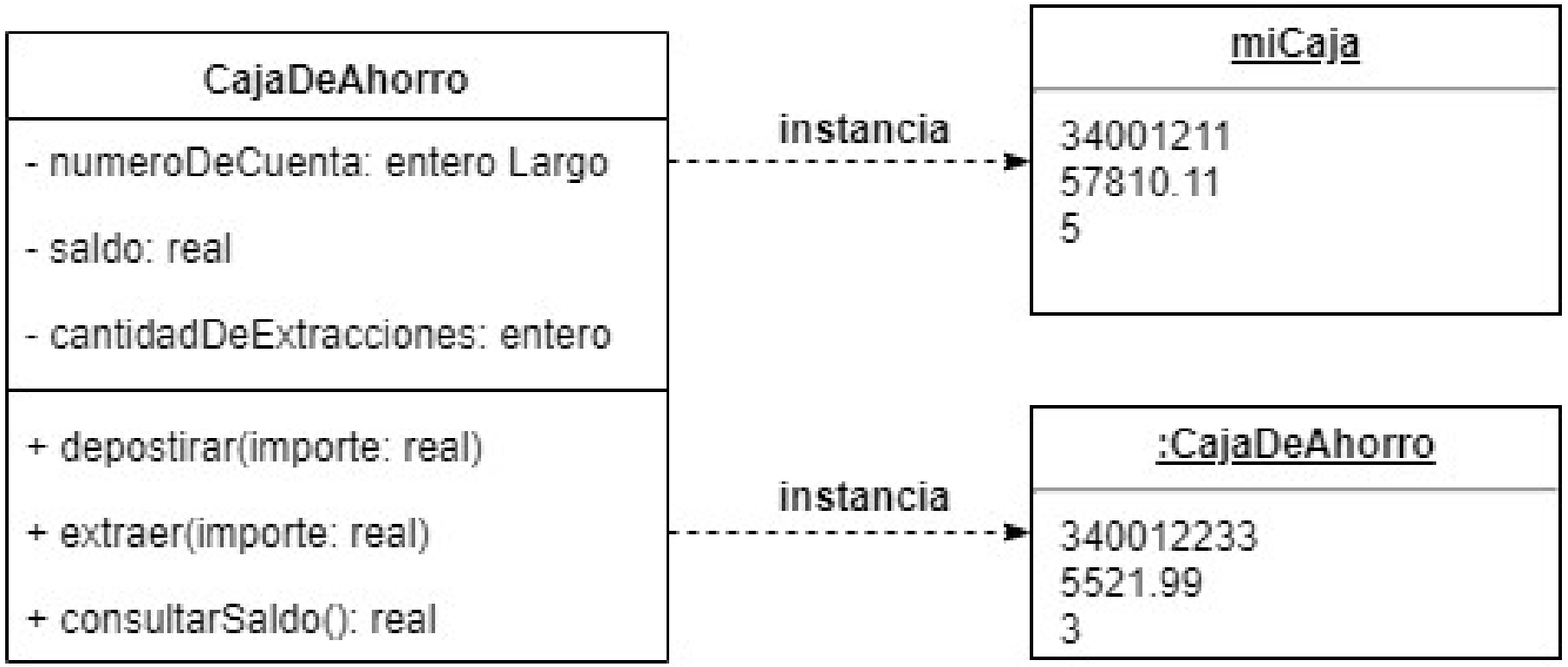
- Cada objeto es instancia de una clase.

Clases

- Cada clase se representa en un rectángulo con tres compartimientos:
  - nombre de la clase
  - atributos de la clase
  - operaciones de la clase



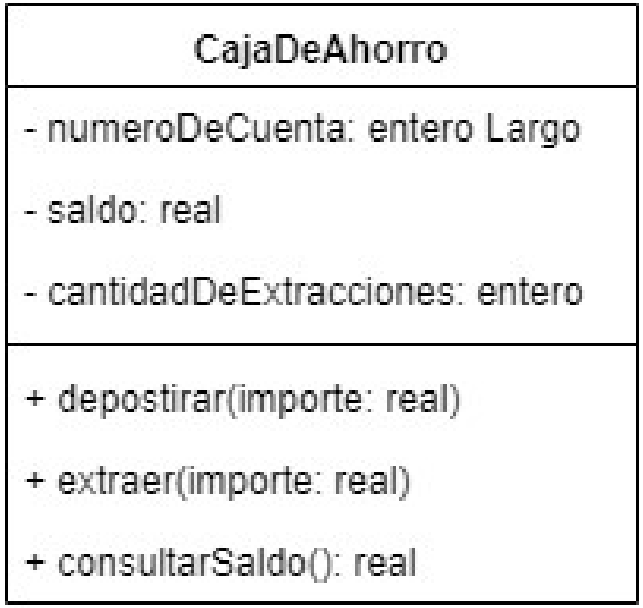
Clases

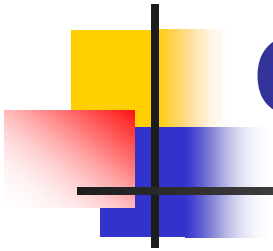

CajaDeAhorro
- numeroDeCuenta: entero Largo
- saldo: real
- cantidadDeExtracciones: entero
+ depostirar(importe: real)
+ extraer(importe: real)
+ consultarSaldo(): real
instancia
instancia
miCaja
34001211
57810.11
5
:CajaDeAhorro
340012233
5521.99
3

Clases - Visibilidad

Los atributos de una clase no deberían ser manipulables directamente por el resto de objetos, no obstante existen distintos **niveles de encapsulación** también llamados niveles de visibilidad.

Reglas de visibilidad
+ Atributo público # Atributo protegido - Atributo privado
+ Método público # Método protegido - Método privado





Clases – Métodos y Mensajes

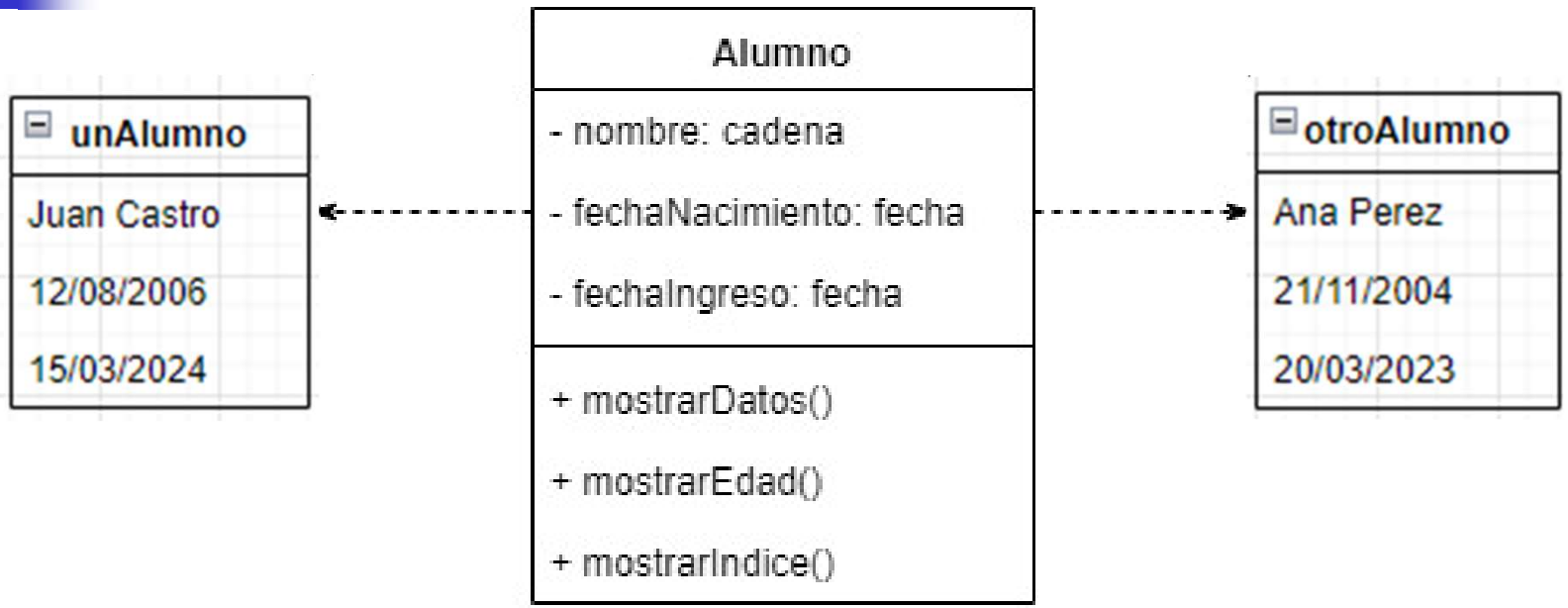
- Los objetos tienen la posibilidad de actuar. La acción sucede cuando un objeto recibe un **mensaje**, que es una solicitud que pide al objeto que se comporte de manera determinada.

- Cada objeto recibe, interpreta y responde a mensajes enviados por otros objetos.

- Los comportamientos u operaciones que caracterizan un conjunto de objetos residen en la clase y se llaman **métodos**.

- Los **métodos** son el **código** que se **ejecuta** para **responder a un mensaje**, y el mensaje es la llamada o invocación a un método.

Clases – Métodos y Mensajes



Mensaje

unAlumno.mostrarEdad()

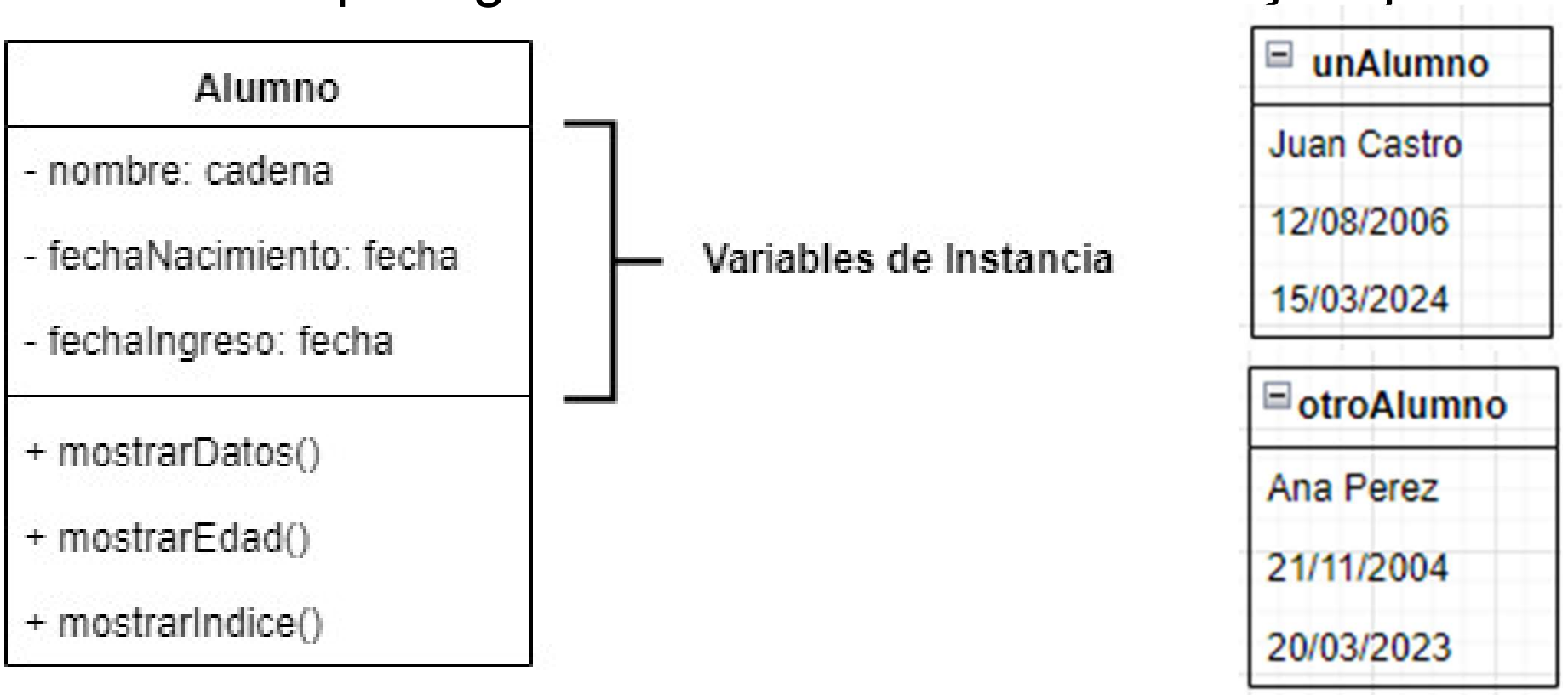
otroAlumno.mostrarEdad()

Método

Alumno.mostrarEdad()

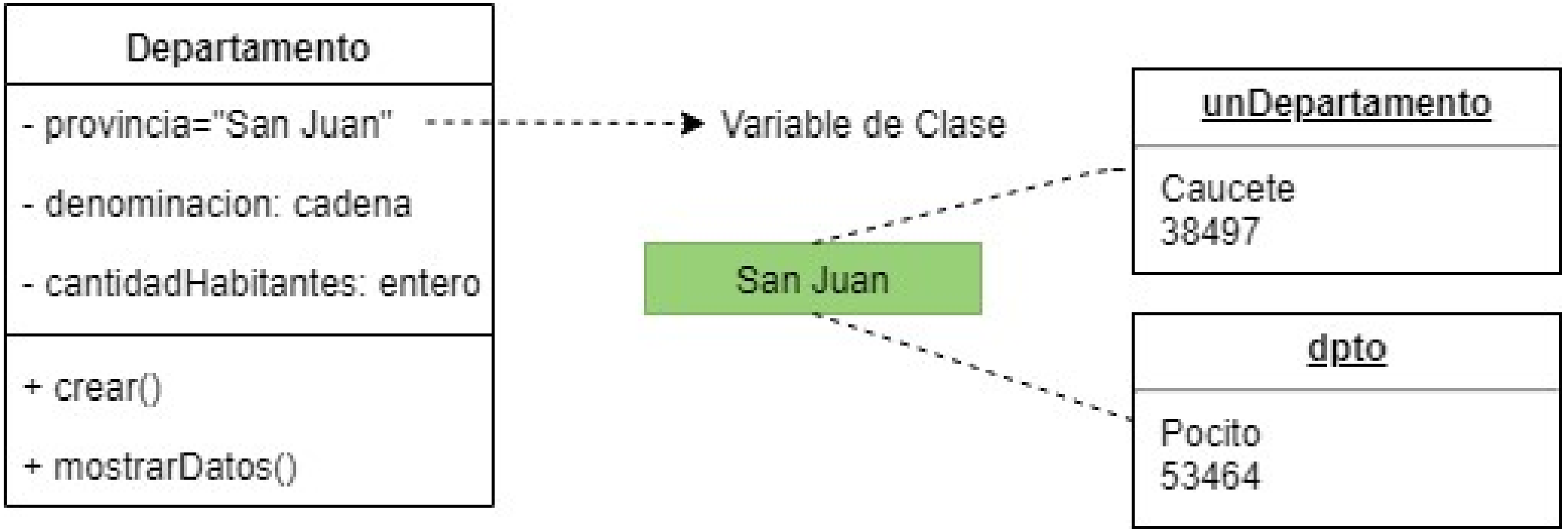
Clases – variables de clases y de instancia

Variables de instancia: Las variables de instancia o miembros dato se usan para guardar los atributos de un objeto particular.



Clases – variables de clases y de instancia

Variables de clase: son aquellos atributos que tienen el mismo valor para cada objeto de la clase. Si el valor de la variable de clase es cambiado para una instancia, el mismo cambia para todas las instancias de la clase y subclase.
Representa un área de memoria compartida por todos los objetos de la clase



Atributos de la instancia dpto:

provincia="San Juan", denominacion="Pocito", cantidadHabitantes=53464

REVISION

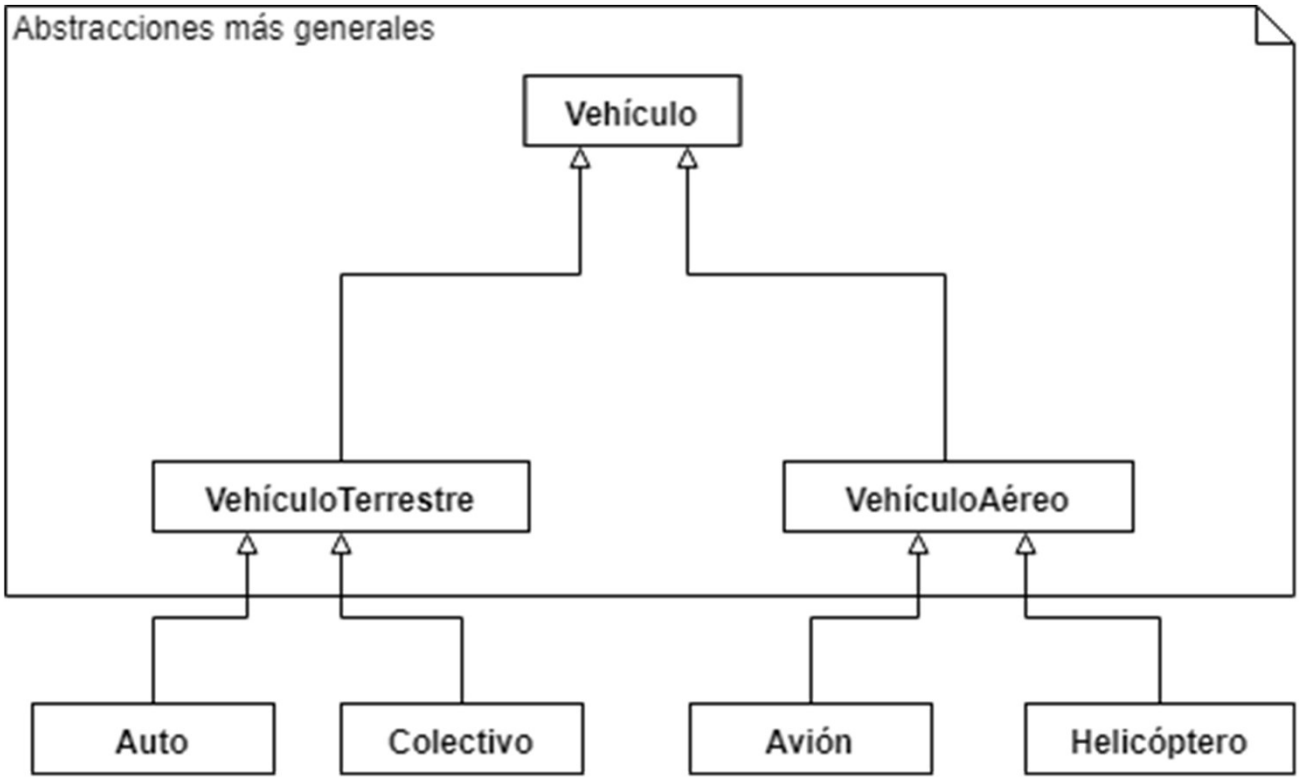
A) Dada la clase ALUMNO, grafique dos instancias de dicha clase usando UML.
B) Indique claramente el estado de uno de los objetos propuestos y diga los posibles comportamientos.

ALUMNO
- facultad="FCEFyN" - matricula: cadena - ingreso: entero - promedio: flotante
+ mostrarDatos() + actualizarPromedio()

1) Indique claramente la diferencia entre método y mensaje.
2) Defina claramente el concepto de objeto y de clase.

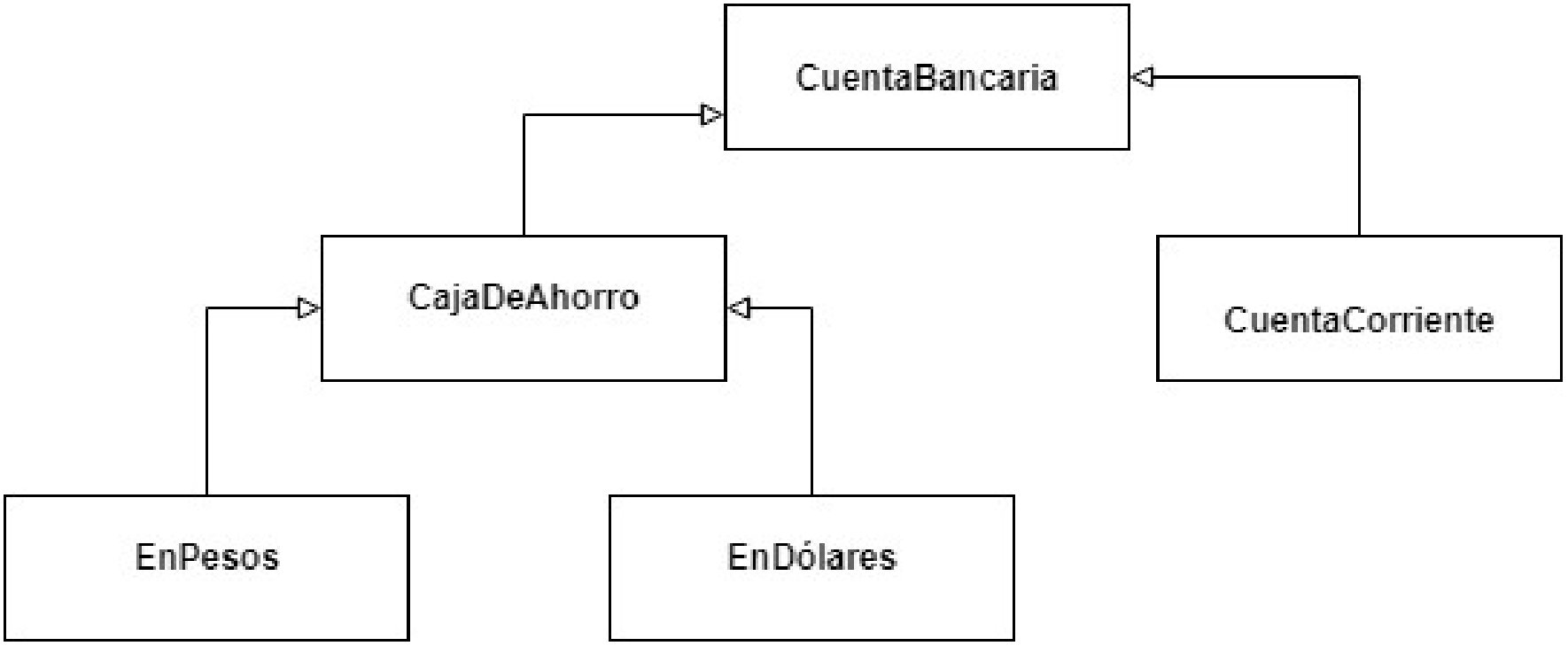
Herencia – Generalización y Especialización

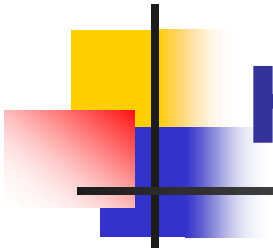
La **generalización** consiste en factorizar los elementos comunes (atributos, métodos y restricciones) de un conjunto de clases en una clase más general llamada superclase.



Herencia – Generalización y Especialización

La **especialización** permite capturar las particularidades de un conjunto de objetos **no discriminadas** por las clases ya identificadas.



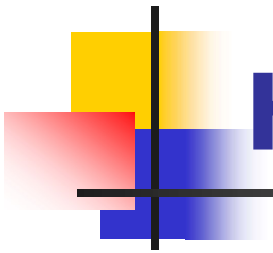


Herencia – Generalización y Especialización

- Las clases se ordenan según una jerarquía, una superclase es una abstracción de sus subclases.

- Los árboles de clases no crecen a partir de una raíz. Por el contrario, se determinan partiendo de las hojas porque éstas pertenecen al mundo real mientras que los niveles superiores son abstracciones construidas para ordenar y comprender.

- Una **subclase** identifica el comportamiento de un conjunto de objetos que hereda las características de la clase padre y adicionan algunas específicas que ésta no posee.

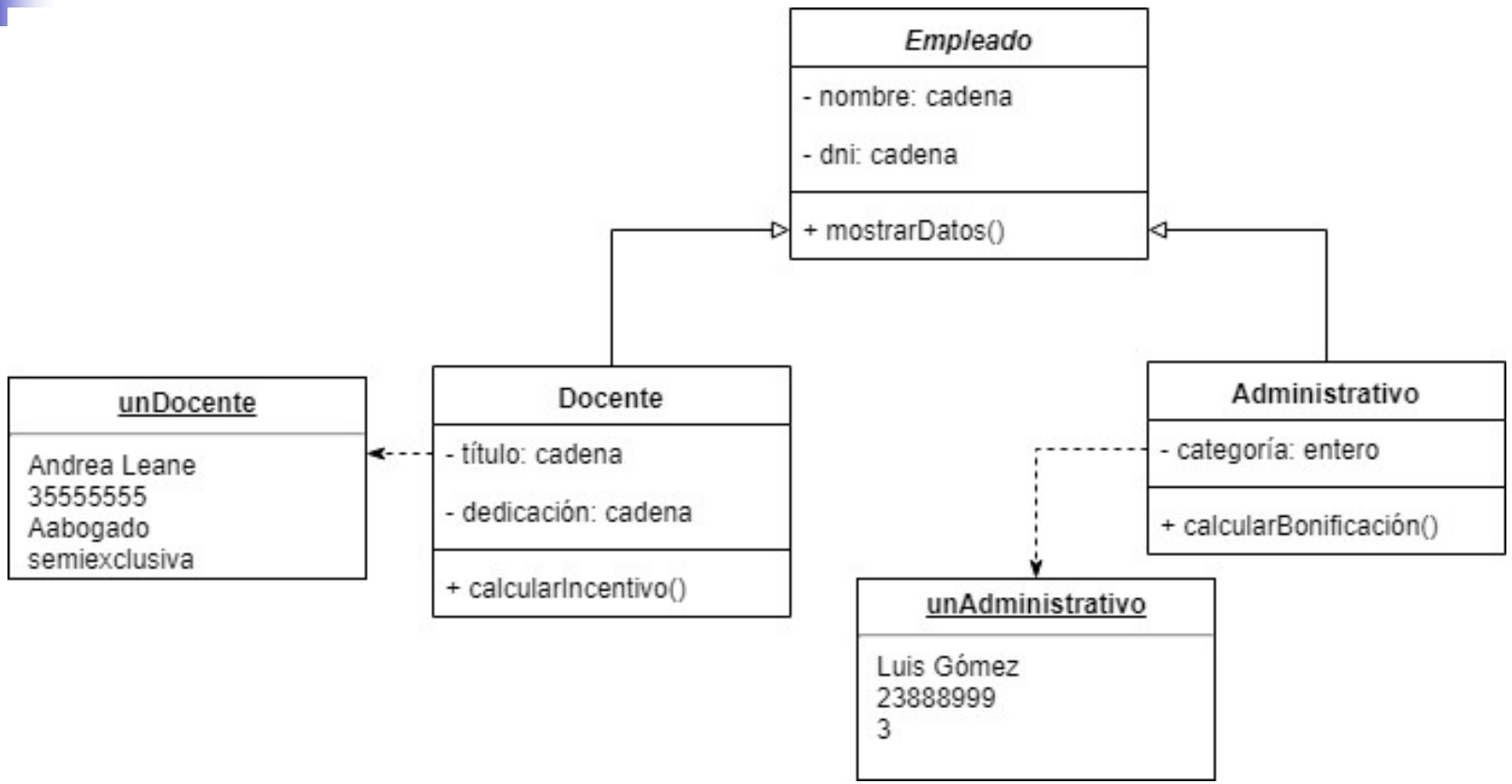


Herencia – clase abstracta y clase concreta

Clase Abstracta: Una clase es abstracta cuando no existe un objeto que sea instancia directa de ella, pero sí existe una subclase de ella que es instanciable.
Generalmente se utilizan para resumir los comportamientos comunes a un conjunto de subclases.
El nombre de las clases abstractas va en cursiva (UML).

Clase Concreta: Una clase concreta es aquella que es instanciable, es decir que existe al menos un objeto de esa clase.

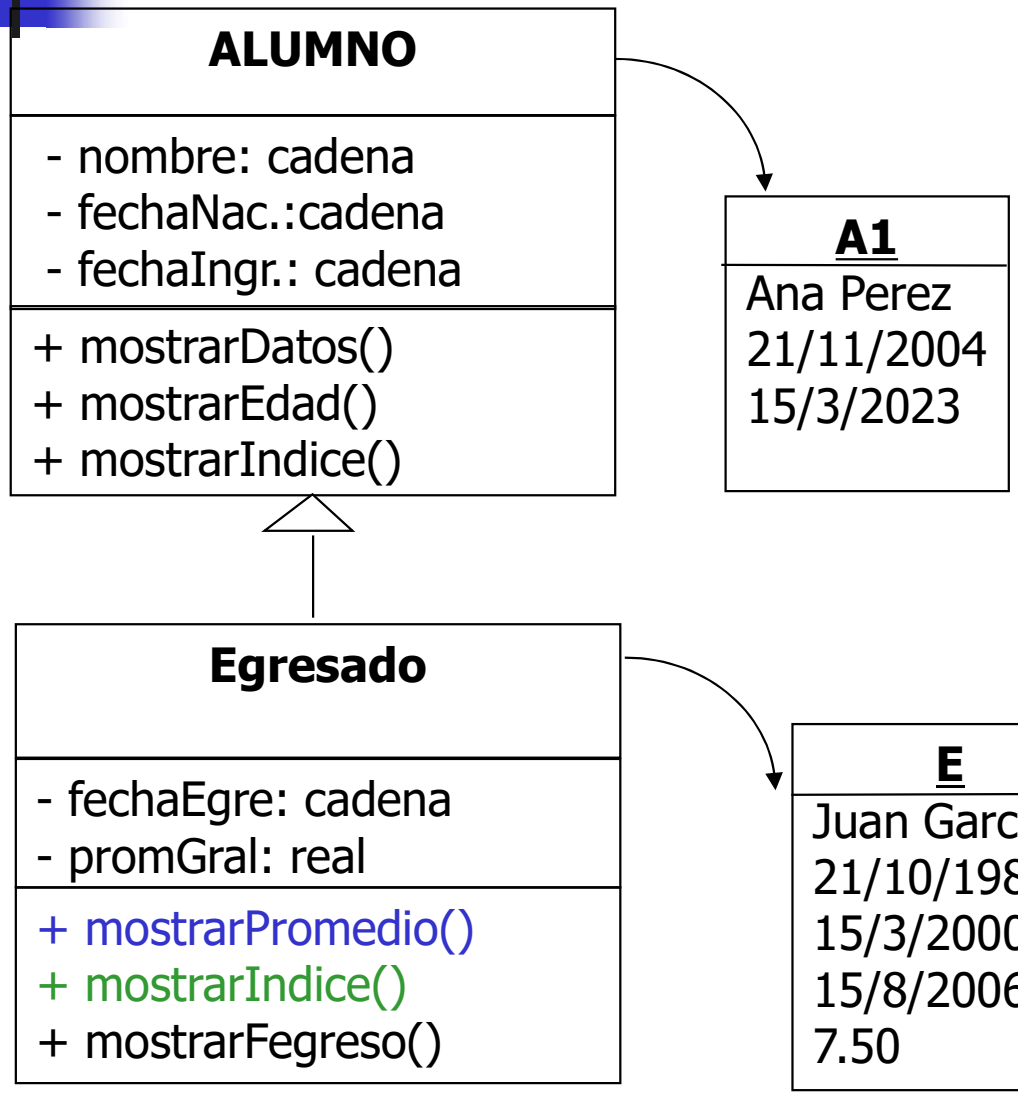
Herencia – clase abstracta y clase concreta



En este contexto, donde sólo hay empleados docentes y administrativos, la clase **Empleado** es una **clase abstracta**, y las clases **Docente** y **Administrativo**, son **clases concretas**.

Actividad para realizar en clase: Indicar métodos propios de la clase egresado, métodos heredados y métodos redefinidos

Herencia – métodos y mensajes



A1.mostrarEdad()?

E.mostrarEdad()?

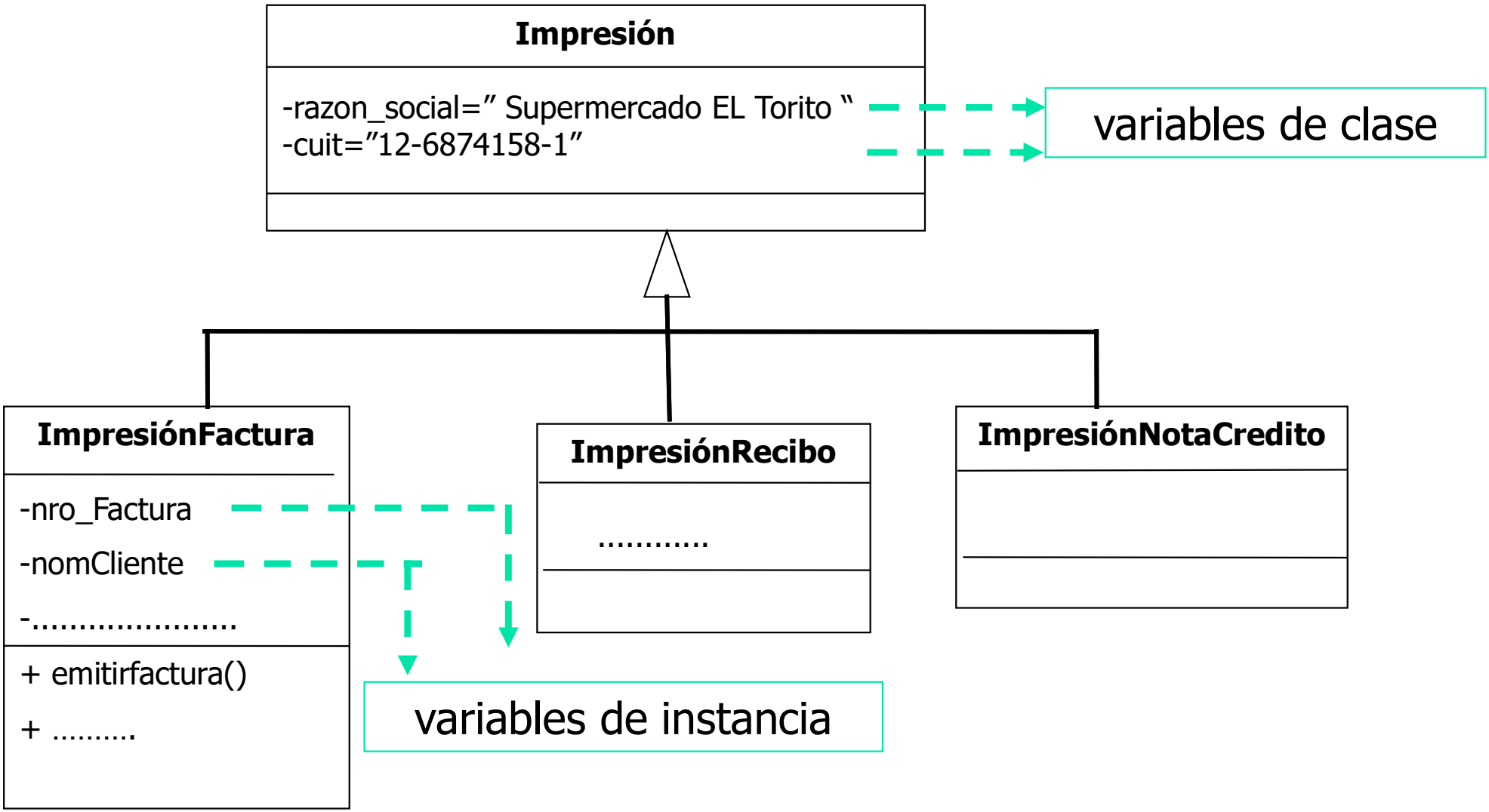
A1.mostrarIndice()?

E.mostrarIndice()?

E.mostrarPromedio()?

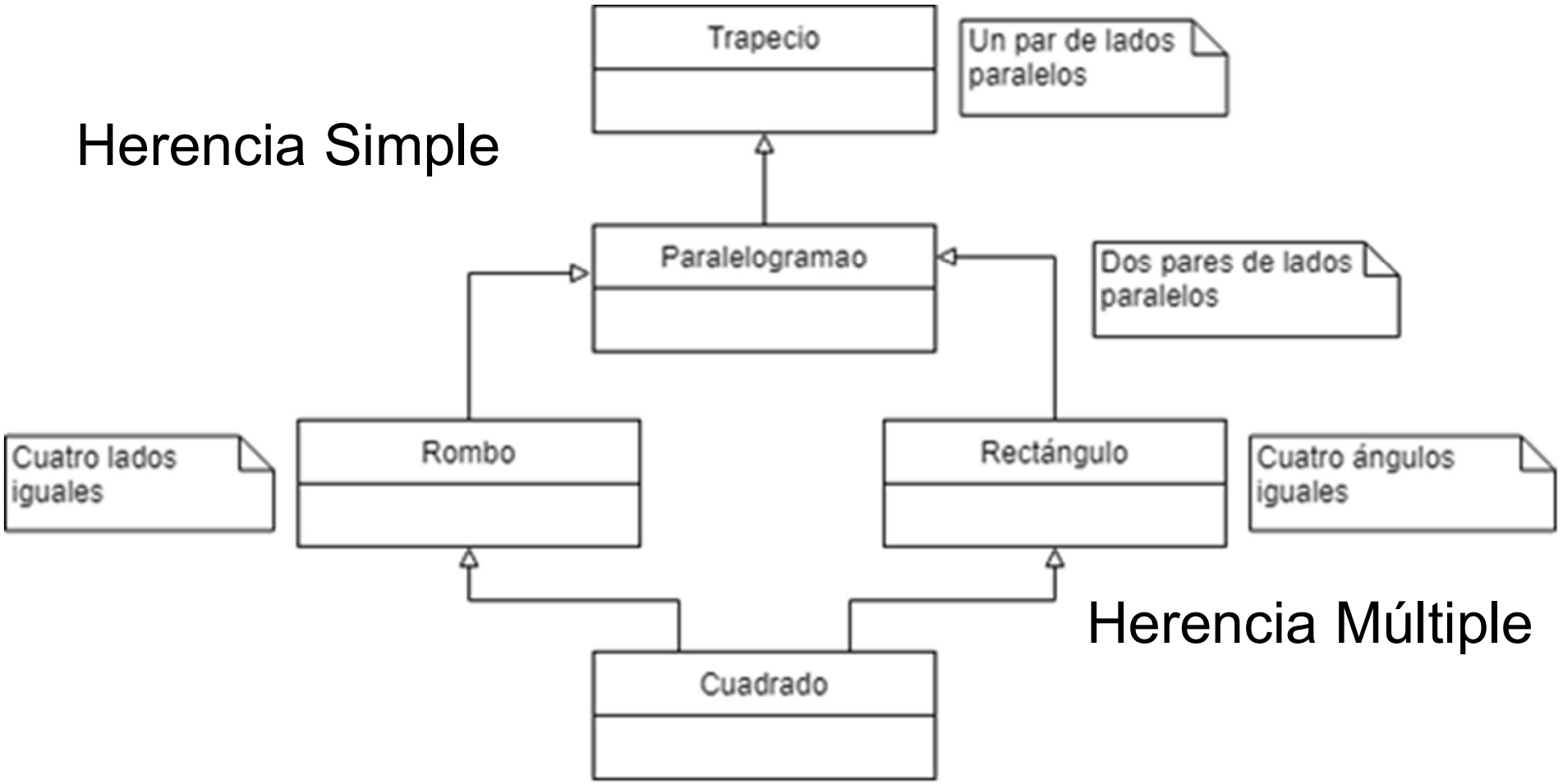
E.mostrarDatos()?

Herencia


Impresión
-razon_social=" Supermercado EL Torito "
-cuit="12-6874158-1"
variables de clase
ImpresiónFactura
-nro_Factura
-nomCliente
-.....
+ emitirfactura()
+ .....
ImpresiónRecibo
.....
ImpresiónNotaCredito
variables de instancia

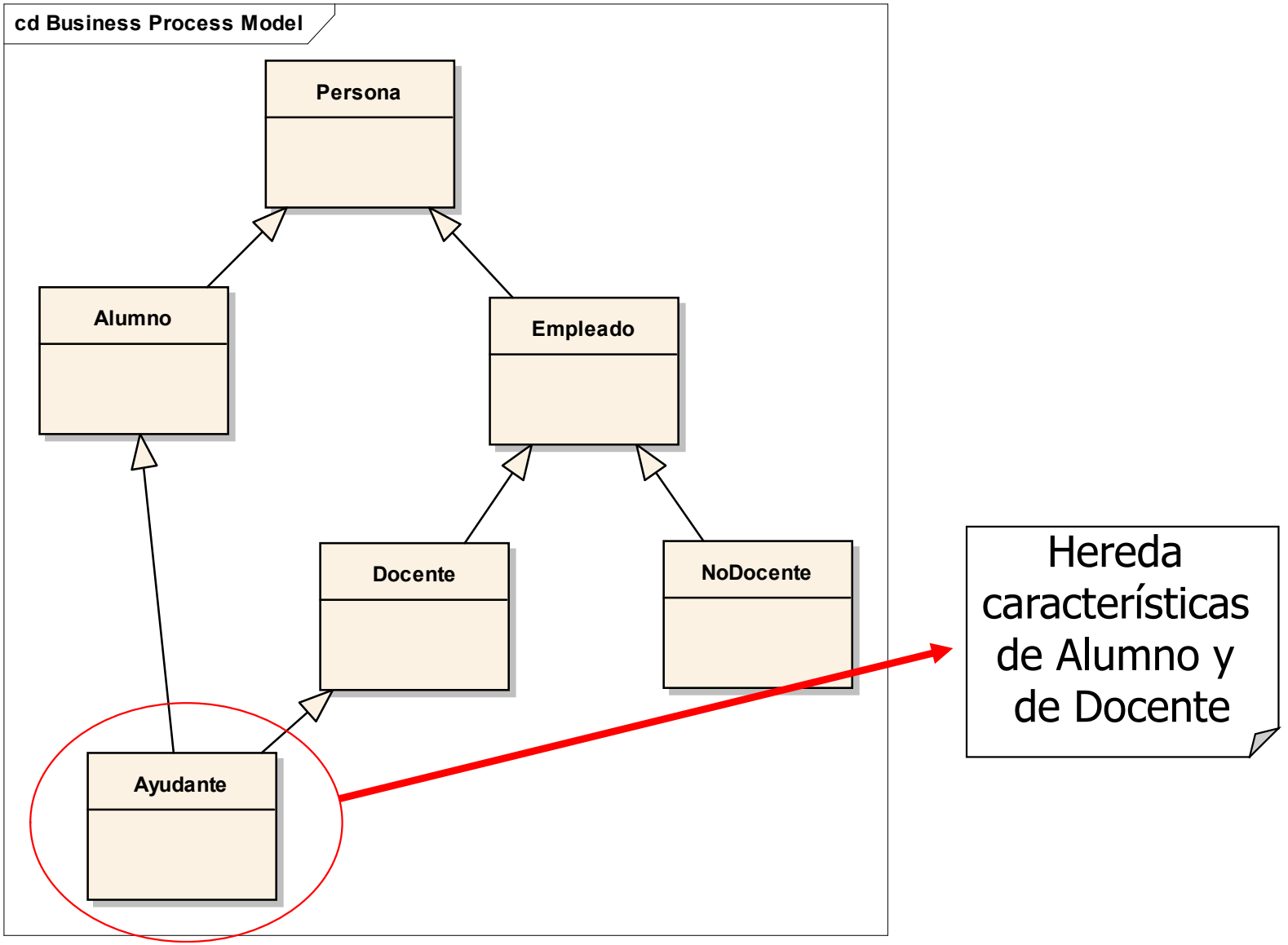
Herencia – Tipos de herencia

Herencia Simple

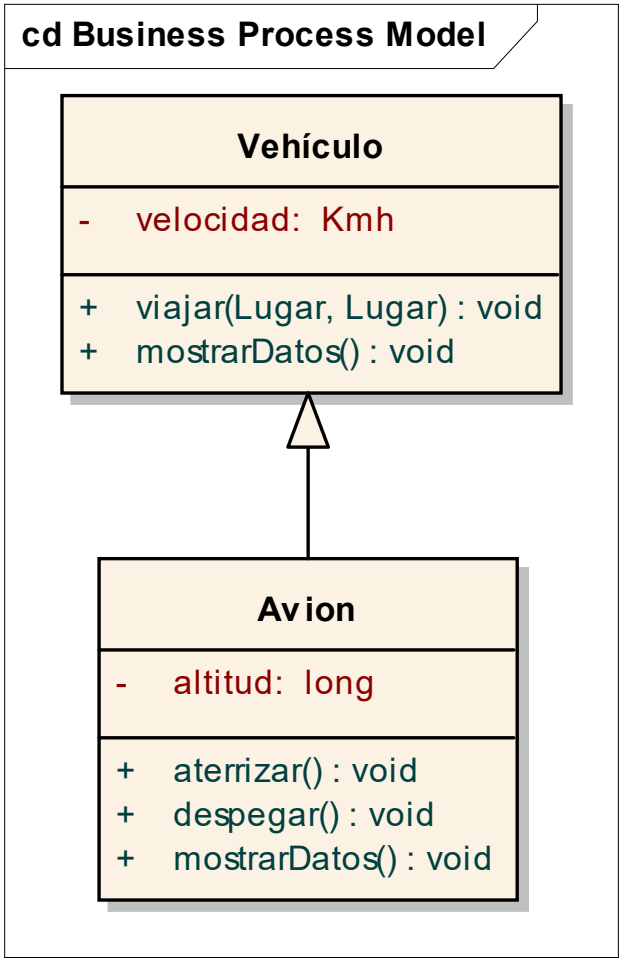


Herencia Múltiple

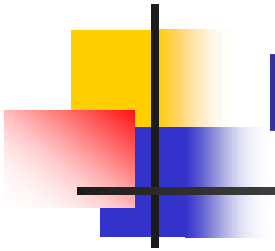
Ejemplo Herencia Múltiple



Herencia



Conclusión: podemos decir que, dentro de una jerarquía de clases, la herencia propaga las características de la clase superior en sus clases descendientes, de modo que varias clases pueden compartir una misma descripción.



Relaciones entre clases

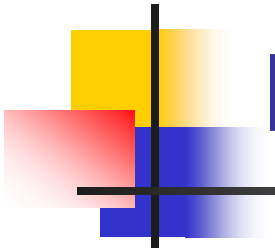
En la primera etapa del DOO, se identifican las clases.

Aunque algunas clases pueden existir aisladas, la mayoría no puede, y deben cooperar unas con otras.

Las relaciones entre las clases expresan cómo se comunican los objetos de esas clases entre sí.

Las **relaciones muestran el acoplamiento** de las clases.

Acoplamiento entre clases es el número de clases con las que una clase concreta está relacionada (acoplada). Una clase está acoplada si sus objetos lo están. Un objeto está acoplado con otro si uno de ellos actúa sobre el otro.



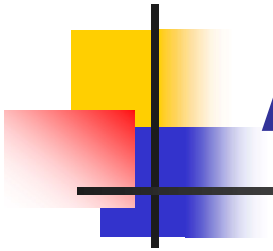
Relaciones entre clases

Según el tipo de acoplamiento que presentan las clases podemos distinguir distintos tipos de relaciones.

Asociación

Agregación

Generalización / Especialización / Herencia (tema ya visto)



Asociación

- Una asociación es una conexión entre dos clases; refleja una conexión que existe en el ámbito de una aplicación.

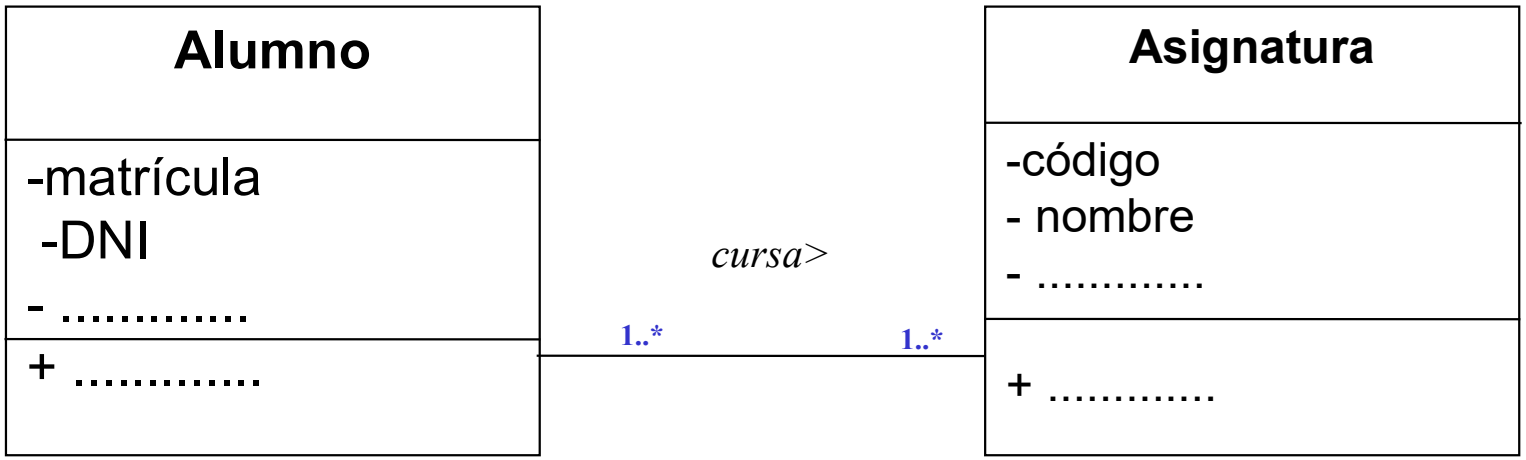
- La comunicación puede ser tanto uni como bi-direccional (por defecto)

-Una asociación puede tener nombre y se representa por una línea continua entre las clases asociadas.

-La asociación no es contenida por las clases, ni subordinada a las clases asociadas.

Asociación

✓Una asociación puede verse como una abstracción de los vínculos que existen entre objetos instancias de las clases asociadas.



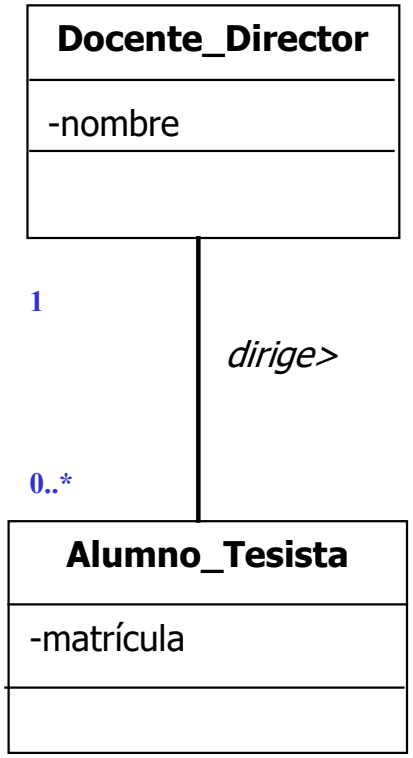
Un alumno está inscripto en una o mas asignaturas - Una asignatura tiene uno o mas inscriptos.

Asociación

En UML, el número de instancias (cardinalidad) que participan en la relación, y se anota en cada extremo de la relación, y se llama multiplicidad.

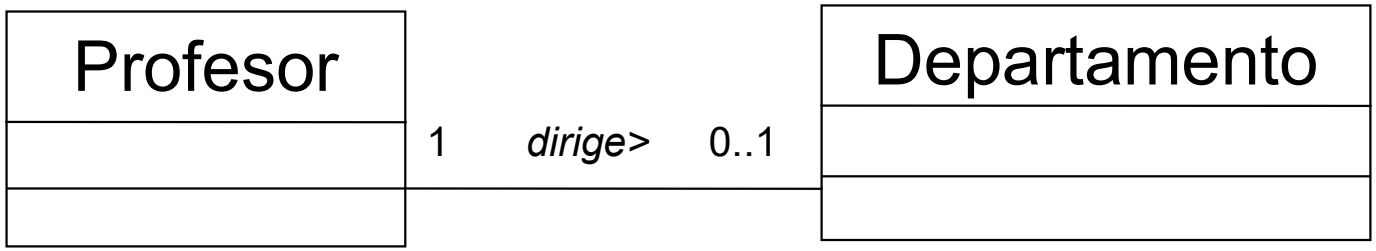
Los valores de multiplicidad más comunes son:

- **1** uno y sólo uno
- **0..1** cero o uno (0 es una relación opcional)
- **m..n** de m a n (m y n enteros naturales)
- ***** de cero a varios
- **0..*** de cero a varios
- **1..*** de uno a varios

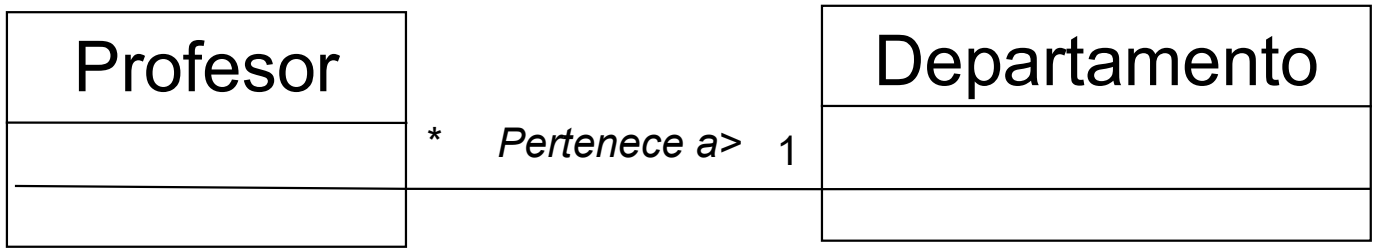


Asociación

Ejemplos:

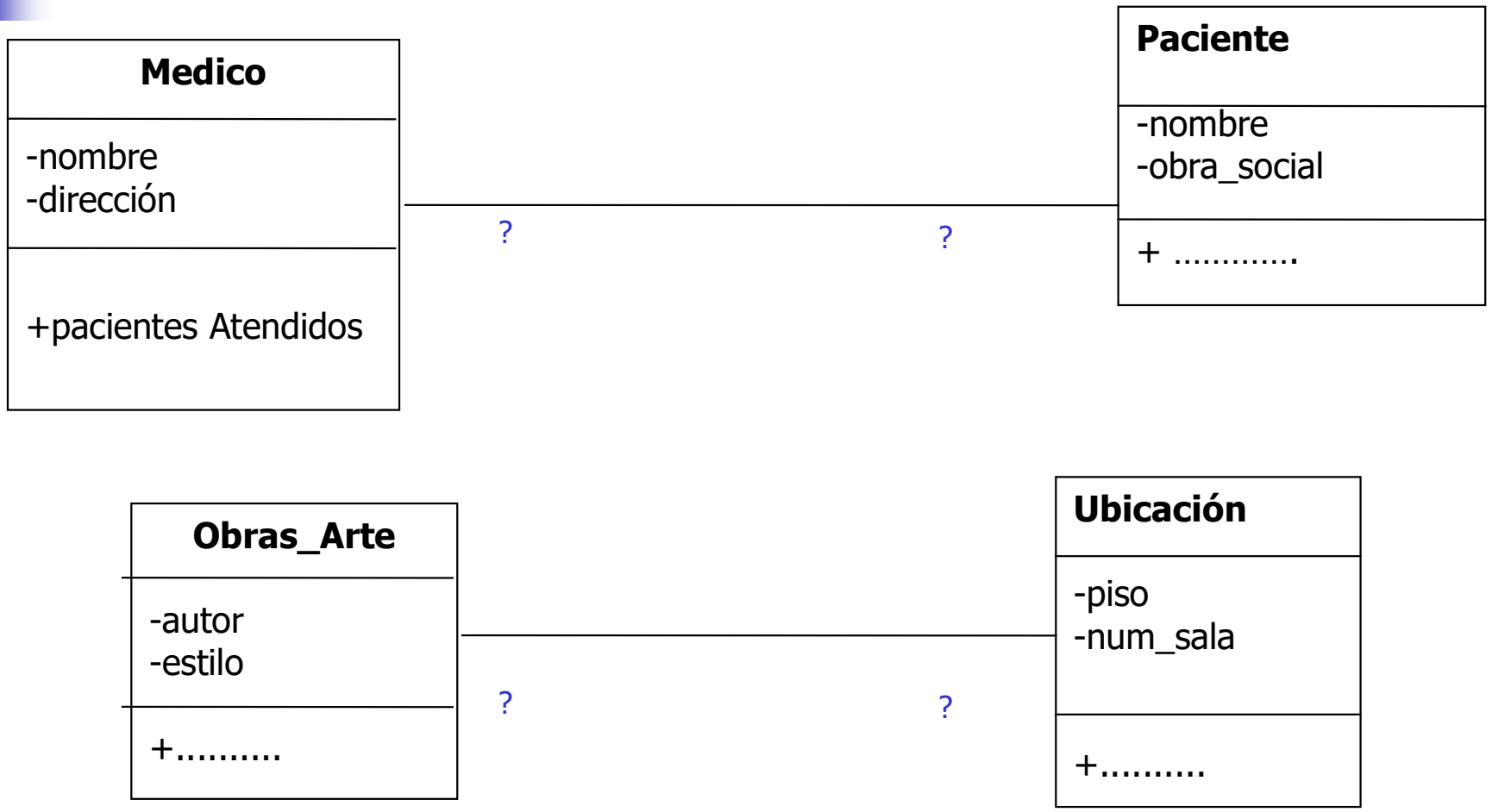


Un departamento tiene un solo profesor que lo dirige y un profesor puede dirigir uno o ningún departamento



A un departamento pertenecen muchos profesores y un profesor pertenece a un solo departamento

Asociación – Actividades Conjuntas



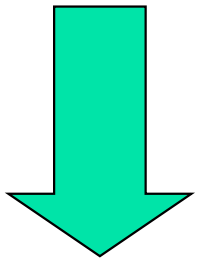
http://www.devx.com/enterprise/Article/28576

Investigue este concepto de asociación en este sitio

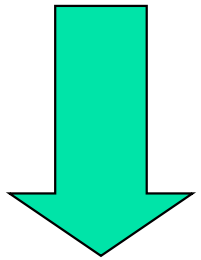
Asociación

Existen situaciones especiales en las que es necesaria agregar información que es propia de la relación y no de las clases.

Para esto se trabaja con dos conceptos distintos según sea el caso:



Clase asociación



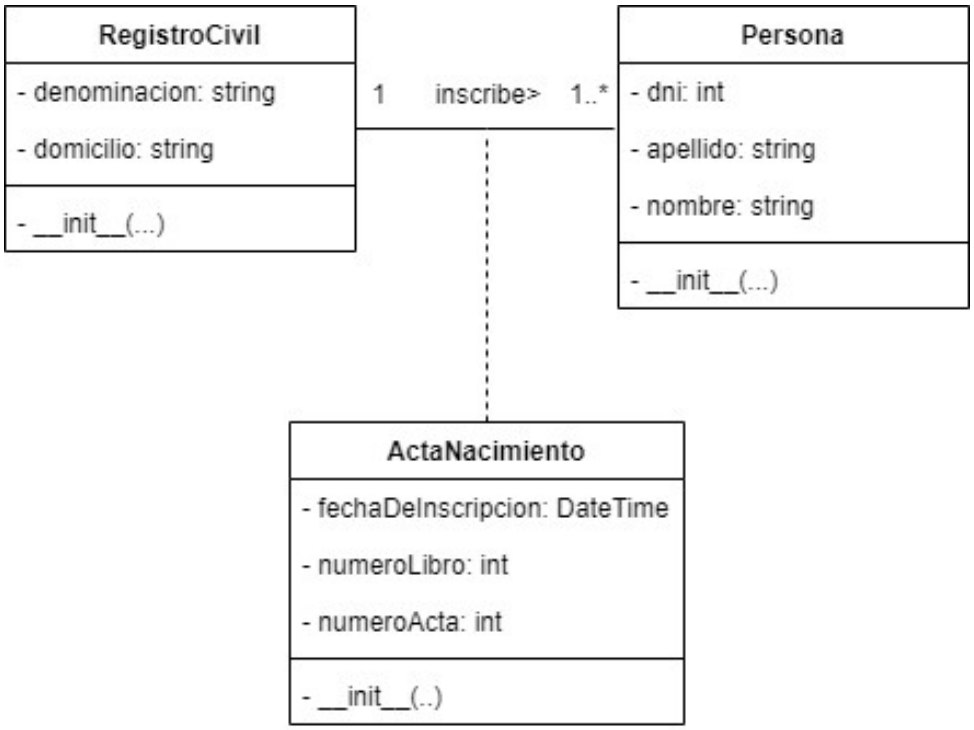
Clase que modela la asociación

http://www.devx.com/enterprise/Article/28576

Investigue este concepto de asociación en este sitio

Asociación – Clase Asociación (1)

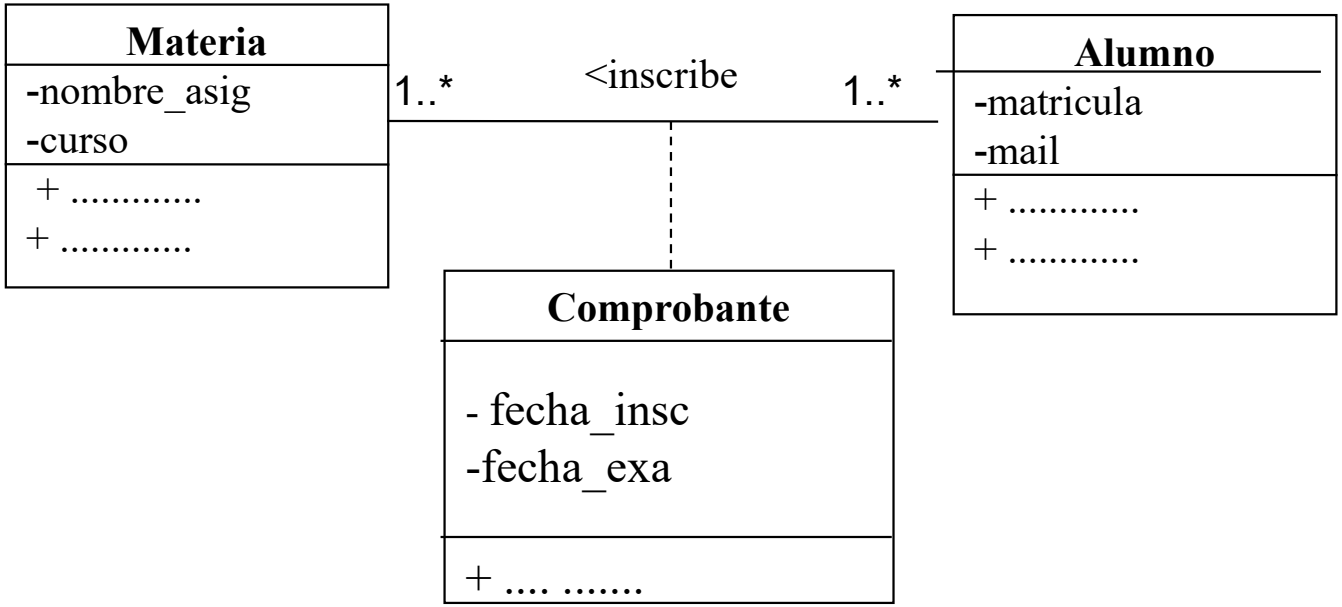
Contexto: Supongamos un sistema que almacena información sobre los registros civiles de la provincia de San Juan, y las personas nacidas en la jurisdicción del mismo. Los nacimientos se registran en un acta de nacimiento, que posee entre otros datos: número de acta, fecha de inscripción, número de libro, etc. Como se puede observar en este ejemplo, si bien, hay una asociación entre Registro Civil y Persona, los datos propios del acta de nacimiento, no son datos inherentes al Registro Civil, ni a la Persona, esta información debe ser almacenada como atributos de una **clase asociación**.



Principio Fundamental:
Una dupla de objetos, instancias de cada una de las clases que participan en la asociación, se relaciona con una única instancia de la clase asociación. No importa la multiplicidad en ambos extremo

Asociación – Clase Asociación (2)

Supongamos un sistema que contempla las **inscripciones a exámenes de turno del mes de julio (CONTEXTO ACOTADO SOLO A UN TURNO)**, donde cada alumno puede inscribirse en una o más asignaturas, y por cada inscripción recibe un comprobante.



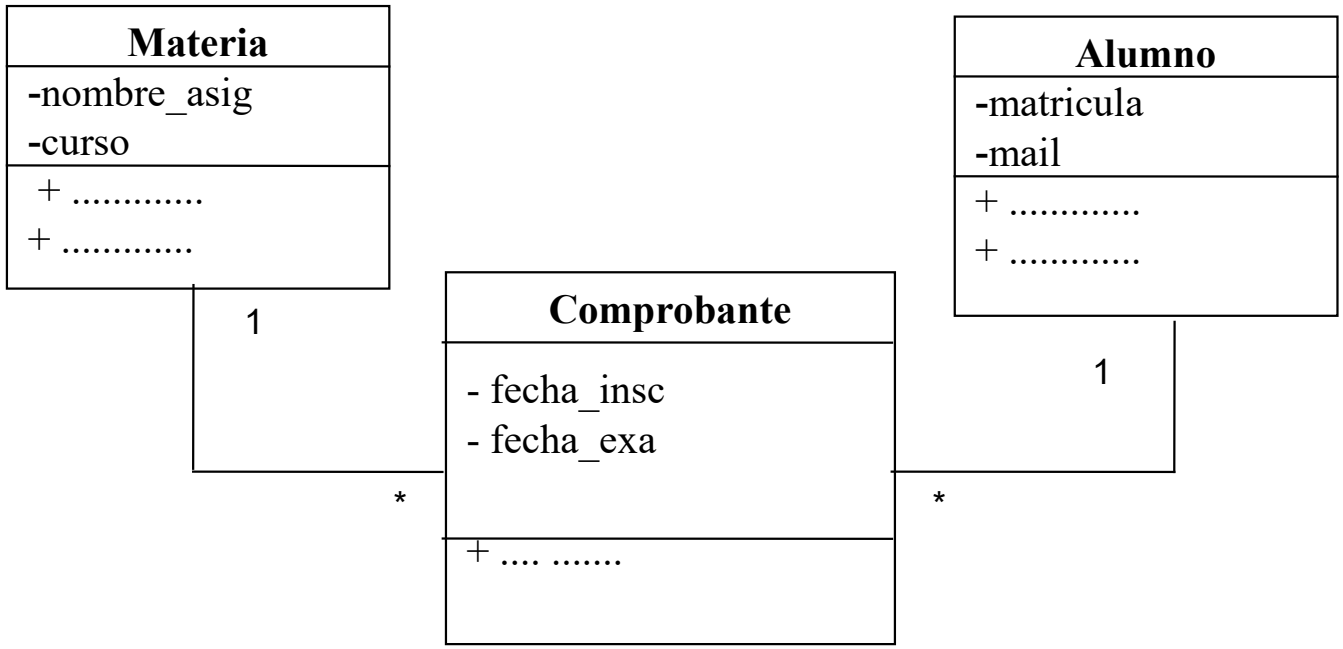
?

Que pasa si contemplo todas las mesas de julio o las mesas de todo el año

Comprobante es una clase Asociación: Para una dupla de objetos (materia, alumno) existe un único objeto asociado a la clase comprobante.

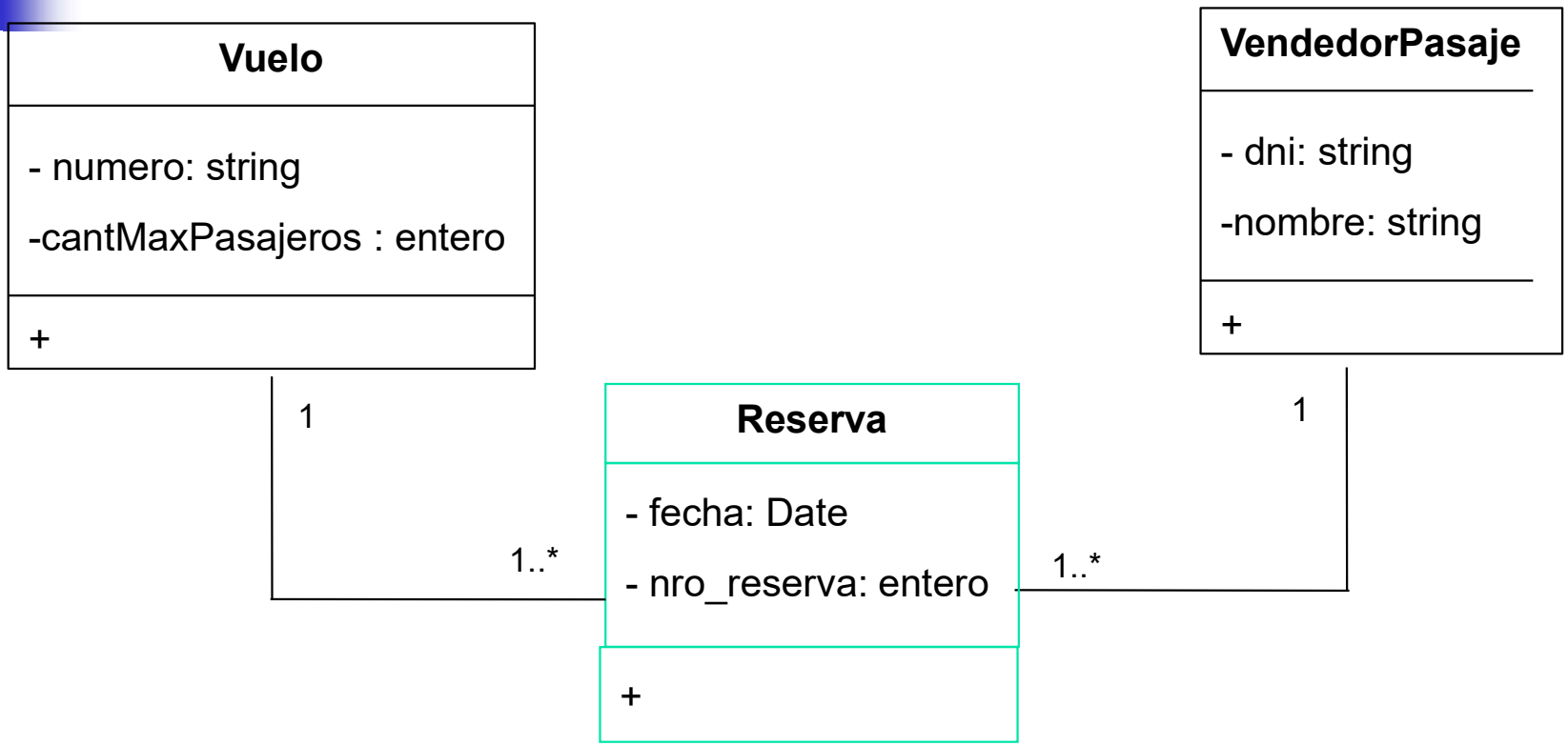
Asociación – Clase que modela la asociación

Supongamos un sistema que contempla las **inscripciones a exámenes de los distintos turnos de examen** (cada alumno puede inscribirse en una o más asignaturas, y por cada inscripción recibe un comprobante)

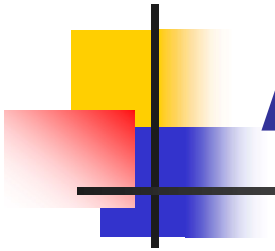


Comprobante es la clase que Modela la asociación entre Materia y Alumno: Para una dupla de objetos (materia, alumno) existe más de objeto asociado a la clase comprobante.

Asociación – Clase que modela la asociación



Para una instancia de **Vuelo** y una instancia de **VendedorPasaje**, existen más de un objeto de la clase **Reserva**. Es decir, podemos tener múltiples reservas por vuelo y vendedor. En este caso no podemos definir a **Reserva** como una clase asociación, sino que existe **una relación de asociación entre Vuelo y VendedorPasaje que se deriva o modela a través de Reserva.**



Agregación

Reutilización y extensión:

Se denomina **reutilización** al uso de clases u objetos desarrollados y probados en un determinado contexto, para incorporar esa funcionalidad en una aplicación diferente a la de origen.
La forma más simple de reutilizar una clase es simplemente haciendo una nueva clase que la contenga. Esto es posible a través de la **composición**.

La **extensión** se basa en aprovechar las clases desarrolladas para una aplicación, utilizándolas para la construcción de nuevas clases, en la misma u otra aplicación.

Como la extensión es una forma particular de reutilización, se suele englobar bajo el concepto común de reutilización.

Un caso típico de extensión se logra a través de la **herencia**, como ya se ha visto anteriormente. A partir de una clase desarrollada pueden generarse una o más subclases que posean características particulares, que heredan atributos y comportamientos de la clase padre.

Agregación

- La agregación consiste en definir como atributos de una clase a objetos de otras clases ya definidas.

- La agregación es una relación *no simétrica* (todo/parte) en la que una de las clases cumple un papel predominante respecto de la otra.

- La agregación declara una dirección a la relación **todo/parte.**

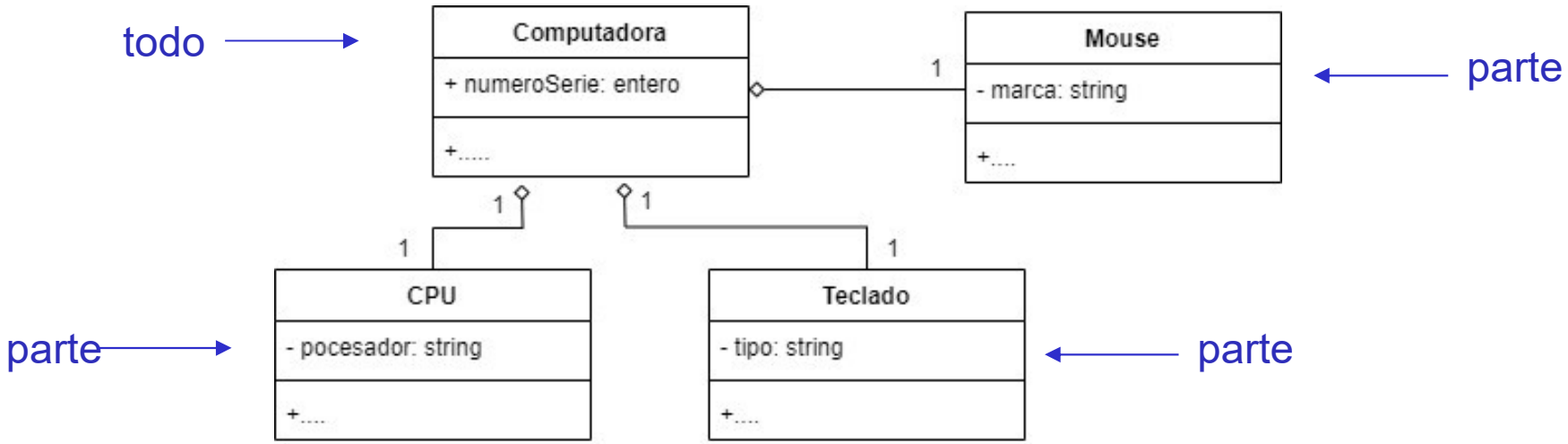
- Gráficamente se representa colocando un rombo del lado de la clase agregado.

Agregación Composición

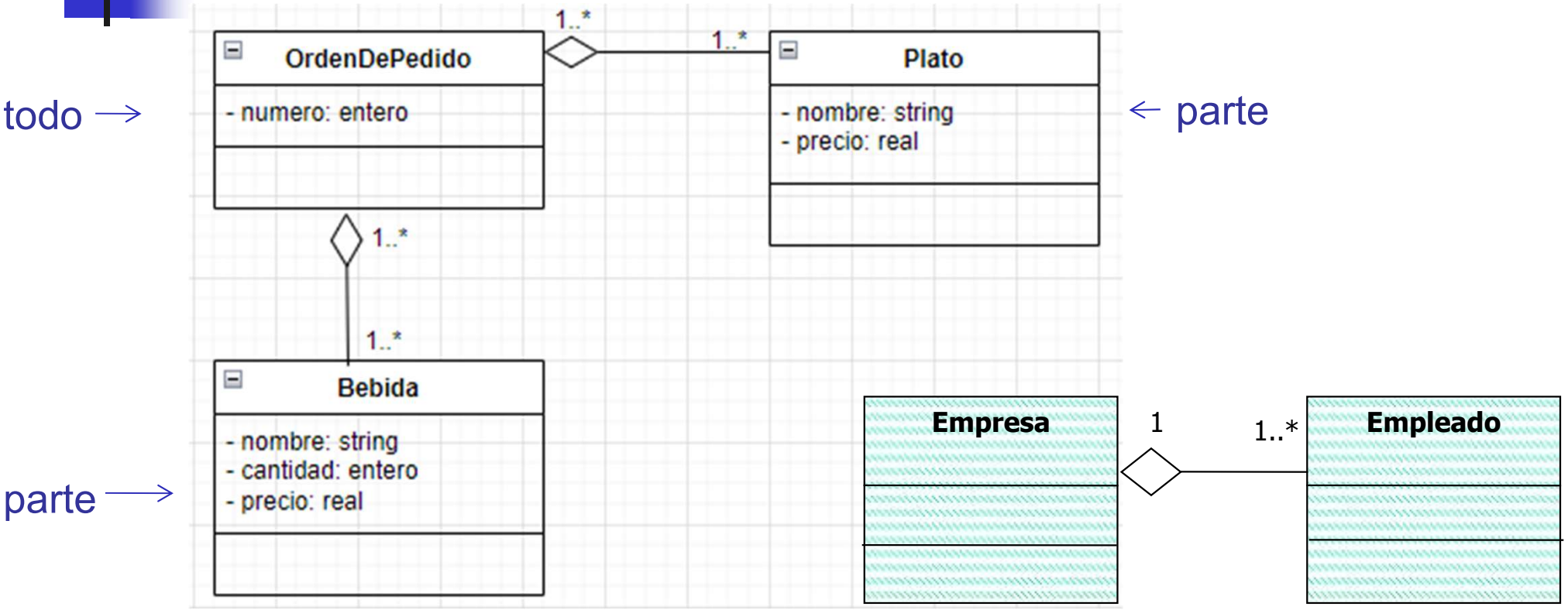
Agregación (propiamente dicha 1)

- Este tipo de relación se presenta en aplicaciones en las cuales un objeto contiene como partes a objetos de otras clases, pero de tal modo que la destrucción del objeto continente no implica la destrucción de sus partes. Esto es, el objeto continente o contenedor **incluye referencias a objetos de otras clases** y que los objetos contenidos pueden existir independientemente del objeto contenedor.

- Los tiempos de vida de los objetos continente y contenido no están tan estrechamente acoplados, de modo que se pueden crear y destruir instancias de cada clase independientemente.



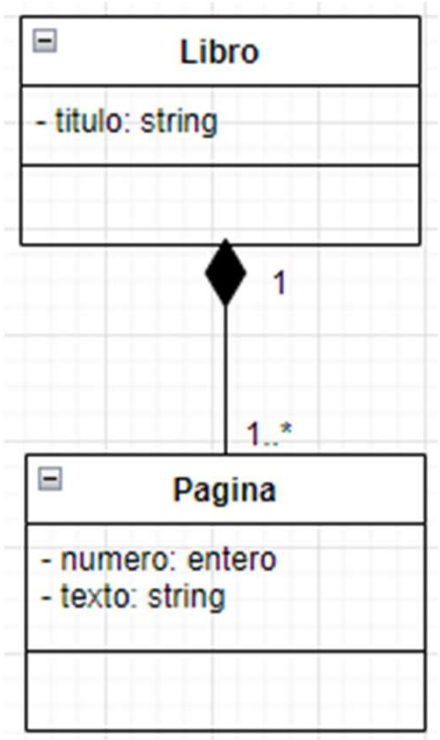
Agregación (propiamente dicha 2)



En el caso de las agregaciones propiamente dichas, no hay restricciones en la multiplicidad del agregado u objeto continente.

Agregación (Composición)

La forma más simple de **reutilizar** una clase es simplemente haciendo una **nueva clase que la contenga.** Esta técnica se llama **composición.**



- Un objeto de una clase contiene como partes a objetos de otras clases y **estas partes están físicamente contenidas por el agregado**.

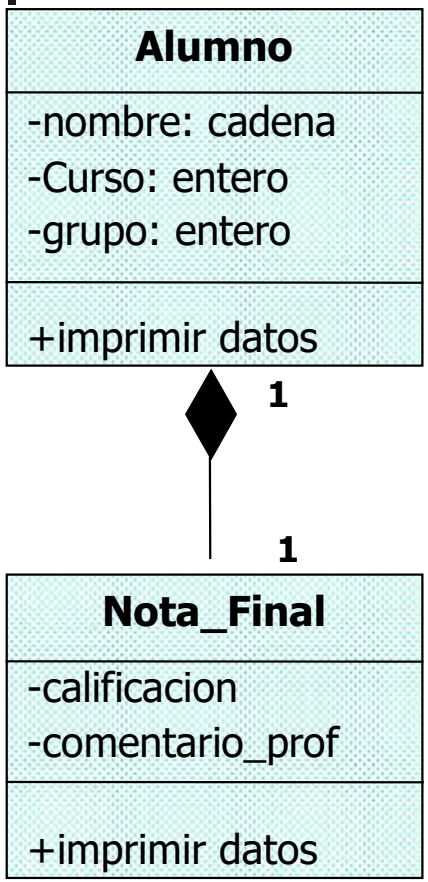
- Los objetos agregados no tienen sentido fuera del objeto resultante.

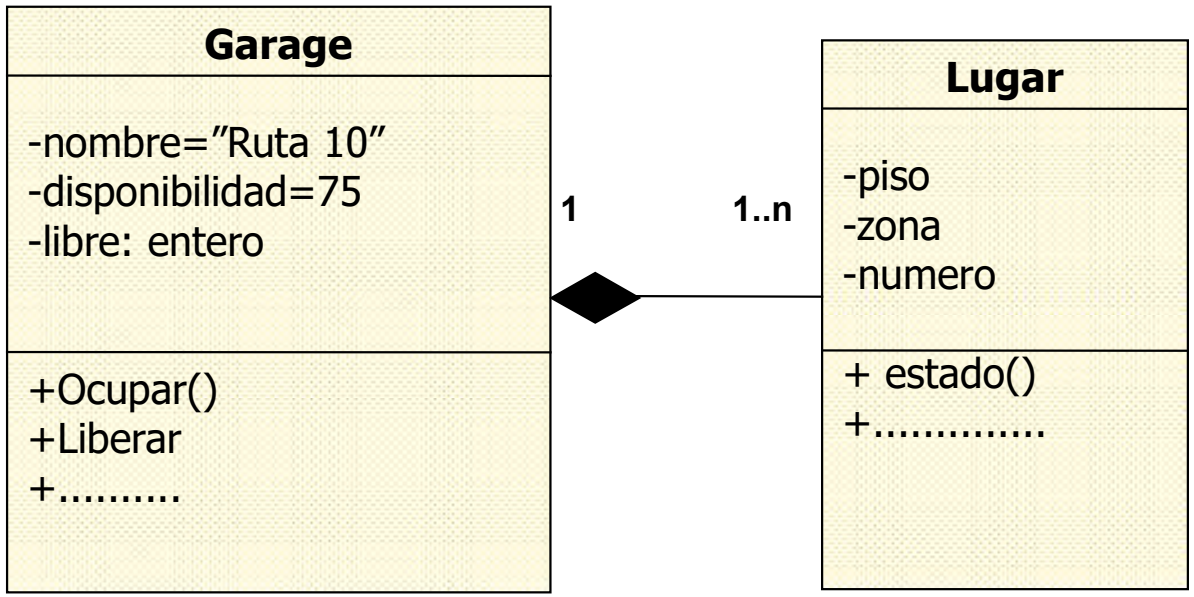
- Los tiempos de vida de los objetos continente y contenido están estrechamente acoplados

- La destrucción del objeto continente implica la destrucción de sus partes

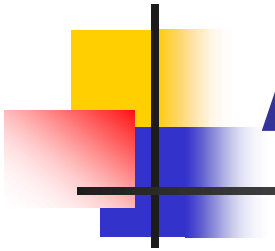
Las composiciones generan una **relación de existencia** entre el todo y cada una de sus partes.

Agregación (Composición)





En el caso de la composición, la multiplicidad del lado agregado puede tomar el valor 0 o 1. El valor 0 se refiere a un atributo no explicitado.



Agregación (Síntesis)

La agregación se utiliza para modelar la relación **todo-parte** (continente-contenido) y se describe mediante la relación **tiene-un** o **parte-de**.

Algunos criterios para detectar agregaciones[1]:

- Los objetos de una clase están subordinados a los objetos de otra clase.
- Una acción sobre objetos de una clase implica una acción sobre objetos de otra clase.
- Los valores de los atributos de un objeto de una clase se propagan en los valores de los atributos de otro objeto de otra clase.

---

[1] Modelado de Objetos. Modelado de objetos con UML / Pierre-Alain Muller . Pág103

Diagrama de Clases

Un diagrama de clases es un diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos.

Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas.

Diagrama de Clases

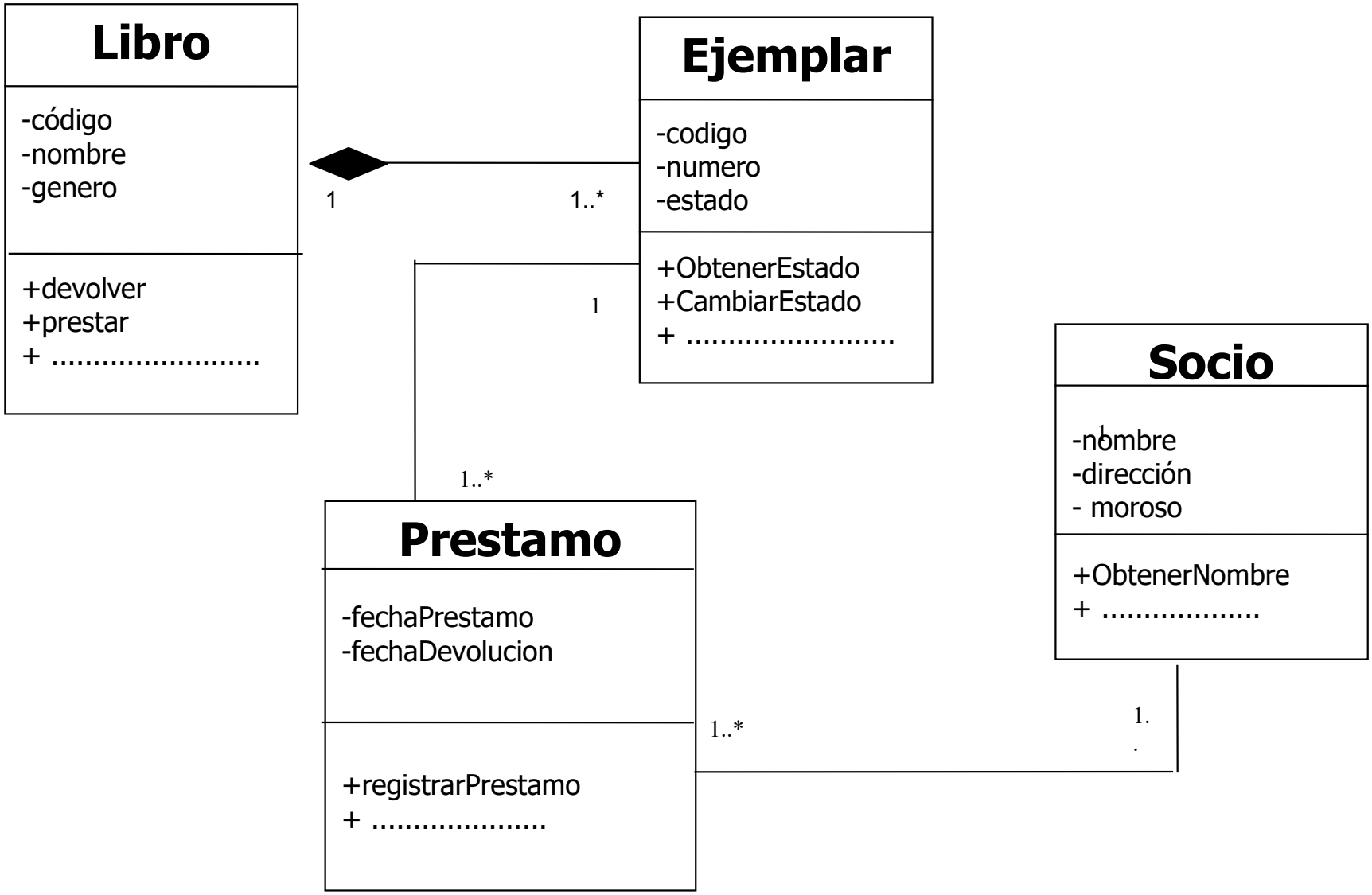


Diagrama de Clases

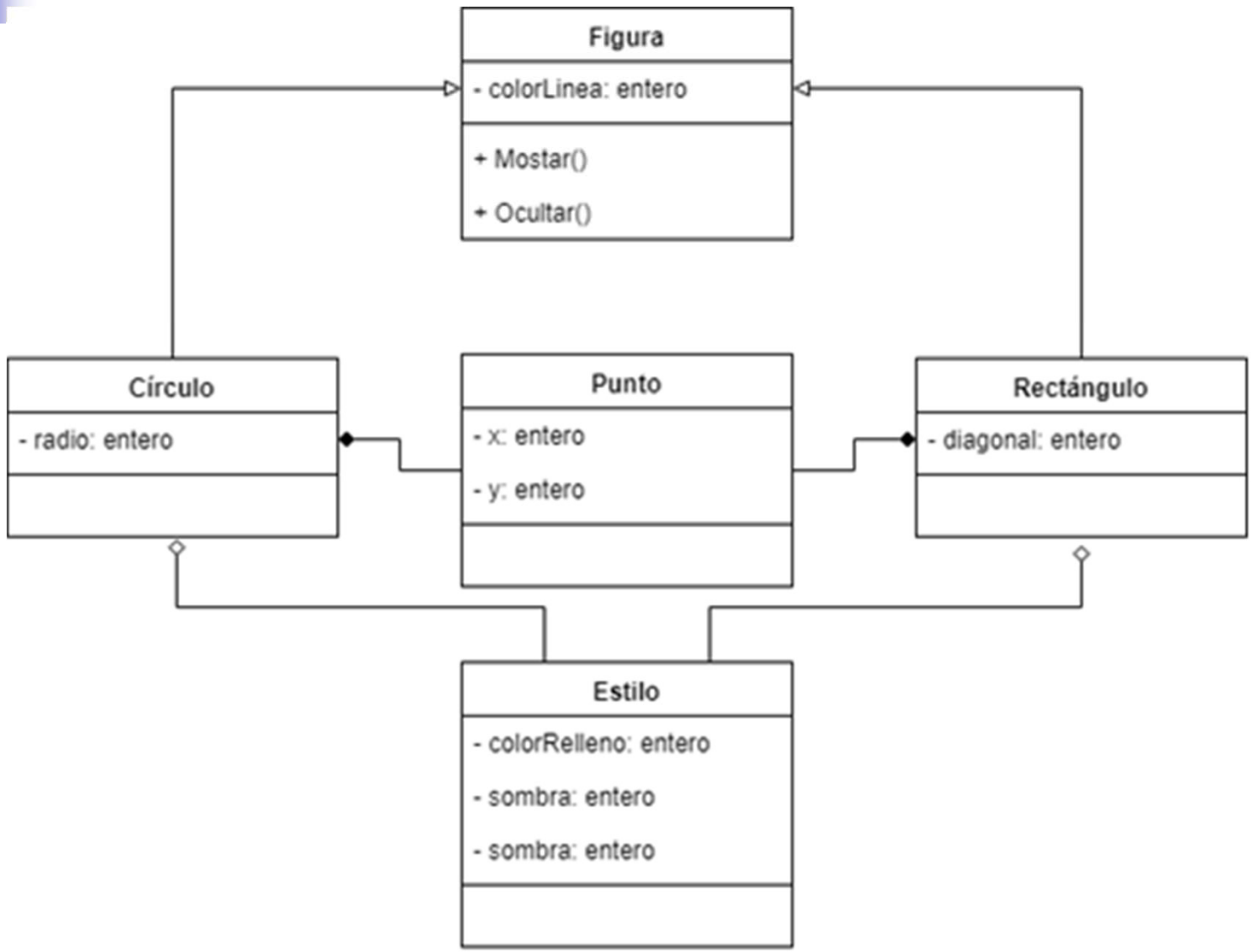
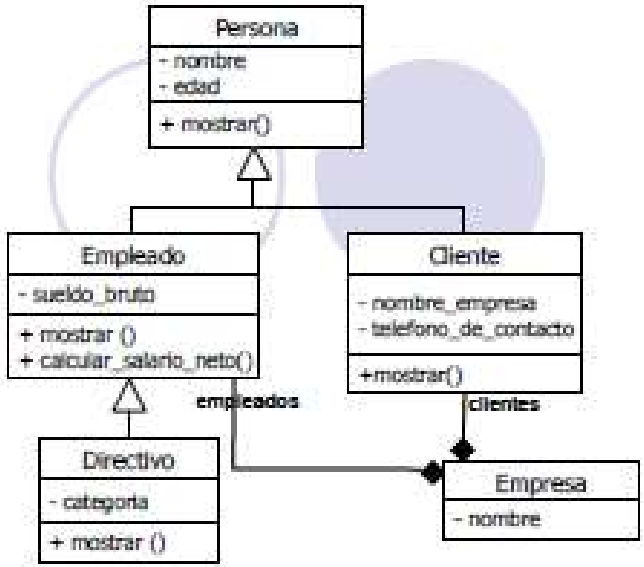


Diagrama de Clases



Objetos

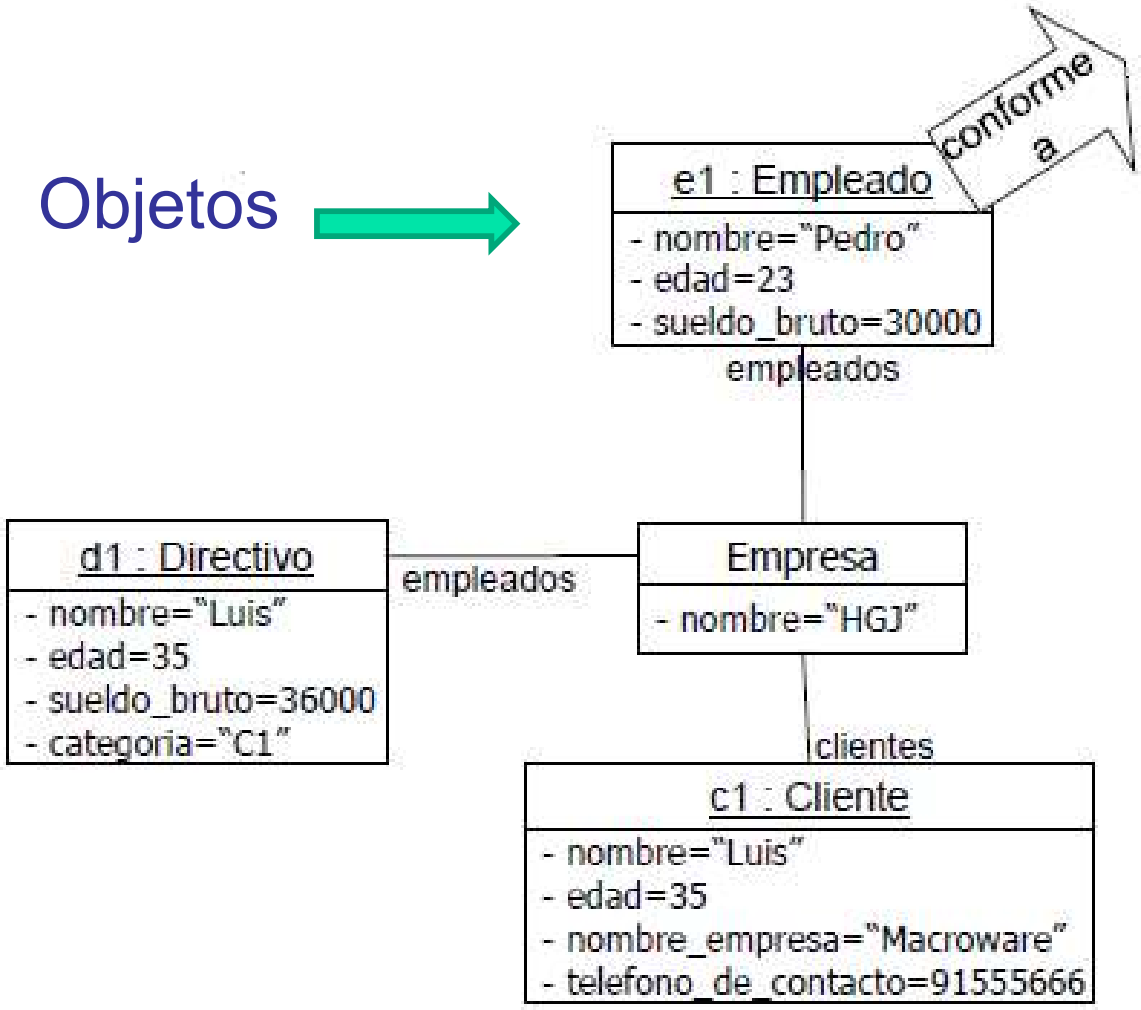
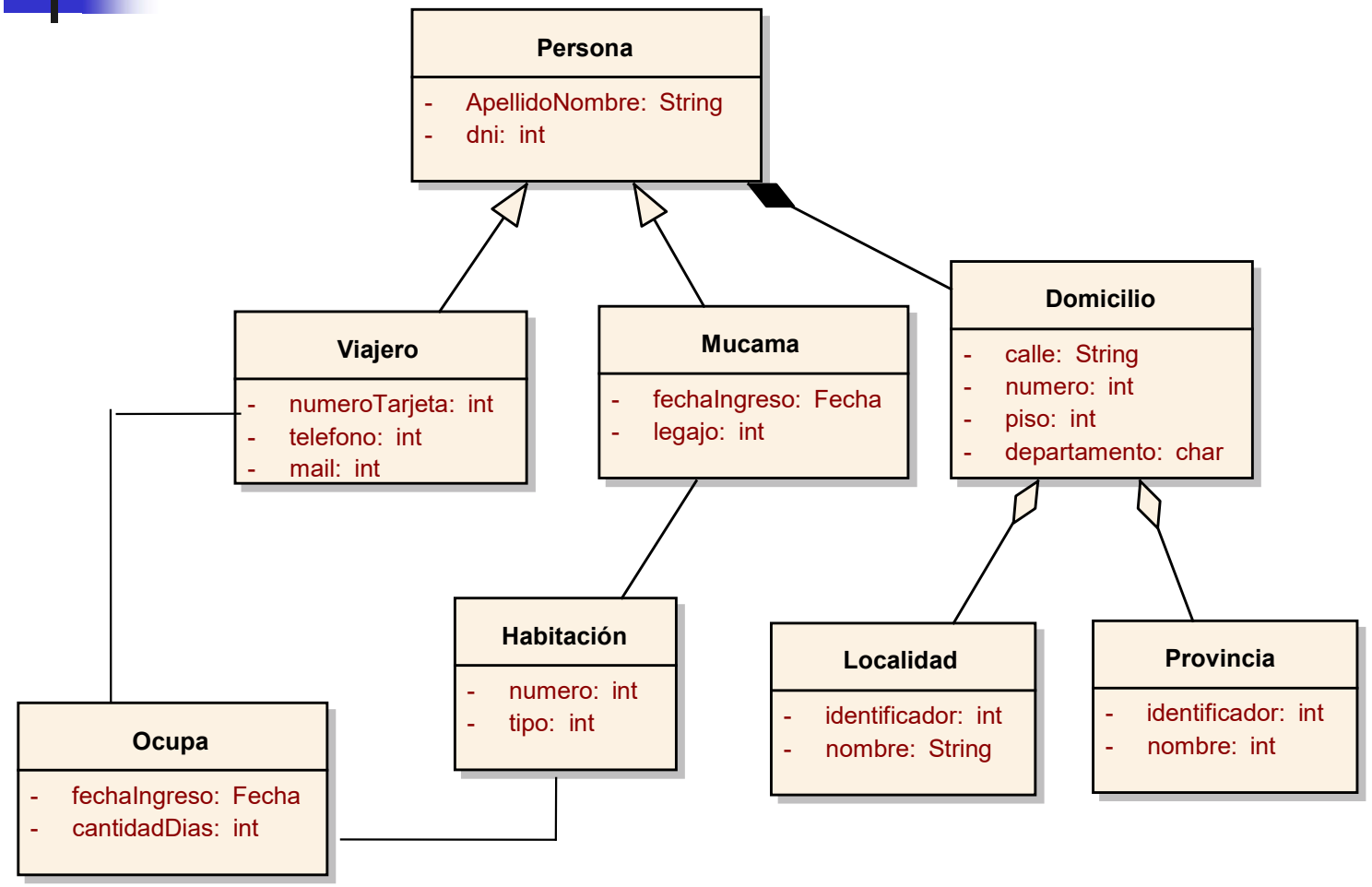
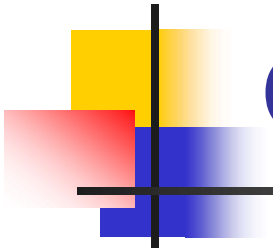


Diagrama de Clases (ACTIVIDAD a consultar)

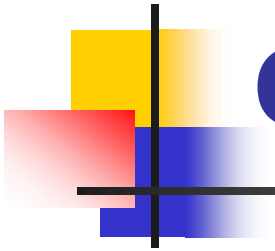


Identifique el **tipo de relaciones existente**s entre las clases y la **cardinalidad** de las mismas.



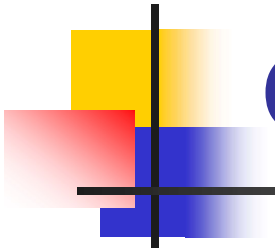
Conceptos Clave (1)

- **Encapsulamiento**
  - Término formal que describe al conjunto de métodos y de datos de un objeto de manera tal que el acceso a los datos se permite solamente a través de los métodos propios de la clase a la que pertenece el objeto.
  - La comunicación entre los distintos objetos se realiza solamente a través de mensajes explícitos.
- **Abstracción**
  - La orientación a objetos fomenta que los programadores y usuarios piensen en las aplicaciones en términos abstractos.
  - A partir de un conjunto de objetos, se piensa en los comportamientos comunes de los mismos para situarlos en superclases, las cuales constituyen un depósito para elementos comunes y reutilizables



Conceptos Clave (2)

- **Polimorfismo**
  - Capacidad que tienen objetos de clases diferentes, relacionados mediante la herencia, a responder de forma distinta a una misma llamada de un método.
  - Fomenta la extensibilidad del software
    - Software escrito para invocar comportamiento polimórfico se escribe en forma independiente del tipo de los objetos a los cuales los mensajes son enviados
    - Nuevos tipos de objetos, que pudieran responder a mensajes existentes, pueden ser agregados en dicho sistema sin modificar el sistema base.



Conceptos Clave (3)

- **Persistencia**
  - designa la capacidad de un objeto de trascender el tiempo o el espacio
  - un objeto persistente conserva su estado en un sistema de almacenamiento permanente (pasivación del objeto)
  - el objeto puede ser reconstruido (activación del objeto) por otro proceso y se comportará exactamente como en el proceso inicial