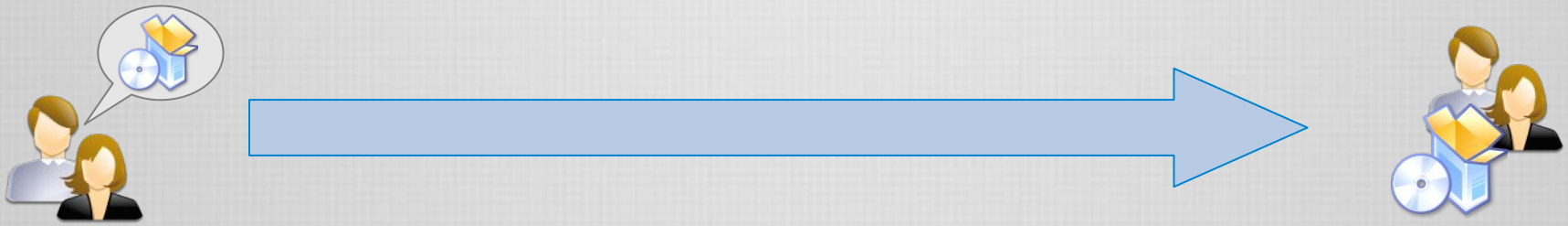
Tecnología de Programación

Martín L. Larrea

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

Proceso de desarrollo de software



Hay varias tareas a realizar en este proceso:

Descubrir qué quiere el cliente

Descubrir cómo lo quiere

Idear cómo debería ser el producto final..

... minimizando los costos

... permitiendo modificaciones inesperadas

... pensando en la reutilización, presente y futura

... y aún así, buscando la simpleza.

El camino hacia la excelencia y la calidad siempre es un camino planificado, nunca improvisado

¿Para qué modelar?

Para proveer una estructura para la solución del problema.

Para proveer las abstracciones necesarias para lidiar con la complejidad.

Para reducir los costos de desarrollo.

Para minimizar el riesgo de cometer errores.

Modelar es simplificar la realidad
Modelamos para comprender lo que construimos

Principios de modelado

El modelado es **esencial** en las disciplinas ingenieriles.
Es importante conocer cuatro principios de modelado: [Booch et al]

La elección de qué modelos crear influye en
cómo el problema es atacado y cómo la solución toma forma.
En física cuántica o astronómica, podemos elegir diferentes modelos matemáticos.
En software, podemos elegir un modelo orientado a objetos, o un modelo imperativo.

Todo modelo puede ser expresado en diferentes niveles de precisión.
La abstracción es importante
En software podemos mostrar el diagrama general del sistema,
o el listado completo de todos los algoritmos usados

Los mejores modelos están conectados a la realidad.
Imaginemos un avión cuyo modelo matemático sólo contempla climas ideales,
o un modelo estructural de un barco que no contempla el peso de lo que transportaría.

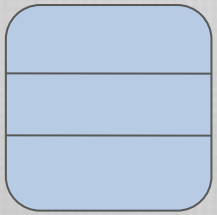
Un sólo modelo no es suficiente.
¿Un solo plano para construir un edificio de 100 pisos?

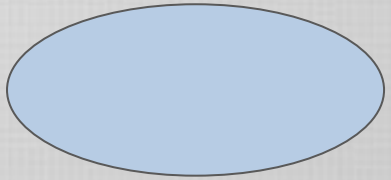
Modelos gráficos

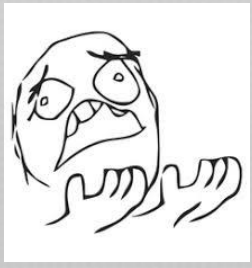
Si bien existen diversos tipos de modelo, los modelos gráficos son especialmente importantes en ingeniería.

El modelado orientado a objetos requiere un buen uso de modelos gráficos.
Han existido varios tipos de diagramas para modelar en orientación a objetos, parecidos entre sí, pero con diferencias relevantes.

Clases en diferentes diagramas históricos:







La tendencia era unificar las notaciones...

Lenguaje de modelado unificado

UML (*Unified Modeling Language*) es un
lenguaje gráfico para el diseño completo de sistemas

Procura abarcar diferentes aspectos del diseño, tanto estáticos como dinámicos, por lo que incluye varios tipos de diagramas.

Es el sucesor de una movida compleja de metodologías OO de los años 80/ 90.
Unifica las más aceptadas de entonces:
el método de Booch, Rumbaugh (OMT), y Jacobson.

Actualmente es un estándar OMG (Object Management Group).

Algunos contribuyentes a UML...

Aonix
Colorado State University
Computer Associates
Concept Five
Data Access
EDS
Enea Data
Hewlett-Packard
IBM
I-Logix
InLine Software
Intellicorp
Kabira Technologies
Klasse Objecten
Lockheed Martin

Microsoft
ObjecTime
Oracle
Ptech
OAO Technology Solutions
Rational Software
Reich
SAP
Softeam
Sterling Software
Sun
Taskon
Telelogic
Unisys

Diagramas de UML

La especificación UML define varios tipos de diagramas.

Básicamente, ocho son los tradicionales desde las primeras versiones:

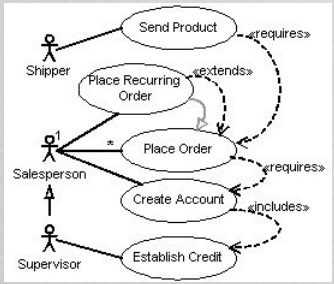
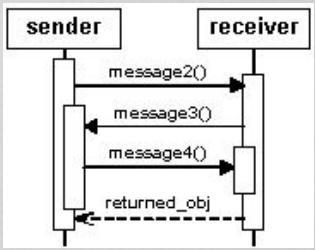


Diagrama de Casos de Uso

Diagramas de Clase y Objetos

Diagrama de Estados

Diagramas de Secuencia



Diagramas de Colaboración

Diagrama de Actividades

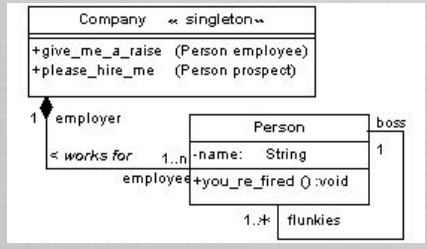


Diagrama de Componentes

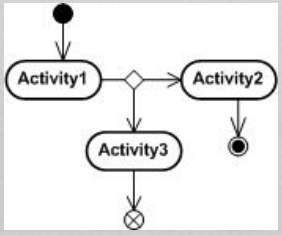
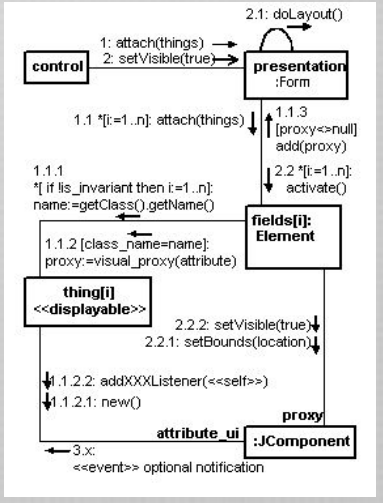


Diagrama de despliegue (deployment)



La versión 2.0 de UML tiene 13 diagramas.

Lleva tiempo aprender todos los diagramas y la semántica asociada.

Veremos únicamente los elementos de UML que son relevantes para el curso.

Clases

Hasta el momento hemos trabajado con
las nociones básicas de orientación a objetos:

Clases

Objetos

Algunas relaciones entre clases (asociación, herencia)

UML tiene diagramas que permiten modelar estos elementos, en diferentes aspectos:

Diagramas de Clase

Diagramas de Objetos

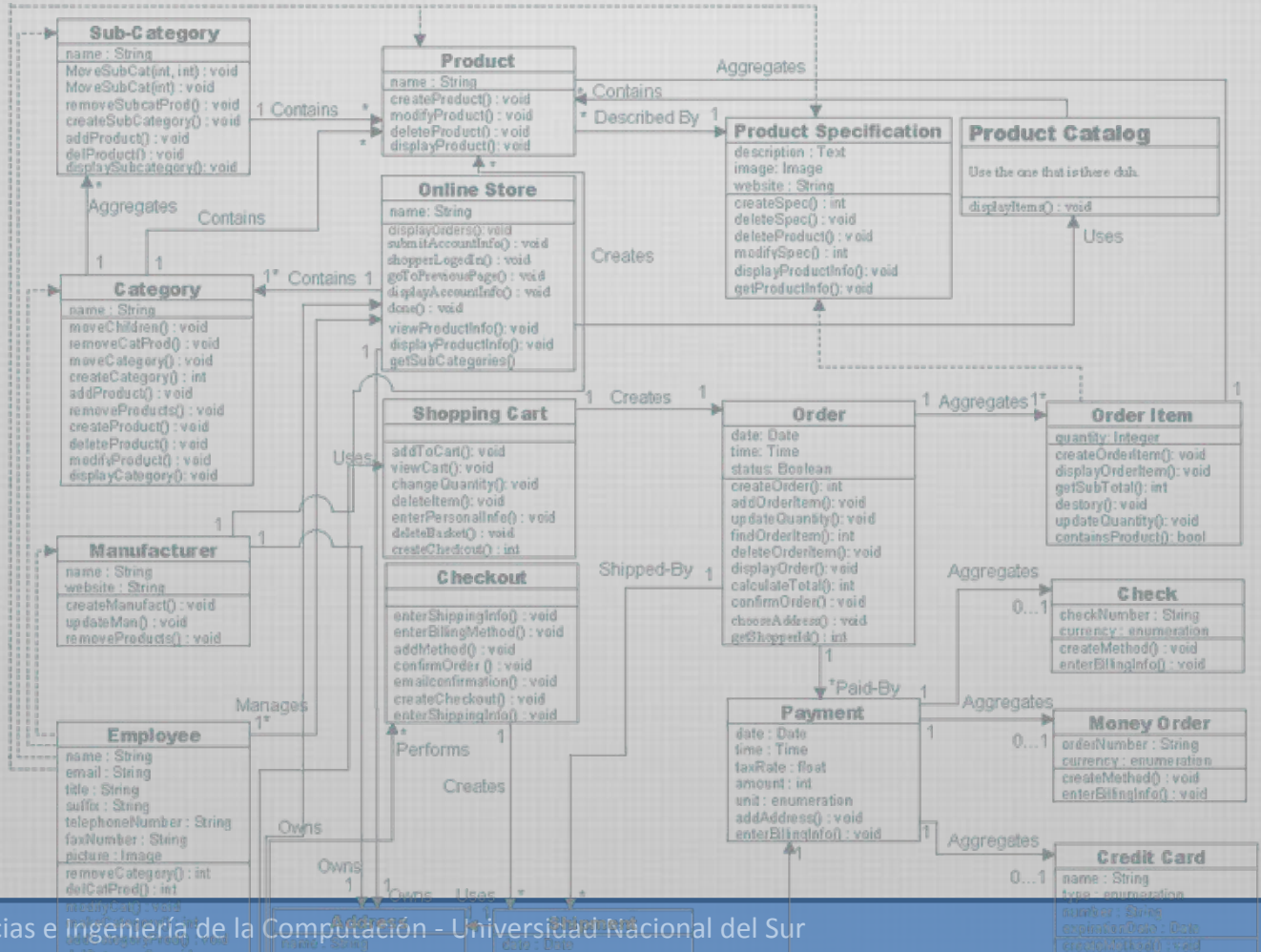
Diagramas de interacción: de Secuencia, de Colaboración

Comencemos a estudiar formalmente cómo nos ayuda UML
a modelar clases, objetos y las relaciones entre estos elementos.

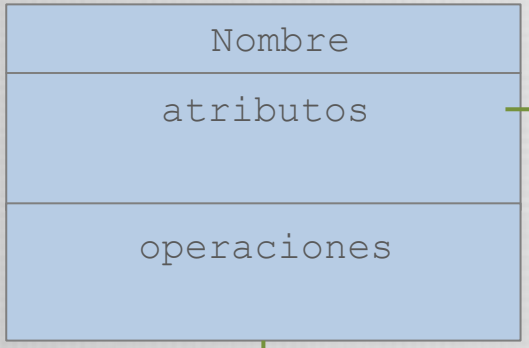
UML - Diagramas de clases

El diagrama de clases es un diagrama de la estructura estática del sistema.

Un diagrama de clases describe los tipos de objetos en el sistema y las dependencias estáticas que existen entre ellos.



UML - Diagramas de clases



Atributos

Sintaxis UML:

<visibilidad><nombre>:<tipo>=<valor por defecto>

La visibilidad puede ser:

- **+** pública
- **#** protegida
- **–** privada

Un atributo subrayado es un atributo de clase.

Operaciones

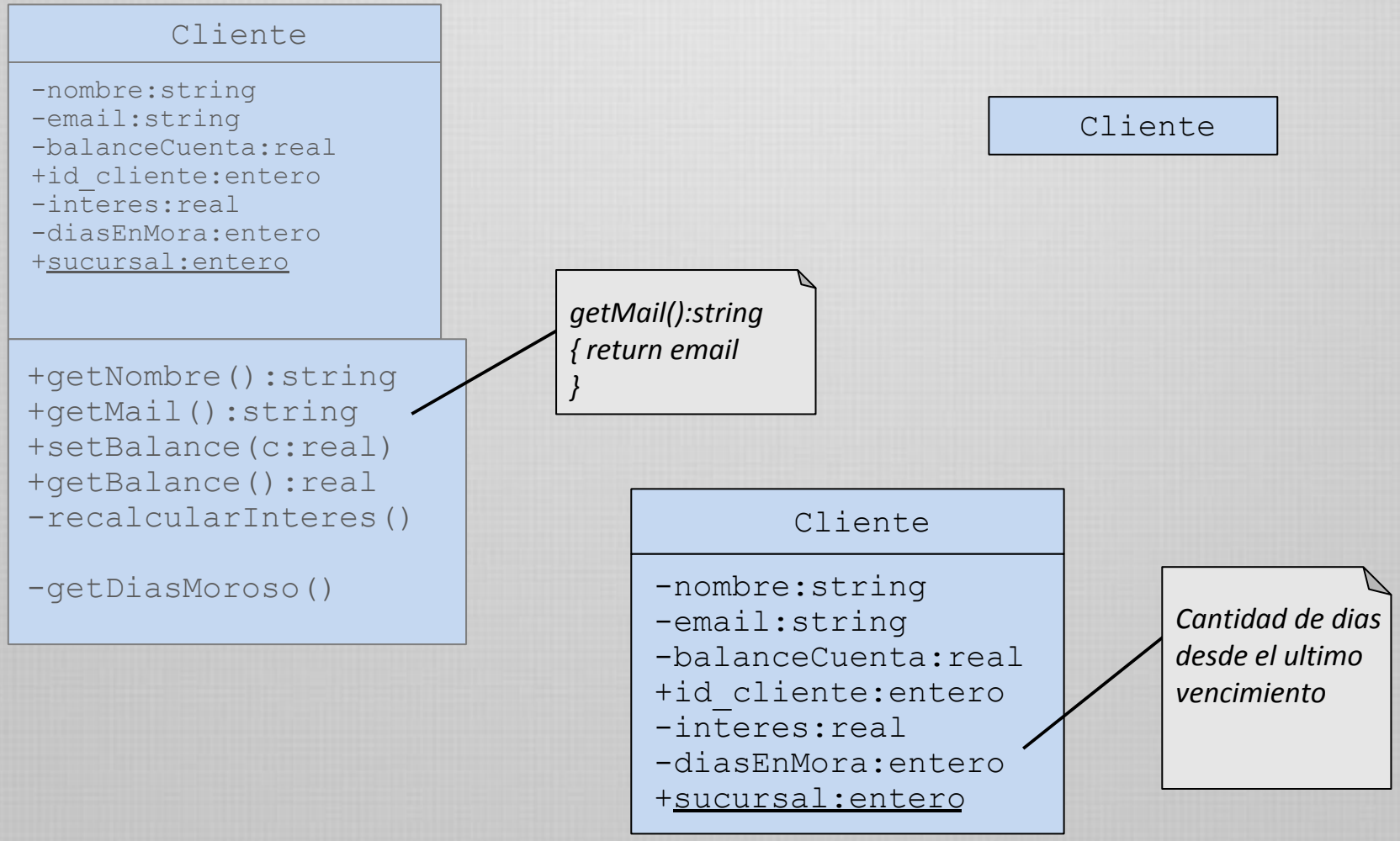
Sintaxis UML:

<visibilidad><nombre>(<lista de parámetros>):<tipo del resultado>

Los tipos de visibilidad son los mismos que para los atributos.
La lista de parámetros tiene la misma sintaxis que los atributos.

Pueden obviarse algunos de estos compartimientos, dependiendo de la vista que se le quiera dar a la clase, según propósitos particulares.

Tres vistas de la misma clase



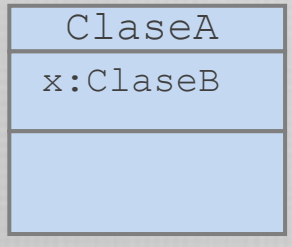
UML - Diagramas de clases

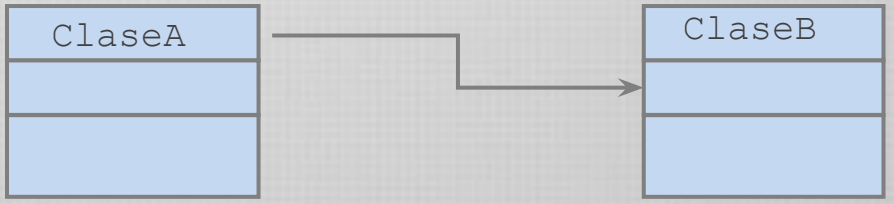
En el mundo real los objetos se relacionan entre sí de alguna forma:

Una persona puede hacer negocios con otra persona.
Un vehículo puede viajar de una ciudad a otra.
Un recibo de compra detalla varios items comprados.
Cada item posee una descripción y una cantidad comprada.

Necesitamos formalizar en el diagrama de clases
estas relaciones que identificamos cuando modelamos

La relación entre clases más común es la asociación:

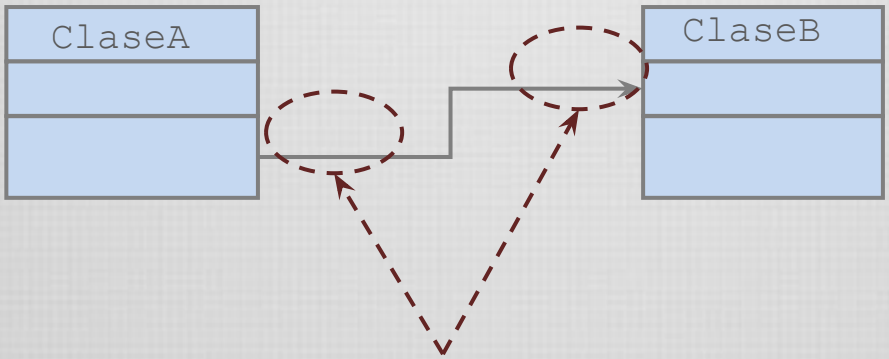




La asociación denota una conexión estructural entre objetos.
Estos pueden tener una existencia separada e independiente,
pero se asocian en algún momento

La direccionalidad de la relación es importante *¿Quién se asocia con quién?*
Las relaciones bidireccionales no poseen flecha.

Multiplicidad



min .. max

0 .. 1

1 .. 100

1 .. *

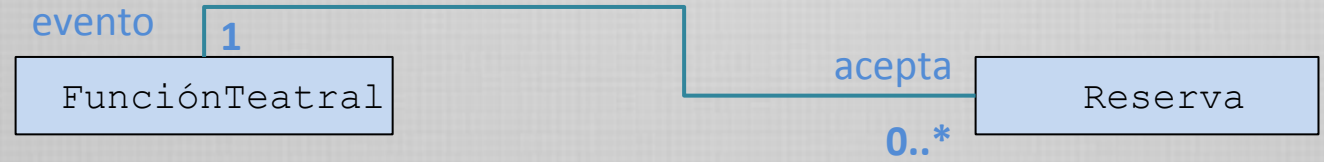
*

Relaciones entre clases – asociaciones binarias

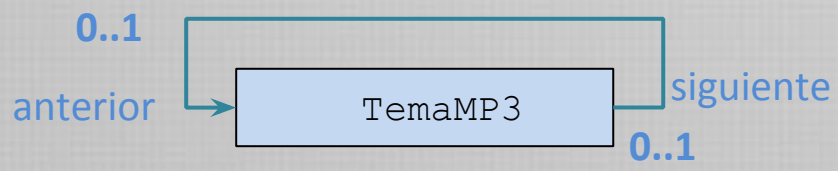
Un contribuyente tiene un domicilio legal

Un alumno universitario puede inscribirse en una o varias carreras.

Se pueden hacer reservas para una función teatral.

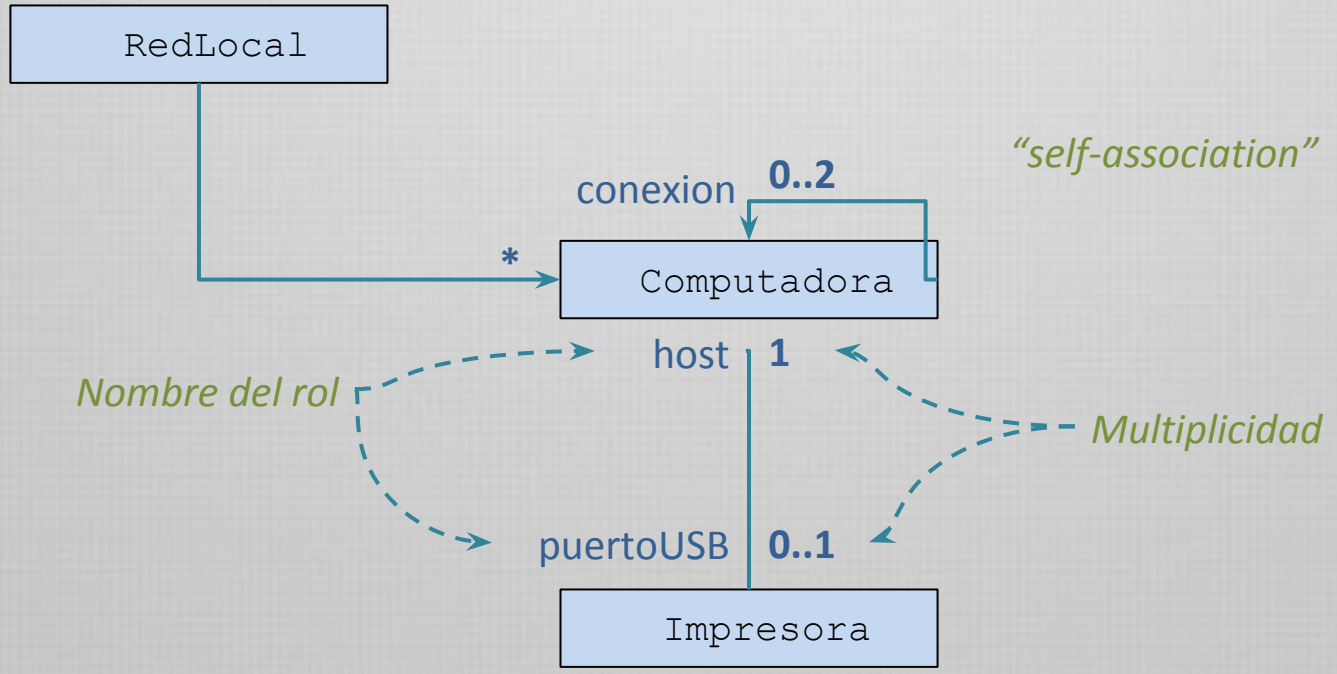


Los archivos mp3 se reproducen uno después del otro.



Relaciones entre clases – asociaciones binarias

Una red local de la empresa comunica varias computadoras.
Una computadora puede conectarse a otras dos.
Cada computadora puede tener una impresora conectada al puerto USB.



Relaciones entre clases

Algunas relaciones entre objetos sugieren más que un conocimiento circunstancial entre ellos...

Un equipo de fútbol profesional está formado por varios jugadores federados.

Aquí los objetos no tienen una existencia separada e independiente, sino que algunos de ellos son agrupados o ensamblados para crear un objeto nuevo.

Es un tipo especial de asociación que implica una fuerte dependencia estructural. Se denomina agregación y denota informalmente la relación "*es parte de*".

Se dibuja con un rombo sin color de relleno
El rombo va en el extremo "contenedor"

Relaciones entre clases

Algunas *relaciones de agregación* denotan restricciones especiales.
No solo los objetos forman parte de otros, sino que el ciclo de vida es el mismo...

Una casa posee varias habitaciones.

El objeto compuesto controla la creación de los objetos que lo componen.
Los objetos que son parte del todo, no pueden existir separadamente.
Esta relación se denomina composición.

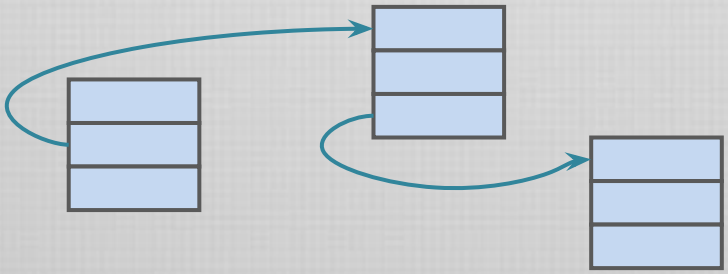
Se dibuja con un rombo con color de relleno
El rombo va en el extremo “contenedor”

Asociaciones, agregaciones y los lenguajes OO

Recordemos que el lenguaje de implementación debe permitirnos conservar lo mejor posible el diseño orientado a objetos.

¿Cómo nos ayuda el lenguaje a implementar estas relaciones entre clases?

Las asociaciones se implementan típicamente como *referencias*.
Los objetos tienen existencia separada y
podemos "navegar" de uno al otro siguiendo una referencia.



La estructura dinámica se puede modificar libremente:
algunos objetos podrán reemplazar a otros,
dos objetos pueden "desasociarse"
algun objeto puede ser eliminado
algún objeto puede ser modificado desde otra referencia

Asociaciones, agregaciones

Asociación

Los objetos se conocen unos a otros para poder trabajar juntos

Agregación

Los objetos juntos conforman una unidad con significado.

Composición

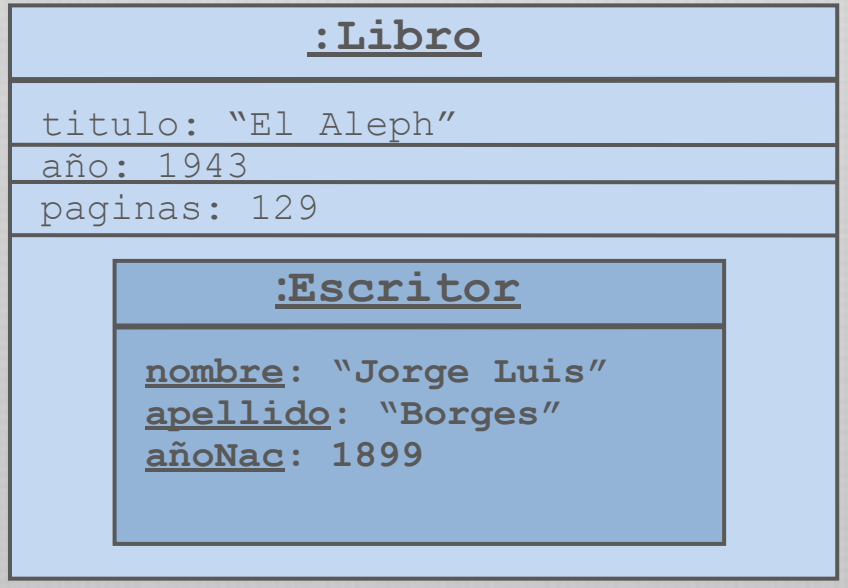
Cada parte puede conformar solo un objeto agregado

Asociaciones, agregaciones y los lenguajes OO

Para las agregaciones y composiciones algunos lenguajes ofrecen un mecanismo diferente para el tratamiento de objetos.

Se denomina usualmente manipulación de objetos con semántica por valor en contraposición a la semántica por referencia.

En la *semántica por valor*, un objeto puede literalmente contener a otro objeto. Este último se denomina subobjeto del primero.



Asociaciones, agregaciones y los lenguajes OO

Una referencia es en realidad una entidad que nos permite acceder a objetos .

En manipulación de objetos con semántica por valor, el objeto es nombrado y accedido directamente, no por medio de referencias.

- La noción de subobjeto implica que el contenido sólo puede ser conocido por el objeto continente
- Es posible sin embargo que el subobjeto referencie a otros objetos que no son el que lo contiene
- Una entidad con semántica por valor no puede ser nula.
- El tiempo de vida del subobjeto es el tiempo de vida del objeto que lo contiene.
- La construcción (creación) de un objeto involucra implícitamente la construcción de todos los subobjetos que contiene.

En el lenguaje Eiffel, se denominan objetos *expandidos*.

Se puede declarar
un objeto *expandido*, o
una clase que siempre produce objetos *expandidos*.

En el lenguaje C++, podemos manipular objetos por referencia o por valor, pero el acceso a funciones miembro es diferente:

```
Empleado *homero;
Empleado lenny;
homero = new Empleado();
a = homero->verTotalAccidentes();
b = lenny.verTotalAccidentes();
```