

El Modelo de Dominio

Empleando el diagrama de clases de UML

Ingeniería de Software – IIC2143

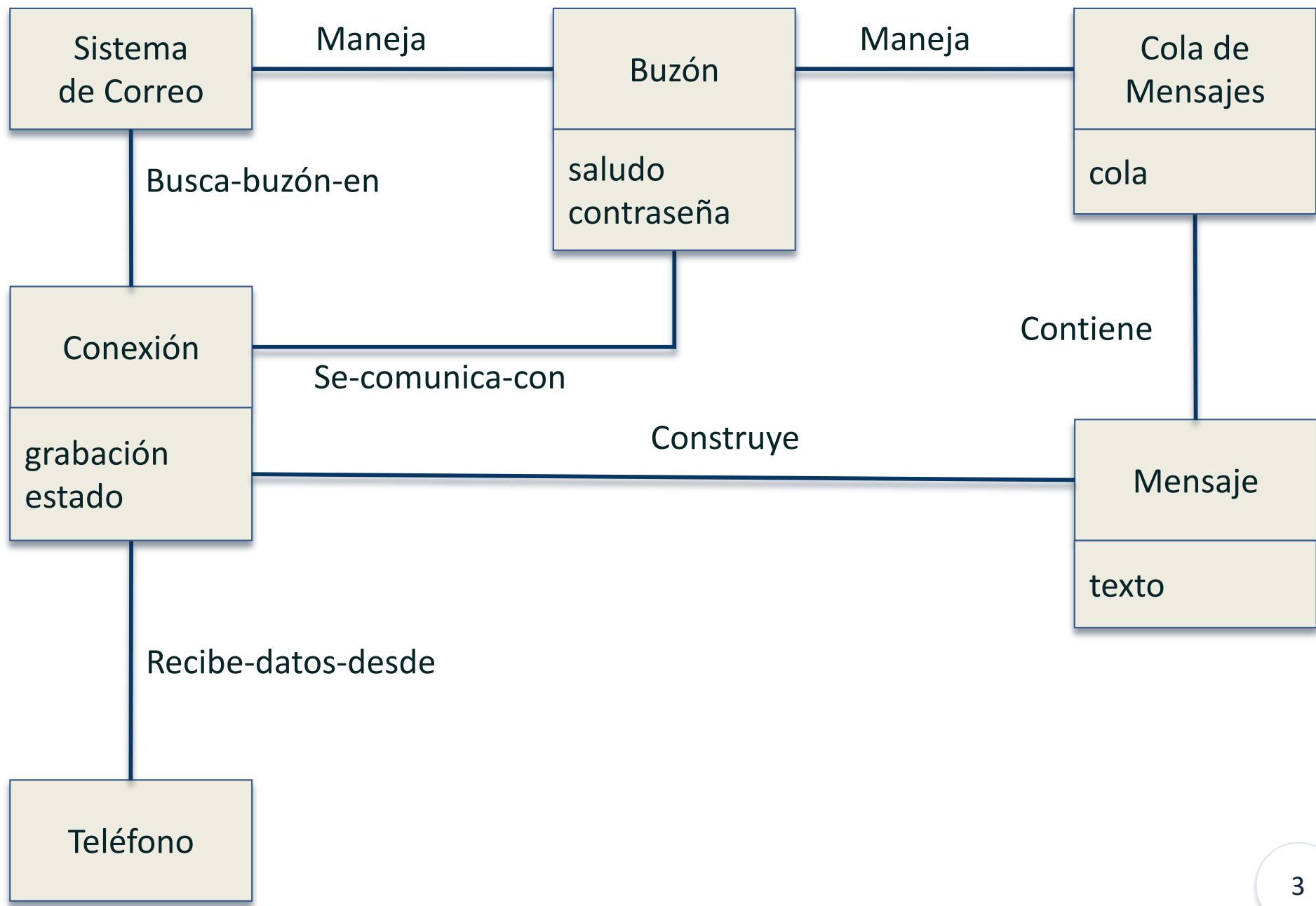
Ejemplo de un modelo de dominio

La próx. diap. muestra un **modelo de dominio** para un sistema de buzón de voz,

... representado mediante un diagrama de clases de UML

El modelo “dice” lo siguiente:

- las conceptos (o clases conceptuales) *Conexión*, *Buzón*, *Cola de Mensajes*, etc. son importantes en este dominio
- un *Buzón* está relacionado con una *Conexión* y con una *Cola de Mensajes* de formas que vale la pena notar
- dos propiedades importantes de un *Buzón* son su saludo y su contraseña
- etc.



¿Cómo representamos un modelo del dominio con UML?

Si construimos modelos de dominio, es útil poder representarlos

Al usar UML, un modelo de dominio es representado mediante un (conjunto de) diagrama(s) de clases, e incluye lo siguiente:

- objetos o conceptos del dominio o clases conceptuales
- relaciones entre clases conceptuales
- atributos de las clases conceptuales

El diagrama de clases de UML

(una manera de representar modelos de dominio)

Las clases son representadas mediante rectángulos

Cada clase tiene uno, dos o tres compartimientos:

- **nombre** (el de más arriba)
- **atributos** —opcional (el que sigue)
- **operaciones** (o métodos) —opcional (el de más abajo; si aparece, también tiene que aparecer el compartimiento de los atributos)

En un modelo de dominio no se incluye el compartimiento de las operaciones

Un modelo de dominio no hace referencia al software

Es una visualización de cosas en un dominio de interés en una situación real

En un modelo de dominio **evitemos incluir lo siguiente:**

- clases del software, como las que uno escribiría en un programa en Java o C#
- artefactos de software, tales como ventanas (de una GUI) o bases de datos (p.ej., la base de datos de clientes)
- responsabilidades, o métodos, en las clases

Un modelo de dominio no es un modelo de datos

Un modelo de datos muestra la información persistente que debe ser almacenada en alguna parte

En un modelo de dominio es válido incluir clases conceptuales sin atributos o que tienen un rol sólo de comportamiento —y no de información— en el dominio:

- p.ej., Sistema de Correo y Teléfono, en la diap. #3

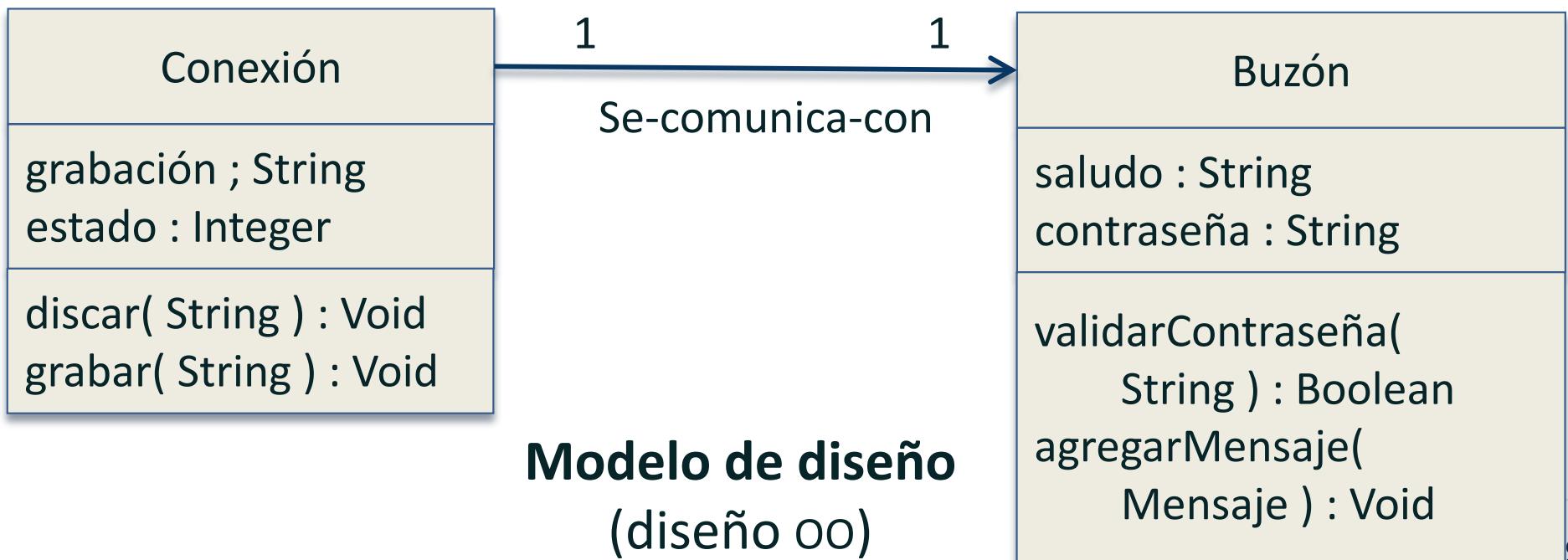
¿Por qué hacer modelos de dominio?

- 1) Ayudan a **entender los conceptos** fundamentales y el vocabulario de un dominio
- 2) Reducen la distancia representacional entre nuestros modelos mentales y los modelos de software cuando, más adelante, diseñemos:
 - los nombres de las clases de software en el diseño pueden estar inspirados en los nombres del modelo de dominio
 - también, los contenidos de información y las responsabilidades de las clases (y objetos) de software pueden provenir del dominio
 - p.ej., ver la próx. diap.

Modelo de dominio (análisis oo)



El modelo de dominio sugiere los objetos para el diseño OO



Sigamos **estos pasos** para construir un modelo de dominio

1. Encontrar los **conceptos** (o clases conceptuales)
2. Dibujarlos como **clases** en un diagrama de clases del UML
3. Agregar **relaciones** entre clases
4. Agregar **atributos** a las clases

Para encontrar clases conceptuales,
hagamos lo siguiente (algunas o todas)

1. Reusemos **modelos existentes**:

- “Model Driven Design Using Business Patterns”, de Pavel Hruby, 2010
- “Enterprise Model Patterns: Describing the World (UML Version)”, de David Hay, 2011

2. Usemos **listas de categorías**:

- la próxima diapositiva contiene categorías comunes de clases conceptuales que vale la pena considerar

3. Identifiquemos **sustantivos**:

- ... en descripciones en texto de un dominio, p.ej., en casos de uso

Categorías comunes de conceptos (o clases conceptuales) y *ejemplos*

Transacciones: *Reserva*

“Líneas” de una transacción: *Identificación del pasajero, Identificación del vuelo, Precio*

Productos o servicios: *Vuelo, Asiento*

Registros de transacciones: *Lista de pasajeros*

Roles de personas u organizaciones (actores): *Pasajero, Línea aérea*

Lugares donde ocurren transacciones: *Aeropuerto, Avión*

Eventos, con hora y lugar: *Vuelo*

Objetos físicos: *Avión*

Descripciones de cosas: *Descripción de vuelo*

Catálogos: *Catálogo de vuelos*

Contenedores de cosas físicas o información: *Avión*

Cosas en un contenedor: *Pasajero*

Otros sistemas: *Control de tráfico aéreo*

Instrumentos financieros: *Cheque*

Programas de actividades, manuales, documentos consultados a menudo: *Programa de reparaciones*

La identificación de sustantivos es una técnica simple

Pero ¡cuidado! :

- no es posible hacer una correspondencia mecánica entre sustantivos y clases
- las palabras en el lenguaje natural son ambiguas
- diferentes sustantivos pueden representar el mismo concepto

Usemos casos de uso (próx. diap.), otros documentos, el conocimiento de expertos

Algunos sustantivos identificados corresponderán a clases conceptuales, otros a atributos

Ejemplo de identificación de sustantivos en un caso de uso

El Suscriptor llama al Sistema y el Sistema le pide el número del buzón. El Suscriptor marca el anexo del destinatario, el Sistema reconoce el buzón y le pide el mensaje. El Suscriptor dice su mensaje y cuelga. El Sistema guarda el mensaje en el buzón del destinatario.

En un modelo de dominio, **usemos los términos del dominio**

Usemos los nombres existentes:

- p.ej., *Conexión, Mensaje* para el sistema de buzón de voz
- p.ej., *Pasajero, Reserva* para una línea aérea

Excluyamos propiedades irrelevantes o fuera del alcance:

- p.ej., el manejo de premios en la línea aérea podría postergarse para una segunda versión del sistema

Definitivamente, no agreguemos cosas que no están ahí

Distingamos entre clases y atributos

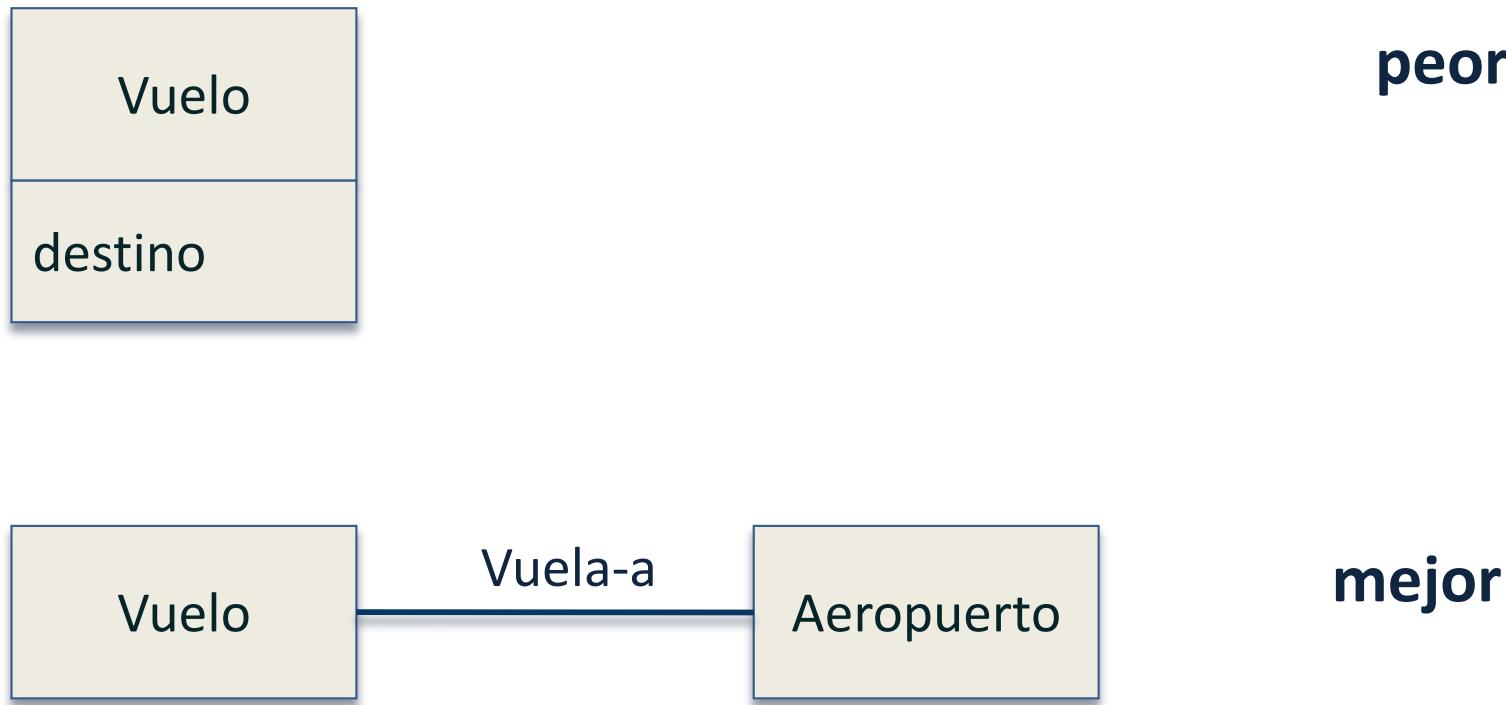
Puede haber confusión con respecto a si algo debe ser una clase conceptual o un atributo

Pauta:

- si cuando pensamos en X pensamos en algo más que sólo un número o un texto, entonces X es seguramente una clase conceptual y no un atributo

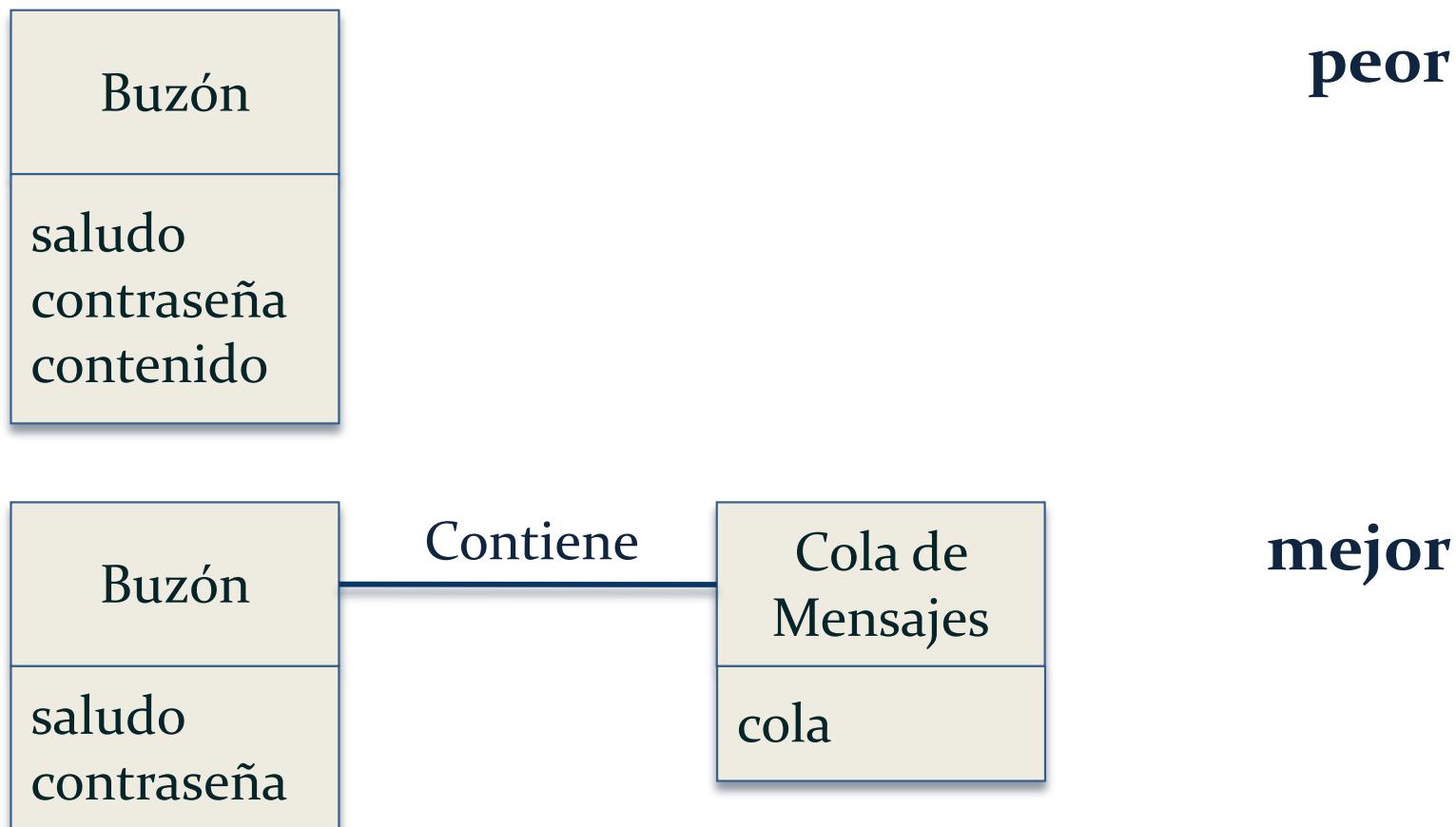
P.ej., ¿debería *destino* ser un atributo de *Vuelo* u otra clase conceptual llamada *Aeropuerto*?

- en el mundo real, un aeropuerto no es simplemente un número o un texto —debería ser un concepto



P.ej., una *Cola de Mensajes* es una estructura conceptualmente compleja, tanto en forma como en funcionamiento:

- es mejor representarla como una clase aparte y no como un atributo *contenido* de *Buzón*

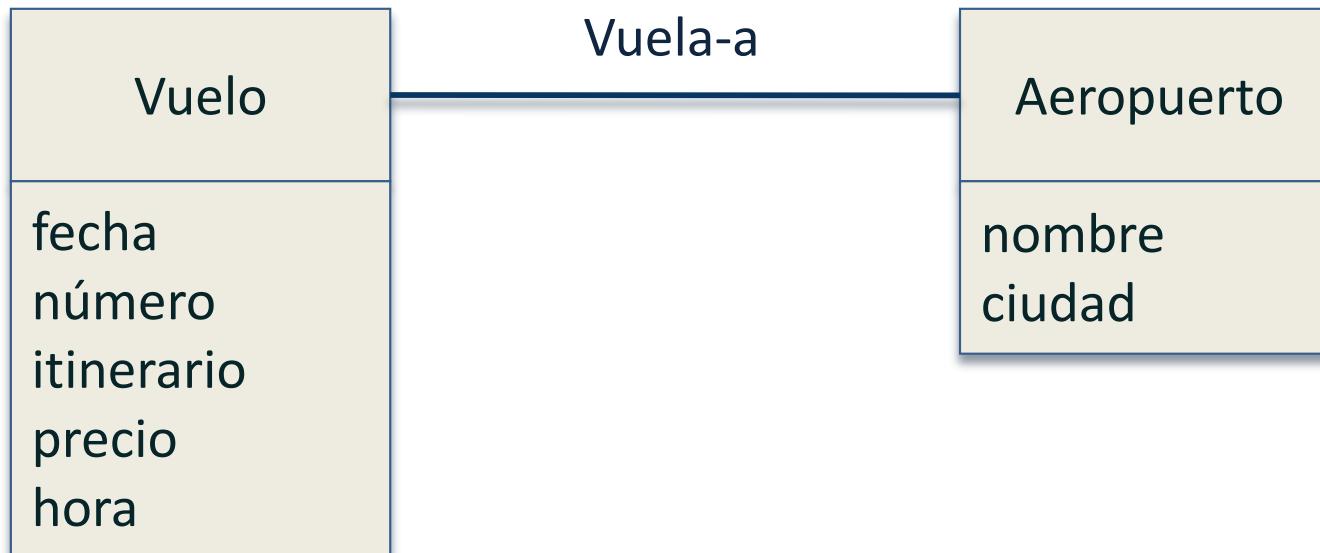


Consideremos la posibilidad de incluir clases que son descripciones de otra cosa

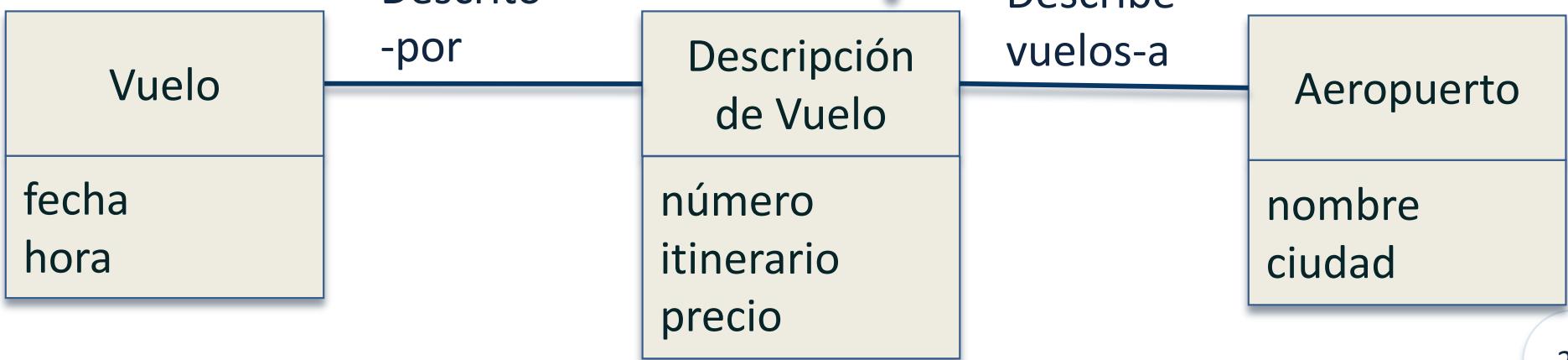
Una **clase descriptiva** contiene información que describe otra cosa:

- p.ej., una *DescripciónDeVueloSantiagoLondres* podría incluir el número, el itinerario y el precio
- ... mientras que cada *VueloSantiagoLondres* incluiría sólo la fecha y hora

peor



mejor



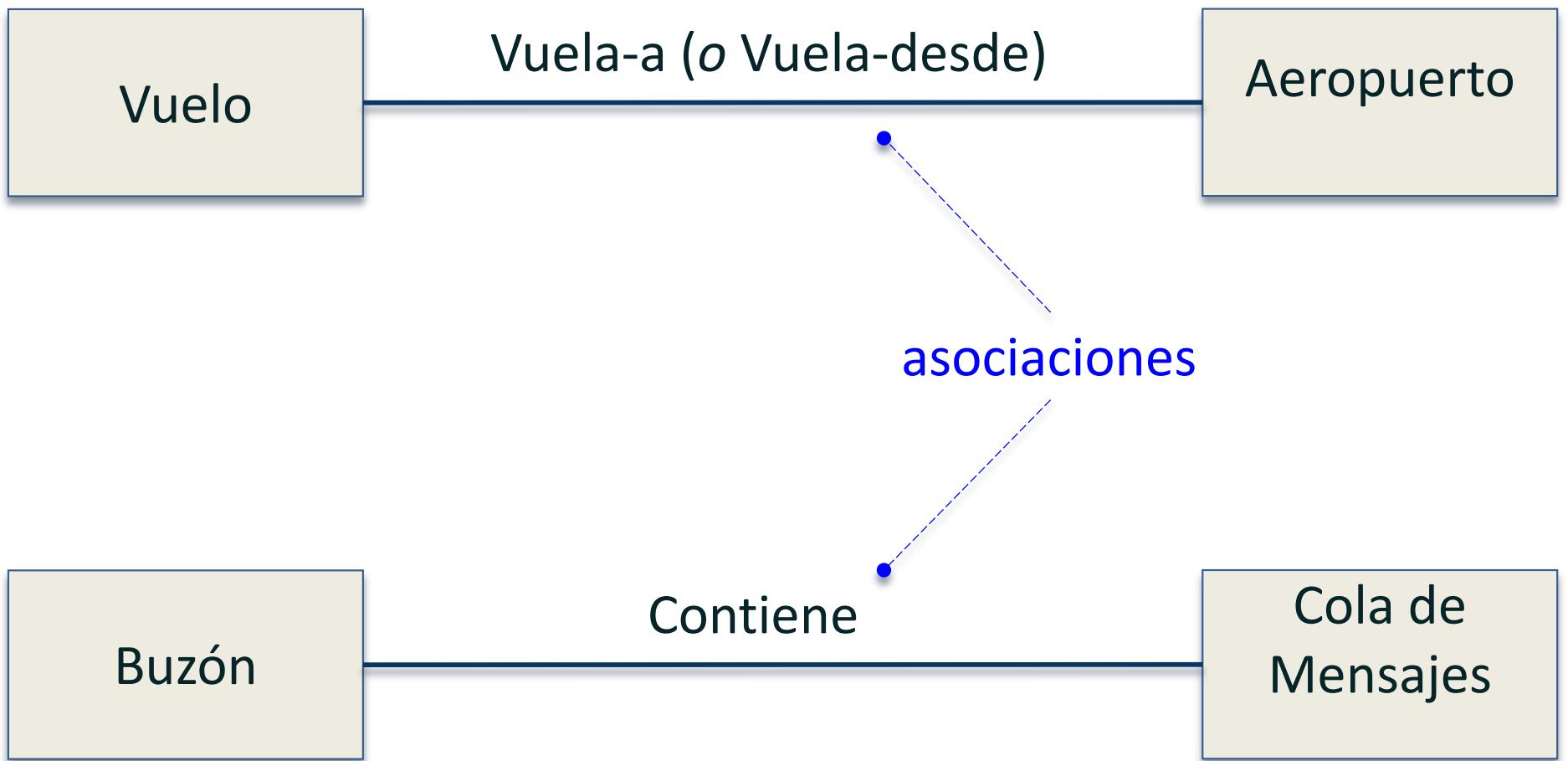
Incluyamos clases descriptivas en los siguientes casos

- 1) Se necesita que haya una descripción de un artículo o servicio, independientemente de si existen instancias concretas de esos artículos o servicios
- 2) La eliminación de instancias de las cosas descritas resulta en una pérdida de información —información que debería mantenerse (pero que fue asociada incorrectamente con las instancias eliminadas)
- 3) Reduce información redundante o duplicada

¿Qué es una asociación en UML?

Una asociación es una **relación entre clases** que indica una conexión significativa e interesante entre instancias de esas clases:

- P.ej., los vuelos de una línea aérea salen desde aeropuertos y llegan a aeropuertos;
- ... por lo tanto, debería existir una asociación entre las clases *Vuelo* y *Aeropuerto*
- P.ej., un buzón de voz típicamente contiene colas de mensajes;
- ... por lo tanto, debería existir una asociación entre las clases *Buzón* y *Cola de mensajes*



¿Cuándo dibujar una asociación?

Cuando implica conocimiento de una relación que debe ser mantenido por algún tiempo, en los siguientes casos:

- es necesaria para **cumplir los requisitos de información** de los casos de uso que estamos desarrollando
- **ayuda a entender el dominio**

Esta necesidad de recordar se refiere a una situación del mundo real, **no a una necesidad de software**:

- aunque durante la implementación, muchas de las mismas necesidades volverán a aparecer

Consideraremos incluir los siguientes tipos de asociaciones

- 1) Asociaciones para las cuales el conocimiento de la relación debe mantenerse por algún tiempo (que puede ser desde milisegundos hasta años, según el contexto) —asociaciones “**por la necesidad de recordar**”

- 2) Asociaciones derivadas de la lista de **asociaciones comunes** (más adelante)

En UML, una asociación es una línea entre dos clases, con un nombre

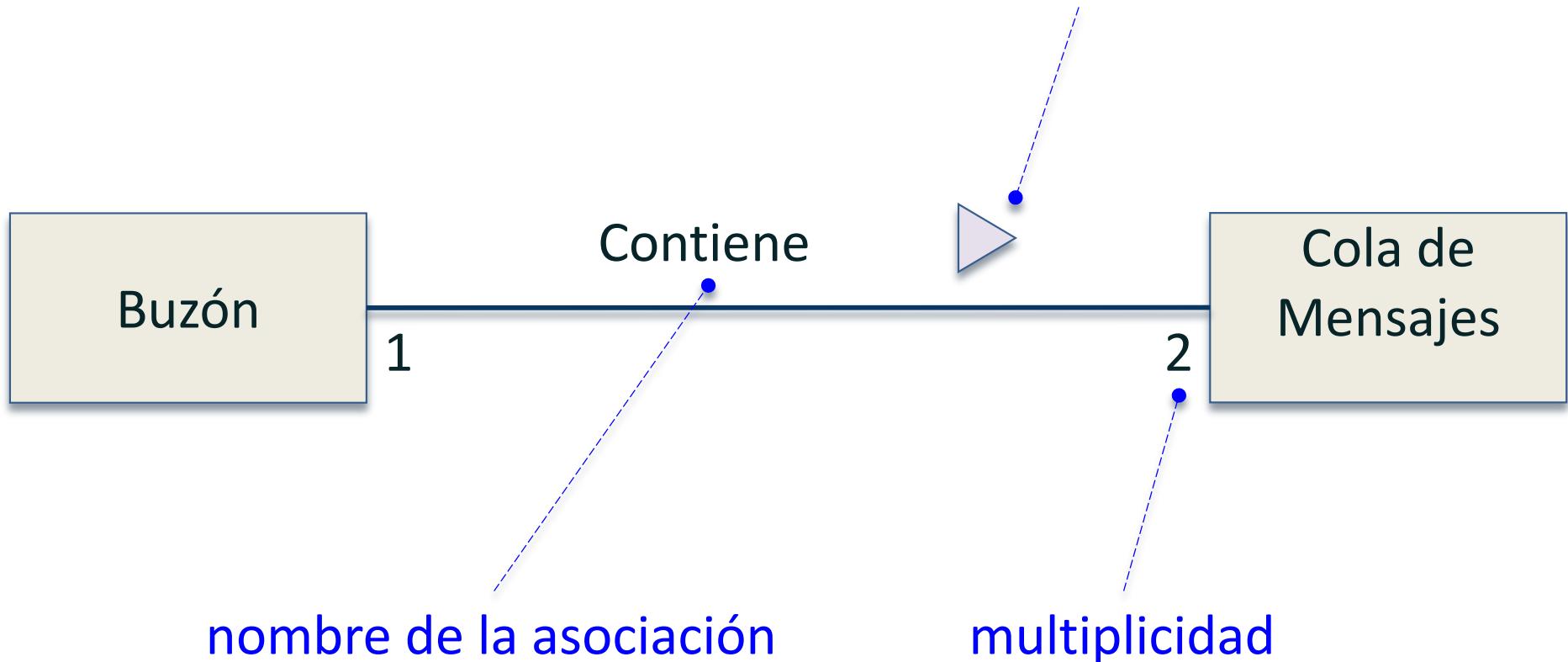
Los extremos pueden indicar multiplicidad:

- relación numérica entre instancias de las clases

Es bidireccional:

- desde instancias de cualquiera de las clases, es posible hacer un recorrido lógico a la otra (no es una afirmación acerca de conexiones entre entidades de software)

flecha de dirección de lectura;
no tiene ningún otro significado;
a menudo, no se incluye



Las asociaciones no representan estructuras de datos

Una asociación en un modelo de dominio **no es una afirmación** acerca de

- flujos de datos
- relaciones de claves foráneas en bases de datos
- variables de instancia (o atributos)
- conexiones entre objetos en una solución de software

... es sólo una afirmación de que una relación es significativa conceptualmente:

- aunque muchas de estas relaciones finalmente van a ser implementadas en software

¿Cómo nombrar a las asociaciones?

Nombremos una asociación basados en el formato

NOMBREdeClase–FraseVerbal–NOMBREdeClase,

en que la frase verbal (el nombre de la asociación) forma una secuencia legible y significativa

Nombres simples tales como “Tiene” o “Usa” a veces no son muy útiles:

- si no mejoran nuestro entendimiento del dominio

Cada extremo de una asociación se llama un **rol**

Los roles opcionalmente pueden tener

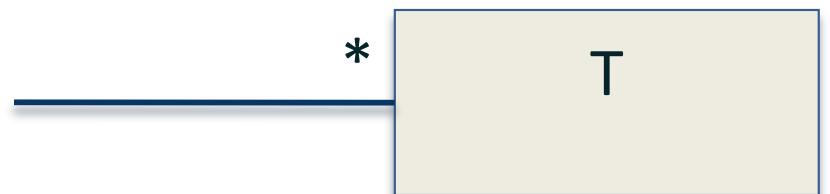
- (una expresión de) **multiplicidad**
- **nombre**
- **navegabilidad**

¿Qué representa la multiplicidad de un rol?

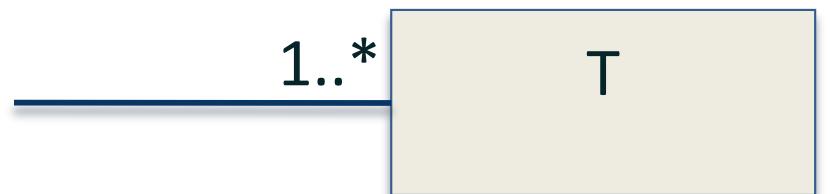
La multiplicidad define **cuántas instancias de una clase A pueden ser asociadas con una instancia de una clase B**:

- p.ej., en la diap. #28, una instancia de *Buzón* es asociada con dos instancias de *Cola de mensajes*
- ... y, desde el otro punto de vista, una instancia de *Cola de mensajes* es asociada con una instancia de *Buzón*

La próx. diap. muestra ejemplos de expresiones de multiplicidad



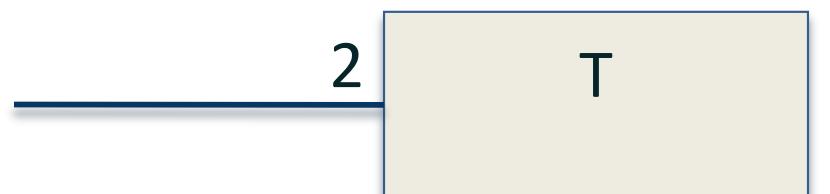
cero o más; muchos



uno o más



uno a 25



exactamente dos



exactamente 1, 2 o 4

Consideraciones sobre la multiplicidad de un rol

Se refiere a **un momento particular en el tiempo**, más que a lo largo del tiempo:

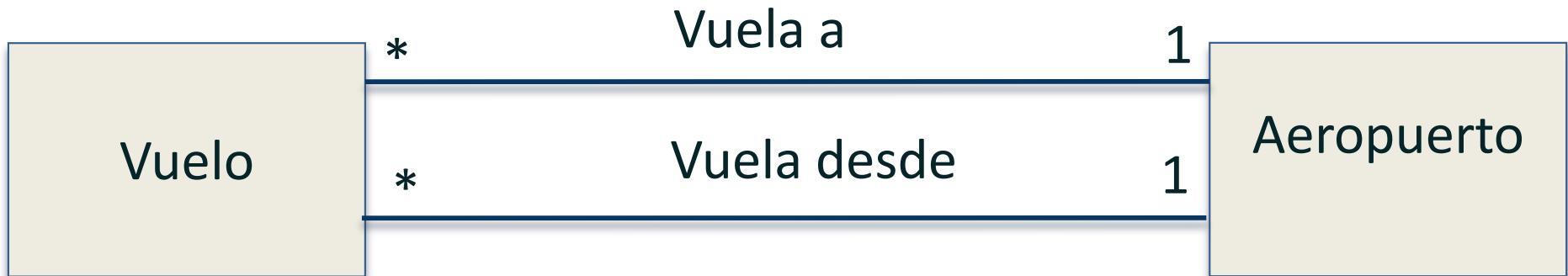
- p.ej., un auto puede ser vendido varias veces durante su vida útil;
- ... pero en un instante de tiempo particular el auto es Propiedad-de sólo un dueño

También depende de cómo queremos usar el modelo

Puede haber **varias asociaciones** entre dos clases

P.ej., en el caso de un *Vuelo* y un *Aeropuerto*:

- las asociaciones *vuela-a* y *vuela-desde* son relaciones claramente diferentes, que deberían mostrarse por separado



Categorías comunes de asociaciones y ejemplos

A es una transacción relacionada con otra transacción B: *Cancelación–Reserva*

A es una línea de una transacción B: *Identificación del pasajero–Reserva*

A es un producto o servicio para una transacción B: *Vuelo–Reserva*

A es un rol relacionado a una transacción B: *Pasajero–Boleto*

A es una parte física o lógica de B: *Asiento–Avión*

A está física o lógicamente contenida en *B*: *Pasajero–Avión*

A es una descripción de *B*: *Descripción de vuelo–Vuelo*

A es conocido en, está registrado en, o está informado en *B*: *Reserva–Lista de pasajeros*

A es un miembro de *B*: *Piloto–Aerolínea*

A es una unidad organizacional dentro de *B*: *Mantenimiento–Aerolínea*

A usa, maneja o es dueño de *B*: *Piloto–Avión*

A está cerca de *B*: *Ciudad–Ciudad*

Ejemplos de asociaciones

El sistema de correo **Maneja** buzones

Una conexión **Recibe-datos-desde** un teléfono

Una conexión **Se-comunica-con** un buzón

Un buzón **Maneja** colas de mensajes

Una cola de mensajes **Contiene** mensajes

¿Qué es un atributo en UML?

Un atributo es una **propiedad lógica de un objeto**

Identifiquemos aquellos atributos de las clases conceptuales **que son necesarios para cumplir los requisitos de información** de los casos de uso en desarrollo:

- p.ej., un caso de uso del sistema de buzón de voz dice que cuando el suscriptor llama a su buzón, el sistema le da un saludo de bienvenida y le pide que ingrese su contraseña
- por lo tanto, la clase *Buzón* necesita tener los atributos *saludo* y *contraseña*

Notación UML para los atributos

```
<visibilidad> <nombre> : <tipo> <multiplicidad> = <default>  
{<property-string>}
```

Normalmente, la visibilidad es privada (-) a menos que se muestre otra cosa

El <property-string> más común es {readOnly}, es decir, sólo para lectura

La multiplicidad se puede usar para indicar

- la presencia opcional de un valor
- el número de objetos que puede haber en un atributo (cuando es una colección de objetos)

Tipos de datos apropiados para los atributos

Normalmente, los tipos de los atributos deberían ser tipos de datos primitivos:

- números, texto (strings), booleans

Otros tipos de datos comunes para atributos son los siguientes:

- fechas, horas, direcciones, colores, números de teléfono, rut's, códigos postales, enumeraciones

En cambio, no es una buena idea modelar conceptos complejos del dominio como atributos:

- éstos deberían ser clases y relacionarse con otras clases por medio de asociaciones

No existe un único modelo de dominio correcto

Los modelos **son aproximaciones** del dominio que estamos tratando de entender:

- el modelo de dominio es primeramente una herramienta para entender y comunicarse dentro de un grupo

Un modelo de dominio **útil** captura los conceptos e información esenciales para entender el dominio en el contexto de los casos de uso vigentes:

- conceptos, terminología, relaciones