UML (I): el modelo estático

Benet Campderrich Falgueras
Recerca Informàtica, S.L.

PID_00198152

Universitat Oberta de Catalunya

Índice

Introducción

Como sabemos, para el análisis y el diseño orientados a objetos utilizaremos los conceptos y las notaciones –esencialmente gráficas– de UML.

Ved también

Consultad el módulo "Introducción a la ingeniería del *software*" de esta asignatura.

UML comprende un cierto número de diagramas interrelacionados mediante conceptos comunes; estos diagramas, sólo para describirlos, los consideraremos agrupados en tres modelos:

- El **modelo estático**, que describe la estructura de clases y objetos.
- Lo que podríamos denominar **modelo dinámico** o de **modelo de comportamiento**, que describe las interacciones entre los objetos dentro del *software*.
- El **modelo de implementación**, que describe la estructura del *software* en términos de los componentes de que consta y su ubicación.

En este módulo veremos el modelo estático. En módulos posteriores trataremos el modelo dinámico. El modelo de implementación queda fuera del alcance de esta asignatura.

Ved también

Consultad el modelo dinámico y el modelo de implementación en el módulo "UML (II): el modelo dinámico" de esta asignatura.

El modelo estático consta, por una parte, de clases y objetos, y por la otra, de relaciones de diferentes tipos entre clases y entre objetos.

Objetivos

El objetivo principal de este módulo es conocer con profundidad el diagrama estático de UML, lo que significa tanto entender los conceptos como saberlos aplicar y, más detalladamente, implica los siguientes aspectos:

1. Profundizar en el concepto de clase, en especial en lo que respecta a los diferentes tipos de clases y conceptos relacionados, y familiarizarse con las notaciones respectivas.

2. Entender los tipos más frecuentes de relaciones entre clases y saberlos identificar.

3. Conocer los principales elementos complementarios de UML aplicables al diagrama estático.

1. Concepto de modelo estático y diagrama de clases

El modelo estático del UML es aquél en el que se describen las clases y los objetos. Se denomina *estático* porque muestra todas las relaciones posibles a lo largo del tiempo, no las que son válidas en un cierto momento.

Ejemplo de modelo estático

Un diagrama estático nos puede mostrar que cada profesor tiene, al menos, una asignatura, y que cada asignatura tiene, al menos, un profesor, pero no nos dice qué asignaturas tiene un profesor concreto.

El modelo estático consta de los dos diagrames siguientes:

- El **diagrama de clases**, que puede contener clases y objetos y relaciones entre éstos, y que se hace siempre.

- El **diagrama de objetos**, que sólo contiene objetos y relaciones entre éstos, es opcional, ya que se utiliza principalmente para realizar ejemplos del diagrama de clases con objetos concretos de éstas.

Un diagrama de clases muestra la estructura estática de las clases en un **dominio** (porción del mundo real considerada por una aplicación); se muestran las clases y las relaciones entre ambas, que pueden ser de herencia, asociación, agregación o uso.

Uso del modelo estático

El modelo estático se utiliza en todas las etapas del ciclo de vida; en las diferentes etapas se documentan diferentes tipos de objetos. En el análisis se consideran objetos del mundo del usuario (por ejemplo: artículos, facturas, clientes, etc.) y en el diseño, en cambio, se consideran objetos de la tecnolgía informática: pantallas, gestores de disco, etc.

El modelo estático pretende ser independiente del lenguaje de programación, pero, sin embargo, si se sabe cuál será, es conveniente no utilizarlo en el análisis de conceptos que sabemos que dicho lenguaje no soporta, si queremos ahorrarnos hacer muchos cambios cuando lleguemos al diseño. También se deberá tener en cuenta que, cuando UML permite describir elementos incompatibles con un lenguaje determinado, raramente la herramienta CASE nos lo impedirá; por lo tanto, será responsabilidad del diseñador del *software* evitar caer en la utilización de conceptos no soportados por el lenguaje de programación.

También puede suceder lo contrario; es decir, que se quieran modelar elementos que la herramienta CASE no soporta, porque UML no los prevé, o por otros motivos; entonces se tendrán que documentar estos aspectos mediante comentarios libres, que permiten todas las herramientas. Algunas herramientas permiten que el usuario defina extensiones, pero si una empresa utiliza esta posibilidad, los diagramas generados no serán transportables a otras empresas.

2. Clasificadores

El **clasificador** es la entidad básica del modelo estático. Un clasificador es más general que una clase; es un conjunto cuyos elementos se denominan **instancias**.

El clasificador en sí mismo no tiene símbolo gráfico, sino que lo tienen sus estereotipos: clase, tipo de dato e interfaz.

Estereotipo

Un estereotipo de un elemento de UML es una variante más restrictiva de dicho elemento; hay estereotipos que forman parte de UML, y también se pueden encontrar estereotipos definidos referidos al diagrama, que son un instrumento para estender UML, pero así se pierde portabilidad.

- El concepto de **clase** es el que ya conocemos de la orientación a objetos, y sus **instancias** son los objetos, que tienen identidad, en el sentido de que incluso dos objetos que coinciden en el valor de todos sus atributos son objetos diferentes si se han creado como tales.

- Por **tipo de dato** entendemos un tipo base ofrecido por algún lenguaje de programación o construido por el programador; tiene operaciones asociadas igual que las clases, pero sus instancias, a diferencia de los objetos, no tienen identidad.

- Una **interfaz** sólo describe las operaciones de una clase que son visibles desde otras clases; se dice que dicha clase implementa la interfaz correspondiente.

Ved también

Véase el módulo "Orientación a objetos" de esta asignatura.

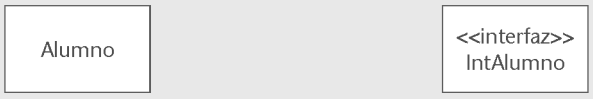
La utilidad del concepto de clasificador radica en el hecho de que los estereotipos mencionados tienen mucho en común, por lo que es suficiente con realizar la indicación una vez en el clasificador. La notación gráfica simplificada es la misma para los tres: un rectángulo.

Ved también

Consultad una notación gráfica ampliada de las clases en el apartado 4 de este módulo didáctico.

Todos los clasificadores deben tener un nombre. En un clasificador se puede indicar la palabra clave del estereotipo (entre comillas latinas, «»); cuando no se indique ningún estereotipo, se tratará de una clase.

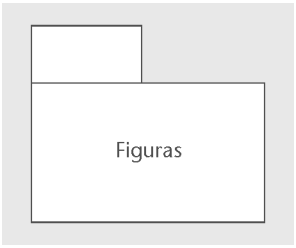
Ejemplo de clase y interfaz



3. Paquetes

Un **paquete** o ***package*** es sólo una "caja" que contiene elementos, como clasificadores, objetos u otros paquetes, así como otras entidades que veremos más adelante, como los casos de uso.

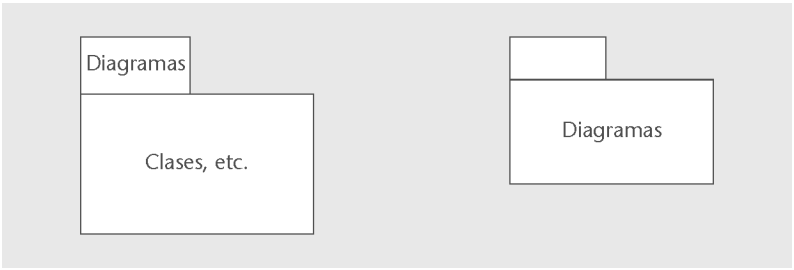
Gráficamente, un paquete se representa así:



Todas las aplicaciones deben tener, por lo menos, un paquete que normalmente se denomina *raíz*. Cada elemento de un paquete tiene visibilidad, es decir, puede ser reconocido o bien desde todos los otros paquetes, o bien sólo desde algunos.

Ejemplo de paquetes

Éstas son dos maneras de representar el mismo paquete:

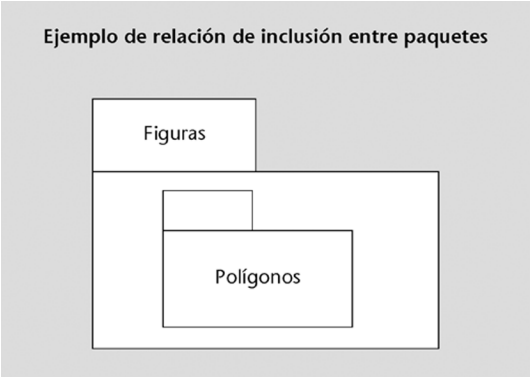


En la primera, se pueden incluir dentro del símbolo del paquete los símbolos de los elementos que contiene; la segunda, simplificada, es más adecuada para representar referencias al paquete (desde otros paquetes, por ejemplo).

Se pueden establecer los siguientes tipos de **relaciones entre paquetes**:

- **De especialización**: si un paquete *A* hereda de otro *B*, todos los elementos de *A*, son casos más restrictivos de elementos de *B*.

- **De inclusión**: si el paquete *A* incluye el *B*, todos los elementos de *B* están también en *A*.



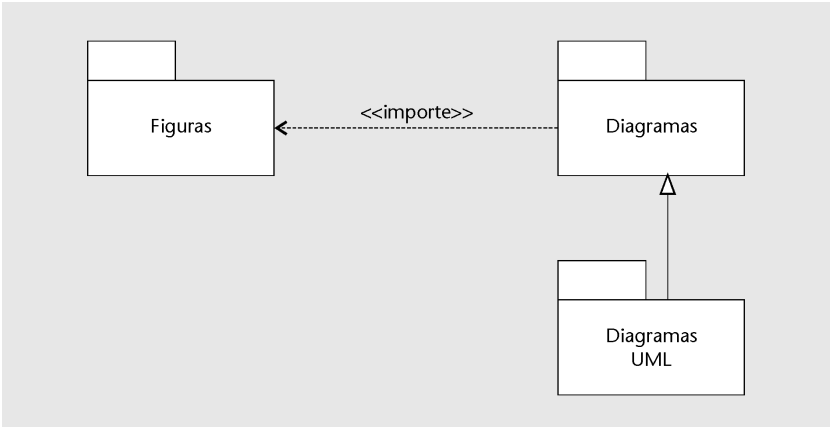
- **De importación**: desde el paquete que importa se reconocen los nombres de los elementos del otro paquete visibles desde el exterior[1].

- **De acceso**: no sólo se reconocen los nombres de los elementos, sino que, además, se pueden utilizar.

[(1)]Por ejemplo, públicos en el sentido de Java.

Ejemplo de relaciones entre paquetes

En la representación gráfica, el paquete *Diagramas* que vemos a continuación importa del paquete *Figuras*, y el paquete *Diagramas de UML* hereda del paquete *Diagramas*; una relación de acceso se representaría con la palabra clave *access*.



4. Clase y conceptos afines

Sobre la orientación a objetos sabemos que una **clase** describe un conjunto de objetos en el que todos tienen los mismos atributos y las mismas operaciones; los atributos y operaciones pueden ser de **instancia**, es decir, ligados a objetos individuales, y de **clase**, que no están relacionados con ningún objeto en particular de la clase. Este mismo es el concepto de clase en UML.

En principio, cada clase es visible (es decir, reconocida) dentro del paquete donde se ha declarado, y su nombre no puede estar repetido en éste, pero desde un paquete se reconocen los nombres de clases –y elementos en general– de otro paquete del que se importa, en el sentido indicado con anterioridad; el nombre de la clase tiene que estar calificado por el del paquete, así Paquete :: Clase.

4.1. Representación ampliada de las clases

Puesto que una clase es un clasificador, se puede utilizar como símbolo de la clase un simple rectángulo con el nombre; sin embargo, dado que una clase consiste en un encapsulado de unos atributos y unas operaciones, también se da una representación gráfica más detallada por medio de un rectángulo dividido en los tres compartimentos siguientes:

Nota

Recordad que como sinónimos de operación se utilizan a menudo *método* y *servicio*; pero en UML, el término *servicio* no se usa, y por *método* se entiende la implementación de una operación.

- El primer compartimento contiene el nombre de la clase.
- El segundo compartimento contiene la lista de los atributos.
- El tercer compartimento corresponde a los servicios de la clase.

El usuario puede crear otros compartimentos, además de los tres obligatorios, para dar información adicional como excepciones, requisitos, etc.

4.1.1. El compartimento del nombre

En la parte superior del compartimento de la clase se puede indicar un estereotipo[2]; algunos estereotipos forman parte de UML –como *metaclass*, del que se hablará más adelante–, y se pueden definir mediante un proyecto concreto, por ejemplo.

(2)Aquí se trata de estereotipos de la clase. Recordemos que la clase, por su parte, es un estereotipo del clasificador.

Ved también

Consultad las metaclases en el subapartado 4.3.3 de este módulo didáctico.

Justo debajo se encuentra el nombre de la clase. Se recomienda que sea un sustantivo en singular que a veces puede tener una segunda palabra que la califique; se recomienda que comience por mayúscula.

El nombre de las clases

Los nombres de las clases tienen que estar bien pensados, ya que la posibilidad de reutilización depende en gran medida de ello, porque cuando queramos reutilizar una clase de una librería que tenga centenares de nombres, la única pista que tendremos es el nombre, y si no se le dio un nombre adecuado, será muy difícil de encontrar.

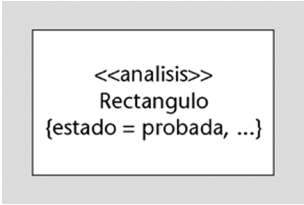
Debajo del nombre se pueden encontrar textos optativos entre llaves ({}) denominados *cadenas de caracteres de propiedades* (*property strings*) o *valores etiquetados* (*tagged values*); los puntos suspensivos que se pueden hallar al final de uno de los apartados indican que hay más elementos, pero que no se ven. La *property string* "abstract" denota una clase abstracta (veáse el apartado 4.2.2).

Formato de los comentarios

Property strings y *tagged values* tienen la forma nombre = *valor*; en el caso de propiedades booleanas, el nombre solo ya indica la presencia de la propiedad.

Ejemplo de clase sólo con el compartimento del nombre

En la figura siguiente, el estereotipo *análisis* podría indicar que la clase *Rectágulo* se ha identificado en la etapa de análisis.



4.1.2. Especificación de los atributos

Cada atributo tiene un nombre o identificador y un tipo. Este último puede ser un tipo simple del lenguaje de programación (por lo menos, durante el diseño y la programación, porque durante el análisis puede ser que las clases se describan sin pensar en ningún lenguaje de programación en concreto), como entero o carácter, o bien un tipo complejo, como una lista de enteros, o también una clase ya definida.

Un atributo, sea de instancia o de clase, se define de la siguiente forma:

visibilidad nombre ':' expresión-de-tipo '=' valor-inicial '{' property string '}'

Se indica que un atributo es **atributo de clase** subrayando su definición.

La **visibilidad de un atributo** indica hasta qué punto las operaciones de otras clases pueden acceder al atributo, y se indica mediante los siguientes símbolos:

- público: "+"
- protegido: "#"
- privado: "–"
- dentro del paquete: "~"

También tienen visibilidad otros elementos del modelo estático, como las operaciones y los extremos de asociación.

En el lugar de los atributos, se pueden utilizar *property strings*, que son, respectivamente, *public*, *protected* o *private*. Las *property strings* son opcionales; además de las mencionadas, podemos encontrar *frozen*, que indica que no se puede cambiar el valor del atributo.

La visibilidad

En UML no hay definiciones del significado de estas opciones de visibilidad, sino que se dejan para los lenguajes de programación; si algún lenguaje incluye más tipos de visibilidad, también se podrán indicar aquí.

En lo referente al nombre de los atributos, se deben tener en cuenta las siguientes pautas:

- Se recomienda que comience por minúscula.
- Cuando se trate de un atributo derivado (es decir, que es redundante con otros a partir de los cuales se puede obtener el valor), el nombre tiene que ir precedido de "/".
- Es conveniente que el nombre cumpla las reglas léxicas del lenguaje de programación, si no queremos que se tenga que cambiar al llegar al diseño. La expresión de tipo y el valor inicial también las deberán respetar.

Se pueden utlizar indicadores de **multiplicidad** como en el caso de los vectores o matrices de acuerdo con el lenguaje.

Uso de los indicadores de multiplicidad

Consideremos:

hijos [0..3]: persona

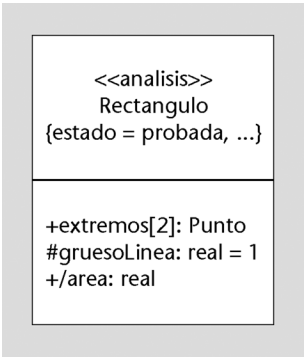
o bien,

hijos [3]: persona;

en el primer caso podría haber entre 0 y 3 hijos, pero en el segundo tiene que haber exactamente tres.

Ejemplo de clase con compartimento de nombre y compartimento de atributos

Como se puede ver en la figura siguiente:



extremos serían las coordenadas de dos vértices opuestos del rectángulo, que lo determinan; *Punto* sería una clase descrita en el mismo paquete; *gruesoLínea* tiene el valor 1 por omisión, y *área* es un atributo derivado, ya que su valor se puede calcular a partir de las coordenadas de los puntos de *extremos*.

4.1.3. Especificación de las operaciones

Una **operación** se define de la siguiente forma:

visibilidad nombre ‘(' lista-de-parámetros ‘):' tipo-de-retorno ‘{'property string‘}'

Se indica que una operación es **operación de clase** subrayando su definición. La visibilidad se indica igual que en el caso de los atributos.

Nota

En la programación conviene que los nombres de las operaciones estén bien pensados, porque son la base del polimorfismo, en el que se aplica que "a igual concepto, igual nombre".

Conviene que el nombre de la operación y de los parámetros y el tipo de los parámetros y del retorno cumplan las reglas del lenguaje de programación. Se recomienda que los nombres de las operaciones comiencen por minúscula.

La **lista de parámetros** se compone de parámetros separados por comas; la sintaxis de cada uno es la siguiente:

```
tipo nombre ':' expresión-de-tipo '=' valor-por-omisión
```

donde:

- *tipo* puede tomar los valores *in* (para indicar un parámetro de entrada), *out* (para indicar un parámetro de salida) o *inout* (para un parámetro tanto de entrada como de salida). Generalmente no se indica nada y por omisión se considera que todos los parámetros son de entrada.
- *nombre* es el nombre del parámetro.
- *expresión-de-tipo* indica la tipología del parámetro (alfanumérico, numérico, booleano...).
- *valor-por-omisión* es opcional y sirve para fijar un valor por defecto para el parámetro.

El *tipo-de-retorno* sólo se tiene que utilizar cuando la operación devuelva un valor como resultado, y también se puede usar un parámetro *out* en su lugar.

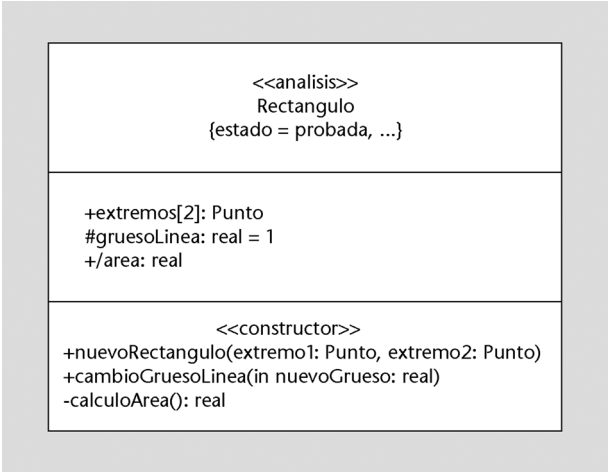
Opcionalmente, puede aparecer *property strings*: *query*, que denota que la operación no modifica el estado del sistema, y para especificar la semántica de concurrencia se pueden utilizar una de éstas: *sequential*, *guarded*[3] o *concurrent*, y *abstract*, que indica que la operación es abstracta (ved el subapartado 4.2.2).

(3)*Guarded* indica que la ejecución puede ser o bien concurrente o bien secuencial, según se cumpla o no una condición.

Se pueden utilizar estereotipos poniéndolos encima de la operación afectada.

Ejemplo de clase con los tres compartimentos

Tal y como se puede apreciar en el gráfico siguiente, la operación *nuevoRectangulo* es del estereotipo (no incluido dentro de UML) *constructor* y tiene dos parámetros *in* (por omisión); la operación *calculoArea* no tiene parámetros pero retorna un valor, cuyo tipo se indica.



4.2. La herencia en el análisis y el diseño

Sabemos que la herencia presupone que existan dos clases, de las cuales una desempeña el papel de superclase y la otra, el de subclase.

La subclase comprende un subconjunto de los objetos de la superclase, los cuales, por tanto, tienen todos los atributos y operaciones de instancia de la superclase (se dice que la subclase los *hereda*) y, además, pueden tener algunos adicionales, específicos de la subclase.

Nota

Se dice que la relación entre una subclase y su superclase es una relación *is_a_kind_of.*

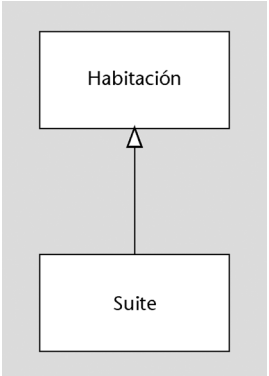
Según se defina primero la superclase o sus subclases, tenemos respectivamente dos tipos de herencia: herencia por especialización y herencia por generalización.

4.2.1. Herencia por especialización

Se llama de esta manera porque lo que se hace es crear una clase más especializada, más restrictiva, a partir de una clase definida con anterioridad.

Ejemplo de especialización

Consideramos que en la gestión de un hotel identificamos la clase *Habitacion* y después nos damos cuenta de que hay una categoría especial de habitaciones que tiene atributos y/o operaciones diferentes, que son las *suites*; esto se representa de la siguiente forma:



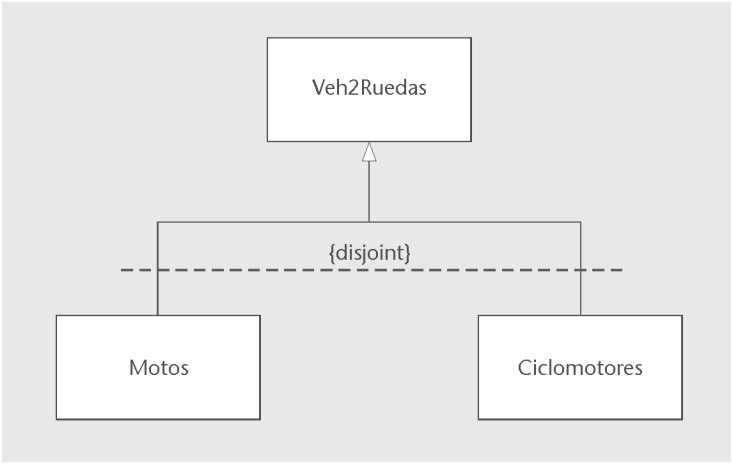
La flecha con punta triangular vacía y cuerpo continuo expresa una relación entre subclase y superclase, y su sentido indica cuál es cada una. También se habla de que hemos creado una jerarquía de herencia, muy sencilla en algunos casos[4], pero que llega a ser un árbol de diferentes niveles si hay superclases que presenten distintas subclases y subclases que sean a la vez superclases de otras. Incluso la jerarquía se convierte en una red si algunas clases tienen más de una superclase (**herencia múltiple**).

[(4)]Como en el caso del ejemplo de especialización.

El proceso mediante el cual reconocemos una subclase dentro de otra clase –que, consecuentemente, pasa a ser superclase de la primera– se denomina **especialización** o **derivación**.

4.2.2. Herencia por generalización. Clases abstractas

Supongamos que, al informatizar los impuestos de un municipio, encontramos, por un lado, los impuestos sobre las motos y por otro, los impuestos sobre los ciclomotores; aunque inicialmente se había considerado que *Motos* y *Ciclomotores* son dos clases diferentes, después se ha observado que tienen algunos atributos y operaciones en común y, en consecuencia, se define una superclase común *Veh2Ruedas*:



En este caso hemos procedido al revés: a partir de las subclases, hemos encontrado la superclase; este proceso se denomina **generalización**.

Otro procedimiento

En el proceso de generalización también se podría haber trazado una flecha independiente desde cada subclase a la superclase.

La propiedad *disjoint* denota que todo objeto de la superclase pueda pertenecer sólo a una de las subclases como máximo.

De la misma forma que en el "Ejemplo de especialización" podía haber habitaciones que no fueran *suites* y, por tanto, había objetos que, de las dos clases, sólo pertenecían a *Habitacion*, en este último, cualquier vehículo de dos ruedas es o bien una moto o bien un ciclomotor, es decir, todos los elementos

de *Veh2Ruedas* pertenecen a una u otra subclase y, por lo tanto, *Veh2Ruedas* es una clase artificial, un simple recurso para no describir dos veces lo que tienen en común *Motos* y *Ciclomotores*; se dice que *Veh2Ruedas* es una clase abstracta.

Una **clase abstracta** es una superclase de la cual no se pueden crear (instanciar) directamente objetos, sino que se tienen que crear necesariamente en alguna de sus subclases. Opcionalmente, y para indicar que se trata de una clase abstracta, su nombre se representa en cursiva.

Por esta razón se dice que las clases abstractas son no instanciables.

Una clase abstracta puede tener **operaciones abstractas**, que son las que sólo están implementadas en las subclases, en principio, de forma diferente en cada una. Como en el caso de una clase abstracta, su nombre se puede representar en cursiva para indicar que se trata de una operación abstracta.

Una operación abstracta debe tener o bien su definición en cursiva, o bien la propiedad {abstract} al final de ella.

4.3. Interfaces

Una interfaz describe un conjunto de operaciones visibles de una clase, sin indicar su implementación. Se dice que dicha clase en cuestión implementa la interfaz. Una interfaz no es una clase, pero equivale a una clase abstracta sin atributos y con todas sus operaciones diferidas.

Nota

Recordad que la interfaz y la clase son dos estereotipos del clasificador.

Las interfaces pueden establecer relaciones de herencia entre sí, pero no pueden participar en asociaciones ni tener estados. Cada interfaz acostumbra a especificar sólo una parte del comportamiento de una clase. Una clase puede implementar diferentes interfaces, si al menos una de éstas no posee todas las operaciones visibles de la clase; en cambio, una interfaz sólo es implementada por una clase.

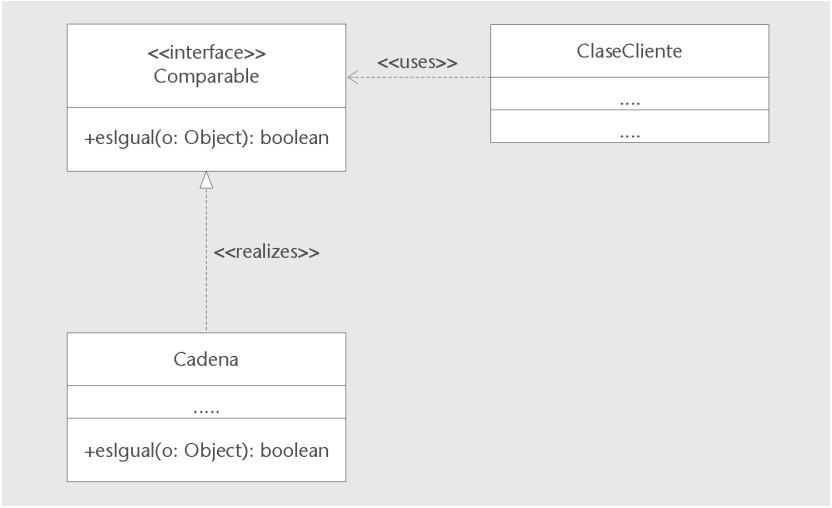
Ved también

Consultad las asociaciones en el subapartado 6.1 de este módulo y los estados en el módulo "UML (II): el modelo dinámico" de esta asignatura.

La notación de la interfaz es como la de la clase pero con el estereotipo *interfaz* y sin el compartimento de atributos, porque no los tiene.

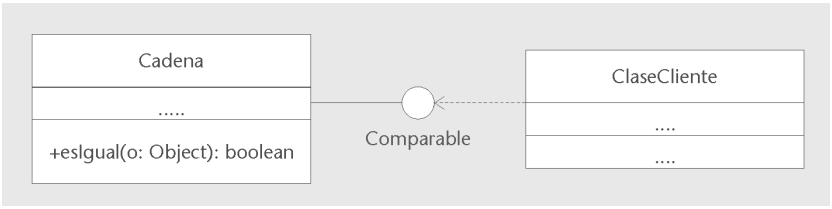
Ejemplo de una interfaz con clase que la utiliza y otra que la implementa

El siguiente ejemplo se ha inspirado en la librería de Eiffel:



La interfaz *Comparable* pretende declarar todas las operaciones que deberán tener las clases que quieran permitir que dos de sus objetos se puedan comparar según algún criterio de igualdad que sólo se define dentro de la implementación de la operación *esIgual* dentro de la clase *Cadena*, que es la que implementa *Comparable*. La clase *ClaseCliente* utiliza la interfaz *Comparable*, en el sentido de que la implementación de alguna operación de *Clase Cliente* llama alguna operación de *Comparable*. ¡Prestad atención a cómo es cada flecha!

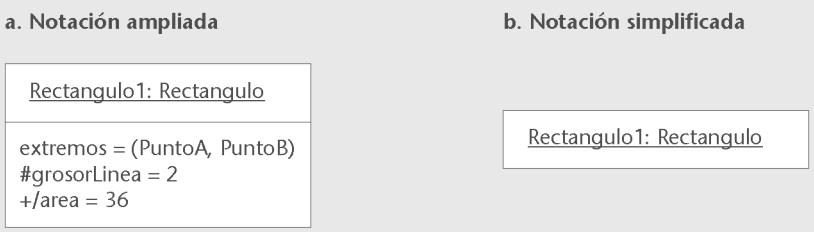
Una notación equivalente más abreviada del mismo ejemplo es ésta:



5. Representación de los objetos

Un objeto se representa gráficamente de una manera muy parecida a las clases: se indican los valores en los atributos de instancia y, opcionalmente, un nombre en el objeto, que va seguido de ":" y del nombre de la clase, todo subrayado. Se puede omitir el tipo de los atributos, así como el compartimento de las operaciones, porque los dos elementos ya se conocen gracias a la especificación de la clase.

Ejemplo de objeto en notación ampliada y en notación simplificada



6. Relaciones entre clases

Sintácticamente, los lenguajes de programación permiten sólo dos tipos de relaciones entre clases: de herencia, que ya hemos visto, y cliente-servidor.

Por **relación cliente-servidor** se entiende que un objeto (el cliente) pida a otro (el servidor), mediante un mensaje, que ejecute una operación de las definidas en la clase del servidor.

Ved también

Consultad las relaciones de herencia entre clases en el subapartado 4.2 de este módulo didáctico.

Por lo que respecta al análisis y el diseño, se consideran otros tipos de relaciones, porque las relaciones cliente-servidor son demasiado simples para describir el mundo real.

Desacuerdo entre autores y el estándar de tipo de relación

Lamentablemente, no existe acuerdo entre los autores más importantes sobre cuáles son estos tipos (y menos todavía sobre su semántica y notación); sin embargo, si UML, convertido en estándar, se acaba imponiendo, es probable que su versión sea aceptada por todos o casi todos. Pero si UML permite modelar gráficamente diferentes tipos de relaciones entre clases (agregación, asociación y uso), el significado de cada uno no queda concretado en los documentos oficiales del OMG y, por lo tanto, la decisión sobre cuándo se debe aplicar un tipo de relación u otro será a menudo una interpretación personal.

6.1. Asociaciones

Dentro de este subapartado veremos los conceptos y terminología utilizados para designar las asociaciones y los diferentes tipos existentes.

6.1.1. Concepto y terminología

Hay una **asociación entre clases** cuando una clase necesita otra u otras para la implementación de sus operaciones, lo cual se cumple por medio del paso de mensajes entre éstas.

Una asociación se define partiendo de la clase, y se concreta en la existencia de **enlaces**[5] entre objetos concretos de las clases relacionadas por la asociación. Dentro de una asociación, se considera que cada clase desempeña un **papel**[6] determinado; cada papel tiene asociada una **cardinalidad**. Entre las mismas clases puede haber asociaciones diferentes con significado diferente.

(5)En inglés, *links*.

(6)En inglés, *role*.

Ved también

La cardinalidad se define en el subapartado 6.1.2 de este módulo didáctico.

Una asociación puede tener nombre, que sirve para identificar su significado, y también se puede dar un nombre a cada uno de los papeles de las clases.

6.1.2. Asociaciones binarias y *n*-arias

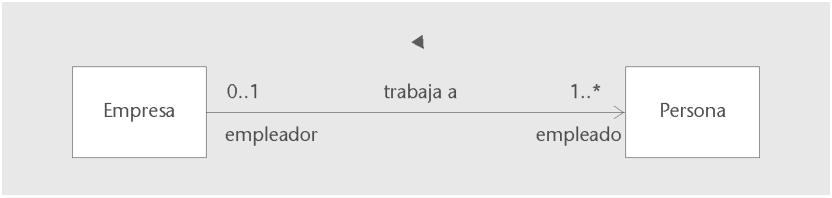
Asociaciones binarias son las que tienen lugar entre dos clases. Las dos clases pueden ser la misma (**asociación reflexiva**), y en este caso es posible permitir que un objeto esté enlazado consigo mismo o que no lo esté.

En una asociación binaria, la **cardinalidad** de un papel *A* es el número de objetos del otro papel *B* al que puede estar enlazado cada objeto de *A*; se indica el valor máximo y mínimo de este número.

Ejemplos de asociaciones binaria y reflexiva

1) Asociación binaria

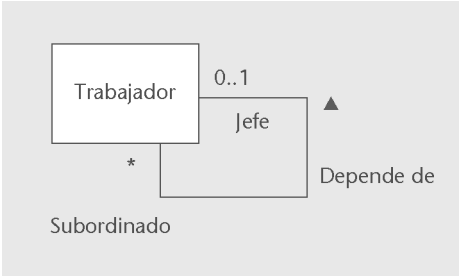
Como se puede apreciar en la siguiente asociación binaria:



La asociación significa que una persona trabaja en una empresa (no al revés, observad el sentido indicado por la punta de flecha coloreada); la empresa es la que ofrece el empleo y la persona desempeña el papel de empleado. Cada persona concreta puede tener una empresa que ofrece el empleo o ninguna, mientras que una empresa puede tener un empleado como mínimo y cualquier número como máximo, según indican las cardinalidades. La punta de flecha abierta encima de la línea de la asociación indica que se puede acceder (navegar) de una empresa hacia sus empleados.

2) Asociación reflexiva

Consideremos la figura siguiente:



El significado de esta asociación es que un trabajador depende de un jefe; tanto el jefe como el subordinado son trabajadores. Cada trabajador puede tener como máximo un jefe, mientras que un jefe puede tener cualquier número de subordinados (el asterisco solo indica que el número puede ser cualquiera, incluso el cero). Un trabajador no puede ser jefe de sí mismo, pero eso no lo indica la notación gráfica.

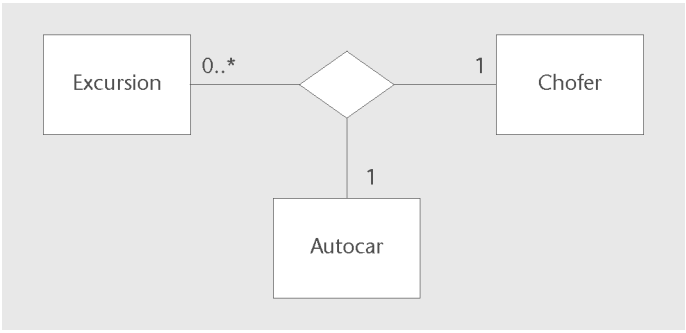
Una **relación ternaria** es aquella que tiene tres papeles[7], y en general una relación *n*-aria es la que tiene *n* papeles. Las relaciones no binarias, ya que no se pueden representar mediante una línea, se representan mediante un rombo.

(7)Esto quiere decir que participan tres clases, diferentes o no.

El significado de la cardinalidad en una asociación ternaria es el siguiente: la cardinalidad del papel *A* expresa los límites al número de objetos de *A* que pueden estar enlazados en cada combinación concreta de un objeto del papel *B* y otro del papel *C*.

Ejemplo de asociación ternaria

Observemos la siguiente asociación:



Un chófer determinado puede conducir un autocar determinado en cualquier número de excursiones ("0..*" es equivalente a "*"), pero en una excursión concreta, un chófer sólo puede conducir un autocar, y en una excursión en particular, un autocar sólo puede tener un chófer.

6.1.3. Clases asociativas

En principio, una asociación no es una clase: no tiene por qué tener atributos ni operaciones, y puede carecer incluso de nombre. No obstante, si una asociación tiene que tener atributos y operaciones propias o bien una de las dos, entonces se debe definir como clase. En este caso se hable de *clase asociativa*.

Establecimiento de enlaces según una asociación

El establecimiento de enlaces entre objetos según una asociación puede ser una operación de alguna de las clases asociadas, igual que la navegación de un objeto a otro enlazado a éste.

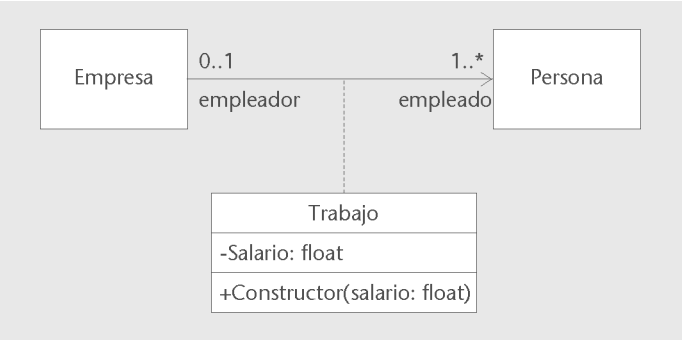
Una clase asociativa se representa como una clase colgada del símbolo de la asociación (la línea, en el caso de una asociación binaria, o el rombo en los otros casos) por medio de una línea discontinua.

Ved también

Consultad el "Ejemplo de asociación binaria" en el subapartado 6.1.2 de este módulo didáctico.

Ejemplo de clase asociativa binaria

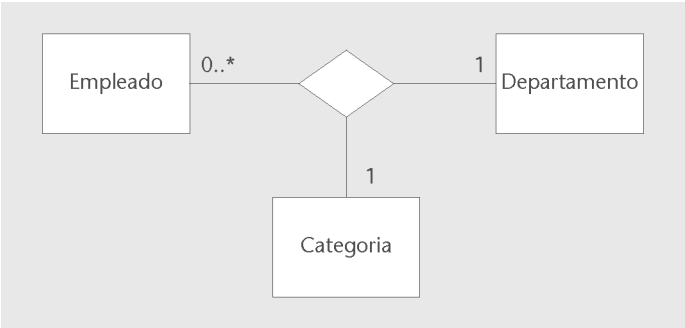
El caso que presentamos aquí es el mismo que el del ejemplo de asociación binaria, excepto por el hecho de que aquí la asociación tiene un atributo. Puesto que la asociación ahora es una clase, se le ha añadido una operación para crear instancias, y también se le ha cambiado el nombre por uno más apropiado para una clase.



6.1.4. Asociaciones calificadas

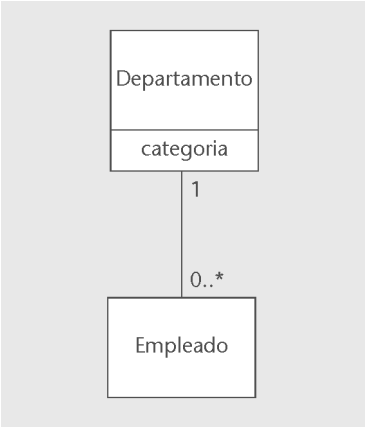
Veremos mejor este concepto si lo introducimos mediante un ejemplo.

Consideremos una clase *Departamento* y otra *Empleados*, que tienen un atributo *categoría*, y queremos establecer una asociación separada entre el departamento y sus empleados para la categoría *A*, la categoría *B*, etc.; una forma de hacerlo es con una relación ternaria, así:



Ejemplo de asociación binaria calificada y de la asociación ternaria equivalente.

Esta solución es un poco forzada, porque obliga a convertir un atributo en una clase independiente, circunstancia que se puede evitar mediante una asociación calificada en la que el atributo calificador sea la categoría del empleado. Esto se representa de la siguiente foma:



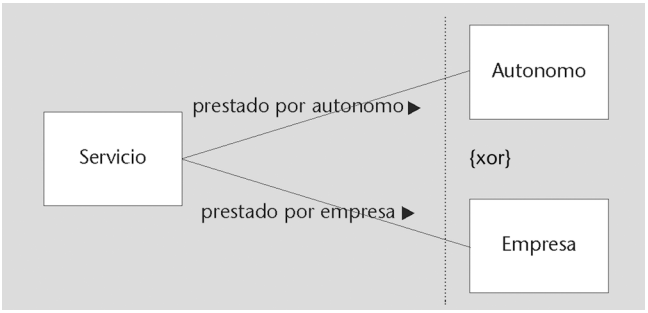
Este diagrama se puede interpretar de esta manera: cada empleado tiene un departamento, y dentro un departamento ha de haber al menos un empleado de cada categoría (mientras que si tuviéramos una asociación binaria ordinaria, prescindiendo de la categoría, cualquier departamento tendría al menos un empleado, de una categoría u otra).

6.1.5. Asociaciones alternativas

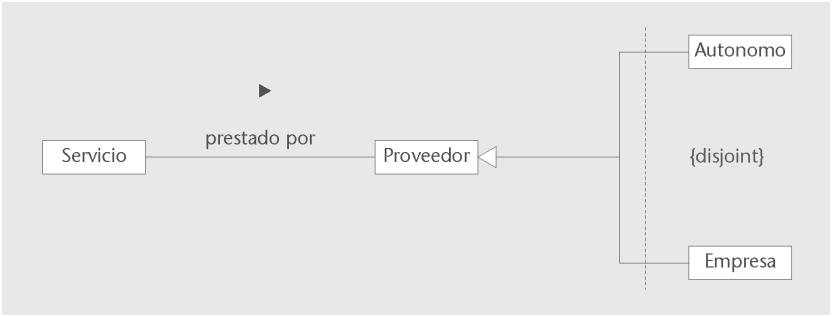
Puede ocurrir que en una clase que participe en dos asociaciones, cada objeto concreto participe en una o en la otra, pero no en las dos. Entonces se habla de *asociaciones alternativas*.

Ejemplo de asociaciones binarias alternativas

Una forma de representar las asociaciones binarias alternativas se puede apreciar en la figura:



Un servicio determinado puede ser prestado por un proveedor autónomo o bien por una empresa, pero nunca por uno y otra a la vez. Esto también se podría describir así:

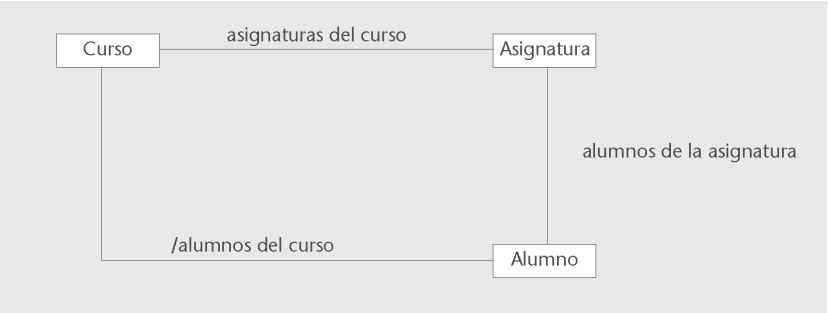


6.1.6. Asociaciones derivadas

Una asociación derivada es una asociación redundante que se puede obtener como combinación de otras relaciones del modelo.

Ejemplo de asociación derivada

En la figura representamos la siguiente asociación derivada:



donde *alumnos del curso* es una asociación derivada, porque los alumnos de un curso se pueden determinar recorriendo las asignaturas del curso y encontrando a los alumnos de cada una. Pero podría ser, también, que esta relación se definiese de una manera no redundante, por ejemplo, que comprendiese a los alumnos que estuviesen matriculados sólo en asignaturas de dicho curso; entonces no se pondría "/", que es el indicador de elemento derivado, como hemos visto al hablar de los atributos derivados en el subapartado 4.1.2.

6.2. Agregaciones y composiciones

En este subapartado consideramos las agregaciones y las composiciones.

6.2.1. Agregaciones

Una agregación[8] es un caso particular de asociación binaria en la que uno de los papeles tiene el significado de 'parte' y el otro, el de 'todo', en algun sentido. Denominaremos *componentes* la clase correspondiente al primer papel y a sus objetos, y *clase* y *objetos agregados*, a los del segundo papel.

(8)Se dice que una agregación es una relación *is-part-of*.

Las **agregaciones** pueden tener diferentes significados:

- **Acoplamiento-piezas**: una máquina y sus piezas, un circuito eléctrico y sus componentes, un sistema de *software* y sus programas; cada parte tiene un papel concreto y no se puede cambiar por ninguna otra.

- **Continente-contenido**: un avión y los pasajeros que transporta, que no constituyen el avión, ya que un avión sin pasajeros es por sí solo un avión.

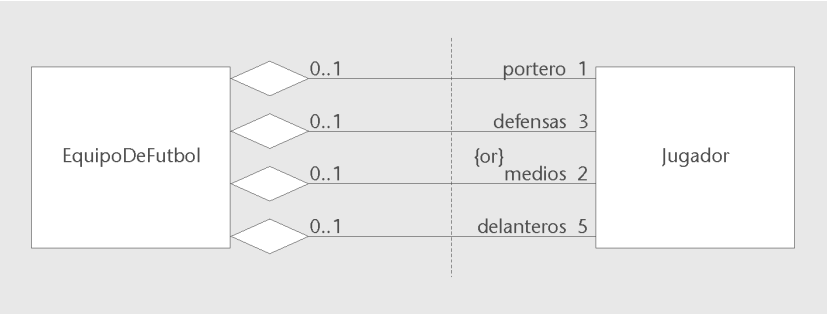
- **Colectivo-miembros**: un grupo excursionista y sus excursionistas, o bien una promoción de alumnos y dichos alumnos; se supone que los miembros no tienen papeles diferenciados y, por tanto, son intercambiables.

Un objeto compuesto puede tener objetos componentes que pertenecen a diferentes clases o, dicho de otra forma, una clase puede tener relaciones de agregación con otras diferentes. Una clase puede desempeñar el papel de clase compuesta en una agregación y el de clase componente en otra, y también es

capaz de realizar los dos papeles en la misma agregación (aunque un objeto no puede ser componente de sí mismo). Entre dos clases puede haber más de una agregación.

Ejemplo de agregaciones

A continuación se representa la composición clásica de un equipo de fútbol:



Cada jugador juega en un equipo (el del club al que pertenece) o en ninguno, y un jugador sólo puede estar en una de las cuatro posiciones, como indica {*or*}. Se supone que las de defensa, por ejemplo, son intercambiables entre sí, o de otro modo habría once agregaciones diferentes. El rombo vacío en la parte de la clase compuesta u objeto compuesto denota que se trata de una agregación.

6.2.2. Composiciones y objetos compuestos

La composición (o agregacióndecomposición) es un caso particular de la agregación.

Nota

En el ejemplo de agregaciones que hemos visto en el subapartado anterior, los jugadores pueden pasar de un equipo a otro y existen por sí mismos incluso cuando no están en ningún equipo. Por tanto, las agregaciones en cuestión no son composiciones.

En una composición, los objetos componentes no tienen vida propia sino que, cuando se destruye el objeto compuesto del que forman parte, también se destruyen. Además, un objeto componente sólo puede formar parte de un objeto compuesto y no puede pasar de un objeto compuesto a otro. Estas restricciones no existen en el caso de agregaciones en general. En una composición denominaremos la clase agregada **clase compuesta**, y el objeto agregado, **objeto compuesto**.

Ejemplos de composición y de objeto compuesto

1) Ejemplo de composición

Consideremos la composición siguiente:

Un centro universitario (facultad, etc.) pertenece a una sola universidad (de hecho, por la parte de la clase compuesta, la cardinalidad en una composición es siempre 1). Una universidad debe tener, al menos, un centro universitario. Suponemos que si desaparece una universidad desaparecen también sus centros universitarios, y que éstos no se trasladan de una universidad a otra.

2) Ejemplo de objeto compuesto

Puesto que en una composición cada objeto componente sólo puede pertenecer a un único objeto compuesto, podemos representar los objetos componentes dentro del objeto compuesto en un compartimento especial.

6.3. Relaciones de dependencia

Una relación de dependencia expresa que un elemento del modelo –denominado **cliente**– depende para su implementación o funcionamiento de otro elemento –denominado **suministrador**[9].

[9]En inglés, *supplier*.

Nota

De la relación de dependencia se suele decir que es una relación *depends_on*.

El símbolo de una relación de dependencia es una flecha de línea discontinua y punta abierta.

Existen diferentes estereotipos estándar, alguno de los cuales ya hemos visto, y se pueden definir otros más. Los estereotipos estándar son los siguientes:

- *access*, *bind*, *import*, *instantiate*, *uses*, que ya hemos visto;
- *derive* significa que un elemento se obtiene de otro por medio de un cálculo o algoritmo;
- *friend* da acceso al cliente a los elementos de visibilidad *private* contenidos en el suministrador;
- *refine* quiere decir que el cliente procede históricamente del suministrador, del cual es una versión nueva o enriquecida (por ejemplo, una clase descrita en el análisis en el que se realizan cambios en el diseño);
- *trace* relaciona elementos que corresponden desde un punto de vista semántico al mismo concepto, como por ejemplo, un elemento y su implementación;
- *call*, *create* y *send*;
- *extend* e *include*, que existen sólo entre casos de uso.

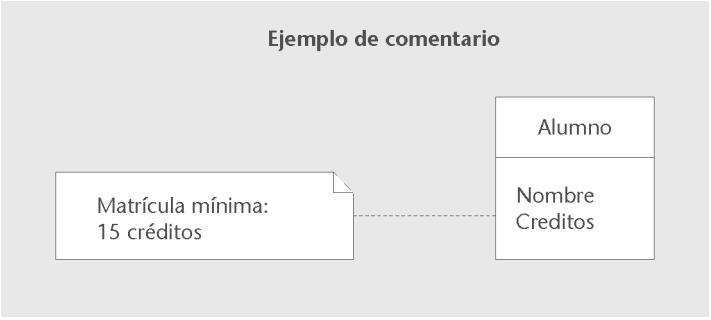
Ved también

Consultad las relaciones entre los casos de uso en el subapartado 2.3 del módulo "UML (II): el modelo dinámico" de esta asignatura.

7. Comentarios y restricciones

7.1. Comentarios

Un comentario se pone dentro de un rectángulo con un vértice doblado, enlazado con un línea discontinua al elemento al que se refiere.



7.2. Restricciones

Las restricciones[10] expresan condiciones que debe cumplir el elemento del modelo al que se asocian. Se representan igual que los comentarios, salvo que van entre llaves {}, lo cual indica que pueden ser interpretadas por las herramientas CASE.

(10) En inglés, *constraints*.

Las especificaciones de UML incluyen un lenguaje para la descripción de las restricciones, denominado OCL[11], pero se puede utilizar UML sin usar este lenguaje.

(11) OCL es la sigla de *Object Constraint Language*.

7.2.1. Las restricciones de las operaciones: la programación por contrato

En UML hay tres tipos de restricciones relativas a las operaciones: precondiciones, postcondiciones e invariantes.

- Las **precondiciones** son restricciones que se deben cumplir antes de ejecutar una operación; su cumplimiento nos garantiza que la operación se ejecuta partiendo de un estado correcto del sistema.

- Las **postcondiciones** se comprueban al acabar la ejecución de una operación, y garantizan que cuando esté terminada la operación, el sistema vuelva a situarse en un estado correcto.

- Las **invariantes** son condiciones que se deben cumplir en todo momento. Se tienen que comprobar al inicio de cualquier operación –excepto los constructores– y al acabar.

Las restricciones pueden servir para diseñar con vistas a hacer **programación por contrato**, que se basa en unas condiciones sobre operaciones y objetos denominadas *aserciones*, que se pueden expresar en forma de restricciones de UML.

Resumen

En este módulo hemos estudiado el modelo estático de UML. Se han descrito los diferentes elementos del modelo, y de cada uno se ha descrito la notación. Estos conceptos son los siguientes:

- Las clases, con sus variantes (clases abstractas) y conceptos relacionados con la clase (clasificador, interfaz) y la herencia.

- Las relaciones entre clases (asociaciones, agregaciones, relaciones de dependencia).

- Los comentarios y las restricciones.

Actividades

1. Un grupo de datos puede estar formado por datos elementales y/o grupos de datos. Representadlo gráficamente.

2. En el departamento comercial de una empresa hay vendedores, de quienes se disponen de cifras de las ventas de cada uno para los seis últimos meses. Representadlo de dos maneras diferentes.

3. Representad una clase *Alumno* con el número de alumnos y una operación para contarlos.

4. En el ejemplo de agregaciones, supongamos que hay jugadores que sólo pueden jugar de porteros, mientras que el resto de jugadores puede estar en cualquier posición salvo en la de portero. ¿Cómo altera este hecho el diagrama?

5. Una empresa de servicios de informática puede tener diferentes proyectos con un mismo cliente, pero cada proyecto pertenece a un único cliente; cada proyecto tiene número y descripción, y los clientes tienen NIF y denominación. Hay dos tipos de empleados, técnicos y jefes de proyecto, todos con NIF y nombre; cada proyecto tiene un jefe y, al menos, un técnico. Un jefe de proyecto puede estar en varios proyectos, pero cada técnico puede estar como máximo en un único proyecto. Realizad el diagrama estático.

Ved también

Consultad el ejemplo de agregaciones en el subapartado 6.2.1 de este módulo didáctico.

Ejercicios de autoevaluación

1. Cuando se hace una generalización, ¿la superclase será siempre una clase abstracta?

2. ¿Cómo se indica que una clase es abstracta?

3. ¿Cuáles son los símbolos y denominaciones de las cuatro opciones estándar de la visibilidad en UML?

4. ¿Cuáles son los tres compartimentos estándar del símbolo de clase?

5. ¿Cómo se sabe si un atributo u operación es de clase o de instancia?

6. ¿Qué significa que haya "/" delante del nombre de un atributo?

7. Si entre dos interfaces existe una relación de especialización, ¿es necesario que estén implementadas por dos clases, una subclase de la otra?

8. En un diagrama, ¿cómo se distingue una composición de una agregación en general?

9. En una asociación, ¿qué diferencia hay entre el sentido de la asociación y el de la navegación?

10. ¿Una relación ternaria equivale a tres relaciones binarias en triángulo?

11. En el ejemplo de las universidades y los centros universitarios, ¿qué pasaría si los centros pudieran pasar de una universidad a una otra?

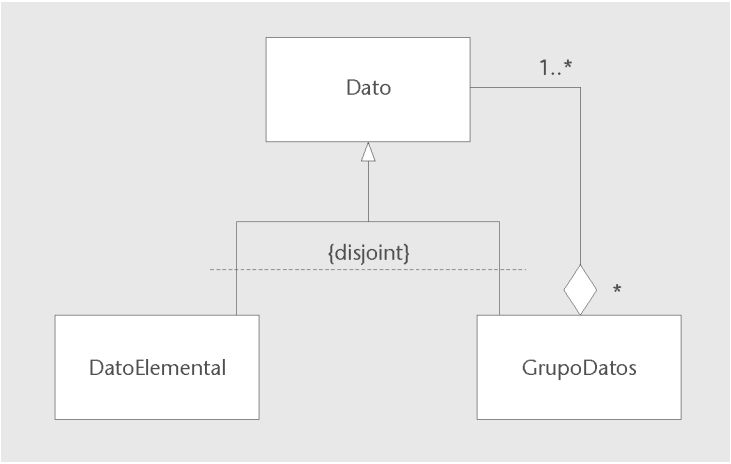
12. ¿Cómo se distingue un comentario de una restricción?

13. Resumid qué significan los diferentes tipos de flechas y líneas que puede haber entre dos clases.

Solucionario

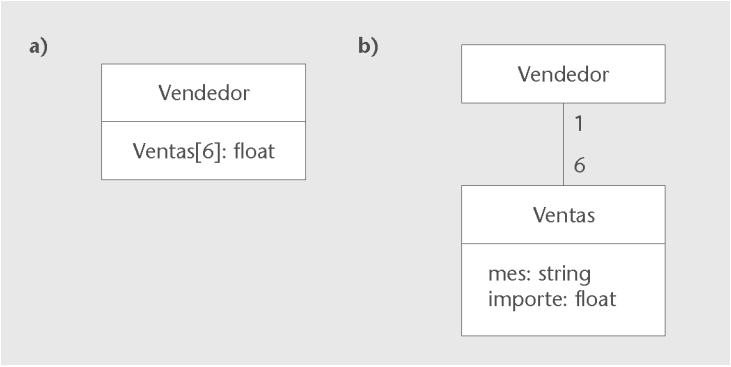
Actividades

1.

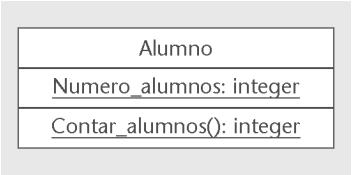


Se ha supesto que un dato puede figurar en diferentes grupos de datos.

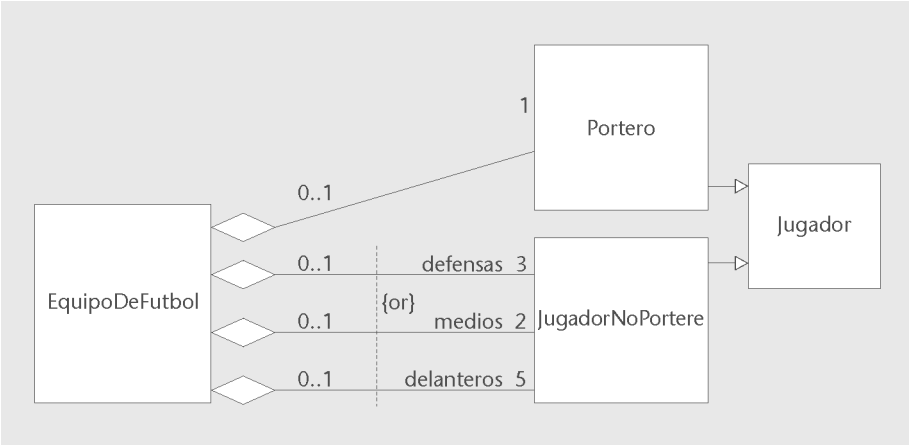
2.



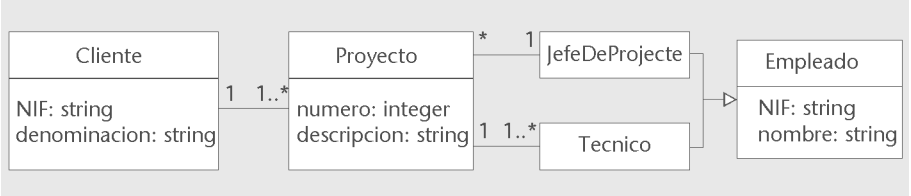
3.



4.



5.



Ejercicios de autoevaluación

1. No necesariamente, porque puede ser que la superclase común a las dos subclases tenga también objetos que no pertenezcan a ninguna de las dos.

2. Existen dos maneras alternativas de hacerlo: con la *property string* {*abstract*}, y poniendo en cursiva el nombre de la clase.

3. privado: "–", protegido: "#" y, público: "+" y de paquete: "~".

4. El del nombre, el de los atributos y el de las operaciones.

5. Si son de clase, su especificación está subrayada.

6. Que es un atributo derivado.

7. No; no hace falta ni siquiera que estén implementadas por dos clases diferentes, sería suficiente con una clase que implementase todas las operaciones de la interfaz subclase, ya que las operaciones de la otra interfaz tienen que estar entre éstas.

8. En una agregación, el rombo está vacío, mientras que en una composición está lleno.

9. El sentido de la asociación tiene un valor semántico; por ejemplo, si tenemos dos clases, *A* y *B*, relacionadas mediante la asociación *pertenece a*, el sentido de la asociación nos indica si es *A* la que pertenece a *B* o viceversa. La punta de flecha que indica navegación en un sentido significa que por medio de la relación se puede acceder a los objetos de la clase de la punta de la flecha desde los de la cola; puede haber navegación en los dos sentidos, mientras que el sentido de la asociación siempre es único.

10. No, porque una asociación ternaria entre las clases *A*, *B* y *C* a un par de objetos, uno de *A* y otro de *B*, le hace corresponder uno o más objetos de *C*, y eso no es ninguna asociación binaria. Lo mismo sucede si permutamos cíclicamente los nombres de las clases.

11. Que ya no se trataría de una composición, sino de una agregación en general.

12. Que las restricciones van entre llaves, hecho que se pretende que eventualmente las pueda hacer interpretables para la herramienta que hace el diagrama.

13. Las clasificaremos primero por tipos de línea y después, por tipos de punta de flecha. El significado indicado corresponde al sentido de la flecha.

- **Línea continua:**

- Punta triangular vacía: subclase de clase.
- Punta abierta: navegación por medio de una asociación o agregación.
- Sin punta: asociación.

- **Línea discontinua:**
  - Punta triangular vacía: la clase implementa la interfaz.
  - Punta abierta: una clase tiene relación de dependencia respecto a la otra.
  - Sin punta: conecta una clase asociativa con la asociación correspondiente (y también enlaza un comentario en el elemento del diagrama al que se aplica).

- **Sin línea:**
  - Punta triangular plena: indica el sentido de una asociación, es decir, el papel de una clase respecto a la otra.

Glosario

asociación binaria Aquélla en la que participan sólo dos clases.

asociación derivada Asociación redundante que resulta de la combinación de otras relaciones.

asociación reflexiva Asociación binaria de una clase consigo misma.

atributo de clase Atributo que tiene un único valor para toda la clase.

atributo derivado Atributo cuyo valor se puede obtener a partir del de otros atributos.

cardinalidad En una asociación binaria, la cardinalidad de un papel *A* es el número de objetos del otro papel *B* al que puede estar enlazado cada objecto de *A*; se indica el valor máximo y mínimo de este número. En una asociación ternaria es lo siguiente: la cardinalidad del papel *A* expresa los límites al número de objetos de *A* que pueden estar enlazados en cada combinación concreta de un objeto del papel *B* y uno del papel *C*.

clase Conjunto de objetos que tienen los mismos atributos y operaciones.

clase abstracta Superclase de la cual no se pueden instanciar objetos porque cada uno de sus objetos debe pertenecer a alguna de sus subclases.

clase asociativa Asociación que, por el mero hecho de tener atributos y/o operaciones propias, llega a ser una clase.

clase diferida Clase abstracta que tiene alguna operación abstracta.
sin.: *clase virtual*.

clase de utilidad Recurso para agrupar procesos y/o datos en forma de una clase que no puede tener objetos.

clase parametrizada Especificación de una clase en la que permanecen sin concretar algunos aspectos de sus operaciones o atributos en forma de parámetros a los que se debe dar un valor.
sin.: *plantilla*.

clase terminal Clase cuyas subclases no se permite que se definan.

clase virtual **Ved** *clase diferida*.

clasificador Concepto que comprende las clases, las interfaces y los tipos de datos.

dominio Porción del mundo real considerada por una aplicación.

especialización Identificación de una o más subclases dentro de una clase.

estereotipo Variante de un elemento de UML. Hay estereotipos estándares, definidos dentro de UML, y se pueden definir como específicos para un proyeto. Los estereotipos se identifican por una palabra clave entre « ».

generalización Definición de una superclase por abstracción de los atributos y operaciones comunes en diferentes clases, que pasan a ser sus subclases.

herencia Característica de las subclases que consiste en tener al menos todos los atributos y operaciones de cada una de sus superclases.

instancia Un objeto es una instancia de su clase. Este concepto se extiende a los clasificadores.

interfaz Conjunto de operaciones de una clase visibles desde otras clases.

modelo estático Modelo que describe las propiedades permanentes del *software* en términos de clases y objetos.

operación abstracta Operación de una superclase que no está implementada en ésta sino sólo en sus subclases.

paquete Elemento del modelo que puede contener elementos de cualquier tipo, incluso paquetes. Sirve para fraccionar el modelo según algún criterio.

plantilla Ved *clase parametrizada*.

visibilidad Propiedad de diferentes elementos del modelo que representa que son reconocidos en un ámbito más o menos grande externo a aquel en el que se han definido.

Bibliografía

Fowler, M. (2004). *UML distilled: A brief guide to the standard object modeling language*. Addison-Wesley.

Esta obra es una muy buena introducción corta a la UML. Tiene un enfoque pragmático más que normativo y es muy aclaratoria. Muestra el uso de todos los diagramas, con ejemplos, y da consejos personales del autor sobre cómo y cuándo hay que utilizar cada parte de la especificación. Pero evidentemente no entra en detalles.

Larman, C. (2005). *Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development*. Prentice Hall.

Larman cubre, en esta obra, el análisis y el diseño de sistemas informáticos usando el lenguaje UML. Da un enfoque ágil y otro de UP, y el libro está escrito de manera iterativa e incremental. Cubre todo el que se explica en este módulo y bastante más, puesto que entra en detalles de diseño e implementación.

Bibliografía complementaria

Booch, G.; Rumbaugh, J.; Jacobson, I. (2005). *The unified modeling language user guide*. Addison-Wesley.

En esta obra los autores de UML escriben una guía de uso del lenguaje de carácter didáctico, más que de referencia. La obra se centra en el lenguaje UML, pero nos da consejos y sugerencias sobre cuándo hay que usar una parte del estándar u otra.

Varios autores (2010). *Unified Modeling Language™ (UML®)*. Object Management Group.

Esta es la especificación de UML. A pesar de su lenguaje formal y normativo, es, evidentemente, la fuente más fiable en caso de duda sobre el lenguaje UML.

Referencias bibliográficas

Génova, G.; Lloréns, J.; Martínez, P. (2001). "Semantics of the minimum multiplicity in ternary associations in UML". *The 4th International Conference on the Unified Modeling Language*. http://www.ie.inf.uc3m.es/ggenova/pub-uml2001.html.

En este artículo los autores dicen que la especificación de UML es ambigua a la hora de explicar la cardinalidad de las asociaciones no binarias y proponen una clarificación.