MODELADO DE OBJETOS

Introducción

El objetivo del presente capítulo es dar a conocer las técnicas UML de modelado estático de objetos.

Dicho modelado se denomina estático porque no describe las interacciones o el ciclo de vida de los objetos, sino que los métodos se introducen desde un punto de vista estático, sin describir su encadenamiento.

Estudiaremos el diagrama de clases. Este diagrama contiene los atributos, métodos y asociaciones de los objetos. Las clases son las que realizan la descripción.

Este diagrama es fundamental para el modelado de un sistema mediante objetos. De todos los diagramas UML, éste es el único obligatorio para ese tipo de modelado.

Veremos de qué manera el lenguaje OCL (*Object Constraint Language* o lenguaje de especificación orientado a objetos) puede extender el diagrama de clases para expresar con mayor riqueza las especificaciones. A continuación, el diagrama de objetos nos mostrará cómo ilustrar la modelización realizada en el diagrama de clases. Por último, descubriremos cómo describir objetos compuestos mediante un diagrama de estructura compuesta.

El uso del OCL, del diagrama de objetos o del diagrama de estructura compuesta es opcional, depende de las especificaciones del proyecto de modelado.

Conocer los objetos del sistema por descomposición

En el capítulo Modelado de la dinámica, estudiamos cómo descubrir los objetos desde un punto de vista dinámico. Primero presentamos los casos de uso en forma de diagrama de secuencias y luego enriquecimos dichos diagramas mediante el envío de mensajes para descubrir los objetos del sistema.

La descomposición de los mensajes hace aparecer los objetos del sistema, ya que conduce a mensajes más finos cuyo destinatario conviene buscar.

Otro posible planteamiento es la descomposición de la información contenida en un objeto. Con frecuencia, esta información es demasiado compleja para ser representada sólo por la estructura de un único objeto. A veces, también debe repartirse entre varios objetos.

Ejemplo

En el ejemplo del capítulo Modelado de la dinámica, el director busca los papeles (la información) de la yegua que desea vender en la base de datos de la granja de cría. La base constituye un objeto de granulado grueso compuesto a su vez por otros objetos, como los papeles de los caballos, las informaciones económicas o contables y los documentos de compraventa de los caballos. Los papeles de una yegua están compuestos, entre otras cosas, por la cartilla de vacunación y los papeles de sus crías. Los papeles de las crías se comparten con otros objetos como, por ejemplo, los papeles del padre semental. Esta descomposición seguía por datos y no por aspectos dinámicos. La figura 6.1 ilustra la composición de `PapelesYegua` *en el diagrama de clases.*

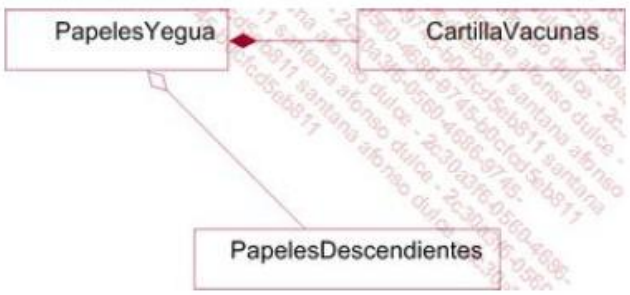


Figura 6.1 - Composición de `PapelesYegua`

NOTA: Recordemos que el granulado de un objeto define su tamaño. El sistema, tomado como un objeto, es de grano grueso o granulado importante. Por el contrario, la cartilla de vacunación de un caballo es un objeto de grano mucho más fino que el sistema.

Ejemplo

La descomposición de un caballo con el fin de mostrar sus diferentes órganos puede hacerse, bien mediante la descomposición de un diagrama de secuencia o bien mediante la descomposición guiada por datos.
La descomposición con el diagrama de secuencia consiste en analizar diferentes envíos de mensajes: dar miedo, correr, comer, dormir. Los mensajes harán aparecer progresivamente los diferentes órganos del caballo. En la figura 6.2, presentamos la descomposición del mensaje `darMiedo`*. Los caballos dilatan las aletas de la nariz cuando están alerta, sorprendidos o tienen miedo. Aprietan la boca cuando están tensos o enfadados. Las coces, finalmente, constituyen un movimiento defensivo.*

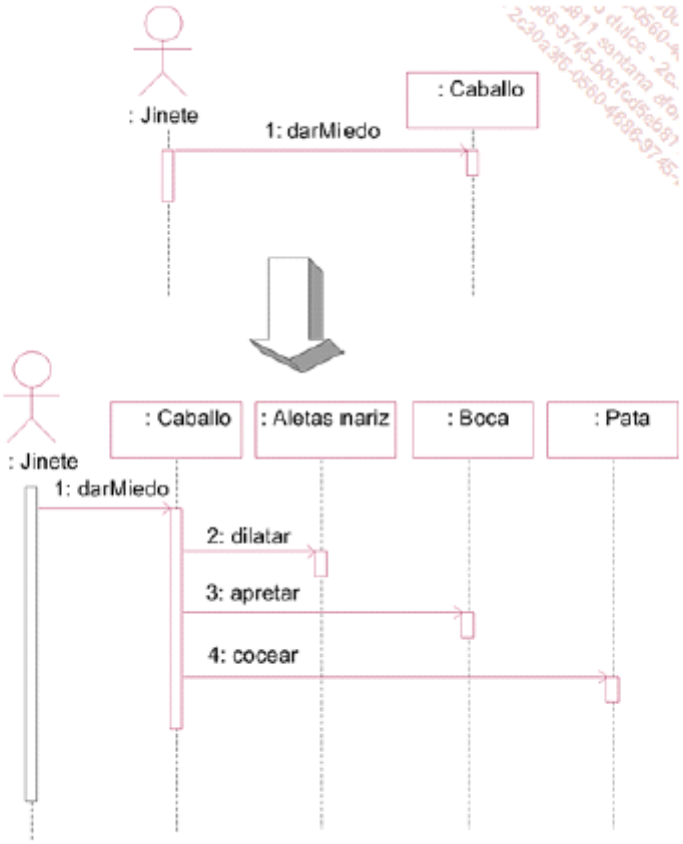


Figura 6.2 - Conocimiento de los objetos mediante enriquecimiento del diagrama de secuencia

La descomposición guiada por datos consiste en estudiar directamente los diferentes órganos de un caballo y tenerlos en cuenta en el diagrama de clases. En la figura 6.3 se representa un caballo compuesto por sus diferentes órganos.

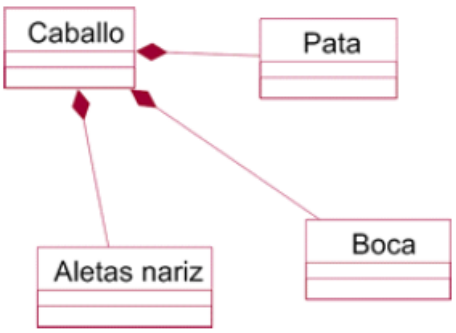


Figura 6.3 - Composición de `Caballo`

NOTA: La descomposición mediante diagramas de secuencia, y la descomposición por datos, son formas naturales de conocer los objetos, lo cual es normal, ya que un objeto es la unión de una estructura y un comportamiento. Por último, conviene destacar que ambos planteamientos no son incompatibles.

La descomposición guiada por datos es más eficaz cuando la persona encargada del modelado conoce bien el tema. La descomposición mediante objetos se realiza entonces de manera inmediata.

Representación de clases

1. La forma simplificada de representación de clases

Los objetos del sistema se describen mediante clases. Presentamos una forma simplificada de representación de las clases en UML en la figura 6.4. La representación consta de tres partes.

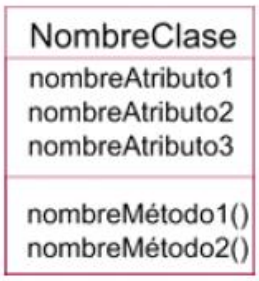


Figura 6.4 - Representación simplificada de una clase en UML

La primera parte contiene el nombre de la clase.

NOTA: Recordemos que el nombre de las clases se escribe en singular y está formado por un nombre común precedido o seguido de uno o varios adjetivos que lo califican. Dicho nombre es representativo del conjunto de objetos que forman la clase, representa la naturaleza de las instancias de una clase.

La segunda parte contiene los atributos. Éstos contienen a su vez la información de los objetos. El conjunto de atributos forma la estructura del objeto.

La tercera parte contiene los métodos, que corresponden a los servicios ofrecidos por el objeto y pueden modificar el valor de los atributos. El conjunto de métodos forma el comportamiento del objeto.

NOTA: El número de atributos y métodos varía de acuerdo con la clase. No obstante, se desaconseja emplear un número elevado de atributos y métodos ya que, en general, éste refleja una mala concepción de la clase.

Ejemplo

La siguiente figura muestra la clase `Caballo` *como ejemplo de representación simplificada de una clase en UML.*

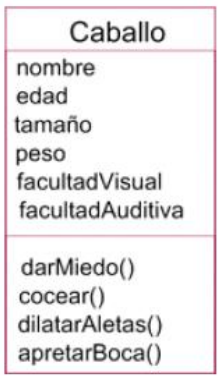


Figura 6.5 - La clase `Caballo`

Esta es la forma más simple de representación de las clases porque no hace aparecer las características de los atributos y de los métodos, a excepción de su nombre. Se utiliza a menudo en las primeras fases del modelado.

Antes de examinar las representaciones más completas, debemos abordar las nociones esenciales de encapsulación, tipo y firma de los métodos.

2. La encapsulación

Introdujimos el concepto de encapsulación en el capítulo Conceptos de la orientación a objetos. Algunos atributos y métodos no se exponen en el exterior del objeto, sino que son encapsulados y reciben el nombre de atributos y métodos privados del objeto.

UML, al igual que la mayoría de lenguajes modernos orientados a objetos, introduce tres posibilidades de encapsulación:

- El atributo privado o el método privado: la propiedad no se expone fuera de la clase, ni tampoco dentro de sus subclases;
- El atributo protegido o el método protegido: la propiedad sólo se expone en las instancias de la clase y de sus subclases;
- La encapsulación de empaquetado: la propiedad sólo se expone en las instancias de clases del mismo empaquetado.

La noción de propiedad privada se utiliza raramente, ya que conduce a establecer una diferencia entre las instancias de una clase y las de sus subclases. Esta diferencia está vinculada a aspectos bastante sutiles de la programación con objetos. La encapsulación de empaquetado, por su parte, procede del lenguaje Java y se reserva a la escritura de diagramas destinados a los desarrolladores.

La encapsulación se representa con un signo más, un signo menos, una almohadilla o una tilde colocados antes del nombre del atributo. En el siguiente cuadro se detalla el significado de los signos.

público	+	elemento no encapsulado visible para todos.
protegido	#	elemento encapsulado visible en las subclases de la clase.
privado	-	elemento encapsulado visible sólo en la clase.
empaquetado	~	elemento encapsulado visible sólo en las clases del mismo empaquetado.

Ejemplo

La siguiente figura muestra la clase `Caballo` *con las características de encapsulación.*

Figura 6.6 - La clase `Caballo` *con las características de encapsulación*

3. Los tipos

En este caso, llamamos variable a cualquier atributo, parámetro o valor de retorno de un método. De manera general, llamamos variable a cualquier elemento que pueda tomar un valor.

El tipo es una especificación aplicada a una variable. Consiste en fijar el conjunto de valores posibles que la variable puede tomar. Dicho conjunto puede ser una clase, en cuyo caso la variable debe contener una referencia a una instancia de la misma y puede ser estándar, como el conjunto de enteros, cadenas de caracteres, booleanos o reales. En estos últimos casos, el valor de la variable debe ser respectivamente un entero, una cadena de caracteres, un valor booleano y un real.

Los tipos estándar se designan del siguiente modo:

- `Integer` para el tipo de los enteros;
- `String` para el tipo de las cadenas de caracteres;
- `Boolean` para el tipo de los booleanos;
- `Real` para el tipo de los reales.

Ejemplo

1 ó 3 ó 10 son ejemplos de valores de entero. «Caballo» es un ejemplo de cadena de caracteres en la cual se ha optado por las comillas como separadores. `False` *y* `True` *son los dos únicos valores posibles del tipo* `Boolean`*.*

3.1415, donde el punto hace la función de separador de decimales, es un ejemplo bien conocido de número real.

Veremos que, en general, solo se recurre a una clase para tipar un atributo si esta es una clase de una biblioteca externa al sistema modelado o una interfaz. Es posible utilizar clases del sistema para dar un tipo a un atributo. Sin embargo, en este caso, es a menudo preferible recurrir a las asociaciones interobjetos.

Por el contrario, el tipo de un parámetro o del retorno de un método puede ser un tipo estándar o una clase, pertenezca o no al sistema.

El tipo de un atributo, de un parámetro y del valor de retorno de un método se especifica en la representación de clase.

Ejemplo

La siguiente figura muestra la clase `Caballo`*, en la cual se ha establecido el tipo de todos los atributos. Los atributos de esta clase utilizan tipos estándar.*

Figura 6.7 - La clase `Caballo` *con el tipo de los atributos*

4. La cardinalidad

Una variable (atributo, parámetro y valor de retorno de un método) puede contener varios valores. En la mayoría de lenguajes de programación, dicha variable se denomina un array o una lista.

La cardinalidad se indica a continuación del tipo con la siguiente sintaxis:

```
[extremoInf..extremoSup]
```

El número de valores que recibe la variable está comprendido entre `extremoInf` y `extremoSup`. Es posible indicar un único extremo para precisar el número de valores que debe recibir exactamente la variable. Es posible utilizar, también, el símbolo * como `extremoSup`. Significa que no existe ningún extremo superior que limite el número de valores que puede recibir la variable. La sintaxis `[0..*]` también puede escribirse `[*]`.

Ejemplo

Supongamos que un caballo puede poseer varios nombres con un mínimo de 1 y un máximo de 3. La sintaxis para el atributo `Nombre` *sería entonces la siguiente:*

```
+ nombre : String[1..3]
```

5. Las propiedades de las variables

UML permite asignar propiedades a una variable (atributo, parámetro y valor de retorno de un método) indicándolo entre llaves. Las siguientes propiedades figuran entre las más utilizadas.

`{readOnly}`: esta propiedad indica que la variable no puede modificarse. Debe inicializarse con un valor por defecto.

`{redefines nombreAtributo}`: esta propiedad solo puede aplicarse a un atributo. Especifica la redefinición del atributo llamado `nombreAtributo` de alguna de las superclases. La redefinición puede realizarse tanto sobre el nombre del atributo como sobre su tipo. En caso de cambio de tipo, el nuevo tipo debe ser compatible con el antiguo, es decir que su conjunto de valores debe estar incluido en el conjunto de valores del antiguo tipo.

`{ordered}`: cuando una variable puede contener varios valores (cardinalidad superior a 1), los valores deben estar ordenados.

`{unique}`: cuando una variable puede contener varios valores (cardinalidad superior a 1), cada valor debe ser único (se prohíbe la existencia de duplicados). Esta propiedad se aplica por defecto.

`{nonunique}`: cuando una variable puede contener varios valores (cardinalidad superior a 1), pueden existir duplicados.

Para asignar varias propiedades a una variable, es preciso separarlas por comas.

Ejemplo

Retomamos el caso en que un caballo puede poseer varios nombres con un mínimo de 1 y un máximo de 3. Queremos, ahora, que el atributo multivalor esté compuesto de nombres únicos y ordenados. Utilizaremos la siguiente sintaxis:

```
+ nombre : String[1..3]{unique, ordered}
```

6. Firma de los métodos

Un método de una clase puede tomar parámetros y devolver un resultado. Los parámetros son valores transmitidos:

- En la ida, al enviar un mensaje que llama a un método;
- O en el retorno de llamada del método.

El resultado es un valor transmitido al objeto que efectúa la llamada cuando ésta se devuelve.

Estos parámetros y el resultado pueden estar tipados. El conjunto constituido por el nombre del método, los parámetros con su nombre, su tipo, su cardinalidad, sus propiedades así como el tipo de resultado con su cardinalidad y sus propiedades se conoce como firma del método.
Una firma adopta la siguiente forma:

```
nombreMétodo (dirección nombreParámetro :
tipo[extremoInf..extremoSup]=ValorPorDefecto{propiedades}, ...) :
tipoResultado[extremoInf..extremoSup]{propiedades}
```

Recordemos que el nombre de los parámetros puede ser nulo y que el tipo de resultado es opcional.

Es posible indicar la dirección en la cual el parámetro se transmite colocando delante del nombre del parámetro una palabra clave. Las tres palabras clave posibles son:

- `in`: el valor del parámetro sólo se transmite al efectuar la llamada;
- `out`: el valor del parámetro sólo se transmite en el retorno a la llamada del método;
- `inout`: el valor del parámetro se transmite en la llamada y en el retorno.

Si no se especifica ninguna palabra clave, el valor del parámetro sólo se transmite en la llamada.

NOTA: Las direcciones `out` y `inout` no son compatibles con llamadas en modo asíncrono en las que aquél que efectúa la llamada no espera el retorno de llamada del método.

Ejemplo

La figura 6.8 muestra la clase `Caballo`, cuyo método `darMiedo` se ha provisto de un parámetro (la intensidad con la que el jinete provoca miedo) y de un retorno (la intensidad del miedo que siente el caballo). Ambos valores son enteros. El resto de métodos no toma parámetros ni devuelve resultados.

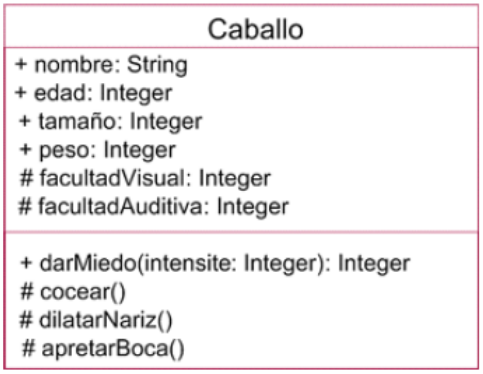


Figura 6.8 - La clase `Caballo` con la firma de los métodos

7. La forma completa de representación de las clases

La representación completa de las clases muestra los atributos con las características de encapsulación, el tipo y los métodos con la firma completa.

También es posible asignar valores predeterminados a los atributos y a los parámetros de un método. El valor predeterminado de un atributo es el que se le atribuye al crear un nuevo objeto. El valor predeterminado de un parámetro se utiliza cuando aquél que llama a un método no proporciona explícitamente el valor del parámetro en el momento de la llamada.

La figura 6.9 ilustra la representación completa de una clase. Por supuesto, es posible escoger una representación intermedia entre la representación simplificada y la representación completa.

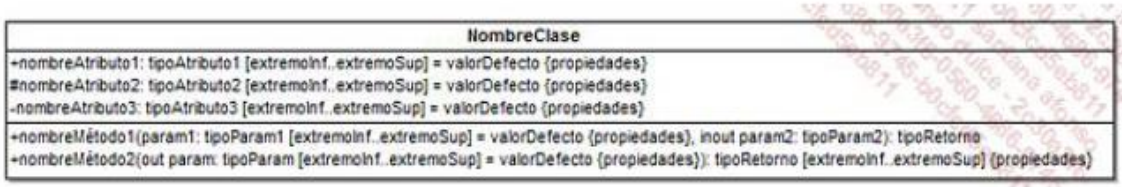


Figura 6.9 - Representación completa de una clase en UML

8. Los atributos y los métodos de clase

Las instancias de una clase contienen un valor específico para cada uno de sus atributos. Este valor, por tanto, no se comparte con el conjunto de instancias. En algunos casos, es preciso utilizar atributos cuyo valor es común a todos los objetos de una clase. Tales atributos comparten su valor al mismo título que su nombre, tipo y valor predeterminado y se conocen como *atributos de clase* porque están vinculados a la clase.

Los atributos de clase se representan mediante un nombre subrayado. Pueden estar encapsulados y poseer un tipo. Se recomienda vivamente asignarles un valor predeterminado.

Ejemplo

Estudiamos el sistema de una carnicería exclusivamente caballar. La siguiente figura introduce una nueva clase que describe una porción de carne de caballo. La venta de este producto está sujeta a una tasa de IVA cuyo montante es similar para todas las porciones. El atributo se destaca y protege, ya que sirve para calcular el precio con el IVA incluido y se expresa en porcentajes, de ahí que su tipo sea `Integer`. El valor predeterminado es 10, es decir 10%.

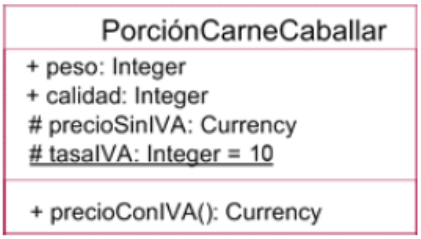


Figura 6.10 - Atributo de clase

`NOTA:` El tipo utilizado para el atributo `precioSinIVA` y el resultado del método `precioConIVA` es `Currency`, que indica que los valores son cantidades monetarias.

Dentro de una clase también pueden existir uno o varios métodos de clase vinculados a la misma. Para llamar a un método de clase, hay que enviar un mensaje a la propia clase y no a una de sus instancias. Los únicos atributos que no se ven afectados por dicho método son los atributos de clase.

Ejemplo

La siguiente figura agrega un método de clase a la clase `PorciónCarneCaballar` que sirve para fijar la tasa de IVA. En efecto, la ley es la que fija dicha tasa y la ley puede modificarse.

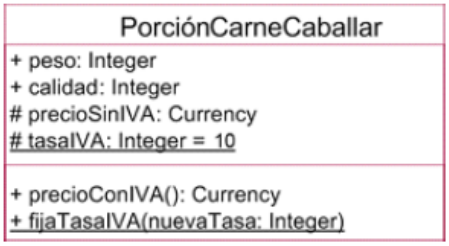


Figura 6.11 - Método de clase

NOTA: En muchas herramientas UML, no se utilizan los términos "atributo de clase" y "método de clase". Estas herramientas dan preferencia a la denominación "atributo estático" o "método estático", denominación que se emplea en lenguajes de programación como C++ o Java.

Los atributos o los métodos de clase no se heredan. La herencia se aplica a la descripción de las instancias, calculada a través de la unión de la estructura y del comportamiento de la clase y de sus superclases. Una subclase puede acceder a un atributo o a un método de clase de una de sus superclases, pero no hereda de ellas. De haber herencia, tendríamos tantos ejemplares de atributos o métodos como subclases poseyera la clase que los introdujo.

Recordemos también que una subclase puede acceder a un atributo o a un método de clase de una de sus superclases, a condición de que no se haya utilizado la encapsulación privada.

9. Los atributos calculados

UML introduce la noción de atributo calculado, cuyo valor viene determinado por una función basada en el valor de otros atributos. Estos atributos poseen un nombre precedido del signo / y van seguidos de una expresión que determina el modo de calcular su valor.

Ejemplo

Retomamos el ejemplo de la clase PorciónCarneCaballar. El método `precioConIVA` *es reemplazado por un atributo calculado* `/precioConIVA`.

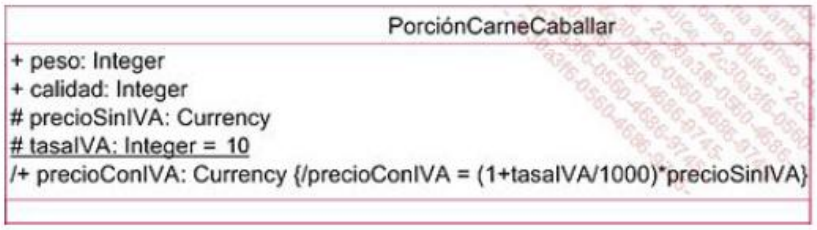


Figura 6.12 - Atributo calculado

Las asociaciones entre objetos

1. Los vínculos entre objetos

En el mundo real, muchos objetos están vinculados entre sí. Dichos vínculos corresponden a una asociación existente entre los objetos.

Ejemplos

- El vínculo existente entre el potro Travieso y su padre;
- El vínculo existente entre el potro Travieso y su madre;
- El vínculo existente entre la yegua Jorgelina y el criadero de caballos al que pertenece;
- El vínculo existente entre el criadero de caballos y su propietario.

En UML, estos vínculos se describen mediante asociaciones, de igual modo que los objetos se describen mediante clases. Un vínculo es un elemento de una asociación. Por consiguiente, una asociación vincula a las clases.

Los elementos de la asociación vinculan entre sí las instancias de las clases.

Las asociaciones tienen un nombre y, como ocurre con las clases, éste es un reflejo de los elementos de la asociación.

Ejemplos

- La asociación padre entre la clase `Descendiente` y la clase `Semental`;
- La asociación madre entre la clase `Descendiente` y la clase `Yegua`;

- La asociación pertenece entre la clase `Caballo` y la clase `CriaderoCaballos`;
- La asociación propietario entre la clase `CriaderoCaballos` y la clase `Persona`.

NOTA: Las asociaciones que hemos estudiado hasta el momento a título de ejemplo establecen un vínculo entre dos clases. Estas asociaciones reciben el nombre de asociaciones binarias. Las asociaciones que vinculan tres clases se denominan asociaciones ternarias y aquellas que vinculan n clases reciben el nombre de asociaciones n-arias. En la práctica, la gran mayoría de asociaciones son binarias y las asociaciones cuaternarias y superiores prácticamente no se usan.

2. Representación de las asociaciones entre clases

La representación gráfica de una asociación binaria consiste en una línea continua que une las dos clases cuyas instancias se vinculan. Las clases se sitúan en los extremos de la asociación.

La siguiente figura muestra la representación de una asociación binaria. El nombre de la asociación se indica encima de la línea.

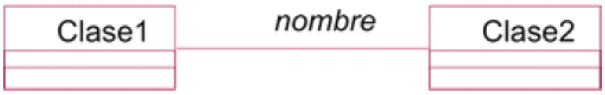


Figura 6.13 - Asociación binaria entre clases

NOTA: Para señalar el sentido de lectura del nombre de la asociación con respecto al nombre de las clases, éste puede precederse del signo < o seguirse del signo >. Si la asociación se sitúa en un eje vertical, el nombre puede ir precedido de ^ o de v.

Los extremos de una asociación también pueden recibir un nombre. Dicho nombre es representativo de la función que desempeñan en la asociación las instancias de la clase correspondiente. Una función tiene la misma naturaleza que un atributo cuyo tipo sería la clase situada en el otro extremo. Por consiguiente, puede ser pública o estar encapsulada de manera privada, protegida o visible únicamente en el empaquetado. Cuando se especifican las funciones, muchas veces no es preciso indicar el nombre de la asociación, ya que éste suele ser el mismo que el de una de las funciones.

La siguiente figura ilustra la representación de una asociación binaria mostrando las funciones.

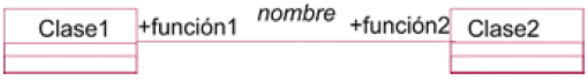


Figura 6.14 - Funciones de una asociación binaria

Ejemplo

La siguiente figura muestra la representación gráfica de las asociaciones introducidas en el ejemplo precedente.

En estas asociaciones se ha indicado el nombre de la asociación o bien sus funciones.

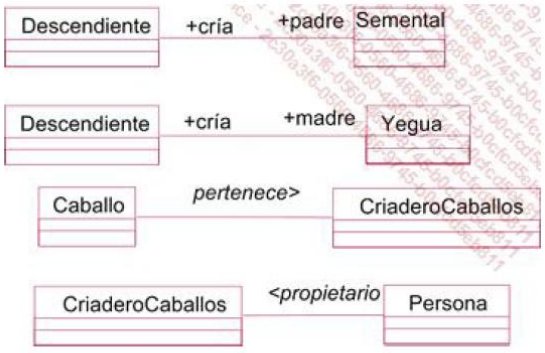


Figura 6.15 - Ejemplos de asociaciones binarias

La representación gráfica de una asociación ternaria y superiores consiste en un rombo que une las diferentes clases. La siguiente figura ilustra la representación de una asociación ternaria.

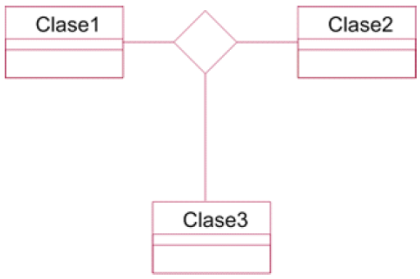


Figura 6.16 - Asociación ternaria entre clases

Ejemplo

La asociación `familia` *que vincula las clases* `Semental`, `Yegua` *y* `Descendiente` *se presenta en la siguiente figura. Cada uno de los elementos constituye un triplete (padre, madre, potro).*

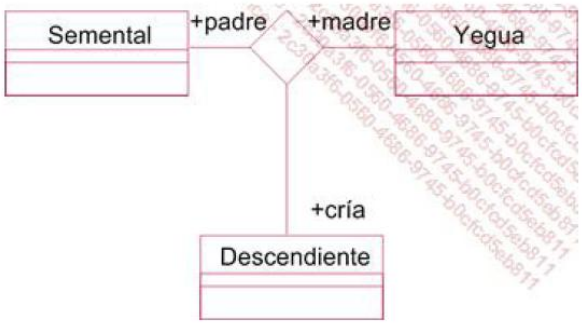


Figura 6.17 - Ejemplo de asociación ternaria

3. La cardinalidad de las asociaciones

La cardinalidad situada en un extremo de una asociación indica a cuántas instancias de la clase situada en ese mismo extremo está vinculada una instancia de la clase situada en el extremo opuesto.

En uno de los extremos de la asociación, es posible especificar la cardinalidad mínima y la máxima con el fin de indicar el intervalo de valores al que deberá pertenecer siempre la cardinalidad.

La sintaxis de especificación de las cardinalidades mínima y máxima se describe en el siguiente cuadro.

Especificación	Cardinalidades
0..1	cero o una vez
1	únicamente una vez
*	de cero a varias veces
1..*	de una a varias veces
M..N	entre M y N veces
N	N veces

NOTA: De no existir una especificación explícita de las cardinalidades mínima y máxima, éstas valen 1.

Ejemplo

En la siguiente figura, retomamos las asociaciones de la figura anterior y les agregamos las cardinalidades mínima y máxima de cada asociación. Un criadero de caballos puede tener varios propietarios y una persona puede ser propietario de varios criaderos.

A modo de ejemplo, la primera asociación se lee de la siguiente manera: un descendiente posee un solo padre, un semental puede tener de cero a varios potros o crías.

Figura 6.18 - Ejemplos de asociaciones descritas con sus cardinalidades

4. La navegación

Por defecto, las asociaciones tienen una navegación bidireccional, es decir, es posible determinar los vínculos de la asociación desde una instancia de cada clase de origen. Las navegaciones bidireccionales resultan más complejas de realizar para los desarrolladores y, en la medida de lo posible, conviene evitarlas.

Para especificar el único sentido útil de navegación durante las fases de modelado cercanas al paso al desarrollo se dibuja la asociación en forma de flecha.

Ejemplo

En el contexto concreto de un criadero, resulta útil conocer los caballos que posee dicho criadero, pero lo contrario, a saber conocer los criaderos que poseen algún caballo, no resulta útil. La siguiente figura muestra la asociación resultante con el sentido de navegación adecuado.

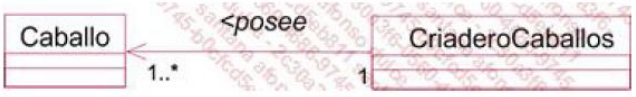


Figura 6.19 - Ejemplo de navegación

5. La asociación reflexiva

Cuando encontramos una misma clase en los dos extremos de una asociación, hablamos de asociaciones reflexivas que unen entre sí las instancias de una misma clase.

En estos casos, resulta preferible asignar un nombre a la función desempeñada por la clase en cada extremo de la asociación.

Las asociaciones reflexivas sirven principalmente para describir dentro del conjunto de instancias de una clase:

- Grupos de instancias;
- Una jerarquía dentro de las instancias.

NOTA: Para los lectores expertos diremos que, en el primer caso, se trata de una asociación que representa una relación de equivalencia y, en el segundo, una asociación que representa una relación de orden.

Ejemplo

Para poder superar las pruebas de selección de un concurso hípico internacional, es preciso que los caballos hayan ganado otros concursos previos. Podemos crear, por consiguiente, una asociación entre el concurso internacional y los concursos celebrados previamente a él. La siguiente figura muestra dicha asociación, que crea una jerarquía dentro de los concursos.

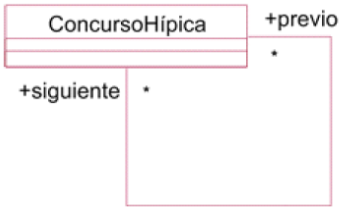


Figura 6.20 - Asociación reflexiva entre concursos hípicos

Ejemplo

La siguiente figura muestra la asociación "ascendente/descendiente directo" entre los caballos. Esta asociación crea una jerarquía dentro de los caballos.
La cardinalidad para los ascendentes es 2, ya que cualquier caballo tiene un padre y una madre.

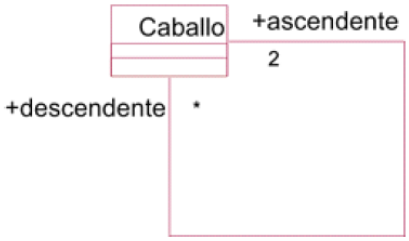


Figura 6.21 - Asociación "ascendiente/descendiente directo" entre caballos

Ejemplo

La siguiente figura muestra la asociación entre los caballos que se encuentran en el mismo criadero. Esta asociación crea grupos dentro del conjunto de instancias de la clase `Caballo`*, ya que cada grupo corresponde a un criadero.*

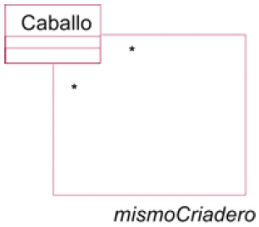


Figura 6.22 - Asociación reflexiva que vincula los caballos de un mismo criadero

6. Las propiedades de los extremos de las asociaciones

Los extremos de las asociaciones pueden poseer propiedades de manera similar a los atributos. Las principales propiedades de los extremos de las asociaciones son las siguientes:

`{ordered}`: cuando un extremo posee una cardinalidad superior a 1, las ocurrencias deben estar ordenadas.

`{nonunique}`: esta propiedad permite indicar que una instancia situada en el extremo donde se encuentra la propiedad puede estar vinculada varias veces a una instancia situada en el otro extremo. La cardinalidad debe ser superior a 1. Esta propiedad no se utiliza por defecto.

Ejemplo

La siguiente figura ilustra el uso de ambas propiedades. La parte superior muestra la descripción de un conjunto ordenado. En un conjunto, un mismo elemento no puede aparecer más de una vez. De manera opuesta, en una lista, un mismo elemento puede aparecer varias veces. En la parte inferior de la figura se describe una lista ordenada. Se utiliza, sin embargo, la propiedad `{nonunique}`.

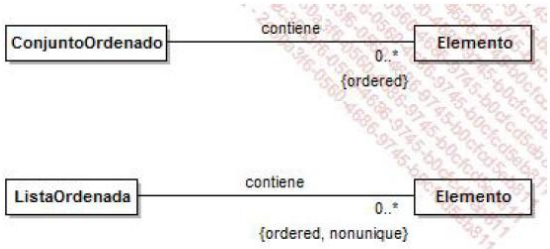


Figura 6.23 - Representación gráfica de las propiedades de los extremos de las asociaciones

7. Las clases- asociaciones

Los vínculos entre las instancias de las clases pueden llevar informaciones. Éstas son específicas a cada vínculo.

En esos casos, la asociación que describe los vínculos recibe el estatus de clase y sus instancias son elementos de la asociación.

Al igual que el resto, estas clases pueden estar dotadas de atributos y operaciones y estar vinculadas a otras clases a través de asociaciones.

La siguiente figura representa gráficamente una clase-asociación que se une a la asociación mediante una línea discontinua.

Figura 6.24 - Representación gráfica de una clase-asociación

Ejemplo

Cuando un cliente compra productos para caballos (productos de mantenimiento, etc.) conviene especificar la cantidad de productos adquiridos mediante una clase-asociación, aquí denominada clase `Adquisición`.

Figura 6.25 - Clase-asociación `Adquisición`

8. La calificación de las asociaciones

En caso de cardinalidad máxima no finita en un extremo de la asociación, si las instancias situadas en dicho extremo son calificables, la calificación puede usarse para pasar de la cardinalidad máxima no finita a una cardinalidad máxima finita.

La calificación de una instancia es un valor o un conjunto de valores que permiten encontrar dicha instancia. Muchas veces, las calificaciones son índices para, por ejemplo, encontrar un elemento en una tabla, o llaves para, por ejemplo, localizar una línea en una base de datos relacional.

La calificación se inserta en el extremo opuesto a donde se encuentra la cardinalidad máxima. Esta calificación se presenta bajo la forma de uno o varios atributos, que califican las instancias de la clase 2 en el ámbito de la clase 1.

Figura 6.26 - Representación gráfica de una calificación

Ejemplo

Una tabla está formada por elementos. La siguiente figura describe dos posibilidades de modelado en UML. La primera no recurre a la calificación, mientras que en la segunda el calificador `índice` *permite encontrar un único elemento de la tabla. Por consiguiente, la cardinalidad máxima pasa a 1.*

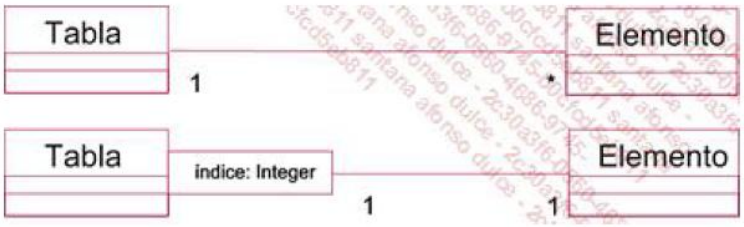


Figura 6.27 - Calificación de los elementos de una tabla

Ejemplo

En las carreras de caballos, todos los caballos poseen un número. La siguiente figura muestra a los participantes en una carrera calificándolos por su número.

Figura 6.28 - Calificación de los participantes en una carrera hípica

9. La expresión de las especificaciones en las asociaciones

UML ofrece la posibilidad de expresar las especificaciones gracias a ciertas construcciones del modelado objeto que ya hemos estudiado: las cardinalidades, el tipo de un atributo, etc.

UML introduce una especificación denominada `{XOR}` (o exclusivo) entre dos asociaciones que poseen al menos en uno de sus extremos una clase común. La especificación expresa que cada instancia de la clase común no puede participar en ambas asociaciones.

Ejemplo

Una comida para un caballo contiene los siguientes elementos:

- *Paja;*
- *Minerales;*

- *Heno o grano.*

La siguiente figura ilustra la comida y sus diferentes constituyentes. Que contenga o bien heno, o bien grano es una restricción expresada mediante el operador `XOR`*. Esta expresa la especificación del o exclusivo entre ambas asociaciones.*

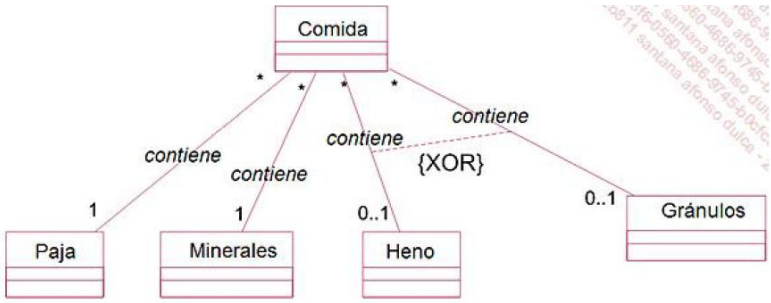


Figura 6.29 - Expresión de la exclusión entre dos asociaciones

Es posible establecer otra especificación entre dos asociaciones que poseen las mismas clases en cada extremo. Se trata de la especificación `{subset}` (subconjunto) que permite indicar que el conjunto de ocurrencias de una de las asociaciones está incluido en el conjunto de ocurrencias de la otra asociación. La representación gráfica es comparable a la de la restricción `{XOR}` con una línea discontinua que va de la asociación de tipo subconjunto hacia la otra asociación.

Ejemplo

La asociación `miembroDirección` *entre una empresa y sus empleados es una asociación de tipo subconjunto de la asociación* `miembro` *entre los empleados y dicha sociedad.*

La siguiente figura ilustra la especificación `{subset}` *entre ambas asociaciones.*

Figura 6.30 - Expresión de subconjunto entre dos asociaciones

No obstante, las especificaciones de este tipo pueden resultar insuficientes. UML propone expresar otras especificaciones en lenguaje natural o en OCL.

OCL es un lenguaje de especificación objeto en forma de condiciones lógicas. Forma parte del conjunto denotaciones UML.

Ya se escriban en lenguaje natural o bien en OCL, las especificaciones se representan en las notas incluidas dentro del diagrama de clases.

Las especificaciones escritas en OCL se expresan sobre el valor de los atributos y de las funciones (extremos de las asociaciones). Las especificaciones deben tener un valor lógico (verdadero o falso).

Para construir una especificación se utilizan todos los operadores aplicables al valor de los atributos en función de su tipo (comparación de enteros, suma de enteros, comparación de cadenas, etc.). En las condiciones pueden emplearse los métodos de los objetos. Para los valores de conjunto (colecciones de objetos), OCL propone un juego de operadores: union, intersection, diferencia. Para las colecciones de objetos, OCL propone sobre todo los operadores siguientes: `collect`, `includes`, `includesAll`, `asSet`, `exists`, `forAll`.

Para designar un atributo o un método en una expresión OCL, hay que elaborar una expresión de ruta que empiece por `self`. A continuación, se designa directamente el atributo o el método por su nombre o se recorre una asociación utilizando el nombre de la función. Es conveniente entonces designar un atributo o un método de la clase situada en el otro extremo de la asociación,

o bien designar de nuevo una función para recorrer otra asociación hasta elegir un atributo o un método.

NOTA: Recorrer una asociación consiste en partir de uno de sus extremos para recuperar las instancias presentadas hasta el otro extremo.

La sintaxis de una expresión de ruta es la siguiente:

```
self.atributo
o
self.método
o
self.función.función. ... .función.atributo
o
self.función.función. ... .función.método
```

Si una especificación OCL no está incluida directamente en el diagrama de clases, entonces hay que precisar sucontexto, según la siguiente sintaxis:

```
Contexto Clase inv Especificación:
```

Ejemplo

Un hipódromo hace correr a varios caballos. Los jockeys que ha contratado deben participar en las carreras que organiza. Esto puede reformularse así: el conjunto de jockeys empleados por el hipódromo debe estar incluido en el conjunto de jockeys que montan los caballos participantes en las carreras del hipódromo.

En OCL, esta especificación se escribe así:

```
Contexto Hipódromo inv jockeysEmpleados:

self.participaCarrera->collect(c | c.monta)->includesAll
(self.empleado)
```

La especificación se apoya en la utilización de las expresiones de camino en OCL. La figura 6.31 muestra el diagrama de clases donde la especificación escrita en OCL se incluye directamente.

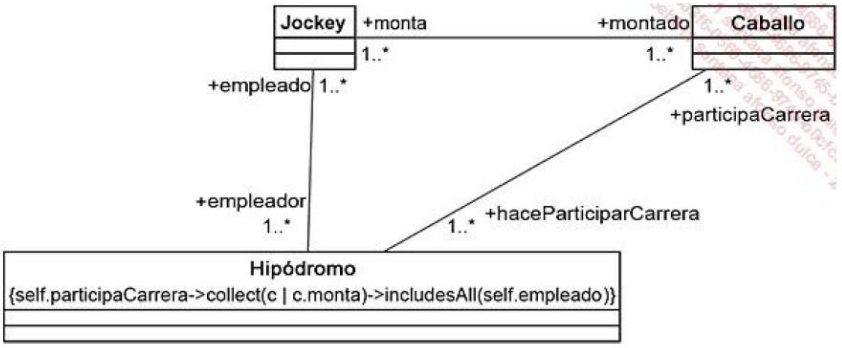


Figura 6.31 - Especificación con expresiones de camino en OCL

10.Los objetos compuestos

Un objeto puede estar compuesto por otros objetos.

En tales casos, nos encontramos ante una asociación entre objetos llamada *asociación de composición*. Ésta asocia un objeto complejo con los objetos que lo constituyen, es decir, sus componentes.

Existen dos formas de composición: fuerte o débil.

a. La composición fuerte o composición

La composición fuerte es una forma de composición en la que los componentes constituyen una parte del objeto compuesto. De esta forma, los componentes no pueden ser compartidos por varios objetos compuestos. Por tanto, la cardinalidad máxima, a nivel del objeto compuesto, es obligatoriamente uno.

La supresión del objeto compuesto comporta la supresión de los componentes.

La siguiente figura muestra la representación gráfica de la asociación de composición fuerte. A nivel del objeto compuesto, la cardinalidad mínima indicada es 0, pero también podría ser 1.

Figura 6.32 - Asociación de composición fuerte

NOTA: En adelante, la asociación de composición fuerte será denominada simplemente composición.

Ejemplo

Un caballo está compuesto, entre otras cosas, por un cerebro. El cerebro no se comparte. La muerte del caballo comporta la muerte del cerebro. Se trata, por tanto, de una asociación de composición.

Ejemplo

Una carrera hípica está constituida por premios. Los premios no se comparten con otras carreras (un premio es específico de una carrera). Si la carrera no se organiza, los premios no se atribuyen y desaparecen. Se trata de una relación de composición.

b. La composición débil o agregación

La composición débil, llamada habitualmente agregación, impone muchas menos especificaciones a los componentes que la composición fuerte. En el caso de la agregación, los componentes pueden ser compartidos por varios compuestos (de la misma asociación de agregación o de varias asociaciones de agregación distintas) y la destrucción del compuesto no conduce a la destrucción de los componentes.

La agregación se da con mayor frecuencia que la composición. En las primeras fases de modelado, es posible utilizar sólo la agregación y determinar más adelante qué asociaciones de agregación son asociaciones de composición.

Ejemplo

Un caballo enjaezado está compuesto, entre otras cosas, por una silla. Una silla está compuesta por una cincha, estribos y una manta de montura. Esta composición es una muestra de agregación. En efecto, la pérdida del caballo no acarrea la pérdida de los objetos y la pérdida de la silla no acarrea la pérdida de sus componentes.

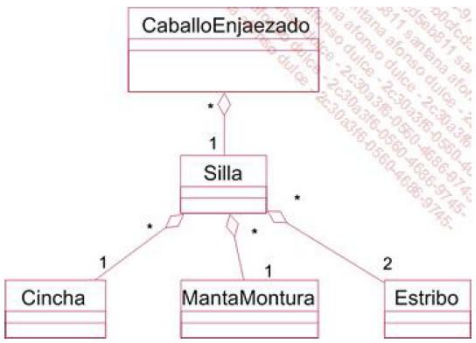


Figura 6.35 - Asociación de agregación entre un caballo enjaezado y su equipamiento y entre una silla y sus componentes

Ejemplo

Un propietario ecuestre posee una colección de caballos. Un caballo domesticado pertenece a una sola colección y puede simultáneamente ser confiado a un criadero o no serlo. Por tanto, puede ser componente de dos agregaciones. La siguiente figura muestra las dos asociaciones.

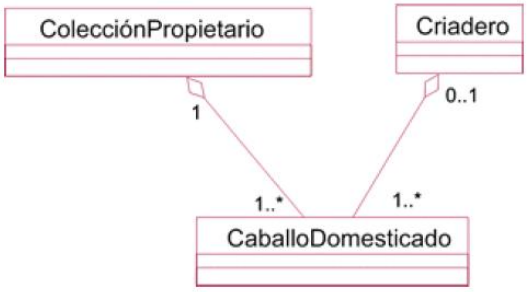


Figura 6.36 - Componente compartido por varias agregaciones distintas

Ejemplo

Un caballo puede pertenecer a varios propietarios. En ese caso, la cardinalidad a nivel de la colección del propietario ya no es 1, sino 1.. para expresar la multiplicidad. El caballo puede entonces ser compartido varias veces en una misma agregación.*

Figura 6.37 - Componente compartido varias veces en una misma agregación

c. Las diferencias entre composición y agregación

La siguiente tabla resume las diferencias entre agregación y composición.

	Agregación	Composición
Representación	rombo transparente	rombo negro
Varias asociaciones comparten los componentes	sí	no
Destrucción de los componentes al destruir el compuesto	no	sí
Cardinalidad a nivel del compuesto	cualquiera	0..1 ó 1

11. La relación de dependencia

La relación de dependencia indica que una clase cliente necesita otra clase proveedora para su propia especificación o para su realización.

Gráficamente, la relación de dependencia se representa mediante una flecha discontinua que va desde la clase `Cliente` hasta la clase `Proveedor`.

Ejemplo

La clase `Cliente` posee un método donde el tipo de alguno de los parámetros o el tipo de retorno es la clase `Proveedor`.

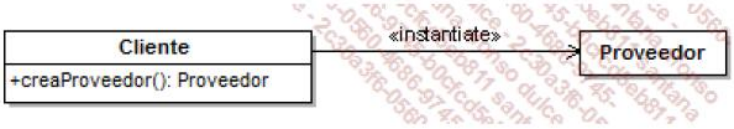


Figura 6.38 - Relación de dependencia

Una relación de dependencia puede estar dotada de un estereotipo para definir su semántica. Los estereotipos más importantes son los siguientes:

«call»: la implantación de la clase `Cliente` invoca a un método de la clase `Proveedor`.

«create»: la implantación de la clase `Cliente` crea una instancia de la clase `Proveedor` para su uso interno.

«derive»: la especificación y la implantación de la clase `Cliente` se obtienen únicamente a partir de la especificación y de la implantación de la clase `Proveedor`. El cliente es, por tanto, una redundancia que se ha construido, por ejemplo, por motivos de optimización o de facilidad de uso.

«instantiate»: el cliente es una fábrica de objetos del proveedor. Una fábrica es una clase cuyo único objetivo es crear instancias de una o varias clases. La figura 6.38 ilustra una fábrica. En efecto, la clase `Cliente` posee un método público `creaProveedor` que crea una nueva instancia de la clase `Proveedor`.

«permit»: la clase `Proveedor` autoriza a la clase `Cliente` a acceder a varios o a la totalidad de sus atributos privados y/o de sus métodos privados.

«refine»: la clase `Cliente` especializa la clase `Proveedor`. El uso de este estereotipo en lugar de la relación de especialización se implementa a menudo tras las primeras etapas de diseño de los diagramas UML de un sistema, antes de reemplazar esta relación por la relación de especialización.

«use»: se trata del estereotipo más general que especifica que la clase `Cliente` necesita, sin proporcionar más precisión, la clase `Proveedor`.

Relación de generalización/especialización entre clases

1. Las clases más específicas y las clases más generales

Una clase es más específica que otra si todas las instancias que la componen son a su vez instancias de la otra clase. La clase más específica es una subclase de la otra clase. Esta última, más general, recibe el nombre de superclase.

La siguiente figura muestra ambas clases así como la relación de generalización/especialización que las une.

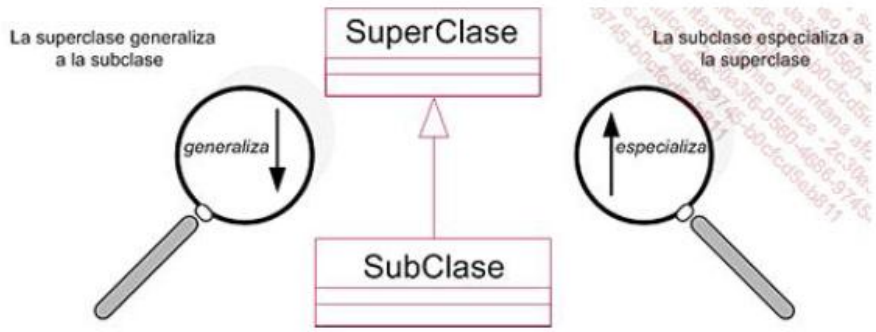


Figura 6.39 - Subclase y superclase

Ejemplo

El caballo es una especialización del équido, que a su vez es una especialización del animal. La zebra es otra especialización del équido. El resultado es la minijerarquía presentada en la siguiente figura.

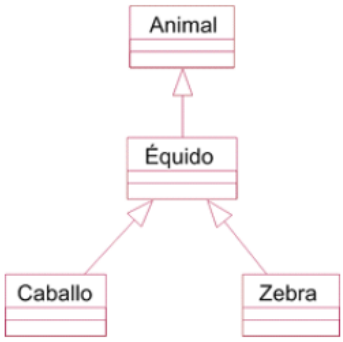


Figura 6.40 - Jerarquía de clases

2. La herencia

Las instancias de una clase son también instancias de su superclase o superclases. Por consiguiente, aprovechan los atributos y métodos definidos en la superclase, además de los atributos y métodos introducidos en la clase.

Esta facultad se conoce como herencia, es decir, una clase hereda los atributos y métodos de las superclases para que sus instancias se beneficien de ellos.

UML ofrece la posibilidad de representar los atributos y los métodos heredados en las subclases precediéndolas del símbolo ^.

NOTA: Recordemos que los atributos y métodos privados de una superclase se heredan en sus subclases, pero no son visibles en ellas.

En la herencia, los métodos o atributos pueden redefinirse en la subclase. Esta redefinición se aplica principalmente a la herencia de las clases abstractas.

Ejemplo

Tal y como se muestra en la siguiente figura, los atributos y métodos de la clase `Caballo` *se heredan en sus dos subclases.*

La herencia significa que, al igual que un caballo de tiro, un caballo de carreras posee un nombre, un peso, una edad y puede caminar y comer.

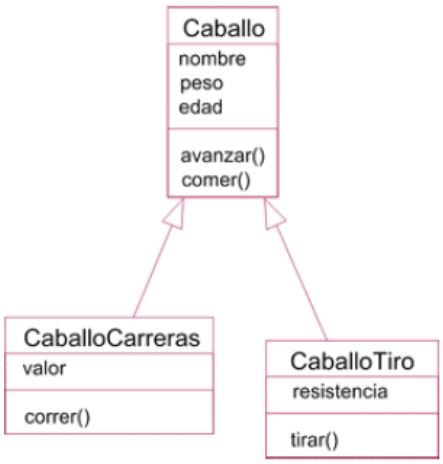


Figura 6.41 - Herencia de atributos y métodos

En la siguiente figura aparece la misma jerarquía indicando los atributos y los métodos heredados que están precedidos del símbolo^. También se han agregado el tipo y la encapsulación de los atributos y de los métodos.

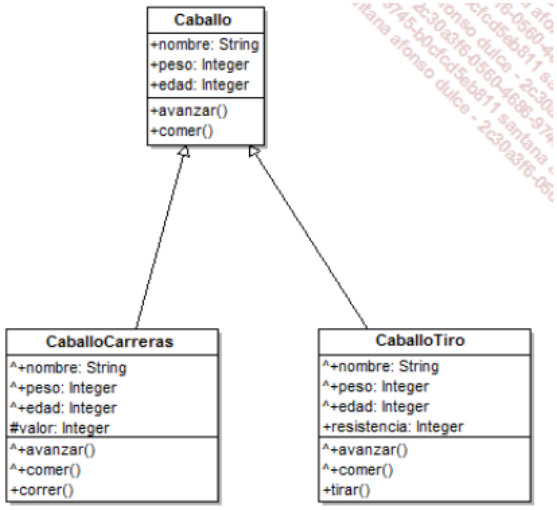


Figura 6.42 - Jerarquía que muestra los atributos y métodos heredados

3. Las clases concretas y abstractas

La siguiente figura muestra la existencia de dos tipos de clases en la herencia: las clases concretas `Caballo` y `Lobo`, que aparecen en la parte más baja de la jerarquía, y la clase abstracta `Animal`.

Una clase concreta posee instancias y constituye un modelo completo de objeto (todos los atributos y métodos se describen completamente).

Por el contrario, una clase abstracta no puede poseer una instancia directa ya que no proporciona una descripción completa. Su finalidad es poseer subclases concretas y sirve para factorizar los atributos y métodos comunes a las subclases.

Con frecuencia la factorización de métodos comunes a las subclases se traduce en la factorización únicamente de la firma. Los métodos introducidos en una clase sólo con la firma y sin código se denominan métodos abstractos.

En UML, las clases o métodos abstractos se representan mediante el estereotipo «`abstract`». Gráficamente se representan explícita, o bien implícitamente, poniendo en cursiva el nombre de la clase o del método.

Ejemplo

Los animales pueden dormir o comer, pero lo hacen de manera distinta de acuerdo con la naturaleza del animal. Estos métodos poseen su firma como única descripción a nivel de la clase `Animal`*. Se trata de métodos abstractos.*

La siguiente figura muestra estos métodos abstractos dentro de la clase abstracta `Animal` *y presenta la redefinición para hacerlos concretos (es decir, la atribución de código) en las clases* `Caballo` *y* `Lobo`*. En efecto, los caballos duermen de pie mientras que los lobos duermen tumbados. Tampoco comen de la misma manera: los lobos son carnívoros mientras que los caballos son herbívoros.*

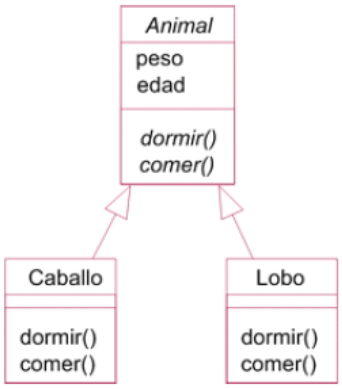


Figure 6.43 - Clases y métodos abstractos

NOTA: La firma es un conjunto formado por el nombre del método, los parámetros con el nombre y el tipo, y el tipo del resultado, a excepción del código del método.

Cualquier clase que posea al menos un método abstracto se considera una clase abstracta. La presencia de un solo método incompleto (el código no está) implica que la clase no es una descripción completa de objetos.

Una clase puede ser abstracta incluso aunque no contenga ningún método abstracto.

4. La expresión de especificaciones sobre la relación de herencia

UML ofrece cuatro especificaciones sobre la relación de herencia entre una superclase y sus subclases:

- La especificación `{incomplete}` significa que el conjunto de subclases está incompleto y que no cubre la superclase o incluso que el conjunto de instancias de las subclases es un subconjunto del conjunto de instancias de la superclase;
- Por el contrario, la especificación `{complete}` significa que el conjunto de subclases está completo y cubre la superclase;
- La especificación `{disjoint}` significa que las subclases no tienen ninguna instancia en común;
- La especificación `{overlapping}` significa que las subclases pueden tener una o varias instancias en común.

Ejemplo

La siguiente figura ilustra una relación de herencia entre la superclase `Équido` y dos subclases: `Caballo` y `Burro`. Estas dos subclases no cubren la clase de los équidos (existen otras subclases, como las cebras). Además, también están las mulas, que derivan de un cruce y son a la vez caballos y burros. De ahí el uso de las especificaciones `{incomplete}` y `{overlapping}`.

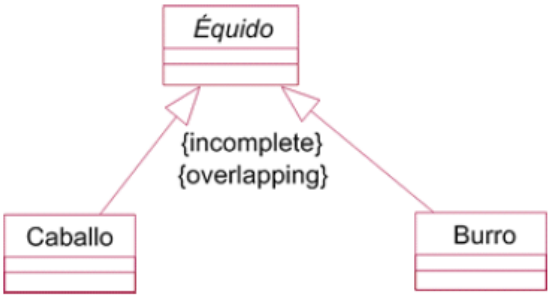


Figura 6.44 - Subclases sin cobertura y con instancias comunes

Ejemplo

La siguiente figura muestra otra relación de herencia entre la superclase `Caballo` y dos subclases: `CaballoMacho` y `CaballoHembra`. Estas dos subclases cubren la clase de los caballos. No existe ningún caballo que sea a la vez macho y hembra. De ahí el uso de las especificaciones `{complete}` y `{disjoint}`.

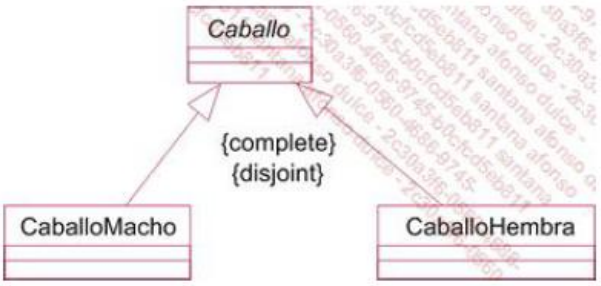


Figura 6.45 - Subclases sin cobertura y sin instancia común

5. La herencia múltiple

Hablamos de herencia múltiple en UML cuando una subclase hereda de varias superclases. La herencia múltiple plantea un solo problema: cuando un mismo método, es decir, un método dotado de la misma firma, se hereda varias veces en la subclase se crea un conflicto. Al recibir un mensaje de llamada al método, es preciso definir un criterio para elegir un método entre todos los heredados.

En tales casos, una solución consiste en redefinir el método en la subclase para suprimir el conflicto.

El mismo problema puede plantearse para un atributo con el mismo nombre introducido en varias superclases. Una posible solución consiste en utilizar la propiedad `{redefines nombreAtributo}` para redefinirlo en la subclase.

NOTA: Si bien la utilización de la herencia múltiple es posible en modelado, a menudo se revela necesario transformar los diagramas de clases para eliminarla al pasar al desarrollo. Pocos son los lenguajes de programación que soportan esta forma de herencia. Por ello, existen diferentes técnicas, entre las cuales se encuentra la transformación de la herencia múltiple en una agregación.

Ejemplo

La siguiente figura retoma las dos subclases `Caballo` y `Burro` e introduce una subclase común, a saber `Mula`. También muestra el método `trotar`, abstracto en la clase `Équido` y concreto en las subclases inmediatas y redefinido en la subclase fruto de su herencia múltiple. Cuando el jinete pide a la mula que trote, ésta responde con reacciones de caballo y de burro a la vez. Puede ser peligrosa como un caballo e irritante como un burro.

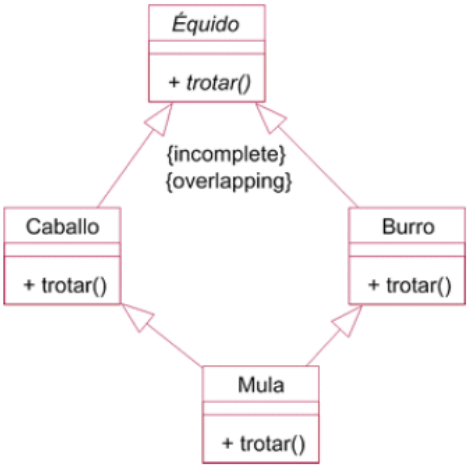


Figura 6.46 - Herencia múltiple con conflicto

6. La factorización de las relaciones entre objetos

Una clase abstracta sirve para factorizar los atributos y métodos de varias subclases. A veces también, es posible factorizar el extremo de una asociación en una superclase con el fin de hacer más simple el diagrama.

Ejemplo

Al igual que los lobos, los caballos poseen dos ojos y una nariz.
La figura 6.48 muestra que, una vez creada, la clase abstracta `Mamífero` *es una superclase de las clases* `Caballo` *y* `Lobo`*, las dos asociaciones de composición se factorizan.*

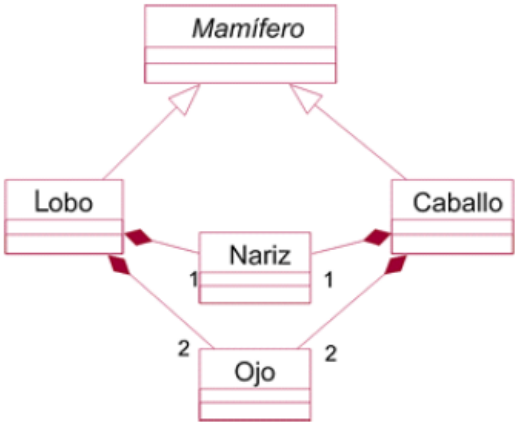


Figura 6.47 - Asociaciones entre objetos no factorizados

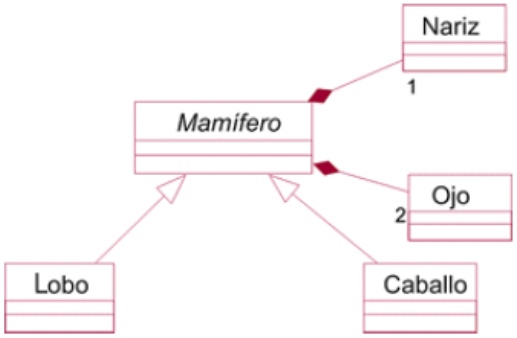


Figura 6.48 - Asociaciones entre objetos factorizados

7. La interfaz

Una interfaz es una clase totalmente abstracta, es decir, no tiene atributo y todos sus métodos son abstractos y públicos. Estas clases no contienen ningún elemento de implantación de los métodos. Gráficamente se representan como una clase con el estereotipo «`interface`».

La implantación de los métodos se realiza mediante una o varias clases concretas, subclases de la interfaz. En ese caso, la relación de herencia existente entre la interfaz y la subclase de implantación se conoce como relación de realización. En las relaciones de herencia entre dos clases, se representa gráficamente mediante una línea discontinua, en lugar de mediante una línea completa.

La siguiente figura muestra esta representación.

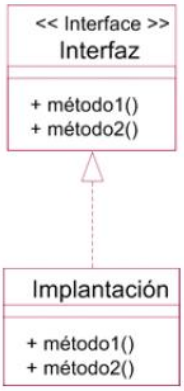


Figura 6.49 - Interfaz y relación de realización

Ejemplo

Un caballo de carreras puede considerarse como una interfaz. La interfaz se compone de varios métodos: `correr`, `detenerse`, *etc.*

A continuación la implantación puede diferir. Las carreras al galope o al trote se realizan sólo con caballos entrenados específicamente para un tipo u otro de carrera. Para los caballos de galope, correr significa galopar. Para los caballos de trote (trotadores), correr significa trotar. Ambos responden a la interfaz `CaballoCarrera`, pero con una implantación diferente resultado de su entrenamiento.

La siguiente figura muestra el ejemplo.

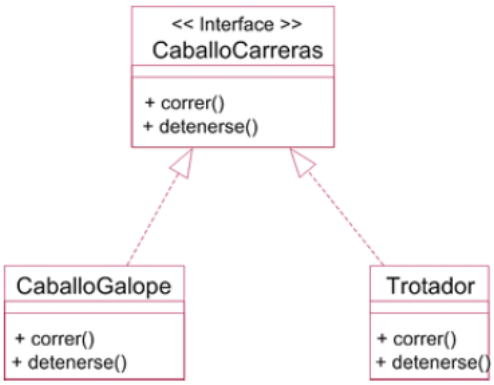


Figura 6.50 - Interfaz y subclases distintas de realización

Se puede también representar con un lollipop o piruleta.

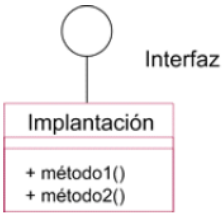


Figura 6.51 - Representación lollipop de una interfaz y de la relación de realización

`NOTA`: Una misma clase puede realizar varias interfaces. Se trata de un caso particular de herencia múltiple. No puede darse ningún conflicto, ya que en la clase de realización sólo se heredan las firmas de los métodos. Si hay varias interfaces con la misma firma, ésta se implanta en la clase común de realización mediante un solo método.

Las clases también pueden depender de una interfaz para realizar sus operaciones. La interfaz se emplea entonces como tipo dentro de la clase (atributo, parámetro o variable local de uno de los métodos).

Decimos que una clase que depende de una interfaz es cliente de ella.

La siguiente figura muestra la relación de dependencia entre una clase y una interfaz.

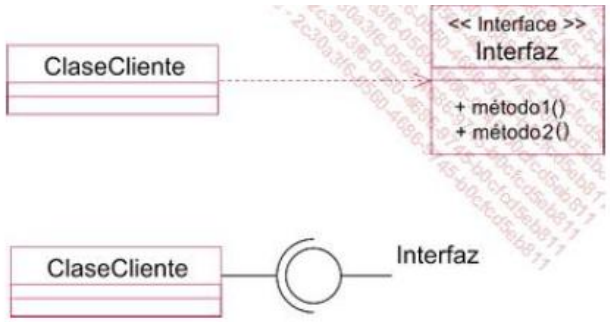


Figura 6.52 - Relación de dependencia entre una clase y una interfaz

Ejemplo

Para poder organizar una carrera se necesitan caballos. La clase `Carrera` *depende por tanto de la interfaz* `CaballoCarrera`*).*

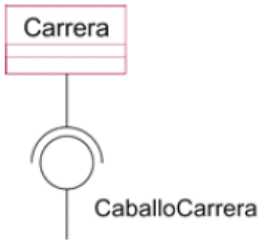


Figura 6.53 - Ejemplo de dependencia entre una clase y una interfaz

Los diferentes estereotipos de clase

Un estereotipo sirve para especializar un elemento UML, como lo hacen los estereotipos de clase «`abstract`» e «`interface`». A continuación describimos los principales estereotipos de clase.

«`abstract`»: este estereotipo indica que una clase es abstracta.

«`auxiliary`»: se trata de una clase secundaria de un sistema. Las clases secundarias sirven de soporte a las clases principales.

«`focus`»: se trata de una clase principal de un sistema. Este estereotipo se opone al estereotipo «`auxiliary`».

«`implementationclass`»: una clase de implementación proporciona la realización de una clase «`type`».

«`type`»: una clase «`type`» se corresponde con un tipo de datos abstracto que introduce atributos y métodos abstractos, pero sin proporcionar ninguna realización. Dicha clase es abstracta y se distingue de una interfaz por la introducción de atributos.

«`interface`»: este estereotipo precisa que una clase sea una interfaz.

«`utility`»: este estereotipo introduce la noción de clase utilitaria que solamente posee atributos de clase dotados de la propiedad `{readOnly}`, es decir, constantes así como métodos de clase. Dicha clase no puede ser instanciada.

«enumeration»: una enumeración es una clase particular que introduce un tipo de datos cuyos valores se precisan mediante una lista fija de constantes. Estos valores se denominan literales. Una enumeración se representa gráficamente por esta lista de literales.

Ejemplo

La siguiente figura ilustra una enumeración en la que la lista de literales se corresponde con los distintos estados de un caballo durante una carrera de obstáculos.

Figura 6.54 - Ejemplo de enumeración

Las clases template

Una clase concreta es un modelo que puede instanciarse para crear objetos. De forma análoga, una clase template es un modelo de clases. Posee parámetros que le confieren un aspecto genérico. Estos parámetros están tipados de forma similar a los parámetros de un método, utilizando todos los tipos disponibles. El principal tipo que encontramos es `Type`, un tipo estándar de UML, como `Integer`o `String`. Sirve para fijar el tipo de los atributos, de los parámetros de método, etc.

NOTA: `Type` es una clase del metamodelo de UML. El metamodelo es un modelo UML donde los elementos descritos son elementos de UML. Contiene la descripción de las clases, de los tipos, de las asociaciones, de los atributos, de los métodos, etc. De este modo, la clase `Type` describe todos los tipos que se encuentran en un modelo UML, incluidos en el metamodelo. Las instancias de `Type` son tanto los tipos estándar de UML como las clases de cualquier modelo o metamodelo. Por ejemplo, `Integer`, `String`, la clase `Caballo` son instancias de `Type`. La clase del metamodelo `Class` que describe las clases y la propia clase `Type` son también instancias de `Type`. Precisamos también que no es necesario conocer el metamodelo de UML para utilizar las clases template.

El valor de estos parámetros está fijado por una relación de enlace (binding) del template que describe el enlace entre una clase instanciada y su clase template. Esta relación de enlace es el equivalente, para las clases template, de la relación de instanciación que vincula un objeto con su clase.

La representación clásica de una clase template es comparable a la de una clase. Los parámetros se introducen en un rectángulo situado en la esquina superior derecha de la representación de la clase respetando la siguiente sintaxis:

```
nombreParámetro : type=valorDefecto
```

Donde `nombreParámetro`es el nombre del parámetro, `type`su tipo, y `valorDefecto` el valor por defecto del parámetro. El tipo y el valor por defecto son opcionales.

Ejemplo

La figura siguiente ilustra la representación de la clase template `ColecciónGenérica`. Posee un parámetro `T` que tiene como tipo la clase `Type`. Implementa una colección cuyos elementos se almacenan en un array introducido por el atributo `contenido`. El método `agrega` inserta un elemento, el método `retira` extrae un elemento y el método `getContenido` devuelve su contenido en forma de array.

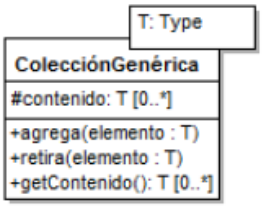


Figura 6.55 - Ejemplo de clase template

La relación de enlace está basada en la palabra clave `bind`, que permite detallar cómo se fija el valor de cada parámetro. La sintaxis del enlace es la siguiente:

```
<<bind>><nombreParámetro -> valor, ... >
```

Ejemplo

La siguiente figura ilustra la relación de enlace entre la clase `MiColección` que describe mi colección de caballos y la clase template `ColecciónGenérica`. El valor de `T` se fija igual a `Caballo`.

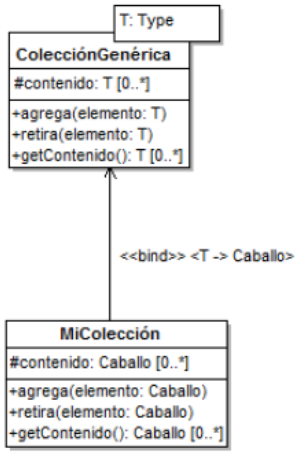


Figura 6.56 - Ejemplo de relación de enlace de una clase template

La siguiente figura es otra representación donde la clase `MiColección` no se ha detallado, lo cual no es obligatorio.

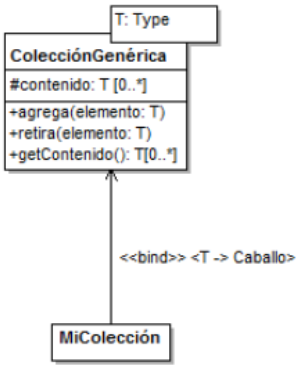


Figura 6.57 - Otra representación de la relación de enlace

Los objetos o instancias

1. La representación de los objetos

El diagrama de clases es una representación estática del sistema. También puede mostrar los objetos, es decir, en un momento determinado, las instancias creadas y sus vínculos cuando el sistema está activo.

Las instancias se representan dentro de un rectángulo con su nombre subrayado y, eventualmente, el valor de uno o varios atributos.

El nombre de una instancia presenta la forma siguiente:

```
nombreInstancia : nombreClase
```

El nombre de la instancia es opcional.

El valor del atributo presenta esta forma:

```
nombreAtributo = valorAtributo
```

Ejemplo

La siguiente figura muestra la instancia `Jorgelina`*, instancia de la clase* `Caballo`*.*

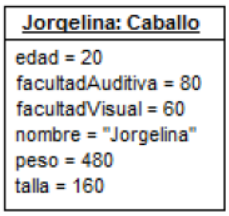


Figura 6.58 - Ejemplo de objeto: la llegua Jorgelina

2. La relación de instanciación

La relación de instanciación describe el vínculo que existe entre una instancia y su clase. Esta relación se describe en el nombre de la instancia cuyo sufijo es el nombre de la clase. También es posible precisar esta relación entre la instancia y su clase utilizando una relación de dependencia dotada del estereotipo «`instanceOf`». Esta última representación es, sin embargo, menos habitual.

Ejemplo

La siguiente figura muestra la instancia `Jorgelina` *y su clase* `Caballo`*. La relación de instanciación se indica de dos maneras.*

En primer lugar, el nombre Jorgelina tiene como sufijo el nombre Caballo. A continuación, la figura incluye una relación de dependencia entre `Jorgelina`*y la clase* `Caballo`*. Esta relación está dotada del estereotipo «*`instanceOf`*».*

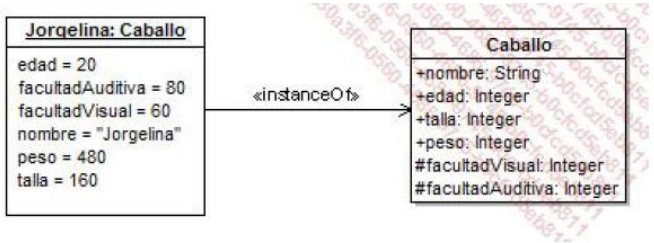


Figura 6.59 - Representación de la relación de instanciación mediante la relación de dependencia

3. Los vínculos entre objetos

Los vínculos entre instancias se representan mediante simples líneas continuas. Estos vínculos son los elementos de las relaciones interobjetos. Por este motivo, de manera similar a como se hace con los nombres de los objetos, el nombre de los vínculos se representa gráficamente con un estilo subrayado y tiene como sufijo el nombre de la asociación. Es posible, también, representar el nombre de los roles sobre la representación gráfica de los vínculos.

Estos nombres de rol no están subrayados, puesto que no se corresponden ni con instancias ni con ocurrencias.

Ejemplo

La siguiente figura ilustra un ejemplo de representación de objetos. Las clases correspondientes están representadas encima. Los distintos enlaces que unen los productos y los clientes muestran representaciones diferentes. El enlace entre `Heno` *y* `Fien` *muestra únicamente los roles. El enlace entre* `Paja` *y* `Fien` *muestra únicamente el nombre del enlace, constituido por dos puntos seguidos del nombre de la asociación. El enlace entre* `Paja` *y* `Lorenzo` *muestra el rol* `productoComprado` *y el nombre del enlace tiene como sufijo el nombre de la asociación. Por último, el enlace entre* `Agua` *y* `Lorenzo` *no muestra ninguna de sus características.*

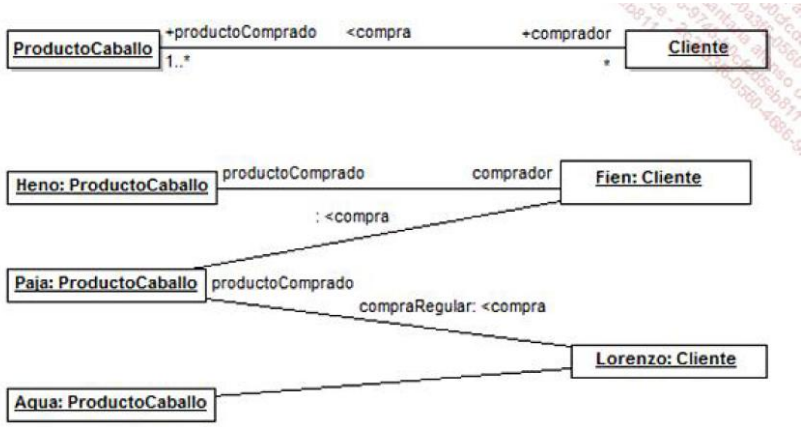


Figura 6. 60 - Ejemplos de representación de objetos

Diagrama de estructura compuesta

1. La descripción de un objeto compuesto

La finalidad principal del diagrama de estructura compuesta es describir con precisión objetos compuestos. Estos diagramas no sustituyen a los diagramas de clases, sino que los completan.
En los diagramas de estructura compuesta, el objeto compuesto se describe mediante un clasificador, mientras que sus componentes se describen mediante las partes. Un clasificador y una parte están asociados a una clase, cuya descripción completa se realiza en un diagrama de clases.

Observemos el objeto compuesto descrito en el diagrama de clases de la siguiente figura. Posee un componente derivado de una composición fuerte y otro derivado de una agregación.

Figura 6.61 - Objeto compuesto

La figura 6.62 muestra el diagrama de estructura compuesta correspondiente a ese objeto. Los componentes están integrados dentro del clasificador que describe el objeto compuesto. El tipo de las partes es la clase del componente. La cardinalidad se indica entre corchetes. Su valor por defecto es uno. Los componentes derivados de agregaciones aparecen representados mediante una línea de puntos, los componentes derivados de composiciones fuertes aparecen representados mediante una línea continua.

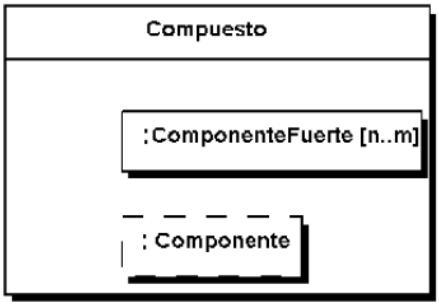


Figura 6.62 - Diagrama de estructura compuesta

Ejemplo

La siguiente figura muestra un ejemplo de diagrama de estructura compuesta en el que se describe un automóvil en tanto que objeto compuesto. El diagrama de clases correspondiente aparece representado abajo.

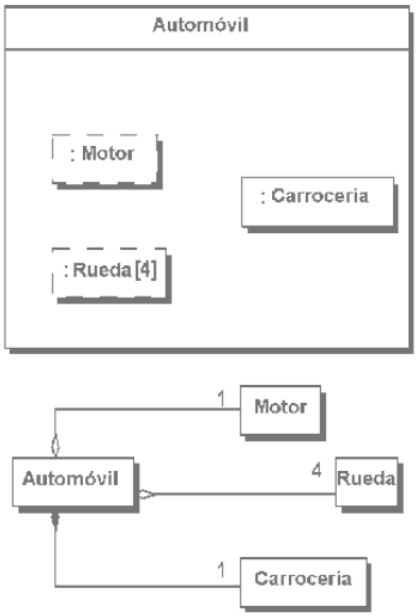
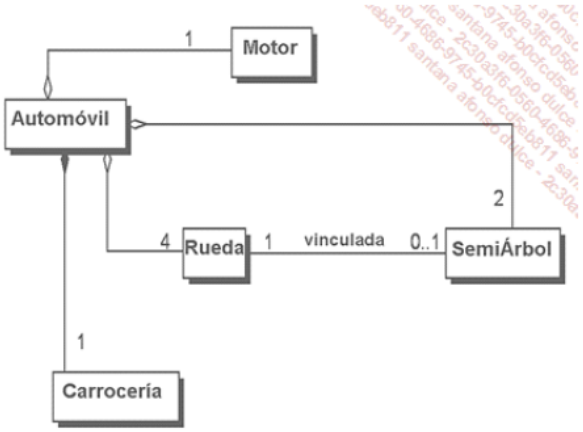


Figura 6.63 - Ejemplo de diagrama de estructura compuesta

NOTA: En el diagrama de estructura compuesta, `Motor`, `Carrocería` y `Rueda` no son clases, sino partes. Las partes se consideran siempre dentro de clasificadores.

El diagrama de clases de la siguiente figura muestra nuevamente un automóvil en tanto que objeto compuesto. Introduce la asociación vinculada entre las ruedas y los semiárboles que se ocupan de la transmisión entre el motor y las ruedas delanteras, que son las ruedas motrices.



La cardinalidad de la asociación vinculada es 0..1 en el extremo del semiárbol. En efecto, la cardinalidad de las ruedas delanteras es uno y la de las traseras es cero. Esta última información no puede ser descrita en el diagrama de clases, a menos que se introduzcan dos subclases de `Rueda`: `RuedaDelantera` y `RuedaTrasera`. No obstante, el inconveniente de incorporar dos subclases para precisar una cardinalidad es que el diagrama de clases se hace más enrevesado. La opción no es, por tanto, demasiado deseable.

El diagrama de estructura compuesta permite especificar la función de una parte. La función describe el uso de la parte dentro del objeto compuesto. La siguiente figura introduce tres partes para las ruedas correspondientes a la rueda delantera izquierda, la rueda delantera derecha y las dos ruedas traseras, respectivamente. La cardinalidad de las partes se adapta en consecuencia. El nombre de la función se indica en la parte antes del tipo.

NOTA: Esta notación no es la misma que la usada en el diagrama de objetos. Efectivamente, la notación para especificar una instancia utiliza el estilo subrayado.

En la siguiente figura se han introducido también dos partes correspondientes a cada semiárbol.

Entre cada parte correspondiente a una rueda delantera y cada parte correspondiente a un semiárbol, un conector representa la asociación `vinculada`. Los conectores sirven para unir dos partes. Están tipificados por una asociación interobjetos, del mismo modo que una parte está tipificada por una clase.

NOTA: Si existen varios conectores entre dos partes y éstos están tipificados por la misma asociación, es posible distinguirlos atribuyéndoles nombres de función al modo de los nombres de función de las partes.

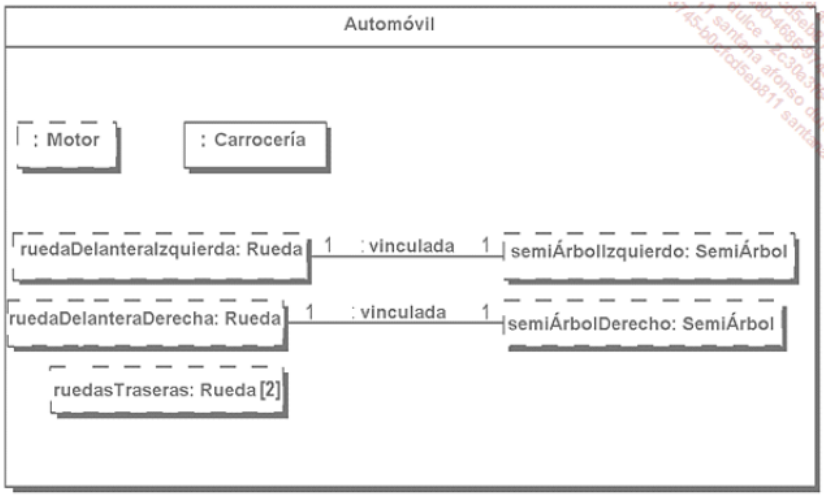


Figura 6.65 - Diagrama de estructura compuesta con funciones y conectores

Los conectores pueden también vincular las partes entre ellas a través de puertos. Un puerto es un punto de interacción. Posee una interfaz que constituye su tipo y define el conjunto de interacciones posibles. Las interacciones definidas por un puerto se hacen con los otros puertos vinculados a él mediante un conector.

Los puertos también pueden introducirse en los clasificadores. En ese caso, el objetivo de los puertos es servir de pasarela entre las partes internas del clasificador y los objetos externos a éste (su entorno).

Desde el punto de vista de la encapsulación, se trata de puertos generalmente públicos. Y se conocen, por tanto, fuera del clasificador.

NOTA: Un puerto definido en el clasificador puede también definirse como privado. Se reserva entonces a una comunicación interna entre el clasificador y sus partes. No realiza funciones de pasarela entre el interior y el exterior del clasificador.

Una parte puede poseer varios puertos, cada uno de ellos dotado de su propia interfaz. Varios puertos de una misma parte pueden tipificarse con la misma interfaz. En esos casos es posible distinguirlos asignándoles nombres defunción diferentes, a la manera de lo que se hace con las partes y los conectores.

La siguiente figura muestra la misma descomposición en partes del objeto `Automóvil` que en el último ejemplo. Entre el motor y los semiárboles de transmisión se han agregado algunos conectores. Los conectores entre las partes están unidos a través de un puerto representado en forma de cuadrado blanco. También se ha añadido un puerto en el clasificador, que está tipificado por la interfaz `Orden` y conectado a un puerto del motor igualmente tipificado por esa interfaz.

En lo que se refiere a las interacciones, la figura muestra que:

- La clase `Automóvil` puede interactuar con el exterior para recibir órdenes destinadas al motor y que le son transmitidas;
- El motor se comunica con los semiárboles (transmisión de movimiento);
- Cada semiárbol se comunica con las ruedas (transmisión de movimiento).

NOTA: Los conectores no están tipificados por razones de simplificación. De igual modo, la interfaz de los puertos de los semiárboles, de las ruedas y del puerto del motor conectado a los semiárboles no se ha indicado. Podría tratarse de la interfaz *`Transmisión`*.

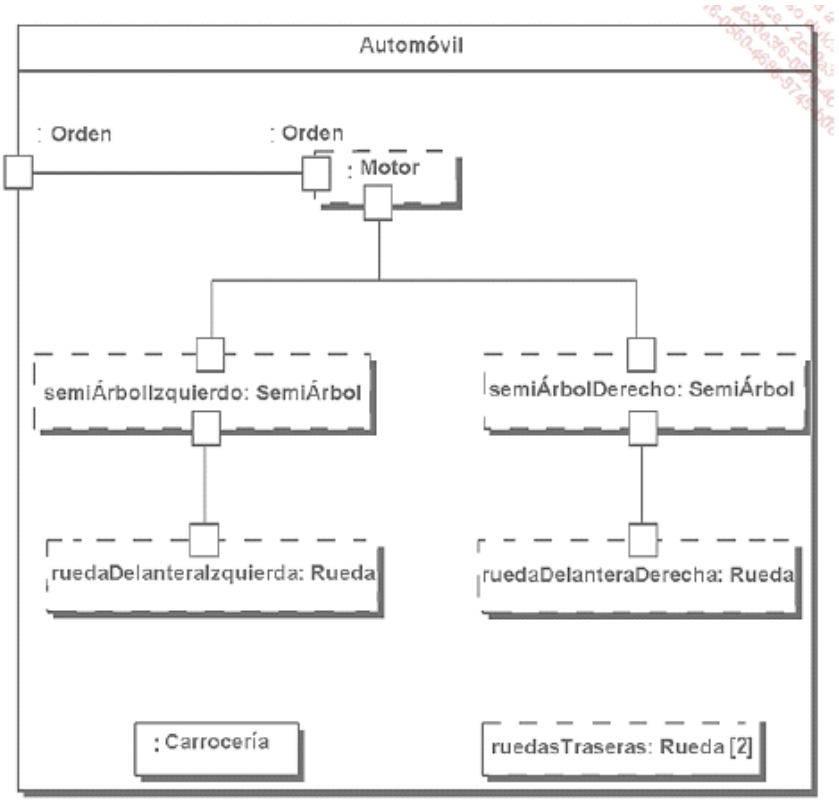


Figura 6.66 - Diagrama de estructura compuesta en el que se han introducido una serie de puertos

2. La colaboración

Una colaboración describe un conjunto de objetos que interactúan entre sí con el fin de ilustrar la funcionalidad de un sistema. Cada objeto participa en esa funcionalidad efectuando un papel concreto.

En las colaboraciones, los objetos se describen como en un clasificador, con los mismos elementos: partes, conectores, puertos, etc.

Las colaboraciones pueden usarse para describir los patrones de diseño (*design patterns*) utilizados en la programación con objetos.

La siguiente figura muestra el diagrama de clases de uno de esos patrones; a saber, el patrón `Composite`.

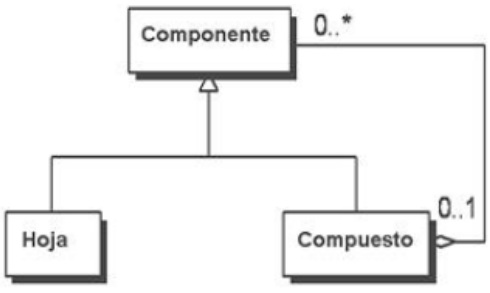


Figura 6.67 - Pattern `Composite`

La finalidad del patrón `Composite` es ofrecer un marco de diseño de una composición de objetos cuya profundidad puede variar. Un componente puede ser una hoja o un objeto compuesto, a su vez, por hojas y otros compuestos.

Uno de los ejemplos que mejor ilustran este tipo de patrón es el sistema de archivos del disco duro de un ordenador personal, compuesto de archivos y carpetas. Cada carpeta puede, a su vez, contener archivos u otras carpetas.
En la siguiente figura se muestra una colaboración en la que se describe el patrón `Composite`. En ella están representadas las dos partes de la colaboración, a saber, la de la clase `Hoja` y la de la clase `Compuesto`. Toda la hoja está necesariamente contenida en un solo y único compuesto, es decir, que existe al menos un compuesto. Los compuestos pueden estar contenidos en otros compuestos o no (caso del compuesto raíz).

NOTA: Las colaboraciones no sustituyen el diagrama de clases, sino que lo completan. Como mencionamos al principio: el diagrama de estructura compuesta no está llamado a reemplazar el diagrama de clases, sino a completarlo.

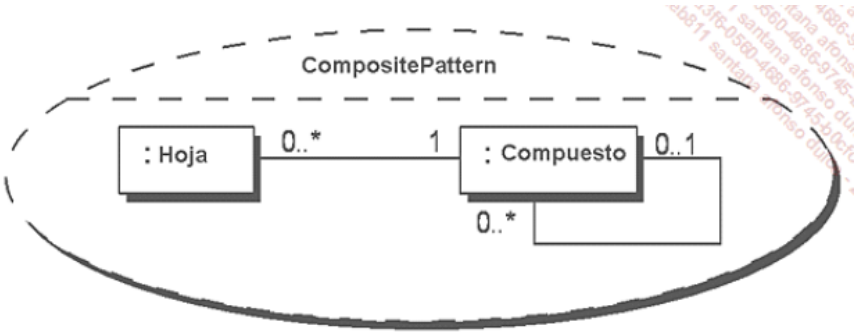


Figura 6.68 - Colaboración en la que se describe el patrón Composite

El diagrama de estructura compuesta ofrece la posibilidad de describir una aplicación de una colaboración fijando las funciones de las partes. Es lo que hemos hecho en la figura 6.69 tomando nuestro ejemplo de archivos como aplicación de la colaboración del patrón `Composite`.

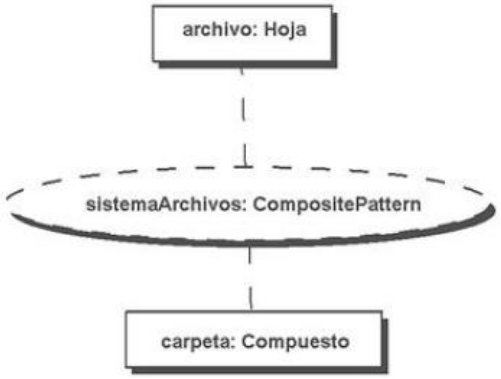


Figura 6.69 - Aplicación de la colaboración Composite al sistema de archivos

Conclusión

En el presente capítulo hemos estudiado los diagramas de clases. Las clases describen los atributos y los métodos de sus instancias, así como los atributos y métodos de clase.
Las asociaciones entre objetos son obligatorias al elaborar un diagrama de clases. Dichas asociaciones constituyen la base de la interacción de las instancias a través del envío de mensajes cuando el sistema está activo.

Las relaciones de herencia entre clases son, asimismo, indispensables, ya que favorecen la factorización de elementos comunes, permitiendo así reducir el tamaño del diagrama.

La expresión de las especificaciones en UML, en lenguaje natural o en OCL conduce a enriquecer aún más la semántica expresada en el diagrama.

Por último, hemos estudiado el diagrama de estructura compuesta que complementa el diagrama de clases ofreciendo una descripción más fina de los objetos compuestos.