

Introducción

Todo gira en torno de una visión. Un sistema complejo toma forma cuando alguien tiene la visión de cómo la tecnología puede mejorar las cosas. Los desarrolladores tienen que entender completamente la idea y mantenerla en mente mientras crean el sistema que le dé forma.

El éxito de los proyectos de desarrollo de aplicaciones o sistemas se debe a que sirven como enlace entre quien tiene la idea y el desarrollador. El UML (*Lenguaje Unificado de Modelado*) es una herramienta que cumple con esta función, ya que le ayuda a capturar la idea de un sistema para comunicarla posteriormente a quien esté involucrado en su proceso de desarrollo; esto se lleva a cabo mediante un conjunto de símbolos y diagramas. Cada diagrama tiene fines distintos dentro del proceso de desarrollo.

El objetivo de este libro es darle, en 24 horas de estudio, las bases sobre el UML. Cada hora le presentará ejemplos para mejorar la comprensión e incluirá ejercicios que le permitirán practicar sus recién adquiridos conocimientos.

Dividí este libro en tres partes: la primera parte le da un panorama del UML y le explica la orientación a objetos, misma que conforma los conceptos fundamentales de la diagramación de objetos y clases. Examinaremos los casos de uso (una estructura para mostrar la forma en que un sistema lucirá ante el usuario, para luego mostrar cómo hacer diagramas de esta estructura). Las horas restantes de la primera parte le permitirán trabajar con el resto de los diagramas UML.

La segunda parte le muestra una metodología simplificada para el desarrollo, enriquecida con el estudio de un caso ficticio. Así, las horas de la segunda parte le mostrarán la forma en que el UML se adapta al contexto de un proyecto de desarrollo. Verá la forma en que los elementos del UML funcionan en conjunto para modelar un sistema.

Por último, en la tercera parte aplicaremos el UML para diseñar patrones y sistemas incrustados, y examinaremos su campo de aplicación en dos áreas más.

Existen diversos fabricantes que cuentan con paquetes que le permitirán generar diagramas UML y coordinarlos en un modelo. Los más notables son Rational Rose y SELECT Enterprise, aunque cabe mencionar que Visual UML es otro digno contendiente. Microsoft está autorizado para utilizar la tecnología de Rational y así comercializa Visual Modeler, un subconjunto de Rational Rose. No obstante, en este libro todo lo que necesitará será un lápiz y papel para dibujar los diagramas y una sana curiosidad respecto al estado actual del diseño de sistemas.

¡Iniciemos!

Convenciones utilizadas en este libro

En los diagramas incluidos en este libro no utilizamos vocales con acento, ni la letra ñe. Esto debido a que el alfabeto inglés, de donde procede la mayoría de los lenguajes de programación, no los incluye y el empleo de esos caracteres en sus diagramas podría acarrearle problemas tanto en el UML como en el lenguaje de programación que piense



Una Nota presenta interesantes secciones de información relacionadas con el tema que se trate.

TERMINO NUEVO

El icono Término Nuevo resalta las definiciones de vocablos nuevos y esenciales. El vocablo, en sí, aparecerá en *cursiva*.



PARTE I

Para iniciar

Hora

- 1 Introducción al UML
- 2 Orientación a objetos
- 3 Uso de la orientación a objetos
- 4 Uso de relaciones
- 5 Agregación, composición, interfaces y realización
- 6 Introducción a los casos de uso
- 7 Diagramas de casos de uso
- 8 Diagramas de estados
- 9 Diagramas de secuencias
- 10 Diagramas de colaboraciones
- 11 Diagramas de actividades
- 12 Diagramas de componentes
- 13 Diagramas de distribución
- 14 Nociones de los fundamentos del UML
- 15 Adaptación del UML en un proceso de desarrollo



Edited with the demo version of
Infix Pro PDF Editor

To remove this notice, visit:
www.iceni.com/unlock.htm



HORA 1

Introducción al UML

El UML (Lenguaje Unificado de Modelado) es una de las herramientas más emocionantes en el mundo actual del desarrollo de sistemas. Esto se debe a que permite a los creadores de sistemas generar diseños que capturen sus ideas en una forma convencional y fácil de comprender para comunicarlas a otras personas.

En esta hora se tratarán los siguientes temas:

- Por qué es necesario el UML
- La concepción del UML
- Diagramas del UML
- Para qué tantos diagramas

TÉRMINO NUEVO

En el contexto de este libro considere a un *sistema* como una combinación de software y hardware que da una solución a un problema de negocios. El *desarrollo de sistemas* es la creación de un programa para un *cliente*, este último es quien tiene el problema que debe ser resuelto. Un *analista* es el que documenta el problema del cliente y lo comunica a los *desarrolladores*, que son los programadores que generarán el programa que resolverá el problema y lo distribuirán en equipos de computación.

La comunicación de la idea es de suma importancia. Antes del advenimiento del UML, el desarrollo de sistemas era, con frecuencia, una propuesta al azar. Los analistas de sistemas intentaban evaluar los requerimientos de sus clientes, generar un análisis de requerimientos en algún tipo de notación que ellos mismos comprendieran (aunque el cliente no lo comprendiera), dar tal análisis a uno o varios programadores y esperar que el producto final cumpliera con lo que el cliente deseaba.

Dado que el desarrollo de sistemas es una actividad humana, hay muchas posibilidades de cometer errores en cualquier etapa del proceso, por ejemplo, el analista pudo haber malentendido al cliente, es decir, probablemente produjo un documento que el cliente no pudo comprender. Tal vez ese documento tampoco fue comprendido por los programadores quienes, por ende, pudieron generar un programa difícil de utilizar y no generar una solución al problema original del cliente.

¿Alguien se preguntará por qué muchos de los sistemas en uso son ineficientes, engorrosos y difíciles de utilizar?

Por qué es necesario el UML

En los principios de la computación, los programadores no realizaban análisis muy profundos sobre el problema por resolver. Si acaso, garabateaban algo en una servilleta. Con frecuencia comenzaban a escribir el programa desde el principio, y el código necesario se escribía conforme se requería. Aunque anteriormente esto agregaba un aura de aventura y atrevimiento al proceso, en la actualidad es inapropiado en los negocios de alto riesgo.

Hoy en día, es necesario contar con un plan bien analizado. Un cliente tiene que comprender qué es lo que hará un equipo de desarrolladores; además tiene que ser capaz de señalar cambios si no se han captado claramente sus necesidades (o si cambia de opinión durante el proceso). A su vez, el desarrollo es un esfuerzo orientado a equipos, por lo que cada uno de sus miembros tiene que saber qué lugar toma su trabajo en la solución final (así como saber cuál es la solución en general).

Conforme aumenta la complejidad del mundo, los sistemas informáticos también deberán crecer en complejidad. En ellos se encuentran diversas piezas de hardware y software que se comunican a grandes distancias mediante una red, misma que está vinculada a bases de datos que, a su vez, contienen enormes cantidades de información. Si desea crear sistemas que lo involucren con este nuevo milenio ¿cómo manejará tanta complejidad?

La clave está en organizar el proceso de diseño de tal forma que los analistas, clientes, desarrolladores y otras personas involucradas en el desarrollo del sistema lo comprendan y convengan con él. El UML proporciona tal organización.

Un arquitecto no podría crear una compleja estructura como lo es un edificio de oficinas sin crear primero un anteproyecto detallado; asimismo usted tampoco podría generar un complejo sistema en un edificio de oficinas sin crear un plan de diseño detallado. La idea

es que así como un arquitecto le muestra un anteproyecto a la persona que lo contrató, usted deberá mostrarle su plan de diseño al cliente. Tal plan de diseño debe ser el resultado de un cuidadoso análisis de las necesidades del cliente.

Otra característica del desarrollo de sistemas contemporáneo es reducir el periodo de desarrollo. Cuando los plazos se encuentran muy cerca uno del otro es absolutamente necesario contar con un diseño sólido.

Hay otro aspecto de la vida moderna que demanda un diseño sólido: las adquisiciones corporativas. Cuando una empresa adquiere a otra, la nueva organización debe tener la posibilidad de modificar aspectos importantes de un proyecto de desarrollo que esté en progreso (la herramienta de desarrollo, el lenguaje de codificación, y otras cosas). Un anteproyecto bien diseñado facilitará la conversión. Si el diseño es sólido, un cambio en la implementación procederá sin problemas.

La necesidad de diseños sólidos ha traído consigo la creación de una notación de diseño que los analistas, desarrolladores y clientes acepten como pauta (tal como la notación en los diagramas esquemáticos sirve como pauta para los trabajadores especializados en electrónica). El UML es esa misma notación.

La concepción del UML

El UML es la creación de Grady Booch, James Rumbaugh e Ivar Jacobson. Estos caballeros, apodados recientemente “Los tres amigos”, trabajaban en empresas distintas durante la década de los años ochenta y principios de los noventa y cada uno diseñó su propia metodología para el análisis y diseño orientado a objetos. Sus metodologías predominaron sobre las de sus competidores. A mediados de los años noventa empezaron a intercambiar ideas entre sí y decidieron desarrollar su trabajo en conjunto.



Las horas 2, “Orientación a objetos”, y 4, “Uso de relaciones”, tratan de la orientación a objetos. Los conceptos de orientación a objetos tienen un papel fundamental en el desarrollo de este libro.

En 1994 Rumbaugh ingresó a Rational Software Corporation, donde ya trabajaba Booch. Jacobson ingresó a Rational un año después; el resto, como dicen, es historia.

Los anteproyectos del UML empezaron a circular en la industria del software y las reacciones resultantes trajeron consigo considerables modificaciones. Conforme diversos corporativos vieron que el UML era útil a sus propósitos, se conformó un consorcio del UML. Entre los miembros se encuentran DEC, Hewlett-Packard, Intellicorp, Microsoft, Oracle, Texas Instruments y Rational. En 1997 el consorcio produjo la versión 1.0 del UML y lo puso a consideración del OMG (Grupo de administración de objetos) como respuesta a su propuesta para un lenguaje de modelado estándar.

El consorcio aumentó y generó la versión 1.1, misma que se puso nuevamente a consideración del OMG. El grupo adoptó esta versión a finales de 1997. El OMG se encargó de la conservación del UML y produjo otras dos revisiones en 1998. El UML ha llegado a ser el estándar de facto en la industria del software, y su evolución continúa.

Diagramas del UML

El UML está compuesto por diversos elementos gráficos que se combinan para conformar diagramas. Debido a que el UML es un lenguaje, cuenta con reglas para combinar tales elementos. En lugar de indicarle a usted cuáles son los elementos y las reglas, veamos directamente los diagramas ya que los utilizará para hacer el análisis del sistema.



Este enfoque es similar a aprender un idioma extranjero mediante el uso del mismo, en lugar de aprender sus reglas gramaticales y la conjugación de sus verbos. Después de un tiempo de hablar otro idioma se le facilitará la conjugación de verbos y la comprensión de las reglas gramaticales.

TERMINO NUEVO

La finalidad de los diagramas es presentar diversas perspectivas de un sistema, a las cuales se les conoce como *modelo*. El modelo UML de un sistema es similar a un modelo a escala de un edificio junto con la interpretación del artista del edificio. Es importante destacar que un modelo UML describe lo que supuestamente hará un sistema, pero no dice cómo implementar dicho sistema.

A continuación se describirán brevemente los diagramas más comunes del UML y los conceptos que representan. Posteriormente, en la parte I verá cada uno de los diagramas con mayor detenimiento. Recuerde que es posible generar híbridos de estos diagramas y que el UML otorga formas de organizarlos y extenderlos.

Diagrama de clases

Piense en las cosas que le rodean (una idea demasiado amplia, pero ¡inténtelo de cualquier forma!). Es probable que muchas de esas cosas tengan atributos (propiedades) y que realicen determinadas acciones. Podríamos imaginar cada una de esas acciones como un conjunto de tareas.

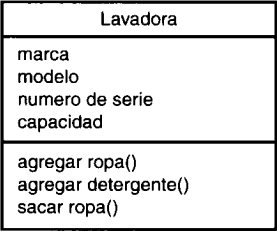
TERMINO NUEVO

También se encontrará con que las cosas naturalmente se albergan en categorías (automóviles, mobiliario, lavadoras...). A tales categorías las llamaremos clases. Una *clase* es una categoría o grupo de cosas que tienen atributos y acciones similares. He aquí un ejemplo: cualquier cosa dentro de la clase Lavadoras tiene atributos como son la marca, el modelo, el número de serie y la capacidad. Entre las acciones

de las cosas de esta clase se encuentran: “agregar ropa”, “agregar detergente”, “activarse” y “sacar ropa”.

La figura 1.1 le muestra un ejemplo de la notación del UML que captura los atributos y acciones de una lavadora. Un rectángulo es el símbolo que representa a la clase, y se divide en tres áreas. El área superior contiene el nombre, el área central contiene los atributos, y el área inferior las acciones. Un diagrama de clases está formado por varios rectángulos de este tipo conectados por líneas que muestran la manera en que las clases se relacionan entre sí.

FIGURA 1.1
*El símbolo UML
de una clase.*



¿Qué objetivo tiene pensar en las clases, así como sus atributos y acciones? Para interactuar con nuestro complejo mundo, la mayoría del software moderno simula algún aspecto del mundo. Décadas de experiencia sugieren que es más sencillo desarrollar aplicaciones que simulen algún aspecto del mundo cuando el software representa clases de cosas reales. Los diagramas de clases facilitan las representaciones a partir de las cuales los desarrolladores podrán trabajar.

A su vez, los diagramas de clases colaboran en lo referente al análisis. Permiten al analista hablarle a los clientes en su propia terminología, lo cual hace posible que los clientes indiquen importantes detalles de los problemas que requieren ser resueltos.

Diagrama de objetos

TERMINO NUEVO

Un objeto es una instancia de clase (una entidad que tiene valores específicos de los atributos y acciones). Su lavadora, por ejemplo, podría tener la marca Laundatorium, el modelo Washmeister, el número de serie GL57774 y una capacidad de 7 Kg.

La figura 1.2 le muestra la forma en que el UML representa a un objeto. Vea que el símbolo es un rectángulo, como en una clase, pero el nombre está subrayado. El nombre de la instancia específica se encuentra a la izquierda de los dos puntos (:), y el nombre de la clase a la derecha.

FIGURA 1.2

El símbolo UML del objeto.

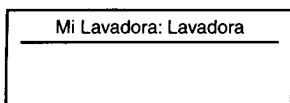


Diagrama de casos de uso

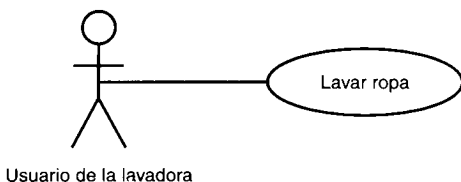
TERMINO NUEVO

Un caso de uso es una descripción de las acciones de un sistema desde el punto de vista del usuario. Para los desarrolladores del sistema, ésta es una herramienta valiosa, ya que es una técnica de aciertos y errores para obtener los requerimientos del sistema desde el punto de vista del usuario. Esto es importante si la finalidad es crear un sistema que pueda ser utilizado por la gente en general (no sólo por expertos en computación).

Posteriormente trataremos este tema con mayor detalle; por ahora, le mostraré un ejemplo sencillo. Usted utiliza una lavadora, obviamente, para lavar su ropa. La figura 1.3 le muestra cómo representaría esto en un diagrama de casos de uso UML.

FIGURA 1.3

Diagrama de casos de uso de uso UML.



TERMINO NUEVO

A la figura correspondiente al Usuario de la lavadora se le conoce como actor. La elipse representa el caso de uso. Vea que el actor (la entidad que inicia el caso de uso) puede ser una persona u otro sistema.

Diagrama de estados

En cualquier momento, un objeto se encuentra en un estado en particular. Una persona puede ser recién nacida, infante, adolescente, joven o adulta. Un elevador se moverá hacia arriba, estará en estado de reposo o se moverá hacia abajo. Una lavadora podrá estar en la fase de remojo, lavado, enjuague, centrifugado o apagada.

El diagrama de estados UML, que aparece en la figura 1.4, captura esta pequeña realidad. La figura muestra las transiciones de la lavadora de un estado al otro.

El símbolo que está en la parte superior de la figura representa el estado inicial y el de la parte inferior el estado final.

FIGURA 1.4

*Diagrama de estados
UML.*

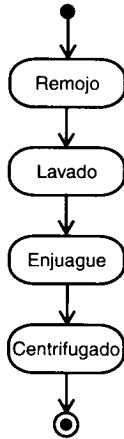


Diagrama de secuencias

Los diagramas de clases y los de objeto representan información estática. No obstante, en un sistema funcional los objetos interactúan entre sí, y tales interacciones suceden con el tiempo. El diagrama de secuencias UML muestra la mecánica de la interacción con base en tiempos.

Continuando con el ejemplo de la lavadora, entre los componentes de la lavadora se encuentran: una manguera de agua (para obtener agua fresca), un tambor (donde se coloca la ropa) y un sistema de drenaje. Por supuesto, estos también son objetos (como verá, un objeto puede estar conformado por otros objetos).

¿Qué sucederá cuando invoque al caso de uso Lavar ropa? Si damos por hecho que completó las operaciones “agregar ropa”, “agregar detergente” y “activar”, la secuencia sería más o menos así:

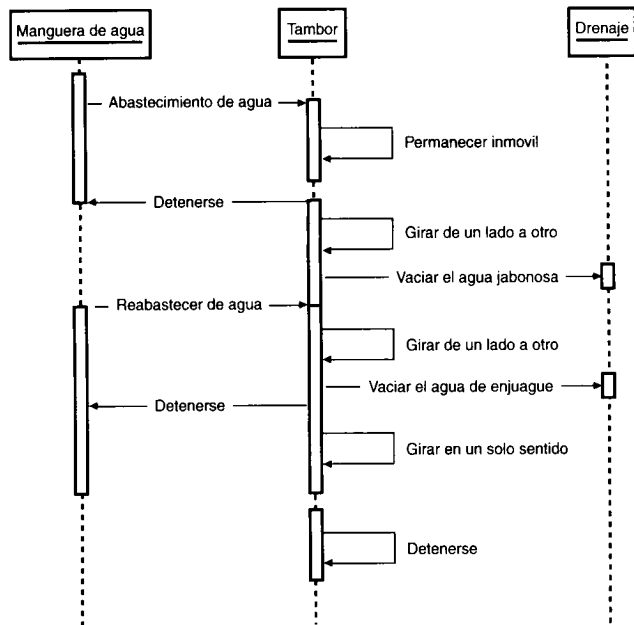
1. El agua empezará a llenar el tambor mediante una manguera.
2. El tambor permanecerá inactivo durante cinco minutos.
3. La manguera dejará de abastecer agua.
4. El tambor girará de un lado a otro durante quince minutos.
5. El agua jabonosa saldrá por el drenaje.
6. Comenzará nuevamente el abastecimiento de agua.
7. El tambor continuará girando.

8. El abastecimiento de agua se detendrá.
9. El agua del enjuague saldrá por el drenaje.
10. El tambor girará en una sola dirección y se incrementará su velocidad por cinco minutos.
11. El tambor dejará de girar y el proceso de lavado habrá finalizado.

La figura 1.5 presenta un diagrama de secuencias que captura las interacciones que se realizan a través del tiempo entre el abastecimiento de agua, el tambor y el drenaje (representados como rectángulos en la parte superior del diagrama). En este diagrama el tiempo se da de arriba hacia abajo.

FIGURA 1.5

Diagrama de secuencias UML.



Por cierto, volviendo a las ideas acerca de los estados, podríamos caracterizar los pasos 1 y 2 como el estado de remojo, 3 y 4 como el estado de lavado, 5 a 7 como el estado de enjuague y del 8 al 10 como el estado de centrifugado.

Diagrama de actividades

Las actividades que ocurren dentro de un caso de uso o dentro del comportamiento de un objeto se dan, normalmente, en secuencia, como en los once pasos de la sección anterior. La figura 1.6 muestra la forma en que el diagrama de actividades UML representa los pasos del 4 al 6 de tal secuencia.

FIGURA 1.6
Diagrama de actividades UML.

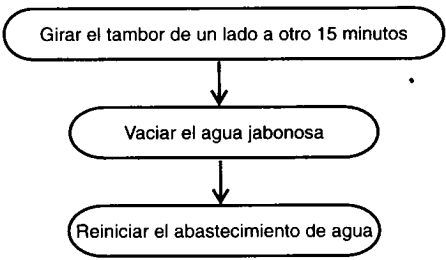


Diagrama de colaboraciones

Los elementos de un sistema trabajan en conjunto para cumplir con los objetivos del sistema, y un lenguaje de modelado deberá contar con una forma de representar esto. El diagrama de colaboraciones UML, diseñado con este fin, se muestra en la figura 1.7. Este ejemplo agrega un cronómetro interno al conjunto de clases que constituyen a una lavadora. Luego de cierto tiempo, el cronómetro detendrá el flujo de agua y el tambor comenzará a girar de un lado a otro.

FIGURA 1.7
Diagrama de colaboraciones UML.

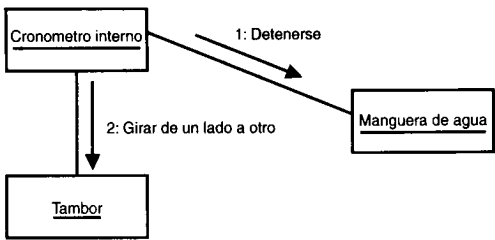


Diagrama de componentes

Este diagrama y el siguiente dejarán el mundo de las lavadoras, dado que están íntimamente ligados con los sistemas informáticos.

El moderno desarrollo de software se realiza mediante componentes, lo que es particularmente importante en los procesos de desarrollo en equipo. Sin extenderme mucho en este punto le mostraré, en la figura 1.8, la manera en que el UML representa un componente de software.

FIGURA 1.8
Diagrama de componentes UML.

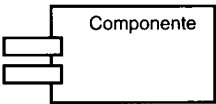
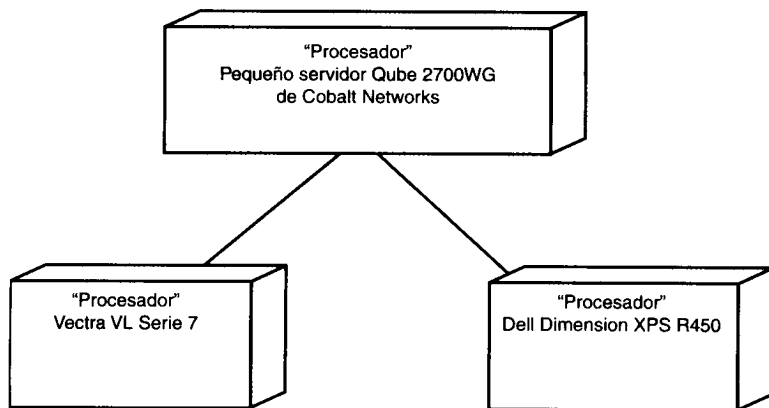


Diagrama de distribución

El diagrama de distribución UML muestra la arquitectura física de un sistema informático. Puede representar los equipos y dispositivos, mostrar sus interconexiones y el software que se encontrará en cada máquina. Cada computadora está representada por un cubo y las interacciones entre las computadoras están representadas por líneas que conectan a los cubos. La figura 1.9 presenta un ejemplo.

FIGURA 1.9

Diagrama de distribución UML.



Otras características

Anteriormente, mencioné que el UML proporciona características que le permiten organizar y extender los diagramas.

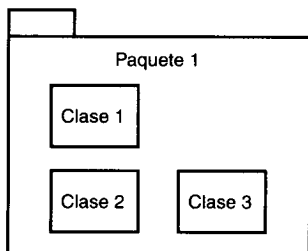
Paquetes

TERMINO NUEVO

En algunas ocasiones se encontrará con la necesidad de organizar los elementos de un diagrama en un grupo. Tal vez quiera mostrar que ciertas clases o componentes son parte de un subsistema en particular. Para ello, los agrupará en un *paquete*, que se representará por una carpeta tabular, como se muestra en la figura 1.10.

FIGURA 1.10

El paquete UML le permite agrupar los elementos de un diagrama.



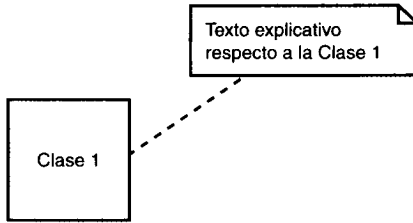
Notas

TÉRMINO NUEVO

Es frecuente que alguna parte del diagrama no presente una clara explicación del porqué está allí o la manera en que trabaja. Cuando éste sea el caso, la *nota* UML será útil. Imagine a una nota como el equivalente gráfico de un papel adhesivo. La nota es un rectángulo con una esquina doblada, y dentro del rectángulo se coloca la explicación. Usted adjunta la nota al elemento del diagrama conectándolos mediante una línea discontinua.

FIGURA 1.11

En cualquier diagrama, podrá agregar comentarios aclaratorios mediante una nota.



Estereotipos

TÉRMINO NUEVO

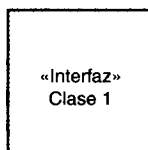
El UML otorga varios elementos de utilidad, pero no es un conjunto minucioso de ellos. De vez en cuando diseñará un sistema que requiera algunos elementos hechos a la medida. Los *estereotipos* o clisés le permiten tomar elementos propios del UML y convertirlos en otros. Es como comprar un traje del mostrador y modificarlo para que se ajuste a sus medidas (contrario a confeccionarse uno completamente nuevo). Imagine a un estereotipo como este tipo de alteración. Lo representará como un nombre entre dos pares de paréntesis angulares y después los aplicará correctamente.

TÉRMINO NUEVO

El concepto de una interfaz provee un buen ejemplo. Una interfaz es una clase que realiza operaciones y que no tiene atributos, es un conjunto de acciones que tal vez quiera utilizar una y otra vez en su modelo. En lugar de inventar un nuevo elemento para representar una interfaz, podrá utilizar el símbolo de una clase con «Interfaz» situada justo sobre el nombre de la clase, como se muestra en la figura 1.12.

FIGURA 1.12

Un estereotipo le permite crear nuevos elementos a partir de otros existentes.



Para qué tantos diagramas

Como puede ver, los diagramas del UML le permiten examinar un sistema desde distintos puntos de vista. Es importante recalcar que en un modelo UML no es necesario que aparezcan todos los diagramas. De hecho, la mayoría de los modelos UML contienen un subconjunto de los diagramas que he indicado.

TERMINO NUEVO

¿Por qué es necesario contar con diferentes perspectivas de un sistema? Por lo general, un sistema cuenta con diversas personas implicadas las cuales tienen enfoques particulares en diversos aspectos del sistema. Volvamos al ejemplo de la lavadora. Si diseñara el motor de una lavadora, tendría una perspectiva del sistema; si escribiera las instrucciones de operación, tendría otra perspectiva. Si diseñara la forma general de la lavadora vería al sistema desde una perspectiva totalmente distinta a si tan sólo tratara de lavar su ropa.

El escrupuloso diseño de un sistema involucra todas las posibles perspectivas, y el diagrama UML le da una forma de incorporar una perspectiva en particular. El objetivo es satisfacer a cada persona implicada.

Resumen

El desarrollo de sistemas es una actividad humana. Sin un sistema de notación fácil de comprender, el proceso de desarrollo tiene una gran cantidad de errores.

El UML es un sistema de notación que se ha convertido en estándar en el mundo del desarrollo de sistemas. Es el resultado del trabajo hecho por Grady Booch, James Rumbaugh e Ivar Jacobson. El UML está constituido por un conjunto de diagramas, y proporciona un estándar que permite al analista de sistemas generar un anteproyecto de varias facetas que sean comprensibles para los clientes, desarrolladores y todos aquellos que estén involucrados en el proceso de desarrollo. Es necesario contar con todos esos diagramas dado que cada uno se dirige a cada tipo de persona implicada en el sistema.

Un modelo UML indica *qué* es lo que supuestamente hará el sistema, mas no *cómo* lo hará.

Preguntas y respuestas

- P** He visto que se refiere al Lenguaje Unificado de Modelado como “UML” y como “el UML”. ¿Cuál es el correcto?
- R** Los creadores del lenguaje prefieren el uso de “el UML”.
- P** Ha indicado que el UML es adecuado para los analistas. No obstante, el diagrama de distribución no parece ser algo muy útil en la fase de análisis en el desarrollo de un sistema. ¿No sería más apropiado para una fase posterior?

- R** En realidad nunca será demasiado pronto para empezar a pensar en la distribución (u otras cuestiones que, tradicionalmente, se dejan para fases posteriores del desarrollo). Aunque es cierto que el analista se interesa por hablar con los clientes y usuarios, en las fases tempranas del proceso el analista debería pensar en los equipos y componentes que constituirían el hardware del sistema. En algunas ocasiones, el cliente dicta esto; en otras, el cliente desea una recomendación del equipo de desarrollo. Ciertamente, un arquitecto de sistemas encontrará útil al diagrama de distribución.
- P** **Ha mencionado que es posible hacer diagramas híbridos. ¿UML, perdón, el UML, impone limitaciones respecto a los elementos que podrá combinar en un diagrama?**
- R** No. El UML no establece límites, no obstante, con frecuencia se da el caso de que un diagrama contenga un tipo de elemento. Podrá colocar símbolos de clases en un diagrama de distribución, pero ello no será muy útil.

Taller

Ya se ha iniciado en el UML. Ahora deberá reafirmar su conocimiento de esta gran herramienta al responder algunas preguntas y realizar los ejercicios. Las respuestas aparecerán en el Apéndice A, “Respuestas a los cuestionarios”.

Cuestionario

1. ¿Porqué es necesario contar con diversos diagramas en el modelo de un sistema?
2. ¿Cuáles diagramas le dan una perspectiva estática de un sistema?
3. ¿Cuáles diagramas le dan una perspectiva dinámica de un sistema (esto es, muestran el cambio progresivo)?

Ejercicios

1. Suponga que creará un sistema informático que jugará ajedrez con un usuario. ¿Cuáles diagramas UML serían útiles para diseñar el sistema? ¿Por qué?
2. Para el sistema del ejercicio que ha completado, liste las preguntas que formularía a un usuario potencial y por qué las haría.

HORA 2

Orientación a objetos

Es fundamental que comprenda todo lo relacionado a la orientación a objetos para el proceso que realizará; específicamente, es importante que conozca algunos conceptos sobre la orientación a objetos.

En esta hora se tratarán los siguientes temas:

- Abstracción
- Herencia
- Polimorfismo
- Encapsulamiento o encapsulación
- Envío de mensajes
- Asociaciones
- Agregación

La orientación a objetos ha tomado por asalto y en forma legítima al mundo del software. Como medio para la generación de programas, tiene varias ventajas. Fomenta una metodología basada en componentes para el desarrollo

de software, de manera que primero se genera un sistema mediante un conjunto de objetos, luego podrá ampliar el sistema agregándole funcionalidad a los componentes que ya había generado o agregándole nuevos componentes, y finalmente podrá volver a utilizar los objetos que generó para el sistema cuando cree uno nuevo, con lo cual reducirá sustancialmente el tiempo de desarrollo de un sistema.

La orientación a objetos es tan importante para el diseño de software que el OMG (Grupo de administración de objetos), una corporación no lucrativa que establece las normas para el desarrollo orientado a objetos, predice que los ingresos obtenidos por el software orientado a objetos serán de 3 millardos de dólares en los siguientes tres a cinco años. El UML influye en esto al permitirle generar modelos de objetos fáciles de usar y comprender para que los desarrolladores puedan convertirlos en software.

La orientación a objetos es un paradigma (un paradigma que depende de ciertos principios fundamentales). En esta hora comprenderá dichos principios y verá qué es lo que hace funcionar a los objetos y cómo utilizarlos en el análisis y diseño. En la siguiente hora, empezará a aplicar el UML a tales principios.

Objetos, objetos por doquier

Los objetos concretos y virtuales, están a nuestro alrededor, ellos conforman nuestro mundo. Como indiqué en la hora anterior, el software actual simula al mundo (o un segmento de él), y los programas, por lo general, imitan a los objetos del mundo. Si comprende algunas cuestiones básicas de los objetos, entenderá cómo se deben mostrar éstos en las representaciones de software.

TERMINO NUEVO

Antes que nada, un objeto es la instancia de una clase (o categoría). Usted y yo, por ejemplo, somos instancias de la clase Persona. Un objeto cuenta con una *estructura*, es decir atributos (propiedades) y acciones. Las acciones son todas las *actividades* que el objeto es capaz de realizar. Los atributos y acciones, en conjunto, se conocen como *características* o *rasgos*.

Como objetos de la clase Persona, usted y yo contamos con los siguientes atributos: altura, peso y edad (puede imaginar muchos más). También realizamos las siguientes tareas: comer, dormir, leer, escribir, hablar, trabajar, etcétera. Si tuviéramos que crear un sistema que manejara información acerca de las personas (como una nómina o un sistema para el departamento de recursos humanos), sería muy probable que incorporáramos algunos de sus atributos y acciones en nuestro software.

En el mundo de la orientación a objetos, una clase tiene otro propósito además de la categorización. En realidad es una plantilla para fabricar objetos. Imagínelo como un molde de galletas que produce muchas galletas (algunos alegarían que esto es lo mismo que la categorización, pero evitemos dicho debate).

Regresemos al ejemplo de la lavadora. Si en la clase Lavadora se indica la marca, el modelo, el número de serie y la capacidad (junto con las acciones de agregar ropa, agregar detergente y sacar ropa), tendrá un mecanismo para fabricar nuevas instancias a partir de su clase; es decir, podrá crear nuevos objetos (vea la figura 2.1).



En la hora 3, "Uso de la orientación a objetos", verá que los nombres de las clases, como *lavadora*, se escribirán como Lavadora, y si constara de dos palabras se escribiría como LavadoraIndustrial, y las características como número de serie se escribirán como numeroSerie.

Esto es particularmente importante en el desarrollo de software orientado a objetos. Aunque este libro no se enfoca a la programación, le ayudará a comprender la orientación a objetos si sabe que las clases en los programas orientados a objetos pueden crear nuevas instancias.

FIGURA 2.1

La clase Lavadora (modelo original) es una plantilla para generar nuevas instancias de Lavadoras.

Lavadora
marca modelo numeroSerie capacidad
agregarRopa() agregarDetergente() sacarRopa()

Es importante que recuerde que el propósito de la orientación a objetos es desarrollar software que refleje particularmente (es decir, que modele) un esquema del mundo. Entre más atributos y acciones tome en cuenta, mayor será la similitud de su modelo con la realidad. En el ejemplo de la lavadora, tendrá un modelo más exacto si incluye los siguientes atributos: volumen del tambor, cronómetro interno, trampa, motor y velocidad del motor. Podría hacerlo más preciso si incluye las acciones de agregar blanqueador, cronometrar el remojo, cronometrar el lavado, cronometrar el enjuague y cronometrar el centrifugado (vea la figura 2.2).

FIGURA 2.2

La adición de atributos y acciones al modelo lo acerca a la realidad.

Lavadora
marca modelo numeroSerie capacidad volumenTambor cronometroInterno trampa motor velocidadMotor
agregarRopa() agregarDetergente() sacarRopa() agregarBlanqueador() cronometrarRemojo() cronometrarLavado() cronometrarEnjuague() cronometrarCentrifugado()

Algunos conceptos

La orientación a objetos se refiere a algo más que tan sólo atributos y acciones; también considera otros aspectos. Dichos aspectos se conocen como *abstracción*, *herencia*, *polimorfismo* y *encapsulamiento* o *encapsulación*. Otros aspectos importantes de la orientación a objetos son: el *envío de mensajes*, las *asociaciones* y la *agregación*. Examinemos cada uno de estos conceptos.

Abstracción

TERMINO NUEVO

La abstracción se refiere a quitar las propiedades y acciones de un objeto para dejar sólo aquellas que sean necesarias. ¿Qué significa esto último?

Diferentes tipos de problemas requieren distintas cantidades de información, aun si estos problemas pertenecen a un área en común. En la segunda fase de la creación de la clase Lavadora, se podrían agregar más atributos y acciones que en la primera fase. ¿Vale la pena?

Valdría la pena si usted pertenece al equipo de desarrollo que generará finalmente la aplicación que simule con exactitud lo que hace una lavadora. Un programa de este tipo (que podría ser muy útil para los ingenieros de diseño que actualmente estén trabajando en el diseño de una lavadora) deberá ser tan completo que permita obtener predicciones exactas respecto a lo que ocurriría cuando se fabrique la lavadora, funcione a toda su capacidad y lave la ropa. De hecho, para este caso podrá quitar el atributo del número de serie, dado que posiblemente no será de mucha ayuda.

Por otra parte, si va a generar un software que haga un seguimiento de las transacciones en una lavandería que cuente con diversas lavadoras, posiblemente no valdrá la pena. En este programa no necesitará todos los atributos detallados y operaciones del párrafo anterior, no obstante, quizá necesite incluir el número de serie de cada objeto Lavadora.

En cualquier caso, con lo que se quedará luego de tomar su decisión respecto a lo que incluirá o desechará, será una abstracción de una lavadora.

Herencia

TERMINO NUEVO

Como ya se mencionó anteriormente, una clase es una categoría de objetos (y en el mundo del software, una plantilla sirve para crear otros objetos). Un objeto es una instancia de una clase. Esta idea tiene una consecuencia importante: como instancia de una clase, un objeto tiene todas las características de la clase de la que proviene. A esto se le conoce como *herencia*. No importa qué atributos y acciones decida usar de la clase Lavadora, cada objeto de la clase heredará dichos atributos y operaciones.

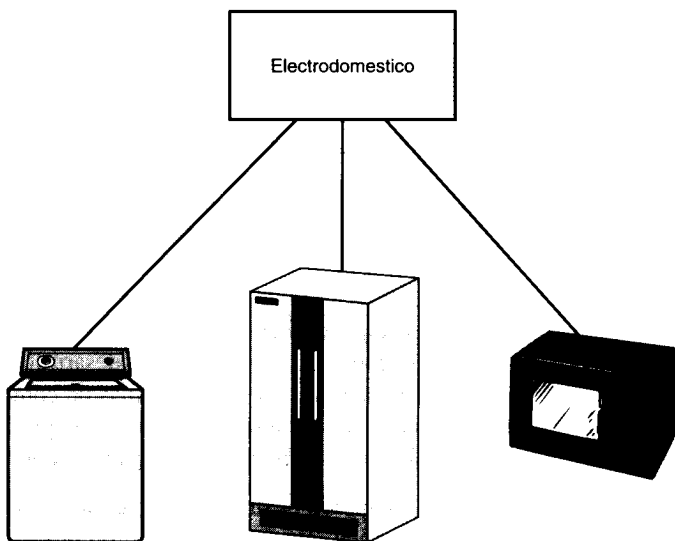
Un objeto no sólo hereda de una clase, sino que una clase también puede heredar de otra. Las lavadoras, refrigeradores, hornos de microondas, tostadores, lavaplatos, radios, licuadoras y planchas son clases y forman parte de una clase más genérica llamada: Electrodomesticos. Un electrodoméstico cuenta con los atributos de interruptor y cable eléctrico, y las operaciones de encendido y apagado. Cada una de las clases Electrodomestico heredará los mismos atributos; por ello, si sabe que algo es un electrodoméstico, de inmediato sabrá que cuenta con los atributos y acciones de la clase Electrodomestico.

TERMINO NUEVO

Otra forma de explicarlo es que la lavadora, refrigerador, horno de microondas y cosas por el estilo son *subclases* de la clase Electrodomestico. Podemos decir que la clase Electrodomestico es una *superclase* de todas las demás. La figura 2.3 le muestra la relación de superclase y subclase.

FIGURA 2.3

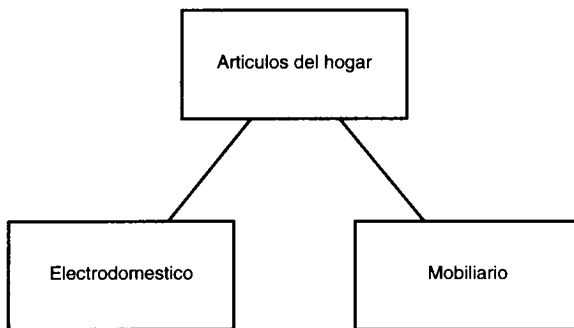
Los electrodomésticos heredan los atributos y acciones de la clase Electrodomestico. Cada electrodoméstico es una subclase de la clase Electrodomestico. La clase Electrodomestico es una superclase de cada subclase.



La herencia no tiene por qué terminar aquí. Por ejemplo, *Electrodomestico* es una subclase de *Articulos del hogar*, como le muestra la figura 2.4. Otra de las subclases de *Articulos del hogar* podría ser *Mobiliario*, que tendrá sus propias subclases.

FIGURA 2.4

Las superclases también pueden ser subclases, y heredar de otras superclases.



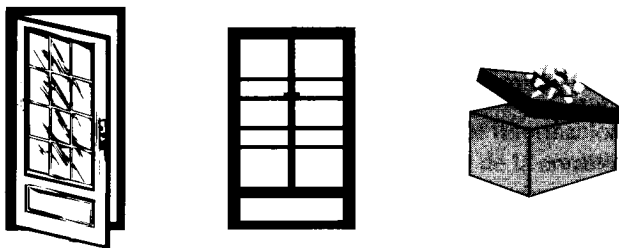
Polimorfismo

TERMINO NUEVO

En ocasiones una operación tiene el mismo nombre en diferentes clases. Por ejemplo, podrá abrir una puerta, una ventana, un periódico, un regalo o una cuenta de banco, en cada uno de estos casos, realizará una operación diferente. En la orientación a objetos, cada clase “sabe” cómo realizar tal operación. Esto es el *polimorfismo* (vea la figura 2.5).

FIGURA 2.5

En el polimorfismo, una operación puede tener el mismo nombre en diversas clases, y funcionar distinto en cada una.



En primera instancia, parecería que este concepto es más importante para los desarrolladores de software que para los modeladores, ya que los desarrolladores tienen que crear el software que implemente tales métodos en los programas de computación, y deben estar conscientes de diferencias importantes entre las operaciones que pudieran tener el mismo nombre. Y podrán generar clases de software que “sepan” lo que se supone que harán.

No obstante, el polimorfismo también es importante para los modeladores ya que les permite hablar con el cliente (quien está familiarizado con la sección del mundo que será modelada) en las propias palabras y terminología del cliente. En ocasiones, las palabras y terminología del cliente nos conducen a palabras de acción (como “abrir”) que pueden

tener más de un significado. El polimorfismo permite al modelador mantener tal terminología sin tener que crear palabras artificiales para sustentar una unicidad innecesaria de los términos.

Encapsulamiento

En cierto comercial televisivo, dos personas discuten acerca de todo el dinero que ahorrarían si marcaran un prefijo telefónico en particular antes de hacer una llamada de larga distancia.

Uno de ellos pregunta con incredulidad: “¿Cómo funciona?”

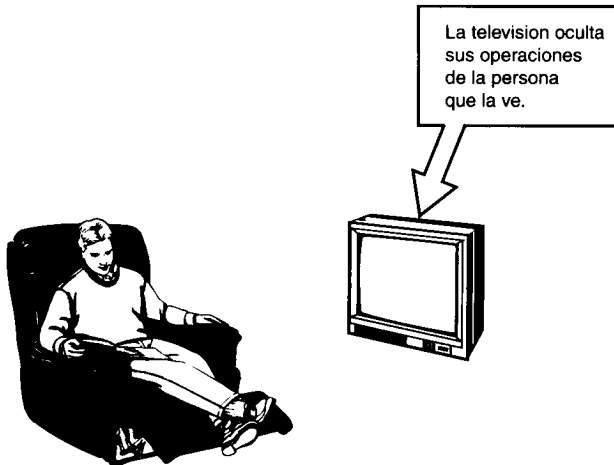
Y el otro responde: “¿Cómo hacen que las rosetas de maíz estallen? ¿A quién le importa?”

TERMINO NUEVO

La esencia del *encapsulamiento* (o encapsulación) es que cuando un objeto trae consigo su funcionalidad, esta última se oculta (vea la figura 2.6). Por lo general, la mayoría de la gente que ve la televisión no sabe o no se preocupa de la complejidad electrónica que hay detrás de la pantalla ni de todas las operaciones que tienen que ocurrir para mostrar una imagen en la pantalla. La televisión hace lo que tiene que hacer sin mostrarnos el proceso necesario para ello y, por suerte, la mayoría de los electrodomésticos funcionan así.

FIGURA 2.6

Los objetos encapsulan lo que hacen; es decir, ocultan la funcionalidad interna de sus operaciones, de otros objetos y del mundo exterior.



¿Cuál es la importancia de esto? En el mundo del software, el encapsulamiento permite reducir el potencial de errores que pudieran ocurrir. En un sistema que consta de objetos, éstos dependen unos de otros en diversas formas. Si uno de ellos falla y los especialistas de software tienen que modificarlo de alguna forma, el ocultar sus operaciones de otros objetos significará que tal vez no será necesario modificar los demás objetos.

En el mundo real, también verá la importancia del encapsulamiento en los objetos con los que trabaje. Por ejemplo, el monitor de su computadora, en cierto sentido, oculta sus operaciones de la CPU, es decir, si algo falla en su monitor, lo reparará o lo reemplazará; pero es muy probable que no tenga que reparar o reemplazar la CPU al mismo tiempo que el monitor.

TERMINO NUEVO

Ya que estamos en el tema, existe un concepto relacionado. Un objeto oculta lo que hace a otros objetos y al mundo exterior, por lo cual al encapsulamiento también se le conoce como *ocultamiento de la información*. Pero un objeto tiene que presentar un “rostro” al mundo exterior para poder iniciar sus operaciones. Por ejemplo, la televisión tiene diversos botones y perillas en sí misma o en el control remoto. Una lavadora tiene diversas perillas que le permiten establecer los niveles de temperatura y agua. Los botones y perillas de la televisión y de la lavadora se conocen como *interfaces*.

Envío de mensajes

Mencioné que en un sistema los objetos trabajan en conjunto. Esto se logra mediante el envío de mensajes entre ellos. Un objeto envía a otro un mensaje para realizar una operación, y el objeto receptor ejecutará la operación.

Una televisión y su control remoto pueden ser un ejemplo muy intuitivo del mundo que nos rodea. Cuando desea ver un programa de televisión, busca el control remoto, se sienta en su silla favorita y presiona el botón de encendido. ¿Qué ocurre? El control remoto le envía, literalmente, un mensaje al televisor para que se encienda. El televisor recibe el mensaje, lo identifica como una petición para encenderse y así lo hace. Cuando desea ver otro canal, presiona el botón correspondiente del control remoto, mismo que envía otro mensaje a la televisión (cambiar de canal). El control remoto también puede comunicar otros mensajes como ajustar el volumen, silenciar y activar los subtítulos.

Volvamos a las interfaces. Muchas de las cosas que hace mediante el control remoto, también las podrá hacer si se levanta de la silla, va a la televisión y presiona los botones correspondientes (¡alguna vez lo habrá hecho ya!). La interfaz que la televisión le presenta (el conjunto de botones y perillas) no es, obviamente, la misma que le muestra al control remoto (un receptor de rayos infrarrojos). La figura 2.7 le muestra esto.

Asociaciones

Otro acontecimiento común es que los objetos se relacionan entre sí de alguna forma. Por ejemplo, cuando enciende su televisor, en términos de orientación a objetos, usted se asocia con su televisor.

La asociación “encendido” es en una sola dirección (una vía), esto es, usted enciende la televisión, como se ve en la figura 2.8. No obstante, a menos que vea demasiada televisión, ella no le devolverá el favor. Hay otras asociaciones que son en dos direcciones, como “casamiento”.

FIGURA 2.7

Ejemplo de un mensaje enviado de un objeto a otro: el objeto "control remoto" envía un mensaje al objeto "televisión" para encenderse. El objeto "televisión" recibe el mensaje mediante su interfaz, un receptor infrarrojo.

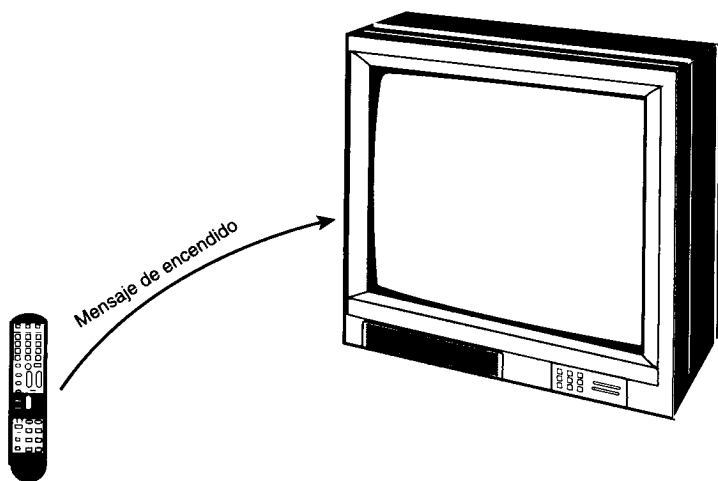
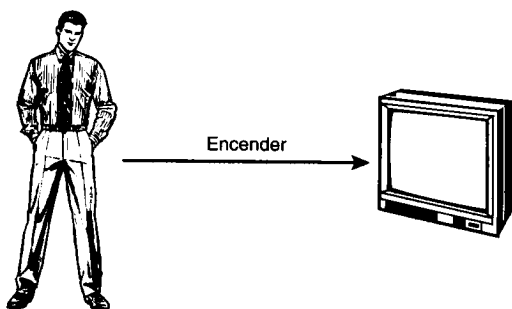


FIGURA 2.8

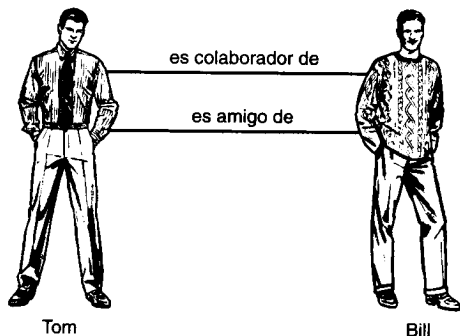
Con frecuencia los objetos se relacionan entre sí de alguna forma. Cuando usted enciende su televisión, tendrá una asociación en una sola dirección con ella.



En ocasiones, un objeto podría asociarse con otro en más de una forma. Si usted y su colaborador son amigos, ello servirá de ejemplo. Usted tendría una asociación "es amigo de", así como "es colaborador de", como se aprecia en la figura 2.9.

FIGURA 2.9

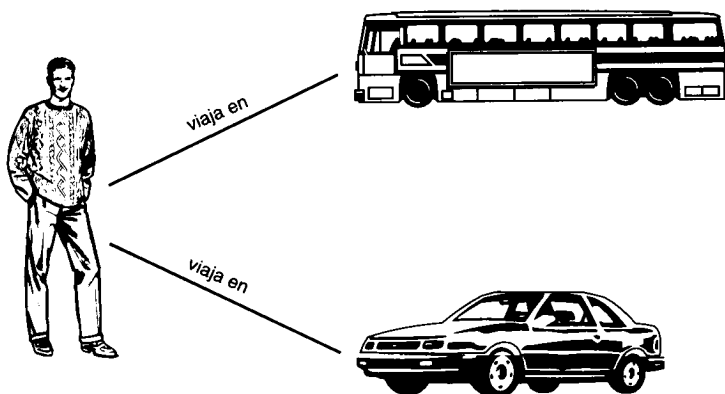
En ocasiones, los objetos pueden asociarse en más de una forma.



Una clase se puede asociar con más de una clase distinta. Una persona puede viajar en automóvil, pero también puede hacerlo en autobús (vea la figura 2.10).

FIGURA 2.10

Una clase puede asociarse con más de una clase distinta.



TERMINO NUEVO

La **multiplicidad** (o diversificación) es un importante aspecto de las asociaciones entre objetos. Indica la cantidad de objetos de una clase que se relacionan con otro objeto en particular de la clase asociada. Por ejemplo, en un curso escolar, el curso se imparte por un solo instructor, en consecuencia, el curso y el instructor están en una asociación de uno a uno. Sin embargo, en un seminario hay diversos instructores que impartirán el curso a lo largo del semestre, por lo tanto, el curso y el instructor tienen una asociación de uno a muchos.

Podrá encontrar todo tipo de multiplicidades si echa una mirada a su alrededor. Una bicicleta rueda en dos neumáticos (multiplicidad de uno a dos), un triciclo rueda en tres, y un vehículo de 18 ruedas, en 18.

Agregación

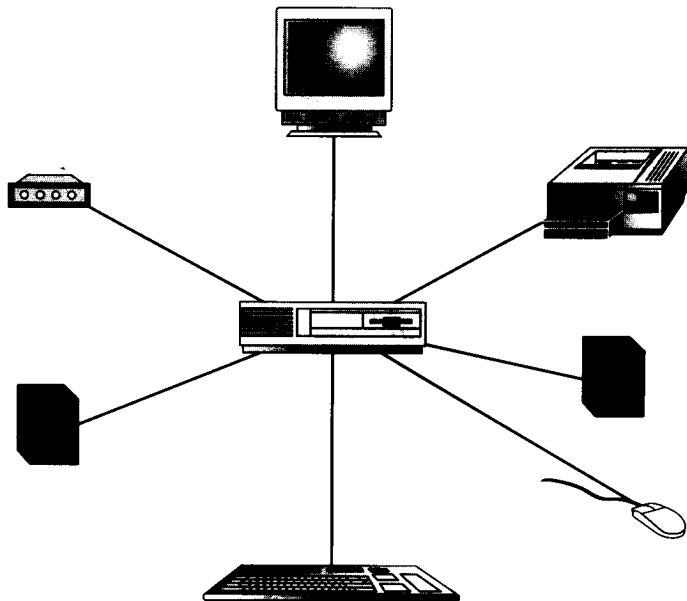
Vea su computadora: cuenta con un gabinete, un teclado, un ratón, un monitor, una unidad de CD-ROM, uno o varios discos duros, un módem, una unidad de disquete, una impresora y, posiblemente, bocinas. Dentro del gabinete, junto con las citadas unidades de disco, tiene una CPU, una tarjeta de vídeo, una de sonido y otros elementos sin los que, sin duda, difícilmente podría vivir.

TERMINO NUEVO

Su computadora es una **agregación** o adición, otro tipo de asociación entre objetos. Como muchas otras cosas que valdrían la pena tener, su equipo está constituido de diversos tipos de componentes (vea la figura 2.11). Tal vez conozca varios ejemplos de agregaciones.

FIGURA 2.11

Una computadora es un ejemplo de agregación: un objeto que se conforma de una combinación de diversos tipos de objetos.



TERMINO NUEVO

Un tipo de agregación trae consigo una estrecha relación entre un objeto agregado y sus objetos componentes. A esto se le conoce como *composición*.

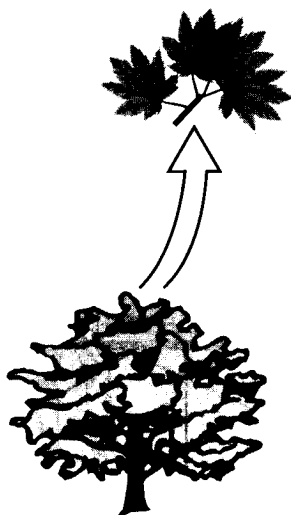
El punto central de la composición es que el componente se considera como tal sólo como parte del objeto compuesto. Por ejemplo: una camisa está compuesta de cuerpo, cuello, mangas, botones, ojales y puños. Suprima la camisa y el cuello será inútil.

En ocasiones, un objeto compuesto no tiene el mismo tiempo de vida que sus propios componentes. Las hojas de un árbol pueden morir antes que el árbol. Si destruye al árbol, también las hojas morirán (vea la figura 2.12).

La agregación y la composición son importantes dado que reflejan casos extremadamente comunes, y ello ayuda a que cree modelos que se asemejen considerablemente a la realidad.

FIGURA 2.12

En una composición, un componente puede morir antes que el objeto compuesto. Si destruye al objeto compuesto, destruirá también a los componentes.



La recompensa

Los objetos y sus asociaciones conforman la columna vertebral de la funcionalidad de los sistemas. Para modelarlos, necesitará comprender lo que son las asociaciones. Si está consciente de los posibles tipos de asociaciones, tendrá una amplia gama de posibilidades cuando hable con los clientes respecto a sus necesidades, obtendrá sus requerimientos y creará los modelos de los sistemas que los ayudarán a cumplir con sus retos de negocios.

TERMINO NUEVO

Lo importante es utilizar los conceptos de la orientación a objetos para ayudarle a comprender el área de conocimiento de su cliente (su *dominio*), y esclarecer sus puntos de vista al cliente en términos que él o ella puedan comprender.

Es allí donde se aplica el UML. En las siguientes tres horas, aprenderá a aplicar el UML para visualizar los conceptos que ha aprendido durante esta hora.

Resumen

La orientación a objetos es un paradigma que depende de algunos principios fundamentales. Un objeto es una instancia de una clase. Una clase es una categoría genérica de objetos que tienen los mismos atributos y acciones. Cuando crea un objeto, el área del problema en que trabaje determinará cuántos de los atributos y acciones debe tomar en cuenta.

La herencia es un aspecto importante de la orientación a objetos, un objeto hereda los atributos y operaciones de su clase. Una clase también puede heredar atributos y acciones de otra.

El polimorfismo es otro aspecto importante, ya que especifica que una acción puede tener el mismo nombre en diferentes clases y cada clase ejecutará tal operación de forma distinta.

Los objetos ocultan su funcionalidad de otros objetos y del mundo exterior. Cada objeto presenta una interfaz para que otros objetos (y personas) puedan aprovechar su funcionalidad.

Los objetos funcionan en conjunto mediante el envío de mensajes entre ellos. Los mensajes son peticiones para realizar operaciones.

Por lo general, los objetos se asocian entre sí y esta asociación puede ser de diversos tipos. Un objeto en una clase puede asociarse con cualquier cantidad de objetos distintos en otra clase.

La agregación es un tipo de asociación. Un objeto agregado consta de un conjunto de objetos que lo componen y una composición es un tipo especial de agregación. En un objeto compuesto, los componentes sólo existen como parte del objeto compuesto.

Preguntas y respuestas

P Usted dijo que la orientación a objetos ha tomado por asalto al mundo del software. ¿Qué no hay algunas aplicaciones importantes que no están orientadas a objetos?

R Sí, y se conocen como sistemas “heredados” (programas que en muchos casos son ejecutados para mostrar su época). La orientación a objetos ofrece diversas ventajas, como la reusabilidad y un rápido periodo de desarrollo. Por tales razones, muy probablemente verá las nuevas aplicaciones (y las versiones rediseñadas de varias aplicaciones antiguas) escritas bajo el esquema de la orientación a objetos.

Taller

Para repasar lo que ha aprendido de la orientación a objetos, intente responder a algunas preguntas y realizar los siguientes ejercicios. Las respuestas las encontrará en el Apéndice A, “Respuestas a los cuestionarios”.

Cuestionario

1. ¿Qué es un objeto?
2. ¿Cómo trabajan los objetos en conjunto?
3. ¿Qué establece la multiplicidad?
4. ¿Pueden asociarse dos objetos entre sí en más de una manera?

Ejercicios

Esta es una hora teórica, así que no incluí ejercicios. Verá algunos en las siguientes horas.

HORA 3

Uso de la orientación a objetos

A continuación conjugaremos las características del UML con los conceptos de la orientación a objetos. Aquí reafirmará su conocimiento de la orientación a objetos al tiempo que aprenderá otras cosas del UML.

En esta hora se tratarán los siguientes temas:

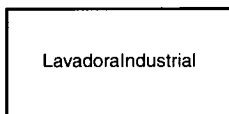
- Concepción de una clase
- Atributos
- Operaciones
- Responsabilidades y restricciones
- Qué es lo que hacen las clases y cómo encontrarlas

Concepción de una clase

Como lo indiqué en la primera hora, en el UML un rectángulo es el símbolo que representa una clase. El nombre de la clase es, por convención, una palabra con la primera letra en mayúscula y normalmente se coloca en la parte superior del rectángulo. Si el nombre de su clase consta de dos palabras, únalas e inicie cada una con mayúscula (como en LavadoraIndustrial en la figura 3.1).

FIGURA 3.1

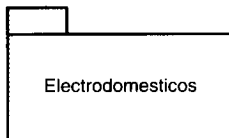
La representación
UML de una clase.



Otra estructura del UML, el paquete, puede jugar un papel en el nombre de la clase. Como indiqué en la hora 1, “Introducción al UML”, un paquete es la manera en que el UML organiza un diagrama de elementos. Tal vez recuerde que el UML representa un paquete como una carpeta tabular cuyo nombre es una cadena de texto (vea la figura 3.2).

FIGURA 3.2

Un paquete del UML.



TERMINO NUEVO

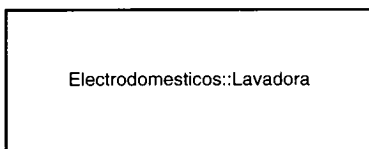
Si la clase “Lavadora” es parte de un paquete llamado “Electrodomesticos”, podrá darle el nombre “Electrodomesticos::Lavadora”. El par de dos puntos separa al nombre del paquete, que está a la izquierda, del nombre de la clase, que va a la derecha. A este tipo de nombre de clase se le conoce como *nombre de ruta* (vea la figura 3.3).



Posiblemente haya notado que en los nombres se han evitado los caracteres acentuados (como en Electrodomesticos) y la letra ñe. Esto se debe a que en el alfabeto inglés, tales caracteres no están contemplados y no podemos asegurar que el utilizarlos en sus identificadores no le traiga problemas, tanto en el UML como en el lenguaje de programación que piense utilizar para traducir los modelos. Por ello, evitaremos los acentos en todos los diagramas que se presentan a lo largo de este libro, de igual manera, evitaremos el uso de la letra ñe, misma que sustituiremos —en su caso— por “ni” (como en Anio, en lugar de Año).

FIGURA 3.3

Una clase con un
nombre de ruta.



Atributos

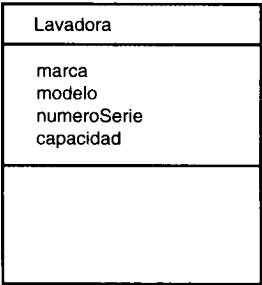
TERMINO NUEVO

Un *atributo* es una propiedad o característica de una clase y describe un rango de valores que la propiedad podrá contener en los objetos (esto es, instancias) de la clase. Una clase podrá contener varios o ningún atributo. Por convención, si el atributo consta de una sola palabra se escribe en minúsculas; por otro lado, si el nombre contiene más de una palabra, cada palabra será unida a la anterior y comenzará con una letra mayúscula, a excepción de la primer palabra que comenzará en minúscula. La lista

de nombres de atributos iniciará luego de una línea que la separe del nombre de la clase, como se aprecia en la figura 3.4.

FIGURA 3.4

Una clase y sus atributos.



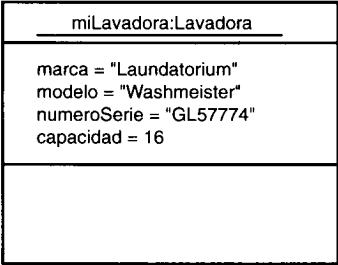
Todo objeto de la clase tiene un valor específico en cada atributo. La figura 3.5 le muestra un ejemplo. Observe que el nombre de un objeto inicia con una letra minúscula, y está precedido de dos puntos que a su vez están precedidos del nombre de la clase, y todo el nombre está subrayado.



El nombre `miLavadora:Lavadora` es una *instancia con nombre*; pero también es posible tener una *instancia anónima*, como `:Lavadora`.

FIGURA 3.5

Un objeto cuenta con un valor específico en cada uno de los atributos que lo componen.



El UML le da la opción de indicar información adicional de los atributos. En el símbolo de la clase, podrá especificar un tipo para cada valor del atributo. Entre los posibles tipos se encuentran cadena (string), número de punto flotante (float), entero (integer) y booleano (boolean), así como otros tipos enumerados. Para indicar un tipo, utilice dos puntos (:) para separar el nombre del atributo de su tipo. También podrá indicar un valor predefinido para un atributo. La figura 3.6 le muestra las formas de establecer atributos.



Aunque no parece haber restricción en la designación de tipos a las variables, utilizaremos los nombres en inglés para ceñirnos a los tipos que aparecen en los lenguajes de programación.

FIGURA 3.6

Un atributo puede mostrar su tipo así como su valor predeterminado.

Lavadora
marca: string = "Laundatorium" modelo: string numeroSerie: string capacidad: integer

Operaciones

TERMINO NUEVO

Una *operación* es algo que la clase puede realizar, o que usted (u otra clase) pueden hacer a una clase. De la misma manera que el nombre de un atributo, el nombre de una operación se escribe en minúsculas si consta de una sola palabra. Si el nombre constara de más de una palabra, únalas e inicie todas con mayúscula exceptuando la primera. La lista de operaciones se inicia debajo de una línea que separa a las operaciones de los atributos, como se muestra en la figura 3.7.

FIGURA 3.7

La lista de operaciones de una clase aparece debajo de una línea que las separa de los atributos de la clase.

Lavadora
marca modelo numeroSerie capacidad
agregarRopa() sacarRopa() agregarDetergente() activar()

TERMINO NUEVO

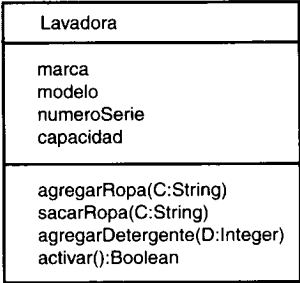
Así como es posible establecer información adicional de los atributos, también lo es en lo concerniente a las operaciones. En los paréntesis que preceden al nombre de la operación podrá mostrar el parámetro con el que funcionará la operación junto con su tipo de dato. La *función*, que es un tipo de operación, devuelve un valor luego que finaliza su trabajo. En una función podrá mostrar el tipo de valor que regresará.

TERMINO NUEVO

Estas secciones de información acerca de una operación se conocen como la *firma* de la operación. La figura 3.8 le muestra cómo representar la firma.

FIGURA 3.8

La firma de una operación.

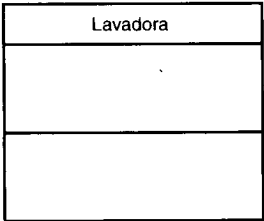


Atributos, operaciones y concepción

Hasta ahora, hemos tratado a las clases como entidades aisladas, y hemos visto todos los atributos y operaciones de una clase. No obstante, en la práctica mostrará más de una clase a la vez; cuando lo haga, no será muy útil que siempre aparezcan todos los atributos y operaciones, ya que el hacerlo le crearía un diagrama muy saturado. En lugar de ello podrá tan sólo mostrar el nombre de la clase y dejar ya sea el área de atributos o el de operaciones (o ambas) vacía, como se muestra en la figura 3.9.

FIGURA 3.9

En la práctica, no siempre mostrará todos los atributos y operaciones de una clase.

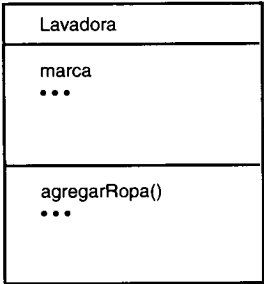


TERMINO NUEVO

En ocasiones será bueno mostrar algunos (pero no todos) de los atributos u operaciones. Para indicar que sólo enseñará algunos de ellos, seguirá la lista de aquellos que mostrará con tres puntos (...), mismos que se conocen como *puntos suspensivos*. A la omisión de ciertos o todos los atributos y operaciones se le conoce como *abreviar* una clase. La figura 3.10 le muestra el uso de los puntos suspensivos.

FIGURA 3.10

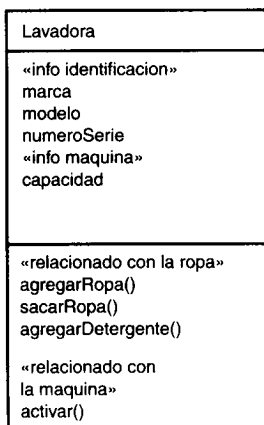
Los puntos suspensivos indican atributos u operaciones que no se encuentran en todo el conjunto.



Si usted tiene una larga lista de atributos u operaciones podrá utilizar un estereotipo para organizarla de forma que sea más comprensible. Un estereotipo es el modo en que el UML le permite extenderlo, es decir, crear nuevos elementos que son específicos de un problema en particular que intente resolver. Como lo mencioné en la hora 1, usted muestra un estereotipo como un nombre bordeado por dos pares de paréntesis angulares. Para una lista de atributos, podrá utilizar un estereotipo como encabezado de un sub-conjunto de atributos, como en la figura 3.11.

FIGURA 3.11

Podrá usar un estereotipo para organizar una lista de atributos u operaciones.



El estereotipo es una estructura flexible, la cual podrá utilizar de diversos modos. Por ejemplo, podrá utilizar el estereotipo sobre el nombre de una clase en un símbolo de clase para indicar algo respecto al papel de la clase.

Responsabilidades y restricciones

TERMINO NUEVO

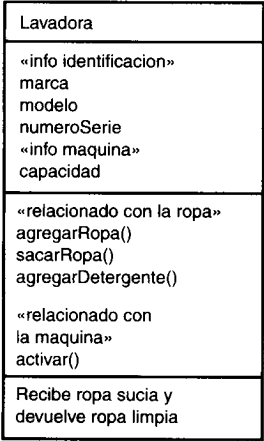
El símbolo de clase le permite establecer otro tipo de información de sí misma.

En un área bajo la lista de operaciones, podrá mostrar la responsabilidad de la clase. La *responsabilidad* es una descripción de lo que hará la clase, es decir, lo que sus atributos y operaciones intentan realizar en conjunto. Una lavadora, por ejemplo, tiene la responsabilidad de recibir ropa sucia y dar por resultado ropa limpia.

En el símbolo, indicará las responsabilidades en un área inferior a la que contiene las operaciones (vea la figura 3.12).

FIGURA 3.12

En un símbolo de clase, que irá debajo de la lista de operaciones, escribirá las responsabilidades de la clase.



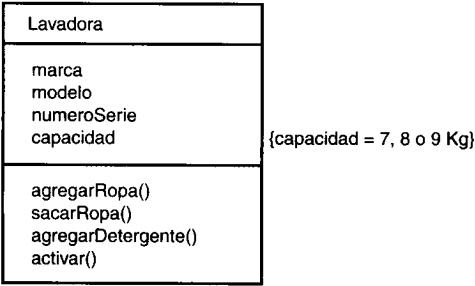
La idea es incluir información suficiente para describir una clase de forma inequívoca.

TERMINO NUEVO

Una manera más formal es agregar una *restricción*, un texto libre bordeado por llaves; este texto especifica una o varias reglas que sigue la clase. Por ejemplo, suponga que en la clase Lavadora usted desea establecer que la capacidad de una lavadora será de 7, 8 o 9 Kg (y así, “restringir” el atributo capacidad de la clase Lavadora). Usted escribirá {capacidad = 7, 8 o 9 Kg} junto al símbolo de la clase Lavadora. La figura 3.13 le muestra cómo hacerlo.

FIGURA 3.13

La regla entre llaves restringe al atributo capacidad para contener uno de tres posibles valores.



El UML funciona con otra forma –aún más formal– de agregar restricciones que hacen más explícitas las definiciones. Es todo un lenguaje conocido como *OCL (Lenguaje de restricción de objetos)*. OCL cuenta con su propio conjunto de reglas, términos y operadores, lo que lo convierte en una herramienta avanzada y, en ocasiones, útil.

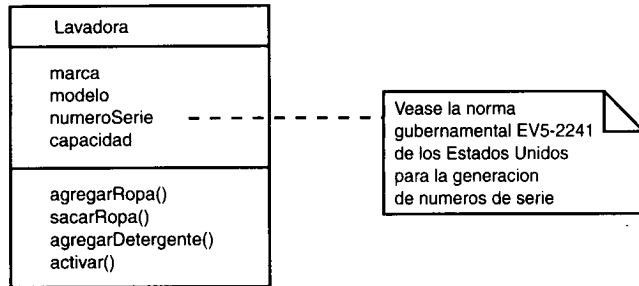
Notas adjuntas

Por encima y debajo de los atributos, operaciones, responsabilidades y restricciones, puede agregar mayor información a una clase en la figura de notas adjuntas.

Con frecuencia agregará una nota a un atributo u operación. La figura 3.14 le muestra una nota que se refiere a una norma gubernamental que indica dónde encontrar la manera en que se generan los números de serie para los objetos de la clase Lavadora.

FIGURA 3.14

Una nota adjunta proporciona mayor información respecto a la clase.



Una nota puede contener tanto una imagen como texto.

Qué es lo que hacen las clases y cómo encontrarlas

Las clases son el vocabulario y terminología de un área del conocimiento. Conforme hable con los clientes, analice su área de conocimiento y diseñe sistemas de computación que resuelvan los problemas de dicha área, comprenderá la terminología y modelará los términos como clases en el UML.

En sus conversaciones con los clientes preste atención a los sustantivos que utilizan para describir las entidades de sus negocios; ya que dichos sustantivos se convertirán en las clases de su modelo. También preste atención a los verbos que escuche, dado que constituirán las operaciones de sus clases. Los atributos surgirán como sustantivos relacionados con los nombres de la clase. Una vez que tenga una lista básica de las clases, pregunte a los clientes qué es lo que cada clase hace dentro del negocio. Sus respuestas le indicarán las responsabilidades de la clase.

Suponga que usted es un analista que generará un modelo del juego de baloncesto, y que entrevista a un entrenador para comprender el juego. La conversación podría surgir como sigue:

Analista: “Entrenador, ¿de qué se trata el juego?”

Entrenador: “Consiste en arrojar el balón a través de un aro, conocido como cesto, y hacer una mayor puntuación que el oponente. Cada equipo consta de cinco jugadores: dos defensas, dos delanteros y un central. Cada equipo lleva el balón al cesto del equipo oponente con el objetivo de hacer que el balón sea encestando.”

Analista: “¿Cómo se hace para llevar el balón al otro cesto?”

Entrenador: “Mediante pases y dribles. Pero el equipo tendrá que encestar antes de que termine el lapso para tirar.”

Analista: “¿El lapso para tirar?”

Entrenador: “Así es, son 24 segundos en el baloncesto profesional, 30 en un juego internacional, y 35 en el colegial para tirar el balón luego de que un equipo toma posesión de él.”

Analista: “¿Cómo funciona el puntaje?”

Entrenador: “Cada canasta vale dos puntos, a menos que el tiro haya sido hecho detrás de la línea de los tres puntos. En tal caso, serán tres puntos. Un tiro libre contará como un punto. A propósito, un tiro libre es la penalización que paga un equipo por cometer una infracción. Si un jugador infracciona a un oponente, se detiene el juego y el oponente puede realizar diversos tiros al cesto desde la línea de tiro libre.”

Analista: “Hábleme más acerca de lo que hace cada jugador.”

Entrenador: “Quienes juegan de defensa son, en general, quienes realizan la mayor parte de los dribles y pases. Por lo general tienen menor estatura que los delanteros, y éstos, a su vez, son menos altos que el central (que también se conoce como ‘poste’). Se supone que todos los jugadores pueden burlar, pasar, tirar y rebotar. Los delanteros realizan la mayoría de los rebotes y los disparos de mediano alcance, mientras que el central se mantiene cerca del cesto y dispara desde un alcance corto.”

Analista: “¿Qué hay de las dimensiones de la cancha? Y ya que estamos en eso ¿cuánto dura el juego?”

Entrenador: “En un juego internacional, la cancha mide 28 metros de longitud y 15 de ancho; el cesto se encuentra a 3.05 m del piso. En un juego profesional, el juego dura 48 minutos, divididos en cuatro cuartos de 12 minutos cada uno. En un juego colegial e internacional, la duración es de 40 minutos, divididos en dos mitades de 20 minutos. Un cronómetro del juego lleva un control del tiempo restante.”

La plática podría continuar, pero hagamos una pausa y veamos con qué contamos. Aquí hay varios sustantivos que ha descubierto: balón, cesto, equipo, jugadores, defensas, delanteros, centro (o poste), tiro, lapso para tirar, línea de los tres puntos, tiro libre, infracción, línea de tiro libre, cancha, cronómetro del juego.

Y los verbos: tirar, avanzar, driblar (o burlar), pasar, infraccionar, rebotar. También cuenta con cierta información adicional respecto a algunos de los sustantivos (como las

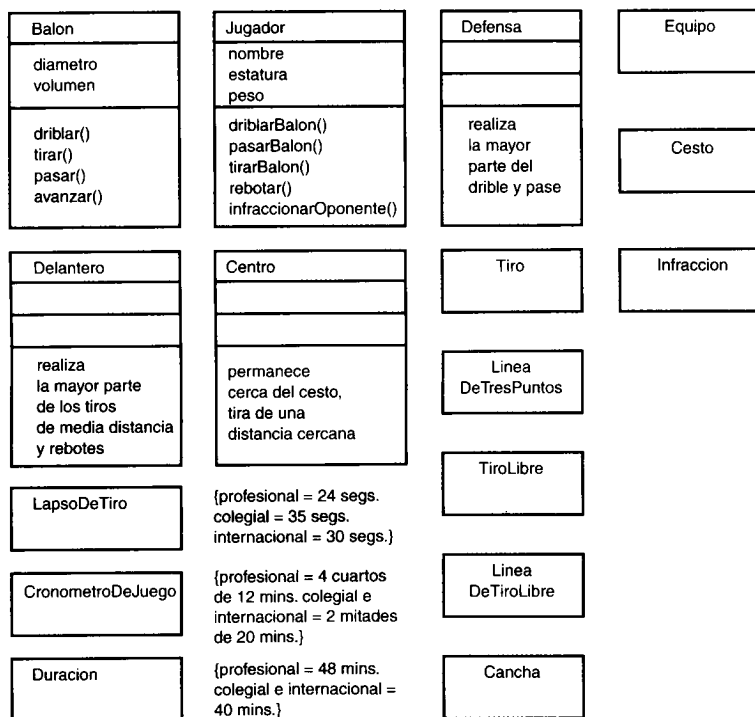
estaturas relativas de los jugadores de cada posición, las dimensiones de la cancha, la cantidad total de tiempo en un lapso de tiro y la duración de un juego).

Finalmente, su propio sentido común podría entrar en acción para generar ciertos atributos por usted mismo. Usted sabe, por ejemplo, que el balón cuenta con ciertos atributos, como volumen y diámetro.

A partir de esta información, podrá crear un diagrama como el de la figura 3.15. En él se muestran las clases y se proporcionan ciertos atributos, operaciones y restricciones. El diagrama también muestra las responsabilidades. Podría usar este diagrama como fundamento para otras conversaciones con el entrenador para obtener mayor información.

FIGURA 3.15

Un diagrama inicial para modelar el juego de baloncesto.



Resumen

Un rectángulo es, en el UML, la representación simbólica de una clase. El nombre, atributos, operaciones y responsabilidades de la clase se colocan en áreas delimitadas dentro del rectángulo. Puede utilizar un estereotipo para organizar las listas de atributos y operaciones y además abreviar una clase al mostrar sólo un subconjunto de sus atributos y operaciones. Esto hace un diagrama de clases menos complejo.

Podrá mostrar el tipo de un atributo, su valor inicial y enseñar los valores con que funciona una operación, así como sus tipos. En una operación, esta información se conoce como firma.

Para reducir la ambigüedad en la descripción de una clase agregue restricciones. El UML también le permite indicar mayor información respecto a una clase mediante notas adjuntas al rectángulo que la representa.

Las clases representan el vocabulario de un área del conocimiento. Las conversaciones con el cliente o un experto en el área dejarán entrever los sustantivos que se convertirán en clases en un modelo, y los verbos se transformarán en operaciones. Podrá utilizar un diagrama de clases como una forma de estimular al cliente a que diga más respecto a su área y que ponga en evidencia cierta información adicional.

Preguntas y respuestas

- P** Usted mencionó el uso del “sentido común” para generar el diagrama de clases del baloncesto. Ello suena bien en tal instancia pero, ¿qué ocurre cuando tengo que analizar un área desconocida para mí (donde el sentido común no será de mucha ayuda)?
- R** Por lo general, contará con cierto apoyo en un área desconocida para usted. Antes de que se reúna con un cliente o con un experto en el campo, intente convertirse en un “subexperto”. Prepárese para la reunión y lea cuanta documentación relacionada tenga a la mano. Pregunte a sus entrevistados respecto a documentos o manuales que hayan escrito. Cuando haya terminado de leer, tendrá cierto conocimiento básico y podrá realizar las preguntas indicadas.
- P** ¿En qué momento tendría que mostrar la firma de una operación?
- R** Tal vez, luego de la fase de análisis de un proceso de desarrollo, conforme se adentre en el diseño. La firma es una sección de información que los desarrolladores podrían encontrar muy útil.

Taller

Para repasar lo que ha aprendido respecto a la orientación a objetos, intente responder a las siguientes preguntas. Las respuestas las encontrará en el Apéndice A, “Respuestas a los cuestionarios”.

Cuestionario

1. ¿Cómo representa una clase en el UML?
2. ¿Qué información puede mostrar en un símbolo de clase?
3. ¿Qué es una restricción?
4. ¿Para qué adjuntaría una nota a un símbolo de clase?

Ejercicios

1. He aquí una breve (e incompleta) descripción del balompié:

Un equipo de balompié (o fútbol soccer) consiste en 11 jugadores de campo (1 portero y el resto, jugadores de cancha que, en ocasiones, se organizan en cuatro defensas, tres centrales y tres delanteros). Los jugadores pueden usar cualquier parte de su cuerpo (excepto las manos) para introducir el balón a la portería del equipo contrario. La única excepción a esta regla la tiene el portero, quien puede utilizar también las manos para jugar el balón, pero sólo dentro del área de meta. El campo de juego es un rectángulo de una longitud máxima de 120 m y mínima de 90 m; y con una anchura no mayor de 90 m, ni menor de 45. Para partidos internacionales, la longitud será de 110 m como máximo y 100 como mínimo; y una anchura no superior a 75 m ni inferior a 64. En cualquier caso, deberá ser mayor la longitud que la anchura. El campo de juego se dividirá en dos mitades transversales de igual tamaño. El centro del campo será marcado con un punto visible, alrededor del cual se trazará una circunferencia de 9.15 m de radio. La meta del juego es pasar el balón a los delanteros, quienes están mejor preparados para patear el balón a la portería. El portero (o arquero) es la última línea de defensa que intentará bloquear, con cualquier parte de su cuerpo, los tiros de sus opositores. Cada vez que evita un gol, es decir, que el balón entre a la portería, habrá salvado su meta. Cada gol equivale a un punto. Un juego dura 90 minutos, divididos en dos periodos de 45 minutos cada uno.

Válgase de la anterior información para crear un diagrama como el de la figura 3.15. Si usted conoce más del balompié de lo que he descrito, agregue tal información a su diagrama.

2. Si usted conoce más del baloncesto de lo que hay en la figura 3.15, agregue la información a tal diagrama.

HORA 4



Uso de relaciones

En la hora anterior creamos un conjunto de clases que representaban el vocabulario del baloncesto. Aunque ello le da las bases para una mayor exploración de lo que es el baloncesto, tal vez haya sentido que algo le falta.

Ese “algo” es un sentido en el que las clases se relacionan entre sí. Si observa el modelo (vea la figura 3.15), verá que no se indica la manera en que un jugador se relaciona con un balón, ni cómo los jugadores conforman un equipo, ni la forma en que procede el juego. Es como si hubiera construido una lista de elementos, en lugar de una representación de un área del conocimiento. Es importante saber cómo se conectan las clases entre sí.

Ahora trazaremos las conexiones entre las clases y completaremos la representación.

En esta hora se tratarán los siguientes temas:

- Asociaciones
- Multiplicidad
- Asociaciones calificadas

- Asociaciones reflexivas
- Herencia y generalización
- Dependencias

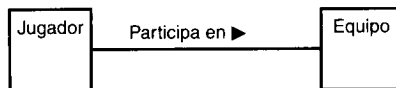
Asociaciones

TERMINO NUEVO

Cuando las clases se conectan entre sí de forma conceptual, esta conexión se conoce como *asociación*. El modelo inicial de baloncesto le dará algunos ejemplos. Examinemos uno de ellos: la asociación entre un jugador y un equipo. Podrá caracterizar tal asociación con la frase: “un jugador participa en un equipo”. Visualizará la asociación como una línea que conectará a ambas clases, con el nombre de la asociación (“participa en”) justo sobre la línea. Es útil indicar la dirección de la relación, y lo hará con un triángulo relleno que apunte en la dirección apropiada. La figura 4.1 le muestra cómo visualizar la asociación “Participa en” entre el jugador y el equipo.

FIGURA 4.1

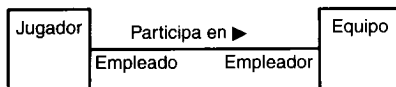
Una asociación entre un jugador y un equipo.



Cuando una clase se asocia con otra, cada una de ellas juega un papel dentro de tal asociación. Puede representar estos papeles en el diagrama escribiéndolos cerca de la línea que se encuentra junto a la clase que juega el papel correspondiente. En la asociación entre un jugador y un equipo, si el equipo es profesional, éste es un empleador y el jugador es un empleado. La figura 4.2 le muestra cómo representar dichos papeles.

FIGURA 4.2

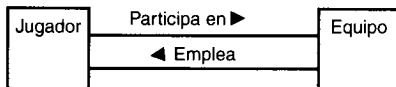
Por lo general, en una asociación cada clase juega un papel. Puede representar tales papeles en el diagrama.



La asociación puede funcionar en dirección inversa: un equipo emplea a jugadores. Podrá mostrar ambas asociaciones en el mismo diagrama con un triángulo relleno que indique la dirección de cada asociación, como en la figura 4.3.

FIGURA 4.3

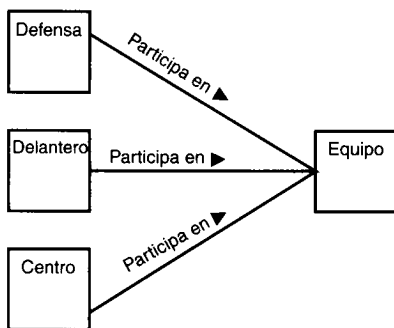
Pueden aparecer dos asociaciones entre clases en el mismo diagrama.



Las asociaciones podrían ser más complejas que tan sólo una clase conectada a otra. Varias clases se pueden conectar a una. Si toma en cuenta los defensas, delanteros y central, así como sus asociaciones con la clase Equipo, tendrá el diagrama de la figura 4.4.

FIGURA 4.4

Pueden asociarse diversas clases con una en particular.

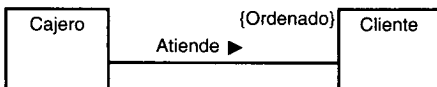


Restricciones en las asociaciones

En ocasiones una asociación entre dos clases debe seguir cierta regla. Ésta se indica al establecer una restricción junto a la línea de asociación. Por ejemplo: un Cajero atiende a un Cliente, pero cada Cliente es atendido en el orden en que se encuentre en la formación. Puede capturar este modelo colocando la palabra *ordenado* entre llaves (para indicar la restricción) junto a la clase Cliente, como se ve en la figura 4.5.

FIGURA 4.5

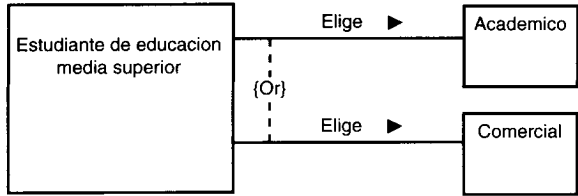
Puede establecer una restricción en una asociación. En este caso, la asociación Atiende está restringida para que el Cajero atienda al Cliente en turno.



Otro tipo de restricción es la relación O (distinguida como {Or}) en una línea discontinua que conecte a dos líneas de asociación. La figura 4.6 modela a un estudiante de educación media superior que elegirá entre un curso académico o uno comercial.

FIGURA 4.6

La relación O entre dos asociaciones en una restricción.



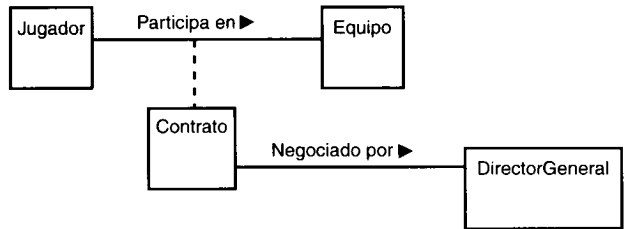
Clases de asociación

TERMINO NUEVO Una asociación, al igual que una clase, puede contener atributos y operaciones. De hecho, cuando éste sea el caso, usted tendrá una *clase de asociación*.

Puede concebir a una clase de asociación de la misma forma en que lo haría con una clase estándar, y utilizará una línea discontinua para conectarla a la línea de asociación. Una clase de asociación puede tener asociaciones con otras clases. La figura 4.7 le muestra una clase de asociación para la asociación “Participa en” entre un jugador y un equipo. La clase de asociación, Contrato, se asocia con la clase DirectorGeneral.

FIGURA 4.7

Una clase de asociación modela los atributos y operaciones de una asociación. Se conecta a una asociación mediante una línea discontinua, y puede asociarse a otra clase.

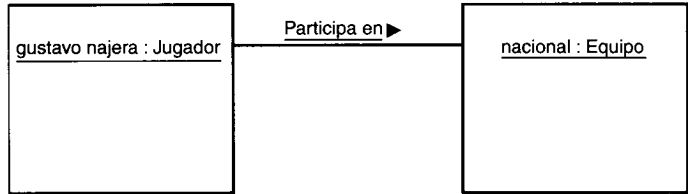


Vínculos

Así como un objeto es una instancia de una clase, una asociación también cuenta con instancias. Si podemos imaginar a un jugador específico que juega para un equipo específico, la relación “Participa en” se conocerá como *vínculo*, y usted lo representará como una línea que conecta a dos objetos. Tal como tuvo que subrayar el nombre de un objeto, deberá subrayar el nombre de un vínculo, como en la figura 4.8.

FIGURA 4.8

Un vínculo es la instancia de una asociación. Conecta a los objetos en lugar de las clases. Deberá subrayar el nombre del vínculo, como se hace en el nombre de un objeto.



Multiplicidad

La asociación trazada entre Jugador y Equipo sugiere que las dos clases tienen una relación de uno a uno. No obstante, el sentido común nos indica que éste no es el caso. Un equipo de baloncesto cuenta con cinco jugadores (sin contar a los sustitutos). La asociación Tiene (Has) debe participar en este recuento. En la otra dirección, un jugador puede participar sólo en un equipo, y la asociación “Participa en” debe responder de esto.

TERMINO NUEVO

Tales especificaciones son ejemplos de la *multiplicidad*: la cantidad de objetos de una clase que se relacionan con un objeto de la clase asociada. Para representar los números en el diagrama, los colocará sobre la línea de asociación junto a la clase correspondiente, como se denota en la figura 4.9.

FIGURA 4.9

La multiplicidad señala la cantidad de objetos de una clase que pueden relacionarse con un objeto de una clase asociada.



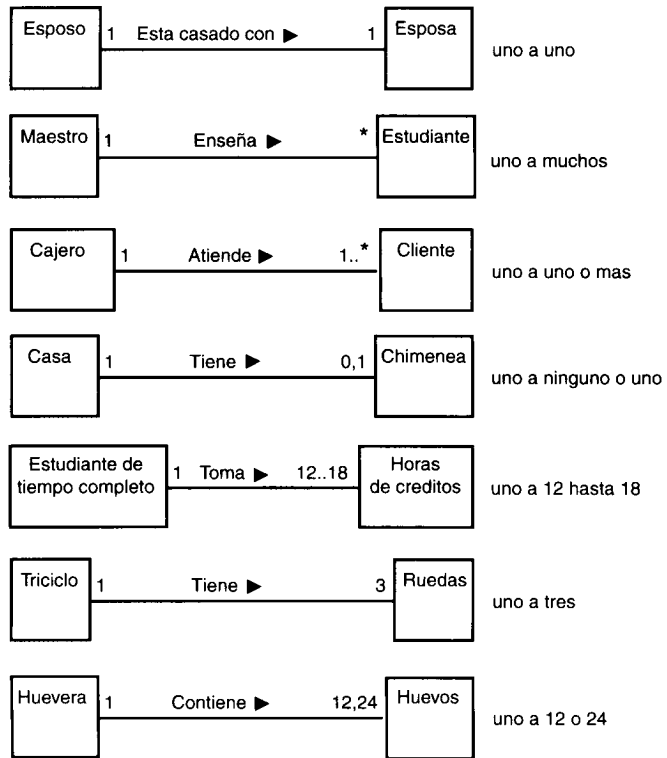
La multiplicidad de este ejemplo no es la única que existe. Hay varios tipos de multiplicidades (una multiplicidad de multiplicidades, por decirlo así). Una clase puede relacionarse con otra en un esquema de uno a uno, uno a muchos, uno a uno o más, uno a ninguno o uno, uno a un intervalo definido (por ejemplo: uno a cinco hasta diez), uno a exactamente n (como en este ejemplo), o uno a un conjunto de opciones (por ejemplo, uno a nueve o diez). El UML utiliza un asterisco (*) para representar *más* y para representar *muchos*. En un contexto O se representa por dos puntos, como en “1..*” (“uno o más”). En otro contexto, O se representa por una coma, como en “5, 10” (“5 o 10”). La figura 4.10 le muestra cómo concebir las posibles multiplicidades.



Cuando la clase A tiene una multiplicidad de uno a ninguno o uno con la clase B, la clase B se dice que es opcional para la clase A.

FIGURA 4.10

Posibles multiplicidades y cómo representarlas en el UML.



Asociaciones calificadas

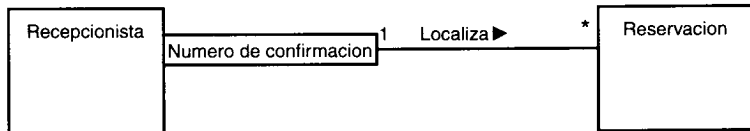
Cuando la multiplicidad de una asociación es de uno a muchos, con frecuencia se presenta un reto muy particular: la búsqueda. Cuando un objeto de una clase tiene que seleccionar un objeto particular de otro tipo para cumplir con un papel en la asociación, la primera clase deberá atenerse a un atributo en particular para localizar al objeto adecuado. Normalmente, dicho atributo es un identificador que puede ser un número de identidad. Por ejemplo, cuando usted realiza una reservación en un hotel, el hotel le asigna un número de confirmación. Si usted quiere hacer preguntas respecto a la reservación, deberá proporcionar el número de confirmación.

TÉRMINO NUEVO

En el UML la información de identidad se conoce como *calificador*. Su símbolo es un pequeño rectángulo adjunto a la clase que hará la búsqueda. La figura 4.11 muestra la representación. La idea es reducir, con eficiencia, la multiplicidad de uno a muchos a una multiplicidad de uno a uno.

FIGURA 4.11

Un calificador en una asociación resuelve el problema de la búsqueda.

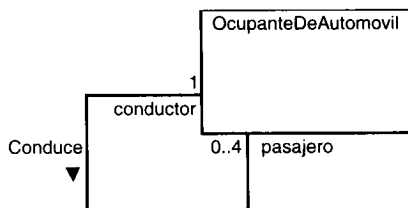


Asociaciones reflexivas

En ocasiones, una clase es una asociación consigo misma. Esto puede ocurrir cuando una clase tiene objetos que pueden jugar diversos papeles. Un **OcupanteDeAutomovil** puede ser un **Conductor** o un **Pasajero**. En el papel del conductor, el **OcupanteDeAutomovil** puede llevar ninguno o más **OcupanteDeAutomovil**, quienes jugarán el papel de pasajeros. Esto lo representará mediante el trazado de una línea de asociación a partir del rectángulo de la clase hacia el mismo rectángulo de la clase, y en la línea de asociación indicará los papeles, nombre de la asociación, dirección de la asociación y multiplicidad como ya lo hizo antes. La figura 4.12 le presenta este ejemplo.

FIGURA 4.12

En una asociación reflexiva, trazará la línea de la clase hacia sí misma y podrá incluir los papeles, nombre de la asociación y su dirección, así como su multiplicidad.



Herencia y generalización

Uno de los sellos distintivos de la orientación a objetos es que captura uno de los mayores aspectos del sentido común en cuanto a la vida diaria: si usted conoce algo de una categoría de cosas, automáticamente sabrá algunas cosas que podrá transferir a otras categorías. Si usted sabe que algo es un electrodoméstico, ya sabrá que contará con un interruptor, una marca y un número de serie. Si sabe que algo es un animal dará por hecho que come, duerme, tiene una forma de nacer, de trasladarse de un lugar a otro y algunos otros atributos (y operaciones) que podría listar si pensara en ello por algunos instantes.

TÉRMINO NUEVO

La orientación a objetos se refiere a esto como *herencia*. El UML también lo denomina *generalización*. Una clase (la clase secundaria o subclase) puede heredar los atributos y operaciones de otra (la clase principal o superclase). La clase principal (o madre) es más genérica que la secundaria (o hija).



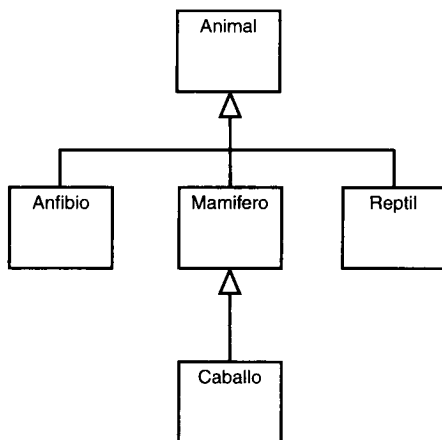
En la generalización, una clase secundaria (hija) es sustituible por una clase principal (madre). Es decir, donde quiera que se haga referencia a la clase madre, también se hace referencia a la clase hija. Sin embargo, en el caso contrario no es aplicable.

La jerarquía de la herencia no tiene que finalizar en dos niveles: una clase secundaria puede ser principal para otra clase secundaria. Un Mamífero es una clase secundaria de Animal, y Caballo es una clase secundaria de Mamífero.

En el UML representará la herencia con una línea que conecte a la clase principal con la secundaria. En la parte de la línea que se conecta con la clase principal, colocará un triángulo sin rellenar que apunte a la clase principal. Este tipo de conexión se interpreta con la frase *es un tipo de*. Un Mamífero *es un tipo de* Animal, y un Caballo *es un tipo de* Mamífero. La figura 4.13 le muestra esta particular jerarquía de la herencia, junto con otras clases. Observe la apariencia del triángulo y las líneas cuando varias clases secundarias son herencia de una clase principal. Al disponer el diagrama de este modo, trae por resultado un diagrama más ordenado en lugar de mostrar todas las líneas y triángulos, aunque el UML no le prohíbe colocarlos todos en la imagen. También vea que no colocó los atributos y operaciones heredadas en los rectángulos de las subclases, dado que ya los había representado en la superclase.

FIGURA 4.13

Una jerarquía de herencia en el reino animal.



Cuando modele la herencia, tenga la seguridad de que la clase secundaria satisfaga la relación *es un tipo de* con la clase principal. Si no se cumple tal relación, tal vez una asociación de otro tipo podría ser más adecuada.

Con frecuencia las clases secundarias agregan otras operaciones y atributos a los que han heredado. Por ejemplo: un Mamífero tiene pelo y da leche, dos atributos que no se encuentran en la clase Animal.

TERMINO NUEVO

Una clase puede no provenir de una clase principal, en cuyo caso será una *clase base* o *clase raíz*. Una clase podría no tener clases secundarias, en cuyo caso será una clase final o clase hoja. Si una clase tiene exactamente una clase principal, tendrá una *herencia simple*. Si proviene de varias clases principales, tendrá una *herencia múltiple*.

Descubrimiento de la herencia

En el proceso de plática con un cliente, un analista descubrirá la herencia de varias formas. Es posible que las clases candidatas que aparezcan incluyan tanto clases principales como clases secundarias. El analista deberá darse cuenta que los atributos y operaciones de una clase son generales y que se aplicarán a, quizá, varias clases (mismas que agregarán sus propios atributos y operaciones).

El ejemplo del baloncesto de la hora 3, “Uso de la orientación a objetos”, tiene las clases Jugador, Defensa, Delantero y Central. El Jugador tiene atributos como nombre, estatura, peso, velocidadAlCorrer y saltoVertical. Tiene operaciones como driblar(), pasar(), rebotar() y tirar(). Las clases Defensa, Delantero y Centro heredarán tales atributos y operaciones, y agregarán los suyos. La clase Defensa podría tener las operaciones correrAlFrente() y quitarBalon(). El Central podría tener retacarBalon(). De acuerdo con los comentarios del entrenador respecto a las estaturas de los jugadores, el analista tal vez quisiera colocar restricciones en las estaturas para cada posición.

Otra posibilidad es que el analista note que dos o más clases tienen ciertos atributos y operaciones en común. El modelo del baloncesto tiene un CronometroDeJuego (que controla el tiempo que resta en un periodo de juego) y un LapsoDeTiro (que controla el tiempo restante desde el instante que un equipo tomó posesión del balón, hasta que intente encestar). Si nos damos cuenta de que ambos controlan el tiempo, el analista podría formular una clase Reloj con una operación controlarTiempo() que podrían heredar tanto CronometroDeJuego como LapsoDeTiro.



Dado que LapsoDeTiro controla 24 segundos (profesional) o 35 segundos (colegial) y el CronometroDeJuego controla 12 minutos (profesional) o 20 minutos (colegial), controlarTiempo() será polimórfico.

Clases abstractas

En el modelo del baloncesto, el par de clases que mencioné —Jugador y Reloj— son útiles puesto que funcionan como clases principales para clases secundarias importantes. Las clases secundarias son importantes en el modelo dado que finalmente usted querrá

tener instancias de tales clases. Para desarrollar el modelo, necesitará instancias de Defensa, Delantero, Centro, CronometroDeJuego y LapsoDeTiro.

No obstante, Jugador y Reloj no proporcionan ninguna instancia al modelo. Un objeto de la clase Jugador no serviría a ningún propósito, así como tampoco uno de la clase Reloj.

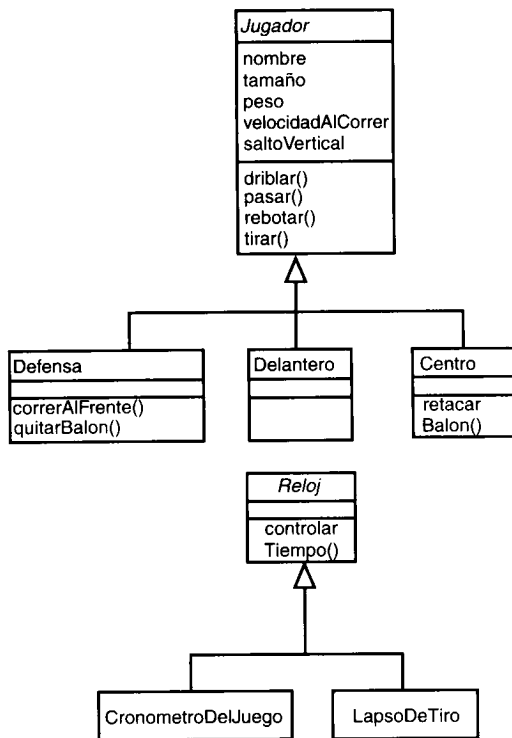
TERMINO NUEVO

Las clases como Jugador y Reloj —que no proveen objetos— se dice que son *abstractas*. Una clase abstracta se distingue por tener su nombre en cursivas.

La figura 4.14 muestra las dos clases abstractas y sus clases secundarias.

FIGURA 4.14

Dos jerarquías de herencia con clases abstractas en el modelo de baloncesto.



Dependencias

TERMINO NUEVO

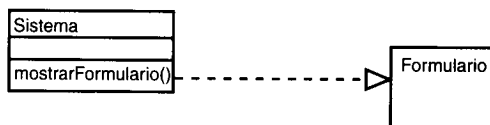
En otro tipo de relación, una clase utiliza a otra. A esto se le llama dependencia. El uso más común de una dependencia es mostrar que la firma de la operación de una clase utiliza a otra clase.

Suponga que diseñará un sistema que muestra formularios corporativos en pantalla para que los empleados los llenen. El empleado utiliza un menú para seleccionar el formulario por llenar. En su diseño, tiene una clase Sistema y una clase Formulario. Entre sus

muchas operaciones, la clase Sistema tiene mostrarFormulario(f:Form). El formulario que el sistema desplegará, dependerá, obviamente, del que elija el usuario. La notación del UML para ello es una línea discontinua con una punta de flecha en forma de triángulo sin relleno que apunta a la clase de la que depende, como muestra la figura 4.15.

FIGURA 4.15

Una flecha representada por una línea discontinua con una punta de flecha en forma de triángulo sin relleno simboliza una dependencia.



Resumen

Sin las relaciones, un modelo de clases sería poco menos que una lista de cosas que representarían un vocabulario. Las relaciones le muestran cómo se conectan los términos del vocabulario entre sí para dar una idea de la sección del mundo que se modela. La asociación es la conexión conceptual fundamental entre clases. Cada clase en una asociación juega un papel, y la multiplicidad especifica cuántos objetos de una clase se relacionan con un objeto de la clase asociada. Hay muchos tipos de multiplicidad. Una asociación se representa como una línea entre los rectángulos de clases con los papeles y multiplicidades en cada extremo. Al igual que una clase, una asociación puede contener atributos y operaciones.

Una clase puede heredar atributos y operaciones de otra clase. La clase heredada es secundaria de la clase principal que es de la que se hereda. Descubrirá la herencia cuando encuentre clases en su modelo inicial que tengan atributos y operaciones en común. Las clases abstractas sólo se proyectan como bases de herencia y no proporcionan objetos por sí mismas. La herencia se representa como una línea entre la clase principal y la secundaria, con un triángulo sin rellenar que se adjunta (y apunta a) la clase principal.

En una dependencia, una clase utiliza a otra. El uso más común de una dependencia es mostrar que una firma en la operación de una clase utiliza a otra clase. Una dependencia se proyecta como una línea discontinua que reúne a las dos clases en la dependencia, con una punta de flecha en forma de triángulo sin relleno que adjunta (y apunta a) la clase de la que se depende.

Preguntas y respuestas

P ¿En alguna ocasión se le puede poner nombre a una relación de herencia, como se hace en una asociación?

R El UML no le impide que adjudique un nombre a una relación de herencia, pero por lo general esto no es necesario.

Taller

El cuestionario y los ejercicios se han diseñado para reafirmar su conocimiento del UML en el área de las relaciones. Cada pregunta y ejercicio requiere que usted piense en la simbología del modelado que ha aprendido y la aplique a una situación. Las respuestas se encuentran en el Apéndice A, “Respuestas a los cuestionarios”.

Cuestionarios

1. ¿Cómo representaría la multiplicidad?
2. ¿Cómo descubrirá la herencia?
3. ¿Qué es una clase abstracta?
4. ¿Cuál es el efecto de un calificador?

Ejercicios

1. Tome como base el modelo del baloncesto de la Hora 3, y agregue vínculos que expresen las relaciones que ha visto en esta hora. Si conoce el juego del baloncesto, siéntase con libertad de agregar los vínculos que representen su conocimiento.
2. De acuerdo con un viejo adagio: “Un abogado que se defiende a sí mismo, tiene por cliente a un tonto.” Cree un modelo que refleje esta pieza de sabiduría.

HORA 5



Agregación, composición, interfaces y realización

Continuaremos con las relaciones entre clases y comprenderá nuevos conceptos respecto a las clases y sus diagramas.

En esta hora se tratarán los siguientes temas:

- Agregaciones
- Composiciones
- Contextos
- Interfaces y realizaciones
- Visibilidad

Ya ha visto lo concerniente a asociación, multiplicidad y herencia y está casi listo para crear diagramas de clases significativos. Conforme explore otros tipos de relaciones y detalles relacionados con las clases comprenderá las piezas finales del rompecabezas. La meta final es crear una idea estática de un sistema, con todas las conexiones entre las clases que lo conforman.

Agregaciones

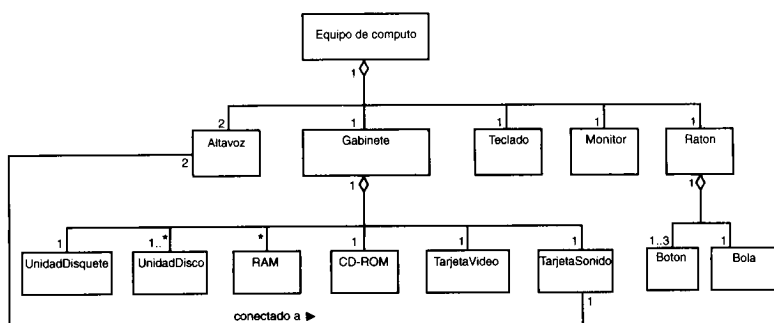
TERMINO NUEVO

En ocasiones una clase consta de otras clases. Éste es un tipo especial de relación conocida como *agregación* o *acumulación*. Los componentes y la clase que constituyen son una asociación que conforma un todo. En la hora 2, "Orientación a objetos", mencioné que su computadora es un conjunto de elementos que consta de gabinete, teclado, ratón, monitor, unidad de CD-ROM, una o varias unidades de disco duro, módem, unidad de disquete, impresora y, posiblemente, altavoces. Además de las unidades de disco, el gabinete contiene la memoria RAM, una tarjeta de vídeo y una tarjeta de sonido (tal vez algunos otros elementos).

Puede representar una agregación como una jerarquía dentro de la clase completa (por ejemplo el sistema computacional) en la parte superior, y los componentes por debajo de ella. Una línea conectará el todo con un componente mediante un rombo sin relleno que se colocará en la línea más cercana al todo. La figura 5.1 le muestra el sistema de cómputo como una agregación.

FIGURA 5.1

Una asociación por agregación se representa por una línea entre el componente y el todo con un rombo sin relleno que conforma al todo.



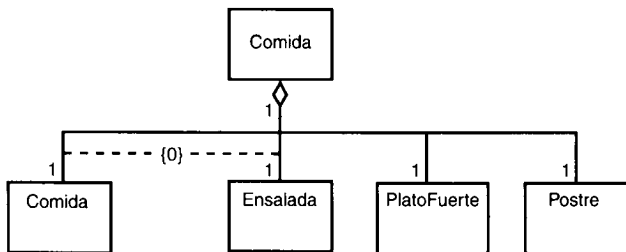
Aunque este ejemplo le muestra cada componente correspondiente a un todo, en una agregación éste no será necesariamente el caso. Por ejemplo: en un sistema casero de entretenimiento, un control remoto podría ser un componente de una televisión, aunque también podría ser un componente de una reproductora de casetes de vídeo.

Restricciones en las agregaciones

En ocasiones el conjunto de componentes posibles en una agregación se establece dentro de una relación O. En ciertos restaurantes, una comida consta de sopa o ensalada, el plato fuerte y el postre. Para modelar esto, utilizaría una restricción: la palabra O dentro de llaves con una línea discontinua que conecte las dos líneas que conforman al todo, como lo muestra la figura 5.2.

FIGURA 5.2

Puede establecer una restricción a una agregación para mostrar que un componente u otro es parte del todo.

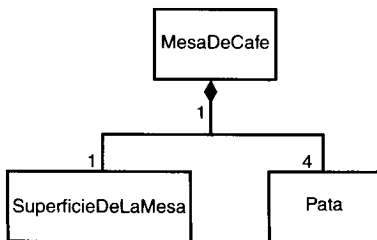


Composiciones

Una composición es un tipo muy representativo de una agregación. Cada componente dentro de una composición puede pertenecer tan sólo a un todo. Los componentes de una mesa de café (la superficie de la mesa y las patas) establecen una composición. El símbolo de una composición es el mismo que el de una agregación, excepto que el rombo está relleno (vea la figura 5.3).

FIGURA 5.3

En una composición, cada componente pertenece solamente a un todo. Un rombo relleno representa esta relación.



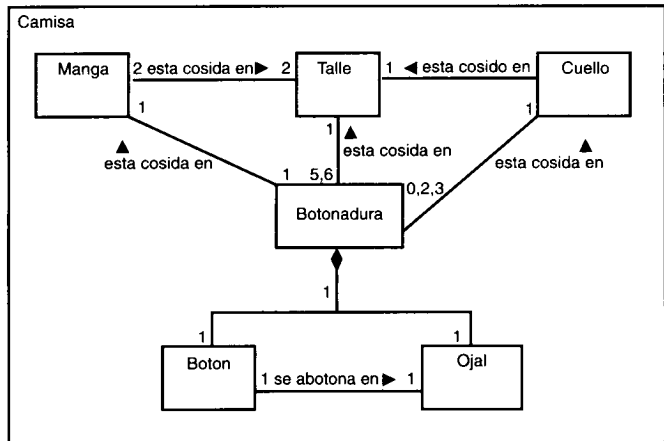
Contextos

Cuando modele un sistema podrían producirse, con frecuencia, agrupamientos de clases, como agregaciones o composiciones. En tal caso, deberá enfocar su atención en un agrupamiento o en otro, y el diagrama de contexto le proporciona la característica de modelaje que requiere para tal fin. Las composiciones figuran en gran medida dentro de los diagramas de contexto. Un diagrama de contexto es como un mapa detallado de alguna sección de un mapa de mayores dimensiones. Pueden ser necesarias varias secciones para capturar toda la información detallada.

He aquí un ejemplo: suponga que está creando un modelo de una camisa y la forma en que se podría combinar con algún atuendo y un guardarropa. Un tipo de diagrama de contexto (vea la figura 5.4) le mostrará la camisa como un gran rectángulo de clase, con un diagrama anidado en el interior, el cual le muestra cómo los componentes de la camisa están relacionados entre sí. Éste es un diagrama de contexto de composición (dado que la sola camisa reúne a cada componente se le denomina *de composición*).

FIGURA 5.4

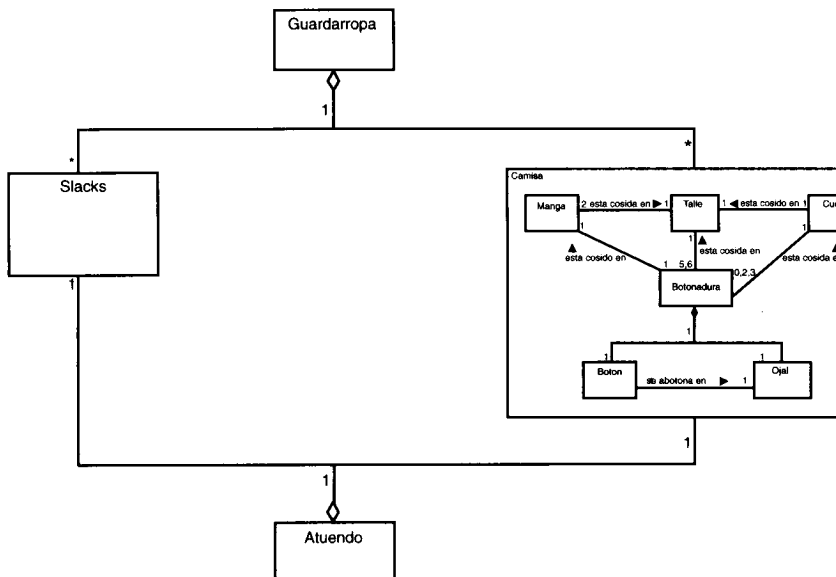
Un diagrama de contexto de composición le muestra los componentes de una clase como un diagrama anidado dentro de un enorme rectángulo de clase.



El diagrama de contexto de composición enfoca la atención en la camisa y sus componentes. Para mostrar la camisa en el contexto del guardarropa y de algún atuendo, tendrá que ampliar su ámbito. Un diagrama de contexto del sistema lo hará por usted. Podrá mostrar la forma en que la clase Camisa se conecta con las clases Guardarropa y Atuendo, como se ve en la figura 5.5.

FIGURA 5.5

Un diagrama de contexto del sistema le muestra los componentes de una clase y la forma en que la clase se relaciona con las otras que hay en el sistema.



Podrá ver de cerca alguna otra clase y presentar sus detalles en algún otro diagrama de contexto.

Interfaces y realizaciones

Una vez que haya creado varias clases, tal vez se dé cuenta que no pertenecen a una clase principal, pero en su comportamiento debe incluir algunas de las mismas operaciones con las mismas firmas de la primera clase. Podría codificar las operaciones en una de las clases y reutilizarlas en otras. Una segunda posibilidad es que desarrolle una serie de operaciones para las clases en un sistema, y reutilizarlas para las clases de otro sistema.

TERMINO NUEVO

De cualquier manera, deseará contar con algún medio para capturar el conjunto reutilizable de operaciones. La interfaz es la estructura del UML que le permite hacerlo. Una *interfaz* es un conjunto de operaciones que especifica cierto aspecto de la funcionalidad de una clase, y es un conjunto de operaciones que una clase presenta a otras.

Con un ejemplo podríamos aclarar lo anterior. El teclado que usted utiliza para comunicarse con su equipo es una interfaz reutilizable. Su operación basada en la opresión de teclas ha provenido de la máquina de escribir. La disposición de las teclas es casi la misma que en una máquina de escribir, pero el punto principal es que la operación por opresión de teclas ha sido cedida de un sistema a otro. Otras operaciones (Mayús, Bloq Mayús y Tab) también se integraron a partir de la máquina de escribir.

Por supuesto, el teclado de una computadora incluye diversas operaciones que no encontrará en una máquina de escribir: Control, Alt, RePág, AvPág y otras. Así pues, la interfaz puede establecer un subconjunto de las operaciones de una clase y no necesariamente todas ellas.

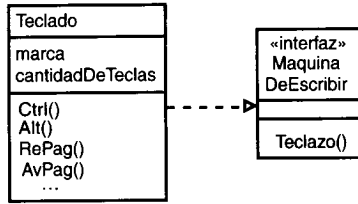
Puede modelar una interfaz del mismo modo en que modelaría una clase, con un símbolo rectangular. La diferencia será que, como un conjunto de operaciones, una interfaz no tiene atributos. Recordará que puede omitir los atributos de la representación de una clase. ¿Entonces, cómo distinguiría entre una interfaz y una clase que no muestra sus atributos? Una forma es utilizar la estructura “estereotipo” y especificar la palabra «interfaz» sobre el nombre de la interfaz en el rectángulo. Otra es colocar la letra “I” al principio del nombre de una interfaz.

TERMINO NUEVO

En cierto sentido, es como si el teclado de la computadora garantizara que esta parte de su funcionalidad “haría las veces” del teclado de una máquina de escribir. Bajo este esquema, la relación entre una clase y una interfaz se conoce como *realización*. Esta relación está modelada como una línea discontinua con una punta de flecha en forma de triángulo sin rellenar que adjunte y apunte a la interfaz. La figura 5.6 le muestra cómo se lleva a cabo esto.

FIGURA 5.6

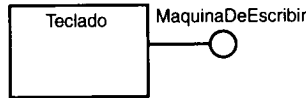
Una interfaz es un conjunto de operaciones que realiza una clase. Esta última se relaciona con una interfaz mediante la realización, misma que se indica por una línea discontinua con una punta de flecha en forma de triángulo sin rellenar que apunte a la interfaz.



Otra forma (omitida) de representar una clase y su interfaz es con un pequeño círculo que se conecte mediante una línea a la clase, como se ve en la figura 5.7.

FIGURA 5.7

La forma omitida de representar una clase que realice una interfaz.



Una clase puede realizar más de una interfaz, y una interfaz puede ser realizada por más de una clase.

Visibilidad

TERMINO NUEVO

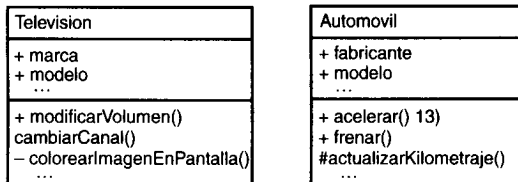
El concepto de visibilidad está muy relacionado con las interfaces y la realización. La *visibilidad* se aplica a atributos u operaciones, y establece la proporción en que otras clases podrán utilizar los atributos y operaciones de una clase dada (o en operaciones de una interfaz). Existen tres niveles de visibilidad: Nivel *público*, en el cual la funcionalidad se extiende a otras clases. En el nivel *protegido* la funcionalidad se otorga sólo a las clases que se heredan de la clase original. En el nivel *privado* sólo la clase original puede utilizar el atributo u operación. En una televisión, `modificarVolumen()` y `cambiarCanal()` son operaciones públicas, en tanto que `dibujarImagenEnPantalla()` es privada. En un automóvil, `acelerar()` y `frenar()` son operaciones públicas, pero `actualizarKilometraje()` o `actualizarMillaje()` es protegida.

La realización, como podría imaginar, implica que el nivel público se aplique a cualquier operación en una interfaz. La protección de operaciones mediante cualquiera de los otros niveles tal vez no tendría sentido, dado que una interfaz se orienta a ser realizada por diversas clases.

Para indicar el nivel público, anteceda el atributo u operación con un signo de suma (+), para revelar un nivel protegido, antecédalo con un símbolo de número (#), y para indicar el nivel privado, antecédalo con un guión (-). La figura 5.8 muestra los atributos y operaciones públicos, protegidos y privados tanto en una televisión como en un automóvil.

FIGURA 5.8

Los atributos y operaciones públicos y privados, tanto de una televisión como de un automóvil.



Ámbito

TERMINO NUEVO

El ámbito es otro concepto referente a los atributos y operaciones, y la forma en que se relacionan dentro de un sistema. Hay dos tipos de ámbitos, el de instancia y el de archivador. En el primero cada instancia cuenta con su propio valor en un atributo u operación. En un ámbito de *archivado*, sólo habrá un valor del atributo u operación en todas las instancias de la clase. Un atributo u operación con el ámbito de archivador, aparece con su nombre subrayado. Este tipo de ámbito se utiliza con frecuencia cuando un grupo específico de instancias (ningunas otras) tienen que compartir los valores exactos de un atributo privado. El ámbito de instancia es, por mucho, el tipo más común de ámbito.

Resumen

Para completar sus nociones de clases y la forma en que se conectan, es necesario comprender algunas relaciones adicionales. Una agregación establece una asociación para conformar un todo: una clase “todo” se genera de clases que la componen. Un componente en una agregación puede ser parte de más de un todo. Una composición es una conformación muy íntimamente ligada con la agregación en el sentido de que un componente de una composición puede ser parte solamente de un todo. La representación del UML de las agregaciones es similar a la representación de las composiciones. La línea de asociación que une la parte con un todo tiene un rombo. En una agregación, el rombo no está relleno, en tanto que en una composición sí lo está.

Un diagrama de contexto enfoca la atención en una clase específica dentro de un sistema. Un diagrama de contexto de composición es como un mapa detallado de un mapa mayor. Muestra un diagrama de clases anidado dentro de un gran símbolo rectangular de clase. Un diagrama de contexto de sistema muestra la forma en que el diagrama de clases compuestas se relaciona con otros objetos del sistema.

Una realización es una asociación entre una clase y una interfaz, una colección de operaciones que cierta cantidad de clases podrá utilizar. Una interfaz se representa como una clase sin atributos. Para distinguirla de una clase cuyos atributos hayan sido omitidos del diagrama, el estereotipo «interfaz» aparecerá por encima del nombre de la interfaz. Otra posibilidad es la de anteceder el nombre de la interfaz con una “I” mayúscula. La realización se representa en el UML mediante una línea discontinua con una punta de flecha en forma de triángulo sin rellenar que conecta a la clase con la interfaz. Otra forma para representar una realización es con una línea continua que conecte a una clase con un pequeño círculo, para que el círculo se interprete como la interfaz.

En términos de visibilidad, todas las operaciones en una interfaz son *públicas*, de modo que cualquier clase podrá utilizarlas. Los otros dos niveles de visibilidad son *protegido* (la funcionalidad se extiende a las clases secundarias de aquella que contiene los atributos y operaciones) y *privado* (atributos y operaciones que se pueden utilizar sólo dentro de la clase que los contiene). Un signo de suma (+) denota a la visibilidad pública, el símbolo de número (#) la protegida y el guión (-) la privada.

El ámbito es otro aspecto de los atributos y operaciones. En un ámbito de instancia, cada objeto de una clase cuenta con su propio valor en un atributo u operación. En un ámbito de archivador, sólo hay un valor para un atributo u operación en particular a través de un conjunto de objetos de una clase. Los objetos que no estén en este conjunto no podrán acceder al valor contenido en el ámbito de archivador.

Preguntas y respuestas

- P** ¿Se considera transitiva a la agregación? Es decir, si la clase 3 es un componente de la clase 2, y la clase 2 es un componente de la clase 1, ¿la clase 3 será un componente de la clase 1?
- R** Así es, la agregación es transitiva. En nuestro ejemplo, los botones y la bola del ratón son parte del ratón, a la vez que son parte de la computadora.
- P** ¿La palabra “interfaz” implica “interfaz de usuario” o GUI?
- R** No. Es algo más genérico. Una interfaz es tan sólo un conjunto de operaciones que una clase presenta a las demás clases. De hecho, una de estas operaciones podría ser (aunque no necesariamente) la del usuario.

Taller

El cuestionario y los ejercicios verificarán y fortalecerán su conocimiento respecto al tema de las agregaciones, composiciones, contextos e interfaces. Las respuestas las podrá ver en el Apéndice A, “Respuestas a los cuestionarios”.

Cuestionario

1. ¿Cuál es la diferencia entre una agregación y una composición?
2. ¿Qué es la realización?
3. Mencione los tres niveles de visibilidad y describa lo que significa cada uno de ellos.

Ejercicios

1. Cree un diagrama de contexto de composición de una revista. Tome en cuenta la tabla de contenido, la editorial, los artículos y las columnas. Luego, cree un diagrama de contexto del sistema que muestre a la revista junto con el suscriptor y el comprador en el puesto de revistas.

2. En la actualidad, el tipo más popular de GUI es la interfaz WIMP (ventanas, iconos, menús y puntero, por sus siglas en inglés). Dibuje un diagrama de clases de la interfaz WIMP, y haga uso de todo el conocimiento adecuado del UML que ha adquirido hasta ahora. Además de las clases indicadas en las siglas, incluya los elementos relacionados como las barras de desplazamiento y el cursor, así como cualquiera de las otras clases necesarias.