Conceptos de Programación Orientada a Objetos

¿Qué es un objeto en el mundo real?

Un objeto es un elemento real o abstracto, que tiene un estado, un comportamiento y una identidad. Un objeto es, pues, una mesa, una silla, un alumno, una clase, etc., pues son elementos reales que se pueden comprender y están bien definidos. Un objeto también es un concepto abstracto como un elemento denominado «Ordenador» que es capaz de recibir un conjunto de números y los ordena ascendente o descendentemente.

En el mundo real, a menudo se encuentran muchos objetos individuales todos del mismo tipo. Puede haber miles de otras bicicletas en existencia, todos de la misma marca y modelo. Cada bicicleta se construyó desde el mismo conjunto de planos, por lo que contiene los mismos componentes. En términos orientados-objeto, se dice que la bicicleta es un ejemplo de la clase de objetos conocidos como las bicicletas. Una clase es el modelo de objetos individuales que se crean.

Las características que definen un objeto son tres: su estado, su comportamiento y su identidad.

a) Estado

Viene determinado para el conjunto de propiedades o atributos que tiene el objeto (que es su estructura estática), junto con los valores que pueden asumir cada uno de esos atributos (su estructura dinámica).

Véanse ambos ejemplos con datos concretos:

Objeto alumno "Luis Pérez Gómez".

• Estructura estática (su contenido no importa).

— Nombre y apellidos:
— Edad:
— Sexo:
— Dirección:
— Curso:
— Asignaturas:

• Estructura dinámica (en 1996).

— Nombre y apellidos: Luis Pérez Gómez
— Edad: 17
— Sexo: varón
— Dirección: calle del Pez, 24
— Curso: 3º BUP
— Asignaturas: Matemáticas, Física, Química, Inglés, Historia

• Estructura dinámica (en 1997).

— Nombre y apellidos: Luis Pérez Gómez
— Edad: 18
— Sexo: varón
— Dirección: calle Atocha, 18
— Curso: COU
— Asignaturas: Curso Completo

NOTA: El alumno Luis Pérez Gómez ha cambiado, entre 1996 y 1997, de edad, dirección, curso y asignaturas, porque es un objeto activo.

Objeto "Ordenador".

En el caso del objeto abstracto «Ordenador», sus atributos son las celdas donde almacena los números y una variable que indica si los números están ordenados ascendentemente, descendentemente o están sin ordenar (es su estructura estática). Mientras que el contenido de las celdas y de la variable, en un momento dado, es su estructura dinámica.

• Estructura estática

— Celdas de memoria: c1, c2, …, cn.
— Variable de estado: ve.

• Estructura dinámica

Si se le pide al objeto que ordene ascendentemente los números: 4, -5, 3, 10, -1, 2 su estructura dinámica va variando de la siguiene forma:

— Estado inicial: Los números se leen y se almacenan en las celdas.

c1	**c2**	**c3**	**c4**	**c5**	**c6**
4	-5	3	10	-1	2

ve=0

— Estado final ordenado: Los números han sido ordenados ascendentemente.

c1	**c2**	**c3**	**c4**	**c5**	**c6**
-5	-1	2	3	4	10

ve=1

NOTA: Se ha supuesto que la variable de estado (ve) del objeto «Ordenador» toma los valores 0, 1 ó 2, según que la información esté sin ordenar, ordenada ascendentemente u ordenada descendentemente.

El hecho de que un objeto tenga un estado implica que ocupa un espacio, sea en el mundo real o en la memoria de la computadora, exista durante un tiempo, cambie el valor de sus atributos, y sea creado y destruido.

Así, el objeto alumno «Luis Pérez Gómez» descrito anteriormente se crea cuando se le matricula en el colegio; existe como alumno durante toda su vida escolar; cambia sus datos al cambiar de curso o por otras circunstancias, y deja de ser alumno cuando sale del colegio.

b) **Comportamiento**

El comportamiento de un objeto viene determinado por la forma de actuar al recibir un mensaje para que realice una acción.
Un «mensaje» es una orden que se manda a un objeto para que realice una operación con un propósito específico.

Así, en el caso del objeto «Ordenador», se le podrían mandar los siguientes mensajes:

— «leer_números», para que leyera números enteros desde el teclado y los almacenara en las celdas de memoria: c1,c2,…,cn.

— «ordenar_ascendentemente», para que el contenido almacenado en c1,c2,…,cn, quedase ordenado ascendentemente.

— «ordenar_descendentemente», para obtener un contenido con orden descendente.

— «indicar_estado», para que especificara mediante la variable de estado ve si los números están sin ordenar (ve=0) ordenados ascendentemente (ve=1), u ordenados descendentemente (ve=2).

— «mostrar_números», para que visualizase por la pantalla de la computadora los números almacenados en las celdas de memoria c1,c2,…,cn.

Un mensaje viene definido por tres componentes:

— El objeto receptor del mensaje, es decir, al objeto que se envía el mensaje.

— El selector del mensaje, que es la acción que se quiere que realice el objeto.

— Los argumentos, que son los valores que se transmiten con el selector y que, en muchos casos, pueden ser nulos.

Por ejemplo, Ordenador.Ordenar_ascendentemente(), es un mensaje que se envía al objeto receptor Ordenador, para que ordene ascendentemente los números (selector), y no se transmite ningún argumento. Mientras que los mensajes:

```
Luis_Perez_Gomez.Asignaturas(curso)
Luis_Perez_Gomez.Asignaturas(todas)
```

Señalan: el primero, que se desea conocer las asignaturas del curso en que Luis está matriculado, y el segundo, que se desea conocer todas las asignaturas en que está matriculado. En estos dos ejemplos, el argumento del selector (curso o todas) permite seleccionar unas asignaturas u otras.

c) **Identidad**

Se entiende por identidad de un objeto la propiedad característica que tiene ese objeto que le distingue de todos los demás. Realmente, es difícil encontrar un dato específico que permanentemente identifique al objeto.

Así, si se hace referencia a Luis Pérez Gómez, ¿se puede seleccionar algún dato que lo identifique durante toda su vida?:

— ¿Su nombre? - Lo puede cambiar en el Registro Civil.
— ¿Su edad? - Cambia continuamente.
— ¿Su DNI?, ¿Su sexo? - También pueden cambiar.

Pero es indudable que pasará el tiempo y Luis Pérez no perderá su identidad y se podrá diferenciar de todas las demás personas.

Si en la vida real es difícil identificar permanentemente un objeto que evoluciona, mucho más lo es en el mundo de la computadora. Normalmente se utiliza, como en la vida real, un nombre definido por el programador que lo identifique.

Así, en el caso del objeto anterior que lee datos y los ordena ascendente o descendentemente se le ha identificado con la palabra «Ordenador».

¿Por qué es importante estudiar los objetos del mundo real?

Los objetos concretos y virtuales, están a nuestro alrededor, ellos conforman nuestro mundo. 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 como se deben mostrar éstos en las representaciones de software.

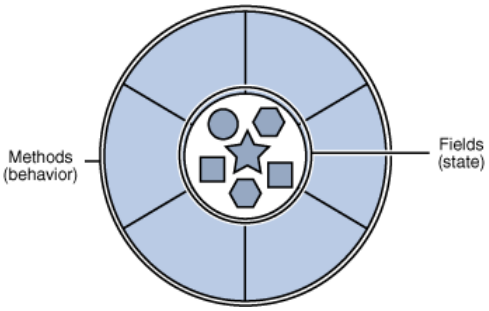
Los objetos son clave para entender orientado a objetos de tecnología. Mira a tu alrededor en este momento y usted encontrará muchos ejemplos de objetos del mundo real: su perro, su escritorio, su televisión, su bicicleta.

Objetos del mundo real tienen dos características: Todos ellos tienen estado y comportamiento. Los perros tienen un estado (nombre, color, raza, hambre) y comportamiento (ladridos, ir a buscar, meneando la cola). Las bicicletas también tienen un estado (equipo actual, la cadencia de pedaleo actual, velocidad actual) y comportamiento (el cambio de marchas, el cambio de cadencia de pedaleo, la aplicación de los frenos). Identificar el estado y el comportamiento de los objetos del mundo real es una gran manera de empezar a pensar en términos de programación orientada a objetos.

Gire a la derecha un momento a observar los objetos del mundo real que están en su área inmediata. Para cada objeto que ve, hágase dos preguntas: "¿Qué estados posibles este objeto puede ser?" y "¿Qué posible comportamiento se puede realizar este objeto?". Asegúrese de anotar sus observaciones. Mientras lo hace, te darás cuenta de que los objetos del mundo real varían en complejidad, su lámpara de escritorio puede tener solo dos estados posibles (dentro y fuera) y dos posibles comportamientos (encender, apagar), pero su radio de escritorio puede tener otros estados (encendido, apagado, volumen actual, la estación actual) y comportamiento (encender, apagar, aumentar el volumen, disminuir el volumen, buscar, explorar, y canción). También puede observar que algunos objetos, a su vez, también contienen otros objetos. Estas observaciones del mundo real se traducen en el mundo de la programación orientada a objetos.

Los objetos de software

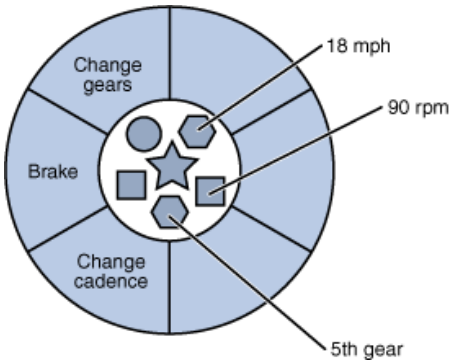
Un objeto es un paquete de software del estado y el comportamiento relacionado. Los objetos de software a menudo se utilizan para modelar los objetos del mundo real que se encuentra en la vida cotidiana. Más adelante se explica cómo el estado y el comportamiento están representados dentro de un objeto, introduce el concepto de encapsulación de datos, y explica los beneficios de diseñar el software de esta manera.



Un objeto de software.

Los objetos de software son conceptualmente similares a los objetos reales del mundo: también ellos constan de estado y el comportamiento relacionado. Un objeto almacena su estado en los campos (variables en algunos lenguajes de programación) y expone su comportamiento a través de métodos (funciones en algunos lenguajes de programación). Los Métodos operaran en el estado interno de un objeto y servir como mecanismo principal para la

comunicación de objeto a objeto. Ocultar el estado interno y requerir toda la interacción se realiza a través de los métodos de un objeto que se conoce como encapsulación de datos - un principio fundamental de la programación orientada a objetos.
Un objeto es un paquete de software del estado y el comportamiento relacionado. Los objetos de software a menudo se utilizan para modelar los objetos del mundo real que se encuentra en la vida cotidiana.
Considere una bicicleta, por ejemplo:



una bicicleta modelada como un objeto de software.

Al atribuir el estado (velocidad actual, cadencia actual del pedal, y equipo actual) y proporcionar métodos para cambiar ese estado, el objeto permanece en el control de cómo el mundo exterior es autorizado para utilizarla. Por ejemplo, si la bicicleta sólo tiene 6 velocidades, un método para cambiar de marcha puede rechazar cualquier valor que es menor que 1 o mayor que 6.

Construir códigos en objetos individuales de software proporciona una serie de beneficios, incluyendo:

1. Modularidad: El código fuente de un objeto puede ser escrito y mantenido de forma independiente del código fuente de otros objetos. Una vez creado, un objeto puede ser fácilmente pasado a otro sistema.
2. Encapsulamiento: Al interactuar sólo con los métodos de un objeto, los detalles de su implementación interna permanecen ocultos del mundo exterior.
3. Reutilizar código: Si un objeto ya existe (tal vez escrito por otro desarrollador de software), puede utilizar ese objeto en su programa. Esto permite a los especialistas implementar/aplicar/probar hacer depuraciones de niveles complejos, desarrollar objetos de tareas específicas, en los cuales se puedan confiar para ejecutar en su propio código.
4. Portabilidad y depuración con facilidad: Si un objeto particular, resulta ser un problema, simplemente tendrá que sacarlo de su aplicación y conectar un objeto diferente como su reemplazo. Esto es análogo a la solución de problemas mecánicos en el mundo real. Si se rompe un perno, que sustituya lo roto, no toda la máquina.

Algunos conceptos

La orientación a objetos se refiere a algo más que tan solo 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 envió de mensajes, las asociaciones y la agregación. Examinemos cada uno de estos Conceptos.

Abstracción

La abstracción se refiere a quitar las propiedades y acciones de un objeto para dejar solo 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.
Supongamos una lavadora a la cual en la primera fase le definimos los siguientes atributos: marca, modelo, número de serie y capacidad.
En una 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 generara finalmente una 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 valdría la pena.
En este programa no necesitaré 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 incluiré o desecharé, será una abstracción de una lavadora.

Modularidad

En programación modular, y más específicamente en programación orientada a objetos, se denomina Modularidad a la propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes.
Estos módulos que se puedan compilar por separado, pero que tienen conexiones con otros módulos.
Al igual que la encapsulación, los lenguajes soportan la Modularidad de diversas formas.
Según Bertrand Meyer "El acto de particionar un programa en componentes individuales para reducir su complejidad en algún grado. . . . A pesar de particionar un programa es útil por esta razón, una justificación más poderosa para particionar un programa es que crea una serie de límites bien definidos y documentados en el programa. Estos límites, o interfaces, son muy valiosos en la comprensión del programa"[1]
Por su parte Barbara Liskov establece que "popularización" consiste en dividir un programa en módulos que pueden ser compilados de forma separada, pero que tienen conexiones con otros módulos"[2]

Herencia

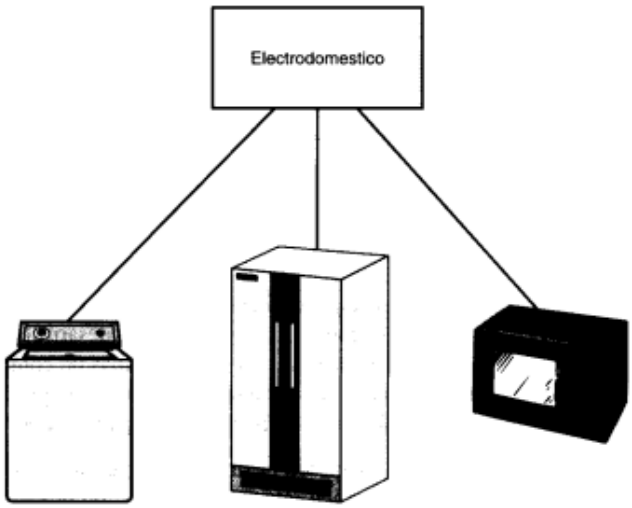
Como ya se menciono anteriormente, una clase es una categoría de objetos (y en el mundo del software, una planilla sirve para crear otros objetos). Un objeto es una instancia de una clase. Esta idea tiene una consecuencia importante como instancia de una clase, un objeto tiene todas las características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

Un objeto no solo hereda de una clase, sine 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; Electrodomésticos. Un electrodoméstico cuenta con los atributos de interruptor y cable eléctrico, y las operaciones de encendido y apagado. Cada una de las clases Electrodoméstico heredara los mismos atributos; por ello, si sabe que algo es un electrodoméstico, de inmediato sabré que cuenta con los atributos y acciones de la clase Electrodoméstico.

Otra forma de explicarle es que la lavadero, refrigerador, home de microondas y cosas por el estilo son subclases de la clase Electrodoméstico. Podemos decir que la clase Electrodoméstico es una superclase de todas las demás. La figura 2.3 le muestra la relación de superclase y subclase,

FIGURA 2.3

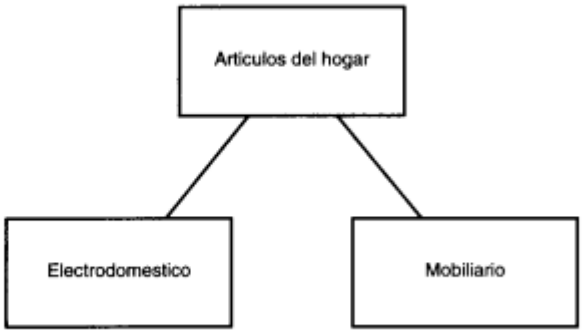
Los electrodomésticos heredan los atributos y acciones de la clase Electrodomestico. Cada electrodoméstico es una subclase de la clase Electro-domestico. La clase Electrodomestico es una superclase de cada subclase.



La herencia no tiene por qué terminar aquí. Por ejemplo, Electrodoméstico es una subclase de Artículos del hogar, como le muestra la figura 2.4. Otra de las subclases de Artículos 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

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, realizara una operación diferente. En la orientación a objetos, cada clase "sabe" como 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.

Por ejemplo, podemos crear dos clases distintas: Pez y Ave que heredan de la superclase Animal. La clase Animal tiene el método abstracto mover que se implementa de forma distinta en cada una de las subclases (peces y aves se mueven de forma distinta).

El concepto de polimorfismo se puede aplicar tanto a funciones como a tipos de datos. Así nacen los conceptos de funciones polimórficas y tipos polimórficos. Las primeras son aquellas funciones que pueden evaluarse o ser aplicadas a diferentes tipos de datos de forma indistinta; los tipos polimórficos, por su parte, son aquellos tipos de datos que contienen al menos un elemento cuyo tipo no está especificado.

Encapsulamiento

Al hablar de los objetos, es importante comentar el concepto de encapsulación. Se dice que un objeto está encapsulado (literalmente, está dentro de una cápsula) cuando está protegido del acceso indiscriminado de cualquier persona.

Así, cuando se tiene en las manos un televisor, se ve que se puede encender o apagar, poner la emisora que más guste, aumentar o disminuir el volumen, pero no se puede mover físicamente el dial (esto se realiza con un mando), ni tocar el indicador de encendido, ni acceder a otras cosas que están dentro del televisor. Esto último lo debe tocar un profesional. Por ello, se dice que:

— El dial, el indicador de encendido y la circuitería están encapsuladas.

— Los mandos de sintonía, volumen, encendido/apagado, etc., no están encapsulados y son de acceso normal para todos los usuarios del transistor.

Resumiendo, la encapsulación es el proceso que aplica el diseñador de un objeto para ocultar aquellos detalles del objeto que no son específicamente necesarios para su uso. A la encapsulación se le denomina también «ocultamiento de información».

Así, en el ejemplo del objeto «Ordenador», parece claro que deberían estar encapsuladas la celdas de memoria c1,c2,…,cn; y la variable de estado ve, ya que el usuario que maneja este objeto no necesita acceder directamente a estos datos. Mientras que no deberían estar encapsuladas las operaciones que realiza: «leer_números», «ordenar_ascendentemente», …, «mostrar_números».

Por lo tanto, un objeto, que es una unidad indivisible, tiene una parte interna que no es accesible al usuario y una parte externa que sí que es accesible. A esta parte accesible se le llama la interfaz (o protocolo) del objeto y lo constituye el conjunto de servicios que proporciona el objeto. Es realmente mediante esta interfaz como el usuario maneja el objeto.

La interfaz del televisor son el interruptor de encendido/apagado, los mandos de sintonía, volumen,…; todos ellos, elementos que permiten su manejo.

¿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 significara que tal vez no será necesario modificar los demás objetos

En el mundo real, también vera 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 reparara o lo reemplazaré; pero es muy probable que no tenga que reparar o reemplazar la CPU al mismo tiempo que el monitor.

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 si 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.

En el mundo real, también vera 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 reparara o lo reemplazaré; pero es muy probable que no tenga que reparar o reemplazar la CPU al mismo tiempo que el monitor.

Un objeto es, entonces, un elemento autónomo de información, que tiene una estructura definida por sus atributos o propiedades y que proporciona unos servicios que definen su comportamiento. Su estructura está encapsulada (está oculta al acceso del usuario) y sus servicios no. Estos servicios, que definen la interfaz del objeto, son lo que solicita el usuario para que el objeto realice una acción específica.

Envió de mensajes

Mencioné que en un sistema los objetos trabajan en conjunto. Esto se logra mediante el envió de mensajes entre ellos. Un objeto envía a otro un mensaje para realizar una operación, y el objeto receptor ejecutara 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 . 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 si 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 via), esto es, usted enciende la televisión, como se ve en la tigura 2.8. No obstante, ella no le devolveré 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.

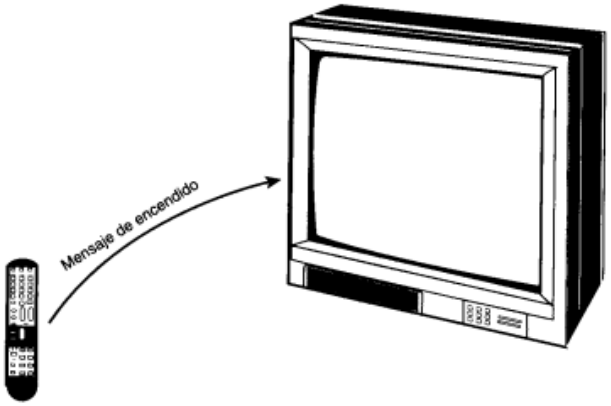
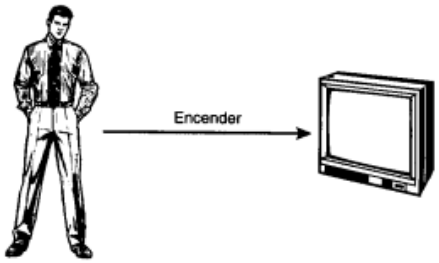


FIGURA 2.8

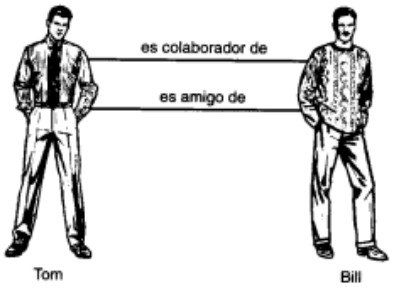
Con frecuencia los objetos se relacionan entre sí de alguna forma. Cuando usted enciende su televisión, tendrá una asociación en una sola dirección con ella.



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

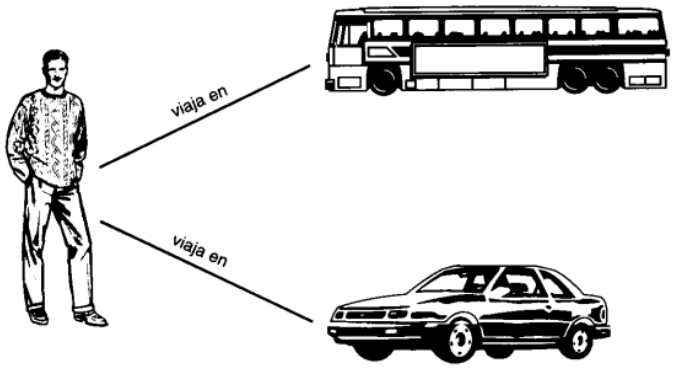
En ocasiones, los objetos pueden asociarse en más de una forma.



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 iigura 2.10).

FIGURA 2.10

Una clase puede asociarse con más de una clase distinta.

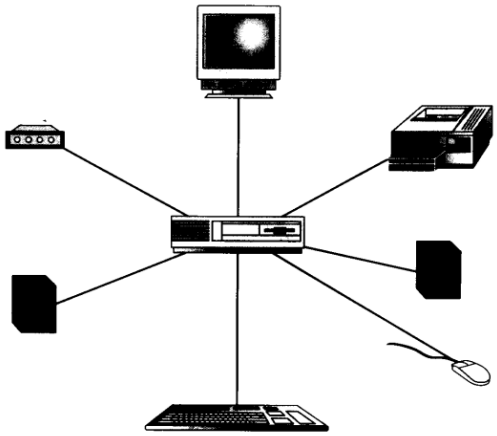


La multiplicidad (o diversificación) es un importante aspecto de las asociaciones entre objetos. Indica la cantidad de objetos de una clase que se relacionan con otro objeto en particular de la clase asociada. Por ejemplo, en un curso escolar, el curso se imparte por un solo instructor, en consecuencia, el curso y el instructor estén 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.
Podré 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

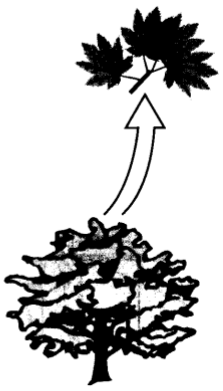
Vea su computadora: cuenta con un gabinete, un teclado, un ratón, un monitor, una unidad de CD-ROM, uno o varios discos duros. Un modem, una unidad de disquete, una impresora y, posiblemente, bocinas. Dentro del gabinete, junto con las citadas unidades de disco, tiene una CPU, una tarjeta de video, una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

FIGURA 2.11
Una computadora es un ejemplo de agregación: un objeto que se conforma de una combinación de diversos tipos de objetos.



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 2.12
En una composición, un componente puede morir antes que el objeto compuesto. Si destruye al objeto compuesto, destruirá también a los componentes.



Un tipo de agregación trae consigo una estrecha relación entre un objeto agregado y sus objetos componentes. A esta se le conoce como composición.
El punto central de la composición es que el componente se considera como tal solo como parte del objeto compuesto. Por ejemplo una camisa esté 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 morirían (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.

¿Qué es una clase?

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 mas). También realizamos las siguientes tareas: comer, dormir, leer, escribir, hablar, trabajar, etcetera. Si tuviéramos que crear un sistema que manejara información acerca de las personas (como una nomina 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.

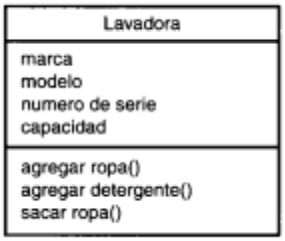
Piense en las cosas que le rodean (una idea demasiado amplia, pero piénselo de cualquier forma). Es probable que muchas de esas cosas tengan atributos (propiedades) y que realicen determinadas acciones. Podríamos imaginar cada una de esas acciones como un conjunto de tareas.

También se encontrara 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: y agregar ropa", "'agregar detergente", "activarse" y "secar 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 hablarles 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

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 esta subrayado. El nombre de la instancia específica se encuentra a la izquierda de los dos puntos (z), y el nombre de la clase a la derecha.

FIGURA 1.2

El símbolo UML del objeto.

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), tendré un mecanismo para fabricar nuevas instancias a partir de su clase; es decir, podré crear nuevos objetos (vea la figura 2. 1)

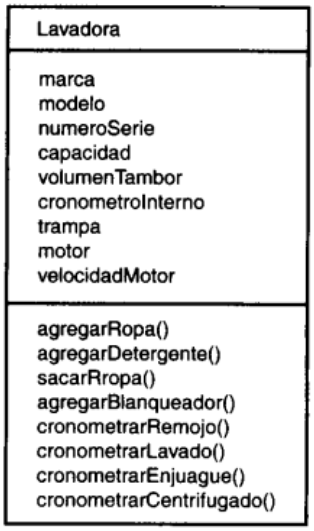
FIGURA 2.1

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

Es importante que recuerde que el propósito de la orientación a objetos es desarrollar software que recicle 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).

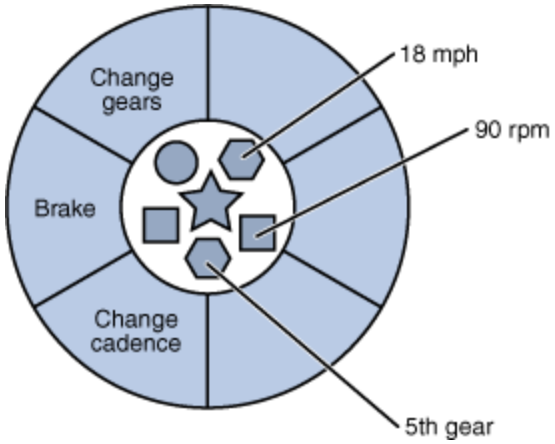
FIGURA 2.2

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



Desarrollo de ejemplo 1

Considere una bicicleta, por ejemplo:



Una bicicleta modelada como un objeto de software.

Al atribuir el estado (velocidad actual, cadencia actual del pedal, y equipo actual) y proporcionar métodos para cambiar ese estado, el objeto permanece en el control de cómo el mundo exterior es autorizado para utilizarla. Por ejemplo, si la bicicleta sólo tiene 6 velocidades, un método para cambiar de marcha puede rechazar cualquier valor que es menor que 1 o mayor que 6.

A continuación clase Bycycle es una posible implementación de una bicicleta

```
/*
 * Copyright (c) 1995, 2008, Oracle and/or its affiliates. All rights reserved.
 *

 */

class Bicycle {

    int cadence = 0;
    int speed = 0;
    int gear = 1;

    void changeCadence(int newValue) {
        cadence = newValue;
    }

    void changeGear(int newValue) {
        gear = newValue;
    }

    void speedUp(int increment) {
        speed = speed + increment;
    }

    void applyBrakes(int decrement) {
        speed = speed - decrement;
    }

    void printStates() {
        System.out.println("cadence:"+cadence+" speed:"+speed+" gear:"+gear);
    }
}
```

El diseño de esta clase se basa en la discusión previa de los objetos de la bicicleta. Los campos de `cadencia`, `velocidad`, y `artes` representan del objeto el Estado y los métodos (`changeCadence` , `changeGear` , `aceleración` , etc) definen su interacción con el mundo exterior.

Usted puede haber notado que las bicicletas de clase no contiene el método main. Eso es porque no es una aplicación completa, es sólo el modelo para bicicletas que pueden ser utilizados en una aplicación. La responsabilidad de crear y utilizar nuevas bicicletas objetos pertenece a otra clase en la aplicación.

Aquí está un `BicycleDemo` clase que crea dos independientes `de bicicletas` objetos e invoca a sus métodos:

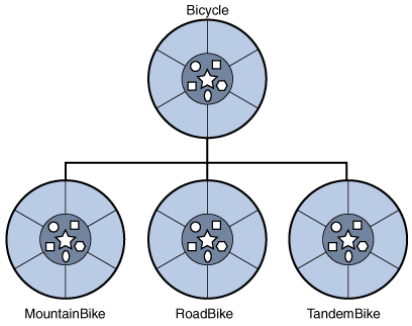
```
/*
 * Copyright (c) 1995, 2008, Oracle and/or its affiliates. All rights reserved.
 *
 */

class BicycleDemo {
    public static void main(String[] args) {

        // Create two different Bicycle objects
        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle();

        // Invoke methods on those objects
        bike1.changeCadence(50);
        bike1.speedUp(10);
        bike1.changeGear(2);
        bike1.printStates();

        bike2.changeCadence(50);
        bike2.speedUp(10);
        bike2.changeGear(2);
        bike2.changeCadence(40);
        bike2.speedUp(10);
        bike2.changeGear(3);
        bike2.printStates();
    }
}
```

Ejemplo de la herencia

Diferentes tipos de objetos a menudo tienen una cierta cantidad en común entre sí. Las bicicletas de montaña, bicicletas de carretera y bicicletas tándem, por ejemplo, todos comparten las características de las bicicletas (velocidad actual, cadencia de pedaleo actual, equipo actual). Sin embargo, cada uno también define características adicionales que las hacen diferentes: bicicletas tándem tienen dos asientos y dos juegos de manillares, las bicicletas de carretera tienen manillar gota; algunas bicicletas de montaña tienen un anillo de las cadenas adicionales, dándoles una relación de transmisión más baja.

Programación orientada a objetos permite que las clases hereden de uso general del estado y el comportamiento de otras clases. En este ejemplo, la bicicleta se convierte en la superclase de BTT , RoadBike y TandemBike . En el lenguaje de programación Java, cada clase se le permite tener una superclase directa, y cada superclase tiene el potencial para un número ilimitado de las subclases :

Una jerarquía de clases de bicicleta.



La sintaxis para crear una subclase es simple. Al comienzo de su declaración de la clase, utilice la extiende palabra clave, seguido por el nombre de la clase que hereda de:

```
extends Bicycle {

     // new fields and methods defining a mountain bike would go here

}
```

Esto da BTT todos los mismos campos y métodos de bicicletas , pero permite su código para centrarse exclusivamente en las características que lo hacen único. Esto hace que el código para que sus clases sean fáciles de leer. Sin embargo, debe tener cuidado de documentar apropiadamente el estado y el comportamiento que cada superclase define, ya que el código no aparecerá en el archivo de código fuente de cada subclase

¿Qué es una interfaz?

Como ya has aprendido, los objetos definen su interacción con el mundo exterior a través de los métodos que se exponen. Métodos de ser objeto de interfaz con el mundo exterior, los botones en el frente de su televisor, por ejemplo, son la interfaz entre el usuario y el cableado eléctrico en el otro lado de su carcasa de plástico. Al presionar el botón "POWER" para encender el televisor encendido y apagado.

En su forma más común, una interfaz es un conjunto de métodos relacionados con cuerpos vacíos. El comportamiento de una bicicleta, si se especifica como una interfaz, podría aparecer como sigue:

```
interfaz de bicicletas {changeCadence void (newValue int); / /
revoluciones de la rueda por changeGear vacío minuto (newValue int);
aceleración nula (incremento int); applyBrakes void (decremento int);}
```

Para implementar esta interfaz, el nombre de la clase iba a cambiar (para una marca en particular de la bicicleta, por ejemplo, como ACMEBicycle), y tendrá que utilizar los instrumentos clave en la declaración de clase:

clase ACMEBicycle implementa bicicletas {/ / resto de esta clase en ejecución como antes}

Implementación de una interfaz permite a una clase para ser más formal sobre el comportamiento que se compromete a proporcionar. Interfaces de formalizar un contrato entre la clase y el mundo exterior, y este contrato se cumpla en tiempo de generación por el compilador. Si sus demandas de clase para implementar

una interfaz, todos los métodos definidos por esa interfaz deben aparecer en su código fuente antes de la clase se compilará con éxito.

¿Qué es un paquete?

Un paquete es un espacio de nombres que organiza un conjunto de clases e interfaces relacionadas. Conceptualmente se puede pensar en paquetes como algo similar a las carpetas diferentes en su equipo. Usted puede guardar las páginas HTML en una carpeta, las imágenes en otro, y secuencias de comandos o aplicaciones en otro. Dado que el software escrito en el lenguaje de programación Java puede estar compuesto por cientos o miles de clases individuales, tiene sentido mantener las cosas organizadas mediante la colocación de las clases e interfaces relacionadas en paquetes.
La plataforma Java proporciona una enorme biblioteca de clases (un conjunto de paquetes) adecuado para su uso en sus propias aplicaciones. Esta biblioteca es conocida como la "Interfaz de programación de aplicación", o "API" para abreviar. Sus paquetes representan las tareas más comúnmente asociados con-programación de propósito general. Por ejemplo, una cadena de objeto contiene el estado y el comportamiento de las cadenas de caracteres, un archivo objeto permite a un programador para crear, eliminar, inspeccionar, comparar o modificar un archivo en el sistema de ficheros, un zócalo de objetos permite la creación y el uso de sockets de red ; varios objetos GUI botones de control y casillas de verificación y cualquier otra cosa relacionada con las interfaces gráficas de usuario. Hay literalmente miles de clases para elegir. Esto le permite, el programador, para centrarse en el diseño de su aplicación en particular, en lugar de la infraestructura necesaria para hacerlo funcionar.

La plataforma Java API Especificación contiene la lista completa de todos los paquetes, interfaces, clases, campos y métodos proporcionados por la plataforma Java 6, Standard Edition. Cargue la página en su navegador y favoritos. Como programador, que se convertirá en su más importante pieza de la documentación de referencia.

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?

Preguntas y Ejercicios: Conceptos de Programación Orientada a Objetos.

Preguntas.

Los objetos del mundo real contienen ___ y ___.

El estado de un objeto de software se almacena en ___.

El comportamiento de un objeto de software se expone a través de ___.

Ocultación de datos interna del mundo exterior, y el acceso sólo a través de métodos expuestos públicamente se conoce como ___ datos.

Un modelo de objeto de software se llama ___.

El comportamiento común puede ser definido en un ___ y heredadas en un ___ ___ utilizando la palabra clave.

Una colección de métodos sin aplicación se llama ___.

Un espacio de nombres que organiza las clases y las interfaces de funcionalidad se llama ___.

El término API significa ___?

Ejercicios

1. Crear nuevas clases para cada objeto del mundo real que observó al comienzo de este camino. Referencia a la clase de bicicletas si se olvida la sintaxis necesaria.
2. Para cada nueva clase que ha creado anteriormente, crear una interfaz que define su comportamiento, a continuación, requieren su clase para ponerlo en práctica. Omitir uno o dos métodos y tratar de compilar. ¿Cómo es el error?

Respuestas a las preguntas

1. Objetos del mundo real contienen **el estado** y **comportamiento**.
2. software objeto del estado A se almacena en **los campos**.
3. software objeto de comportamiento de A se expone a través de **métodos**.
4. Ocultación de datos interna del mundo exterior, y el acceso sólo a través de los métodos expuestos públicamente se conoce como datos de **encapsulación**.
5. Un modelo de objeto de software se llama **clase**.
6. comportamiento común puede ser definido en una **superclase** y heredado en una **subclase** con la **extiende** palabra clave.
7. Una colección de métodos sin aplicación recibe el nombre de **la interfaz**.
8. Un espacio de nombres que organiza las clases y las interfaces por la funcionalidad que se llama un **paquete**.
9. El término API significa **Application Programming Interface** .

Respuestas a los ejercicios

1. Sus respuestas pueden variar dependiendo de los objetos del mundo real que está modelando.
2. Sus respuestas pueden variar aquí también, pero el mensaje de error específicamente una lista de los métodos necesarios que no se han aplicado.

```
/*
 * Copyright (c) 1995, 2008, Oracle and/or its affiliates. All rights reserved.


 */

class Bicycle {

      int cadence = 0;
      int speed = 0;
      int gear = 1;

      void changeCadence(int newValue) {
          cadence = newValue;
      }

      void changeGear(int newValue) {
          gear = newValue;
      }

      void speedUp(int increment) {
          speed = speed + increment;
      }

      void applyBrakes(int decrement) {
          speed = speed - decrement;
      }

      void printStates() {
          System.out.println("cadence:"+cadence+" speed:"+speed+" gear:"+gear);
      }
}
```

Cadenas de caracteres: la clase String

Para representar cadenas de caracteres en Java se emplea una clase, String, que es la que proporciona soporte para las operaciones que se realizan más comúnmente con cadenas de caracteres. La definición de un String es similar a la de un tipo primitivo. Las cadenas de caracteres que vayan incluidas en el código Java como literales deben ir rodeadas por comillas dobles:

```
String e ;   //declarado pero no inicializado
String e ="";   //cadena vacía
String e = "Hola";  //inicialización y asignación
```

La concatenación de dos cadenas de caracteres en Java es muy sencilla: se realiza con el operador +, es decir "sumando" las cadenas de caracteres. Lo ilustraremos con un ejemplo:

```
String saludo = "hola";
String nombre = "Pepe";
String saludaPepe = "";
saludaPepe = saludo + nombre;// saludaPepe toma el valor "holaPepe"
```

Si una cadena la intentamos concatenar con otro tipo de variable automáticamente se convierte la otra variable a String (en el caso de que la otra variable sea un objeto para realizar la conversión se invocará el método toString () de dicho objeto), de tal modo que en Java es perfectamente válido el siguiente código:

```
String saludo = "hola";
int  n = 5;
saludo = saludo + "" + n;// saludo toma el valor "hola 5"
```

Algunos métodos útiles de la clase String

A diferencia de C, las operaciones comunes sobre cadenas de caracteres, como la comparación o la extracción de una sabadeña, no se realizan a través de funciones (o métodos estáticos), sino que se realizan a través de métodos de la propia clase String . Por ejemplo, en la clase String hay un método que permite la extracción de una sabadeña de caracteres de otra. Su sintaxis es:

```
cadena.substring( int posici&oacute;nInicial, int posici&oacute;nFinal);
```

donde posiciónInicial y posiciónFinal son, respectivamente, la posición del primer carácter que se desea extraer y del primer carácter que ya no se desea extraer. Veamos un ejemplo de uso:

```
String saludo = "hola";
String subsaludo = saludo.substring(0,2);// subsaludo toma el valor "ho"
```

También puede extraerse un char de una cadena; para ello se emplea el método charAt(posición) , siendo posición la posición del carácter que se desea extraer. Se empieza a contar en 0.Para comparar cadenas de caracteres no puede emplearse el operador == . Este operador realiza una comparación de identidad, es decir, nos permitiría comprobar si dos objetos String son realmente un mismo objeto. Pero no nos permite comprobar si dos objetos String diferentes contienen el mismo texto. El segundo caso es el que se corresponde con la semántica de "igualdad" que habitualmente empleamos los seres humanos: no nos importa si son el mismo objeto, sino si su contenido es el mismo. Para comparar dos cadenas de caracteres debe emplearse otro método de String: equals. Su sintaxis es:

```
cadena1.equals(cadena2);
```

el método devuelve true si las dos cadenas son iguales y false si son distintas.¿Es posible pasar una cadena de caracteres a minúsculas o a mayúsculas ? ¿y reemplazar un carácter por otro? ¿O encontrar la primera ocurrencia de un carácter de una cadena?. La respuesta es sí, todas estas operaciones, y muchas otras, son posibles. ¿Cómo se realizan?. Empleando métodos de la clase String . ¿Qué métodos?. A esta pregunta, ya me niego a responder. Recomiendo al lector que antes de continuar leyendo el artículo consulte el javadoc de la clase String y se familiarice con las operaciones que soporta. En el listado 2 vemos un código que demuestra varias operaciones que se realizan con Strings. Este código también permite mostrar como en Java se distingue entre mayúsculas y minúsculas, de tal modo que "Hola" no es la misma cadena que caracteres que "hola".

```java
//LISTADO 2: Operaciones frecuentes con cadenas de caracteres
//código Ejemplo7.java del CD
 String saludo = "Hola";
 String saludo2 ="hola";
 int n = 5;
//Imprime por consola la sabadeña formada por los caracteres
//comprendidos entre el caractero 0 de saludo y hasta el  carácter 2
 System.out.println( saludo.substring(0,2));
//ejemplo de concatenacion
 System.out.println( saludo +" " + n);
//Imprime el resultado del test de igualdad entre saludo y saludo2.
 System.out.println( "saludo == saludo2 "+ saludo.equals(saludo2));
```