

Programación Orientada a Objetos

La era del gran enfoque monolítico aplicado al desarrollo de sistemas para el procesamiento de datos está llegando a su fin. Quienes se dedican al mejoramiento de sistemas se han dado cuenta de que existen diversas formas de especificar y programar sistemas. Aunque el enfoque tradicional IF.. THEN, DO WHILE es popular en muchas situaciones, está lejos de ser la mejor o la única forma viable. Por ejemplo, un sistema de diagnóstico médico podría implementarse mejor con reglas y una máquina de inferencias. En otras situaciones, el condicionamiento de una red neuronal y una máquina de inferencias. En otras situaciones, el condicionamiento de una red neuronal o un algoritmo genético podrían producirse los resultados deseados con mayor eficacia. Por otra parte, una lógica de predicados o una expresión funcional podría ser la mejor forma de especificar una aplicación. En otras palabras, si un sistema puede especificarse de diversas formas, ¿porqué no elegir la mejor mezcla que nos lleve a solucionar el problema dado? ¿Por qué limitarse a un modo de hacer las cosas cuando una combinación de métodos puede producir un resultado más económico, seguro, comprensible y presentable?

Existen muchos métodos para hacer desarrollos; siempre existirán y deberán existir. Las diversas tareas pueden tener características diferentes que requieran, desde luego, distintos tipos de métodos. El reto es la selección e integración de los procedimientos.

La respuesta, entonces, radica en permitir a los estudiosos de los sistemas elegir los métodos que resuelvan mejor el problema en cuestión. Sin embargo, ¿será posible integrar con éxito una anarquía de tal naturaleza? Por ejemplo, ¿será posible reunir de forma útil las reglas, la lógica, las funciones, las redes neuronales, las técnicas estructuradas, SQL, cliente-servidor, etc.? Sin una interconexión coherente, el desarrollo de sistemas será más complicado de lo que ya es. La orientación a objetos es una forma de conseguir esa coherencia; por su propia naturaleza, es una técnica de organización. Como tal, es una herramienta de integración muy útil.

La OO para sistemas de cómputo

Durante muchos años, la orientación a objetos fue asociada de manera exclusiva con un tipo particular de lenguaje de programación. Sin embargo, en la actualidad las nociones utilizadas por los lenguajes de programación OO (OOPL) se aplican como filosofía general al desarrollo de sistemas. Esto no significa que el desarrollo de sistemas OO deba especificarse en términos de clases que incluyan de manera física las definiciones de variables objeto y métodos codificados. Tampoco quiere decir que un sistema se especifique en términos de código heredado. Aunque las nociones de estructuras de clase y herencia se utilizan para definir la orientación a objetos, de hecho, solamente son *implementaciones* de OO. Desde el punto de vista conceptual, la orientación a objetos se interpreta ahora de manera más general.

En primer lugar, esta interpretación más amplia significa que la OO es una forma de organizar nuestras ideas con respecto a nuestro mundo. Esta organización se basa en los tipos de cosas, o *tipos de objetos*, de nuestro mundo. En consecuencia, podemos definir los atributos, las operaciones que se pueden realizar, las reglas, el aprendizaje de estos tipos de objetos, etc. En vez de una unidad física con variables y métodos, un método OO más general proporciona una manera de organizar nuestro conocimiento en forma conceptual. La orientación a



objetos ofrece, entonces, un índice de nuestro conocimiento, sin importar que éste se exprese en términos de reglas, lógica, funciones, lenguajes relacionales, redes neuronales o cualquier otra cosa. Además si ampliamos esta idea, podemos utilizar la OO como un método para organizar e interconectar diferentes tecnologías de software, incluyendo la base de conocimiento, la computación paralela, la reingeniería empresarial y el desarrollo rápido de aplicaciones.

La OO no se limita a los sistemas de información

En muchas organizaciones, el personal de sistemas de información (SI) desarrolla el software de aplicación. Aunque la información sea un producto importante de los sistemas de software, ciertamente no es todo lo que está implicado. Los sistemas para la transferencia de existencias, el monitoreo de pacientes y el control de plantas no son sistemas de información. Su principal meta no es la *información* que proporcionan, sino el *proceso* que realizan. De hecho, ofrecer información no es un proceso como cualquier otro de software. Las operaciones sobre un objeto **Existencia** incluir **crear, terminar, transferir, desplegar, desplegarActividadActual, o imprimirFechaInicialDeOferta**. En otras palabras, quien se dedica al desarrollo de sistemas OO no se limita al mundo de la información; puede especificar la automatización de *cualquier* tipo de proceso. Además, tiene la posibilidad de especificar y programar el sistema con diversos métodos (como reglas, funciones, lógica, SQL). Tal vez un nombre más descriptivo para este tipo de persona sería el de *ingeniero de software de aplicación*. Así, el término *sistema de información* estaría limitado a los procesos que proporcionan información.

La OO para sistemas en general

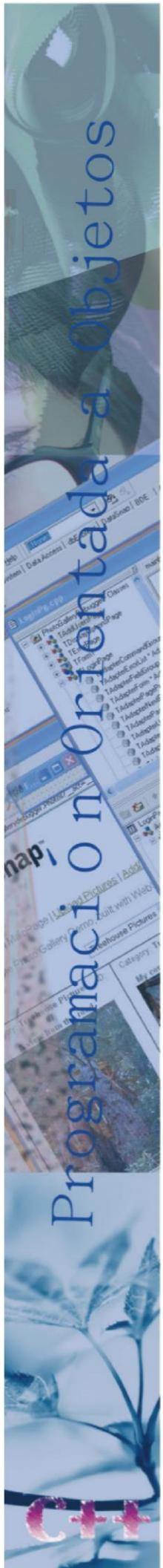
Además de trascender la idea de un lenguaje único de programación, ¿es posible que el enfoque OO nos lleve más allá de los sistemas de software en general? Es seguro que la ingeniería de software es una especialidad que sobrevivirá por muchos años. Sin embargo, es sólo uno de los mecanismos para la implementación de sistemas. En particular, en esta era de reingeniería de procesos empresariales (BPR), se enfatiza *cualquier* proceso empresarial, y no sólo los automatizados mediante software.

Sin embargo, la cuestión es saber con certeza si un enfoque OO se puede aplicar con efectividad a los sistemas empresariales en general. ¿Podemos utilizar la OO para comprender y especificar procesos relacionados o no con el software? Por supuesto, la respuesta es sí. De hecho, la OO surgió de la necesidad de simular sistemas de forma más sencilla, no sólo de información, sino de cualquier tipo. Los fundadores de Simula determinaron que la sola administración de un gran proceso no era la respuesta, sino que una solución consta de varios componentes que interactúan entre sí. Éstos no están basados en una modularización *ad hoc*, sino en los tipos, las clases o las cosas por simular. Así resulta más fácil localizar y controlar la estructura y el comportamiento del sistema.

Aunque Simula se desarrolló como el primer lenguaje orientado a objetos, su herencia más importante fue la filosófica. La orientación a objetos proporciona una forma de construir cualquier tipo de sistema, sin importar el modo de implantación. Además, esta misma especificación de ingeniería orientada a objetos puede servir como guía en muchas otras disciplinas, ya sea que impliquen personas, máquinas o computadoras.

El análisis Orientado a objetos no es un enfoque que modele la realidad, sino la manera en que las personas la comprenden y la procesan, principalmente mediante los conceptos adquiridos. Así pues, la comprensión de los fundamentos de la orientación a objetos requiere comprender primero qué son los conceptos y la forma de utilizarlos en el análisis orientado a objetos.





Conceptos y realidad

La formación de conceptos nos ayuda a ordenar nuestra vida. Los psicólogos han propuesto que, cuando un bebé inicia su vida, su mundo es una enorme confusión. A muy temprana edad, desarrolla la noción de ser alimentado. Poco después, aprende a distinguir los sonidos de sus padres. Al parecer, los seres humanos poseemos una capacidad innata de percibir irregularidades y reconocer los diferentes objetos que ofrece el mundo. Llegamos a desarrollar conceptos tales como *azul* y *cielo* y después aprendemos a combinar conceptos para formar otros nuevos, como *cielo azul*. Al crecer, desarrollamos construcciones conceptuales elaboradas que aumentan el significado, la precisión y la utilidad. Por ejemplo, el cielo sólo es azul en días despejados, o bien, el cielo no es realmente azul, sólo parece azul por el efecto de la atmósfera en el planeta Tierra.



Un concepto es una idea o noción de algo que aplicamos a las cosas, u objetos, en forma consciente.

Objetos

Nuestra realidad consta de los *conceptos* que poseemos y de las cosas u *objetos* a los que se aplican dichos conceptos.

Un objeto es cualquier cosa a la que se aplica un concepto. Es un caso particular de concepto.

La palabra *objeto* es utilizada en varias formas por los practicantes de la POO. Las palabras *objeto* e *instancia* (o caso particular) se pueden emplear como sinónimos. Un ejemplo sencillo de objeto es la pluma sobre la mesa de su tía (un caso específico, particular de pluma). Otros objetos podrían ser la ciudad donde usted vive, usted mismo, su trabajo, un sonido, cierto proceso, un hecho particular, un instante en el tiempo, un registro en una base de datos, etc. Observe que todos estos son objetos no son datos. El análisis orientado a objetos investiga los objetos sin prejuicio de si van a ser datos o no. De esta forma, analizamos la comprensión humana antes de trabajar con bits, bytes, campos y registros.

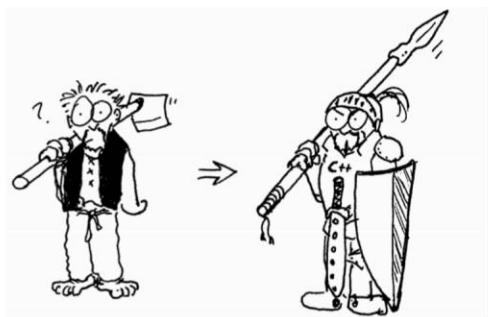
De esta manera, cualquier cosa en la que podamos pensar, a la que podamos referirnos, describir, analizar o experimentar, será un objeto siempre que tengamos los conceptos para hacerlo así.

Conceptos y Principios orientados a objetos

Vivimos en un mundo de objetos. Estos objetos existen en la naturaleza, en entidades hechas por el hombre, en los negocios y en los productos que usamos. Ellos pueden ser clasificados, descritos, organizados, combinados, manipulados y creados. Por esto no es sorprendente que se proponga una visión orientada a objetos para la creación de software de computadora, una abstracción que modela el mundo de forma tal que nos ayuda a entenderlo y gobernarlo mejor.

Un enfoque orientado a objetos para el desarrollo de software se propuso por primera vez a finales de los años 60's. Sin embargo, las tecnologías de objetos han necesitado casi veinte años para llegar a ser ampliamente usadas. Durante la primera mitad de los años 90, la ingeniería de software orientada a objetos se convirtió en el paradigma de elección para muchos productores de software y un creciente número de sistemas de información y profesionales de la ingeniería. A media que pase el tiempo las tecnologías de objetos sustituirán a los enfoques clásicos de desarrollo de software. Una pregunta importante es ¿Por qué?.





La respuesta (como muchas otras respuestas a interrogantes sobre ingeniería del software) no es sencilla. Algunas personas argumentarían que los profesionales del software sencillamente añoraban un nuevo enfoque, pero esta visión es muy simplista. Las tecnologías de objeto llevan un número de beneficios inherentes que proporcionan ventajas a los niveles de dirección y técnico.

Las tecnologías de objeto llevan a reutilizar y la reutilización (de componentes de software) lleva a un desarrollo de software más rápido y programas de mejor calidad. El software orientado a objetos es más fácil de mantener debido a que su estructura es inherentemente descompuesta.

Esto lleva a menores efectos colaterales cuando se deben hacer cambios y provoca menos frustración en el ingeniero de software y el cliente. Adicionalmente, los sistemas orientados a objetos son más fáciles de adaptar y escalar (por ejemplo, pueden crearse grandes sistemas ensamblando subsistemas reutilizables).

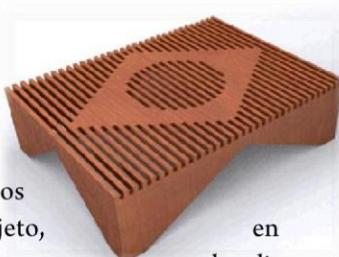
El paradigma orientado a objetos

Durante muchos años el término orientado a objetos (OO) se usó para significar un enfoque de desarrollo de software que usaba uno de los lenguajes orientados a objetos (Ada 96, C++, Eiffel, Smalltalk). Hoy en día el paradigma OO encierra una completa visión de la ingeniería del software.

Los beneficios de la tecnología orientada a objetos se fortalecen si se usa antes y durante el proceso de ingeniería de software. Esta tecnología orientada a objetos considerada debe hacer sentir su impacto durante todo el proceso de ingeniería de software. Un simple uso de la programación orientada a objetos (POO) no brindara los mejores resultados. Los ingenieros de software y sus directores deben considerar estos elementos como análisis orientado a objetos (AROO), del diseño orientado a objetos (DOO), análisis del diseño orientado a objetos (ADOO), de sistemas de gestión de bases de datos orientadas a objetos (SGBDOO) y de ingeniería de software orientada a objetos asistida por computadora (ISOAAC).

Conceptos de orientación a objetos

Para poder entender la visión orientada a objetos, consideremos un ejemplo de un objeto del mundo real –la cosa sobre la que usted está sentado ahora mismo–, una silla. La **silla** es un miembro (el término “instancia” también se usa) de una clase mucho más grande de objetos que llamaremos **mobiliario**. Un conjunto de atributos genéricos puede asociarse con cada objeto, la clase **mobiliario**. Por ejemplo, todo mueble tiene un costo, dimensiones, peso, localización y color, entre otros muchos posibles atributos. Éstos son aplicables a cualquier elemento sobre el que se hable, una mesa o silla, un sofá o un armario. Como **silla** es un miembro de la clase **mobiliario**, **silla** hereda todos los atributos definidos para esta clase.



Una vez definida la clase, los atributos pueden reutilizarse al crear nuevas instancias de la clase. Por ejemplo, supongamos que debemos definir un nuevo objeto llamado **sillesa** (un cruce entre una silla y una mesa) que es un miembro de la clase mobiliario. La **sillesa** hereda todos los atributos de mobiliario.





Se ha intentado definir una clase describiendo sus atributos, pero algo falta. Todo objeto en la clase **mobiliario** puede manipularse de varias maneras. Puede comprarse y venderse, modificarse físicamente (por ejemplo, puede usted eliminar una pata o pintar el objeto de azul) o moverse de un lugar a otro. Cada una de estas operaciones (otros términos son “servicios” o “métodos” modificarán uno o mas atributos del objeto. Por ejemplo, si el atributo *localización* es un dato compuesto definido

Localización = edificio + piso + habitación

entonces una operación denominada *move* modificaría uno o más de los elementos dato (*edificio*, *piso* o *habitación*) que conforman el atributo *localización*. Para hacer esto, *move* debe tener “conocimiento” sobre estos elementos. La operación *move* puede usarse para una silla o una mesa, debido a que ambas son instancias de la clase **mobiliario**. Todas las operaciones válidas (por ejemplo, *comprar*, *vender*, *pesar*) de la clase **mobiliario** están “conectadas” a la definición del objeto y son heredadas por todas las instancias de esta clase.

El objeto **silla** (y todos los objetos en general) encapsula datos (los valores de los atributos que definen la silla), operaciones (las acciones que se aplican para cambiar los atributos de la silla), otros objetos (pueden definirse objetos compuestos), constantes (fijar valores) y otra información relacionada. El encapsulamiento significa que toda esta información se encuentra empaquetada bajo un nombre y puede de reutilizarse como una especificación o componente de un programa.

Se puede resumir entonces a la orientación a objetos de la siguiente forma resumida:

Orientación a objetos = objetos + clasificación + herencia + comunicación



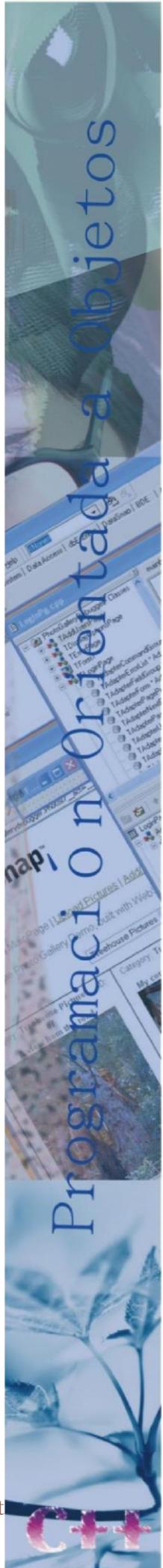
Clases y objetos

Una **clase** es un concepto OO que encapsula las abstracciones de datos y procedimientos que se requieren para describir el contenido y comportamiento de alguna entidad del mundo real.

Las abstracciones de datos (atributos) que describen la clase están encerradas por una “muralla” de abstracciones procedimentales (llamadas métodos o servicios) capaces de manipular los datos de alguna manera. La única forma de alcanzar los atributos (y operar sobre ellos) es ir a través de algunos de los métodos que forman la muralla. Por lo tanto, la clase encapsula datos (dentro de la muralla) y el proceso que manipula los datos (los métodos que componen la muralla). Esto posibilita el ocultamiento de la información y reduce el impacto de efectos colaterales asociados a cambios. Como estos métodos tienden a manipular un número limitado de atributos, esto es una alta cohesión, y como la comunicación ocurre sólo a través de los métodos que encierra la “muralla”, la clase tiende a un acoplamiento con otros elementos del sistema.

Puesto de otra manera, una clase es una descripción generalizada (por ejemplo, una plantilla, un patrón o una copia) que describe una colección de objetos similares. Por definición, todos los objetos que existen dentro de una clase heredan sus atributos y las operaciones disponibles para la manipulación de los atributos. Una *superclase* es una colección de clases y una *subclase* es una instancia de una clase.

Estas definiciones implican la existencia de una *jerarquía de clases* en la cual los atributos y operaciones de la superclase son heredados por subclases que pueden añadir, cada una de ellas, atributos “privados” y métodos.



Atributos

Los atributos están asociados a clases y objetos, y que ellos describen la clase o el objeto de alguna manera.

Las entidades de la vida real están a menudo descritas con palabras que indican características estables. La mayoría de los objetos físicos tienen características tales como forma, peso, color y tipo de material. Las personas tienen características incluyendo fechas de nacimiento, padres, nombres y color de los ojos. Una característica puede verse como una relación binaria entre una clase y cierto dominio.



La relación binaria anteriormente señalada implica que un atributo puede tomar un valor definido por un dominio enumerado. En la mayoría de los casos, un dominio es simplemente un conjunto de valores específicos. Por ejemplo, suponga que una clase **coche** tiene un atributo *color*. El dominio de valores de *color* es (blanco, negro, plata, gris, azul, rojo, amarillo, verde). En situaciones más complejas, el dominio puede ser un conjunto de clases. Continuando con el ejemplo, la clase **coche** también tiene un atributo *motor* que abarca los siguientes valores de dominio: (valor 15 opción económica, 16 válvulas opción deportiva, 24 válvulas opción deportiva, 32 válvulas opción de lujo). Cada una de las opciones indicadas tiene un conjunto de atributos específicos de ella.

Las características (valores del dominio) pueden aumentarse asignando un valor por defecto (característica) a un atributo. Por ejemplo, el atributo *motor* destacado antes tiene el valor por defecto 16 válvulas opción deportiva. Esto puede ser también útil para asociar una probabilidad con una característica particular a través de pares <valor, probabilidad>. Considere el atributo *color* para coche. En algunas aplicaciones (por ejemplo, planificación de la producción) puede ser necesario asignar una probabilidad a cada uno de los colores (blanco y negro son más probable como colores de coches).

Operaciones, métodos y servicios

Un objeto encapsula datos (representados como una colección de atributos) y los algoritmos que procesan estos datos. Estos algoritmos son llamados *operaciones*, *métodos* o *servicios* y pueden ser vistos como módulos en un sentido convencional.

Cada una de las operaciones encapsuladas por un objeto proporciona una representación de uno de los comportamientos del objeto. Por ejemplo, la operación *DeterminarColor* para el objeto **automóvil** extraerá el color almacenado en el atributo *color*. La consecuencia de la existencia de esta operación es que la clase **automóvil** ha sido diseñada para recibir un estímulo (mensaje) que requiere el color de una instancia particular de una clase. Cada vez que un objeto recibe un estímulo, éste inicia un comportamiento. Éste puede ser tan simple como determinar el color del coche o tan complejo como la iniciación de una cadena de estímulos que se pasan entre una variedad de objetos diferentes. En este último caso, considere un ejemplo en el cual un estímulo inicial recibido por el objeto nº 1 resulta en una generación de otros dos estímulos que se envían al objeto nº 2 y al objeto nº 3. Las operaciones encapsuladas por el segundo y tercer objetos actúan sobre el estímulo devolviendo información necesaria para el primer objeto. El objeto nº 1 usa entonces la información devuelta para satisfacer el comportamiento demandado por el estímulo inicial.



Mensajes

Los mensajes son el medio a través del cual los objetos interactúan. Usando la terminología introducida en la sección precedente, un mensaje estimula la ocurrencia de cierto comportamiento en el objeto receptor. El comportamiento se realiza cuando se ejecuta una operación.

Una operación dentro de un *objeto emisor* genera un mensaje de la forma:

Mensaje: [destino, operación, parámetros]

donde *destino* define el objeto *receptor* el cual es estimulado por el mensaje, *operación* se refiere al método que recibe el mensaje y *parámetros* proporciona información requerida para el éxito de la operación.

El paso de mensajes mantiene comunicados un sistema orientado a objetos. Los mensajes proporcionan una visión interna del comportamiento de objetos individuales, y del sistema OO como un todo.

Encapsulamiento, herencia y polimorfismo

Aunque la estructura y terminología ya definida diferencian los sistemas OO a partir de sus componentes convencionales, tres características de sistemas orientados a objetos los hacen únicos. Como ya se ha observado, las clases OO y los objetos derivados de ella encapsulan los datos y las operaciones que trabajan sobre estos en un único paquete. Esto proporciona un número importante de beneficios:

- Los detalles de implementación interna de datos y procedimientos están ocultos al mundo exterior (ocultamiento de la información). Esto reduce la propagación de efectos colaterales cuando ocurren cambios.
- Las estructuras de datos y las operaciones que las manipulan están mezcladas en una entidad sencilla: la clase. Esto facilita la reutilización de componentes.
- Las interfaces entre objetos encapsulados están simplificadas. Un objeto que envía un mensaje no se tiene que preocupar de los detalles de estructuras de datos internos del receptor. Por tanto, se simplifica la interacción y el acoplamiento del sistema tiende a reducirse.



La herencia es una de las diferencias clave entre sistemas convencionales y sistemas OO. Una subclase **Y** hereda todos los atributos y operaciones asociadas con su superclase **X**. Esto significa que todas las estructuras de datos y algoritmos originalmente diseñados e implementados para **X** están inmediatamente disponible para **Y** (no es necesario trabajo extra). La reutilización se realiza directamente. Cualquier cambio en los datos u operaciones contenidas dentro de una superclase se hereda inmediatamente por todas las subclases que se derivan de la superclase. Debido a esto, la jerarquía de clases se convierte en un mecanismo a través del cual los cambios (a altos niveles) pueden propagarse inmediatamente a través de todo el sistema.

Es importante destacar que en cada nivel de la jerarquía de clases, pueden añadirse nuevos atributos y operaciones a aquellos que han sido heredados de niveles superiores de la jerarquía. De hecho, cada vez que se debe crear una nueva clase, el ingeniero de software tiene varias opciones:

- La clase puede diseñarse y construirse de la nada. Esto es, no se usa la herencia.
- La jerarquía de clases puede ser rastreada para determinar si una clase ascendiente contiene la ma-



yoría de los atributos y operaciones requeridas.

- La nueva clase hereda de su clase ascendiente, y pueden añadirse los elementos requeridos.
- La jerarquía de clases puede reestructurarse de tal manera que los atributos y operaciones requeridos puedan heredarse por la nueva clase.
- Las características de una clase existente pueden sobrescribirse y se pueden implementar versiones privadas de atributos u operaciones para la nueva clase.

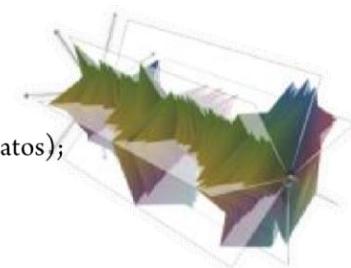
En algunos casos, es tentador heredar algunos atributos y operaciones de una clase y otros de otra clase. Esta acción se llama *herencia múltiple* y es controvertida. En general, la herencia múltiple complica la jerarquía de clases y crea problemas potenciales en el control de la configuración. Como las secuencias de herencia múltiple son más difíciles de seguir, los cambios en la definición de una clase que reside en la parte superior de una jerarquía pueden tener impactos no deseados originalmente en las clases definidas en zonas inferiores de la arquitectura.

El *polimorfismo* es una característica que reduce en gran medida el esfuerzo necesario para extender un sistema ORIENTADO A OBJETOS. Para entender el polimorfismo, considere una aplicación convencional que debe dibujar cuatro tipos diferentes de gráficos: gráficos de línea, gráficas de pastel, histogramas y diagramas de Kiviat. Idealmente, una vez que se han recogido los datos necesarios para un tipo particular de gráfico, el gráfico debe dibujarse él mismo. Para realizar esto en una aplicación convencional (y mantener la cohesión entre módulos), será necesario desarrollar módulos de dibujo para cada tipo de gráfico. Entonces, dentro del diseño para cada tipo de gráfico, se deberá añadir una lógica de control semejante a la que sigue:

case of tipo_gráfico:

```
if tipo_gráfico = gráfico_linea then DibujarLinea (datos);
    if tipo_gráfico = gráfico_tarta then DibujarTarta(datos);
    if tipo_gráfico = gráfico_histograma then DibujarHistograma (datos);
    if tipo_gráfico = gráfico_kiviat then DibujarKiviat(datos);
```

end case;



Aunque este diseño es razonablemente evidente, añadir nuevos tipos de gráfico puede ser artesanal o complicado. Un nuevo módulo de dibujo deberá crearse para cada tipo de gráfico y entonces la lógica de control deberá actualizarse para cada gráfico.

Para resolver este problema en un sistema OO, cada uno de los gráficos mencionados arriba se convierte en una subclase de una clase general llamada **grafico**. Usando un concepto llamado *sobrecarga*, cada subclase define una operación llamada *dibujar*. Un objeto puede enviar un mensaje *dibujar* a cualquiera de los objetos instanciados de cualquiera de las subclases. El objeto que recibe el mensaje invocará su propia operación *dibujar* para crear el gráfico apropiado. Por esto, el diseño anteriormente mostrado se reduce a:

```
tipo_gráfico dibujar
```

Cuando se debe añadir un nuevo tipo de gráfico al sistema, se crea una subclase con su propia operación *dibujar*. Pero no se requieren cambios dentro de un objeto que quiere dibujar un gráfico pues el mensaje *tipo_gráfico dibujar* permanece sin cambiar. En resumen, el polimorfismo permite que un número de operaciones diferentes tengan el mismo nombre. Esto reduce el acoplamiento entre objetos, haciendo a cada uno más independiente.