Capítulo 2
El diseño orientado a objetos

1. Enfoque procedural y descomposición funcional

Antes de enunciar los aspectos básicos de la programación orientada a objetos, vamos a repasar el enfoque procedural con ayuda de un ejemplo concreto de organización de código.

La programación procedural es un paradigma de programación que considera a los diferentes actores de un sistema como objetos prácticamente pasivos que un procedimiento central utilizará para una función dada.

Tomemos como ejemplo la distribución de agua corriente en los domicilios e intentemos emular este concepto en una aplicación muy sencilla. El análisis procedural (como el análisis de objetos) revela la siguiente lista de objetos:

– el grifo del lavabo;
– el depósito del agua;
– un sensor del nivel de agua, con contador en el depósito;
– la bomba de alimentación que envía el agua del río.

El código del programa "procedural" consistirá en crear un conjunto de variables que represente los argumentos de cada componente y después escribir el bucle de operación de la gestión central, verificando los valores leídos y actuando en función del resultado de las pruebas. Veremos que, por un lado, están las variables y por otro, las acciones.

2. La transición hacia el enfoque orientado a objetos

La programación orientada a objetos es un paradigma de programación que considera los diferentes actores de un sistema como objetos activos y relacionados. El enfoque orientado a objetos es mucho más cercano a la realidad.

En nuestro ejemplo, el usuario abre el grifo; este libera presión y el agua fluye desde el depósito hasta el lavabo; el sensor/flotador del depósito llega a un nivel que activa la bomba; el usuario cierra el grifo; alimentado por la bomba, el depósito del agua se rellena y el sensor/flotador alcanza un nivel que detiene la bomba.

En este enfoque, se comprueba que los objetos interactúan; no existe ninguna operación central que defina dinámicamente el funcionamiento de los objetos. En su lugar, hubo un análisis funcional que condujo a la creación de diferentes objetos, su montaje y a establecer sus relaciones.

El código del programa "orientado a objetos" va a seguir esta realidad, proponiendo los objetos descritos anteriormente, pero definiendo los métodos de intercambio adecuados entre estos objetos que conducirán al funcionamiento esperado.

Observación

Los conceptos en la orientación a objetos son muy próximos a la realidad...

3. Las características de la POO

3.1 El objeto, la clase y la referencia

3.1.1 El objeto

El objeto es el elemento básico de la POO. El objeto es la unión:

– de una lista de variables de estado,

– de una lista de comportamientos,

– de una identificación.

Las variables de estado cambian durante el ciclo de vida del objeto. Supongamos el caso de un lector de música digital. Cuando se compra el aparato, los estados de este objeto podrían ser:

– memoria libre = **100%**

– tasa de carga de la batería = **bajo**

– aspecto exterior = **nuevo**

A medida que se usa, sus estados van a cambiar. Rápidamente, la memoria libre va a descender, la tasa de carga de la batería cambiará, así como el aspecto exterior, en función del cuidado que tenga el usuario con el aparato.

Los comportamientos del objeto definen lo que puede hacer este objeto: Reproducir la música - Ir a la pista siguiente - Ir a la pista anterior - Subir el volumen, etc. Una parte de los comportamientos del objeto son accesibles desde el exterior del objeto: botón Play, botón Stop, etc. y otra parte solo desde el interior: lectura de la tarjeta de memoria, decodificación de la música desde el archivo, etc. Hablamos de "encapsulación" para definir un límite entre los comportamientos accesibles desde el exterior y los comportamientos internos.

La identificación de un objeto es información, separada de la lista de estados, que permite diferenciar el objeto de sus congéneres (es decir, de otros objetos del mismo tipo). La identificación puede ser un número de referencia o una cadena de caracteres única, formada durante la creación del objeto; también puede ser una dirección de memoria. En realidad, su forma depende totalmente de la problemática asociada.

El objeto POO puede estar relacionado con una entidad real, en nuestro caso el lector digital, pero también puede estar relacionado con una entidad totalmente virtual, como una cuenta de cliente, una entrada en un directorio telefónico, etc. La finalidad del objeto informático es gestionar la entidad física o emularla.

Incluso si esto no está en su naturaleza básica, el objeto se puede hacer persistente, es decir, que sus estados se pueden registrar en un soporte que memoriza la información de manera intemporal. A partir de este registro, el objeto se podrá recrear cuando el sistema lo necesite.

3.1.2 La clase

La clase es el "molde" a partir del que se va a crear el objeto en memoria. La clase contiene los estados y comportamientos comunes de un mismo tipo. Los valores de estos estados estarán contenidos en sus objetos.

Todas las cuentas corrientes de un mismo banco contienen los mismos argumentos (datos de contacto del titular, saldo, etc.) y todos tienen las mismas funciones (pago a débito o crédito, etc.). Estas definiciones deben estar contenidas en una clase y, cada vez que un cliente abra una nueva cuenta, esta clase servirá de modelo para crear el objeto cuenta.

La pantalla de los reproductores de música digitales ofrece un mismo modelo en diferentes colores, con tamaños de memoria modulables, etc. Cada uno de los dispositivos de visualización es un objeto fabricado a partir de la información de una única clase. Durante su realización, se seleccionan los atributos del aparato en función de criterios estéticos y comerciales.

Una clase puede contener muchos atributos. Pueden ser de tipo primitivo – enteros, caracteres, etc. – y también de tipos más complejos. De hecho, una clase puede contener una o varias clases de otros tipos. En este caso, hablamos de composición o incluso de "fuerte acoplamiento". La destrucción de la clase principal implica, evidentemente, la destrucción de las clases que contiene. Por ejemplo, si una clase *hotel* contiene una lista de *habitaciones*, la destrucción del *hotel* implica la destrucción de las *habitaciones*.

Una clase puede hacer "referencia" a otra clase; en este caso, el acoplamiento se llama "débil" y los objetos pueden vivir de manera independiente. Por ejemplo, su PC está conectado a su impresora. Si su PC se estropea, su impresora funcionará con el nuevo PC. Hablamos de asociación.

Además de sus atributos, la clase también contiene una serie de "comportamientos", es decir, una serie de métodos con firma y código adjunto. Estos métodos se "copian" directamente en los objetos y se utilizan tal cual.

Se declara la clase y su contenido en un mismo archivo fuente con ayuda de una sintaxis que estudiaremos en detalle. Los desarrolladores C++ aprecian el hecho de que además no exista, por un lado, una parte de definiciones y por otra, una parte de implementaciones. De hecho, en C# el archivo de programa (de extensión .CS) contiene las definiciones de todos los estados y eventualmente un valor "inicial" y las definiciones e implementaciones de todos los comportamientos. Veremos que en C# existe una solución que permite definir una clase en varios archivos, para que las personas que participan en un mismo proyecto puedan trabajar en paralelo, sin que una persona afecte al trabajo de otra.

3.1.3 La referencia

Los objetos se construyen a partir de la clase, por un proceso llamado instanciación y, por tanto, cualquier objeto es una instancia de una clase. Cada instancia empieza con una ubicación única en memoria. Los desarrolladores conocen esta ubicación en memoria con el nombre de *puntero*. C y C++ se han convertido en una *referencia* para los desarrolladores C# y Java.

Cuando el desarrollador necesita un objeto durante una operación, debe:

– declarar y nombrar una variable del tipo de la clase que va a utilizar;

– instanciar el objeto y registrar su referencia en esta variable.

Cuando se realiza esta instanciación, el programa accederá a las propiedades y métodos del objeto utilizando la variable que contiene su referencia. Cada instancia es única. Por el contrario, varias variables pueden "apuntar" a una misma instancia. Cuando ninguna variable apunta a una instancia dada, el recolector de basura registra esta instancia como instancia para ser destruida.

3.2 La encapsulación

La encapsulación consiste en crear un tipo de caja negra, que contiene internamente un mecanismo protegido y externamente un conjunto de comandos que van a permitir manipularla. Este juego de comandos se hace de tal manera que será imposible alterar el mecanismo, protegido frente a casos de uso incorrecto. La caja negra será tan opaca que será imposible para el usuario intervenir directamente sobre el mecanismo.

Como habrá entendido, la caja negra no es otra cosa que un objeto con métodos públicos de "alto nivel", que controlan con rigor los argumentos que se pasan antes de utilizarlos en una operación. Respetando el principio de encapsulación, el usuario del objeto jamás podrá acceder "directamente" a sus datos. Gracias a este control, su objeto se utilizará correctamente, por lo que será más fiable y su puesta a punto más sencilla. En el mundo Java, los métodos que permiten acceder en modo lectura a los datos son los descriptores de acceso, y los que permiten acceder en modo escritura son los modificadores. Veremos que C# ofrece una solución muy elegante, las propiedades, que permiten conservar el lado práctico del acceso directo a los datos del objeto, respetando los conceptos principales de la encapsulación.

3.3 La herencia

Otra noción importante de la POO afecta a la herencia. Para explicarla, imaginemos que debemos construir un sistema de gestión de los diferentes tipos de empleados de una empresa. Como resultado de un análisis rápido, se extrae una lista de propiedades comunes a todos los puestos. De hecho, aunque el empleado sea temporal, directivo, mando intermedio o incluso director, siempre tiene un nombre, un apellido y un número de seguridad social. A esto se le llama generalización; consiste en factorizar los elementos comunes de un conjunto de clases en una clase más general, llamada superclase en Java y "clase de base" en C# y C++. La clase que hereda de la superclase, se llama subclase o clase heredada.