Introducción a los conceptos de Programación Orientada a Objetos

D.1 Introducción

Después de aprender C, probablemente aprenderá uno o más lenguajes orientados a objetos basados en C o influenciados por C. Estos incluyen Java, C++, C#, Objective-C, Python, Swift y muchos más. Estos lenguajes suelen admitir varios paradigmas de programación:

- Programación procedimental,
- Programación orientada a objetos,
- Programación genérica, y
- Programación funcional.

Este apéndice presenta una visión general de la terminología y los conceptos de la programación orientada a objetos.

D.2 Lenguajes de programación orientada a objetos

C dio origen a una nueva generación de lenguajes de programación que van más allá del modelo de programación procedimental de C. A medida que aumenta la demanda de software nuevo y más potente, es importante desarrollar software de forma rápida, correcta y económica. Los **objetos**, o más precisamente, las **clases** de las que provienen, son esencialmente componentes de software reutilizables. Existen objetos de fecha, objetos de tiempo, objetos de audio, objetos de video, objetos de automóviles, objetos de personas, etc. Casi cualquier sustantivo puede representarse razonablemente como un objeto de software en términos de **atributos** (p. ej., nombre, color y tamaño) y **comportamientos** (p. ej., cálculo, movimiento y comunicación). Los grupos de desarrollo de software pueden utilizar un enfoque de diseño e implementación modular y orientado a objetos para ser más productivos que con las técnicas populares anteriores. Los programas orientados a objetos suelen ser más fáciles de entender, corregir y modificar.

D.3 El automóvil como objeto[1]

Para ayudarle a comprender los objetos y su contenido, considere una analogía sencilla. Suponga que quiere conducir un coche y acelerarlo presionando el pedal del acelerador. ¿Qué debe suceder antes de poder hacerlo? Pues bien, antes de poder conducir un coche, alguien tiene que diseñarlo. Un coche suele empezar con planos de ingeniería, similares a los que describen el diseño de una casa. Estos planos incluyen el diseño del pedal del acelerador. El pedal oculta al conductor los complejos mecanismos que aceleran el coche, al igual que el pedal del freno oculta los mecanismos que lo frenan y el volante los que lo hacen girar. Esto permite a las personas conducir un coche incluso con poco o ningún conocimiento de cómo funcionan los motores, los frenos y la dirección.

Así como no se pueden cocinar según los planos de una cocina, no se puede conducir un coche según los planos de ingeniería. Antes de poder conducir un coche, debe construirse a partir de los planos de ingeniería que lo describen. Un coche completo tiene un pedal de acelerador para acelerar, pero ni siquiera eso es suficiente. El coche no acelerará por sí solo (¡ojalá!), así que el conductor debe pisar el pedal para acelerarlo.

D.4 Métodos y clases

Realizar una tarea en un programa orientado a objetos requiere un **método**. Los métodos albergan las sentencias del programa que realizan sus tareas. Cada método oculta estas sentencias al usuario, al igual que el pedal del acelerador de un coche oculta al conductor los mecanismos que lo hacen ir más rápido. En la programación orientada a objetos, una unidad de programa llamada **clase** alberga el conjunto de métodos que realizan las tareas de la clase. Por ejemplo, una clase que representa una cuenta bancaria podría contener un método para depositar dinero en una cuenta, otro para retirar dinero de una cuenta y un tercero para consultar el saldo de la cuenta. Una clase es similar en concepto a los planos de ingeniería de un coche, que albergan los diseños del pedal del acelerador, el volante, etc.

D.5 Instanciación

Así como alguien tiene que construir un automóvil a partir de sus planos de ingeniería antes de poder conducirlo, es necesario construir un objeto de una clase antes de que un programa pueda realizar las tareas que definen sus métodos. Este proceso se denomina **instanciación**. Un objeto se denomina entonces una **instancia** de su clase.

[1] Mientras lee el resto de este apéndice, piense en cómo los coches autónomos afectarían la discusión

D.6 Reutilización

Al igual que los planos de ingeniería de un coche se pueden reutilizar muchas veces para construir muchos coches, se puede reutilizar una clase muchas veces para construir muchos objetos. Reutilizar clases existentes al crear nuevas clases y programas ahorra tiempo y esfuerzo. La reutilización también ayuda a construir sistemas más fiables y eficaces. Las clases y componentes existentes suelen ser sometidos a exhaustivas pruebas y depuración (búsqueda y eliminación de errores) y ajustes de rendimiento. Así como el concepto de piezas intercambiables fue crucial para la Revolución Industrial, las clases reutilizables son cruciales para la revolución del software impulsada por la tecnología de objetos.

En lenguajes orientados a objetos como C++, Java, C#, Python, Swift y muchos más, se suele utilizar un enfoque de **bloques de construcción** para crear los programas. Para evitar reinventar la rueda, se utilizan piezas existentes de alta calidad siempre que sea posible. Esta reutilización de software es una ventaja clave de la programación orientada a objetos.

D.7 Mensajes y llamadas a métodos

Al conducir un coche, al pisar el acelerador se envía un mensaje al coche para que realice una tarea; es decir, para que vaya más rápido. De forma similar, se envían mensajes a un objeto. Cada mensaje se implementa como una **llamada a un método** que indica a un método del objeto que realice su tarea. Por ejemplo, un programa podría llamar al método de depósito de un objeto de cuenta bancaria para aumentar el saldo de la cuenta en una cantidad específica.

D.8 Atributos y variables de instancia

Un coche, además de tener capacidades para realizar tareas, también tiene atributos como su color, número de puertas, nivel de gasolina en el tanque, velocidad actual y su historial de kilómetros totales recorridos (es decir, la lectura del odómetro). Al igual que sus capacidades, los atributos del coche se representan como parte de su diseño en sus diagramas de ingeniería (que, por ejemplo, incluyen un odómetro y un indicador de combustible). Al conducir un coche, estos atributos se conservan con el vehículo. Cada coche mantiene sus propios atributos. Por ejemplo, cada coche sabe cuánta gasolina tiene en su propio tanque, pero no cuánta hay en los tanques de los demás.

De forma similar, un objeto tiene atributos que conserva a medida que se utiliza en un programa. Estos atributos se especifican como parte de la clase del objeto. Por ejemplo, un objeto de cuenta bancaria tiene un atributo “saldo” que representa la cantidad de dinero en la cuenta. Cada objeto de cuenta bancaria conoce el saldo

de la cuenta que representa, pero no los saldos de las demás cuentas del banco. Los atributos se especifican mediante las **variables de instancia** de la clase. Los atributos y métodos de una clase (y de sus objetos) están estrechamente relacionados, por lo que las clases los integran.

D.9 Herencia

Se puede crear fácilmente una nueva clase de objetos mediante **herencia**: la nueva clase (denominada **subclase**) comienza con las características de una clase existente (denominada **superclase**), posiblemente personalizándolas y añadiéndoles características únicas. En nuestra analogía del coche, un objeto de la clase convertible es ciertamente un objeto de la clase más general “automóvil”, pero más específicamente, el techo se puede subir o bajar.

D.10 Análisis y Diseño Orientado a Objetos (OOAD)

Muchos programadores crean el código (es decir, las instrucciones del programa) para sus programas sin una fase de planificación inicial. Este enfoque puede funcionar para programas pequeños como los que presentamos en los primeros capítulos de este libro. Pero ¿qué sucedería si le pidieran crear un sistema de software para controlar miles de cajeros automáticos para un banco? ¿O si le pidieran trabajar en un equipo de 1000 desarrolladores de software que construyen la próxima generación del sistema de control de tráfico aéreo estadounidense?

Para crear las mejores soluciones para proyectos tan grandes y complejos, debería seguir un proceso de **análisis** detallado para determinar los **requisitos** de su proyecto; es decir, definir qué se supone *que* debe hacer el sistema. A continuación, desarrollaría un diseño que satisfaga esos requisitos; es decir, especificaría *cómo* debería hacerlo el sistema. Idealmente, antes de escribir cualquier código, debería seguir este proceso y revisar cuidadosamente el diseño, para que lo revisen otros profesionales del software. Si este proceso implica analizar y diseñar su sistema desde una perspectiva orientada a objetos, se denomina **análisis y diseño orientado a objetos** (**OOAD**). La programación en un lenguaje orientado a objetos se denomina **programación orientada a objetos** (**POO**) y permite implementar un diseño orientado a objetos como un sistema operativo.