CAPÍTULO 3 : PROGRAMACIÓN ORIENTADA A OBJETOS

La programación orientada a objetos (OOP) es una metodología de diseño de software y un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que combinan estado (es decir, datos) y comportamiento (esto es, métodos). La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que se comunican entre ellos para realizar tareas. Esto difiere de los lenguajes procedurales tradicionales, en los que los datos y los procedimientos (funciones) están separados y sin relación. Estos métodos están pensados para hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.

3.1. PROPIEDADES DE LA OOP

La programación clásica presenta ciertos problemas, que han ido haciéndose cada vez más graves, a medida que se construían aplicaciones y sistemas informáticos más complejos, entre los que destacan los siguientes:

- Es difícil modificar y extender los programas, pues suele haber datos compartidos por varios subprogramas, que introducen interacciones ocultas entre ellos.

- Es difícil mantener los programas. Casi todos los sistemas informáticos grandes tienen errores ocultos, que no surgen a la luz hasta después de muchas horas de funcionamiento.

- Es difícil reutilizar los programas, es prácticamente imposible aprovechar en una aplicación nueva los subrutinas que se diseñaron para otra.

La OOP trata de encontrar solución a estos problemas mediante los siguientes conceptos:

Abstracción -> Cada objeto en el sistema sirve como modelo de un agente abstracto que puede realizar trabajo, informar y cambiar su estado, y comunicarse con otros objetos en el sistema sin revelar cómo se implementan estas características.

Encapsulación -> Asegura que los objetos no pueden cambiar el estado interno de otros objetos de manera inesperada; sólo los propios métodos internos del objeto pueden acceder a su estado. Cada tipo de objeto expone una interfaz a otros objetos que especifica cómo pueden interactuar con él.

Polimorfismo -> Las referencias y las colecciones de objetos pueden contener objetos de diferentes tipos, y la invocación de un comportamiento en una referencia producirá el comportamiento real para el tipo del referente. Cuando esto ocurre en tiempo de ejecución se denomina "asignación dinámica". Algunos lenguajes, como C++,

proporcionan medios más estáticos de polimorfismo, tales como la sobrecarga de métodos.

Herencia -> Organiza y facilita el polimorfismo y la encapsulación permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir y extender su comportamiento sin tener que reimplementar el propio. Suele hacerse habitualmente agrupando los objetos en clases, y las clases en árboles que reflejan un comportamiento común.

3.2. BENEFICIOS DEL USO DE LA OPP

Día a día los costos del hardware decrecen. Así surgen nuevas áreas de aplicación: procesamiento de imágenes y sonidos, bases de datos multimedia, automatización de oficinas, entornos de ingeniería software, etc.

Lamentablemente, los costos de producción de software siguen aumentando; el mantenimiento y la modificación de sistemas complejos suele ser una tarea complicada; cada aplicación, aunque tenga aspectos similares a otras, suele encarecer un nuevo proyecto, etc.

Todos estos problemas aún no han sido solucionados de forma completa. Sin embargo los objetos son portables, al menos teóricamente, y la herencia permite volver a usar un mismo código, así que es más sencillo modificar código existente al no interaccionar los objetos excepto a través de mensajes. En consecuencia, un cambio en la codificación de un objeto no afecta la operación con otro siempre que los métodos que los relacionan permanezcan intactos.

La introducción de tecnología de objetos como una herramienta conceptual para analizar, diseñar e implementar aplicaciones permite obtener aplicaciones más modificables, fácilmente extensibles y a partir de componentes reutilizables. Esta propiedad de volver a usar el código disminuye el tiempo que se utiliza en el desarrollo y hace que la implementación del software sea más intuitiva ya que la gente piensa de forma natural en términos de objetos más que en términos de algoritmos en software.

3.3. PROBLEMAS ACTUALES

El problema de un sistema orientado a objeto surge en la implementación del mismo. Muchas compañías oyen acerca de los beneficios de un sistema orientado a objetos e invierten gran cantidad de recursos, para darse luego cuenta que han impuesto una “nueva cultura” que es ajena a los programadores actuales. Los problemas principales que suelen aparecer son:

- Curvas de aprendizaje largas. Un sistema orientado a objetos involucra la conceptualización de todos los elementos de un programa, desde subsistemas a datos, en forma de objetos. Toda la comunicación entre los objetos debe realizarse en forma

de mensajes. Esta no es la forma en que están actualmente escritos los programas orientados a objetos, al hacer la transición a un sistema orientado a objetos la mayoría de los programadores deben capacitarse de nuevo antes de usarlo.

- Dependencia del lenguaje. A pesar de la portabilidad conceptual de los objetos en un sistema orientado a objetos, en la práctica existen muchas dependencias. Muchos lenguajes orientados a objetos están compitiendo actualmente para dominar el mercado. Cambiar el lenguaje de implementación de un sistema orientado a objetos no es una tarea sencilla, la elección de un lenguaje tiene ramificaciones de diseño muy importantes.

- Determinación de las clases. Una clase es una especie de "molde" que se utiliza para crear nuevos objetos. En consecuencia es importante crear el conjunto de clases adecuado para un proyecto. Desafortunadamente la definición de las clases es más un arte que una ciencia. Si bien hay muchas jerarquías de clase predefinidas, normalmente se deben crear clases específicas para la aplicación que se está desarrollando. Luego, pasado un determinado tiempo, se da cuenta que las clases que se establecieron no son posibles; en este caso se necesita reestructurar la jerarquía de clases devastando totalmente la planificación original.