TEMA 27

PROGRAMACIÓN ORIENTADA A OBJETOS. OBJETOS. CLASES. HERENCIA. POLIMORFISMO. LENGUAJES.

ÍNDICE

8. BIBLIOGRAFÍA

1. INTRODUCCIÓN

Actualmente, una de las áreas más candentes en la industria y en el ámbito académico es la orientación a objetos. La orientación a objetos promete mejoras de amplio alcance en la forma de diseño, desarrollo y mantenimiento del software, ofreciendo una solución a largo plazo a los problemas y preocupaciones que han existido desde el comienzo en el desarrollo de software: la falta de portabilidad del código y la reusabilidad, códigos difíciles de modificar, ciclos de desarrollo largos y técnicas de codificación no intuitivas.

Un lenguaje orientado a objetos ataca estos problemas. Tiene tres características básicas: debe estar basado en objetos, basado en clases y capaz de tener herencia de clases. Muchos lenguajes cumplen uno o dos de estos puntos; muchos menos cumplen los tres. La barrera más difícil de sortear es, usualmente, la herencia.

El concepto de programación orientada a objetos (OOP) no es nuevo, lenguajes clásicos como Smalltalk se basan en ella. Dado que la OOP se basa en la idea natural de la existencia de un mundo lleno de objetos y que la resolución del problema se realiza en términos de objetos, un lenguaje se dice que está basado en objetos si soporta objetos como una característica fundamental del mismo.

El elemento fundamental de la OOP es, como su nombre lo indica, el **objeto**. Podemos definir un objeto como un conjunto complejo de datos y programas que poseen estructura y forman parte de una organización.

Esta definición especifica varias propiedades importantes de los objetos. En primer lugar, un objeto no es un dato simple, sino que contiene en su interior cierto número de componentes bien estructurados. En segundo lugar, cada objeto no es un ente aislado, sino que forma parte de una organización jerárquica o de otro tipo.

2. OBJETOS

2.1. Estructura de un objeto

Un objeto puede considerarse como una especie de cápsula dividida en tres partes:

Relaciones
Propiedades
Métodos

Cada uno de estos componentes desempeña un papel totalmente independiente.

Las **relaciones** permiten que el objeto se vincule con la organización de la que forma parte, y están formadas esencialmente por **punteros** a otros objetos.

Las **propiedades** distinguen un objeto determinado de los restantes que forman parte de la misma organización y tienen **valores** que dependen de la propiedad de que se trate. Las propiedades de un objeto pueden ser heredadas por sus descendientes en la organización.

Los **métodos** son las operaciones que pueden realizarse sobre el objeto, que normalmente estarán incorporados en forma de programas (**código**) que el objeto es capaz de ejecutar y que también pone a disposición de sus descendientes a través de la herencia.

2.2. Encapsulamiento y ocultación

Como hemos visto, cada objeto es una estructura compleja en cuyo interior hay datos y programas, todos ellos relacionados entre sí, como si estuvieran encerrados conjuntamente en una cápsula. Esta propiedad: **encapsulamiento**, es una de las características fundamentales de la programación orientada a objetos.

El interior de esta cápsula no es visible. Los objetos son inaccesibles, e impiden que otros objetos, los usuarios, o incluso los programadores conozcan cómo está distribuida la información o qué información hay disponible. En esencia, un objeto es una caja negra que no permite distinguir los detalles de lo que hay en su interior. Esta propiedad de los objetos se denomina **ocultación** de la información.

Esto no quiere decir, sin embargo, que sea imposible conocer lo necesario respecto a un objeto y a lo que contiene. De esa forma, no podríamos hacer gran cosa con él. Lo que sucede es que las peticiones de

información a un objeto deben realizarse a través de los cauces adecuados, es decir, mediante mensajes dirigidos a ese objeto, con la orden de realizar la operación pertinente. La respuesta a estas ordenes será la información requerida, siempre que el objeto considere que quien envía el mensaje está autorizado para obtenerla.

El hecho de que cada objeto pueda considerarse como una cápsula, facilita enormemente que un objeto determinado pueda ser transportado a otro punto de la organización, o incluso a otra organización totalmente diferente que precise de él. Si el objeto ha sido bien construido, sus métodos seguirán funcionando en el nuevo entorno sin problemas. Esta cualidad hace que la OOP sea muy apta para la reutilización de programas.

3. ORGANIZACIÓN DE LOS OBJETOS. CLASES

En principio, los objetos forman siempre una organización jerárquica, en el sentido de que ciertos objetos son superiores a otros de cierto modo. Existen varios tipos de jerarquías: serán simples cuando su estructura pueda ser representada por medio de un árbol. En otros casos puede ser más compleja.

En cualquier caso, sea la estructura simple o compleja, podrán distinguirse en ella tres **niveles de objetos**:

- La **raíz de la jerarquía**. Se trata de un objeto que suele ser único y especial (algunos sistemas de OOP permiten multiplicidad de raíces y, por ende, de jerarquías). Se caracteriza por estar situado en el nivel más elevado de la estructura jerárquica, es decir, no existe ningún objeto a un nivel más alto que él. Suele recibir un nombre muy genérico, que indica su categoría especial, básica, distinta de la de todos los demás objetos de la organización, como, por ejemplo, objeto Madre, Raíz o Entidad.

- Los **objetos intermedios**. Son todos aquellos que descienden directa o indirectamente de la raíz y que a su vez tienen descendientes. Representan conjuntos o clases de objetos, que pueden ser muy generales o muy especializados, según las necesidades de la aplicación. Normalmente reciben nombres genéricos que denotan al conjunto de objetos que representan, por ejemplo, Ventana, Cuenta, Fichero... En conjunto reciben el nombre de **clases** o **tipos** si descienden directamente de la raíz y de **subclase** o **subtipos** si descienden de una clase o de una subclase.

  Una segunda clasificación de los objetos intermedios se basa en conocer si sus descendientes son también intermedios, como ellos, o bien son objetos terminales. En el primer caso el objeto se llama **clase abstracta**; en el segundo**, clase no abstracta** o clase de objetos terminales.

- Los **objetos terminales**. Son todos aquellos que descienden de una clase o subclase y no tienen descendientes. Suelen llamarse **casos particulares**, **instancias** o ítems porque representan los elementos del conjunto representado por la clase o subclase a la que pertenecen.

3.1. Clases

Una clase es la descripción de un conjunto de objetos. Consta de métodos y datos que resumen las características comunes de un conjunto de objetos. Muestra el comportamiento general de un grupo de objetos.

Un objeto se crea mediante el envío de un *mensaje de construcción* a la clase.

Análogamente para la destrucción del objeto, la clase recibe un *mensaje de destrucción*.

Un programa orientado a objetos se resume en 3 sucesos:

1. Creación de objetos cuando se necesitan, mediante un mensaje de construcción a la clase.

2. Intercambio de mensajes entre objetos o entre usuario de objeto y objeto (diferencia fundamental entre lenguajes POO puros e híbridos).

3. Eliminar objetos cuando no se necesitan, mediante un mensaje de destrucción a la clase.

3.2. Relaciones

Un objeto está formado por tres componentes principales: relaciones, propiedades y métodos. Las relaciones entre objetos son, precisamente, los enlaces que permiten a un objeto relacionarse con aquellos que forman parte de la misma organización.

No todas las relaciones son equivalentes entre sí. Las hay de dos tipos fundamentales:

- **Relaciones jerárquicas**. Son esenciales para la existencia misma de la aplicación porque la construyen. Estrictamente hablando, existe un solo tipo de organización jerárquica: la relación padre-hijo. Se trata de una relación bidireccional que podríamos definir así:

  Un objeto es padre de otro cuando existe una relación jerárquica directa entre ellos, según la cual el primer objeto se encuentra situado inmediatamente encima del segundo en la organización en la que ambos forman parte. Asimismo, si un objeto es padre de otro, el segundo es hijo del primero.

  Decimos que un objeto A es antepasado de otro B, si A es padre de B o si A es padre de un antepasado de B.
  Decimos que un objeto B es descendiente de otro A, si B es hijo de A o si B es hijo de un descendiente de A.

Una organización jerárquica simple puede definirse como aquella en la que un objeto puede tener un solo padre, mientras que en una organización jerárquica compleja un hijo puede tener varios padres.

- **Relaciones semánticas**. Expresan relaciones entre los objetos que no tienen nada que ver con la organización de la que forman parte. Sus propiedades y consecuencias sólo dependen de los objetos en sí mismos (de su significado) y no de su posición en la organización jerárquica.

Ejemplo: Supongamos que vamos a construir un diccionario informatizado que permita al usuario obtener la definición de una palabra cualquiera. En dicho diccionario, las palabras son objetos y la organización jerárquica es la que proviene de forma natural de la estructura de nuestros conocimientos sobre el mundo.

La raíz del diccionario podría llamarse TEMAS. De éste término genérico descenderán tres grandes ramas de objetos llamadas VIDA, MUNDO y HOMBRE. El primero (vida) comprenderá las ciencias biológicas: Biología y Medicina. El segundo (mundo), las ciencias de la naturaleza inerte: las Matemáticas, la Física, la Química y la Geología. El tercero (hombre) comprenderá las ciencias humanas: la Geografía, la Historia, etc.

Estableceremos la relación trabajo entre los objetos NEWTON y OPTICA y la interpretaremos diciendo que significa que Newton trabajó en óptica. La relación es, evidentemente, semántica, pues no establece ninguna connotación jerárquica entre NEWTON y OPTICA y su interpretación depende exclusivamente del significado de ambos objetos.

La existencia de esta relación nos permitirá responder a preguntas como:

¿Quién trabajó en óptica?
¿En qué trabajó Newton?
¿Quién trabajó en Física?

Las dos primeras se deducen inmediatamente de la existencia de la relación trabajo. Para la tercera observamos que si Newton trabajó en óptica automáticamente sabemos que trabajó en Física, por ser óptica una rama de la Física (en nuestro diccionario, el objeto ÓPTICA es hijo del objeto FÍSICA). Entonces gracias a la OOP podemos responder a la tercera pregunta sin necesidad de establecer una relación entre NEWTON y FÍSICA, apoyándonos sólo en la relación definida entre NEWTON y ÓPTICA y en que ÓPTICA es hijo de FÍSICA. De este modo se elimina toda redundancia innecesaria y la cantidad de información que tendremos que definir para todo el diccionario será mínima.

3.3. Propiedades

Todo objeto puede tener cierto número de propiedades, cada una de las cuales tendrá, a su vez, uno o varios valores. En programación orientada a objetos, las propiedades corresponden a las clásicas variables de la programación clásica. Son, por tanto, datos encapsulados dentro del objeto, junto con los métodos (programas) y las relaciones (punteros a otros objetos).

Las propiedades de un objeto pueden tener un valor único o pueden contener un conjunto de valores más o menos estructurados (matrices, vectores, listas, etc.). Además, los valores pueden ser de cualquier tipo (numérico, alfabético, etc.) si el sistema de programación lo permite.

Pero existe una diferencia con las variables, y es que las propiedades se pueden heredar de unos objetos a otros. En consecuencia, un objeto puede tener una propiedad de dos maneras diferentes:

- **Propiedades propias**. Están definidas dentro de la cápsula del objeto.

- **Propiedades heredadas**. Están definidas en un objeto diferente, antepasado de éste (padre, abuelo, etc.). A veces estas propiedades se llaman propiedades miembro porque el objeto las posee por el mero hecho de ser miembro de una clase de la organización.

3.4. Métodos

Podemos definir un **método** como un programa procedimental escrito en cualquier lenguaje, que está asociado a un objeto determinado y cuya ejecución sólo puede desencadenarse a través de un mensaje recibido por éste o por sus descendientes.

Son sinónimos de método todos aquellos términos que se han aplicado tradicionalmente a los programas, como procedimiento, función, rutina, etc. Sin embargo, es conveniente utilizar el término método para que se distingan claramente las propiedades especiales que adquiere un programa en el entorno OOP, que afectan fundamentalmente a la forma de invocarlo (únicamente a través de un mensaje) y a su campo de acción, limitado a un objeto y a sus descendientes, aunque posiblemente no a todos.

Si los métodos son programas, se deduce que podrán tener argumentos o parámetros. **Ejemplo**: Considerando el método “suma”, aplicable a la clase de los números enteros. Uno de estos números enteros puede ser el 3. En efecto, un número como el 3 puede considerarse como uno de los objetos terminales en un sistema de OOP y la suma es, evidentemente, una de las operaciones que se pueden realizar con él. Pero también es obvio que para realizar dicha operación necesitaremos un argumento: el valor que queremos sumarle al 3, por ejemplo, el 5. En un sistema de OOP, la expresión 3+5 puede considerarse como un mensaje que se envía al objeto 3 para realizar la operación sum, con argumento 5.

Puesto que los métodos pueden heredarse de unos objetos a otros, un objeto puede disponer de un método de dos maneras diferentes:

- **Métodos propios**. Están definidos dentro de la cápsula del objeto.

- **Métodos heredados**. Están definidos en un objeto diferente, antepasado de éste (padre, abuelo, etc.). A veces estos métodos se llaman métodos miembro porque el objeto los posee por el mero hecho de ser miembro de una clase de la organización.

3.4.1. Polimorfismo

Una de las características fundamentales de la OOP es el polimorfismo, que no es otra cosa que la posibilidad de construir varios métodos diferentes con el mismo nombre. Esto es una consecuencia inmediata del hecho de que los métodos son operaciones propias de los objetos, por lo que dos objetos diferentes podrán tener operaciones, también diferentes en su actuación, expresadas con el mismo nombre. Esto conlleva la habilidad de enviar un mismo mensaje a objetos de clases diferentes. Estos objetos recibirían el mismo mensaje global pero responderían a él de formas diferentes; por ejemplo, un mensaje "+" a un objeto ENTERO significaría suma, mientras que para un objeto STRING significaría concatenación ("pegar" strings uno seguido con otro).

Polimorfismo es invocar métodos distintos con el mismo mensaje (ligadura en tiempo de ejecución).

Para ello es necesaria una jerarquía de herencia: una clase base que contenga un método polimórfico, que es redefinido en las clases derivadas (no anulado).

Se permite que los métodos de los hijos puedan ser invocados mediante un mensaje que se envía al padre.

Este tipo de clase que se usa para implementar el polimorfismo se conoce como **clase abstracta**

3.4.2. Demonios

Es un tipo especial de métodos, relativamente poco frecuente en los sistemas de OOP, que se activa automáticamente cuando sucede algo especial. Es decir, es un programa, como los métodos ordinarios, pero se diferencia de estos porque su ejecución no se activa con un mensaje, sino que se desencadena automáticamente cuando ocurre un suceso determinado: la asignación de un valor a una propiedad de un objeto, la lectura de un valor determinado, etc.

Los demonios, cuando existen, se diferencian de otros métodos por que no son heredables y porque a veces están ligados a una de las propiedades de un objeto, mas que al objeto entero.

4. HERENCIA

Una de las características fundamentales de la programación dirigida a objetos es la transmisión de las propiedades y los programas de unos objetos a otros a través de la organización jerárquica a la que pertenecen. Es precisamente esta propiedad la que permite que sea fácil volver a utilizar desarrollos anteriores en aplicaciones completamente nuevas, lo que simplifica y reduce extraordinariamente el esfuerzo necesario para construirlas.

Herencia es la capacidad de un objeto (clase) para utilizar las estructuras y los métodos existentes en antepasados o ascendientes.

Es la reutilización de código desarrollado anteriormente.

Cuando usamos herencia decimos que hacemos *programación por herencia*: Definición de nuevos tipos a partir de otros con los que comparten algún tipo de característica.

La herencia es *unidireccional*, pues un objeto puede heredar de uno de sus antepasados, pero nunca de sus descendientes. En otras palabras, un hijo puede heredar de su padre, pero un padre no puede heredar de su hijo.

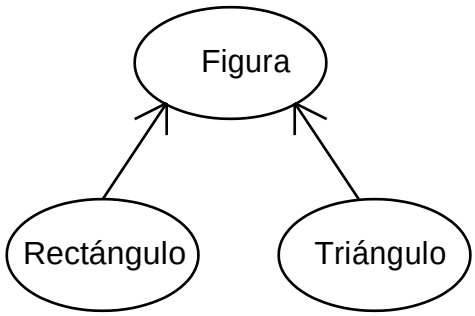
La herencia afecta tanto a propiedades como a métodos, pero sólo se heredarán los métodos y/o propiedades que interesen. Los métodos y/o propiedades heredados no aparecerán en la descripción del objeto hijo (subclase), estarán contenidos de forma implícita.

Con la herencia no se necesita repetir código para todas las subclases de una misma clase, y se garantiza un comportamiento homogéneo para todas las subclases que tienen un mismo padre.

Si en una subclase interesa heredar una propiedad con distinto valor y/o tipo al del padre, bastará que aparezca en su definición con el nuevo valor y/o tipo. Esta nueva definición afectará a todos los posibles hijos de la subclase. Cuando esta situación afecta a un método, habrá que escribir un nuevo código para el método en el objeto hijo; dando lugar al polimorfismo cuando este cambio afecta a los parámetros y/o valor de retorno del método.

4.1. Herencia simple

Se aplica cuando el objeto que recibe la herencia tiene un solo padre. Un tipo derivado se crea a partir de una única clase base. Esto ocurre siempre que la organización jerárquica de los objetos sea simple, o también en aquellos objetos que perteneciendo a jerarquías múltiples tengan un solo padre y sus antepasados tengan también un padre único.



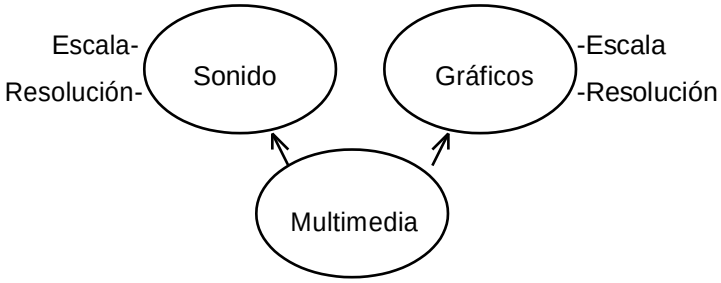
4.3. Herencia múltiple

Se aplica únicamente en las jerarquías complejas a aquellos objetos que tengan más de un padre o, aunque no tengan más que uno, alguno de sus antepasados tenga varios padres. Una clase tiene más de una ascendiente inmediato.

La herencia múltiple puede plantear 2 tipos de problemas:

1. La *herencia repetida*: ej.: profesor universitario hereda 2 veces los atributos de persona.

2. Produce *ambigüedad* respecto a los atributos o los métodos. En la clase base pueden haber atributos que se llamen igual.



5. MENSAJES

La programación orientada a objetos no se realiza a base de llamadas a subrutinas, sino de mensajes entre los objetos.

Los objetos se comunican mediante llamadas a funciones miembro o **métodos**: se dice entonces que *un objeto envía un mensaje a otro objeto.*

Un mensaje es una petición enviada por un objeto a otro objeto para que active un método. La estructura general de un mensaje es la siguiente:

Objeto método [parámetros]

donde

Objeto es el nombre del objeto al que se dirige el mensaje.

método es el nombre del método que se desea ejecutar, entre los que pertenecen al objeto que recibe el mensaje. El método puede ser propio del objeto (definido en él) o heredado de uno de sus antepasados.

parámetros son los argumentos del método.

En OOP, los mensajes son equivalentes a las llamadas de subrutinas en la programación procedimental. La llamada a una subrutina se expresaba:

subrutina (parámetros)

Observamos que los mensajes son muy parecidos, pues el nombre del método corresponde al de la subrutina, mientras que los parámetros ocupan una posición equivalente. Sin embargo, en los mensajes tenemos un elemento adicional y fundamental, el nombre del objeto que recibe el mensaje.

El nombre del objeto que envía el mensaje no aparece en dicho mensaje. Esto se debe a que, a causa del encapsulamiento, la ejecución del mensaje sólo depende del objeto que lo recibe, nunca del que lo envía. Por tanto, su nombre no es necesario y se omite. De hecho, el mensaje no tiene por qué originarse por un objeto. A veces, el usuario de la aplicación puede lanzarlo desde una unidad de entrada/salida. O bien, podría llegar de una máquina diferente.

Estrictamente hablando, no son los objetos quienes envían mensajes a otros objetos para que ejecuten métodos. Los mensajes de originan siempre (excepto en los casos mencionados) en un método. Esto es lógico, pues los métodos forman el componente activo del objeto, su conducta, en contraposición a sus propiedades o datos, que forman la componente pasiva.

5.1. Ligamiento dinámico

Es una de las características fundamentales de la OOP, que podemos definir así: Capacidad de retrasar hasta el instante de la recepción del mensaje la decisión sobre la clase de objeto que lo recibe y el método concreto que debe ejecutarse.

El ligamiento dinámico es el que da la capacidad a la programación dirigida a objetos para buscar en las jerarquías los métodos y propiedades heredados cuando un objeto ha recibido un mensaje.

El código correspondiente a una función (método en OOP) se asociará en tiempo de ejecución y no en tiempo de compilación. Afecta también a las variables (propiedades en OOP), cuya dirección se busca en tiempo de ejecución en lugar de quedar definida en tiempo de compilación. Estas búsquedas se realizarán en función del objeto receptor del mensaje.

Sin ligamiento dinámico no serían posibles la herencia y el polimorfismo.

6. LENGUAJES DE LA OOP

Podemos clasificarlos en dos grandes grupos: los que sólo permiten realizar programación orientada a objetos (lenguajes puros) y los que permiten mezclarla con la programación procedimental clásica (lenguajes híbridos).

6.1. Lenguajes puros

Se dice que un lenguaje de programación orientada a objetos es puro si no permite salirse del entorno de la OOP para realizar programación clásica. En ellos, cualquier relación entre un programa y otros diferentes debe realizarse a través de un mensaje dirigido a un objeto. En estos lenguajes suele predominar el concepto de la elegancia frente a otras consideraciones, como podrían ser la eficiencia o la flexibilidad del programador para el manejo de los objetos.

Smalltalk

No existen datos que no sean objetos, y no hay otra forma de manejarlos más que a través de mensajes enviados al objeto correspondiente. Además, los objetos ocultan su información, que sólo se puede obtener por los cauces apropiados (encapsulamiento).

La mayor parte de los sistemas Smalltalk son intérpretes. Como en casi todos los sistemas de este tipo, al usuario se le proporciona un espacio de trabajo en la memoria principal, donde se almacenan los objetos junto con sus métodos. Este espacio de trabajo puede, si se desea, guardarse en disco tal y como está, es decir, el fichero correspondiente será una imagen del contenido de la memoria principal, por lo que podrá restaurarse posteriormente mediante los mandatos adecuados.

Características:

- Todas las entidades que maneja **Smalltalk** son objetos.
- Todas las clases derivan de una clase base llamada Object.
- Existe herencia simple.
- Usa métodos y paso de mensajes.
- Tiene una tipificación débil: la comprobación de los tipos se hace en tiempo de ejecución.
- Soporta concurrencia, pero pobre.
- Se comercializa con un conjunto de bibliotecas de clases predefinidas, agrupadas en categorías o jerarquías (E/S, etc...)
- Existe una versión, Smalltalk V para computadoras IBM.
- C++ y CLOS se han apoyado en características de Smalltalk

Actor

Los métodos y las clases predefinidas de Actor están principalmente orientados a realizar las funciones que un programador en lenguaje C debería utilizar para crear algún objeto en el entorno Windows. La diferencia principal entre una función del C relacionada con Windows y un mensaje de Actor es que este último oculta muchos de los detalles de bajo nivel. Con ello se consigue una reducción del código y una gran simplificación conceptual a la hora de manejar los objetos de Windows.

6.2. Lenguajes híbridos

Se dice que un lenguaje de la programación dirigida a objetos es híbrido si permite salirse del entorno y de la forma de programar OOP para realizar programación procedimental. Es decir, permite el uso de la instrucción CALL u otras equivalentes. En estos lenguajes es teóricamente posible mezclar las dos formas de programar en proporciones cualesquiera, e incluso prescindir por completo de la programación orientada a objetos, para realizar programación clásica. En estos lenguajes suelen predominar consideraciones como la eficiencia o la flexibilidad del programador sobre la elegancia del diseño.

Algunos lenguajes de este tipo son:

- **C++ [1986]**
  Diseñado por **Stroustrup**, es un híbrido basado en **C** y **Smalltalk**. Ha tenido mucho éxito.
- **Object Pascal [1987]**
  Extensión de **Pascal**, lo popularizó **Borland** con **Turbo Pascal 5.5**
- **Object COBOL [1992-3]**
  Extensión de **COBOL**, nuevo LPOO que parece haber tenido aceptación.
- **Delphi [1995]**
  Proviene de **Object Pascal**, creado por **Borland**.

C++

Representa el resultado de los esfuerzos realizados para proporcionar las ventajas de la programación orientada a objetos a un lenguaje clásico, muy extendido en la programación de sistemas. Se trata de una extensión del lenguaje C que representa claras influencias del lenguaje Simula, que a su vez puede considerarse como el precursor de los lenguajes OOP.

Clases en C++

El lenguaje C++ entiende por clase a un tipo de datos definido por el programador que tiene toda la información necesaria para construir un objeto de dicho tipo y el conjunto de operaciones que permiten manejarlo (métodos). Las clases de C++ le permiten disfrutar de los beneficios de la modularidad, de la misma manera que en lenguajes como Ada o Modula-2.

El acceso a cada uno de los miembros (atributos o métodos) de una clase o estructura puede pertenecer a uno de los siguientes niveles:

- **Público** ("public"). Un miembro público puede utilizarse en cualquier lugar donde se tenga acceso a la clase.
- **Privado** ("private"). Un miembro privado sólo puede utilizarse en métodos declarados dentro de la misma clase.
- **Protegido** ("protected"). Un miembro protegido puede utilizarse en métodos declarados dentro de la misma clase y en métodos de clases descendientes de ella.

Constructores y destructores

Una vez definida una clase y los métodos asociados a ella, la siguiente cuestión que se plantea es cómo crear y destruir casos particulares u objetos pertenecientes a ella. El lenguaje C++ incorpora dos métodos especiales que pueden utilizarse con todas las clases definidas por el programador, que se encargan de la creación (e iniciación) y de la destrucción de los objetos. Estos métodos son creados automáticamente por el compilador, si el programador no los especifica.

Herencia

El lenguaje C++ permite la herencia simple y múltiple. La herencia se establece explícitamente durante la declaración de la clase, especificando cuál es la superclase ("clase paterna") de la que estamos creando.

Funciones virtuales

Entre los elementos de C++ que hemos descrito hasta el momento, nos falta mencionar la manera de conseguir algún tipo de ligamiento dinámico. El compilador de C++ tiene tres formas principales de resolver la localización del destinatario de un método:

- Detectando el tipo de argumento utilizado para llamar a una función polimorfa.
- Anteponiendo al nombre del método, o miembro funcional, el de la clase a la que pertenece seguido del operador ::.
- Lanzando un mensaje hacia un objeto utilizando la sintaxis

  nombre_del_objeto.nombre_del_método(argumentos)

Turbo Pascal

La incorporación a Turbo Pascal de elementos de programación orientada a objetos se hizo paralelamente a la aparición del sistema Turbo C++. Como ambos compiladores pertenecen a la empresa Borland, presentan un alto grado de paralelismo como lenguajes de OOP.

Definición de clases

Turbo Pascal utiliza internamente una estructura para almacenar la información relativa al objeto e incorpora un nuevo identificador ("object") que permite especificar que dicha estructura define una clase. Este identificador corresponde a "class" en C++. En dicha estructura se almacenan los atributos y los nombres de los métodos que actúan sobre esta clase de objetos. En Turbo Pascal se pueden definir como métodos tanto funciones como procedimientos.

Herencia

La herencia es otro de los conceptos de OOP que incorpora el Turbo Pascal y se establece en la declaración de las clases de los objetos, especificando que una clase es descendiente de otra. Con ello se heredan automáticamente los atributos y métodos del antecesor. Esto no impide que se puedan redefinir los métodos de un antecesor en uno de sus descendientes, si fuera necesario que dicho método tuviera un comportamiento diferente para esa clase.

Encapsulamiento

No existen todos los niveles de acceso que en C++ permiten al programador un alto grado de control sobre la visibilidad del objeto ante sus descendientes. El único modificador de acceso que puede establecer un programador sobre alguna característica de una clase es "private", que determina que dicha característica no sea visible desde fuera del objeto.

Constructores y destructores

En Turbo Pascal existen también los métodos constructores y destructores que garantizan la inicialización de los objetos y la liberación correcta de la memoria cuando dejan de ser utilizados. Su diferencia con C++ es que deben ser declarados por el programador, ya que el compilador no genera una visión por defecto de los mismos. Estos métodos se declaran con los modificadores especiales "Init" (inicializar) y "Done" (terminado).

Funciones virtuales

Tiene el mismo tipo de ligamiento dinámico que C++. Para utilizarlo es necesario que se defina como virtual el método sobre el que se quiere establecer el ligamiento dinámico en la clase que lo declare, así como en todos sus descendientes, con lo que se crea una tabla de métodos virtuales para dicha clase. Esta tabla contiene información relativa a la dirección del procedimiento o función particular que ejecuta cada uno de los métodos virtuales de la clase.

6.3. Comparación entre lenguajes OOP puros e híbridos:

	Puros	***Híbridos***
Ventajas	Más potencia. Flexibilidad para modificar el lenguaje. Más fácil de asimilar (enseñar).	Más rápido. Más fácil el paso de programación estructurada a POO. Implementación de operaciones-algoritmos fundamentales más sencilla.
Inconvenientes	Más lento. Implementación de operaciones-algoritmos fundamentales muy complicada.	Trabaja con 2 paradigmas a la vez. Modificar el lenguaje es difícil.

7. CONSIDERACIONES FINALES

7.1. Beneficios que se obtienen del desarrollo con OOP

Día a día los costos del Hardware decrecen. Así surgen nuevas áreas de aplicación cotidianamente: procesamiento de imágenes y sonido, bases de datos multimedia, automatización de oficinas, ambientes de ingeniería de software, etc. Aún en las aplicaciones tradicionales encontramos que definir interfaces hombre-máquina tipo Windows suele ser bastante conveniente.

Lamentablemente, los costos de producción de software siguen aumentando; el mantenimiento y la modificación de sistemas complejos suele ser una tarea trabajosa; cada aplicación, (aunque tenga aspectos similares a otra) suele encararse como un proyecto nuevo, etc.

Todos estos problemas aún no han sido solucionados en forma completa. Pero como los objetos son portables (teóricamente) mientras que la herencia permite la reusabilidad del código orientado a objetos, es más sencillo modificar código existente porque los objetos no interaccionan excepto a través de mensajes; en consecuencia un cambio en la codificación de un objeto no afectará la operación con otro objeto siempre que los métodos respectivos 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 reusables. Esta reusabilidad del código disminuye el tiempo que se utiliza en el desarrollo de software y hace que dicho desarrollo sea más intuitivo porque la gente piensa naturalmente en términos de objetos más que en términos de algoritmos de software.

7.2. Problemas derivados de la utilización de OOP en la actualidad

Un sistema orientado a objetos, por lo visto, puede parecer un "paraíso virtual". El problema, sin embargo, surge en la implementación de tal sistema. Muchas compañías perciben los beneficios de un sistema orientado a objetos e invierten gran cantidad de recursos en él, luego comienzan a darse cuenta que han impuesto una nueva cultura que es ajena a los programadores actuales. Específicamente los siguientes temas suelen aparecer repetidamente:

Curvas de aprendizaje largas. Un sistema orientado a objetos ve al mundo en una forma única. Involucra la conceptualización de todos los elementos de un programa, desde subsistemas a los datos, en la forma de objetos. Toda la comunicación entre los objetos debe realizarse en la forma de mensajes. Esta no es la forma en que están escritos los programas orientados a objetos actualmente; al hacer la transición a un sistema orientado a objetos la mayoría de los programadores deben capacitarse nuevamente antes de poder 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; por ejemplo C++ soporta el concepto de herencia múltiple mientras que Smalltalk no lo soporta; en consecuencia la elección de un lenguaje tiene ramificaciones de diseño muy importantes.

Determinación de las clases. Una clase es un molde que se utiliza para crear nuevos objetos. En consecuencia es importante crear el conjunto de clases adecuado para un proyecto. Desgraciadamente la definición de las clases es más un arte que una ciencia. Si bien hay muchas jerarquías de clase predefinidas usualmente se deben crear clases específicas para la aplicación que se este desarrollando. Luego, en 6 meses ó 1 año nos damos cuenta de que las clases que se establecieron no son posibles; en ese caso será necesario reestructurar la jerarquía de clases devastando totalmente la planificación original.

Rendimiento. En un sistema donde todo es un objeto y toda interacción es a través de mensajes, el tráfico de mensajes afecta al rendimiento. A medida que la tecnología avanza y la velocidad de microprocesamiento, potencia y tamaño de la memoria aumentan, la situación mejorará; pero en la situación actual, un diseño de una aplicación orientada a objetos que no tiene en cuenta el rendimiento no será viable comercialmente.

Idealmente, habría una forma de atacar estos problemas eficientemente al mismo tiempo que se obtienen los beneficios del desarrollo de una estrategia orientada a objetos. Debería existir una metodología fácil de aprender e independiente del lenguaje, y fácil de reestructurar que no drene el rendimiento del sistema

8. BIBLIOGRAFÍA

Alfonseca, M.
Programación orientada a objetos
Anaya Multimedia, 1992

O'Brien, Stephen
Turbo Pascal 6
Mc Graw-Hill, 1991

Atkinson, Lee
Programación en Borland C++
Anaya Multimedia, 1992