Introducción a la Programación Orientada a Objetos

Luis R. Izquierdo

1 IMPORTANTE NOTA PRELIMINAR

Este documento es un apéndice de mi proyecto fin de carrera. Lo escribí después de leer tres o cuatro libros sobre el tema y consultar algunas páginas de Internet. Lo cierto es que, sinceramente, no recuerdo las fuentes que utilicé, así que me temo que, por mucho que me gustaría, me es imposible citarlas. No obstante, los conceptos que se presentan en este documento pueden encontrarse en cualquier libro de programación orientada a objetos, con definiciones probablemente mucho más rigurosas que las que yo aquí presento. Dejo este documento de forma libre en la red con la esperanza de que pueda ser útil, y sin ningún ánimo de atribuirme méritos que no me corresponden. Mi única intención es que, leyendo este breve documento, el lector pueda aprender los conceptos básicos de la programación orientada a objetos y disfrutar al mismo tiempo.

2 INTRODUCCIÓN

Es importante aclarar desde un principio la diferencia que existe entre programación orientada a objetos y un lenguaje orientado a objetos.

La programación orientada a objetos es una *"filosofía",* un modelo de programación, con su teoría y su metodología, que conviene conocer y estudiar antes de nada. Un lenguaje orientado a objetos es un lenguaje de programación que permite el diseño de aplicaciones orientadas a objetos. Dicho esto, lo normal es que toda persona que vaya a desarrollar aplicaciones orientadas a objetos aprenda primero la "filosofía" (o adquiera la forma de pensar) y después el lenguaje, porque "filosofía" sólo hay una y lenguajes muchos. En este documento veremos brevemente los conceptos básicos de la programación orientada a objetos desde un punto de vista global, sin particularizar para ningún lenguaje de programación específico.

3 UNA FORMA NUEVA DE PENSAR

Es muy importante destacar que cuando hacemos referencia a la programación orientada a objetos no estamos hablando de unas cuantas características nuevas añadidas a un lenguaje de programación. Estamos hablando de una nueva forma de pensar acerca del proceso de descomposición de problemas y de desarrollo de soluciones de programación.

La programación orientada a objetos surge en la historia como un intento para dominar la complejidad que, de forma innata, posee el software. Tradicionalmente, la forma de enfrentarse a esta complejidad ha sido empleando lo que llamamos programación estructurada, que consiste en descomponer el problema objeto de resolución en subproblemas y más subproblemas hasta llegar a acciones muy simples y fáciles de codificar. Se trata de descomponer el problema en acciones, en verbos. En el ejemplo de un programa que resuelva ecuaciones de segundo grado, descomponíamos el problema en las siguientes *acciones*: primero, *pedir* el valor de los coeficientes a, b y c; después, *calcular* el valor del discriminante; y por último, en función del signo del discriminante, *calcular* ninguna, una o dos raíces.

Como podemos ver, descomponíamos el problema en acciones, en verbos; por ejemplo el verbo pedir, el verbo hallar, el verbo comprobar, el verbo calcular…

La programación orientada a objetos es otra forma de descomponer problemas. Este nuevo método de descomposición es *la descomposición en objetos*; vamos a fijarnos no en lo que hay que hacer en el problema, sino en cuál es el escenario real del mismo, y vamos a intentar simular ese escenario en nuestro programa.

Los lenguajes de programación tradicionales no orientados a objetos, como C, Pascal, BASIC, o Modula-2, basan su funcionamiento en el concepto de *procedimiento* o *función*. Una función es simplemente un conjunto de instrucciones que operan sobre unos argumentos y producen un resultado. De este modo, un programa no es más que una sucesión de llamadas a funciones, ya sean éstas del sistema operativo, proporcionadas por el propio lenguaje, o desarrolladas por el mismo usuario.

En el caso de los lenguajes orientados a objetos, como es el caso de C++ y Java, el elemento básico no es la función, sino un ente denominado precisamente *objeto*. Un objeto es la representación en un programa de un *concepto,* y contiene toda la información necesaria para abstraerlo: datos que describen sus atributos y operaciones que pueden realizarse sobre los mismos.

La programación orientada a objetos es una nueva forma de pensar, una manera distinta de enfocar los problemas. Ahí radica la dificultad de aprender un lenguaje totalmente orientado a objetos, como es Java, sin conocer previamente los pilares de la programación orientada a objetos. Hecha esta importante aclaración, conviene destacar que Java, más que un lenguaje *orientado a* objetos, es un lenguaje *de* objetos. Java incorpora el uso de la orientación a objetos como uno de los pilares básicos y fundamentales del lenguaje. Esto constituye una importante diferencia con respecto a C++. C++ está pensado para su utilización como lenguaje orientado a objetos, pero también es cierto que con C++ se puede escribir código sin haber oído nada de la programación orientada a objetos. Esta situación no se da en Java, dotado desde las primeras etapas de su diseño de esta filosofía, y donde no cabe obviar la orientación a objetos para el desarrollo de programas, por sencillos que éstos sean. Al contrario que en C++, en Java nada se puede hacer sin usar al menos un objeto.

4 UN PRIMER EJEMPLO

Si nos detenemos a pensar sobre cómo se nos plantea un problema cualquiera en la realidad podremos ver que lo que hay en la realidad son entidades (otros nombres que podríamos usar para describir lo que aquí llamo entidades son “agentes” u “objetos”). Estas entidades poseen un conjunto de propiedades o *atributos,* y un conjunto de *métodos* mediante los cuales muestran su *comportamiento.* Y no sólo eso, también podremos descubrir, a poco que nos fijemos, todo un conjunto de *interrelaciones* entre las entidades, guiadas por el intercambio de *mensajes*; las entidades del problema responden a estos mensajes mediante la ejecución de ciertas *acciones.* El siguiente ejemplo, aunque pueda parecer un poco extraño, creo que aclarará algunos conceptos y nos servirá como introducción para desarrollarlos con profundidad.

Imaginemos la siguiente situación: un domingo por la tarde estoy en casa viendo la televisión, y de repente mi madre siente un fuerte dolor de cabeza; como es natural, lo primero que hago es tratar de encontrar una caja de aspirinas.

Lo que acabo de describir es una situación que probablemente no resulte muy extraña a muchos de nosotros. Vamos a verla en clave de objetos: el objeto hijo ha recibido un mensaje procedente del objeto madre. El objeto hijo responde al mensaje o evento ocurrido mediante una acción: buscar aspirinas. La madre no tiene que decirle al hijo dónde debe buscar, es responsabilidad del hijo resolver el problema como considere más oportuno. Al objeto madre le basta con haber emitido un mensaje. Continuemos con la historia.

El hijo no encuentra aspirinas en el botiquín y decide acudir a la farmacia de guardia más cercana para comprar aspirinas. En la farmacia es atendido por una señorita que le pregunta qué desea, a lo que el hijo responde: "una caja de aspirinas, por favor". La farmacéutica desaparece para regresar al poco tiempo con una caja de aspirinas en la mano. El hijo paga el importe, se despide y vuelve a su casa. Allí le da un comprimido a su madre, la cual al cabo de un rato comienza a experimentar una notable mejoría hasta la completa desaparición del dolor de cabeza.

El hijo, como objeto responsable de un cometido, sabe lo que debe hacer hasta conseguir una aspirina. Para ello entra en relación con un nuevo objeto, la farmacéutica, quien responde al mensaje o evento de petición del objeto hijo con la búsqueda de la aspirina. El objeto farmacéutica es ahora el responsable de la búsqueda de la aspirina. El objeto farmacéutica lanza un mensaje al objeto hijo solicitando el pago del importe, y el objeto hijo responde a tal evento con la acción de pagar.

Como hemos podido ver, en esta situación nos hemos encontrado con objetos que se diferenciaban de los demás por un conjunto de características o propiedades, y por un conjunto de acciones que realizaban en respuesta a unos eventos que se originaban en otros objetos o en el entorno.

También podemos darnos cuenta de que, aunque todos los objetos tienen propiedades distintas, como el color del cabello, el grado de simpatía o el peso, todos tienen un

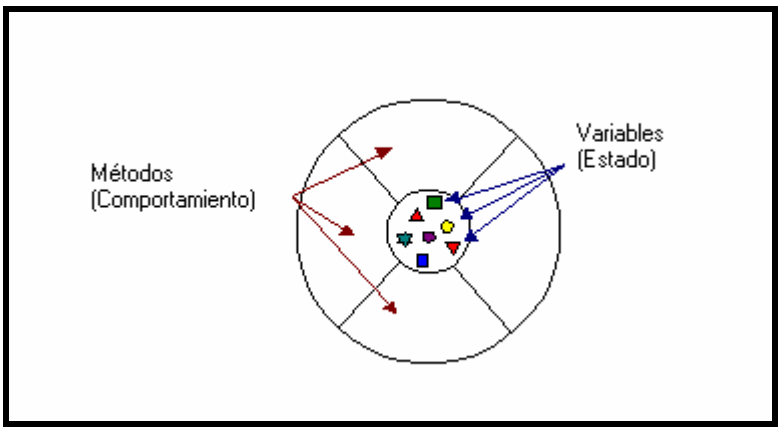
conjunto de atributos en común por ser ejemplos de una entidad superior llamada “ser humano”. A este patrón de objetos (en nuestro caso “ser humano”) lo llamaremos *clase*.

Con este ejemplo espero que se entienda que los objetos son instancias o casos concretos de las clases, que no son más que plantillas que definen las variables y los métodos comunes a todos los objetos de un cierto tipo. La clase “ser humano” tendrá, entre sus muchas *variables miembro* o variables que la componen: color del cabello, color de los ojos, estatura, peso, fecha de nacimiento, etc. A partir de una clase se podrán generar todos los objetos que se deseen especificando valores particulares para cada una de las variables definida por la clase. Así, encontraremos el objeto farmacéutica, cuyo color de cabello es rubio, color de ojos azul, estatura 175 cm., peso 50 Kg., y así sucesivamente.

A continuación veremos con más detalle qué son los objetos, las clases, los mensajes y otros conceptos básicos de la programación orientada a objetos.

5 ¿QUÉ ES UN OBJETO?

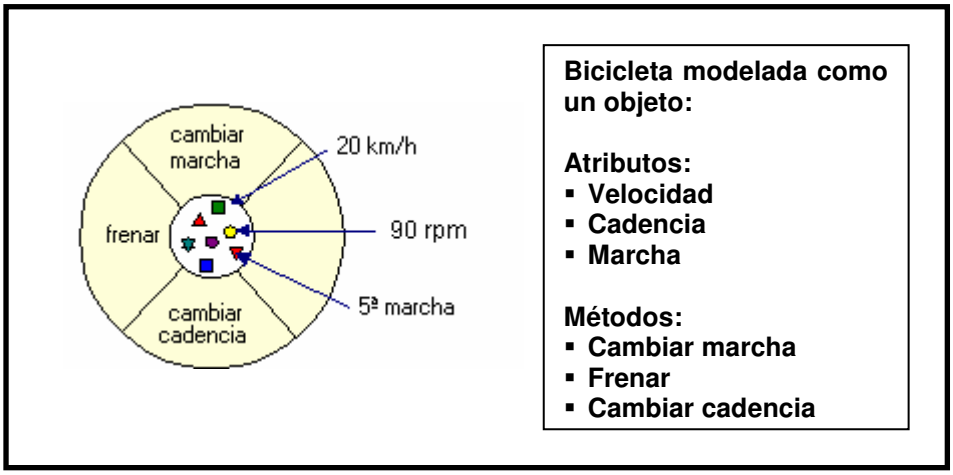
Un objeto no es más que un conjunto de variables (o datos) y métodos (o funciones) relacionados entre sí. Los objetos en programación se usan para modelar objetos o entidades del mundo real (el objeto hijo, madre, o farmacéutica, por ejemplo). Un objeto es, por tanto, la representación en un programa de un concepto, y contiene toda la información necesaria para abstraerlo: datos que describen sus atributos y operaciones que pueden realizarse sobre los mismos. La siguiente figura muestra una representación visual de un objeto.



Representación visual de un objeto

Los atributos del objeto (estado) y lo que el objeto puede hacer (comportamiento) están expresados por las variables y los métodos que componen el objeto respectivamente. Por ejemplo, un objeto que modelase una bicicleta en el mundo real tendría variables que indicararían el estado actual de la bicicleta: su velocidad es de 20 km/h, su cadencia de pedaleo 90 r.p.m. y su marcha actual es la 5ª. Estas variables se conocen formalmente como *variables instancia* o *variables miembro* porque contienen el estado de un objeto bicicleta particular y, en programación orientada a objetos, un objeto particular se denomina una instancia.

Además de estas variables, el objeto bicicleta podría tener métodos para frenar, cambiar la cadencia de pedaleo, y cambiar de marcha (la bicicleta no tendría que tener un método para cambiar su velocidad pues ésta es función de la cadencia de pedaleo, la marcha en la que está y de si los frenos están siendo utilizados o no, entre otros muchos factores). Estos métodos se denominan formalmente *métodos instancia* o *métodos miembro*, ya que cambian el estado de una instancia u objeto bicicleta particular. La siguiente figura muestra una bicicleta modelada como un objeto:



El diagrama del objeto bicicleta muestra las variables objeto en el núcleo o centro del objeto y los métodos rodeando el núcleo y protegiéndolo de otros objetos del programa. Este hecho de empaquetar o proteger las variables miembro con los métodos miembro se denomina **encapsulación**. Este dibujo conceptual que muestra el núcleo de variables miembro del objeto protegido por una membrana protectora de métodos o funciones miembro es la representación ideal de un objeto y es el ideal que los programadores de

objetos suelen buscar. Sin embargo, debemos matizarlo. A menudo, por razones prácticas, es posible que un objeto desee exponer alguna de sus variables miembro, o proteger otras de sus propios métodos o funciones miembro. Por ejemplo, Java permite establecer 4 niveles de protección de las variables y de las funciones miembro para casos como éste. Los niveles de protección determinan qué objetos y clases pueden acceder a qué variables o a qué métodos.

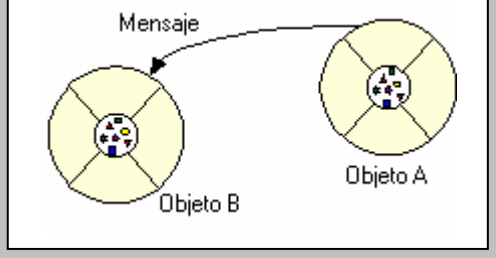
De cualquier forma, el hecho de encapsular las variables y las funciones miembro relacionadas proporciona dos importantes beneficios a los programadores de aplicaciones:

- **Capacidad de crear módulos:** El código fuente de un objeto puede escribirse y mantenerse independiente del código fuente del resto de los objetos. De esta forma, un objeto puede pasarse fácilmente de una parte a otra del programa. Podemos dejar nuestra bicicleta a un amigo, y ésta seguirá funcionando.
- **Protección de información:** Un objeto tendrá una interfaz pública perfectamente definida que otros objetos podrán usar para comunicarse con él. De esta forma, los objetos pueden mantener información privada y pueden cambiar el modo de operar de sus funciones miembros sin que esto afecte a otros objetos que usen estas funciones miembro. Es decir, no necesitamos entender cómo funciona el mecanismo de cambio de marcha para hacer uso de él.

6 ¿QUÉ ES UN MENSAJE?

Normalmente un único objeto por sí solo no es muy útil. En general, un objeto aparece como un componente más de un programa o una aplicación que contiene otros muchos objetos. Es precisamente haciendo uso de esta interacción como los programadores consiguen una funcionalidad de mayor orden y modelar comportamientos mucho más complejos. Una bicicleta (a partir de ahora particularizaremos) colgada de un gancho en el garaje no es más que una estructura de aleación de titanio y un poco de goma. Por sí sola, tu bicicleta (por poner una bicicleta en concreto) es incapaz de desarrollar ninguna actividad. Tu bicicleta es realmente útil en tanto que otro objeto (tú) interactúa con ella (pedalea).

Los objetos de un programa interactúan y se comunican entre ellos por medio de mensajes. Cuando un objeto A quiere que otro objeto B ejecute una de sus funciones miembro (métodos de B), el objeto A manda un mensaje al objeto B.

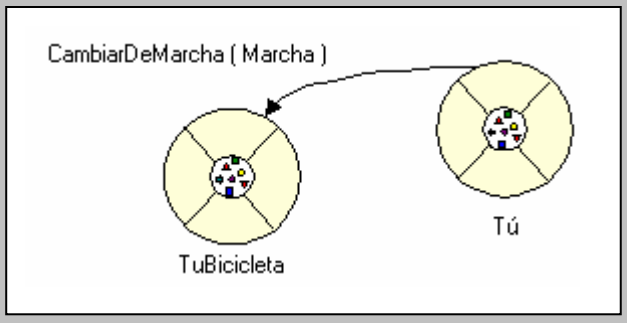


Cuando un objeto A quiere que otro objeto B ejecute una de sus funciones miembro (métodos de B), el objeto A manda un mensaje al objeto B.

En ocasiones, el objeto que recibe el mensaje necesita más información para saber exactamente lo que tiene que hacer; por ejemplo, cuando se desea cambiar la marcha de una bicicleta, se debe indicar la marcha a la que se quiere cambiar. Esta información se pasa junto con el mensaje en forma de parámetro.

La siguiente figura muestra las tres partes que componen un mensaje:

1. El objeto al cual se manda el mensaje (TuBicicleta).
2. El método o función miembro que debe ejecutar (CambiarDeMarcha).
3. Los parámetros que necesita ese método (Marcha)



El Objeto "Tú" manda el mensaje "CambiarDeMarcha" con el parámetro "Marcha" al Objeto "TuBicicleta".

Estas tres partes del mensaje (objeto destinatario, método y parámetros) son suficiente información para que el objeto que recibe el mensaje ejecute el método o la función miembro solicitada. Los mensajes proporcionan dos ventajas importantes:

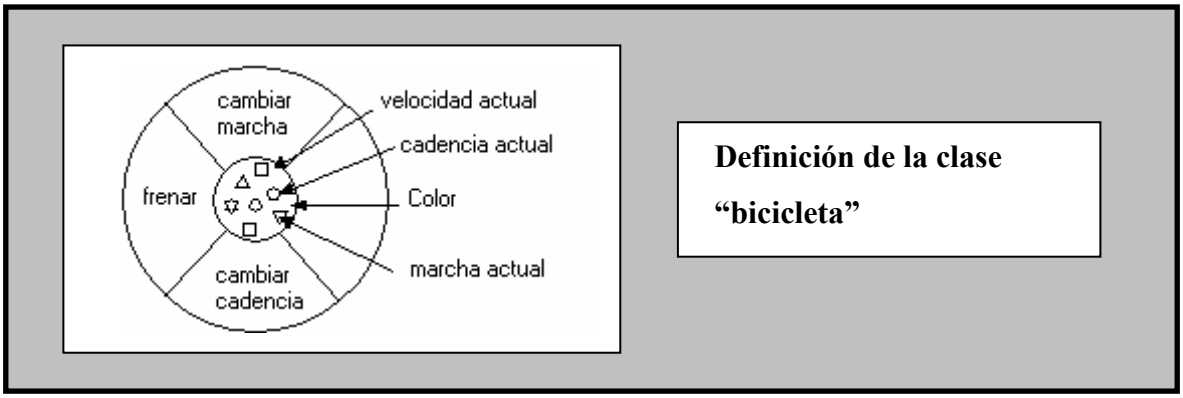
- El comportamiento de un objeto está completamente determinado (a excepción del acceso directo a variables miembro públicas) por sus métodos, así que los mensajes representan todas las posibles interacciones que pueden realizarse entre objetos.
- Los objetos no necesitan formar parte del mismo proceso, ni siquiera residir en un mismo ordenador para mandarse mensajes entre ellos (y de esta forma interactuar).

7 ¿QUÉ ES UNA CLASE?

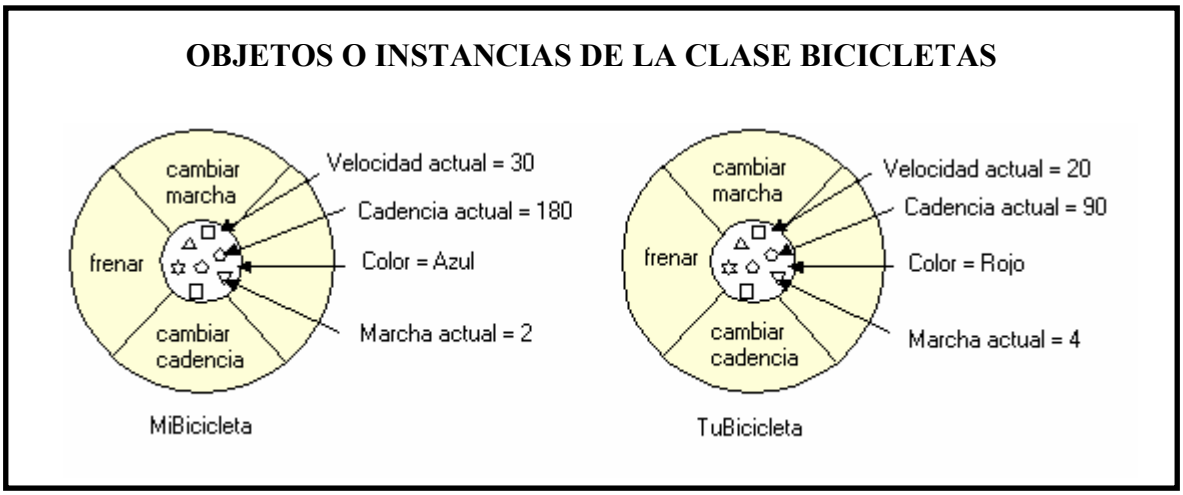
Normalmente en el mundo real existen varios objetos de un mismo tipo, o como diremos enseguida, de una misma clase. Por ejemplo, mi bicicleta es una de las muchas bicicletas que existen en el mundo. Usando la terminología de la programación orientada a objetos, diremos que mi bicicleta es una **instancia** de la **clase** de objetos conocida como *bicicletas*. Todas las bicicletas tienen algunos estados o atributos (color, marcha actual, cadencia actual, dos ruedas) y algunos métodos (cambiar de marcha, frenar) en común. Sin embargo, el estado particular de cada bicicleta es independiente del estado de las demás bicicletas. La particularización de estos atributos puede ser diferente. Es decir, una bicicleta podrá ser azul, y otra roja, pero ambas tienen en común el hecho de tener una variable "color". De este modo podemos definir una plantilla de variables y métodos para todas las bicicletas. Las plantillas para crear objetos son denominadas clases.

Una **clase** es una plantilla que define las variables y los métodos que son comunes para todos los objetos de un cierto tipo.

En nuestro ejemplo, la clase bicicleta definiría variables miembro comunes a todas las bicicletas, como la marcha actual, la cadencia actual, etc. Esta clase también debe declarar e implementar los métodos o funciones miembro que permiten al ciclista cambiar de marcha, frenar, y cambiar la cadencia de pedaleo, como se muestra en la siguiente figura:



Después de haber creado la clase bicicleta, podemos crear cualquier número de objetos bicicleta a partir de la clase. Cuando creamos una instancia de una clase, el sistema reserva suficiente memoria para el objeto con todas sus variables miembro. Cada instancia tiene su propia copia de las variables miembro definidas en la clase.



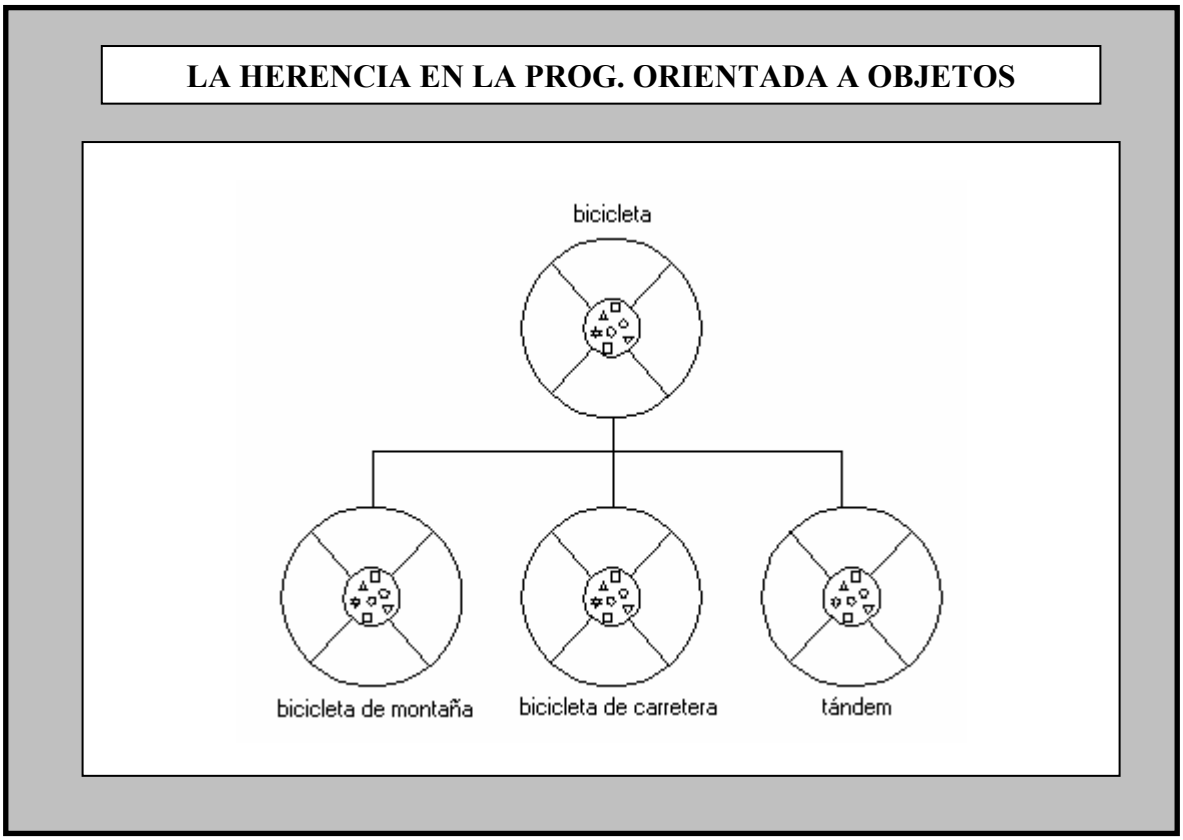
8 HERENCIA

Una vez que hemos visto el concepto de clase y el de objeto, estamos en condiciones de introducir otra de las características básicas de la programación orientada a objetos: el uso de la **herencia**.

El mecanismo de herencia permite definir nuevas clases partiendo de otras ya existentes. Las clases que *derivan* de otras heredan automáticamente todo su comportamiento, pero además pueden introducir características particulares propias que las diferencian.

Como hemos visto, los objetos se definen a partir de clases. Con el mero hecho de conocer a qué clase pertenece un objeto, ya se sabe bastante sobre él. Puede que no sepamos lo que es la “Espada”, pero si nos dicen que es una bicicleta, ya sabremos que tiene dos ruedas, manillar, pedales...

La programación orientada a objetos va más allá, permitiéndonos definir clases a partir de otras clases ya construidas. Por ejemplo, las bicicletas de montaña, las de carretera y los tándems son todos, en definitiva, bicicletas. En términos de programación orientada a objetos, son subclases o clases derivadas de la clase bicicleta. Análogamente, la clase bicicleta es la clase base o superclase de las bicicletas de montaña, las de carretera y los tándems. Esta relación se muestra en la siguiente figura.



Cada subclase hereda los estados (en forma de declaración de variables) de la superclase de la cual deriva. Las bicicletas de montaña, las de carretera y los tándems comparten algunos estados: cadencia, velocidad... Además, cada subclase hereda los métodos de su superclase.

Las bicicletas de montaña, las de carretera y los tándems comparten algunos comportamientos: frenar y cambiar la cadencia de pedaleo, por ejemplo.

Sin embargo, las clases derivadas no se encuentran limitadas por los estados y comportamientos que heredan de su superclase. Muy al contrario, estas subclases pueden añadir variables y métodos a aquellas que han heredado. Los tándems tienen dos asientos y dos manillares; algunas bicicletas de montaña tienen una catalina adicional con un conjunto de marchas con relaciones de transmisión mucho más cortas. Las clases derivadas pueden incluso sobrescribir los métodos heredados y proporcionar implementaciones más especializadas para esos métodos. Por ejemplo, si nuestra bicicleta de montaña tuviera una catalina extra, podríamos sobrescribir el método "CambiarDeMarcha" para poder usar esas nuevas marchas.

Además, no estamos limitados a un único nivel de herencia. El árbol de herencias o jerarquía de clases puede ser tan extenso como necesitemos. Los métodos y las variables miembro se heredarán hacia abajo a través de todos los niveles de la jerarquía. Normalmente, cuanto más abajo está una clase en la jerarquía de clases, más especializado es su comportamiento. En nuestro ejemplo, podríamos hacer que la clase bicicleta derivase de una superclase de vehículos.

La herencia es una herramienta clave para abordar la resolución de un problema de forma organizada, pues permite definir una relación jerárquica entre todos los conceptos que se están manejando. Es posible emplear esta técnica para descomponer un problema de cierta magnitud en un conjunto de problemas subordinados a él. La resolución del problema original se consigue cuando se han resuelto cada uno de los problemas subordinados, que a su vez pueden contener otros. Por consiguiente, la capacidad de descomponer un problema o concepto en un conjunto de objetos relacionados entre sí cuyo comportamiento es fácilmente identificable puede ser extraordinariamente útil para el desarrollo de programas informáticos.

La herencia proporciona las siguientes ventajas:

- Las clases derivadas o subclases proporcionan comportamientos especializados a partir de los elementos comunes que hereda de la clase base. A través del

mecanismo de herencia los programadores pueden reutilizar el código de la superclase tantas veces como sea necesario.

- Los programadores pueden implementar las llamadas superclases abstractas, que definen comportamientos genéricos. Las clases abstractas definen e implementan parcialmente comportamientos, pero gran parte de estos comportamientos no se definen ni se implementan totalmente. De esta forma, otros programadores pueden hacer uso de estas superclases detallando esos comportamientos con subclases especializadas. El propósito de una clase abstracta es servir de modelo base para la creación de otras clases derivadas, pero cuya implantación depende de las características particulares de cada una de ellas. Un ejemplo de clase abstracta podría ser en nuestro caso la clase vehículos. Esta clase sería una clase base genérica, a partir de la cual podríamos ir creando todo tipo de clases derivadas.