IV

INTRODUCCIÓN A LA PROGRAMACIÓN ORIENTADA A OBJETOS

Los lenguajes computacionales han experimentado una impresionante evolución desde que se construyeron las primeras computadoras. Éstos han permitido escribir programas como una serie de procedimientos que actúan sobre los datos, donde los datos están separados de los procedimientos. La programación Orientada a Objetos trata a los datos y a los procedimientos que operan sobre ellos como un solo objeto.

La programación estructurada se utiliza desde los setenta y es uno de los métodos más utilizados en el campo de la programación. Uno de los principales conceptos que introduce es la abstracción, que permite concentrarnos en conocer que tarea realiza un procedimiento, sin preocuparnos en cómo esta tarea es realizada. Sin embargo a medida que los programas crecen las estructuras de datos sobre las cuales operan los procedimientos cobran mayor importancia. Los tipos de datos son procesados en muchas funciones dentro de un programa estructurado y cuando se producen cambios en estos tipos de datos, deben hacerse modificaciones en cada sentencia que actúa sobre estos tipos dentro del programa. Otro problema es que dado que muchas funciones acceden a datos compartidos, la disposición de los datos no se puede cambiar sin modificar todas las funciones que los utilizan. En conclusión con la programación estructurada podemos empaquetar código en funciones pero ¿qué sucede con los datos?.

La programación orientada a objetos surge como un nuevo paradigma que permite acoplar el diseño de programas a situaciones del mundo real, las entidades centrales son los objetos, que son tipos de datos que encapsulan con el mismo nombre estructuras de datos y las operaciones o algoritmos que manipulan esos datos. Algunas de las ventajas de esta aproximación son la reutilización de código, el desarrollo de prototipos en forma sencilla, desarrollo de interfaces gráficas en forma rápida, bases de datos más eficientes.

V.1. FUNDAMENTOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS.

Este paradigma de programación orientada a objetos se fundamenta en las siguientes propiedades:

Abstracción: que permite representar las características esenciales de un objeto, sin preocuparse de las no esenciales.

Encapsulación: es la propiedad que permite asegurar que el contenido de la información de un objeto está oculto al mundo exterior. La encapsulación permite la división de un programa en módulos.

Estos módulos se implementan mediante clases (ver sección 5.2), las clases representan la encapsulación de abstracciones.

Modularidad: es la propiedad que permite subdividir una aplicación en partes más pequeñas cada una de las cuales debe ser tan independiente como sea posible.

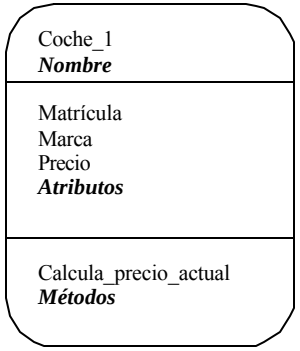
Jerarquía: es una propiedad que permite ordenar las abstracciones. Las dos jerarquías más importantes son las clases y los objetos. Las clases se relacionan unas con otras por medio de la relación herencia mediante la cual pueden definirse nuevos objetos a partir de los existentes.

V.2 CONCEPTOS FUNDAMENTALES DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

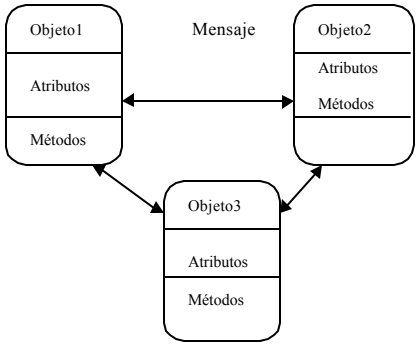
A continuación se especifican conceptos, cuya comprensión es necesaria dentro de este nuevo paradigma:

1. **Objeto**: unidad que permite combinar datos y funciones que operan sobre esos datos. Dentro de un objeto residen los datos (atributos) de los lenguajes tradicionales, tales como números, arreglos, cadenas y registros que caracterizan el estado del objeto. Así como funciones o subrutinas (métodos) que operan sobre ellos. Los métodos dentro del objeto son el único medio de acceder a los datos privados de un objeto. No se puede acceder a los datos directamente. Los datos están ocultos, y eso asegura que no se pueden modificar accidentalmente por funciones externas al objeto. Los datos y funciones asociados se dicen que están encapsulados en una entidad única o módulo.
   Ejemplo de objetos:
   a) Objetos físicos: aviones en un sistema de control de tráfico aéreo, casas, parques.
   b) Elementos de interfaces gráficas de usuario: ventanas, menús, teclado, cuadros de diálogo.
   c) Animales: animales vertebrados, animales invertebrados
   d) Tipos de datos definidos por el usuario: Datos complejos, Puntos de un sistema de coordenadas
   e) Alimentos: carnes, frutas, verduras.

La representación gráfica de un objeto se muestra a continuación:

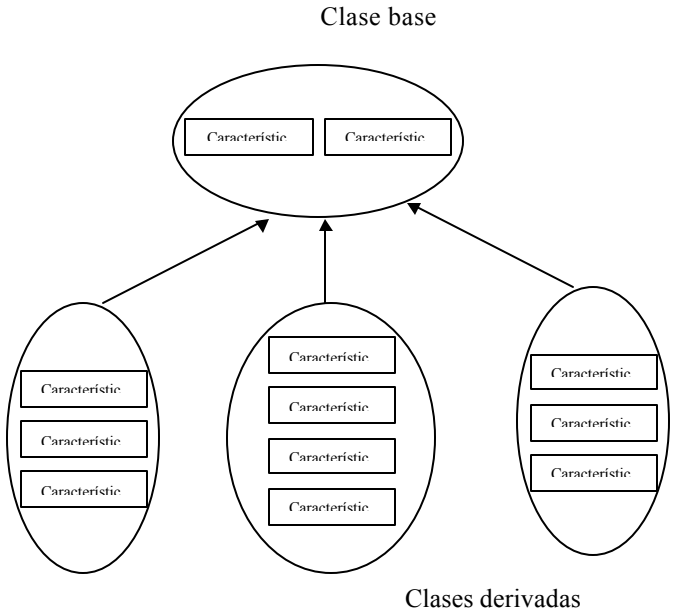


2. **Métodos y mensajes**: Un programa orientado a objetos consiste en un número de objetos que se comunican unos con otros llamando a procedimientos o funciones que reciben el nombre de *métodos*. Un *método* es el procedimiento o función que se invoca para actuar sobre un objeto. Los métodos determinan como actúan los objetos cuando reciben un mensaje. Un *mensaje* es el resultado de cierta acción efectuada por un objeto. El conjunto de mensajes a los cuales puede responder un objeto se le conoce como *protocolo del objeto*. Por ejemplo, el protocolo de un icono puede constar de mensajes invocados por el clic del botón de un ratón cuando el usuario localiza sobre el icono el apuntador de dicho ratón. De esta forma los mensajes son el único conducto que conectan al objeto con el mundo exterior.
   Cuando se ejecuta un programa orientado a objetos ocurren tres sucesos. Primero, los objetos se crean a medida que se necesitan. Segundo. Los mensajes se mueven de un objeto a otro (o del usuario a un objeto) a medida que el programa procesa información o responde a la entrada del usuario. Tercero, cuando los objetos ya no se necesitan, se borran y se libera la memoria.



3. **Clase** es la descripción abstracta de un conjunto de objetos; consta de métodos y datos que resumen características comunes del conjunto de objetos. Se pueden definir muchos objetos de una misma clase. Es decir una clase es la declaración de un tipo de objetos. Cada vez que se construye un objeto a partir de una clase se crea lo que se conoce como instancia de esa clase. Por consiguiente un objeto es una instancia de una clase. La clase tiene dos propositos definir abstracciones y favorecer la modularidad. Existen ciertas clases que se conocen como clases abstractas, estas clases no tiene instancias, pueden no existir en la realidad pero son conceptos útiles, que ocupan un lugar en la jerarquía de clases, actuando como un depósito de métodos y atributos compartidos para las subclases de nivel inferior.

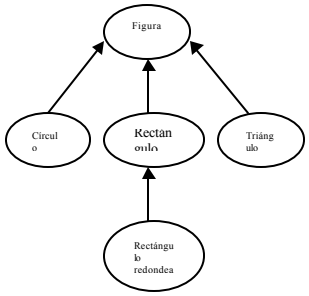
4. **Herencia** propiedad que permite a los objetos ser construidos a partir de otros objetos. Dicho de otra forma la capacidad de un objeto para utilizar estructuras de datos y métodos presentes en sus antepasados. El objetivo principal de la herencia es la *reutilización*, poder utilizar código desarollado con anterioridad.
   La herencia se basa en la división de clases básicas en subclases. Dicha división se fundamenta en el concepto de jerarquía.
   La herencia supone una clase base y una jerarquía de clases que contiene las clases derivadas. Las clases derivadas pueden heredar el código y los datos de su clase base, añadiendo su propio código especial y datos, incluso cambiar aquellos elementos de la clase base que necesitan ser diferentes.
   No debe confundirse la relación de los objetos con las clases, con la relación de una clase base con sus clases derivadas. Los objetos existentes en la memoria de la computadora expresan las características exactas de su clase. Las clases derivadas heredan características de su clase base, pero añaden otras características propias, nuevas.



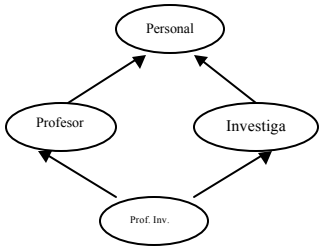
Las clases derivadas pueden a su vez servir como clases base para definir nuevas clases y de esta forma enfatizar la reutilización.
Existen dos tipos de herencia:

a) Herencia simple: Una clase puede tener sólo un ascendente. Es decir una subbase puede heredar datos y métodos de una única clase base

b) Herencia múltiple: Una clase puede tener más de un ascendente inmediato, adquirir datos y métodos de más de una clase.



Herencia simple



Herencia múltiple

La mayor parte de los lenguajes orientados a objetos permiten la herencia simple no así la herencia múltiple porque pueden surgir ambiguedades. Al aplicar la herencia múltiple si las clases utilizadas para definir una clase nueva tienen un método con el mismo nombre aparecerán problemas de ambigüedad que deberán resolverse con una operación de prioridad que el lenguaje de programación deberá soportar y entender.
Cuando se define una subclase a partir de una clase base, la subclase no tiene que heredar todas las características de la clase base, puede seleccionarse únicamente las características de útilidad, esto se conoce como *herencia selectiva*.

5. **Polimorfismo:** es el uso de un nombre o un símbolo para representar o significar más de una acción. Los operadores aritméticos de los lenguajes de programación tradicionales son ejemplo de esta propiedad ya que el símbolo + cuando se utiliza con operandos enteros representa un conjunto de operaciones de máquina diferente al conjunto empleado si los operandos son reales. Otra ilustración sería tener una clase figura que puede aceptar los mensajes dibujar, borrar y mover. Cualquier tipo derivado de una figura es un tipo de figura y puede recibir el mismo mensaje. Cuando se envía un mensaje dibujar, esta tarea será distinta según que la clase sea un triángulo un cuadrado o una elipse. Esto el polimorfismo.