## *Algoritmo*

Secuencia finita de instrucciones, cada una con significado preciso, que puede ejecutarse en una cantidad finita de esfuerzo en un tiempo finito (Aho A. et al).

Pueden estar escritos es:

1. Lenguaje Natural. Es el lenguaje común (coloquial).
2. Lenguaje Estructurado. Es un lenguaje mas limitado que el anterior, con reglas de sintaxis y semántica definidas, esto quiere decir que consiste en crear programas con instrucciones agrupadas en un estricto orden secuencial, el cual es imprescindible conservar para la resolución de un problema.
    1. Pseudocódigo; lenguaje universal para comunicarse entre programadores, esto quiere decir que es un conjunto de instrucciones en lenguaje natural, como el castellano o el inglés, de acuerdo a la persona que desarrollará un algoritmo basado en dicho lenguaje natural, en conclusión, es elaborar el algoritmo usando palabras y frases que se comprendan fácilmente.
    2. Código; lenguaje orientado a un tipo de compilador específico, para ser interpretado por el computador, en otras palabras es un conjunto de instrucciones que son parte de un lenguaje de programación especifico que se escriben en orden secuencial y se almacenan en un archivo al que se denomina programa, cuando el programa es pequeño se le denomina mini-programa o con el nombre de macro (en ingles se le denomina script).
3. Lenguaje Simbólico. Es una representación que usa símbolos predefinidos para diagramar un algoritmo, con el fin de que sea fácil de seguir la lógica de la solución que se desea expresar en forma de un flujo de pasos a realizar, indicando el inicio y el termino de los mismos

## *Lenguaje de Programación*

Los lenguajes de programación permiten comunicarse con los ordenadores o computadoras. Una vez identificada una tarea, el programador debe traducirla o codificarla a una lista de instrucciones que la computadora entienda. Un programa informático para determinada tarea puede escribirse en varios lenguajes. Según la función, el programador puede optar por el lenguaje que implique el programa menos complicado. También es importante que el programador elija el lenguaje más flexible y más ampliamente compatible para el caso de que el programa tenga varias aplicaciones.

## *Programa*

- Es un conjunto de instrucciones lógicas que tienen la finalidad de llevar a cabo una tarea específica.
- Secuencia de instrucciones mediante las cuales se ejecutan diferentes acciones de acuerdo con los datos que se estén procesando.
- Es un algoritmo desarrollado para ser utilizado por la computadora.
- Expresión de un algoritmo en un lenguaje preciso (de programación) que puede llegar a entender una máquina de cómputo.

## *Software*

Es el conjunto de programas, procedimientos y documentos relacionados con el sistema hardware. Es la herramienta de que se vale el usuario para obtener el resultado esperado de un procesamiento

de datos.

Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación. (IEEE)

## *Otros Conceptos*

- Lenguaje maquina
- Lenguaje bajo nivel
- Programa fuente
- Programa objeto
- Compilador
- Intérprete

## *La complejidad inherente al software*

Como Brooks sugiere, "la complejidad del software es una propiedad esencial, no accidental" Entre otras según Booch, ésta se deriva de:

- La complejidad del dominio del problema.
- La dificultad de gestionar el proceso de desarrollo.

### La complejidad del dominio del problema

Los problemas que se intentan resolver con software implican normalmente elementos de ineludible complejidad, en los que se encuentran:

- Una gran cantidad de requisitos, en muchas ocasiones contradictorios.
- Difíciles interacciones entre los usuarios de un sistema y sus desarrolladores: los usuarios encuentran generalmente muy difícil dar precisión sobre sus necesidades de forma que los desarrolladores puedan comprender.
- Los usuarios y desarrolladores tienen diferentes perspectivas de la naturaleza del problema y hacen suposiciones diferentes sobre la naturaleza de la solución.
- Los requisitos de un sistema software cambian durante su desarrollo.

### La dificultad de gestionar el proceso de desarrollo

Hoy es usual encontrar sistemas en funcionamiento cuyo tamaño se mide en centenares de millares, o incluso millones de líneas de código, por ejemplo el kernel de linux en su versión 2.6.36 tiene ya más de 13 millones de líneas de código[1]. Esta característica se facilita descomponiendo nuestra implementación en centenares y a veces millones de módulos independientes. Esta cantidad de trabajo exige el uso de un equipo de desarrolladores, aunque se trata por todos los medios de que este equipo sea lo más pequeño posible. Ahora bien, a medida que haya más desarrolladores, se producen comunicaciones entre ellos más complejas, e incluso con coordinación difícil entre ellos, particularmente si el equipo está disperso geográficamente, como suele ser el caso de proyectos grandes.

1 Disponible en: http://www.h-online.com/open/features/What-s-new-in-Linux-2-6-36-1103009.html?page=6

## *La crisis del software*

En 1968 una conferencia sobre software, patrocinada por la OTAN, asumió los términos ingeniería del software y crisis del software. Con estos términos se quería expresar que el software era caro, poco fiable y escaso. Las metodologías y técnicas estructurales que reinaron en la década de los setenta y ochenta no eliminaron el problema, y de hecho la crisis del software continúa hoy en día. Pese a las muchas herramientas y métodos utilizados, los problemas del diseño descendentes permanecen igual, posiblemente debido a que la complejidad del problema ha crecido considerablemente.

Entre las diferentes fases del ciclo de vida del software (Fig 1.1), el mantenimiento, aunque en tiempos fue despreciada su importancia, se considera actualmente como uno de los problemas más rigurosos en el desarrollo del software .

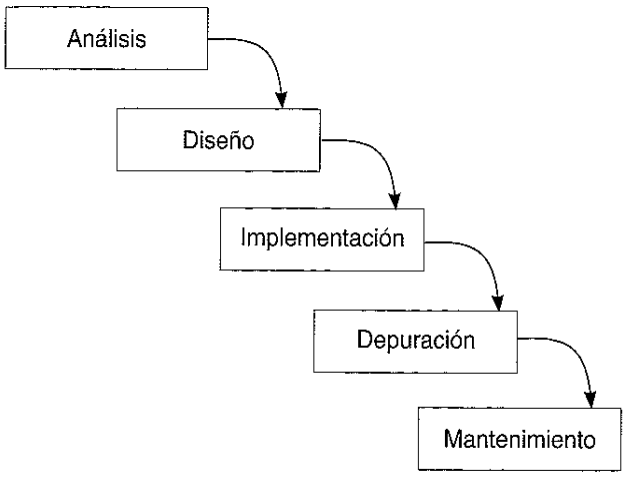


**Figura 1.1.** Ciclo de vida del software.

Muchos investigadores sugieren que los costes de software requieren más de la mitad de los costes y recursos globales en el desarrollo de software .

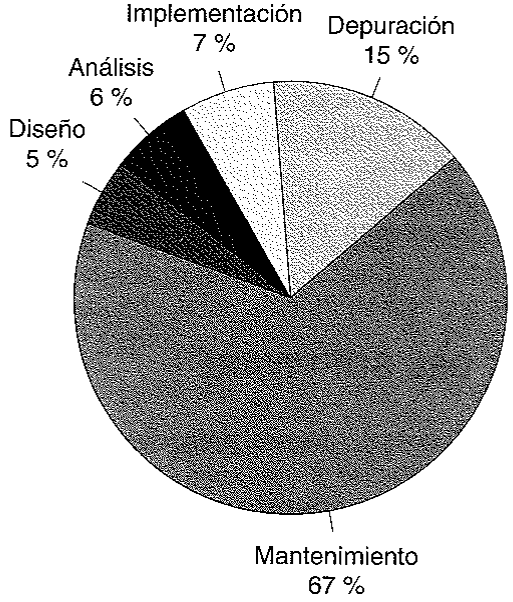


**Figura 1.2.** Costes de las diferentes fases del ciclo de vida de un proyecto software.

Los cambios realizados en la evolución de un programa son el punto débil de los métodos tradicionales de desarrollo de software, siendo paradójicamente uno de los puntos fuertes de los métodos de desarrollo de software orientado a objetos .

En 1986, Fredrick P. Brooks, en un famoso artículo, apuntaba que en los últimos diez años no se había producido ningún progreso significativo en el desarrollo de software, y analizaba críticamente todas las tecnologías más prometedoras. Aunque él confesaba que tenía más confianza en la programación orientada a objetos que en cualquier otra tecnología, mantenía dudas sobre sus ventajas efectivas.

Luego, las propuestas de reusabilidad o reutilización, "reusability", de componentes de software, se consideraron como bloques iniciales para la construcción del programa, de modo similar a la construcción de cualquier objeto complejo (tal como un automóvil) que se construye ensamblando sus partes .

En respuesta al artículo de Brooks, Brad Cox, el inventor de Objective-C, publicó un artículo en el que esencialmente rebatía las tesis de Brooks:

> Existe una bala de plata. Es un arma tremendamente potente, impulsada por vastas fuerzas económicas a la que nuevos obstáculos técnicos sólo pueden resistir brevemente .
>
> La bala de plata es un cambio cultural lugar de un cambio tecnológico. Es un nuevo paradigma; una revolución industrial basada en partes reutilizables e intercambiables que modificarán el universo del software, de igual modo que la revolución industrial cambió la fabricación.

Por consiguiente, la POO (Programación Orientada a Objetos) no sólo son nuevos lenguajes de programación, sino un nuevo modo de pensar y diseñar aplicaciones que pueden ayudar a resolver problemas que afectan al desarrollo del software. Sin embargo, el lenguaje debe ser capaz de soportar el nuevo paradigma, siendo por consiguiente una parte esencial de esta evolución.

## *Paradigmas de Programación*

El significado de paradigma (paradigma en latín, parádeigma en griego) en su origen significaba un ejemplo ilustrativo, en particular, un enunciado modelo que mostraba todas las inflexiones de una palabra.

Thomas Kuhn en The Structure of Scientific Revolutions (1962), introdujo la noción de paradigma como “logros científicos universalmente reconocidos que durante un tiempo proporcionan problemas y soluciones modelo para una comunidad de profesionales”

Peter Wegner en 1988 extendió la noción a los lenguajes de programación considerando a los paradigmas como “patrones de pensamiento para la resolución de problemas”.

Existen muchos paradigmas de programación, pero los cuatro más importantes son:

- Paradigma Imperativo
- Paradigma Funcional
- Paradigma Lógico
- Paradigma Orientado a Objetos

### Paradigma Imperativo[1]

En este paradigma se describe la computación en términos de enunciados que cambian el estado de un programa. Los programas imperativos definen secuencias de comandos que la computadora debe realizar.

El término imperativo se usa en oposición a la programación declarativa que expresa qué es el lo que se tiene que hacer sin describir cómo, en términos de secuencias de acciones que se deben tomar. La programación funcional y lógica son ejemplos de esta aproximación declarativa.

La programación procedural o por procedimientos es programación imperativa en la cual el programa está constituido de uno o más procedimentos (también conocidos como funciones o subrutinas). Los términos procedural e imperativo son usualmente usados como sinónimos.

La programación estructurada puede ser vista como una rama de la programación imperativa, su fama la tiene principalmente por restringir el uso del enunciado GOTO. Su base teórica es el teorema de la programa estructurado que dice que mediante tres formas de combinar los enunciados es posible escribir cualquier función computable. Estas tres formas son:

- Secuencias
- Selección
- Iteración o repetición

Desde los años 60 se impulsó el uso de la programación estructurada para mejorar el mantenimiento y la calidad total de los programas imperativos.

Un programa estructurado se compone de funciones, segmentos, módulos y/o subrutinas, cada una con una sola entrada y una sola salida. Cada uno de estos módulos (aún en el mismo programa completo), se denomina programa apropiado cuando, además de estar compuesto solamente por las tres estructuras básicas, tiene sólo una entrada y una salida y en ejecución no tiene partes por las cuales nunca pasa ni tiene ciclos infinitos.

#### *Secuencia*

Indica que las instrucciones de un programa se ejecutan una después de la otra, en el mismo orden en el cual aparecen en el programa. Se representa gráficamente como una caja después de otra, ambas con una sola entrada y una única salida.

Las cajas A y B pueden ser definidas para ejecutar desde una simple instrucción hasta un módulo o programa completo, siempre y cuando que estos también sean programas apropiados.

#### *Selección*

También conocida como la estructura SI-CIERTO-FALSO, plantea la selección entre dos alternativas con base en el resultado de la evaluación de una condición o predicado; equivale a la instrucción IF de todos los lenguajes de programación

#### *Iteración*

También llamada la estructura HACER-MIENTRAS-QUE, corresponde a la ejecución repetida de una instrucción mientras que se cumple una determinada condición.

Pascal, C y BASIC son ejemplos de los lenguajes imperativos más importantes.

A continuación vemos los paradigmas declarativos más importantes:

## Paradigma Programación Lógica[2]

Los sistemas de programación lógica al programador establecer una colección de axiomas o clausulas desde los cuales se pueden comprobar teoremas. El usuario de un programa lógico establece un teorema o meta y el motor del lenguaje intenta encontrar un colección de clausulas y pasos de inferencia que juntos implican la meta. De los varios lenguajes lógicos existentes el más común y más usado se llama Prolog.

En casi todos los lenguajes lógicos los axiomas se representan mediante clausulas con dos partes: una cabeza o consecuente *H*, y un cuerpo consistente en términos $B_i$.

$$H \leftarrow B_{1,} B_{2,} B_{3,} \ldots, B_n$$

El significado del enunciado anterior es que cuando todos los $B_i$ son verdaderos podemos deducir que $H$ también es verdadero. Este tipo de clausulas permiten representar la mayoría pero no todos los enunciados lógicos.

Para obtener nuevos enunciados, un sistema de programación lógica combina los enunciados ya existentes, cancelando términos equivalentes, a través de un proceso que se conoce como resolución. Por ejemplo:

$$\frac{\begin{array}{c} C \leftarrow A, B \\ D \leftarrow C \end{array}}{D \leftarrow A, B}$$

Se considera la programación lógica es potencialmente de más alto nivel que la programación funcional o la imperativa. El auge del paradigma declarativo se debe a que el área de la lógica formal de las matemáticas ofrece un sencillo algoritmo de resolución de problemas adecuado para, usarse en un sistema de programación declarativo de propósito general.

Algunas aplicaciones de la programación lógica se dan en los campos de:

- Interfaces de Lenguaje Natural.
- Sistemas Expertos.
- Resolución de ecuaciones simbólicas.
- Inteligencia Artificial en general.
- Simulación.

## Paradigma Funcional [3]

El paradigma funcional es un paradigma de programación que trata la computación como la evaluación de funciones matemáticas, hace énfasis en la funciones, en contraste con el paradigmas imperativo que hace énfasis en los cambios de estado. La programación funcional tiene sus raíces en el cálculo lambda un sistema formal desarrollado en 1930 para investigar la definición de funciones, la aplicación de funciones y recursión.

En la práctica, la diferencia entre una función matemática y la definición de función en la programación imperativa es que la funciones imperativas pueden tener efectos de lado y cambiar el valor cálculos ya realizados. En cambio, el código funcional, el valor de retorno de una función sólo de los argumentos de entrada de la función, así que siempre que se llame a una función con los mismos valores se obtendrá el mismo resultado.

Los lenguajes de programación funcional puros han sido ampliamente enseñados en ambientes académicos, no en comerciales. Sin embargo, algunos lenguajes funcionales prominentes como Scheme (dialecto Lisp), Erlang, Objective Caml y Haskell han sido usados en aplicaciones comerciales e industriales por una amplia variedad de organizaciones.

Un ejemplo de la actualidad e importancia de la programación funcional es el lenguaje multiparadigma F#, recientemente incorporado en el .NET framework de Microsoft y completamente soportado en el Visual Studio 2010.

## Un ejemplo comparativo [2]

Cuando el programador imperativo dice:

Para calcular el mcd (máximo común divisor) de a y b, revise si a y b son iguales, si los son, imprima uno y deténgase. En caso contrario, reemplace el mayor por la diferencia y repita.

El programador funcional dice:

El mcd de a y b se define como a cuando a y b son iguales, y se define como mcd de c y d cuando a y b no son iguales, donde c es el menor entre a y b y d es su diferencia. Para calcular el mcd de un par de números, expanda y simplifique su definición hasta que termine.

El programador lógico dice:

La proposición de que el mcd de a y b es igual a g es verdadera cuando a, b y g son iguales o si existen números c y d tales que es verdadero que c es el minimo entre a y b y es verdadero que d es la diferencia entre a y b, y es verdadero que g es el mcd de c y d.

Para calcular el mcd de un par de números busque un numero g para el cual se cumplan las reglas anteriores.

### Paradigma Orientado a Objetos[4]

En 1988 Peter Wegner extendió la noción de paradigma de Thomas Kuhn a los lenguajes de programación considerando a los paradigmas como “patrones de pensamiento para la resolución de problemas”. En este sentido, la programación orientada a objetos (POO) es un nuevo paradigma. La orientación a objetos fuerza a reconsiderar nuestro pensamiento sobre la computación, sobre lo que significa realizar computación y sobre cómo se estructura la información dentro del computador.

Bobrow y Stefik definen un estilo de programación como «un medio de organización de programas sobre la base de algún modelo conceptual de programación y un lenguaje apropiado para hacer programas en un estilo claro» Sugieren que existen cuatro clases de estilos de programación:

| | |
|---|---|
| Orientados a procedimientos | Algoritmos |
| Orientados a objetos | Clases y objetos |
| Orientados a lógica | Expresado en cálculo de predicados, ej.: Prolog. |
| Orientados a reglas | Reglas activadas por condiciones, ej.: CLIPS y Prolog. |

No existe ningún estilo de programación idóneo para todas las clases de programación. La orientación a objetos se acopla a la simulación de situaciones del mundo real.

En POO, 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.

El paradigma orientado a objetos, se basa en los conceptos de objetos y clases de objetos. Un objeto es una variable equipada con un conjunto de operaciones que le pertenecen o están definidas para ellos.

## *Orientación a Objetos*

La orientación a objetos puede describirse como el conjunto de disciplinas (ingeniería) que desarrollan y modelizan software que facilitan la construcción de sistemas complejos a partir de componentes.

El atractivo intuitivo de la orientación a objetos es que proporciona conceptos y herramientas con las cuales se modela y representa el mundo real tan fielmente como sea posible. Las ventajas de la orientación a objetos son muchas en programación y modelación de datos. Como apuntaban Ledbetter y Cox (1985):

> La programación orientada a objetos permite una representación más directa del modelo de mundo ideal en el código. El resultado es que la transformación radical normal de los requisitos del sistema (definido en términos de usuario) a la especificación del sistema (definido en términos de computador) se reduce considerablemente.

Las técnicas orientadas a objetos proporcionan mejoras y metodologías para construir sistemas de software complejos a partir de unidades de software modularizado y reutilizable. Se necesita un

nuevo enfoque para construir software en la actualidad. Este nuevo enfoque debe ser capaz de manipular tanto sistemas grandes como pequeños y debe crear sistemas fiables que sean flexibles, mantenibles y capaces de evolucionar para cumplir las necesidades de cambio.

## *Programación Orientada a Objetos*

### Definición

La programación orientada a objetos es una técnica estructurada siendo los objetos los principales elementos de construcción. La programación orientada a objetos es el producto de la evolución del paradigma estructurado a fin de conseguir que el software sea un producto industrial, la programación orientada a objetos es una programación que envía mensajes a los objetos [7].

Grady Booch, autor del método de diseño orientado a objetos, define la programación orientada a objetos (POO) como :

> "Un método de implementación en el que los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representan una instancia de alguna clase, y cuyas clases son todas miembros de una jerarquía de clases unidas mediante relaciones de herencias".

De la definición anterior podemos mencionar tres características básicas de la POO:

- Debe estar basado en objetos.
- Debe estar basado en clases.
- Debe ser capaz de tener herencia de clases.

### Historia de los Lenguajes Orientados a Objetos [4]

El primer lenguaje de programación que introdujo el concepto de clase fue Simula-67, como entidad que contenía datos y las operaciones que manipulaban los datos. Asimismo, introdujo también el concepto de herencia. Simula-67 es considerado el primer lenguaje orientado a objetos. El siguiente lenguaje orientado a objetos, y seguramente el más popular desde un enfoque conceptual exclusivamente de objetos, es Smalltalk, cuya primera versión comercial se desarrolló en 1976, y en 1980 se popularizó con la aparición de Smalltalk-80, que es considerado por algunos como el lenguaje orientado a objetos por excelencia, se caracteriza por soportar todas las propiedades fundamentales de la orientación a objetos, dentro de un entorno integrado de desarrollo, con interfaz interactivo de usuario basado en menús. Entre los lenguajes orientados a objetos que se han desarrollado a partir de los ochenta destacan extensiones de lenguajes tradicionales tales como C++ y Objective-C (extensiones de C), Modula-2 y Object Pascal (extensión de Pascal) , Object Cobol y Java.

Un lenguaje que impulsó mucho la expansión de la orientación a objetos es: C++.

Otro lenguaje como Smalltalk orientado a objetos puro es Eiffel, creado por Bertrand Meyer y que soporta todas las propiedades fundamentales de objetos. Actualmente se usa como plataforma de desarrollo en finanzas, aplicaciones aeroespaciales, aplicaciones para salud, videojuegos y otras industrias, existen plugins para desarrollar en Eiffel con Visual Studio .NET 2003 [6].

Ada ha sido también un lenguaje -en este caso basado en objetos- que soporta la mayoría de las propiedades orientadas a objetos. Sin embargo, desde la versión Ada-95 ya soporta herencia y polimorfismo. Actualmente Ada es un estándar internacional definido por ISO y ANSI conocido como Ada 2005.

En los últimos años algunos de los lenguajes orientados a objetos más comercialmente usados son: Visual Basic.NET, C# y Java.

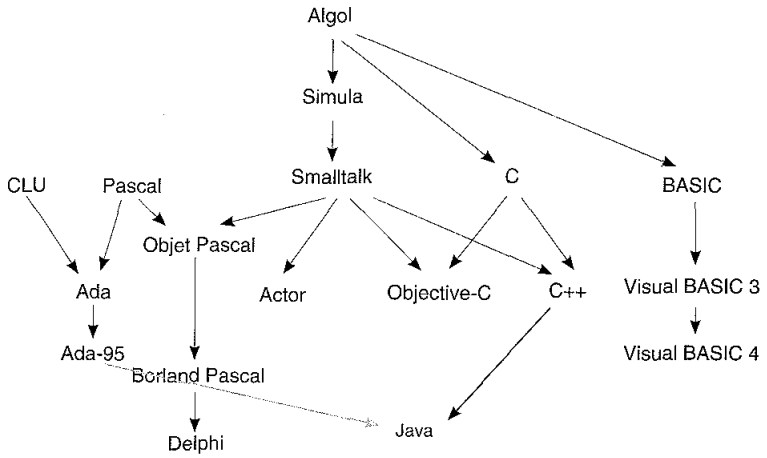


**Figura 1.5.** Evolución de los lenguajes orientados a objetos.

## Ventajas de la Programación Orientada a Objetos

- Uniformidad. Ya que la representación de los objetos implica tanto el análisis como el diseño y la codificación de los mismos.
- Comprensión. Tanto los datos que componen los objetos, como los procedimientos que los manipulan, están agrupados en clases, que se corresponden con las estructuras de información que el programa trata.
- Flexibilidad. Al tener relacionados los procedimientos que manipulan los datos con los datos a tratar, cualquier cambio que se realice sobre ellos quedará reflejado automáticamente en cualquier lugar donde estos datos aparezcan.
- Estabilidad. Dado que permite un tratamiento diferenciado de aquellos objetos que permanecen constantes en el tiempo sobre aquellos que cambian con frecuencia permite aislar las partes del programa que permanecen inalterables en el tiempo.
- La POO es especialmente adecuada para realizar determinadas aplicaciones, sobre todo la realización de prototipos y simulación de programas.
- Los mecanismos de encapsulación de POO soportan un alto grado de reutilización de código, que se incrementa por sus mecanismos de herencia
- El entorno de las bases de datos, la POO se adjunta bien a los modelos semánticos de datos, para solucionar las limitaciones de los modelos tradicionales
- Interfaces de usuario gráficos (icono) y visuales. Los interfaces de usuario se modelan muy bien utilizando objetos.

## Objetos [7]

Un objeto es una agrupación de código, compuesta de propiedades y métodos, que pueden ser manipulados como una entidad independiente. Las propiedades definen los datos o información del objeto, permitiendo consultar o modificar su estado; mientras que los métodos son las rutinas que definen su comportamiento.

Un objeto es una pieza que se ocupa de desempeñar un trabajo concreto dentro de una estructura organizativa de nivel superior, formada por múltiples objetos, cada uno de los cuales ejerce la tarea particular para la que ha sido diseñado.

La idea fundamental en los lenguajes orientados a objetos es combinar en una sola unidad datos y funciones que operan sobre estos datos, tal unidad se denomina objeto.

Es la instancia de una clase. Una clase es la representación abstracta de un concepto en el mundo real, y proporciona la base a partir de la cual creamos instancias de objetos específicos. Como ejemplo, puede crear una clase que defina a un cliente. Después puede crear una nueva instancia de la clase cliente para tener un objeto utilizable de Cliente. Para poder crear un objeto de la clase cliente, debe crear una nueva instancia basada en esa clase. Por ejemplo:

```
Private Objetocliente as Clasecliente

Objetocliente = New ClaseCliente()
```

Cada objeto es un elemento único de la clase en la que se basa. Si una clase es como un molde, entonces un objeto es lo que se crea a partir del molde. La clase es la definición de un elemento; el objeto es el elemento. El molde para una figura de cerámica en particular, es como una clase; la figura es el objeto.

La herencia, el hecho de pertenecer a una estructura organizativa, hace la diferencia entre un TDA y un objeto.

Ej.: Definir un TDACola y un TDAColaConPrioridades vrs definir una clase Cola y una clase ColaConPrioridades.

## Estructura de los Objetos [7]

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

- Relaciones
- Propiedades (datos miembro)
- Métodos (funciones miembro)

#### *Relaciones*

Las relaciones entre objetos son, precisamente, los enlaces que permiten a un objeto relacionarse con aquellos que forman parte de la misma organización.

Las hay de dos tipos fundamentales:

- Relaciones jerárquicas. Son esenciales para la existencia misma de la aplicación porque la construyen. Son bidireccionales, es decir, un objeto es padre de otro cuando 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.

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

  Se refiere a la antes mencionada jerarquía de especialización.

  **Importante: El objetivo final de la herencia es la reutilización de código.**

- Relaciones semánticas. Se refieren a las relaciones que no tienen nada que ver con la organización de la que forman parte los objetos que las establecen. Sus propiedades y consecuencia solo dependen de los objetos en sí mismos (de su significado) y no de su posición en la organización.

  Se refiere a la antes mencionada jerarquía de agregación.

#### *Propiedades (atributos o campos)*

Todo objeto puede tener cierto número de propiedades, cada una de las cuales tendrá, a su vez, uno

o varios valores. En POO, las propiedades corresponden a las clásicas "variables" de la programación estructurada. Son, por lo 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 mas 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.

Los atributos están asociados a clases y objetos, y describen la clase o el objeto de alguna manera. Las entidades de la vida real están a menudo descritas con palabras que indican características estables. La mayoría de los objetos físicos tienen características tales como forma, peso, color y tipo de material. Las personas tienen características como fecha de nacimiento, padres, nombre y color de los ojos. Una característica puede verse como una relación binaria entre una clase y cierto dominio.

Los atributos describen el estado del objeto. Un atributo consta de de dos partes: un nombre de atributo y un valor de atributo.

La “relación” binaria implica que un atributo puede tomar un valor definido por un dominio enumerado. En la mayoría de los casos, un dominio es simplemente un conjunto de valores específicos. Por ejemplo, supongamos que una clase Coche tiene un atributo color. El dominio de valores de color es blanco, negro, plata, gris, azul, rojo, amarillo, verde.

Las características (valores del dominio) pueden aumentarse asignando un valor por defecto (característica) a un atributo. Por ejemplo, el atributo color tiene el valor por defecto negro.

Las propiedades pueden ser heredadas o propias.

#### *Métodos*

Una operación que realiza acceso a los datos. Podemos definir método como un programa procedimental o procedural 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.

Los métodos describen el comportamiento asociado a un objeto, representan las acciones que pueden realizarse por un objeto o sobre un objeto, la ejecución de un método puede conducir a cambiar el estado del objeto o dato local del objeto, en la POO el cuerpo de un método consta de un bloque de código procedimental que ejecuta la acción requerida, todos los métodos que alteran o acceden a los datos de un objeto se define dentro del objeto, Un método dentro de un objeto se activa  por un mensaje que se envía por otro objeto al objeto que contiene el método

Si los métodos son programas, se deduce que podrían tener argumentos, o parámetros. 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 incluidos 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.\

#### *Otros Conceptos*

#### *Constructores*

Un constructor es un método especial que se ejecuta durante la creación de un objeto. Dentro de un constructor se colocan instrucciones para inicializar los objetos al ser creados. Normalmente, el nombre de un constructor es el mismo nombre de la clase que lo contiene.

Una clase puede tener varios constructores con tal que se diferencien en el tipo o el número de parámetros que reciben.

#### *Enumeraciones*

Las enumeraciones son elementos que se pueden utilizar para indicar el conjunto de valores posibles de un campo de una clase. Esto permite limitar los valores que se pueden asignar a una propiedad específica.

#### *Eventos*

En la programación orientada a objetos, y más aún, en la programación visual, los eventos son mensajes que se envían para informar que algo ha ocurrido. Son mensajes que actúan como notificaciones para indicar a otros objetos que ha ocurrido una situación específica en el sistema.

#### *Modificadores de accesibilidad*

Los modificadores de accesibilidad son una característica de la encapsulación que nos permite especificar el nivel de acceso que tendran los programas sobre los miembros de nuestras clases. Existen tres modificadores de accesibilidad:

- Public: Indica que un miembro puede ser accesado desde cualquier parte del programa.
- Private: Limita el acceso solo a los miembros dentro de la misma clase.
- Protected: Limita el acceso a la clase contenedora o a los tipos derivados de esta clase

### Identidad del Objeto [7]

La identidad expresa que aunque dos objetos sean exactamente iguales en sus atributos, son distintos entre sí. De esta forma incluso una serie de objetos carro recién fabricados son distintos los unos de los otros.

La afirmación anterior, aunque parece obvia, tiene importancia cuando descendemos al nivel de programación. En este ámbito cada uno de los objetos tiene un controlador por el cual se identifica. Este puede ser una variable, una estructura de datos, una cadena de caracteres, etc. El controlador será distinto para cada uno de los objeto, aunque las referencias a éstos sean uniformes e independientes del contenido, permitiendo crear agrupaciones de objetos con el mismo tratamiento.

### Clasificación [7]

Con la clasificación comienza la verdadera programación orientada a objetos. Ellos nos obliga a una abstracción del concepto de objeto denominada clase.

Las clases permiten la agrupación de objetos que comparten las mismas propiedades y comportamiento. Si bien clase y objeto suelen usarse como sinónimos, no lo son.

El esfuerzo del programador ante una aplicación orientada a objetos se centra en la identificación de las clases, sus atributos y operaciones asociadas. El programador no crea directamente los objetos.

Las propiedades de cada clase deben cumplir una serie de premisas:

- Las propiedades deber ser significativas dentro del entorno de la aplicación es decir, deben servir para identificar claramente y de una manera única (y univoca) a cada uno de los objetos.

- El número de propiedades de un objeto debe ser el mínimo para realizar todas las operaciones que requiera la aplicación.

## Clases

Una clase es la descripción de un conjunto de objetos, consta de métodos y datos que resumen características comunes de un conjunto de objetos, se pueden definir muchos objetos de la misma clase, cada vez que se construyen un objeto a partir de una clase estamos creando lo que se llama instancia de esa clase, una instancia es una variable de tipo objeto.

A partir del concepto de clase podemos decir que un objeto es una instancia de una clase.

Una clase es esencialmente un proyecto, a partir del cual puede crear objetos. Una clase define las características de un objeto, incluyendo las propiedades que definen los tipos de datos que ese objeto puede contener y los métodos que describen el comportamiento del objeto. Estas características determinan la manera en que otros objetos pueden acceder y trabajar con los datos que se incluyen en el objeto.

La clase determina el conjunto de puntos clave que ha de cumplir un objeto para ser considerado perteneciente a dicha clase o categoría, ya que no es obligatorio que dos objetos creados a partir de la misma clase sean exactamente iguales, basta con que cumplan las especificaciones clave de la clase.

Aunque objetos distintos de una misma clase pueden tener ciertas propiedades diferentes, deben tener el mismo comportamiento o métodos. Ej.: dos carros de distinto color.

## Requisitos para que un lenguaje sea orientado a objetos [4]

La orientación a objetos trata de cumplir las necesidades de los usuarios finales, así como las propias de los desarrolladores de productos software. Estas tareas se realizan mediante la modelización del mundo real. El soporte fundamental es el modelo objeto. Los cuatro elementos (propiedades) más importantes de este modelo son:

- Abstracción
- Encapsulación
- Modularidad
- Jerarquía

Como sugiere Booch, si alguno de estos elementos no existe, se dice que el modelo no es orientado a objetos. Revisemos cada una:

- Abstracción: es la propiedad que permite representar las características esenciales de un objeto, sin preocuparse de las restantes características (no esenciales), una abstracción se centra en la vista externa de un objeto, de modo que sirva para separar el comportamiento esencial de un objeto de su implementación.
- Encapsulación: es la propiedad que permite asegurar que el contenido de la información de un objeto esta oculta al mundo exterior, el objeto A no conoce que hace el objeto B, y viceversa, (la encapsulación se conoce como ocultación de la información), en esencia es el proceso de ocultar todos los secretos de un objeto que no contribuya a sus características esenciales.
- Modularidad: es la propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas módulos) cada una de las cuales deben de ser independiente como sea posible de la aplicación en sí y de las restantes partes.

La modularización: consiste en dividir un programa en módulos que se puedan compilar por separado, pero que tiene conexiones con otros módulos.

- Jerarquía: es una propiedad que permite una ordenación de las abstracciones. Las dos jerarquías más importantes de un sistema complejo son : estructura de clases (especialización), y estructura de objetos (agregación), las jerarquías de especialización se conocen como herencia, básicamente un herencia define una relación entre clases, en donde una clase comparte la estructura o comportamiento definido en una o mas clases (herencia simple o herencia múltiple) la agregación es el concepto que permite el agrupamiento físico estructuras relacionadas lógicamente, así un camión se compone de ruedas, motor, sistema de transmisión y chasis, en consecuencia un camión es una agregación, y ruedas, motor, transmisión y chasis son agregados de camión.
- Polimorfismo: esta propiedad no suele ser considerada como fundamental en los diferentes modelos de objetos propuestos, pero, dada su importancia, no tiene sentido considerar un objeto modelo que no soporte esta propiedad. Polimorfismo implica que:
  - Una misma entidad de programación pueda tomar muchas formas.
  - Se pueda referir a dos objetos clases distintas a través de una misma clase antepasada de ambos.
  - Se pueda invocar el mismo método en dos objetos de clases distintas y obtener un comportamiento diferente.

## *Lista de Referencias*


[1] Programming paradigm. (2010, September 13). In *Wikipedia, The Free Encyclopedia*. Retrieved 05:48, September 14, 2010, from http://en.wikipedia.org/w/index.php?title=Programming_paradigm&oldid=384602252

[2] Scott M (2006). Programming Languages Pragmatics

[3] Functional programming. (2010, September 10). In *Wikipedia, The Free Encyclopedia*. Retrieved 14:21, September 14, 2010, from http://en.wikipedia.org/w/index.php?title=Functional_programming&oldid=383995245

[4] Aguilar L. J (1996). Conceptos fundamentales de programación orientada a objetos. En J. Domínguez (Ed.) *Programación Orientada a Objetos* (pp 66 – 111). España: McGraw-Hill.

[5] Schuerer K. , Maufrais C, Letondal C, Deveaud E, & Petit M (2008). *Introduction to Programming using Python Programming Course for Biologists at the Pasteur Institute* .Extraído el 15 de Septiembre de 2010 desde http://www.pasteur.fr/formation/infobio/python/support.pdf

[6] Eiffel (programming language). (2010, 25 de agosto). In *Wikipedia, The Free Encyclopedia*. Recuperado 06:31, agosto 31, 2010 de http://en.wikipedia.org/w/index.php?title=Eiffel_(programming_language)&oldid=380870822

[7] Duarte H. *Material para la clase Programación Orientada a Objetos.*