Paradigmas de programación

Programming paradigms

Autores: Martin Alejandro Carvajal Rada, Juan Manuel Suarez Aguirre

*Ingeniería de Sistemas, Universidad Tecnológica de Pereira, Pereira, Colombia*

Correo-e: m.carvajal1@utp.edu.co

***Resumen*— En este documento voy a hablar sobre algunos paradigmas de programación entre estos están (Imperativo, Comparativo, Lógico, Funcional, Orientado a objetos, Orientado a aspectos)**

***Palabras clave—* Paradigma, programación, Paradigma imperativo, Paradigma declarativo, Paradigma lógico, Paradigma funcional, Java, PHP, Estructuras Cíclicas de Control, Subexpresiones, Lenguaje, Software, Programas,** **IDE**

***Abstract*— In this paper I will talk about some programming paradigms among these are (Imperative, Comparative, Logical, Functional, Object-oriented, Aspect-oriented)**

***Key Word* — Paradigm, programming, Paradigm imperative, Declarative paradigm, Logical paradigm, Functional paradigm, Java, PHP, Control Cyclic Structures, Sub-expressions, Language, Software, Programs, IDE**

1. INTRODUCCIÓN

Para programar necesitamos tener un conjunto de herramientas:

Entorno de desarrollo integrado (IDE): es el conjunto de todas las herramientas que necesitamos para escribir y ejecutar un programa. Básicamente existen dos caminos posibles: que el lenguaje sea interpretado o compilado. El entorno de desarrollo que usemos estará determinado por el lenguaje que utilicemos.

Lenguaje de programación en gran parte estará determinado por el paradigma que utilicemos.

Paradigma de programación en si no es una herramienta si no una característica del lenguaje que utilicemos.

1. CONTENIDO

PARADIGMA IMPERATIVO

Uno de los paradigmas más representativos de la programación y de hecho el primer paradigma formalmente aceptado es el imperativo. Imperar significa mandar, ordenar, y eso es exactamente lo que hacemos al programar.

También es llamado procedimental o algorítmico. La programación imperativa se rige por dos conceptos básicos para la construcción de programas: la estructura y el módulo. De ahí que se hable de programación estructurada y de programación modular. La programación en el paradigma imperativo consiste en determinar qué datos son requeridos para el cálculo, asociar a estas direcciones de memoria y efectuar, paso a paso, una secuencia de transformaciones en los datos almacenados de forma tal que el estado final represente el resultado correcto.

La programación imperativa se basa en tres conceptos importantes:

•Celda de memoria variable

•Operaciones de asignación

•Operaciones de repetición

Celdas de memoria: donde se almacenan valores que son referenciados y modificados durante la ejecución del programa, a las que se suele asignar un nombre o identificador.

Operaciones de asignación: son las que permiten asociar los datos o valores a la posición de la memoria o variable que los contiene.

Repetición: normalmente un programa iterativo realiza su tarea ejecutando una y otra vez una secuencia de pasos elementales.

Ventajas y desventajas de la programación imperativa

Ventajas:

* Su relativa simplicidad y facilidad de implementación de los compiladores e intérpretes
* La capacidad de reutilizar el mismo código en diferentes lugares en el programa sin copiarlo.
* Una forma fácil seguir la pista de flujo del programa.
* La capacidad de ser muy modular o estructurado.
* Necesita menos memoria solamente.

Desventajas:

* Los datos son expuestos a la totalidad del programa, así que no hay seguridad para los datos.
* Dificultad para relacionarse con los objetos del mundo real.
* Difícil crear nuevos tipos de datos reduce la extensibilidad.
* Se da importancia a la operación de datos en lugar de los datos mismos

Los lenguajes imperativos pueden resolver prácticamente cualquier problema en cualquier área: desde simples hasta complejos cálculos matemáticos.

Se pueden hacer cualquier tipo de aplicaciones:

* De nóminas
* De control aéreo
* De inteligencia artificial
* De control de dosis de medicamentos
* Para cajeros automáticos
* Para naves espaciales
* Para dispositivos móviles
* Aplicaciones en línea y tiempo real

Hay que recordar que fue el primer paradigma que le vino a poner orden a la manera de hacer programas, y por tanto su filosofía marco la línea a seguir para resolver problemas de la vida cotidiana.

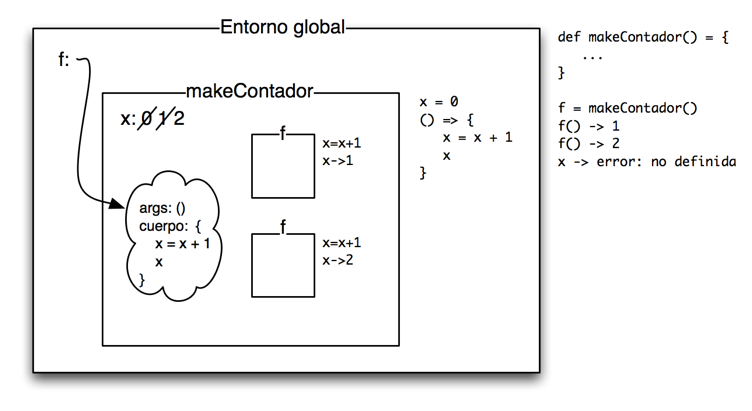
Elementos básicos:

La instrucción básica del paradigma imperativo es la asignación. Elementos que intervienen:

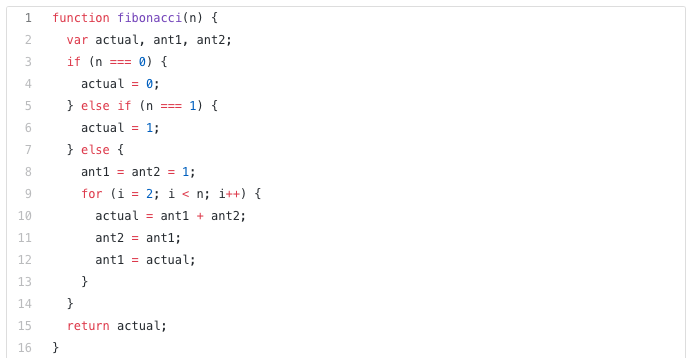
* Variables: En este paradigma las variables son identificadores asociados a celdas de memoria, las cuales contienen valores o referencias y pueden ser modificadas.
* Expresiones
* Valores literales
* Funciones predefinidas
* Operadores

Variantes entre distintos lenguajes:

* Símbolo de asignación (:=, =), auto asignación, autoincremento
* Equivalencia expresión/sentencia



La programación imperativa es la más usada y la más antigua, el ejemplo principal es el [lenguaje de máquina](https://es.wikipedia.org/wiki/Lenguaje_de_m%C3%A1quina). Ejemplos de lenguajes puros de este paradigma serían el [C](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_C), [BASIC](https://es.wikipedia.org/wiki/BASIC) o [Pascal](https://es.wikipedia.org/wiki/Pascal_(lenguaje_de_programaci%C3%B3n)).



La imagen anterior es un buen ejemplo de cómo puede ser aplicada la programación imperativa en un algoritmo de Fibonacci.

Los lenguajes de programación imperativos generalmente hacen uso de procedimientos, rutinas o funciones impuras para establecer mecanismos de control, que potencialmente generan efectos secundarios y mutan el estado del programa durante su ejecución, a diferencia de los lenguajes de programación declarativos, en donde los mecanismos de control están dados por funciones o expresiones puramente matemáticas que carecen de efectos secundarios.

Estructuras Cíclicas de Control

La mayoría de lenguajes de programación imperativos tienen estructuras cíclicas de control tales como ***while***, ***do..while***, ***for*** y ***loop*** para iterar sobre un determinado bloque de códigos dada una condición o expresión, mientras que la gran mayoría de lenguajes declarativos, especialmente los funcionales (subparadigma declarativo), carecen de este tipo de estructuras cíclicas, de hecho la única forma de representar un flujo cíclico es a través de la [recursión](https://es.wikipedia.org/wiki/Recursi%C3%B3n) (funciones que se llaman a sí misma) o través de funciones de alto nivel tales como ***map*** y ***reduce*** (que internamente usan recursión).

A continuación veremos dos implementaciones básicas de la función ***map***, que transforma secuencialmente los elementos de una lista según lo especificado en la función anónima que se pasa como segundo parámetro:



PARADIGMA DECLARATIVO

El paradigma declarativo es un [paradigma de programación](https://www.ecured.cu/index.php?title=Paradigma_de_programaci%C3%B3n&action=edit&redlink=1) que está basado en el desarrollo de [programas](https://www.ecured.cu/Software) especificando o "declarando" un conjunto de condiciones, proposiciones, afirmaciones, restricciones, ecuaciones o transformaciones que describen el problema y detallan su solución. La solución es obtenida mediante mecanismos internos de control, sin especificar exactamente cómo encontrarla (tan solo se le indica a la [computadora](https://www.ecured.cu/Computadora) qué es lo que se desea obtener o qué es lo que se está buscando). No existen asignaciones destructivas, y las variables son utilizadas con transparencia referencial

En la programación declarativa las sentencias que se utilizan lo que hacen es describir el problema que se quiere solucionar; se programa diciendo lo que se quiere resolver a nivel de usuario, pero no las instrucciones necesarias para solucionarlo. Esto último se realizará mediante mecanismos internos de [inferencia](https://www.ecured.cu/index.php?title=Inferencia&action=edit&redlink=1) de información a partir de la descripción realizada.

En realidad, la programación declarativa es un término que agrupa los siguientes paradigmas de programación:

* Programación lógica. Los problemas se representan por medio de lógica matemática.
* Programación funcional. Todo se resuelve por medio de la evaluación de funciones matemáticas.
* Lenguajes de dominio específico (DSLs). Lenguajes descriptivos para un propósito específico, tales como HTML, CSS y SQL.
* Lenguajes híbridos. Un ejemplo son los archivos “make” que combinan la descripción de dependencias entre componentes, con instrucciones imperativas para compilar o instalar una aplicación.

Ventajas de la programación declarativa:

Cuando pensamos en los beneficios de programar en forma declarativa, en general se empieza pensando en las ventajas propias del lenguaje a utilizar. Por ejemplo, si se está usando un lenguaje funcional, la principal ventaja es que al lidiar puramente con funciones, no necesitamos preocuparnos por el estado de la información, ya que los datos sean inmutables. Por otro lado, en el caso de los lenguajes basado en reglas, los programas son más claros y entendibles incluso por los usuarios.

A pesar de todo esto, la ventaja más importante de la programación declarativa consiste en que el indicar a la computadora“qué” tarea es la que tiene que hacer, en lugar de “cómo” hacerla nos protege de cambios en el contexto tecnológico. En ese sentido, el qué perdura mucho más que el cómo.

**Un primer ejemplo**

Veamos un ejemplo de cómo nos puede ayudar la programación declarativa. Supongamos un caso sencillo de un programa que suma los números del 1 al 100. Una posible solución procedural podría ser un programa similar al siguiente:

int suma = 0;  
for (int i = 1 to 100)  
    suma += i;  
return suma;

Una solución declarativa podría ser simplemente:

suma = Sum(1, 100)

Dado este ejemplo sencillo, lo primero que uno advierte es que se necesita un lenguaje de más alto nivel que dé soporte a las cosas que declaramos: en este caso alguien tiene que implementar el Sum, y el programador no tiene idea cómo se está resolviendo el proceso de la suma.

Un lenguaje declarativo es un tipo de [lenguaje de programación](http://enciclopedia.us.es/index.php/Lenguaje_de_programaci%C3%B3n) basado más en las matemáticas y en la lógica que los [lenguajes imperativos](http://enciclopedia.us.es/index.php/Lenguaje_imperativo), más cercanos estos al razonamiento humano. Los lenguajes declarativos no dicen cómo hacer una cosa, sino, más bien, qué cosa hacer. A diferencia de los imperativos, no suele haber declaración de variables ni tipos.

Ejemplo de un programa en un lenguaje declarativo: Este sería el código de un programa que determina el [factorial](http://enciclopedia.us.es/index.php/Factorial) de un número en un lenguaje declarativo inventado

factorial(0) = 1

factorial(n) = n\*factorial(n-1)

Como se puede ver, el programa utiliza simplemente 2 líneas. En una está el "caso base", que nos dice que el factorial del número 0 es 1. La otra línea nos dice que el factorial de un número n es n por el factorial de n-1. Un ejemplo de ejecución sería:

factorial(3)?=3\*factorial(2)=3\*2\*factorial(1)=3\*2\*1\*factorial(0) = 3\*2\*1\*1 = 6

Nótese que el caso base es necesario, sino el programa entraría en un bucle infinito, sin terminar jamás.

PARADIGMA LÓGICO

Consiste en la aplicación del corpus de conocimiento sobre [lógica](https://www.ecured.cu/L%C3%B3gica) para el diseño de [lenguajes de programación](https://www.ecured.cu/Lenguajes_de_programaci%C3%B3n); no debe confundirse con la disciplina de la [lógica computacional](https://www.ecured.cu/index.php?title=L%C3%B3gica_computacional&action=edit&redlink=1). La programación lógica es un tipo de [paradigmas de programación](https://www.ecured.cu/Paradigmas_de_programaci%C3%B3n) dentro del paradigma de [programación declarativa](https://www.ecured.cu/Programaci%C3%B3n_declarativa). La programación lógica gira en torno al concepto de predicado, o relación entre elementos.

La [lógica matemática](https://www.ecured.cu/L%C3%B3gica_matem%C3%A1tica) es la manera más sencilla, para el intelecto humano, de expresar formalmente problemas complejos y de resolverlos mediante la aplicación de reglas, hipótesis y teoremas. De ahí que el concepto de *"programación lógica"* resulte atractivo en diversos campos donde la programación tradicional es un fracaso. La lógica ha estado muy relacionada históricamente con las computadoras y los lenguajes de programación. Presentemos algunos ejemplos:

* Los circuitos de las computadoras son diseñados con la ayuda del álgebra booleana (George Boole)
* Datos y expresiones booleanos son usados en casi todos los lenguajes de programación para el control de acciones del programa.
* Enunciados lógicos se usan para especificaciones formales que describen el comportamiento de un programa, lo que permite realizar sobre estas pruebas de corrección.
* Enunciados lógicos vistos como un lenguaje de programación y ejecutados en la computadora.

La mayoría de los lenguajes de programación lógica se basan en la teoría lógica de primer orden, aunque también incorporan algunos comportamientos de orden superior. En este sentido, destacan los lenguajes funcionales, ya que se basan en el cálculo lambda, que es la única teoría lógica de orden superior que es demostradamente computable (hasta el momento).

El cálculo de predicados de primer orden consta de un alfabeto y de dos clases de expresiones definidas a partir de los símbolos de este alfabeto, los *términos* y las *fórmulas*. El alfabeto del lenguaje consta de los siguientes conjuntos:

* V={ x, y, z, ....} cuyos elementos se denominan símbolos de variables (individuales).
* F={ f, g, h, ….} , donde cada elemento f es un símbolo *funcional* n-ario (n ≥ 0), por ejemplo sucesor(x). Si n = 0 el símbolo de función se denomina símbolo de constante. Las constantes m{as empleadas son true y false.
* R={ R, S, .....} , donde cada elemento R es un símbolo de relación n-aria ( n ≥ 0). Si n = 0 el símbolo de relación se denomina símbolo de constante proposicional o proposición.

Un conjunto finito de símbolos denominados operadores lógicos:

* : denominado negación,
* ∧ : denominado conjunción,
* ∨ : denominado disyunción,
* ⇒ : denominado implicación,
* ⇔: denominado bicondicional

Ejemplos x, a, f(x), g(a, f(b)) son términos:

En consideraciones posteriores jugará un papel importante una clase de términos denominados listas, que en la programación lógica se presenta con la notación: [ t1, t2, ..., tn ]

con cabeza t1 el car y cola [ t2, ..., tn].

La lista [X|Y] se denomina una lista patrón o un esquema de lista que denota cualquier lista con un primer elemento X y resto Y.

Ejemplos

[1, 2, 3]

[a, 3, [3], [[3]]]

[[X, Y|[2,3]]

[[X|Y], Z, [X|W]]

Fórmulas.

* Si R es un símbolo relacional n-ario ( n ≥ 0 ) y tl,t2,...,tn son términos, entonces R(t1,t2,...,tn) es una fórmula elemental o átomo.
* Si A es una fórmula elemental, entonces A es una fórmula.
* Si A es una fórmula, entonces A es una fórmula.
* Si A y B son fórmulas, entonces [A ∨ B], [A ∧ B], [A=>B] y [A <=> B] son fórmulas.
* Si x es una variable que ocurre en la fórmula A, entonces se dice que x está acotada o ligada en A, si A es el alcance del cuantificador ∀(x) (∃ (x)).

Obsérvese que el programa lógico no solamente es capaz de responder si una determinada hipótesis es verdadera o falsa. También es capaz de determinar que valores de la incógnita hacen cierta la hipótesis. Este ejemplo es claramente académico. Sin embargo, consideremos el siguiente ejemplo: el sistema de control de semáforos de una ciudad. El estado de cada uno de los semáforos (verde, rojo o ámbar) constituye los hechos del mundo real.

El programa en sí consiste en unas pocas reglas de sentido común: determinados semáforos no pueden permanecer simultáneamente en verde, un semáforo solamente puede transitar de verde a ámbar y de ámbar a rojo, etc. La hipótesis es el estado en el que deberían estar cada uno de los semáforos en el siguiente instante de tiempo.

Éste es un ejemplo imposible de resolver mediante programación tradicional, ya que la lógica subyacente al comportamiento de los semáforos en su conjunto queda enmascarada por simples órdenes imperativas del tipo "cambiar color de tal o cual semáforo".

PARADIGMA FUNCIONAL

Introducción

El objetivo del paradigma funcional es conseguir lenguajes expresivos y matemáticamente elegantes, en los que no sea necesario bajar al nivel de la máquina para describir el proceso llevado a cabo por el programa, y evitando el concepto de estado del cómputo. La secuencia de computaciones llevadas a cabo por el programa se regiría única y exclusivamente por la reescritura de definiciones más amplias a otras cada vez más concretas y definidas, usando lo que se denominan definiciones dirigidas.

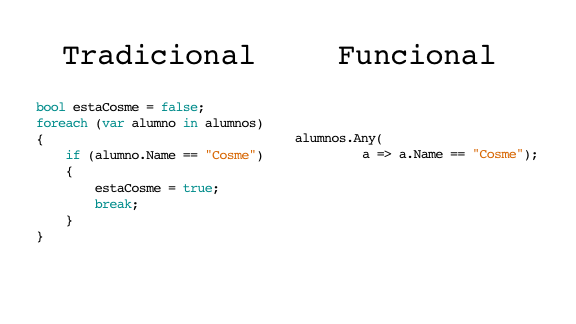
Todo esto con el objetivo de familiarizar a los estudiantes con un lenguaje elegante en el cual se pueda manejar más fácilmente y así los programas sean menos extensos y complejos.

Otro de los objetivos primordiales de dicho paradigma es buscar satisfacer las necesidades del usuario con respecto a operaciones matemáticas y convertirse en un lenguaje más expresivo.

Características:

Los programas escritos en un lenguaje funcional están constituidos únicamente por definiciones de funciones, entendiendo éstas no como subprogramas clásicos de un lenguaje imperativo, sino como funciones puramente matemáticas, en las que se verifican ciertas propiedades como la transparencia referencial (el significado de una expresión depende únicamente del significado de sus subexpresiones), y por tanto, la carencia total de efectos laterales.

Otras características propias de estos lenguajes son la no existencia de asignaciones de variables y la falta de construcciones estructuradas como la secuencia o la iteración (lo que obliga en la práctica a que todas las repeticiones de instrucciones se lleven a cabo por medio de funciones recursivas).

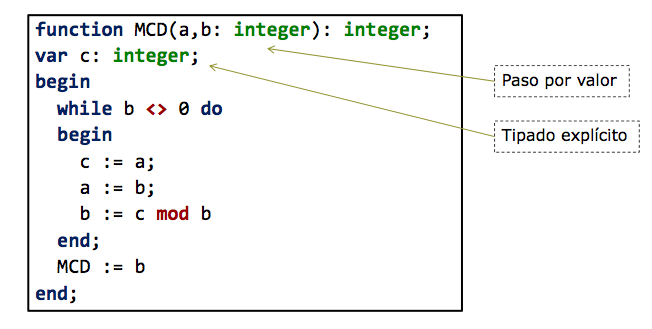


Existen dos grandes categorías de lenguajes funcionales: los funcionales puros y los híbridos. La diferencia entre ambos estriba en que los lenguajes funcionales híbridos son menos dogmáticos que los puros, al admitir conceptos tomados de los lenguajes procedimentales, como las secuencias de instrucciones o la asignación de variables.

En contraste, los lenguajes funcionales puros tienen una mayor potencia expresiva, conservando a la vez su transparencia referencial, algo que no se cumple siempre con un lenguaje funcional híbrido.

Entre los lenguajes funcionales puros, cabe destacar a Haskell y Miranda. Los lenguajes funcionales híbridos más conocidos son Lisp, Scheme, Ocaml y Standard ML (estos dos últimos, descendientes del lenguaje ML).

Programación funcional:



El componente básico de los lenguajes funcionales es la noción de función y su estructura de control esencial la aplicación de una función. Entre los lenguajes funcionales se encuentran ISWIM, ML, LISP y todos sus derivados, como Scheme. Las características fundamentales de los lenguajes funcionales de programación son, según [SET92]:

1. El valor de una expresión depende sólo de los valores de sus subexpresiones, si las tiene. La programación funcional pura es una programación sin asignaciones. En realidad, la mayoría de los lenguajes funcionales son impuros, ya que permiten asignaciones. Sin embargo, su estilo de programación es diferente al de los lenguajes de programación imperativa.

2. Almacenamiento implícito. El programador no debe preocuparse en manejar el almacenamiento de datos. Una consecuencia de esto es que la implementación del lenguaje debe realizar una "recolección de basura" para recuperar la memoria que se ha usado y no se volverá a utilizar.

3. Las funciones son valores de primera clase. Una función puede ser el valor de una expresión, pasarse como argumento o colocarse en una estructura de datos. Esto permite potentes operaciones.

Al decir que un lenguaje de programación es funcional, se suele hacer referencia al paradigma de programación funcional.

En términos general, al desarrollar software en el paradigma de programación, en contraste con el paradigma de programación imperativo, se tiende a enfatizar más la evaluación de expresiones que la ejecución secuencial de comandos.

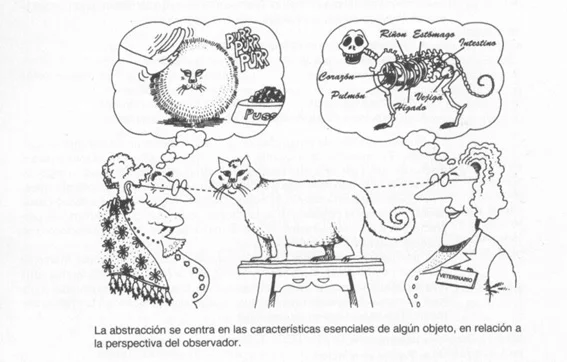
En el fondo, para ejecutar programas los computadores ejecutan, de manera secuencial, instrucciones de código de máquina. De cierta forma, ésto hace que los lenguajes imperativos sean más cercanos a la manera en que funcionan los computadores, al forzar a los programadores a formular sus algoritmos como una serie de pasos que se realizarán de manera secuencial.

PARADIGMA ORIENTADA A OBJETOS

El paradigma orientado a objetos o POO es una técnica por la cual pretendemos modelar nuestro mundo, pasarlo a un lenguaje que lo entienda el ordenador mediante los objetos. Estos objetos representan la funcionalidad de algo, y ese “algo” tiene la capacidad de interactuar, es decir, los objetos interactual con objetos. Es importante profundizar y tener claro lo que es el concepto de objeto.

El objeto en Java es una abstracción del mundo real, pero no deja de ser una abstracción nuestra.

Concepto de abstracción



En resumen: ¿Qué significa Orientación a Objetos? El software se organiza como una colección de objetos que contienen características (atributos o propiedades) y comportamiento (métodos).

Por ejemplo: Una granja será un objeto en Java si tiene funcionalidad, es decir, si a parte de tener atributos tiene metodos. Matizando, esa granja a parte de características deberá tener un comportamiento. En los artículos siguientes trataremos el ejemplo de la granja y desarrollaremos una aplicación se dedique a la gestión de animales en granjas.

Los lenguajes de programación orientados a objetos tienen su origen en un lenguaje que fue diseñado por los profesores Ole-Johan Dahl y Kristen Nygaard en Noruega. Este lenguaje de programación orientado a objetos fue el “Simula 67” que fue un lenguaje creado para hacer simulaciones de naves.

Los lenguajes de programación orientadas a objetos son lenguajes dinámicos en los que estos objetos se pueden crear y modificar sobre la marcha. Esta programación orientada a objetos (POO) tomo auge a mediados de los años ochenta debido a la propagación de las interfaces gráficas de usuarios, para lo que los lenguajes de programación orientados a objetos están especialmente dotados.

Los principales lenguajes de programación orientados a objetos son:

Ada, C++, C#, VB.NET, Clarion, Delphi, Eiffel, Java, Lexico (en castellano), Objective-C, Ocaml, Oz, PHP, PowerBuilder, Python, Ruby y Smalltalk.

No todos estos lenguajes de programación orientados a objetos son específicamente orientados a objetos. Sino que algunos de ellos se le han añadido extensiones orientadas a objetos.

PARADIGMA ORIENTADA A ASPECTOS

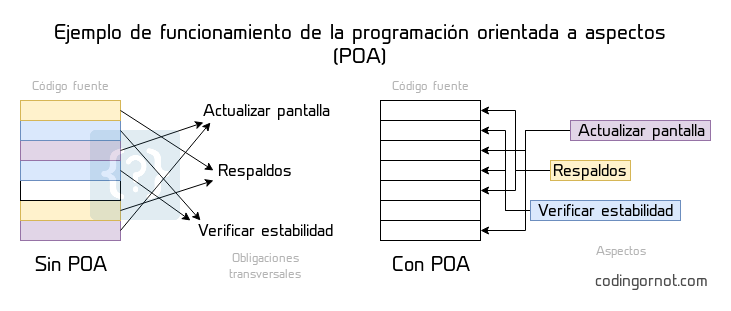
La programación orientada a aspectos (AOP - Aspect Oriented Programming) es un paradigma de programación que intenta formalizar y representar de forma concisa los elementos que son transversales a todo el sistema. En los lenguajes orientados a objetos, la estructura del sistema se basa en la idea de clases y jerarquías de clases.

La herencia permite modularizar el sistema, eliminando la necesidad de duplicar código. No obstante, siempre hay aspectos que son transversales a esta estructura: el ejemplo más clásico es el de control de permisos de ejecución de ciertos métodos en una clase:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | public class MiObjetoDeNegocio {     public void metodoDeNegocio1() throws SinPermisoException {          chequeaPermisos();          //resto del código          ...     }       public void metodoDeNegocio2() throws SinPermisoException {          chequeaPermisos();          //resto del código          ...     }       protected void chequeaPermisos() throws SinPermisoException {           //chequear permisos de ejecucion        ...     }  } |

Como vemos, estructurando adecuadamente el programa se puede minimizar la repetición de código, pero es prácticamente imposible eliminarla.

La situación se agravaría si además tuviéramos que controlar permisos en objetos de varias clases. El problema es que en un lenguaje orientado a objetos los aspectos transversales a la jerarquía de clases no son modularizables ni se pueden formular de manera concisa con las construcciones del lenguaje.



La programación orientada a aspectos intenta formular conceptos y diseñar construcciones del lenguaje que permitan modelar estos aspectos transversales sin duplicación de código. En nuestro ejemplo, se necesitaría poder especificar de alguna manera concisa que antes de ejecutar ciertos métodos hay que llamar a cierto código.

Lenguajes:

* [AspectC++](https://en.wikipedia.org/wiki/AspectC%2B%2B) es un compilador que permite desarrollar aspectos en [C++](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_C%2B%2B).
* [AspectJ](https://es.wikipedia.org/wiki/AspectJ) es una extensión [Java](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java) del proyecto [Eclipse](https://es.wikipedia.org/wiki/Eclipse_(software)) para ayudar en el desarrollo orientado a aspectos.
* [Aspect](http://search.cpan.org/perldoc?Aspect), un módulo [Perl](https://es.wikipedia.org/wiki/Perl) disponible en [CPAN](https://es.wikipedia.org/wiki/CPAN) para la Programación Orientada a Aspectos (en inglés).
* [PHP-AOP (AOP.io)](http://aop.io/) es una lib que proporciona todo el paradigma de la POA en PHP.
* phpAspect es una extensión PHP para implementar el paradigma de la POA, que, mediante árboles de decisión XML, realiza el weaving del software para ser ejecutado como PHP estándar.
* [FLOW3](https://es.wikipedia.org/w/index.php?title=FLOW3&action=edit&redlink=1) es un [framework](https://es.wikipedia.org/wiki/Framework) MVC de [PHP](https://es.wikipedia.org/wiki/PHP) incluye un [módulo](https://web.archive.org/web/20120831112649/http:/flow3.typo3.org/documentation/guide/partiii/aspectorientedprogramming.html) para poder realizar Programación orientada a Aspectos en nuevos desarrollos.
* [AOP con SpringFramework 2.5](http://static.springframework.org/spring/docs/2.5.x/reference/aop.html) es un [Framework](https://es.wikipedia.org/wiki/Framework) de [Java](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java) que permite programar en el paradigma de Aspectos utilizando [Anotación Java](https://es.wikipedia.org/wiki/Anotaci%C3%B3n_Java).
* [Aspyct AOP](https://web.archive.org/web/20120906032119/http:/old.aspyct.org/doku.php?id=aspyct) es un módulo de [Python](https://es.wikipedia.org/wiki/Python) que permite incluir Programación orientada a Aspectos a programas ya existentes escritos en [Python](https://es.wikipedia.org/wiki/Python) o a nuevos desarrollos.

1. CONCLUSIONES

Después de hacer esta investigación acerca de los paradigmas de programación nos hemos dado cuenta que todos los paradigmas juegan un papel importante en el mundo de la programación ya que a través de estos podemos hacer un excelente planeamiento para desarrollar un software.

Nos estructuran nuestra programación y nos dicen cómo resolver los problemas si es a través de sentencias lógicas o matemáticas.

También ahora se pueden comprender mejor las diferentes definiciones de cada tipo de paradigmas. Los lenguajes de programación a veces pueden tener influencias de otros tipos de paradigmas.

1. REFERENCIAS
2. <https://www.4rsoluciones.com/blog/que-son-los-paradigmas-de-programacion-2/>
3. <https://rootear.com/desarrollo/poo>
4. <https://medium.com/@Loopa/paradigmas-de-programaci%C3%B3n-programaci%C3%B3n-imperativa-y-programaci%C3%B3n-declarativa-4c4a4182fd87>
5. <https://es.slideshare.net/JFREDYOLAYARAMOS/paradigma-imperativo-39302522>
6. <http://enciclopedia.us.es/index.php/Lenguaje_declarativo>
7. <https://sg.com.mx/revista/24/programacion-declarativa>
8. <http://wiki.uqbar.org/wiki/articles/paradigma-logico---introduccion.html>
9. <https://www.ecured.cu/Programaci%C3%B3n_l%C3%B3gica>