Unidad 1: Introducción POO

Cátedra Introducción a Programación Orientada a Objetos
Tecnicatura en Desarrollo de Aplicaciones Web
Facultad de Informática
Buenos Aires 1400 (8300) Neuquén
Universidad Nacional del Comahue, Argentina

Evolución de los lenguajes de programación

La historia de los lenguajes de programación se ha desarrollado en base a una sola idea central: hacer que la implementacion sea cada vez lo más simple, flexible y portable posible. La POO da un nuevo paso hacia ese fin. A medida que se van desarrollando los lenguajes, se va desarrollando también la posibilidad de resolver problemas cada vez más complejos. En la evolución de cada lenguaje, llega un momento en el que los programadores comienzan a tener dificultades a la hora de manejar programas que sean de un cierto tamaño y sofisticación. Esta evolución en los lenguajes, ha sido impulsada por:

—Los avances tecnológicos

—Los avances conceptuales

—Los avances en cuanto a enfoque de la programación

1. EVOLUCIÓN EN CUANTO A LA TECNOLOGÍA

Un ordenador no es más que un conjunto de microinterruptores que pueden estar apagados o encendidos. Si están apagados (no dejan pasar corriente), decimos que su estado es cero y en caso contrario uno.

Un programa no es más que una sucesión de instrucciones que se ejecutarán secuencialmente, es decir, una detrás de otra. Si programar consitiera en introducir secuencias de ceros y unos (lo que llamamos bits); la gran probabilidad que de error que existe al introducir sucesiones de 0 y 1 es muy alta; además, una vez cometido un error, intentar encontrarlo y corregirlo puede ser una tarea imposible.

Cronológicamente el primer avance importante vino con la aparición de los *lenguajes ensambladore*s. Estos lenguajes se los conoce como lenguajes de "**bajo nivel**", ya que estan vinculados a la forma de trabajo de la máquina con la que programamos.

Quizás se pueda pensar que no hay gran diferencia entre programar en hexadecimal (o en binario), y hacerlo a través de un ensamblador, y es cierto, pero este logro no lo es tanto en cuanto al modo de trabajo (que seguía siendo precario y tedioso) como en cuanto a la nueva concepción que ello implica: por primera vez lo que se programaba no era entendible directamente por la máquina, sino que debe ser previamente traducido para que la máquina lo entienda. Un lenguaje ensamblador lo único que hace es transcribir unos nemónicos (palabras fáciles de recordar) a la secuencia ceros y unos a los que el nemónico representa y que sí son entendibles por la máquina.

El problema de los lenguajes ensambladores es que estan fuertemente vinculados a la máquina. Un programa escrito en lenguaje ensamblador solo podrá ser ejecutado en la máquina donde fue diseñado.

El siguiente paso, vino con la aparición de los "**lenguajes de alto nivel**". El proceso de desarrollo de una aplicación con un lenguaje de alto nivel es mucho más rápido, simple y, por tanto, resulta más fácil detectar y corregir errores. Esto se debe principalmente a dos factores: por un lado, cada instrucción en un lenguaje de alto nivel puede equivaler a varias decenas e incluso cientos de instrucciones en ensamblador; por otra parte, la sintaxis de las instrucciones y los nemónicos que usamos se parecen algo más al lenguaje cotidiano.

Realizar un programa que en un lenguaje de alto nivel nos puede llevar unos días, en un lenguaje ensamblador es tarea de meses. Si tenemos esto en cuenta, comprenderemos que después de escribir un programa en código máquina y de haberlo depurado para eliminar los errores (bugs en terminología informática), a nadie le quedan ganas de ponerse a re-escribirlo para otra máquina distinta. Con lo que la portabilidad queda mermada

considerablemente. Hecho que no debe ocurrir con un lenguaje de alto nivel.

Los programas realizados con lenguajes de alto nivel, realizan los mismos procesos de un modo más lento que de haberlos escrito con uno de bajo nivel, este problema no es demasiado grave si analizamos la evolución en la potencia de cálculo de las máquinas en los últimos diez años. Salvo para programas muy específicos, o de gran rendimiento, no habría ningún problema, en escribir nuestras aplicaciones en un lenguaje de alto o bajo nivel.

En teoría, y solo en teoría, puesto que un lenguaje de alto nivel es independiente de la máquina donde se ejecute, el mismo código que escribimos para una máquina puede ser traducido al lenguaje hexadecimal en esa máquina o en otra que disponga de traductor para ese lenguaje. Por ejemplo, si yo escribo un programa en lenguaje BASIC para un ordenador, este mismo programa podría ejecutarse en cualquier otra máquina que disponga de traductor de BASIC. Pero aquí aparece la torre de Babel de la informática. Existen traductores de lenguaje BASIC para multitud de máquinas diferentes, el problema reside en que cada máquina tiene un dialecto del lenguaje BASIC distinto a los demás, con lo que la portabilidad se hace imposible. Esto esta intimamente relacionado con cuestiones de marketting, política de empresas y sociedad de consumo y libre mercado, algo que tiene muy poco que ver con los investigadores y las buenas intenciones.

2. EVOLUCIÓN EN CUANTO A LA CONCEPTUALIZACIÓN

El primer avance en metodología de programación, vino con la **Programación Estructurada** (en este concepto vamos a incluir el propio y el de técnicas de Programación con Funciones -también llamado procedural-, ya que ambos se hallan íntimamente relacionados, no creemos, que se pueda concebir la programación estructurada sin el uso masivo de funciones).

La programación en ensamblador es lineal, es decir, las instrucciones se ejecutan en el mismo orden en que las escribimos. Podemos, sin embargo, alterar este orden haciendo saltos desde una instrucción a otro lugar del programa distinto a la instrucción que le sigue a la que se estaba procesando.

Este sistema de trabajo es complicado, ya que obliga al programador a retener en su cabeza permanentemente todo el código escrito hasta un momento determinado para poder seguir escribiendo el programa; además a la hora de leerlo, el programador se pierde con facilidad porque debe ir saltando continuamente de unos trozos de código a otros.

Al implementar un programa utilizando las técnicas de programación estructurada, los saltos no son una buena práctica de programación. Si bien el traductor (ya sea intérprete o compilador) cambia nuestro programa a código máquina, lo convierte a estilo lineal, pero eso es asunto de la máquina, nosotros escribimos y corregimos nuestro programa de un modo claro, y podemos seguir el flujo de la información con facilidad.

Lo que se intenta, es que el programador pueda hacer programas cada vez más extensos sin perderse en un entramado de líneas de código interdependientes. Un programa en estilo lineal se parece a la red neuronal del cerebro, si usted ha visto alguna vez una foto de estas, no lo habrá olvidado.

Para evitar esto, junto con la programación estructurada aparece un concepto que permite abarcar programas más amplios con menor esfuerzo: el de función. La idea es muy simple: muchas veces realizo procesos que se repiten y en los que sólo cambia algún factor, si trato ese proceso como un subprograma al que llamo cada vez que lo necesito, y cada vez que lo llamo puedo cambiar ese factor, estaré reduciendo el margen de error, al reducir el número de líneas que necesito en mi programa, ya que no tengo que repetir todas esas líneas cada vez que quiera realizar el proceso, con una sola línea de llamada al subprograma será suficiente; además, de haber algún fallo en este proceso el error queda circunscrito al trozo de código de la función.

Así, las funciones podemos entenderlas como unas cajas negras, que reciben y devuelven valores. Solo hay que programarlas una vez, se pueden testear por separado y comprobar que funcionan correctamente, una vez implementadas pueden ser reutilizadas cada vez que sea necesario.

Simultáneamente al concepto de función aparece el de variables de ámbito reducido. En un lenguaje estructurado, las variables son conocidas solo por aquellas partes del programa donde son definidas; pudiendo re-usar los nombres de las variables sin que haya colisión, siempre que estas se utilicen en ámbitos distintos.

POO - Programación Orientada al Objeto

Ofrece mucho mayor dominio sobre el programa liberándonos aún más de su control. Hasta ahora, el control del programa era tarea del programador, si usted ha realizado algún programa de magnitud considerable lo habrá padecido. El programador tenía que controlar y mantener en su mente cada proceso que se realizaba y los efectos colaterales que pudieran surgir entre distintos procesos, lo que llamamos colisiones.

En POO, el programa se controla a sí mismo y la mente del programador se libera enormemente pudiendo realizar aplicaciones mucho más complejas al exigir menor esfuerzo de atención, ya que los objetos son entidades autónomas que se controlan (si han sido creados correctamente) a sí mismos.

Esto es posible principalmente porque los objetos nos impiden mezclar sus datos con otros métodos distintos a los suyos. En programación estructurada, una función trabaja sobre unos datos, y no debería modificar datos

que no le corresponde hacer, pero de eso tiene que encargarse el programador, en POO es el propio sistema de trabajo el que impide que esto ocurra. Además, la re-usabilidad del código escrito es mucho mayor que con el uso de funciones, y las portabilidad es también mayor.

3. EVOLUCIÓN EN CUANTO AL ENFOQUE

La evolución de los lenguajes de programación, en cuanto a enfoque es también una evolución conceptual, pero ésta es tan profunda que supone un cambio drástico en cuanto al modo de concebir el tratamiento de la programación. En este sentido, y dependiendo del autor a quien se consulte, existen dos o tres enfoques diferentes:

—**Programación procedural**: Entre los lenguajes que trabajan de forma procedural encontramo: Java, C, Pascal, BASIC, Cobol, Fortran, APL, RPG, Clipper, etc. En ellos, se definen los datos y las instrucciones necesarias para el manejo de esos datos. Hay que especificar al lenguaje cómo alcanzar el objetivo que se persigue, es decir, donde buscar la información, cómo manipularla, cuando parar, etc.

—**Programación declarativa:** ProLog es un lenguaje de este tipo, este lenguaje fue desarrollado en la universidad de Marsella hacia 1970 por Alan Clomerauer y sus colegas. ProLog, se basa en manipulaciones lógicas, utiliza la lógica proposicional -lógica de predicados- para realizar sus deducciones. En ProLog se declaran hechos, que son manipulados por el lenguaje para extraer las conclusiones que resulten inferibles de estos hechos.

—**Programación orientada a objetos:** puede definirse como un conjunto de reglas a seguir para hacer mas fácil la tarea de programar. Un lenguaje de programación, es algo que todos más o menos conocemos: un conjunto de instrucciones entendibles directamente o traducibles al lenguaje del ordenador con el que trabajemos; combinando estas instrucciones realizamos programas. Para poder aplicar POO al 100 %, es necesario que el lenguaje nos proporcione una serie de mecanismos inherentes al propio lenguaje. En cualquier caso, la POO es casi 100 % procedural y, desde luego, no es en absoluto declarativa.

4. ¿QUÉ ES LA POO?

Como ya se ha comentado, la POO es un conjunto de técnicas que intervien en el proceso de producción de software; aumentando la productividad y permitiendo abordar proyectos de mayor envergadura. Usando estas técnicas, se incrementa criterios de calidad tales como el reuso ya que los objetos que se implementan quedan disponibles para futuros proyectos.

Hasta aquí, no hay ninguna diferencia con las funciones, una vez escritas, estas nos sirven siempre. La diferencia radica en que con POO es posible re-usar ciertos comportamientos de un objeto, ocultando otros o redefiniendolo para que los objetos se comporten de acuerdo a nuevas necesidades.

Por ejemplo si tenemos un coche y queremos que sea más rápido, no construimos un coche nuevo; simplemente le cambiamos el carburador por otro más potente, cambiamos las ruedas por otras más anchas para conseguir mayor estabilidad y le añadimos un sistema turbo. Pero seguimos usando todas las otras piezas de nuestro coche.

Desde el punto de vista de la POO ¿Qué hemos hecho? Hemos modificado dos **cualidades** de nuestro objeto (*métodos*): el carburador y las ruedas. Hemos añadido un *método* nuevo: el sistema turbo.

En programación tradicional, estabamos obligados a construir un coche nuevo por completo. Dicho en términos de POO, si se quiere construir un objeto que comparte ciertas cualidades con otro que ya tenemos creado, no tenemos que volver a crearlo desde el principio; simplemente, decimos qué queremos usar del antiguo en el nuevo y qué nuevas características tiene nuestro nuevo objeto.

Con POO podemos incorporar objetos que otros programadores han implementado, similar al armado de una estanteria obteniendo cada una de sus partes que seran ensambladas para obtener el resultado final. También podemos modificar el comportamiento de los objetos construidos por otros programadores sin tener que saber "como" se han construido.

Los objetos son protagonistas del paradigma de **Programación Orientada a Objetos**. Ya hemos utilizado objetos en Python sin mencionarlo explícitamente. Es más, todos los tipos de datos que Python nos provee son, en realidad, objetos.

A su vez, a las variables que un objeto contiene, se las llama atributos.

SABIAS QUE...

La Programación Orientada a Objetos introduce nueva terminología, y una gran parte es simplemente darle un nuevo nombre a cosas que ya estuvimos usando. Esto si bien parece raro es algo bastante común en el aprendizaje humano. Para poder pensar abstractamente, los humanos necesitamos asignarle distintos nombres a cada cosa o proceso. De la misma manera, para poder hacer un cambio en una forma de ver algo ya establecido (realizar un cambio de paradigma), suele ser necesario cambiar la forma de nombrar a los elementos que se comparten con el paradigma anterior, ya que sino es muy difícil realizar el salto al nuevo paradigma.