

## Programación y lenguajes

En estas notas se describen características generales de los programas y lenguajes de programación, así como de la programación orientada a objetos, llamada a veces por la sigla OOP (Object Oriented Programming).

### Programa de computador

Un *programa de computador* indica a un computador lo que debe hacer. En general, un programa de computador indica a un computador lo que debe hacer con ciertos *datos*. El programa se considera una entidad diferente de los datos. A veces se llama *software* a un programa de computador, pero el término software es también genérico para designar un conjunto de programas para una aplicación específica (a veces *paquete de software*), o aún para designar a todos los programas de computador, por contraposición a *hardware*, la parte física de la máquina.

En la mayoría de los computadores, un *sistema operativo* (conjunto de programas que permite funcionar al computador) se ocupa de cargar y ejecutar el programa.

La mayoría de los computadores actuales se construyen según la *arquitectura Von Neuman*: el programa se carga desde un dispositivo periférico, típicamente un disco duro, hacia la memoria principal de la máquina, luego se ejecutan las instrucciones del programa una a una secuencialmente, en el orden en que están escritas. Algunas instrucciones permiten cambiar la ejecución hacia otra instrucción del programa diferente de la siguiente en orden. Estas instrucciones se llaman *instrucciones de bifurcación*. También puede cambiar el orden de ejecución una *interrupción*, una señal dirigida al procesador de la máquina, originada en algún evento, como digitar una tecla.

Las instrucciones de un programa de computador, para poder ser ejecutadas, deben estar en *código de máquina*, lo único que una máquina de determinado tipo es capaz de entender. Los lenguajes de máquina para procesadores de diferentes tipos y marcas son, en principio, totalmente distintos. El lenguaje de máquina es difícil e incómodo para los humanos, por lo que los programas se escriben en un lenguaje de programación de computador más fácilmente comprensible. El programa así escrito en lenguaje de computador se denomina *fuentes* o *código fuente*. El programa escrito en lenguaje de programación de computador debe traducirse a código de máquina para poder ejecutarse, ya sea en un paso previo todo de una vez o mientras se va ejecutando.

Muchos programas de computación corren hoy día en máquinas virtuales. Una *máquina virtual* es también un programa, pero admite un conjunto de instrucciones uniforme construido encima de cada plataforma (hardware más sistema operativo). El programa original corre dentro de la máquina virtual, quien a su vez ejecuta las instrucciones propias del computador de base. Esto permite escribir programas de computador que corran en diferentes tipos de máquina, con sólo disponer del programa de máquina virtual para ese tipo de máquina. La capacidad de un mismo programa de computador escrito en un lenguaje de programación para correr en diferentes tipos de máquina se denomina *portabilidad*.

### Lenguaje de Programación

Un *lenguaje de programación* o *lenguaje de computación* es una forma normalizada de escribir instrucciones para un computador. Está compuesto por un conjunto *lexicográfico* (caracteres alfanuméricos, puntuación, otros) más un conjunto de reglas *sintácticas* (forma de escribir) y *semánticas* (significado de lo escrito). Un lenguaje de programación permite especificar con toda precisión los datos sobre los cuales actuará el programa de computador, cómo esos datos serán transformados, almacenados y transmitidos, previendo asimismo acciones a realizar en diferentes circunstancias.

El propósito primario de un lenguaje de programación es permitir a un programador expresar

sus propósitos computacionales más fácilmente que en código de máquina o en lenguajes de bajo nivel cercanos al código de máquina. Una instrucción en lenguaje de programación generalmente se traduce en muchas instrucciones en código de máquina.

Aunque se hicieron varios intentos, no ha sido posible crear un lenguaje de programación universal útil para cualquier propósito. La variedad de lenguajes de programación halla sus razones en la gran diferencia de tareas a encarar (comerciales, científicas, animaciones, juegos); el adiestramiento requerido, por haber lenguajes de programación muy difíciles de aprender y usar, junto a otros muy sencillos; las preferencias de los programadores; la diferencia de costo en tiempo al correr en un pequeño microcontrolador o en un supercomputador, según las aptitudes y complejidad de los diferentes lenguajes.

Existen lenguajes de propósito específico para ciertas aplicaciones: PHP para desarrollo de sitios web, Perl para manipulación de textos, C para construir sistemas operativos y compiladores (llamada *programación de sistemas*).

## Lenguajes interpretados y lenguajes compilados

Los lenguajes de programación hacen al programa menos dependiente del tipo de máquina o el ambiente (sistema operativo) donde se ejecutan: no son ejecutados directamente por la máquina, sino convertidos al código de máquina propio de cada tipo de máquina. Según la forma en que se realiza esta traducción, los lenguajes pueden ser interpretados o compilados.

Si la traducción del lenguaje de programación a código de máquina se realiza como una tarea previa, toda de una vez, y luego se corre el código de máquina (programa ejecutable) así generado, el proceso se denomina *compilación*, y el programa que realiza la traducción se llama *compilador*. El compilador es un programa capaz de tomar el código escrito en lenguaje de programación comprensible por humanos, llamado *código fuente*, y generar el programa en *código objeto*, que puede ser código de máquina ejecutable directamente por el procesador de la máquina, o código coincidente con una especificación de máquina virtual.

Si la traducción se realiza durante la ejecución del programa, convirtiendo cada paso del código fuente en código de máquina y ejecutándolo inmediatamente, se dice que el lenguaje es *interpretado*, y el programa que realiza la traducción se denomina un *intérprete*. El intérprete es capaz de tomar el programa escrito en lenguaje de programación comprensible por humanos e ir traduciendo y ejecutando cada paso sin proceso previo alguno; se va ejecutando a medida que se traduce. Aunque las definiciones no son idénticas, los lenguajes interpretados suelen llamarse también *lenguajes de scripting*; un *script* es un programa escrito y corrido en forma inmediata, frecuentemente usado en tareas de administración de sistemas. La denominación lenguaje de scripting para referirse a un lenguaje interpretado es equívoca; un lenguaje interpretado puede permitir escribir programas tan largos y complejos como un lenguaje compilado.

## Características de los lenguajes de programación

Un lenguaje de programación es un conjunto de especificaciones formales de vocabulario (léxico), forma de escribir (sintaxis) y significado (semántica). Estas especificaciones suelen incluir:

- Datos y Estructuras de Datos.
- Instrucciones y Control de Flujo.
- Mecanismo de Referencias y Reuso.
- Filosofía de Diseño.

## Datos y Estructuras

En un computador los datos se guardan internamente como secuencias de estados sí – no (estados binarios). No obstante, estos datos normalmente representan información del mundo real tal como números de cuenta bancaria, nombres de personas o medidas. Los lenguajes de programación organizan estos datos binarios en conceptos de alto nivel relacionados con el mundo real.

La forma en que cada lenguaje de programación organiza los datos binarios en conceptos de alto nivel se denomina el *sistema de tipos* de ese lenguaje, donde “tipos” debe entenderse como “tipos de datos”. Muchos lenguajes tienen tipos estáticos de datos; son tipos predefinidos para datos individuales como números enteros o de punto flotante de cierta precisión y tamaños máximos, caracteres y cadenas de caracteres, valores booleanos. Para cada tipo de datos los lenguajes de programación definen ciertas operaciones posibles. Los nombres de variables usados para designar valores pueden estar asociados a un único tipo de datos, y pueden participar sólo en las operaciones habilitadas para ese tipo de datos.

La mayoría de los lenguajes disponen también alguna forma de generar estructuras más complejas de datos definidas en base a combinaciones de los tipos de datos básicos, llamadas *estructuras de datos*, a las cuales puede asociarse un nombre distintivo. Esto permite luego declarar un nombre de variable cuyo tipo de dato es esa nueva estructura de datos, invocando su nombre. Un tipo de datos Persona puede estar compuesto por Nombre, Apellido, Dirección, siendo estos tres datos cadenas de caracteres.

Los lenguajes Orientados a Objetos permiten definir entidades de software denominadas *objetos*, en las cuales coexisten un conjunto de variables de ciertos tipos y también funciones capaces de realizar diferentes acciones involucrando los valores de esas variables. Las variables definidas dentro de un objeto se llaman *atributos* del objeto y las funciones definidas dentro de un objeto se llaman *métodos*. Un objeto se define de tal manera que representa muy cercanamente cualidades y comportamientos de algún objeto o entidad del mundo real o conceptual. Los objetos en un programa pueden operar e interactuar entre sí como subprogramas independientes. Estas interacciones se diseñan para reproducir en un modelo las interacciones entre objetos o conceptos del mundo real.

La definición de las estructuras de datos es una decisión de diseño de la mayor importancia. Definidas las estructuras de datos, los algoritmos mediante los cuales se manejarán esas estructuras surgen más o menos fácilmente, tanto más cuanto más acertada haya sido la definición de estructuras para el problema en particular.

## **Instrucciones y Control de Flujo**

Una vez definidas las estructuras de datos, deben indicarse las acciones a realizar sobre ellos. Acciones simples se expresan en *sentencias* usando palabras clave para designar las acciones, o mediante construcciones gramaticales bien definidas. Cada lenguaje permite a su vez combinar estas sentencias agrupándolas de diferentes maneras. Además de las sentencias de manipulación de datos, existen sentencias de control de flujo para estructurar acciones repetitivas, acciones alternativas dependiendo de alguna condición, acciones diferentes según diferentes casos, y otras.

## **Mecanismo de Referencias y Reuso.**

En lo esencial una *referencia* es una manera indirecta de designar cierto espacio de almacenamiento. La forma más simple es a través de un nombre. Los lenguajes de programación pueden tener diversas formas de indirección, en particular referencias que son punteros a otras áreas de almacenamiento.

También es posible asignar un nombre a un grupo de instrucciones. Esto permite escribir una sentencia donde se invoque la acción de un grupo de instrucciones usando su nombre simbólico. Las formas más comunes son los procedimientos y las funciones. Este uso de nombres simbólicos da al lenguaje flexibilidad y capacidad de reuso.

## **Filosofía de Diseño.**

Cada lenguaje ha sido concebido según una filosofía o diseño específico. Ciertos aspectos se destacan más según el uso de las estructuras de datos, o su notación favorece una determinada manera de resolver un problema o expresar su estructura.

Los lenguajes de programación son construcciones artificiales, por lo cual demandan una

especificación muy estricta de las operaciones deseadas. No tienen ninguna tolerancia a errores, pero tratan de informar al programador todo lo posible sobre la naturaleza de las anomalías detectadas.

## Programación Orientada a Objetos

La programación orientada a objetos considera a un programa de computador como una colección de unidades independientes, *los objetos*, cada uno de los cuales incluye estado y comportamiento. Los objetos son capaces de intercambiar mensajes entre sí, los cuales son atendidos por diferentes partes del código, realizando de esta forma las tareas esperadas de ese programa.

Los *objetos* combinan estado (sus datos o *atributos*) y comportamiento (procedimientos o *métodos*) en una entidad única distinguida de manera unívoca por un *identificador*. Un objeto responde a un mensaje recibido de otro objeto ejecutando el método correspondiente a ese tipo de mensaje.

El modelo más común de orientación a objetos es el basado en *clases*. Un objeto se construye tomando como modelo una clase. Una *clase* es una definición abstracta de atributos y métodos propios de esa clase, identificada con un *nombre de clase*. Un objeto es explícitamente construido a partir de una clase determinada, de la cual se dice es una *instancia*. Para fijar ideas, puede pensarse en una clase como un molde, y cada instancia de esa clase como un objeto (físico) moldeado usando ese molde. Para resumir: *un objeto es una instancia de una clase*.

Pueden crearse tantos objetos de una clase como se quiera. Cada objeto tendrá sus atributos, sus métodos y un *identificador de objeto* para distinguirlo de todos los demás objetos.

Más formalmente, la Programación Orientada a Objetos es un paradigma de programación en el cual un sistema de software se modela como un conjunto de objetos que interactúan unos con otros. La programación orientada a objetos se reconoce por estas características [PFL02, 6.1]:

- *Identidad*: datos organizados en entidades discretas, distinguibles, llamadas objetos. Un *objeto* tiene estado (datos o *atributos*) y comportamiento (procedimientos o *métodos*); se identifica con un *nombre* o *referencia* (“handle”, mango, manija o asa) por la cual un objeto se distingue de todos los demás. El objeto empaqueta juntos los datos y la funcionalidad; los objetos son la base de la modularidad y la estructura en un programa orientado a objetos.
- *Abstracción*: la capacidad de un programa para ignorar algunos aspectos de la información que está manipulando, enfocando en lo esencial. Cada objeto en el sistema sirve como modelo abstracto capaz de realizar tareas, informar su estado, cambiar su estado y comunicarse con otros objetos del sistema sin mostrar al exterior cómo se han implementado estas capacidades. La abstracción es una técnica que ayuda a determinar qué información debe ser visible y cuál debe ser oculta.
- *Clasificación*: agrupa objetos con características (atributos) y comportamiento (métodos) similares en una clase. La *clase* define los atributos (nombres y tipos de datos) y métodos (codificación de funciones) comunes a todos los objetos de la clase.
- *Encapsulamiento*: asegura que los usuarios de un objeto no puedan cambiar el estado interno de ese objeto de maneras inesperadas; sólo los métodos internos del objeto tienen permitido el acceso a su estado. Cada clase expone hacia el exterior una interfaz que especifica cómo las otras clases pueden interactuar con la clase actual. Esto permite al programador de la clase cambiar la implementación interna sin afectar el código que hace uso de esa clase, mientras mantenga inalterada la interfaz. La *interfaz* es la parte visible exteriormente del objeto, muestra cómo se puede hacer para interactuar con él. La *implementación* es la construcción interna del objeto, sus atributos y métodos. Un objeto Vehículo puede tener un atributo LitrosCombustible, invisible al usuario, y un método CargarCombustible(litros) visible al usuario; invocar el método CargarCombustible de ese objeto equivale a *enviar un mensaje* a ese objeto. La interfaz es la expresión CargarCombustible(litros), que nos indica la forma de enviar el mensaje, con nombre (CargarCombustible) y parámetros (cantidad de litros). El encapsulamiento busca el

agrupamiento y empaquetamiento de información relacionada (cohesión); no tiene funciones de seguridad. En esto difiere del *ocultamiento de información*; el encapsulamiento oculta o no según las necesidades.

- **Polimorfismo:** uso de un mismo nombre para invocar operaciones distintas sobre objetos de diferente tipo. Diferentes clases de objetos pueden tener la misma interfaz o responder el mismo mensaje (basado en el nombre del mensaje) y actuar de manera apropiada según la naturaleza de ese objeto. Por ejemplo un objeto de clase Avion puede recibir un mensaje Acelerar(valor) e implicará aumentar el empuje de sus reactores; un objeto de la clase Automóvil puede recibir el mensaje Acelerar(valor) e implicará aumentar las revoluciones del motor y/o cambiar de marcha. Ambos responden el mismo mensaje, pero de manera apropiada a la naturaleza de cada Vehículo. El polimorfismo permite incorporar nuevas clases sin cambiar el código ya existente.
- **Herencia:** permite organizar las clases jerárquicamente según las semejanzas y diferencias entre ellas, en una estructura donde se comienza definiendo una clase lo más amplia posible para luego definir subclases más especializadas de esa clase amplia. Una subclase hereda los atributos y métodos de su superclase, pudiendo a su vez incorporar atributos y métodos propios o modificar los heredados. La herencia organiza y facilita el polimorfismo y el encapsulamiento permitiendo definir objetos como tipos más especializados de objetos ya existentes. Estos pueden compartir y extender su comportamiento sin implementar nuevamente ese comportamiento. Las subclases se definen como extensión de clases preexistentes. Puede definirse una clase Vehículo, y luego Avión y Automóvil como extensiones de la clase Vehículo, agregando cada una sus particularidades a las generalidades compartidas por todos los vehículos.
- **Persistencia:** capacidad del nombre, estado y comportamiento para trascender en el espacio y en el tiempo; estas cualidades se conservan cuando el objeto es transformado. Un cambio de precio en el combustible Nafta puede requerir conservar el precio anterior para estadísticas, pero el objeto sobrevive aunque sus atributos hayan cambiado.

Algunas representaciones orientadas a objetos no exigen la persistencia o la clasificación; suele denominarse a estas representaciones *basadas en objetos*, en lugar de orientadas a objetos.

La Programación Orientada a Objetos se considera un paradigma más que un estilo o tipo de programación porque la orientación a objetos significa un cambio en el modo de desarrollar software, impone a los programadores e ingenieros de software una nueva forma de pensar, desde la creación del modelo del problema hasta la implementación de la solución.

El paradigma de OOP (Object Oriented Programming, Programación Orientada a Objetos) es esencialmente de diseño, no de programación. Se diseña un sistema definiendo las clases y objetos que existirán en ese sistema; el código que efectivamente realiza las tareas no importa al objeto ni a quienes usan el objeto, por el encapsulamiento. Lo difícil de OOP es entonces el diseño de un sistema de objetos adaptado al problema a resolver. Por ejemplo, en un sistema de objetos bien diseñado, el código asociado a un método suele ser breve y realizar una función bien específica; aún métodos complejos se programan haciendo referencia a otros métodos, que pueden ser de una superclase o de otra clase accesible, buscando así evitar redundancias y mantener cada método realizando una función bien específica.