Módulo 1 Lección 1

¿Qué es un programa en un lenguaje de programación?

Antes de empezar a programar en un lenguaje particular debes tener claro qué es un **programa**.

Un **programa** es, simplemente, un proceso de transformación de cero (o más) entradas en una (o más) salidas.

No siempre un programa recibe la entrada, trabaja, entrega su salida y termina; por ejemplo, en un videojuego la entrada la está dando continuamente el jugador y la salida es también continua y consiste en cambios de escenarios en la pantalla.

Entre otros tipos de problemas para los que queremos programas se encuentran:

- Una lista de alumnos para asentar ahí una calificación. Su entrada es, por un lado la lista de alumnos, que la proporciona la División de Estudios Profesionales, mientras que las calificaciones las proporciona el profesor en otro momento posterior.
- Multas: si deseo saber si al auto que manejo tiene o no multas levantadas, ingreso al sistema de placas del Distrito Federal, que es un programa. Recibe como datos las placas de todos los coches. En un segundo (tercer, cuarto,...) momento, un oficial de tránsito le aplica una infracción. Como salida tiene una carta dirigida al dueño del automóvil indicándole la multa. Si el dueño del automóvil accede al sistema, da como entrada la placa del auto que desea consultar y el sistema da como salida las multas acumuladas para ese automóvil.
- Generador de los primeros 100 números primos[1]. Este programa no recibe nada como entrada, pues trabaja siempre a partir del número 2 (una constante) y se lanza a verificar cuándo junta 100 números primos.

Si bien, como vimos en el último ejemplo, los programas pueden no recibir entradas, tienen forzosamente que producir al menos una salida. Si no lo hace, no nos vamos a enterar nunca si ya trabajó o no. Esta salida puede ser, como ya vimos:

- En los vídeo juegos, los distintos escenarios continuos.
- Puede ser una respuesta a una pregunta dada, como las multas.
- Puede ser información almacenada como el caso de una nómina o las listas de alumnos.

Como además quieres ser un programador responsable, debes cuidar que tu programa:

- Se ejecute bien (que haga lo que tiene que hacer), esto es que dé la solución correcta al problema. Por ejemplo, si tu sistema de placas contesta tu pregunta con las multas de una placa que no es la que le estás dando, el programa está mal.
- Que sepa defenderse de errores del usuario: Sería terrible que cada vez que alguien teclee una placa que no está en circulación, el sistema *"truene"* y tengas que volver a empezar porque te equivocaste al teclear.
- Si es que el programa no puede seguir adelante, debes preveer que tenga una *"muerte digna"*: le avise al usuario cuál es la razón por la que lo va a abandonar y terminar la ejecución.

[1] Un número primo es aquel donde sólo hay resultado de número entero cuando se divide entre sí mismo (el número que se está revisando) y cuando se divide entre 1 (uno).

- Por último el programa no puede quedarse dando vueltas sin recibir datos ni proporcionar salidas, o seguir pidiendo indefinidamente el mismo dato sin hacer nada al respecto: no debe "ciclarse" y siempre debe terminar.

Algoritmo

Éstos y otros conceptos más están asociados a una palabra que debes haber oído ya, la de *algoritmo*, que es un método de solución para un problema dado que, además de cero o más entradas y una o más salidas, debe cumplir con:

- Debes tener para este método de solución una descripción de cómo obtener la solución, que esté disponible y sea finita (con un número finito de instrucciones).

- Cada paso de tu solución debe poder realizarse en el contexto en el que lo estás planteando. El problema y la solución planteada determinan cuáles son sus características para poderse llevar a cabo. Por ejemplo, si le dices a un niño de dos años que lleve a cabo una suma de número arbitrarios, aunque le des las instrucciones, no lo va a poder hacer; o si le pides a mi esposo que haga un pastel, tampoco tiene él los mecanismos para llevar a cabo las tareas que son parte del método de solución.

- Por último, tu método de solución debe terminar de ejecutarse y dar la respuesta correcta.

No siempre tienes un algoritmo para responder a algunas preguntas, o la computadora se va a tardar tanto en responder (años o miles de años) que no tiene caso probar ese método de solución. Te inventas alguna otra solución, que aunque no sea *"la"* solución correcta sea una buena aproximación calculada u obtenida en mucho menos tiempo. A este último tipo de métodos de solución se les llama *heurísticas*.

Programas y algoritmos

La diferencia obvia es que un programa está escrito en un lenguajes de programación y lo puede ejecutar una computadora en mucho menos tiempo que una ser humano. El término *algoritmo* es más general y no forzosamente se refiere a programas, e incluye procesos que pueden ser ejecutados por una persona.

Los algoritmos y los programas se parecen en que son métodos de solución para determinados problemas y que los programas son una forma de escribir cierto tipo de algoritmos.

El proceso de programar

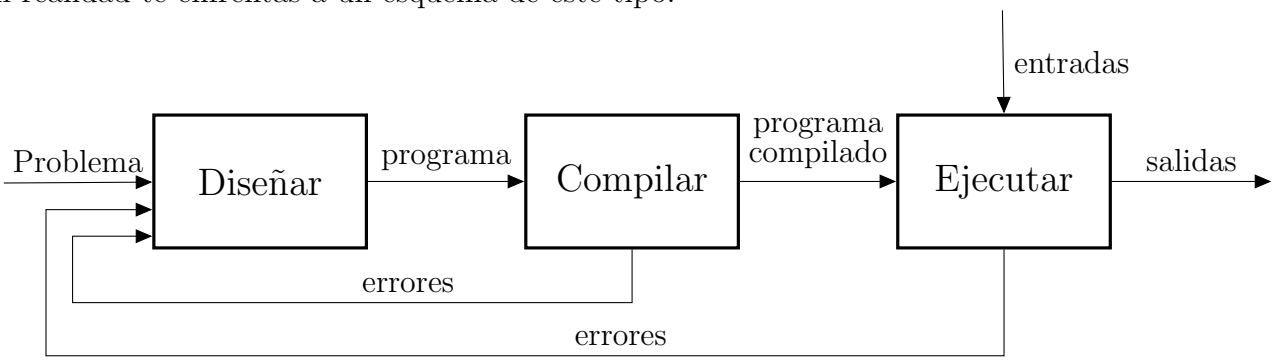
Si quieres resolver algún problema, primero tienes que describir la solución en forma de algoritmo, esto es, que cumpla con las características dadas. A este paso en el proceso le llamamos *diseñar* la solución. Sin embargo, la solución debe situarse en el tipo de lenguaje de programación que vas a usar para que el siguiente paso sea sencillo.

Una vez que tienes diseñada la solución, deberás *traducirla* al lenguaje de programación que hayas elegido. No es lo mismo un problema de gráficas, que de nómina que de videojuegos, y según el tipo de problema que deseas resolver, tu solución será más fácil de programar en cierta clase de lenguajes y no en otra. En el caso de este curso vamos a usar Java, que es un lenguaje del que decimos que es orientado a objetos (OO), por lo que el método de diseño que verás se ajusta a este tipo de lenguaje, para que la traducción del diseño al lenguaje Java sea sencilla.

Una vez que tienes tu programa elaborado y listo para el lenguaje Java, tienes que dárselo como entrada a un programa que se encarga de traducir tu programa a un lenguaje en el que trabaje la computadora. Lo único que sabe ejecutar la computadora es su propio lenguaje de máquina, que consiste de operaciones muy sencillas pero llevadas a cabo a una gran velocidad. AL programa que va a traducir tu programa escrito en Java a un lenguaje de máquina se le llama un *compilador* y te va a reportar *errores de sintaxis*: si es que estás *hablando* en Java, que estén bien formados tus enunciados, que estés hablando correctamente, que respete la sintaxis definida para Java.

Una vez que tu programa ya no tiene errores de la sintaxis de Java, ejecutas el programa para ver si te da los resultados que esperas. Java tiene forma de que escribas programas que verifiquen si tu programa hace lo que tiene que hacer, pero no los veremos en este módulo introductorio.

Realmente estos tres pasos se van repitiendo. Como resultado de la compilación y la ejecución es posible que requieras regresar a modificar tu diseño; o que al haber cambiado el diseño tengas que modificar el programa; también es posible que tu diseño no sea fácil de transcribir al lenguaje de programación en el que vas a trabajar, lo que también te puede obligar a modificar el programa. En realidad te enfrentas a un esquema de este tipo:



Organización en Java

Para solucionar un problema usando Java para ello, nos veremos en la necesidad, en general, de *usar*, incluir o *elaborar* varios programas, módulos o paquetes. Dos objetivos fundamentales de los lenguajes orientados a objetos es el poder compartir y reutilizar lo hecho por terceros, y hacer esto de manera hasta un poco "ciega", conociendo nada más las entradas que esperan y las salidas que proveen cada uno de los elementos de nuestro programa, módulo o paquete –a esto último se le llama *encapsulamiento*–. A cada una de estas unidades, unidas por compartir el problema que desean resolver, le llamaremos un *proyecto*.

Hoy en día para programar se usan ambientes de desarrollo integral (IDE por sus siglas en inglés[2]). Un IDE te permite realizar todas las tareas relacionadas con un proyecto particular (elaborar distintos programas o módulos, compilar, ejecutar, verificar errores, generar documentación, depurar, entre otras tareas) sin nunca salir de ese ambiente. Esto es muy cómodo no únicamente para aprender a programar sino también, y especialmente, para agilizar el desarrollo de nuevos proyectos. Como Java es un lenguaje muy utilizado en el mundo real, se han creado muchos ambientes de desarrollo para él. Muchos de estos ambientes son difíciles de usar para un neófito, por lo que elegimos DrJava, que está diseñado precisamente para *aprender a programar* y que se puede obtener e instalar de manera gratuita y fácil.

[2]**I**ntegrated **D**evelopment **E**nvironmente

Dicho lo anterior, resumimos cuál va ser tu proceso de programar una solución para un problema dado:

- Lo primero es pensar en una solución que puedas llevar a cabo con las herramientas que tienes.
- Como en tu caso la herramienta va a ser Java, tu solución deberá estar pensada en términos de la orientación a objetos.
- Realizarás un diseño de solución, que no es el proyecto en sí, pero si en realidad está pensado para la herramienta que vas a usar, será relativamente fácil de traducir. Para ello abordaremos una metodología sencilla para diseño orientado a objetos que se llama *de tarjetas de responsabilidades* y que podrás (deberás) hacer con papel y lápiz.
- Una vez que tengas el diseño, usando el IDE, traduces tu diseño al lenguaje de programación (esto lo escribes directamente, cada módulo en un archivo).
- Puedes ir pidiéndole al IDE que *compile* tu programa (lo traduzca a un programa ejecutable por la computadora), buscando ir escribiendo unidades completas susceptibles de ser compiladas. Conforme vas agregando texto a tu programa, vas viendo si el programa compila bien; también puedes escribir todo esa unidad y compilarla al terminar de escribirla.
- Al compilar el IDE te puede reportar errores de *sintaxis* –de forma–. Puede ser que se trate de una equivocación al teclear o de un problema de no haber cumplido con las reglas del lenguaje. Revisas de cuál de ellos se trata y lo corriges. Si no hay forma de corregirlo, seguramente tu diseño es equivocado y debes regresar a esa etapa.
- Una vez que el IDE no te reporta errores de sintaxis, le pides que ejecute tu unidad.
- La ejecución puede terminar antes de tiempo –le llamamos *abortar* o *tronar*– en cuyo caso deberás regresar al diseño o a la programación y ver qué puedes hacer para evitar que aborte.
- La ejecución puede terminar pero con la solución incorrecta. Deberás regresar al diseño o a la programación y ver qué puedes hacer para que dé la respuesta correcta.
- Si tu módulo recibe entradas, es conveniente probar que funciona bien incluso con entradas equivocadas, para ver que *se defiende* y, en todo caso, *muere dignamente*.
- En general, esta es la idea de la depuración[3], iterar sobre el diseño, la programación y la verificación hasta que el programa haga lo que se supone que debe hacer.

Nos vemos en la siguiente lección en la que te mostraremos con más detalle en qué consiste la orientación a objetos.

[3]En inglés, *debugging*