Introducción a la programación

Encarnación Sánchez Gallego

¿PRIMERA VEZ PROGRAMANDO?

Realizar test de conocimientos previos

ÍNDICE

Introducción

La razón principal por la que una persona utiliza un ordenador es para **resolver problemas** (en el sentido más general de la palabra), o en otras palabras, procesar una información para obtener un resultado a partir de unos datos de entrada.

Los **ordenadores** resuelven los problemas mediante la **utilización de programas** escritos por los programadores. Los programas de ordenador no son entonces más que métodos para resolver problemas. Por ello, para escribir un programa, lo primero es que el programador sepa resolver el problema que estamos tratando.

El programador debe identificar cuáles son los **datos de entrada** y a partir de ellos obtener los **datos de salida**, es decir, la solución, a la que se llegará por medio del procesamiento de la información que se realizará mediante la utilización de un método para resolver el problema que denominaremos **algoritmo**.

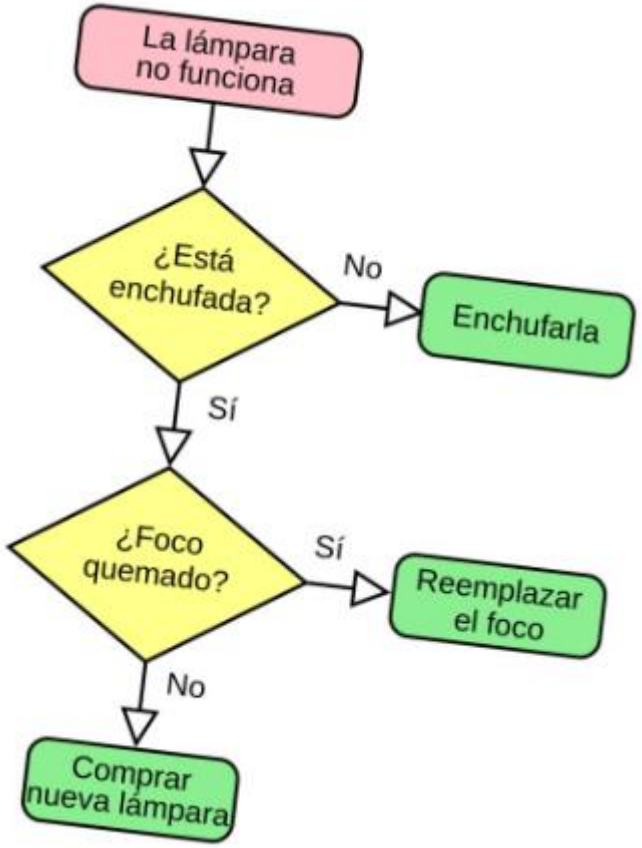
ÍNDICE

- Introducción
- **Algoritmo**
- Fases de un desarrollo
- Lenguajes de programación.
- Paradigmas
- Herramientas y entornos de desarrollo
- Lenguajes compilados e interpretados

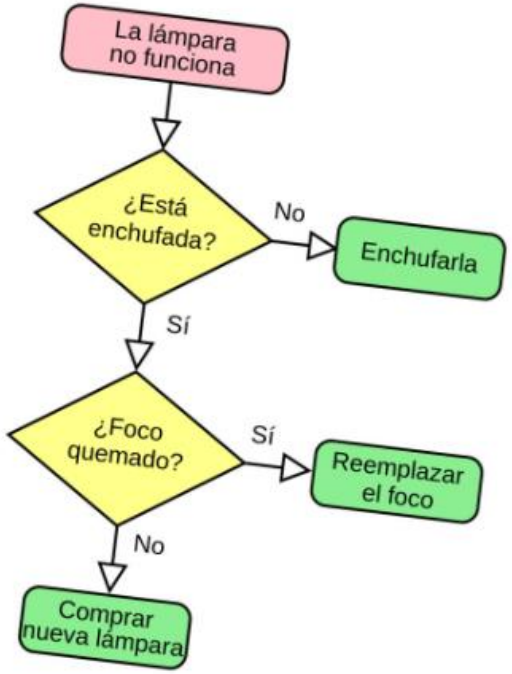
Algoritmo

Por algoritmo entendemos un conjunto ordenado y finito de operaciones que permiten resolver un problema que además cumplen las siguientes características:

- Tiene un número finito de pasos
- Acaba en un tiempo finito. Si no acabase nunca, no se resolvería el problema.
- Todas las operaciones deben estar definidas de forma precisa y sin ambigüedad.
- Puede tener varios datos de entrada y como mínimo un dato de salida.



Ejemplo de algoritmo



Vamos a definir el algoritmo para f**reír un huevo** podría ser el siguiente:

Datos de entrada: Huevo, aceite, sartén, fuego.
Datos de salida: huevo frito.

Procedimiento:

1. Poner el aceite en la sartén.
2. Poner la sartén al fuego.
3. Cuando el aceite esté caliente, cascar el huevo e introducirlo.
4. Cubrir el huevo de aceite.
5. Cuando el huevo esté hecho, retirarlo.

La codificación de un algoritmo en un ordenador se denomina **programa.**

Ciclo de vida de un programa

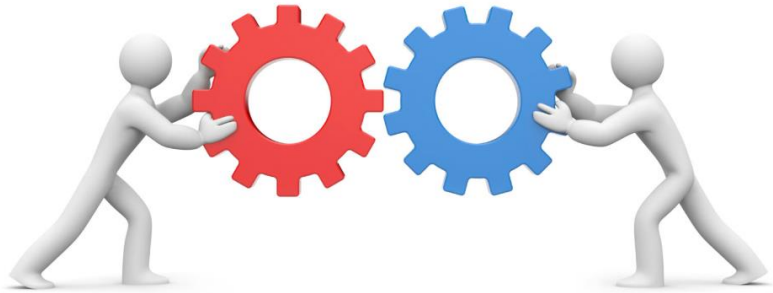
La creación de cualquier programa (software o sw) implica la realización de tres pasos genéricos:

- Definición ¿Qué hay que desarrollar?
- Desarrollo.
- Mantenimiento.

ÍNDICE

- Introducción
- Algoritmo
- **Fases de un desarrollo**
- Lenguajes de programación.
- Paradigmas
- Herramientas y entornos de desarrollo
- Lenguajes compilados e interpretados

Fase de definición



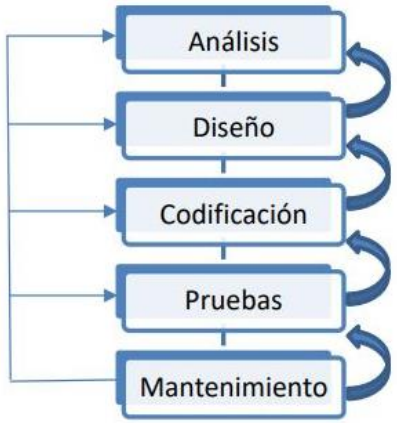
Se intenta **caracterizar** el sistema que se ha de construir. Se debe determinar la información que ha de usar el sistema, qué funciones debe realizar, qué condiciones existen, cuáles son las interfaces del sistema y qué criterios de validación se utilizarán.

El **estudio y definición del problema** dan lugar al planteamiento del problema que se escribirá en la **documentación del programa**. Si no se sabe lo que se busca, no se lo reconoce si se lo encuentra. Es decir que, si no sabemos con claridad qué es lo que tenemos que resolver, no podremos encontrar una solución. Aquí se declara cuál es la situación de partida y el entorno de datos de entrada, los resultados deseados, dónde deben registrarse y cuál será la situación final a la que debe conducir el problema después de ser implementado.

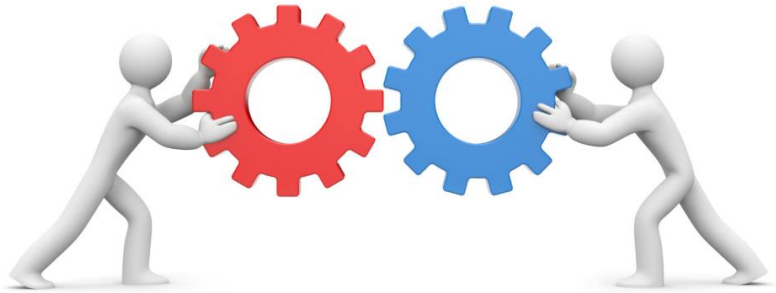
Fase de desarrollo

En esta fase se diseñan **estructuras de datos y de los programas**, **se escriben y documentan** éstos, y se **prueba el software**.

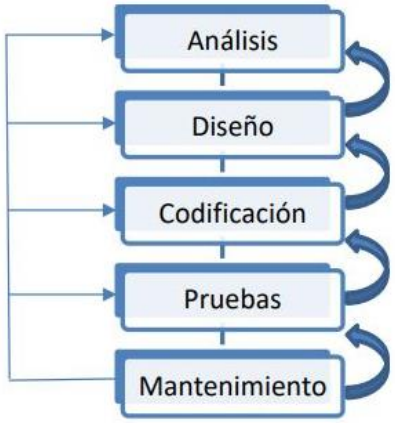
En esta etapa del ciclo de vida de desarrollo de programas, los analistas trabajan con los requerimientos del software desarrollados en la etapa de análisis. Se determinan todas las tareas que cada programa realiza, como así también, la forma en que se organizarán estas tareas cuando se codifique el programa.



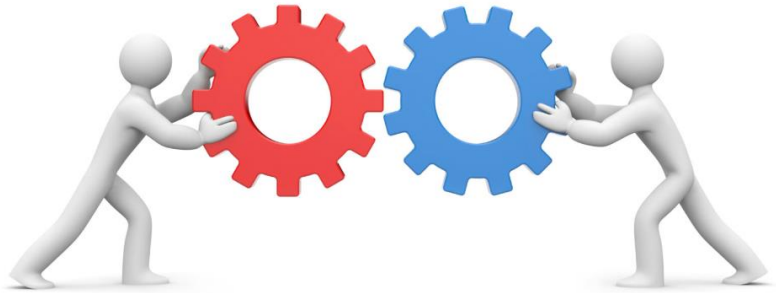
Fase de desarrollo



Los problemas cuando son **complejos**, se pueden resolver más eficientemente con el ordenador cuando se descomponen en **subproblemas** que sean más fáciles de solucionar que el original. La descomposición del problema original en subproblemas más simples y a continuación dividir estos subproblemas en otros más simples que pueden ser implementados para su solución en el ordenador se denomina **diseño descendente**.

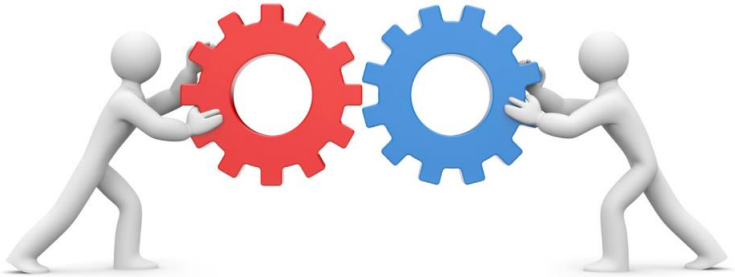


Fase de desarrollo



En esta fase, se convierte el algoritmo en programa, escrito en un lenguaje de programación de alto nivel como C, Java, etc. La codificación del programa suele ser **una tarea pesada** que requiere un conocimiento completo de las características del lenguaje elegido para conseguir un programa eficaz. Sin embargo, si el diseño del algoritmo se ha realizado en detalle con acciones simples y con buena legibilidad, el proceso de codificación puede reducirse a una simple tarea mecánica. Las reglas de sintaxis que regulan la codificación variarán de un lenguaje a otro y el programador deberá conocer en profundidad dichas reglas para poder diseñar buenos programas.

Fase de mantenimiento



Una vez obtenido el programa fuente, es necesaria su **traducción** al **código máquina**, ya que los programas escritos en un lenguaje de alto nivel no son directamente ejecutables por el ordenador.

Según el **tipo de traductor** que se utilice, los lenguajes de alto nivel se clasifican en **lenguajes interpretados y lenguajes compilados.**

ÍNDICE

- Introducción
- Algoritmo
- Fases de un desarrollo
- **Lenguajes de programación.**
- Paradigmas
- Herramientas y entornos de desarrollo
- Lenguajes compilados e interpretados

LENGUAJES DE PROGRAMACIÓN

Un lenguaje de programación es una notación para describir algoritmos y estructuras de datos para resolver problemas concretos en un ordenador.

Los lenguajes de programación se pueden clasificar

- Según la **evolución histórica**:

  https://www.youtube.com/watch?v=MtuqC0L2_S0

- Según su **nivel de abstracción**
- Según la **forma de ejecución**
- Según el **paradigma de programación**

Clasificación según su nivel de abstracción

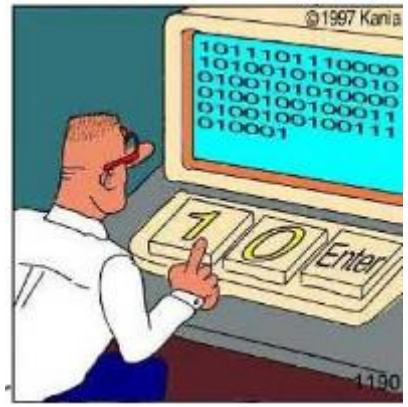
Son lenguajes de programación dependientes de **arquitectura de la máquina** que los soporta. Los programadores están obligados a conocer perfectamente la máquina, ya que estos lenguajes manejan directamente **recursos del sistema: memoria, registros del microprocesador …**

1.-LENGUAJE MÁQUINA

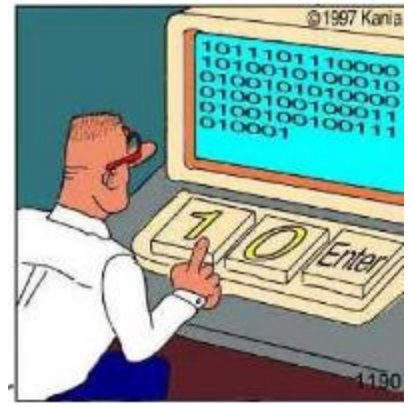
Es el lenguaje usado directamente por el procesador y está formado por un conjunto de **instrucciones codificadas en binario.**

La tecnología de la máquina tan sólo le permite entender en 0 y 1 (bits) y es de esa forma, como se codificaban las órdenes en los comienzos de la informática.

https://www.youtube.com/watch?v=KjBU3mNAfto



1.-LENGUAJE MÁQUINA



El segmento de código en lenguaje Java es:

int counter = 0;

counter = counter + 1;

podría ser traducido a lenguaje de máquina como:

00010100010001000100010000100010101011110

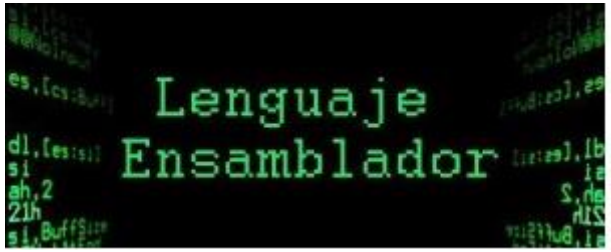
00000111010100011111000010001000010101010

Antes de los años 50, los programadores eran los encargados de introducir los códigos binarios correspondientes a las distintas operaciones que debía realizar la computadora.

Los programas eran totalmente dependientes de la máquina. Los programas en código máquina eran difíciles de leer e interpretar, por ello surgen los lenguajes de programación.

2.-LENGUAJE ENSAMBLADOR

Para paliar el uso del lenguaje máquina y como evolución de los mismos aparecen los lenguajes ensambladores. La idea surge del uso de **palabras mnemotécnicas**, en lugar de las largas secuencias de ceros y unos, para referirse a las distintas operaciones disponibles en el juego de instrucciones que soporta cada máquina en particular.




```
;name of the program:one.asm
;
.model small
.stack
.code
        mov AH,1h    ;Selects the 1 D.O.S. function
        Int 21h      ;reads character and return ASCII
                          ; code to register AL
        mov DL,AL    ;moves the ASCII code to register DL
        sub DL,30h   ;makes the operation minus 30h to
                          ; convert 0-9 digit number
        cmp DL,9h    ;compares if digit number it was
                          ; between 0-9
        jle digit1   ;If it true gets the first number
                          ; digit (4 bits long)
        sub DL,7h    ;If it false, makes operation minus
                          ; 7h to convert letter A-F digit1:
        mov CL,4h    ;prepares to multiply by 16
        shl DL,CL    ;multiply to convert into four bits upper
        int 21h           ;gets the next character
        sub AL,30h   ;repeats the conversion operation
        cmp AL,9h    ;compares the value 9h withthe content
                          ; of register AL
        jle digit2   ;If true, gets the second digit number
        sub AL,7h    ;If no, makes the minus operation 7h
                          ; digit2:
        add DL,AL    ;adds the second number digit
        mov AH,4CH
        Int 21h      ;21h interruption
        End     ;finish the program code
```

2.-LENGUAJE ENSAMBLADOR

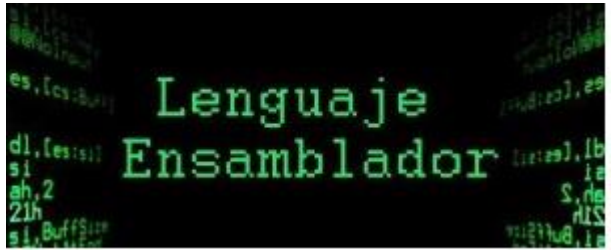
El ordenador no puede ejecutar directamente un programa escrito en lenguaje ensamblador, por lo que es indispensable un programa capaz de realizar la **traducción** al lenguaje máquina.

Este programa se conoce como Ensamblador siendo también específico de cada tipo de procesador.

Los programadores están obligados a conocer perfectamente la máquina, ya que estos lenguajes manejan directamente recursos del sistema: memoria, registros del microprocesador ...

Los programas son totalmente dependientes de la máquina

https://www.youtube.com/watch?v=KbEqb2utOFU



3. LENGUAJES DE ALTO NIVEL

Los programadores de la etapa anterior (hasta el 1959 no surge el manual de programación del UNIVAC) seguían obligados a pensar a la hora de diseñar los algoritmos en términos de instrucciones de máquina básicas, aún muy alejados de la manera natural en que nos comunicamos las personas.

Siguiendo en el camino de acercamiento hombre- máquina y en el intento de paliar los problemas derivados del uso de ensambladores, se desarrollaron los llamados lenguajes de alto nivel.

Están **orientados al programador**, en lugar de a la máquina, por lo que utilizan palabras del lenguaje natural (en inglés), por lo que son más fáciles de leer, escribir y modificar que los lenguajes de bajo nivel .

Son independientes de la máquina (aunque pueden incorporar instrucciones no estándares) y por tanto, el programador no necesita conocer el hardware específico de la máquina., C++, C#, Java, Logo, Python, Pascal, C...

CLASIFICACIÓN SEGÚN LA FORMA DE EJECUCIÓN

CLASIFICACIÓN SEGÚN LA FORMA DE EJECUCIÓN

1.- LENGUAJES COMPILADOS:

- Los algoritmos escritos en un lenguaje de alto nivel necesitan para su ejecución un programa llamado **compilador**, capaz de realizar la traducción al lenguaje máquina (Turbo Pascal, Turbo C, Borland C++, Visual C++).
- La **traducción** se efectúa de manera que cada instrucción escrita en lenguaje de alto nivel se transforma **en una o varias instrucciones en lenguaje máquina.**
- El compilador traduce completamente el texto escrito en lenguaje de alto nivel y una vez acabada la traducción, informa de los posibles errores. El programador deberá corregir esos errores y, sólo entonces, se genera la traducción lista para ejecutar.

CLASIFICACIÓN SEGÚN LA FORMA DE EJECUCIÓN

2.-LENGUAJES INTERPRETADOS:

- Existen lenguajes de alto nivel cuyos traductores, llamados Intérpretes, realizan lo que se podría llamar **traducción** simultánea, es decir, cada instrucción escrita en un lenguaje interpretado es analizada, traducida y ejecutada tras su verificación.
- La **principal diferencia** con los lenguajes compilados, es que una vez traducida cada instrucción, se ejecuta y a continuación se elimina. No se guarda una versión del programa traducido a código máquina.
- Suelen ser **más lentos** que los compilados, ya que han de ser traducidos mediante se ejecutan.
- Se necesita tener el intérprete ejecutándose en la máquina para ejecutar el programa, en los compilados no se necesita tener el compilador.
- Ejemplos de lenguajes interpretados: PHP, JavaScript, Python, Perl, Logo Rubi, ASP, Basic, etc.

CLASIFICACIÓN SEGÚN LA FORMA DE EJECUCIÓN

- Actualmente hay lenguajes en el mercado que se podrán llamar **pseudo-compilados o pseudo-interpretados,** por ejemplo Java. El código fuente se compila para obtener un código binario en forma de **byte-codes**, que son estructuras similares a las instrucciones de máquina, con la característica de no ser específicas de ninguna máquina en particular.
- Este código es interpretado por la máquina virtual Java, que es en definitiva quien lo traduce al código máquina del procesador.

3.-MÁQUINAS VIRTUALES DE PROCESO

- Las máquinas virtuales de proceso se utilizan para independizar la ejecución de un proceso del hardware subyacente, el más conocido es **la máquina virtual Java**.
- Es importante no confundir las **máquinas virtuales** de proceso con las máquinas virtuales de sistema, ejemplos de software para crear máquinas virtual de sistema son Vmware, Workstation y Virtual Box.

Ejercicio Lenguajes de Programación

Práctica:

Haz una clasificación de los lenguajes de programación que encuentres en internet, según lo comentado hasta ahora en clase.

ÍNDICE

- Introducción
- Algoritmo
- Fases de un desarrollo
- Lenguajes de programación.
- **Paradigmas**
- Herramientas y entornos de desarrollo
- Lenguajes compilados e interpretados

Clasificación según el paradigma de programación

Existe una enorme variedad de lenguajes de programación, no sólo en cuanto a su sintaxis, sino también en cuanto a su comportamiento o semántica.

Hemos visto que cada año el **número de lenguajes se incrementa**, de forma que para los informáticos es prácticamente imposible conocer cada nuevo lenguaje que aparece.

Pero eso no es un problema, ya que todos esos lenguajes tienen características comunes y se pueden agrupar en cuatro grandes grupos o modelos computacionales llamados **paradigmas**.

Todos los **lenguajes** pertenecen a uno de esos **cuatro paradigmas**. De forma que, si se conocen las características de los paradigmas de programación, es muy sencillo aprender a programar en un nuevo lenguaje, porque tendrá las características del paradigma de programación al que pertenezca.

Clasificación según el paradigma de programación

El origen de la palabra paradigma entendida como un marco general en el que se desarrollan teorías científicas se encuentra en el trabajo de 1962 del filósofo e historiador de la **ciencia Thomas S. Kuhn**, La estructura de las revoluciones científicas.

Esa palabra ha sido después adoptada por el mundo de la computación para definir un conjunto de ideas y principios comunes de grandes grupos de lenguajes de programación.

La definición de la palabra paradigma más cercana a lo que se quiere decir en la expresión paradigma de programación es la siguiente:

Un marco filosófico y teórico de una escuela o disciplina científica en el que se formulan teorías, leyes y generalizaciones y los experimentos realizados en soporte de ellas.

Clasificación según el paradigma de programación

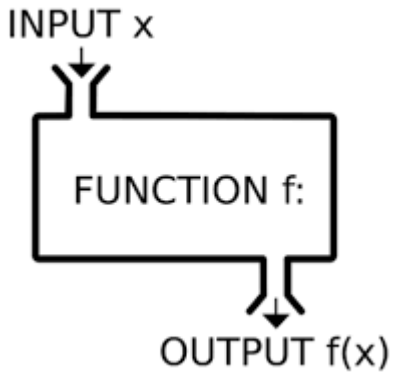
Un **paradigma** define un conjunto de **reglas, patrones y estilos de programación** que son usados por un grupo de lenguajes de programación.

Podemos distinguir cuatro grandes paradigmas de programación:

- Paradigma funcional
- Paradigma lógico
- Paradigma estructurado
- Paradigma orientado a objetos

Paradigma funcional

El programa es una colección de **funciones matemáticas** cada una de ellas con su entrada y su resultado. Las funciones interactúan y se combinan las unas con las otras utilizando **condicionales, recursivas y composición funcional**.



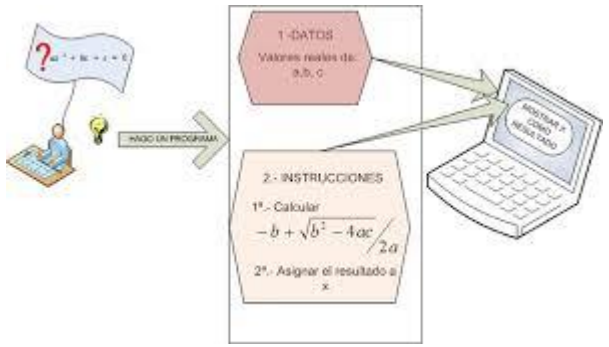
Paradigma funcional

Lenguajes: LISP, Scheme, Haskell, Scala, Clojure. Ejemplo de código (LISP):

```
(define (factorial x)
        (if (= x 0)
                1
                (* x (factorial (- x 1)))))
(factorial 8)
40320
(factorial 30)
265252859812191058636308480000000
```

Paradigma lógico (declarativo)

El programa es una colección de declaraciones lógicas sobre el resultado que debería conseguir una función, en lugar de cómo debería obtenerse dicho resultado. La ejecución del programa aplica estas declaraciones para obtener una serie de soluciones posibles a un problema. La programación lógica ofrece un vehículo natural para la expresión no determinista, **ya que las soluciones a muchos problemas no son únicas, sino múltiples.**



Paradigma lógico (declarativo)

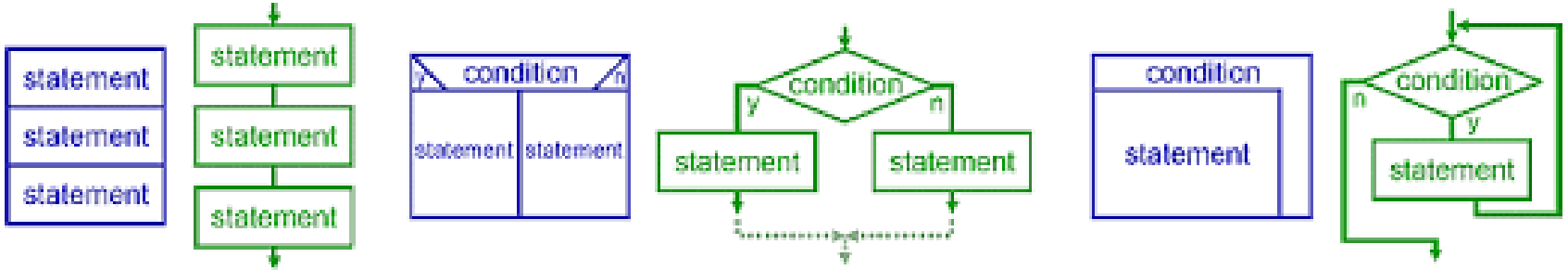
Lenguajes: Prolog, Mercury, Oz. Ejemplo de código (Prolog):

```
padrede('juan', 'maria'). % juan es padre de maria
padrede('pablo', 'juan'). % pablo es padre de juan
padrede('pablo', 'marcela').
padrede('carlos', 'debora').
hijode(A,B) :- padrede(B,A).
abuelode(A,B) :- padrede(A,C), padrede(C,B).
hermanode(A,B) :- padrede(C,A) , padrede(C,B), A \== B.
familiarde(A,B) :- padrede(A,B).
familiarde(A,B) :- hijode(A,B).
```

```
familiarde(A,B) :- hermanode(A,B).
?- hermanode('juan', 'marcela').
yes
?- hermanode('carlos', 'juan').
no
?- abuelode('pablo', 'maria').
yes
?- abuelode('maria', 'pablo').
no
```

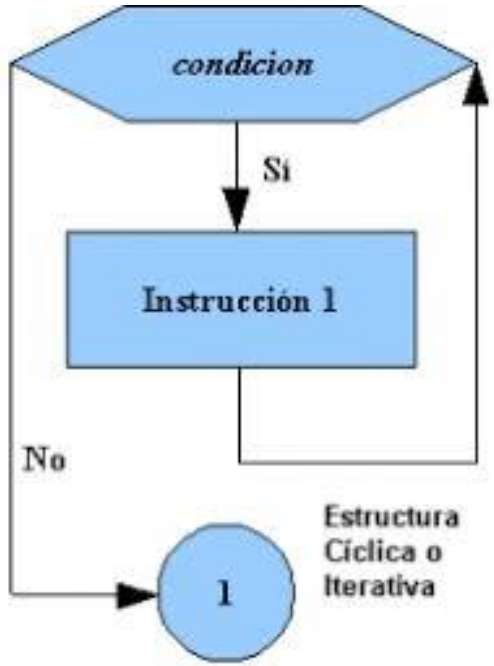
Paradigma estructurado

El programa es una serie de pasos, cada uno de cuales realiza un cálculo, recupera una entrada o tiene como resultado una salida. La abstracción procedimental es un bloque de creación esencial en la programación estructurada, al igual que las sentencias condicionales, asignaciones, bucles y secuencias.



Paradigma estructurado

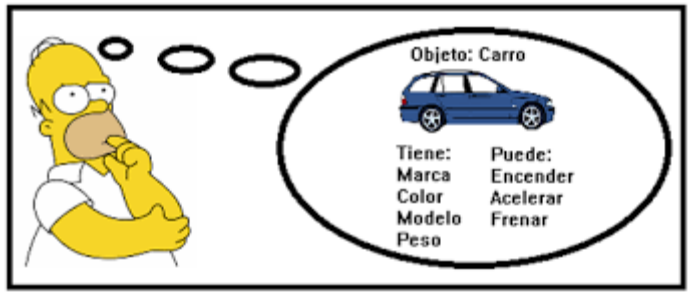
Lenguajes de programación estructurada son Cobol, Fortran, C.

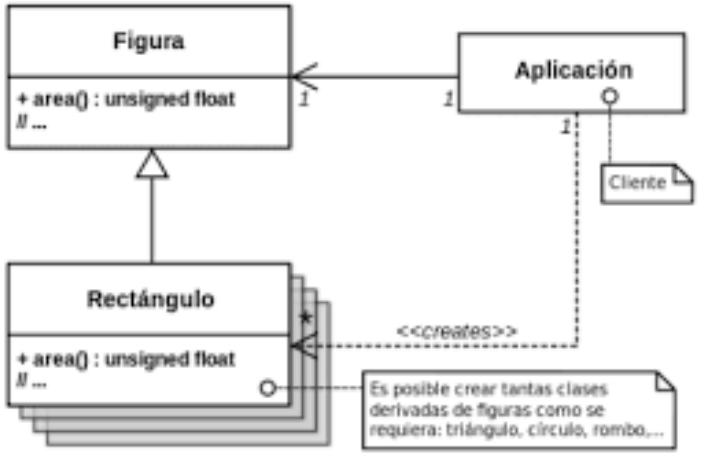



```
type
        tDimension = 1..100;
        eMatriz(f,c: tDimension) = array [1..f,1..c] of real;
        tRango = record
                f,c: tDimension value 1;
        end;
        tpMatriz = ^eMatriz;
procedure EscribirMatriz(var m: tpMatriz);
var filas,col : integer;
begin
        for filas := 1 to m^.f do begin
                for col := 1 to m^.c do
                        write(m^[filas,col]:7:2);
                writeln(resultado);
                writeln(resultado)
        end;
```

Paradigma orientado a objetos

El programa es una **colección de objetos** que interactúan los unos con los otros trasladándose mensajes que transforman su estado. Las clases y la herencia de objetos son bloques de creación esenciales de la programación OO.





Paradigma orientado a objetos

Lenguajes orientados a objetos

son Smalltalk, Java, C++, Eiffel.

Ejemplo (Java):


```
public class Bicicleta {
        public int marcha;
        public int velocidad;
        public Bicicleta(int velocidadInicial, int marchaInicial
                marcha = marchaInicial;
                velocidad = velocidadInicial;
        }
        public void setMarcha(int nuevoValor) {
                marcha = nuevoValor;
        }
        public void frenar(int decremento) {
                velocidad -= decremento;
        }
        public void acelerar(int incremento) {
                velocidad += incremento;
        }
}
```

Ejercicio Lenguajes de Programación en paradigmas

Práctica:

De los lenguajes de programación que has localizado en la actividad anterior, clasifícalos en los distintos paradigmas

ÍNDICE

HERRAMIENTAS Y ENTORNOS PARA EL DESARROLLO DE PROGRAMAS

Un entorno de desarrollo de software es una combinación de herramientas que automatiza o soporta al menos una gran parte de **la tareas (o fases)** del desarrollo: **análisis** de requisitos, **diseño** de arquitectura, diseño detallado, **codificación**, **pruebas** de unidades, **pruebas** de integración y validación, **gestión de configuración**, **mantenimiento**, etc. Las herramientas deben estar bien integradas, pudiendo interoperar unas con otras.

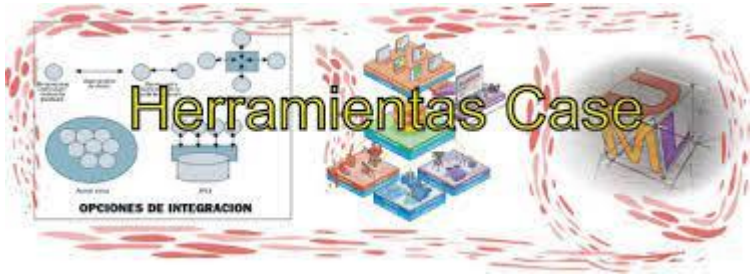
Están formados por el conjunto de **instrumentos** (hardware, software, procedimientos, ...) que facilitan o automatizan las actividades de desarrollo.

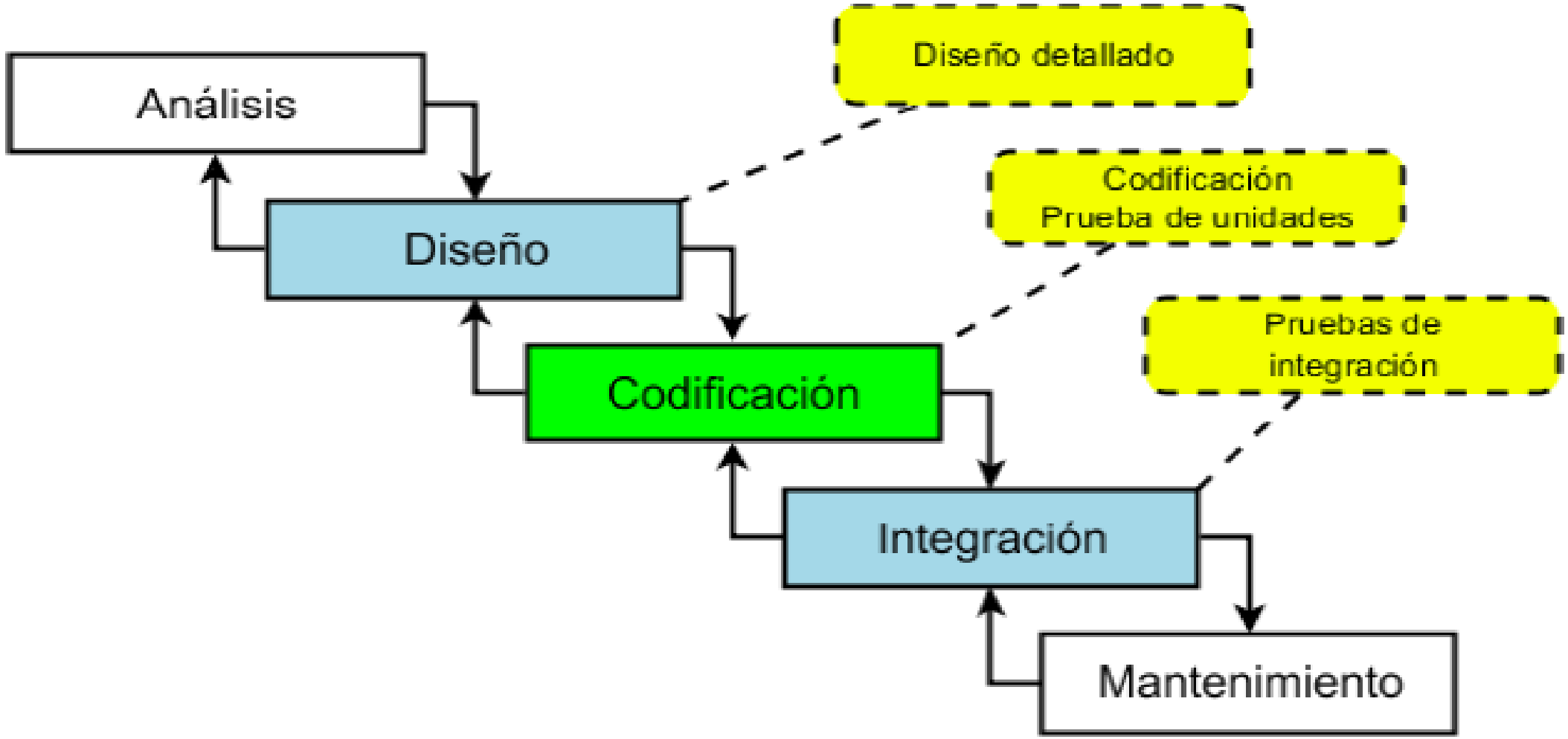
HERRAMIENTAS Y ENTORNOS PARA EL DESARROLLO DE PROGRAMAS

CASE: Computer-Aided Software Engineering

- Con este término genérico se denominan los productos software que dan soporte informático al desarrollo
- Sería deseable automatizar todo el desarrollo, pero normalmente se automatiza sólo en parte
- **Productos CASE**: son cada uno de los instrumentos o herramientas software de apoyo al desarrollo

Las actividades mejor soportadas por herramientas de desarrollo son normalmente la centrales: codificación y pruebas de unidades. El conjunto de herramientas que soportan estas actividades constituyen lo que se llama un entorno de programación. A veces se utilizan las siglas **IDE** (Integrated Development Environment) para designar estos entornos, aunque no son un entorno de desarrollo completo, sino sólo una parte de él.




Análisis
Diseño
Codificación
Integración
Mantenimiento
Diseño detallado
Codificación
Prueba de unidades
Pruebas de
integración

Funciones de un Entorno de Programación

Como se ha dicho, la misión de un **Entorno de Programación** es dar soporte a la preparación de programas, es decir, a las actividades de codificación y pruebas.

- Las tareas esenciales de la fase de codificación son:
- **Edición** (creación y modificación) del código fuente
- **Proceso/ejecución** del programa
- **Interpretación** directa (código fuente)
- **Compilación** (código máquina) - montaje - ejecución
- **Compilación** (código intermedio) - interpretación

Otras funciones:

- **Examinar** (hojear) el código fuente
- **Analizar** consistencia, calidad, etc.
- **Ejecutar en modo depuración**
- **Ejecución automática de pruebas**
- **Control de versiones**
- **Generar documentación**, reformar código
- ... y otras muchas más ...

ÍNDICE

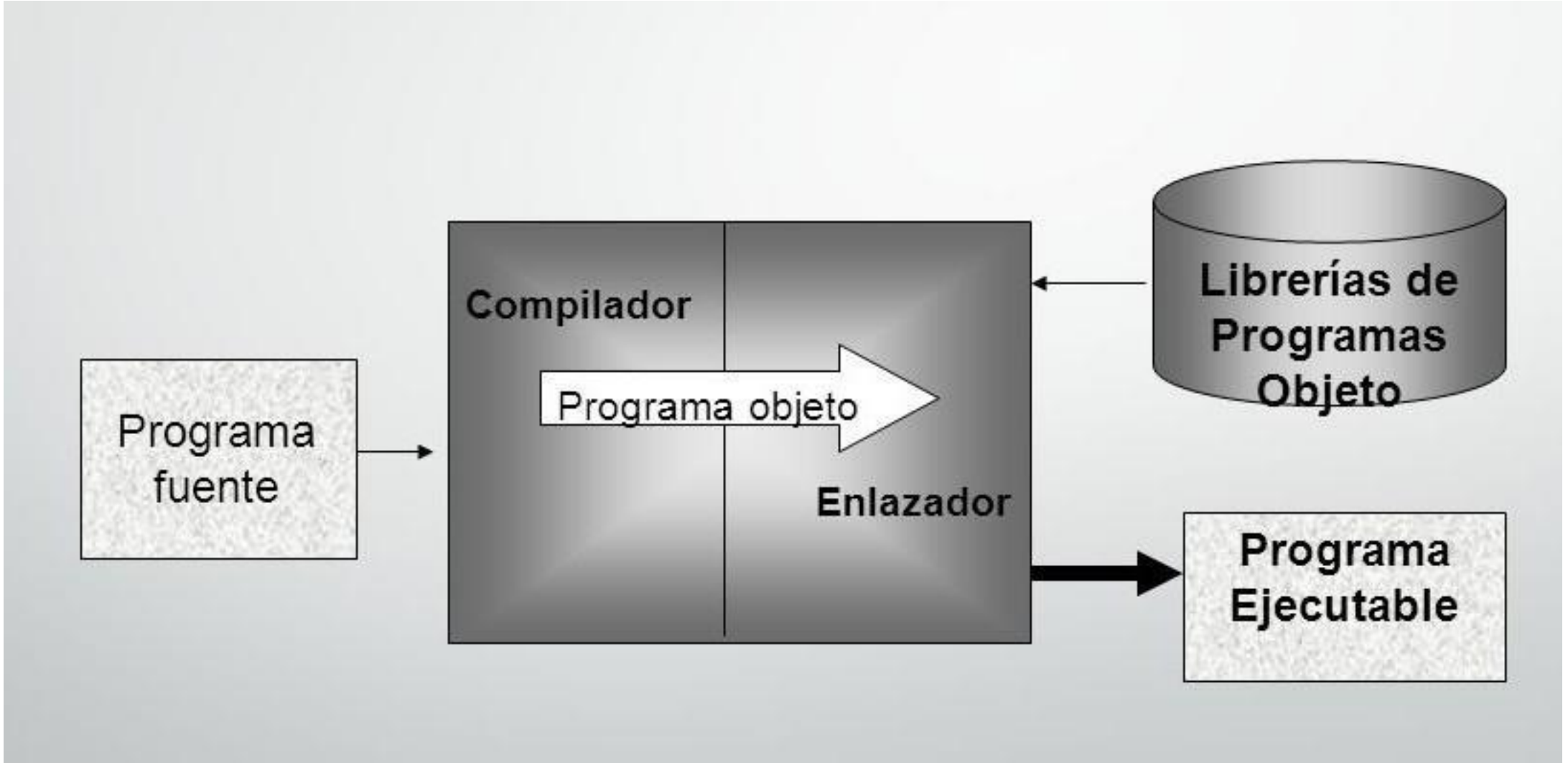
Lenguajes compilados y lenguajes interpretados

Son lenguajes **interpretados** aquellos en los que el sistema traduce una instrucción y la ejecuta, y así sucesivamente con las restantes.

Son lenguajes **compilados** aquellos en los que, primero se traduce el programa fuente completo, obteniéndose un código intermedio o módulo objeto (programa objeto); después, se fusiona éste con rutinas o librerías necesarias para su ejecución en un proceso llamado linkado y que obtiene como resultado un módulo ejecutable (programa ejecutable). La ventaja de los lenguajes compilados, frente a los interpretados, son su rápida ejecución y, en caso de necesitar posteriores ejecuciones del mismo programa, se hará del ejecutable almacenado.

COMPILADORES

- El proceso de traducción de un programa a código máquina se lleva a cabo mediante dos programas **compilador y enlazador**.
- Si el compilador en el proceso de **traducción** devuelve algún error (falta algún signo de puntuación, sentencias mal escritas, tipos de datos no compatibles, variables no definidas, etc) no se generará el programa objeto. Será necesario modificar el programa fuente y pasarlo de nuevo al compilador.


Compilador
Programa objeto
Enlazador
Librerías de Programas Objeto
Programa fuente
Programa Ejecutable

Compilador Java

```
1:   istore_1
2:   iload_1
3:   sipush  1000
6:   if_icmpge       44
9:   iconst_2
10:  istore_2
11:  iload_2
12:  iload_1
13:  if_icmpge       31
16:  iload_1
17:  iload_2
18:  irem             # remainder
19:  ifne    25
22:  goto    38
```

Intérprete JVM