

# 0.- Introducción

---

1	Problemas, algoritmos y programas. ....	2
1.1	Problemas .....	2
1.2	Algoritmos.....	3
1.3	Programas .....	6
2	Paradigmas de la programación. ....	6
2.1	Programación imperativa.....	6
2.2	Programación declarativa .....	7
3	Lenguajes de programación. El lenguaje Java .....	7
3.1	El lenguaje Java .....	7
3.2	Independencia de la plataforma .....	8
3.3	Applets, JavaScript y Aplicaciones Java.....	<b>¡Error! Marcador no definido.</b>
4	Herramientas y entornos de desarrollo. ....	9
4.1	El JDK.....	9
5	Errores en los programas.....	10
5.1	Errores de codificación (o sintácticos) .....	10
5.2	Errores de ejecución .....	10
5.3	Errores lógicos .....	10

# 1 Problemas, algoritmos y programas.

## 1.1 Problemas

Podríamos decir que la **PROGRAMACIÓN** es una forma de resolución de **PROBLEMAS**.

Para que un problema pueda resolverse utilizando un programa informático, éste tiene que poder resolverse de forma mecánica, es decir, mediante una secuencia de instrucciones u operaciones que se puedan llevar a cabo de manera “automática” por un **ORDENADOR**.

EJEMPLOS de problemas resolubles mediante un ordenador:

- Determinar el producto de dos números  $a$  y  $b$ .
- Determinar la raíz cuadrada positiva del número 2.
- Determinar la raíz cuadrada positiva de un número  $n$  cualquiera.
- Determinar si el número  $n$ , entero mayor que uno, es primo.
- Dada la lista de palabras, determinar las palabras repetidas.
- Determinar si determinada palabra pertenece al idioma castellano.
- Ordenar y listar alfabéticamente todas las palabras del castellano.
- Dibujar en pantalla un círculo de radio  $r$ .
- Separar las sílabas de una palabra  $p$ .
- A partir de la fotografía de un vehículo, leer y reconocer su matrícula.
- Traducir un texto de castellano a inglés.
- Detectar posibles tumores a partir de imágenes radiográficas.

Por otra parte, el científico Alan Turing, demostró que existen **problemas irresolubles**, de los que **ningún** ordenador será capaz de obtener **nunca** su solución.

Los problemas deben definirse de forma general y precisa, evitando ambigüedades.

EJEMPLO: Raíz cuadrada.

- Determinar la raíz cuadrada de un número  $n$
- Determinar la raíz cuadrada de un número  $n$ , entero no negativo, cualquiera.

EJEMPLO: Dividir

- Calcular la división de dos números de dos números  $a$  y  $b$
- Calcular el cociente entero de la división  $a/b$ , donde  $a$  y  $b$  son números enteros y  $b$  es distinto de cero. ( $5/2 = 2$ )
- Calcular el cociente real de la división  $a/b$ , donde  $a$  y  $b$  son números reales y  $b$  es distinto de cero ( $5/2 = 2.5$ )

## 1.2 Algoritmos

Dado un problema P, un **ALGORITMO** es un conjunto de reglas o pasos que indican cómo resolver P en un tiempo finito

EJEMPLO: Secuencias de reglas básicas que utilizamos para realizar operaciones aritméticas: sumas, restas, productos y divisiones.

Supongamos que se nos plantea el problema de multiplicar 981 por 1234

$$\begin{array}{r} 981 \\ \times 1234 \\ \hline 3924 \\ 2943 \phantom{0} \\ 1962 \phantom{00} \\ 981 \phantom{000} \\ \hline 1210554 \end{array}$$

➤ Se multiplica sucesivamente el *multiplicando* (981) por cada una de las cifras del *multiplicador* (1234), tomadas de derecha a izquierda, escribiendo estos resultados intermedios uno tras otro, desplazando cada línea un lugar a la izquierda; finalmente se suman todas estas filas para obtener el *producto* (1210554)

EJEMPLO: Algoritmo para desayunar

### Inicio

Sentarse

Servirse café con leche

Servirse azúcar

Si tengo tiempo

    Mientras tenga apetito

        Untar mantequilla en una tostada

        Añadir mermelada

        Comer la tostada

    Fin Mientras

Fin Si

Beberse el café con leche

Levantarse

Fin

Un algoritmo, por tanto, no es más que la secuencia de pasos que se deben seguir para solucionar un problema específico. La descripción o nivel de detalle de la solución de un problema en términos algorítmicos depende de qué o quién debe entenderlo, interpretarlo y resolverlo.

### 1.2.1 Características de los algoritmos

Un algoritmo, para que sea válido, tiene que tener ciertas características fundamentales:


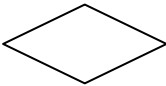



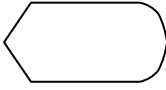
- **Generalidad:** han de definirse de forma general, utilizando identificadores o parámetros. Un algoritmo debe resolver toda una clase de problemas y no un problema aislado particular.
- **Finitud:** han de llevarse a cabo en un tiempo finito, es decir, el algoritmo ha de acabar necesariamente tras un número finito de pasos.
- **Definibilidad:** han de estar definidos de forma exacta y precisa, sin ambigüedades.
- **Eficiencia:** han de resolver el problema de forma rápida y eficiente.


### 1.2.2 Representación de algoritmos

Los métodos más usuales para representar algoritmos son los **diagramas de flujo** y el **pseudocódigo**. Ambos son sistemas de representación independientes de cualquier lenguaje de programación. Hay que tener en cuenta que el diseño de un algoritmo constituye un paso previo a la codificación de un programa en un lenguaje de programación determinado (C, C++, Java, Pascal). La independencia del algoritmo del lenguaje de programación facilita, precisamente, la posterior codificación en el lenguaje elegido.

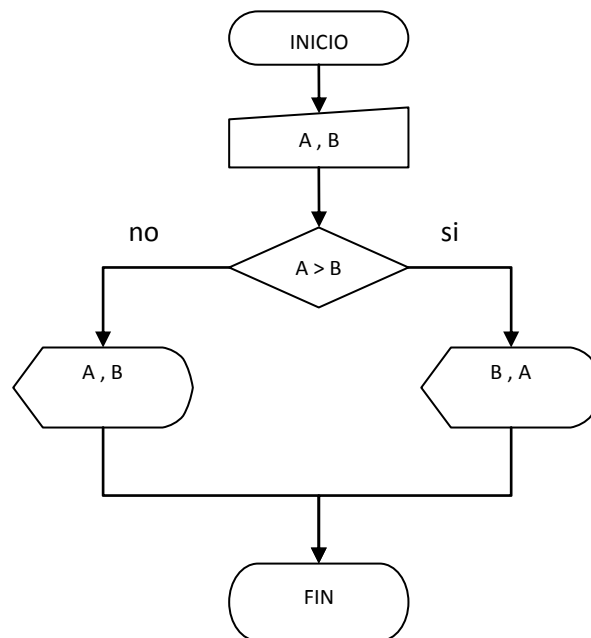
**Diagrama de flujo.**- Un diagrama de flujo (*Flowchart*) es una de las técnicas de representación de algoritmos más antiguas y más utilizadas, aunque su empleo disminuyó considerablemente con los lenguajes de programación estructurados. Un diagrama de flujo utiliza símbolos estándar que contienen los pasos del algoritmo escritos en esos símbolos, unidos por flechas denominadas líneas de flujo que indican la secuencia en que deben ejecutarse.

Los símbolos más utilizados son:

<b>Proceso</b>		Cualquier tipo de operación que pueda originar cambio de valor, formato, operaciones aritméticas etc
<b>Decisión</b>		Indica operaciones lógicas o de comparación entre datos y en función del resultado determina el camino a seguir.
<b>Terminal</b>		Representa el comienzo y el final de un programa o un módulo
<b>Entrada / Salida de información</b>		Este símbolo se puede subdividir en otros de teclado, pantalla, impresora disco etc.
<b>Teclado</b>		Representa las entradas de datos desde teclado
<b>Pantalla</b>		Representa las salidas de datos en pantalla

<b>Dirección del flujo</b>		Indica el sentido de ejecución de las operaciones
----------------------------	---	---

EJEMPLO: Mostrar dos números ordenados de menor a mayor



**Pseudocódigo.** El pseudocódigo es un lenguaje de descripción de algoritmos que está muy próximo a la sintaxis que utilizan los lenguajes de programación. Nace como medio para representar las estructuras de control de *programación estructurada*.

El pseudocódigo no se puede ejecutar nunca en el ordenador, sino que tiene que traducirse a un lenguaje de programación (codificación). La ventaja del pseudocódigo, frente a los diagramas de flujo, es que se puede modificar más fácilmente si detecta un error en la lógica del algoritmo, y puede ser traducido fácilmente a los lenguajes estructurados como Pascal, C, fortran, Java, etc.

El Pseudocódigo utiliza palabras reservadas (en sus orígenes se escribían en inglés) para representar las sucesivas acciones. Para mayor legibilidad utiliza la **identación** -sangría en el margen izquierdo- de sus líneas.

### Palabras reservadas del pseudocódigo:

**Inicio**  
**Fin**  
**Leer ( )**  
**Escribir ( )**

**Si** condición **Entonces**  
    acción1  
**SiNo**  
    acción2  
**FinSi**

EJEMPLO: Mostrar dos números ordenados de menor a mayor

**Mientras** condición  
    acción  
**FinMientras**

**Repetir**  
    acción  
**Hasta** condición

**Inicio**  
    **Leer** (A, B)  
    **Si** (A>B) **Entonces**  
        **Escribir** (B, A)  
    **SiNo**  
        **Escribir** (A, B)  
    **FinSi**  
**Fin**

**Para** valor-inicial **Hasta** valor-final **Hacer**  
    acción  
**FinPara**

## 1.3 Programas

Un **programa** es un algoritmo escrito en una notación precisa para que pueda ser ejecutado por un ordenador.

## 2 Paradigmas de la programación.

Los paradigmas de programación son las distintas formas de plantear la solución automatizada de un problema mediante un programa. Existen muchos paradigmas de programación diferentes, cada uno de ellos tiene sus **propias características** y tratan de solucionar los problemas clásicos del desarrollo de software desde diferentes perspectivas y filosofías.

### 2.1 Programación imperativa

Los programas expresan la secuencia de pasos que tiene que realizar el ordenador para llegar a la solución del problema. Podríamos decir que la programación imperativa es la programación natural para las CPUs, que se basan en ese paradigma al nivel **más básico**.

Existen, a su vez, varios paradigmas que han evolucionado a partir de la programación imperativa:

- **Programación estructurada:** Se basa en el uso de estructuras de control y en la no utilización de instrucciones de ruptura de secuencia (goto, resume, exit, ...)
- **Programación modular:** Consiste en agrupar instrucciones en bloques llamados funciones o subrutinas con el objetivo de descomponer los programas en piezas más manejables y favorecer la reutilización del software.
- **Programación Orientada a Objetos:** Quizás se trata del más utilizado en la actualidad. Con este paradigma, la programación se basa en la utilización de objetos,

elemento que aglutina datos junto a las instrucciones necesarias para manejarlos. Se persigue la reutilización de código y la capacidad de reflejar más fielmente los elementos del mundo real que se representan en el programa.

## 2.2 Programación declarativa

En la programación declarativa no es necesario definir algoritmos, puesto que se detalla la solución del problema en lugar de cómo llegar a esa solución.

- **Programación funcional:** Basada en el uso de funciones matemáticas.
- **Programación lógica:** Basada en la determinación de una serie de reglas lógicas.

## 3 Lenguajes de programación. El lenguaje Java

En la actualidad todos estamos habituados a utilizar programas. Para ejecutarlos basta, normalmente, con hacer doble clic en el icono correspondiente.

Para que el ordenador sea capaz de ejecutar el programa, este debe encontrarse en lo que se llama **código máquina**. El código máquina se compone de números codificados en binario que se corresponden con instrucciones que es capaz de ejecutar el microprocesador y los datos que acompañan a dichas instrucciones.

Sin embargo, los programadores de ordenadores, no escriben sus programas en código máquina, pues resulta poco intuitivo. En su lugar utilizan **lenguajes de programación**, más próximos al pensamiento lógico-matemático que al conjunto de instrucciones que conoce la máquina.

Existen multitud de lenguajes de programación distintos. Unos son de propósito general, otros están especializados en la elaboración de determinado tipo de aplicaciones. Pero ninguno de ellos puede ser ejecutado directamente por el procesador del ordenador. El procesador solo puede ejecutar código máquina. Para que un programa escrito en un lenguaje de programación se pueda ejecutar, es necesario realizar un proceso de traducción. Este proceso se realiza de forma automática mediante un programa que puede ser de dos tipos: compilador o intérprete.

### 3.1 El lenguaje Java

**Java** es un lenguaje de programación que tiene gran protagonismo en la actualidad. Es un lenguaje imperativo que incorpora los elementos necesarios para poder realizar una programación estructurada, modular y orientada a objetos.

Permite realizar aplicaciones de propósito general, pero además se dice que es el lenguaje de Internet: Applets, Servlets, páginas JSP o JavaScripts utilizan Java como lenguaje de programación.

Es un lenguaje **multiplataforma**, lo que permite que un mismo programa pueda ejecutarse sobre distintos sistemas operativos y tipos de ordenadores.

### 3.2 Independencia de la plataforma

Una plataforma es un determinado entorno de ejecución, que define el modo de trabajo del conjunto Software/Hardware que la conforma. En el mundo comercial se encuentran multitud de plataformas:

- Hardware basado en microprocesadores intel y S.O. Windows.
- Hardware basado en microprocesadores intel y S.O. Linux.
- Hardware basado en microprocesadores intel y S.O. OS/2.
- Hardware macintosh y S.O. MacOS
- Hardware basado en procesadores SPARC y S.O. Solaris.
- Etc.

Aquel software que haya sido desarrollado para una de estas plataformas no funcionará en el resto, lo que supone un hándicap importante para la portabilidad de la aplicaciones. Y este es precisamente el reto de Java: todo programa hecho en Java puede ejecutarse potencialmente sobre cualquier máquina y sistema operativo.

La clave de esta independencia de plataforma radica en que precisamente, tal y como se ha comentado anteriormente, java es una plataforma en si mismo, lo que proporciona el grado de abstracción necesario para que un programa java pueda funcionar en cualquier sistema.

Pero, ¿Cómo se consigue esto?. Pues haciendo de java un lenguaje en parte compilado y en parte interpretado. Mediante el uso de un compilador se traduce el código fuente escrito en lenguaje Java a un lenguaje intermedio cercano al lenguaje máquina pero independiente del ordenador y del sistema operativo en que se ejecuta. A las instrucciones obtenidas se les denomina bytecodes. Estas instrucciones son posteriormente interpretadas y ejecutadas por una Máquina Virtual De Java (JVM).

Para cada una de las arquitecturas Hardware/Software, se debe desarrollar una Máquina Virtual de Java específica.

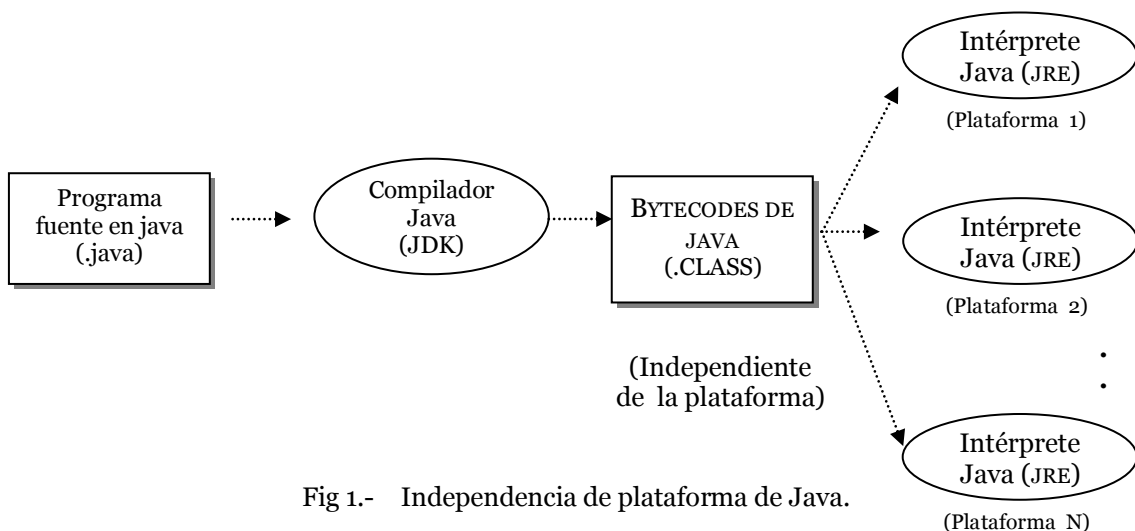


Fig 1.- Independencia de plataforma de Java.



## 4 Herramientas y entornos de desarrollo.

Un **entorno de desarrollo** es el conjunto de herramientas que utiliza el programador con la finalidad de desarrollar una aplicación en un lenguaje determinado. Generalmente y dependiendo del lenguaje de programación, el entorno de programación está formado por un **editor de texto**, un **compilador**, un **enlazador**, **bibliotecas de funciones**, **bibliotecas de clases**, etc.

Todas estas herramientas que el programador necesita pueden ser elementos aislados, o bien, aparecer cohesionadas entre sí a través de un interfaz gráfico, formando una aplicación. En este último caso se denominan entornos integrados de desarrollo.

Un **entorno integrado de desarrollo** o IDE (*Integrated Development Enviroment*) consta al menos de un editor de texto o código, un compilador y/o intérprete y un depurador de errores. Los actuales pueden incorporar además gestión de proyectos, control de versiones, asistencia y ayuda en la escritura del programa, funciones para facilitar la construcción de interfaces gráficas de usuario (GUI, *Graphic User Interface*).

En general, un IDE es específico para determinado lenguaje de programación pero existen algunos que sirven para varios lenguajes. Además, existen IDE multiplataforma, es decir, que pueden ejecutarse en distintos sistemas operativos y/o familias de ordenadores.

De entre los que permiten desarrollar aplicaciones Java cabe destacar:

### De código libre / Gratuitos

- Eclipse
- NetBeans
- ...

### Propietarios / Comerciales

- JBuilder
- IntelliJ IDEA
- JCreator
- ...

### 4.1 El JDK

El Java Development Kit (Kit de desarrollo de Java) es la herramienta que Oracle pone a disposición de los programadores para que estos puedan iniciarse de forma fácil y gratuita en el desarrollo de aplicaciones Java. Al mismo tiempo el JDK es una forma de presentar al público cada nueva especificación de Java.

El JDK es un entorno de desarrollo para escribir aplicaciones Java y Applets. Tanto el compilador como el resto de herramientas se ejecutan desde el shell del S.O. y no poseen interfaz gráfica. No se trata de un entorno de desarrollo **integrado**.

El contenido del JDK se puede dividir en cuatro partes claramente diferenciadas. Las clases base JAVA, el código fuente de estas, la documentación, y finalmente las utilidades.

#### 4.1.1 Algunas utilidades del JDK

**javac:** Es el compilador de Java. Convierte el código fuente escrito en Java a bytecode.

**java:** Es el intérprete de Java. Se encarga de ejecutar los bytecodes resultantes de compilar los programas. Es la máquina virtual que se nos entrega con el JDK.

**javadoc:** Genera documentación en archivos HTML basada en el código fuente y en los comentarios que contiene. Naturalmente para su correcto funcionamiento se deben seguir determinadas reglas.

**jdb:** Es el depurador. Permite recorrer línea a línea el programa, analizando las variables y los puntos de interrupción. Por desgracia, como el resto de herramientas, funciona mediante línea de comandos, lo cual dificulta bastante su uso.

**javap:** Desensamblador de Java. Muestra las funciones a las que se puede acceder y, que se encuentran dentro de los archivos class compilados.

## 5 Errores en los programas.

En las distintas fases por las que pasa un programa, durante su codificación, su prueba o su explotación, pueden detectarse errores. Los errores pueden ser de distinto tipo:

### 5.1 Errores de codificación (o sintácticos)

Son errores por una mala utilización de las reglas del lenguaje: palabras reservadas mal escritas, instrucciones incompletas, etc. Estos errores se detectan durante la compilación, cuando el programa se traduce del lenguaje de programación a código máquina. La herramienta de traducción nos avisa del error, por tanto son fáciles de detectar y corregir.

### 5.2 Errores de ejecución

Se producen durante la ejecución del programa porque se realiza una operación no permitida o porque se produce una situación imprevista de la que el programa no se puede recuperar. Por ejemplo, la memoria es insuficiente para realizar alguna operación. El compilador no detecta este tipo de errores y, en general, son más difíciles de detectar y de corregir.

### 5.3 Errores lógicos

El programa no produce error, ni durante la compilación ni durante su ejecución. Simplemente no hace lo que tiene que hacer o no produce el resultado que se esperaba.