

UD01: Elementos de un programa informático



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 la palabra p es del 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, reconocer y leer 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.

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.

Los algoritmos son independientes de los lenguajes de programación y de las computadoras donde se ejecutan. Un mismo algoritmo puede ser expresado en diferentes lenguajes de programación y podría ser ejecutado en diferentes dispositivos. Piensa en una receta de cocina, ésta puede ser expresada en castellano, inglés o francés, podría ser cocinada en fogón o vitrocerámica, por un cocinero o más, etc. Pero independientemente de todas estas circunstancias, el plato se preparará siguiendo los mismos pasos.

La **diferencia** fundamental entre **algoritmo** y **programa** es que, en el segundo, los pasos que permiten resolver el problema, deben escribirse en un determinado lenguaje de programación para que puedan ser ejecutados en el ordenador y así obtener la solución.

1.2.1. Características de los algoritmos

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


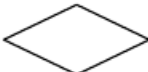


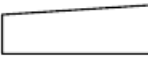


- **Generalidad:** han de definirse de forma general, utilizando identificadores o parámetros. Un algoritmo debe resolver toda una clase de problemas y no un problema aislado particular.
- **Finitud:** han de llevarse a cabo en un tiempo finito, es decir, el algoritmo ha de acabar necesariamente tras 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

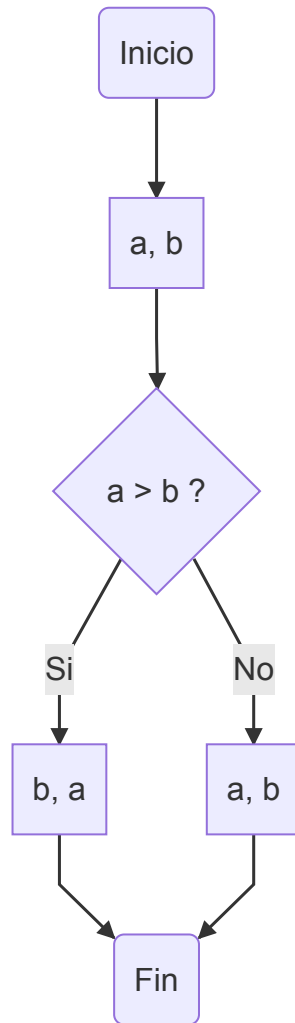
Los métodos más usuales para representar algoritmos son los diagramas de flujo y el pseudocódigo. Ambos son sistemas de representación independientes de cualquier lenguaje de programación. Hay que tener en cuenta que el diseño de un algoritmo constituye un paso previo a la codificación de un programa 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.

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:

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

Ejemplo: Mostrar dos números ordenados de menor a mayor



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 indentación -sangría en el margen izquierdo- de sus líneas.

Ejemplo: Mostrar dos números ordenados de menor a mayor

```

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

1.3. Programas

La **diferencia** fundamental entre **algoritmo** y **programa** es que, en el segundo, los pasos que permiten resolver el problema, deben escribirse en un determinado lenguaje de programación para que puedan ser ejecutados en el ordenador y así obtener la solución.

Los lenguajes de programación son sólo un medio para expresar el algoritmo y el ordenador un procesador para ejecutarlo. El diseño de los algoritmos será una tarea que necesitará de la creatividad y conocimientos de las técnicas de programación. Estilos distintos, de distintos programadores a la hora de obtener la solución del problema, darán lugar a programas diferentes, igualmente válidos.

Pero cuando los problemas son complejos, es necesario descomponer éstos en subproblemas más simples y, a su vez, en otros más pequeños. Estas estrategias reciben el nombre de diseño descendente (Metodología de diseño de programas, consistente en la descomposición del problema en problemas más sencillos de resolver) o diseño modular (top-down design) (Metodología de diseño de programas, que consiste en dividir la solución a un problema en módulos más pequeños o subprogramas. Las soluciones de los módulos se unirán para obtener la solución general del problema). Este sistema se basa en el lema **divide y vencerás**.

1.4. Estructura y Bloques Fundamentales de un programa.

```
public class holamundo {
    // programa Hola Mundo
    public static void main(String[] args) {
        /* lo único que hace este programa es mostrar
           la cadena "Hola Mundo!" por pantalla */
        System.out.println("Hola Mundo!");
    }
}
```

En Java generalmente una clase lleva el identificador `public` y corresponde con un fichero. El nombre de la clase coincide con el del fichero `.java` respetando mayúsculas y minúsculas.

```
public class holamundo {
    [...]
}
```

El código java en las clases se agrupa en funciones o métodos. Cuando java ejecuta el código de una clase busca la función o método `main()` para ejecutarla. Es público (`public`) estático (`static`) para llamarlo sin instanciar la clase. No devuelve ningún valor (`void`) y admite parámetros (`Strings [] args`) que en este caso no se han utilizado.

```
[...]
    public static void main (Strings [] args)
    {
        [...]
    }
[...]
```

El código de la función `main` se escribe entre las llaves. Por ejemplo:

```
[...]
    System.out.println("Hola Mundo");
[...]
```

Muestra por pantalla el mensaje `Hola Mundo`, ya que la clase `System` tiene un atributo `out` con dos métodos: `print()` y `println()`. La diferencia es que `println` muestra mensaje e introduce un retorno de carro.

Todas las instrucciones menos las llaves `{ }` terminan con punto y coma `(;)`.

1.5. Sangrado o Indentación

El sangrado (también conocido como indentación) deberá aplicarse a toda estructura que esté lógicamente contenida dentro de otra. El sangrado será de un tabulador. **Es suficiente entre 2 y 4 espacios.** Para alguien que empieza a programar suele ser preferible unos 4 espacios, ya que se ve todo más claro.

Las líneas no tendrán en ningún caso demasiados caracteres que impidan que se pueda leer en una pantalla. **Un número máximo recomendable suele estar entre unos 70 y 90 caracteres, incluyendo los espacios de sangrado.** Si una línea debe ocupar más caracteres, tiene que dividirse en dos o más líneas, para ello utiliza los siguientes principios para realizar la división:

- Tras una coma.
- Antes de un operador, que pasará a la línea siguiente.
- Una construcción de alto nivel (por ejemplo, una expresión con paréntesis).
- La nueva línea deberá alinearse con un sangrado lógico, respecto al punto de ruptura

Unos pocos ejemplos, para comprender mejor:

Dividir tras una coma:

```
funcion(expresionMuuuuyLarga1,
        expresionMuuuyyyyLarga2,
        expresionMuuuyyyLarga3);
```

Mantener la expresión entre paréntesis en la misma línea:

```
nombreLargo = nombreLargo2*
              (nombreLargo3 + nombreLargo4)+
              4*nombreLargo5;
```

Siempre hay excepciones. Puede resultar que al aplicar estas reglas, en operaciones muy largas, o expresiones lógicas enormes, el sangrado sea ilegible. En estos casos, el convenio se puede relajar.

2. Java

2.1. ¿Qué y cómo es Java?

Java es un lenguaje sencillo de aprender, con una sintaxis parecida a la de C++, pero en la que se han eliminado elementos complicados y que pueden originar errores. Java es orientado a objetos, con lo que elimina muchas preocupaciones al programador y permite la utilización de gran cantidad de bibliotecas ya definidas, evitando reescribir código que ya existe. Es un lenguaje de programación creado para satisfacer nuevas necesidades que los lenguajes existentes hasta el momento no eran capaces de solventar.

Una de las principales virtudes de Java es su independencia del hardware, ya que el código que se genera es válido para cualquier plataforma. Este código será ejecutado sobre una máquina virtual denominada Máquina Virtual Java (MVJ o JVM – Java Virtual Machine), que interpretará el código convirtiéndolo a código específico de la plataforma que lo soporta. De este modo el programa se escribe una única vez y puede hacerse funcionar en cualquier lugar. Lema del lenguaje: *"Write once, run everywhere"*.

Antes de que apareciera Java, el lenguaje C era uno de los más extendidos por su versatilidad. Pero cuando los programas escritos en C aumentaban de volumen, su manejo comenzaba a complicarse.

Mediante las técnicas de programación estructurada y programación modular se conseguían reducir estas complicaciones, pero no era suficiente.

Fue entonces cuando la Programación Orientada a Objetos (POO) entra en escena, aproximando notablemente la construcción de programas al pensamiento humano y haciendo más sencillo todo el proceso. Los problemas se dividen en objetos que tienen propiedades e interactúan con otros objetos, de este modo, el programador puede centrarse en cada objeto para programar internamente los elementos y funciones que lo componen.

Las características principales de lenguaje Java se resumen a continuación:

- El código generado por el compilador Java es independiente de la arquitectura.
- Está totalmente orientado a objetos.
- Su sintaxis es similar a C y C++.
- Es distribuido, preparado para aplicaciones TCP/IP.
- Dispone de un amplio conjunto de bibliotecas.
- Es robusto, realizando comprobaciones del código en tiempo de compilación y de ejecución.
- La seguridad está garantizada, ya que las aplicaciones Java no acceden a zonas delicadas de memoria o de sistema. (*ejem, ejem!*)

2.2. Breve historia.

Java surgió en 1991 cuando un grupo de ingenieros de Sun Microsystems trataron de diseñar un nuevo lenguaje de programación destinado a programar pequeños dispositivos electrónicos. La dificultad de estos dispositivos es que cambian continuamente y para que un programa funcione en el siguiente dispositivo aparecido, hay que reescribir el código. Por eso la empresa Sun quería crear un lenguaje independiente del dispositivo.

Pero no fue hasta 1995 cuando pasó a llamarse Java, dándose a conocer al público como lenguaje de programación para computadores. Java pasa a ser un lenguaje totalmente independiente de la plataforma y a la vez potente y orientado a objetos. Esa filosofía y su facilidad para crear aplicaciones para redes TCP/IP ha hecho que sea uno de los lenguajes más utilizados en la actualidad.

El factor determinante para su expansión fue la incorporación de un intérprete Java en la versión 2.0 del navegador Web Netscape Navigator, lo que supuso una gran revuelo en Internet. A principios de 1997 apareció Java 1.1 que proporcionó sustanciales mejoras al lenguaje. Java 1.2, más tarde rebautizado como Java 2, nació a finales de 1998.

El principal objetivo del lenguaje Java es llegar a ser el nexo universal que conecte a los usuarios con la información, esté ésta situada en el ordenador local, en un servidor Web, en una base de datos o en cualquier otro lugar.

Para el desarrollo de programas en lenguaje Java es necesario utilizar un entorno de desarrollo denominado JDK (Java Development Kit), que provee de un compilador y un entorno de ejecución (JRE – Java RunEnvironment) para los bytecodes generados a partir del código fuente. Al igual que las diferentes versiones del lenguaje han incorporado mejoras, el entorno de desarrollo y ejecución también ha sido mejorado sucesivamente.

Java 2 es la tercera versión del lenguaje, pero es algo más que un lenguaje de programación, incluye los siguientes elementos:

- Un lenguaje de programación: Java.
- Un conjunto de bibliotecas estándar que vienen incluidas en la plataforma y que son necesarias en todo entorno Java. Es el Java Core.
- Un conjunto de herramientas para el desarrollo de programas, como es el compilador de bytecodes, el generador de documentación, un depurador, etc.
- Un entorno de ejecución que en definitiva es una máquina virtual que ejecuta los programas traducidos a bytecodes.

2.3. Compilar y ejecutar un programa Java. Uso de la consola.

Veamos los pasos para compilar e interpretar nuestro primer programa escrito en lenguaje Java.

2.3.1. PASO 1: Creación del código fuente

Abrimos un editor de texto (da igual cual sea, siempre que sea capaz de almacenar “texto sin formato” en código ASCII). Una vez abierto escribiremos nuestro primer programa, que mostrará un texto “Hola Mundo” en la consola. De momento no te preocupes si no entiendes lo que escribes, más adelante le daremos sentido. Ahora solo queremos ver si podemos ejecutar java en nuestro equipo.

El código de nuestro programa en Java será el siguiente:

```
/* Ejemplo Hola Mundo */
public class Ejemplo {
    public static void main(String[ ] arg) {
        System.out.println("Hola Mundo");
    }
}
```

A continuación guardamos nuestro archivo y le ponemos como nombre `Ejemplo.java`. Debemos seguir una norma dictada por Java, hemos de hacer coincidir nombre del archivo y nombre del programa, tanto en mayúsculas como en minúsculas, y la extensión del archivo habrá de ser siempre “.java”.

```

~: nano — Konsole
Fitxer  Edita  Visualitza  Adreces d'interès  Arranjament  Ajuda
GNU nano 4.8  Ejemplo.java  Modificat
/* Ejemplo Hola Mundo */
public class Ejemplo {
    public static void main(String[] arg) {
        System.out.println("Hola Mundo");
    }
}
^G Ajuda  ^O Desa  ^W On és  ^K Retalla  ^J Justifica  ^C Pos Act
^X Surt  ^R Llegeix  ^\ Reemplaça  ^U Enganxa te  ^T Ortografia  ^_ Vés a línia

```

Debemos recordar exactamente la ruta donde guardamos el archivo de ejemplo `Ejemplo.java`.

2.3.2. PASO 2: Compilación del programa

Vamos a proceder a compilar e interpretar este pequeño programa Java (no te preocupes si todavía no entiendes el significado de las palabras compilar e interpretar, lo veras en la asignatura de `Entornos de Desarrollo`). Para ello usaremos la consola. Una vez en la consola debemos colocarnos en la ruta donde previamente guardamos el archivo `Ejemplo.java`.

A continuación daremos la instrucción para que se realice **el proceso de compilación del programa**, para lo que escribiremos `javac Ejemplo.java`, donde `javac` es el nombre del compilador (**java compiler**) que transformará el programa que hemos escrito nosotros en lenguaje Java al lenguaje de la máquina virtual Java (`bytecode`), dando como resultado un nuevo archivo `Ejemplo.class` que se creará en este mismo directorio. Comprueba que no aparezca ningún error y que `javac` esté instalado en tu sistema (desde la consola lo puedes comprobar con el comando `javac --version` y debería aparecer el número de versión que tienes instalada). Si aparecen los dos archivos tanto `Ejemplo.java` (código fuente) como `Ejemplo.class` (bytecode creado por el compilador) puedes continuar.

```
$ javac Ejemplo.java
```

2.3.3. PASO 3: Ejecución del programa

Finalmente, vamos a pedirle al intérprete que ejecute el programa, es decir, que transforme el código de la máquina virtual Java en código máquina interpretable por nuestro ordenador y lo ejecute. Para ello escribiremos en la ventana consola: `java Ejemplo`.

El resultado será que se nos muestra la cadena `Hola Mundo`. Si logramos visualizar este texto en pantalla, ya hemos desarrollado nuestro primer programa en Java.

```
$ java Ejemplo
Hola Mundo
```

3. Variables, identificadores, convenciones.

3.1. Variables

Una **variable** es una zona en la memoria del computador con un valor que puede ser almacenado para ser usado más tarde en el programa. Las variables vienen determinadas por:

- un **nombre**, que permite al programa acceder al valor que contiene en memoria. Debe ser un identificador válido.
- un **tipo de dato**, que especifica qué clase de información guarda la variable en esa zona de memoria
- un **rango de valores** que puede admitir dicha variable.

Las variables declaradas dentro de un bloque `{ }` son accesibles solo dentro de ese bloque. Una variable local no puede ser declarada como `static`. Una variable no puede declararse fuera de la clase.

Visibilidad, ámbito o scope de una variable es la parte de código del programa donde la variable es accesible y utilizable. Las variables de un bloque son visibles y existen dentro de dicho bloque. Las funciones miembro de clase podrán acceder a todas las variables miembro de dicha clase pero no a las variables locales de otra función miembro.

Al nombre que le damos a la variable se le llama identificador. Los identificadores permiten nombrar los elementos que se están manejando en un programa. Vamos a ver con más detalle ciertos aspectos sobre los identificadores que debemos tener en cuenta.

3.2. Identificadores

Un **identificador** en Java es una secuencia ilimitada sin espacios de letras y dígitos Unicode, de forma que el primer símbolo de la secuencia debe ser una letra, un símbolo de subrayado (`_`) o el símbolo dólar (`$`). Por ejemplo, son válidos los siguientes identificadores:

- `x5`
- `απη`
- `NUM_MAX`
- `numCuenta`

Unicode es un código de caracteres o sistema de codificación, un alfabeto que recoge los caracteres de prácticamente todos los idiomas importantes del mundo. Además, el código Unicode es “compatible” con el código ASCII, ya que para los caracteres del código ASCII, Unicode asigna como código los mismos 8 bits, a los que les añade a la izquierda otros 8 bits todos a cero. La conversión de un carácter ASCII a Unicode es inmediata.

3.3. Convenciones

Normas de estilo para nombra variables

A la hora de nombrar un identificador existen una serie de normas de estilo de uso generalizado que, no siendo obligatorias, se usan en la mayor parte del código Java. Estas reglas para la nomenclatura de variables son las siguientes:

- Java distingue las mayúsculas de las minúsculas. Por ejemplo, `Alumno` y `alumno` son variables diferentes.
- No se suelen utilizar identificadores que comiencen con `$` o `_`, además el símbolo del dólar, por convenio, no se utiliza nunca.
- No se puede utilizar el valor booleano (`true` o `false`) ni el valor nulo (`null`).
- Los identificadores deben ser lo más descriptivos posibles. Es mejor usar palabras completas en vez de abreviaturas crípticas. Así nuestro código será más fácil de leer y comprender. En muchos casos también

hará que nuestro código se auto-documente. Por ejemplo, si tenemos que darle el nombre a una variable que almacena los datos de un cliente sería recomendable que la misma se llamara algo así como `FicheroClientes` o `ManejadorCliente`, y no algo poco descriptivo como `C133`.

Además de estas restricciones, en la siguiente tabla puedes ver otras convenciones, que no siendo obligatorias, sí son recomendables a la hora de crear identificadores en Java.

Identificador	Convención	Ejemplo
nombre de variable	Comienza por letra minúscula, y si tienen más de una palabra se colocan juntas y el resto comenzando por mayúsculas. A esto se le llama <i>lowerCamelCase</i> .	numAlumnos, suma
nombre de constante	En letras mayúsculas, separando las palabras con el guión bajo, por convenio el guión bajo no se utiliza en ningún otro sitio	TAM_MAX, PI
nombre de una clase	Comienza por letra mayúscula, y si tienen más de una palabra se colocan juntas y el resto comenzando por mayúsculas. A esto se le llama <i>upperCamelCase</i> .	String, MiTipo
nombre de función	Comienza por letra minúscula, y si tienen más de una palabra se colocan juntas y el resto comenzando por mayúsculas. A esto se le llama <i>lowerCamelCase</i> .	modificaValor, obtieneValor

Puedes consultar estas y otras convenciones sobre código Java en este [enlace](#).

Palabras reservadas

Las palabras reservadas, a veces también llamadas palabras clave o keywords, son secuencias de caracteres formadas con letras ASCII cuyo uso se reserva al lenguaje y, por tanto, no pueden utilizarse para crear identificadores.

Las palabras reservadas en Java son:

```
abstract, continue, for, new, switch, assert, default, goto, package, synchronized, boolean, do, if,
private, this, break, double, implements, protected, throw, byte, else, import, public, throws, case,
enum, instanceof, return, transient, catch, extends, int, short, try, char, final, interface, static,
void, class, finally, long, strictfp, volatile, const, float, native, super, while
```

4. Tipos de datos.

Los tipos de datos se utilizan para declarar variables y el compilador sepa de antemano que tipo de información contendrá la variable.

Java dispone de los siguientes tipos de datos simples:

Tipo de dato	Representación	Tamaño (Bytes)	Rango de Valores	Valor por defecto	Clase Asociada
byte	Numérico Entero con signo	1	-128 a 127	0	Byte
short	Numérico Entero con signo	2	-32768 a 32767	0	Short
int	Numérico Entero con signo	4	-2147483648 a 2147483647	0	Integer
long	Numérico Entero con signo	8	-9223372036854775808 a 9223372036854775807	0	Long
float	Numérico en Coma flotante de precisión simple Norma IEEE 754	4	-3.4×10^{-38} a 3.4×10^{38}	0.0	Float
double	Numérico en Coma flotante de precisión doble Norma IEEE 754	8	-1.8×10^{-308} a 1.8×10^{308}	0.0	Double
char	Carácter Unicode	2	\u0000 a \uFFFF	\u0000	Character
boolean	Dato lógico	-	true ó false	false	Boolean
void	-	-	-	-	Void

Ejemplo de declaración y asignación de valores a variables:

Tipo de datos	código
byte	<code>byte a;</code>
short	<code>short b, c=3;</code>
int	<code>int d=-30;</code> <code>int e=0xC125; //la 0x significa Hexadecimal</code>
long	<code>long b=46240;</code> <code>long b=5L; // La L en este caso indica Long</code>
char	<code>char car1='c'</code> <code>char car2=99; //car1 y car2 son iguales, la c equivale al ascii 99</code> <code>char letra = '\u0061'; //código unicode del carácter "a"</code>
float	<code>float pi=3.1416;</code> <code>float pi=3.1416F; //La F significa float</code> <code>float medio=1/2; //0.5</code>
double	<code>double millon=1e6; // 1x10^6</code> <code>double medio=1/2D; //0.5, la D significa double</code> <code>double z=.123; //si la parte entera es 0 se puede omitir</code>
boolean	<code>boolean primero;</code> <code>boolean par=false;</code>

5. Tipos referenciados

A partir de los ocho tipos datos primitivos, se pueden construir otros tipos de datos. Estos tipos de datos se llaman tipos referenciados o referencias, porque se utilizan para almacenar la dirección de los datos en la memoria del ordenador.

```
int[] arrayDeEnteros;  
Cuenta cuentaCliente;
```

En la primera instrucción declaramos una lista de números del mismo tipo, en este caso, enteros. En la segunda instrucción estamos declarando la variable u objeto `cuentaCliente` como una referencia de tipo `Cuenta`.

Cualquier aplicación de hoy en día necesita no perder de vista una cierta cantidad de datos. Cuando el conjunto de datos utilizado tiene características similares se suelen agrupar en estructuras para facilitar el acceso a los mismos, son los llamados datos estructurados.

Son datos estructurados los `arrays`, `listas`, `árboles`, etc. Pueden estar en la memoria del programa en ejecución, guardados en el disco como ficheros, o almacenados en una base de datos.

Además de los ocho tipos de datos primitivos que ya hemos descrito, Java proporciona un tratamiento especial a los textos o cadenas de caracteres mediante el tipo de dato `String`. Java crea automáticamente un nuevo objeto de tipo `String` cuando se encuentra una cadena de caracteres encerrada entre comillas dobles. En realidad se trata de objetos, y por tanto son tipos referenciados, pero se pueden utilizar de forma sencilla como si fueran variables de tipos primitivos:

```
String mensaje;  
mensaje= "El primer programa";
```

Hemos visto qué son las variables, cómo se declaran y los tipos de datos que pueden adoptar. Anteriormente hemos visto un ejemplo de creación de variables, en esta ocasión vamos a crear más variables, pero de distintos tipos primitivos y los vamos a mostrar por pantalla. Los tipos referenciados los veremos en la siguiente unidad.

Para mostrar por pantalla un mensaje utilizamos `System.out`, conocido como la salida estándar del programa. Este método lo que hace es escribir un conjunto de caracteres a través de la línea de comandos. Podemos utilizar `System.out.print` o `System.out.println`. En el segundo caso lo que hace el método es que justo después de escribir el mensaje, sitúa el cursor al principio de la línea siguiente.

El texto en color gris que aparece entre caracteres `//` son comentarios que permiten documentar el código, pero no son tenidos en cuenta por el compilador y, por tanto, no afectan a la ejecución del programa.

6. Tipos enumerados

Los tipos de datos enumerados son una forma de declarar una variable con un conjunto restringido de valores. Por ejemplo, los días de la semana, las estaciones del año, los meses, etc. Es como si definiéramos nuestro propio tipo de datos.

La forma de declararlos es con la palabra reservada `enum`, seguida del nombre de la variable y la lista de valores que puede tomar entre llaves. A los valores que se colocan dentro de las llaves se les considera como constantes, van separados por comas y deben ser valores únicos.

La lista de valores se coloca entre llaves, porque un tipo de datos `enum` no es otra cosa que una especie de clase en Java, y todas las clases llevan su contenido entre llaves.

Al considerar Java este tipo de datos como si de una clase se tratara, no sólo podemos definir los valores de un tipo enumerado, sino que también podemos definir operaciones a realizar con él y otro tipo de elementos, lo que hace que este tipo de dato sea más versátil y potente que en otros lenguajes de programación.

En el siguiente ejemplo puedes comprobar el uso que se hace de los tipos de datos enumerados.

```
public class tiposEnumerados {  
    public enum Dias {Lunes, Martes, Miercoles, Jueves, Viernes, Sábado, Domingo}  
  
    public static void main(String[] args) {  
        Dias diaActual = Dias.Martes;  
        Dias diaSiguiente = Dias.Miercoles;  
  
        System.out.print("Hoy es:");  
        System.out.println(diaActual);  
        System.out.println("Mañana\nes\n"+diaSiguiente);  
    }  
}
```

El resultado después de la ejecución será:

```
Hoy es:Martes  
Mañana  
es  
Miercoles
```

Tenemos una variable `Dias` que almacena los días de la semana. Para acceder a cada elemento del tipo enumerado se utiliza el nombre de la variable seguido de un punto y el valor en la lista. Más tarde veremos que podemos añadir métodos y campos o variables en la declaración del tipo enumerado, ya que como hemos comentado un tipo enumerado en Java tiene el mismo tratamiento que las clases.

En este ejemplo hemos utilizado el método `System.out.print`. Como podrás comprobar si lo ejecutas, la instrucción `print` escribe el texto que tiene entre comillas pero no salta a la siguiente línea, por lo que la instrucción `println` escribe justo a continuación.

Sin embargo, también podemos escribir varias líneas usando una única sentencia. Así lo hacemos en la instrucción `println`, la cual imprime como resultado tres líneas de texto. Para ello hemos utilizado un carácter especial, llamado carácter escape (`\`). Este carácter sirve para darle ciertas órdenes al compilador, en lugar de que salga impreso en pantalla. Después del carácter de escape viene otro carácter que indica la orden a realizar, juntos reciben el nombre de secuencia de escape. La secuencia de escape `\n` recibe el nombre de carácter de nueva línea. Cada vez que el compilador se encuentra en un texto ese carácter, el resultado es que

mueve el cursor al principio de la línea siguiente. En el próximo apartado vamos a ver algunas de las secuencias de escape más utilizadas.

7. Constantes y literales.

Las constantes se utilizan para almacenar datos que no varían nunca, asegurándonos que el valor no va a poder ser modificado.

Podemos declarar una constante utilizando:

```
final <tipo de datos> <nombre de la constante> = <valor>;
```

El calificador final indica que es constante. A continuación indicaremos el tipo de dato, el nombre de la constante y el valor que se le asigna.

```
final double IVA = 0.21;
```

Los literales pueden ser de tipo simple, null o string, como por ejemplo 230, null o "Java".

Respecto a los literales existen unos caracteres especiales que se representan utilizando secuencias de escape:

Secuencia de escape	Significado	Secuencia de escape	Significado
\b	Retroceso	\r	Retorno de carro
\t	Tabulador	\"	Carácter comillas dobles
\n	Salto de línea	\'	Carácter comillas simples
\f	Salto de página	\	Barra diagonal

8. Operadores y expresiones.

8.1. Operadores Aritméticos

Los **Operadores Aritméticos** permiten realizar operaciones matemáticas:

Operador	Uso	Operación
+	A + B	Suma
-	A - B	Resta
*	A * B	Multipliación
/	A / B	División
%	A % B	Módulo o resto de una división entera

Ejemplo:

```
double num1, num2, suma, resta, producto, division, resto;
num1 =8;
num2 =5;
suma = num1+num2;      // 13
resta = num1 - num2;    // 3
producto = num1 * num2; // 40
division = num1 / num2; // 1.6
resto = num1 % num2;    // 3
```

8.2. Operadores Relacionales

Los **Operadores Relacionales** permiten evaluar (la respuesta es un booleano: si o no) la igualdad de los operandos:

Operador	Uso	Operación
<	A < B	A menor que B
>	A > B	A mayor que B
<=	A <= B	A menor o igual que B
>=	A >= B	A mayor o igual que B
!=	A != B	A distinto de B
==	A == B	A igual a B

Por ejemplo:

```

int valor1 = 10;
int valor2 = 3;
boolean compara;
compara = valor1 > valor2; // true
compara = valor1 < valor2; // false
compara = valor1 >= valor2; // true
compara = valor1 <= valor2; // false
compara = valor1 == valor2; // false
compara = valor1 != valor2; // true

```

8.3. Operadores Lógicos

Los **Operadores Lógicos** permiten realizar operaciones lógicas:

Operador	Uso	Operación
&& o &	A && B o A & B	A AND B. El resultado será true si ambos operadores son true y false en caso contrario.
o	A B o A B	A OR B. El resultado será false si ambos operandos son false y true en caso contrario
!	!A	NOT A. Si el operando es true el resultado es false y si el operando es false el resultado es true.
^	A ^ B	A XOR B. El resultado será true si un operando es true y el otro false, y false en caso contrario.

Ejemplo:

```

double sueldo = 1400;
int edad = 34;
boolean logica;
logica = (sueldo>1000 & edad<40); //true
logica = (sueldo>1000 && edad >40); //false
logica = (sueldo>1000 | edad>40); //true
logica = (sueldo<1000 || edad >40); //false
logica = !(edad <40); //false
logica = (sueldo>1000 ^edad>40); //true
logica = (sueldo<1000 ^edad>40); //false

```

Para representar resultados de operadores Lógicos también se pueden usar tablas de verdad a las que conviene acostumbrarse:

A	B	A && B	A B	!A
false	false	false	false	true
true	false	false	true	false
false	true	false	true	true
true	true	true	true	false

8.4. Operadores Unarios o Unitarios

Los **Operadores Unarios** o **Unitarios** permiten realizar incrementos y decrementos:

Operador	Uso	Operación
++	A++ o ++A	Incremento de A
--	A-- o --A	Decremento de A

Ejemplo:

```
int m = 5, n = 3;
m++; // 6
n--; // 2
```

En el caso de utilizarlo como prefijo el valor de asignación será el valor del operando más el incremento de la unidad. Y si lo utilizamos como sufijo se asignará el valor del operando y luego se incrementará la unidad sobre el operando.

```
int A = 1, B;
B = ++A; // A vale 2 y B vale 2
B = A++; // A vale 3 y B vale 2
```

8.5. Operadores de Asignación

Los **Operadores de Asignación** permiten asignar valores:

Operador	Uso	Operación
=	A = B	Asignación (como ya hemos visto)
*=	A *= B	Multipliación y asignación. La operación A*=B equivale a A=A*B
/=	A /= B	División y asignación. La operación A/=B equivale a A=A/B
%=	A %= B	Módulo y asignación. La operación A%=B equivale a A=A%B
+=	A += B	Suma y asignación. La operación A+=B equivale a A=A+B
-=	A -= B	Resta y asignación. La operación A-=B equivale a A=A-B

Ejemplo:

```
int dato1 = 10, dato2 = 2, dato;
dato=dato1; // dato vale 10
dato2*=dato1; // dato2 vale 20
dato2/=dato1; // dato2 vale 2
dato2+=dato1; // dato2 vale 12
dato2-=dato1; // dato2 vale 2
dato1%=dato2; // dato1 vale 0
```

Los operadores tienen diferente Prioridad por lo que es interesante utilizar paréntesis para controlar las operaciones sin necesidad de depender de la prioridad de los operadores.

Operador	Utilización	Resultado
<<	A << B	Desplazamiento de A a la izquierda en B posiciones. Multiplica por 2 el número B de veces.
>>	A >> B	Desplazamiento de A a la derecha en B posiciones, tiene en cuenta el signo. Divide por 2 el número B de veces.
>>>	A >>> B	Desplazamiento de A a la derecha en B posiciones, no tiene en cuenta el signo. (simplemente agrega ceros por la izquierda)
&	A & B	Operación AND a nivel de bits
	A B	Operación OR a nivel de bits
^	A ^ B	Operación XOR a nivel de bits
~	~A	Complemento de A a nivel de bits

[illegible]

8.7. Operador condicional `?:`

El **operador condicional** `?:` sirve para evaluar una condición y devolver un resultado en función de si es verdadera o falsa dicha condición. Es el único operador ternario de Java, y como tal, necesita tres operandos para formar una expresión.

El primer operando se sitúa a la izquierda del símbolo de interrogación, y siempre será una expresión booleana, también llamada condición. El siguiente operando se sitúa a la derecha del símbolo de interrogación y antes de los dos puntos, y es el valor que devolverá el operador condicional si la condición es verdadera. El último operando, que aparece después de los dos puntos, es la expresión cuyo resultado se devolverá si la condición evaluada es falsa.

```
condición ? exp1 : exp2
```

Por ejemplo, en la expresión:

```
(x>y)?x:y;
```

Se evalúa la condición de si x es mayor que y, en caso afirmativo se devuelve el valor de la variable x, y en caso contrario se devuelve el valor de y.

Ejemplo para calcular qué número es mayor:

```
int mayor, exp1 = 15, exp2 = 25;
mayor=(exp1>exp2)?exp1:exp2;
// mayor valdrá 25
```

El operador condicional se puede sustituir por la sentencia `if...then...else` que veremos más adelante.

8.8. Prevalencia de operadores

Los operadores tienen diferente **Prioridad** por lo que es interesante utilizar paréntesis para controlar las operaciones sin necesidad de depender de la prioridad de los operadores.

Prevalencia de operadores, ordenados de arriba a abajo de más a menos prioridad:

Descripción	Operadores
operadores posfijos	op++ op--
operadores unarios	++op --op +op -op ~ !
multiplicación y división	* / %
suma y resta	+ -
desplazamiento	<< >> >>>
operadores relacionales	< > <= >=
equivalencia	== !=
operador AND	&
operador XOR	^
operador OR	
AND booleano	&&
OR booleano	
condicional	?:
operadores de asignación	= += -= *= /= %= &= ^= = <<= >>= >>>=

Por ejemplo:

```
int x, y1 = 6, y2 = 2, y3 = 8;
x = y1 + y2 * y3;    // 22
x = (y1 + y2) * y3;  // 64
```


9. Conversiones de tipo.

Existen dos tipos de conversiones: Implícitas y Explícitas. Debemos evitar las conversiones de tipos ya que pueden suponer pérdidas de información.

9.1. Conversiones Implícitas

Las **Conversiones Implícitas** se realizan de forma automática y requiere que la variable destino tenga más precisión que la variable origen para poder almacenar el valor.

Ejemplo:

```
// Conversión Implícita
byte origen = 5;
short destino;
destino=origen; // 5
```

9.2. Conversión Explícita

En la **Conversión Explícita** el programador fuerza la conversión con la operación llamada "cast":

Ejemplo:

```
// Conversión Explícita
short origen2 = 3;
byte destino2;
destino2=(byte)origen2; // 3
```

10. Comentarios.

Los comentarios son muy importantes a la hora de describir qué hace un determinado programa. A lo largo de la unidad los hemos utilizado para documentar los ejemplos y mejorar la comprensión del código. Para lograr ese objetivo, es normal que cada programa comience con unas líneas de comentario que indiquen, al menos, una breve descripción del programa, el autor del mismo y la última fecha en que se ha modificado.

Todos los lenguajes de programación disponen de alguna forma de introducir comentarios en el código. En el caso de Java, nos podemos encontrar los siguientes tipos de comentarios:

- Comentarios de **una sola línea**. Utilizaremos el delimitador `//` para introducir comentarios de sólo una línea.

```
// comentario de una sola línea
```

- Comentarios de **múltiples líneas**. Para introducir este tipo de comentarios, utilizaremos una barra inclinada y un asterisco (`/*`), al principio del párrafo y un asterisco seguido de una barra inclinada (`*/`) al final del mismo.

```
/* Esto es un comentario  
de varias líneas */
```

- Comentarios **Javadoc**. Utilizaremos los delimitadores `/**` y `*/`. Al igual que con los comentarios tradicionales, el texto entre estos delimitadores será ignorado por el compilador. Este tipo de comentarios se emplean para generar documentación automática del programa. A través del programa javadoc, incluido en JavaSE, se recogen todos estos comentarios y se llevan a un documento en formato `.html`.

```
/** Comentario de documentación.  
Javadoc extrae los comentarios del código y  
genera un archivo html a partir de este tipo de comentarios  
*/
```

11. Herramientas útiles para empezar

11.1. Generar números aleatorios.

Podemos generar números aleatorios entre 0 y 1 utilizando el método `random` de la clase `Math`.

```
Math.random()
```

Ejemplo:

```
double numero;
int entero;
numero = Math.random();
System.out.println("El número es: "+numero);
numero = Math.random()*100;
System.out.println("El número es: "+numero);
entero = (int)(Math.random()*100);
System.out.println("El número sin decimales es: "+entero);
```

11.2. Introducir un texto desde el teclado.

Este método de leer texto y números desde consola no nos servirá cuando comencemos a usar IDE's.

Podemos introducir texto desde el teclado utilizando `System.console().readLine()`;

Ejemplo 1: Introducción de texto.

```
String texto;
System.out.print("Introduce un texto: ");
texto = System.console().readLine();
System.out.println("El texto introducido es: "+ texto);
```

Ejemplo 2: Introducción de un número entero.

```
String texto2;
int entero2;
System.out.print("Introduce un número: ");
texto2 = System.console().readLine();
entero2 = Integer.parseInt(texto2);
System.out.println("El número introducido es:"+entero2);
```

Ejemplo 3: Introducción de un número decimal.

```
String texto3;
double doble3;
System.out.print("Introduce un número decimal: ");
texto3 = System.console().readLine();
doble3 = Double.parseDouble(texto3); // convertimos texto a doble
System.out.println("Número decimal introducido es: "+doble3);
```

12. Ejemplo UD01

```
public class EjemploUD01 {

    // variable de clase precisa static para poder usarse dentro de la funcion main()
    static double dto = 0.25;

    public static void main(String[] args) {

        // Declaración y asignación de valores a variables
        byte a;
        a = 127;
        short c = 3;
        int d = -30;
        int e = 0xC125;
        long l = 5L;
        char car1 = 99; //car1 es la letra "c" equivale al ascii 99
        char letra = '\u0061'; //código unicode del carácter "a"
        double b = 5F;
        double f = 1e6;
        float g = 1 / 2F;
        boolean par = false;

        // Declaración y asignación constantes y literales
        final double IVA = 0.21;
        System.out.println("1a linea\n2a linea\n3a \"linea\"");

        //Muestra por pantalla literales y contenidos de variables.
        System.out.println("Hola Mundo");
        System.out.println("a vale " + a);
        System.out.println("b vale " + b);
        System.out.println("c vale " + c);
        System.out.println("d vale " + d);
        System.out.println("e vale " + e);
        System.out.println("f vale " + f);
        System.out.println("g vale " + g);
        System.out.println("g vale " + l);
        System.out.println("g vale " + car1);
        System.out.println("g vale " + letra);
        System.out.println("g vale " + par);

        // uso de la constante
        double precio = 430;
        double preciofinal = precio + ((precio - (precio * dto)) * IVA) - (precio * dto);
        System.out.println(IVA);
        System.out.println(preciofinal);

        // Operadores aritméticos
        double num1, num2, suma, resta, producto, division, resto;
        num1 = 8;
        num2 = 5;
        suma = num1 + num2;      // 13
        resta = num1 - num2;     // 3
        producto = num1 * num2;  // 40
        division = num1 / num2;  // 1.6
        resto = num1 % num2;     // 3
    }
}
```

```

System.out.println("Suma: " + suma);
System.out.println("Resta: " + resta);
System.out.println("Producto: " + producto);
System.out.println("División: " + division);
System.out.println("Resto: " + resto);

// Operadores Relacionales
int valor1 = 10;
int valor2 = 3;
boolean compara;
compara = valor1 > valor2; // true
System.out.println("Mayor:" + compara);
compara = valor1 < valor2; // false
System.out.println("Menor:" + compara);
compara = valor1 >= valor2; // true
System.out.println("Mayor o igual: " + compara);
compara = valor1 <= valor2; // false
System.out.println("Menor o igual: " + compara);
compara = valor1 == valor2; // false
System.out.println("Igual: " + compara);
compara = valor1 != valor2; // true
System.out.println("Distinto: " + compara);

//Operadores Lógicos
double sueldo = 1400;
int edad = 34;
boolean logica;
logica = (sueldo > 1000 & edad < 40); //true
System.out.println("AND: " + logica);
logica = (sueldo > 1000 && edad > 40); //false
System.out.println("AND: " + logica);
logica = (sueldo > 1000 | edad > 40); //true
System.out.println("OR: " + logica);
logica = (sueldo < 1000 || edad > 40); //false
System.out.println("OR: " + logica);
logica = !(edad < 40); //false
System.out.println("NOT: " + logica);
logica = (sueldo > 1000 ^ edad > 40); //true
System.out.println("XOR: " + logica);
logica = (sueldo < 1000 ^ edad > 40); //false
System.out.println("XOR: " + logica);

//Operadores Unarios o Unitarios
int m = 5, n = 3;
m++; // 6
n--; // 2
System.out.println("Incremento: " + m);
System.out.println("Decremento: " + n);

int A = 1, B;
B = ++A; // A vale 2 y B vale 2
System.out.println("A vale: " + A + " B vale: " + B);
B = A++; // A vale 3 y B vale 2
System.out.println("A vale: " + A + " B vale: " + B);

//Operadores de Asignación
int dato1 = 10, dato2 = 2, dato;
dato = dato1; // dato vale 10

```



```

short destino;
destino = origen; // 5
System.out.println("Implícita: " + destino);

// Conversión Explícita
byte destino2;
short origen2 = 3;
destino2 = (byte) origen2; // 3
System.out.println("Explícito: " + destino2);

// comentario de una sola línea

/* Esto es un comentario
de varias líneas */

/** Comentario de documentación.
Javadoc extrae los comentarios del código y
genera un archivo html a partir de este tipo de comentarios
*/

//Generar número aleatorios
double numero;
int entero;
numero = Math.random();
System.out.println("El número es: "+numero);
numero = Math.random()*100;
System.out.println("El número es: "+numero);
entero = (int)(Math.random()*100);
System.out.println("El número sin decimales es: "+entero);

//Introducir texto desde teclado
String texto;
System.out.print("Introduce un texto: ");
texto = System.console().readLine();
System.out.println("El texto introducido es: "+ texto);

//Introducir un número entero desde teclado
String texto2;
int entero2;
System.out.print("Introduce un número: ");
texto2 = System.console().readLine();
entero2 = Integer.parseInt(texto2);
System.out.println("El número introducido es:"+entero2);

//Introducir un número decimal desde teclado
String texto3;
double doble3;
System.out.print("Introduce un número decimal: ");
texto3 = System.console().readLine();
doble3 = Double.parseDouble(texto3); // convertimos texto a doble
System.out.println("Número decimal introducido es: "+doble3);
}
}

```

13. Píldoras informáticas relacionadas

- [Curso Java. Estructuras principales I. Vídeo 4](#)
- [Curso Java. Estructuras principales II. Vídeo 5](#)
- [Curso Java. Estructuras principales III. Declaración variables Eclipse. Vídeo 6](#)
- [Curso Java. Estructuras principales IV. Constantes y Operadores. Vídeo 7](#)
- [Curso Java. Estructuras principales V. Constantes y Operadores II. Vídeo 8](#)

14. Fuentes de información

- [Wikipedia](#)
- [Programación \(Grado Superior\) - Juan Carlos Moreno Pérez \(Ed. Ra-ma\)](#)
- Apuntes IES Henri Matisse (Javi García Jimenez?)
- Apuntes AulaCampus
- [Apuntes José Luis Comesaña](#)
- [Apuntes IOC Programació bàsica \(Joan Arnedo Moreno\)](#)
- [Apuntes IOC Programació Orientada a Objectes \(Joan Arnedo Moreno\)](#)