

---

---

# ANEXO: ASPECTOS BÁSICOS DE JAVA

---

---

## A.1.- PALABRAS CLAVE JAVA

---

Esta es simplemente una referencia alfabética de las palabras clave reservadas en el lenguaje Java:

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

## A.2.- JAVA Y C

---

Java es como C y C++ en muchos detalles...

- Los identificadores se forman con una letra seguida de letras o dígitos. También pueden incluir o empezar en \$ y \_.
- Como en C, la capitalización de las letras es significativa. Java distingue entre mayúsculas y minúsculas tanto para identificadores como para palabras clave del lenguaje.
- Las llaves engloban los bloques: clases, funciones, etc. empiezan en { y acaban en }. Las funciones y las clases tienen una cabecera y un cuerpo, éste indicado entre llaves.
- Las cabeceras de funciones tienen el mismo formato: especificadores + tipo devuelto + nombre + lista de parámetros entre paréntesis.  

```
[modificadores] tipoDevuelto nombreFuncion( [listaParámetros] )
```
- Los tipos preceden a los nombres en las declaraciones de variables.
- En las declaraciones puede asignarse un valor inicial, indicando un igual (=) y el valor después del nombre de la variable.
- Los comentarios van entre /\* y \*/ o bien (como en C++) hay comentarios de línea, que empiezan en // y acaban con el final de esa línea física.

Pero no es como C++ en todos...

Precisamente simplifica a C++ en muchos aspectos. Concretamente:

- El ámbito de las variables es el bloque en el que se definen, con excepción de la variable contador del for que, si se declara en el mismo, deja de existir al final de su cuerpo.

- No permite sobrecarga de operadores (se usa el + con String pero nada más).
- No permite herencia múltiple (en su lugar usa interfaces).
- Tiene gestión de memoria dinámica automática, tanto en su creación como especialmente en su liberación. Por esto mismo, no tiene métodos destructores como tal.
- En cuanto a la localización de los fuentes, las declaraciones y las implementaciones se hacen en el mismo lugar, en el mismo fichero. Desaparece la necesidad de los ficheros de cabecera y de las declaraciones adelantadas.

## A.3.- COMENTARIOS

---

Java tiene 3 tipos de comentarios. Ya hemos visto que 2 de ellos son como en C/C++:

- `/**` comenta hasta el final de la línea.
- `/*` comenta hasta el primer `*/` (no se permite anidamiento).

El tercer tipo de comentario es exclusivo de Java, se utiliza para generar documentación con la herramienta proporcionada por JavaSoft, llamada "javadoc":

- `/**` comenta hasta el primer `*/`.

## A.4.- TIPOS DE DATOS

---

Los tipos se clasifican en:

- tipos primitivos
- clases

### A.4.1.- Tipos de datos primitivos

Los tipos primitivos se dividen en:

- Numéricos
  - Enteros
    - byte
    - short
    - int
    - long

- Reales
  - float
  - double
- Carácter: char
- Lógicos: boolean

## Tipos Numéricos Enteros

Para construir una variable entera hay que hacer dos cosas: primero, decidir el tamaño máximo y mínimo que se espera que tenga el número; segundo, declarar el entero dándole un nombre.

En la siguiente tabla se ve los tamaños máximos y mínimos admitidos por los tipos enteros de Java:

Tipo	Mínimo valor negativo	Máximo valor negativo	Mínimo valor positivo	Máximo valor positivo
byte	-128	-1	0	127
short	-32.768	-1	0	32.767
int	-2.147.483.684	-1	0	
long	9.223.372.036.854.755.808	-1	0	9.223.372.036.854.755.808

El tipo byte ocupa 1 byte, short ocupa 2 bytes, int ocupa 4 y long ocupa 8 bytes.

Los literales enteros son cualquier secuencia de dígitos (sin punto decimal). Por omisión se toman de tipo int, excepto si el valor no cabe en 32 bits, en cuyo caso se define un long. Si se quiere indicar explícitamente un literal de tipo long (64 bits), hay que añadir al número el sufijo L. Por ejemplo:

34                      34L

Cuando el literal empieza por 0x, se considera una constante hexadecimal. Por otro lado, si comienza por 0 se considera octal. Por ejemplo:

16                      016                      0x16

Cuando no se indica un valor inicial, toda variable entera se inicializa automáticamente a 0.

## Tipos Numéricos Reales

Como en los tipos de enteros, los tipos de coma flotante o reales, se presentan en varios tamaños: float, que se almacena en 32 bits (con 7 dígitos de precisión) y double, que se almacena en 64 bits (con 15 dígitos de precisión). El rango de valores que abarcan se muestra en la siguiente tabla:

Tipo	Mínimo valor negativo	Máximo valor negativo	Mínimo valor positivo	Máximo valor positivo
float	-3.40282347 e+38	-1.40239846e-45	1.40239846e-45	3.40282347 e+38
double	- 1.79769313486 23147 e+308	- 4.940656458412 46544 e-324	4.940656458412 46544e-324	1.797693134862 3147 e+308

Los literales reales son los que tienen punto decimal o los expresados en notación exponencial (mantisa, E, exponente), como 1.34E18. Por omisión son double. También se pueden utilizar los sufijos F ó D para explicitar su tipo: float o double, respectivamente. Por ejemplo:

3.14

3.14F

Toda variable real no inicializada explícitamente comienza con valor 0.0.

## Tipo Carácter

El tipo carácter se denota con la palabra clave char. Permite almacenar un carácter codificado en Unicode, que es un estándar para codificación de caracteres internacionales mucho más general que el ASCII. Utiliza dos bytes por cada carácter, mientras que el código ASCII sólo utilizan uno. Unicode es compatible con la parte estándar del código ASCII (los 7 bits de menor peso, o sea, los 128 primeros caracteres ASCII). Por otro lado, con Unicode se permite almacenar caracteres de otros idiomas, como el chino, el japonés y el coreano, que no utilizan alfabetos, sino un extenso conjunto de ideogramas, uno por cada palabra. Estos ideogramas no se podían representar con 8 bits, que es lo que puede almacenar el código ASCII.

Unicode es el código nativo de Java, así como de los sistemas operativos más recientes, como Windows NT. Puede ser que otros sistemas operativos no soporten Unicode, por eso en ellos, aunque Java lo permita, el entorno no lo podrá apreciar.

Las constantes de carácter van entre comillas simples. Por ejemplo:

'A'            'Z'            '1'

Como en C, los caracteres especiales pueden indicarse con códigos de escape: '\u0001' para indicar un código Unicode con sus cuatro dígitos hexadecimales. Así, por ejemplo:

'\u0041'        ==        'A'

Las secuencias de escape más comunes son: '\n' línea nueva, '\r' retorno de carro, '\t' tabulador. También hay otras como: '\b', '\f', '\", '\'', '\\', y '\ddd' en octal.

Las variables char se inicializan con valor de carácter nulo (código 0, '\u0000').

## Tipo Lógico

El tipo lógico se denota por la palabra clave boolean. Sólo puede tomar dos valores true y false (verdadero y falso). Por definición, un boolean ocupa sólo un bit de almacenamiento, y tiene un valor predeterminado false.

## A.4.2.- Tipos de datos referencia

Todos los demás tipos en Java son clases. Y las clases internamente se gestionan como referencias. Esto es, cada vez que creamos una instancia x de la clase X, el espacio que se reserva es el necesario para una dirección (32 bits).

Todos los datos de tipo objeto descienden, directa o indirectamente, de la clase Object.

Toda variable de un tipo de clase T (o sea, referencia) mantiene una referencia nula, expresada por la palabra clave null, o bien apunta a una instancia de la clase T o alguna de sus subclases.

Por omisión, su valor es null.

## A.5.- OPERADORES

---

### A.5.1.- Operadores sobre enteros

Java admite 8 tipos de operadores para números enteros:

- Operadores de asignación

Hay 5 tipos de operadores de asignación:

- = Para asignación simple; coloca el valor de la parte derecha en la variable de la parte izquierda.
- += Añade el valor de la derecha a la variable de la izquierda y almacena el resultado en la variable.
- -= Sustraer el valor de la derecha de la variable de la izquierda y almacena el resultado en la variable.
- \*= Multiplica la variable de la parte izquierda por el valor de la parte derecha y almacena el resultado en la variable.
- /= Divide la variable de la parte izquierda por el valor de la parte derecha y almacena el resultado en la variable.

Por ejemplo:

```
unEntero = 5; // Coloca el valor 5 en la variable unEntero.
unNumero +=3; // Añade 3 a la variable unNumero y almacena el resultado
              // en esa variable.
```

- Operadores de comparación

Se puede comprobar la igualdad usando el operador: ==, y la desigualdad con: !=.

También existen los operadores: < (menor que), <= (menor o igual que), > (mayor que), >= (mayor o igual que).

- Operadores de signo unitarios

Se puede cambiar el signo de un valor utilizando el operador: -. Así, si unaVariable contiene el valor 5, -unaVariable será igual a -5.

- Operadores de suma, resta, multiplicación y división

Estos operadores son: + (suma), - (resta), \* (multiplicación), / (división), % (módulo o resto). Estos dos últimos operadores son los únicos operadores que producen una excepción si el divisor es cero. Por lo demás, no se generan errores de overflow ni underflow.

- Operadores de incremento y decremento

Son ++ (incremento) y -- (decremento). Se puede aplicar estos operadores antes o después de una variable (como prefijo o sufijo, respectivamente). Si se utiliza el operador en la posición prefijo, el valor de la variable cambiará antes de que el operador devuelva el valor. Si se coloca el operador en posición sufijo, se devolverá el valor de la variable, y a continuación se aplicará el operador (para incrementar o disminuir la variable). Por ejemplo:

```
int num1;
int num2 = 4;

num1 = ++num2; // Resultado:
               //   num2 = 5;
               //   num1 = 5;

num1 = num2++; // Resultado:
               //   num2 = 5;
               //   num1 = 4;
```

- Operadores de desplazamiento a nivel de bit

Viendo un número entero como un patrón de bits, se le pueden aplicar una serie de operadores de desplazamiento:

- >> Desplaza el patrón un número determinado de lugares hacia la derecha (y el bit de signo se copiará a la derecha).

- >>> (Operador lógico de desplazamiento a la derecha) Con él se impide que copie el signo al realizar el desplazamiento.
- << Desplazamiento a la izquierda. Por supuesto, no hay diferencia entre el desplazamiento a la izquierda lógico y aritmético, puesto que no hay bit de signo en el extremo derecho.
- Operador de negación lógica a nivel de bit

Se puede complementar cada bit de un entero utilizando el operador ~.
- Operadores AND, OR y XOR a nivel de bit
  - & Operador AND.
  - | Operador OR.
  - ^ Operador XOR.

Los operadores de igualdad y comparación siempre producen un resultado boolean (true o false). Los otros operadores binarios producen un int o un long (nunca un byte o un short). Si alguno de los dos operandos es un long, el resultado es un long. Si el resultado de aplicar el operador desborda un int, se convertirá en un long. De lo contrario los operadores devuelven un int.

### A.5.2.- Operadores sobre reales

Los números de coma flotante admiten 5 tipos de operadores:

- Operadores de asignación
- Operadores de comparación
- Operadores de signo unitarios
- Operadores de suma, resta, multiplicación y división
- Operadores incremento y decremento

Los símbolos para estos operadores son idénticos a los utilizados para los enteros. Las reglas de combinación de tipos son análogas a las utilizadas en el caso de los enteros. El resultado de una operación binaria que involucre al menos a un double, es un double.

### A.5.3.- Operadores sobre booleanos

Se pueden aplicar 10 operadores a los booleanos:

- Asignación (=)

- Igualdad (==)
- Desigualdad (!=)
- NOT lógico (!) para cambiar de true a false y viceversa.
- AND (también & ó &&)
- XOR (^)
- if-then-else (?:)

## A.6.- CONVERSIÓN DE TIPOS

---

Java es un lenguaje con sistema de tipos fuerte (strongly typed). Esto es, toda variable y expresión tiene un tipo conocido en compilación. Esto permite detectar muchos errores sin necesidad de ejecutar el código.

Veamos las conversiones de tipos más importantes que tenemos que considerar:

### Conversiones sobre enteros y reales

Toda operación con enteros se hace en 32 bits (tipo int), de no ser que alguno de los operandos sea un long, en cuyo caso la operación se realiza en 64 bits.

Si cualquiera de los operandos de una expresión numérica es real, la operación es real entonces, convirtiéndose los operandos pertinentes a ese tipo.

### Conversiones de booleanos

No se puede convertir ningún tipo de datos a booleano. En todo caso hay que escribir la expresión correspondiente, como (n != 0) para un número y (obj != null) para una instancia.

### Conversiones entre tipos primitivos

Java tiene dos tipos de conversiones:

- Implícitas (o realizadas automáticamente por el compilador).
- Explícitas (o realizadas por el programador mediante un cast).

Conversiones implícitas:

Los tipos byte y short se convierten a int antes de operar.

Cuando en una expresión hay operandos de distinto tipo se promociona el de menor tipo.

```
double d = 3.4;
```

```
int i = 5;

i + d;    // Suma real

double x = 5/2;
```

En las asignaciones se realiza una conversión implícita si el tipo de la derecha es menor o igual que el de la izquierda.

```
d = i;    //Ok
i = d;    // Error. Pérdida de información.
```

### Conversiones explícitas:

Cuando se desea provocar una conversión se utiliza el operador cast.

```
( tipoAConvertir )expresion
```

Ejemplo:

```
5.4
(int)5.4
```

Usando un cast se pueden realizar aquellas conversiones que el compilador no realiza por defecto.

## A.7.- ESTRUCTURAS DE CONTROL

---

### Estructura alternativa if

```
if (<expresión booleana>
{
    <sentencias>
}
else {
    <sentencias>
}
```

- Los paréntesis son obligatorios.
- Las llaves son opcionales si sólo hay una sentencia.
- El "else" es opcional.

Ejemplo:

```
if (a > b)
c = a;
else
c = b;
```

- Es importante emparejar cada if con su else.

```
int a = 4, b = 6, c = 0;
if (a > b)
```

```
if (a > 5)
    c = 1;
else
    c = 2;
```

## Operador ternario.

(<expresión booleana>) ? <expr2> : <expr3>

- Evalúa la expresión booleana:
- Si es cierta devuelve el resultado de evaluar la segunda expresión.
- Si es false devuelve el resultado de evaluar la tercera expresión.

### Ejemplo:

```
c = (a > b) ? a : b; // Max(a, b)
```

- Es una forma compacta de if.

```
if (a > b)
    c = a;
else
    c = b;
```

## Bucle while

```
while (<expresión booleana>)
{
    // sentencias
}
```

- Los paréntesis son obligatorios.
- Las llaves son opcionales si solo hay una sentencia.

```
int i = 1;
int tabla = 8; // Tabla multiplicar del 8
while (i < 11)
    System.out.println(i++ * tabla);
```

## Bucle do-while

```
do
{
    <sentencias>
} while (<expresión booleana>);
```

- Paréntesis y ";" obligatorios.
- Llaves opcionales si sólo hay una sentencia.

```
int i = 1;
int tabla = 8; // Tabla multiplicar del 8
do
    System.out.println(i * tabla);
while (++i < 11);
```

## Bucle for

```
for (<init>; <expr booleana> ; <reinit>)
{
    <sentencias>
}
```

- Es equivalente a:

```
{
    <init>
    while (<expr booleana>)
    {
        <sentencias>
        <reinit>
    }
}
```

### Ejemplo:

```
for (int i = 1; i < 11; i++)
    System.out.println(i * tabla);
```

- Las variables de la inicialización no son visibles fuera del for.

```
for (int i = 1; i < 11; i++)
    System.out.println(i * tabla);
i = 5;    // Error
```

- Todas las expresiones son opcionales.

```
for ( ; ; )
    System.out.println(".");
```

- Se pueden usar comas para iniciar/reiniciar varias expresiones a la vez:

```
for (i=0, j=f(x) ; i < 10 && j >= k ; i++, j= i * k)
    System.out.println(".");
```

## Ruptura de bucles

- BREAK: Produce la salida de un bucle for, while, do-while y switch.

```
while (c = leerCaracter()) {
    if (c == '\n')
        break;
    escribirCaracter(c);
}
```

- Solo afecta al bucle inmediato.

```
while (true) {
    while(true) {
        break;
        System.out.println("0");
    }
    System.out.println("1");
}
System.out.println("2");
```

- Se pueden etiquetar los bucles para indicar el nivel de salida.

```
nivell1:
while (true) {
    while(true) {
        break nivell1;
        System.out.println("0");
    }
    System.out.println("1");
}
System.out.println("2");
```

- CONTINUE: Finaliza la iteración actual y pasa a evaluarse la condición para la próxima iteración.

- Válida en bucles while, do-while y for.

```
while (c = leerCaracter()) {
    if (c == '\n')
        continue;
    escribirCaracter(c);
}
```

- Solo afecta al bucle inmediato.

```
while (true) {
    System.out.println("1");
    while(true) {
        System.out.println("0");
        continue;
    }
}
System.out.println("2");
```

- Se pueden etiquetar los bucles para indicar el nivel de salida.

```
nivell1:
while (true) {
    System.out.println("1");
    while(true) {
        System.out.println("0");
        continue nivell1;
    }
}
System.out.println("2");
```

## Estructura switch

- Permite seleccionar entre distintas alternativas de tipo entero.
- Estructura primitiva (legado de C) y que generalmente indica un diseño pobre (¿herencia?).

```
switch (<expr entera>) {
    case <cte1>:
        <sentencias>;
    case <cte2>:
        <sentencias>;
```

```
    case <cte3>:  
        <sentencias>;  
    default:  
        sentencias;  
}
```

- Se evalúa la expresión entera y se pasa el control a las sentencias cuyo case coincida con el valor.
- Si ninguna coincide
- Si hay default se ejecutan sus sentencias.
- Si no lo hay se sale del switch.
- Una vez seleccionada una rama se ejecutan todas las sentencias siguientes (default incluido).
- El switch no permite rangos ni expresiones que no sean enteras.

Ejemplo:

```
switch (contador) {  
    case 1:  
        System.out.println("1");  
    case 2:  
        System.out.println("2");  
        break;  
    case 3:  
        System.out.println("3");  
    default:  
        System.out.println("4");  
}
```