**Convenios y reglas para nombrar variables.**

| **Convenciones sobre 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. | 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. | String, MiTipo |
| Nombre de un método. | Comienza con letra minúscula. | modifica\_Valor, obtiene\_Valor |

**Palabras reservadas.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Palabras clave en Java** | | | | |
| 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 | strictfcode | volatile |
| const | float | native | super | while |

No son palabras reservadas, pero no se pueden usar como identificadores: true, false, null

**Variables referencia:** variables cuyo tipo es una clase. (Objetos)

**Variables miembro:** Son las variables que se crean dentro de una clase, fuera de cualquier método.

**Variables locales:** Son las variables que se crean y usan dentro de un método o, en general, dentro de cualquier bloque de código.

Los **float** se emplean para representar los números en coma flotante de simple precisión de 32 bits, de los cuales 24 son para la mantisa y 8 para el exponente. La mantisa es un valor entre -1.0 y 1.0.

Los **double** representan los números en coma flotante de doble precisión de 64 bits, de los cuales 53 son para la mantisa y 11 para el exponente.

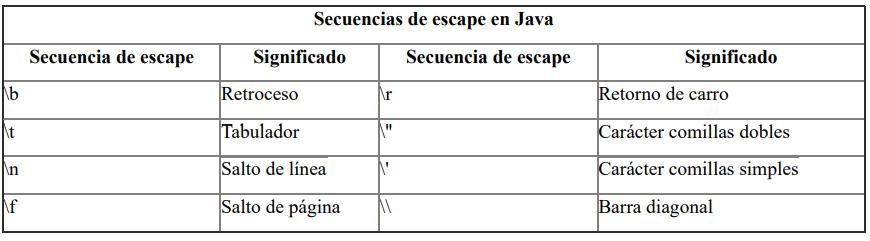
**Literales**

[Booleanos, enteros, reales, caracteres, cadenas de caracteres]

Octal: por ejemplo 024. Un número en octal **siempre empieza por cero** (seguido de dígitos octales (del 0 al 7)).

(Hexadecimal: Empieza por 0x)

Un literal **carácter** puede escribirse como un carácter entre comillas simples como 'a', 'ñ', 'Z', 'p', etc. o por su código de la tabla Unicode, anteponiendo la **secuencia de escape ‘\’ si el valor lo ponemos en octal o ‘\u’ si ponemos el valor en hexadecimal**.



**Operaciones con String:**

Obtención de longitud. Si necesitamos saber la longitud de un String, utilizaremos el método length() .

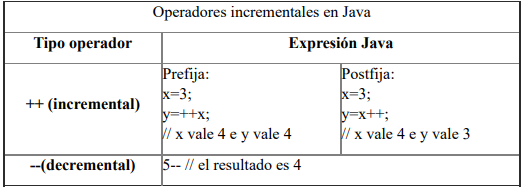
Concatenación. Se utiliza el operador + o el método concat() para concatenar cadenas de caracteres.

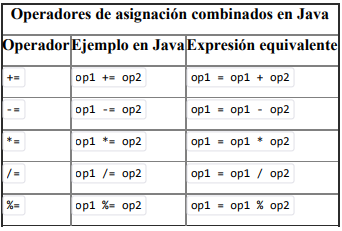
Comparación. El método equals() nos devuelve un valor booleano que indica si las cadenas comparadas son o no iguales. El método equalsIgnoreCase () hace lo propio, ignorando las mayúsculas de las cadenas a considerar.

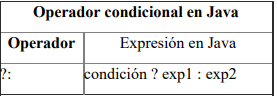
Obtención de subcadenas. Podemos obtener cadenas derivadas de una cadena original con el método substring (), al cual le debemos indicar el inicio y el fin de la subcadena a obtener.

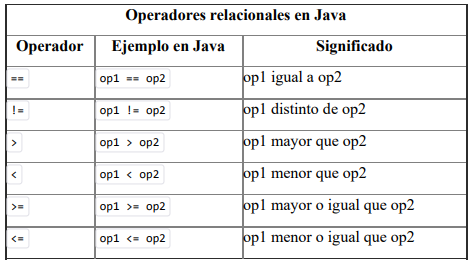
Cambio a mayúsculas/minúsculas. Los métodos toUpperCase() y toLowerCase() devuelven una nueva variable que transforma en mayúsculas o minúsculas, respectivamente, la variable inicial.

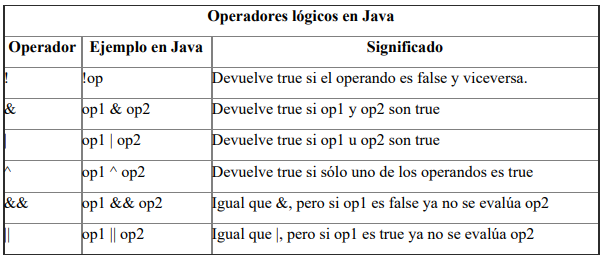
Valueof: Utilizaremos este método para convertir un tipo de dato primitivo ( int , long , float , etc.) a una variable de tipo String .





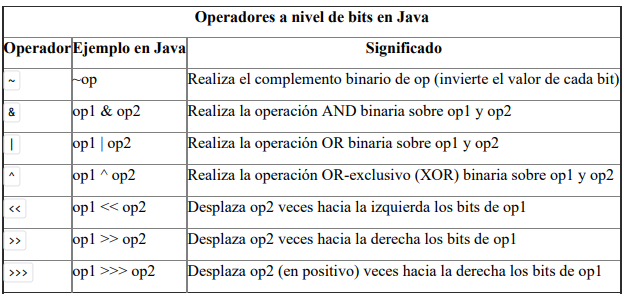




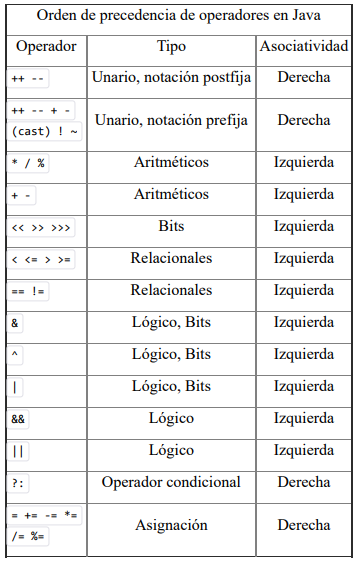


**op1 ^ op2 Devuelve true si sólo uno de los operandos es true (OR exclusivo)**

[WTF?



]



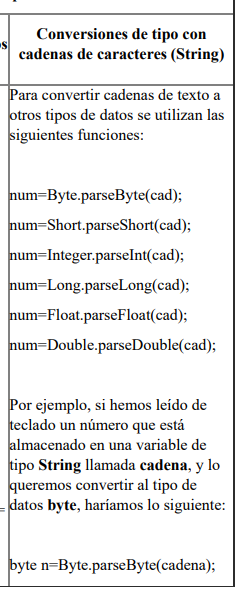
**Los operadores de asignación, el operador condicional ( ?: ), los operadores incrementales ( ++, -- ) y el casting son asociativos por la derecha. (realiza primero las operaciones a su derecha y, después, se aplica)**

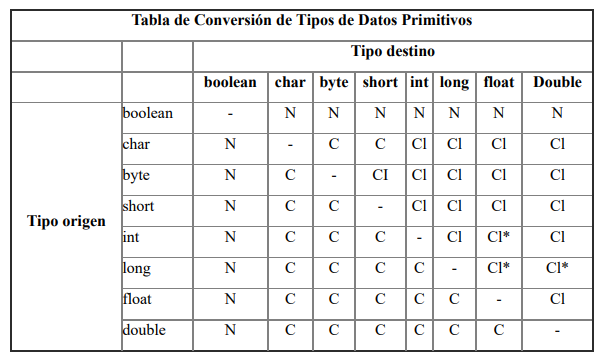
**(cast)**:

int a; byte b;

byte b = (byte) a; // Hace falta pasar expresamente “a” a byte mediante cast (rango menor a int)

(haciendo eso se pueden perder datos)





**Arrays**

int[] n; // Declaración del array.

n = new int[10]; //Creación del array reservando para el un espacio en memoria.

int[] m=new int[10]; // Declaración y creación en un mismo lugar.