

# UT2 Identificación de los elementos de un programa informático en Java

1.- Estructura y bloques fundamentales.   
2.- Variables.   
3.- Tipos de datos.   
4.- Literales.   
5.- Constantes.   
6.- Operadores y expresiones.   
7.- Conversiones de tipo.   
8.- Comentarios.

## Estructura y bloques fundamentales de un programa Java

El código fuente de un programa en Java debe estar escrito en un **fichero de texto con extensión ".java"**. Para crear o editar el fichero debe utilizarse un editor de texto sin formato como el bloc de notas de Windows, el editor de texto de Linux, o alguno de los programas y entornos de desarrollo específicos para la creación de aplicaciones.

public class HolaMundo {

public static void main(String args[]) {

System.out.println("Hola Mundo!");

}

}

La primera línea de código indica el nombre de la clase Java que estamos desarrollando. Todos los programas Java están formados por una o más clases. Es importante tener en cuenta que el nombre de la clase debe corresponder exactamente con el nombre del fichero de texto que contiene el código fuente. En el caso de este ejemplo, el código debe almacenarse en un fichero de texto denominado "HolaMundo.java".

public class HolaMundo

Las llaves { } indican el inicio y fin de cada bloque. Siempre deben ir en pareja, es decir, que por cada llave de apertura debe existir siempre una llave de cierre.

La siguiente línea realiza la declaración del método llamado "main". Cada clase Java que forma una aplicación contendrá uno o varios métodos, que son bloques de código que permiten tenerlo más organizado. Si una determinada clase Java contiene un método con el nombre "main", la ejecución de dicha clase comenzará por el código contenido en ese método.

public static void main(String args[])

La última línea de código del ejemplo, contiene las sentencias que se van a ejecutar. En este caso tan solo hay una sentencia, pero podría haber todas las que se quisiera. Aquí en concreto se utiliza la llamada a *System.out.println()* que permite mostrar en pantalla una serie de caracteres. El texto mostrado es el indicado entre comillas: "Hola Mundo!".

System.out.println("Hola Mundo!");

## Variables

Las ***variables*** identifican datos mediante un nombre simbólico, haciendo referencia a un espacio de memoria principal en los que se sitúan los datos, para que puedan ser utilizados por el procesador, y así poder hacer cualquier tipo de operación con ellos.

Los **nombres utilizados para identificar a las variables** deben cumplir una serie de **condiciones**:

* No pueden empezar por un dígito numérico.
* No pueden utilizarse espacios, y los únicos caracteres especiales válidos son el guión bajo (\_) y el símbolo del dólar ($).
* Son sensibles a las mayúsculas y minúsculas, es decir, las variables "suma" y "Suma" se consideran variables distintas.

No pueden utilizarse como nombres de variables las palabras reservadas de Java, que son las siguientes:



Es costumbre empezar los nombres de las variables por una letra minúscula.

Cuando el nombre de una variable está formado por más de una palabra, se suele utilizar una letra mayúscula para distinguir el comienzo de las palabras. Por ejemplo: sumaTotal.

Es recomendable utilizar nombres que hagan referencia al contenido que va a almacenar para facilitar la comprensión del código. Es mucho más claro utilizar el nombre "suma" que "s".

**Ejemplos** de nombres de variables **válidos**: indice, ventas, compras, saldoGeneral, importetotal, contador\_lineas, $valor, num2.

**Ejemplos** de nombres de variables **no válidos**: 3valores, suma&total, super, edad media.

Antes de poder utilizar una variable, esta debe ser declarada en el programa. La declaración de variables se debe realizar siguiendo el siguiente formato de sentencia:

tipoDato nombreVariable;

Donde tipoDato es uno de los tipos de datos básicos o el nombre de una clase (byte, short, int, long, float, double, boolean, char, String, etc), y nombreVariable es el nombre que se desea asignar a la variable siguiendo las normas anteriores.

Es posible declarar más de una variable de un mismo tipo en la misma línea separando los nombres con comas:

tipoDato nombreVariable1, nombreVariable2, nombreVariable3;

***Ejemplos*** de declaraciones de variables:

int num1, num2, suma;

char letraNIF;

String saludoInicial;

boolean mayorEdad;

Es posible asignar un valor inicial a las variables en el momento de declararlas. Para inicializar variables se debe seguir el siguiente formato de sentencia:

tipoDato nombreVariable = valorInicial;

Donde valorInicial puede ser un valor literal, otra variable declarada anteriormente o una expresión combinando valores literales y variables con operadores. El valorInicial debe ser del mismo tipo de dato que la variable que se está declarando.

Ejemplos de declaraciones de variables con inicialización:

int num1 = 34;

int doble = num1 \* 2;

String saludo = "Hola";

char letraA = 'A', letraB = 'B';

En el código del programa es posible asignar valores a las variables que previamente han sido declaradas. Al hacerlo, el valor que guardaba la variable anteriormente se perderá. Se debe utilizar el siguiente formato:

nombreVariable = valor;

Donde valor puede ser de nuevo un valor literal, una variable declarada anteriormente (puede ser la misma variable) o una expresión combinando valores literales y variables con operadores. El valor debe ser del mismo tipo de dato que la variable a la que se está asignando el nuevo valor.

package ejemplos;

public class AsignacionVariables {

public static void main(String[] args) {

int a=5, b=0, c;

b = a \* 3; // Se cambia el valor de b a 15

c = a; // Se guarda en c el valor de a que es 5

a = a + 6; // Se suma 6 al valor que tenía a. Ahora a vale 11

b = a - c; // b guarda 11 - 5 , luego vale 6

System.out.println("La variable a contiene: " + a);

System.out.println("La variable b contiene: " + b);

System.out.println("La variable c contiene: " + c);

}

}

Se muestra lo siguiente:

La variable a contiene: 11

La variable b contiene: 6

La variable c contiene: 5

Las variables pueden ser utilizadas dentro del bloque de código en el que han sido declaradas, es decir, dentro de las llaves "{" y "}" que marcan el inicio y el fin de un bloque de código. Se denomina **ámbito** de la variable al bloque de código en el que se declara la variable.

package ejemplos;

public class AmbitoVariables {

static int variableGlobal;

public static void main(String[] args) {

int variableDelMain = 10;

/\*Aquí se pueden usar variableGlobal y

variableDelMain. No se puede usar

variableDeOtroMetodo \*/

System.out.println("variableGlobal " + variableGlobal);

System.out.println("variableDelMain " + variableDelMain);

otroMetodo();

}

static void otroMetodo() {

int variableDeOtroMetodo=90;

/\* Aquí se pueden usar variableGlobal y

variableDeOtroMetodo. No se puede usar

variableDelMain \*/

System.out.println("variableGlobal " + variableGlobal);

//System.out.println("variableDelMain " + variableDelMain);

System.out.println("variableDeOtroMetodo " + variableDeOtroMetodo);

}

}

## Tipos de datos básicos

El lenguaje de programación Java permite la utilización de los siguientes tipos de datos básicos:

**Números enteros**: Representan a los números enteros (sin parte decimal) con signo (pueden ser positivos o negativos). Se dispone de varios tipos de datos, ocupando cada uno de ellos un espacio distinto en memoria. Cuanta más capacidad de almacenamiento, más grande es el rango de valores permitidos, aunque ocupará más espacio de memoria principal. Se dispone de los siguientes tipos:

* **byte**: Ocupan 8 bits (1 byte), permitiendo almacenar valores entre -128 y 127.
* (2⁸ = 256; necesitas 1 bit para el signo y uno de los valores será 0; 2⁷ = 128;)
* **short**: Ocupan 16 bits (2 bytes), permitiendo almacenar valores entre -32.768 y 32.767.
* **int**: Ocupan 32 bits (4 bytes), permitiendo almacenar valores entre -2.147.483.648 y 2.147.483.647. Es el tipo de datos por defecto para los valores numéricos enteros. Este tipo de datos es lo suficientemente grande para almacenar los valores numéricos que vayan a usar tus programas. solo se suelen usar los tipos anteriores si se pueden producir problemas con el espacio de memoria.
* **long**: Ocupan 64 bits (8 bytes), permitiendo almacenar valores entre

-9.223.372.036.854.775.808 y 9.223.372.036.854.775.807.

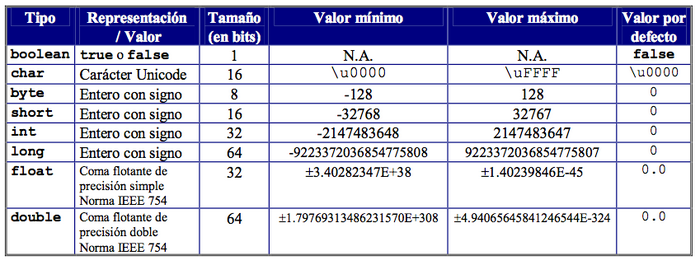
**Números reales**: Representan a los números reales con parte decimal y signo positivo o negativo. Hay dos tipos de datos numéricos reales que permiten obtener mayor o menor precisión. Utilizan un método para almacenar los datos que puede ocasionar que el valor original varíe levemente del valor almacenado realmente. Cuanta más precisión se utilice, habrá menor variación.

* **float:** Ocupan 32 bits (4 bytes). Se le denomina de simple precisión. Almacenan valores desde -3.40282347E+38 a + 3.40282347E+38

(E+x significa: 10 elevado a x)

* **double**: Ocupan 64 bits (8 bytes). Se le denomina de doble precisión. Es el tipo de datos por defecto para los valores numéricos reales. Almacenan valores desde:

- 1.79769313486231570E+308 a +1.79769313486231570E+308



**Valores lógicos:** Representan dos únicos posibles valores: verdadero y falso.

* **boolean:** Ocupan 1 bit, pudiendo almacenar los valores true (verdadero) y false (falso).

**Caracteres:** Representan las letras, dígitos numéricos y símbolos contenidos en la tabla de caracteres Unicode.

* **char:** Ocupan 16 bits (2 bytes). Permite representar un único carácter, encerrado entre comillas simples, por ejemplo la letra 'A'.

(https://unicode-table.com/en/)

* **String:** Realmente **no es un tipo de dato básico de Java**, pero por su interés se incluye aquí. Permite representar un conjunto de caracteres, encerrado entre comillas dobles, por ejemplo: "Saludos para todos".

(String empieza con mayúsculas porque es el nombre de una clase, una clase es el nombre de una plantilla)

## Valores literales – *Leer en casa*

Un valor literal es la representación de un valor fijo en el código fuente de un programa.

Los valores correspondientes a los tipos de datos numéricos enteros (byte, short, int y long) se pueden expresar usando el sistema numérico decimal, octal o hexadecimal. Los números en sistema decimal se expresan de la manera habitual, simplemente escribiendo su valor con los dígitos numéricos, por ejemplo, 284. No se pueden emplear separadores de millares, por lo que para indicar el valor 1.000.000 (un millón) se debe escribir como 1000000. Para representar valores negativos se añade el carácter "-" (guion) delante del número, como es habitual es la escritura normal, por ej -376.

Para representar un valor numérico entero en sistema octal, debe ir precedido del carácter 0 (cero), por ejemplo, el valor 284 se representa en octal como 0434. Asimismo, para representar un valor en el sistema hexadecimal, se debe emplear el prefijo 0x, por lo que el valor 284 se representa en hexadecimal como 0x11C.

Por defecto, los valores literales numéricos enteros se almacenan en memoria con el formato del tipo de dato "int". Si se desea almacenar como "long", con el fin de poder obtener resultados con valores muy altos en los cálculos, se debe emplear el sufijo L en mayúscula o minúscula (sería recomendable utilizar la L mayúscula por el parecido de la letra minúscula con el valor 1). Por ejemplo, el valor 284L correspondería al valor entero 284 utilizando 64 bits (tipo long) para almacenarlo en memoria.

Para representar valores literales de los tipos de datos numéricos reales (float y double) se puede emplear el sistema decimal o la notación científica. En el sistema decimal se expresan los números con parte decimal de la forma usual, utilizando el punto como separador de la parte entera y decimal. En este caso tampoco se puede emplear los separadores de millares. Así, por ejemplo, el valor 21.843,83 se debe expresar como 21843.83 en el código fuente.

Los números reales expresados en notación científica deben emplear la letra "E" o "e" para separar la parte correspondiente al exponente. El valor 7,433·10^6 se debe expresar como 7.433e6 en el código fuente. Si el exponente es negativo se escribe el guion detrás de la letra E, por ejemplo, 7,433·10^-6 se expresa como 7.433e-6, y si el valor es negativo se indica el guion al principio, por ejemplo, -7,433·10^6 se expresa como -7.433e6.

Por defecto, los valores literales numéricos reales se almacenan en memoria con el formato del tipo de dato "double". Si se desea almacenar como "float", con el fin de emplear menos espacio de memoria y necesitar menos precisión en los resultados de los cálculos, se debe emplear el **sufijo F en mayúscula o minúscula**. Por ejemplo, el valor 21843.83F correspondería al valor entero 21843.83 utilizando 32 bits (tipo float) para almacenarlo en memoria, en vez de 64 bits.

Los valores literales de los tipos char (carácter) y String (cadena de caracteres), pueden contener cualquier carácter Unicode. Los valores de tipo char se debe expresar encerrados entre comillas simples (en los teclados españoles habituales se encuentra junto a la tecla del cero), por ejemplo, la letra A se debe expresar como 'A'. Por otro lado, las cadenas de caracteres (tipo String) se expresan entre comillas dobles (en la tecla del 2), por ejemplo, el texto Saludos a todos, se debe escribir como "Saludos a todos".

**En caso de que se necesite escribir un carácter de la tabla de caracteres Unicode que no se encuentre en el teclado, se puede hacer indicando el código hexadecimal correspondiente a dicho carácter precedido del modificador \u (utilizando la barra invertida situada en la tecla junto al 1)**. En todo caso se debe encerrar entre comillas simples o dobles según se vaya a tratar como carácter o dentro de una cadena de caracteres. Por ejemplo, para escribir la letra griega beta (ß) se puede emplear '\u00DF', o para escribir la palabra España es posible utilizar "Espa\u00F1a".

El lenguaje de programación Java también soporta un pequeño conjunto de caracteres especiales que se pueden utilizar en los valores literales char y String:

\b (retroceso), \t (tabulador), \n (nueva línea), \f (salto de página), \r (retorno de carro), \" (comilla doble), \' (comilla simple), \\ (barra invertida).

Por ejemplo, la cadena de caracteres "La palabra "hola" es un saludo" se tiene que escribir en el código fuente como: "La palabra \"hola\" es un saludo", para que no confunda las comillas dobles de inicio y fin de la cadena de caracteres.

El carácter especial '\n' permite introducir un salto de línea dentro de una cadena de caracteres. Por ejemplo, "Primera línea\nSegunda línea" mostraría ese texto separado en dos líneas.

Los valores literales para el tipo de dato boolean sólo pueden ser true o false, que corresponden a los valores Verdadero y Falso. Hay que observar que se deben indicar sin comillas de ningún tipo, ya que no son cadenas de caracteres.

En un programa, es posible mostrar cualquiera de estos valores literales a través de la salida estándar (terminal, ventana de salida, etc) utilizando la siguiente sentencia:

System.out.println(valorLiteral);

Donde valorLiteral debe ser el valor que se desea mostrar manteniendo las normas comentadas.

Ejemplos:

//Mostrar un valor numérico entero

System.out.print("Número entero: ");

System.out.println(284);

//Mostrar un valor numérico real

System.out.print("Número real: ");

System.out.println(21843.83);

//Mostrar un carácter

System.out.print("Carácter: ");

System.out.println('A');

//Mostrar una cadena de caracteres

System.out.print("Cadena de caracteres: ");

System.out.println("Saludos a todos");

//Mostrar un valor lógico

System.out.print("Valor lógico: ");

System.out.println(true);

## Constantes

Son similares a las variables en cuanto que son datos a los que se hace referencia mediante un nombre y a los que se les asigna un valor. Pero a diferencia de las variables, a las constantes no se les puede modificar el valor asignado.

El formato de declaración de las constantes es prácticamente igual que el utilizado para las variables. La única diferencia es que se debe indicar el modificador final delante del tipo de dato que almacenará:

**final** tipoDato NOMBRECONSTANTE = valor;

Al igual que en la declaración de variables, el valor que se asigna en la declaración puede ser un valor literal, una variable o una expresión.

Ejemplo:

**final double** PI = 3.1415926536;

Por convenio, el nombre de las constantes se escribe en mayúsculas.

El beneficio de usar constantes es evitar la repetición de escribir un mismo valor en el programa y facilitar su modificación.

El empleo de las constantes a lo largo del programa es igual que el utilizado con las variables. Se indica su nombre dentro de cualquier expresión o como parámetro para métodos como println().

System.out.println("Perímetro = " + 2\*PI\*radio);

## Operadores

El lenguaje de programación Java incorpora una serie de operadores que permiten realizar cálculos y escribir expresiones que realicen una serie de operaciones sobre los datos.

### Operadores Aritméticos:

+ (suma)

- (resta)

\* (multiplicación)

/ (división entera o con decimales según operandos)

% (resto de la división)

Todos estos operadores aritméticos deben utilizarse con dos operandos, situados delante y detrás de los operadores, pudiéndose encadenar las operaciones. Se pueden incluir espacios para aclarar más el código. Los datos usados como operandos deben ser de alguno de los tipos de datos numéricos (byte, short, int, long, float o double).

***Ejemplos***:

4 + 3

8 - 5 + 2

6 \* 2 / 3

8.5 - 3 + 4.3

El resultado de la división tendrá decimales o no según el tipo de operandos que se utilice. Si los dos son enteros, el resultado no tendrá decimales, pero si al menos uno de los operandos es de tipo numérico real (float o double) el resultado será de ese tipo. Ejemplos:

8 / 2 resulta 4

7 / 2 resulta 3

7.0 / 2 resulta 3.5

7 / 2.0 resulta 3.5

7.4 / 2 resulta 3.7

8 / 2.5 resulta 3.2

8.5 / 2.5 resulta 3.4

Es posible modificar el tipo de dato de cualquier operando indicando delante el nuevo tipo de dato entre paréntesis. Así se hace una conversión de tipo (casting):

(double)7 / 2 resulta 3.5

7 / (float)2 resulta 3.5

El operador resto se debe utilizar con tipos de datos numéricos enteros *(a mi me ha funcionado también con double)*. El resultado será el resto de la división entre los dos operandos. Ejemplos:

7 % 2 resulta 1

8 % 3 resulta 2

El resultado de cualquier operación aritmética será del tipo de dato más grande que se utilice en los operandos. Por ejemplo, si se hace una operación entre dos números enteros (int) el resultado será del mismo tipo, pero si se hace entre un int y un long es resultado es de tipo long.

Ejemplos:

2147483647 \* 2 resulta un dato incorrecto, en concreto -2, porque se están multiplicando dos int y el resultado sobrepasa el límite de los enteros.

2147483647L \* 2 resulta 4294967294 porque el primer operando es de tipo long (se ha indicado L al final).

Los caracteres pueden ser utilizados para realizar cálculos aritméticos. En caso de que aparezca algún carácter en una expresión aritmética, se toma el valor numérico que le corresponde a cada carácter en la tabla de codificación Unicode.

Ejemplos:

'A' + 1 resulta 66, ya que el carácter 'A' tiene el código 65.

(char) ('A' + 1) resulta 'B' ya que se ha hecho una conversión de tipos del resultado.

### Operadores Relacionales

Los operadores relacionales permiten comparar dos valores numéricos.

> (mayor que)

>= (mayor o igual que)

< (menor que)

<= (menor o igual que)

== (igual que)

!= (distinto de)

Cada uno de estos operadores relacionales debe emplearse con dos valores numéricos a ambos lados, pudiendo ser dos valores literales o resultados de expresiones aritméticas.

Ejemplos:

4 > 3 resulta true.

7 <= 2 resulta false.

5 + 2 == 4 + 3 resulta true.

4 \* 3 != 12 resulta false.

### Operadores Lógicos

Los operadores lógicos permiten unir valores o expresiones lógicas, obteniendo como resultado si es verdadera o falsa la expresión combinada. Son los siguientes:

&& (Y lógico - conjunción)

|| (O lógico - disyunción)

! (NO lógico)

Los operadores && y || deben utilizarse con dos valores o expresiones lógicas a ambos lados, mientras que el operador de negación ! solo se aplica al valor o expresión lógica que tenga a su derecha. El resultado que se obtiene utilizando estos operadores se obtiene de la siguiente tabla de verdad:

El **operador Y** (&&) resulta true solo si ambos operandos son true:

true && true resulta true

true && false resulta false

false && true resulta false

false && false resulta false

El **operador O** (||) resulta true si al menos uno de los operandos son true:

true || true resulta true

true || false resulta true

false || true resulta true

false || false resulta false

El **operador NO** (!) resulta true si el operando es false:

!true resulta false.

!false resulta true.

Ejemplos:

4 > 3 && 5 <= 5 resulta true,

porque las dos expresiones son true.

4 > 3 && 5 != 5 resulta false,

porque al menos una expresión es false.

4 > 3 || 5 != 5 resulta true,

porque al menos una expresión es true.

4 > 3 && !(5 != 5) resulta true,

porque las dos expresiones son true.

### Operadores de Asignación

Permiten asignar valores a variables. El operador de asignación elemental es el igual (=) que asigna un valor o el resultado de una expresión a una variable, siguiendo el siguiente formato:

nombreVariable = valorAsignado;

El tipo de dato del valor asignado debe ser del mismo tipo que el de la variable a la que se pretende asignar. En caso de que no sean del mismo tipo, hay que utilizar algún tipo de conversión como el casting, utilizando entre paréntesis el nombre del tipo de dato al que se quiere convertir.

int numEntero;

numEntero = 364; //Correcto

numEntero = 264.38; //Incorrecto

numEntero = (int)264.83; //Correcto, se asigna el valor 264 (lo trunca)

También se puede utilizar el operador de asignación para guardar en una variable el resultado de una expresión que dé como resultado un valor del mismo tipo que la variable.

nombreVariable = Expresión;

int numEntero;

numEntero = 364 \* 2 + 266;

Además del operador igual (=), se pueden emplear otros operadores de asignación, que a la vez que asignan un valor realizan un cálculo:

**+=** (le suma a la variable un valor y guarda el resultado en la misma variable).

**-=** (le resta a la variable un valor y guarda el resultado en la misma variable).

**\*=** (multiplica la variable por un valor y guarda el resultado en la misma variable).

**/=** (divide la variable por un valor y guarda el resultado en la misma variable).

**%=** (obtiene el resto de dividir la variable por un valor y guarda el resultado en la misma variable).

calculo = 5;

calculo += 6; //Incrementa en 6 el valor de la variable: calculo = calculo +6

System.out.println(calculo); //Muestra 11

//Duplica el valor de cálculo: calculo = calculo\*2

calculo \*= 2;

System.out.println(calculo); //Muestra 22

Por tanto, una sentencia como a+=3 es lo mismo que a=a+3.

### Operadores Incrementales

Son los operadores que nos permiten incrementar las variables en una unidad

++ (incrementa en 1 el valor de una variable).

-- (decrementa en 1 el valor de una variable).

Ejemplo:

num1 = 3;

num2 = 7;

num1++; //Suma 1 a num1

System.out.println(num1); //Muestra 4

num2--; //resta 1 a num2

System.out.println(num2); //Muestra 6

Por tanto, una sentencia como x++ es los mismo que x=x+1, y x-- es lo mismo que x=x-1.

Hay dos formas de utilizar el incremento y el decremento, x++ ó ++x. La diferencia estriba en el modo en el que se comporta la asignación.

Ejemplo:

int x=5, y=5, z;

z=x++; /\* z vale 5, x vale 6 porque primero se asigna

el valor de x a z después se incrementa x \*/

z=++y; /\* z vale 6, y vale 6 porque primero incrementa

la variable y, después se asigna el valor a z \*/

### Operador Condicional. - (ha dicho que nos cuesta entenderlo)

*(No es lo mismo un operador condicional que una sentencia)*

Existe un operador condicional, que permite asignar a una variable un valor u otro dependiendo de una expresión condicional. El formato es el siguiente:

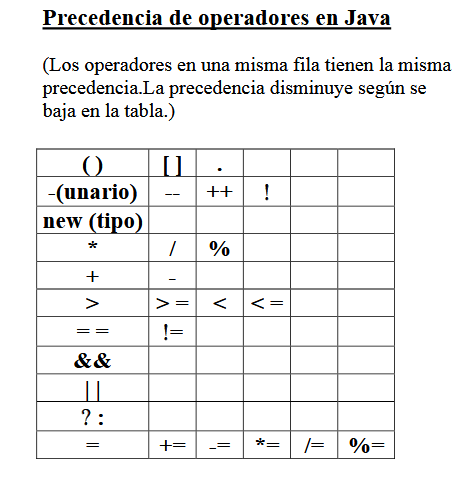
variable = condición ? valor1 : valor2;

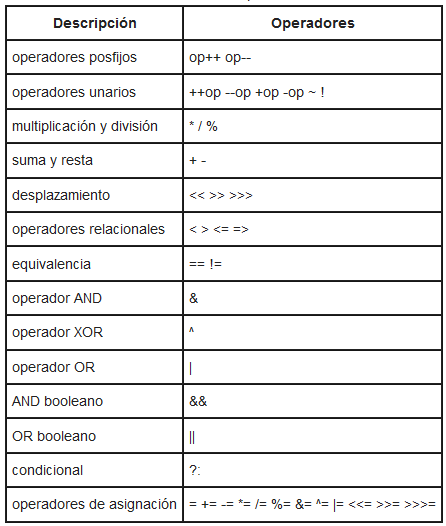
La condición que se indica debe ser una variable booleana o una expresión condicional que resulte un valor de tipo booleano (true o false).

Ejemplo:

mensaje = (num1 >= 0) ? "Positivo" : "Negativo";

Si se mostrara en pantalla posteriormente el valor de la variable mensaje, aparecerá "Positivo" si el valor de la variable num1 es mayor o igual que cero, y "Negativo" en caso contrario.





## Conversiones **d**e tipo

Java es un lenguaje fuertemente tipificado, lo que significa que es bastante estricto al momento de asignar valores a una variable (*esto ya no suele ser así, pero en versiones previas lo era*). El compilador sólo permite asignar un valor del tipo declarado en la variable; no obstante, en ciertas circunstancias es posible realizar conversiones que permiten almacenar en una variable un tipo diferente al declarado. En Java es posible realizar conversiones en todos los tipos básicos, con excepción de boolean, que es incompatible con el resto de los tipos.

Las conversiones de tipo pueden realizarse de dos maneras: implícitamente y explícitamente.

### Conversión implícita.

Las conversiones implícitas se realizan de manera automática, es decir, el valor o expresión que se va a asignar a una variable es convertido automáticamente por el compilador, antes de almacenarlo en la variable.

Para que una conversión pueda realizarse de manera automática (implícitamente), el tipo de la variable destino debe ser de tamaño igual o superior al tipo de origen, si bien esta regla tiene dos excepciones:

a) Cuando la variable destino es entera y el origen es decimal (float o double), la conversión no podrá ser automática.

b) Cuando la variable destino es "char" y el origen es numérico; independientemente del tipo específico, la conversión no podrá ser automática.

Ejemplos de conversiones implícitas:

// Declaraciones.

int j = 5, i;

short s = 10;

char c = 'ñ';

float f;

// Conversiones implícitas.

i = c; // Conversión implícita de char a int.

f = j; // Conversión implícita de int a float.

i = s; // Conversión implícita de short a int.

Los siguientes ejemplos de conversión implícita provocarían un error:

// Declaraciones.

int i;

long l = 20;

float ft = 2.4f;

char c;

byte b = 4;

// Conversiones implícitas.

i = l; // Error, el tipo destino es menor al tipo origen.

c = b; // Cuando la variable destino es "char" y el origen es numérico; independientemente del tipo específico, la conversión no podrá ser automática.

i = ft; // Cuando la variable destino es entera y el origen es decimal (float o double), la conversión no podrá ser automática.

### Conversión explícita.

Cuando no se cumplan las condiciones para una conversión implícita, ésta podrá realizarse de manera explícita utilizando la expresión:

variable\_destino = (tipo\_destino) dato\_origen;

Con esta expresión se obliga al compilador que convierta "dato\_origen" a "tipo\_destino" para que pueda ser almacenado en "variable\_destino". A esta operación se le conoce como "casting" o "conversión”.

Ejemplos de conversiones explícitas:

// Declaraciones.

char c;

byte b;

int i = 400;

double d = 34.6;

// Conversiones explícitas.

c = (char)d; // Se elimina la parte decimal (trunca), no se redondea.

b = (byte)i; // Se provoca una pérdida de datos, pero la conversión es posible.

public class Conversiones {

public static void main (String [] args) {

int a = 2;

double b = 3.0;

float c = (float) (20000\*a/b + 5);

System.out.println("Valor en formato float: " + c);

System.out.println("Valor en formato double: " +

(double) c);

System.out.println("Valor en formato byte: " +

(byte) c);

System.out.println("Valor en formato short: " +

(short) c);

System.out.println("Valor en formato int: " + (int) c);

System.out.println("Valor en formato long: " + (long) c);

}

}

## Comentarios.

Cuando se escribe código en general es útil realizar comentarios explicativos. Los comentarios no tienen efecto como instrucciones para el ordenador, simplemente sirven para que cuando una persona lea el código pueda comprender mejor lo que lee. En Java existen dos formas de poner comentarios.

La primera es cuando la línea de comentario solo ocupa una línea de código. En este caso se pone dos barras inclinadas (//) antes del texto.

// Comentario de una línea

En el caso de comentarios de más de una línea se pone /\* para empezar y \*/ para finalizar.

El código nos quedará de la siguiente forma:

/\* Comentario

de varias

líneas \*/

# UT2 Literales en Java

## LITERAL JAVA

Un literal Java es un valor de tipo entero, real, lógico, carácter, cadena de caracteres o un valor nulo (null) que puede aparecer dentro de un programa.

Por ejemplo: 150, 12.4, “ANA”, null, ‘t’.

### LITERAL JAVA DE TIPO ENTERO

Puede expresarse en decimal (base 10), octal (base 8) y hexadecimal (base 16).

El signo + al principio es opcional y el signo – será obligatorio si el número es negativo.

El tipo de un literal entero es **int** a no ser que su valor absoluto sea mayor que el de un int o se especifique el sufijo l o L en cuyo caso será de tipo **long**.

#### Literal Java de tipo entero en Base decimal

Está formado por 1 o más dígitos del 0 al 9.

El primer dígito debe ser distinto de cero.

Por ejemplo:

1234 literal java entero de tipo int

1234L literal java entero de tipo long

123400000000 literal java entero de tipo long

#### Literal Java de tipo entero en Base octal

Está formado por 1 o más dígitos del 0 al 7.

El primer dígito debe ser cero.

Por ejemplo:

01234

025

#### Literal Java de tipo entero en Base hexadecimal

Está formado por 1 o más dígitos del 0 al 9 y letras de la A a la F (mayúsculas o minúsculas).

Debe comenzar por 0x ó 0X.

Por ejemplo:

0x1A2

0x430

0xf4

### LITERAL JAVA DE TIPO REAL

Son números en base 10, que deben llevar un parte entera, un punto decimal y una parte fraccionaria.  Si la parte entera es cero, puede omitirse.

El signo + al principio es opcional y el signo – será obligatorio si el número es negativo.

Por ejemplo:

12.2303

-3.44

+10.33

0.456

.96

También pueden representarse utilizando la notación científica. En este caso se utiliza una E (mayúscula o minúscula) seguida del exponente (positivo o negativo). El exponente está formado por dígitos del 0 al 9.

Por ejemplo, el número en notación científica 14\*10-3 se escribe: 14E-3

Más ejemplos de literal Java de tipo real:

2.15E2 -> 2.15 \* 102

.0007E4 -> 0.0007 \* 104

-50.445e-10 -> -50.445 \* 10-10

El tipo de estos literales es siempre **double**, a no ser que se añada el sufijo F ó f para indicar que es float.

Por ejemplo:

2.15 literal real de tipo double

2.15F literal real de tipo float

También se pueden utilizar los sufijos d ó D para indicar que el literal es de tipo double:

12.002d literal real de tipo double

### LITERAL JAVA DE TIPO CARÁCTER

Contiene un solo carácter encerrado entre comillas simples.

Es de tipo **char**.

Las secuencias de escape se consideran literales de tipo carácter.

Por ejemplo:

'a'

'4'

'\n'

'\u0061'

### LITERAL JAVA DE CADENAS DE CARACTERES

Está formado por 0 ó más caracteres encerrados entre comillas dobles.

Pueden incluir secuencias de escape.

Por ejemplo:

“Esto es una cadena de caracteres”

“Pulsa \”C\” para continuar”

“” -> cadena vacía

“T” -> cadena de un solo carácter, es diferente a ‘t’ que es un carácter

Las cadenas de caracteres en Java son objetos de tipo **String**.

*Nota: Sobre atajos de teclado en Eclipse:  
Ctrl+May+L nos visualiza en Eclipse todos los atajos de teclado*

# UT2 Ampliación Operadores

## Operadores lógicos

|  |  |  |  |
| --- | --- | --- | --- |
| **Operador** | **Descripción** | **Uso** | **Retorna cierto si** |
| && | and | op1 && op2 | op1 y op2 son true. Solo evalúa op2 si op1 es true |
| || | or | op1 || op2 | op1 o op2 es true. Solo evalúa op2 si op1 es false |
| ! | not | !op | op es false |

## Operadores lógicos a nivel de bits

|  |  |  |
| --- | --- | --- |
| **Operador** | **Descripción** | **Uso** |
| & | and | op1 & op2 |
| | | or | op1 | op2 |
| ^ | or exclusivo | op1 ^ op2 |
| ~ | complemento | ~op1 |

El operador **AND** **bit por bit (&)** realiza un and lógico entre cada par de bits paralelos que forman los operandos. Si ambos bits valen 1 el bit resultante vale 1, de lo contrario vale 0. Por ejemplo:

|  |
| --- |
| int bits\_orig = 6; --------// binario 00000000 00000000 00000000 00000**110** |
| int bits\_bandera = 5; -----// binario 00000000 00000000 00000000 00000**101** |
| int respuesta = (bits\_orig & bits\_bandera);--------- --// 000000 00000**100** |

El operador **OR bit por bit (|)** realiza un or lógico entre cada par de bits paralelos que forman los operandos. Si alguno de los bits vale 1 el bit resultante vale 1, de lo contrario vale 0. Por ejemplo:

|  |
| --- |
| int bits\_orig = 6; --------// binario 00000000 00000000 00000000 00000**110** |
| int bits\_bandera = 5; -----// binario 00000000 00000000 00000000 00000**101** |
| int respuesta = (bits\_orig | bits\_bandera);--------- --// 000000 00000**111** |

El operador **XOR bit por bit (^)** realiza un or exclusivo entre cada par de bits paralelos que forman los operandos. Si ambos bits tienen valores opuestos el bit resultante vale 1, de lo contrario vale 0. Por ejemplo:

|  |
| --- |
| int bits\_orig = 6; --------// binario 00000000 00000000 00000000 00000**110** |
| int bits\_bandera = 5; -----// binario 00000000 00000000 00000000 00000**101** |
| int respuesta = (bits\_orig ^ bits\_bandera);--------- --// 000000 00000**011** |

El operador **NOT bit por bit (~)** invierte los bits del operando. Por ejemplo:

|  |
| --- |
| byte valor = 6;-------------------// binario **00000110** |
| byte respuesta = (~ valor);---- --// binario **11111001** |

## Operadores de desplazamiento de bits

Existen muchas operaciones que requieren desplazar los valores de los bits que componen un número a la izquierda o a la derecha. Los operadores de desplazamiento de bits suelen utilizarse para llevar a cabo operaciones muy rápidas de multiplicación y de división de enteros. Un desplazamiento a la izquierda equivale a una multiplicación por 2 y un desplazamiento a la derecha a una división por 2.

Un desplazamiento no es una rotación. A medida que se desplazan los bits hacia un extremo se van rellenando con ceros por el otro extremo. Los bits que salen se pierden.

|  |  |  |
| --- | --- | --- |
| **Operador** | **Uso** | **Operación** |
| << | op1 << op2 | Desplaza a la izquierda los bits del primer operando op1 tantas veces como indica el segundo operando op2( por la derecha siempre entra un cero). |
| >> | op1 >> op2 | Desplaza a la derecha los bits del primer operando op1 tantas veces como indica el segundo operando op2( por la izquierda entra siempre el bit más significativo anterior). |
| >>> | op1 >>> op2 | Desplaza a la derecha los bits del primer operando op1 tantas veces como indica el segundo operando op2(sin signo- por la izquierda entra siempre un cero) |

A diferencia de C y C++, Java siempre conserva el bit de signo (el bit izquierdo) después de hacer un desplazamiento. Este tipo de desplazamiento se conoce como desplazamiento aritmético o desplazamiento de extensión de signo. Las siguientes instrucciones desplazan un valor, primero dos bits a la izquierda y luego dos bits a la derecha, con los operadores de desplazamiento de bits de Java:

|  |
| --- |
| int original = -3; ---------------- //- binario 10000000 00000000 00000000 00000011 |
| int izquierda = original << 2; ---- // izquierda vale -12 ---binario 10000000 00000000 00000000 00001100 |
| int derecha = izquierda >> 2;-----  // derecha vale -3 --- --binario 10000000 00000000 00000000 00000011 |
| int x = 7; ----------------------- // x vale 7--------------binario 00000000 00000000 00000000 00000111 |
| x = x << 1;----------------------- // x vale 14------------ binario 00000000 00000000 00000000 00001110 |

El **operador >>>** realiza un desplazamiento a la derecha similar al operador >>. La diferencia es que el operador >>> no conserva el bit de signo, sino que coloca el bit de signo en cero, a diferencia del operador >>, que lo pone en uno cuando es negativo, de forma que permanezca negativo. Normalmente se utiliza el operador >>> cuando el operador no representa una cantidad con signo, sino que es una máscara de bits de algún tipo en el que no importa el bit de signo. El siguiente ejemplo muestra el uso del operador >>> para desplazar el valor 256 dos bits a la derecha, dando como resultado 64:

|  |
| --- |
| int mascara = 256;------------------ // mascara 256-------------- binario 00000000 00000000 00000001 00000000 |
| int respuesta = mascara >>> 2;------ // respuesta 64------------- binario 00000000 00000000 00000000 01000000 |

Nota:

Java convierte a tipo int los tipo short y byte antes de realizar el desplazamiento de bits. Esto significa que el tipo byte tendrá 32 bits en lugar de 8. Después de ejecutar el desplazamiento trunca el dato para convertirlo al tipo original.

# UT2 Java Scanner para la lectura de datos

## Java Scanner para lectura de datos

La clase Scanner está disponible a partir de Java 5 y facilita la lectura de datos en los programas Java. Primero veremos varios ejemplos de lectura de datos en Java con Scanner y después explicaremos en detalle cómo funciona.

Para utilizar Scanner en un programa tendremos que hacer lo siguiente:

### 1. Escribir el import

La clase Scanner se encuentra en el paquete java.util por lo tanto se debe incluir al inicio del programa la instrucción:

import java.util.Scanner;  
 import java.util.\*;

### 2. Crear un objeto Scanner

Tenemos que crear un objeto de la clase Scanner asociado al dispositivo de entrada. Si el dispositivo de entrada es el teclado escribiremos:

Scanner sc = new Scanner(System.in);

*//(System.in representa la entrada de datos por defecto en mi programa)*

Se ha creado el objeto sc asociado al teclado representado por System.in, una vez hecho esto podemos leer datos por teclado.

#### Ejemplos de lectura:

Para leer podemos usar el método nextXxx() donde Xxx indica el tipo, por ejemplo nextInt() para leer un entero, nextDouble() para leer un double, etc.

*Ejemplo* de lectura por teclado de un número entero:

int n;

System.out.print("Introduzca un número entero: ");

n = sc.nextInt();

*Ejemplo* de lectura de un número de tipo double:

double x;

System.out.print("Introduzca número de tipo double: ");

x = sc.nextDouble();

*Ejemplo* de lectura de una cadena de caracteres (una línea hasta intro):

String s;

System.out.print("Introduzca texto: ");

s = sc.nextLine();

*Ejemplo* de programa Java con lectura de datos con Scanner:

El programa pide que se introduzca el nombre de la persona y lo muestra por pantalla. A continuación lee por teclado el radio de una circunferencia de tipo double y muestra su longitud. Además lee un entero y muestra su cuadrado.

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in); //crear un objeto Scanner

String nombre;

double radio;

int n;

System.out.print("Introduzca su nombre: ");

nombre = sc.nextLine(); //leer un String/linea

System.out.println("Hola " + nombre + "!!!");

System.out.print("Introduzca el radio de la circunferencia: ");

radio = sc.nextDouble(); //leer un double

System.out.println("Longitud de la circunferencia: " + 2\*Math.PI\*radio);

System.out.print("Introduzca un número entero: ");

n = sc.nextInt(); //leer un entero

System.out.println("El cuadrado es: " + Math.pow(n,2));

}

}

#### Funcionamiento de la clase Java Scanner.

De forma resumida podemos decir que cuando se introducen caracteres por teclado, el objeto Scanner toma toda la cadena introducida y la divide en elementos llamados tokens (los guarda en un buffer interno).

El carácter predeterminado que sirve de separador de tokens (*en el .nextLine()* ) es el espacio en blanco. Por ejemplo, si introducimos:

Esto es un ejemplo, lectura de datos.

Scanner divide la cadena en los siguientes tokens:

Esto

es

un

ejemplo,

lectura

de

datos.

Si introducimos la cadena:

12 20.001 Lucas w

Los tokens que se crean son:

12

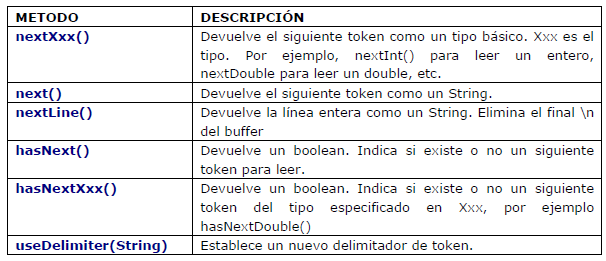
20.001

Lucas

W

A continuación, utilizando los métodos que proporciona la clase Scanner se puede acceder a esos tokens y trabajar con ellos en el programa.

Ya hemos visto el método nextXxx(). Además la clase Scanner proporciona otros métodos, algunos de los métodos más usados son:



#### Cómo limpiar el buffer de entrada en Java

Cuando en un programa se leen por teclado datos numéricos y datos de tipo carácter o String debemos tener en cuenta que al introducir los datos y pulsar intro estamos también introduciendo en el buffer de entrada el intro.

Es decir, cuando en un programa introducimos un dato y pulsamos el intro como final de entrada, el carácter intro también pasa al buffer de entrada.

Buffer de entrada si se introduce un 5: 5\n

En esta situación, la instrucción:

n = sc.nextInt();

Asigna a n el valor 5 pero el intro permanece en el buffer.

*Buffer de entrada* después de leer el entero: \n

Si ahora se pide que se introduzca por teclado una cadena de caracteres:

System.out.print("Introduzca su nombre: ");

nombre = sc.nextLine(); //leer un String

El método nextLine() extrae del buffer de entrada todos los caracteres hasta llegar a un intro y elimina el intro del buffer.

En este caso asigna una cadena vacía \n a la variable nombre y limpia el intro. Esto provoca que el programa no funcione correctamente, ya que no se detiene para que se introduzca el nombre.

***Solución***:

Se debe **limpiar el buffer** de entrada si se van a leer datos de tipo carácter a continuación de la lectura de datos numéricos.

La forma más sencilla de limpiar el buffer de entrada en Java es ejecutar la instrucción:

sc.nextLine();

Lo podemos comprobar si cambiamos el orden de lectura del ejemplo y leemos el nombre al final:

import java.util.Scanner;

public class Ejemplo {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String nombre;

double radio;

int n;

System.out.print("Introduzca el radio de la circunferencia: ");

radio = sc.nextDouble();

System.out.println("Longitud de la circunferencia: " + 2\*Math.PI\*radio);

System.out.print("Introduzca un número entero: ");

n = sc.nextInt();

System.out.println("El cuadrado es: " + Math.pow(n,2));

System.out.print("Introduzca su nombre: ");

nombre = sc.nextLine(); //leemos el String después del double

System.out.println("Hola " + nombre + "!!!");

}

}

Si lo ejecutamos obtendremos:

*Introduzca el radio de la circunferencia: 34*

*Longitud de la circunferencia: 213.62830044410595*

*Introduzca un número entero: 3*

*El cuadrado es: 9.0*

*Introduzca su nombre: Hola !!!*

Comprobamos que no se detiene para pedir el nombre.

Una solución es escribir la instrucción

sc.nextLine();

Después de la lectura del int y antes de leer el String:

n = sc.nextInt();

System.out.println("El cuadrado es: " + Math.pow(n,2));

sc.nextLine();

System.out.print("Introduzca su nombre: ");

nombre = sc.nextLine();

System.out.println("Hola " + nombre + "!!!");

Ahora la ejecución es correcta:

*Introduzca el radio de la circunferencia: 23*

*Longitud de la circunferencia: 144.51326206513048*

*Introduzca un número entero: 5*

*El cuadrado es: 25.0*

*Introduzca su nombre: Lucas*

*Hola Lucas!!!*

Hay una solución más elegante: leer siempre líneas con sc.nextLine(); y realizar la conversión al tipo correspondiente después:

import java.util.Scanner;

public class Scanner4 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String nombre, radioS, nS;

double radio;

int n;

System.out.print("Introduzca el radio de la circunferencia: ");

radioS = sc.nextLine();

radio = Double.parseDouble(radioS);

//https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Double.html#parseDouble(java.lang.String)

System.out.println("Longitud de la circunferencia: " + 2\*Math.PI\*radio);

System.out.print("Introduzca un numero entero: ");

nS = sc.nextLine();

n = Integer.parseInt(nS);

System.out.println("El cuadrado es: " + Math.pow(n,2));

System.out.print("Introduzca su nombre: ");

nombre = sc.nextLine(); //leemos el String despues del double

System.out.println("Hola " + nombre + "!!!");

sc.close();

}

}

# UT2 Ejercicios de expresiones

**Ejemplos:**

|  |  |
| --- | --- |
| **Expresión algebraica** | **Expresión aritmética algorítmica** |
| [https://1.bp.blogspot.com/-Yprv285kbDM/VQDqr2B40EI/AAAAAAAAMIM/adExySKON-w/s1600/eq.png](http://1.bp.blogspot.com/-Yprv285kbDM/VQDqr2B40EI/AAAAAAAAMIM/adExySKON-w/s1600/eq.png) | x\*\*2 + y\*\*4 |
| [https://2.bp.blogspot.com/--Ib7mwPHA0c/VQDm5rltj8I/AAAAAAAAMIA/zSfRAD7yhZI/s1600/eq1.png](http://2.bp.blogspot.com/--Ib7mwPHA0c/VQDm5rltj8I/AAAAAAAAMIA/zSfRAD7yhZI/s1600/eq1.png) | x\*\*2 / (a\*\*3 + b\*\*3) |
| [https://4.bp.blogspot.com/-cf7MX76AhZI/VQDrLYDLD1I/AAAAAAAAMIU/jP69Uhnd89g/s1600/eq2.png](http://4.bp.blogspot.com/-cf7MX76AhZI/VQDrLYDLD1I/AAAAAAAAMIU/jP69Uhnd89g/s1600/eq2.png) | u + x\*\*2 / y |
| [https://1.bp.blogspot.com/-605-e1GP8xs/VQDsptXNVuI/AAAAAAAAMIo/4-JbjFtpFWs/s1600/eq3.png](http://1.bp.blogspot.com/-605-e1GP8xs/VQDsptXNVuI/AAAAAAAAMIo/4-JbjFtpFWs/s1600/eq3.png) | (a + b) / (a + c\*\*2 / (d + e)) |

**Deducir el valor de las expresiones siguientes: Siendo:   
A = 5; B = 25; C = 10**

1. A + B / C 7,5

2. (A + B) / C 3

3. A + B % C 10

**Si el valor de A es 4, el valor de B es 5 y el valor de C es 1, evaluar las siguientes expresiones:**

1. B \* A - B \* B / 4 \* C

20-25/4\*1 13,75

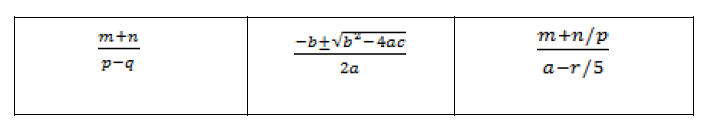
2. (A \* B) / 3 \*3

20/3\*3 20

3. ( ( ( B + C ) / 2 \* A + 10 ) \* 3 \* B ) - 6

((6/2\*4+10)\*3\*5)-6 22\*15-6=324

**Realizar las conversiones de expresiones matemáticas a expresiones algorítmicas indicando el orden de ejecución de cada una de ellas**



(m+n)/(p-q)

(-b+(Math.pow(Math.pow(b, 2) - 4\*a\*c, 1/2)))/(2\*a)

(-b-(Math.pow(Math.pow(b, 2) - 4\*a\*c, 1/2)))/(2\*a)

((m+n)/p)/((a-r)/5)

**Evaluar las expresiones lógicas aplicando la jerarquía de operadores.**

|  |  |
| --- | --- |
| 1. ((A \* B) < (B + C)) && (A= = C)  (12<6) && (3==2) FALSE | A=3, B=4 y C=2 |
| 2. ((A + B) > C) || ((B / D > B))  (7 > 3) || (5/5>5) T || (1>5) T | A=2, B=5, C=3 y D=5 |
| 3. (A/B) \* C + (A / B)  2\*3 + (2) 8 | A = 4, B = 2, C = 3 |
| 4. PI \* X\*X>Y || 2\* PI \* X <=Z  3.141592\*1\*1 > 4 ||  2\*3.141592\*1 <= 10  F | | T T  5. X>3 && Y==4 || X+Y<=Z  F && T || 5 <= 10 F || T T  6. X>3 && (Y==4 || X+Y<=Z)  F && (F || 5<=10) F && (F) F  7. !( Y/2==2\*X) && !( Y<PI-E\*Z)  !(4/2==2) && !(4<3.141592-27.18281)  !( T ) && !( F ) F && T F | X=1,  Y=4,  Z=10,  PI=3.141592  E=2.718281 |
| 8. A==B%C  5 == 4%3 5 == 1 F  9. 6/C < C % 6  6/3 < C % 6 2 < 3 T  10. C + B – 1 != A || B >= -B \* A && A \* A<=10  3 + 4 – 1 != 5 || 4 >= -4 \*5 && 5 \*5 <=10  6 != 5 || 4 >= -20 && 25 <= 10  T || T && F T || F T  11. B % A / C  4 % 5 / 3 4/3  12. !(X \*A > Y/B)  !(0.05 \* 5 > 2.3/4) !(0.25 > 0.575) !(F) T | A=5,  B=4,  C=3,  X=0.05,  Y=2.3 |

**Convertir en expresiones numéricas los siguientes enunciados.**

1. Elabore una expresión que sólo permita valores entre 1 y 10.

(x>=1 && x<=10)

2. Elabore una expresión que permita valores entre 1 y 3, y entre 5 y 7 exclusivamente.

(x>=1 && x<=3) || (x>=5 && x<=7)

3. Elabore una expresión que permita edades entre 18 y 25 años.

(edad>=18 && x<=25)

**Comprobar qué resultado se obtiene en estas 2 expresiones. ¿Son el mismo resultado? Justificar la respuesta**

1. 7==4+3 || 6<2 && 5>=8 T || F && F T || F T

2. (7==4+3 || 6<2) && 5>=8 (T || F) && F T && F F