Programación en Java 101

ÍNDICE:

1.) Estructura de bloques:

```
public class HolaMundo {
public static void main (String[]args) {
        System.out.println("Hola mundo");
        }
}
```

Cada llave de apertura y cierre { } pertenece a un bloque concreto.
Los bloques son el inicio y el fin de cada estructura.

2.) Método main:

TODOS los programas empiezan por el método main().
Debe ser público, estático, y mantener los parámetros de entrada. Y el nombre debe coincidir con el nombre del archivo en el que se ha escrito.

public static void main (String [] args) {

}

-Es público, se puede llamar desde cualquier punto
-Es estático, se llama sin instanciar la clase
-No devuelve valor (void)
-Admite parámetros (String [] args)

3.) Comentarios:

// una sola línea

/* . . . */ una o más líneas

/** . . . */ se usa en la herramienta javadoc. Para documentación.

Sintaxis básica 01:

-Cada sentencia se define con un ; (semicolon).
-Un bloque es un conjunto de sentencias agrupadas entre llaves { }
-Se pueden usar espacios para entender mejor el código. Java los ignora.
-Variable = Un elemento que PUEDE SER MODIFICADO. Casi siempre usamos variables para almacenar valores.

4.) Tipos de Datos:

-Delimitan qué tipo de valores pueden tener y qué tipo de operaciones se pueden hacer.
Java es de tipado estático => Cada variable debe ser declarada ANTES de usarla.

4.1) Datos primitivos:

BYTE > 8 bits (-128..+127) // byte a;

SHORT > 16 bits // short b, c = 3;

INT > 32 bits // int d = -30;

LONG > 64 bits // long b = 434123; long b = 5L <—L = Long

CHAR > carácter simple UNICODE // char car1 = ‘c’; char car2 = ’99’; <- esto es una c
FLOAT > 32 bits (7 dígitos decimales) // float pi = 3.1416F; <— F = Float

DOUBLE > 64 bits (16 dígitos decimales) // double millon = 1e6D; <— D = Double. 1e6 = 1x10^6 (1.000.000)

BOOLEAN > 1 bit (True/False) // esNumero = true;

4.2) Tipos enumerados:

Se usa enum y representan un conjunto de valores. Ej. Los días de la semana.
public enum Dias {LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO}

Para acceder a un enum usamos variable.valor:

Dias diaActual = Dias.MARTES;
Dias diaSiguiente = Dias.MIERCOLES;

4.3) Literales:

Los literales son aquellos valores que podemos asignar, son los tipos primitivos, los strings (cadena de caracteres) y la expresión “null” (Valor nulo o desconocido).

Todo lo que hemos visto son literales: ‘a’ , 322 , 3.14 , “ANA” , null
Los literales pueden expresarse en decimal, octal, hexadecimal y binario.
Decimal = 21 // Octal = 025 // Hexadecimal = 0x15 // Binario = 0b1001001
El valor por defecto será de tipo INT a menos que se especifique => 123L (tipo Long)
El valor por defecto de un decimal será DOUBLE a menos que se especifique => 2.15F
Se pueden añadir _ entre dígitos para leerlo mejor: 0b100_001_111

4.4) Números Reales:

En base 10, con parte entera un punto (.) y una parte fraccionaria.
Si la parte entera es 0 puede omitirse.
Se usará una E (e) seguida del exponente, ya sea positivo o negativo.
Ejemplos: 1e2 = 1x10^2 56.34E-45 = 56.34 x 10^-45

4.5) Cadenas de Caracteres (String):

Se consideran objetos. Pero no necesitan el operador “new” para crearlos.
Deben ir entre “ ” comillas dobles.
Pueden incluir secuencias de escape.
Ejemplos: “Pulsa \”C\” para continuar” // “” (vacía) // “T” (Un sólo carácter)

Secuencias de escape:
\b > retrocede un espacio
\t > tabulación
\n > nueva línea
\f > salto de página
\r > retorno de carro
\” > comilla doble
\’ > comilla simple
\\ > barra invertida

5.) Constantes:

Su valor NO SE PUEDE MODIFICAR.
Se declaran e instancian en la misma línea:

final static <tipo_dato> <nombre> = <valor>

-final indica que es constante
-static indica que sólo existe una copia
-tipo_dato será el tipo (int, double…)
-nombre el identificador
-valor es el valor que NO SE MODIFICARÁ

Ej. **final static double PI = 3.141592**;

A LAS CONSTANTES LAS NOMBRAREMOS EN MAYÚSCULAS

6.) Variables:

Todas aquellas zonas de memoria donde se almacena información.
Se pueden modificar.
Existen las de tipo primitivo y las referencias a objetos.
Deben llevar un tipo, un identificador y un dato o valor.

Podemos declararlas vacías o con valor en una línea o en varias.
int num1;
int num1 = 5;
int num2 = 10;

6.1) Ambito de variables:

La zona del código donde se puede trabajar con esa variable.
Atributos: Solo asociadas a la clase a la que pertenecen
Parametros de método: Solamente en el método del que son parámetros
Variables Locales: deben inicializarse, sólo se puede usar en el método dentro del que han sido declaradas. (Ej. Las variables dentro del método main sólo podrán ser usadas en él)
Variables de bloque: Sólo en el bloque donde han sido declaradas. Ej. Un bloque en la aplicación “calculadora” que usa variables auxiliares para calcular el resto de una división y almacenarlo.

Las variables miembro (atributos) se inicializan a un valor por defecto (Cero o null)
Las locales no se inicializan por defecto, y deberemos hacerlo manualmente.

A la hora de identificar elementos es mejor seguir un estilo concreto:
Paquetes: **lowercase** (ej. package ejerciciosjava;)
Clases e interfaces: **UpperCamelCase** (ej. EjerciciosJava)
Variables y métodos: **lowerCamelCase** (ej. esNumeroPar)
Constantes: **SCREAMING_SNAKE_CASE** (ej. DIAS_DEL_MES)

7.) Operadores

Realizan operaciones. Una expresión es una combinación de operadores y operandos.
Su resultado se puede usar como parte de otra expresión.

7.1) Aritméticos:

\+ => suma **(op1 + op2)**
\- => resta **(op1 - op2)**
* => multiplica **(op1 * op2)**
/ => divide **(op1 / op2)**
% => calcula el resto **(op1 % op2)**

Cuando al menos 1 de los operandos es de un tipo, los operandos tomarán el tipo más grande.
Int > long
Float > double
Int > double
También existen operadores uranios como -op1 que cambia el signo.

7.2) Relacionales:

> => mayor que
< => menor que
== => igual que
!= => distinto que
>= => mayor o igual que
<= => menor o igual que
Todos estos devuelven un boolean como resultado.

7.3) Asignación:

= => guarda el valor de un operando en el primero
+= => guarda la suma de un operando en el primero
-= => guarda la resta de un operando en el primero
***=** => guarda la multiplicación de un operando en el primero
/=, %=, &=, |=, ^=, <<=, >>=, >>>= => lo mismo para estos operandos

7.4) Condicional: op ? op : op;
Es como un if simplón.
Necesita tres operandos, primero una expresión, y luego los valores de true y false.
(x > y)? True : false; (¿Es X mayor que Y? Sí o No)

7.5) Evaluar expresiones:

Usaremos !, &, |, ^, && y ||
! (!op) > Distinto de. Devuelve true si el operando es false.
& (op1 & op2) > AND (Y) Devuelve true si AMBOS operandos son true.
| (op1 | op2) > OR (O) Devuelve true si op1 O op2 son true.
^ (op1 ^ op2) > XOR (O estricto) Devuelve true si SOLO UNO de los operandos es true.
&& (op1 && op2) > AND (Y) Igual que & pero se detiene cuando op1 es false.
|| (op1 || op2) > OR (O) Igual que | pero se detiene cuando op1 es true.

8.) Cadenas de caracteres

Se trabaja llamando al String por su nombre, seguido de un punto, y la operación.
(Ej. mensaje.charAt(0))

8.1) Creación:

No es necesario crearla con “new”. Bastará con introducirlo entre comillas dobles “ ”

8.2) Obtención de un caracter:

Usaremos el método **charAt(POSICION)** (EMPIEZA DESDE LA POSICIÓN CERO)
Ej. mensaje.charAt(3); > Devolverá el carácter en la posición 3 del mensaje.

8.3) Longitud:

Usaremos el método **length**.
Ej. mensaje.length( ); > Nos devolverá un número del número total de caracteres.

8.4) Concatenación:

Juntará dos cadenas de caracteres. Usa el “ **+** ” o el método **concat( )**.
Ej. “Hola ” + “Qué tal” > > “Hola Qué tal”

8.5) Comparación:

Con el método **equals** nos devuelve un booleano según si las dos cadenas son o no iguales.
El método **equalsIgnoreCase** ignora las mayúsculas.
*NO SE DEBE USAR == *

8.6) Subcadenas:
Nos permite coger fragmentos de cadenas usando el método **substring**, que debemos indicarle el inicio y el final.

8.7) Cambio mayúsculas / minúsculas:
Los métodos t**oUpperCase** y **toLowerCase** devuelven una nueva variable con la cadena transformada en mayúsculas y minúsculas, respectivamente.

8.8) Conversiones:
Usamos el método **valueOf** para convertir de un dato primitivo a una variable de tipo String.
Precedencia:
Es de vital importancia la precedencia. Se le llama asociatividad a si empieza por la izquierda o la derecha.

x++ y x- -	> 1º / Derecha
++x, - -x y !	> 2º / Derecha
* / y %	> 3º / Izquierda
+ -	>4º / Izquierda
Relacionales (<, >= etc.)	>6º / Izquierda
Relacionales (== y !=)	>7º / izquierda
&	>8º / Izquierda
^	>9º / Izquierda
|	>10º / Izquierda
&&	>11º / Izquierda
||	>12º / Izquierda
?: (Condicional)	>13º / Derecha
Asignación (=, +=, -=, etc.)	>14º / Derecha

9.) Casting (Conversión de tipos):

Pasar de un tipo a otro de variables declaradas.
Se le llama conversión, cast, casting, moldeado o tipado.
Se puede forzar pero **PUEDE PERDERSE INFORMACIÓN.**
Las conversiones seguras son de un tipo pequeño a uno grande. (Int a Long)
Las conversiones se resuelven en tiempo de compilación.
Un tipo pequeño se convierte en otro más grande => **UPCASTING.**
Las conversiones forzosas se llaman EXPLÍCITAS o **DOWNCASTING.**

```
Ej.   Int dato1 = 5;
      byte dato2;
      dato2 = (byte) dato1;        => El tipo se especifica entre paréntesis.
```

Se puede convertir de **manera automática siempre y cuando sean conversiones seguras** del **mismo tipo**. Se llama "**promoción**" de valor.
INT => LONG.
FLOAT => DOUBLE.