

Estructura básica de datos:

1. Salida de datos por pantalla (**Escritura**).
2. Entrada de datos por el usuario (**Lectura**).
3. Operaciones matemáticas.
4. Declaración y uso de variables.
5. Tipos de datos elementales.

1) SALIDA BÁSICA POR PANTALLA:

- a. **System.out.println** → salto de línea
- b. **System.out.print**
- c. **System.out.printf** → Marcas de formato
 - i. **%d** para datos enteros
 1. **%5d** o **%05d** mostrar al menos 5 cifras, se rellena con espacios a la izquierda o ceros a la izquierda.
 - ii. **%f** para datos reales
 1. **%5.2f** mostrar 5 elementos (entero, punto coma y decimal) de los cuales 2 son decimales. Espacios hacia la izquierda, ceros a la derecha.
 - iii. **%s** para cadenas de texto
 - iv. **%c** para caracteres

LAS MARCAS DE FORMATO: EJEMPLO

```
int numero = 3;
String texto = "Hola";

System.out.print("El número es " + numero + " y el texto es " + texto);

System.out.printf("El número es %d y el texto es %s", numero, texto);
```

Se pueden enlazar (**concatenar**) con el operador **+**.

2) ENTRADA DE DATOS POR EL USUARIO:

- a. **Scanner**: permite procesar tipo de datos simples de una fuente de entrada.
- b. **System.in**.
- c. Incluir **import** .
- d. Instrucciones:
 - i. **nextLine()** leer texto (hasta que se pulse intro).
 - ii. **nextInt()** leer un entero.
 - iii. **nextFloat()** leer un float.
 - iv. **nextBoolean()** leer un boolean.
- e. **System.console().readLine()**

3) OPERADORES: Son elementos que permiten realizar una operación determinada en un programa.

- a. **Aritméticos:** Operaciones aritméticas.
 - i. **Suma (+) , Resta (-) , Multiplicación (*) , División (/) y Resto de división (%)**
 - ii. Precedencia:
 - 1. *, / , % de izquierda a derecha
 - 2. + , - de izquierda a derecha
 - 3. El uso de paréntesis permite variar precedencia.
- b. **Asignación:** asignar valor o modificarlo.
 - i. **Asignación simple (=) , Autosuma (+=) , Autorresta (-=) , Automultiplicación (*=) , Autodivisión (/=), Autorresto (%=)**
- c. **Relacionales:**
 - i. **Mayor que (>) , Mayor o igual que (>=) , menor que (<) , menor o igual que (<=), igual que (==) , Distinto de (!=)**
 - ii. **.equals()**
- d. **Lógicos:**
 - i. **Y (&&) , O (||) , No (!) (TABLAS DE VERDAD)**
 - ii. Precedencia:
 - 1. Paréntesis
 - 2. Negaciones
 - 3. &&
 - 4. ||

***Comprobación cortocircuito:** Si la primera comprobación es falsa (&&) la comprobación es falsa, en un O si la primera comprobación es verdadera (||) la comprobación es verdadera

4) DECLARACIÓN DE VARIABLES: Espacio de memoria donde podemos almacenar un valor.

- a. **tipoDato nombreVariable = valor;**

5) TIPOS DE DATOS:

- a. Tipos de datos enteros
 - i. **byte (1) , short (2) , int (4) , long(8)**
- b. Tipos de datos reales:
 - i. **float (6-7 cifras) , double (15 cifras)**

Desbordamiento: hacer que salga más allá de los límites el valor de un dato. Esto tiene como consecuencia → Valores no esperados / errores de programa

c. Tipo de dato Carácter:

- i. **Char**: Permite representar cualquier carácter o símbolos sueltos (letras, dígitos, puntuación, tabulaciones, saltos de línea..)
- ii. Se representan entre ' '.
- iii. Secuencias de escape: '\n' → Salto de línea| '\t' → tabulación| '\"' → mostrar comillas dobles por pantalla| '\"' → comillas simple por pantalla| '\\' → barra invertida por pantalla

d. Tipo de datos Cadena de caracteres:

- i. **String**: representa una secuencia de caracteres.
- ii. Se representa entre " ".
- iii. Secuencias de escape como en char.

```
/*
*Autoincremento: n++, ++n
*Autodecremento: n--, --n
*/
```

PRE Y POST EN EXPRESIONES

```
int i = 3, j = 2, k = 1;

i = j++ * k;
// Primero i = 2 * 1 = 2
// Después j = j + 1 = 3
```

```
int i = 3, j = 2, k = 1;

i = ++j * k;
// Primero j = j + 1 = 3
// Después i = 3 * 1 = 3
```

CONVERSIÓN BÁSICA CON TYPECAST

- Cuando Java sabe cómo convertir un tipo de dato en otro, podemos forzar a que lo haga poniendo el tipo de dato entre paréntesis, delante de la variable o expresión que queramos convertir
- Aplicable para convertir entre tipos enteros, reales y caracteres. Las cadenas de texto NO entran en esta dinámica

```
int numero = 3;
float numero2 = 3.14159f;
int numero3 = (int)numero2; // numero3 = 3
float numero4 = (float)numero; // numero4 = 3.0
```

CONVERSIÓN DE CADENAS DE TEXTO

- La instrucción `Integer.parseInt(texto)` permite convertir un texto a número entero (`int`)
- La instrucción `Float.parseFloat(texto)` permite convertir un texto a número real (`float`)
- Existen otras instrucciones similares para convertir al resto de tipos numéricos: `Double.parseDouble(texto)`, `Byte.parseByte(texto)`, `Short.parseShort(texto)` y `Long.parseLong(texto)`

```
int numero = Integer.parseInt("21");      // numero = 21
float numero2 = Float.parseFloat("3.14"); // numero2 = 3.14f
```

```
int nota1, nota2;
String textoNota1, textoNota2;
System.out.println("Escribe una nota:");
textoNota1 = System.console().readLine();
nota1 = Integer.parseInt(textoNota1);
System.out.println("Escribe otra nota:");
textoNota2 = System.console().readLine();
nota2 = Integer.parseInt(textoNota2);
int suma = nota1 + nota2;
float media = (float)suma / 2;
System.out.println("La media es " + media);
```

CONVERSIÓN A CADENAS DE TEXTO

- La instrucción `String.valueOf(...)` permite convertir a texto el valor que indiquemos entre paréntesis
- También se puede convertir a texto un tipo simple concatenándolo con una cadena vacía

```
String texto = String.valueOf(23); // texto = "23"
String texto2 = "" + 23;           // texto = "23"
```

Programación estructurada:

1. **Condiciones:** Nos permite elegir entre varios caminos o alternativas posibles, en función de una condición o condiciones que deberán cumplirse para uno u otro.
2. **Bucles:** Permitirán repetir un conjunto de instrucciones un número determinada de veces, o mientras se cumpla una determinada condición.

1) TIPO DE DATO BOOLEANO:

- a. Solo puede tomar dos valores: **True** o **False**.
- b. Se representan por la palabra **boolean**.

2) ESTRUCTURA IF:

- a. La estructura evalúa una expresión booleana entre paréntesis y ejecuta una instrucción o conjunto de instrucciones si dicha expresión es cierta.
- b. **if-else** – Si la condición evaluada no es cierta y no se ejecuta la instrucción if podemos usar **else** para ejecutar la segunda instrucción.
- c. **Else – if** - ejecución de varios caminos.

```
int numero = ...;

if (numero > 5)
{
    // Instrucciones si es cierto
}
else if (numero < 0)
{
    // Instrucciones si es falso lo
    // anterior pero es cierto esto
}
else
{
    // Instrucciones si todo lo anterior
    // es falso
}
```

3) ESTRUCTURA SWITCH:

- a. Permite evaluar el valor de una variable o expresión y ejecutar un conjunto de instrucciones u otro en función de dicho valor.
- b. Se coloca la variable o expresión a evaluar dentro del paréntesis de la partícula **Switch** y se define un bloque **case** por los posibles valores que contemplamos.
- c. Hay que poner **break** sino al ejecutar un **case** se ejecutarán el resto de los siguientes. Es decir, para salir del bloque **switch** en general.
- d. **Default** para dar respuesta en el caso de que ninguno de los casos de la expresión coincida con los examinados.

```

int numeroDia = ...;

switch(numeroDia)
{
    case 1:
        System.out.println("LUNES");
    case 2:
        System.out.println("MARTES");
    case 3:
        System.out.println("MIÉRCOLES");
    case 4:
        System.out.println("JUEVES");
    case 5:
        System.out.println("VIERNES");
    case 6:
        System.out.println("SÁBADO");
    case 7:
        System.out.println("DOMINGO");
}

```

4) OPERADOR TERNARIO:

SINTAXIS Y EJEMPLO

```

int numero = ...;
String mensaje;

if (numero >= 0)
{
    mensaje = "Es positivo";
}
else
{
    mensaje = "Es negativo";
}

System.out.println(mensaje);

```

condicion ? valor_si_true : valor_si_false

`mensaje = (numero >= 0) ? "Es positivo" : "Es negativo";`

5) ESTRUCTURA REPETITIVA WHILE:

- a. Permite ejecutar una instrucción o instrucciones que engloba mientras se cumpla la condición expresada en paréntesis.

while(condicion)

```

{
    // Instrucciones...
}

```

EJEMPLO

Contar de 1 a 10

```

int n = 1;
while(n <= 10)
{
    System.out.println(n);
    n++;
}

```

6) ESTRUCTURA REPETITIVA DO-WHILE:

- a. Permite ejecutar la instrucción o instrucciones que engloba mientras se cumpla la condición expresada en paréntesis.
- b. Diferencia con **While** la condición se evalúa después de ejecutar al menos una vez el conjunto de instrucciones. Lo que permite la ejecución previa a la evaluación.

```
do
{
    // Instrucciones...
} while (condicion);
```

EJEMPLO

• Contar de 1 a 10

```
int n = 1;
do
{
    System.out.println(n);
    n++;
} while (n <= 10);
```

7) ESTRUCTURA REPETITIVA FOR:

- a. Permite ejecutar la instrucción o instrucciones que engloba un número determinado de veces, mientras se cumpla la condición expresada entre paréntesis.
- b. Se diferencia de la estructura `while` o `do-while` en que la variable que se incrementa y hace de contador de repeticiones se integra y modifica en la propia estructura `for`.

```
for (valor_inicial; condicion; incremento)
{
    // Instrucciones...
}
```

EJEMPLO

• Contar de 1 a 10

```
int n = 1;
while (n <= 10)
{
    System.out.println(n);
    n++;
}
```

```
for (int n = 1; n <= 10; n++)
{
    System.out.println(n);
}
```

8) ESTRUCTURA REPETITIVA BREAK Y CONTINUE:

- a. **BREAK:** Permite salir de la estructura repetitiva donde se esté de forma inmediata
 - i. Se suele hacer cuando dentro de una repetición “normal” se da una condición excepcional que obligue a interrumpir las iteraciones.
 - ii. No recomendable.
- b. **CONTINUE:** fuerza una repetición más de la estructura iterativa donde se esté, de forma inmediata.
 - i. Suele hacerse cuando dentro de una repetición “normal”, se da una condición excepcional que obligue a dejar de ejecutar la instrucción restante y pasar a la siguiente repetición del bucle.
 - ii. No recomendable.

EJEMPLO

```
int n = 1;
while (n <= 10)
{
    System.out.println(n);
    if (n == 5)
        break;
    n++;
}
```

EJEMPLO

```
int n = 1;
while (n <= 10)
{
    System.out.println(n);
    if (n == 5)
        continue;
    n++;
}
```

```
for (int n = 1; n <= 10; n++)
{
    if (n == 5)
        continue;
    System.out.println(n);
}
```

9) RECOMENDACIONES DE USO:

- a. **While:**
 - i. Numero de repeticiones indefinidas.
 - ii. Ni siquiera se sabe si habrá una primera repetición.
 - iii. Ej: *Programa que, dado un número, determine cuantas veces seguidas es divisible por dos.*
- b. **Do..While:**
 - i. Numero de repeticiones indefinidas.
 - ii. Al menos se sabe que habrá una repetición.
 - iii. Ej: *Programa que le pida al usuario números hasta que este escriba un 0.*
- c. **While:**
 - i. Numero de repeticiones conocido e invariable antes de empezar el bucle
 - ii. Ej: *Programa que cuente del 1 al 10.*

10) BUCLES ANIDADOS:

- a. Se pueden incluir bucles dentro de bucles, tantos niveles como se necesiten.
- b. Esto permite repetir el bucle tantas veces como indique el exterior.