

Programación

# Tema 1

Programación Estructurada y Modular

# Tema 1

- 
1. Tipos y operadores
  2. Constantes y Variables
  3. Algoritmos
  4. Programación estructurada
  5. Programación modular

# 1.- Tipos y operadores

- Los tipos de datos nos sirven para clasificar nuestra información de entrada y salida y trabajar con ellos.
- Un tipo lo define en base al valor que puede almacenar y las operaciones que se pueden realizar con él.
- **Tipos simples:**
  - Se llama tipo de dato a una clase concreta de objetos o valores:
  - Números enteros : int, long , short , byte
  - Números reales float , double
  - Caracter char
  - Logicos : boolean
  - **String** : tipos especial de datos
- **Tipos compuestos:**
  - arrays, listas, etc, que los veremos más adelante.

# 1.- Tipos y operadores

```
public static void main(String[] args) {  
    // Tipos de datos  
    // Tipos simples  
  
    // Numeros  
    int entero = 1;  
    float realCorto = (float) 1.0;  
    double realLargo = 1.0f;  
    long enteroLargo = 1L;  
    short enteroCorto = 1;  
    byte tipoByte=1;  
  
    // logicos  
    boolean variableTrue = true;  
    boolean variableFalso = false;  
  
    // Caracteres  
    char caracter ='c';  
  
    // Cadenas  
    String cadena ="hola mundo";  
}
```


# 1.- Tipos y operadores

## Lenguajes tipados y no tipados

- Lenguajes tipados:
  - Los lenguajes tipados son aquellos que requieren que el programador declare el tipo de dato que se va a utilizar en cada variable.
  - En estos lenguajes, el compilador o intérprete comprueba que el tipo de dato de cada variable coincida con el tipo de dato que se ha declarado.
  - Ejemplo Java, C
- Lenguajes débilmente tipados y/o tipado dinámico:
  - Son aquellos que no requieren que el programador declare el tipo de dato que se va a utilizar en cada variable.
  - El compilador o intérprete deduce el tipo de dato de cada variable en tiempo de ejecución.
  - Ejemplo : Python

## 2.- Operadores y variables

### Operaciones

- Aritméticas: +, -, \*, /, %, ++, --
- Relacionales: ==, !=, >, <, >=, <= 
- Lógicas: &&, ||, !
- Asignación: =, +=, -=

### Variables

- Nombres que asignamos a los datos en memoria que vamos a utilizar

### 3.- Algoritmo

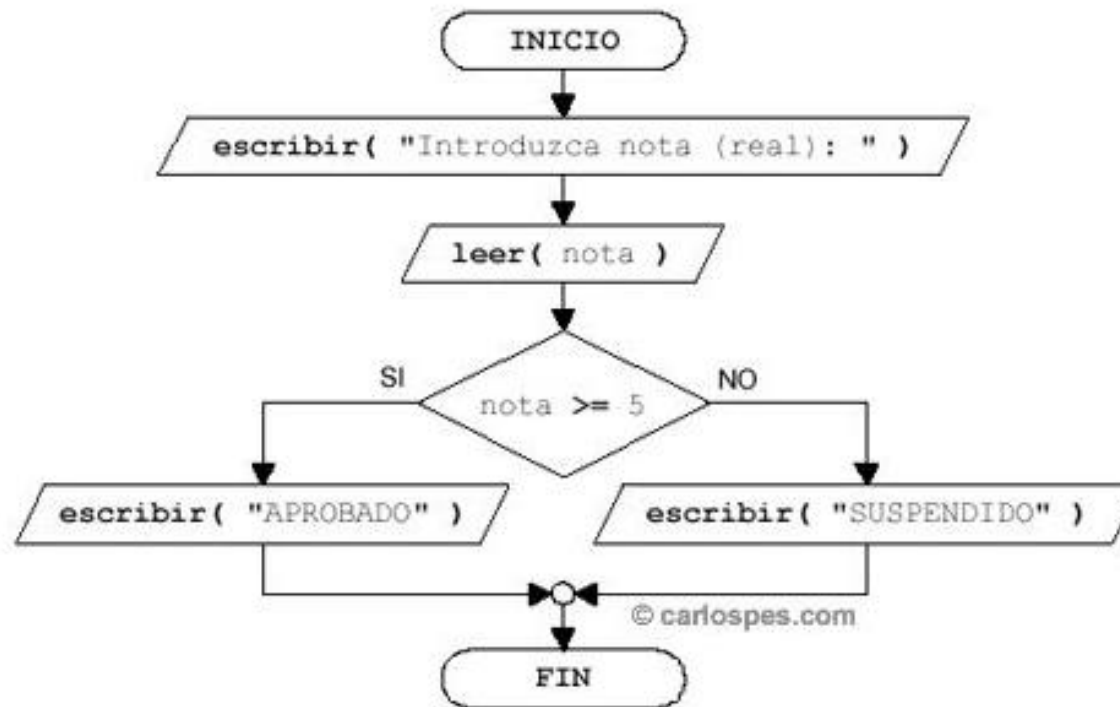
- Un algoritmo es una secuencia ordenada de pasos que conducen a la solución de un problema. Tienen tres características:
  - Son precisos en el orden de realización de los pasos.
  - Están bien definidos de forma que usando un algoritmo varias veces con los mismos datos, dé la misma solución.
  - Son finitos, deben acabarse en algún momento.
- Los algoritmos deben representarse de forma independiente del lenguaje de programación que luego usaremos.
- Usaremos ordinogramas o diagramas de flujo para representarlos y pseudocódigo

### 3.- Algoritmo

SÍMBOLO	NOMBRE	FUNCIÓN
	Inicio / Fin	Es el inicio y el final de un proceso
	Línea de flujo	Es el orden que llevan las actividades u operaciones
	Entrada / Salida	Son las lectura de los datos de la entrada y la impresión de datos en la salida
	Proceso	Representa las operaciones de cualquier tipo
	Decisión	Se analiza una situación con verdadero o falso



### 3.- Algoritmo



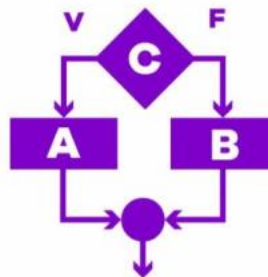
## 4.-Programación Estructurada

- La programación estructurada es un paradigma de programación que se basa en la estructuración de los programas en tres elementos básicos:
  - **Secuencia:** Los programas se ejecutan secuencialmente, es decir, una instrucción tras otra.
  - **Condicionales:** Los programas pueden ejecutar una parte de código u otra dependiendo de una condición.
  - **Bucles:** Los programas pueden ejecutar una parte de código varias veces

Secuencia



Selección o condicional



Iteración (ciclo o bucle)



## 4.-Programación Estructurada

### Secuencias

- Las secuencias son la base de la programación estructurada. Se ejecutan una tras otra, de arriba a abajo.

```
// Secuencia
System.out.println("Hola");
System.out.println("Introduce tu nombre:");

// leemos el nombre
Scanner teclado = new Scanner(System.in);
String nombre = teclado.next();
System.out.println("Encantado de conocerte "+nombre);
```

## 4.-Programación Estructurada

### Condicionales

Los condicionales nos permiten ejecutar una parte de código u otra dependiendo de una condición.

Pueden ser:

- Condicionales simples:
- Condicionales Múltiples
- Multicondicionales : switch/case



## 4.-Programación Estructurada

### Condicionales

- Condicionales simples:

Se ejecuta una parte de código u otra dependiendo de una condición (if).  
La parte de código que se ejecuta se llama rama verdadera y la otra rama falsa.

La parte asociada a else es opcional.

```
int edadAlumno = 20;

// Condicionales simples

if (edadAlumno >= 18) {
    System.out.println("Eres mayor de edad");
} else {
    System.out.println("Eres menor de edad");
}
```

## 4.-Programación Estructurada

### Condicionales

- Condicionales múltiples:
  - Pueden tener varios casos (if-else-if-else). Se ejecuta una parte de código u otra dependiendo de una condición.

```
// Condicionales múltiples
if (edadAlumno >= 18) {
    System.out.println("Eres mayor de edad");
} else if (edadAlumno >= 16) {
    System.out.println("Casi eres mayor de edad");
} else {
    System.out.println("Eres menor de edad");
}
```

## 4.-Programación Estructurada

### Condicionales

- Switch-case

```
// Condicionales múltiples
char vocal = 'a';
switch (edadAlumno) {
    case 'a': {
        System.out.println("Eres la primera vocal");
    }
    case 'e': {
        System.out.println("Eres la segunda vocal");
    }
    case 'i': {
        System.out.println("Eres la tercera vocal");
    }
    case 'o': {
        System.out.println("Eres la cuarta vocal");
    }
    case 'u': {
        System.out.println("Eres la ultima vocal");
    }
    default: {
        System.out.println("No eres vocal");
    }
};
```

## 4.-Programación Estructurada

### Bucles

- Los bucles nos permiten ejecutar una parte de código varias veces. Existen varios tipos
- **Indefinidos:**
  - while
  - do-while
- **Definidos**
  - for



## 4.-Programación Estructurada

### Bucles

- **Indefinidos:**

- Se ejecutan hasta que se cumple una condición (while).
- Se ejecuta el bloque de código mientras se cumpla una condición.
- Si queremos que se ejecute al menos una vez, debemos usar do-while.
- Es importante manejar correctamente las variables que se usan en la condición (banderas o flags) para evitar bucles infinitos.

```
// Bucles indefinidos
// Ejemplo while
int i = 0;
while (i < 10) {
    System.out.println(i);
    i++;
}

// Ejemplo do-while
int j = 0;
do {
    System.out.println(j);
    j++;
} while (j<10);
```

## 4.-Programación Estructurada

- **Definidos**

- Se ejecutan un número determinado de veces (for) en base a un paso de iteración.
- Si el paso es 1 no se suele indicar.

```
// Bucles definidos
// Ejemplo for
for (int i=0;i<10;i++) {
    System.out.println(i);
}

// Con paso 2
for (int i=0;i<10;i+=2) {
    System.out.println(i);
}
```

```
// descendente
for (int i=10;i>0;i--) {
    System.out.println(i);
}

// Descendente con paso 3
for (int i=10;i>0;i-=3) {
    System.out.println(i);
}
```

## 4.-Programación Estructurada

- Comentarios

- Los comentarios son fragmentos de código que no se ejecutan. Se usan para documentar el código y explicar lo que hace. Pueden ser de una línea o de varias.

```
// Comentarios
// Comentario de una línea

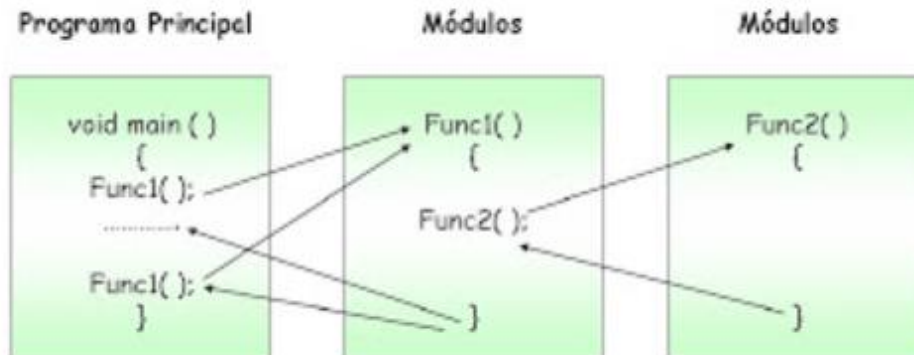
/*
Comentario de varias líneas
Para que los veas
*/
```

## 5.- Programación Modular

- La programación modular es un paradigma de programación
- Se basa en la **modularización** de los programas en funciones.
- Las funciones son bloques de código que realizan una tarea concreta y devuelven un valor.
  - **Si no devuelven ningún valor se denominan procedimientos.**
- Las ventajas que ofrece la programación modular son:
  - Facilita la resolución del problema.
  - Aumenta la claridad y legibilidad de todo el programa.
  - Permite que varios programadores trabajen en el mismo proyecto.
  - Reduce el tiempo de desarrollo ya que se pueden reutilizar esos módulos en varios programas.
  - Aumenta la fiabilidad porque es más sencillo diseñar y depurar módulos y el mantenimiento es más fácil.

## 5.- Programación Modular

- La descomposición modular se basa en la técnica “Divide y Vencerás”.
- Esta técnica tiene dos pasos:
  - Identificación de los subproblemas y construcción de los módulos que lo resuelven.
  - Combinación de los módulos para resolver el problema original.



## 5.- Programación Modular

### • Funciones

- Las funciones son bloques de código que realizan una tarea concreta y devuelven un valor.

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
    suma(2,4);  
    suma(4,5);  
}  
  
private static void suma(int a, int b) {  
    // TODO Auto-generated method stub  
    System.out.println("Sumamos los numeros "+a+" y "+b+":");  
    int resul = a + b;  
    System.out.println("\t resultado = "+resul);  
}
```

## 5.- Programación Modular

- **Parámetros**

- Paso por valor
  - Se crea una copia del valor del parámetro y se pasa a la función.
  - Si se modifica el valor del parámetro dentro de la función, no se modifica el valor original.
  - Lenguajes C
- **Paso por referencia**
  - Se pasa la dirección de memoria, por lo tanto si modificamos el parámetro dentro de la función, se modifica el valor original.
  - El parámetro del subalgoritmo, es decir, las modificaciones que sufra el parámetro, se reflejan en la variable que usamos en la llamada.
    - En pseudocódigo reflejaremos esta situación con la palabra “Ref”.

## 5.- Programación Modular

### Ámbito de una variable

- El ámbito de una variable es el lugar donde se puede utilizar.
- Las variables pueden tener ámbito local o global.
- Las variables locales solo se pueden utilizar dentro de la función donde se han declarado.
- Las variables globales se pueden utilizar en cualquier parte del programa.
  - Se deben intentar no abusar de las variables globales ya que pueden provocar errores en el programa



## 5.- Programación Modular

### Paquete o Módulo

- Un paquete o módulo es un conjunto de funciones y procedimientos que realizan una tarea concreta.
  - Por ejemplo:
    - un paquete de funciones matemáticas,
    - un paquete de funciones de entrada y salida,
    - un paquete de funciones de gestión de arrays, etc.
  - Usamos estos paquetes para agrupar funciones o clases que realizan una tarea concreta y que podemos reutilizar en otros programas o en otras partes del mismo.