Tipos de Datos

		NOMBRE	TIPO	OCUPA	RANGO APROXIMADO
TIPOS DE DATOS EN JAVA	TIPOS PRIMITIVOS (sin métodos; no son objetos; no necesitan una invocación para ser creados)	byte	Entero	1 byte	-128 a 127
		short	Entero	2 bytes	-32768 a 32767
		int	Entero	4 bytes	$2*10^9$
		long	Entero	8 bytes	Muy grande
		float	Decimal simple	4 bytes	Muy grande
		double	Decimal doble	8 bytes	Muy grande
		char	Carácter simple	2 bytes	---
		boolean	Valor true o false	1 byte	---
	TIPOS OBJETO (con métodos, necesitan una invocación para ser creados)	Tipos de la biblioteca estándar de Java	String (cadenas de texto) Muchos otros (p.ej. Scanner, TreeSet, ArrayList...)		
		Tipos definidos por el programador / usuario	Cualquiera que se nos ocurra, por ejemplo Taxi, Autobus, Tranvia		
		arrays	Serie de elementos o formación tipo vector o matriz. Lo consideraremos un objeto especial que carece de métodos.		
		Tipos envoltorio o wrapper (Equivalentes a los tipos primitivos pero como objetos.)	Byte		
			Short		
			Integer		
			Long		
			Float		
			Double		
			Character		
			Boolean		

Tipos de Datos

PRIMITIVOS

- Int
- Short
- Long
- Double
- Float
- Boolean
- Chart
- Byte

REFERENCIADOS

- String
- Array
- Character
- Collections

Variables y Constantes

Una **variable** es un nombre que se asocia con una porción de la memoria de la computadora, en la que se guarda un valor determinado. Por ejemplo, en matemáticas se usa la x como variable para asignarle valores.

Las **constantes** son valores que se mantienen siempre de igual manera y que no dependen de una asignación para obtener el mismo. Por ejemplo, el valor del IVA (21%).

Variables y Constantes

Reglas para nombres de variables

1. Representativo de lo que se guarda
2. Debe comenzar con letras
3. No debe contener espacios en blanco
4. No puede tener caracteres especiales o de puntuación

Variables y Constantes

Declaración y asignación de variables

Declaración: Tipo de dato + nombre de variable.
Por ejemplo: Int edad;

Asignación: Tipo de dato + nombre de la variable + valor.
Por ejemplo: int edad = 15;

```
1
2
3  class Persona {
4
5    public static void main(String[] args) {
6
7          int edad;
8          String nombre;
9
10         edad = 20;
11         nombre = "Julio";
12
13       System.out.println( "nombre: " + nombre + "\nedad: " + edad);
14    }
15 }
```

Estructura e Indentación del código

En Java, la indentación del código se refiere a la forma en que se organizan las líneas de código dentro de un programa de manera que sea fácil de leer y entender.

La indentación ayuda a destacar bloques de código relacionados y a identificar fácilmente el inicio y el final de cada bloque.

Estructura e Indentación del código

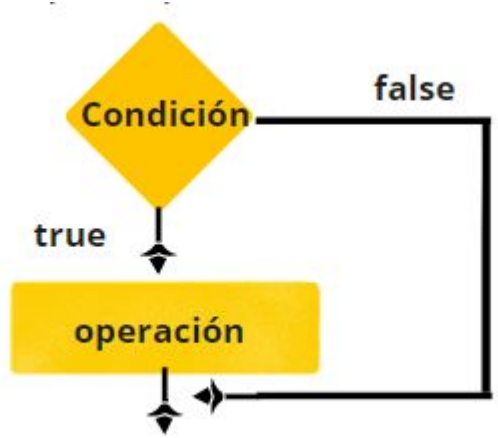
Hay algunas reglas básicas a seguir al indentar código en Java:

- Use sangrías o tabulaciones para indentar el código en lugar de espacios en blanco.
- Indente el código dentro de bloques de código, como los bloques de código de una clase, método o estructura de control de flujo (if, for, etc.).
- Alinee el código en bloques relacionados. Por ejemplo, alinee los puntos y coma de las declaraciones de variables en una clase o los puntos y coma de las instrucciones en un método.
- Evite utilizar una indentación excesiva. Un nivel o dos de indentación suele ser suficiente para la mayoría de los programas.

ESTRUCTURAS CONDICIONALES: IF

La estructura de control IF permite decidir entre dos opciones resultantes de la evaluación de una sentencia.

Si la evaluación es positiva se ejecuta una parte del código, de lo contrario el código dentro de la condición no se ejecuta o se establece otra serie de acciones para que se ejecuten en ese caso.

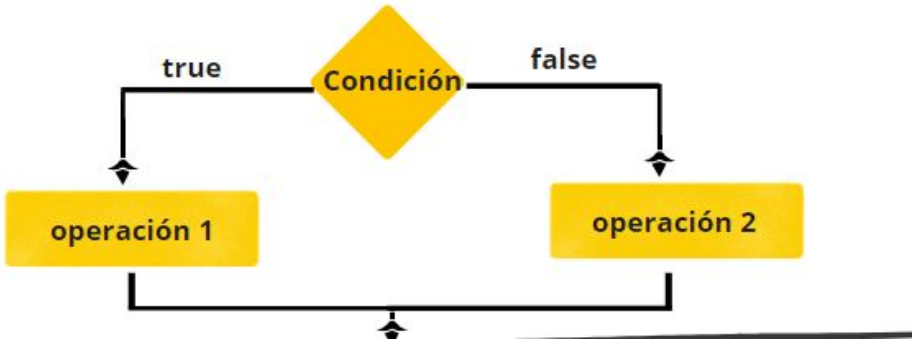


ESTRUCTURAS CONDICIONALES: IF

```java
import java.util.Scanner;

class Persona {

  public static void main(String[] args) {

        int edad;
        String nombre;

        Scanner scanner = new Scanner(System.in);

       System.out.print("Ingrese su nombre: ");
          nombre = scanner.nextLine();
       System.out.print("Ingrese su edad: ");
         edad =  scanner.nextInt();

           if (edad >= 18)

      System.out.println( nombre + " es mayor de edad" );
    }
}
```

ESTRUCTURAS CONDICIONALES: IF/ ELSE

La estructura de control **IF ELSE** es la extensión de la sentencia IF. Significa "de lo contrario" y permite la ejecución de un bloque de código si el resultado de la evaluación de la condición es falso. La sentencia **ELSE** se ejecuta solamente si la expresión **IF** se evalúa como falsa.



ESTRUCTURAS CONDICIONALES: IF/ ELSE

```java
import java.util.Scanner;

class Persona {

  public static void main(String[] args) {

        int edad;
        String nombre;

        Scanner scanner = new Scanner(System.in);

       System.out.print("Ingrese su nombre: ");
          nombre = scanner.nextLine();
       System.out.print("Ingrese su edad: ");
         edad =  scanner.nextInt();

        if (edad >= 18) {

       System.out.println( nombre + " es mayor de edad" );
    }
       else {
            System.out.println( nombre + " es menor de edad" );
       }
}
}
```

ESTRUCTURAS CONDICIONALES: SWITCH

La estructura Switch permite múltiples caminos a partir de la evaluación de una sola condición/expresión.

La expresión puede ser una variable o cualquier otro tipo, siempre y cuando se evalúe un valor simple.

La construcción se ejecuta mediante la evaluación de la condición y un conjunto de sentencias **case**.

Cada **case** es una posible respuesta a la evaluación de esa condición,si el valor que se busca coincide con algún case, se ejecuta el mismo hasta la sentencia **break** o hasta el final del switch, dependiendo del caso.

ESTRUCTURAS CONDICIONALES: SWITCH

```java
import java.util.Scanner;

class Persona {

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

        System.out.print("Ingrese un numero del 1 al 7: ");
        int dia = scanner.nextInt();

        switch(dia) {
            case 1:
                System.out.println("Hoy es Lunes.");
                break;
            case 2:
                System.out.println("Hoy es Martes.");
                break;
            case 3:
                System.out.println("Hoy es Miercoles.");
                break;
            case 4:
                System.out.println("Hoy es Jueves.");
                break;
            case 5:
                System.out.println("Hoy es Viernes.");
                break;
            case 6:
                System.out.println("Hoy es Sabado.");
                break;
            case 7:
                System.out.println("Hoy es Domingo.");
                break;
            default:
                System.out.println("Numero invalido.");
                break;
        }
    }
}
```

ESTRUCTURAS REPETITIVAS: WHILE

La estructura **WHILE** permite ejecutar una porción de código **"mientras"** se cumpla una determinada condición.
La **condición se evalúa siempre al inicio del bucle.**

Si la condición es verdadera ingresa al bucle y ejecuta el código.
Si la condición del bucle While se evalúa a falso cuando se ejecuta el bucle por primera vez, el cuerpo del bucle no se ejecutará nunca.

La condición lógica en un bucle While, debe ser modificada por una sentencia en el cuerpo del bucle, en caso contrario, el **bucle es infinito.**

ESTRUCTURAS REPETITIVAS: WHILE

```
class Persona {

  public static void main(String[] args) {


    int i = 1;
       while(i <= 5) {
           System.out.println(i);
           i++;
       }
}
}
```

ESTRUCTURAS REPETITIVAS: DO/ WHILE

La estructura do-while es un tipo de bucle en Java que permite ejecutar un bloque de código al menos una vez, y luego seguir ejecutándolo mientras se cumpla una cierta condición.

ESTRUCTURAS REPETITIVAS: DO/ WHILE

```java
import java.util.Scanner;

class Persona {

  public static void main(String[] args) {

  Scanner scanner = new Scanner(System.in);

    int numero;
        do {
            System.out.print("Ingrese un numero entre 1 y 10: ");
            numero = scanner.nextInt();
        } while (numero < 1 || numero > 10);

        System.out.println("Numero ingresado: " + numero);
}
}
```

ESTRUCTURAS REPETITIVAS: FOR

El bucle **FOR** se utiliza cuando se quiere repetir un conjunto de sentencias un número determinado de veces, usando una variable numérica capaz de controlar el número de iteraciones.

Es decir que el número de iteraciones se conoce por anticipado, y por ello no se precisa poner ninguna condición de salida para detener el bucle.

En su lugar un contador cuenta el número de iteraciones fijas y se termina cuando llega al valor final previamente definido.

Está compuesto por tres partes:

1. **Inicialización de la variable:** que utilizaremos en la condición -se ejecuta solo una vez al principio del ciclo-.
2. **Condición de fin de vida del ciclo:** se evalúa esta expresión al comienzo de cada iteración.
3. **Modificación de la variable:** se ejecuta al final de cada iteración.

ESTRUCTURAS REPETITIVAS: FOR

```java
public class EjemploFor {
    public static void main(String[] args) {
        for (int i = 1; i <= 5; i++) {
            System.out.println(i);
        }
    }
}
```

EJERCICIOS

1. Escribir un programa que solicite al usuario ingresar una cantidad de números enteros, y luego lea esos números y determine cuántos de ellos son números pares. Utilice un bucle while para leer los números, y una estructura if para determinar si cada número es par o impar.

2. Escribir un programa que solicite al usuario ingresar un número, y luego determine si ese número es primo o no. Utilice un bucle do-while para solicitar al usuario que ingrese un número válido, y una estructura if-else para determinar si el número es primo o no.

3. Escribir un programa que solicite al usuario que ingrese un número del 1 al 7, y luego imprima el día de la semana correspondiente a ese número (1 para lunes, 2 para martes, etc.). Utilice un bucle for para verificar que el número ingresado esté dentro del rango permitido, y una estructura switch para imprimir el día de la semana correspondiente.