Taller JAVA Introductorio

Presented By

Adler Pérez
Henri Martínez
Angel González
Fernando Alvarado

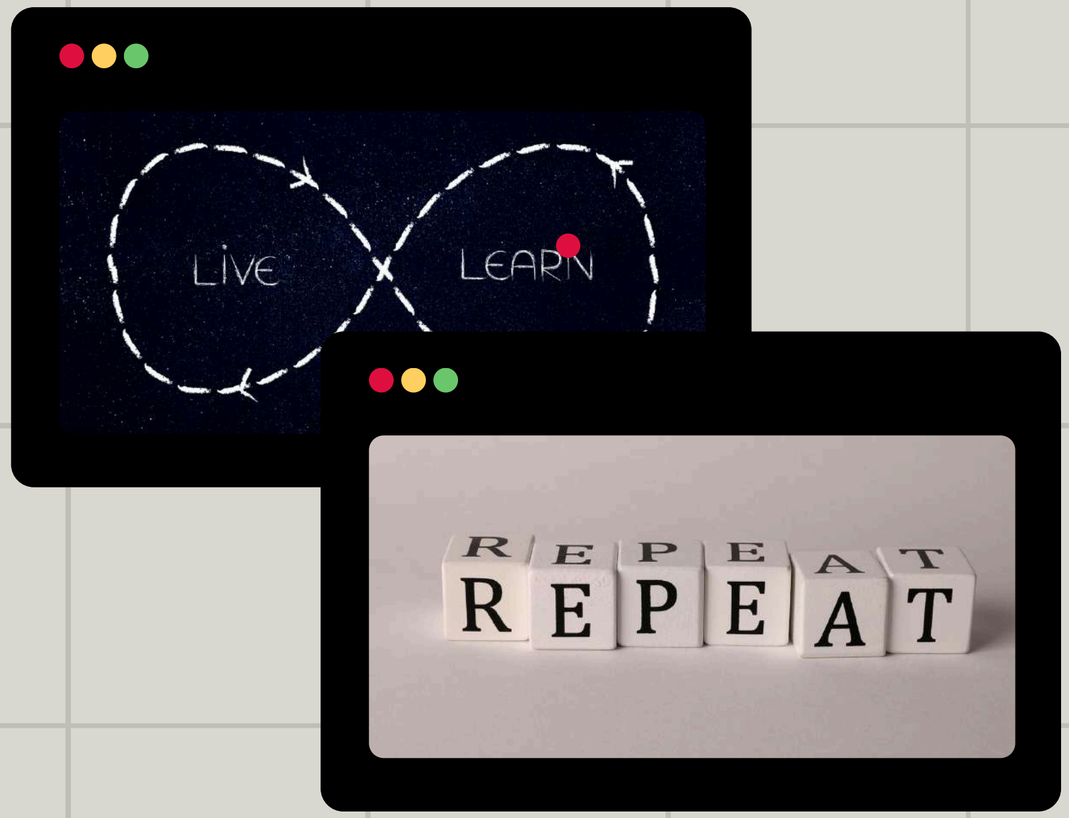
Matemática para Computación 1

Martes 25 de marzo del 2025

Bucles

Un bucle es una estructura que nos permite repetir instrucciones varias veces sin escribir código repetitivo.

- Automatizan tareas repetitivas.
- Hacen el código más eficiente y fácil de mantener.
- Se usan en algoritmos, procesamiento de datos, videojuegos, etc.



Ejemplo práctico: Evitando código repetitivo

Sumando 5 números SIN BUCLE

```java
Scanner sc = new Scanner(System.in);

int suma = 0;

System.out.print("Introduce el primer numero: ");
suma = sc.nextInt();
System.out.print("Introduce el segundo numero: ");
suma += sc.nextInt();
System.out.print("Introduce el tercer numero: ");
suma += sc.nextInt();
System.out.print("Introduce el cuarto numero: ");
suma += sc.nextInt();
System.out.print("Introduce el quinto numero: ");
suma += sc.nextInt();

System.out.println("La suma de los números es: " + suma);
```

Sumando 5 números CON BUCLE

```java
Scanner sc = new Scanner(System.in);

int suma = 0;

for (int i = 0; i < 5; i++) {
    System.out.println("Introduce un número: ");
    suma += sc.nextInt();
}

System.out.println("La suma de los números es: " + suma);
```

Los bucles nos evitan escribir código innecesariamente.

Bucles en java

Java tiene tres tipos de bucles, cada uno se usa para casos distintos.

- **for** → Se usa cuando sabemos cuántas veces queremos repetir la acción.
- **while** → Se usa cuando repetimos algo mientras se cumpla una condición.
- **do-while** → Similar al while, pero se ejecuta al menos una vez.

for

📌 ¿Cuándo usarlo?

- Cuando sabemos cuántas veces se ejecutará el código.
- Útil para recorrer arreglos, iteraciones contadas, etc.

```
for (inicialización; condición; incremento) {
    // Código a ejecutar
}
```

for

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

```
run:
1
2
3
4
5
6
7
8
9
10
BUILD SUCCESSFUL
```

for

```java
Scanner sc = new Scanner(System.in);
int suma = 0;

for (int i = 0; i < 3; i++) {
    System.out.println("Introduce un número: ");
    suma += sc.nextInt();
}

System.out.println("El promedio es: " + suma / 3);
```

while

📌 ¿Cuándo usarlo?

- Cuando no sabemos cuántas veces se ejecutará.
- Se ejecuta mientras la condición sea verdadera.

```
while (condicion){
    // Codigo a ejecutar
}
```

while

```java
Scanner sc = new Scanner(System.in);
String contrasena = "1234";
String contrasenaIntroducida = "";

while (!contrasena.equals(contrasenaIntroducida)) {
    System.out.println("Introduce la contraseña:");
    contrasenaIntroducida = sc.nextLine();
}

System.out.println("Contraseña correcta");
```

while (PRECAUCIÓN)

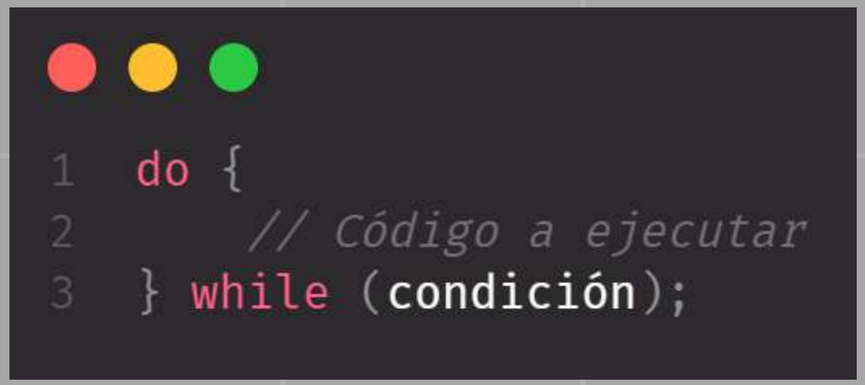
```java
//Bucle while infinito, no se incrementa el contador
int contador = 0;
while (contador < 10) {
    System.out.println(contador);
}

//Forma correcta de hacerlo
int contador = 0;
while (contador < 10) {
    System.out.println(contador);
    contador++;
}
```

do while

📌 ¿Cuándo usarlo?

- Cuando no sabemos cuántas veces se ejecutará.
- Se ejecuta mientras la condición sea verdadera.

Similar a while, pero garantiza al menos una ejecución.



do while

```java
Scanner scanner = new Scanner(System.in);
int opcion;
do {
    System.out.println("1. Saludar");
    System.out.println("2. Despedirse");
    System.out.println("3. Salir");
    System.out.print("Elige una opción: ");
    opcion = scanner.nextInt();

    switch (opcion) {
        case 1:
            System.out.println("Hola, ¿cómo estás?");
            break;
        case 2:
            System.out.println("Adiós");
            break;
        case 3:
            System.out.println("Saliendo...");
            break;
        default:
            System.out.println("Opción no válida");
            break;
    }

} while (opcion != 3);
```

Sentencias importantes

break

Sale del bucle antes de que termine.

continue

Salta una iteración y sigue con la siguiente.

```java
for (int i = 1; i <= 10; i++) {
    if (i == 5) {
        continue; // Salta la iteración cuando i es 5
    }
    if (i == 8) {
        break; // Sale del bucle cuando i es 8
    }
    System.out.println(i);
}
```

```
run:
1
2
3
4
6
7
BUILD SUCCESSFUL
```

Bucles anidados

📌 ¿Qué son los bucles anidados?

- Un bucle anidado es un bucle dentro de otro bucle.

Se usa cuando necesitamos recorrer estructuras en múltiples dimensiones (ej. tablas)

Bucles anidados

```java
for (int i = 1; i <= 5; i++) {  // Bucle externo
    System.out.println("Bucle externo, iteración " + i + ":");

    for (int j = 1; j <= 3; j++) {  // Bucle interno
        System.out.print("(" + i + "," + j + ") ");
    }

    System.out.println();  // Salto de línea después de cada iteración del bucle externo
}
```

```
run:
Bucle externo, iteración 1:(1,1) (1,2) (1,3)
Bucle externo, iteración 2:(2,1) (2,2) (2,3)
Bucle externo, iteración 3:(3,1) (3,2) (3,3)
Bucle externo, iteración 4:(4,1) (4,2) (4,3)
Bucle externo, iteración 5:(5,1) (5,2) (5,3)
BUILD SUCCESSFUL (total time: 0 seconds)
```

Ejercicio práctico

Hacer una calculadora, debe tener un menú que nos deje elegir entre las siguientes opciones:

1. Suma
2. Resta
3. Imprimir todas las tablas de multiplicar (1 al 10)
4. Salir

Al seleccionar la opción debe pedir 2 números (si la opción lo requiere) e imprimir el resultado.

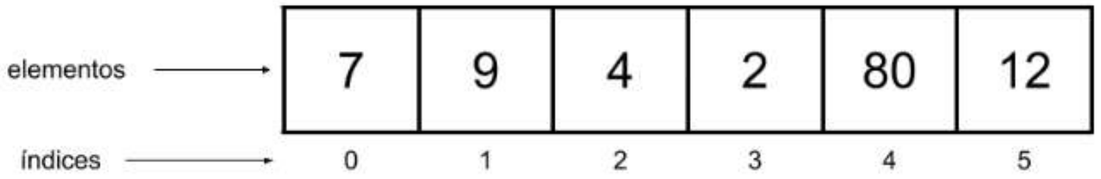
Arreglos estáticos

El Arreglo es un tipo de dato NO primitivo. Como el ***String*** es un conjunto de caracteres, el ***Arreglo*** es un conjunto de múltiples valores, de un mismo tipo de dato.

En Java se pueden crear arreglos de forma sencilla, arreglos estáticos, esto quiere decir que su tamaño será fijo. El tamaño será lo que el programador especifique en la declaración.

Cada elemento en un arreglo tiene un índice, esto indica la posición que tiene este elemento dentro del arreglo.

Siempre los índices inician en 0, luego 1, después 2, y así sucesivamente. En este tipo de arreglos no se pueden eliminar elementos como tal, solo reemplazarlos.



Declaración e inicialización

Valores iniciales por defecto

Sintáxis: *tipo[] nombreArreglo = new tipo[cantidadValores];*

```
int[] edades = new int[8];
```

```
String[] nombres = new String[25];
```

Con elementos iniciales

Sintáxis: *tipo[] nombreArreglo = {elemento1, elemento2, elemento3...};*

```
int[] edades = {17, 18, 21, 24, 19};
```

```
String[] nombres = {"Juan", "Ana", "Marcos"};
```

Acceso a elementos

Puedes acceder a los elementos, especificando el índice del elemento (Tomando en cuenta los límites)

```
String[] carros = {"Volvo", "BMW", "Toyota", "Mazda"};
System.out.println(carros[2]);
// Salida: Toyota
```

La misma lógica para cambiar un elemento de un arreglo

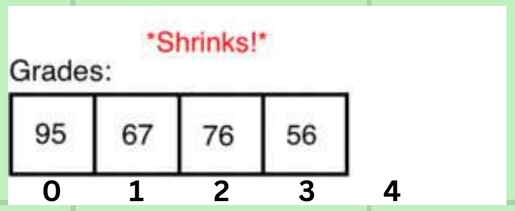
```
String[] carros = {"Volvo", "BMW", "Toyota", "Mazda"};
carros[1] = "Ford";
System.out.println(carros[1]);
// Salida: Ford (Ya no es BMW)
```

Longitud

Para saber cuántos elementos tiene un arreglo, se usa la propiedad *length*

```
String[] carros = {"Volvo", "BMW", "Toyota", "Mazda"};
System.out.println(carros.length);
// Salida: 4
```

Arreglos dinámicos

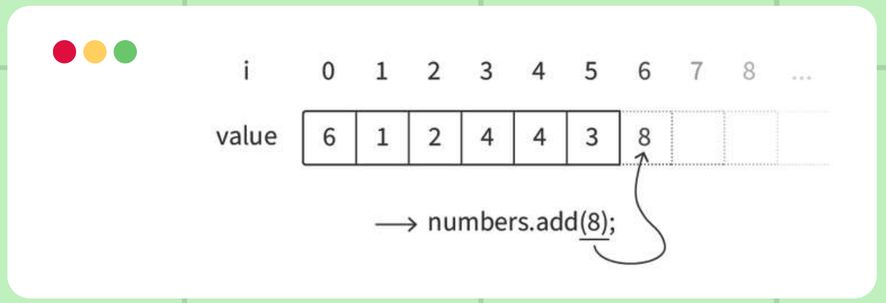


Los arreglos dinámicos son exactamente lo mismo que los arreglos estáticos, solo que son la versión mejorada. Los arreglos dinámicos no tienen un tamaño fijo. Pueden ir aumentando en tamaño, si se requiere.

Los elementos de una ArrayList, que es una clase en Java brindada para tener un arreglo dinámico, se pueden añadir, y eliminar, conforme se necesite en el programa.

De igual forma, cada elemento en un arreglo dinámico tiene un índice, esto indica la posición que tiene cada elemento dentro del arreglo dinámico.

Estos índices pueden tener un elemento distinto, dependiendo si se elimina un elemento del arreglo dinámico



Tipo Primitivo	Clase Wrapper
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Declaración y asignación de elementos

Se crea un objeto de tipo *ArrayList* (para esto se debe importar la clase *java.util.ArrayList*)

Sintáxis: *Arraylist<tipo> nombreArreglo = new ArrayList<>();*

```
import java.util.ArrayList;
ArrayList<String> carros = new ArrayList<>();
```

Para añadir (agregar) elementos al ArrayList

Sintáxis: *nombreArreglo.add(elemento);*
Sintáxis: *nombreArreglo.add(índice, elemento);*

```
cars.add("Volvo");
cars.add("BMW");
```
⟶ ["Volvo", "BMW"]

```
cars.add(0, "Kia");
```
⟶ ["Kia", "Volvo", "BMW"]

Obtener un elemento del ArrayList

Sintáxis: *nombreArreglo.get(índice);*

```
cars.get(1);    // Salida: Volvo
cars.get(4);    // Salida: ERROR!
```

Cambiar un elemento del ArrayList

Sintáxis: *nombreArreglo.set(índice, nuevoElemento);*

```
cars.set(0, "Opel");
cars.set(1, "Ford");
```
⟶ ["Opel", "Ford", "BMW"]

Eliminar un elemento del ArrayList

Sintáxis: *nombreArreglo.set(índice, nuevoElemento);*

```
cars.remove(1);
```
⟶ ["Opel", "BMW"]

Eliminar todos los elementos del ArrayList

```
cars.clear();
```

Obtener la longitud de un ArrayList

```
cars.size();    // Salida: 2
```

Ejercicios

```
String[] equipos = {"Equipo A", "Equipo B", "Equipo C", "Equipo D", "Equipo E",
                    "Equipo F", "Equipo G", "Equipo H", "Equipo I", "Equipo J"};
int[] puntajes = new int[10]; // Inicialmente, todos los equipos tienen 0 puntos
```

En un torneo de fútbol, donde **participan 10 equipos** y cada equipo tiene un nombre y un puntaje fijo, ¿qué estructura usarías y cómo serían los arreglos creados?

```
ArrayList<String> carrito = new ArrayList<>();
carrito.add("Laptop");
carrito.remove("Laptop");
```

Si estás creando un programa donde los usuarios pueden **agregar y eliminar productos** de un carrito de compras, ¿qué estructura usarías y cómo serían los arreglos creados?

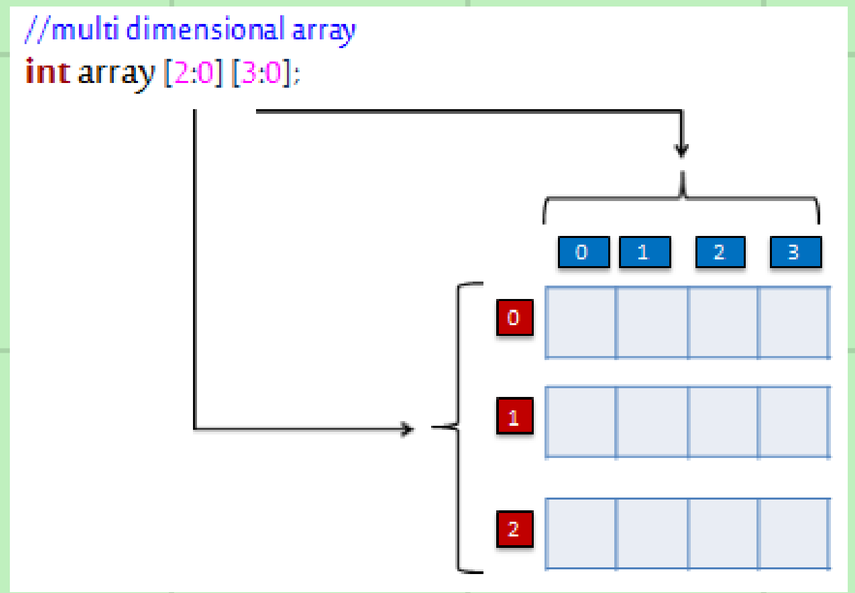
Ejemplo en Vivo Arreglos

Arreglos multidimensionales

Los arreglos multidimensionales, o matrices, son arreglos, en los que cada uno de sus elementos guarda otro arreglo, formando una matriz, esta puede ser de 2 o más direcciones. Aunque lo más habitual es que sean de 2 dimensiones.

Cada elemento en un arreglo multidimensional tiene tantos índices como dimensiones del arreglo.

Es decir, en una matriz de 2 dimensiones, cada elemento tendrá entonces 2 índices para poder posicionarlo. Uno de ellos representará las filas, y el otro, la columna en donde esté el elemento.




```
myArray[1][0]= 4;
```

	0	1	2
0	1 [0][0]	2 [0][1]	3 [0][2]
1	4 [1][0]	5 [1][1]	6 [1][2]

Declaración e inicialización

Valores iniciales por defecto

Sintáxis: *tipo[][] nombreArreglo = new tipo[cantValores1][cantValores2];*

```
int[][] edades = new int[4][2];
```

```
String[][] nombres = new String[3][5];
```

Con elementos iniciales

Sintáxis: *tipo[][] nombreArreglo = {arreglo1, arreglo2, arreglo3, ...};*

```
int[][] edades = {{17, 20}, {30, 27}};
```

```
String[][] nombres = {{"Juan", "Ana"},
                      {"Mia", "Gio"}};
```

La cantidad de [] dependerá de las dimensiones que se requieran

Acceso y cambio de elementos

Puedes acceder a un elemento, especificando la cantidad necesaria de índices que tiene el elemento. En una matriz de 2 dimensiones se deben especificar 2 índices: Uno que indica el número del arreglo, y el otro para especificar el elemento adentro de ese arreglo (Tomando en cuenta los límites)

```
int[][] numbers = {{1, 2, 3}, {4, 5}, {6, 7, 8}};
System.out.println(numbers[0][2]);
// Salida: 3
```

Se aplica la misma lógica para cambiar un elemento de un arreglo

```
int[][] numbers = {{1, 2, 3}, {4, 5}, {6, 7, 8}};
numbers[0][2] = 33;
numbers[2][0] = 66;
System.out.println(numbers[0][2]);
// Salida: 33
```

Recorrer arreglos y matrices (FOR)

Arreglos

```java
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (int i = 0; i < cars.length; i++) {
  System.out.println(cars[i]);
}
```

- La inicialización siempre será donde querramos iniciar el recorrido, normalmente 0, el índice inicial del arreglo
- La condición es cuantas veces queremos que corra el ciclo, normalmente será el tamaño del arreglo
- La actualización es cómo se darán los saltos, normalmente será de uno en uno

Adentro del ciclo irá lo que queremos hacer con cada elemento del arreglo, ya sea imprimir, reemplazar el contenido, u obtener el valor para realizar operaciones o acciones con este.

Matrices

```java
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
for (int i = 0; i < myNumbers.length; ++i) {
  for (int j = 0; j < myNumbers[i].length; ++j) {
    System.out.println(myNumbers[i][j]);
  }
}
```

Recorrer arreglos y matrices (FOR-EACH)

Arreglos

```java
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String i : cars) {
  System.out.println(i);
}
```

Adentro del ciclo irá lo que queremos hacer con cada elemento del arreglo, ya sea imprimir, reemplazar el contenido, u obtener el valor para realizar operaciones o acciones con este.

Este ciclo es de uso exclusivo para recorrer los elementos de arreglos o matrices.
Hay que tener muy en cuenta el tipo de dato del que se compone el arreglo.

Sintáxis:

```
for (tipo nombreElemento : nombreArreglo) {
    ...
}
```

Matrices

```java
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
for (int[] row : myNumbers) {
  for (int i : row) {
    System.out.println(i);
  }
}
```

Aquí se observa, que en el primer ciclo for-each, el tipo de datos es de tipo: *arreglo de enteros -> int[]*

Ejemplo en Vivo
Arreglos, Matrices y Ciclos

Métodos

Un método es un bloque de código reutilizable que:

- Realiza una tarea específica (como por ejemplo calcular, imprimir o validar).
- En la ejecución de esta tarea, puede recibir datos (parámetros) y puede devolver un resultado (valor de retorno).

Imaginen un método con la funcionalidad de un microondas:

- Recibe parámetros (tiempo, potencia).
- Ejecuta una acción (calentar la comida).
- Devuelve un resultado ("Comida caliente")

Estructura de la declaración de un Método

La declaración de un método es la definición de su estructura y comportamiento.

```
[modificador] [tipo-retorno] [nombre]([parámetros]) {
    // Cuerpo (instrucciones)
    [return valor;] // Si no es void
}
```

```
public static void saludar() {
    System.out.println("Hola, buenos días");
}
```

Parte	Ejemplo	¿Para qué sirve?
Modificador	public static	Define quién puede usarlo (public, private, static, etc.).
Tipo de retorno	int, String, void	Especifica qué devuelve. Si es void, no retorna nada.
Nombre	calcularIVA	Debe ser un verbo + sustantivo (en camelCase).
Parámetros	(double precio)	Datos que necesita para trabajar (opcionales).
Cuerpo	{...}	Instrucciones que ejecuta.
Return	return total;	Obligatorio solo si no es void.

Llamada a un método

Una llamada a método es la ejecución del código contenido en un método. Cuando llamas a un método:

- El flujo del programa salta a ese método.
- Se ejecutan las instrucciones de método.
- Después de ejecutar las instrucciones, el flujo retorna al punto donde se llamó al método.

```
nombreMetodo(argumentos);
```

```
saludar(); // Imprime "Hola, buenos días"
```

Parte	Descripción
Nombre del método	Identificador del método a ejecutar.
Argumentos	Valores concretos que coinciden con los parámetros declarados.

Métodos sin retorno (Void)

```java
public static void saludar() {
    System.out.println("Hola, buenos días");
}
```

Características:

- No devuelven valores
- Usan palabra clave void

Útiles para:

- Mostrar información (System.out.println).
- Modificar atributos de objetos.
- Ejecutar procesos sin necesidad de resultado.

Ejemplo en Vivo

Métodos Void

Métodos con Retorno

```java
public static int calcularEdad() {
        int añoNacimiento = 1998;
        int añoActual = 2025;
        return añoActual - añoNacimiento;
    }
```

Características Clave

- En lugar de Void, deben declarar el tipo de dato del retorno antes de indicar el nombre.
- Requieren una sentencia return con un valor del tipo declarado dentro del cuerpo del método.
- Todos los caminos de ejecución deben terminar en return.
- Pueden usarse en expresiones (asignaciones, operaciones, etc.).
- Solo se puede retornar un valor a la vez.
- El tipo de retorno debe coincidir

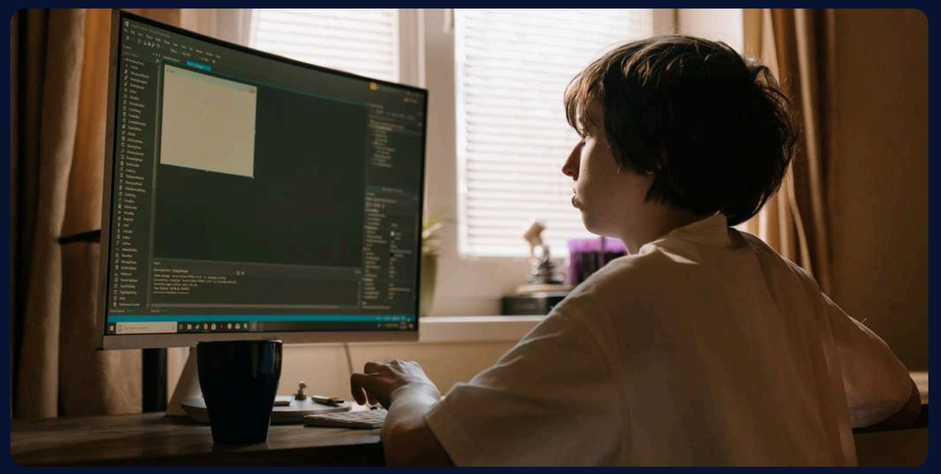
Útiles para:

- Cuando se necesitan resultados de cálculos
- Para validaciones o comprobaciones.

```java
int edadActual = calcularEdad();
```

Tipo de Retorno	Ejemplo	Descripción
Primitivos	int, double, boolean	Valores básicos
Objetos	String, ArrayList, Scanner	Instancias de clases
Arreglos	int[], String[] []	Colecciones de datos

Ejemplo en Vivo: Métodos con retorno



Ejercicio

Contador de vocales

Se requiere crear un metodo **CON RETORNO** que **DEVUELVA** el **NÚMERO** de vocales (a,e,i,o,u) eN la siguiente frase:

“La felicidad es contagiosa”

El nombre de la funcion debe ser **contarVocales();**

Ejercicio

Se recomienda utilizar el siguiente codigo para empezar

```java
public static int contarVocales() {
    String texto = "La felicidad es contagiosa";
    int contador = 0;

    // Convertimos el texto en un arreglo de caracteres
    char[] letras = texto.toCharArray();
}
```

GRACIAS!