

# JAVA

---

## Estructura basica de un programa

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

- **Public class:** Define la clase publica.
- **Public static void main**(String[] args){ : Método principal que ejecuta el programa
- **System.out.print("\n")** "sout": Imprime texto en la consola.

---

## Variables y tipos de datos

**int** : Enteros. Ej: int numero = 10;

**double** : Decimales. Ej: double precio = 19.99;

**char** : caracteres individuales. Ej: char letra = 'A';

**boolean** (String 1, String2) : Verdadero o falso. Ej: boolean esJavaFacil = true;

**String** : Cadena de texto. Ej: String saludo ="Hola, Java";

---

## Operadores

+ suma

- resta

\* multiplicación

/ división

% módulo ej. int resto = 10%3;

> mayor que if (a > b)

< menor que if (a < b)

Igualdad == if (a == b)

Distinto != if (a != b)

Mayor o igual >= if (a >= b)

Menor o igual <= if (a <= b)

AND: && `if(a > b && c > d)`

OR: || `if ( a > b || c > d)`

NOT: ! `If( !condición)`

---

## Estructuras de control

- Condicionales

```
if (condicion) {  
    // código si la condición es verdadera  
} else {  
    // código si la condición es falsa  
}
```

- Switch

```
switch (variable) {  
    case 1:  
        // código para el caso 1  
        break;  
    case 2:  
        // código para el caso 2  
        break;  
    default:  
        // código por defecto  
}
```

---

## Bucles

- **For** (fori)

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

- **While y do-while**

- **While:**

```
java Copiar código  
  
while (condicion) {  
    // código que se ejecuta mientras la condición sea verdadera  
}
```

- **Do-while:**

```
java Copiar código  
  
do {  
    // código que se ejecuta al menos una vez  
} while (condicion);
```

---

## Funciones y métodos

- Declaración de métodos

```
public static int sumar(int a, int b) {  
    return a + b;  
}
```

- Llamada a métodos

```
int resultado = sumar (3,4);
```

```
System.out.println("Resultado: " + resultado);
```

```
public class Calculadora {  
  
    // Método para sumar dos números  
    public static int sumar(int a, int b) {  
        return a + b; // Devuelve la suma de a y b  
    }  
  
    public static void main(String[] args) {  
        // Aquí es donde llamamos al método sumar  
        int resultado = sumar(5, 10); // Llamada al método  
        System.out.println("La suma es: " + resultado); // Muestra el resultado  
  
        // Puedes llamar al método sumar con diferentes valores  
        int otraSuma = sumar(7, 3);  
        System.out.println("Otra suma es: " + otraSuma);  
    }  
}
```

Ejemplo completo

---

## Clases y objetos

```
public class Main {  
    public static void main(String[] args) {  
        // Crear el primer objeto de la clase Coche  
        Coche coche1 = new Coche("Toyota", "Corolla", 2020);  
  
        // Llamar a los métodos del objeto coche1  
        coche1.mostrarDetalles();  
        coche1.acelerar();  
  
        System.out.println(); // Línea en blanco para separar la salida  
  
        // Crear otro objeto de la clase Coche  
        Coche coche2 = new Coche("Ford", "Mustang", 2023);  
  
        // Llamar a los métodos del objeto coche2  
        coche2.mostrarDetalles();  
        coche2.acelerar();  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        // Crear el primer objeto de la clase Coche  
        Coche coche1 = new Coche("Toyota", "Corolla", 2020);  
  
        // Llamar a los métodos del objeto coche1  
        coche1.mostrarDetalles();  
        coche1.acelerar();  
  
        System.out.println(); // Línea en blanco para separar la salida  
  
        // Crear otro objeto de la clase Coche  
        Coche coche2 = new Coche("Ford", "Mustang", 2023);  
  
        // Llamar a los métodos del objeto coche2  
        coche2.mostrarDetalles();  
        coche2.acelerar();  
    }  
}
```

- \* La palabra clave **this** en Java es una referencia especial que se usa dentro de una clase para referirse al **objeto actual**. Es muy útil para distinguir entre los **atributos de la clase** y los **parámetros del método**, así como para llamar a otros constructores y métodos de la misma clase.

---

## Expresiones comunes

- Concatenación de cadenas

```
System.out.println("Hola" + "," + "mundo");
```

```
String Saludo = "Hola" + "," + "mundo";
```

- Conversiones de tipo

```
Int num = Integer.parseInt("123");
```

```
String strNum = String.valueOf(456);
```

## Arreglos (Arrays)

### - Declaración de un Array:

```
Int [ ] numeros = new int[5];    // Array de 5 elementos
enteros
```

```
String [ ] nombres = {"Juan", "Ana ", "Luis"}; // Array
inicializado
```

### - Acceso a elementos del array.

```
System.out.println(numeros[0]); // Muestra el primer elemento

numeros[1] = 10;                // Asigna valor al segundo
elemento
```

### - Recorrer un Array

```
for (int i = 0; i < numeros.length; i++) {

    System.out.println(numeros[i]);

}
```

\* Ejemplo de tener un array y querer imprimir cada nombre:

```
public class EjemploForEach {
    public static void main(String[] args) {
        // Definir un array de nombres
        String[] nombres = {"Ana", "Juan", "Pedro", "Luisa"};

        // Usar un bucle for-each para imprimir cada nombre
        for (String nombre : nombres) {
            System.out.println(nombre);
        }
    }
}
```

Devuelve :

Ana

Juan

Pedro

Luisa

### - Propiedades del Array:

**length** : devuelve el tamaño del array. `numeros.length()`

---

## ArrayList

Clase de la colección de Java que representa una lista dinámica, es decir, puede cambiar de tamaño.

- Importar la clase

```
import java.util.ArrayList;
```

- Declaración del ArrayList

```
ArrayList<String> listaNombres = new ArrayList<>();
```

- Operaciones comunes en ArrayList:

- **Agregar elementos** add: `listaNombres.add("Carlos");`  
`listaNombres.add("Ana");`
- **Acceder a un elemento** get:  
`System.out.println(listaNombres.get(0));` // Imprime "Carlos"
- **Modificar un elemento** set: `listaNombres.set(0, "Juan");` //  
Cambia "Carlos" por "Juan"
- **Eliminar un elemento** remove: `listaNombres.remove(1);` // Elimina  
el elemento en la posición 1 (Ana)
- **Verificar un tamaño** size:  
`System.out.println(listaNombres.size());` // Devuelve el  
número de elementos
- **Recorrer un ArrayList:**

```
for (String nombre : listaNombres) {  
    System.out.println(nombre);  
}
```

## Listas

La interfaz `List` es una estructura de datos más general en Java que permite almacenar una secuencia de elementos y proporciona métodos para manipularlos. `ArrayList` es una implementación de `List`.

- Declaración de una lista usando `List`:

```
import java.util.List;
import java.util.ArrayList;

List<Integer> numeros = new ArrayList<>();
numeros.add(10);
numeros.add(20);
numeros.add(30);
```

Otras implementaciones de `List`:

- `LinkedList`: Es una lista enlazada donde cada elemento es un nodo que contiene el valor y un enlace al siguiente nodo.

```
import java.util.LinkedList;

LinkedList<String> linkedList = new LinkedList<>();
linkedList.add("Elemento1");
linkedList.add("Elemento2");
```



### EJEMPLO ArrayList, List, LinkedList

```
import java.util.List;
import java.util.ArrayList;
import java.util.LinkedList;

public class ListVsArrayListVsLinkedListExample {

    public static void main(String[] args) {

        // Usando List implementado por ArrayList
        List<String> arrayList = new ArrayList<>();
        arrayList.add("Apple");
        arrayList.add("Banana");
        arrayList.add("Cherry");

        System.out.println("Elementos en arrayList
(ArrayList):");

        for (String fruit : arrayList) {

            System.out.println(fruit);

        }

        // Accediendo a un elemento por índice en ArrayList
        System.out.println("\nElemento en el índice 1 en
arrayList: " + arrayList.get(1));

        // Eliminando un elemento en ArrayList
        arrayList.remove(0); // Elimina "Apple"

        System.out.println("\nElementos después de eliminar en
arrayList:");

        for (String fruit : arrayList) {

            System.out.println(fruit);

        }

    }

}
```

```
// Usando List implementado por LinkedList

List<String> linkedList = new LinkedList<>();

linkedList.add("Dog");
linkedList.add("Cat");
linkedList.add("Rabbit");


System.out.println("\nElementos en linkedList
(LinkedList):");

for (String animal : linkedList) {
    System.out.println(animal);
}

// Accediendo a un elemento por índice en LinkedList

System.out.println("\nElemento en el índice 1 en
linkedList: " + linkedList.get(1));


// Eliminando un elemento en LinkedList

linkedList.remove(0); // Elimina "Dog"

System.out.println("\nElementos después de eliminar en
linkedList:");

for (String animal : linkedList) {
    System.out.println(animal);
}

// Usando directamente ArrayList (Clase)

ArrayList<Integer> myArrayList = new ArrayList<>();

myArrayList.add(10);
myArrayList.add(20);
myArrayList.add(30);
```

```
System.out.println("\nElementos en myArrayList:");

for (Integer number : myArrayList) {

    System.out.println(number);

}

// Usando directamente LinkedList (Clase)
LinkedList<Integer> myLinkedList = new LinkedList<>();

myLinkedList.add(100);
myLinkedList.add(200);
myLinkedList.add(300);

System.out.println("\nElementos en myLinkedList:");

for (Integer number : myLinkedList) {

    System.out.println(number);

}

// Añadiendo elementos al inicio y al final en
LinkedList

myLinkedList.addFirst(50);
myLinkedList.addLast(400);

System.out.println("\nElementos en myLinkedList
después de añadir al inicio y al final:");

for (Integer number : myLinkedList) {

    System.out.println(number);


}

}

}
```

### Salida esperada

yaml

 Copiar código

Elementos en arrayList (ArrayList):

Apple

Banana

Cherry

Elemento en el índice 1 en arrayList: Banana

Elementos después de eliminar en arrayList:

Banana

Cherry

Elementos en linkedList (LinkedList):

Dog

Cat

Rabbit

Elemento en el índice 1 en linkedList: Cat

Elementos después de eliminar en linkedList:

Cat

Rabbit

Elementos en myArrayList:

10

20

30

Elementos en myLinkedList:

100

200

300

Elementos en myLinkedList después de añadir al inicio y al final:

50

100

200

300

400

## Excepciones en Java

Las **excepciones** en Java son eventos que interrumpen el flujo normal de ejecución de un programa. Se producen cuando ocurre un error, como intentar dividir por cero o acceder a un índice inexistente en un array.

Java maneja excepciones mediante la clase base `Exception`. Existen dos tipos principales de excepciones:

- **Excepciones comprobadas (checked):** Deben ser capturadas o declaradas en el método mediante `throws`.
- **Excepciones no comprobadas (unchecked):** Heredan de `RuntimeException` y no es obligatorio capturarlas.

### Tipos comunes de excepciones y ejemplos

#### - `ArithmeticException`

Se lanza cuando ocurre un error aritmético, como dividir un número por cero.

```
try {  
    int resultado = 10 / 0;  
} catch (ArithmeticException e) {  
    System.out.println("Error: No se puede dividir por  
cero.");  
}
```

#### - `NullPointerException`

Se lanza cuando intentas usar un objeto que no ha sido inicializado (es `null`).

```
String texto = null;  
try {  
    System.out.println(texto.length()); // Esto lanzará  
    NullPointerException  
}  
catch (NullPointerException e) {  
    System.out.println("Error: Intento de acceder a un  
objeto nulo."); }  
}
```

### - **ArrayIndexOutOfBoundsException**

Se lanza cuando intentas acceder a una posición que está fuera del rango de un array.

```
int[] numeros = {1, 2, 3};

try {
    System.out.println(numeros[5]); // Índice fuera del rango
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("Error: Índice de array fuera de
límites.");
}
```

### - **NumberFormatException**

Se lanza cuando intentas convertir una cadena que no es un número válido a tipo numérico.

```
String texto = "abc";

try {
    int numero = Integer.parseInt(texto); // Esto lanzará
NumberFormatException
} catch (NumberFormatException e) {
    System.out.println("Error: No se puede convertir el texto
en número.");
}
```

### - **ClassCastException**

Se lanza cuando intentas convertir un objeto a un tipo incompatible.

```
Object obj = "Hola";

try {

    Integer num = (Integer) obj; // Esto lanzará
    ClassCastException

} catch (ClassCastException e) {

    System.out.println("Error: No se puede convertir el objeto
a Integer.");

}
```

### - **FileNotFoundException**

Se lanza cuando no se encuentra el archivo especificado. Es comprobada.

```
import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner;

try {

    File archivo = new File("archivo_que_no_existe.txt");

    Scanner lector = new Scanner(archivo);

} catch (FileNotFoundException e) {

    System.out.println("Error: Archivo no encontrado.");

}
```

## - IOException

Se lanza cuando ocurre un error en la entrada/salida de datos.  
Es comprobada.

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

try {
    BufferedReader lector = new BufferedReader(new
FileReader("archivo.txt"));

    String linea = lector.readLine();

    lector.close();
} catch (IOException e) {
    System.out.println("Error: Problema de E/S.");
}
```

## Manejo de excepciones

- try catch y finally.

El bloque `try` contiene el código que puede lanzar una excepción y `catch` la maneja.

El bloque `finally` se ejecuta siempre, haya o no una excepción. Es útil para liberar recursos.

```
try {
    int resultado = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Error: División por cero.");
} finally {
    System.out.println("Este bloque se ejecuta siempre.");
}
```



## Metodos vistos Java

### Métodos de clase String

- **length()**: longitud de la cadena.

```
String str = "Hola";

int len = str.length(); // 4
```

- **toUpperCase()** / **toLowerCase()**: mayúsculas / minúsculas

```
String upper = str.toUpperCase(); // "HOLA"
```

- **charAt (int index)**: Devuelve el carácter en una posición específica.

```
char c = str.charAt(1); // 'o'
```

- **contains(String seq)**: Comprobar si contiene una secuencia de caracteres.

```
boolean hasHola = str.contains("Hola"); // true
```

- **trim()**: Elimina espacios al inicio y al final.

```
String trimmed = "  Hola  ".trim(); // "Hola"
```

- **replace(String target, String replacement)**: Reemplaza una subcadena.

```
String replaced = "Java".replace("a", "o"); // "Jovo"
```

- **indexOf()**: busca la posición de un carácter o subcategoría de una cadena.

```
int index = cadena.indexOf(String o char);
```

- **HashSet<Character>**: almacena elementos únicos y no permite duplicarlos:

```
HashSet<Character> conjunto = new HashSet<>();
```

- **Scanner** sc = new Scanner ( [system.in](#)) // String cadena = sc.nextLine ();

- **String.endsWith(final)**: final de una cadena.

- **StringBuilder cadena:** manejar y modificar cadenas.

```
public class Main {  
    public static void main(String[] args) {  
        StringBuilder sb = new StringBuilder("Bienvenido");  
        // Agregar texto  
        sb.append(" a Java");  
        System.out.println(sb); // "Bienvenido a Java"  
        // Insertar texto  
        sb.insert(10, " al mundo");  
        System.out.println(sb); // "Bienvenido al mundo a  
Java"  
        // Reemplazar texto  
        sb.replace(0, 10, "Hola");  
        System.out.println(sb); // "Hola al mundo a Java"  
        // Eliminar texto  
        sb.delete(5, 13);  
        System.out.println(sb); // "Hola a Java"  
        // Invertir texto  
        sb.reverse();  
        System.out.println(sb); // "avaJ a aloH"  
    }  
}
```

- **Substring:** retorna una subcadena de la cadena original( la corta)

miércoles, 6 de noviembre de 2024