Fundamentos de programación

1º DAM. José Manuel Bermudo Ancio

Índice

Introducción. ¿Qué es la programación?

Es el arte de escribir instrucciones para que un ordenador realice tareas específicas.

Permite crear aplicaciones y resolver problemas de forma automatizada.

Introducción. Importancia de la programación en la actualidad

Es fundamental en el desarrollo de tecnologías y soluciones para diversos campos.

Impulsa la innovación y la transformación digital en la sociedad.

Introducción. Lenguajes de programación y su uso

Un lenguaje de programación es un conjunto de símbolos y caracteres combinados entre sí, de acuerdo con una sintaxis definida, utilizado para transmitir órdenes o instrucciones al ordenador.

Existen diferentes enfoques:

- Alto nivel/bajo nivel
- Compilados/Interpretados
- Según su uso: scripting/web...

Hay múltiples lenguajes, cada uno con su propósito y enfoque específico.

Ejemplos: Java, Python, C++, JavaScript, entre otros.

Breve introducción a Java

Creado por Sun Microsystems en 1995, ahora propiedad de Oracle.

Lenguaje de programación versátil y robusto.

Lenguaje compilado e interpretado.

Necesita que en el sistema destino exista un JRE.

Plataformas compatibles con Java

Java SE: Plataforma Estándar para aplicaciones de escritorio y desarrollo general.

Java EE: Plataforma Empresarial para aplicaciones empresariales y web.

Java ME: Plataforma Micro Edition para dispositivos móviles y embebidos.

Aplicaciones y casos de uso de Java

Utilizado en aplicaciones web, móviles y de escritorio.

Aplicaciones empresariales, juegos, IoT, entre otros.

Hasta hace poco era la base de todas las apps desarrolladas para Android. Hoy en día, reemplazada en favor de Kotlin.

Uso de Entornos Integrados de Desarrollo (IDE)

Herramientas de software para facilitar la programación.

Unifican la escritura, depuración y compilación.

Importancia de los IDEs

Aumentan la productividad del programador.

Ambiente integrado y amigable.

Demostración de un IDE para Java

Descargaremos e instalaremos 2 IDEs diferenes:

- Eclipse
- IntelliJ IDEA

Por favor, no utilicéis ninguna de las instalaciones para otras materias. Pueden coexistir diferentes versiones si están en diferentes carpetas.

Ejercicio práctico

Crear un nuevo proyecto y comprobar la estructura creada.

Herramientas de desarrollo y depuración

Autocompletado y sugerencias de código.

Resaltado de sintaxis y depuración de errores.

Ventajas de usar un IDE

Facilita la organización y estructuración del código.

Agiliza la escritura con autocompletado y sugerencias.

Permite la depuración gráfica de errores.

Integración de herramientas adicionales (como GIT).

Ejercicios prácticos.

- Escribir un programa simple en Java para imprimir un mensaje en la consola.
- Escribir un programa que pregunte el nombre de una persona y la salude.
- Investiga cuál es el problema del fichero ejemploErrores.java utilizando el “debugger” del IDE.

Git

Git es un sistema de control de versiones ampliamente utilizado en el desarrollo de software. Nos permite realizar un seguimiento de los cambios realizados en nuestros proyectos, lo que facilita la colaboración entre desarrolladores y el mantenimiento del código a lo largo del tiempo.

Git

Para explicar mejor los conceptos de git, he creado un pdf aparte. Buscad en la plataforma “Git -súper- Express.pdf”

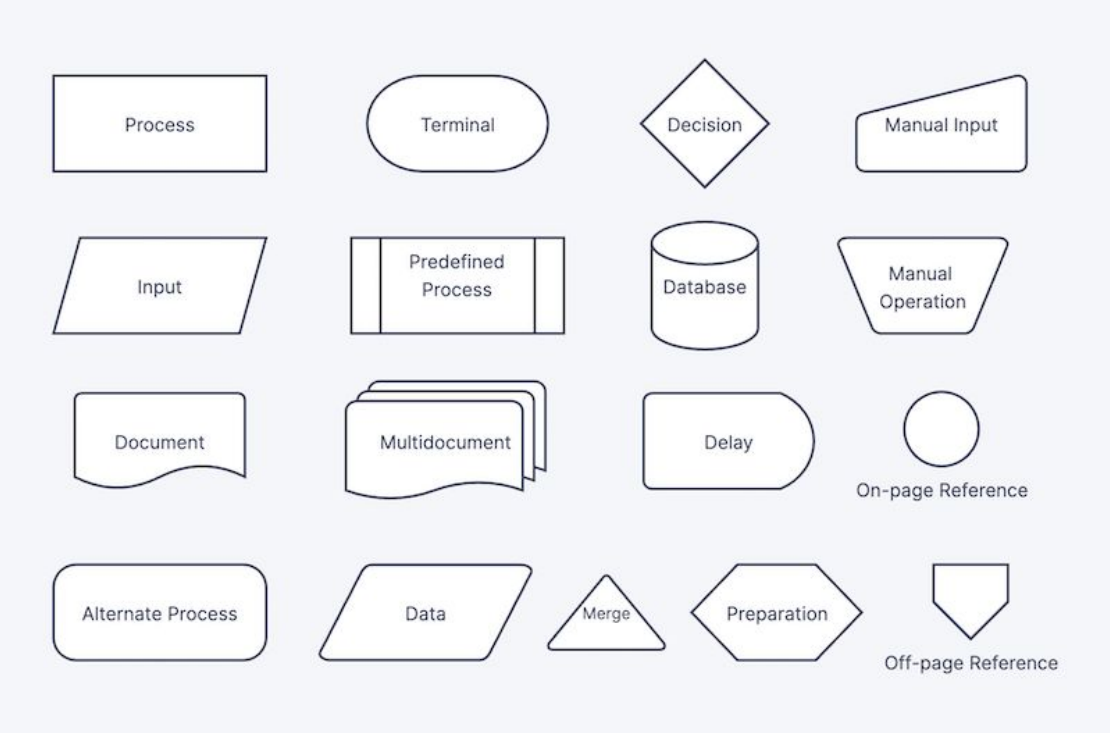
Diagramas de Flujo y PSeInt

Representación gráfica de los pasos de un algoritmo.

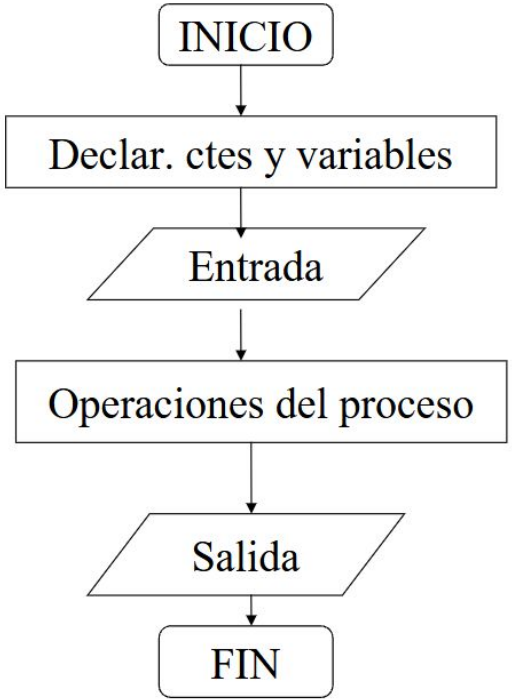
Ayudan a visualizar la secuencia lógica de un programa.

Símbolos: Inicio, Proceso, Decisión, Entrada/Salida, Conector, Fin.

Diagramas de Flujo y PSeInt



Diagramas de Flujo y PSeInt



Importancia de los Diagramas de Flujo

Facilitan el diseño y planificación de un programa.

Permiten identificar errores y mejorar la lógica.

Ayudan en la comunicación entre programadores y equipos de desarrollo.

PSeInt

Herramienta gratuita para escribir algoritmos y diagramas de flujo.

Facilita la comprensión de la lógica de programación.

Permite validar y depurar algoritmos antes de codificar en un lenguaje específico.

Descarga desde aquí.

PSeInt. Ejercicios

1. Introducir un nombre por teclado y saludarlo.
2. Realizar la suma de dos números introducidos.
3. Calcular la resta de dos números.

Crear primero el diagrama y posteriormente, visualizar el código.

Realizar ahora el boletín 1.1

Introducción a la programación en Java

Tipos de datos en Java:

En Java, los tipos de datos determinan qué tipo de valores pueden ser almacenados en las variables.

Las variables de java son **fuertemente tipadas.**

A continuación, veremos los principales tipos de datos que podemos utilizar en Java.

Tipos de datos en java

Tipos Primitivos:

- Los tipos primitivos son los tipos de datos básicos incorporados en Java.
- Se dividen en cuatro categorías principales:
  - Enteros: int, long, short, byte.
  - Decimales: double, float.
  - Carácter: char.
  - Booleano: boolean.

Tipos de datos en java

Enteros:

- **int (32 bits)**: Representa números enteros en el rango de -2,147,483,648 a 2,147,483,647.
- **long (64 bits)**: Permite almacenar números enteros más grandes en el rango de -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807.
- **short (16 bits)**: Representa enteros más pequeños en el rango de -32,768 a 32,767.
- **byte (8 bits)**: Permite almacenar valores enteros más pequeños en el rango de -128 a 127

Tipos de datos en java

Decimales:

- **double (64 bits)**: Almacena números en coma flotante de doble precisión con mayor rango y precisión.

- **float (32 bits)**: Almacena números en coma flotante con menor rango y precisión.

Tipos de datos en java

Carácter:

- **char**: Representa un único carácter Unicode y se define entre comillas simples (' ').

Booleano:

- **boolean**: Representa valores de verdadero (true) o falso (false).

Tipos de datos en java

```
int coordenadaX = 23;

double salario = 2150.35;

long kmsAlSol = 1234565788785L;

float pi = 3.1415f;

boolean encontrado = false;

char letraA = ‘A’;
```

Variables en java

En Java, las variables son contenedores que almacenan valores que pueden cambiar durante la ejecución del programa. A continuación, veremos cómo declarar y utilizar variables en Java.

Declaración de Variables

Para crear una variable, primero debemos declararla especificando su tipo de datos y nombre.

Ejemplo de declaración:

int edad;

(Declaramos una variable de tipo int llamada "edad").

Asignación de Valores

Después de declarar una variable, podemos asignarle un valor utilizando el operador de asignación (=).

Ejemplo de asignación:

edad = 25;

(Asignamos el valor 25 a la variable "edad").

Inicialización de Variables

Podemos declarar y asignar un valor a una variable al mismo tiempo.

Ejemplo de inicialización:

int altura = 180;

(Declaramos una variable de tipo int llamada "altura" y le asignamos el valor 180).

Inferencia de tipo

Podemos crear una variable “sin tipo” y dejar que java lo “infiera” (deduzca) a partir de java 10.

Para ello tenemos la palabra reservada **var**:

var miVariable = 3; //miVariable ha quedado definido como int. Si intentamos cambiarla, obtendremos un error.

Conversiones de tipos

Es posible cambiar el tipo de una variable a otro diferente, siempre que ambos sean compatibles.

En java podemos realizar la conversión de tipos implícita entre tipos numéricos de menor a mayor rango. Por ejemplo de int a float, de float a double, de short a long, etc.

Entre los tipos int y char, la conversión es directa, pues internamente es el mismo tipo de dato.

Conversiones de tipos

Las conversiones implícitas no necesitan nada especial en su creación:

int x = 3;

long y = x; // Al ser una conversión a un tipo de mayor rango no tenemos que poner nada más

Conversiones de tipos

Cuando la conversión sea de un tipo de mayor rango a otro de menor rango, debemos hacer lo que se conoce como “casting”:

```
double x = 6.3;

int y = (int) x;
```

Entre paréntesis estamos diciéndole a la jvm que estamos haciendo a propósito la conversión al tipo señalado, a sabiendas de que podemos estar perdiendo información. En esta caso, por ejemplo, y valdrá 6, pues los tipos int no guardan decimales.

Conversiones de tipos

```
int entero = 3;

double real = 5.2;

entero = real; // error

entero = (int) real; // entero valdrá 5. Explícita

real = entero; // real valdría 5.0. Implícita
```

Conversiones de tipos

El resultado de cualquier expresión es del tipo correspondiente al del operando de mayor jerarquía, en el orden: **double, float, int**

Ejemplo:

int i = 10, k;

double j = 2.0;

k= i/j; //daría error. El resultado de la expresión es un double que no se puede asignar a un entero

Conversiones de tipos

En Java, una variable entera puede asignarse a un carácter y viceversa. La conversión se realizará a través del código ASCII. Ejemplos:

```
int entero = 65;

char caracter;

caracter = entero; // El carácter tomará el valor ‘A’

caracter = ‘Z’;

entero = caracter; // El entero tomará el valor 90
```

Conversiones de tipos

Cuidado: La división entre variables enteras da un entero, aunque el resultado se asigne a un double o float.

Para que la división entre enteros tenga un resultado con decimales tendrá que hacerse un casting.

Conversiones de tipos

Suponer la variable x definida como **double**

x = 10/4; // x toma el valor 2.0

x = 10.0/ 4; // x toma el valor 2.5

x = (double) 10 /4; // x toma el valor 2.5*

x = (double) (10/4); // x tomaría el valor 2.0

* Observa que en la tercera instrucción se está realizando un casting al número 10, por lo que pasa a tratarse como un double

Reglas para Nombres de Variables

Los nombres de variables deben comenzar con una letra, un guión bajo (_) o el símbolo del dólar ($), aunque por regla general, comienzan siempre por una letra.

No pueden comenzar con un número o contener espacios en blanco.

Java distingue entre mayúsculas y minúsculas en los nombres de las variables.

Utilizaremos siempre la nomenclatura **camelCase**.

Ámbito de las Variables

El ámbito de una variable es la parte del programa donde puede ser accesible.

Las variables declaradas dentro de un bloque de código sólo son accesibles dentro de ese bloque.

Las variables declaradas fuera de todos los bloques son accesibles en todo el método.

Ámbito de las Variables

```
public class MiClase {
        boolean variableAccesibleEnTodaLaClase;
        public static void main (String[] args) {
                int variableAccesibleEnTodoElMetodo;
                while (int i = 0; i < 3; i++) {
                        // i únicamente es accesible dentro del bloque while
                }
        }
}
```

Uso de Constantes y Literales en Java

En Java, las constantes y literales son valores fijos que se utilizan en el código para representar datos que no cambian durante la ejecución del programa. Aprendamos cómo declarar y utilizar constantes y literales en Java.

Constantes

Una constante es una variable cuyo valor no puede modificarse una vez que se le asigna un valor inicial.

Se declaran usando la palabra clave final, y su nombre se escribe en mayúsculas para indicar que es una constante.

Ejemplo de declaración:

final double PI = 3.14159;

(El valor de PI no puede modificarse posteriormente).

Literales

Un literal es una representación directa de un valor en el código fuente.

Los literales numéricos pueden ser enteros (por ejemplo, 10), decimales (por ejemplo, 3.14) o en notación científica (por ejemplo, 1.5e6).

Los literales de texto se definen entre comillas (por ejemplo, "Hola, mundo!").

Ejemplo de uso:

```
if (edad > 25)
```

Operadores y Expresiones en Java

Los operadores son símbolos especiales que se utilizan para realizar operaciones en Java. Aprendamos sobre los operadores más comunes y el orden de prioridad de los mismos.

Operadores Aritméticos

- ‘+’ Suma: Realiza la suma de dos valores.
- ‘-’ Resta: Realiza la resta de dos valores.
- ‘*’ Multiplicación: Realiza la multiplicación de dos valores.
- ‘/’ División: Realiza la división de dos valores.
- ‘%’ Módulo: Devuelve **el resto de la división entera** de dos valores.

Operadores de Asignación

- ‘=’ Asignación: Asigna un valor a una variable.
- ‘+=’ Suma y asigna: Suma un valor a una variable y guarda el resultado en la misma variable.
- ‘-=’ Resta y asigna: Resta un valor a una variable y guarda el resultado en la misma variable.
- ‘*=’ Multiplicación y asigna: Multiplica una variable por un valor y guarda el resultado en la misma variable.
- ‘/=’ División y asigna: Divide una variable por un valor y guarda el resultado en la misma variable.

Operadores de Asignación

```
int a = 3;

a += 3; // ahora a vale 6

a -= 1; // ahora a vale 5

a *= 6 // ahora a vale 30

a /= 5 // ahora a vale 6
```

Operadores de Comparación

- ‘==’ Igual a: Comprueba si dos valores son iguales.
- ‘!=’ Diferente de: Comprueba si dos valores son diferentes.
- ‘>’ Mayor que: Comprueba si el valor de la izquierda es mayor que el de la derecha.
- ‘<’ Menor que: Comprueba si el valor de la izquierda es menor que el de la derecha.
- ‘>=’ Mayor o igual que: Comprueba si el valor de la izquierda es mayor o igual que el de la derecha.
- ‘<=’ Menor o igual que: Comprueba si el valor de la izquierda es menor o igual que el de la derecha

Operadores Lógicos

- ‘&&’ AND lógico: Devuelve true si ambas expresiones son verdaderas.
- ‘||’ OR lógico: Devuelve true si al menos una de las expresiones es verdadera.
- ‘!’ NOT lógico: Devuelve el valor opuesto de la expresión (si es verdadera, devuelve false; si es falsa, devuelve true).

Orden de Prioridad de los Operadores (de mayor a menor)

- Parentesis ()
- Operadores de signo: +, - (no confundir con suma y resta)
- Operadores Unarios: ++, --
- Operadores Aritméticos: *, /, %
- Operadores de Suma y Resta: +, -
- Operadores de Comparación: ==, !=, >, <, >=, <=
- Operadores Lógicos: &&, ||, !

Preincremento y Postincremento en Java

El preincremento (++variable) y postincremento (variable++) son operadores que incrementan el valor de una variable numérica en una unidad. Sin embargo, su comportamiento difiere ligeramente.

Preincremento y Postincremento en Java

- Preincremento (++variable): Primero incrementa el valor de la variable y luego utiliza el nuevo valor en la expresión en la que se encuentra.
- Postincremento (variable++): Utiliza el valor original de la variable en la expresión y luego incrementa su valor en una unidad.

Debemos tener mucho cuidado con la evaluación:

Preincremento y Postincremento en Java

int a = 5;

if (a++ > 5) : esta sentencia evaluará a falso, porque a vale 5 en el momento de evaluarse. Sin embargo:

if (++a > 5) : esta sentencia sí evaluará a true, porque en el momento de evaluarse a se ha pre-incrementado y vale 6.

Preincremento y Postincremento en Java

Lo explicado con el preincremento y postincremento es aplicable al decremento:

a = 3;

a--; //ahora a vale 2

Ejercicios

Hora de practicar con el boletín 1.2. Hacer el bloque de variables.

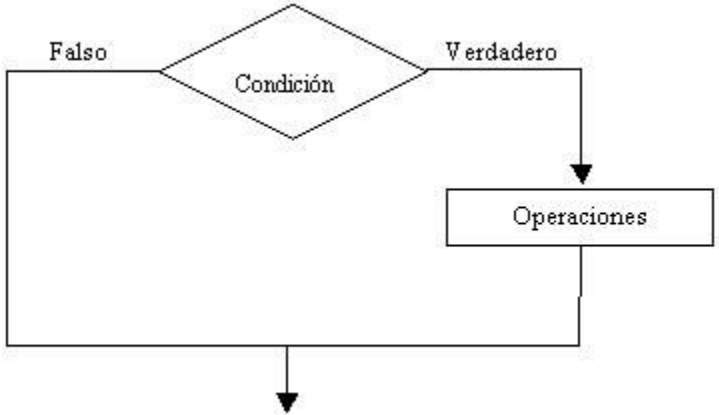
Control de flujo

El control de flujo en Java nos permite tomar decisiones y repetir acciones según ciertas condiciones. Esto nos permite crear programas más complejos y adaptativos.

Veamos algunas de las estructuras de control de flujo que ofrece Java

Control de flujos: estructuras de selección

If-else: Nos permite tomar decisiones entre dos opciones o más.



Estructuras de selección

Podemos tener múltiples ramas:

si (condición):

    hacer algo

sino si (sub-condición):

    hacer otra cosa

sino si (sub-condición):

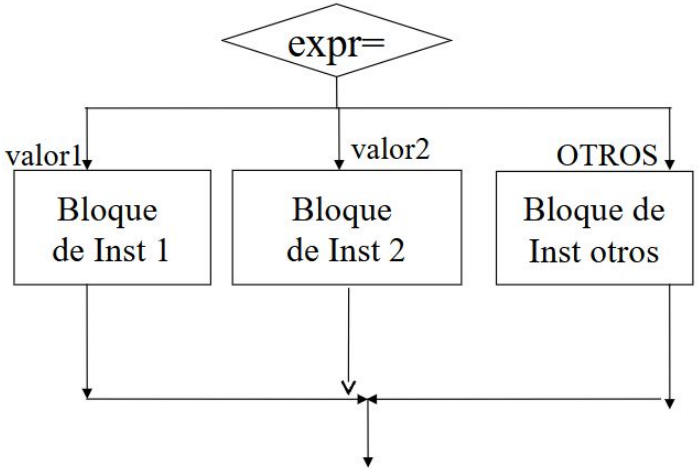
    hacer otra cosa distinta

en otro caso:

    hacer algo (o no)

Estructuras de selección

Switch: Es útil cuando necesitamos tomar decisiones basadas en el valor de una variable.



Estructuras de selección

¿Qué tipo de variables podemos usar en un switch?

- char, int (desde las versiones iniciales)
- String y enums (desde java 8)

Estructuras de selección

Estructura de un switch.

Estructura básica:

```
switch (variable){
     case valor1:
          acciones;
          break;
     case valor2:
          acciones;
          break;
     default:
          acciones;
}
```

Estructuras de selección

Expresión (podemos devolver valores con yield):

```
String a = switch (operador) {
    case '?':
        yield "Signo de interrogación";
    case '*':
        yield "Asterisco";
    default:
        yield "Desconocido";
};
```

Estructuras de selección

Expresión con operador flecha:

```
String a = switch (operador) {
    case '?' -> "Signo de interrogación";
    case '*' -> {
        int z = 2;
        yield "Asterisco";
    };
    default -> "Desconocido";
};
```

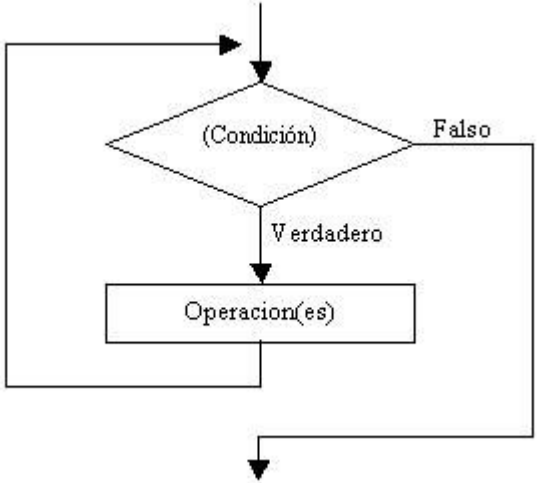
Estructuras de selección: ejercicios

if-else: boletín 1.1. Bloque de ejercicios if-else.

switch: boletín 1.1. Bloque de ejercicios switch.

Control de flujos: estructuras de repetición

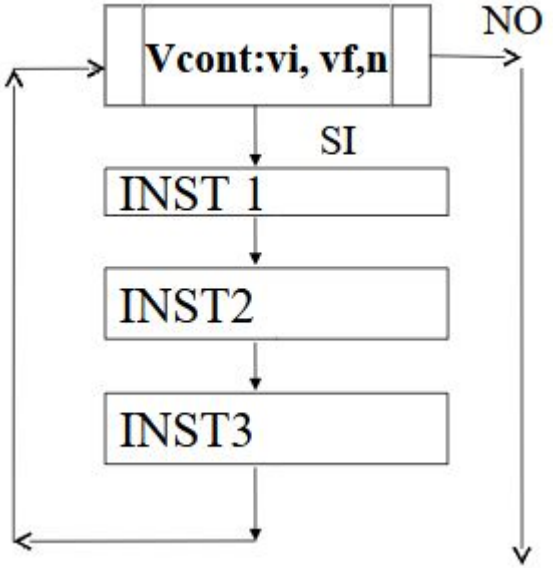
Las estructuras de repetición nos permiten ejecutar un bloque de código múltiples veces.



Bucle for

El bucle **for** es útil cuando sabemos cuántas veces queremos repetir una acción.

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



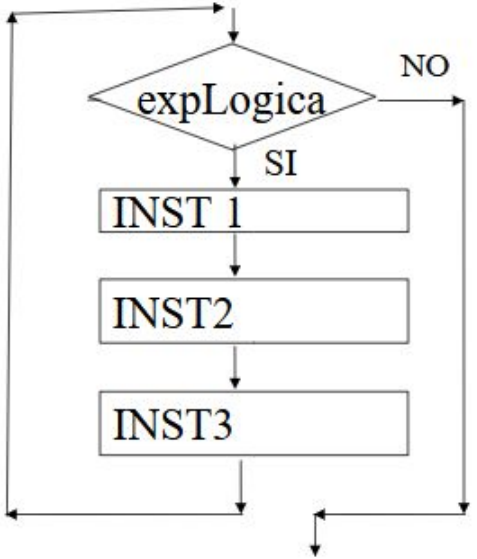
Bucle for

Ejercicio: Escribe un programa que sume los números del 1 al 10 usando un bucle for.

Bucle while

El bucle **while** repite una acción mientras una condición sea verdadera.

```
String input = "";
while (!input.equals("salir")) {
        System.out.println("Escribe 'salir' para terminar.");
        // Código para tomar la entrada del usuario
}
```

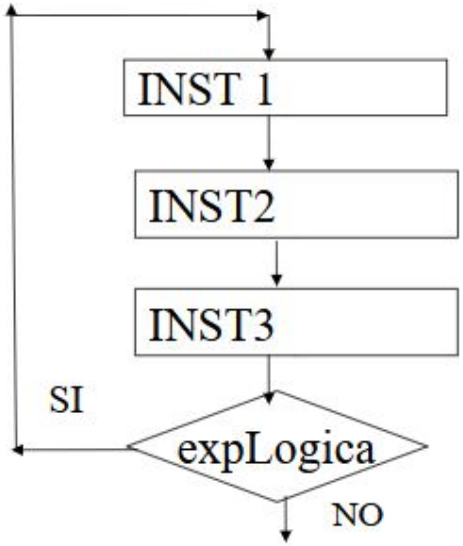


Bucle while

Ejercicio: Escribe un programa que encuentre el primer divisor de un número usando un bucle while.

Bucle do-while

El bucle **do-while** asegura que el código dentro del bucle se ejecute al menos una vez.




```
int opcion;
do {
        System.out.println("Selecciona una opción válida.");
        // Código para tomar la opción del usuario
} while (opcion < 1 || opcion > 3);
```

Bucle do-while

Escribe un programa que solicite una contraseña válida usando un bucle do-while

Bucle for extendido

El bucle **for extendido** simplifica el recorrido de arrays y colecciones.

Ejemplo

```
String[] nombres = {"Alicia", "Pedro", "Carlos"};
for (String nombre : nombres) {
        System.out.println(nombre);
}
```

Bucle for extendido

Ejercicio (resuelto): Escribe un programa que sume todos los elementos de un array de enteros usando el bucle for extendido.

Solución en la próxima página

Bucle for extendido

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

int acum = 0;

for (int n: miArray) {

      acum += n;

}

System.out.println(acum);
```

Métodos y funciones

Un método es una colección de declaraciones que realizan una operación específica. En Java, los métodos son la forma en que encapsulamos comportamientos y acciones para nuestros objetos.

Sintaxis de un método

La sintaxis general de un método en Java es la siguiente:

```
[Modificador de acceso] [Tipo de retorno]
[nombre del método] ([lista de parámetros])

{

   [cuerpo del método]

}
```

Sintaxis de un método

Los componentes de un método son:

- Modificadores de acceso (public, private, protected)
- Tipo de retorno (int, double, String, etc.)
- Nombre del método
- Lista de parámetros (opcional)
- Cuerpo del método (las sentencias que se ejecutan cuando se llama al método)

Métodos sin retorno

Un método puede no devolver nada, en cuyo caso usamos la palabra clave **void**. Por ejemplo, el método

```
public void saludar() {
    System.out.println("¡Hola Mundo!");
}
```

simplemente imprime un saludo, pero no devuelve ningún valor.

Métodos con retorno

Un método puede devolver un valor. Por ejemplo,

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

toma dos números enteros como parámetros y devuelve su suma.

Parámetros de un método

Los parámetros son variables que se pasan a un método cuando se invoca. Por ejemplo, en

```
public int suma(int a, int b)
```

a y b son los parámetros.

Invocación de un método

Para utilizar un método, debemos llamarlo o invocarlo. Por ejemplo, si tenemos un método

public void saludar()...

podemos invocarlo así:

saludar();

Sobrecarga de métodos

La sobrecarga de métodos ocurre cuando en una misma clase se declaran varios métodos con el mismo nombre, pero con diferentes parámetros. Por ejemplo, podríamos tener dos métodos llamados suma pero uno que toma dos números enteros y otro que toma tres:

```
public void suma(int a, int b) {...
```

```
public void suma(int a, int b, int c) {...
```

```
public void suma(int x, int y) {...
```
//**Error**, ya existe un método con el nombre suma y que acepta dos enteros, independientemente de su nombre

Conclusiones

- La programación impulsa la innovación y transformación digital.
- Java es un lenguaje robusto y versátil para diversas aplicaciones.
- Los IDEs aumentan la productividad en el desarrollo.
- Los diagramas de flujo son una herramienta valiosa en la programación.
- Java es un lenguaje fuertemente tipado, requiriendo una definición precisa de las variables.
- Las estructuras de control permiten programas dinámicos.
- Los métodos son una pieza esencial de la programación en Java. Nos permiten encapsular comportamiento y crear código más modular, reutilizable y fácil de leer.

Fuentes

- Oracle. "The Java™ Tutorials." https://docs.oracle.com/javase/tutorial/.
- Hackr.io - "10 Best Java IDEs For Real Developers" - https://hackr.io/blog/best-java-ides
- Stack Overflow - "Why should I use an IDE" - https://stackoverflow.com/questions/208193/why-should-i-use-an-ide
- Zenflowchart - “Símbolos de Diagrama de Flujo: Una Guía Completa” - https://www.zenflowchart.com/diagrama-de-flujo-simbologia
- Oracle. “Primitive Data Types” - https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html
- Medium.com. “The Evolution Of Switch Statement From Java 7 to Java 17” https://medium.com/@javatechie/the-evolution-of-switch-statement-from-java-7-to-java-17-4b5eee8d29b7
- Oracle. “Java Language Changes” - https://docs.oracle.com/en/java/javase/20/language/java-language-changes.html