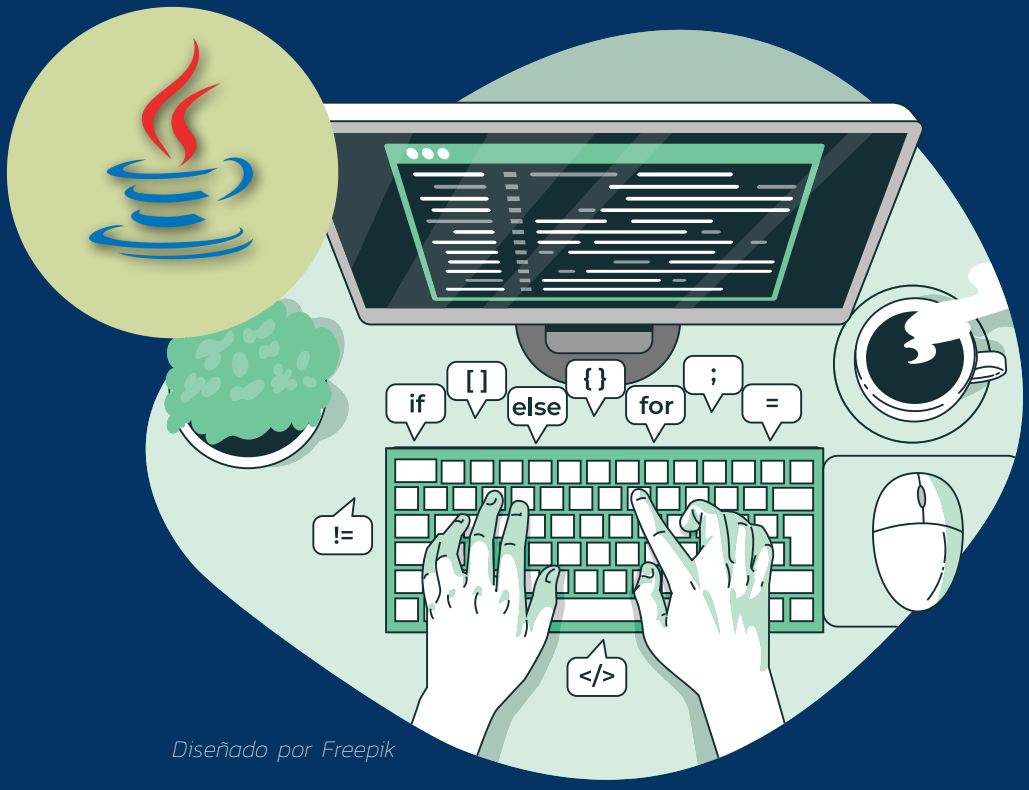
Gobierno del CHACO | Ministerio de la Producción y el Desarrollo Económico Sostenible

INFORMATORIO

ETAPA 3: ESPECIALIZACIONES

JAVA



Diseñado por Freepik

Apunte Nº 1

PROGRAMACIÓN IMPERATIVA

1

Año 2025

Gobierno del CHACO | Ministerio de la Producción y el Desarrollo Económico Sostenible

INFORMATORIO

¿Qué es un paradigma de programación?

Los paradigmas de programación son una forma de pensar y estructurar el código, y cada lenguaje de programación tiene su propio paradigma.

Algunos paradigmas se ocupan principalmente de las implicancias para el modelo de ejecución del lenguaje, como permitir efectos secundarios o si la secuencia de operaciones está definida por el modelo de ejecución. Otros paradigmas se refieren principalmente a la forma en que se organiza el código, como agrupar un código en unidades junto con el estado que modifica el código. Sin embargo, otros se preocupan principalmente por el estilo de la sintaxis y la gramática.

En específico, Java tiene como paradigma de programación a la Programación Orientada a Objetos. Sin embargo, con el objetivo de conocer otras metodologías de programación, Java nos permite experimentar otros paradigmas como son el imperativo y el estructurado.

¿Qué es la programación imperativa?

La programación imperativa es un paradigma de programación de software que se centra en describir paso a paso cómo realizar una tarea. En este enfoque, el código se organiza en una secuencia de instrucciones que el programa ejecuta de manera ordenada.
En la programación imperativa, el desarrollador tiene el control total sobre el flujo del programa y puede manipular directamente los datos y el estado del sistema.

Algunos los pilares fundamentales de la programación imperativa son:

- Las instrucciones se ejecutan secuencialmente, una después de la otra.
- Los programas imperativos hacen un uso extensivo de variables, sus estados y las estructuras de datos mutables.

Ejemplos de lenguajes imperativos: C, Python y Java

En el siguiente ejemplo veremos de qué modo la programación imperativa es lo más sencillo, son pasos, líneas tras línea le decimos a la máquina que debe hacer sin tareas complejas. Sin embargo, la programación imperativa nos limita considerablemente a hacer programas sencillos, muy cortos a lo que nosotros podríamos desear crear.

```java
int value1 = 5;

System.out.println(value1 );


double value3 = 10 * 20.5d;
System.out.println(10 * 20.5d);

int value4 = 1 + 2 * 4;
System.out.println(value4 + value3);
```

imagen 1

Tipos de datos

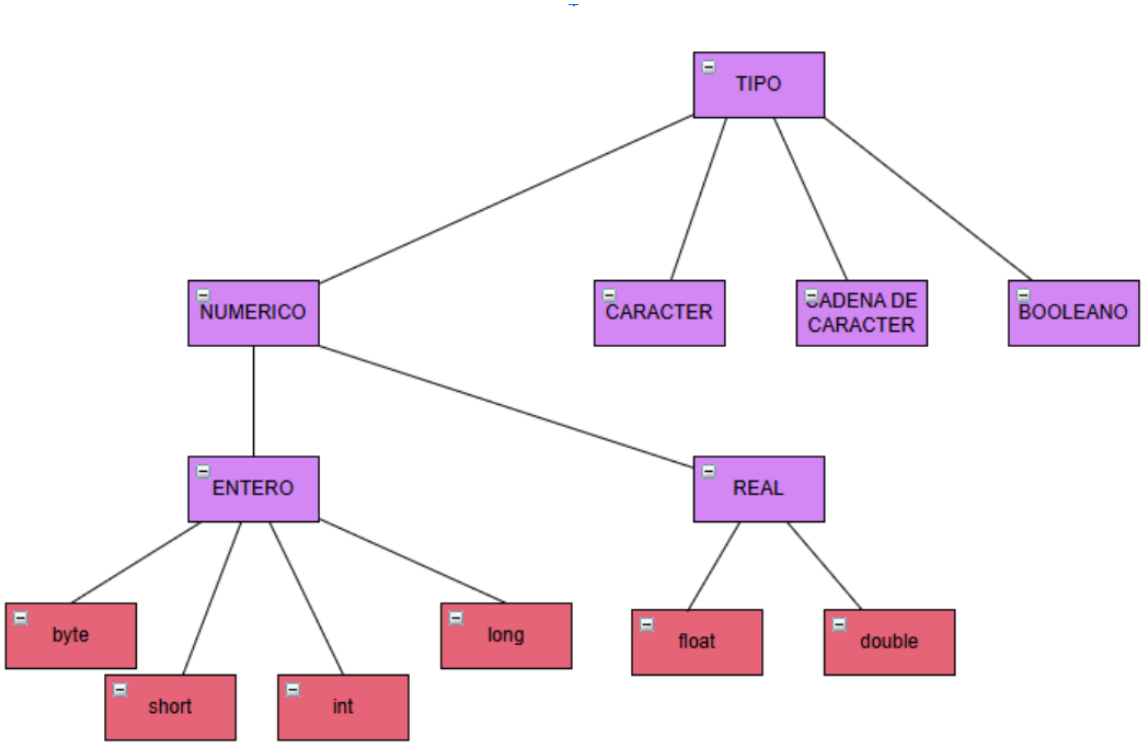


imagen 2

Tipo de datos primitivos

Los tipos primitivos como su nombre lo da a entender, son los tipos de datos más básicos y simples del sistema de tipos de Java y por ello es bastante fácil usarlos.

Tipo primitivo char

Este tipo de dato permite representar caracteres aislados, es decir, por medio de un único char no podemos representar palabras completas sino más bien letras individuales. Por ejemplo, entonces la palabra "carácter" estaría conformada por un total de ocho chars 'c', 'a', 'r', 'a', 'c', 't', 'e' y 'r'. Un char tiene un peso fijo (independientemente de la letra) de 16 bit.

Tipo boolean

El tipo de dato primitivo boolean nos permite representar valores lógicos o booleanos (falso y verdadero). Una variable boolean, solo puede tomar dos posibles valores false (falso) o true (verdadero), este tipo de variables son especialmente usadas para evaluar condiciones en las cuales es necesario conocer el valor de verdad de cierta operación lógica. Este tipo de datos representa un bit de información.

Tipo de dato byte

Este tipo de datos representa pequeños números enteros (8 bit), puede contener números entre -128 y 127.

El tipo short

Este es usado para representar números enteros más grandes que byte (aunque no demasiado) tiene un peso de 16 bit y varía entre -32768 y 32767.

Tipo primitivo char

```
char unChar = 'a';
System.out.println("unChar = " + unChar);
```

imagen 3

Tipo boolean

```
boolean miVerdadero = true;
boolean miFalse = false;
```

imagen 4

Tipo de dato byte

```
//Primitivo de Byte
byte minValue = -128;
byte maxValue = 127;
```

imagen 5

Tipo de dato short

```
//Primitivo de Short
short minValueShort = -32768;
short maxValueShort = 32767;
```

imagen 6

Tipo primitivo int

Este tipo de dato es uno de los más populares entre una gran variedad de programadores, pues generalmente cuando se piensa en un número entero inmediatamente se coloca la variable como tipo int, sin embargo generalmente int suele ser más grande de lo que llegamos a necesitar, desperdiciando así algunos bits en memoria. Un tipo int tiene un peso de 32 bit con signo, es decir va desde -2 a la 31 hasta 2 a la 32, aproximadamente desde -2.147'483.648 hasta 2.147'483.647.

El tipo long

El tipo primitivo long es usado para representar números enteros realmente grandes, del orden de -2 a la 63 hasta 2 a la 64. Vemos entonces que un long tiene un tamaño de 64 bit y varía aproximadamente desde -9.223''372.036.854'775.808 hasta 9.223''372.036.854'775.807.

Tipo primitivo float

El tipo float es quizás el segundo más usado, es útil cuando queremos representar números decimales, como por ejemplo resultados de divisiones, raíces o similares. Son también grandes (su tamaño es de 32 bit) y cubren un gran rango de posibles valores.

Tipo de dato double

Este tipo es el tipo de dato numérico más grande, con este podemos representar casi cualquier número que necesitemos, es demasiado improbable que necesitemos un número que supere la capacidad de éste (aunque existen unos pocos casos en los que sí), tienen un peso de 64 bit.

En este último ejemplo *(imagen 10)*, vemos que guardamos un float dentro de un double, esto es porque double es más grande como tipo de dato que float.

Tipo de dato int

```
int minValueInt = -2147483648;
int maxValueInt = 2147483647;
```

imagen 7

Tipo de dato long

```
long valueLong = 2147483647;
```

imagen 8

Tipo de dato float

```
float miMaximoValorDeFloat = 3.40f;
```

imagen 9

Tipo primitivo double:

```
double valorConFloat = 3.40f;
```

imagen 10

Clases Wrapper

Java provee clases para brindar la funcionalidad que los tipos de datos primitivos (int, double , char, etc) no pueden proveer (justamente) por ser solo tipos de datos primitivos. A estas clases se las suele denominar wrappers (envoltorios) y permiten, entre otras cosas, realizar conversiones entre cadenas y números, obtener expresiones numéricas en diferentes bases, etcétera.

Ejemplo de código:

```
// --- operaciones con el tipo int --- int i = 43;
// convierto de int a String
String sInt = Integer.toString(i);
// convierto de String a int
int i2 = Integer.parseInt(sInt);
// --- operaciones con el tipo double --- double d = 24.2;
// convierto de double a String
String sDouble = Double.toString(d);
// convierto de String a double
double d2 = Double.parseDouble(sDouble);
```

imagen 11

A continuación se presenta una tabla a modo de resumen:

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

imagen 12

Numéricos

En los tipos de datos numéricos, encontramos los tipos enteros y los valores reales. En Java tenemos las clases Wrapper de cada tipo de dato primitivo (byte, short, int y long), las cuales son Byte, Short, Integer y Long.

Operadores

Los operadores son funciones matemáticas, las cuales se clasifican en tres tipos:

Tipos de operadores

Prefija -X (<operador><expresión>)
Infija X+1 (<expresión><operador><expresión>)
Sufija X! (<expresión><operador>)

Operadores aritméticos

Operador	Descripción
+	suma
-	resta
*	multiplicación
/	división
%	módulo
+=	acumulador
-=	restador
*=	multiplicador
/=	divisor

imagen 13

Operadores lógicos

Operador	Descripción
&&	and binario
||	or binario
!	not

imagen 14

Operadores relacionales

Operador	Descripción
==	igual
!=	distinto
>	mayor que
<	menor que
>=	mayor o igual que
<=	menor o igual que

imagen 15

Operadores lógicos de bit

Operador	Descripción
&	and binario
|	or binario

imagen 15

Operadores lógicos de desplazamiento de bit

Operador	Descripción
<<	desplazamiento a izquierda
>>	desplazamiento a derecha
>>>	desplazamiento a derecha incluyendo el bit de signo

imagen 16

Como java analiza qué operación hacer primero?

La precedencia de operadores en Java es el orden en el que se evalúan los operadores en una expresión. Es un concepto fundamental en Java que todo programador debe comprender.

Comprender la precedencia de los operadores permite escribir expresiones más precisas y evitar errores lógicos en el código.
Para alterar el orden de evaluación, se pueden utilizar paréntesis. La expresión dentro de los paréntesis se evalúa primero.
Java tiene en memoria la siguiente tabla que vemos a continuación, en la cual en orden de prioridad (de arriba hacia abajo) java comienza a resolver los operadores que tengamos en una sentencia de código:

Comprender la precedencia de los operadores permite escribir expresiones más precisas y evitar errores lógicos en el código.

Prior.	Operador	Tipo de operador	Operación
1	++ -- +, - ~ ¡	Aritmético Aritmético Aritmético Integral Booleano	Incremento previo o posterior (unario) Incremento previo o posterior (unario) Suma unaria, Resta unaria Cambio de bits (unario) Negación (unario)
2	(tipo)	Cualquiera	
3	*, /, %	Aritmético	Multiplicación, división, resto
4	+, - +	Aritmético Cadena	Suma, resta Concatenación de cadenas
5	<< >> >>>	Integral Integral Integral	Desplazamiento de bits a izquierda Desplazamiento de bits a derecha con inclusión de signo Desplazamiento de bits a derecha con inclusión de cero
6	<, <= >, >= instanceof	Aritmético Aritmético Objeto, tipo	Menor que, Menor o igual que Mayor que, Mayor o igual que Comparación de tipos
7	== ¡= == ¡=	Primitivo Primitivo Objeto Objeto	Igual (valores idénticos) Desigual (valores diferentes) Igual (referencia al mismo objeto) Desigual (referencia a distintos objetos)
8	& &	Integral Booleano	Cambio de bits AND Producto booleano
9	^ ^	Integral Booleano	Cambio de bits XOR Suma exclusiva booleana
10	| |	Integral Booleano	Cambio de bits OR Suma booleana
11	&&	Booleano	AND condicional
12	||	Booleano	OR condicional
13	? :	Booleano, cualquiera, cualquiera	Operador condicional (ternario)
14	= *=, /=, %= +=, -= <<=, >>= >>>= &=, ^=, |=	Variable, cualquiera	Asignación Asignación con operación

imagen 17

Variables y constantes

Una **variable** es un espacio de memoria. Cada variable tiene un tipo, nombre y un valor que puede cambiar.

El nombre de la variable se escribe con **lowerCase**.

Una **constante** es un espacio de memoria. Cada variable tiene un tipo, nombre y un valor que no puede cambiar. Cada constante se define con final.

El nombre de la constante se escribe con **SCREAMING_SNAKE_CASE**.

Nombre válidos para variables y constantes:

- Se puede utilizar cualquier combinación de letras, números, $ y _
- No se pueden utilizar palabras reservadas (Keywords) como identificador.
- No puede comenzar con caracter numérico.

Ejemplos :

- int_1 = 10 **// OK**
- char break; **//error**
- int 3aj; **//error**
- float car.t; **//error**

Tipos de conversiones

PROMOCIÓN

Transforma un dato de un tipo a otro con el mismo o mayor espacio de memoria para almacenar un dato.
Ejemplo:

```
//Conversion de tipo promocion
int value1 = 5;
```

imagen 18

CONTRACCIÓN

Transforma un dato de un tipo a otro de menor espacio en memoria para almacenar dato con la consecuente posible pérdida de información.
Ejemplo:

```
//Conversion de tipo contraccion
int value2 = (int)5.69f; //Cast
System.out.println(value2);
```

imagen 19

CONVERSIÓN IMPLÍCITA

Cuando se combinan dos operandos de distinto tipo, se convierte el de menor precisión al de mayor precisión.
Ejemplo:

```java
//Conversion de tipo implicita
double value3 = 10 * 20.5d;
System.out.println(10 * 20.5d);
```

imagen 20