Introducción al lenguaje

www.devtalles.com

Origen de Java: de Oak a Java

Desarrollado por Sun Microsystems en el año 1995 y está basado en C++.

Inicialmente, comenzó a desarrollarse en 1991 para tarjetas inteligentes y sintonizadores de TV.

Integrado por un equipo de 13 personas al mando de James Gosling.

www.devtalles.com

Origen de Java: de Oak a Java

Inicialmente llamado Oak, pero, finalmente, fue renombrado a Java.

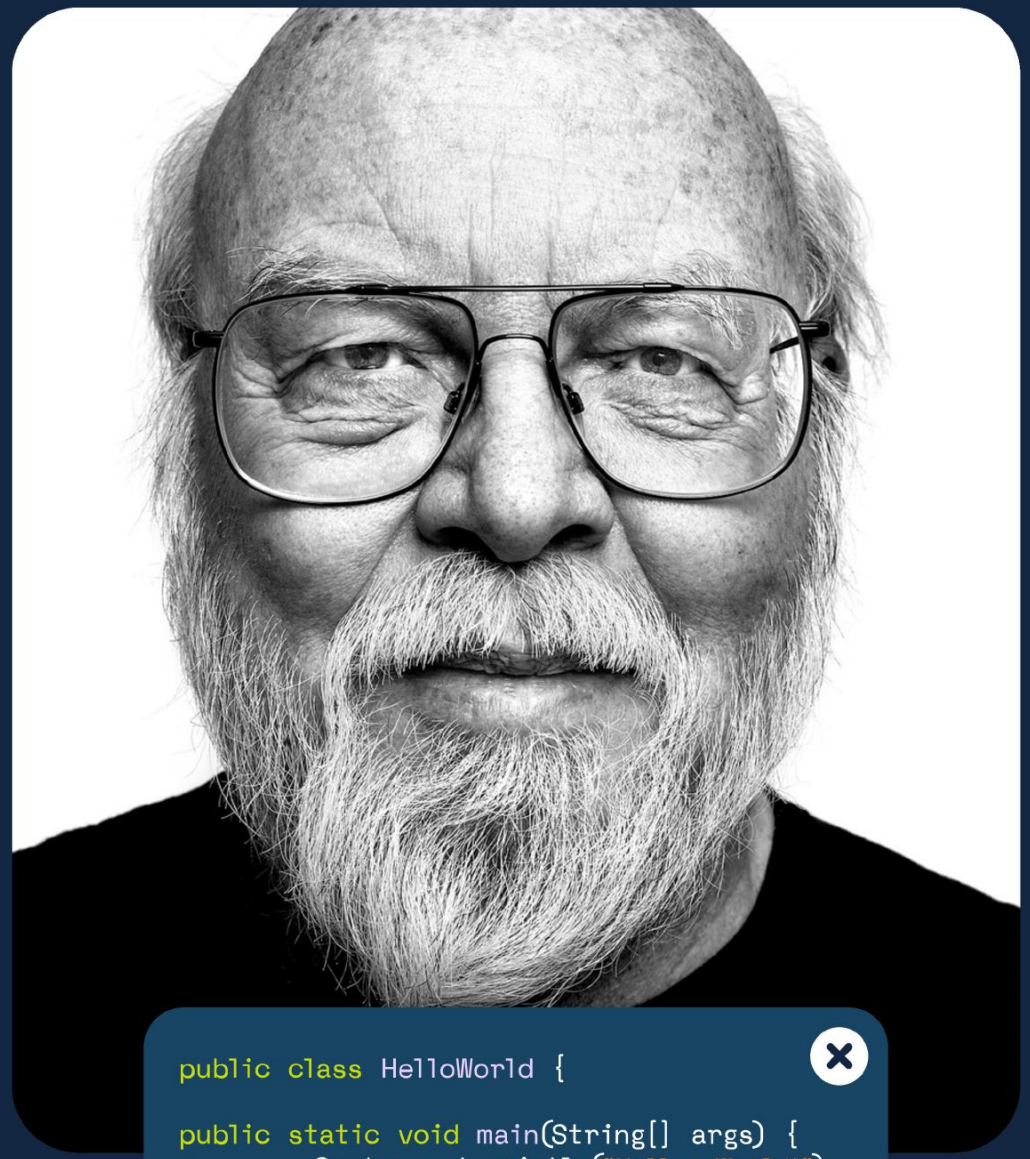
Lanzado públicamente y oficial en 1996 incluyendo Soporte web con el navegador HotJava.

Luego en 1998 aparecen las primeras versiones de Java EE (J2EE 1.0).

www.devtalles.com

"¡Escríbelo una vez, ejecútalo donde quieras!"

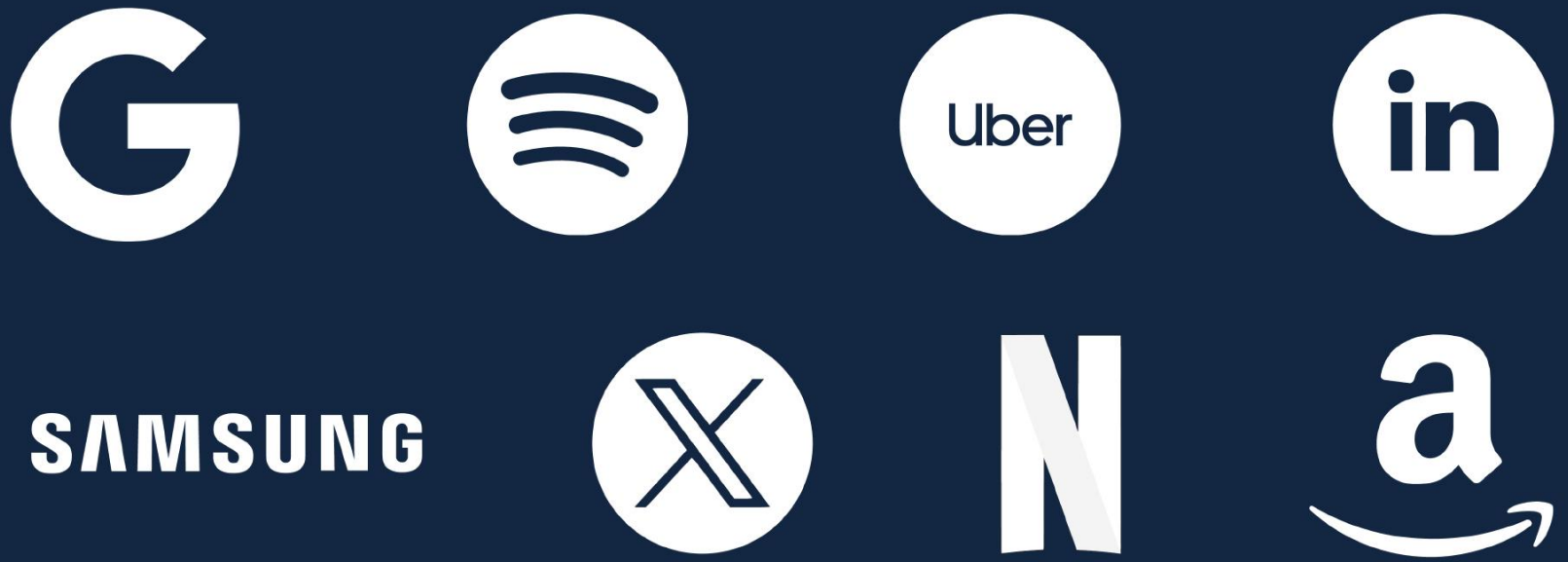
James Gosling



```java
public class HelloWorld {

public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

www.devtalles.com

Empresas que usan Java



www.devtalles.com

Independiente de la plataforma

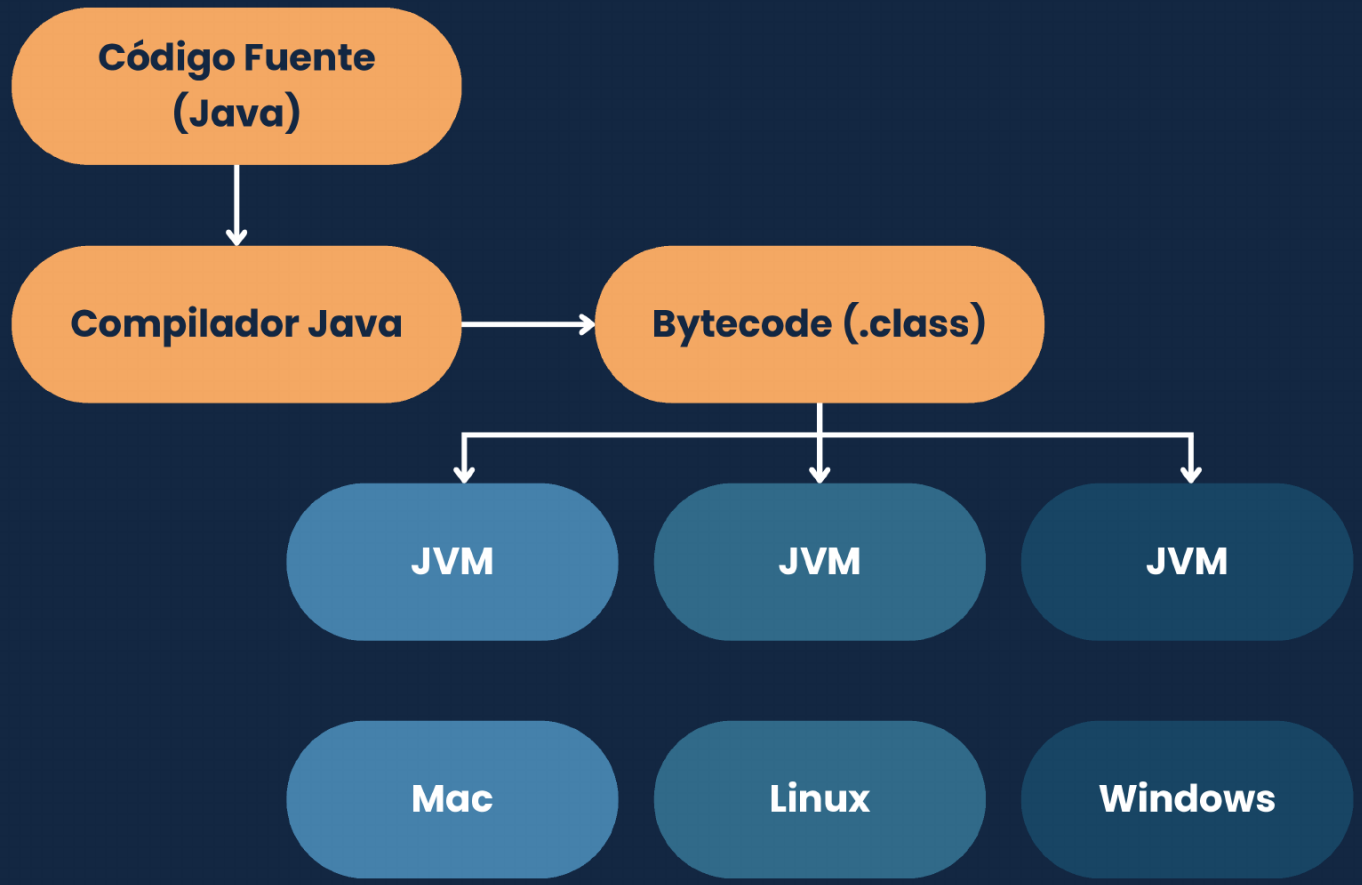
La idea de Gosling era que el lenguaje Java fuera **multiplataforma**.

Un lenguaje independiente del sistema operativo con un entorno de ejecución llamado JVM (máquina virtual de Java).

www.devtalles.com

El Bytecode

El bytecode lo crea el compilador de Java (javac). Javac como un traductor intermedio, toma el código fuente en Java (.java), lo convierte en bytecode y lo deja listo para que la JVM lo ejecute, sin importar el S.O. (Mac, Linux, Windows).



www.devtalles.com

Características

- Muchos tipos de aplicaciones (consola, desktop, web, backend, etc).
- Multiplataforma.
- Programación orientada a objetos.
- Lenguaje tipado.

www.devtalles.com

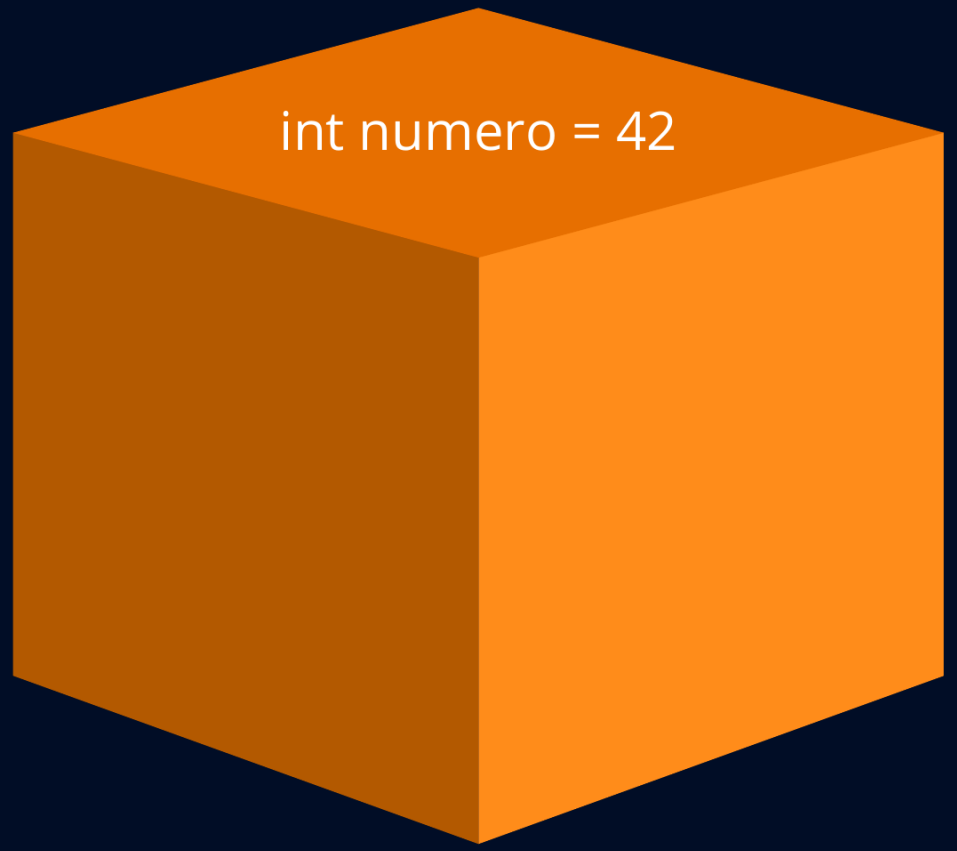
Fundamentos de Programación

www.devtalles.com

Gabriel Chaldú

Variables

Espacio en memoria donde se almacena un valor que puede cambiar durante la ejecución del programa.



Variables

DECLARACIÓN DE VARIABLE

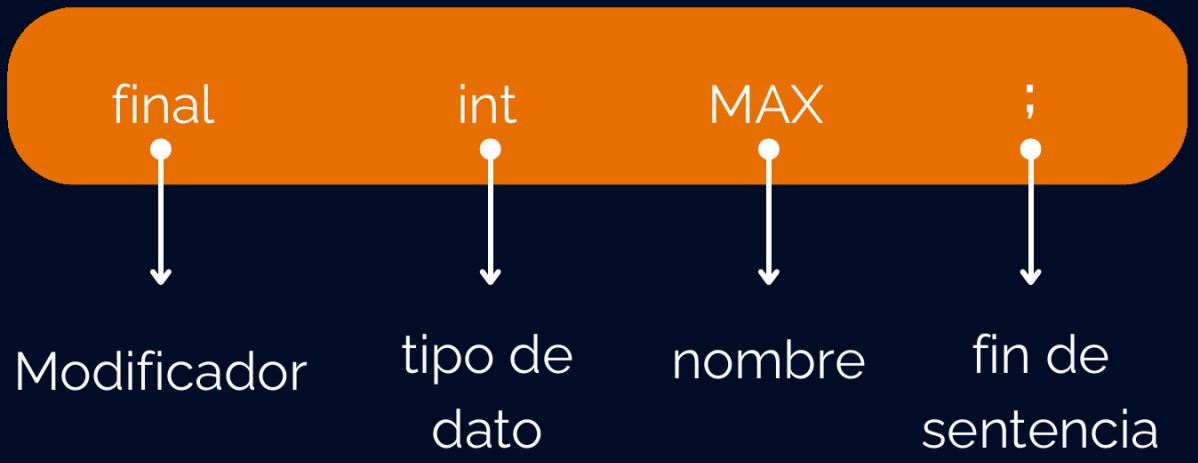
Variable sin inicializar: valor por defecto = 0

Constantes

Espacio en memoria donde se almacena un valor que **no** puede cambiar durante la ejecución del programa.

Constantes

DECLARACIÓN DE CONSTANTES



NO tiene valor por defecto

Reglas para definir nombres de variables y constantes

Reglas obligatorias (deben cumplirse):

01
No pueden contener espacios

02
Deben comenzar con una letra, $ o _

03
Solo pueden contener letras, números, $ y _

04
No pueden ser palabras reservadas

Reglas para definir nombres de variables

Prácticas recomendadas (no obligatorias, pero útiles):

01
Usa camelCase

02
Elige nombres significativos

03
Java es sensible a mayúsculas

04
Evita nombres muy largos

Reglas para definir nombres de constantes

Prácticas recomendadas (no obligatorias, pero útiles):

01
Usa UPPER_CASE

02
Elige nombres significativos

03
Java es sensible a mayúsculas

04
Evita nombres muy largos

Tipos de variables en Java

Existen dos categorías de datos principales

01
Primitivas

02
Referencia

¿Qué son los tipos primitivos?

Son tipos de datos que contienen un solo valor.

Gabriel Chaldú

Primitivos

Tipos de dato	Tamaño	Valor por defecto	Uso común
byte	1 byte	0	Números pequeños (ahorro de memoria).
short	2 bytes	0	Números pequeños (más grande que byte).
int	4 bytes	0	Números enteros generales.
long	8 bytes	0L	Números enteros muy grandes.
float	4 bytes	0.0f	Números decimales con precisión simple.
double	8 bytes	0.0d	Números decimales con precisión doble.
char	2 bytes	\u0000	Almacenar caracteres individuales.
boolean	1 bit*	false	Valores lógicos (verdadero/falso).

Tipo de dato:

boolean

- Un solo bit.
- Expresa un valor VERDADERO o FALSO.

```java
boolean isLoading = false;
boolean isActive = true;
```

www.devtalles.com

Gabriel Chaldú

Tipo de dato:
char

- Usa el código UNICODE y ocupa cada carácter 16 bits.

```
char aCharacter = 'a';
char oneCharacter = '1';
char unicode = '\u0021';
```

www.devtalles.com

Gabriel Chaldú

Tipo de dato:

byte, short, int, long

- byte, short, int y long
- Un entero es un número del conjunto Z = {..., -2, -1, 0, 1, 2, ...}.
- Difieren en las precisiones y pueden ser positivos o negativos.

```java
byte enteroByte = 127;           // Rango máximo positivo de byte
short enteroShort = 32767;      // Rango máximo positivo de short
int enteroInt = 2147483647;     // Rango máximo positivo de int
long enteroLong = 92233720368547758O7L; // Rango máximo positivo de long
```

www.devtalles.com

Gabriel Chaldú

Tipo de dato:
float, double

Tipos de datos para representar números reales con precisión simple (float) o doble (double).

```
float valorFloat = 3.4028235e38f;  // Rango máximo positivo de float
double valorDouble = 1.7976931348623157e308; // Rango máximo positivo de double
```

www.devtalles.com

Gabriel Chaldú

¡Nos vemos!

Gabriel Chaldú

Fundamentos de Programación

www.devtalles.com

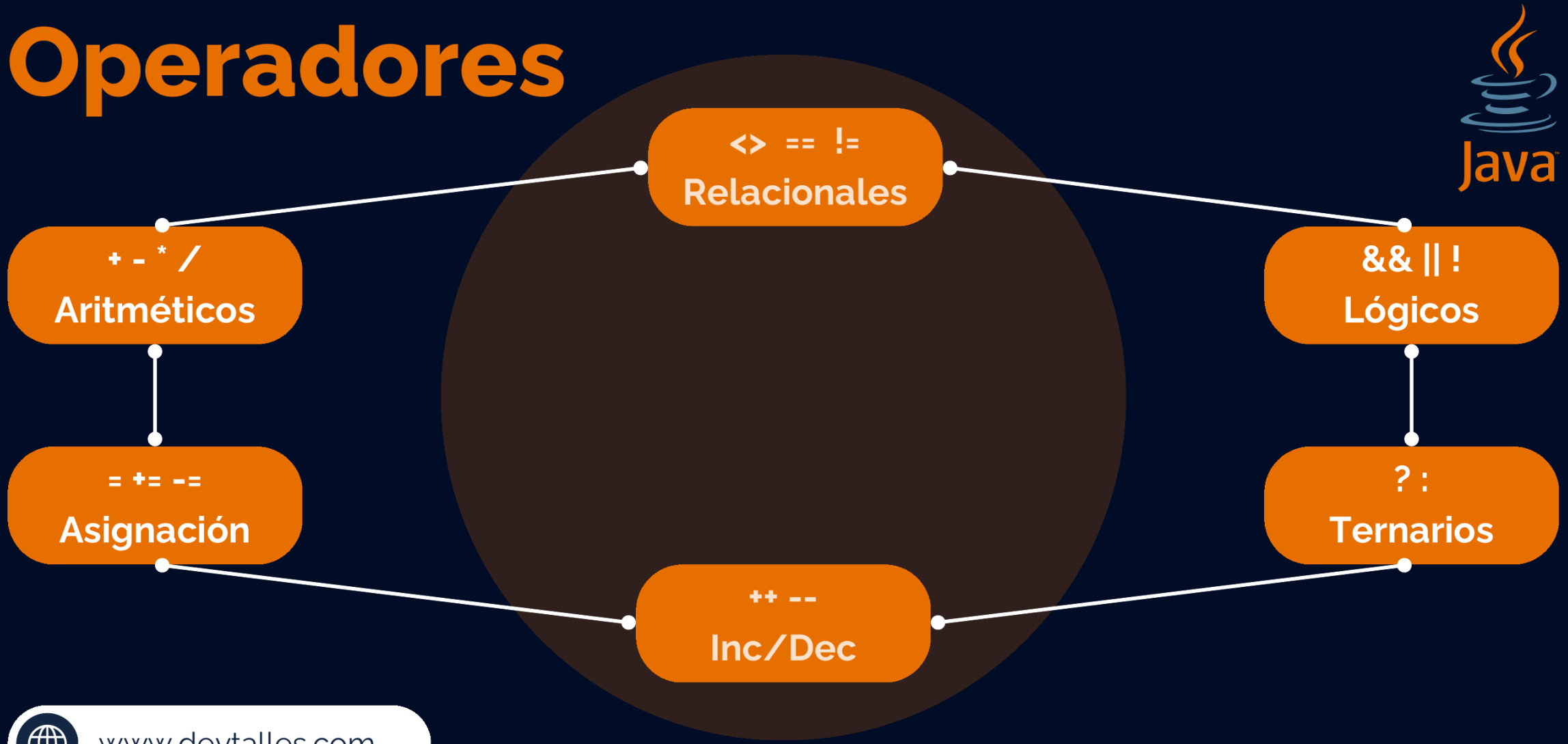
Gabriel Chaldú

Operadores básicos

Los operadores son:

- Son símbolos que permiten realizar operaciones en una o más variables.
- Se agrupan en diferentes categorías según su función.

Operadores



www.devtalles.com

Gabriel Chaldú

Operadores: Asignación

- Sirven para asignar valores a las variables.

Operadores de Asignación

`=` `+=` `-=` `*=` `/=` `%=`

Gabriel Chaldú

Ejemplos: Asignación


```java
int a = 10;  // Asignación simple
System.out.println("Asignación (=): " + a);

a += 5;  // Equivalente a: a = a + 5;
System.out.println("Suma y asignación (+=): " + a);

a -= 3;  // Equivalente a: a = a - 3;
System.out.println("Resta y asignación (-=): " + a);

a *= 2;  // Equivalente a: a = a * 2;
System.out.println("Multiplicación y asignación (*=): " + a);

a /= 4;  // Equivalente a: a = a / 4;
System.out.println("División y asignación (/=): " + a);

a %= 3;  // Equivalente a: a = a % 3;
System.out.println("Módulo y asignación (%=): " + a);
```

Gabriel Chaldú

Operadores: Aritméticos

- Permiten realizar operaciones matemáticas básicas.

Gabriel Chaldú

Ejemplos: Aritméticos


```
System.out.println("Suma: " + (a + b));          // Adición
System.out.println("Resta: " + (a - b));         // Sustracción
System.out.println("Multiplicación: " + (a * b)); //
System.out.println("División: " + (a / b));      // División entera
System.out.println("Módulo: " + (a % b)); // Resto de la división
```

Operadores: Relacionales

- Comparan dos valores y devuelven un valor booleano (true o false).

Ejemplos: Relacionales


```java
int a = 10, b = 20;

System.out.println("a == b: " + (a == b));  // Igualdad
System.out.println("a != b: " + (a != b));  // Diferente de
System.out.println("a > b: " + (a > b));    // Mayor que
System.out.println("a < b: " + (a < b));    // Menor que
System.out.println("a >= b: " + (a >= b));  // Mayor o igual que
System.out.println("a <= b: " + (a <= b));  // Menor o igual que
```

Gabriel Chaldú

Operadores: Lógicos

- Permite combinar expresiones booleanas para tomar decisiones basadas en múltiples condiciones.

Gabriel Chaldú

Ejemplos: Lógicos


```java
boolean a = true, b = false;

System.out.println("a && b: " + (a && b));  // AND lógico
System.out.println("a || b: " + (a || b));  // OR lógico
System.out.println("!a: " + (!a));          // NOT lógico
```

www.devtalles.com

Gabriel Chaldú

Operadores: Unarios

- Se aplican a un solo operando.

Ejemplos: Unarios


```java
int a = 5;
int b = -a; // Negación unaria
boolean flag = true;

System.out.println("Negación unaria: " + b); // -5
System.out.println("Negación lógica: " + !flag); // false

// Incremento y decremento
int c = 10;
System.out.println("Valor original: " + c);
System.out.println("Post-incremento: " + (c++)); // 10, luego c es
System.out.println("Pre-incremento: " + (++c)); // 12
System.out.println("Post-decremento: " + (c--)); // 12, luego c es
System.out.println("Pre-decremento: " + (--c)); // 10
```

¡Nos vemos!

Gabriel Chaldú

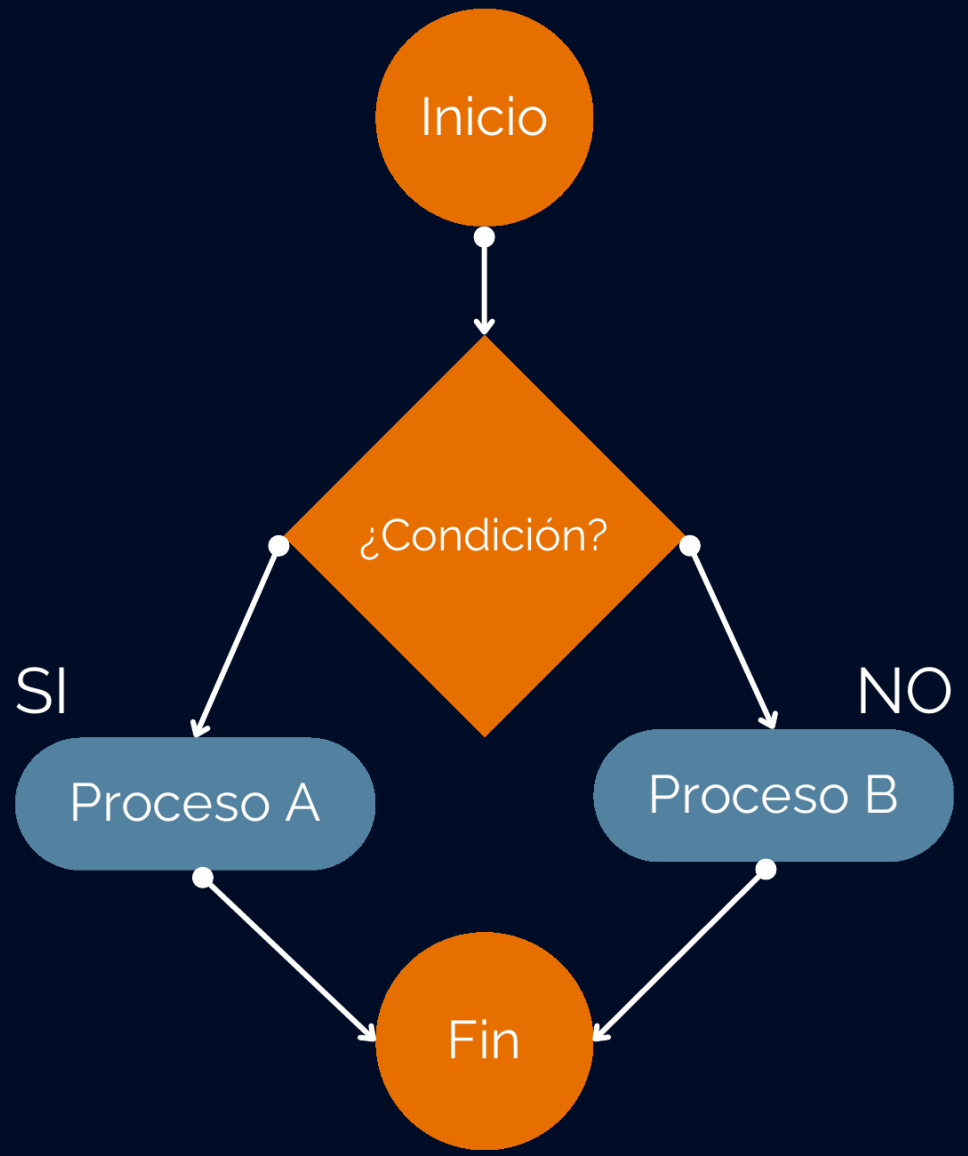
Fundamentos de Programación

www.devtalles.com

Gabriel Chaldú

Flujo de control

El flujo de control en Java define cómo se ejecutan las instrucciones en un programa. Se divide en tres tipos principales: secuencial, condicional y repetitivo.

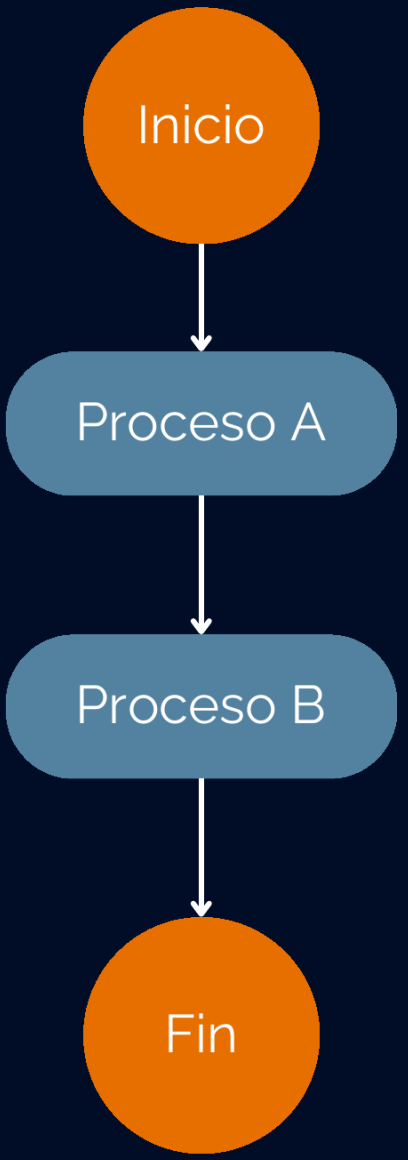


www.devtalles.com

Secuencial

Flujo básico: las instrucciones se ejecutan en el orden en que aparecen.

```java
int a = 10;  // Asignación simple
System.out.println("Asignación (=): " + a);
```

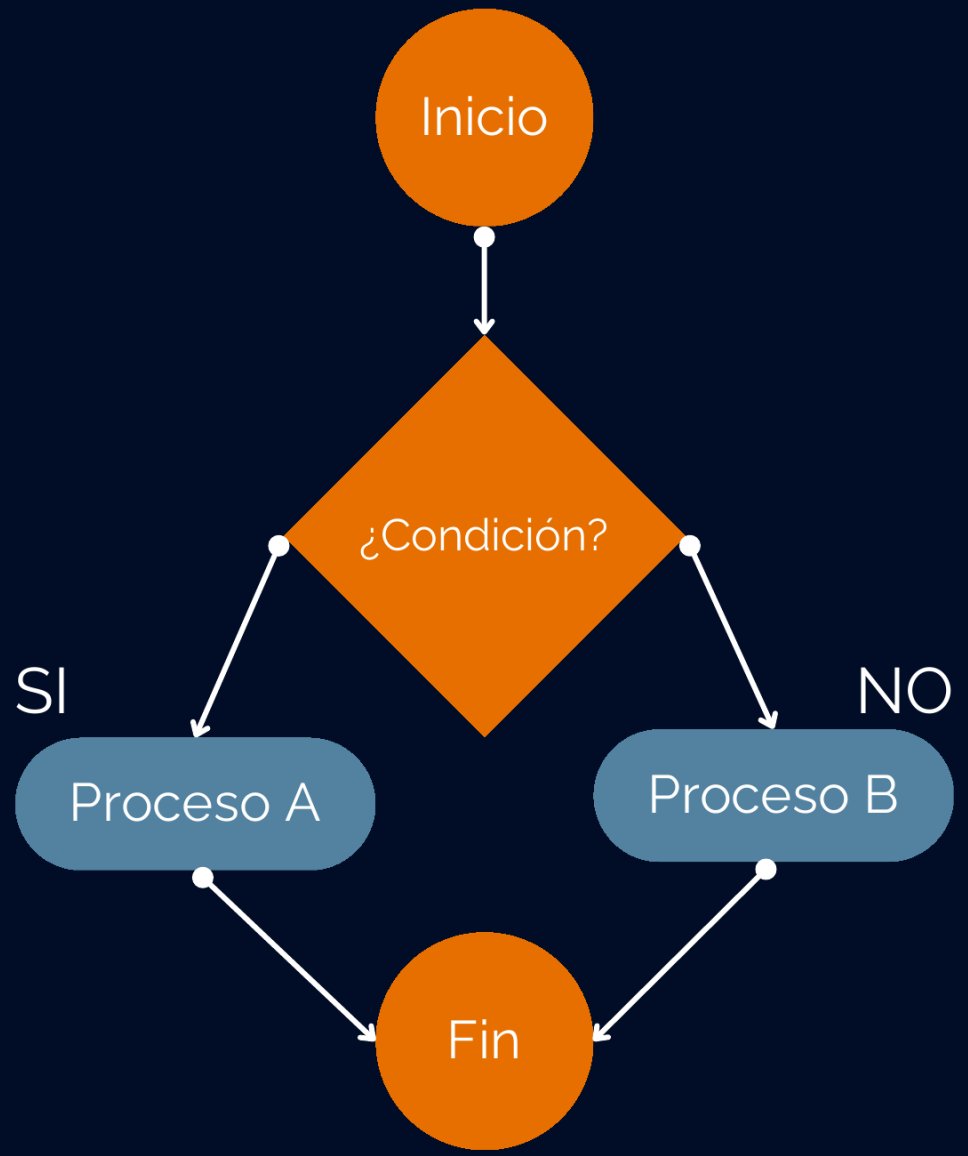


Condicional

Permite tomar decisiones basadas en condiciones.

```java
int numero = scanner.nextInt();

if (numero > 0) {
    System.out.println("El número es positivo.");
} else if (numero < 0) {
    System.out.println("El número es negativo.");
} else {
    System.out.println("El número es cero.");
}
```

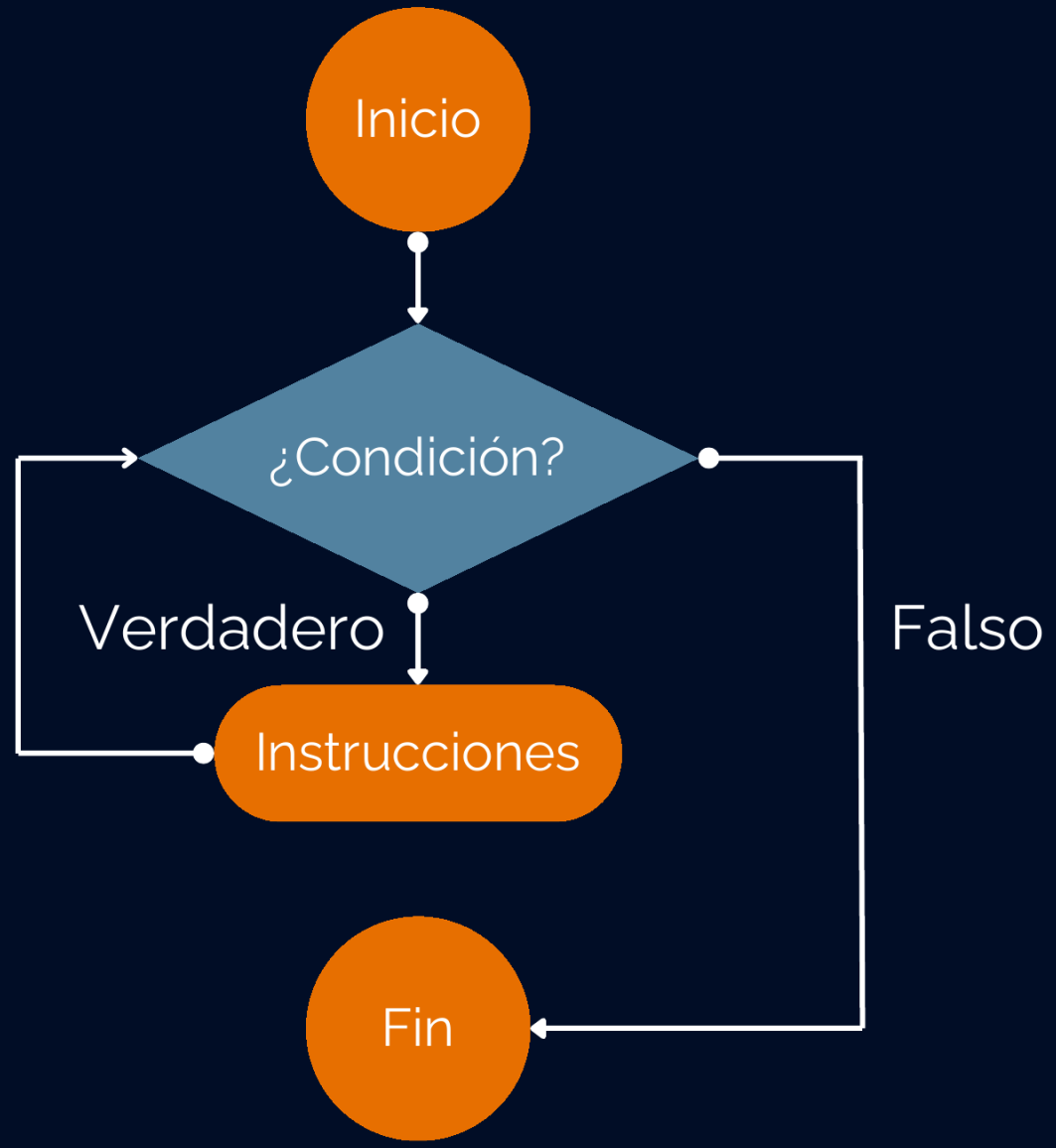
www.devtalles.com



Repetitivos

Repite bloques de código mientras se cumpla una condición.

```java
for (int i = 1; i <= 5; i++) {
    System.out.println("Número: " + i);
}
```



www.devtalles.com

¡Nos vemos!

Gabriel Chaldú

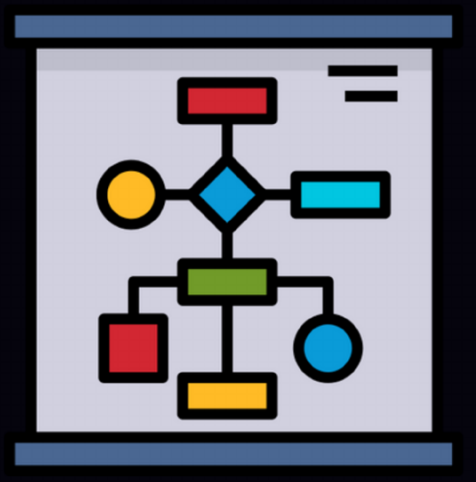
{dev/talles}

Programación orientada a objetos

Por: Gabriel Chaldú

¿Qué es la POO?

Es un paradigma de programación que utiliza ***objetos*** para representar datos y métodos asociados.



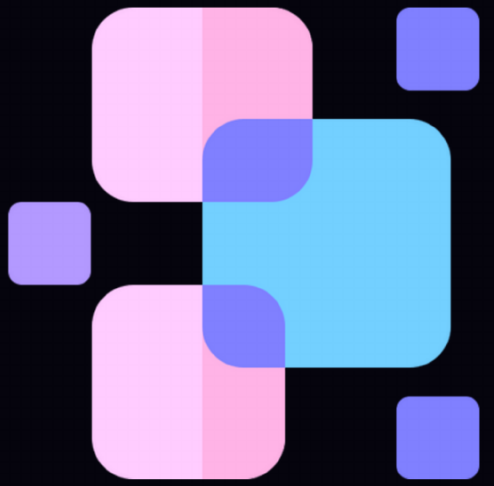
{dev/talles}

Pilares fundamentales de la POO

Por: Gabriel Chaldú

Abstracción

Proceso de ***seleccionar los atributos y métodos*** esenciales de un objeto para definir su estructura y comportamiento en nuestro sistema.



Abstracción

Mundo real → Vehicle → Código

Vehicle
brand model year
start() stop() accelerate()

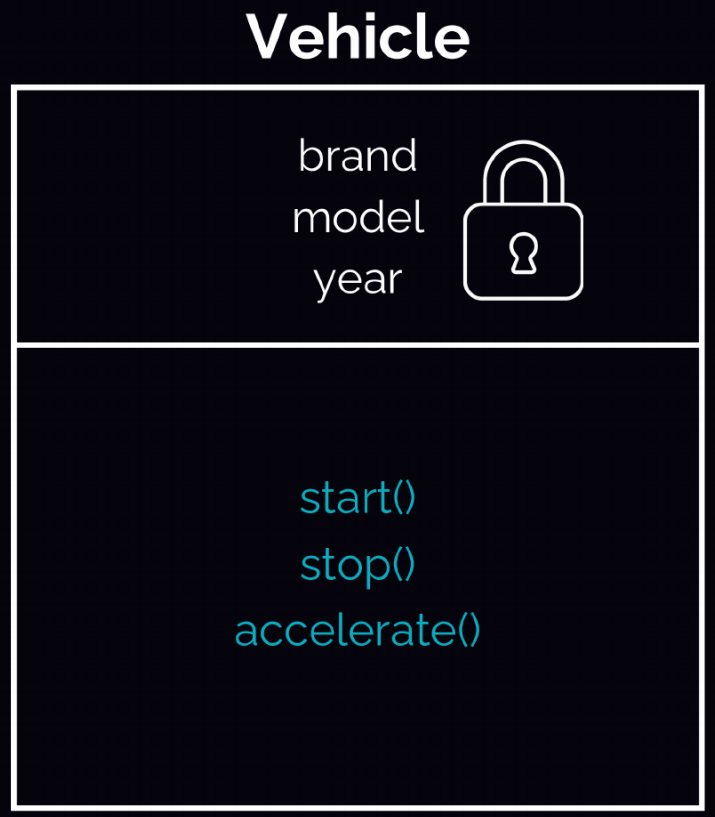
```java
public class Vehicle {
    private String brand;
    private String model;
    private int year;

    public Vehicle(String brand, String model, int year) {
        this.brand = brand;
        this.model = model;
        this.year = year;
    }

    public void start() {
        System.out.println("El vehículo está encendido.");
    }

    public void stop() {
        System.out.println("El vehículo está apagado.");
    }

    public void accelerate() {
        System.out.println("El vehículo está acelerando.");
    }

    // Getters and setters omitted for brevity
}
```

Encapsulamiento

Agrupa datos y métodos que operan sobre esos datos, ***restringiendo*** el acceso directo desde fuera de la clase.

Encapsulamiento

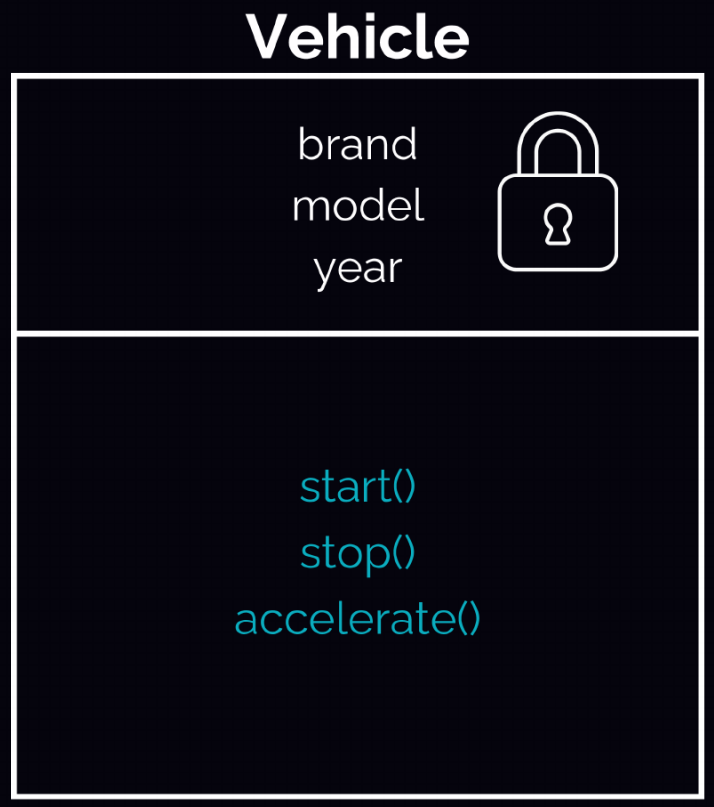
```java
public class Vehicle {
    private String brand;
    private String model;
    private int year;

    public Vehicle(String brand, String model, int year) {
        this.brand = brand;
        this.model = model;
        this.year = year;
    }

    public void start() {
        System.out.println("El vehículo está encendido.");
    }

    public void stop() {
        System.out.println("El vehículo está apagado.");
    }

    public void accelerate() {
        System.out.println("El vehículo está acelerando.");
    }

    // Getters and setters omitted for brevity
}
```



object.brand

object.start()

Encapsulamiento



¿Cómo accedo a los atributos private?

Getter y Setter:

```
String getBrand();
void setBrand(String brand);
```

Herencia

Permite crear nuevas clases basadas en clases existentes, facilitando la reutilización y ***extensión*** de código.

Se usa cuando **varias clases comparten atributos y métodos en común.**

Herencia

Vehicle - superclase

brand
model
year

start()
stop()
accelerate()

Car - subclase

Vehicle

brand
model
year

start()
stop()
accelerate()

doors
openDoors()

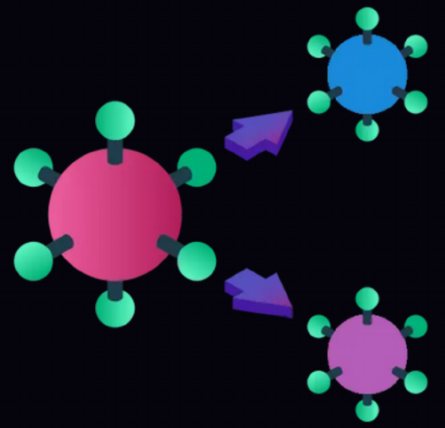
Herencia

```java
public class Vehicle {
    private String brand;
    private String model;
    private int year;

    public Vehicle(String brand, String model, int year) {
        this.brand = brand;
        this.model = model;
        this.year = year;
    }

    public void start() {
        System.out.println("El vehículo está encendido.");
    }

    public void stop() {
        System.out.println("El vehículo está apagado.");
    }

    public void accelerate() {
        System.out.println("El vehículo está acelerando.");
    }

    // Getters and setters omitted for brevity
}
```

```java
public class Car extends Vehicle {
    private int doors;

    public Car(String brand, String model, int year, int doors) {
        super(brand, model, year);
        this.doors = doors;
    }

    public void openDoors() {
        System.out.println("Las puertas del auto están abiertas.");
    }

    // Getter and setter for doors omitted for brevity
}
```

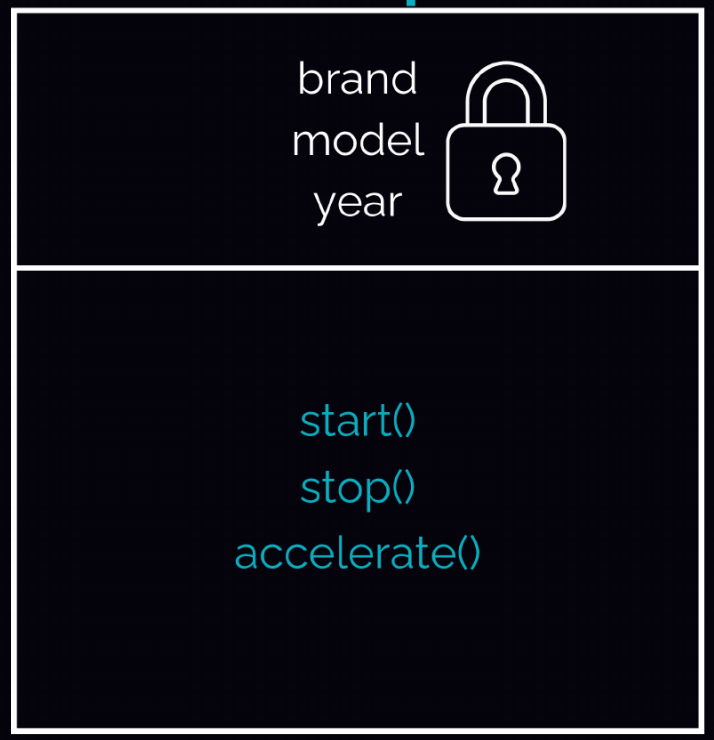
Polimorfismo

Permite que distintas clases sean tratadas de manera uniforme a través de una interfaz, posibilitando que un mismo método tenga comportamientos distintos según la clase que lo implemente.

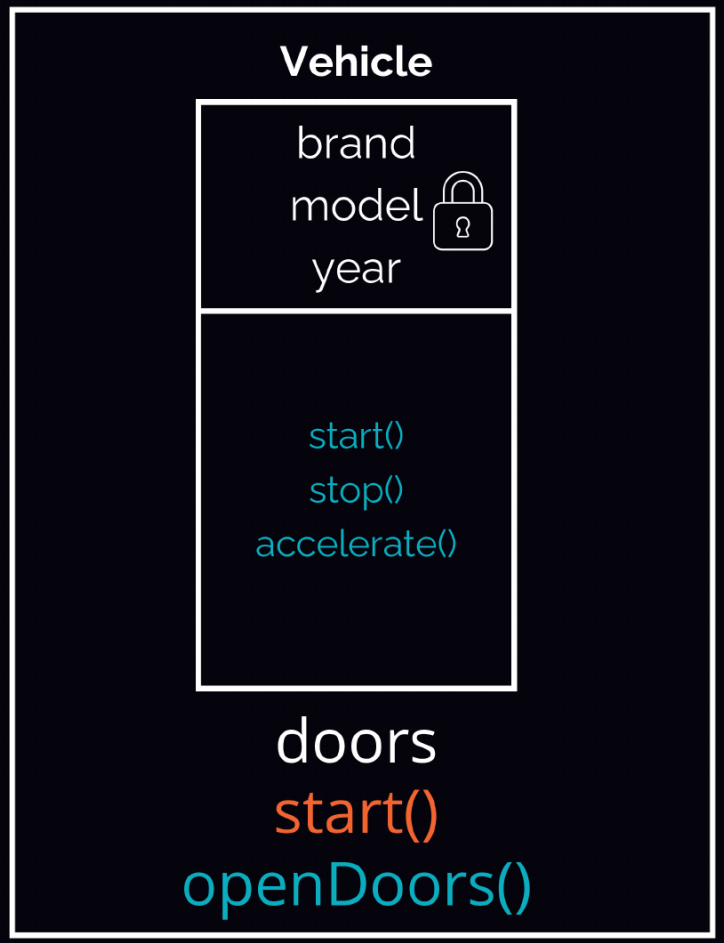


Polimorfismo

Vehicle - superclase



Car - subclase



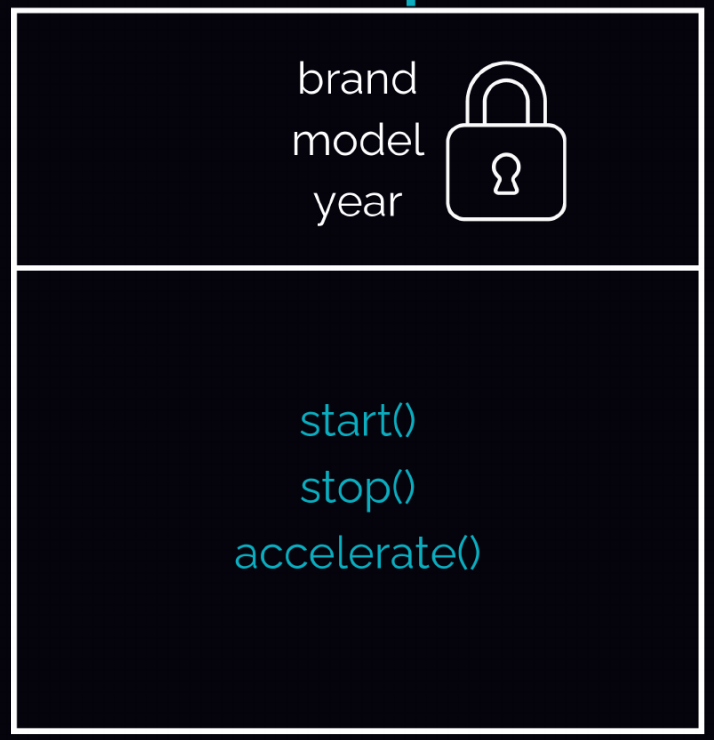
A partir del PADRE

Car ferrari = new Car();

Vehicle ferrari = new Car();

Polimorfismo

Vehicle - superclase



Car - subclase

Sobreescritura

```
start(){
}
```

Sobrecarga

```
openDoors(Vehicle v){
}
```

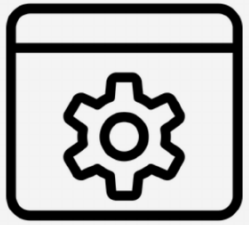
{dev/talles}

Conclusión

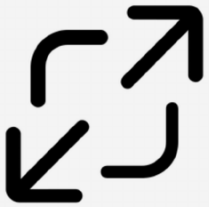
Por: Gabriel Chaldú

Ventajas de la POO

Reutilización

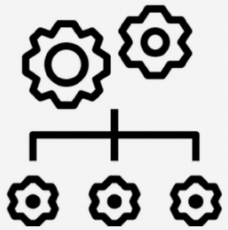


Extensibilidad

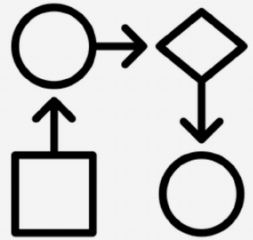


Ventajas de la POO

Modularidad



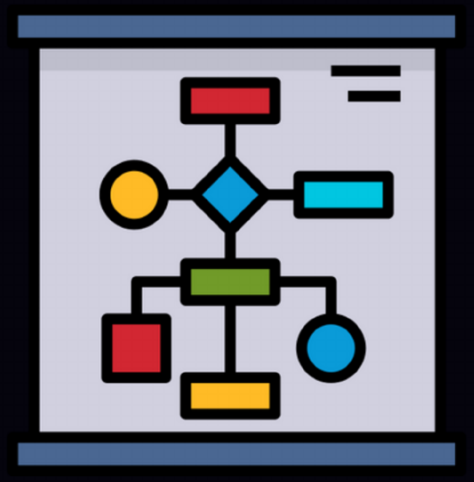
Modelado Natural



P.O.O.

Es esencial para el desarrollo de software moderno, ofreciendo una estructura clara y eficiente para manejar la complejidad.

Comprender sus fundamentos es crucial para diseñar sistemas robustos y mantenibles.



{dev/talles}

¡A seguir mejorando!