

FUNDAMENTOS DE PROGRAMACIÓN EN JAVA





FUNDAMENTOS DE PROGRAMACIÓN EN JAVA

¿QUÉ VAMOS A VER?

- Tipos de datos en Java.
- Operadores en Java.
- Sentencias de Control.
- Sentencias de Ramificación.





TIPOS DE DATOS EN JAVA

- Sabemos que una variable es un espacio en memoria en el que guardamos nuestros datos.
- Siempre vamos a tener la cualidad de poder designar un valor directamente a nuestras variables, o hacerlo a partir de otra variable, por ejemplo: **a = b**.
- Veremos que en Java los tipos de datos son más complejos y en esencia coexisten dos tipos de datos: los de **tipo primitivo** y los de **tipo objeto**.
- Los datos de tipo primitivo no necesitan ser invocados para ser creados, por ende, no tienen métodos que dependan de ellos.





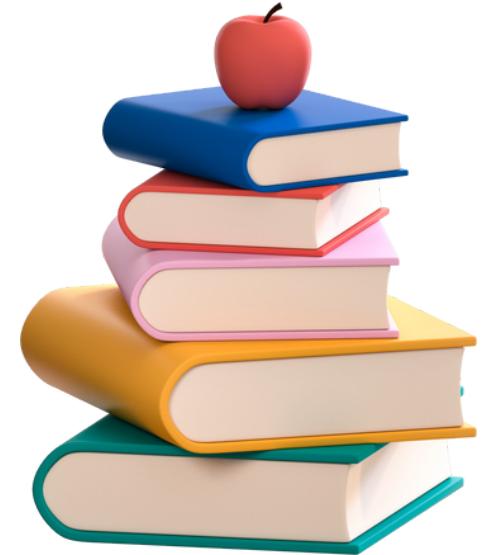
TIPOS DE DATOS PRIMITIVOS

Nombre	Tipo	Tamaño	Rango	Valor por defecto
Byte	Entero	1 byte	-128 a 127	0
Short	Entero	2 bytes	-32.768 a 32.767	0
Int	Entero	4 bytes	-2.147.483.648 a 2.147.483.647	0
Long	Entero	8 bytes	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	0
Float	Decimal simple	4 bytes	1.4e-45 a 3.4e38	0.0f
Double	Decimal doble	8 bytes	4.9e-324 a 1.8e308	0.0
Char	Carácter simple	2 bytes	Un carácter simple.	'\u0000'
Boolean	Valor de verdad	1 byte	True o False	False



STRING (CADENA DE TEXTO)

- Es fundamental para el manejo de cadenas de texto.
- Al ser una clase más que un tipo de dato posee una amplia gama de métodos para manipular cadenas.
- Los más comunes son:
- **length():** Se utiliza para obtener la longitud de un String.
- **charAt(index):** Se utiliza para acceder a un carácter individual en un String. *Index* es la posición del carácter que queremos obtener.



```
String texto = "Hola Mundo";  
int longitud = texto.length();|
```

```
char letra = texto.charAt(index: 1);|
```



STRING (CADENA DE TEXTO)

- **substring(inicio, fin)**: Se utiliza para obtener una subcadena de un String. *Inicio* marca el índice del primer carácter y *fin* marca el índice final.

```
String subcadena = texto.substring(beginIndex: 0, endIndex: 4);
```

- **equals()**: Se utiliza para comparar dos String. retorna True o False. Es importante tener en cuenta que es sensible a las mayúsculas y minúsculas.

```
String texto = "Hola Mundo";
String texto2 = "Hola Mundo";

boolean sonIguales = texto.equals(anObject: texto2);
```

- **equalsIgnoreCase()**: Funciona prácticamente igual a equals(), excepto porque ignora si existe alguna diferencia entre mayúsculas y minúsculas.

```
String texto = "Hola Mundo";
String texto2 = "hola mundo";

boolean sonIguales = texto.equalsIgnoreCase(anotherString: texto2);
```



STRING (CADENA DE TEXTO)

- **replace(target, replacement)**: Se utiliza para reemplazar todas las ocurrencias de un carácter o subcadena por otro. *Target* corresponde al carácter que queremos reemplazar y *replacement* al nuevo.

```
String texto = "Hola Mundo";
String reemplazado = texto.replace(target: "o", replacement: "Ø");
```

- **Integer.parseInt(String)**: Se utiliza para convertir una cadena de texto en un entero de tipo *int*.

```
String cadena = "123";
int numero = Integer.parseInt(s: cadena);
```

- **Double.parseDouble(String)**: Se utiliza para convertir una cadena de texto en un número de doble precisión, o de tipo *double*.

```
String cadena = "123.45";
double nDouble = Double.parseDouble(s: cadena);
```



STRING (CADENA DE TEXTO)

- **format()**: Se utiliza para crear cadenas de texto usando marcadores de posición que van a ser reemplazados por valores entregados.

```
int edad = 30;
String nombre = "Goku";

String mensaje = String.format(format:"Hola, mi nombre es %s y tengo %d años.", args: nombre, args: edad);
```

- **contains()**: Se utiliza para saber si existe una frase dentro de un String. Nos va a retornar un valor booleano según corresponda.

```
String texto = "Hola mundo";
boolean contiene = texto.contains(s: "mundo");
```

- **toUpperCase()**: Se utiliza para transformar una cadena de texto a mayúsculas.

```
String texto = "Hola mundo";
String cMayus = texto.toUpperCase();
```

- **toLowerCase()**: Se utiliza para transformar una cadena de texto a minúsculas.

```
String texto = "Hola mundo";
String cMinus = texto.toLowerCase();
```



SCANNER

- Este objeto se utiliza para obtener datos de entrada del usuario, viene a suplir lo que en PSelnt hacíamos como “Leer”.
- Forma parte del paquete **java.util**, por lo tanto, debemos importarlo en nuestro código antes de poder utilizarlo.
- **nextLine()**: Obtiene una línea de texto como entrada.
- **nextInt()**: Obtiene un entero como entrada.
- **nextDouble()**: Obtiene un número decimal como entrada.
- **nextBoolean()**: Obtiene un valor booleano como entrada.

```
Scanner input = new Scanner(source: System.in);
```

```
System.out.println("Ingresa tu nombre: ");
String name = input.nextLine();
```





OPERADORES EN JAVA

OPERADORES ARITMÉTICOS

Realizan operaciones simples aritméticas simples en datos primitivos.

Nombre	Símbolo	Descripción	Ejemplo
Suma	+	Suma dos números.	$a + b$
Resta	-	Resta dos números.	$a - b$
Multiplicación	*	Multiplica dos números.	$a * b$
División	/	Divide dos números.	a / b
Módulo	%	Resto de la división de dos números.	$a \% b$



OPERADORES EN JAVA

OPERADORES RELACIONALES

Se utilizan para verificar relaciones, retornando así un dato de tipo booleano.

Nombre	Símbolo	Descripción	Ejemplo
Igualdad	<code>==</code>	Comprueba si dos variables son iguales.	<code>a == b</code>
Desigualdad	<code>!=</code>	Comprueba si dos variables son diferentes.	<code>a != b</code>
Menor que	<code><</code>	Comprueba si la primera variable es menor que la segunda.	<code>a < b</code>
Menor o igual que	<code><=</code>	Comprueba si la primera variable es menor o igual que la segunda.	<code>a <= b</code>
Mayor que	<code>></code>	Comprueba si la primera variable es mayor que la segunda.	<code>a > b</code>
Mayor o igual que	<code>>=</code>	Comprueba si la primera variable es mayor o igual que la segunda.	<code>a >= b</code>



OPERADORES EN JAVA

OPERADORES LÓGICOS

Se utilizan para probar más de una condición a la hora de tomar decisiones.

Nombre	Símbolo	Descripción	Ejemplo
AND	&&	Devuelve true si ambas condiciones se cumplen, de lo contrario devuelve false .	(a + b) && (a == c)
OR		Devuelve true si al menos una de las condiciones se cumple, de lo contrario devuelve false .	(a + b) (a == c)
NOT	!	Invierte el valor booleano de una variable.	!(a > b)



OPERADORES EN JAVA

OPERADORES DE ASIGNACIÓN

Se utilizan para entregar un valor a una variable.

Nombre	Símbolo	Descripción	Ejemplo
Simple	=	Asigna el valor a una variable.	a = b
Con Suma	+=	Suma el valor asignado con el ya existente.	a += b
Con Resta	-=	Resta el valor asignado con el ya existente.	a -= b
Con Multiplicación	*=	Multiplica el valor asignado con el ya existente.	a *= b
Con División	/=	Divide el valor asignado con el ya existente	a /= b
Con Módulo	%=	Obtiene el resto de la división del valor asignado con el ya existente.	a %= b



OPERADORES EN JAVA

OPERADORES INCREMENTALES Y DECREMENTALES

Se utilizan para disminuir o aumentar en uno el valor de una variable.

Nombre	Símbolo	Descripción	Ejemplo
Pre-incremento	<code>++</code>	Incrementa en 1 el valor de la variable antes de la evaluación.	<code>++a</code>
Post-incremento	<code>++</code>	Incrementa en 1 el valor de la variable luego de la evaluación.	<code>a++</code>
Pre-decremento	<code>--</code>	Decrementa en 1 el valor de la variable antes de la evaluación.	<code>--b</code>
Post-decremento	<code>--</code>	Decrementa en 1 el valor de la variable luego de la evaluación.	<code>b--</code>



SENTENCIA IF-ELSE

- Proporciona la posibilidad de ejecutar selectivamente otras sentencias, basándose en el criterio de una condicional.
- **If:** Es la sentencia más simple de todas. Tiene solo un camino a seguir y es que el bloque de código solo se ejecuta si la condición es true.

```
if (a == b) {  
    System.out.println("Hola Mundo");  
}
```

```
if (a == b) System.out.println("Hola Mundo");
```

- **If-else:** Esta sentencia permite ejecutar código en caso de que también la condición retorne false.
- **If-else if:** Se evalúan múltiples condiciones que, en caso de no cumplirse, si lo hará el bloque de código dentro de else.

```
if (a == b) {  
    System.out.println("Hola Mundo");  
} else if (c == d) {  
    System.out.println("Dentro de un else if");  
} else {  
    System.out.println("Dentro de else");  
}
```



SENTENCIA IF-ELSE

- **If Anidados:** Su finalidad es tener un mayor control en el flujo del programa, es por eso que va a permitir que ciertos bloques de código solo ocurran si antes se cumplió o no X condición.

```
if (a == b) {  
    if (c == d) {  
        if (e == f) {  
            System.out.println("Hola Mundo");  
        } else {  
            System.out.println("Chao Mundo");  
        }  
    } else {  
        System.out.println("Hulala");  
    }  
} else {  
    System.out.println("Salir salir");  
}
```

- **Operador Ternario:** Es una forma más simple de escribir una condicional if-else.

```
String mensaje = (a == b) ? "Esto es true" : "Esto es false";
```





EJEMPLO

- Desarrolle un programa que asigne notas basadas en la puntuación de un examen.
- Para obtener una calificación “Sobresaliente” debe obtener una puntuación del 90% o superior, un “Notable” se obtiene con el 80% o superior, “Bien” se obtiene con una nota superior a 70% y un “Suficiente” con un puntaje de 60% o superior.
- En cualquier otro caso, se obtiene una calificación “insuficiente”.





EJERCICIO

- Escribe un programa que solicite al usuario ingresar un número.
- Se debe determinar si el número ingresado es positivo o negativo.
- Muestra el resultado por consola.





EJERCICIO

- Solicita al usuario ingresar tres números.
- Determina cuál de ellos es el número mayor.
- Muestra por consola el resultado.





EJERCICIO

- Crea un programa que actúe como una calculadora básica.
- Se le solicitará al usuario ingresar dos números y una operación matemática (+, -, x u /).
- Luego, calculará la operación seleccionada y mostrará el resultado por pantalla.
- Si el símbolo de la operación es diferente a los permitidos, se le debe especificar al usuario que existe un error.





SENTENCIA SWITCH-CASE

- Es una excelente elección cuando tenemos diferentes “caminos a seguir” y dependemos del dato almacenado en una única variable.
- Cada opción va a estar representada por la palabra reservada **case**.
- Es importante tener en cuenta la sentencia **break** que nos va a permitir cortar la ejecución de nuestra sentencia.
- Si existe la eventualidad de que ninguno de los casos se cumpla, tendremos la opción de **default**.

```
switch (number) {  
    case 1:  
        System.out.println("Número 1");  
        break;  
    case 2:  
        System.out.println("Número 2");  
        break;  
    case 3:  
        System.out.println("Número 3");  
        break;  
    default:  
        System.out.println("No contiene un número");  
        break;  
}
```



SENTENCIA SWITCH-CASE

- En las nuevas versiones de Java existe la posibilidad de que nos encontremos con una nueva sintaxis de switch-case.
- Es importante conocer ambas para entender su funcionamiento a la hora de afrontarlas.
- Esta nueva sintaxis omite el uso de **break**.

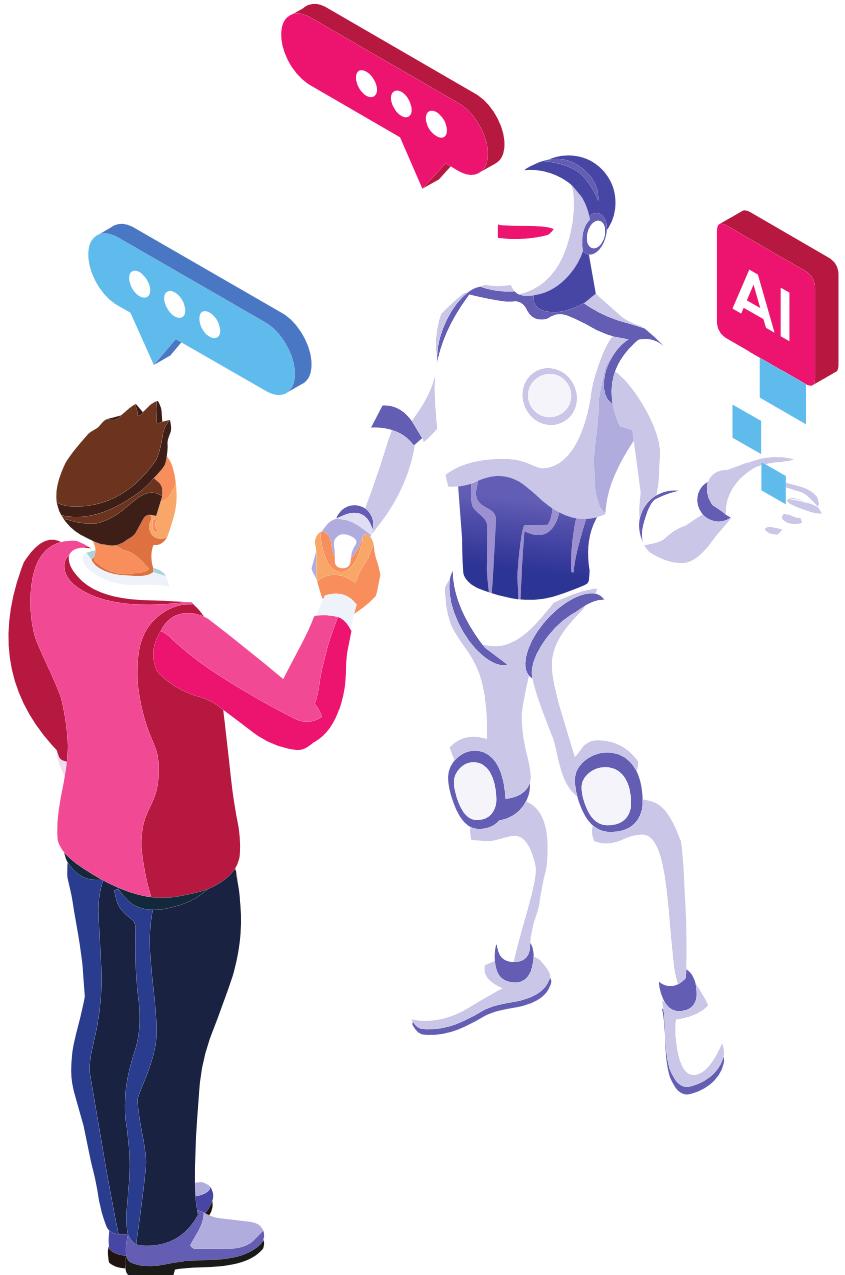


```
switch (number) {  
    case 1 -> System.out.println("Número 1");  
    case 2 -> System.out.println("Número 2");  
    case 3 -> System.out.println("Número 3");  
    default -> System.out.println("No contiene un número");  
}
```



EJERCICIO

- Pide al usuario un número del 1 al 7.
- Muestra por consola el día de la semana correspondiente al número ingresado.





EJERCICIO

- Tomemos de base el ejercicio anterior y cambiemos un poco su funcionamiento.
- Pide al usuario que ingrese el nombre del día de la semana.
- Devuelve por consola una respuesta correspondiente a:
 - Inicio de semana.
 - Mediados de semana.
 - Último día de la semana.
 - Fin de semana.
- Ten en cuenta que el usuario puede ingresar el nombre tanto en mayúsculas como minúsculas.





EJERCICIO

- Vamos a crear una aplicación sostenida en el tiempo. Simularemos el sistema de un banco que escalaremos según aprendamos nuevo contenido.
- En primera instancia le pediremos al usuario que se registre.
 - (Nombre completo, rut, fecha de nacimiento, email, teléfono, dirección/ciudad/región, etc...)
- Una vez los datos sean validados, se muestran por pantalla.
- **Tips:** Podemos ir más al detalle e incluso ingresar el tipo de cuenta, sueldo y beneficios que tiene el cliente.





AWAKELAB

#programmingbootcamp

nodovirtual.awakelab.cl

 jELOU futurejob by  Adalid Chile