Elementos básicos de la programación

UD02 · Programación · 1º DAM

Carlos Barroso · CFGS Desarrollo de Aplicaciones Multiplataforma

La unidad más rentable del curso

Datos, operaciones y decisiones: **lo que vas a usar en todos los programas que escribas**, en Java y en cualquier otro lenguaje.

Y la más densa. Ve haciendo los ejercicios de cada bloque sobre la marcha, no al final.

Qué vamos a ver

1. IntelliJ IDEA: el primer proyecto
2. Variables e identificadores
3. Tipos de datos, literales y constantes
4. Entrada y salida por consola
5. Operadores
6. Conversiones de tipo
7. Estructuras de selección
8. Estructuras de repetición

1 · El IDE

Ahora sí

IntelliJ IDEA Community

- **Community**, que la Ultimate es de pago
- JDK **25** al crear el proyecto
- ▶ para ejecutar · `Ctrl` + `Mayús` + `F10`
- La salida y la entrada, en la ventana **Run**

Por dentro sigue llamando al mismo compilador y a la misma JVM que usabas en el terminal. **El IDE es comodidad, no magia.**

2 · Variables

Qué es una variable

> Un espacio en memoria, con un **nombre** y un **tipo**, cuyo contenido puede cambiar durante la ejecución.

```
int edad;          // declaración
edad = 17;         // asignación
int cursos = 2;    // las dos cosas a la vez
```

Como una taquilla: un número, un tamaño y un contenido que cambias cuando quieras.

`=` no significa «igual»

`=` **es asignación**: coge lo de la derecha y guárdalo en lo de la izquierda.

```
contador = contador + 1;   // no es un disparate matemático
```

Para comparar se usa `==`, que es otra cosa.

Confundirlos es el error número uno del primer trimestre.

Cómo se llaman

Reglas del lenguaje:

- Letras, dígitos, `_` y `$`; **nunca empezando por dígito**
- Sin espacios ni signos de puntuación
- No pueden ser palabras reservadas (`int`, `class`, `for`…)
- Distingue mayúsculas: `nota` ≠ `Nota` ≠ `NOTA`

Convenios: `notaMedia` (variables) · `IVA_GENERAL` (constantes) · `CuentaBancaria` (clases)

El nombre tiene que decir qué guarda

`n` no dice nada. `numeroDeAlumnos` sí.

Escribirlo cuesta dos segundos con el autocompletado.
Entender dentro de un mes qué era `x2` cuesta mucho más.

3 · Tipos de datos

Los ocho primitivos

Tipo	Guarda	Rango aproximado
`byte` · `short`	Enteros pequeños	±127 · ±32.767
`int`	Enteros	±2.147 millones
`long`	Enteros grandes	±9,2 trillones
`float` · `double`	Decimales	7 · 15 dígitos
`char`	Un carácter	`'a'` , `'€'`
`boolean`	Cierto o falso	`true` / `false`

El 90 % del tiempo: `int` , `double` , `boolean` y `char` .

Los decimales no son exactos

```
IO.println(0.1 + 0.2);   // 0.30000000000000004
```

No es un fallo de Java: en binario hay números sin representación exacta, igual que 1/3 no la tiene en decimal.

Nunca uses `double` para dinero en un programa serio. Para eso está `BigDecimal`.

Literales

```
long milisegundos = 1298332800000L;   // la L es obligatoria
float altura = 1.75f;                 // la f también
char inicial = 'C';                   // comillas simples
String nombre = "Lucía";              // comillas dobles

int millon = 1_000_000;               // el _ solo ayuda a leer
int binario = 0b1010;                 // 10
int hexadecimal = 0xFF;               // 255
```

Constantes y `var`

```
final double IVA = 0.21;   // si alguien la cambia, no compila
final int AFORO = 540;

var nombre = "Lucía";      // String, deducido
var edad = 17;             // int
```

`var` **no** significa «sin tipo»: el tipo se decide al declararla y no cambia.

```
var edad = 17;
edad = "diecisiete";   // ✗ error
```

Primitivos y referenciados

Primitivo: la variable contiene el valor.

Referenciado: contiene la dirección donde está el objeto. Puede ser `null`.

```
String a = IO.readln("Escribe hola: ");

if (a == "hola")      { ... }   // ✗ compara direcciones
if (a.equals("hola")) { ... }   // ✓ compara contenido
```

Fuente de errores número uno de todo primero de DAM.

Cosas que hacer con un `String`

```
String texto = "Programación";

texto.length()           // 12
texto.toUpperCase()      // "PROGRAMACIÓN"
texto.charAt(0)          // 'P'
texto.substring(0, 4)    // "Prog"
texto.trim()             // sin espacios en los extremos
texto.equalsIgnoreCase("programación")   // true
```

Se empieza a contar en 0.

4 · Entrada y salida

Hablar con el usuario

```
IO.println("Hola");                    // escribe y salta de línea
IO.print("Sin salto");                 // se queda ahí
String n = IO.readln("¿Nombre? ");     // pregunta y lee
```

`IO.readln` siempre devuelve **texto**:

```
int edad = Integer.parseInt(IO.readln("¿Edad? "));
double nota = Double.parseDouble(IO.readln("¿Nota? "));
```

¿Y el `Scanner`?

```
Scanner sc = new Scanner(System.in);
int edad = sc.nextInt();
```

Es la forma clásica y funciona. Necesita un `import`, crear un objeto y entender qué es `System.in`: **tres cosas que aún no hemos explicado**.

Usamos `IO`. Pero reconoce el `Scanner`, porque lo vas a ver en todas partes.

Dar formato

```
double total = 1234.5678;

IO.println("Total: " + total);                    // 1234.5678
IO.println("Total: %.2f €".formatted(total));  // 1234,57 €
IO.println("%-15s%5d".formatted("Camisetas", 3));
```

`%d` enteros · `%f` decimales · `%s` texto · `%-10s` alineado · `%%` un porcentaje

Y `\n`, `\t`, `\"`, `\\` dentro de las cadenas.

5 · Operadores

Aritméticos: cuidado con la división

```
int a = 7, b = 2;

a / b             // 3   ← división entera, se trunca
7.0 / 2           // 3.5
(double) a / b    // 3.5
a % b             // 1   ← el resto
```

Si los dos operandos son enteros, el resultado es entero.

`%` no es un operador menor: `n % 2 == 0` detecta pares, `n % 10` da la última cifra.

Relacionales y lógicos

```
==  !=  >  <  >=  <=          // devuelven true o false

if (edad >= 18 && tieneCarnet) { ... }   // y
if (dia == 6 || dia == 7)      { ... }   // o
if (!aprobado)                 { ... }   // no
```

Cortocircuito: si la primera parte de un `&&` es falsa, la segunda ni se evalúa.

```
if (divisor != 0 && total / divisor > 10) { ... }   // seguro
```

El ternario

```
int mayor = (a > b) ? a : b;
String msg = (nota >= 5) ? "Aprobado" : "Suspenso";
```

¿condición? entonces esto : si no, esto otro

Bien para casos cortos. Si empiezas a anidarlos, usa un `if`.

En la duda, paréntesis

```
nota1 + nota2 + nota3 / 3      // ✗ solo divide la tercera
(nota1 + nota2 + nota3) / 3    // ✓
```

No cuestan nada y evitan discusiones.

Conversiones de tipo

Implícita (de pequeño a grande, sin pérdida): la hace Java sola.

```
int entero = 7;
double decimal = entero;   // 7.0
```

Explícita (*casting*, de grande a pequeño): la pides tú y asumes la pérdida.

```
double precio = 10.99;
int truncado = (int) precio;   // 10
```

Texto → número **no es un casting**: `Integer.parseInt("42")`.

6 · Selección

`if` / `else if` / `else`

```
if (nota < 5) {
    IO.println("Insuficiente");
} else if (nota < 6) {
    IO.println("Suficiente");
} else if (nota < 9) {
    IO.println("Notable");
} else {
    IO.println("Sobresaliente");
}
```

Se comprueban **de arriba abajo** y se ejecuta el primero que se cumple. El orden importa.

Pon siempre las llaves

Java te deja omitirlas cuando hay una sola instrucción.

No lo hagas.

El día que añadas una segunda línea creerás que está dentro del `if` y no lo estará. Ese error ha provocado agujeros de seguridad famosos en software real.

`switch` como expresión

```
String nombreDia = switch (dia) {
    case 1 -> "Lunes";
    case 2 -> "Martes";
    case 6, 7 -> "Fin de semana";
    default -> "Día no válido";
};
```

Con flechas **no hace falta** `break`. La forma clásica sí lo necesita, y olvidarlo ha causado tantos errores que por eso nació esta sintaxis.

`switch` para valores concretos; `if` para intervalos.

7 · Repetición

`while` : no sé cuántas vueltas

```
while (intentos < 4) {
    String clave = IO.readln("Combinación: ");
    if (clave.equals("1234")) break;
    intentos++;
}
```

Se comprueba **antes** de entrar: puede no ejecutarse ninguna vez.

⚠️ Si dentro no cambia nada que afecte a la condición → **bucle infinito**.

`do-while` : al menos una vez

```
do {
    IO.println("1. Vender  2. Consultar  3. Salir");
    opcion = Integer.parseInt(IO.readln("Opción: "));
    ...
} while (opcion != 3);
```

La condición se comprueba **al final**. Es la estructura de los menús.

`for` : sé cuántas vueltas

```
for (int i = 1; i <= 10; i++) {
    IO.println(i + " x 3 = " + (i * 3));
}
```

- **Inicialización** — una vez, al principio
- **Condición** — antes de cada vuelta
- **Actualización** — al final de cada vuelta

Todo junto en una línea: por eso es el que menos se olvida de actualizar.

Bucles anidados

```
for (int fila = 1; fila <= altura; fila++) {
    for (int col = 1; col <= fila; col++) {
        IO.print("*");
    }
    IO.println();
}
```

Una pirámide es exactamente eso: un bucle de filas y, dentro, uno de columnas.

`break` y `continue`

```
for (int i = 1; i <= 100; i++) {
    if (i % 3 == 0) continue;   // salta esta vuelta
    if (i > 50) break;          // sal del bucle
    IO.println(i);
}
```

Cómodos, pero con moderación: un bucle con cuatro `break` repartidos no hay quien lo siga.

Depurar es más rápido que adivinar

Punto de interrupción en el margen · botón 🐞 · `F8` para avanzar

Es la traza a mano de la UD01, pero hecha por el ordenador.

Y es lo que menos usa el alumnado de primero.

La práctica de la unidad

Auditorio Municipal de Martos: venta de entradas, facturas con código único, cambio en el menor número de monedas y consulta del estado de cada evento.

Con lo de esta unidad tenéis todo lo necesario. Empezad por el menú vacío y añadid una cosa cada vez.

A programar

apuntes.carlosprofe.es

31 ejercicios con solución, ordenados por bloques

Descargar estas diapositivas en PDF