Introducción a la programación

UD01 · Programación · 1º DAM

Carlos Barroso · CFGS Desarrollo de Aplicaciones Multiplataforma

Esta unidad no va de Java

Va de aprender a **pensar los problemas antes de escribirlos**.

- Los lenguajes de moda dentro de diez años todavía no existen.
- Lo que aprendas aquí te va a servir con todos ellos.

Java aparece al final. Y solo para saludar.

Qué vamos a ver

1. Qué es un algoritmo y en qué se diferencia de un programa
2. Del problema a la solución: análisis, diseño, codificación, pruebas
3. Cómo se representa un algoritmo: pseudocódigo y diagramas de flujo
4. Paradigmas de programación
5. Cómo llega tu código hasta el procesador
6. **Scratch**: programar sin sintaxis
7. Java: historia, JVM y primer programa

1 · Programar es resolver problemas

Y el código es el último paso

Con qué máquina estás tratando

Un ordenador es **rapidísimo** y **absolutamente literal**.

- Hace exactamente lo que le dices.
- En el orden en que se lo dices.
- Sin interpretar tus intenciones.
- Sin avisarte de que probablemente querías decir otra cosa.

> Programar consiste, sobre todo, en **quitar ambigüedad**.

Resolver un problema, con y sin ordenador

En la vida real	En programación
Observas la situación	**Análisis**: qué datos hay, qué se espera
Piensas cómo resolverlo	**Diseño**: los pasos, sin ambigüedad
Lo haces	**Codificación**: lo traduces a un lenguaje
Compruebas que ha salido bien	**Pruebas**: lo ejecutas y verificas

Primero correcto. Después rápido.

Un programa rapidísimo que da resultados incorrectos no vale nada.
Uno correcto pero lento, al menos, sirve.

Cuatro ideas para todo el ciclo

- **Abstracción** — quedarte con lo que importa e ignorar el resto
- **Divide y vencerás** — un problema grande son muchos problemas pequeños
- **Encapsulación** — cada parte esconde *cómo* lo hace y enseña *qué* hace
- **Modularidad** — cada parte hace **una** cosa, y su nombre lo dice

No es teoría de examen: es la diferencia entre un programa que puedes modificar dentro de seis meses y uno que da miedo tocar.

2 · Algoritmos y programas

Algoritmo

> Secuencia ordenada de pasos, descrita sin ambigüedad, que resuelve un problema.

De *al-Juarismi*, matemático persa del siglo IX. Mil años más viejo que los ordenadores.

Una receta, el montaje de una estantería o la división a mano son algoritmos.

Las tres condiciones

- **Preciso** — cada paso dice exactamente qué hacer y en qué orden
  «Añade sal al gusto» no vale.
- **Definido** — con los mismos datos, siempre el mismo resultado
  Si no, no te puedes fiar ni puedes depurarlo.
- **Finito** — termina
  Lo que no termina no es un algoritmo, es un bucle infinito.

Algoritmo ≠ programa

Algoritmo: la idea.
Programa: esa idea escrita en un lenguaje que el ordenador ejecuta.

La receta de la tortilla es la misma en castellano o en inglés, en gas o en vitrocerámica.

Consecuencia práctica: el algoritmo que diseñas hoy te sirve en Scratch, en Java y en el lenguaje que aprendas dentro de cinco años.

Pseudocódigo

Lenguaje natural con estructura. No hay un estándar: importa que se entienda.

```
INICIO
    Leer numero
    Si numero < 1 o numero > 100 entonces
        Escribir "Fuera de rango"
    Si no
        Si numero módulo 2 = 0 entonces
            Escribir "Par"
        Si no
            Escribir "Impar"
        Fin_si
    Fin_si
FIN
```

Diagramas de flujo

Símbolo	Significa
Óvalo	Inicio o fin
Rectángulo	Acción o cálculo
Rombo	Decisión (sí / no)
Romboide	Entrada o salida
Flecha	Por dónde sigue

Diagrama cuando hay muchas decisiones o se lo explicas a alguien que no programa.

Pseudocódigo cuando el algoritmo es largo: cincuenta cajas no las dibuja nadie.

3 · Las fases de la programación

De la idea al programa en marcha

1. **Análisis** — ¿qué datos necesito? ¿qué resultado se espera?
2. **Diseño** — el algoritmo, y su **traza** con datos concretos
3. **Codificación** — traducirlo al lenguaje, respetando su sintaxis
4. **Pruebas** — datos normales, datos límite y datos absurdos
5. **Explotación y mantenimiento** — corregir, adaptar, ampliar

No es una escalera de un solo sentido: volver atrás es normal. Saltarse fases, no.

La traza

Recorrer el algoritmo **a mano**, con datos concretos, anotando cuánto vale cada dato en cada momento.

Es hacer tú de ordenador. Es aburrido.

Y es donde aparecen la mitad de los fallos, cuando arreglarlos todavía cuesta un minuto.

El mantenimiento es la fase larga

La mayor parte del tiempo de un programador no se va en escribir código nuevo, sino en **leer y modificar código que ya existe**.

Muchas veces, el tuyo de hace dos años.

Ciclo de vida: los modelos

- **Cascada** — una fase tras otra, sin volver atrás
  Obliga a acertar los requisitos a la primera. Casi nunca pasa.
- **Prototipos** — una maqueta rápida para concretar qué quiere el cliente
- **Incremental** — versiones pequeñas que funcionan, y se van ampliando
- **Espiral** — incremental + análisis de riesgos en cada vuelta

Hoy en la empresa: métodos ágiles (Scrum, Kanban). Ciclos de una o dos semanas y entrega continua.

4 · Paradigmas de programación

¿Le dices *cómo* o le dices *qué*?

Imperativo — describes cómo hacerlo, paso a paso

- No estructurado → *código espagueti* a base de `goto`
- **Estructurado** → secuencia, selección y repetición (UD04)
- **Orientado a objetos** → objetos con datos y comportamiento (UD03)

Declarativo — describes qué quieres, y el lenguaje resuelve el cómo

- Funcional (Haskell, Lisp) · Lógico (Prolog) · **SQL** (UD08)

En la práctica, mezclados

Java es orientado a objetos, pero desde Java 8 incorpora programación funcional (lambdas, *streams*).

Python, JavaScript o Kotlin están igual de mezclados.

La pregunta útil no es *«¿de qué paradigma es este lenguaje?»*, sino **«¿qué enfoque le va mejor a este problema?»**

5 · De tu código al procesador

Niveles de lenguaje

- **Lenguaje máquina** — binario puro. Un programa por modelo de procesador.
- **Ensamblador** — mnemotécnicos (`MOV` , `ADD` , `MUL`). Bajo nivel: sigues atado al hardware.
- **Alto nivel** — parecido al lenguaje humano e independiente del hardware.

El procesador solo entiende el primero. Todo lo demás **hay que traducirlo**.

Compilado vs. interpretado

	Compilado	Interpretado
Cuándo se traduce	Antes, entero	Durante, instrucción a instrucción
Qué produce	Un ejecutable	Nada, se ejecuta al vuelo
Errores de sintaxis	Al compilar	Al llegar a esa línea
Ejemplos	C, C++, Rust, Go	Python, JavaScript, PHP

Lo que más te afecta no es la velocidad: es cuándo te enteras de los fallos.

Java: las dos cosas

1. Escribes el código fuente → `Hola.java`
2. Se compila a **bytecode** → `Hola.class`
   Instrucciones que no son de ningún procesador concreto
3. La **JVM** traduce ese bytecode al lenguaje máquina de tu ordenador

Lo que cambia entre Windows, macOS o Linux es la JVM. **Tu programa, no.**

> *Write once, run anywhere*

¿Y no es lento?

Lo fue en 1998. Hoy la JVM lleva un compilador **JIT** (*just-in-time*):

- Detecta durante la ejecución las partes que más se repiten
- Las compila a código máquina nativo
- Optimiza con información que **solo existe mientras el programa corre**

El precio está en el arranque, no en el rendimiento sostenido.

6 · Scratch

Programar sin sintaxis

Por qué Scratch en un ciclo superior

Los bloques que no encajan, **no encajan**: no puedes equivocarte escribiendo.

Toda tu atención va a lo único que importa ahora: **la lógica**.

Cuando en la UD04 aparezca un `while`, no estarás aprendiendo qué es un bucle.
Solo cómo se escribe.

El entorno

- **Escenario** — el fondo. `x` de −240 a 240, `y` de −180 a 180
- **Objetos** (*sprites*) — cada personaje, con sus disfraces, sus sonidos y **su propio programa**
- **Bloques** — Movimiento, Apariencia, Sonido, Eventos, Control, Sensores, Operadores, Variables y Mis bloques

Que cada objeto tenga su propio código es exactamente la idea que en la UD03 llamaremos «objeto».

El juego de naves, paso a paso

1. **La nave se mueve** → bucle + condiciones
2. **La nave dispara** → `repetir hasta que`
3. **Los torpedos se acaban** → **variables**
4. **Aparecen enemigos** → azar + **funciones** (`definir inicio`)
5. **Las cosas chocan** → sensores
6. **Termina la partida** → **mensajes** entre objetos
7. **El enemigo final** → todo lo anterior junto

Lo que estás aprendiendo en realidad

En Scratch	En Java
`por siempre`, `repetir hasta que`	`while`, `for` — UD04
`si … entonces / si no`	`if / else` — UD04
Variables	Variables y tipos — UD02
Mis bloques	Métodos — UD04
Objetos y clones	Clases e instancias — UD03
`enviar` / `al recibir`	Comunicación entre objetos — UD03
`número aleatorio entre 1 y 10`	`Math.random()` — UD04

7 · Java

De dónde viene

- **1991** · Sun Microsystems busca un lenguaje para pequeños dispositivos
  Cada aparato, un procesador distinto → reescribir el programa entero
- **1995** · Se presenta como Java, justo cuando internet llena el mundo de máquinas distintas
- **2010** · Oracle compra Sun
- **Hoy** · **OpenJDK**, libre y de código abierto

Una versión cada 6 meses · una **LTS** cada 2 años: 8, 11, 17, 21 y **25**

Qué te vas a encontrar

- **Independiente de la plataforma** — bytecode + JVM
- **Orientado a objetos**, con programación funcional desde Java 8
- **Fuertemente tipado** — el compilador caza errores antes de ejecutar
- **Memoria automática** — el recolector de basura libera lo que ya no se usa
- **Biblioteca estándar enorme** y un ecosistema gigantesco

Tu primer programa

```
void main() {
    IO.println("¡Hola, DAM!");
}
```

```
java Hola.java
```

Ni compilar a mano, ni crear un proyecto, ni abrir un IDE.

Sigue habiendo compilación: simplemente no tienes que pedirla tú.

Lo mismo, en Java clásico

```
public class Hola {
    public static void main(String[] args) {
        System.out.println("¡Hola, DAM!");
    }
}
```

Cuatro conceptos que hoy no tocan: `class`, `public`, `static` y arrays.

Reconócelo bien: es como se ha escrito Java durante treinta años, y es lo que vas a encontrar en internet.

Que una IA te dé código que funciona no significa que te dé el código actual

Pídele un «hola mundo» en Java y comprueba qué versión te da.
Después, pregúntale por qué.

¿Empezamos?

apuntes.carlosprofe.es

Teoría, ejercicios con solución y la práctica de la unidad

Descargar estas diapositivas en PDF