

Tema 25

Programación estructurada

Estructuras básicas, funciones y procedimientos

1.1 Introducción

 Paradigma que organiza el código de forma lógica, secuencial y modular.

- Base para aprender POO, scripting y desarrollo web
- Aporta claridad, mantenibilidad y facilidad de depuración

1.2 Origen y objetivos

- ◆ Propuesta por Dijkstra en los años 70
- ◆ Alternativa al uso del `goto`

 Objetivos:

- Legibilidad
- Reutilización
- Reducción de errores
- Flujo controlado
- Colaboración efectiva

1.3 Estructuras básicas de control

Tipos principales:

- Secuencia → instrucciones en orden
- Selección → `if`, `else`, `switch-case`, ternario
- Iteración → `for`, `while`, `do-while`

Ejemplo:

```
if (nota >= 5) {  
    System.out.println("Aprobado");  
} else {  
    System.out.println("Suspenso");  
}
```

1.4 Funciones y procedimientos

Funciones

- Devuelven valores con `return`
- Encapsulan lógica

Procedimientos

- No devuelven nada (`void` en Java)

Parámetros y ámbito

- Java → paso **por valor** (objetos: referencia modificable)
- Ámbito local/global: depende del bloque o clase
- **Recursividad**: llamada a sí misma + caso base

1.5 Diseño modular

🧩 Organización del programa en funciones/métodos

✅ Ventajas:

- Reutilización
- Pruebas independientes
- Código limpio y mantenible

📌 Buenas prácticas:

- Funciones cortas y con nombre claro
- Evitar duplicación
- Uso coherente de `return`

1.6 Diseño top-down, cohesión y acoplamiento

Diseño top-down

- Se parte del problema general y se divide en subproblemas
- Se planifica antes de codificar (pseudocódigo, diagramas)

Cohesión

- Alta cohesión: función hace **una sola cosa clara**
- Favorece comprensión y mantenimiento

Acoplamiento

- Bajo acoplamiento = funciones **independientes**
- Mejora pruebas, escalabilidad y modularidad

1.7 Trazado y depuración estructurada

 La estructura lógica permite depurar paso a paso

Herramientas útiles:

- `System.out.println()`
- Depurador del IDE (breakpoints, inspección de variables)

 Apoyos:

- Indentación
- Comentarios explicativos
- Revisión de condiciones y bucles

1.8 Buenas prácticas

 Evita:

- Funciones excesivamente largas
- Nombres ambiguos
- Código duplicado

 Aplica:

- Nombres descriptivos
- Modulación en bloques pequeños
- Comentarios necesarios y útiles

1.9 Eficiencia algorítmica

 Comparar soluciones estructuradas:


- **Iterativa vs. recursiva**
 - Ej: factorial, suma, búsqueda

Notación Big-O

- $O(1)$, $O(n)$, $O(n^2)$... para medir eficiencia

 Técnicas:

- **Memoización**
- **Refactorización**

 ¿Una solución clara también es eficiente?

1.10 Comparación con otros paradigmas

Característica	Programación estructurada	Programación orientada a objetos
Unidad principal	Funciones	Clases y objetos
Aplicación típica	Algoritmos, lógica	Aplicaciones complejas
Manejo de datos	Directo	Encapsulado
Reutilización	Modular	Herencia, polimorfismo
Ejemplos de uso	Scripts, algoritmos	Interfaces gráficas, sistemas

Conclusión

 La programación estructurada es la base de:

- La lógica computacional
- El diseño claro y mantenible
- La transición hacia paradigmas más complejos como POO

 Dominarla permite escribir código comprensible, modular y eficaz.