



Universidad  
**Tecmilenio**®

# ESTRUCTURAS DE DATOS

**Alumno:** Edwin Manuel Aguilar Márquez

**Actividad 3**

**Profesor:** Adalberto Emmanuel Rojas  
Perea

## Documento de Explicación y Reporte de Ejecución

### 1. Serie de Fibonacci (Recursiva)

#### Diseño

- **Objetivo:** Calcular el enésimo número de la serie de Fibonacci utilizando recursión.
- **Entrada:** Un número entero  $n$  que indica la posición en la serie.
- **Salida:** El valor de Fibonacci en la posición  $n$  o la serie completa hasta  $n$ .
- **Estrategia:**
  - Caso base:  $F(0) = 0$ ,  $F(1) = 1$ .
  - Caso recursivo:  $F(n) = F(n-1) + F(n-2)$ .
- **Complejidad:** Exponencial  $O(2^n)$  en esta versión simple (sin memoización).

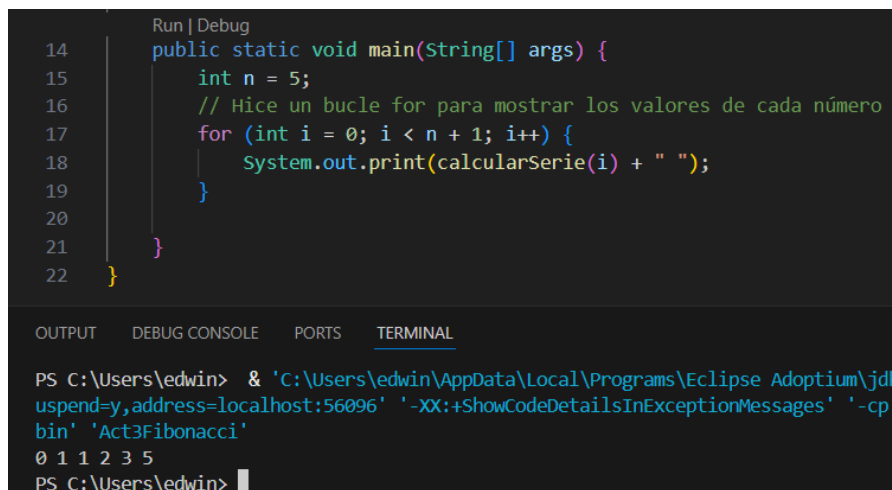
#### Implementación

- Método `calcularSerie(int n)` que retorna un entero.
- Uso de condiciones `if` para los casos base y la llamada recursiva para  $n > 1$ .
- Bucle `for` en `main` para mostrar la serie completa hasta  $n$ .

#### Funcionamiento

1. Si  $n$  es 0 o 1, retorna directamente.
2. Si  $n > 1$ , llama recursivamente a `calcularSerie(n-1)` y `calcularSerie(n-2)`.
3. La suma de las llamadas recursivas devuelve el valor en la posición  $n$ .

#### Capturas de pantalla



```
Run | Debug
14 public static void main(String[] args) {
15     int n = 5;
16     // Hice un bucle for para mostrar los valores de cada número
17     for (int i = 0; i < n + 1; i++) {
18         System.out.print(calcularSerie(i) + " ");
19     }
20 }
21 }
22 }
```

OUTPUT    DEBUG CONSOLE    PORTS    TERMINAL

```
PS C:\Users\edwin> & 'C:\Users\edwin\AppData\Local\Programs\Eclipse Adoptium\jdk
uspend=y,address=localhost:56096' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp'
bin' 'Act3Fibonacci'
0 1 1 2 3 5
PS C:\Users\edwin>
```

## 2. Suma de Subconjuntos (Subset Sum)

### Diseño

- **Objetivo:** Determinar si existe un subconjunto de números enteros que suma un valor objetivo.
- **Entrada:** Un arreglo de enteros y un valor objetivo objetivo.
- **Salida:** true si existe el subconjunto, false en caso contrario.
- **Estrategia:**
  - Recursión considerando dos posibilidades para cada elemento: incluirlo o no.
  - Caso base:  $\text{objetivo} == 0 \rightarrow \text{true}$ ,  $n == 0 \parallel \text{objetivo} < 0 \rightarrow \text{false}$ .
- **Complejidad:** Exponencial  $O(2^n)$  en el peor caso, ya que explora todas las combinaciones.

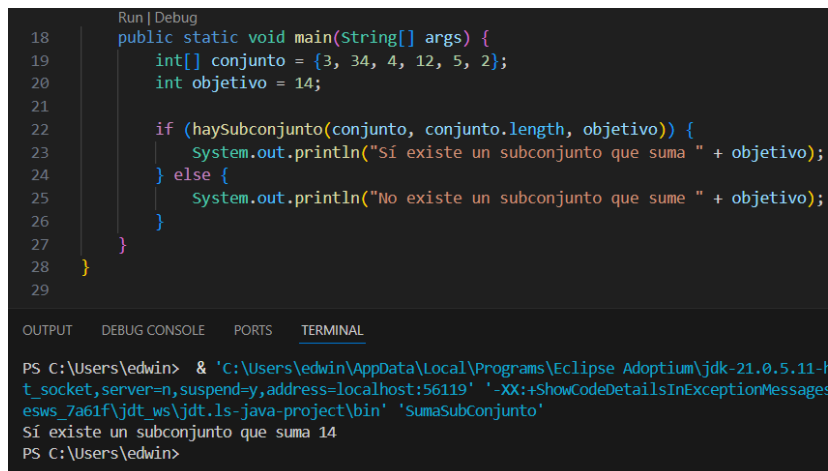
### Implementación

- Método `haySubconjunto(int[] conjunto, int n, int objetivo)` retorna booleano.
- Llamadas recursivas para incluir o excluir el último elemento del arreglo.
- main para probar con un conjunto de ejemplo {3, 34, 4, 12, 5, 2} y objetivo 14.

### Funcionamiento

1. Revisa si el objetivo se alcanzó o si no hay elementos.
2. Explora recursivamente las combinaciones posibles.
3. Retorna true si alguna combinación suma el objetivo.

### Capturas de pantalla



```
Run | Debug
18 public static void main(String[] args) {
19     int[] conjunto = {3, 34, 4, 12, 5, 2};
20     int objetivo = 14;
21
22     if (haySubconjunto(conjunto, conjunto.length, objetivo)) {
23         System.out.println("Sí existe un subconjunto que suma " + objetivo);
24     } else {
25         System.out.println("No existe un subconjunto que suma " + objetivo);
26     }
27 }
28 }
29

OUTPUT  DEBUG CONSOLE  PORTS  TERMINAL
PS C:\Users\edwin> & 'C:\Users\edwin\AppData\Local\Programs\Eclipse Adoptium\jdk-21.0.5.11-hotspot\bin\java.exe' -Xmx1024m -Djava.net.preferIPv4Stack=true -Djava.awt.headless=true -Djava.library.path=C:\Users\edwin\AppData\Local\Programs\Eclipse Adoptium\jdk-21.0.5.11-hotspot\bin\java.exe -jar C:\Users\edwin\AppData\Local\Programs\Eclipse Adoptium\jdk-21.0.5.11-hotspot\bin\java.exe esws_7a61f\jdt_ws\jdt.ls-java-project\bin\SumaSubconjunto
Sí existe un subconjunto que suma 14
PS C:\Users\edwin>
```

### 3. Sudoku (Backtracking)

#### Diseño

- **Objetivo:** Resolver un tablero de Sudoku de 9x9 usando backtracking.
- **Entrada:** Una matriz 9x9 con algunos valores llenos y ceros en celdas vacías.
- **Salida:** Tablero completo con todos los números válidos según reglas de Sudoku.
- **Estrategia:**
  - Buscar la primera celda vacía.
  - Probar números del 1 al 9.
  - Validar fila, columna y subcuadro 3x3.
  - Si ningún número es válido → backtracking (retroceder).

#### Implementación

- Función solveSudoku(int[][] tablero) que aplica backtracking.
- Función isValid() para verificar si un número puede colocarse.
- Función mostrarTablero() para imprimir el tablero final.

#### Funcionamiento

1. Recorre todas las celdas en búsqueda de vacíos.
2. Intenta colocar un número válido.
3. Si se completa, retorna true; si no, retrocede.
4. El tablero final se imprime mostrando los subcuadros delimitados.

## Capturas de pantalla

```
1 public class Sudoku {
71     // Imprime el tablero
72     private static void mostrarTablero(int[][] tablero) {
73         for (int f = 0; f < N; f++) {
74             if (f % 3 == 0 && f != 0) {
75                 System.out.println(x:"-----+-----+-----");
76             }
77             for (int c = 0; c < N; c++) {
78                 if (c % 3 == 0 && c != 0) {
79                     System.out.print(s:" | ");
80                 }
81                 System.out.print(" " + tablero[f][c] + " ");
82             }
83             System.out.println();

```

OUTPUT    DEBUG CONSOLE    PORTS    TERMINAL

```
t_socket,server=n,suspend=y,address=localhost:56147' '-XX:+ShowCodeDetailsInExceptionMes
esws_7a61f\jdt_ws\jdt.ls-java-project\bin' 'Sudoku'
Sudoku resuelto:
5 3 4 | 6 7 8 | 9 1 2
6 7 2 | 1 9 5 | 3 4 8
1 9 8 | 3 4 2 | 5 6 7
-----+-----+-----
8 5 9 | 7 6 1 | 4 2 3
4 2 6 | 8 5 3 | 7 9 1
7 1 3 | 9 2 4 | 8 5 6
-----+-----+-----
9 6 1 | 5 3 7 | 2 8 4
2 8 7 | 4 1 9 | 6 3 5
3 4 5 | 2 8 6 | 1 7 9
PS C:\Users\edwin>
```

## Reflexión sobre Recursividad y Divide y Vencerás

- La recursividad permite resolver problemas complejos dividiéndolos en subproblemas más simples, como en Fibonacci y Subset Sum.
- Los algoritmos de divide y vencerás facilitan el manejo de grandes espacios de solución, haciendo más manejable la búsqueda de combinaciones o configuraciones válidas.
- En Sudoku, el backtracking combina recursión con búsqueda sistemática para explorar soluciones posibles, mostrando cómo la recursión puede ser aplicada en problemas de decisión y construcción.
- Comprender estas técnicas es fundamental para optimizar y estructurar soluciones a problemas que serían prácticamente imposibles de resolver de manera iterativa simple.