Universidad Tecmilenio

1 DE SEPTIEMBRE DE 2025

ESTRUCTURA DE DATOS
ACTIVIDAD 3

JUAN ESTEBAN CAMPOS CRUZ
AL05064315

IDS

DESCRIPCIÓN

El objetivo de esta actividad es que implementes y apliques tus conocimientos de recursividad y algoritmos de divide y vencerás mediante la resolución de problemas clásicos en programación. Se te proporcionará una serie de problemas que deberás resolver utilizando técnicas recursivas, recursividad indirecta y algoritmos de backtracking. Desarrollarás programas en Java para cada problema, documentando tu proceso y razonamiento.

OBJETIVO

- Aplicar conceptos de recursividad directa e indirecta.
- Implementar algoritmos de backtracking y divide y vencerás.
- Desarrollar la habilidad para resolver problemas complejos mediante la descomposición en subproblemas

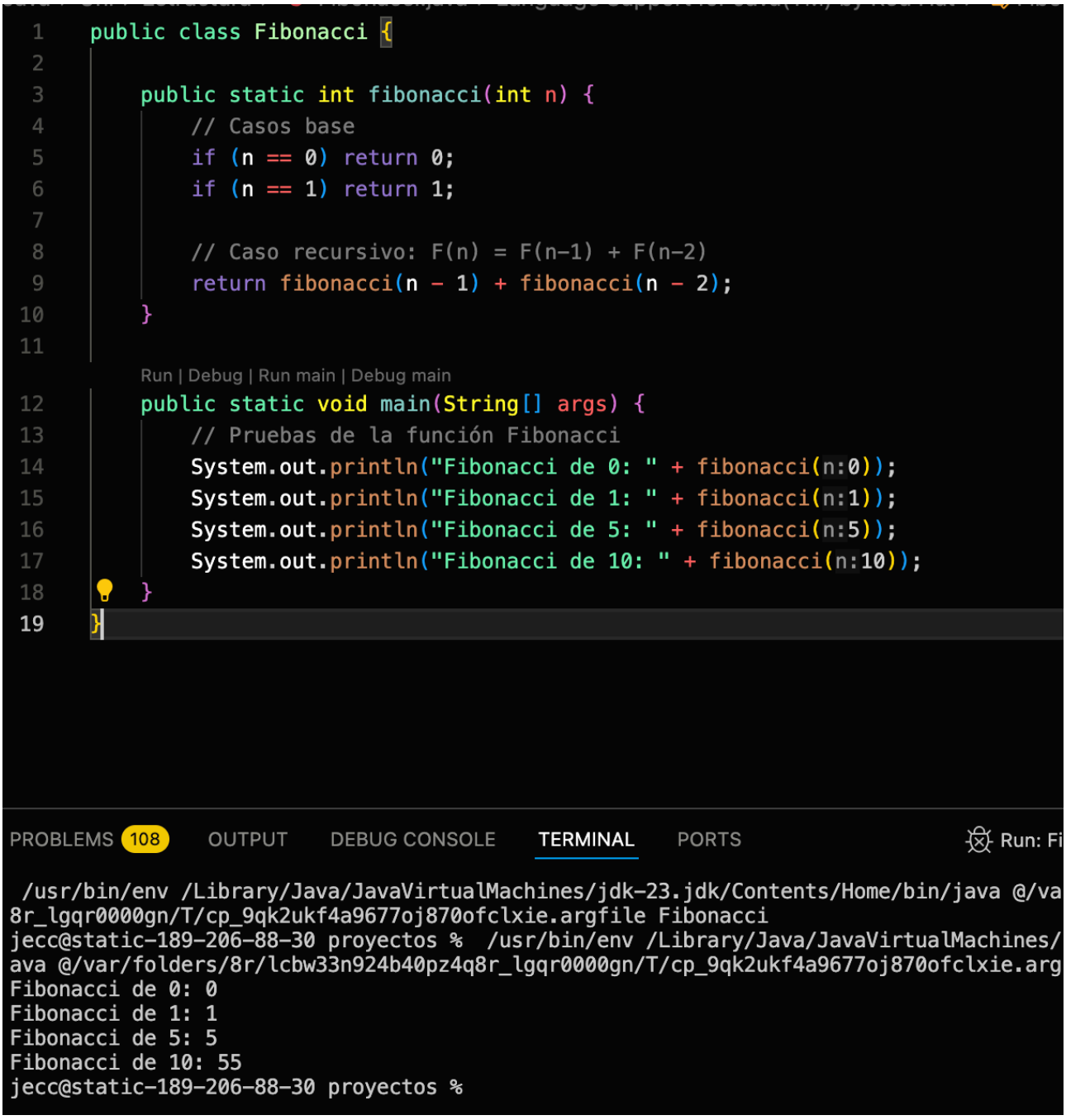
INSTRUCCIONES

1. Primer problema: Serie de Fibonacci recursiva. Implementa una función recursiva en Java que calcule el enésimo número en la serie de Fibonacci. Asegúrate de incluir tanto el caso base como el caso recursivo.

2. Segundo problema: Suma de subconjuntos (Subset Sum). Desarrolla un algoritmo recursivo para determinar si existe un subconjunto de un conjunto dado de enteros que sume un valor objetivo.

3. Tercer problema: Algoritmo de backtracking para el problema del sudoku. Implementa un algoritmo de backtracking que resuelva un Sudoku. El programa debe llenar las celdas vacías del tablero de Sudoku dado.

PRIMER PROBLEMA: SERIE DE FIBONACCI RECURSIVA



```java
public class Fibonacci {

    public static int fibonacci(int n) {
        // Casos base
        if (n == 0) return 0;
        if (n == 1) return 1;

        // Caso recursivo: F(n) = F(n-1) + F(n-2)
        return fibonacci(n - 1) + fibonacci(n - 2);
    }

    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        // Pruebas de la función Fibonacci
        System.out.println("Fibonacci de 0: " + fibonacci(n:0));
        System.out.println("Fibonacci de 1: " + fibonacci(n:1));
        System.out.println("Fibonacci de 5: " + fibonacci(n:5));
        System.out.println("Fibonacci de 10: " + fibonacci(n:10));
    }
}
```

```
PROBLEMS 108   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   Run: Fi

 /usr/bin/env /Library/Java/JavaVirtualMachines/jdk-23.jdk/Contents/Home/bin/java @/va
8r_lgqr0000gn/T/cp_9qk2ukf4a9677oj870ofclxie.argfile Fibonacci
jecc@static-189-206-88-30 proyectos %  /usr/bin/env /Library/Java/JavaVirtualMachines/
ava @/var/folders/8r/lcbw33n924b40pz4q8r_lgqr0000gn/T/cp_9qk2ukf4a9677oj870ofclxie.arg
Fibonacci de 0: 0
Fibonacci de 1: 1
Fibonacci de 5: 5
Fibonacci de 10: 55
jecc@static-189-206-88-30 proyectos %
```

Explicación del Código

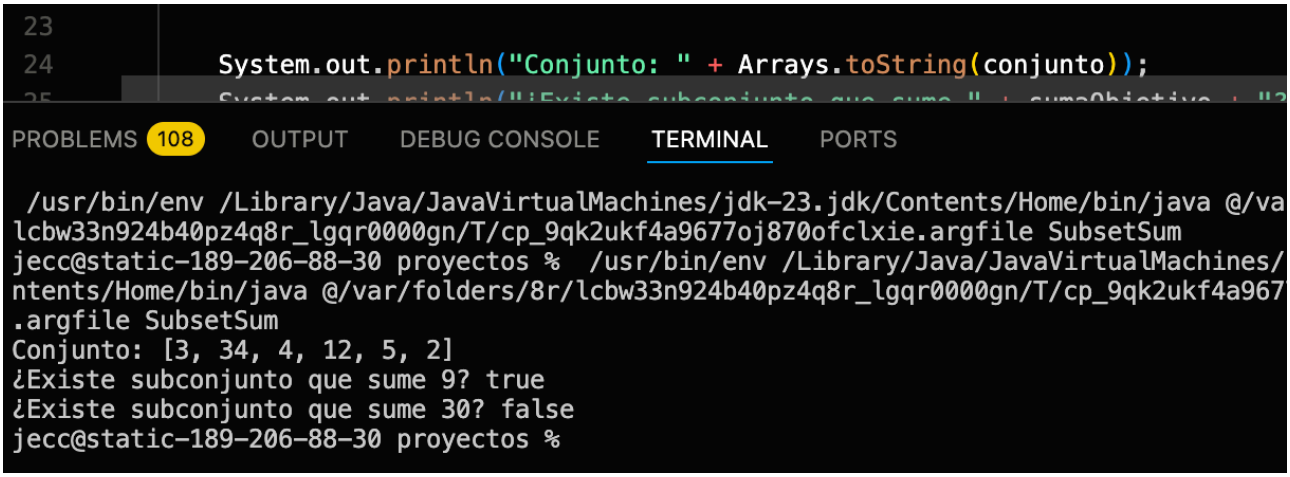
La serie de Fibonacci es una secuencia donde cada número es la suma de los dos anteriores: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

El algoritmo recursivo implementa directamente esta definición:

- Casos base:
  - Si n = 0, retorna 0
  - Si n = 1, retorna 1
- Caso recursivo:
  - Para n > 1, retorna fibonacci(n-1) + fibonacci(n-2)

SEGUNDO PROBLEMA: SUMA DE SUBCONJUNTOS (SUBSET SUM)

```
1   mport java.util.Arrays;
2
3   ublic class SubsetSum {
4
5       public static boolean existeSubconjuntoSuma(int[] conjunto, int n, int sum
6           // Casos base
7           if (sumaObjetivo == 0) return true; // Subconjunto vacío suma 0
8           if (n == 0 && sumaObjetivo != 0) return false; // Sin elementos y suma
9
10          // Si el último elemento es mayor que la suma objetivo, ignorarlo
11          if (conjunto[n-1] > sumaObjetivo) {
12              return existeSubconjuntoSuma(conjunto, n-1, sumaObjetivo);
13          }
14
15          // Probar incluyendo o excluyendo el último elemento
16          return existeSubconjuntoSuma(conjunto, n-1, sumaObjetivo) ||
17                 existeSubconjuntoSuma(conjunto, n-1, sumaObjetivo - conjunto[n-
18      }
19
        Run | Debug | Run main | Debug main
20      public static void main(String[] args) {
21          int[] conjunto = {3, 34, 4, 12, 5, 2};
22          int sumaObjetivo = 9;
23
24          System.out.println("Conjunto: " + Arrays.toString(conjunto));
```



Explicación del Código

El problema de la suma de subconjuntos busca determinar si existe un subconjunto de números que sumen exactamente un valor objetivo.

El algoritmo recursivo funciona de la siguiente manera:

- Casos base:
  - Si la suma objetivo es 0, retorna true (el subconjunto vacío suma 0)
  - Si no hay elementos y la suma objetivo no es 0, retorna false
- Caso recursivo:
  - Si el último elemento es mayor que la suma objetivo, se ignora y se verifica con los elementos restantes
  - Se prueban dos posibilidades: excluir el último elemento o incluirlo restando su valor a la suma objetivo

TERCER PROBLEMA: SOLUCIONADOR DE SUDOKU CON BACKTRACKING

```
    private static final int TAMANIO = 9;
    static boolean resolverSudoku(int[][] tablero) {
        for (int fila = 0; fila < TAMANIO; fila++) {
            for (int col = 0; col < TAMANIO; col++) {
                if (tablero[fila][col] == 0) { // Encontrar celda vacía
                    for (int num = 1; num <= 9; num++) { // Probar números de
                        if (esValido(tablero, fila, col, num)) {
                            tablero[fila][col] = num; // Asignar número
```

PROBLEMS 108 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
 /usr/bin/env /Library/Java/JavaVirtualMachines/jdk-23.jdk/Contents/Home/bin/java @/va
lcbw33n924b40pz4q8r_lgqr0000gn/T/cp_9qk2ukf4a9677oj870ofclxie.argfile Sodoku
jecc@static-189-206-88-30 proyectos %  /usr/bin/env /Library/Java/JavaVirtualMachines/
ntents/Home/bin/java @/var/folders/8r/lcbw33n924b40pz4q8r_lgqr0000gn/T/cp_9qk2ukf4a967
.argfile Sodoku
Tablero inicial:
5 3 . | . 7 . | . . .
6 . . | 1 9 5 | . . .
. 9 8 | . . . | . 6 .
------+-------+------
8 . . | . 6 . | . . 3
4 . . | 8 . 3 | . . 1
7 . . | . 2 . | . . 6
------+-------+------
. 6 . | . . . | 2 8 .
. . . | 4 1 9 | . . 5
. . . | . 8 . | . 7 9

Solución encontrada:
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
jecc@static-189-206-88-30 proyectos %
```

Explicación del Código

El algoritmo de backtracking para resolver Sudoku funciona de la siguiente manera:

1. Encontrar celda vacía: Busca la primera celda con valor 0 (vacía)
2. Probar valores: Intenta colocar números del 1 al 9 en la celda vacía
3. Verificar validez: Comprueba si el número es válido según las reglas del Sudoku
    - No debe repetirse en la misma fila
    - No debe repetirse en la misma columna
    - No debe repetirse en la subcuadrícula 3x3
4. Llamada recursiva: Si el número es válido, se asigna y se llama recursivamente a la función
5. Backtracking: Si la llamada recursiva no lleva a una solución, se deshace la asignación (se vuelve a poner 0) y se prueba con el siguiente número

Conclusión

Estos tres problemas demuestran la aplicación de diferentes técnicas algorítmicas:

1. **Fibonacci**: Recursividad directa con casos base simples
2. **Subset Sum**: Recursividad con múltiples llamadas para explorar diferentes opciones
3. **Sudoku**: Backtracking con verificación de restricciones

Cada técnica tiene sus ventajas y desventajas en términos de complejidad y aplicabilidad, pero todas comparten el principio de descomponer problemas complejos en subproblemas más simples, que es fundamental en el diseño de algoritmos.