UNIVERSIDAD TECMILENIO

Actividad 3

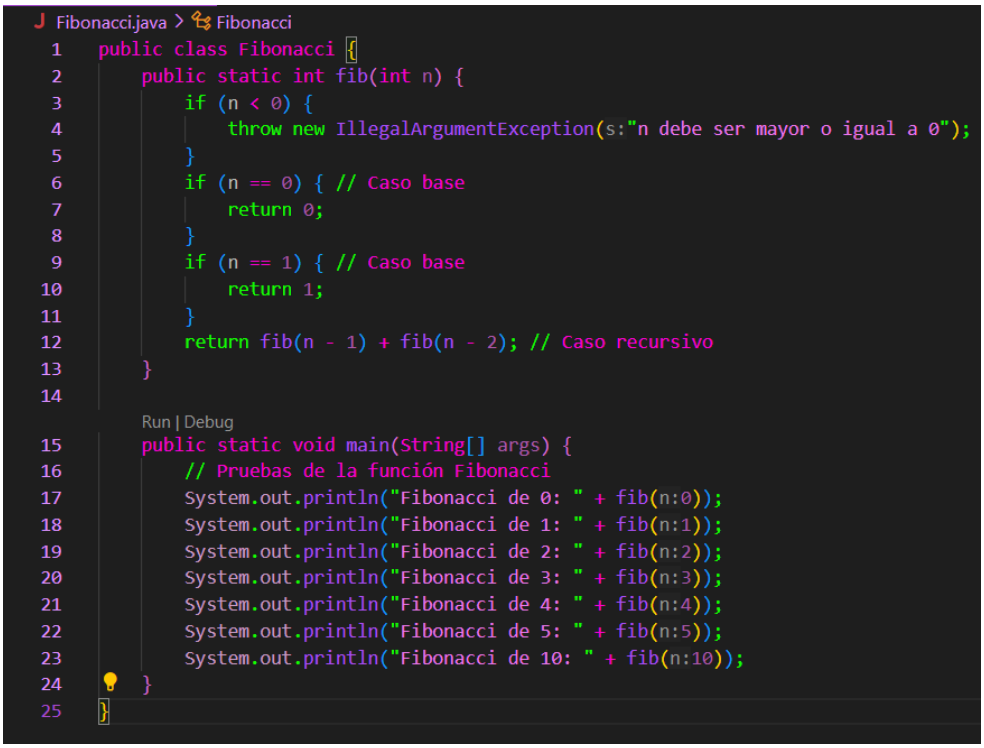
Estructuras de datos

Adalberto Emmanuel
Rojas Perea

Leonardo Pereda Chánez
AL03087353

Documentación de código

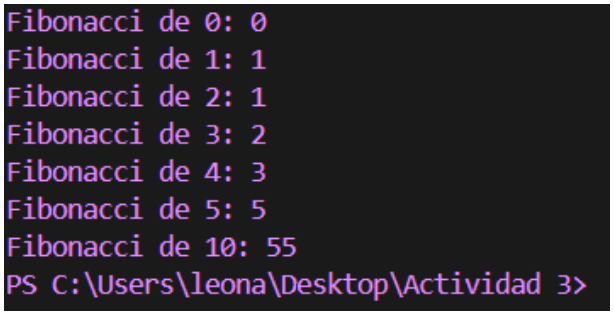
Fibonacci.java



```java
Fibonacci.java > Fibonacci
public class Fibonacci {
    public static int fib(int n) {
        if (n < 0) {
            throw new IllegalArgumentException(s:"n debe ser mayor o igual a 0");
        }
        if (n == 0) { // Caso base
            return 0;
        }
        if (n == 1) { // Caso base
            return 1;
        }
        return fib(n - 1) + fib(n - 2); // Caso recursivo
    }

    Run | Debug
    public static void main(String[] args) {
        // Pruebas de la función Fibonacci
        System.out.println("Fibonacci de 0: " + fib(n:0));
        System.out.println("Fibonacci de 1: " + fib(n:1));
        System.out.println("Fibonacci de 2: " + fib(n:2));
        System.out.println("Fibonacci de 3: " + fib(n:3));
        System.out.println("Fibonacci de 4: " + fib(n:4));
        System.out.println("Fibonacci de 5: " + fib(n:5));
        System.out.println("Fibonacci de 10: " + fib(n:10));
    }
}
```

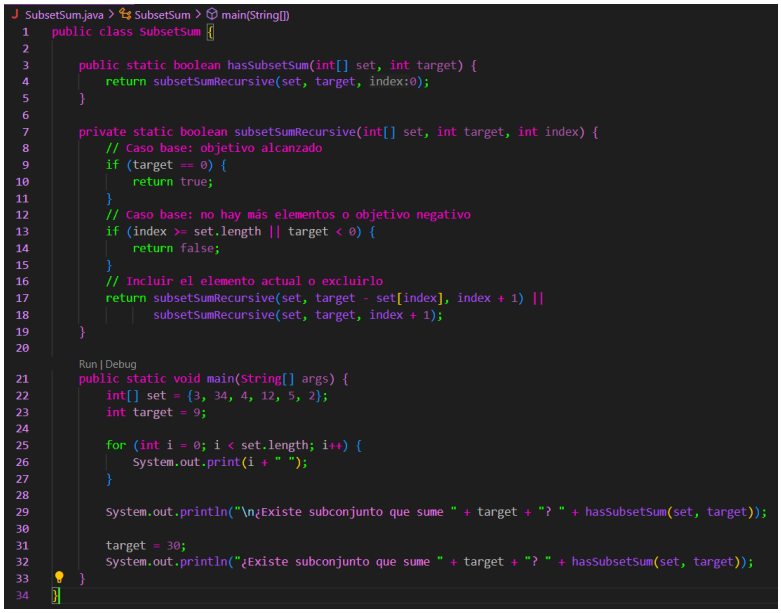
La clase **Fibonacci.java** utiliza la técnica de recursividad, en donde la función se llama a sí misma, dividiéndose en 2 llamadas recursivas que a su vez se dividen hasta llegar a los casos base.

El método main llama varias veces al método de fib() con el fin de que se realicen pruebas del funcionamiento del método recursivo.



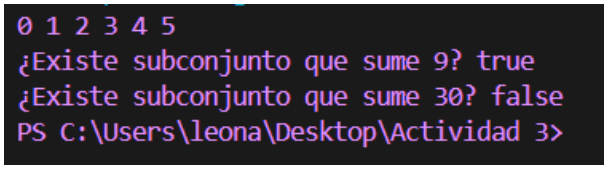
```
Fibonacci de 0: 0
Fibonacci de 1: 1
Fibonacci de 2: 1
Fibonacci de 3: 2
Fibonacci de 4: 3
Fibonacci de 5: 5
Fibonacci de 10: 55
PS C:\Users\leona\Desktop\Actividad 3>
```

SubsetSum.java



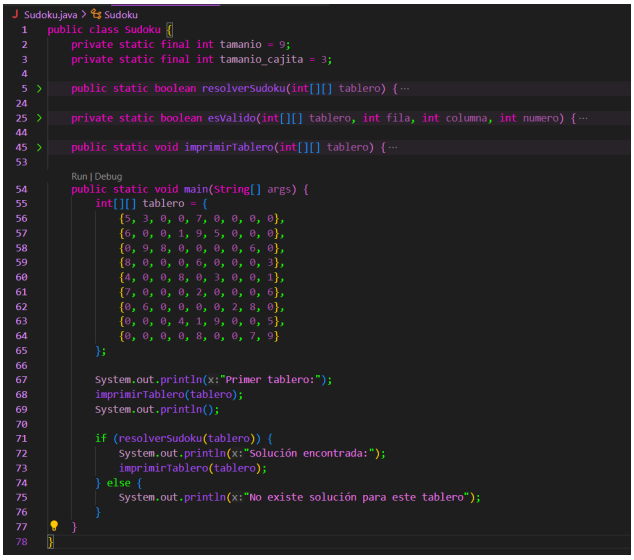
```java
public class SubsetSum {

    public static boolean hasSubsetSum(int[] set, int target) {
        return subsetSumRecursive(set, target, index:0);
    }

    private static boolean subsetSumRecursive(int[] set, int target, int index) {
        // Caso base: objetivo alcanzado
        if (target == 0) {
            return true;
        }
        // Caso base: no hay más elementos o objetivo negativo
        if (index >= set.length || target < 0) {
            return false;
        }
        // Incluir el elemento actual o excluirlo
        return subsetSumRecursive(set, target - set[index], index + 1) ||
               subsetSumRecursive(set, target, index + 1);
    }

    Run | Debug
    public static void main(String[] args) {
        int[] set = {3, 34, 4, 12, 5, 2};
        int target = 9;

        for (int i = 0; i < set.length; i++) {
            System.out.print(i + " ");
        }

        System.out.println("\n¿Existe subconjunto que sume " + target + "? " + hasSubsetSum(set, target));

        target = 30;
        System.out.println("¿Existe subconjunto que sume " + target + "? " + hasSubsetSum(set, target));
    }
}
```

En la clase **SubsetSum.java**, el algoritmo divide el problema original en subproblemas más pequeños de manera recursiva, utilizando la técnica de “Divide y vencerás”, donde los resuelve de manera independiente y combina sus soluciones.



```
0 1 2 3 4 5
¿Existe subconjunto que sume 9? true
¿Existe subconjunto que sume 30? false
PS C:\Users\leona\Desktop\Actividad 3>
```

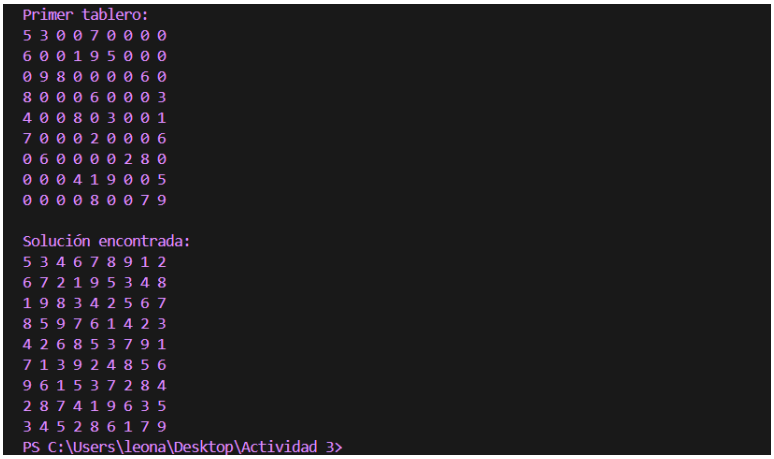
Sudoku.java



```java
public class Sudoku {
    private static final int tamanio = 9;
    private static final int tamanio_cajita = 3;

    public static boolean resolverSudoku(int[][] tablero) {...

    private static boolean esValido(int[][] tablero, int fila, int columna, int numero) {...

    public static void imprimirTablero(int[][] tablero) {...

    Run | Debug
    public static void main(String[] args) {
        int[][] tablero = {
            {5, 3, 0, 0, 7, 0, 0, 0, 0},
            {6, 0, 0, 1, 9, 5, 0, 0, 0},
            {0, 9, 8, 0, 0, 0, 0, 6, 0},
            {8, 0, 0, 0, 6, 0, 0, 0, 3},
            {4, 0, 0, 8, 0, 3, 0, 0, 1},
            {7, 0, 0, 0, 2, 0, 0, 0, 6},
            {0, 6, 0, 0, 0, 0, 2, 8, 0},
            {0, 0, 0, 4, 1, 9, 0, 0, 5},
            {0, 0, 0, 0, 8, 0, 0, 7, 9}
        };

        System.out.println(x:"Primer tablero:");
        imprimirTablero(tablero);
        System.out.println();

        if (resolverSudoku(tablero)) {
            System.out.println(x:"Solución encontrada:");
            imprimirTablero(tablero);
        } else {
            System.out.println(x:"No existe solución para este tablero");
        }
    }
}
```

La clase **Sudoku.java** toma como técnica el backtracking para poder resolver el tablero del sudoku, realiza una búsqueda que construye soluciones hasta que ya no se puede avanzar más.

Los métodos primero encuentran la primera celda vacía, luego va probando con los números del 1 al 9 y los verifica si ese mismo número está dentro de la fila, columna o cajita, si es válido coloca el número y sigue avanzando hasta el caso de que ningún número funciona, retrocede y prueba otras alternativas.



```
Primer tablero:
5 3 0 0 7 0 0 0 0
6 0 0 1 9 5 0 0 0
0 9 8 0 0 0 0 6 0
8 0 0 0 6 0 0 0 3
4 0 0 8 0 3 0 0 1
7 0 0 0 2 0 0 0 6
0 6 0 0 0 0 2 8 0
0 0 0 4 1 9 0 0 5
0 0 0 0 8 0 0 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
PS C:\Users\leona\Desktop\Actividad 3>
```

Reflexión

La recursividad y los algoritmos de divide y vencerás son fundamentales para la resolución de problemas complejos, primeramente con la recursividad, es una herramienta potente que aparte de muestra código más legible, hace más sencillo de abordar problemas complejos, mientras que los algoritmos de divide y vencerás, divide problemas en subproblemas independientes, y cada uno de estos subproblemas se resuelven con la misma estrategia de división, y llevar a cabo su implementación es más sencilla, poniendo como ejemplo la torre de Hanoi, divides el problema en 3, colocar los n-1 discos en la torre auxiliar; colocar el disco más grande en la torre de destino; colocar los n-1 discos de la torre auxiliar en la torre de destino, su escritura en código es muy similar a como si fuese un pseudocódigo.