Sudoku Solver - Java Code

```java
// This is the main class for the Sudoku solver
public class Solution {

    // ---
    // isSafe Method
    // ---

    // Checks if a number 'dig' can be safely placed at a specific cell (row, col)
    static boolean isSafe(int[][] mat, int row, int col, int dig) {

        // ---
        // 1. Check the Row
        // ---

        // Loop through all columns in the current row
        for (int j = 0; j < 9; j++) {
            // If the digit 'dig' is already present in this row, it's not safe
            if (mat[row][j] == dig) return false;
        }

        // ---
        // 2. Check the Column
        // ---

        // Loop through all rows in the current column
        for (int i = 0; i < 9; i++) {
            // If the digit 'dig' is already present in this column, it's not safe
            if (mat[i][col] == dig) return false;
        }

        // ---
        // 3. Check the 3x3 Subgrid
        // ---

        // Calculate the starting row of the 3x3 subgrid
        int srow = (row / 3) * 3;
        // Calculate the starting column of the 3x3 subgrid
        int scol = (col / 3) * 3;

        // Loop through the rows of the 3x3 subgrid
        for (int i = srow; i < srow + 3; i++) {
            // Loop through the columns of the 3x3 subgrid
            for (int j = scol; j < scol + 3; j++) {
                // If the digit 'dig' is already present in this subgrid, it's not safe
                if (mat[i][j] == dig) return false;
            }
        }

        // If the digit passes all three checks, it's safe to place
        return true;
    }

    // ---
    // helper Method (Recursive Backtracking)
    // ---

    // This is the recursive function that solves the Sudoku puzzle using backtracking
    static boolean helper(int[][] mat, int row, int col) {
        // Base case: If we've successfully filled all rows (i.e., row becomes 9), the puzzle is solved
        if (row == 9) return true;

        // Calculate the coordinates of the next cell to process
        int nextRow = row, nextCol = col + 1;

        // If we reach the end of the current row (col becomes 9), move to the next row and reset column
        if (nextCol == 9) {
            nextRow = row + 1;
        nextCol = 0;
    }

    // ---
    // Skip Filled Cells
    // ---

    // If the current cell is not empty (i.e., it has a pre-filled value),
    // we skip it and move on to the next cell recursively
    if (mat[row][col] != 0) {
        return helper(mat, nextRow, nextCol);
    }

    // ---
    // Try Digits 1 to 9
    // ---

    // Loop through possible digits from 1 to 9 to find a valid one for the current empty cell
    for (int dig = 1; dig <= 9; dig++) {
        // Check if placing the current digit is safe
        if (isSafe(mat, row, col, dig)) {
            // If it's safe, place the digit in the current cell
            mat[row][col] = dig;

            // Recursively call the helper function for the next cell
            // If the recursive call returns true (meaning a solution was found down this path),
            // we've solved the puzzle, so we return true immediately
            if (helper(mat, nextRow, nextCol)) return true;

            // ---
            // Backtrack
            // ---

            // If the previous recursive call didn't lead to a solution, it means our current choice
            // We reset the cell to 0 to "undo" our choice and try the next digit.
            mat[row][col] = 0;
        }
    }

    // If the loop finishes without finding any digit (1-9) that fits,
    // it means there's no solution from this path, so we return false
    return false;
}

// ---
// solveSudoku Method (Main entry point)
// ---

// This is the public method that starts the solving process
static void solveSudoku(int[][] mat) {
    // Start the backtracking process from the top-left cell (0, 0)
    helper(mat, 0, 0);
}

// ---
// printBoard Method
// ---

// A utility method to print the Sudoku board to the console
static void printBoard(int[][] mat) {
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            // Print each number followed by a space
            System.out.print(mat[i][j] + " ");
        }
        // Move to the next line after each row is printed
        System.out.println();
    }
}

// ---
// main Method (Driver code)
```

```
    // ---

    // The main method where the program execution begins
    public static void main(String[] args) {
        // Define the initial Sudoku puzzle board
        int[][] board = {
            {3, 0, 6, 5, 0, 8, 4, 0, 0},
            {5, 2, 0, 0, 0, 0, 0, 0, 0},
            {0, 8, 7, 0, 0, 0, 0, 3, 1},
            {0, 0, 3, 0, 0, 0, 0, 6, 8},
            {9, 0, 0, 8, 6, 3, 0, 0, 5},
            {0, 5, 0, 0, 9, 0, 6, 0, 0},
            {1, 3, 0, 0, 0, 0, 2, 5, 0},
            {0, 0, 0, 0, 0, 0, 0, 7, 4},
            {0, 0, 5, 2, 0, 6, 3, 0, 0}
        };

        // Print the original unsolved Sudoku board
        System.out.println("Original Sudoku:");
        printBoard(board);

        // Call the solver method to solve the board in-place
        solveSudoku(board);

        // Print a new line for spacing
        System.out.println("\nSolved Sudoku:");
        // Print the now-solved Sudoku board
        printBoard(board);
    }
}
```