```
import java.util.Scanner;

/* A Backtracking program in
Java to solve Sudoku problem */
class SudokuSolver
{
    public static boolean isSafe(int[][] board,int row, int col,int num)
    {
        // Row has the unique (row-clash)
        for (int d = 0; d < board.length; d++)
        {

            // Check if the number we are trying to
            // place is already present in
            // that row, return false;
            if (board[row][d] == num) {
                return false;
            }
        }

        // Column has the unique numbers (column-clash)
        for (int r = 0; r < board.length; r++)
        {

            // Check if the number
            // we are trying to
            // place is already present in
            // that column, return false;
            if (board[r][col] == num)
            {
                return false;
            }
        }

        // Corresponding square has
        // unique number (box-clash)
        int sqrt = (int)Math.sqrt(board.length);
        int boxRowStart = row - row % sqrt;
        int boxColStart = col - col % sqrt;

        for (int r = boxRowStart;
            r < boxRowStart + sqrt; r++)
        {
            for (int d = boxColStart;
                d < boxColStart + sqrt; d++)
            {
                if (board[r][d] == num)
                {
                    return false;
```

```
                }
            }
        }

        // if there is no clash, it's safe
        return true;
    }

    public static boolean solveSudoku(
        int[][] board, int n)
    {
        int row = -1;
        int col = -1;
        boolean isEmpty = true;
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                if (board[i][j] == 0)
                {
                    row = i;
                    col = j;

                    // We still have some remaining
                    // missing values in Sudoku
                    isEmpty = false;
                    break;
                }
            }
            if (!isEmpty) {
                break;
            }
        }

        // No empty space left
        if (isEmpty)
        {
            return true;
        }

        // Else for each-row backtrack
        for (int num = 1; num <= n; num++)
        {
            if (isSafe(board, row, col, num))
            {
                board[row][col] = num;
                if (solveSudoku(board, n))
                {
                    // print(board, n);
```

```
                    return true;
                }
                else
                {
                    // replace it
                    board[row][col] = 0;
                }
            }
        }
        return false;
    }

    public static void print(
        int[][] board, int N)
    {

        // We got the answer, just print it
        for (int r = 0; r < N; r++)
        {
            for (int d = 0; d < N; d++)
            {
                System.out.print(board[r][d]);
                System.out.print(" ");
            }
            System.out.print("\n");

            if ((r + 1) % (int)Math.sqrt(N) == 0)
            {
                System.out.print("");
            }
        }
    }

    // Driver Code
    public static void main(String args[])
    {

      Scanner input = new Scanner(System.in);

      /**
       * first Sudoku :
0 0 5 3 0 0 0 0 0
8 0 0 0 0 0 0 2 0
0 7 0 0 1 0 5 0 0
4 0 0 0 0 5 3 0 0
0 1 0 0 7 0 0 0 6
0 0 3 2 0 0 0 8 0
0 6 0 5 0 0 0 0 9
0 0 4 0 0 0 0 3 0
```

```
0 0 0 0 0 9 7 0 0
        * second sudoku :
8 0 0 0 0 0 0 0 0
0 0 3 6 0 0 0 0 0
0 7 0 0 9 0 2 0 0
0 5 0 0 0 7 0 0 0
0 0 0 0 4 5 7 0 0
0 0 0 1 0 0 0 3 0
0 0 1 0 0 0 0 6 8
0 0 8 5 0 0 0 1 0
0 9 0 0 0 0 4 0 0
        */


            int[][] board = new int[9][9];
        for (int i=0; i<9; i++) {
            for (int j=0; j<9; j++) {
                board[i][j] = input.nextInt();
            }
        }
        System.out.println("\nsolution to the problem :");
            int N = board.length;

            if (solveSudoku(board, N))
            {
                  // print solution
                  print(board, N);
            }
            else {
                  System.out.println("No solution");
            }
        input.close();
      }
}
```