Task 2: Rat in a Maze

mplement a function bool SolveMaze(int[,] maze) that uses backtracking to find a path from the top left corner to the bottom right corner of a maze. The maze is represented by a 2D array where 1s are paths and 0s are walls. Find a rat's path through the maze. The maze size is 6x6.

ANS:

```
package com.Day22;
public class RatInMaze {

  // Size of the maze
  static final int N = 6;
  // Function to print the solution matrix
  void printSolution(int sol[][]) {
     for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
           System.out.print(" " + sol[i][j] + " ");
        System.out.println();
     }
  }
  // Utility function to check if x, y is valid index for N*N maze
  boolean isSafe(int maze[][], int x, int y) {
     // x, y must be within the maze and maze[x][y] must be 1
     return (x >= 0 && x < N && y >= 0 && y < N && maze[x][y] == 1);
  }
  // This function solves the Maze problem using Backtracking.
  // It mainly uses solveMazeUtil() to solve the problem.
  boolean solveMaze(int maze[][]) {
     int sol[][] = new int[N][N];
     if (solveMazeUtil(maze, 0, 0, sol) == false) {
        System.out.println("Solution doesn't exist");
        return false;
     }
     printSolution(sol);
     return true;
  }
  // A recursive utility function to solve Maze problem
  boolean solveMazeUtil(int maze[][], int x, int y, int sol[][]) {
     // If (x, y is goal) return true
     if (x == N - 1 && y == N - 1 && maze[x][y] == 1) {
```

```
            sol[x][y] = 1;
            return true;
        }
        // Check if maze[x][y] is valid
        if (isSafe(maze, x, y) == true) {
            // Check if the current block is already part of the solution
path.
            if (sol[x][y] == 1)
                return false;
            // Mark x, y as part of the solution path
            sol[x][y] = 1;
            // Move forward in x direction
            if (solveMazeUtil(maze, x + 1, y, sol))
                return true;
            // If moving in x direction doesn't give solution then
            // Move down in y direction
            if (solveMazeUtil(maze, x, y + 1, sol))
                return true;
            // If moving in y direction doesn't give solution then
            // Move backward in x direction
            if (solveMazeUtil(maze, x - 1, y, sol))
                return true;
            // If moving in x direction backward doesn't give solution
then
            // Move up in y direction
            if (solveMazeUtil(maze, x, y - 1, sol))
                return true;
            // If none of the above movements work then
            // BACKTRACK: unmark x, y as part of solution path
            sol[x][y] = 0;
            return false;
        }
        return false;
    }
    public static void main(String args[]) {
        RatInMaze rat = new RatInMaze();
        int maze[][] = {{1, 0, 0, 0, 0, 0},
```

```
                {1, 1, 0, 1, 1, 1},
                {0, 1, 0, 0, 0, 1},
                {1, 1, 1, 1, 0, 0},
                {0, 0, 0, 1, 0, 1},
                {0, 0, 0, 1, 1, 1}};
        rat.solveMaze(maze);
    }
}
```

OUTPUT:

```
1  0  0  0  0  0
1  1  0  0  0  0
0  1  0  0  0  0
0  1  1  1  0  0
0  0  0  1  0  0
0  0  0  1  1  1
```