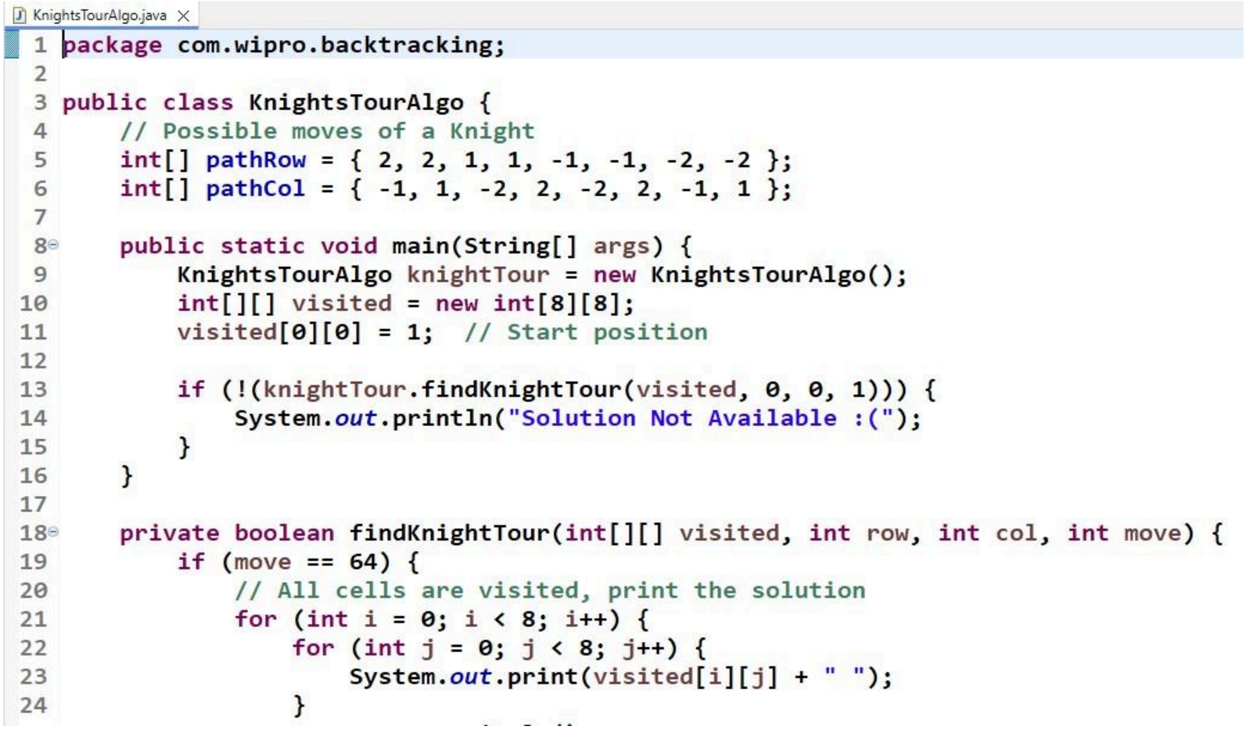
# Day 16 and 17

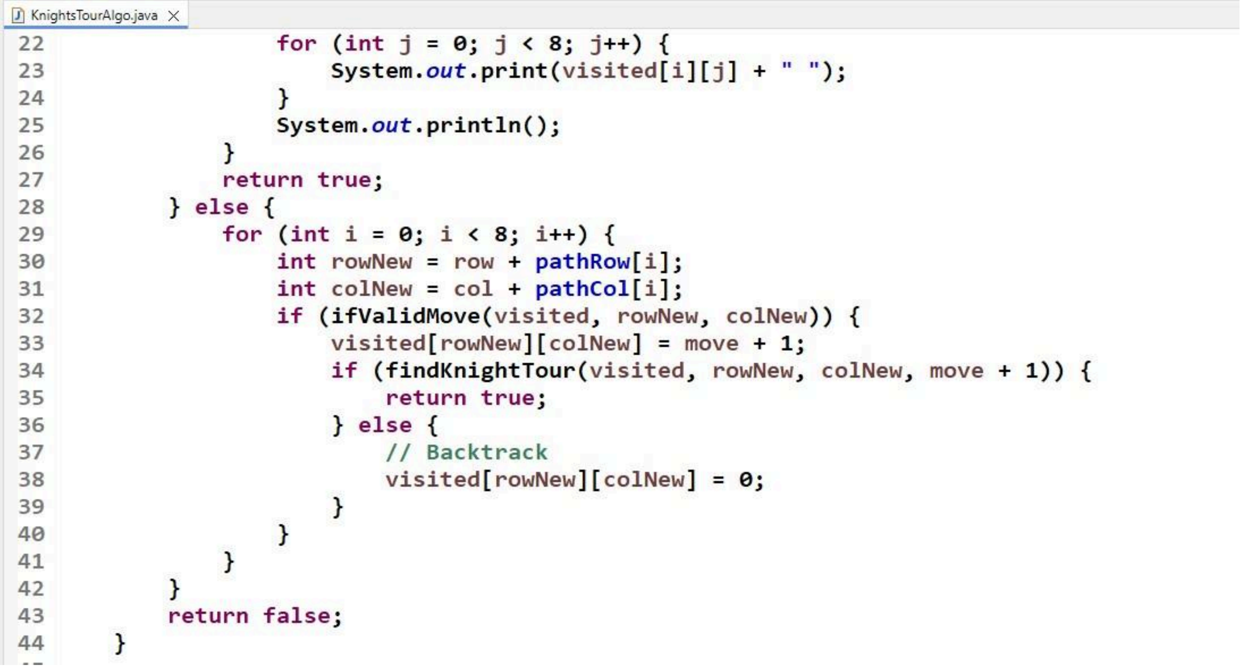
**1]** Task 1: The Knight’s Tour Problem
Create a function bool SolveKnightsTour(int[,] board, int moveX, int moveY, int moveCount, int[] xMove, int[] yMove) that attempts to solve the Knight's Tour problem using backtracking. The function should return true if a solution exists and false otherwise. The board represents the chessboard, moveX and moveY are the current coordinates of the knight, moveCount is the current move count, and xMove[], yMove[] are the possible next moves for the knight. Fill the chessboard such that the knight visits every square exactly once. Keep the  chessboard size to 8x8.

Solution:-
Code -

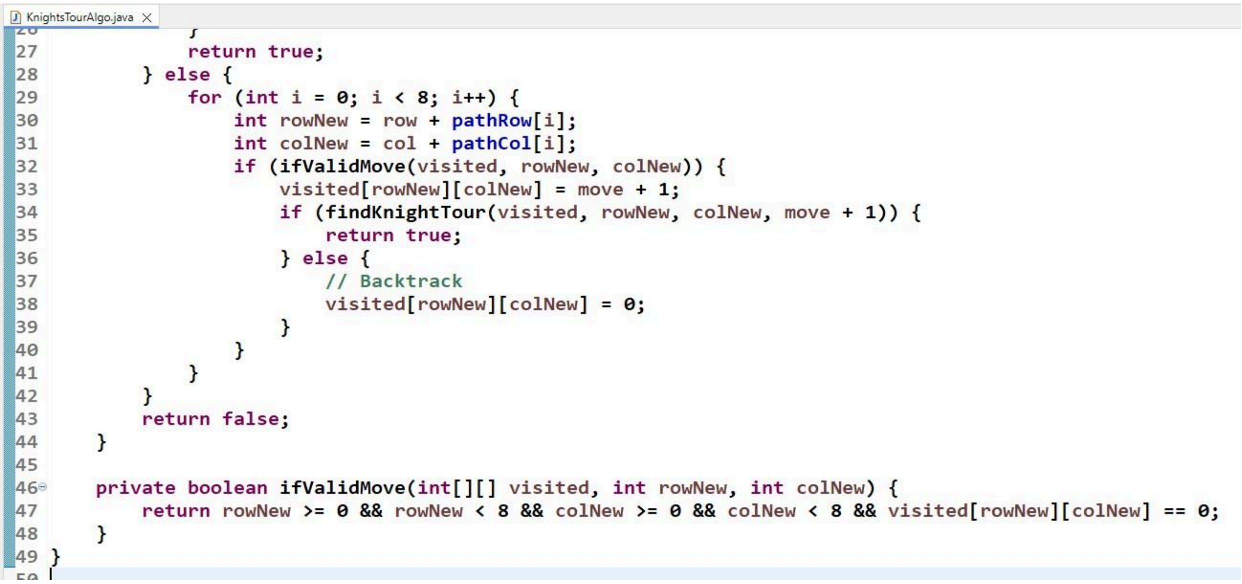


KnightsTourAlgo.java

```java
package com.wipro.backtracking;

public class KnightsTourAlgo {
    // Possible moves of a Knight
    int[] pathRow = { 2, 2, 1, 1, -1, -1, -2, -2 };
    int[] pathCol = { -1, 1, -2, 2, -2, 2, -1, 1 };

    public static void main(String[] args) {
        KnightsTourAlgo knightTour = new KnightsTourAlgo();
        int[][] visited = new int[8][8];
        visited[0][0] = 1;  // Start position

        if (!(knightTour.findKnightTour(visited, 0, 0, 1))) {
            System.out.println("Solution Not Available :(");
        }
    }

    private boolean findKnightTour(int[][] visited, int row, int col, int move) {
        if (move == 64) {
            // All cells are visited, print the solution
            for (int i = 0; i < 8; i++) {
                for (int j = 0; j < 8; j++) {
                    System.out.print(visited[i][j] + " ");
                }
```

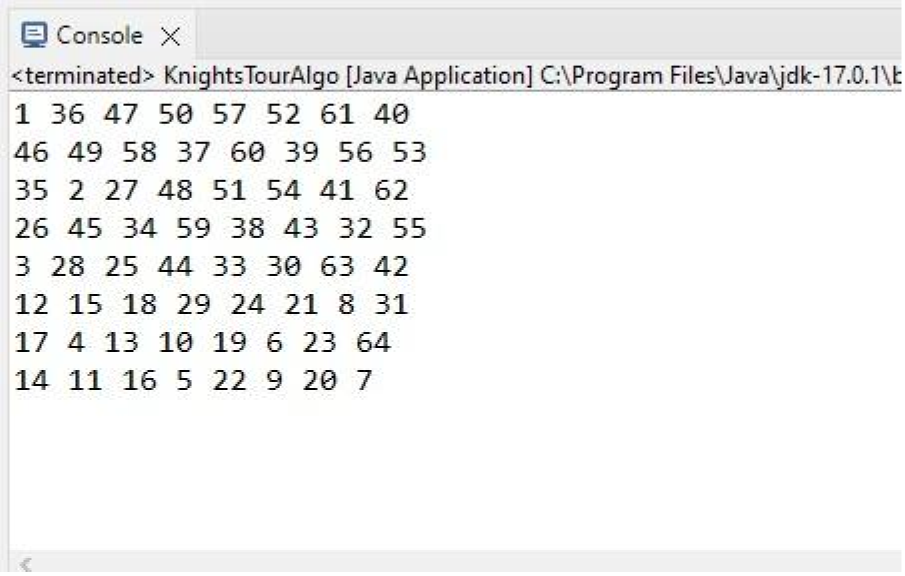


```java
                for (int j = 0; j < 8; j++) {
                    System.out.print(visited[i][j] + " ");
                }
                System.out.println();
            }
            return true;
        } else {
            for (int i = 0; i < 8; i++) {
                int rowNew = row + pathRow[i];
                int colNew = col + pathCol[i];
                if (ifValidMove(visited, rowNew, colNew)) {
                    visited[rowNew][colNew] = move + 1;
                    if (findKnightTour(visited, rowNew, colNew, move + 1)) {
                        return true;
                    } else {
                        // Backtrack
                        visited[rowNew][colNew] = 0;
                    }
                }
            }
        }
        return false;
    }
```



```java
            return true;
        } else {
            for (int i = 0; i < 8; i++) {
                int rowNew = row + pathRow[i];
                int colNew = col + pathCol[i];
                if (ifValidMove(visited, rowNew, colNew)) {
                    visited[rowNew][colNew] = move + 1;
                    if (findKnightTour(visited, rowNew, colNew, move + 1)) {
                        return true;
                    } else {
                        // Backtrack
                        visited[rowNew][colNew] = 0;
                    }
                }
            }
        }
        return false;
    }

    private boolean ifValidMove(int[][] visited, int rowNew, int colNew) {
        return rowNew >= 0 && rowNew < 8 && colNew >= 0 && colNew < 8 && visited[rowNew][colNew] == 0;
    }
}
```

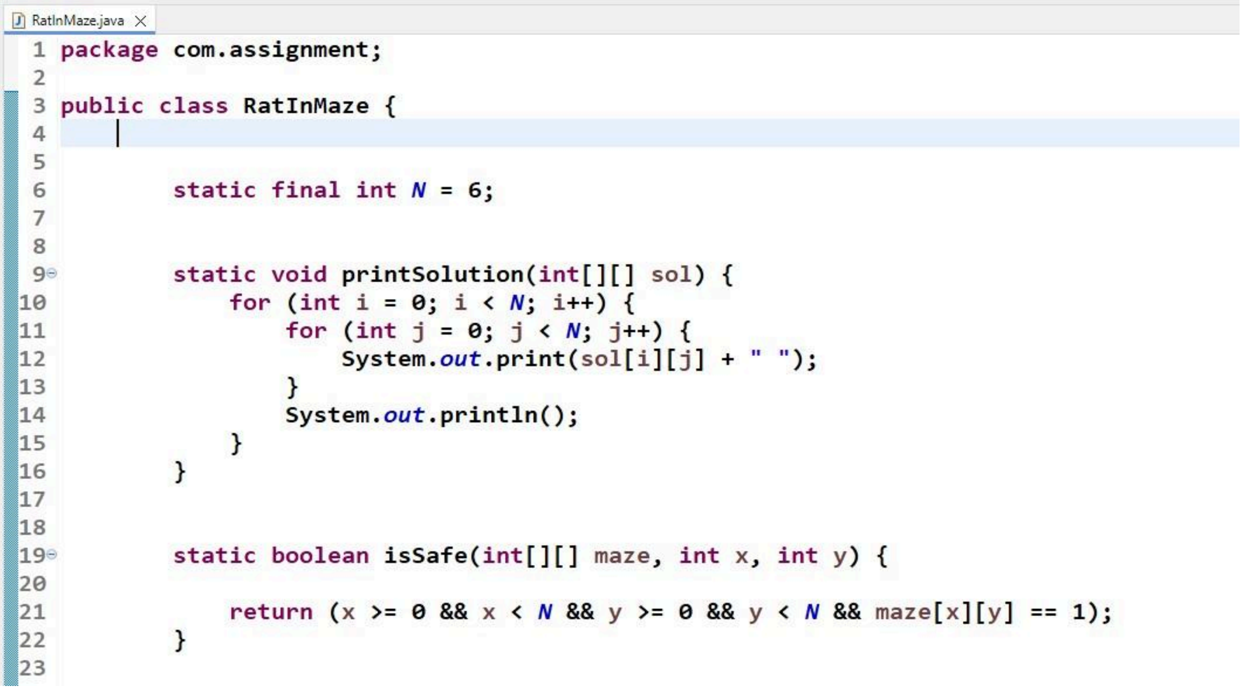
Output -



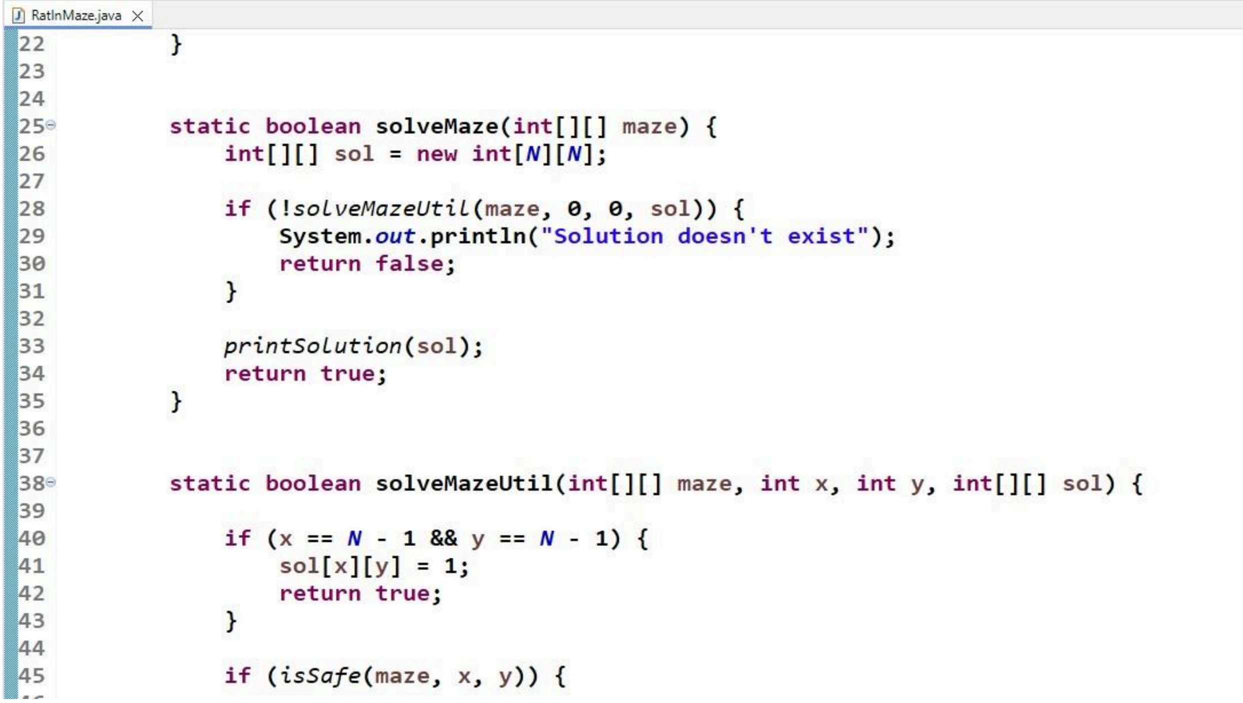
```
<terminated> KnightsTourAlgo [Java Application] C:\Program Files\Java\jdk-17.0.1\b
1 36 47 50 57 52 61 40
46 49 58 37 60 39 56 53
35 2 27 48 51 54 41 62
26 45 34 59 38 43 32 55
3 28 25 44 33 30 63 42
12 15 18 29 24 21 8 31
17 4 13 10 19 6 23 64
14 11 16 5 22 9 20 7
```

**2]** Task 2: Rat in a Maze
implement 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.

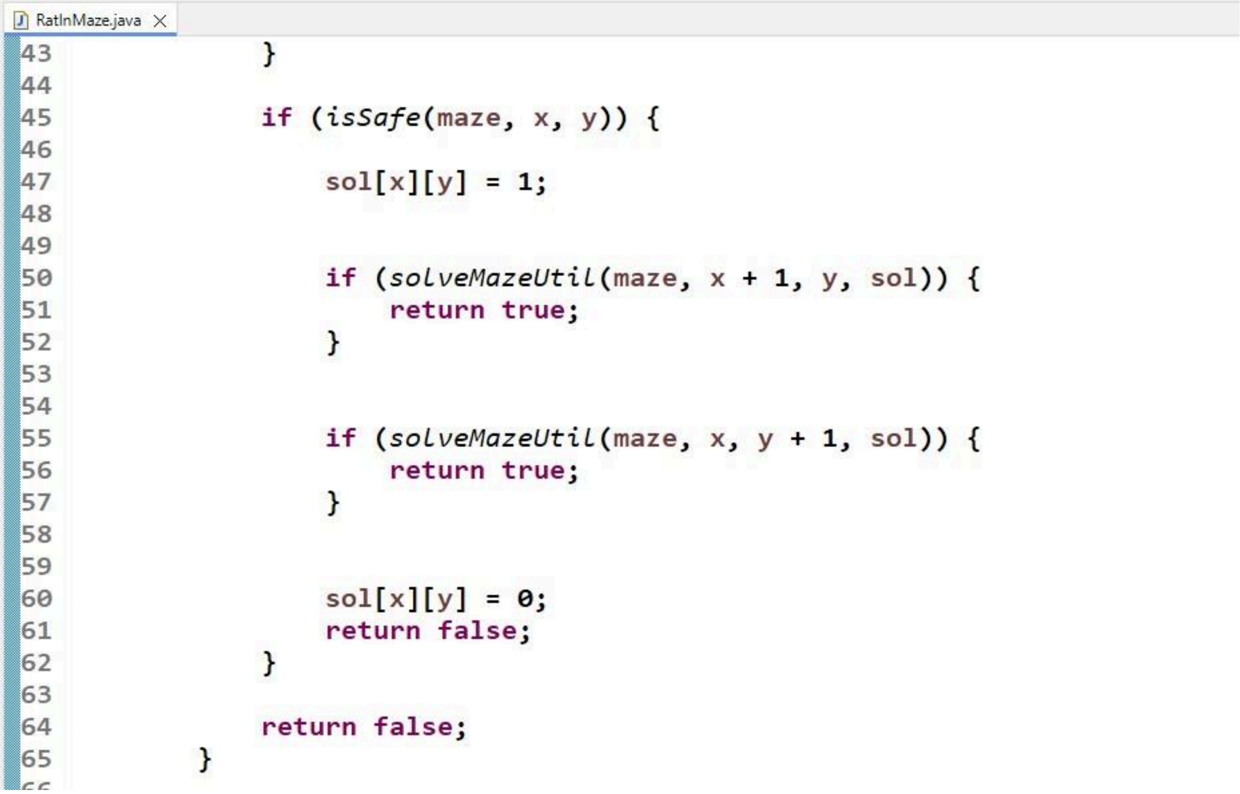
Solution:-
Code -



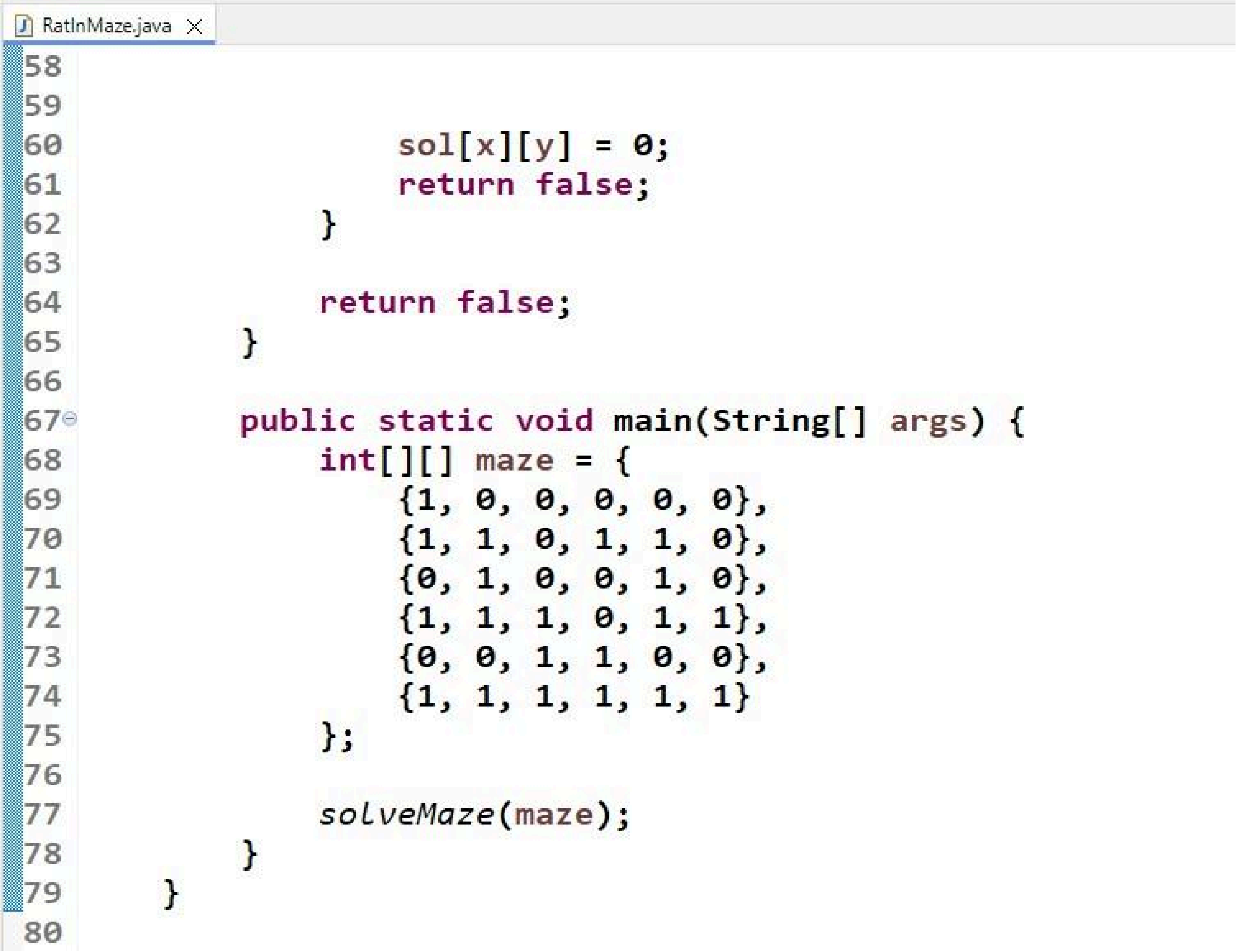
```
package com.assignment;

public class RatInMaze {


        static final int N = 6;


        static 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();
            }
        }


        static boolean isSafe(int[][] maze, int x, int y) {

            return (x >= 0 && x < N && y >= 0 && y < N && maze[x][y] == 1);
        }

```



```
        }


        static boolean solveMaze(int[][] maze) {
            int[][] sol = new int[N][N];

            if (!solveMazeUtil(maze, 0, 0, sol)) {
                System.out.println("Solution doesn't exist");
                return false;
            }

            printSolution(sol);
            return true;
        }


        static boolean solveMazeUtil(int[][] maze, int x, int y, int[][] sol) {

            if (x == N - 1 && y == N - 1) {
                sol[x][y] = 1;
                return true;
            }

            if (isSafe(maze, x, y)) {
```

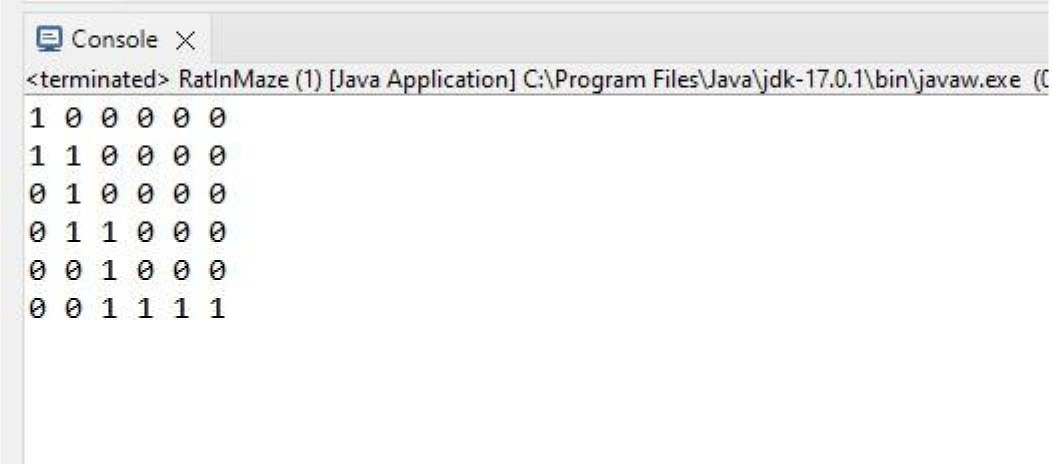


```
            }

            if (isSafe(maze, x, y)) {

                sol[x][y] = 1;


                if (solveMazeUtil(maze, x + 1, y, sol)) {
                    return true;
                }


                if (solveMazeUtil(maze, x, y + 1, sol)) {
                    return true;
                }


                sol[x][y] = 0;
                return false;
            }

            return false;
        }
```



```java

                sol[x][y] = 0;
                return false;
            }

            return false;
        }

        public static void main(String[] args) {
            int[][] maze = {
                {1, 0, 0, 0, 0, 0},
                {1, 1, 0, 1, 1, 0},
                {0, 1, 0, 0, 1, 0},
                {1, 1, 1, 0, 1, 1},
                {0, 0, 1, 1, 0, 0},
                {1, 1, 1, 1, 1, 1}
            };

            solveMaze(maze);
        }
    }
```

Output -

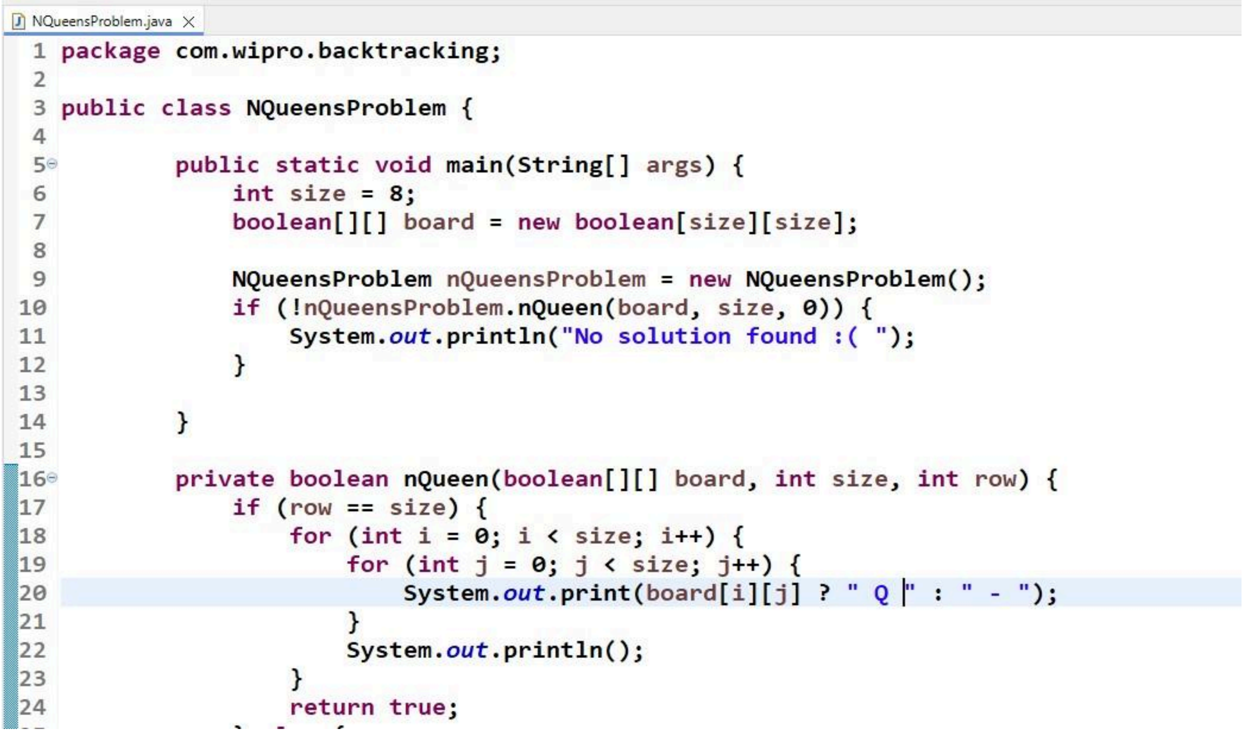


```
<terminated> RatInMaze (1) [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe  ((
1 0 0 0 0 0
1 1 0 0 0 0
0 1 0 0 0 0
0 1 1 0 0 0
0 0 1 0 0 0
0 0 1 1 1 1
```

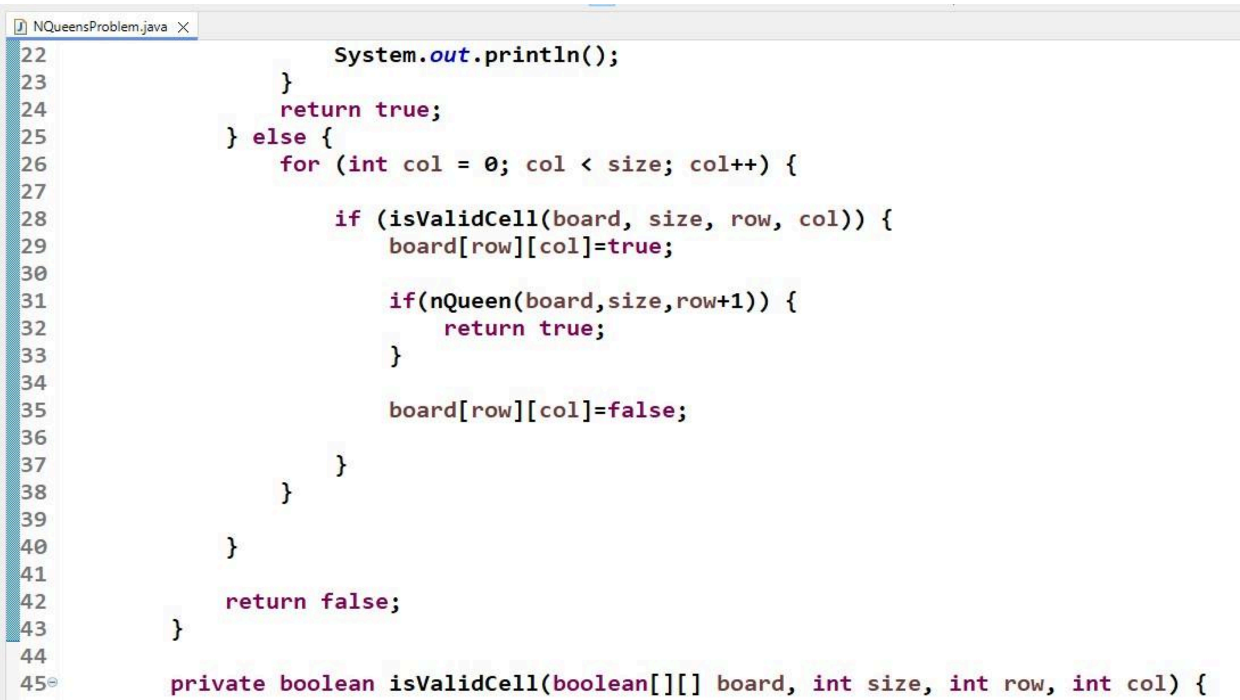
## **3]** Task 3: N Queen Problem

Write a function bool SolveNQueen(int[,] board, int col) in C# that places N queens on an N x N chessboard so that no two queens attack each other using backtracking. Place N queens on the board such that no two queens can attack each other. Use a standard 8x8 chessboard.

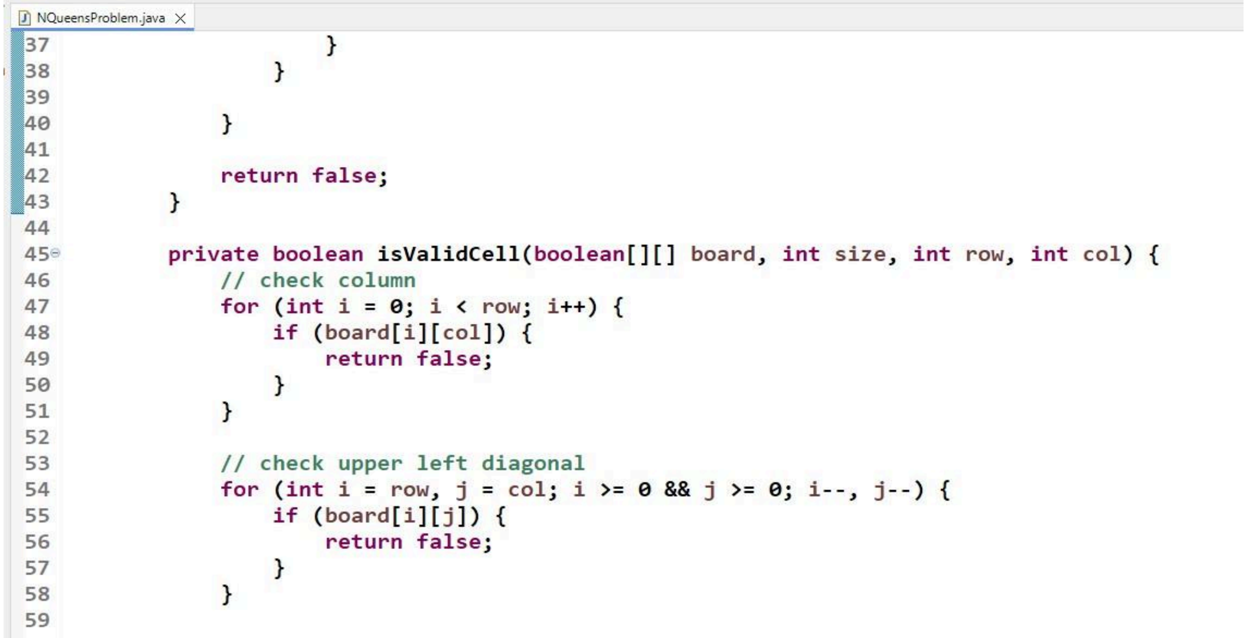
Solution:-
Code -



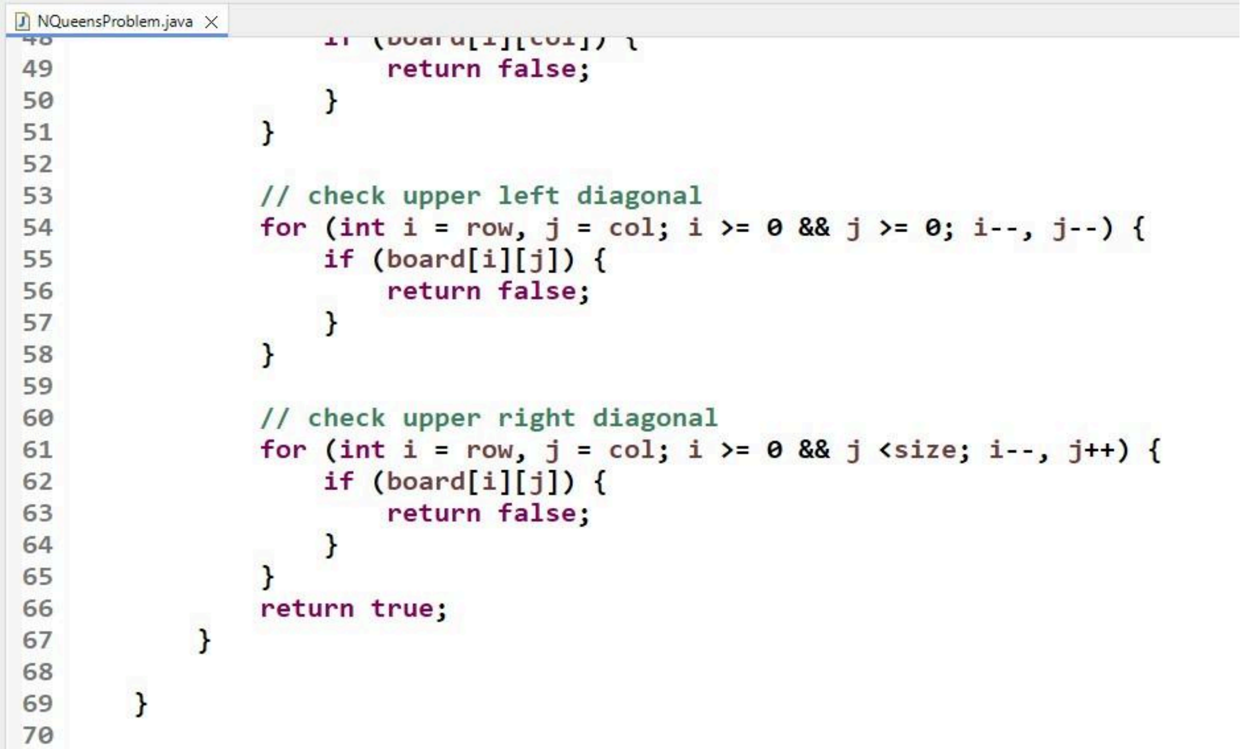
```
package com.wipro.backtracking;

public class NQueensProblem {

        public static void main(String[] args) {
            int size = 8;
            boolean[][] board = new boolean[size][size];

            NQueensProblem nQueensProblem = new NQueensProblem();
            if (!nQueensProblem.nQueen(board, size, 0)) {
                System.out.println("No solution found :( ");
            }

        }

        private boolean nQueen(boolean[][] board, int size, int row) {
            if (row == size) {
                for (int i = 0; i < size; i++) {
                    for (int j = 0; j < size; j++) {
                        System.out.print(board[i][j] ? " Q " : " - ");
                    }
                    System.out.println();
                }
                return true;
```



```java
                System.out.println();
            }
            return true;
        } else {
            for (int col = 0; col < size; col++) {

                if (isValidCell(board, size, row, col)) {
                    board[row][col]=true;

                    if(nQueen(board,size,row+1)) {
                        return true;
                    }

                    board[row][col]=false;

                }
            }

        }

        return false;
    }

    private boolean isValidCell(boolean[][] board, int size, int row, int col) {
```

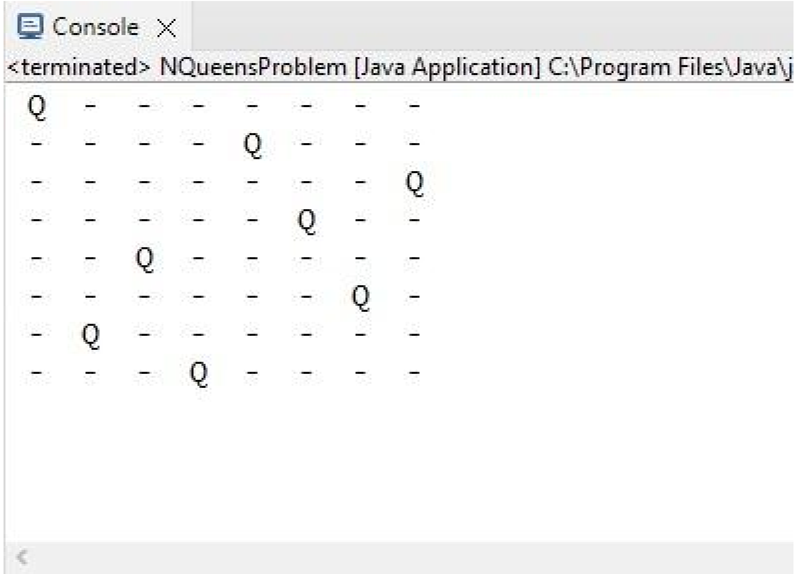


```java
                }
            }

        }

        return false;
    }

    private boolean isValidCell(boolean[][] board, int size, int row, int col) {
        // check column
        for (int i = 0; i < row; i++) {
            if (board[i][col]) {
                return false;
            }
        }

        // check upper left diagonal
        for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
            if (board[i][j]) {
                return false;
            }
        }

```



```java
                if (board[i][col]) {
                    return false;
                }
            }

            // check upper left diagonal
            for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
                if (board[i][j]) {
                    return false;
                }
            }

            // check upper right diagonal
            for (int i = row, j = col; i >= 0 && j <size; i--, j++) {
                if (board[i][j]) {
                    return false;
                }
            }
            return true;
        }

    }
```

Output -



```
<terminated> NQueensProblem [Java Application] C:\Program Files\Java\j
 Q  -  -  -  -  -  -  -
 -  -  -  -  Q  -  -  -
 -  -  -  -  -  -  -  Q
 -  -  -  -  -  Q  -  -
 -  -  Q  -  -  -  -  -
 -  -  -  -  -  -  Q  -
 -  Q  -  -  -  -  -  -
 -  -  -  Q  -  -  -  -
```