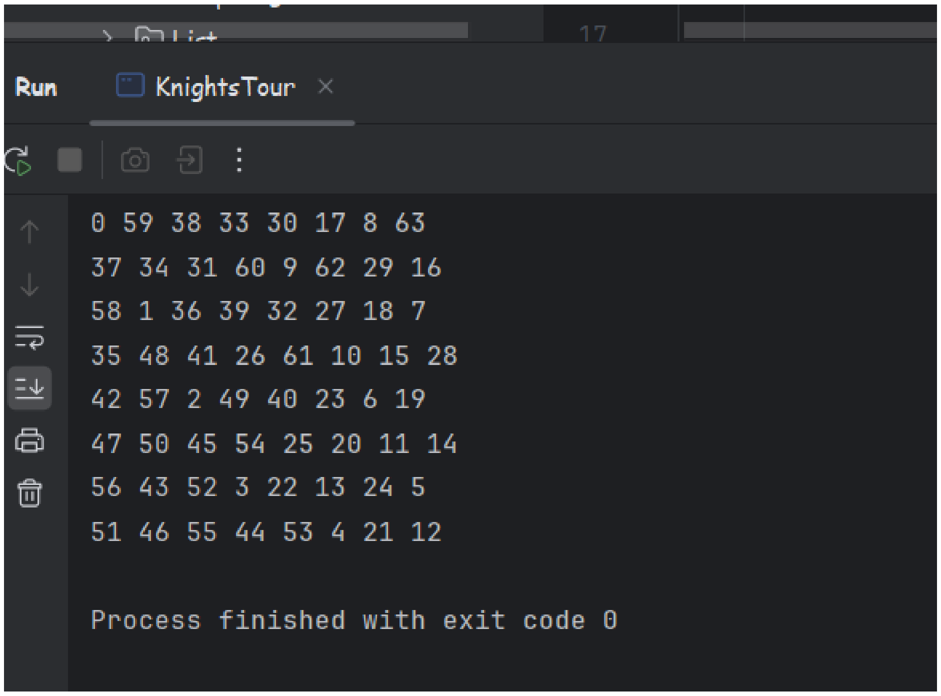
Task 1: The Knight's Tour Problem

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.

Ans:

```java
package Day16_17;

public class KnightsTour {

    private static final int N = 8;


    private static boolean isSafe(int x, int y, int[][] board) {
        return (x >= 0 && x < N && y >= 0 && y < N && board[x][y] == -1);
    }

    public static boolean solveKnightsTour(int[][] board, int moveX, int moveY, int moveCount, int[]
xMove, int[] yMove) {
        int nextX, nextY;
        if (moveCount == N * N) {
            return true;
        }

        for (int k = 0; k < N; k++) {
            nextX = moveX + xMove[k];
            nextY = moveY + yMove[k];
            if (isSafe(nextX, nextY, board)) {
                board[nextX][nextY] = moveCount;
                if (solveKnightsTour(board, nextX, nextY, moveCount + 1, xMove, yMove)) {
                    return true;
                } else {
                    board[nextX][nextY] = -1;
                }
            }
        }

        return false;
    }

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

        for (int x = 0; x < N; x++) {
            for (int y = 0; y < N; y++) {
                board[x][y] = -1;
            }
        }

        int[] xMove = { 2, 1, -1, -2, -2, -1, 1, 2 };
        int[] yMove = { 1, 2, 2, 1, -1, -2, -2, -1 };

        board[0][0] = 0;
        if (!solveKnightsTour(board, 0, 0, 1, xMove, yMove)) {
            System.out.println("Solution does not exist");
        } else {
            printSolution(board);
        }
    }

    private static void printSolution(int[][] board) {
        for (int x = 0; x < N; x++) {
            for (int y = 0; y < N; y++) {
                System.out.print(board[x][y] + " ");
            }
            System.out.println();
        }
    }
}
```

Output:



```
0 59 38 33 30 17 8 63
37 34 31 60 9 62 29 16
58 1 36 39 32 27 18 7
35 48 41 26 61 10 15 28
42 57 2 49 40 23 6 19
47 50 45 54 25 20 11 14
56 43 52 3 22 13 24 5
51 46 55 44 53 4 21 12

Process finished with exit code 0
```

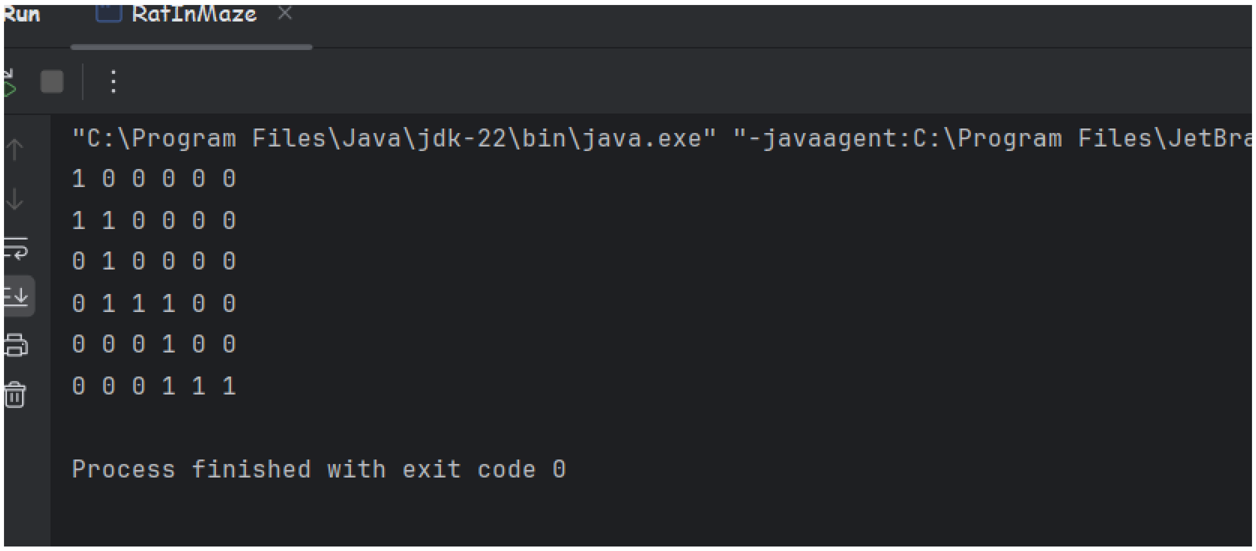
Task 2: Rat in a Maze
Implement a function bool SolveMaze(int[,] maze) that uses backtracking to find a path from the top left corner to the bottom right corner of a maze. The maze is represented by a 2D array where 1s are paths and 0s are walls. Find a rat's path through the maze. The maze size is 6x6.

Ans:

```
package Day16_17;

public class RatInMaze {
    private static final int N = 6;
    private 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();
        }
    }

    private static boolean isSafe(int[][] maze, int x, int y) {
        return (x >= 0 && x < N && y >= 0 && y < N && maze[x][y] == 1);
    }

    public 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;
}

private 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)) {

        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;
}

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

        solveMaze(maze);
    }
}
```

Output:



```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBra
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

Process finished with exit code 0
```

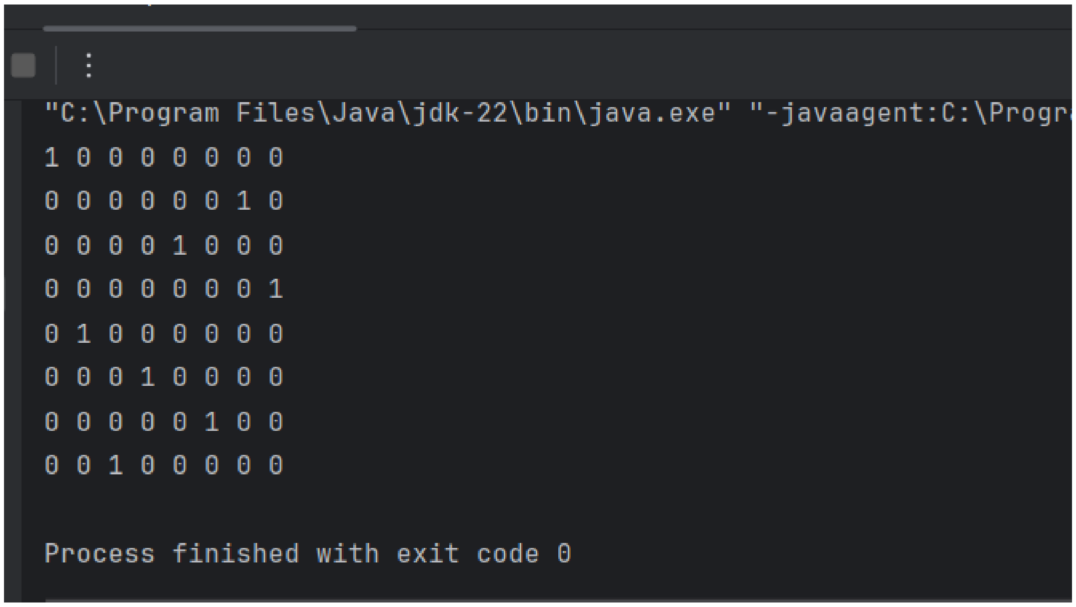
Task 3: N Queen Problem

Write a function bool SolveNQueen(int[,] board, int col) in Java 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.

```
package Day16_17;

public class NQueenProblem {
    private static final int N = 8;

    private static void printSolution(int[][] board) {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                System.out.print(board[i][j] + " ");
            }
            System.out.println();
        }
```

```
}

private static boolean isSafe(int[][] board, int row, int col) {
    int i, j;

    for (i = 0; i < col; i++) {
        if (board[row][i] == 1) {
            return false;
        }
    }

    for (i = row, j = col; i >= 0 && j >= 0; i--, j--) {
        if (board[i][j] == 1) {
            return false;
        }
    }

    for (i = row, j = col; j >= 0 && i < N; i++, j--) {
        if (board[i][j] == 1) {
            return false;
        }
    }

    return true;
}

private static boolean solveNQueenUtil(int[][] board, int col) {

    if (col >= N) {
        return true;
    }

    for (int i = 0; i < N; i++) {

        if (isSafe(board, i, col)) {

            board[i][col] = 1;


            if (solveNQueenUtil(board, col + 1)) {
                return true;
            }
            board[i][col] = 0;
        }
        }

        return false;
    }

    public static boolean solveNQueen() {
        int[][] board = new int[N][N];

        if (!solveNQueenUtil(board, 0)) {
            System.out.println("Solution does not exist");
            return false;
        }

        printSolution(board);
        return true;
    }

    public static void main(String[] args) {
        solveNQueen();
    }
}
```

Output:



```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Progr
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0

Process finished with exit code 0
```