DAY 22:

ASSIGNMENT 2:

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.

has context menu

ANSWER:

```
public class NQueenProblem {

   private static final int N = 8;

   // Function to print the solution matrix
   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();
      }
   }

   // Function to check if a queen can be placed on board[row][col]
   private static boolean isSafe(int[][] board, int row, int col) {
      int i, j;

      // Check this row on the left side
      for (i = 0; i < col; i++) {
```

```
        if (board[row][i] == 1) {
            return false;
        }
    }

    // Check upper diagonal on the left side
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--) {
        if (board[i][j] == 1) {
            return false;
        }
    }

    // Check lower diagonal on the left side
    for (i = row, j = col; i < N && j >= 0; i++, j--) {
        if (board[i][j] == 1) {
            return false;
        }
    }

    return true;
}

// Function to solve the N Queen problem using backtracking
private static boolean solveNQueenUtil(int[][] board, int col) {
    // If all queens are placed, return true
    if (col >= N) {
        return true;
    }

    // Consider this column and try placing this queen in all rows one by one
    for (int i = 0; i < N; i++) {
```

```
        if (isSafe(board, i, col)) {
            // Place this queen in board[i][col]
            board[i][col] = 1;

            // Recur to place rest of the queens
            if (solveNQueenUtil(board, col + 1)) {
                return true;
            }

            // If placing queen in board[i][col] doesn't lead to a solution,
            // then remove queen from board[i][col] (backtrack)
            board[i][col] = 0;
        }
    }

    // If the queen cannot be placed in any row in this column, return false
    return false;
}

// Function to solve the N Queen problem
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();
    }
}
```

Explanation:

1. isSafe Function: Checks if it's safe to place a queen at board[row][col]. This means no other queens should be in the same row, same upper diagonal, or same lower diagonal.

2. solveNQueenUtil Function: Uses backtracking to place queens column by column. If a queen can be placed in a row of the current column, it recursively attempts to place queens in the subsequent columns. If placing a queen in any row of the current column doesn't lead to a solution, it backtracks and removes the queen from the current cell.

3. solveNQueen Function: Initializes the board and starts the solving process. If a solution is found, it prints the board. If not, it prints that no solution exists.

This program will output the board configuration with queens placed in a way such that no two queens can attack each other.