Implement a solution for a Constraint Satisfaction Problem using Backtracking for n-queens problem.

Program:

```
import java.util.Scanner;

public class NQueens {

    private static int N;
    private static int backtrackCount;

    // 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 column on upper side
        for (i = 0; i < row; i++)
            if (board[i][col] == 1)
                return false;

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

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

        return true;
    }

    // A recursive utility function to solve N Queens problem
    private static boolean solveNQUtil(int[][] board, int row) {
```

```
        // Base case: If all queens are placed then return true
        if (row >= N)
            return true;

        // Consider this row and try placing this queen in all
columns one by one
        for (int i = 0; i < N; i++) {
            // Check if the queen can be placed on board[row][i]
            if (isSafe(board, row, i)) {
                // Place this queen in board[row][i]
                board[row][i] = 1;

                // Print the backtracking order
                System.out.println("Backtracking " +
(++backtrackCount) + ":");
                printSolution(board);

                // Recur to place rest of the queens
                if (solveNQUtil(board, row + 1))
                    return true;

                // If placing queen in board[row][i] doesn't lead to
a solution then remove queen from board[row][i]
                board[row][i] = 0; // BACKTRACK
            }
        }

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

    // This function solves the N Queens problem using Backtracking.
    public static void solveNQ(int n) {
        N = n;
        backtrackCount = 0;
```

```
        int[][] board = new int[N][N];

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

    // A utility function to print solution
    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();
        }
        System.out.println();
    }

    // Main method
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the value of n: ");
        int n = scanner.nextInt();
        solveNQ(n);
    }
}
```

Output:

```
Enter the value of n: 4
Backtracking 1:
 1  0  0  0
 0  0  0  0
 0  0  0  0
 0  0  0  0

Backtracking 2:
 1  0  0  0
 0  0  1  0
 0  0  0  0
 0  0  0  0

Backtracking 3:
 1  0  0  0
 0  0  0  1
 0  0  0  0
 0  0  0  0

Backtracking 4:
 1  0  0  0
 0  0  0  1
 0  1  0  0
 0  0  0  0

Backtracking 5:
 0  1  0  0
 0  0  0  0
 0  0  0  0
 0  0  0  0

Backtracking 6:
 0  1  0  0
 0  0  0  1
 0  0  0  0
 0  0  0  0

Backtracking 7:
 0  1  0  0
 0  0  0  1
 1  0  0  0
 0  0  0  0

Backtracking 8:
 0  1  0  0
 0  0  0  1
 1  0  0  0
 0  0  1  0
```