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:

```
package com.Day21;
public class KnightsTourProblem {
  private static final int N = 8;
  // Possible moves for the knight
  private static final int[] xMove = {2, 1, -1, -2, -2, -1, 1, 2};
  private static final int[] yMove = {1, 2, 2, 1, -1, -2, -2, -1};
  // Function to check if (x, y) are valid coordinates for the knight
  private static boolean isSafe(int x, int y, int[][] board) {
    return (x >= 0 && x < N && y >= 0 && y < N && board[x][y] ==
-1);
  }
  // Function to solve Knight's Tour problem
  private static boolean solveKnightsTour(int[][] board, int moveX,
int moveY, int moveCount) {
    // Base case: If all squares are visited
    if (moveCount == N * N) {
      return true;
    }
    // Try all next moves from the current coordinate
    for (int k = 0; k < N; k++) {
      int nextX = moveX + xMove[k];
      int nextY = moveY + yMove[k];
      if (isSafe(nextX, nextY, board)) {
        board[nextX][nextY] = moveCount;
        if (solveKnightsTour(board, nextX, nextY, moveCount + 1))
{
          return true;
        } else {
          // Backtrack
```

```
                board[nextX][nextY] = -1;
            }
        }
    }
    return false;
}
// Function to initiate the Knight's Tour problem
public static void solve() {
    int[][] board = new int[N][N];
    // Initialization of board
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            board[i][j] = -1;
        }
    }
    // Starting position
    int startX = 0, startY = 0;
    board[startX][startY] = 0;
    // Start solving the Knight's Tour problem
    if (!solveKnightsTour(board, startX, startY, 1)) {
        System.out.println("Solution does not exist");
    } else {
        printSolution(board);
    }
}
// Function to print the solution
private static void printSolution(int[][] board) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            System.out.printf("%2d ", board[i][j]);
        }
        System.out.println();
    }
}
public static void main(String[] args) {
    solve();
}
```

```
}
```

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
```