Name : Tejaswini Ani Kamble

Email : teju000kamble@gmail.com

Assignments : Day 16 and 17

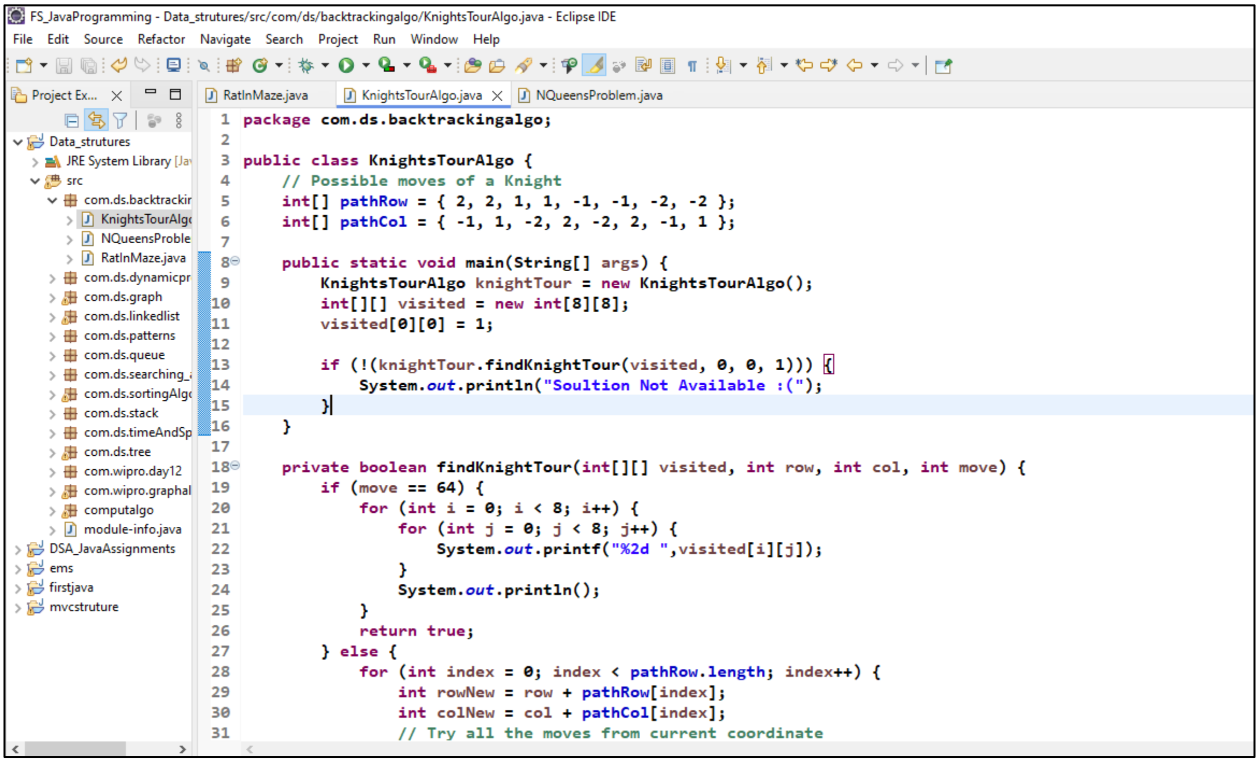
Task 1: The Knight’s Tour Problem

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.

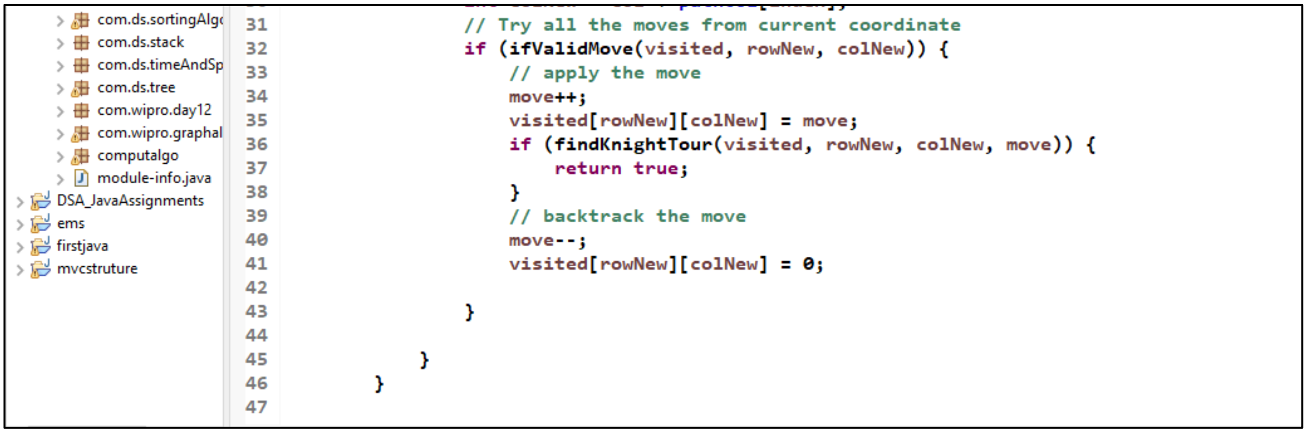
Ans: Source Code

```
package com.ds.backtrackingalgo;

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;

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

```
    private boolean findKnightTour(int[][] visited, int row, int col, int move)
 {
        if (move == 64)
{
            for (int i = 0; i < 8; i++) {
                for (int j = 0; j < 8; j++) {
                    System.out.printf("%2d ",visited[i][j]);
                }
                System.out.println();
            }
            return true;
        } else
 {
            for (int index = 0; index < pathRow.length; index++)
 {
                int rowNew = row + pathRow[index];
                int colNew = col + pathCol[index];
                // Try all the moves from current coordinate
                if (ifValidMove(visited, rowNew, colNew))
{
                    // apply the move
                    move++;
                    visited[rowNew][colNew] = move;
                    if (findKnightTour(visited, rowNew, colNew,
move)) {

                        return true;
                    }
                    // backtrack the move
                    move--;
                    visited[rowNew][colNew] = 0;

                }

            }
        }

        return false;
    }

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

}
```

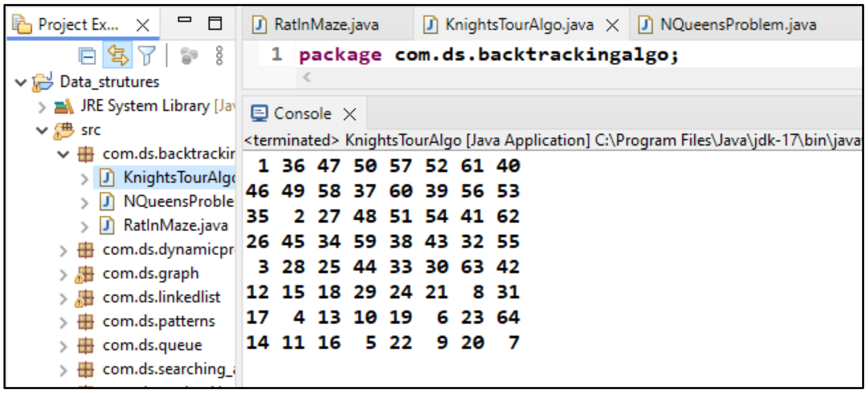


```
package com.ds.backtrackingalgo;

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;

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

    private boolean findKnightTour(int[][] visited, int row, int col, int move) {
        if (move == 64) {
            for (int i = 0; i < 8; i++) {
                for (int j = 0; j < 8; j++) {
                    System.out.printf("%2d ",visited[i][j]);
                }
                System.out.println();
            }
            return true;
        } else {
            for (int index = 0; index < pathRow.length; index++) {
                int rowNew = row + pathRow[index];
                int colNew = col + pathCol[index];
                // Try all the moves from current coordinate
```



```
                // Try all the moves from current coordinate
                if (ifValidMove(visited, rowNew, colNew)) {
                    // apply the move
                    move++;
                    visited[rowNew][colNew] = move;
                    if (findKnightTour(visited, rowNew, colNew, move)) {
                        return true;
                    }
                    // backtrack the move
                    move--;
                    visited[rowNew][colNew] = 0;

                }

            }
        }

```

Output:



```
package com.ds.backtrackingalgo;
```

```
<terminated> KnightsTourAlgo [Java Application] C:\Program Files\Java\jdk-17\bin\java
 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
```

Task 2: Rat in a Maze

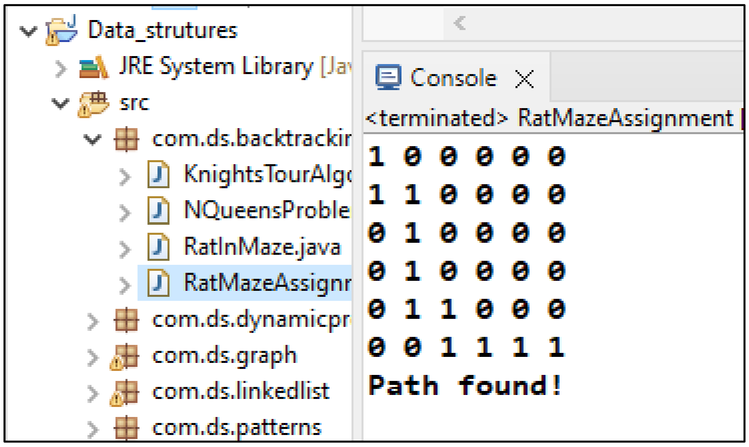
mplement a function bool SolveMaze(int[,] maze) that uses backtracking to find a path from the top left corner to the bottom right corner of a maze. The maze is represented by a 2D array where 1s are paths and 0s are walls. Find a rat's path through the maze. The maze size is 6x6.

Ans: Source Code

```
package com.ds.backtrackingalgo;

public class RatMazeAssignment
{
   private static final int MAZE_SIZE = 6;

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

      if (solveMaze(maze))
```

```
 {
      System.out.println("Path found!");
    } else {
      System.out.println("No path found :(");
    }
  }

  public static boolean solveMaze(int[][] maze)
{
    int[][] solution = new int[MAZE_SIZE][MAZE_SIZE];
    if (!findPath(maze, 0, 0, solution))
 {
      return false;
    }

    printSolution(solution);
    return true;
  }
private static boolean findPath(int[][] maze, int row, int col, int[][] solution)
 {
    if (row == MAZE_SIZE - 1 && col == MAZE_SIZE - 1)
{
      solution[row][col] = 1;
      return true;
    }
    if (isValidMove(maze, row, col))
 {     solution[row][col] = 1;
      if (findPath(maze, row, col + 1, solution))
{
        return true;
      }
      if (findPath(maze, row + 1, col, solution))
 {
        return true;
      }
      solution[row][col] = 0;
      return false;
    }

    return false;
    }

    private static boolean isValidMove(int[][] maze, int row, int col) {

return row >= 0 && row < MAZE_SIZE && col >= 0 && col < MAZE_SIZE
&& maze[row][col] == 1;
    }

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

Output:



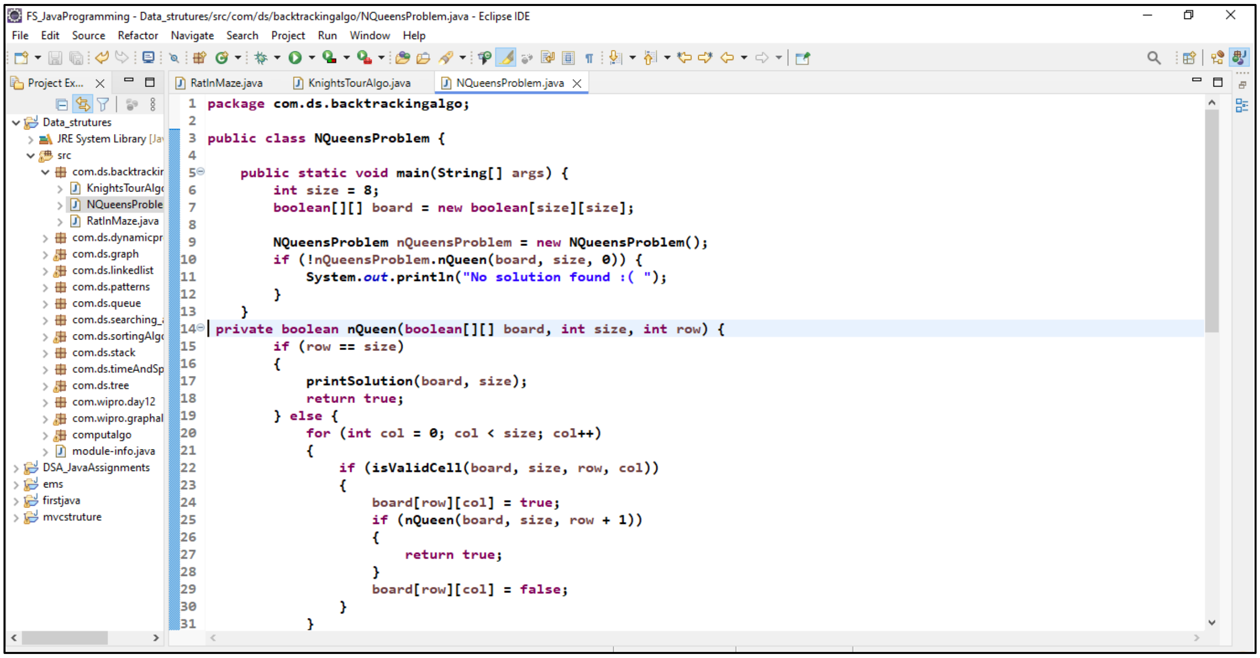
```
<terminated> RatMazeAssignment
1 0 0 0 0 0
1 1 0 0 0 0
0 1 0 0 0 0
0 1 0 0 0 0
0 1 1 0 0 0
0 0 1 1 1 1
Path found!
```

Task 3: N Queen Problem

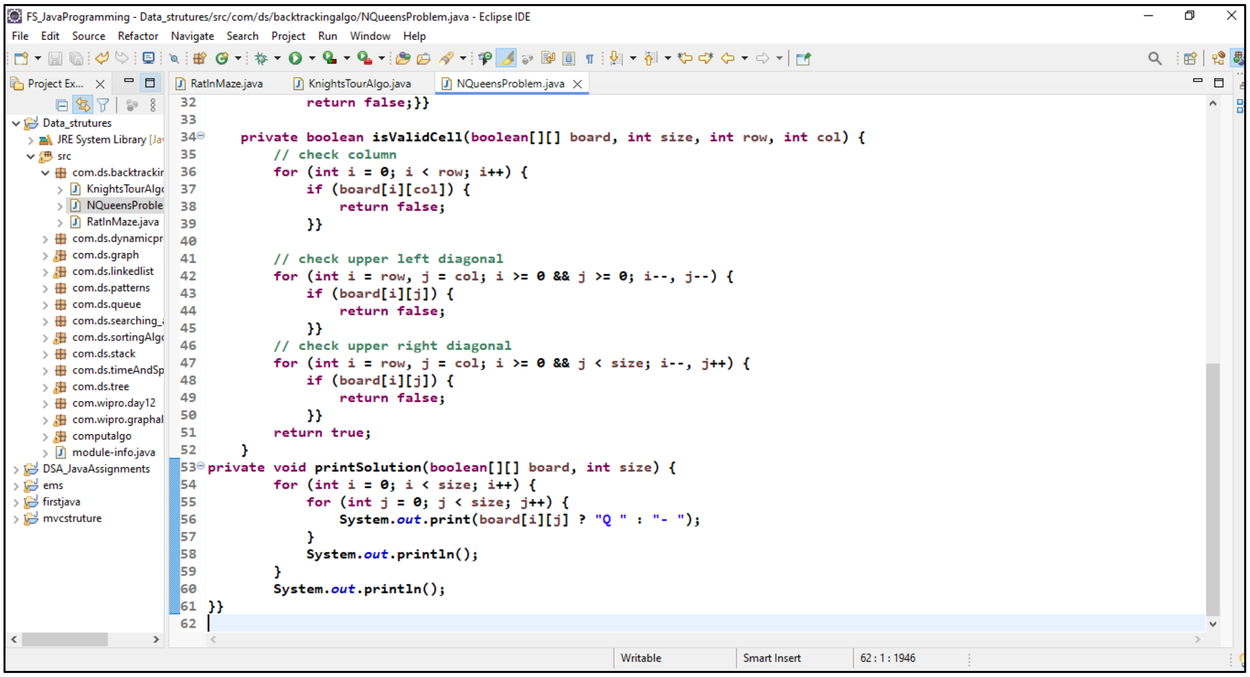
Write a function bool SolveNQueen(int[,] board, int col) in C# that places N queens on an N x N chessboard so that no two queens attack each other using

backtracking. Place N queens on the board such that no two queens can attack each other. Use a standard 8x8 chessboard.

Ans: Source Code

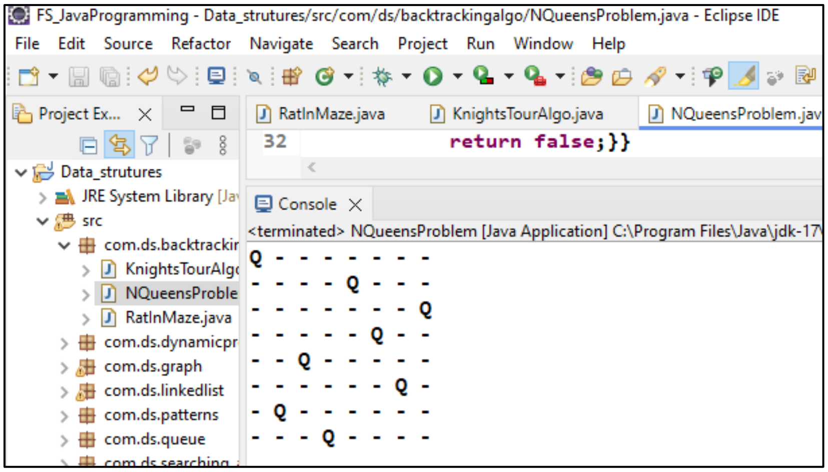


```java
package com.ds.backtrackingalgo;

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)
        {
            printSolution(board, size);
            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;
                }
            }
```



```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;
            }}
        // check upper right diagonal
        for (int i = row, j = col; i >= 0 && j < size; i--, j++) {
            if (board[i][j]) {
                return false;
            }}
        return true;
    }
private void printSolution(boolean[][] board, int 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();
        }
        System.out.println();
}}

```

Output:



```
            return false;}}
```

```
<terminated> NQueensProblem [Java Application] C:\Program Files\Java\jdk-17\
Q - - - - - - -
- - - - Q - - -
- - - - - - - Q
- - - - - Q - -
- - Q - - - - -
- - - - - - Q -
- Q - - - - - -
- - - Q - - - -
```