# Assignment Day-16&17

## Core Java with DS and Algorithms

**Name: Joshnitha Rangolu**

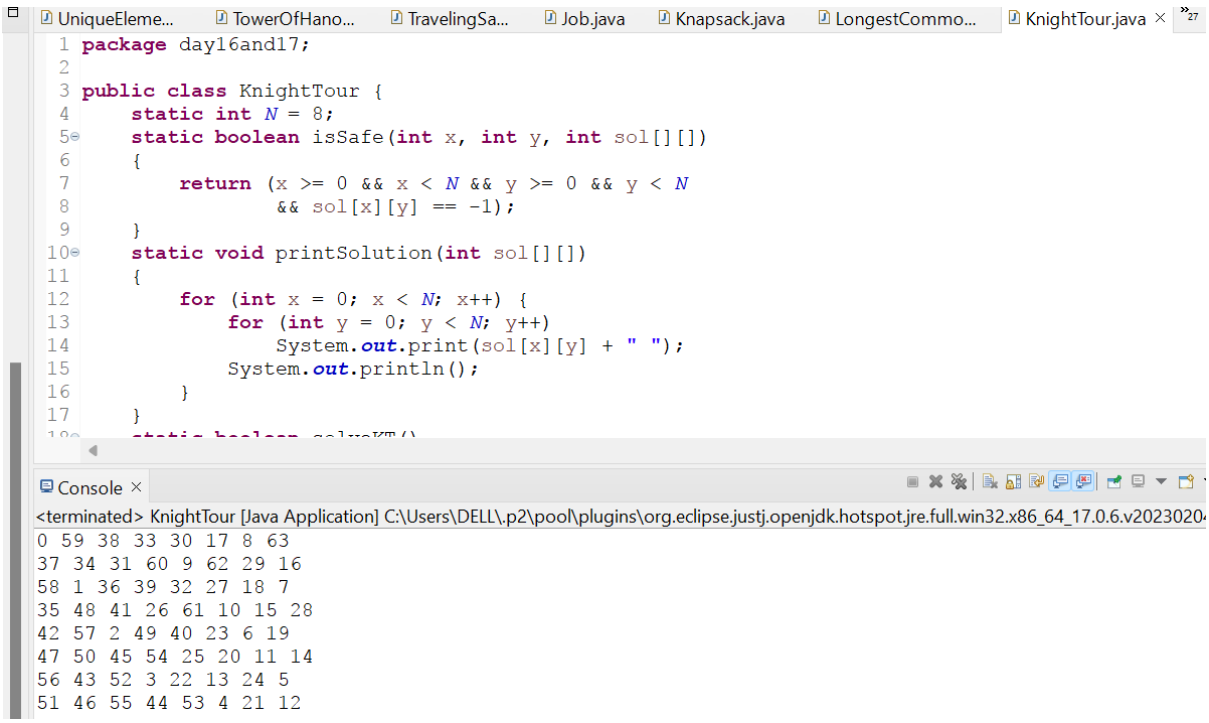
### Task 1: The Knight's Tour Problem

**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.**

```
package day16and17;

public class KnightTour {

static int N = 8;

static boolean isSafe(int x, int y, int sol[][])

{

return (x >= 0 && x < N && y >= 0 && y < N

&& sol[x][y] == -1);

}

static void printSolution(int sol[][])

{

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

for (int y = 0; y < N; y++)

System.out.print(sol[x][y] + " ");

System.out.println();

}

}

static boolean solveKT()

{
```

```
int sol[][] = new int[8][8];

for (int x = 0; x < N; x++)

for (int y = 0; y < N; y++)

sol[x][y] = -1;

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

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

sol[0][0] = 0;

if (!solveKTUtil(0, 0, 1, sol, xMove, yMove)) {

System.out.println("Solution does not exist");

return false;

}

else

printSolution(sol);

return true;

}

static boolean solveKTUtil(int x, int y, int movei,

int sol[][], int xMove[],

int yMove[])

{

int k, next_x, next_y;

if (movei == N * N)

return true;

for (k = 0; k < 8; k++) {

next_x = x + xMove[k];

next_y = y + yMove[k];

if (isSafe(next_x, next_y, sol)) {

sol[next_x][next_y] = movei;

if (solveKTUtil(next_x, next_y, movei + 1,
```

```
sol, xMove, yMove))

return true;

else

sol[next_x][next_y]

= -1; // backtracking

}

}

return false;

}

public static void main(String args[])

{

solveKT();

}

}
```

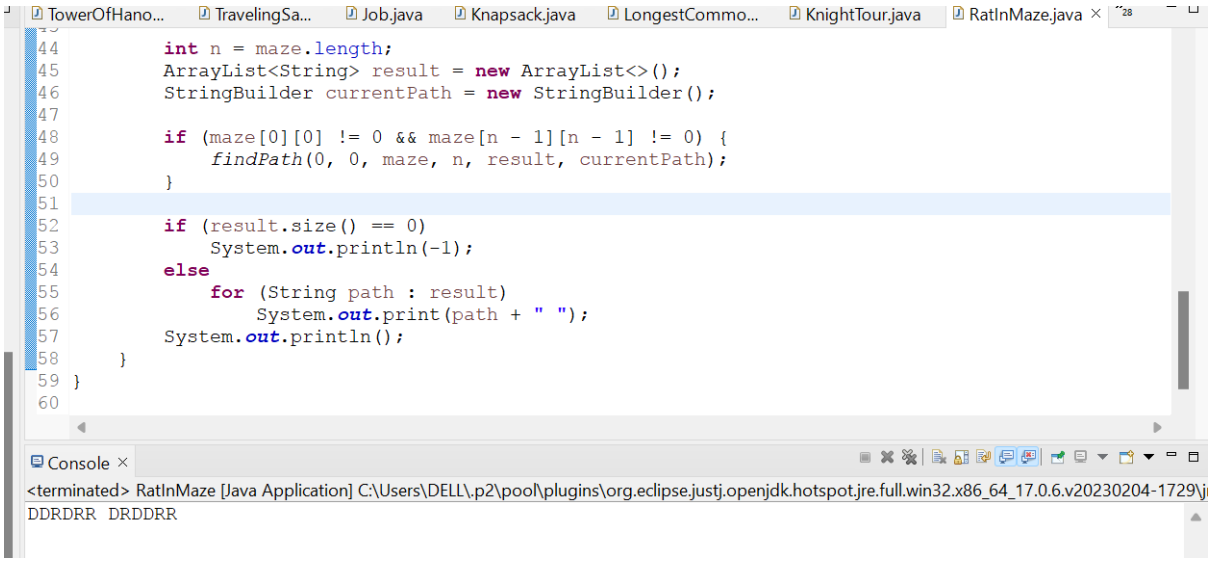


**Task 2: Rat in a Maze**

**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.**

```
package day16and17;

import java.util.ArrayList;

public class RatInMaze {

static String direction = "DLRU";

static int[] dr = { 1, 0, 0, -1 };

static int[] dc = { 0, -1, 1, 0 };

static boolean isValid(int row, int col, int n,int[][] maze)

{

return row >= 0 && col >= 0 && row < n && col < n

&& maze[row][col] == 1;

}

static void findPath(int row, int col, int[][] maze,

int n, ArrayList<String> ans,

StringBuilder currentPath)

{

if (row == n - 1 && col == n - 1) {

ans.add(currentPath.toString());

return;

}

maze[row][col] = 0;

for (int i = 0; i < 4; i++) {

int nextrow = row + dr[i];

int nextcol = col + dc[i];

if (isValid(nextrow, nextcol, n, maze)) {
```

```
currentPath.append(direction.charAt(i));
findPath(nextrow, nextcol, maze, n, ans, currentPath);
currentPath.deleteCharAt(currentPath.length() - 1);
}
}
maze[row][col] = 1;
}
public static void main(String[] args)
{
int[][] maze = { { 1, 0, 0, 0 },
{ 1, 1, 0, 1 },
{ 1, 1, 0, 0 },
{ 0, 1, 1, 1 } };
int n = maze.length;
ArrayList<String> result = new ArrayList<>();
StringBuilder currentPath = new StringBuilder();
if (maze[0][0] != 0 && maze[n - 1][n - 1] != 0) {
findPath(0, 0, maze, n, result, currentPath);
}
if (result.size() == 0)
System.out.println(-1);
else
for (String path : result)
System.out.print(path + " ");
System.out.println();
}
}
```




## 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 day16and17;

public class NQueenProblem {

final int N = 8;

void printSolution(int board[][])

{

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

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

if (board[i][j] == 1)

System.out.print("Q ");

else

System.out.print(". ");

}

System.out.println();

}

}
```

```
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;
}
boolean solveNQUtil(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 (solveNQUtil(board, col + 1) == true)
return true;
board[i][col] = 0;
}
}
return false;
}
```

```
boolean solveNQ()

{

int board[][] = { { 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0 },

{0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0 }};

if (solveNQUtil(board, 0) == false) {

System.out.print("Solution does not exist");

return false;

}

printSolution(board);

return true;

}

public static void main(String args[])

{

NQueenProblem Queen = new NQueenProblem();

Queen.solveNQ();

}

}
```

TravelingSa... Job.java Knapsack.java LongestCommo... KnightTour.java RatInMaze.java *NQueenProb...

```java
package day16and17;

public class NQueenProblem {
     final int N = 8;
        void printSolution(int board[][])
        {
            for (int i = 0; i < N; i++) {
                for (int j = 0; j < N; j++) {
                    if (board[i][j] == 1)
                        System.out.print("Q ");
                    else
                        System.out.print(". ");
                }
                System.out.println();
            }
        }
        boolean isSafe(int board[][], int row, int col)
```

Console

<terminated> NQueenProblem [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.6.v20230204-

```
Q . . . . . . .
. . . . . . Q .
. . . . Q . . .
. . . . . . . Q
. Q . . . . . .
. . . Q . . . .
. . . . . Q . .
. . Q . . . . .
```