Lab 3

The Knight's Tour is an ancient and famous chess puzzle. The object is to move a knight from one square to another on an empty chessboard until it has visited every square **exactly once**. Write a program that solves this puzzle using the idea of depth-first search. It's best to make the board size variable so that you can attempt solutions for smaller boards. The regular 8×8 board can take years to solve on a desktop computer, but a 5×5 board takes only a minute or so. An animation of the Knight's Tour problem can be found from the following link:
https://en.wikipedia.org/wiki/Knight%27s_tour#/media/File:Knights-Tour-Animation.gif

It may be easier to think of a new knight being created and remaining on the new square when a move is made. This way, a knight corresponds to a vertex, and a sequence of knights can be pushed onto a **stack**. When the board is completely filled with knights (the stack is full), you win. In this problem the board is traditionally numbered sequentially, from 1 at the lower-left corner to 64 at the upper-right corner (or 1 to 25 on a 5×5 board). When looking for its next move, a knight must 1) make a legal knight's move, 2) not move off the board or onto an already-occupied (visited) square. A legal knight's move is defined as moving to a square that is two squares horizontally and one square vertically, or two squares vertically and one square horizontally. The following codes may help you to understand how to move a knight on a chessboard.

```
//-------------------------------------------------------------
private static  int[][] moves8 =
     { {+1,-2}, {+2,-1}, {+2,+1}, {+1,+2},
       {-1,+2}, {-2,+1}, {-2,-1}, {-1,-2}  };

   public int getNextPos()               // picks next
     {                                 // possible
     while(cycle < 8)                    // move
        {
        int dx = moves8[cycle][0];        // get move in
        int dy = moves8[cycle][1];        //    (x,y) format

        int x = (position-1)%N;           // translate from j
        int y = (position-1)/N;           // to (x,y) format
        x = x + dx;                     // add move to
        y = y + dy;                     //    position

        cycle++;                         // used this move
        if(x>=0 && x<N && y>=0 && y<N)    // on the board?
          {                            // yes
          int nextPos = x + y*N + 1;     // (x,y) to j
```

```
            if(Board.get(nextPos)==false)  // unoccupied cell?
              {                          // yes
              return nextPos;              // found a move
              }
            }  // end if(x>=0...)
          }  // end while                  // no possible move
        cycle = 0;                        // reset move index
        return -1;                        // failure
        }  // end getNextPos()
```

After executing your program with different inputs, your program should output something like these:

```
Enter board size (8 for 8x8 board): 5
Enter the beginning square (1 to 25): 2
FAILURE:
Total Number of Moves=3658842
Moving Sequence: (2)

Enter board size (8 for 8x8 board): 5
Enter the beginning square (1 to 25): 1
SUCCESS:
Total Number of Moves=57
Moving Sequence: 1 8 5 14 25 18 9 20 23 16 7 4 15 24 17 6 3 10 13 2 11
22 19 12 (21)

Enter board size (8 for 8x8 board): 8
Enter the beginning square (1 to 64): 1
SUCCESS:
Total Number of Moves=6484066
Moving Sequence: 1 11 5 15 32 47 64 54 39 24 30 40 55 61 46 31 16 22 7
13 23 8 14 29 44 38 48 63 53 59 49 34 19 4 21 6 12 2 17 27 37 52 58 41
35 20 10 25 42 57 51 36 26 9 3 18 28 43 33 50 60 45 62 (56)
```

Note: if you implement the getNextPos() function in a different way, you may expect different "Total Number of Moves" and "Moving Sequence".