```
Output was cut down to save paper

Enter M then N. Where N >= M
5
8
-----------------
|Q| | | | | | | |
-----------------
| | |Q| | | | | |
-----------------
| | | | |Q| | | |
-----------------
| |Q| | | | | | |
-----------------
| | | |Q| | | | |
-----------------

-------------------------------------------------------------
Enter M then N. Where N >= M
8
10
---------------------
|Q| | | | | | | | | |
---------------------
| | | | |Q| | | | | |
---------------------
| | | | | | | |Q| | |
---------------------
| | | | | |Q| | | | |
---------------------
| | |Q| | | | | | | |
---------------------
| | | | | | |Q| | | |
---------------------
| |Q| | | | | | | | |
---------------------
| | | |Q| | | | | | |
---------------------
```

```java
import java.util.LinkedList;
import java.util.Random;

public class Queens {

    //Board class (inner class)
    private class Board{

        private char[][] array;                 //2d Array is the board
        private int rows;                       //filled rows

        //Constructor
        private Board(int m, int n){
            array = new char[m][n];             //create array

            //initialize array to blanks
            for(int i = 0; i < m; i++)
                for(int j = 0; j < n; j++)
                    array[i][j] = ' ';

            rows = 0;
        }
    }

    private int m;
    private int n;

    //Queens Class Constructor
    public Queens(int m, int n){
        this.m = m;
        this.n = n;
    }

    //Solve Queens Problem
    public  void solve(){
        LinkedList<Board> list = new LinkedList<Board>(); //List of boards
        LinkedList<Board> complete = new LinkedList<Board>(); //List of complete boards
        Random rand = new Random();            //RNG

        Board board = new Board(m,n);          //Create empty board
        list.addFirst(board);                  //Add to list

        while(!list.isEmpty()) {                //While list has boards
            board = list.removeFirst();         //Remove first board

            if(complete(board)) {               //If the board is a solution
                int choice = rand.nextInt(1);
                if(choice == 0)
                    complete.addFirst(board);
                else
                    complete.addLast(board);
            }else{
                LinkedList<Board> children = generate(board);

                for(int i = 0; i < children.size(); i++)
                    list.addFirst(children.get(i));
            }
        }
        if(complete.isEmpty())
            System.out.println("No Solution");      //Print if there
 is no solution
        else
            printList(complete);
    }

    private void printList(LinkedList<Board> complete) {
        while(!complete.isEmpty())
            display(complete.removeFirst());
    }

    //Method generates children of a board
    private LinkedList<Board> generate(Board board){
        LinkedList<Board> children = new LinkedList<Board>(); //
 Children list

        for(int i = 0; i < m; i++){             //Generate children
            Board child = copy(board);          //Create copy of
 parent
            child.array[child.rows][i] = 'Q';   //Put queen in the
 row

            if(check(child, child.rows, i))
                children.addLast(child);

            child.rows ++;                      //Increment Filled
 Rows
        }
        return children;                        //Return List of
 children
    }

    //Method checks whether queen at a given location causes conflict
    private boolean check(Board board, int x, int y){
        for(int i = 0; i < m; i ++)             //Go thru all
 locations
            for(int j = 0; j < n; j++){
                if(board.array[i][j] == ' ');   //If empty ignore
                else if(x == i && y == j);      //If same location
 ignore
                else if(x == i || y == j || x+y == i+j || x-y == i-j)
                    return false;               //Conflict if in same
  row, column, or diagonal
            }
```

```java
        return true;                            //No conflicts
    }

    //Method makes copy of board
    private Board copy(Board board){
        Board result = new Board(m,n);          //Empty board

        for(int i = 0; i < m; i++)              //Copy given board to empty board
            for(int j = 0; j < n; j++)
                result.array[i][j] = board.array[i][j];

        result.rows = board.rows;               //Copy filled rows

        return result;                          //Return copy
    }

    //Checks if board is complete
    private Boolean complete(Board board){
        return(board.rows == m);                //Check number filled rows equals board size
    }

    //Displays board
    private void display(Board board){
        for(int j = 0; j < n + 1; j++)              //Top horizontal line
            System.out.print("--");

        System.out.println();

        for(int i = 0; i < m; i++){             //Every row
            System.out.print("|");            //First Line
            for(int j = 0; j < n; j++)          //Slots
                System.out.print(board.array[i][j] + "|");

            System.out.println();               //Next Line

            for(int j = 0; j < n + 1; j++)          //Horizontal line
                System.out.print("--");

            System.out.println();               //Next Line
        }
    }
}

```

```java
import java.util.Scanner;
public class QueensTester {
    public static void main(String[] args){
        Scanner keyIn = new Scanner(System.in);
        int m;
        int n;
        Queens q;
        System.out.println("Enter M then N. Where N >= M");
        m = keyIn.nextInt();
        n = keyIn.nextInt();

        if(m <= n) {
            q = new Queens(m, n);
            q.solve();
        }
        else
            System.out.println("Error: N must be >= M");
    }
}

```