import java.util.Arrays;

public class QBoard implements IQueenState {
    private int[] myBoardRows;
    private static int NONE = 100000; // invalid row, and calculations ok

    public QBoard(int n) {
        myBoardRows = new int[n];
        Arrays.fill(myBoardRows, NONE);
    }

    public boolean safeToPlace(int row, int col) {
        // check each column to see if conflict exists
        for (int c = 0; c < myBoardRows.length; c++) {
            if (myBoardRows[c] == row) return false;
            if (myBoardRows[c] == row - col) return false;
            if (myBoardRows[c] == row + col) return false;
        }
        return true;
    }

    public void setQueen(int row, int col, boolean value) {
        myBoardRows[col] = value ? row : NONE;
    }

    public void print() {
        for (int r = 0; r < myBoardRows.length; r++) {
            for (int c = 0; c < myBoardRows.length; c++) {
                if (myBoardRows[c] == r) {
                    System.out.print("Q");
                } else {
                    System.out.print(".");
                }
            }
            System.out.println();
        }
    }
}

/**
 * Interface for an nxn board for the N−Queens problem. The interface
 * is meant to facilitate graphical/non−graphical views of a board.
 * @author ola
 */
public interface IQueenState {
    /**
     * Determine if a queen can be placed at (row,col) on the board, return true
     * if the queen can be placed without attack by previously placed queens, false
     * otherwise.
     * @param row is row being considered for placement
     * @param col is column being considered for placement
     * @return true iff queen can be placed at (row,col) without attack by other
     * queens placed
     */
    public boolean safeToPlace(int row, int col);

    /**
     * Set or unset a queen at (row,col) depending on value, e.g., value == true
     * means queen is placed at (row,col), otherwise queen is removed from (row,
     * col)
     * @param row is row at which queen state is set
     * @param col is column at which queen state is set
     * @param value determines if queen is placed (true) or removed (false)
     */
    public void setQueen(int row, int col, boolean value);

    /**
     * Print some version of the board indicating where queens are placed.
     */
    public void print();
}
public class Queens {
  private IQueenState myBoard;
  private int mySize;
  private int myCount;

  public Queens(int n) {
    mySize = n;
    myBoard = new QBoardGUI(n);
    if (solve(0)) {
      myBoard.print();
    }
  }

  /**
   * Queens have been placed in all columns [0..col), try to place
   * a queen in column <code>col</code> and all columns after
   * it, returning true if this is possible, false otherwise.
   * @param col is left−most column with no queen in it
   * @return true if a queen can be placed in all columns [col..size)
   */
  public boolean solve(int col) {
    if (col == mySize) return true;
    // try each row until all are tried
    for (int r = 0; r < mySize; r++) {
      if (myBoard.safeToPlace(r, col)) {
        myBoard.setQueen(r, col, true);
        if (solve(col + 1)) {
          //myCount++;
          return true;
        }
        myBoard.setQueen(r, col, false);
      }
    }
    return false;
  }

  public int getCount() {
    return myCount;
  }

  public static void main(String[] args) {
    int size = 8;
    double start = System.currentTimeMillis();
    Queens q = new Queens(size);
    System.out.println("# ways = " + q.getCount());
    double end = System.currentTimeMillis();
    System.out.printf("time: %f", (end - start) / 1000.0);
  }
}