```java
package hanoi;

import util.Stack;

/**
 * Contains all the methods to solve and display the Hanoi tower problem
 *
 * @author Kevin Farine, Timothee Van Hove
 */
public class Hanoi {
    /**
     * The number of needles (towers) used in the problem.
     */
    private static final int NB_NEEDLES = 3;

    /**
     * The towers name to be displayed.
     */
    private static final String[] TOWER_NAMES = {"One", "Two", "Three"};

    /**
     * The container of the needles.
     */
    private final Stack[] needles;

    /**
     * The number of disks stacked on the first needle defined by the user.
     */
    private final int nbDisks;

    /**
     * Displayer of the Hanoi towers problem execution.
     */
    private final HanoiDisplayer displayer;

    /**
     * The number of moves required to solve the problem with n disks.
     */
    private int nbMoves;

    /**
     * Hanoi generic constructor.
     *
     * @param nbDisks The number of disks stacked on the first needle.
     * @param displayer The displayer that displays the execution of the Hanoi towers problem.
     */
    public Hanoi(int nbDisks, HanoiDisplayer displayer) {
        if (nbDisks < 1){
            throw new RuntimeException("The disks count cannot be < 1");
        }

        this.nbDisks = nbDisks;
        this.displayer = displayer;

        // Stacks initialization
        needles = new Stack[NB_NEEDLES];
        for (int i = 0; i < NB_NEEDLES; ++i) {
            needles[i] = new Stack();
        }

        // Puts all the disks onto the first needle beginning by the last one
        for (int i = nbDisks; i > 0; --i) {
            needles[0].push(i);
        }
    }

    /**
     * Hanoi constructor used for the console display.
     *
     * @param disks The number of disks stacked on the first needle.
     */
    public Hanoi(int disks) {
        this(disks, new HanoiDisplayer());
    }

    /**
     * Solves the Hanoi towers problem using a recursive algorithm.
     */
    public void solve() {
        displayer.display(this);
        hanoiAlgorithm(needles[0], needles[1], needles[2], nbDisks);
    }

    /**
     * Returns a 2d array representation of the needles current status.
     *
     * @return a 2d int array representing the status of the 3 needles.
     */
    public int[][] status() {
        int[][] status = new int[NB_NEEDLES][];

        for (int i = 0; i < NB_NEEDLES; ++i) {
            //Get the current state of each stack
            Object[] currentState = needles[i].getCurrentState();
            status[i] = new int[currentState.length];

            for (int j = 0; j < currentState.length; ++j) {
                //Cast the value of each Item in the stack to int
                status[i][j] = (int)currentState[j];
            }
        }
        return status;
    }

    /**
     * Indicates if the run is finished (when 2^n - 1 moves have been done).
     *
     * @return True if the run is finished, false if not.
     */
    public boolean finished() {
        return nbMoves == Math.pow(2, nbDisks) - 1;
    }

    /**
     * Indicates the number of moves that were required to solve the problem with n disks.
     *
     * @return The number of moves to solve the problem.
     */
    public int turn() {
        return nbMoves;
    }

    /**
     * String representation of the current state of the 3 needles.
     *
     * @return The string representation of the needles current state.
     */
    @Override
    public String toString() {
        StringBuilder state = new StringBuilder("-- Turn: " + turn() + "\n");
        for (int i = 0; i < NB_NEEDLES; ++i) {
            state.append(String.format("%-7s%s", TOWER_NAMES[i] + ":", needles[i]));
            if(i != NB_NEEDLES - 1){
                state.append("\n");
            }
        }
        return state.toString();
    }

    /**
     * Hanoi recursive algorithm used to solve the Hanoi towers problem with 3 needles.
     * Complexity in O(n^2 - 1).
     * @param from The first needle. At the beginning all disks are placed onto this needle.
     * @param via The intermediate needle.
     * @param to The final needle. At the end all disks are placed onto that needle.
     * @param nbDisks The number of disks to put on the first needle.
     */
    private void hanoiAlgorithm(Stack from, Stack via, Stack to, int nbDisks) {
        if (nbDisks > 0) {
            hanoiAlgorithm(from, to, via, nbDisks - 1);

            // Transfers the top disk between 2 stacks
            to.push(from.pop());
            ++nbMoves;
            displayer.display(this);
            hanoiAlgorithm(via, from, to, nbDisks - 1);
        }
    }
}
```