```java
/**
 *  -------------------------------------------------------------------------------
 * @Fichier      : Hanoi.java
 * @Labo         : Laboratoire 7 : Tours de Hanoï
 * @Auteurs      : Slimani Walid & Baume Oscar
 * @Date         : 09.11.2022
 *
 * @Description  : Ce fichier définit la classe Hanoi. Cette classe permet de
 *                 modéliser et de résoudre le casse tête des tours de Hanoi.
 * @Remarque     : / Aucune remarque
 * @Modification : / Aucune modification
 *  -------------------------------------------------------------------------------
 **/

package hanoi;
import util.Stack;

public class Hanoi {
    // region ctor

    /**
     * Nom                   : Hanoi
     * Description           : Permet de construire le casse-tête des tours de Hanoï en spécifiant le
     nombre
     *                         de disque et en spécifiant le HanoiDisplayer qui va afficher l'état des
     *                         tours de Hanoï
     * @param nbRing         : Nombre de disque à déplacer
     * @param hanoiDisplayer : Objet de type HanoiDisplayer permettant d'afficher l'état du casse-tête
     * @return               : L'objet Hanoi construit par le constructeur
     **/
    public Hanoi(int nbRing, HanoiDisplayer hanoiDisplayer) {
        this(nbRing);

        if (hanoiDisplayer == null) {
            throw new RuntimeException("Le Hanoï displayer est null");
        }

        this.hanoiDisplayer = hanoiDisplayer;
    }

    /**
     * Nom           : Hanoi
     * Description   : Permet de construire le casse-tête des tours de Hanoï en spécifiant le nombre
     *                 de disque
     * @param nbRing : Nombre de disque à déplacer
     * @return       : L'objet Hanoi construit par le constructeur
     **/
    public Hanoi(int nbRing) {
        if (nbRing <= 0) {
            throw new RuntimeException("Le nombre de disque n'est pas valable");
        }

        this.nbRing = nbRing;

        for (int i = 0; i < nbStack; ++i) {
            stacks[i] = new Stack<>();
        }

        for (int val = nbRing; val > 0; val--) {
            stacks[0].push(val);
        }
    }
    // endregion

    // region paramètre
    private final int nbStack = 3;  // Nombre d'aiguille
    private Stack<Integer>[] stacks = new Stack[nbStack];   // Aiguilles modélisant les tours de Hanoï
    private HanoiDisplayer hanoiDisplayer;  // Objet permettant d'afficher les tours de Hanoï
    private final int nbRing;   // Nombre de disques utilisés pour résoudre le casse-tête
    private int turn;   // Nombre de déplacement de disque effectué
    // endregion

    // region methodes

    /**
     * Nom           : solve
     * Description   : Permet de résoudre le casse-tête en affichant l'état des tours
```

```
     *                à chaque fois qu'un disque est déplacé
     * @return       : void
     **/
    public void solve() {
        if (hanoiDisplayer != null) {
            hanoiDisplayer.display(this);
        } else {
            System.out.println(this);
        }
        solve(nbRing, stacks[0], stacks[2], stacks[1]);
    }


    /**
     * Nom           : solve
     * Description   : Algorithme récursif permettant de résoudre le casse-tête
     * @param n      : Nombre de disque à transférer
     * @param from   : Représente l'aiguille numéro 1 (celle ou tous les disques sont empilés au
     commencement)
     * @param to     : Représente l'aiguille numéro 3 (celle ou tous les disques sont empilés à la fin)
     * @param other  : Représente l'aiguille numéro 2 (aiguille intérmédiaire)
     * @return       : void
     **/
    private void solve(int n, Stack from, Stack to, Stack other) {
        if (n == 1) {
            move(from, to);
            if (hanoiDisplayer != null) {
                hanoiDisplayer.display(this);
            } else {
                System.out.println(this);
            }
            return;
        }
        solve(n - 1, from, other, to);
        move(from, to);
        if (hanoiDisplayer != null) {
            hanoiDisplayer.display(this);
        } else {
            System.out.println(this);
        }
        solve(n - 1, other, to, from);
    }

    /**
     * Nom           : move
     * Description   : Permet de déplacer un disque d'une aiguille à une autre
     * @param from   : Représente l'aiguille ou se situe le disque à déplacer
     * @param to     : Représente l'aiguille ou il faut déplacer le disque
     * @return       : void
     **/
    private void move(Stack<Integer> from, Stack<Integer> to) {
        try {
            if (to.getHead() != null && from.top() >= to.top())
                throw new RuntimeException("Disque trop grand pour etre deplacer");

        } catch (RuntimeException e) {
        }
        int val = from.pop();
        to.push(val);
        turn++;
    }


    /**
     * Nom           : status
     * Description   : Permet de retourner le statu de chaque aiguille
     * @return       : Un tableau de Stack (un tableau de 3 aiguilles)
     **/
    public int[][] status() {
        int[][] out = new int[nbStack][];

        for (int i = 0; i < nbStack; ++i) {
            Object[] tour = stacks[i].toArray();
            int[] t = new int[tour.length];

            for (int j = 0; j < tour.length; ++j) {
                t[j] = (Integer) tour[j];
```

```
            }
            out[i] = t;
        }
        return out;
    }


    /**
     * Nom           : finished
     * Description   : Indique si la résolution du casse-tête est terminé
     * @return       : Retourne true si le casse-tête est résolu
     **/
    public boolean finished() {
        int[][] test = status();
        return test[nbStack - 1].length == nbRing;
    }

    /**
     * Nom           : turn
     * Description   : Retourne le nombre disque déplacé
     * @return       : Entier indiquant combien de disque ont été déplacé
     **/
    public int turn() {
        return turn;
    }


    /**
     * Nom           : toString
     * Description   : Permet d'afficher l'état actuel des tours de Hanoï
     * @return       : String représentant l'état des 3 aiguilles du casse-tête
     **/
    public String toString() {
        final String name[] = new String[]{"One", "Two", "Three"};
        String out = "-- Turn : " + turn + "\n";
        for (int i = 0; i < nbStack; i++) {
            out += String.format("%-6s %s", name[i] + ":", stacks[i]);
        }
        return out;
    }
    // endregion
}
```