```java
/**
 *  -------------------------------------------------------------------------------------
 * @Fichier      : Main.java
 * @Labo         : Laboratoire 7 : Tours de Hanoï
 * @Auteurs      : Slimani Walid & Baume Oscar
 * @Date         : 09.11.2022
 *
 * @Description  : Le but de ce programme est de résoudre le problème des tours de Hanoï.
 *                 De plus, afin de visualiser la résolution du casse-tête une interface
 *                 graphique est présente.
 * @Remarque     : Le nombre de minimum de coup nécessaire pour résoudre le casse-tête est
 *                 (2^nombre de disque) - 1 (Exemple : pour 3 disques il faut (2^3) - 1 = 7).
 * @Modification : / Aucune modification
 *  -------------------------------------------------------------------------------------
**/

import hanoi.*;
import hanoi.gui.JHanoi;

public class Main {
    public static void main(String[] args) {

        if (args.length == 0) {
            new JHanoi();
        }

        if (args.length == 1) {
            int nbrDisk = 0;

            try {
                nbrDisk = Integer.parseInt(args[0]);
            }

            catch (Exception e) {
                throw new RuntimeException("Entré non valable");
            }

            if (nbrDisk <= 0) {
                throw new RuntimeException("Le nombre de disques doit être supérieur à 0");
            }

            Hanoi h = new Hanoi(nbrDisk);
            h.solve();
        }
    }
}
```

```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
}
```

```java
/**
 *  -------------------------------------------------------------------------------------
 * @Fichier      : HanoiDisplayer.java
 * @Labo         : Laboratoire 7 : Tours de Hanoï
 * @Auteurs      : Slimani Walid & Baume Oscar
 * @Date         : 09.11.2022
 *
 * @Description  : Ce fichier définit la classe HanoiDisplayer qui s'occupe d'afficher
 *                 l'état de la tour de Hanoï lors de la résolution de cette dernière.
 * @Remarque     : / Aucune remarque
 * @Modification : / Aucune modification
 *  -------------------------------------------------------------------------------------
 **/

package hanoi;

public class HanoiDisplayer {
    // region Méthode

    /**
     * Nom         : display
     * Description : Affiche l'état actuel de la tour de Hanoi
     * @param h    : La tour de Hanoï à afficher
     * @return     : void
     **/
    public void display(Hanoi h) {
        System.out.println(h);
    }
    // endregion
}
```

```java
/**
 *  -------------------------------------------------------------------------------------
 * @Fichier      : TestHanoi.java
 * @Labo         : Laboratoire 7 : Tours de Hanoï
 * @Auteurs      : Slimani Walid & Baume Oscar
 * @Date         : 25.11.2022
 *
 * @Description  : Le but de ce programme est d'effectuer divers tests afin de contrôler
 *                 l'efficacité de notre classe Hanoï.
 * @Remarque     : Le nombre de minimum de coup nécessaire pour résoudre le casse-tête est
 *                 (2^nombre de disque) - 1 (Exemple : pour 3 disques il faut (2^3) - 1 = 7).
 * @Modification : / Aucune modification
 *  -------------------------------------------------------------------------------------
 **/

package test;
import hanoi.*;

public class TestHanoi {
    public static void main(String[] args) {
        // region Test de construction
        System.out.println("Construction(nbring) sans erreur : \n");
        Hanoi h1 = new Hanoi(3);

        System.out.println("Construction(nbring, displayer) sans erreur :\n");
        Hanoi h2 = new Hanoi(3, new HanoiDisplayer());

        try {
            System.out.println("Construction(nbring) avec erreur :");
            Hanoi h3 = new Hanoi(0);
        } catch (Exception e) {
            System.out.println(e + "\n");
        }

        try {
            System.out.println("Construction(nbring) avec erreur :");
            Hanoi h4 = new Hanoi(-3);
        } catch (Exception e) {
            System.out.println(e + "\n");
        }

        try {
            System.out.println("Construction(nbring, displayer) avec erreur :");
            Hanoi h5 = new Hanoi(3, null);
        } catch (Exception e) {
            System.out.println(e + "\n");
        }
        // endregion

        // region Test du nombre de déplacements de disques et du booléen finished
        Hanoi h = new Hanoi(3, new HanoiDisplayer());
        System.out.println("finished = " + h.finished());
        System.out.println("nbr de coups avant résolution = " + h.turn());
        try {
            h.solve();
        }
        catch (Exception e) {
            System.out.println(e + "\n");
        }
        System.out.println("finished = " + h.finished());
        System.out.println("nbr de coups après résolution = " + h.turn());
        System.out.println("Pour info, le nombre minimum de déplacement est 2^nDisk - 1 = 2^3 - 1 = 8 -
        1 = 7");
        // endregion
    }
}
```

```java
/**
 *  ------------------------------------------------------------------------------------
 * @Fichier      : TestStack.java
 * @Labo         : Laboratoire 7 : Tours de Hanoï
 * @Auteurs      : Slimani Walid & Baume Oscar
 * @Date         : 25.11.2022
 *
 * @Description  : Le but de ce programme est d'effectuer divers tests afin de contrôler
 *                 l'efficacité de notre classe Stack.
 * @Remarque     : / Aucune remarque
 * @Modification : / Aucune modification
 *  ------------------------------------------------------------------------------------
 **/

package test;
import util.*;

public class TestStack {

    public static void main(String[] args) {
        // region Test des fonctionnalités de la classe Stack
        System.out.println("Construction de la stack :");
        Stack<String> stack = new Stack<>();
        System.out.println(stack);
        //test pop sur stack vide
        try{
            stack.pop();
        } catch (RuntimeException e){
            System.out.println(e.getMessage());
        }
        // test top sur stack vide
        try{
            stack.top();
        } catch (RuntimeException e){
            System.out.println(e.getMessage());
        }
        // test création iterateur sur stack vide
        try{
            ElementIterator it = new ElementIterator(stack.getHead());
        } catch (RuntimeException e){
            System.out.println(e.getMessage());
        }

        stack = new Stack<String>();
        System.out.println("On ajoute dans la stack les valeurs suivantes : 7, 14, 95, 33 et 444 :");
        stack.push("7");
        stack.push("14");
        stack.push("95");
        stack.push("33");
        stack.push("444");
        System.out.println(stack);

        System.out.println("On dépile deux fois donc 444 et 33 disparaissent :");
        stack.pop();
        stack.pop();
        System.out.println(stack);

        System.out.println("On ajoute 23 :");
        stack.push("23");
        System.out.println(stack);

        System.out.println("On dépile deux fois donc 23 et 95 disparaissent, puis on affiche le sommet
        (14) :");
        stack.pop();
        stack.pop();
        System.out.println("Top = " + stack.top() + "\n");

        System.out.println("Affichage de la stack :");
        System.out.println(stack);
        // endregion

        // region Test de la génération d'un tableau d'objets à partir d'une Stack
        System.out.println("Construction d'une autre stack (10, 20, 30, 40, 50) : ");
        Stack<String> stack2 = new Stack<>("10", "20", "30", "40", "50");
        System.out.println(stack2);

        System.out.println("Génération d'un tableau d'objets à partir de la deuxième stack :");
```

```
        Object[] tab = stack2.toArray();

        System.out.print("[ ");
        for(Object o : tab) {
            System.out.print("<" + o + ">" + " ");
        }
        System.out.println("]\n");
        // endregion

        // region Test des itérateurs
        System.out.println("Création d'un itérateur qui pointe sur le sommet de la deuxième stack :");
        ElementIterator it = new ElementIterator(stack2.getHead());
        System.out.println("l'itérateur pointe sur -> " + it + "\n");

        System.out.println("Tant que la pile contient des élément on itére et affiche le contenu :");
        while(it.hasNext()) {
            System.out.println("next -> " + it.next());
        }
        // endregion
    }
}
```

```java
/**
 *  -----------------------------------------------------------------------------------
 * @Fichier      : ElementStack.java
 * @Labo         : Laboratoire 7 : Tours de Hanoï
 * @Auteurs      : Slimani Walid & Baume Oscar
 * @Date         : 09.11.2022
 *
 * @Description  : Ce fichier définit la classe ElementStack. Cette classe permet de
 *                 modéliser les éléments contenus dans la Stack
 * @Remarque     : / Aucune remarque
 * @Modification : / Aucune modification
 *  -----------------------------------------------------------------------------------
 **/

package util;

class ElementStack<T> {
    // region ctor

    /**
     * Nom          : ElementStack
     * Description  : Permet de construire un element d'une stack
     * @param value : Valeur de l'élément de type <T>
     * @param next  : Element suivant d'un élément de la pile
     * @return      : L'objet ElementStack construit par le constructeur
     **/
    public ElementStack(T value, ElementStack<T> next) {
        this(value);
        this.next = next;
    }

    /**
     * Nom          : ElementStack
     * Description  : Permet de construire un element d'une stack qui ne possède pas d'élément suivant
     * @param value : Valeur de l'élément de type <T>
     * @return      : L'objet ElementStack construit par le constructeur
     **/
    public ElementStack(T value) {
        next = null;
        this.value = value;
    }
    // endregion

    // region paramètre
    private T value;    // Valeur de l'élément
    private ElementStack<T> next;   // Elément suivant de la Stack
    // endregion

    // region methodes

    /**
     * Nom         : getNext
     * Description : Retourne l'élement suivant qui est dans la Stack
     * @return     : L'élément suivant en question
     **/
    public ElementStack<T> getNext() {
        return next;
    }

    /**
     * Nom         : getValue
     * Description : Retourne la valeur de l'élément courant
     * @return     : La valeur de type <T> de l'élément courant
     **/
    public T getValue() {
        return value;
    }

    /**
     * Nom         : toString
     * Description : Retourne la valeur de l'élément courant
     * @return     : String représentant la valeur de l'élément courant
     **/
    public String toString() {
        return (String) value;
    }
    // endregion
```

```
}
```

```java
/**
 *  -----------------------------------------------------------------------------------
 * @Fichier      : Stack.java
 * @Labo         : Laboratoire 7 : Tours de Hanoï
 * @Auteurs      : Slimani Walid & Baume Oscar
 * @Date         : 09.11.2022
 *
 * @Description  : Ce fichier définit la classe Stack. Cette classe permet de
 *                 modéliser la strucutre de donnée "Pile".
 * @Remarque     : La pile modélisée est en réalité une liste simplement chaînée
 * @Modification : / Aucune modification
 *  -----------------------------------------------------------------------------------
 **/

package util;

public class Stack<T> {
    // region ctor

    /**
     * Nom          : Stack
     * Description  : Permet de construire une Stack avec un nombre d'élément variable
     * @param values : Valeur(s) de(s) élément(s) à ajouter dans la pile
     * @return       : L'objet Stack construit par le constructeur
     **/
    public Stack(T... values) {
        for (T val : values) {
            push(val);
        }
    }
    // endregion

    // region paramètre
    private ElementStack<T> head;   // Représente le sommet de la pile
    // endregion

    // region method

    /**
     * Nom           : push
     * Description   : Permet d'ajouter un élément au sommet de la pile
     * @param  value : valeur de l'élément à ajouter au sommet de la pile
     * @return       : void
     **/
    public void push(T value) {
        if (head == null) {
            head = new ElementStack<>(value);
        } else {
            head = new ElementStack<>(value, head);
        }
    }

    /**
     * Nom           : pop
     * Description   : Permet d'enlever l'élément se situant au sommet de la pile
     * @return       : L'élément de type <T> qui était au sommet de la pile
     **/
    public T pop() {
        if (head == null)
            throw new RuntimeException("pop() : La stack est vide");

        ElementIterator it = new ElementIterator(head);
        T value = head.getValue();
        head = it.next();
        return value;
    }

    /**
     * Nom           : top
     * Description   : Retourne la valeur de l'élément se situant au sommet de la pile
     * @return       : La valeur de type <T> de l'élément qui est au sommet de la pile
     **/
    public T top() {
        if (head == null) {
            throw new RuntimeException("top() : La stack est vide");
        }
        return head.getValue();
```

```
    }

    /**
     * Nom           : toString
     * Description   : Permet d'afficher la pile
     * @return       : String représentant l'état de la pile
     **/
    public String toString() {
        String rt = "[ ";
        ElementIterator it = new ElementIterator(head);

        if (head != null) {
            rt += "<" + head.getValue() + "> ";
            while (it.hasNext()) {
                rt += "<" + it.next().getValue() + "> ";
            }

        }
        return rt + "]\n";
    }

    /**
     * Nom           : toArray
     * Description   : Permet de convertir une stack en un tableau "d'Object"
     * @return       : Le tableau "d'Object"
     **/
    public Object[] toArray() {
        Object out[] = new Object[0];
        int pos = 0;
        if (head != null) {
            ElementIterator it = new ElementIterator(head);
            pos++;
            while (it.hasNext()) {
                it.next();
                pos++;
            }
            it = new ElementIterator(head);
            out = new Object[pos];
            pos = 0;
            out[pos++] = head.getValue();
            while (it.hasNext()) {
                out[pos++] = it.next().getValue();
            }
        }
        return out;
    }

    /**
     * Nom         : getHead
     * Description : retourne le sommet de la pile
     * @return     : Un ElementStack qui est le sommet de la pile
     * Remarque    : /!\ cette fonction a été implémenté UNIQUEMENT pour pouvoir tester le bon
     fonctionnement
     *               de la classe Stack (voir le test "Test des itérateurs" dans la classe "TestStack")
     /!\
     **/
    public ElementStack<T> getHead() {
        if(head == null) throw new RuntimeException("getHead() : la stack est vide");
        return head;
    }
    // endregion
}
```

```java
/**
 *  -------------------------------------------------------------------------------------
 * @Fichier      : ElementIterator.java
 * @Labo         : Laboratoire 7 : Tours de Hanoï
 * @Auteurs      : Slimani Walid & Baume Oscar
 * @Date         : 09.11.2022
 *
 * @Description  : Ce fichier définit la classe ElementIterator. Cette classe permet de
 *                 parcourir la stack.
 * @Remarque     : / Aucune remarque
 * @Modification : / Aucune modification
 *  -------------------------------------------------------------------------------------
 **/

package util;

public class ElementIterator {
    // region ctor

    /**
     * Nom         : ElementIterator
     * Description : Permet de construire un itérateur permettant d'itérer sur une Stack contenant des
     *               StackElement. A la construction, l'itérateur pointe sur le StackElement qui est
     *               passé en paramètre du constructeur
     * @param it  : ElementStack sur lequel pointe l'itérateur
     * @return    : L'objet ElementIterator construit par le constructeur
     **/
    public ElementIterator(ElementStack it) {
        this.it = it;
    }
    // endregion

    // region Paramètre
    private ElementStack it;    // Element de la stack pointé par l'itérateur
    // endregion

    // region Méthode

    /**
     * Nom         : next
     * Description : Retourne l'élement suivant qui est dans la Stack
     * @return     : L'élément suivant en question
     **/
    public ElementStack next() {
        if (hasNext()) {
            it = it.getNext();
            return it;
        }
        return null;

    }

    /**
     * Nom         : hasNext
     * Description : Indique si l'élément courrant est suivi d'un autre élément
     * @return     : Vrai si l'élément courrant est suivi par un autre élément
     **/
    public boolean hasNext() {
        if(it == null) throw new RuntimeException("L'iterateur est null");
        return it.getNext() != null;
    }

    /**
     * Nom         : toString
     * Description : Indique la valeur de l'élément sur lequel pointe l'itérateur
     * Remarque    : /!\ cette fonction a été implémenté UNIQUEMENT pour pouvoir tester le bon
     fonctionnement
     *      *        de la classe Stack (voir le test "Test des itérateurs" dans la classe "TestStack")
     /!\
     * @return     : String contenant la valeur de l'ElementStack sur lequel pointe l'itérateur
     **/
    public String toString() {
        return (String) it.getValue();
    }
    // endregion
}
```