

## Main.java - Sarah Jallon Tegest Bogale

```
1 import hanoi.Hanoi;
2 import hanoi.gui.JHanoi;
3
4
5 /**
6  * Permet de lancer le programme en ligne de commande ou avec l'interface graphique
7  *
8  * @author Bogale Tegest & Jallon Sarah
9  */
10 public class Main {
11     /**
12      * Lance le programme en ligne de commande ou avec l'interface graphique
13      * @param args arguments donnés en ligne de commande
14      * @throws RuntimeException si le nombre de disques est < 1 ou si hanoi Displayer est null
15      */
16     public static void main(String[] args) {
17         if (args.length == 1) {
18             Hanoi hanoi = new Hanoi(Integer.parseInt(args[0]));
19             hanoi.solve();
20         } else if (args.length == 0) {
21             new JHanoi();
22         } else {
23             System.out.println("To use the interface run the program without arguments");
24             System.out.println("To use the console run the program with the number of disks");
25         }
26     }
27 }
28 }
```

## TestHanoi.java - Sarah Jallon Tegest Bogale

```
1 package test;
2
3 import hanoi.Hanoi;
4 import hanoi.HanoiDisplay;
5
6 import java.util.Arrays;
7
8 /**
9  * Classe de test mettant en œuvre toutes les fonctionnalités de la classe Hanoi
10  *
11  * @author Bogale Tegest & Jallon Sarah
12  */
13 public class TestHanoi {
14     private static HanoiDisplay hanoiDisplay = new HanoiDisplay();
15     private static Hanoi hanoi;
16
17     public static void main(String[] args) {
18
19         System.out.println("Test the construction of hanoi with a given disks");
20         testHanoi();
21         System.out.println("Test the algorithm used");
22         testSolve();
23         System.out.println("Tests the representation of hanoi in an array of 2 dimensions");
24         testStatus();
25         System.out.println("Hanoi with number of disks less than 1 should raise a RuntimeException: ");
26         testWrongDisk();
27     }
28
29     /**
30      * Test la bonne construction de la tour de hanoi
31      */
32     private static void testHanoi() {
33         hanoi = new Hanoi(5);
34         System.out.println("Hanoi : ");
35         hanoiDisplay.display(hanoi);
36     }
37
38
39     /**
40      * Construction de la tour de hanoi avec un nombre de disques < 1
41      */
42     private static void testWrongDisk() {
43         try {
44             hanoi = new Hanoi(0);
45         } catch (RuntimeException e) {
46             System.out.println(e.getMessage());
47         }
48     }
49
50
51     /**
52      * Teste l'algorithme pour la résolution et la méthode finished()
53      */
54     private static void testSolve() {
55         hanoi = new Hanoi(5);
56         System.out.println("Solve : ");
57         hanoi.solve();
58         System.out.println("When solved the value of finished is true : " + hanoi.finished());
59     }
60
61
62     /**
63      * Teste la représentation en tableau de tableau de la pile
64      */
65     private static void testStatus() {
66         hanoi = new Hanoi(5);
67         System.out.println("Status : " + Arrays.deepToString(hanoi.status()));
68     }
69
70
71 }
72
```

## TestStack.java - Sarah Jallon Tegest Bogale

```
1 package test;
2
3 import util.*;
4
5 import java.util.Arrays;
6 import java.util.EmptyStackException;
7
8
9 /**
10  * Classe de test mettant en œuvre toutes les fonctionnalités des classes du package util,
11  * soit Stack, Element et Examiner
12  * @author Bogale Tegest & Jallon Sarah
13  */
14 public class TestStack {
15     private static final String passed = "Passed";
16     private static final String failed = "Failed";
17
18     public static void main(String[] args) {
19         System.out.println();
20         System.out.println("***** TEST push(Object o) & toString() *****");
21         testPush();
22         System.out.println();
23         System.out.println("***** TEST pop() *****");
24         testPop();
25         System.out.println();
26         System.out.println("***** TEST hasNext() *****");
27         testHasNext();
28         System.out.println();
29         System.out.println("***** TEST stackToArray() *****");
30         testStackToArray();
31         System.out.println("***** TEST next() *****");
32         testNext();
33     }
34 }
35
36 /**
37  * Permet de tester l'empilement des valeurs correctes, incorrectes, null ainsi que la représentation sous la
38  * forme de chaîne de caractère du contenu de la pile
39  */
40 private static void testPush() {
41     Stack s = creatTestStack(4);
42     String correct = "[ <1> <2> <3> <4> ]";
43     String wrongStringRep = "[ <1 <2> <3> <4> ]";
44     String inCorrect = "[ <2> <3> <4> ]";
45
46     Stack emptyStack = creatTestStack(0);
47     System.out.println("The representation of an empty stack is [ ] = " + emptyStack);
48     System.out.println("Wrong string representation should fail: "
49         + (testResults(s.toString(), wrongStringRep).equals(failed) ? passed : failed));
50     System.out.println("Creation of a stack with correct expected values: "
51         + testResults(s.toString(), correct));
52     System.out.println("Creation of a stack with incorrect expected values should not work: "
53         + (testResults(s.toString(), inCorrect).equals(failed) ? passed : failed));
54     System.out.println("Creation of a stack with correct values and correct string representation: "
55         + testResults(correct, s.toString()));
56
57     s.push(null);
58     System.out.println("Pushing \"null\" object to the stack is possible. Result: " + s);
59 }
60
61
62 /**
63  * Permet de tester le désempilement des valeurs correctes, incorrectes et null
64  */
65 private static void testPop() {
66     Stack s = creatTestStack(2);
67     s.pop();
68     String correct = "[ <2> ]";
69     String inCorrect = "[ <1> <2> ]";
70
71     System.out.println("Pop from a non empty stack works: " + testResults(s.toString(), correct));
72     System.out.println("The stack value after a pop is different from the original and is correct: "
73         + (testResults(s.toString(), inCorrect).equals(failed) ? passed : failed));
74     System.out.println("Pop on empty stack should throw EmptyStackException (null) : ");
75     try {
76         s.pop();
77         s.pop();
78     } catch (EmptyStackException e) {
79         System.out.println(e.getMessage());
80     }
81
82     Stack nulls = creatTestStack(0);
83     nulls.push(null);
84     System.out.println("We can pop a null value. Before: " + nulls);
85     nulls.pop();
86     System.out.println("After pop(): " + nulls);
87 }
88
89 }
```

```

90  /**
91   * Test le fonctionnement de hasNext()
92   */
93
94  private static void testHasNext() {
95      Stack s = creatTestStack(3);
96      Examiner e = s.examiner();
97      int cnt = 0;
98      while (e.hasNext()) {
99          e.next();
100         ++cnt;
101     }
102     System.out.println("hasNext iterates until the stack is empty: "
103         + testResult(cnt, s.size()));
104
105     Examiner e2 = s.examiner();
106     for (int i = 0; i < 10; ++i) {
107         if (e2.hasNext()) {
108             ++cnt;
109         }
110     }
111     System.out.println("The iterator doesn't overflow : "
112         + (testResult(cnt, s.size()).equals(failed) ? passed : failed));
113 }
114
115 /**
116 * Test le fonctionnement de hasNext()
117 */
118
119 private static void testNext() {
120     Stack s = creatTestStack(3);
121     Examiner e = s.examiner();
122
123     System.out.println("Printing stack values using next() works as expected: ");
124     for (int i = 0; i < s.size(); ++i) {
125         System.out.println(e.next());
126     }
127     System.out.println("If the iterator overflow an EmptyStackException is raised(null): ");
128     try {
129         for (int j = 0; j < s.size() + 3; ++j) {
130             System.out.println(e.next());
131         }
132     } catch (EmptyStackException ex) {
133         System.out.println(ex.getMessage());
134     }
135 }
136
137 }
138
139 /**
140 * Test le fonctionnement de stackToArray
141 */
142 private static void testStackToArray() {
143     Stack s = creatTestStack(4);
144     String correctArray = "[1, 2, 3, 4]";
145     String incorrectArray = "[1 2 3 4]";
146     System.out.println("Check with correct array: " +
147         testResult(Arrays.toString(s.stackToArray()), correctArray));
148     System.out.println("Check with incorrect array: " +
149         (testResult(Arrays.toString(s.stackToArray()), incorrectArray).equals(failed) ? passed : failed));
150
151     Stack empty = creatTestStack(0);
152     System.out.println("Representation of empty stack to array: " + Arrays.toString(empty.stackToArray()));
153 }
154
155 /**
156 * Permet de créer et remplir une pile
157 */
158 private static Stack creatTestStack(int size) {
159     Stack s = new Stack();
160     for (int i = size; i > 0; --i) {
161         s.push(i);
162     }
163     return s;
164 }
165
166 /**
167 * Permet de comparer la sortie avec le bon affichage
168 */
169 private static String testResult(String a, String b) {
170     if (a.equals(b)) return passed;
171     else return failed;
172 }
173
174 /**
175 * Permet de comparer si on a le bon nombre de résultats.
176 */
177 private static String testResult(int a, int b) {
178     if (a == b)

```

```
179         return passed;  
180     else return failed;  
181 }  
182  
183 }  
184
```

## Stack.java - Sarah Jallon Tegest Bogale

```
1 package util;
2
3 import java.util.EmptyStackException;
4
5 /**
6  * Classe qui implémente la pile
7  *
8  * @author Bogale Tegest & Jallon Sarah
9  */
10 public class Stack {
11     private Element top = null;
12     private int size;
13
14     /**
15      * Permet d'empiler un objet sur le sommet de la pile.
16      * Il est possible d'empiler un objet null
17      *
18      * @param o l'objet à empiler
19      */
20     public void push(Object o) {
21         top = new Element(o, top);
22         ++size;
23     }
24
25     /**
26      * Permet de dépiler un objet du sommet de la pile
27      *
28      * @return retourne l'objet dépiler
29      * @throws EmptyStackException si la pile est vide
30      */
31     public Object pop() {
32         if (top == null) {
33             throw new EmptyStackException();
34         }
35         Object o = top.data;
36         top = top.next;
37         --size;
38         return o;
39     }
40
41     /**
42      * Permet de redéfinir la méthode toString() afin d'obtenir la représentation sous la forme de
43      * chaîne de caractères du contenu de la pile,
44      *
45      * @return rend le contenu de la pile sous forme de chaîne de caractère
46      */
47     @Override
48     public String toString() {
49         Examiner ex = examiner();
50         StringBuilder sb = new StringBuilder();
51         sb.append("[ ");
52         while (ex.hasNext()) {
53             sb.append("<").append(ex.next()).append(">").append(" ");
54         }
55         sb.append("]");
56         return sb.toString();
57     }
58
59     /**
60      * Permet d'obtenir un tableau d'objets représentant
61      * l'état actuel de la pile (l'indice 0 contenant l'élément placé au
62      * sommet de la pile)
63      *
64      * @return le tableau d'objet représentant l'état actuel de la pile
65      */
66     public Object[] stackToArray() {
67         Object[] tab = new Object[size];
68         int i = 0;
69         Examiner ex = examiner();
70         while (ex.hasNext()) {
71             tab[i++] = ex.next();
72         }
73         return tab;
74     }
75
76     /**
77      * Permet de retourner un itérateur le sur le sommet de la pile
78      *
79      * @return un itérateur
80      */
81     public Examiner examiner() {
82         return new Examiner(top);
83     }
84
85     /**
86      * Permet de retourner la taille de la pile
87      *
88      * @return la taille de la pile
89      */
90 }
```

```
90     public int size() {
91         return size;
92     }
93
94     /**
95      *Permet de savoir si la stack est vide
96      * @return vrai si vide, faux sinon
97      */
98     public boolean isEmpty(){
99         return size() == 0;
100     }
101
102 }
```

```
1 package util;
2
3 /**
4  * Classe qui implémente les éléments à ajouter à la pile
5  *
6  * @author Bogale Tegest & Jallon Sarah
7  */
8 class Element {
9     Object data;
10     Element next;
11
12     /**
13      * Constructeur avec l'objet stocké et l'élément suivant
14      *
15      * @param data l'objet stocké dans l'élément
16      * @param next élément suivant l'actuel
17      */
18     Element(Object data, Element next) {
19         this.data = data;
20         this.next = next;
21     }
22 }
23
24
```



```
1 package util;
2
3 import java.util.EmptyStackException;
4
5 /**
6  * Classe qui implémente l'itérateur de la pile
7  *
8  * @author Bogale Tegest & Jallon Sarah
9  */
10 public class Examinator {
11     private Element current;
12
13     /**
14      * Constructeur, avec l'élément de départ de l'itérateur.
15      * À sa création, l'itérateur se réfère au début de la stack (avant le 1er élément).
16      *
17      * @param el élément sur lequel l'itérateur se réfère
18      */
19     Examinator(Element el) {
20         current = el;
21     }
22
23     /**
24      * Méthode qui rend l'élément suivant de l'itérateur
25      *
26      * @return retourne l'Objet stocké dans l'élément suivant de l'itérateur
27      * @throws EmptyStackException s'il n'y a plus d'élément à itérer
28      */
29     public Object next() {
30         if (!hasNext()) {
31             throw new EmptyStackException();
32         }
33         Object o = current.data;
34         current = current.next;
35         return o;
36     }
37
38     /**
39      * Méthode qui vérifie si l'élément sur lequel l'itérateur se réfère a un élément après
40      *
41      * @return rend true si l'itérateur n'est pas en fin de la pile, false sinon
42      */
43     public boolean hasNext() {
44         return current != null;
45     }
46 }
47
```

## Hanoi.java - Sarah Jallon Tegest Bogale

```
1 package hanoi;
2
3 import util.Stack;
4
5 /**
6  * Classe qui implémente les tours de hanoi et ses méthodes pour la résolution ainsi que l'affichage
7  *
8  * @author Bogale Tegest & Jallon Sarah
9  */
10 public class Hanoi {
11     private final Stack[] needles = new Stack[3];
12     private final HanoiDisplay hanoiDisplay;
13     private final int disks;
14     private int cntTurn = 0;
15
16     /**
17      * Constructeur pour l'affichage dans la GUI
18      *
19      * @param disks nombre de disques à déplacer
20      * @param hanoiDisplay instance de hanoi displayer pour l'affichage
21      * @throws RuntimeException si le nombre de disques est inférieur à 1 ou si hanoiDisplay est null
22      */
23     public Hanoi(int disks, HanoiDisplay hanoiDisplay) {
24         if (disks < 1) throw new RuntimeException("Error: number of disks should be greater than 0.");
25         if (hanoiDisplay == null) throw new RuntimeException("Error: hanoi displayer must be different from null.");
26         this.disks = disks;
27         this.hanoiDisplay = hanoiDisplay;
28
29         for (int i = 0; i < needles.length; ++i) {
30             needles[i] = new Stack();
31         }
32         for (int i = disks; i > 0; --i) {
33             needles[0].push(i);
34         }
35     }
36
37     /**
38      * Constructeur pour l'affichage dans la console
39      *
40      * @param disks nombre de disques à déplacer
41      */
42     public Hanoi(int disks) {
43         this(disks, new HanoiDisplay());
44     }
45
46     /**
47      * Fonction de transfert récursive qui permet de déplacer n disques d'un baton d'origine à un
48      * baton de destination en passant par un baton intermédiaire
49      *
50      * @param src baton d'origine
51      * @param by baton intermédiaire
52      * @param dest baton de destination
53      * @param n nombre de disques à déplacer
54      */
55     private void move(Stack src, Stack by, Stack dest, int n) {
56         if (n < 1)
57             return;
58         move(src, dest, by, n - 1);
59         dest.push(src.pop());
60         ++cntTurn;
61         hanoiDisplay.display(this);
62         move(by, src, dest, n - 1);
63     }
64
65     /**
66      * Algorithme de résolution du problème des tours de Hanoi, repris dans le cours
67      * de ASD2022.
68      */
69     public void solve() {
70         hanoiDisplay.display(this);
71         move(needles[0], needles[1], needles[2], disks);
72     }
73
74     /**
75      * Permet d'obtenir le statut courant des tours
76      *
77      * @return un tableau de tableaux représentant les 3 batons et leurs contenus
78      */
79     public int[][] status() {
80         Object[][] hanoiStatus = new Object[needles.length][];
81         int[][] t = new int[hanoiStatus.length][];
82         for (int i = 0; i < t.length; ++i) {
83             hanoiStatus[i] = needles[i].stackToArray();
84             t[i] = new int[hanoiStatus[i].length];
85             for (int j = 0; j < t[i].length; ++j) {
86                 t[i][j] = (int) hanoiStatus[i][j];
87             }
88         }
89     }
```

```
90         return t;
91     }
92
93     /**
94      * Permet de savoir la résolution du problème des tours de hanoi est terminée
95      *
96      * @return vrai si terminé, faux sinon
97      */
98     public boolean finished() {
99         return needles[0].isEmpty() && needles[1].isEmpty() && needles[2].size() == disks;
100     }
101
102
103     /**
104      * Permet de connaître le nombre d'étapes déjà réalisées dans la résolution du
105      * problème de Hanoi
106      *
107      * @return le nombre de l'étape courante
108      */
109     public int turn() {
110         return cntTurn;
111     }
112
113     /**
114      * Permet de redéfinir la méthode toString() afin d'obtenir la représentation sous la forme de
115      * chaîne de caractère des trois tours de Hanoi utilisant Stack.toString()
116      *
117      * @return rend le contenu des trois tours sous forme de chaîne de caractères
118      */
119     @Override
120     public String toString() {
121         return "One: " + needles[0] +
122             "\nTwo: " + needles[1] +
123             "\nThree: " + needles[2];
124     }
125 }
126 }
127
```

## HanoiDisplayer.java - Sarah Jallon Tegest Bogale

```
1 package hanoi;
2
3 /**
4  * Classe qui implémente l'affichage de hanoi sur la console
5  *
6  * @author Bogale Tegest & Jallon Sarah
7  */
8 public class HanoiDisplayer {
9
10     /**
11      * Permet d'afficher l'état courant du problème de Hanoi dans la console
12      *
13      * @param h instance de Hanoi que nous souhaitons afficher
14      */
15     public void display(Hanoi h) {
16         System.out.println("-- Turn: " + h.turn());
17         System.out.println(h);
18     }
19
20 }
21
```