```java
package hanoi;

import hanoi.gui.JHanoi;

/**
 * Represente le classique probleme des Tours de Hanoi.
 * <p>
 * Trois piles sont utilisees pour representer l'etat courant des tours de
 * Hanoi. Fournit une methode de resolutoon du problème, avec affichage
 * graphique ou console, au choix.
 *
 * @author Sacha Bron
 * @author Valentin Minder
 * @date 13.11.2014
 */
public class Hanoi {

    /**
     * Represente les trois piles initialisees vides.
     */
    private Pile[] pilesArray = { new Pile(), new Pile(), new Pile() };

    /**
     * Represente la taille maximale de la pile (definie par le nombre de
     * disques utilises).
     */
    private int size = 0;

    /**
     * Represente le nombre de coup joues jusque la (initialisation = 0).
     */
    private int turn = 0;

    /**
     * Constructeur (utilise pour resolution console et graphique).
     *
     * @param disks
     *            nombre de disks
```

```
40      * @throw {@link IllegalArgumentException} si le nombre de
  disque n'est pas
41      *        strictement positif
42      */
43     public Hanoi(int disks) {
44         if (disks <= 0) {
45             throw new IllegalArgumentException(
46                     "Positive number of disks required!");
47         }
48         size = disks;
49         // remplissage de la premiere pile.
50         for (int i = 0; i < disks; i++) {
51             pilesArray[0].empile(size - i);
52         }
53     }
54 
55     /**
56      * Resoud le probleme avec affichage de maniere "graphique".
57      * <p>
58      * Deplace tous les disques de la premiere aiguille à la
  troisieme en
59      * affichant les etats successifs des aiguilles dans la
  fenetre frame,
60      * instance de la classe hanoi.gui.JHanoi
61      *
62      * @param frame
63      */
64     public void solve(JHanoi frame) {
65         frame.display();
66         solveHanoi(size, 0, 2, 1, frame);
67         return;
68     }
69 
70     /**
71      * Resoud le probleme avec affichage de maniere "console".
72      * <p>
73      * Deplace tous les disques de la premiere aiguille à la
  troisieme en
74      * affichant les etats successifs des aiguilles dans la
  console.
75      */
76     public void solveConsoleOnly() {
77         System.out.println("--- turn " + turn + " ---");
78         System.out.println("One:    " + pilesArray[0]);
```

```
79          System.out.println("Two:   " + pilesArray[1]);
80          System.out.println("Three: " + pilesArray[2]);
81          solveHanoi(size, 0, 2, 1, null);
82      }
83
84      /**
85       * Methodes de resolution recursive de Hanoi (affichage
    console ou
86       * graphique).
87       * <p>
88       *
89       * Explication par pseudo-code <br>
90       * (n est le nombre de disque, D, A, I sont les piles
    respectivement de
91       * depart, d'arrivee et intermediaire
92       * <p>
93       * procedure Hanoi(n, D, A, I) <br>
94       * __si n != 0 <br>
95       * ____Hanoi(n-1, D, I, A) // (1)<br>
96       * ____Deplacer le disque de D vers A // (2) <br>
97       * ____Hanoi(n-1, I, A, D) // (3)<br>
98       * fin-procedure
99       * <p>
100      *
101      * Explication en francais: pour deplacer un pile de n disques
    de depart D a
102      * arrivee A, il faut <br>
103      * (1) deplacer la pile de n-1 disques (sans le plus grand
    disque) du depart
104      * D a l'intermediaire I, <br>
105      * (2) deplacer le plus grand cercle du depart D a l'arrivee
    A, <br>
106      * (3) finalement deplacer de n-1 disques (sans le plus grand
    disque) la
107      * pile de l'intermediaire I a l'arrive A.
108      *
109      * <p>
110      * Les index des piles doivent etre entre 0 et 2, et etre
    complets (chacune
111      * des piles doit etre representee, donc tous differents et la
    somme doit
112      * etre 0 + 1 + 2 = 3).
113      *
114      * @param numberOfDisks
```

```
115      *            nombre de disques a deplacer
116      * @param indexPileDepart
117      *            index de la pile de depart
118      * @param indexPileArrivee
119      *            index de la pile d'arrivee
120      * @param indexPileInterm
121      *            index de la pile utilisee comme intermediaire
122      * @param frame
123      *            JHanoi Frame (peut etre null pour affichage
    console
124      *            uniquement)
125      */
126     public void solveHanoi(int numberOfDisks, int indexPileDepart,
127             int indexPileArrivee, int indexPileInterm, JHanoi
    frame) {
128         // test des arguments...
129         if (numberOfDisks <= 0 || numberOfDisks > size) {
130             throw new IllegalArgumentException(
131                     "Number of disks must be: 0 < disks <= size of
    Hanoi towers");
132         }
133         if (indexPileDepart + indexPileArrivee + indexPileInterm !
    = 3) {
134             throw new IllegalArgumentException(
135                     "Piles idnex must represent the three piles:
    0, 1, 2, and their sum must be 3.");
136         }
137         if (indexPileDepart == indexPileArrivee
138                 || indexPileDepart == indexPileInterm
139                 || indexPileArrivee == indexPileInterm) {
140             throw new IllegalArgumentException(
141                     "Piles idnex must represent the three piles:
    0, 1, 2, and must be all different");
142         }
143         if (0 > indexPileDepart || 0 > indexPileArrivee || 0 >
    indexPileInterm
144                 || indexPileDepart > 2 || indexPileArrivee > 2
145                 || indexPileInterm > 2) {
146             throw new IllegalArgumentException(
147                     "Piles idnex must represent the three piles:
    0, 1, 2, and must be 0 <= index < 3");
148         }
149
150         // appel fonction privee
```

```
151         solveHanoiRecursive(numberOfDisks, indexPileDepart,
    indexPileArrivee,
152                 indexPileInterm, frame);
153     }
154 
155     /**
156      * Voir ci dessus @link{@link Hanoi#solveHanoi(int, int, int,
    int, JHanoi)}
157      */
158     private void solveHanoiRecursive(int numberOfDisks, int
    indexPileDepart,
159             int indexPileArrivee, int indexPileInterm, JHanoi
    frame) {
160         // condition d'arret de la recursion
161         if (numberOfDisks > 0) {
162 
163             // appel recursif
164             solveHanoiRecursive(numberOfDisks - 1,
    indexPileDepart,
165                     indexPileInterm, indexPileArrivee, frame);
166 
167             // deplacement courrant: UN objet de depart a arrivee
168             Pile pileDepart = pilesArray[indexPileDepart];
169             Pile pileArrivee = pilesArray[indexPileArrivee];
170             int k = (int) pileDepart.depile();
171             pileArrivee.empile(k);
172             turn++;
173             if (frame != null) {
174                 frame.display();
175             } else {
176                 System.out.println("--- turn " + turn + " ---");
177                 System.out.println("One:   " + pilesArray[0]);
178                 System.out.println("Two:   " + pilesArray[1]);
179                 System.out.println("Three: " + pilesArray[2]);
180             }
181 
182             // appel recursif
183             solveHanoiRecursive(numberOfDisks - 1,
    indexPileInterm,
184                     indexPileArrivee, indexPileDepart, frame);
185         }
186     }
187 
188     /**
```

```
189      * Rend un tableau de tableaux représentant l'etat des
    aiguilles. Pour un
190      * tel tableau t, l'element t[i][j] correspond a la taille du
    j-eme disque
191      * (en partant du haut) de la i-eme aiguille.
192      * <p>
193      * Attention, le tableau n'est pas de taille 3 x n, car seules
    les cases
194      * occupees sont representees! La 2e dimension n'est donc pas
    homogène!
195      *
196      * @return un tableau representant l'etat des piles
197      */
198     public int[][] status() {
199         int[][] statusIntIntArray = new int[3][];
200         Object[] statusPileIndexAsObjectArray;
201         for (int indexPile = 0; indexPile < 3; indexPile++) {
202             statusPileIndexAsObjectArray =
    pilesArray[indexPile].toArray();
203             statusIntIntArray[indexPile] = new
    int[statusPileIndexAsObjectArray.length];
204             for (int indexVertical = 0; indexVertical <
    statusPileIndexAsObjectArray.length; indexVertical++) {
205                 statusIntIntArray[indexPile][indexVertical] =
    (int) statusPileIndexAsObjectArray[indexVertical];
206             }
207         }
208         return statusIntIntArray;
209     }
210
211     /**
212      * Rend true si la solution du probleme a ete atteinte, false
    sinon.
213      * <p>
214      * Retourne true si les pile 0 et 1 sont vides (et donc la
    pile 2 contient
215      * tous les disques).
216      *
217      * @return true si termine, false sinon.
218      */
219     public boolean finished() {
220         return !pilesArray[0].iterateur().possedeSuivant()
221                 && !pilesArray[1].iterateur().possedeSuivant();
222     }
```

```
223
224     /**
225      * Rend le nombre de disques deja deplaces.
226      *
227      * @return nombre de disque deplaces
228      */
229     public int turn() {
230         return turn;
231     }
232
233 }
234
```