Quentin Forestier, Herzig Melvyn

POO1 – 25.11.2020

Tours de Hanoï

Laboratoire 7



Table des matières

[Algorithme utilisé 2](#_Toc56947125)

[Description des classes 2](#_Toc56947126)

[Stack 2](#_Toc56947127)

[Element 2](#_Toc56947128)

[Examinator 2](#_Toc56947129)

[Hanoi 2](#_Toc56947130)

[HanoiDisplayer 2](#_Toc56947131)

[Diagramme de classe 3](#_Toc56947132)

[Contrainte 3](#_Toc56947133)

[Tests de la pile 4](#_Toc56947134)

[Tests Hanoi 5](#_Toc56947135)

[Conclusion des tests 8](#_Toc56947136)

[Réponse 8](#_Toc56947137)

# Algorithme utilisé

Afin de résoudre le problème des tours de Hanoï, nous avons opté pour une solution itérative afin de ne pas être limité par la taille limitée de la pile de récursion.

Nous utilisons deux fonctions :

* Public void solve()
* Private void transfer(Stack s1, Stack s2)

La fonction transfer nécessite deux piles en paramètre. Elle cherche quelle pile a l’élément le plus petit pour le déplacer sur l’autre.

La fonction solve est l’élément central. En fonction du numéro du tour de résolution, elle détermine entre quelles piles nous devons déplacer un disque pour avancer la résolution. Il est inspiré de la solution proposée sur le site GeeksForGeeks[[1]](#footnote-1) et du cours d’ASD1. L’algorithme s’arrête dès que nous avons effectués le nombre maximal de déplacement pour N disques. Le nombre maximal est obtenu avec la formule 2^N-1.

# Description des classes

## Stack

Package : util Visibilité : publique

Implémente le concept de pile LIFO. Elle emplie des objets Element qui encapsule des objets Object.

## Element

Package : util Visibilité : package

Eléments d’une pile. Un ensemble d’Elements forment une liste simplement chaînée.

## Examinator

Package : util Visibilité : publique

Itérateur permettant de parcourir une pile de haut en bas.

## Hanoi

Package : hanoi Visibilité : publique

Classe modélisant le problème des tours de Hanoï avec un nombre de disques quelconque. Les aiguilles sont modélisées par trois Stack.

## HanoiDisplayer

Package : hanoi Visibilité : publique

Par défaut, permet de représenter un objet Hanoï dans la console.

# Diagramme de classe

Contrainte :

Un élément ne peut pas être suivi par lui même

# Tests de la pile

Pour tester l’implémentation de la classe Stack nous avons écrit TestStack.java. Les résultats ont été vérifiés manuellement et tous ont été validés. Pour afficher le tableau nous avons créé une fonction spécifique. Les tests sont en vert en cas de réussite ou rouge en cas d’échec.

|  |  |
| --- | --- |
|  | Doit afficher [ ] |
|  | Comme la pile est vide la taille vaut 0 et est isEmpty true |
|  | Top sur pile vide lance une RuntimeException |
|  | Pop sur pile vide lance une RuntimeException |
|  | Sur une pile vide, la création de l’examinator fonctionne mais son utilisation lance une NullPointerExcepton |
|  | Affiche { } |
|  | Affiche  [ <abc> <4> <a> ] |
|  | Le contenu du tableau est similaire à la pile. |
|  | La pile n’est plus vide et la taille vaut 3 |
|  | Le top retourne abc |
|  | Dépiler le top résulte en une pile  [ <4 <a> ] non vide de taille 2 |
|  | Le reste de la pile est parcouru avec l’examinator jusqu’à la fin |

# Tests Hanoi

Afin de vérifier le fonctionnement de notre résolveur du problème de Hanoï, nous avons écrit TestHanoi.java. Le programme prend deux arguments :

* Le mode d’affichage : « graphique » ou « console ».
* Si le mode est « console » le programme demande en deuxième argument le nombre de disque.

**Test 1 : Réussite**

Arguments utilisés : « console -6 » et « graphique »

Nous avons vérifié si le programme acceptait un nombre de disque négatif. Le test est réussi si le programme retourne une erreur qui indique qu’un nombre de disque négatif est interdit.

Résultats (similaire dans les deux cas) :



**Test 2 : Réussite**

Arguments utilisés : « graphique »

Le but de ce test est de vérifier si le programme résolvait correctement le problème avec un nombre de disque impair, dans ce cas 3. Le test est réussi si tous les mouvements sont légaux et le problème résolu en 7 tours en mode graphique.

Résultat :

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

**Test 3 : Réussite**

Arguments utilisés : « graphique »

Le but de ce test est de vérifier si le programme résolvait correctement le problème avec un nombre de disque pair, dans ce cas 4. Le test est réussi si tous les mouvements sont légaux et le problème résolu en 15 tours en mode graphique.

Résultat :

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | [ . . . ] |  |
|  |  |  |

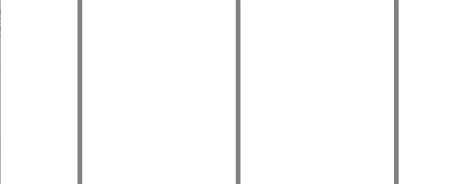
Nous avons retiré 8 tours par souci de place. Le test a été réalisé en 15 tours.

**Test 4 : Réussite**

Arguments utilisés : « graphique »

Le but du test est de vérifier le comportement du programme avec 0 disque. Le test est réussi si le programme affiche les trois aiguilles vide une unique fois.

Résultat :



**Test 5 : Réussite**

Arguments utilisés : « console 3 »

Le but de ce test est de vérifier si le programme résolvait correctement le problème avec un nombre de disque impair, dans ce cas 3. Le test est réussi si tous les mouvements sont légaux, le problème résolu en 7 tours et l’affichage console correcte.

Résultat :

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

**Test 6 : Réussite**

Arguments utilisés : « console 4 »

Le but de ce test est de vérifier si le programme résolvait correctement le problème avec un nombre de disque pair, dans ce cas 4. Le test est réussi si tous les mouvements sont légaux, le problème résolu en 15 tours et l’affichage console correcte.

Résultat :

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | [ . . . ] |  |
|  |  |  |

Nous avons retiré 8 tours par souci de place. Le test a été réalisé en 15 tours.

# Conclusion des tests

Les tests sont réussis. La pile fonctionne comme prévu, ce qui a permis de tester le résolveur des tours de Hanoï. Ce dernier fonctionne aussi bien en mode console qu’en mode graphique. Le problème ne peut être résolu avec un nombre de disque négatif.

# Réponse

*En supposant des moines surentraînés capables de déplacer un disque à la seconde, combien de temps reste-t-il avant que l’univers disparaisse (celui-ci a actuellement 13.7 milliards d’années)*

Il y a 64 disques à déplacer de la première aiguille à la troisième :

Soit nbDepl = #de déplacements nécessaires pour résoudre le problème.  
 nbDepl = 2^64-1 = 18 446 744 073 709 551 616

Soit secParAn = # nombre de secondes dans une année.  
 secParAn = 3600 \* 24 \* 365 = 31 536 000

Le nombre de déplacement total représente :

nbDepl / secParAn = 584 942 417 355.072032 ans  
 = 584.94 milliards d’années

Ainsi, l’univers vivra encore pendant (584.94 – 13.7 = ) 571.24 milliards d’années

1. <https://www.geeksforgeeks.org/iterative-tower-of-hanoi/> [↑](#footnote-ref-1)