HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Tours de Hanoï

LABORATOIRE 7

Quentin Forestier, Herzig Melvyn

POO1 – 25.11.2020

Table des matières

Algorithme utilisé

Afin de résoudre le problème des tours de Hanoï, nous avons opté pour une solution itérative afin de ne pas être limités 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] et du cours d'ASD1. L'algorithme s'arrête dès que nous avons effectué le nombre maximal de déplacements pour N disques. Le nombre maximal est obtenu avec la formule 2^N-1.

Description des classes

Stack

Package : util Visibilité : public

Implémente le concept de pile LIFO. Elle empile des objets Element qui encapsulent des objets Object.

Element

Package : util Visibilité : package

Éléments d'une pile. Un ensemble d'Elements forment une liste simplement chaînée.

Examinator

Package : util Visibilité : public

Itérateur permettant de parcourir une pile de haut en bas.

Hanoi

Package : hanoi Visibilité : public

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

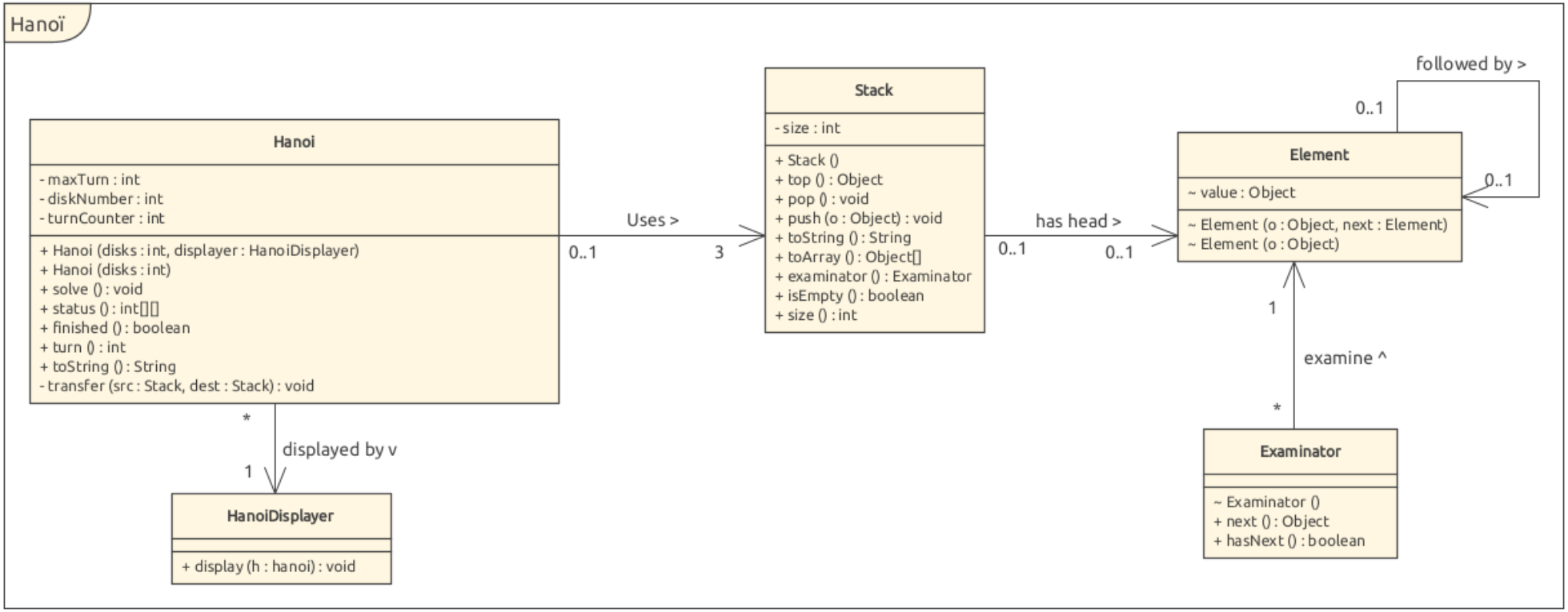
HanoiDisplayer

Package : hanoi Visibilité : public

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

[1] https://www.geeksforgeeks.org/iterative-tower-of-hanoi/

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.

------ Pile vide création et affichage string [ ]	Doit afficher [ ]
------ Pile vide size et isEmpty Pile vide : true Taille : 0	Comme la pile est vide la taille vaut 0 et est isEmpty true
------ Pile vide top Exception reçue top on empty stack	Top sur pile vide lance une RuntimeException
------ Pile vide pop Exception reçue pop on empty stack	Pop sur pile vide lance une RuntimeException
------ Pile vide Examinator: Création de l'examinator avec succès NullPointerException reçue lors de next()	Sur une pile vide, la création de l'examinator fonctionne, mais son utilisation lance une NullPointerExcepton
------ Pile vide toArray: { }	Affiche { }
------ Pile vide push 'a' 4 "abc" : [<abc> <4> <a>]	Affiche [<abc> <4> <a>]
------ Pile remplie toArray: Array { abc 4 a } Etat de la pile [<abc> <4> <a>]	Le contenu du tableau est similaire à la pile.
------ Pile remplie size et isEmpty Pile vide : false Taille : 3	La pile n'est plus vide et la taille vaut 3
------ Pile remplie top Top: abc	Le top retourne abc
------ Pile remplie pop [<4> <a>] Pile vide : false Taille : 2	Dépiler le top résulte en une pile [<4> <a>] non vide de taille 2

------ Pile remplie Examinator: Création de l'examinator2 avec succès hasNext: true | next : 4 hasNext: true | next : a hasNext: false	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 disques.

Test 1 : Réussite

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

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

Résultats (similaire dans les deux cas) :

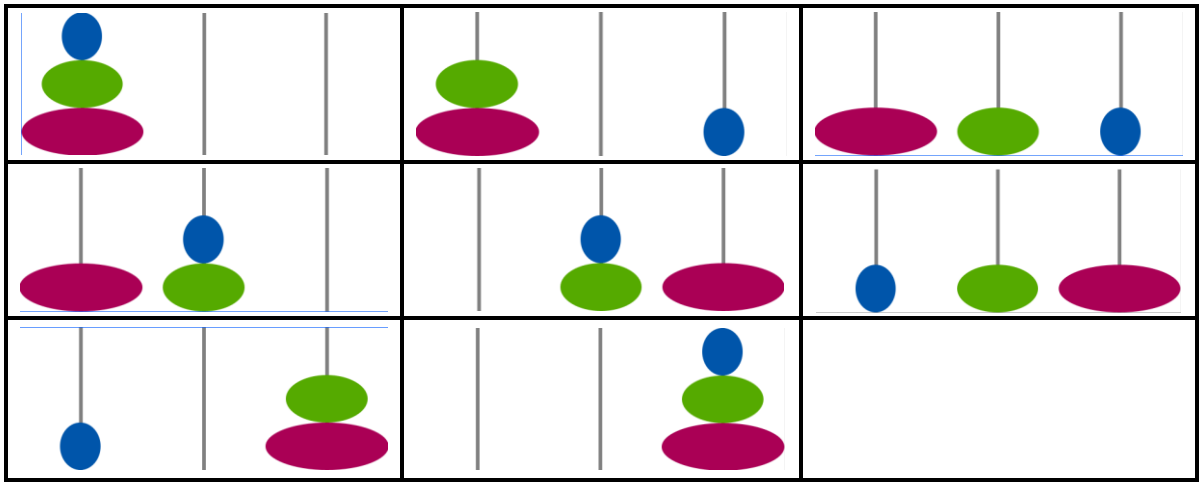
```
Exception in thread "AWT-EventQueue-0" java.lang.RuntimeException Create breakpoint : Nombre de disque inférieur à 0 impossible.
    at hanoi.Hanoi.<init>(Hanoi.java:37)
```

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 disques 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 :

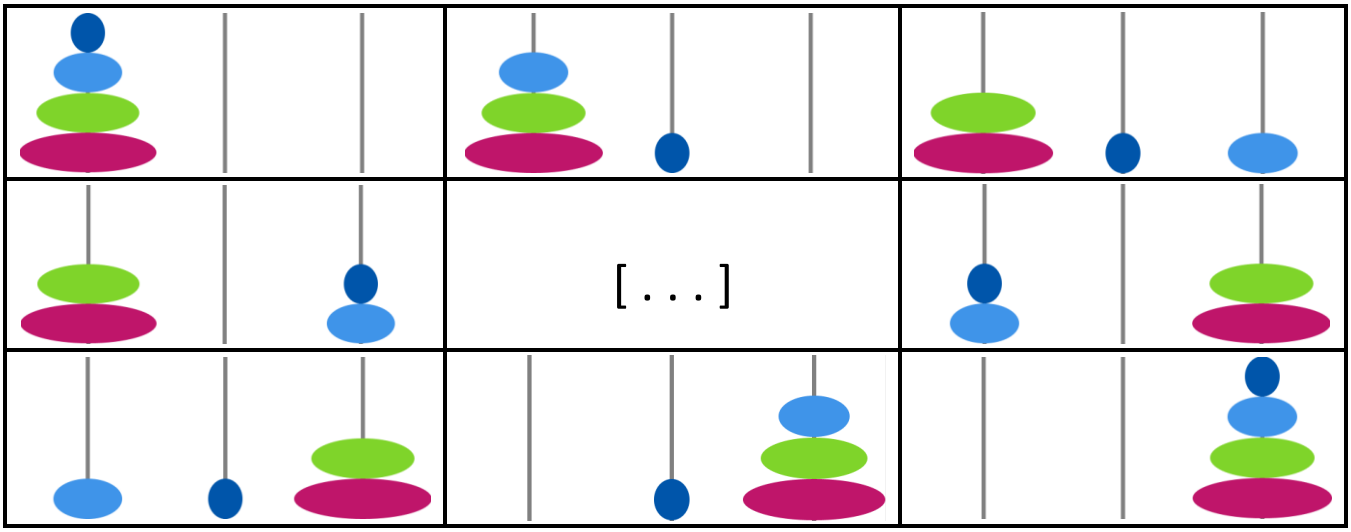


Test 3 : Réussite

Arguments utilisés : « graphique »

Le but de ce test est de vérifier si le programme résolvait correctement le problème avec un nombre de disques 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 vides 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 disques 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 :

-- Turn : 0 One : [<1> <2> <3>] Two : [ ] Three: [ ]	-- Turn : 1 One : [<2> <3>] Two : [ ] Three: [<1>]	-- Turn : 2 One : [<3>] Two : [<2>] Three: [<1>]
-- Turn : 3 One : [<3>] Two : [<1> <2>] Three: [ ]	-- Turn : 4 One : [ ] Two : [<1> <2>] Three: [<3>]	-- Turn : 5 One : [<1>] Two : [<2>] Three: [<3>]
-- Turn : 6 One : [<1>] Two : [ ] Three: [<2> <3>]	-- Turn : 7 One : [ ] Two : [ ] Three: [<1> <2> <3>]	

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 disques 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 :

-- Turn : 0 One : [<1> <2> <3> <4>] Two : [ ] Three: [ ]	-- Turn : 1 One : [<2> <3> <4>] Two : [<1>] Three: [ ]	-- Turn : 2 One : [<3> <4>] Two : [<1>] Three: [<2>]
-- Turn : 3 One : [<3> <4>] Two : [ ] Three: [<1> <2>]	[. . .]	-- Turn : 12 One : [<1> <2>] Two : [ ] Three: [<3> <4>]
-- Turn : 13 One : [<2>] Two : [<1>] Three: [<3> <4>]	-- Turn : 14 One : [ ] Two : [<1>] Three: [<2> <3> <4>]	-- Turn : 15 One : [ ] Two : [ ] Three: [<1> <2> <3> <4>]

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 déplacements 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