Labo 7 Hanoi

Sarah Jallon & Tegest Bogale

Professeur : Pier Donini
Assistant : Gregoire Decorvet
30.11.2022

Introduction

Le but de ce laboratoire est d'implémenter l'algorithme de tours de hanoi en utilisant une pile et de faire fonctionner le programme avec une interface graphique ainsi qu'en ligne de commande. Ce rapport permet dans un premier temps de répondre à la question posée et ensuite d'expliquer les différents choix d'implémentations des différentes classes, l'hypothèse de travail ainsi que l'algorithme utilisé.

Légende hindoue

Dans le grand temple de Brahmâ à Bénarès, sous la coupole qui marque le centre du monde, repose un socle de cuivre équipé de trois aiguilles de diamant. Au commencement du monde 64 disques d'or étaient enfilés sur une aiguille, les plus grands en bas et les plus petits en haut, et formaient la Tour de Brahmâ. Les moines les déplacent, un à la fois, d'une aiguille à l'autre en suivant l'immuable loi de Brahmâ: aucun disque ne peut être posé sur un autre disque plus petit. Quand la Tour de Brahmâ sera finalement reconstruite sur une aiguille différente de celle d'origine le monde tombera en poussière et disparaîtra.

Question

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 13.8 milliards d'années) ?

Réponse

Nous savons que l'algorithme de Hanoi pour n disques opère 2^n -1 opérations (donc déplacements). Si nous avons 64 disques, la résolution des tours de Hanoi demande alors 18 446 744 073 709 551 615 déplacements. Les moines auraient alors besoin d'autant de secondes pour le résoudre. Si nous divisons ce nombre par le nombre de secondes dans une année (31 536 000) nous obtenons alors le nombre d'années nécessaires aux moines pour déplacer les disques qui est de un peu plus de 584 942 417 355 années.
Sachant que notre univers a 13 800 000 000 ans, nous allons les soustraire à notre total.
Il reste donc 571 142 417 355 années à notre univers, ce qui correspond à plus de 571 milliards d'années. Nous comprenons que les moines en ont encore pour un bon bout de temps.

Description de l'algorithme utilisé

Nous avons utilisé l'algorithme récursif vu lors du cours ASD.

pseudo-code:

```
Si n > 0, ALORS:
   Transférer n-1 disques de O vers I via D
   transférer le disque estant de O vers D
   Transférer n-1 disques de I vers D via O
Fin si
```

Tant que n est supérieur à 0, on effectue l'appel récursif sur la fonction de transfert. On déplace de l'aiguille d'origine(O) vers l'intermédiaire (I) via celui de destination (D). Puis, on déplace le disque restant de O vers D. Pour finir, de I vers D via O.

Description des classes

Classe Hanoi

Nous avons choisi de représenter les trois tours de hanoi par un tableau de 3 piles.
La classe Hanoi permet d'instancier un problème des tours de Hanoi à partir de 1 disque (nous considérons que 0 disque n'est pas pertinent) et met à disposition une fonction de résolution du problème, utilisant un algorithme récursif de transfert. Sa visibilité est package afin d'interagir avec HanoiDisplayer.

Classe Stack

Une instance de la classe "Stack" référence l'élément qui se trouve au sommet. En se basant sur l'implémentation des collections java par ex. LinkedList, nous avons autorisé l'empilement de "null".
Par ailleurs, il est possible d'instancier une pile sans qu'elle soit référencée par une instance de Hanoi d'où le choix de cardinalité 0..1.

Classe Examinator

Nous avons choisi de mettre la classe "Examinator" ainsi que ses méthodes "next" et "hasNext" publiques afin de garantir leur utilisation en dehors de la classe "Stack". Cependant, le constructeur a une visibilité package car son instanciation ne doit être possible que par la "Stack".

Il faut constater qu'à la création, l'itérateur se réfère au début de la stack avant le premier élément.

Classe Element

La classe "Element" est interne à la classe "Stack". Les utilisateurs n'ont pas besoin d'y avoir accès depuis l'extérieur. Ainsi elle a une visibilité package tout comme ses attributs et méthodes.

Classe HanoiDisplayer

Elle permet d'afficher l'état de la classe Hanoi en console. Elle utilise la fonction toString() fournie par la classe Hanoi. Elle a une visibilité publique et peut ainsi être utilisée à l'extérieur.

Tests

Stack

testPush

En créant et empilant des éléments sur la pile, nous testons le bon fonctionnement de la méthode "push(Object o)" . Par la même occasion, nous testons la représentation sous forme de chaîne de caractères du contenu de la pile implémentée par la redéfinition de la méthode toString().

Test	**Résultat attendu**	**Résultat obtenu**
Affichage d'une pile vide	[ ]	[ ]
Affichage d'une pile contenant les valeurs de [1,4]	[<1> <2> <3> <4>]	[<1> <2> <3> <4>]
Une mauvaise représentation de chaîne devrait échouer. Exemple [<1 <2> <3> <4>]	réussi	réussi
Création d'une pile avec des valeurs attendues correctes	réussi	réussi
La création d'une pile avec des valeurs attendues incorrectes ne devrait pas fonctionner	réussi	réussi

Empiler un objet "null" sur la pile est possible.	[<null> <1> <2> <3> <4>]	[<null> <1> <2> <3> <4>]

testPop

Test	**Résultat attendu**	**Résultat obtenu**
Désempiler un élément de la pile	réussi	réussi
Désempiler un élément d'une pile vide	EmptyStackException : null	EmptyStackException: null
La valeur de la pile après un pop est différente de l'originale	réussi	réussi
Il est possible de désempiler un objet "null". Pile de départ : [<null>]	[ ]	[ ]

testHasNext

Test	**Résultat attendu**	**Résultat obtenu**
itération jusqu'à ce que la pile soit vide	réussi	réussi
L'itérateur ne déborde pas	réussi	réussi

testNext

Test	**Résultat attendu**	**Résultat obtenu**
Imprimer les valeurs de la stack contenant les valeur de [1,3]	1 2 3	1 2 3
Si l'itérateur déborde une exception est levée	EmptyStackException : null	EmptyStackException : null

testStackToArray

Test	Résultat attendu	Résultat obtenu
Vérification avec un résultat correct	réussi	réussi
Vérification avec un résultat erroné	réussi	réussi
Vérification avec une stack vide	[]	[]

Hanoi

tests construction

Test	Résultat attendu	Résultat obtenu
Construction d'un hanoi	-- Turn: 0 One: [<1> <2> <3> <4>] Two: [ ] Three: [ ]	-- Turn: 0 One: [<1> <2> <3> <4>] Two: [ ] Three: [ ]
Construction avec disks <1	runtime Exception	runtime Exception

test solve() et finished()

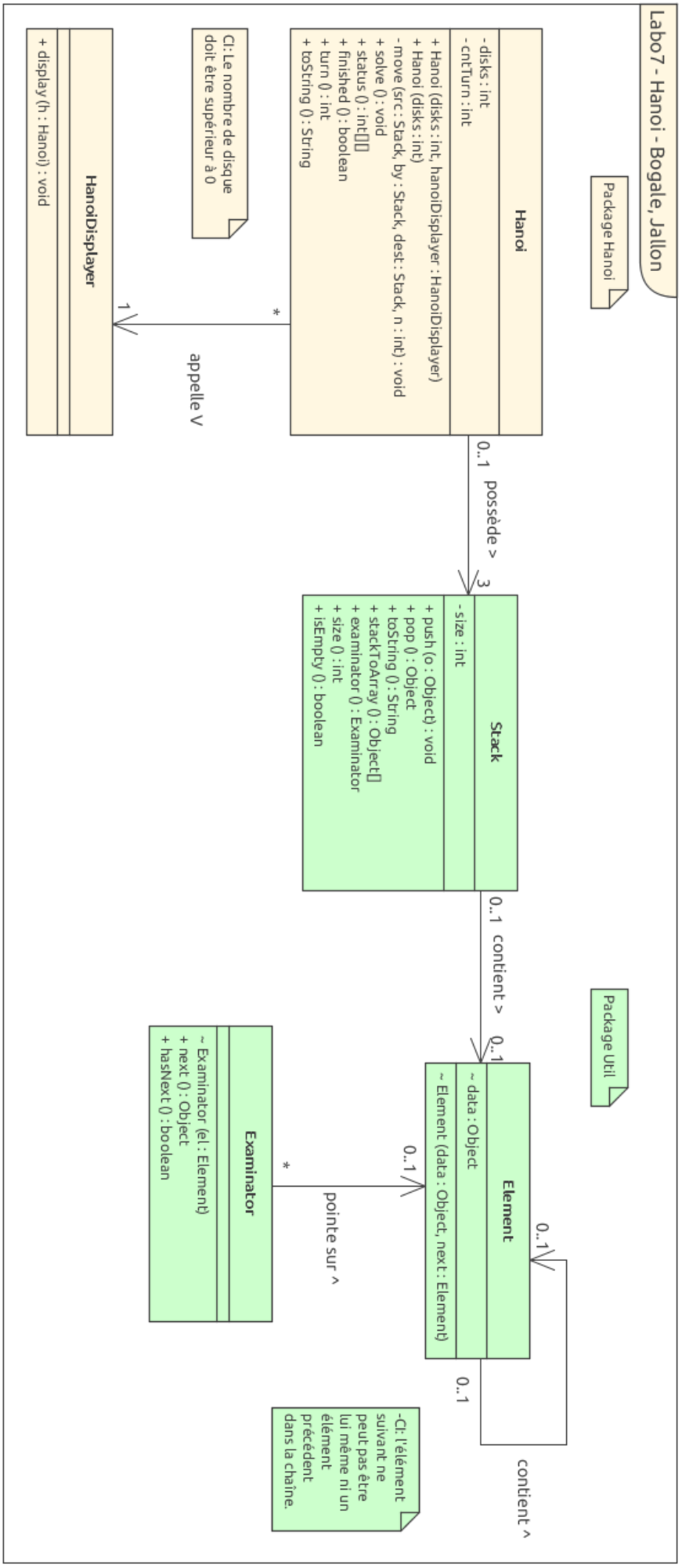
Test	Résultat attendu	Résultat obtenu
imprimer la solution	solution affichée	solution affichée
Vérification que la fonction finished fonctionne	true	true

test status()

Test	Résultat attendu	Résultat obtenu
imprimer sous la forme tableau	[[1, 2, 3, 4, 5], [], []]	[[1, 2, 3, 4, 5], [], []]

Conclusion

Nous avons trouvé intéressant de travailler avec une interface, et de voir notre travail prendre forme lorsque notre algorithme était réalisé. Notre plus grande difficulté a été la compréhension globale de l'interaction entre les différentes classes et le fonctionnement des classes du package util. Une fois cela compris, l'implémentation nous a semblé fluide et en lien avec le cours théorique.


Labo7 - Hanoi - Bogale, Jallon
Package Hanoi
Hanoi
- disks : int
- cntTurn : int
+ Hanoi (disks : int, hanoiDisplayer : HanoiDisplayer)
+ Hanoi (disks : int)
- move (src : Stack, by : Stack, dest : Stack, n : int) : void
+ solve () : void
+ status () : int[][]
+ finished () : boolean
+ turn () : int
+ toString () : String
CI: Le nombre de disque doit être supérieur à 0
HanoiDisplayer
+ display (h : Hanoi) : void
1
*
appelle V
0..1
possède >
3
Stack
- size : int
+ push (o : Object) : void
+ pop () : Object
+ toString () : String
+ stackToArray () : Object[]
+ examinator () : Examinator
+ size () : int
+ isEmpty () : boolean
0..1
contient >
0..1
Package Util
Element
~ data : Object
~ Element (data : Object, next : Element)
0..1
contient ^
0..1
-CI: l'élément suivant ne peut pas être lui même ni un élément précédent dans la chaîne.
Examinator
~ Examinator (el : Element)
+ next () : Object
+ hasNext () : boolean
*
pointe sur ^
0..1