HEIG-VD POO

Rapport laboratoire n°7

Tours de Hanoï

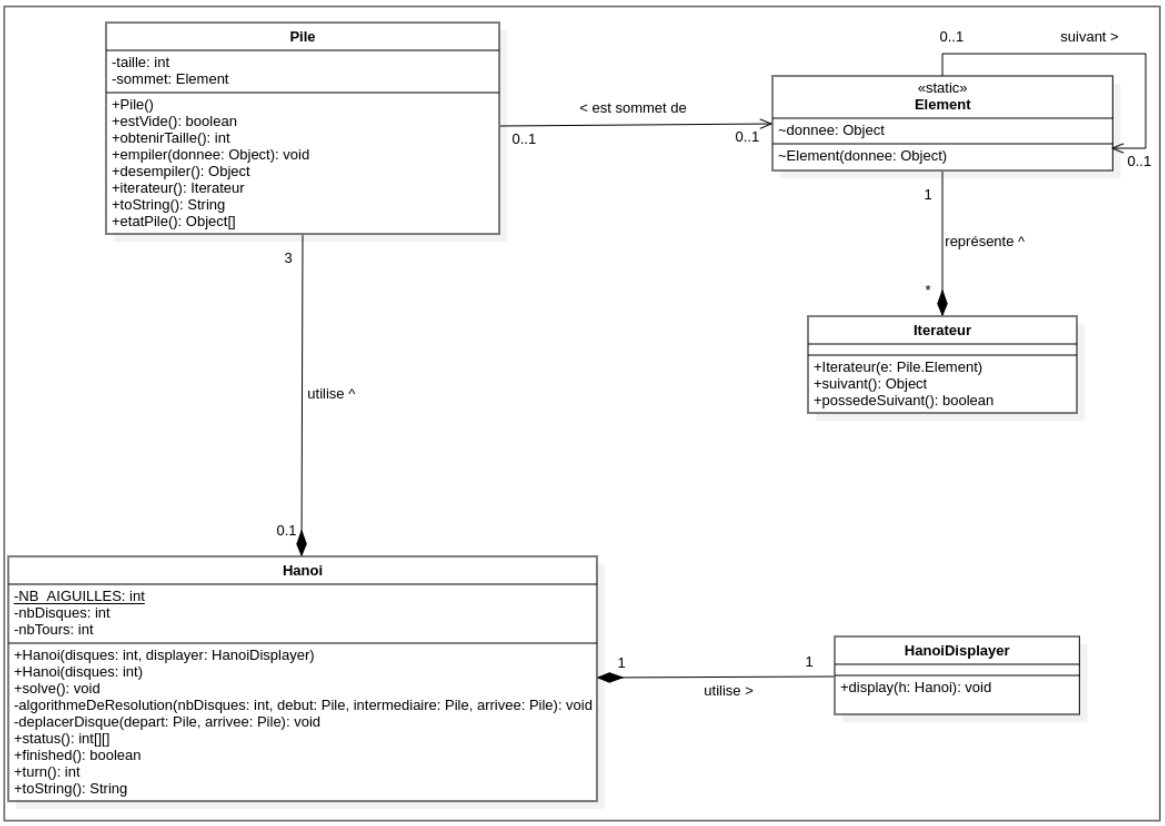
DECOPPET Joris, DUCOMMUN Hugo, MARTINS

12-9-2022

1 Introduction

Ce rapport concerne le laboratoire n°7 du cours de POO dans lequel nous avons dû implémenter le célèbre jeu des tours d'Hanoï. Le travail est séparé en deux parties majeurs. La première étant la création d'une structure de données de type « Pile » pouvant stocker des objets de types variés. La seconde partie consiste à ré-utiliser cette pile pour l'implémentation et la résolution de jeu. Il était aussi possible d'afficher le jeu de deux façons différentes, une étant via la console et la seconde via une interface graphique. L'utilisateur quant à lui pouvait choisir la taille de la tour.

2 Diagramme des classes



3 Description des classes

3.1 Structure de données

3.1.1 Elément

Cette classe a pour but de représenter un élément qui est utilisé dans la pile. Cet élément est composé d'une référence vers l'élément le suivant dans la structure, ainsi que d'une donnée pouvant être de type quelconque.

Cette classe a la particularité d'être en visibilité « Package » étant donné qu'elle n'a pas besoin d'être accédée par d'autres classes en dehors du package « Util ». Seules les entités liées à la pile ont besoin de pouvoir accéder à des éléments.

Cette classe est aussi une classe statique interne à la classe Pile. Cela améliore grandement la structure du code étant donné qu'un élément est utile uniquement dans le cas de la pile.

3.1.2 Itérateur

Un itérateur permet de référencé un élément. Il est notamment utile pour parcourir les éléments de la pile. Cet itérateur respecte les normes Java comme nous les avons vus en cours. Notamment pour la méthode « suivant » qui retourne la valeur de l'élément courant et passe au prochain élément.

Comme pour les éléments, cette classe est en visibilité « Package » pour les mêmes raisons.

3.1.3 Pile

Cette classe réunit les fonctionnalités des deux dernièrement présentées. Elle représente la structure de données en tant que telle. Elle permet d'avoir une pile fonctionnelle, de l'afficher et de la parcourir.

3.2 Jeu des tours d'Hanoï

3.2.1 Hanoï

Dans cette classe, on s'occupe du jeu lui-même. C'est-à-dire que nous avons nos trois aiguilles qui sont présentées sous la forme d'un tableau de trois piles. La classe implémente aussi l'algorithme de résolution du jeu qui sera détaillé dans la section suivante. L'affichage du jeu quant à lui se passe dans une classe séparée.

3.2.2 HanoiDisplayer

Cette dernière classe nous permet de faire le lien entre le jeu et son affichage par l'utilisateur. Elle permet d'appeler les méthodes nécessaires présentent dans Hanoi pour afficher le jeu. La version graphique du jeu quant à elle est gérée par une libraire externe (JHanoi) qui nous était fournie par les enseignants. Nous n'avons fait que de l'utiliser en lui fournissant le jeu à afficher et sa résolution.

4 Description de l'algorithme

Pour la réalisation de l'algorithme de résolution du jeu, nous avons décidé de partir sur sa version récursive. Nous étions déjà familiers avec celui-ci grâce au cours d'ASD et il est surtout bien plus court et facile à implémenter. Nommons les aiguilles de 1 à 3, respectivement de gauche à droite. Et utilisons un jeu composé de « N » disques où N est un entier strictement positif.

- Déplacer N-1 disques de la première tour à la seconde. Ce mécanisme se réalise grâce des appels récursifs.
- Il ne reste alors que le plus grand des disques non-placé sur la première aiguille. On déplace celui-ci sur la troisième aiguille.

- On déplace les N-1 premiers disques de la seconde à la troisième aiguille.

Cela nous demande alors de réaliser $2^N - 1$ déplacements pour résoudre ce problème.

5 Liste des tests réalisés

Tous les tests cités ci-dessous ont été concluants. Le résultat attendu était à chaque fois le résultat obtenu.

Description du test	Résultat attendu
Itérateur	
Itérer sur des éléments d'une pile	Les éléments sont itérés dans l'ordre et dans leur totalité
Itérer sur une pile vide provoque une exception	Exception levée
Un itérateur possédant un élément suivant indique que c'est le cas	L'itérateur indique bien qu'un élément suivant est présent
Pile	
Une pile vide est indiquée vide	La pile est vide, alors « vrai » est retourné
Taille de la pile annoncée correctement	La pile a la taille espérée
Ajouter des éléments sur la pile et les désempiler pour vérifier qu'ils sortent dans le bon ordre.	Les éléments empilés dans l'ordre A -> B -> C ressortent dans l'ordre C -> B -> A
Désempiler une pile vide	Une exception est levée
Affichage de la pile	Une pile remplie avec les éléments A -> B -> C s'affiche de la manière [<C> <B> <A>]
L'état de la pile est correctement réalisé dans un tableau	Une pile remplie avec les éléments A -> B -> C nous donne le tableau suivant [0] -> C, [1] -> B, [2] -> A
Hanoi	
Résolution du jeu correcte	A la fin du jeu les disques sont dans le bon ordre sur la dernière aiguille
Jeu terminé	Lorsque le jeu est terminé, la méthode corresponde nous indique que c'est le cas
Jeu en cours	Tant que le jeu n'est pas fini, la méthode nous indique qu'il est en cours
Affichage du statut du jeu	Le jeu nous est donné sous le format d'un tableau bidimensionnel-

6 Réponse à la question

Comme expliqué lors de la description de notre algorithme. Le nombre de mouvements nécessaires pour compléter un problème de N disques est de $2^N - 1$. Dans le cas où les moines doivent déplacer 64 disques, on a alors $2^{64} - 1$ mouvements et étant donné qu'ils font un mouvement par seconde, on a alors ce même nombre de secondes.

La conversion de ce très grand nombre en milliards d'années nous donne alors précisément 584.9 milliards d'années. Si on y soustrait l'âge de l'univers, on obtient alors **571.2 milliards d'années** restants avant la fin de l'univers. Tous ces calculs ont été réalisés grâce à WolframAlpha.