POO - Labo 7 - Les tours de Hanoi

Par Adrien Allemand et James Smith

POO - Labo 7 - Les tours de Hanoi
- Question 1
- Algorithme de Hanoi
- Diagramme des classes
- Description des classes
- Sources du programme

Question 1

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) ?

Tel que décrit, la situation initiale où tout les disks se trouvent sur la première tour dans l'ordre croissant et doivent finir sur la dernière tour dans l'ordre croissant, corresponds au pire cas de l'algorithme de résolution du problème. Ainsi la formule pour calculer le nombre de déplacements nécessaires est donnée par :

$nombre\ de\ coups = 2^{nombre\_de\_disks} - 1$

Pour 64 disks, le temps basé sur le nombre de déplacement sera donc :

$2^{64} - 1 = 18'446'744'073'709'551'616\ secondes$

pour avoir un résultat plus représentatif passons le en milliard d'années en le divisant par :

$60 * 60 * 24 * 365,2421875 * 1000000000 = 3,1556925e16$

(nous nous basons sur la durée d'une année selon [Johann Heinrich von Mädler])

Ainsi nous obtenons que l'univers va disparaitre au bout de (arrondit à la 6ième décimalle soit au millier d'années)

$au\ bout\ de\ 584.554549\ milliards\ d'ann\acute{e}es$

Etant donné que celui-ci a actuellement $13,7\ milliards\ d'ann\acute{e}es$ il nous reste donc $584.5 - 13.7 = 570.8\ milliards\ d'ann\acute{e}es$ avant que l'univers ne disparaisse.

Algorithme de Hanoi

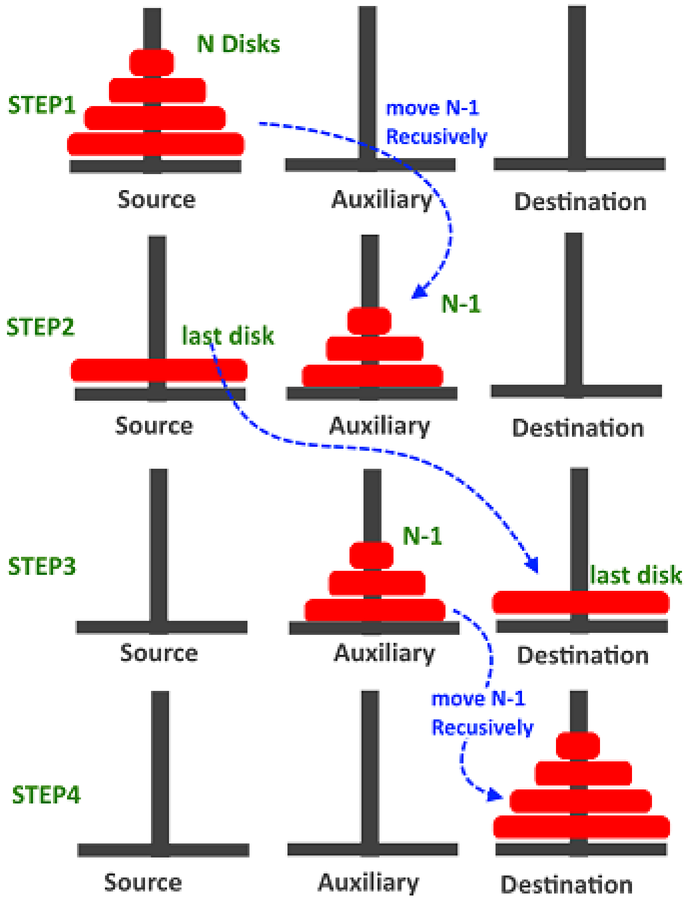
```
private void transfert(Pile from, Pile via, Pile to, int n){
  if(n > 0){
    transfert(from,to,via,n-1);
    to.stack(from.unstack());
    cmpt++;                  // incrémentation du nombre de tour
    displayer.display(this); // Affichage de status
    transfert(via,from,to,n-1);
  }
}
```

Pour résoudre le jeu d'Hanoi, nous avons décidé de prendre une solution récursive que nous avions déjà étudié en ASD1.

- Pile from - est la Pile de départ
- Pile via - est la Pile qui servira d'intérmédiaire
- Pile to - est la Pile ou les disques finiront.

L'algorithme consiste à déplacé tous les disques sauf le plus gros sur la tour intérmédiaire, puis déplacer le plus gros sur la pile de destination et redéplacer les disques sur la tour de destination.

Pour mieux visualisé voilà une représentation graphique:



L'algorithme utilisé est récursif et entièrement basé sur la fonction `transfer()`

```
// Appel initial (From pile A, via pile B, to pile c, nombre de disques)
private void transfert(Pile from, Pile via, Pile to, int n){
    // jusqu'à ce que le dernier étage (n = 1) ait été transféré
    if(n > 0){
        // Appel récursif pour transférer toute la tour sauf le dernier disque sur la pile
        // intermédiaire.
        transfert(from,to,via,n-1);
        // transfer du plus petit disque de la pile from à la pile to
        to.stack(from.unstack());
        // Appel récursif pour reconstruire la tour depuis la pile intermédiaire à la pile
        // de destination.
        transfert(via,from,to,n-1);
    }
}
```

La formule utilisée pour calculer le nombre de déplacements requis se démontres facilement par réccurence :

$n = nombre\ de\ disques$

$2^n - 1 = nombre\ de\ mouvements\ n\acute{e}cessaires$

Le résultat est vrait pour $n = 1$ car il faut bien un seul déplacement pour déplacer 1 disque de la première à la troisième aiguille.

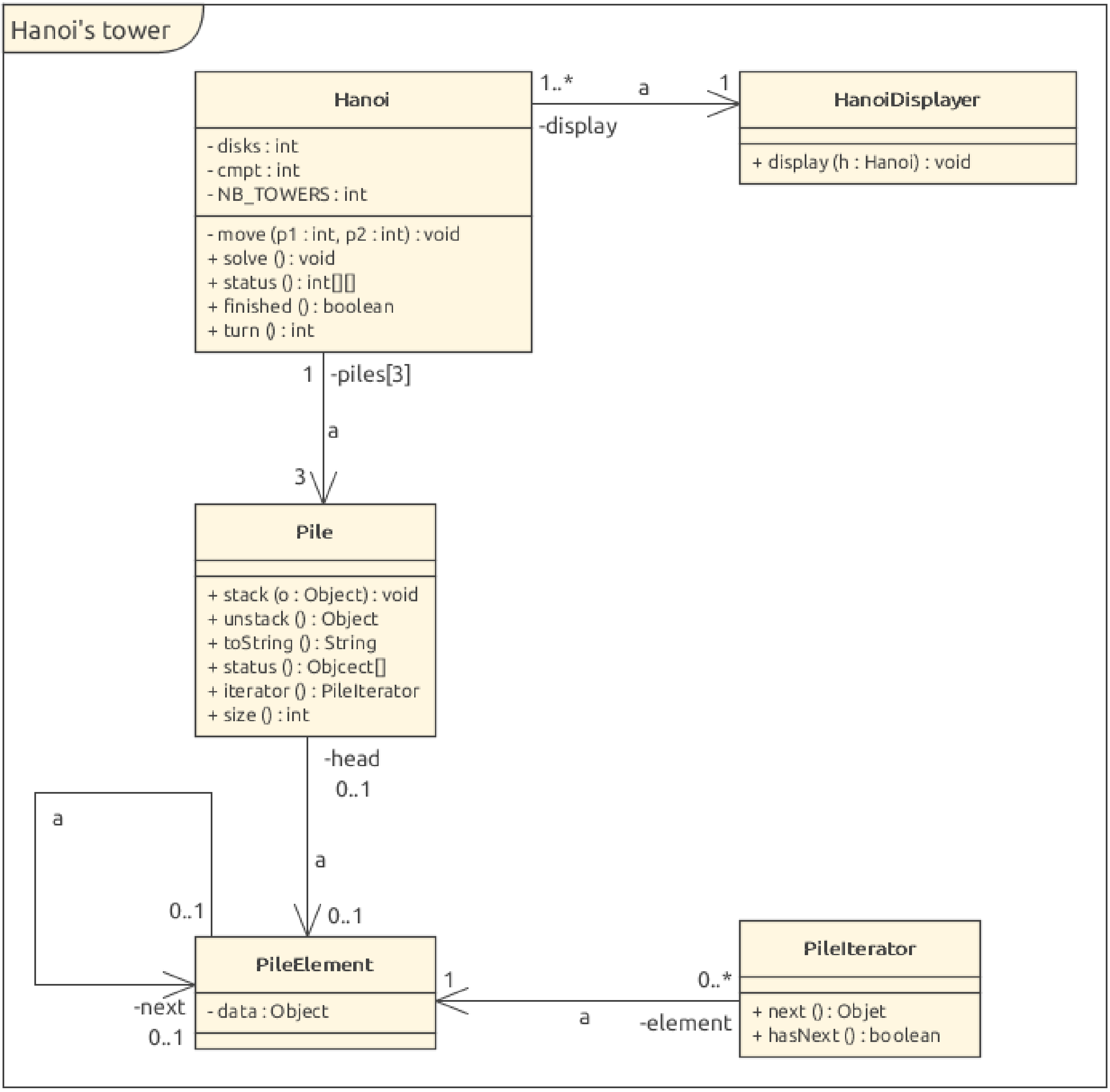
Supposons que l'algorithme soit vrait pour n. On a alors $2^n - 1$ coups pour n anneaux.

Pour un anneau de plus, soit $n + 1$ anneaux, on doit transférer une tour de taille n sur la pile intermédiaire (le via ci-dessus) en $2^n - 1$ coups, puis on doit déplacer l'anneau supplémentaire de la pile de départ à la pile d'arrivée, en $1\ coup$, puis redéplacer toute la tour de taille n de la pile intermédiaire à la pile finale, au dessus de l'anneau supplémentaire qui est déjà en place.

Au total pour $n + 1$ disques on a donc $(2^n - 1) + 1 + (2^n - 1)$ déplacements soit $2(2^n - 1) + 1 = 2^{n+1} - 1$ qui corresponds à la formule que l'on a supposé vrai.

Comme démontré ci dessus le pas initial est correcte et le pas d'incrémentation est correct la formule est donc démontrée.

Diagramme des classes



Description des classes

Voir la $JAVADOC$ fournie en annexe.

Sources du programme