Les interfaces: Les Tours de Hanoï

Projet 3 / GL / LP DA2I / semestre 1 / 2017-2018

La légende des Tours de Hanoï

Une légende raconte qu'on trouve à Hanoï un temple bouddhiste très ancien, dans lequel se dressent trois mâts d'airain. Sur l'un de ces mâts sont enfilés soixante-quatre disques de jade formant une pyramide.

Les bonzes qui s'affairent dans le temple doivent déplacer tous les disques du premier mât vers celui du milieu, en obéissant à des règles simples:

- seul un disque du sommet d'une pile peut être déplacé;
- on ne peut déplacer qu'un disque à la fois;
- un disque ne peut être posé que sur un disque plus grand ou dans une pile vide;
- à tout moment, on peut déplacer un disque de n'importe quelle pile vers n'importe quelle autre pourvu qu'on respecte toutes les règles ci-dessus.

Lorsqu'ils auront terminé, dit la légende, ce sera la fin du monde.

Informatisons les Tours de Hanoï!

Pour modéliser avec des objets ce problème, il faut disposer de *piles* et de *disques*. Les piles implémenteront l'interface `Pile` vue au TP3. Quant aux disques, ils doivent implémenter les interfaces `Comparable` (à cause de la règle sur les dimensions), et `Disque` (un disque est un objet ayant un diamètre).

Q1.

Récupérez et lisez attentivement les fichiers `Disque.java` et `DisqueHanoi.java` fournis sur Moodle.

Q2.

Écrivez, en vous inspirant du code de `PileTableau`, une classe `PileHanoi` (qui est une pile de disques de Hanoï et qui implante donc l'interface `Pile`), qui respecte les règles suivantes:

- l'empilement ne peut avoir lieu que si le disque à empiler est plus petit que celui du sommet;
- le déplacement d'un disque vers une autre pile ne peut donc se faire qu'aux mêmes conditions;
- l'affichage de la pile commence par la base, et on se contente d'afficher le diamètre des disques.

Q3.

Récupérez également `Hanoi.java` pour tester votre classe `PileHanoi`. Ce programme initialise les trois piles nécessaires et les affiche, puis attend des ordres de déplacement d'une pile à l'autre en affichant le résultat à chaque fois.

- Exemple d'exécution:

```
A : 4 3 2 1
B:
C:
Déplacement de : A
Vers : B
Je déplace un disque de A vers B
A : 4 3 2
B: 1
C:
Déplacement de : A
Vers : B
Impossible !
A : 4 3 2
B: 1
C:
Déplacement de : STOP
OK ! Terminé...
```

Q4.

Essayez de résoudre à la main le problème des tours de Hanoï pour un, deux, trois et quatre disques avec cette méthode. Notez tous les déplacements de disques nécessaires et comparez-les.

Automatisation de la résolution

Pour résoudre automatiquement le problème des Tours de Hanoï pour n disques, il «suffit» de savoir résoudre le problème pour $n-1$ disques. En effet, supposons qu'on soit parti d'une situation où quatre disques sont empilés sur le premier mât et qu'on sache déplacer des piles de 3 disques (ou moins) d'un mât à un autre. Dans ce cas, il suffit d'utiliser cette connaissance pour

1. déplacer les trois plus petits disques sur le deuxième mât,
2. déplacer le plus grand est sur le troisème mât, et
3. déplacer les trois plus petits disques du deuxième vers le troisième mât;

Q5.

Écrivez dans `PileHanoi` une méthode récursive `void deplacerDesDisques(int n, Pile dest, Pile interm)` qui déplace n disques (un par un), de la pile courante vers la pile `dest` en passant si besoin par la pile `interm`.

Q6.

Écrivez ensuite une méthode `static void resoudreAuto(PileHanoi a, PileHanoi b, PileHanoi c)` dans la classe `Hanoi` qui procède automatiquement à la résolution en affichant chaque déplacement de disque.

Q7.

Enfin, rajoutez dans le `main` un test sur les arguments de la ligne de commande pour que :

- `java Hanoi` lance la résolution en mode manuel (comme précédemment)
- `java Hanoi --auto` lance la résolution en mode automatique

Gestion de l'affichage

On voudrait maintenant pouvoir appliquer à chaque pile un algorithme d'affichage *a priori* quelconque, indépendant pour chaque pile. On propose pour cela de rajouter à `PileHanoi` un attribut dédié à l'affichage et de modifier ainsi la méthode `toString`:

```
public class PileHanoi implements ... { // ...
    private Affichage algoAffichage = new AffichageSimple() ; // affichage p
    //    ...
    public PileHanoi(String nom, Affichage a) { // constructeur specifiant u
        this.nom = nom ;
        algoAffichage = a ;
        // ...
    }
    public String toString() {
        Disque [] lesDisques = ...; // les disques contenus dans la pile cou
        //  ...
        return nom + " : " + algoAffichage.affichage_tableau(lesDisques, nbD
    } // ...
}
```

Q8.

À partir de l'interface `Affichage` (qui vous est donnée également), écrivez la classe `AffichageSimple` correspondante, qui se contente d'écrire le diamètre des disques.

On doit obtenir visuellement la même chose qu'auparavant.

Q9.

Placez le fichier `affichage.jar` dans votre propre répertoire `classes` et créez maintenant une des piles avec `new PileHanoi("jolie pile", new AffichageJoli())`. Que remarquez-vous ? Expliquez ce qui se passe. NB: Il faut compiler en utilisant l'option `-classpath classes/affichage.jar` pour indiquer au compilateur où trouver le bytecode de la classe `AffichageJoli`, dont vous ne possédez pas les sources. Les fichiers JAR sont des archives destinées à empaqueter du *bytecode* et d'autres fichiers nécessaires au bon fonctionnement d'une application (cf. doc. JDK).

Q10.

Faites de même avec `AffichageGraphique` (consultez la documentation fournie d'abord!).

Questions subsidiaires

Q11.

On appelle $H(n)$ le nombre d'opérations (déplacements de disques) nécessaires pour résoudre le problème des Tours de Hanoï lorsqu'il y a `n` disques. Étant donnée la méthode récursive employée, déterminez comment calculer $H(n+1)$ en fonction de $H(n)$. Déduisez-en la valeur de $H(n)$ en fonction de n. Faites une vérification expérimentale de votre formule pour $n \leq 6$. Calculez enfin la date approximative de la fin du monde.

Q12.

Proposez une implémentation pour la classe `AffichageJoli`.

Auteur: Sébastien Picault

Emacs 25.2.1 (Org mode 9.1.1)