TP n°3

Piles, Tri et Entrepot

Ce TP nous permettra de nous familiariser avec la programmation avec des classes. Nous implémenterons une structure de données simple (une pile) dans l'exercice 1, avant de nous en servir pour les exercices suivants.

La classe `Scanner`. La classe `Scanner` de la bibliothèque `java.util` de java permet la lecture de données sur l'entrée standard (clavier) ou dans un fichier. Nous parlerons ici simplement de la lecture au clavier. Pour utiliser la classe `Scanner`, il faut d'abord l'importer :
`import java.util.Scanner;`
Ensuite il faut créer un objet : `Scanner sc = new Scanner(System.in);`

Pour récupérer les données, il faut faire appel sur l'objet `sc` aux méthodes décrites ci-dessous. Ces méthodes parcourent la donnée suivante lue sur l'entrée et la retourne :

- `String next()` : donnée de la classe String qui forme un mot,
- `String nextLine()` : donnée de la classe String qui forme une ligne,
- `int nextInt()` : donnée entière de type int,
- `double nextDouble()` : donnée réelle de type double.

Il peut être utile de vérifier le type d'une donnée avant de la lire. Avec le code qui suit nous nous assurons que l'utilisateur donne un entier.

```
while (!sc.hasNextInt()) {
   System.out.println("Un nombre entier, s'il vous plait.");
   sc.nextLine();
}
int num = sc.nextInt();
System.out.println("Merci d'avoir donné le nombre " + num + ".");
```

Il existe d'autres méthodes de la classe `Scanner` (voir la documentation de Java). Quelques exemples :

- `boolean hasNext()` : renvoie `true` s'il y a une donnée à lire,
- `boolean hasNextLine()` : renvoie `true` s'il y a une ligne à lire,
- `boolean hasNextDouble()` : renvoie `true` s'il y a un double à lire.

Dans les exercices qui suivent on utilisera la classe `Scanner` pour les saisies au clavier.

Exercice 1 Une pile est un ensemble d'objets gérés de telle manière que le dernier élément ajouté est le premier que l'on peut extraire. On considère qu'un élément de pile encapsule une valeur entière.

1. Définir la classe `ElementPile` qui représente un élément d'une pile. Les attributs de cette classe seront privés. Définir les constructeurs et les accesseurs de cette classe.
2. Définir la classe `Pile`. Le constructeur de cette classe construira la pile vide. Les éléments seront gardés dans un tableau de dimension 100. Un champ privé tiendra en compte le nombre des éléments de la pile et de l'indice du son sommet.
3. Définir une méthode `estVide` permettant de tester si la pile est vide.
4. Définir la méthode `empile`(ajoute un élément au sommet de la pile)
5. Définir la méthode `depile` (retourne le sommet et le retire de la pile)
6. Définir la méthode `sommet` (retourne le sommet de la pile sans le retirer)
7. Définir la méthode `affiche` qui affiche le contenu d'une pile, sans la modifier.

Exercice 2 Définir une classe `TestPile` qui contient deux piles : `A` et `B` et deux méthodes statiques :

- `static void main(String[] args)`
- `static Pile tri(Pile p)`

La méthode `static void main (String[] args)` crée la pile `A` initialement vide et vous permet de la remplir : les éléments sont lus à partir du clavier et empilés les uns sur les autres dans une boucle de demander à chaque fois s'il faut continuer ou moins à l'entrée. En suite la méthode fait de même pour la pile `B` et permet de tester les méthodes `empile`, `depile`, `sommet`, `affiche` pour la pile `A`.

Écrire une méthode `union` qui réalise l'union de deux piles `A` et `B`. Écrivez le code qui vous permet de tester la méthode `union`.

La méthode `static Pile tri(Pile p)` crée une pile vide `res`, remplit cette pile avec les éléments de `p` d'une manière ordonnée, avec le minimum en haut, et renvoie `res`. L'algorithme proposé est le suivant : on utilise une pile auxiliaire `aux` qui est vide au début et tant que la pile `p` n'est pas vide, on considère les deux cas suivants :

- si la pile `res` est vide ou si l'élément au sommet de `p` est plus petit que celui de `res` :
  on retire l'élément au sommet de la pile `p` pour empiler dans la pile `res`, puis si la pile `aux` n'est pas vide on retire tous les éléments de la pile `aux` pour empiler dans la pile `res`.
- sinon :
  on deplace l'élément au sommet de la pile `res` à la pile `aux`.

Testez votre méthode de tri sur la pile `A` $= \{4, 3, 2, 5, 8, 2, 6, 9, 3\}$.

Exercice 3

1. Soit la classe `Produit` définie par un nom, un prix et le nombre de jours restant avant péremption du produit. Soit la classe `Entrepot` définie par un ensemble de produits et le nombre de produits périmés et non périmés. L'ensemble sera vu comme une pile. Vous aurez par ailleurs à utiliser les méthodes de la classe `Pile`. La classe `Pile` doit être modifiée pour gérer les objet `Produit`. On appelera cette modification `PileEntrepot`. Adaptéz toutes les méthodes `empile`, `depile`, `sommet`, `affiche` au cas des entrepôts. Ajoutez à la classe `PileEntrepot` les méthodes suivantes :
   - `suppression` qui ôte de l'ensemble tous les produits périmés et renvoie la somme perdue ;
   - `valeur` qui renvoie la valeur totale des marchandises dans l'entrepôt.
2. Définir une classe `TestPileEntrepot` qui contient deux entrepots : `A` et `B` et deux méthodes statiques :
   - `static void main(String[] args)`

– `static PileEntrepot tri(PileEntrepot pe)`
Comme dans l'exercice précédent la méthode `static void main(String[] args)` crée deux entrepôts `A` e `B` initialement vides et permet à l'utilisateur de les remplir avec des produits à l'aide du clavier. Également modifier la méthode `union` et tester toutes les méthodes. La méthode `trie` trie un entrepot suivant la date de péremption en plaçant en haut de la pile le produit dont le nombre de jour avant péremption est le plus faible (ce nombre peut être négatif).