Institut Galilée ISPG 1 2019-2020
Programmation Orientée Objet

T.P. 1 (Piles et Tri)

Dans ce TP on va faire un simple exercice de programmation avec des classes. On en profitera également pour commencer à se familiariser avec l'environnement de développement eclipse. Eclipse, comme emacs, permet d'écrire des fichiers .java, mais il a beaucoup plus de fonctionnalités, qui rendent la programmation plus rapide et organisée, et permettent de trouver les erreurs et de tester les programmes plus facilement. Vous pouvez commencer à utiliser eclipse comme éditeur, et découvrir au fur et à mesure ses fonctionnalités. Lancez ecplise à partir d'un terminal. Choisissez un répertoire à utiliser comme "workspace" par exemple "POO". Créez un projet java "TP1". Cela créera un répertoire "TP1" qui contient les fichiers .java et les fichiers .class. Eclipse vous permet de compiler et exécuter vos programmes sans passer par le terminal.

Dans les exercices qui suivent, on va utiliser la classe Scanner, qui a eté introduite à partir de la version java 1.5.0. Elle simplifie 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 de la classe `Scanner` : `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,
- `boolean nextBoolean()` : donnée booléenne,
- `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 :

- `boolean hasNext()` : renvoie `true` s'il y a une donnée à lire,
- `boolean hasNext(String pattern)` : renvoie `true` si la prochaine donnée à lire forme le mot pattern,
- `boolean hasNextLine()` : renvoie `true` s'il y a une ligne à lire,
- `boolean hasNextBoolean()` : renvoie `true` s'il y a un booléen à lire,
- `boolean hasNextInt()` : renvoie `true` s'il y a un entier à lire,
- `boolean hasNextDouble()` : renvoie `true` s'il y a un double à lire.

Il existe d'autres méthodes de la classe `Scanner`. Si cela vous intéresse, allez consulter l'API de java. Dans les exercices que suivent on utilisera la classe `scanner` pour les saisies au clavier.

Exercice 1(Piles)

Écrire une classe implantant une pile d'éléments. 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. Comment représenter la pile vide ?
3. Définir la classe `Pile`. Le constructeur de cette classe construira la pile vide.
4. Définir une méthode permettant de tester si une pile est vide.
5. Définir la méthode `empile` (ajoute un élément au sommet de la pile)
6. Définir la méthode `depile` (retourne le sommet et le retire de la pile)
7. Définir la méthode `sommet` (retourne le sommet de la pile sans le retirer)
8. Définir dans la classe `Pile` la méthode `affiche` qui affiche le contenu d'une pile.

Tester la création d'une pile et sa manipulation en empilant puis dépilant divers éléments...

Exercice 2 (Tri par insertion et piles)

Écrire un programme de tri par insertion d'un ensemble de nombres entiers. Les données sont stockées dans une pile A et le programme doit retourner une pile B contenant ces nombres triés avec le minimum au sommet de la pile. L'algorithme proposé est le suivant : on utilise une pile C qui est vide au début. Tant que la pile A n'est pas vide, on considère les deux cas suivants :

— si la pile B est vide ou si l'élément au sommet de A est plus petit que celui de B : on retire l'élément au sommet de la pile A pour empiler dans la pile B, puis si la pile C n'est pas vide on retire tous les éléments de la pile C pour empiler dans la pile B.

— sinon : on deplace l'élément au sommet de la pile B à la pile C.
Définir une classe `Tri` qui contient trois piles A, B et C, une méthode `tri(Pile A, pile B, Pile C)` et la méthode `main()`. La pile A peut être construite à partir d'un tableau d'entiers en utilisant la méthode `empile`. Tester avec la pile A = 4, 3, 2, 5, 8, 2, 6, 9, 3.

Exercice 3

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. 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

1. écrire la méthode `construitEntrepot` qui construit un ensemble à partir de données entrées au clavier.
2. écrire la méthode `trie` qui trie un ensemble 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.
3. écrire la méthode `suppression` qui ôte de l'ensemble tous les produits périmés et renvoie la somme perdue.
4. écrire les méthodes qui réalisent l'union, `union`, et l'intersection, `intersection`, triées de deux ensembles triés. Attention à la valeur de la variable comptant le nombre de produits périmés.