Programmation objet TD 4
Structure de données, Exception, Interface, Classe interne,

Exercice 0:

-Qu'est ce qu'une classe interne ?

-Quelles sont les différences entre une interface et une classe ?

-Quelle différence y a t il entre une exception et une erreur ?

Exercice 1: interface, classe interne

Le type abstrait `Pile` correspond à une structure de données (LIFO) pour laquelle sont définies les opérations `pileVide` : qui teste si la pile est vide, `empiler` qui ajoute un élément au sommet de la pile, `dépiler` qui retire l'élément au sommet de la pile et `sommet` qui retourne l'élément qui est au sommet. On se propose d'implémenter plusieurs types de piles.

a) Ecrire l'interface `Pile` qui décrit le type abstrait Pile (`empile, dépile, pileVide , sommet`).

b) Une pile peut être implémentée de manière contiguë grâce à un tableau. Fournir une classe `PileATableau` qui implémente une pile d'objets, en commençant par les deux champs et le constructeur qui crée le tableau physique.

Rappel :

```
Object[] T ;// crée une variable qui contiendra un tableau
T = new Object[100] ; crée en mémoire le tableau
```

Ex d'utilisation de la pile à tableau :

```
Compte C1 = new Compte(), C2 = new Compte() ;
PileATableau P1, P2 ;
P1 = new PileATableau(10) ;
P1.empile(C1) ;  P1.empile(C2) ;  P1.depile() ;  Compte  C  =
P1.sommet() …
```

c) Ecrire une méthode `viderEtSommerEntiers` qui dépile tous les éléments contenus dans la pile et fait la somme de tous les entiers contenus dans cette pile. Indication: `o instanceof T` permet de tester si l'objet o est de la classe T ou d'une classe descendante de T.

d) Une pile peut également être implémentée à l'aide d'une liste chaînée. Proposer une réalisation en Java de cette implémentation chaînée. On écrira la classe `PileAListe` à un champ ; elle utilisera une classe à programmer `Maillon` qui décrit un maillon et contient un constructeur à deux paramètres (à programmer).

e) Ecrire un programme d'essai qui crée une `pileAListe`, y met 3 éléments (1, 2, 3) puis affiche les éléments de la pile sans modifier la pile.

f) Quels sont les avantages et inconvénients de ces deux implémentations de l'interface `Pile` ?

Remarque : la classe java.util.Stack modélise une pile d'objets

Exercice 2: Exception

On gère les classes `Compte` (composé des champs `crédit` et `débit`, des méthodes `déposer` et `retirer`) et CompteEpargne qui en hérite, et est composée du champ `Taux` et de la méthode `déposer` en redéfinition qui prend 1 euro de commission.

On souhaite gérer les exceptions qui peuvent avoir lieu lors du codage du programme.

Pour ce faire, on souhaite créer une classe `CompteException` qui sera composée d'un champ `numErreur` qui gardera le numéro de l'erreur levée, et d'un champ `méthode` qui stockera le nom de la méthode qui a généré l'erreur.

Les erreurs possibles correspondent au fait de retirer ou déposer de l'argent avec un nombre négatif ou égal à 0. L'erreur aura le code erreur 0 pour un compte simple, 1 pour un compte épargne.

1) Ecrire la classe `CompteException`; celle-ci sera composée des deux champs en privé, d'un constructeur à deux paramètres les remplissant, et des accesseurs en lecture en public.

2) Ecrire les classes `Compte` et `CompteEpargne`

3) Ecrire un programme de test qui crée un compte `C1` et un compte épargne `CE1`; qui demande un nombre entre 1 et 4 (on suppose l'existence d'une fonction `readInt()` qui lit et retourne un nombre); si le nombre est 1 (resp. 2, 3, 4), le programme dépose "-10 euros" sur C1 (resp. dépose "-10 euros" sur CE1, retire "-10 euros" sur C1, retire "-10 euros" sur CE1).

Ce programme est composé de sorte à ce qu'une gestion des erreurs ait lieu: elle affiche pour une erreur le nom de la classe qui a généré l'erreur, et le nom de la méthode qui la contenait.

Exemple: en tapant 1, s'affiche: `"erreur de Compte, dans déposer".`

Exercice 3: Tri de tableau, interface

On souhaite écrire en Java une méthode de tri générique qui permette de trier un tableau d'objets de type quelconque selon un de ses attributs.

1) on dispose d'un tableau de joueurs, définis par leur nom, leur âge, leur score, leur adresse.
a) Ecrire la classe `joueur`.
b) On veut pouvoir les trier suivant chacun de ces critères. La seule propriété nécessaire est que les éléments soient comparables.
Pour cela, on définit une interface `Compare` qui permet de définir le profil d'une comparaison `inf`.

```
public Interface Compare{
{public boolean inf(Object x , Object y)};} //rend vrai si x<y
```

Ecrire une classe de comparateur `CompareAge` qui implémente un comparateur d'age pour les joueurs.
Exemple d'utilisation:

```
joueur J1, J2, J3; J1= new joueur(),J2= new joueur(),J3= new
joueur(),
... // on remplit les joueurs
CompareAge CA; bool B; CA=new ComparAge
B = CA.inf(J1, J2) ; // B est vrai si l age de J1 < l age de
J2
```

Notez qu'on pourrait écrire d'autres classes de comparateurs: `CompareScore`, `CompareAdresse.`
2) Ecrire une classe de tri à bulle générique `TriGénérique` qui reçoit en paramètre un tableau à trier et un comparateur du type de l'interface. Cette classe contient une unique méthode de classe `Tri`.
Exemple d'utilisation (suite)

```
joueur [] T;
```

```
T=new joueur[22];...
TriGenerique.tri(T, new CompareAge())
```

On peut utiliser la méthode `length` qui donne la taille d'un tableau.
Rappel tri à bulle:
T: tableau [0..N] entiers; i,j : entier
pour i de N à 0 faire
    pour j de 1 à i faire si (T[j] < T[j-1]) alors x = T[j-1] ; T[j-1] = T[j] ; T[j] = x
A partir de la version 5, JAVA propose des paramètres de type qui permettent d'obtenir une autre solution.