TD PAC - Java n° 5

Exercice 1
On souhaite représenter des doublets et des triplets d'entiers, manipuler leur somme et leur produit. 1. Construire une classe **Doublet** ayant comme attribut x et y, et les méthodes add qui prend en entrée un autre doublet et retourne un nouveau doublet qui est la somme des deux doublets, toString qui retourne une chaîne de caractères d'écrivant le doublet. Construire la classe **Triplet**, héritant de la classe Doublet, ses méthodes add et toString. Se servir autant que possible des méthodes de la classe Doublet.
2. On souhaiterait manipuler la somme et le produit des attributs contenus dans un doublet ou dans un triplet, et ce indifférement de la classe, par exemple par l'intermédiaire de cette classe :

```
class Traitement {
      static Doublet traitement(SommeProduit ref)
      {  return new Doublet(ref.somme(),ref.produit()); }
}
```

Définir l'interface **SommeProduit**, contenant les méthodes somme et produit.
3. Construire les classes **Doublet2** et **Triplet2**, héritant respectivement de **Doublet** et **Triplet**, implantant l'interface **SommeProduit**.
4. Faire un exemple d'utilisation en affichant le résultat de **Traitement** sur (51 ; 16) et de (64 ; 1 ; 33).

Exercice 2
L'objectif est de créer une classe abstraite **ListeTri** correspondant a une liste d'objets triée. La classe doit comporter :

- un attribut *liste*, de type vecteur ;
- une méthode *plusGrand*, prenant deux objets en entrée et renvoyant vrai si le premier objet est plus grand que le deuxième, faux sinon ;
- une méthode *add*, qui prend un objet en entrée et l'insère au bon endroit dans liste ;
- une méthode *afficher* affichant liste.

Quelles méthodes doivent être abstraites ? Implanter cette classe. Définir la classe heritée **listeTriDoublet**, représentant une liste triée de doublets (reprendre la classe **Doublet2** de l'exercice précedent), en considérant qu'un doublet est plus petit qu'un autre si le produit est plus petit. Peut-on utiliser une interface pour résoudre l'exercice ?

Exercice 3 Définir la classe **Coureur**. Un coureur est défini par la distance qu'il a parcourue et le temps qu'il a mis. On pourra supposer ces deux valeurs données par des entiers. Définir un constructeur, une méthode d'affichage, et une méthode de calcul de sa vitesse moyenne. La méthode *calculVitesse* doit traiter le cas exceptionnel où le coureur a parcouru une distance de 0 en un temps égal à 0. On rappelle que dans le cas d'une division *flottante* par zéro, on n'obtient pas une *ArithmeticException* mais un résultat qui peut valoir *POSITIVE INFINITY, NEGATIVE INFINITY* ou *NaN* qui sont définis comme *static float* dans la classe **Float**. On pourra définir l'exception *DivideByZeroException*.

Exercice 4
Le but est ici de modéliser un petit jeu de plateau. Nous avons deux classes principales, **Plateau** et **Piece** *(ces deux objets ne sont pas ceux du TD precédent)* . Deux types de pièces existent, les pièces A se déplacant en ligne droite et les pièces B se déplacant en diagonal. Les pièces ne bougent que d'une seule case par mouvement. Les pièces sont repérées sur le plateau par le numéro de la ligne et le numéro de la colonne. Elles ont en outre une couleur.
Nous voulons également utiliser l'interface :

```
 interface Graphique{
  void affiche();
}

et l'interface existante en Java :
interface Comparable{
  int compareTo(Object o);
}
```

renvoyant un nombre négatif, zéro, ou un nombre positif lorsque l'objet est plus petit, égal ou supérieur a l'objet passé en paramètre.

1. Sachant que Piece implantera l'interface Comparable, créer une classe **ListePiece** contenant un attribut qui est une liste de pièces triée, avec sa méthode *add* placant une nouvelle pièce au bon endroit dans la liste (on peut se servir de l'exercice 2).
2. Créer la classe abstraite **Piece** et les deux classes de pièces possibles. On utilisera entre autre les deux interfaces, un identificateur unique pour chaque pièce. On affichera un + pour une pièce A et un x pour une pièce B. Une pièce a est plus petite qu'une pièce b si le numero de ligne de a est plus petit que celui de b. En cas d'egalité, c'est la pièce ayant le plus petit numéro de colonne qui est la plus petite. Ne pas oublier de munir les classes de toutes les méthodes nécessaires pour un accès simple aux attributs. Ajouter dans ce but a la classe **ListePiece** les méthodes *getXpos, getYpos*, prenant en entrée un entier i et renvoyant les positions de la pièce numéro i dans la liste, et la méthode *getPiece* renvoyant la pièce numéro i.
3. Créer la classe Plateau. Elle contient une liste triée de pièces, les dimensions du plateau et une méthode d'affichage texte (elle implante Graphique). On affichera un pour une case vide, la pièce sinon (se servir du fait que les pièces sont triées par ligne puis par colonne). Tester sur des exemples en créant une classe Jeu, créant le plateau et quelques pièces.
4. On veut maintenant gérer le déplacement des pièces. Pour cela, on va ajouter une méthode *newPos* a la classe **Piece**, prenant en entrée deux entiers représentant la nouvelle position. Elle fera appel a une nouvelle méthode *del* de plateau, supprimant une pièce du plateau selon son identificateur, et à la méthode *add* de plateau (qui la repositionne). Il faudra modifier la class **Piece** pour qu'elle référence le plateau.
5. Modifier les classes relatives aux pièces en introduisant des exceptions. On veut en particulier qu'une pièce ne puisse pas se situer en dehors du plateau, que deux pièces ne se situent pas sur la même case et que le déplacement soit valide.
6. Tester le tout sur différents exemples en créant une classe **Jeu**, créant le plateau, quelques pièces et quelques mouvements.