

**Programmation Java 2003/2004**  
td-tp chapitre 4  
**Classes dérivées - classes abstraites - interfaces**

**Comptes**

1. Définir un menu dans une classe principale pour créer des comptes et effectuer des opérations.
2. Redéfinir la classes Compte, comme sous-classe de la classe Compte0.  
Redéfinir les classes CompteEnBanque et CompteAvecSauvegarde comme sous-classes de la classe Compte.

**Matrices**

Les classe Matrice, MatriceCarree et Matrice33 sont documentées ci-après.

1. Ecrire des programmes de test pour chacune de ces classes.
2. Définir les classes.

```
-----  
/** implémentation des matrices à 2 dimensions */  
public class Matrice  
  
/** crée une Matrice nulle de dimensions 'n' et 'm' */  
    public Matrice(int n, int m)  
  
/** crée une matrice de dimensions 'n' et 'm' remplie par les  
 * éléments du tableau 'tab'  
 * si une ligne du tableau est trop courte ou trop longue elle  
 * est complétée par des 0 ou tronquée  
 * s'il n'y a pas assez de lignes on complète par des 0, s'il y a  
 * trop de lignes, les dernières ne sont pas prises en compte  
 */  
    public Matrice(int n, int m, int [][] tab)  
  
/** crée une matrice remplie par les éléments du tableau 'tab'  
 * éventuellement complété par des 0 dont la première dimension  
 * est égale au nombre de lignes de 'tab' et la deuxième dimension  
 * est égale au plus grand nombre d'éléments dans les lignes de 'tab'  
 */  
    public Matrice(int [][] tab)  
  
/** rend une représentation sous forme de chaine de cette Matrice */  
    public String toString()  
  
/** rend si possible la Matrice somme de cette Matrice et de la Matrice  
 * 'mat' sinon rend null  
 */  
    public Matrice somme(Matrice mat)  
  
/** rend si possible la Matrice produit de cette Matrice et de la Matrice  
 * 'mat' sinon rend null  
 */  
    public Matrice produit(Matrice mat)  
  
-----  
  
/** implémentation des matrices à 2 dimensions */  
public class MatriceCarree extends Matrice  
  
/** crée une MatriceCarree nulle d'ordre 'n' */  
    public MatriceCarree(int n)  
  
/** crée une MatriceCarree d'ordre 'n' remplie par le  
 * tableau 'tab' éventuellement tronqué ou complété par des 0  
 */  
    public MatriceCarree(int n, int [][] tab)
```

```

/** crée une MatriceCarree remplie par le tableau 'tab' éventuellement
 * complété par des 0 dont la dimension
 * permet de conserver tous les éléments de 'tab'
 */
public MatriceCarree(int [][] tab)

/** crée une MatriceCarree d'ordre 'n' à partir d'une Matrice 'mat'
 * éventuellement tronquée ou complétée par des 0
 */
public MatriceCarree(int n,Matrice mat)

/** rend le carré de cette MatriceCarree */
public MatriceCarree carre()

-----

/** implémentation des matrices 3x3 */
public class Matrice33 extends MatriceCarree

/** crée une matrice nulle 3x3 */
public Matrice33()

/** crée une matrice 3x3 remplie par le tableau 'tab'
 * éventuellement tronqué ou complété par des 0
 */
public Matrice33(int[][] tab)

/** crée une matrice 3x3 à partir d'une MatriceCarree éventuellement
 * tronquée ou complétée par des 0
 */
public Matrice33(MatriceCarree mat)

/** rend le déterminant de cette Matrice33 */
public int determinant()

-----

```

## **Piles** (Exercice IV - partiel avril 2003)

1. On veut implémenter en Java la structure de *pile* (dernier entré, premier sorti). On veut pouvoir *empiler* des objets quelconques (instances de la classe *Object*), *dépiler* le dernier objet entré, renvoyer l'objet au *sommet* de la pile, tester si la pile *est vide*.

- 1.1. Donner la documentation d'une classe `Pile` pour réaliser ces fonctions.
- 1.2. Ecrire une classe principale pour tester les fonctions de la classe `Pile`.
- 1.3. Définir la classe `Pile`.

2. Définir une sous-classe `PileAvecCumul` de la classe `Pile` qui aura les caractéristiques suivantes :

Si on veut *empiler* un `Integer` (resp. un `Double`) alors que le *haut* de la pile est un `Integer` (resp. un `Double`), au lieu d'*empiler* on remplace le *sommet* de la pile par la somme des deux `Integer` (resp. des deux `Double`). Si on veut *empiler* une `String` (resp. `StringBuffer`) alors que le *haut* de la pile est une `String` (resp. `StringBuffer`), au lieu d'*empiler*, on remplace le *sommet* de la pile par la concaténation des deux chaînes.

## **Algorithmes de tri** (texte élaboré et enrichi à partir de l'exercice 2 de l'examen de sept 2001 (Y.Parchemal))

On considère l'interface `TypeElement` défini comme suit :

```

/** permet de définir des types d'éléments en pouvant tester l'appartenance
 * d'un élément à ce type */
interface TypeElement{ public boolean estUnElement(Object o); }

```

1) La classe `TypeIntervalleEntier` ci-dessous implémente l'interface `TypeElement` et définit les intervalles d'entiers. Ecrire une méthode `main` utilisant cette classe.

```

/** implementation de types d'intervalles d'entiers*/
class TypeIntervalleEntier implements TypeElement{
private int min,max;

```

```

public TypeIntervalleEntier(int min,int max){this.min=min;this.max=max;}

public boolean estUnElement(Object o){
    if (!(o instanceof Integer)) return false;
    else {int v = ((Integer)o).intValue();
        return ((v>=min)&& (v<=max));
    }
}
}

```

2) La classe `TriAbstrait` ci-dessous implemente la notion de critère de tri

```

abstract class TriAbstrait{
    public abstract int valeurCle(Object o);
    public Vector trier(Vector v){
        Vector res=(Vector)v.clone();
        for (int i=0;i<res.size()-1;i++){
            Object min = res.elementAt(i);int imin=i;
            for (int j=i+1;j<res.size();j++)
                if (this.valeurCle(min)>this.valeurCle(res.elementAt(j)))
                    {imin=j;min=res.elementAt(j);}
            res.setElementAt(res.elementAt(i),imin);
            res.setElementAt(min,i);
        }
        return res;
    }
}

```

a) pourquoi cette classe est-elle abstraite ?

b) que fait la méthode trier ?

c) écrire et tester une classe concrète `TriValeurAbsolue` dérivée de `TriAbstrait` réalisant le critère de tri suivant : "Les objets considérés sont des entiers. On désire les ranger par ordre croissant de valeur absolue."

d) considérer d'autres critères de tris en définissant d'autres classes dérivées de la classe `TriAbstrait`, par exemple :

- la classe `TriPairImpair` pour ranger tous les nombres pairs avant les nombres impairs
- la classe `TriUsuel` pour trier selon l'ordre usuel croissant
- la classe `TriInverse` pour trier selon l'ordre inverse et tout autre critère que vous pouvez imaginer.

3) La classe `Ensemble` permet de manipuler des ensembles avec les fonctionnalités indiquées implicitement ci-dessous dans la fonction `main` utilisant cette classe `Ensemble`.

```

public static void main(String[] args) {
    TypeElement intervalle=new TypeIntervalleEntier(-300,200);
    Ensemble e = new Ensemble(intervalle);
        e.ajouter(new Integer(3));
        e.ajouter(new Integer(-11));
        e.ajouter(new Integer(-34));
        e.ajouter(new Integer(144));
        e.ajouter(new Integer(-22));
        e.ajouter(new Integer(33));
        e.ajouter(new Integer(-78));
        e.ajouter(new Integer(-277));
        e.ajouter(new String("atyty"));
        e.ajouter(new Integer(300));
    TriAbstrait tri=new TriValeurAbsolue();
    System.out.println(e.getElement(3, tri));
    // le résultat renvoyé ci-dessus est le 3ème élément de l'ensemble en considérant les
    // éléments par ordre croissant de valeur absolue ; pour cet exemple c'est -22
    System.out.println(e.trier(tri)); // l'ensemble trié pour ce critère
}
}

```

Définir et tester la classe `Ensemble` avec tous les tris définis.

4) Définir et tester d'autres types implémentant l'interface `TypeElement`, par exemple :

- la classe `TypeEntier` contenant des entiers quelconques
- la classe `TypeMultiple` contenant des multiples d'un entier donné en paramètre.

## Classe abstraite Nombre et équations dans les complexes (Exercice II - examen juin 2003)

**Le but de cet exercice est de définir une classe abstraite Nombre, deux sous-classes Reel et Complexe de Nombre et une classe Equation.**

Ces classes permettent de résoudre des équations du second degré dont les coefficients sont soit des réels, soit des complexes. Pour simplifier, on ne traitera que des équations de la forme  $x^2 + bx + c$  où le coefficient principal est égal à 1 et où  $b$  et  $c$  sont des réels ou des complexes, instances de la classe Reel ou de la classe Complexe.

**Définir ces classes en respectant les indications ci-dessous.**

La classe Reel comprendra un attribut `val` de type `double` dont la valeur sera celle du réel représenté.

Le constructeur aura comme paramètre un `double` correspondant à la valeur du réel.

La classe Complexe comprendra deux attributs `reel` et `imag` de type `double` dont les valeurs seront les parties réelle et imaginaire du complexe représenté.

Le constructeur aura deux paramètres de type `double` correspondant aux parties réelle et imaginaire du complexe.

Les méthodes `somme`, `produit` et `racineCarree` seront des méthodes d'instance abstraites de la classe Nombre et définies dans ses deux sous-classes.

La méthode `carre` sera une méthode définie dans la classe Nombre.

Si la définition de la méthode `racineCarree` de la classe Reel est simple, la méthode `racineCarree` de la classe Complexe nécessite la résolution d'une équation du second degré à coefficients réels qui sera résolue en faisant appel à la classe Equation.<sup>1</sup>

La classe Equation comprendra deux attributs `b` et `c` instances de la classe Nombre. On appellera `racine1` la méthode renvoyant la plus grande racine et `racine2` l'autre racine. On tiendra compte, bien sûr, de la simplification apportée en se restreignant aux équations dont le coefficient principal est égal à 1.

---

<sup>1</sup> En effet, si  $a$  et  $b$  sont réels et si on cherche  $x$  et  $y$  tels que  $\sqrt{a+ib} = x+iy$  on a  $a+ib = (x+iy)^2 = x^2 - y^2 + 2ixy$ , d'où

$$x^2 - y^2 = a \text{ et } 2xy = b, y = b/2x, x^2 - \left(\frac{b}{2x}\right)^2 = a \text{ et enfin } x^4 - ax^2 - \frac{b^2}{4} = 0$$

L'équation  $X^2 - aX - \frac{b^2}{4} = 0$  ayant, si  $b \neq 0$ , deux racines non nulles de signes différents,  $x^2$  est égal à la racine positive de cette équation (qui est aussi la racine la plus grande) et  $x$  à la racine carrée de  $x^2$ .

(Si  $b=0$ , le calcul de  $\sqrt{a}$  est immédiat, réel ou imaginaire pur selon le signe de  $a$ ).