Série n°4

Héritage et Polymorphisme

Classes abstraites et interfaces

Exercice 1

On s'intéresse à l'implémentation d'une entité Employé. Un employé est décrit par un numéro (NSS), un nom, un prénom, une date de naissance et exerce une fonction dans l'entreprise. Les employés de l'entreprise peuvent être de trois catégories « permanent », « stagiaire » ou « contractuel ». Un employé permanent possède une expérience (en nombre d'années), il a un salaire (égal au salaire de base + primes – retenues, les primes et les retenues sont un pourcentage du salaire de base avec des taux définis). Un contractuel possède aussi une expérience, une date de début de contrat et une durée de contrat ; le contrat peut être renouvelé pour une durée donnée (en nombre de mois), un contractuel possède un salaire calculé de la même façon qu'un employé permanent. Un stagiaire possède une date de début de stage, une durée de stage et un présalaire fixe.
On voudrait pouvoir obtenir des informations sur les entités manipulées (ou modifier des données) grâce aux méthodes définies dans les classes.

1. Proposez une implémentation de l'entité Employé décrite ci-dessus avec les différents types comportant un constructeur, une méthode de saisie, d'affichage et les accesseurs. (on crée d'abord un type Date avec les méthodes nécessaires qu'on utilisera au besoin). NB : respecter le principe d'encapsulation.
2. Ecrire un programme qui crée une entité de type employé « contractuel », renouvelle son contrat pour une durée d donnée et affiche la description complète de l'objet.
3. Créer une structure de type Vector (classe prédéfinie dans java.util) pour stocker les informations d'un nombre n d'employés de différents types (permanent, contractuel, stagiaire). Afficher les noms et prénoms de tous les *stagiaires* ayant une durée de stage égale à 6 mois.

Exercice 2

1. Donner l'implémentation en java d'une classe **Liste** pour une liste chaînée d'entiers, avec les méthodes InsererEntete, InsererAlaFin, InsererApresElément (…), SupprimerPremier, supprimerElement (…), SupprimerDernier, ElementExiste (…), AfficheListe.
2. Redéfinir la méthode toString et la méthode equals de la classe Object
3. Dans une classe **ProgListe**, écrire un programme qui crée une liste d'entiers entrés au clavier, on s'arrête dès la rencontre d'une valeur qui existe déjà dans la liste et supprime ensuite une valeur val donnée de la liste.
3. A partir de la classe **Liste,** implémenter une classe **ListeOrd** qui implémente une liste triée d'entiers, par ordre croissant. (redéfinir les méthodes nécessaires)

4. Que faut-il modifier pour implémenter une liste d'objets ? (Par exemple une liste de nombres complexes).

Exercice 3

Dans cet exercice, on s'intéresse à la manipulation de comptes bancaires de clients. Un compte bancaire est décrit par un ***numéro***, une ***date*** de création, un ***état*** (bloqué ou non), et un ***solde*** (somme d'argent contenue dans le compte), il possède un ***titulaire*** (client, dans la classe compte, on enregistre le numéro du client). Un compte peut être créé, crédité (ajouter argent), débité (retirer argent) ou bloqué.
Un client est décrit par un ***numéro***, un ***nom***, un ***prénom***, une ***date*** de naissance et une ***adresse***. Un client peut changer son adresse. A tout moment, on doit pouvoir afficher les informations des clients et des comptes (redéfinir la méthode toString).

1. Définir en java les classes qui décrivent les deux entités **COMPTE** et **CLIENT**.
2. Ecrire un programme qui crée et remplit deux structures (de type Vector ou ArrayList), l'une regroupant les données de clients et l'autre regroupant les données des comptes. Les informations concernant les comptes et les clients sont entrées au clavier.
3. Ecrire un programme qui affiche les numéros de compte, les soldes, les noms et prénoms de clients ayant des comptes débiteurs (solde négatif).

Dans cette partie, on considère différents types de comptes :

- Un compte « Chèque » où le solde ne peut être que créditeur (positif).
- Un compte « Courant » dont le solde peut devenir négatif grâce à une autorisation du banquier et ne peut descende en dessous d'un seuil S négatif fixe (on suppose que le seuil est fixé à 5000 da).
- Un compte « Epargne » générateur d'intérêts selon un taux fixe (le taux est égal à 7%). Le cumul des intérêts est calculé par rapport au solde et en fonction du taux d'intérêt.

4. Proposer une nouvelle description des classes en utilisant l'héritage.

5. On suppose que les données des comptes et des clients sont déjà stockées dans des structures de type ArrayList (classe prédéfinie dans java.util). Ecrire un programme qui calcule les intérêts cumulés sur tous les comptes Epargne. Les résultats devront être affichés en précisant le nom et prénom des clients.

Exercice 4

On considère les classes **Point** et **PointEspace** définies comme suit :
Point : caractérisée par deux attributs x et y de type double, des constructeurs et les méthodes : String *toString()*,
void *deplacer(double d)*, on déplace des deux coordonnées

void *deplacer (Point p)* (On inclut dans la méthode, le message « déplacer de Point »)
Point *symetrie()*.(On inclut dans la méthode, le message « Symétrie de Point »)

PointEspace : dérivée de la classe Point, à laquelle on ajoute un attribut *z*, des constructeurs et les méthodes :

String *toString()*,

void *deplacer (double d)*, on déplace les trois coordonnées

void *deplacer (PointEspace p)* (On inclut dans la méthode, le message « déplacer de PointEspace »)

PointEspace *symetrie()*. (On inclut dans la méthode, le message « Symétrie de PointEspace »)

Questions :

1. Implémenter les classes **Point** et **PointEspace**
2. Déterminer les méthodes surchargées et les méthodes redéfinies.
3. Les instructions suivantes sont-elles correctes :

```
PointEspace P5 = new PointPlan (10.0, 15.0) ;
Point p1 = new Point(1,4) ;        PointEspace p5 =
 (PointEspace) p1;
PointEspace p2 = new PointEspace (3,6, -1) ;
((PointEspace) p1). deplacer (p2) ;
```

4. Soit le code suivant :

   Point **p1** = new Point(1,4) ; Point **p2** = new Point(1,4) ;
   PoinEspace **p3** = new PointEspace (5,4,3) ;
   Point **p4** = new PointEspace (1,1,1) ;

Remplir le tableau suivant :

Rappel: La *liaison dynamique* est un mécanisme qui permet de déterminer la méthode à exécuter lors de son invocation sur un objet, *au moment de l'exécution* du programme et non au moment de la compilation.

Instruction	Valide (O/N)	Méthode redéfinie (Oui/Non)	Liaison Dynamique	Polymorphisme appliqué (Oui/Non)	Classe de la méthode invoquée
p1.deplacer(p2)					
p1.deplacer(p3)					
p3.deplacer(p1)					
p3.deplacer(p4)					
p4.deplacer(2)					
P4.deplacer(p1) ;					
p4.deplacer(p3);					
p1.symetrie() ;					
p3.symetrie() ;					
p4.symetrie ();					

Exercice 5

On dispose d'une interface **TAB** définie par la donnée de la signature d'une méthode Fusion destinée à effectuer la fusion de deux vecteurs d'entiers.

```
public interface TAB
{
  Public void Fusion (VECT V2, VECT V);
} // fin de l'interface TAB
```

On dispose aussi d'une classe **VECT** implémentée comme suit :

```
public class VECT
{ int t[ ] ; int taille ;
   VECT (int n) {taille = n ; t = new int [taille];} /* création du vecteur */
 void Saisir()
    {java.util.Scanner e = new java.util. Scanner (System.in);
     for (i=0 ; i<taille ; i++) t[i] = e.nextInt(); }
void Afficher()
    {for (i=0 ; i<taille ; i++) ; System.out.print (t[i]+" ");}
} // fin de la classe VECT
```

Questions

1. Redéfinir les méthodes toString et equals dans la classe VECT
2. A partir de l'interface TAB et de la classe VECT, donner l'implémentation d'une classe **VectOrdonné** permettant de créer, saisir, afficher un vecteur trié et fusionner deux vecteurs triés dans l'ordre croissant.
3. Ecrire le programme qui crée deux vecteurs triés, les fusionne et affiche le résultat de la fusion.

Exercice 6

1. On veut définir une classe abstraite **Liste**, sous forme d'une liste chainée d'objets de type Complexe représentant les nombres complexes)
   La classe comporte deux méthodes abstraites *ajouter (Complexe x)* et *supprimer (Complexe x)* et une méthode définie *affiche()* qui affiche les éléments de la liste en appelant la méthode toString() (à redéfinir).
2. Ajouter à la classe, une méthode *remplir (int n)* qui remplit la liste de n éléments quelconques
3. Ecrire une méthode *remove (int n)* qui supprime les n premiers éléments de la liste.
4. A partir de la classe Liste, définir deux sous-classes Pile et File (définir les méthodes nécessaires).
   Rappel : une pile est une structure de type LIFO et file est une structure de type FIFO.