Durée : 1h30

Examen du module POO

Exercice 1 (14 pts)

Dans cet exercice, on s'intéresse à la gestion des tickets de caisse concernant les achats de clients dans un supermarché. Le supermarché propose un ensemble d'articles aux clients. Un article est décrit par une *référence* (String) et un *prix unitaire*.

1. Donner l'implémentation d'une classe **Article** comportant une méthode ***Lire*** pour la saisie au clavier et une méthode ***Afficher***.

On s'intéresse aux achats des clients dans le supermarché. Un client peut acheter plusieurs articles avec des quantités différentes. On appelle **LigneAchat** la description de l'achat d'un article donné, Une LigneAchat est décrite par l'*article* (référence et prix unitaire) acheté, et la *quantité* achetée de l'article.

2. Donner l'implémentation de la classe **LigneAchat** comportant une méthode ***Lire*** pour la saisie au clavier, une méthode ***SommeAchat*** qui calcule le prix de revient de la quantité achetée et une méthode ***Afficher***.

Maintenant, on s'intéresse au ticket de caisse délivré par le caissier au client après paiement des ses achats, on l'appelle **TicketAchat**. Un TicketAchat comporte une *date* (la date du jour, réinitialisée chaque jour), un *numéro de ticket* séquentiel, une ou plusieurs *LigneAchat* et affiche la somme totale des achats.

3. On suppose que chaque client achète exactement 5 articles différents avec les quantités voulues. Donner l'implémentation de la classe **TicketAchat** dans ce cas, en définissant un ***constructeur***, une méthode ***InitDate*** qui initialise la date, une méthode ***InitNumero*** qui initialise à 1 le numéro de ticket, une méthode ***Lire*** pour la saisie des LigneAchat au clavier, une méthode ***Total*** qui calcule la somme totale des achats et une méthode ***Afficher*** qui affiche le ticket à délivrer au client avec les détails de ses achats.

4. A partir de la classe TicketAchat, implémenter une classe **TicketAmélioré** qui permet de donner la *recette* (somme d'argent globale) récoltée dans la caisse en fin de journée.

5. Ecrire le programme qui crée 500 tickets de caisse (TicketAchat) et affiche la recette récoltée en fin de journée.

6. En supposant que le nombre d'articles achetés diffère d'un client à l'autre et n'est pas fixé ni limité, donner une autre implémentation de la classe TicketAchat (de la question 3). Ne pas réécrire les méthodes InitDate et InitNumero.

Exercice 2 (6 pts)

On dispose d'une interface **I** définie par la donnée des signatures de deux méthodes **Add** et **Remove**, respectivement pour l'ajout et la suppression d'un élément dans une structure (vecteur par exemple).

Interface **I**

```
{ Public void Add ( int X) ;
  Public int Remove (); }   // fin de l'interface TAB I
```

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

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

Questions

1. A partir de l'interface **I** et de la classe **VECT**, donner l'implémentation d'une classe **Pile** qui définit une pile d'entiers implémentée par un vecteur.

2. A partir de la classe **Pile**, implémenter une classe **PileTriée** (par ordre croissant).

Indications pour l'exercice 1

1. Pour chaque classe sans constructeur, on utilise le **constructeur fourni par défaut**.
2. La classe Date est supposée prédéfinie avec un constructeur **Date (int, int, int)** et une méthode **Afficher()**.
3. **Java.util.Scanner** : classe pour la lecture de données au clavier
4. **Java.util.Vector** : classe pour la création d'un vecteur de taille variable et illimitée
5. **add(object obj) :** pour ajouter un élément obj à une structure de type Vector
6. **elementAt (i)** : donne l'élément d'indice i dans l'objet de type Vector
7. **Size() :** donne le nombre d'éléments d'un objet de type Vector

Bon Courage

Corrigé de l'examen

Exercice 1(1,5 + 2,5 + 4+ 3+ 2+2)

```
import java.util.Scanner; import java.util.Vector;
class Article {String reference ; float prixunit ;
void Lire () { Scanner e = new Scanner (System.in);
                reference = e.next();  prixunit= e.nextFloat();}
void Afficher() {System.out.println (reference +"   " + prixunit);} }          // fin Article
```

```
class LigneAchat { Article Art  = new Article(); int quantite;
void Lire () { Scanner e = new Scanner (System.in);
                Art.Lire();  quantite= e.nextInt();}
float SommeAchat ()     {return ( Art.prixunit * quantite) ;}
void Afficher() { Art. Afficher(); System.out.println ("   " + quantite +
this.SommeAchat());} }      // fin
```

```
class TicketAchat { static int numero ; static Date date ;
                        LigneAchat   Achats [] = new  LigneAchat [5];       // tableau de 5
LignesAchats
TicketAchat ()       // Constructeur
      { numero++ ;          // pour chaque ticket créé le numéro séquentiel est incrémenté
       for (i= 0 ; i<5 ; i++) Achats[i] = new LigneAchat(); }  // créer les 5 objets
LignesAchat
                    // méthodes
  static void InitNumero () {numero =1;}
  static void InitDate () { date = new Date (31, 5, 2011); }       // la date est fixe pour tous
les tickets de la
                                                                          journée
void Lire () { for (i= 0 ; i<5 ; i++)   Achats[i].Lire(); }        // invoquer la méthode Lire
de la classe
                                                                       LigneAchat
 float Total ()
       {  float t = 0 ; for (i= 0 ; i<5 ; i++)   {t = t +Achats[i].SommeAchat() };
return ( t) ;}
  void Afficher() { System.out.println (numero ;)  date. Afficher();
            for (i= 0 ; i<5 ; i++)   {Achats[i].Afficher();
        System.out.println ( Achat[i].SommeAchat());}        // on affiche les sommes des
LigneAchat
        System.out.println (this.Total() );  } // on affiche la somme globale      }      // fin
TicketAchat
```

```
class TicketAmélioré extends TicketAchat
{ static float Recette ;
  TicketAmélioré ()       // constructeur
     { super () ; }                  // appel du constructeur de la superclasse

static void InitRecette () {Recette =0;}        // initialiser la recette de la journée
```

```
 void CalculRecette ()                    // mettre à jour la recette de la journée après
chaque ticket
     { Recette = Recette + super.Total()}      // on ajoute la somme récoltée pour chaque
TicketAchat
static void Afficher() { System.out.println ("La recette est:" + Recette) }     }      //
fin TicketAmélioré
```

```
class ProgramAchat
{public static void main (String args[])
   { TicketAmelioré T [] = new TicketAmelioré [500];
   TicketAchat. InitNumero(); TicketAchat.InitDate();  TicketAmelioé.InitRecette();
//  initialisations
    for (i= 0 ; i<5 ; i++) { T[i] = new TicketAmélioré();              //Création des
tickets
                          T[i].Lire();              // saisir les infos d'un ticket
                            T[i].CalculRecette() }       // on ajoute le total de chaque
ticket à la recette
   TicketAmélioré.Afficher() ;  }       // on affiche la recette des 500 tickets    }    // fin
Program
```

```
class TicketAchat { static int numero ; static Date date ;
 Vector  <LigneAchat>  A  = new  Vector <LigneAchat>;      // vecteur de type
                                                          LigneAchat de taille
variable et non limitée.
TicketAchat ()                  // Constructeur
{ numero++ ;  }              // pour chaque ticket créé le numéro séquentiel est incrémenté
                  // méthodes
     void Lire () { LigneAchat  La = new LigneAchat () ; La.Lire();  A.add(La) ;}
                                                    // Créer une ligne Achat  La et
l'ajouter au vecteur A
  float Total () {  float t = 0 ; for (i= 0 ; i<A.size() ; i++)
                                  {t = t +A.elementAt(i).SommeAchat() };   return (
t) ;}
   void Afficher() { System.out.println (numero ;)  date. Afficher();
           for (i= 0 ; i<A.size() ; i++)   {A.elementAt(i).Afficher();
       System.out.println ( A.elementAt(i).SommeAchat());}          // on affiche la
somme de chaque
                                                                    LigneAchat
       System.out.println (this.Total() ); }  // on affiche la somme globale      } // fin
TicketAchat
```

Exercice 2 (3+3)

```
class Pile implements I extends VECT
     { int sommet;
      Pile (int n) { super(n); }       // appel au constructeur de la superclasse
       void InitPile () {sommet = -1;}   boolean Pilevide () { return (sommet = =
-1);}
```

```
    int SommetPile () { return (t[sommet]);}
                    // on doit implémenter les méthodes de l'interface
    public void Add(int  x)  { sommet++; t[sommet] =  x ;}  // empiler
    public int Remove ()  { int  x = t[sommet]; sommet - -; return(x); }  // dépiler   }
// fin Pile
```

```
class PileTriee extends Pile
{ PileTriee (int n) { super(n) ;}          // appel au constructeur de la superclasse
   // on doit redéfinir les méthodes Add et Saisir de la classe VECT car les données doivent
être
      stockées dans l'ordre croissant
  public void Add(int  x)          // cette méthode Add ajoute un élément dans une pile
triée
    { Pile S = new Pile (100);      // on crée une pile intermédiaire S pour dépiler les
éléments
      int y ;  while (! super.Pilevide() && super.Sommetpile() < x)
                    { y = super.Remove() ; S.Add(y) ; }      // on appelle la méthode
Add de la classe
                                                              Pile car S est de type
Pile
      super.Add (x)       // empiler la valeur  x  et  remettre les éléments
      while (! S.Pilevide()) { y = S.Remove() ; super.Add(y) ; }}
```

```
    void Saisir ( ) { Scanner e = new Scanner (System.in);   int i, x;
                  for (i=0; i<n; i++)  { x = e.nextInt();  this.Add(x); } }     // fin
PileTriee
                                          // on appelle la méthode Add  redéfinie pour avoir les
éléments dans l'ordre
```