Université Hassan 1er
École Nationale des Sciences Appliquées – Berrechid

MANUEL DES EXERCICES CORRIGÉS ET ANNALS DES EXAMENS

PROGRAMMATION ORIENTÉ OBJETS JAVA

Professeur : MOUMOUN Lahcen

Sommaire

Introduction

Le langage Java est aujourd'hui l'un des langages les plus utilisés et les plus polyvalents dans le monde du développement logiciel. Sa capacité à fonctionner sur une multitude de plateformes, combinée à sa robustesse et à sa flexibilité, en fait un choix privilégié pour de nombreux développeurs. De la création d'applications mobiles aux systèmes d'entreprise, en passant par les applications web, Java s'est imposé comme un standard incontournable.

Ce manuel d'exercices corrigés et d'annales d'examens a été conçu pour accompagner les étudiants dans leur préparation académique. Il leur offre une sélection de sujets variés, inspirés des examens passés, ainsi que des éléments de corrections pour les guider dans leur apprentissage.

L'objectif principal de ce manuel est de permettre aux étudiants de s'entraîner efficacement, en leur fournissant non seulement des exercices typiques, mais aussi des corrigés qui peuvent être exécutés avec un IDE JAVA. Chaque exercice a été soigneusement sélectionné pour couvrir les points essentiels du programme, tout en offrant une progression adaptée et un renforcement des compétences des étudiants.

Les annales d'examens permettent aux étudiants de se familiariser avec le format des épreuves et de tester leurs connaissances dans des conditions réelles. En s’exerçant sur ces sujets, les étudiants pourront non seulement mieux comprendre les attentes des examens, mais aussi développer des stratégies pour gérer leur temps et améliorer leur performance.

Chapitre 1 : Eléments de base du langage JAVA

Exercice 1.1.

Énoncé

Pour calculer le produit de 2 entiers positifs, on propose la multiplication Egyptienne qui consiste à n'utiliser que des additions, multiplications par 2 et divisions par 2 (division entière).

Pour calculer "n = B x C" :

- On ajoute la valeur de B dans n (n est initialisé à 0) si C est impair (ne rien faire si C est pair)
- on divise (division entière) C par 2
- on multiplie B par 2
- et on répète ces 3 opérations jusqu'à ce que C devienne égal à 0
- Lorsque C est nul, dans "n" nous avons le résultat du produit recherché.

Ecrivez un programme JAVA qui calcul et qui affiche le produit de deux entiers positifs en utilisant la multiplication Egyptienne.

Élément de correction

```
public class Ex1 {
    public static void main(String[] args) {
     long b,c,n=0;
     b=3;c=5;
     System.out.print("B * C="+b + " * "+c + " = " );
     while (c!=0) {
              if (c%2!=0)  n+=b;
              c/=2;
              b*=2;
     }
     System.out.println(n );
     }
}
```

Exercice 1.2.

Énoncé

On désire un programme JAVA qui calcul la somme des chiffres d'un nombre entier strictement positif N et recommence le calcul avec le résultat obtenu tant que celui-ci n'est pas compris entre 1 et 9. Après chaque calcul il faut afficher à l'écran la somme obtenue. À la fin de ce processus il faut afficher le nombre compris entre 1 et 9, ainsi, obtenu.

Exemple : Si le nombre N considéré est 123456, On affichera

21 (car 1+2+3+4+5+6=21)
3 (car 2+1=3)
Le nombre obtenu est 3.

Indication : le dernier chiffre décimal d'un nombre entier positif n est n mod 10. Le nombre obtenu en amputant un nombre entier positif n de son dernier chiffre décimal est n div 10 (division entière).

Élément de correction

```
public class Ex2 {
public static void main(String[] args) {
    int n=78192, n2=n, som=0,ch,nb;
    for(nb=0;n2!=0;n2/=10,nb++);
    n2=n;

    for( ; nb>=1; nb--  ){

     ch=n2 / (int) Math.pow(10,nb-1);
     n2= n2 % (int) Math.pow(10,nb-1 );
     som+=ch;
     if (som>=9)som-=9;
     }

    if (som==0)
      System.out.println("le nombre "+n+ " est divisible par 9");
    else
      System.out.println("le nombre "+n+ " n'est pas divisible 9");
}
}
```

Exercice 1.3.

Énoncé

Écrire un programme JAVA qui affiche, pour deux valeurs numériques x et p, la racine carrée du nombre x avec une précision p. On utilisera pour cela une méthode par dichotomie : à la k-ème itération, on cherche x dans l'intervalle [min, sup], on calcule le milieu m de cet intervalle (à vous de trouver comment la calculer). Si l'étendue de cet intervalle (la valeur sup - min) est inférieure $2 \times p$, afficher m. Sinon, vérifiez si la racine se trouve dans [inf, m] ou dans [m, sup], et modifiez les variables inf et sup en conséquence.

Par exemple, calculons la racine carrée de 10 avec une précision 0.5 :

- Commençons par la chercher dans [0, 10], on a m=5, comme $5^2 > 10$, alors $5 > \sqrt{10}$, donc se trouve dans l'intervalle [0, 5].
- On recommence, m= 5/2=2.5, comme $(5/2)^2 < 10$, alors $5/2 < \sqrt{10}$, on poursuit la recherche dans [5/2 , 5]
- On a m = 3.75, comme $3.75^2 > 10$, alors $3.75 > \sqrt{10}$ et $\sqrt{10} \in [2.5,\ 3.75]$
- On a m = 3.125, comme $3.1252 < 10$, alors $3.125 < \sqrt{10}$ et $\sqrt{10} \in [3.125,3.75]$
- Comme l'étendue de l'intervalle [3.125, 3.75] est inférieure 2×0.5, alors m=3.4375 est une approximation à 0.5 près de $\sqrt{10}$.

Élément de correction

```
public class Ex3 {
public static void main(String[] args) {
    float x=2F, p=0.001F;
    float min,max,m;
    min=0; max=x;
    do {
     m=(max+min)/2;
     if (m*m > x) { max=m; }
     else {   min=m;}
    }while ((max-min)>2*p);
    System.out.println("Racine de " + x + " est :" + Math.sqrt(x));
    System.out.println("Racine calculée  de " + x + " est :" + m);

}
}
```

Exercice 1.4.

Énoncé

On peut déterminer si un nombre est divisible par 9 par la méthode suivante : On part du premier chiffre ou de zéro si ce chiffre est 9. On ajoute le deuxième chiffre (s'il y en a un), si le résultat est supérieur ou égal à 9, on lui soustrait 9 sinon on ne fait rien. On répète ensuite la même opération pour les chiffres suivants. Le nombre est divisible par 9 si et seulement si le résultat final est nul. Écrire un programme JAVA qui détermine les chiffres d'un nombre N et qui indique si le nombre N est divisible par 9 en mettant en œuvre la méthode décrite ci-dessus. On considère que le nombre N est un entier positif composé au maximum de 5 chiffres.

Exemple : Pour le nombre N = 78192, l'algorithme effectuera les opérations suivantes :

- Déterminer les chiffres de N: 7, 8, 1, 9 et 2.
- Vérifier si N est divisible par 9 :
  - 7 est différent de 9 ; 7 + 8 = 15, 15 est supérieur ou égal à 9 on lui soustrait 9 pour obtenir 6
  - 6 + 1 = 7, 7 est strictement inférieur à 9
  - 7 + 9 = 16, 16 est supérieur ou égal à 9 on lui soustrait 9 pour obtenir 7
  - 7 + 2 = 9, 9 est supérieur ou égal à 9 on lui soustrait 9 pour obtenir 0
  - Le résultat est nul donc 78192 est divisible par 9 (78192 = 9 x 8688).

Élément de correction

```
public class EX4 {
    public static void main(String[] args) {
     int n,n2;
     n=n2=787652;
     int nbCh=0;

     //while (n2!=0) { n2/=10;nbCh++;}
     for(; n2!=0;  nbCh++) n2/=10;
     int som=0;
     n2=n;
     for (int p=nbCh-1; p>=0; p--) {
            som=som + ( n2 / (int) Math.pow(10, p) );
            if (som>=9)som-=9;

            n2=n2 % (int) Math.pow(10, p);
     }
```

```
        if (som==0) System.out.println("le nombre " + n + " et divisible par 9");
        else System.out.println("le nombre " + n + " et non divisible par 9");

        }
    }
```

Exercice 1.5.

Énoncé

Dans cet exercice nous rappelons les notions suivantes :

- On appelle nombre premier tout entier naturel supérieur à 1 qui possède exactement deux diviseurs, lui-même et l'unité ;
- On appelle diviseur propre de n, un diviseur quelconque de n, n exclu ;
- Un entier naturel est dit parfait s'il est égal à la somme de tous ses diviseurs propres;
- Les nombres n tels que : $(n + a + a^2)$ est premier pour tout a tel que $0 \leq a < (an - 1)$, sont appelés nombres chanceux.

Écrire le code permettant de définir les quatre fonctions : somDiv, estParfait, estPremier et estChanceux :

- La fonction somDiv retourne la somme des diviseurs propres de son argument ;
- Les trois autres fonctions vérifient la propriété donnée par leur définition et retourne un booléen. Plus précisément, si par exemple la fonction estPremier vérifie que son argument est premier, elle retourne True, sinon elle retourne False.

Écrire un programme principal qui comporte une boucle de parcours de l'intervalle [2, 1000] incluant les tests nécessaires pour afficher les nombres parfaits et chanceux.

Élément de correction

```
public class Ex2 {
public static int somDiv(int n) {
        int som=1;
        for( int i=2; i<n; i++) {
                if (n % i == 0 ) {
```

```
                    som+=i; //System.out.print(i+ " ");
                }
        }
        return som;
    }
    public static boolean estParfait(int n) {
        if (somDiv(n)==n) return true;
        return false;
    }
    public static boolean estPremier(int n) {
        if (somDiv(n)==1) return true;
        return false;
    }
    public static boolean estChanceux(int a) {
        int b;
        for(int n=0; n<a-1; n++) {
            b=a+n+n*n;
            if (estPremier(b)==false) return false;
        }
        return true;
    }
    public static void main(String[] args) {
        String strParfait="Les nombres pafaits sont  :";
        String strChanceux="Les nombres Chanceux sont  :";

        for( int i=2; i<=1000; i++) {
            if (estParfait(i))  strParfait += i + "  ";
            if (estChanceux(i)) strChanceux += i + "  ";
        }
        System.out.println(strParfait+ "\n"+ strChanceux);
    }
}
```

Exercice 1.6.

Énoncé

Soit à un entier strictement supérieur à 1. La suite $(U_n)_{n\in\mathbb{N}}$ définie par :

$$\begin{cases} U_0 = a \\ U_{n+1} = \frac{1}{2}\left(U_n + \frac{a}{U_n}\right) \end{cases}$$

S'appelle la suite de Héron (d'Alexandrie). Cette suite est très célébre car elle a le mérite de converger vers $\sqrt{a}$ très rapidement.

Ecrire une méthode alternative puis récursive qui retourne la racine carrée d'un réel (reçu en paramètre) en se basant sur la suite de Héron

Élément de correction

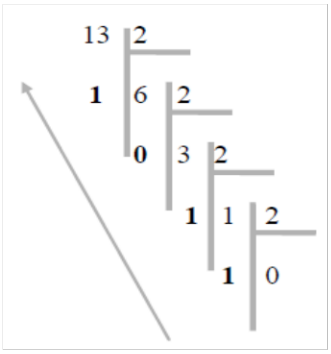
```
public class Ex3 {
  public static float heron(float a, float p) {
        float uc, up=a;
        do {
              uc= 1F/2*(up+a/up);
              if (Math.abs(uc-up)< p)  break;
              up=uc;
        } while (true);
        return uc;}
  public static float heron2(float a, float p, float u) {
        float uc=(1F/2*(u+a/u));
        if (Math.abs( u- uc) <p    ) return uc;
        return heron2(a,p,uc);
  }
  public static void main(String[] args) {
        float a=2F, p=0.01F;
        System.out.println("Racine réel de "+a+" est "+ Math.sqrt(a));
        System.out.println("Racine calculé de "+a+" est "+     heron2(a, p, a));
}    }
```

Exercice 1.7.

On veut convertir un nombre décimal en binaire selon la méthode suivante:

- Tant que le nombre décimal est différent de zéro, on le divise par deux et on retien le reste de la division.
- On inverse les élément retenus.

Exemple: le nombre 13 vaut 1101 en binaire



Exercice 1.8.

On considère la fonction f : IN∗ → IN∗ définie par : f(n) = n/2 si n est pair et f(n)=3n+ 1 sinon

La suite de Syracuse est définie, par récurrence, par un premier terme u_0 donné (entier strictement positif) et par la relation $u_{n+1} = f(u_n)$ ($n \in IN$). Cette suite a la particularité de toujours "passer" par la valeur 1

1. Ecrivez le code de la fonction génératrice de cette suite, et une fonction principale permettant de la tester.
2. On souhaite calculer des termes de la suite (en fonction du premier terme saisi par l'utilisateur). Transformez la méthode principale afin de calculer :

- 10 termes de la suite
- Des termes à volonté, un par un (on demande à l'utilisateur s'il souhaite continuer)
- 1 terme, puis [si l'utilisateur souhaite continuer] 2 nouveaux termes, puis [...] 3 nouveaux termes, et ainsi de suite en arrêtant lorsque l'utilisateur le souhaite
- Tous les termes consécutifs, en arrêtant lorsqu'on rencontre la valeur 1.

Complétez le code pour que le programme calcule et affiche le nombre de termes calculés, la valeur maximale rencontrée, ainsi que la moyenne des termes.

Chapitre 2 : Notion de classes et objets en JAVA

Exercice 2.1.

Énoncé

Réaliser une classe qui permet d'attribuer un numéro unique à chaque nouvel objet créé (1 au premier, 2 au suivant...). On ne cherchera pas à réutiliser les numéros d'objets éventuellement détruits. On dotera la classe uniquement d'un constructeur, d'une méthode getIdent fournissant le numéro attribué à l'objet et d'une méthode getIdentMax fournissant le numéro du dernier objet créé.

Écrire un petit programme d'essai.

Élément de correction

```
public class Ident {
    public Ident(){
     nbIdent++; id=nbIdent;        }
    public int getId(){      return id;        }
    public static int getMaxId(){  return nbIdent;          }
    private int id;
    private static int nbIdent=0;
}
public class testIdent {
 public static void main(String[] args) {
    Ident i1,i2,i3;
     i1=new Ident();
    i2=new Ident();  i3=new Ident();
    System.out.println(i1.getId());
    System.out.println(i2.getId());
    System.out.println(i3.getId());
    System.out.println(Ident.getMaxId());
}
}
```

Exercice 2.2.

Énoncé

On souhaite disposer d'une classe permettant d'effectuer des conversions (dans les deux sens) entre nombre sexagésimaux (durée exprimée en heures, minutes, secondes) et des nombres décimaux (durée exprimée en heures décimales). Pour ce faire, on réalisera une classe permettant de représenter une durée. Elle comportera :

- un constructeur recevant trois arguments de type int représentant une valeurs sexagésimale (heures, minutes, secondes) qu'on supposera normalisée (secondes et minutes entre 0 et 59). Aucune limitation ne portera sur les heures ;
- un constructeur recevant un argument de type double représentant une durée en heures ;
- une méthode getDec fournissant la valeur en heures décimales associée à l'objet,
- des méthodes getH, getM et getS fournissant les trois composantes du nombre sexagésimal associé à l'objet.

On proposera deux solutions :

1. Avec un champ (privé) représentant la valeur décimale,
2. Avec des champs (privés) représentant la valeur sexagésimale.

Élément de correction

1. Avec un champ (privé) représentant la valeur décimale,

```
class SexDec {
    public SexDec (double dec) { this.dec = dec ; }
    public SexDec (int h, int mn, int s) {
            dec = h + mn/60. + s/3600. ; }
    public double getDec() { return dec ; }
    public int getH() { int h = (int)dec ; return h ;}
    public int getM() {
            int h = (int)dec ;
            int mn = (int)(60*(dec-h)) ;
            return mn ;}
```

```
public int getS(){
    int h = (int)dec ;
    double minDec = 60*(dec-h) ;
    int mn = (int)minDec ;
    int sec = (int)(60*(minDec-mn)) ;
    return sec ;
}
private double dec ;
}
public class TSexDec1{
    public static void main (String args[]){
            SexDec h1 = new SexDec(4.51) ;
            System.out.println ("h1 - decimal = " + h1.getDec() +" Sexa = " +
                h1.getH() + " " + h1.getM() + " " + h1.getS()) ;
            SexDec h2 = new SexDec (2, 32, 15) ;
            System.out.println ("h2 - decimal = " + h2.getDec() +" Sexa = " +
                h2.getH() + " " + h2.getM() + " " + h2.getS()) ;
    }
}
```

2. Avec des champs (privés) représentant la valeur sexagésimale.

```
class SexDec{
    public SexDec (double dec){
            h = (int)dec ;
            int minDec = (int)(60*(dec-h)) ;
            mn = (int)minDec ;
            s = (int)(60*(minDec-mn)) ;
    }

    public SexDec (int h, int mn, int s){
            this.h = h ; this.mn = mn ; this.s = s ;
    }
    public double getDec(){return (3600*h+60*mn+s)/3600 ;  }
```

```
    public int getH(){ return h ;}
    public int getM(){ return mn ;}
    public int getS(){ return s ;}
private int h, mn, s ;
}
```

Exercice 2.3.

Énoncé

Réaliser une classe Répertoire permettant de gérer un répertoire téléphonique associant un numéro de téléphone (chaîne de caractères) à un nom. Pour faciliter les choses, on prévoira une classe Abonne destinée à représenter un abonné et disposant des fonctionnalités indispensables.

La classe Repertoire devra disposer des fonctionnalités suivantes :

- Constructeur recevant un argument de type entier précisant le nombre maximum d'abonnés que pourra contenir le répertoire (cette particularité évite d'avoir à se soucier d'une gestion dynamique du répertoire),
- Méthode addAbonne permettant d'ajouter un nouvel abonné ; elle renverra la valeur false si le répertoire est plein, la valeur true sinon,
- Méthode getNumero fournissant le numéro associé à un nom d'abonné fourni en argument,
- Méthode getNAbonnes qui fournit le nombre d'abonnés figurant dans le répertoire,
- Méthode getAbonne fournissant l'abonné dont le rang est fourni en argument,
- Méthode getAbonnesTries fournissant un tableau des références des différents abonnés, rangés par ordre alphabétique (pour simplifier, on supposera que les noms sont écrits en minuscules, sans caractères accentués).
- Écrire un petit programme de test.

Élément de correction

```
public class Abonne {
    public Abonne(String nom, String tel) { this.nom = nom;  this.tel = tel;}
    public String getNom() {       return nom;     }
    public void setNom(String nom) {     this.nom = nom;         }
    public String getTel() { return tel;     }
    public void setTel(String tel) { this.tel = tel; }
    public String toString() {       return " [nom=" + nom + ", tel=" + tel + "]\n";  }
private String nom;
private String tel;
}
public class Repertoire {
    private Abonne tabAbonnes[];
    private int nbAbonnees=0;
    public Repertoire(int tailleMax) { this.tabAbonnes = new Abonne[tailleMax];}
    public Abonne[] getTabAbonnes() { return tabAbonnes;     }
    public Boolean addAbonnee(Abonne ab) {
            if (nbAbonnees > tabAbonnes.length) return false;
            tabAbonnes[nbAbonnees++]=ab;
            return true;
    }
    public Boolean addAbonnee(String nom, String tel) {
            return addAbonnee(new Abonne(nom, tel));
    }
    public Boolean updateAbonnee (String nomModif, String NouvTel) {
            for(int i=0; i<nbAbonnees; i++) {
                    if (tabAbonnes[i].getNom().equals(nomModif)) {
                            tabAbonnes[i].setTel(NouvTel);
                            return true;
                    }
            }
            return false;
    }
```

```
    public Boolean delAbonne(String nomSup) {
        int pos=-1;
        for(int i=0; i<nbAbonnees; i++) {
            if (tabAbonnes[i].getNom().equals(nomSup)) {pos=i;}
        }
        if (pos==-1) return false;
        for(int i =pos; i<nbAbonnees-1; i++)tabAbonnes[i]=tabAbonnes[i+1];
        tabAbonnes[nbAbonnees-1]=null;
        nbAbonnees--;
        return true;
    }
    public String getNumeroTel(String nomRech) {
        for(int i=0; i<nbAbonnees; i++) {
            if (tabAbonnes[i].getNom().equals(nomRech)) {
                return tabAbonnes[i].getTel();          }
        }
        return null;
    }
    public int getNgetNAbonnes() {return nbAbonnees;}
    public String toString() {
        String str="";
        for (int i=0; i< nbAbonnees; i++) {str+= tabAbonnes[i].toString();}
        return str;
    }
}
```

Exercice 2.4.

Énoncé

Pour la gestion d'une bibliothèque on nous demande d'écrire une application traitant des documents

Dans cet exercice, nous allons modéliser le fonctionnement simplifié d'une banque. Cela nous permet de faire interagir plusieurs types d'objet différents: un compte, un client et une banque.

La classe Compte : On suppose que la classe « Compte » contient deux attributs privés: son numéro et son solde, ainsi que quatre méthodes publiques:

- void depot(float valeur); /* pour faire un dépot sur le compte. */
- void retrait(float valeur); /* pour faire un retrait sur le compte. */
- float getSolde(); /* pour obtenir la valeur du solde */
- void afficherSolde(); /* pour afficher le solde */

1. Définir la classe « Compte » puis donner un code pour la tester (écrire une classe de test possédant la méthode main).
2. Considérons une méthode qui aura pour effet de faire un virement vers un autre compte. Cette méthode aura deux arguments: la somme à déplacer, et le compte destinataire: void virer(float valeur, Compte destinataire).

Definir la méthode « virer » de la classe « Compte » puis donner le code permettant de tester son fonctionnement (prévoir des virements entre plusieurs comptes).

La classe Client : Vous devez maintenant écrire une classe « Client » qui possède deux attributs privés: son nom et son compte (chaque objet client doit donc avoir un objet compte comme attribut). La classe « Client » doit également posséder trois méthodes: « getNom », qui renvoie le nom du client, « getSolde », qui renvoie le solde du compte du client, « afficherSolde », qui affiche le solde du client. Prévoir un constructeur permettant de choisir le nom de l'objet client à créer.

1. Définir une première version de la classe « Client » et donner le code de son teste .
2. Nous allons maintenant permettre à chaque client d'avoir plusieurs comptes. Pour cela, au lieu d'avoir un attribut compte, nous allons avoir un attribut comptes, qui sera un tableau contenant plusieurs comptes. Nous considérons que le nombre maximum de comptes associé à un client est 3. Vous devez ajouter à la classe « Client », en plus de l'attribut comptes, l'attribut nbComptes qui contient le nombre de compte du client la méthode « ajouterCompte » qui ajoute un nouveau compte. Vous devez également modifier la méthode « getSolde » pour qu'elle renvoie la somme du solde de tous les comptes du client. Donner la nouvelle classe « Client ».

La classe banque : De même qu'un client peut posséder plusieurs comptes, une banque peut contenir plusieurs clients. On considère une classe « Banque », possédant entre autres les méthodes ajouterClient(String nomduclient), bilanClient(int numeroduclient) qui affiche le bilan de tous les comptes d'un client, et afficherBilan() qui affiche un bilan général de tous les compte de tous les clients. Définir la classe « Banque » et écrire un programme pour la tester

Élément de correction

La classe Compte

```
public class Compte {
        private double solde = 0;
        public void depot(double valeur) {
            if (valeur>0)   solde=solde + valeur;
        }
        public void retrait(double valeur) {
            if (valeur>0)   solde=solde - valeur;
        }
        public double getSolde() {return solde;  }
        public void afficherSolde() {
            System.out.println("Le solde du compte est de "+solde+"¤");
        }
        public void virer(double value,Compte cpt) {
            if (value>0) {
                    cpt.depot(value); retrait(value);
            }
        }
}
```

La classe Client

```
public class Client  {
Scanner sc=new Scanner(System.in);
private Compte [] comptes=new Compte[100];
private int nbComptes=0; private String nom;
public Client(String s) { nom=s; ajouterCompte(); }
public void ajouterCompte(){nbComptes+=1; comptes[nbComptes]=new Compte();}
```

```
public double soldeTotal(){
    double x=0;
    for (int i=1;i<=nbComptes;i++) x=x+comptes[i].getSolde();
    return x;}
public void afficherSolde() {
    System.out.println("Le solde de l'utilisateur est de "+soldeTotal()+"¤");}
public void afficherBilan(){
    System.out.println("Bilan des comptes de M. ou Mme "+nom);
    for (int i=1;i<=nbComptes;i++)
     System.out.println(" Le solde du compte n°"+i+" est de "+comptes[i].getSolde());
    System.out.println();  }
public String getNom() { return nom;}
public Compte getCompte(int numero) { return comptes[numero];}
public void interaction() {
    boolean fini=false;
    while(!fini) {
      System.out.println("Quelle operation voulez-vous effectuer sur le client "+nom);
      System.out.println(" 1) Faire un dépot");
      System.out.println(" 2) Faire un retrait");
      System.out.println(" 3) Faire un virement");
      System.out.println(" 4) Créer un compte");
      System.out.println(" 5) Afficher le bilan des comptes");
      System.out.println(" 6) Revenir au menu princial");
    System.out.println("Votre choix:");   int reponse=sc.nextInt();
    switch (reponse) {
    case 1:
            System.out.println("De quel montant?");
            double val = sc.nextDouble();  comptes[1].depot(val);
            System.out.println("Le depot a été effectué\n");
            break;
    case 2:
            double val1=sc.nextDouble(); comptes[1].retrait(val1);
            System.out.println("Le retrait a été effectué\n");      break;
```

```
        case 3:
                System.out.println("De quel montant?");
                double montant=sc.nextDouble();
                System.out.println("Compte emetteur? ");    int emetteur = sc.nextInt();
                System.out.println("Compte destinataire?"); int destinataire = sc.nextInt();
                if ((emetteur<=nbComptes) && (destinataire<nbComptes)) {
                        comptes[emetteur].virer(montant,comptes[destinataire]);
                        System.out.println("Le virement a étté effectué\n");
                }
                else {  System.out.println("Un des comptes n'existe pas\n");}
                break;
        case 4:
                ajouterCompte();
                System.out.println("Le compte n°"+nbComptes+" a été créé\n");
                break;
        case 5:
                afficherBilan();  break;
        case 6:
                fini=true;
        }
    }
}
}
```

La classe banque

```
    import java.util.Scanner;
    public class Banque {
        Scanner sc=new Scanner(System.in);
        private int nbClients=0;  private Client [] clients=new Client[100];
        public void ajouterClient(String nom) {
                nbClients+=1;   clients[nbClients]=new Client(nom); }
        public void afficherBilan() {
                for (int i=1;i<=nbClients;i++)  clients[i].afficherBilan();
                System.out.println(); }
```

```
    public void interaction() {
        boolean fini=false;
        while (!fini) {
            System.out.println("Quelle operation voulez-vous effectuer?");
            System.out.println(" 1) Ajouter un client");
            System.out.println(" 2) Afficher le bilan de la banque");
            System.out.println(" 3) Effectuer une operation sur un client");
            System.out.println(" 4) Quitter le programme");
            System.out.println("Votre choix:"); int reponse=sc.nextInt();
            switch(reponse){
            case 1:
                System.out.println("Entrez le nom du client: ");
                String nom=sc.nextLine();
                ajouterClient(nom); System.out.println("Le client a été ajouté\n");
                break;
            case 2:
                afficherBilan(); break;
            case 3:
               System.out.println("Choisissez le client:");
               for (int i=1;i<=nbClients;i++)
               System.out.println(" "+i+") "+clients[i].getNom());
               System.out.println("Votre choix:"); int numero=sc.nextInt();
               clients[numero].interaction();  break;
    case 4:
        fini=true;}
  }}
}
```

Test classe banque

```
  public class TestBanque {
    public static void main(String[] args) {
            Banque maBanque=new Banque();
            maBanque.interaction();   }
  }
```

Exercice 2.5.

Énoncé

Nous souhaitons réaliser une solution de gestion d'un ensemble d'agences de location de voitures. Les classes principales de ce programme sont : la classe **Voiture**, la classe **Client**, la classe **Location** et la classe **Agence**.

1. On considère que les voitures et les clients d'une agence sont représentés par les classes suivantes (le signe « - » correspond à un attribut privé et le signe « + » correspond à une méthode publique) :

Client
- numClt :int // numéro du client avec une incrémentation automatique - nomPrenom , telephone : String - email : String **…**
+ Client (nomPrenom :String , telephone : String, email : String) + getXxx() : … // Xxx correspond à un attribut + setXxx(…) : void + toString() : String …

Voiture
- immat : String - marque : String - prixLocation : float **…**
+ Voiture (immat : String , marque: String, prixLoc: int) + getXxx() : … ; + setXxx(…) : void + toString() : String **…**

2. Chaque agence mis à la disposition de ces clients la possibilité de louer plusieurs voitures, à chaque location une instance de la classe Location est créée.

Location
- clt : Client - voiture : Voiture - DateLocation : Date - duree : int // durée de location en nombre de jours …
+ Location (clt : Client , voiture : Voiture, dateLoc : Date, duree : int) + getXxx() : … ; + setXxx(…) : void // Xxx correspond à un attribut de la classe + toString() : String …

Nous considérons que chaque agence gère séparément ses locations et que l'ensemble des voitures et les clients d'une agence sont stocker séparément dans des tableaux de la classe Agence. Cette dernière comporte également un tableau regroupant les différentes locations de l'agence. L'agence offre à ces clients la possibilité de choisir les voitures à

louer en fonction du critère de marque ou de prix. Il est, ainsi, possible de sélectionner la voiture à louer dans la liste des voitures d'une marque donnée ou dans celle des voitures dont le prix de location est compris entre une valeur minimale et une valeur maximale.

Pour des raisons de simplification, nous supposons que l'historique des locations n'est pas pris en considération c.à.d. que la location est supprimée lors du retour de la voiture louée.

Agence
- nomagence : String - listClt : Client [ ] ; - nbClt : int // nombre des clients de l'agence - listVoiture : Voiture [ ] ; - nbVoiture : int // nombre des voitures de l'agence - listLocation : Location [ ] ; - nbLocation : int // nombre de locations de l'agence …
+ Agence (nomAgence : String) // création des tableaux (allocation mémoire nécessaire) + getXxx() : … // Xxx correspond à un attribut de la classe + SetNomAgence(nomAgence : String) : void + getClient(numClt : int) : Client + addClient(clt : Client) : void ; + deleteClient(numClt : int) : Boolean + getVoiture(immat : String) : Voiture + addVoiture(voiture : Voiture) : Boolean ; + deleteVoiture(immat : String) : Boolean + getListVoitureByMarq(marque : String) : Voiture [ ] + getListVoitureByPrix(minPrix : float, maxPrix : float) : Voiture [ ] + addLocation(numClt : int , immat :String) : Boolean + deleteLocation(numClt : int , immat :String) : Boolean + getLocationClient(numClt : int) : Location [ ] …

- Lors de l'ajout d'une voiture il faut s'assurer que chaque voiture possède un numéro d'immatriculation unique. La méthode à définir doit retourner « false » si l'ajout ne peut pas se faire suite à une redondance du numéro d'immatriculation.
- La méthode getClient permet de retourner le client dont le numéro est fourni en argument (la valeur null est retournée si le numéro fourni est introuvable).
- La méthode getListVoitureMarq permet de retourner un tableau de voitures qui correspondent à une marque donnée.

3. Définir une classe de test nommée « TestAgence » et qui contient la méthode « main ». Cette méthode assure les tests suivants :

- Créer une agence nommée « BerrechidVoiture »
- Ajouter 2 Clients et 3 voitures à l'agence « BerrechidVoiture » (utiliser uniquement les marques « BMW » et « AUDI »)

- Affichage des voitures qui correspondent à la marque « AUDI »
- Effectuer une location d'une voiture de la marque « AUDI » par le premier client.
- Inscrire le retour de la voiture louée et marquer la fin de cette location.

Élément de correction

Classe Client

```
public class Client {
    private int numClt; private static int nbClt;
    private String nomPrenom, telephone, email
    private Location [] listLocationClt; int nbLocationClt;
    public Client(String nomPrenom, String telephone, String email) {
            this.nomPrenom = nomPrenom; this.telephone = telephone;
            this.email = email; this.numClt=++nbClt;
            listLocationClt=new Location[100];
    }
    public String getNomPrenom() { return nomPrenom; }
    public void setNomPrenom(String nomPrenom){this.nomPrenom=nomPrenom;}
    public String getTelephone() { return telephone;}
    public void setTelephone(String telephone) {this.telephone = telephone;   }
    public String getEmail() {return email;}
    public void setEmail(String email) { this.email = email;}
    public int getNumClt() {return numClt;}

@Override
    public String toString() {
       return "Client [numClt=" + numClt + ", nomPrenom=" + nomPrenom + ",
           telephone=" + telephone + ", email=" + email          + "]";   }
    @Override
    public boolean equals(Object obj) {
            if (this == obj) return true;
            if (obj == null) return false;
            if (getClass() != obj.getClass()) return false;
            Client other = (Client) obj;
            if (email == null) {
                    if (other.email != null) return false;
            } else if (!email.equals(other.email)) return false;
            return true;
    }
    public Location[] getListLocationClt() {
            Location [] listTemp=new Location[nbLocationClt];
            for(int i =0; i<nbLocationClt;i++) listTemp[i]=listLocationClt[i];
            return listTemp; }


    public String getlocationByCltToString() {
            if (listLocationClt[0]==null) return "";
```

```
            String st= ("Location Clients \n" );
            st+=listLocationClt[0].toString();
            for (int i=1; i<nbLocationClt; i++)
                  st+= ("\n"+listLocationClt[i].toString());
            return st; }
    public boolean addLocation(Location location) {
            listLocationClt[nbLocationClt++]=location; return true;}
}
```

Classe Voiture

```
public class Voiture {
    private String immat, marque ;
    private float prixLocation;
    private Location [] listThisLocation; private int nbLocation;
    public Voiture(String immat, String marque, float prixLocation) {
          this.immat = immat;     this.marque = marque;
          this.prixLocation=prixLocation; this.listThisLocation=new Location [100];}
    public Voiture(String immat, String marque) {
           this.immat = immat;   this.marque = marque;}
    public String getImmat() {     return immat;  }
    public void setImmat(String immat) {this.immat = immat;  }
    public String getMarque() {   return marque;}
    public void setMarque(String marque) {       this.marque = marque; }
    public float getPrixLocation() {        return prixLocation;   }
    public void setPrixLocation(float prixLocation){this.prixLocation=prixLocation;}
    @Override
    public String toString() {
        return "Voiture [immat="+immat +", marque="+marque + ", prixLocation="
                + prixLocation + "]";      }
    @Override
    public boolean equals(Object obj) {
            if (this == obj)  return true;
            if (obj == null) return false;
            if (getClass() != obj.getClass()) return false;
            Voiture other = (Voiture) obj;
            if (immat == null) {
                  if (other.immat != null) return false;
            } else if (!immat.equals(other.immat)) return false;
            return true;      }
    public boolean addLocation(Location location) {
            listThisLocation[nbLocation++]=location;   return true; }
    public String getlocationToString() {
            if (listThisLocation[0]==null) return "";
            String st= ("Location de la voiture " + immat + " :\n" );
            st+=listThisLocation[0].toString();
            for (int i=1; i<nbLocation; i++) st+= ("\n"+listThisLocation[i].toString());
            return st;
    }
}
```

Classe Location

```
import java.util.Date;
public class Location {
    private Client clt; private Voiture voiture; private Date dateLocation;
    private int duree ; // durée de location en nombre de jours
    public Location(Client clt, Voiture voiture, int duree) {
            this.clt = clt;   this.voiture = voiture; this.duree = duree;
            this.dateLocation= new Date();
            // Ajout automatique de la location en cours au tableau des locations du
            // client conserné et celui des locations de la voiture
            this.clt.addLocation(this); this.voiture.addLocation(this);   }
    public Location(Date dateLocation, Client clt, Voiture voiture, int duree) {
            this.dateLocation = dateLocation;     this.clt = clt;
            this.voiture = voiture; this.duree = duree;
            this.clt.addLocation(this); this.voiture.addLocation(this);   }
    public Location(Client clt, Voiture voiture, Date dateLocation, int duree) {
            this.clt = clt;   this.voiture = voiture; this.dateLocation = dateLocation;
            this.duree = duree;     this.clt.addLocation(this);
            this.voiture.addLocation(this);          }
    public Client getClt() { return clt;     }
    public void setClt(Client clt) { this.clt = clt; }
    public Voiture getVoiture() {           return voiture; }
    public void setVoiture(Voiture voiture) {     this.voiture = voiture; }
    public Date getDateLocation() { return dateLocation; }
    public void setDateLocation(Date dateLocation) {
            this.dateLocation = dateLocation;     }
    public int getDuree() {return duree;  }
    public void setDuree(int duree) {      this.duree = duree;}
    @Override
    public String toString() {
            return "Location [ clt=" + clt.getNomPrenom() +
            ", voiture=" + voiture.getImmat() +  ", duree=" + duree      + "]";
    }
    @Override
    public boolean equals(Object obj) {
            if (this == obj) return true;
            if (obj == null) return false;
            if (getClass() != obj.getClass())return false;
            Location other = (Location) obj;
            if (clt == null) {
                    if (other.clt != null)    return false;
            } else if (!clt.equals(other.clt))          return false;
            if (voiture == null) {
                    if (other.voiture != null) return false;
            } else if (!voiture.equals(other.voiture))return false;
            return true;     }
}
```

Classe Agence

```
public class Agence {
    String nomagence;
    private Client [ ] listClt; private int nbClt ; // nombre des clients de l'agence
    private Voiture listVoiture[ ]; private int nbVoiture ; // nb des voitures de l'agence
    private Location [ ] listLocation ; private int nbLocation ; // nb de locations
    public Agence(String nomagence) {
            this.nomagence = nomagence;this.listClt=new Client[100];
            this.listVoiture=new Voiture[20];this.listLocation= new Location[2000];}
    public String getNomagence() {        return nomagence;     }
    public void setNomagence(String nomagence) {this.nomagence=nomagence;}
    public Client[] getListClt() {  return listClt;  }
    public int getNbClt() {return nbClt;}
    public Voiture[] getListVoiture() {    return listVoiture;}
    public int getNbVoiture() {    return nbVoiture;}
    public Location[] getListLocation() {return listLocation;}
    public int getNbLocation() {  return nbLocation;     }
    @Override
    public String toString() {
          return                "Agence[nomagence="+nomagence+",nbClt="+nbClt+",
                      nbVoiture=" +  nbVoiture+", nbLocation="+ nbLocation + "]";}
     public boolean addClt( Client client) {
            if ( getClientByMail(client.getEmail()) !=null)     return false;
            listClt[nbClt++]=client;  return true; }
    public String nomPrCltsToString() {
            if (listClt[0]==null) return "";
            String st= ("Clients de l'agence <" + nomagence + ">:\n");
            st+=listClt[0].getNomPrenom();
            for (int i=1; i<nbClt; i++) st+= ("\n"+listClt[i].getNomPrenom());
            return st; }
    public String emailCltsToString() {
            if (listClt[0]==null) return "";
            String st= ("Emails Clients de l'agence <" + nomagence + ">:\n");
            st+=listClt[0].getEmail();
            for (int i=1; i<nbClt; i++) st+= ("\n"+listClt[i].getEmail());
            return st; }
    public Client getClientByMail(String mail) {
            for (int i=0; i<nbClt;i++) {
                    if (listClt[i].getEmail().equals(mail)) return listClt[i]; }
            return null;      }
    public boolean delClt(String mail) {
            for (int i=0; i<nbClt;i++) {
                    if (listClt[i].getEmail().equals(mail)) {
                    for (int j=i; j< (nbClt-1); j++) {  listClt[j]=listClt[j+1];}
                    listClt[nbClt-1]=null; nbClt--;return true;}              }
            return false; }
    public boolean updateNomPrClt(String email, String nompr) {
            Client clt=getClientByMail(email);
            if (clt==null) return false;
```

```
        clt.setNomPrenom(nompr);   return true;}
public Voiture getVoitureByImmat(String immat) {
        for (int i=0; i<nbVoiture; i++) {
        if (listVoiture[i].getImmat().equals(immat)) {return listVoiture[i]; }
        }
        return null;
}
public boolean addVoiture(Voiture voiture) {
        if ( getVoitureByImmat(voiture.getImmat())!= null) return false;
        listVoiture[nbVoiture++]=voiture;    return true;      }
public String voituresToString() {
        if (nbVoiture==0) return "";
        String st= ("Voitures de l'agence <" + nomagence + ">:\n");
        st+=listVoiture[0].toString();
        for(int i=1; i<nbVoiture; i++) {
                st+="\n";st+=listVoiture[i].toString();}
        return st;
}
public boolean delVoiture(String immat) {
        for (int i=0; i<nbVoiture;i++) {
        if (listVoiture[i].getImmat().equals(immat)) {
                for (int j=i; j< (nbVoiture-1); j++) {
                        listVoiture[j]=listVoiture[j+1];}
                listVoiture[nbVoiture-1]=null; nbVoiture--;
                return true;      }
        }
        return false;
}

public Location getLocation(String emailClt, String immat) {
        Client clt=getClientByMail(emailClt);
        if (clt==null) return null;
        Voiture voiture=getVoitureByImmat(immat);
        if (voiture==null)return null;
        for(int i=0; i<nbLocation; i++) {
                if (listLocation[i].getClt().equals(clt)
                && listLocation[i].getVoiture().equals(voiture))
                return listLocation[i]; }
        return null;
}
public boolean addLocation(Location location) {
        if (getLocation(location.getClt().getEmail(),
                location.getVoiture().getImmat() ) != null )  return false;
        listLocation[nbLocation++]=location;
        return true;      }
public String locationsToString() {
        if (nbLocation==0)return "";
        String st= ("Locations de l'agence <" + nomagence + ">:\n");
        st+=listLocation[0].toString();
        for (int i=1; i<nbLocation; i++) {
```

```
                st+="\n"; st+=listLocation[i].toString();}
            return st; }
    public boolean finLocation(String emailClt, String immat) {
            for (int i=0; i<nbLocation; i++) {
                if (listLocation[i].getClt().getEmail().equals(emailClt)
                  && listLocation[i].getVoiture().getImmat().equals(immat)) {
                for( int j=i; j< nbLocation-1; j++) {
                        listLocation[j]=listLocation[j+1];     }
                listLocation[nbLocation-1]=null; nbLocation--;
                return true;     }
            }
            return false;
    }
}
```

Classe Test

```
public class Test {
    public static void main(String[] args) {
    /*Client [] tabClt=new Client[10];
    for(int i=0; i<6; i++) {
            tabClt[i]=new Client("NomPr"+(i+1), "Tel"+(i+1), "email"+(i+1))}
    for(Client c:tabClt) {  if (c==null) continue; System.out.println(c);          }
    */
    Agence ag=new Agence("Agence Berrechid");
    ag.addClt(new Client("EnSA","066666666", "ensab@uhp.ac.am"));
    //System.out.println(ag);
    //System.out.println(ag.nomPrCltsToString());
    for (int i=1; i<=9; i++)
      ag.addClt(new Client("CLt"+i,"0000"+i, ""+i+"@gmail.com"));
    //System.out.println(ag.getClientByMail("ensab@uhp.ac.am"));
    System.out.println(ag.nomPrCltsToString());
    /* ag.delClt("2@gmail.com");
    ag.addClt(new Client("EnSA2","055555", "ensab@uhp.ac.am"));
    System.out.println(ag.nomPrCltsToString());
    */
    //System.out.println(ag.emailCltsToString());
    //ag.getClientByMail("5@gmail.com").setNomPrenom("EEEEEEE");;
    //ag.updateNomPrClt("5@gmail.com","ZZZZZZZ");
    //System.out.println(ag.nomPrCltsToString());
    for (int i=1; i<=5; i++)   ag.addVoiture(new Voiture("Voit"+i, "MarQ"+i, i*100));
    System.out.println(ag.voituresToString());
    for(int i =1; i<20; i++) {
            //posClt= i/2 //-----> 0,1,1,2,2,3,3,.. , 9,9
            Client clt=ag.getListClt()[i/2];
            //posVoiture= i/4 //-----> 0,0,0,1,1,1,1,2,.. , 4,4,4,4
            Voiture voiture=ag.getListVoiture()[i/4];
            Location loc= new Location(clt, voiture, i);
            ag.addLocation(loc);
    }
```

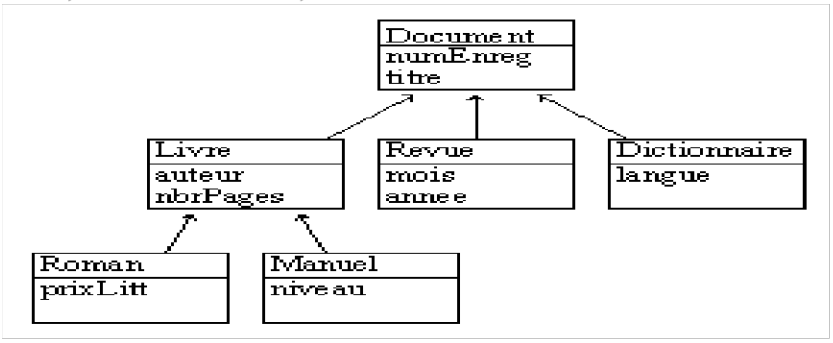
```
        System.out.println(ag.locationsToString());
        /*System.out.println( ag.getlocationByCltToString(
                            ag.getlocationByClt("6@gmail.com") )        );*/
        //Client c=ag.getClientByMail("ensab@uhp.ac.am");
        //System.out.println(c.getlocationByCltToString());
        System.out.println( ag.getVoitureByImmat("Voit3").getlocationToString() );
        }
    }
```

Chapitre 3 : L'héritage et le polymorphisme en JAVA

Exercice 3.1.

Énoncé

Pour la gestion d'une bibliothèque on nous demande d'écrire une application traitant des documents de nature diverse : des livres, qui peuvent être des romans ou des manuels, des revues, des dictionnaires, etc.



Tous les documents ont un numéro d'enregistrement (un entier) et un titre (une chaîne de caractères). Les livres ont, en plus, un auteur (une chaîne) et un nombre de pages (un entier). Les romans ont éventuellement un prix littéraire (un entier conventionnel, parmi : GONCOURT, MEDICIS, INTERALLIE, etc.), tandis que les manuels ont un niveau scolaire (un entier). Les revues ont un mois et une année (des entiers) et les dictionnaires ont une langue (une chaîne de caractères appartenant à un ensemble prédéfini, comme "anglais", "allemand", "espagnol", etc.). Tous les divers objets en question ici (livres, revues, dictionnaires, romans, etc.) doivent pouvoir être manipulées en tant que documents.

1. Définissez les classes Document, Livre, Roman, Manuel, Revue et Dictionnaire, entre lesquelles existeront les liens d'héritage que la description précédente suggère. Dans chacune de ces classes définissez

   - Le constructeur qui prend autant arguments qu'il y a de variables d'instance et qui se limite à initialiser ces dernières avec les valeurs des arguments,
   - Une méthode public String toString() produisant une description sous forme de chaîne de caractères des objets,
   - Si vous avez déclaré private les variables d'instance (c'est conseillé, sauf indication contraire) définissez également des « accesseurs » publics get... permettant de consulter les valeurs de ces variables.

2. Écrivez une classe exécutable TestDocuments qui crée et affiche plusieurs documents de types différents.

3. Une bibliothèque sera représentée par un tableau de documents. Définissez une classe Bibliotheque, avec un tel tableau pour variable d'instance et les méthodes :

   - Bibliotheque(int capacité) - constructeur qui crée une bibliothèque ayant la capacité (nombre maximum de documents) indiquée,
   - void afficherDocuments() - affiche tous les ouvrages de la bibliothèque,
   - Document document(int i) - renvoie le ième document,
   - boolean ajouter(Document doc) - ajoute le document indiqué et renvoie true (false en cas d'échec),
   - boolean supprimer(Document doc) - supprime le document indiqué et renvoie true (false en cas d'échec)
   - void afficherAuteurs() - affiche la liste des auteurs de tous les ouvrages qui ont un auteur (au besoin, utilisez l'opérateur instanceof).

Élément de correction

```
public abstract class Document {
        public Document(int numeroEnreg, String titre) {
                this.numeroEnreg = numeroEnreg;    this.titre = titre;  }
        public int getNumeroEnreg() {   return numeroEnreg;  }
        public String getTitre() {    return titre;  }
        public String toString() {   return titre + " [N° " + numeroEnreg + "]";  }
        public boolean equals(Object obj) {
                return obj instanceof Document
                        && ((Document) obj).numeroEnreg == numeroEnreg;  }
        abstract void affiche();
        private int numeroEnreg;  private String titre;}
public class Livre extends Document {
        public Livre(int numeroEnreg, String titre, String auteur, int nbrPages) {
          super(numeroEnreg, titre); this.auteur = auteur;   this.nbrPages = nbrPages;  }
        public String getAuteur() {   return auteur;  }
        public int getNombrePages () {   return nbrPages;  }
        public String toString() {
          return (super.toString() + " - " + auteur + "(" + nbrPages + " pages)" );  }
        public void affiche(){ System.out.println(toString());  };
        private String auteur;  private int nbrPages;}
public class Dictionnaire extends Document {
    public Dictionnaire(int numeroEnreg, String titre, int langue) {
        super(numeroEnreg, titre);   this.langue = langue;  }
    public int getLangue() {   return langue;  }
    public String toString() {    return super.toString() + " " + langues[langue];  }
    public void affiche(){       System.out.println(toString());  };
    public static final int FRANCAIS = 1;  public static final int ANGLAIS = 2;
    public static final int ALLEMAND = 3; public static final int ESPAGNOL = 4;
```

```
    public static final int RUSSE  = 5;
    public static final String[] langues =
        { null,   "Français", "Anglais", "Allemand", "Espagnol", "Russe" };
    private int langue;}
public class Roman extends Livre {
     public Roman(int numeroEnreg, String titre, String auteur, int nbrPages,  int prix) {
       super(numeroEnreg, titre, auteur, nbrPages);  this.prix = prix;  }
     public int getPrix() {   return prix; }
     public String toString() {
       String res = super.toString();
       switch (prix) {
        case GONCOURT : res += " - GONCOURT"; break;
        case MEDICIS : res += " - MEDICIS"; break;
        case INTERALLIE : res += " - INTERALLIE"; break;
        case ACADEMIE : res += " - ACADEMIE FRANCAISE"; break;
        default : res += " - NEANT"; break;    }    return res; }
      public static final int NEANT   = 0;  public static final int NOBEL   = 1;
      public static final int GONCOURT  = 2;   public static final int MEDICIS  = 3;
      public static final int INTERALLIE = 4;  public static final int ACADEMIE  = 5;
      private int prix;}
public class Revue extends Document {
    public Revue(int numeroEnreg, String titre, int mois, int annee) {
       super(numeroEnreg, titre);  this.mois = mois;    this.annee = annee;  }
    public int getMois() {    return mois;  }
    public int getAnnee() {    return annee;  }
    public String toString() {  return super.toString() + " " + mois + "/" + annee ;  }
    public void affiche(){  System.out.println(toString());  };
    private int mois, annee; }
public class Manuel extends Livre {
     public Manuel(int numeroEnreg, String titre, String auteur, int nombrePages, int niveau) {
       super(numeroEnreg, titre, auteur, nombrePages);    this.niveau = niveau;  }
     public int getNiveau() {   return niveau;  }
     public String toString() {  return super.toString() + " - classe de " + niveau + "An";}
     private int niveau; }
    public class TestDocuments {
        public static void main(String[] args) {
          Document unLivre =new Livre(1002, "Guerre et paix", "L. Tolstoi", 2400);
          Document unRoman  = new Roman(1003, "Les raisins de la colère", "J.
                Steinbeck", 650, Roman.ACADEMIE);
          Document unManuel  = new Manuel(1004, "Algebre", "E. Galois", 120, 2);
          Document uneRevue  = new Revue(1005,"Informatique&psychanalyse",4,2004);
          Document unDico = new Dictionnaire(1006, "Caramba!",
                Dictionnaire.ESPAGNOL);
          System.out.println("un livre:  " + unLivre);
          System.out.println("un roman:  " + unRoman);
          System.out.println("un manuel:  " + unManuel);
          System.out.println("une revue:"+uneRevue); System.out.println("un dico: "+unDico);
    }}
```

Exercice 3.2.

Énoncé

On souhaite modéliser en java le calcul de coûts de transport de marchandises. Les marchandises transportées seront des instances de la classe Marchandise :

Marchandise
- poids : int - volume : int
+ Marchandise (poids:int, volume:int) +getPoids() , getVolume() : int +toString () : String

1. Ecrire la définition de la classe marchandise.

2. Les marchandises sont transportées sous la forme de cargaisons. Une cargaison est par ailleurs également caractérisée par un code, la distance sur laquelle elle est transportée et le responsable (une chaine de caractères contenant le nom et le prénom du responsable) de transport de la garnison. Ces troix renseignements sont communiqués à la construction de la cargaison sous la forme d'un nombre entier de kilomètres et d'une chaine de caractères. Parmis les seules fonctionnalités publiques des cargaisons, on trouve:

   - **ajouter** qui permet d'ajouter une marchandise dans cette cargaison si cela est encore possible.
   - **cout** qui retourne, sous la forme d'un nombre entier en DH, le coût total du transport de cette cargaison.

   Ecrire la définition de la classe abstraite cargaison.

3. On suppose qu'une cargaison ne peut réunir qu'un nombre limité de marchandises qui dépend d'un encombrement total de ces marchandises à ne pas dépasser. Cet encombrement est soit le poids total, soit le volume total des marchandises. Une exception nommée « DepacementCapacite » est levée lorsque l'ajout d'une marchandise provoque le depacement l'encombrement autorisé. On distingue 2 types de cargaisons : Un premier type nommé GarnisonType1 et un deuxième type nommé GarnisonType2. le calcul du cout d'une garnison est donné, ainsi, par le tableau suivant :

Type cargnaison	**Encombrement**	**Cout**
GargnisonType1	Poids maximal	Distance * encombrement * 0.5
GargnisonType2	Volume maximal	(Distance + encombrement)*1.5

Donner la hiérarchie de classes permettant de modéliser les différentes classes de cargaisons et écrire ces classes en java.

4. Définir une classe nommée « TestTransport » qui contient la méthode main(). Cette méthode assure les tests suivants :

   - Créer deux garnisons : la première est de type garnisonType1 alors que la deuxième est de type GarnisonType2.
   - Ajouter deux marchandises à la première garnison et 3 marchandises à la deuxième garnison.
   - Afficher le cout des deux garnisons.

Élément de correction

Classe Marchandise

```
public class Marchandise {
        private int poid;  private int volume;
        public Marchandise(int poid, int volume) {
            this.poid = poid; this.volume = volume; }
        public int getPoid() {      return poid;     }
        public void setPoid(int poid) { this.poid = poid;          }
        public int getVolume() { return volume; }
        public void setVolume(int volume) {     this.volume = volume; }
        @Override
        public String toString() {
            return "Marchandise [poid=" + poid + ", volume=" + volume + "]";}
        @Override
        public boolean equals(Object obj) {
            Marchandise other=(Marchandise) obj; return (  this.poid==other.poid &&
                    this.volume==other.volume ); }
}
```

Classe Cargaison

```
public class Cargaison {
    private String code; private int dist;   private String nomPrResp;
    protected Marchandise listMarchandise[];
    protected int nbMarchandise;
    public Cargaison(String code, int dist,          String nomPrResp) {
        this.code = code; this.dist = dist; this.nomPrResp = nomPrResp;
        this.listMarchandise= new Marchandise[1000]; }
    public String getCode() { return code; }
    public void setCode(String code) { this.code = code;          }
    public int getDist() { return dist;       }
    public void setDist(int dist) { this.dist = dist;}
    public String getNomPrResp() {return nomPrResp; }
    public void setNomPrResp(String nomPrResp) {
        this.nomPrResp = nomPrResp;   }
    public Marchandise[] getListMarchandise() {return listMarchandise;}
    public int getNbMarchandise() {return nbMarchandise;}
```

```
    @Override
    public String toString() {
                return "Cargaison [code=" + code + ", dist=" + dist + ", nomPrResp=" +
                    nomPrResp + ", nbMarchandise="        + nbMarchandise + "]"; }
    public String listMarchandiseToString (){
        String str=this.toString() + "\n================================";
        for (int i=0; i<nbMarchandise;i++) str+=("\n"+listMarchandise[i].toString());
        return str; }
    public float cout (int iii) {return ( (float) getDist()*100.F);  }
}
```

Classe CargaisonType1

```
public class CargaisonType1 extends Cargaison{
    private int poidMax;
    public CargaisonType1(String code, int dist, String nomPrResp, int poidMax) {
        super(code, dist, nomPrResp);    this.poidMax = poidMax;}
    public int getPoidMax() {return poidMax;}
    public void setPoidMax(int poidMax) {this.poidMax = poidMax;}
    @Override
    public String toString(){return "["+super.toString()+",poidMax="+poidMax+"]";}
    public int poidTotal() {
        int poidT=0;
        for(int i=0; i<nbMarchandise; i++)poidT+=listMarchandise[i].getPoid();
        return poidT; }
    public boolean addMarchandise(Marchandise marchandise) {
        if (  (poidTotal() + marchandise.getPoid()) > poidMax  ) return false;
        listMarchandise[nbMarchandise++]=marchandise;
        return true; }
        public float cout () {return ( (float) poidTotal()* getDist()*0.5F); }
    }
```

Classe CargaisonType2

```
public class CargaisonType2 extends Cargaison{
    private int volumeMax;
    public CargaisonType2(String code, int dist, String nomPrResp,
            int volumeMax) {
        super(code, dist, nomPrResp); this.volumeMax = volumeMax; }
    public int getVolumeMax() { return volumeMax;    }
    public void setVolumeMax(int volumeMax) { this.volumeMax = volumeMax; }
    @Override
    public String toString() {
        return "[ " + super.toString()+","+ " volumeMax="+volumeMax+"]";}
    public int volumeTotal() {
        int volumeT=0;
        for(int i=0; i<nbMarchandise; i++)
            volumeT+=listMarchandise[i].getVolume();
        return volumeT;  }
    public boolean addMarchandise(Marchandise marchandise) {
        if (  (volumeTotal() + marchandise.getVolume()) > volumeMax  )return false;
        listMarchandise[nbMarchandise++]=marchandise;  return true; }
    public float cout () {return ( ( (float) volumeTotal()+ getDist() )*1.5F );}
}
```

Classe Test

```
public class Test {
    public static void main(String[] args) {
        Cargaison tabCargaison []= new Cargaison[10];
        for (int i=0; i<tabCargaison.length;i++) {
            if (i%2==0)     tabCargaison[i]=
                            new CargaisonType1(""+i, i*10,"nomPrResp"+i, i*100);
            else  tabCargaison[i]=new CargaisonType2(""+i,i*10,"nomPrResp"+i,i*100);}
        for(Cargaison c: tabCargaison) System.out.println(c.toString()); }
}
```

Exercice 3.3.

On appelle horaire une donnée caractérisant un moment dans une journée, indépendamment de la date de cette journée. Un horaire permet par exemple de préciser le moment du départ d'un train : "le train de 17h42 " où 17h42 représente l'horaire du train évoqué. Nous allons représenter un horaire par une instance de la classe Horaire.

Horaire
- heure: Integer // entre 0 et 23 - minute :Integer // entre 0 et 59
+ Horaire () + Horaire (heure: Integer, minute :Integer) + getXxx(): … // Xxx represente un attribut (heure ou minute) + setXxx(…): … // Xxx represente un attribut (heure ou minute) + compareTo(o: Object):int + equals(h : Horaire) : boolean + ecart (h : Horaire) : Integer + toString(): String

La classe Horaire implémente l'interface Comparable en réalisant la relation d'ordre intuitive sur ces horaires. La méthode equals permet de verifier si deux horaires étant égaux (s'ils ont mêmes heures et mêmes minutes). La méthode ecart fournie le résultat en minutes entre l'objet horaire invoquant la méthode et l'horaire h passé en paramètre. Cette méthode déclenche une exception IllegalArgumentException si l'horaire h est plus tôt que l'horaire invoquant.

1. Donner le code java de la méthode equals.
2. Ecrire le code des méthodes addTemp et compareTo de la classe Temps.
3. Donner le code java de la méthode ecart.

On s'intéresse à la représentation de vols entre aéroports. Les vols sont supposés être tous journaliers et il s'agit de "vols courts" donc les horaires de départ et d'arrivée sont

toujours dans la même journée.). Ces horaires correspondent à des des instances de la classe Horaire définie dans l'exercice précèdent. Dans cet exercice , on ne s'occupe donc pas des jours de départ, seul l'horaire compte (deux objets Aéroport sont donc égaux s'ils ont le même identifiant

Vol
- numero : Integer // numéro unique - depart : Aeroport - dest : Aeroport - heureDepart : Horaire - heureArrivee : Horaire
+ Vol(...) + getXxx(): … // Xxx represente un attribut + setXxx(…): … // Xxx represente un attribut + equals(o : Object) : boolean

Aéroport
- id : String // identifiant unique - lesVolsDepart : ArrayList <Vol>
+ Aéroport (...) + getXxx(): … // Xxx represente un attribut + equals(o : Object) : boolean ...

Dans les questions suivantes si pour résoudre une question vous pensez qu'il manque une méthode, vous en donnerez le code en précisant brièvement pourquoi vous ajoutez cette méthode.

4. Donnez le code java de l'entête de la classe Aeroport ainsi que la déclaration de ses attributs et de son constructeur sachant qu'initialement la liste des vols est vide.

5. Donnez le code d'une méthode ajouteVol qui prend en paramètre un vol et l'ajoute à un aéroport. Cette méthode déclenche une IllegalArgumentException si cet aéroport n'est pas l'aéroport de départ de ce vol.

6. Donnez le code java d'une méthode volsDirects qui prend en paramètre un aéroport destination dest. Cette méthode retourne la liste des vols qui partent de cette instance d'aéroport et dont la destination est dest.

Exercice 3.4.

On s'intéresse dans cet examen à la gestion de tournois de tennis. Les compétitions sont organisées par une fédération qui gère ses joueurs en leur délivrant une licence, on parle alors de licenciés d'une fédération. Cette licence donne aux joueurs le droit de participer aux tournois organisés par la fédération. Ainsi, un joueur se voit attribuer des points en fonction des résultats qu'il obtient à l'occasion de sa participation à ces tournois, en fonction des matchs qu'il y remporte.

A. On considère dans un premier lieu la classe modélisant les informations d'un joueur.

Joueur
- numLic : String - nomPrenom :String - nbPoints=0 : int
+ Joueur (numLic : String , nomPrenom : String) + getXxx() : … // Xxx correspond à un attribut + setNomPrenom (…) : void + AjoutPoints(nbPoint : int) :void + compareTo (Object j) : int + equals (Object j) : boolean // // retourne true pour des numéros de licence identique + toString () : String

1. Ecrire le constructeur de la classe « Joueur »
2. Donner le code de la méthode « compareTo » permettant de comparer deux joueurs. Un joueur sera, ainsi, considéré "plus petit" qu'un autre si son nombre de points est inférieur. Dans ce cas la méthode retourne -1. Lorsque ce nombre de points est supérieur à celui d'un autre, la méthode retourne 1 (en cas d'égalité des nombres de points, la méthode retourne 0)

B. Un match se joue entre 2 joueurs et se conclut nécessairement par un vainqueur.

Match
- joueur1 , joueur2 : Joueur - vainqueur : Joueur
+ Match(j1 : Joueur, j2 : Joueur) + getJoueur1() : Joueur ; + getJoueur2() : Joueur + getVainqueur() : Joueur ; + getVaincu() : Joueur + setVainqueur(vainqueur : Joueur) : boolean + equals (Object m) : boolean + toString () : String

- Les méthodes « getVainqueur » et « getVaincu » retourne null si le match n'a pas encore été joué et donc que le vainqueur n'est pas encore connu (valeur null).
- La méthode « setVainqueur » retourne false si son paramètre n'est pas l'un des deux joueurs du match.

1. Donner le code des méthodes « getVaincu » et « setVainqueur »
2. Ecrire la méthode « equals » qui retourne true lorsque deux matches sont égaux. Deux matches sont, ainsi, considérés égaux s'ils ont été joués par les même joueurs

C. Un tournoi comporte un certain nombre de matchs, sa structure est définie par la classe suivante :

Tournoi
- NumTournoi : int // valeur incrementale - NbPointsVictoire : int // bonus attribué au vainqueur du tournoi (par defaut = 20) - fini : boolean // vaut true si et seulement si le tournoi est terminé - listJoueursTournoi : Joueur [ ] // tableau des joueurs participant au tournoi - listMatches : Match [ ] // tableau des matchs qui ont été programmés pour le tournoi
+ Tournoi(int NbPointsVictoire) + getXxx() : … // Xxx correspond à un attribut + setFini() : void // déclarer que le tournoi est terminé (fini=true) + getNbJoueursTournoi() : int // retourne la nombre de joueurs participant au tournoi (nombre de cases // non null du tableau listJoueursTournoi) + getNbMatches() : int // retourne le nombre de matchs qui ont été programmés pour le tournoi + AjoutJoueurTournoi (j : Joueur) : boolean // ajout du joueur j au tableau listJoueursTournoi + AjoutMatch(j1 : Joueur, j2 : Joueur) :boolean // ajout d'un match au tableau listMatches + SupprimerJoueurTournoi (j : Joueur) :boolean // supprimer un joueur du tableau listJoueursTournoi + SupprimerMatch(m : Match) :boolean // supprimer un match du tableau listMatches + getVainqueurTournoi () : Joueur

- La méthode « getVainqueurTournoi » renvoie le vainqueur du tournoi si le tournoi est fini (sinon elle retourne la valeur null), il s'agit du vainqueur du dernier match du tournoi (matche de la finale)
- L'attribut « NbPointsVictoire » précise le nombre de points qui sera accordé au vainqueur du match de la finale ; la moitié de ce bonus (càd NbPointsVictoire /2) sera attribué au vaincu de cette finale.

1. Donner le constructeur de la classe Tournoi
2. Définir la méthode « AjoutMatch »

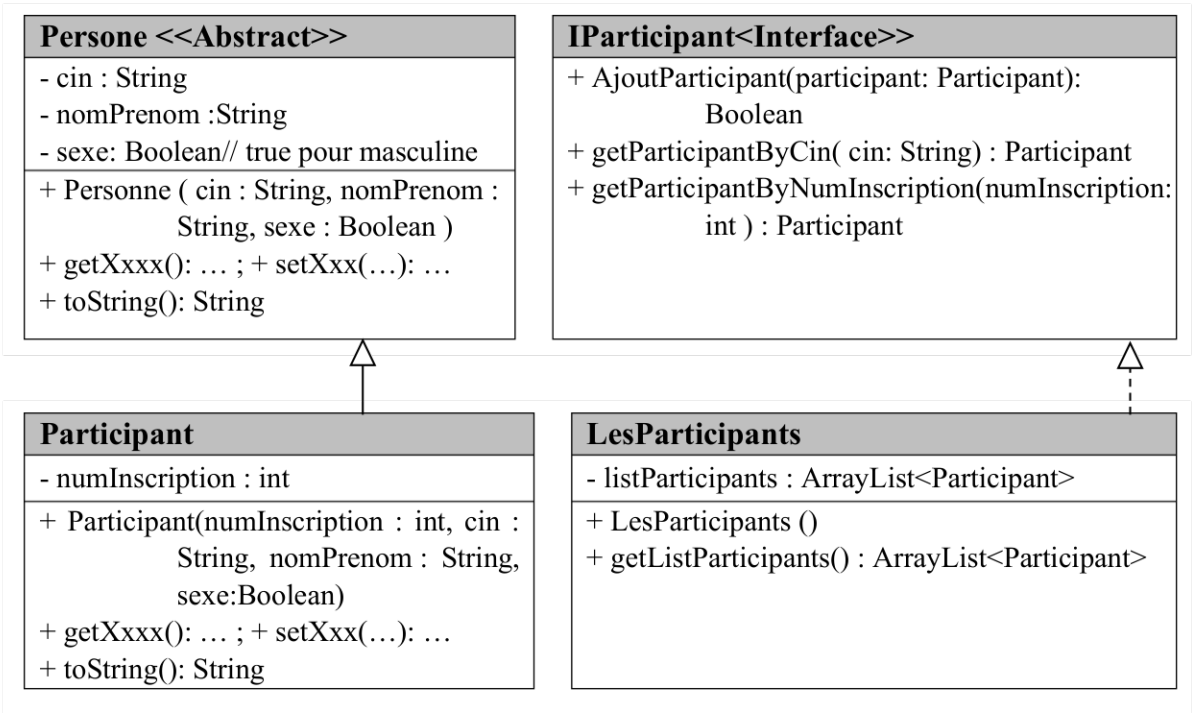
Chapitre 4 : Les collections en JAVA

Exercice 4.1.

Énoncé

On s'intéresse dans cet exercice à la gestion d'un sondage. Ce dernier consiste pour des participants à choisir pour un sujet donné un vote parmi une liste de votes possibles. Pour des raisons de simplification nous nous limitons à 3 choix de votes : les votes Oui, Non et Nul. Il est possible de voter tant qu'un sondage n'est pas clos. Ensuite, une fois que celui-ci est clos, il est possible de déterminer le résultat de ce sondage.

1. On considère dans un premier lieu les classes modélisant les informations des participants :



2. Dans un deuxième lieu nous nous intéressons à la classe vote qui modélise le vote réalisé par un participant. Nous considérons que chaque vote est défini par un id (valeur numérique qui s'incrémente automatiquement), le participant qui a effectué le vote, le sujet sur lequel le participant a voté et le choix de vote correspondant. Deux votes portant sur le même sujet et ayant le même choix sont naturellement égaux. La méthode toString doit également être définie. Le diagramme de la classe Vote est le suivant :

Vote
- id : int - participant: Participant - Sujet : String - choix : String // les choix autorisés sont : OUI, NON et NUL
+ Vote (participant : Participant, sujet : String, choix : String) + getXxxx(): … ; + setXxx(…): …; + toString() : String + equals(vote: Object) : boolean

3. Dans cette partie nous considérons le classes Sondage et Resultat nécessaires pour manipuler les informations du sondage. Ils sont représentées par le diagramme suivant :

Sondage
- Numero : int // valeur incrémentale - listSujets : ArrayList<String> - lesParticipants : LesParticipants - lesvotes : ArrayList<Vote>
+ Sondage () + getXxxx(): … ; + setXxx(…): … + ajoutSujet(sujet: String): boolean + ajoutVote(v : Vote): boolean + estClos(): boolean + clos() : void // clôt le sondage + getVotesParticipant(numInscription : int) : ArrayList<Vote> + getVotesSujet(sujet : String) : ArrayList<Vote> + getResultats() : ArrayList<Resultat> + sauvegarde (file : String) : void

Resultat
- Sujet : String - NbVoteOui : int - NbVoteNoni : int - NbVoteNul : int
+ Resultat(); + getXxxx(): … ; + setXxx(…): … + toString() : String

- La méthode ajouteVote(v) ajoute le vote v à la liste des votes effectués pour le sondage (on ne s'occupe pas ici du fait que chacun des votants ne peut voter qu'une fois, ceci est supposé géré par ailleurs). La méthode ajouteVote accepte les votes tant que le sondage n'est pas clos et que le vote fasse partie des votes autorisés (OUI, NON et NUL).
- La méthode getResultats retourne, une fois le sondage clos, pour chaque sujet le nombre de vote Oui, le nombre des vote Non ainsi que le nombre des votes NUL
- La méthode sauvegarde permet de mémoriser les résultats du sondage (sujet, nombre des votes oui non et null) dans un fichier binaire dont le nom est fourni en paramètre.
- La méthode ajouteVote lève une exception ScrutinClosException si le scrutin est clos. Les méthodes getResultats () et sauvegarde(…) lèvent une exception ScrutinNonClosException dans le cas contraire. Dans les 3 cas, si une exception est levée, la méthode n'a aucun autre effet.

Élément de correction

Classe Personne

```
public abstract class Persone {
        private String cin;          private String nomPrenom;    private Boolean sexe;
    public Persone(String cin, String nomPrenom, Boolean sexe) {
            this.cin = cin;          this.nomPrenom = nomPrenom;this.sexe = sexe;}
    public String getCin() {        return cin;      }
    public void setCin(String cin) {this.cin = cin;}
    public String getNomPrenom() {return nomPrenom;}
    public void setNomPrenom(String nomPrenom){this.nomPrenom = nomPrenom;}
    public Boolean getSexe() {return sexe;}
    public void setSexe(Boolean sexe) { this.sexe = sexe;}
    public Boolean isMen() {return sexe;}
    @Override
    public String toString() {
        return " cin="+cin +", nomPrenom=" + nomPrenom +", sexe="+sexe;}
    @Override
    public boolean equals(Object other) {return this.cin.equals(((Persone) other).cin );}
}
```

Classe Participant

```
public class Participant extends Persone{
    private int numInscription;
    public Participant(String cin, String nomPrenom, Boolean sexe, int
    numInscription) {
        super(cin, nomPrenom, sexe);    this.numInscription = numInscription;}
    public int getNumInscription() {      return numInscription;}
    public void setNumInscription(int numInscription) {
        this.numInscription = numInscription;   }
    @Override
    public String toString() {
        return "Participant[numInscription="+numInscription+ super.toString()+"]";}
    @Override
    public boolean equals(Object other) {
        return this.numInscription== ((Participant)other).numInscription
                && super.equals(other); }
}
```

Classe Soundage

```
public class Soundage {
    private static int nb=0;  private int numero ;
    private String [] listSujets ; private LesParticipants lesParticipants ;
    private Vote lesvotes[]; private boolean close;
    public Soundage() {
        numero=++nb;listSujets= new String[100];
        lesParticipants=new LesParticipants();  lesvotes= new Vote[1000];    }
    private int getNumber(Object [] tab) {
        int i=0;
        for(Object obj:tab ) { if (obj==null) return i;              i++;     }
        return i;
    }
```

```
    public    boolean ajoutSujet(String sujet) {
          int pos= getNumber(listSujets);
          if (pos ==listSujets.length) return false;
          listSujets[pos]=sujet;
          return true;          }
    public    boolean ajoutVote(Vote vote) {
          if (estClos())return false;
          String choiVote=vote.getChoix() ;
          if  (!(choiVote.equals("oui")||choiVote.equals("non") ||choiVote==null ))
             return false;
          int pos= getNumber(lesvotes);
          if (pos ==lesvotes.length) return false;   lesvotes[pos]=vote;
          return true;          }
    public LesParticipants getLesParticipants() {return lesParticipants;}
    public Boolean estClos() {return close;}
    public int getNumero() {return numero;}
    public void setNumero(int numero) {this.numero = numero;}
    public String[] getListSujets() {
          int nb=getNumber(listSujets);    String [] list= new String[nb];
          for( int i=0; i< nb; i++)     list[i]=listSujets[i];
          return list;}
    public void setListSujets(String[] listSujets) {this.listSujets = listSujets;}
    public Vote[] getLesvotes() {return lesvotes;}
    public void setLesvotes(Vote[] lesvotes) {this.lesvotes = lesvotes; }
}
```

Classe Vote

```
public class Vote {
    private int id ;  private Participant participant;
    private String sujet ;     private String choix ;
    public Vote(int id, Participant participant, String sujet, String choix) {
          this.id = id; this.participant = participant; sujet = sujet;this.choix = choix;}
    public int getId() {return id;}
    public void setId(int id) {this.id = id;}
    public Participant getParticipant() {   return participant;}
    public void setParticipant(Participant participant) { this.participant= participant;}
    public String getSujet() {return sujet;}
    public void setSujet(String sujet) {    sujet = sujet;}
    public String getChoix() {return choix;}
    public void setChoix(String choix) { this.choix = choix;}
    @Override
    public String toString() {
          return "Vote [id=" + id + ", participant=" + participant+", Sujet="+ sujet + ","
                + " choix=" + choix + "]";   }
    @Override
    public boolean equals(Object otherVote) {
          return this.sujet.equals(((Vote) otherVote).sujet) &&
                this.participant.equals( ((Vote) otherVote).participant); }
}
```

Interface IPersonne

```
public interface IPersone {
    public Boolean AjoutPersone(Persone p);
    public Boolean SuppresionPersone(Persone p);
    public Boolean ModifPersone(Persone p);
    public Persone getPersoneByCin(String cin);
}
```

Classe LesParticipants

```
public class LesParticipants implements IPersone {
    Participant listParticipant[]=new Participant[100];
    private int getNbParticipant() {
        int i=0;
        for (Participant p : listParticipant ) {if (p==null) return i;   i++;          }
        return i;  }
    @Override
    public Boolean AjoutPersone(Persone p) {
        int pos=getNbParticipant();
        if (pos== listParticipant.length) return false;
        listParticipant[pos]=(Participant) p; return true; }
    @Override
    public Boolean SuppresionPersone(Persone p) {return null;}
    @Override
    public Boolean ModifPersone(Persone p) {  return null;}
    @Override
    public Persone getPersoneByCin(String cin) {return null;}
}
```

Classe Test

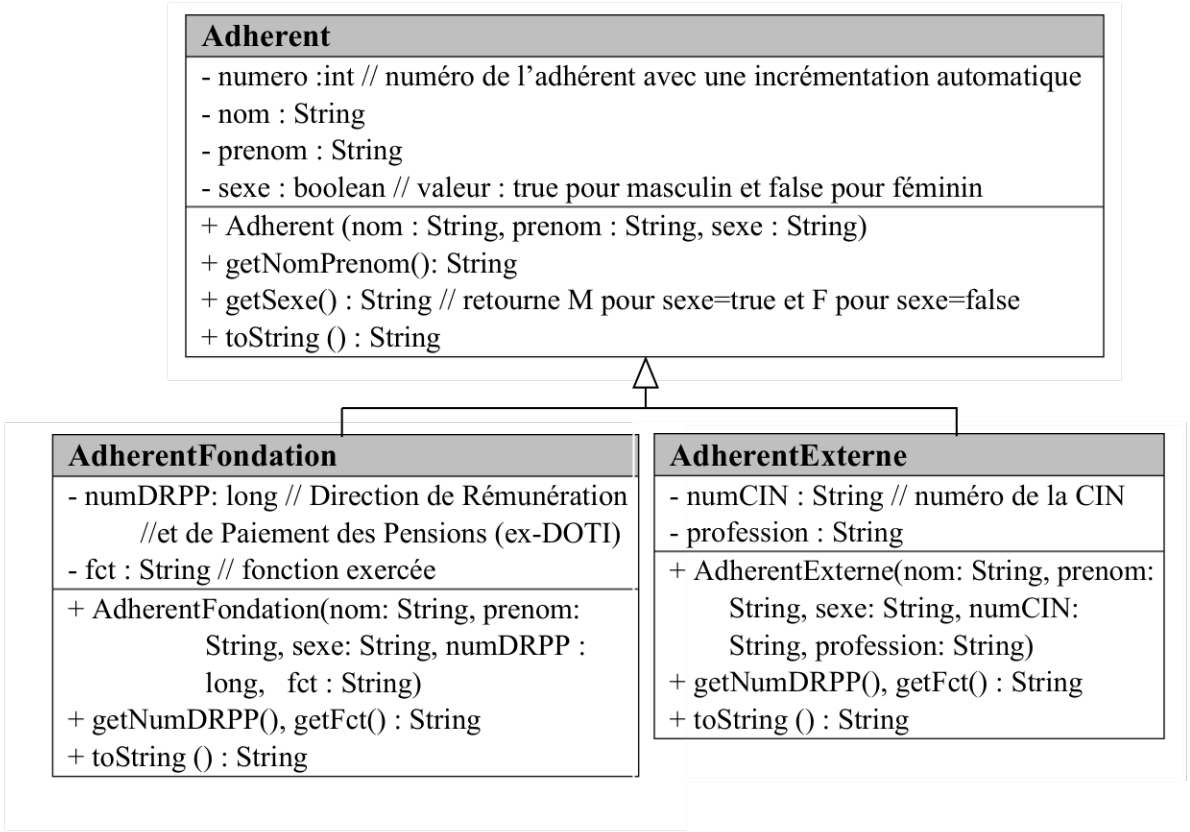
```
public class Test {
    public static void main(String[] args) {
        Scanner clavier = new Scanner(System.in);
        Soundage soundage= new Soundage();
        System.out.println(soundage.getNumero());
        for (int i=1; i<=3; i++) soundage.ajoutSujet("Sujet"+i);
        for (int i=1; i<=10; i++)
            soundage.getLesParticipants().AjoutPersone(
              new Participant("cin"+i, "ENSA"+i, false, i));
        String sujet;
        do {
            System.out.println("list des Sujiets:");
            for ( String s :soundage.getListSujets()) System.out.println(s);
            System.out.print("Sujet choisi :"); sujet=clavier.nextLine();
            System.out.print("votre Cin :"); String cin=clavier.nextLine();
            System.out.print("votre Choix :"); String choix=clavier.nextLine();
            soundage.ajoutVote(  new Vote(10,
             (Participant) soundage.getLesParticipants().getPersoneByCin(cin),
             sujet, choix)   );
        } while ( !sujet.equals("fin"))     ;
        System.out.println(soundage.getLesvotes());
    }}
```

Exercice 4.2.

Énoncé

Le club « Narjiss » est un club sportif dédié à la famille de l'enseignement. L'accès au club est ouvert aux adhérents de la Fondation Mohammed VI des Œuvres Sociales des Enseignants et aussi aux personnes externes. Le club « Narjiss » désire une application JAVA permettant de gérer ses activités sportives. Une des fonctionnalités principales de l'application, que nous traitons dans cet examen, est de permettre d'identifier les adhérents qui participent aux activités du club.

1. Considérons une classe nommée «Adherent» permettant de modéliser les caractéristiques et le comportement d'un adhérent. Afin de distinguer les adhérents de la Fondation des personnes externes, nous proposons d'utiliser deux classes nommées « AdherentFondation » et « AdherentExterne ». La représentation graphique ci-dessous illustre l'hiérarchie et la définition de ces trois classes.



   a. Donner le code de définition de la classe «Adherent»

   b. Définir, sous forme de code, la classe «AdherentFondation»

2. Le club offre différentes séances pour exercer certaines activités sportives (on se limite aux trois activités suivantes : activités de musculation, activités fitness et activités aquatiques). Chaque séance est caractérisée par un code, l'intitulé de l'activité sportive concernée (musculation, aquatiques ou fitness) et deux liste des inscrits à la séance : une première liste des adhérents de la fondation et une deuxième liste des adhérents externes. Une séance est également caractérisée par le nombre maximum d'inscrits (adhérents de la fondation et adhérents externes) à ne pas dépasser, ce nombre dépend de l'activité sportive, il est déterminé selon le tableau suivant :

Activité sportive	**Nombre maximum de participant**
Activités de musculation	15
Activités fitness	14
Activités aquatiques	10

Afin de modéliser une séance d'activité sportive, nous proposons une classe «SeanceSportive » dont la définition est donnée par la représentation graphique ci-dessous :

SeanceSportive
- code , intitule : String - listAderentsFond : ArrayList<AdherentFondation> - listExterne : ArrayList<AdherentExterne> - nbAdherentMax : int // nombre maximum d' inscrits
+ SeanceSportive(code : String, intitule : String) + getCode(), getIntitule() : String ; + getnbAdherentMax() : int + getNbAderentsFond() : int // retourne le nombre des inscrits de la fondation + getNbAdherentExterne() : int // retourne le nombre des inscrits externes + getAderentsFond(i :int) : AdherentFondation // retourne l'objet d'indice i + getAderentsExterne(i :int) : AdherentExterne + toString() : String // retourne l'ensemble des informations de la séance. + addAderentsFond (AdherentFondation adFond) : boolean // ajout à listAderentsFond + addAderentsExterne(AdherentExterne adExt): boolean//ajout à listAderentsExterne

L'ajout d'un nouvel inscrit à une séance d'activité sportive est conditionné par le nombre maximum d'inscrits. Ainsi, une classe nommée « DepacementNbInscrit » est associée à l'exception qui sera levée lorsque l'ajout d'un inscrit (adhérent de la fondation ou personne externe) provoque le déplacement du nombre d'inscrits autorisé.

a. Donner le code du constructeur de la classe « SeanceSportive »

b. Définir, sous forme de code de la classe «DepacementNbInscrit»

c. Écrire la méthode « addAderentsFond » permettant d'assurer l'ajout d'un adhérent de la fondation à la liste des adhérents de la fondation (listAderentsFond). Cette méthode retourne la valeur true si l'ajout a été effectué correctement, ou la valeur false dans le cas contraire

3. Définir une classe nommée « TestClub » qui contient la méthode « main ». Cette méthode assure les tests suivants :

- En utilisant un tableau de cinq adhérents (Type de base Adherent) créer trois adhérents de la fondation puis deux adhérents externes
- En utilisant un tableau de 2 séances d'activité sportive, créer une séance de d'activité de musculation puis une séance d'activité aquatiques
- Inscrire les deux premiers adhérents de la fondation et le deuxième adhérent externe à la première séance crée
- Inscrire le troisième adhérent de la fondation et le premier adhérent externe à la deuxième séance crée
- Afficher pour chaque séance, l'intitulé, les noms des inscrits de la fondation et les noms des inscrits externes .

Élément de correction

```
public class SeanceSportive {
        private String code ; private String intitule ;
        private ArrayList<AdherentFondation> listAderentsFond;
        private ArrayList<AdherentExterne> listExterne;
        private int nbAdherentMax ;  // nombre maximum d' inscrits
        public SeanceSportive(String code, String intitule) {
                this.code = code; this.intitule = intitule;
                listAderentsFond= new ArrayList<AdherentFondation>();
                listExterne= new ArrayList<AdherentExterne>();
                switch (intitule.toUpperCase()) {
                        case "MUSCULATION":
                                this.nbAdherentMax=15; break;
                        case "AQUATIQUES":
                                this.nbAdherentMax=10; break;
                        case "FITNESS":
                                this.nbAdherentMax=14;  break;
        }  }
        public String getCode() { return code;   }
        public String getIntitule() { return intitule;}
        public int getnbAdherentMax() { return nbAdherentMax; }
        public int getNbAderentsFond() { return (listAderentsFond.size() ); }
        public int getNbAdherentExterne() { return (listExterne.size() ); }
        public AdherentFondation getAderentsFond(int i) {
                return (listAderentsFond.get(i));}
```

```
        public AdherentExterne getAderentsExterne(int i) {
                return (listExterne.get(i)); }
        @Override
        public String toString() {
                return "SeanceSportive [code=" + code + ", intitule=" + intitule
                        + ", nbAdherentMax=" + nbAdherentMax
                        + ", listAderentsFond=" + listAderentsFond + ", listExterne="
                                + listExterne + "]"; }
        public boolean addAderentsFond(AdherentFondation adFond)throws
        DepacementNbInscrit{
            if( (getNbAderentsFond()+getNbAdherentExterne())<nbAdherentMax) {
                        listAderentsFond.add(adFond); return true; }
             else throw new DepacementNbInscrit("Dépacement du nombre d'inscrits");
        }
        public boolean addAderentsExterne (AdherentExterne adExterne) throws
        DepacementNbInscrit{
                if((getNbAderentsFond()+getNbAdherentExterne())<nbAdherentMax){
                        listExterne.add(adExterne); return true; }
                else throw new DepacementNbInscrit("Dépacement du nombre d'inscrits");
        }
}
public class Adherent {
        private int numero; // numéro de l'adhérent avec une incrémentation automatique
        private String nom ;
        private String prenom ;
        private boolean sexe ;  // valeur : true pour masculin et false pour féminin
        public Adherent(String nom, String prenom, String sexe) {
                this.nom = nom; this.prenom = prenom;
                if ( sexe.toUpperCase().equals("M") ) this.sexe = true;
                else if ( sexe.toUpperCase().equals("F") ) this.sexe = false;
        }
        public String getNomPrenom() {  return (nom+ " "+ prenom); }
        public String getSexe() { if (sexe) return ("M"); else return ("F"); }
        public String toString() {
                return "Adherent [numero=" + numero + ", nom=" + nom + ", prenom="
                                + prenom + ", sexe=" + sexe + "]"; }
}
public class AdherentExterne extends Adherent {
        private String numCIN; private String profession;
        public AdherentExterne(String nom, String prenom, String sexe,
                        String numCIN, String profession) {
                super(nom, prenom, sexe);
                this.numCIN = numCIN; this.profession = profession;    }
        public String getNumCIN() { return numCIN; }
        public String getProfession() { return profession; }
        public String toString() {
          return AdherentExterne[numCIN="+numCIN+",profession="+profession+"];}
}
```

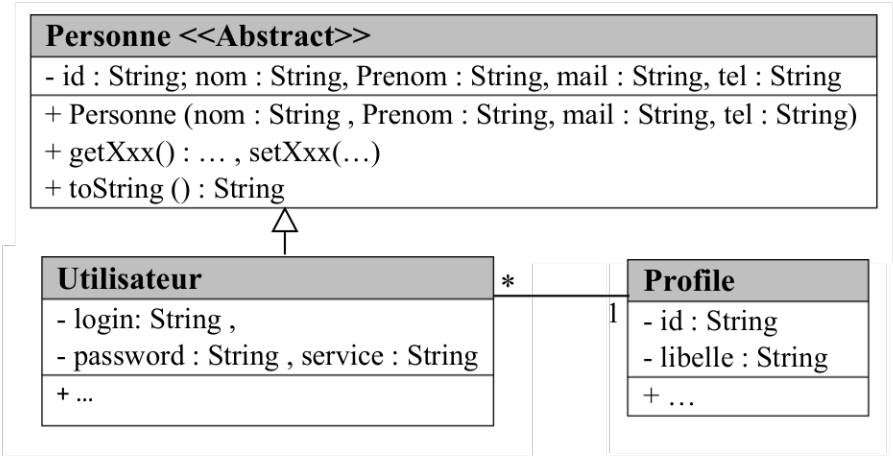
```
public class AdherentFondation  extends Adherent{
        private long numDRPP; private String fct;  // fonction exercée
        public AdherentFondation(String nom, String prenom, String sexe,
                        long numDRPP, String fct) {
                super(nom, prenom, sexe); this.numDRPP = numDRPP;  this.fct = fct;  }
        public long getNumDRPP() {    return numDRPP; }
        public String getFct() { return fct; }
        public String toString() {
                return"AdherentFondation[numDRPP="+ umDRPP +",fct=" +fct +"]";}
}
public class DepacementNbInscrit extends Exception{
        private String message;
        public DepacementNbInscrit(String message) { this.message = message;}
        public String toString() {
                return "DepacementNbInscrit [message=" + message + "]";}
}
public class TestClub {
        public static void main(String[] args) {
          //tableau des adhérents avec 3 adhérents de la fondation et 2 adhérents externes.
            Adherent tabAd[]= new Adherent[5];
            tabAd[0]= new AdherentFondation("NomAdFond1", "PrAdFond1", "M",
            10010, "PEnsSup PA");
            tabAd[1]= new AdherentFondation("NomAdFond2", "PrAdFond2", "F",
            20020, "PEnsSup PH");
            tabAd[2]= new AdherentFondation("NomAdFond3", "PrAdFond3", "F",
            30010, "PEnsSup Admin");
            tabAd[3]= new AdherentExterne("NomAdExt1", "PrAdExt1", "M",
            "W101010", "Ingenieur");
            tabAd[4]= new AdherentExterne("NomAdExt2", "PrAdExt2", "F",
            "D202020", "Entrepreneur");
            //tableau des séances avec 1 activité de musculation 2eme activité aquatiques
            SeanceSportive tabSeances[]= new SeanceSportive[2];
            tabSeances[0]=new SeanceSportive("Mus1", "musculation");
            tabSeances[1]=new SeanceSportive("Aqua1", "aquatiques");
            //Inscription des adhérents  et affichage des informations
            try {
                tabSeances[0].addAderentsFond((AdherentFondation) tabAd[0]);
                tabSeances[0].addAderentsFond((AdherentFondation) tabAd[1]);
                tabSeances[0].addAderentsExterne((AdherentExterne) tabAd[3]);
                tabSeances[1].addAderentsFond((AdherentFondation) tabAd[2]);
                tabSeances[1].addAderentsExterne((AdherentExterne) tabAd[4]);
                for ( SeanceSportive sSport : tabSeances){
                        System.out.println("\nSeance : "+ sSport.getIntitule());
                        System.out.println("======================");
                        System.out.println("Les adhérents de la fondation");
                        System.out.println("----------------------------------");
                for ( int i=0 ; i <sSport.getNbAderentsFond(); i++){
                  System.out.print(sSport.getAderentsFond(i).getNomPrenom()+",");}
                System.out.println("\n\nLes adhérents externes");
```

```
                    System.out.println("----------------------------------------");
                    for ( int i=0 ; i<sSport.getNbAdherentExterne(); i++){
                      System.out.print(sSport.getAderentsExterne(i).getNomPrenom()+",");}
                    System.out.println();}
            } catch (DepacementNbInscrit e) {e.printStackTrace(); }
    }}
```

Exercice 4.3.

Énoncé

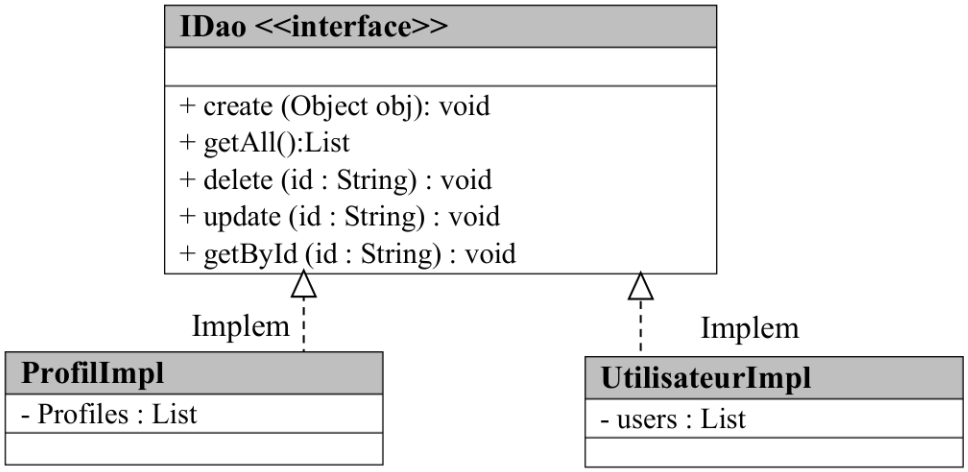
Le directeur des systèmes d'information d'une société industrielle souhaite développer un module pour la gestion des utilisateurs de son service, pour cela il vous a fait appel pour réaliser cette tâche. Le diagramme suivant représente une partie du diagramme de classe a été établi par un analyste afin de répondre à ce besoin.



On suppose que l'ID d'une personne est généré automatiquement : Il est formé des 4 premiers caractères du nom de la personne qui sont suivis du mois et de l'année de sa création. Le mot de passe doit obligatoirement comporter 8 caractères formé par des lettres (majuscule et minuscule) et des chiffres (aucun caractère spécial n'est autorisé).

1. Développer les classes ci-dessus dans le package bean. Le non-respect d'une des contraintes ci-dessus entraine le déclanchement de l'exception correspondante.
2. Développer l'interface ci-dessous dans le package dao

3. Développer les classes ProfilesImpl et UtilisateurImpl dans le package impl



4. Dans une classe de teste :

- Créer les profiles : Chef de projet (CP), Manager (MN), Directeur de projet (DP), Directeur des ressources humaines (DRH), Directeur général (DG),
- Créer des utilisateurs avec différents profiles métiers.
- Afficher la liste des managers.

Élément de correction

La classe Personne

```
package bean;
import java.text.*; import java.util.*;
public  abstract class Personne {
   private String id; private String nom;  private String prenom;  private String mail;
   private String telephone;
   private String calculId(){
    String idStr =nom.substring(0,4) ;
    SimpleDateFormat formatter = new SimpleDateFormat ("MMYY",
    Locale.FRANCE);
    idStr +=formatter.format ( new Date () )  ;    return idStr;
   }
   public Personne(String nom, String prenom, String mail, String telephone) {
    this.nom = nom;  this.id=calculId();  this.prenom = prenom; this.mail = mail;
    this.telephone = telephone; }
   public String getId() { return id; }
   public String getNom() {  return nom; }
   public void setNom(String nom) {  this.nom = nom;  this.id=calculId(); }
   public String getPrenom() { return prenom; }
   public void setPrenom(String prenom) {  this.prenom = prenom; }
   public String getMail() { return mail; }
   public void setMail(String mail) { this.mail = mail; }
   public String getTelephone() {          return telephone; }
```

```
    public void setTelephone(String telephone) {  this.telephone = telephone;}
    public String toString() {
      return " [id=" + id + ", nom=" + nom + ", prenom=" + prenom
                + ", mail=" + mail + ", telephone=" + telephone + "]";}
}
```

La classe Profil

```
package bean;
public class Profil {
    private String id; private String libelle;
    public Profil(String id, String libelle) {          this.id = id;  this.libelle = libelle;  }
    public void setId(String id) { this.id = id;  }
    public String getId() { return id;          }
    public String getLibelle() { return libelle;}
    public void setLibelle(String libelle) { this.libelle = libelle; }
    public String toString() { return " [id=" + id + ", libelle=" + libelle + "]"; }
}
```

La classe Utilisateur

```
package bean;
public class Utilisateur extends Personne{
    private String login; private String password;  private String service;
    private Profil profil;
    public Utilisateur(String nom, String prenom, String mail, String telephone, String
      login, String password, String service, Profil profil) {
      super(nom,prenom, mail, telephone); this.login = login; this.password = password;
      this.service = service; this.profil = profil;
    }
    public String getLogin() { return login; }
    public String getPassword() { return password; }
    public void setPassword(String password) {   this.password = password; }
    public String getService() { return service; }
    public void setService(String service) {          this.service = service; }
    public Profil getProfil() {  return profil;  }
    public void setProfil(Profil profil) {  this.profil = profil;        }
    public String toString() {
      return ( super.toString() + "[login=" + login + ", password=" + password
                + ", service=" + service + ", profil=" + profil + "]" ) ;}
}
```

L'interface IDao

```
package dao;
import java.io.*;
import java.util.ArrayList;
public interface IDao {
    public void create(Object obj);
    public ArrayList getAll();
    public void delete(String id);
    public void update(String id);
    public Object getById(String id);
    public  void enregistrer ()throws FileNotFoundException , IOException;
}
```

La classe ProfilImpl

```
package Imp;
import java.io.*; import java.util.*;
import bean.Profil; import dao.IDao;
public class ProfilImpl implements IDao {
   private ArrayList profiles; String nomFichier;
   public ProfilImpl(String fichier)throws FileNotFoundException , IOException  {
    this.nomFichier = fichier ;
    FileInputStream fis ;  ObjectInputStream ois = null ;
    try {
     fis = new FileInputStream ( fichier );          ois = new ObjectInputStream ( fis);
     profiles = ( ArrayList ) ois.readObject (); }
    catch ( Exception e) { profiles = new ArrayList ();  }
   }
   public void create(Object obj){ profiles.add( obj);     }
   public ArrayList getAll(){ return profiles;  }
   public void delete(String id){  profiles.remove(getById(id));  }
   public void update(String id){
    Scanner sc = new Scanner(System.in);
    System.out.print("\nEntrez le nouveau libelle: ");
    ((Profil)getById(id)).setLibelle(sc.nextLine());
    sc.close();
   }
   public Object getById(String  id){
    for(int i = 0; i < profiles.size(); i++){
    if(((Profil)profiles.get(i)).getId().equals(id)  ){ return profiles.get(i);  }  }
     return null;
   }
   public String getNomFichier() { return nomFichier; }
   public void enregistrer() throws FileNotFoundException , IOException {
    FileOutputStream fos = new FileOutputStream ( nomFichier );
    ObjectOutputStream oos = new ObjectOutputStream ( fos);
    oos.writeObject ( profiles );
   }
}
```

La classe UtilisateurImpl

```
package Imp;
import java.io.*;  import java.util.*;
import bean.Utilisateur; import dao.IDao;
public class UtilisateurImpl implements IDao{
   private ArrayList users; String nomFichier;
   public UtilisateurImpl (String fichier)throws FileNotFoundException ,
    IOException  {
    this.nomFichier = fichier ; FileInputStream fis ;  ObjectInputStream ois = null ;
    try {
     fis = new FileInputStream ( fichier );  ois = new ObjectInputStream ( fis);
     users = ( ArrayList ) ois.readObject (); }
    catch ( Exception e) { users = new ArrayList ();  }
   }
   public void create(Object obj){ users.add(obj); }
```

```
    public ArrayList getAll(){ return users;  }
    public void delete(String id){  users.remove(getById(id));  }
    public void update(String id){
      Scanner sc = new Scanner(System.in);
      System.out.print("\nEntrez le nouveau nom: ");
      ((Utilisateur)getById(id)).setNom(sc.nextLine());
      System.out.print("\nEntrez le nouveau prenom: ");
      ((Utilisateur)getById(id)).setPrenom(sc.nextLine());
      System.out.print("\nEntrez le nouveau mail: ");
      ((Utilisateur)getById(id)).setMail(sc.nextLine());
      System.out.print("\nEntrez le nouveau telephone: ");
      ((Utilisateur)getById(id)).setTelephone(sc.nextLine());
      System.out.print("\nEntrez le nouveau password: ");
      ((Utilisateur)getById(id)).setPassword(sc.nextLine());
      System.out.print("\nEntrez le nouveau service: ");
      ((Utilisateur)getById(id)).setService(sc.nextLine());
      System.out.print("\nEntrez le nouveau libelle du profil a affecter a cet " +
      "utilisateur: ");
      ((Utilisateur)getById(id)).getProfil().setLibelle(sc.nextLine());
      sc.close();;
    }
    public Object getById(String  id){
      for(int i = 0; i < users.size(); i++){ if(((Utilisateur)users.get(i)).getId().equals(id)
      ){ return users.get(i); } }
      return null;
    }
    public Object getByName(String  nom, String pr){
      for(int i = 0; i < users.size(); i++){
                        if( ((Utilisateur)users.get(i)).getNom().equals(nom) &&
                              ((Utilisateur)users.get(i)).getPrenom().equals(pr)    ){ return
                        users.get(i);  }
      }
    return null;
    }
    public String getNomFichier() { return nomFichier; }
    public void enregistrer() throws FileNotFoundException , IOException {
      FileOutputStream fos = new FileOutputStream ( nomFichier );
      ObjectOutputStream oos = new ObjectOutputStream ( fos);   oos.writeObject (
      users );
    }
}
```

La classe Test

```
import java.io.*;
import java.util.Iterator;
import Imp.*; import bean.*;
public class Test {
    public static void main(String[] args) throws FileNotFoundException,
     IOException {
     ProfilImpl p = new ProfilImpl("profil.dat");
```

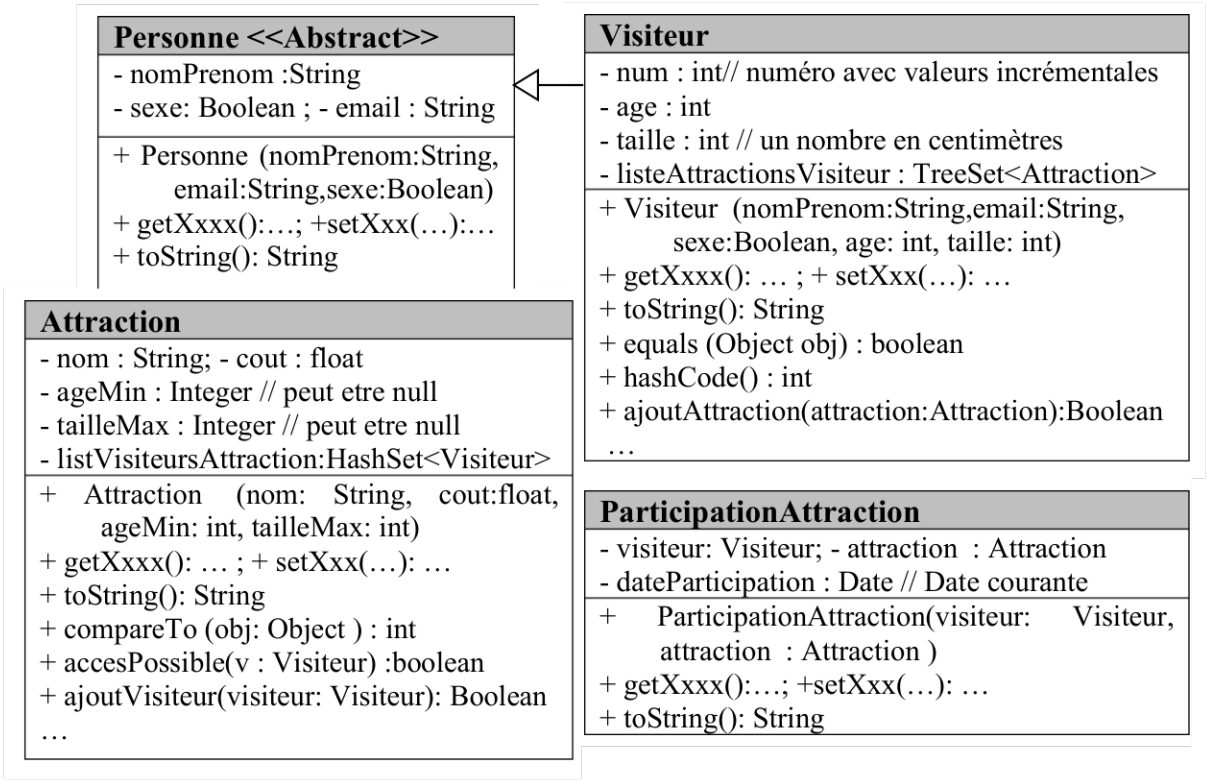
```
        //Création des profils
        p.create(new Profil("MN", "Manager"));
        p.create(new Profil("CP", "Chef de projet"));
        p.create(new Profil("DRH", "Directeur des ressources humaines"));
        p.create(new Profil("DP", "Directeur de projet"));
        p.create(new Profil("DG", "Directeur General"));
        //Liste des profils
        Iterator t = p.getAll().iterator();
        System.out.println("Les différents profils disponibles sont:\n");
        while(t.hasNext()){  System.out.println((Profil)t.next());  }
        UtilisateurImpl u = new UtilisateurImpl("users.dat");;
        //Création des utilisateurs
        u.create(new Utilisateur("EL MERCHICHI",
        "Mouad","elmerchichi.mouad@gmail.com","0634567823",
         "user1", "1234", "Informatique", (Profil)p.getById("MN")));
        u.create(new Utilisateur("SMYEJ", "Zineb", "zineb.smyej@gmail.com",
        "0634567823", "user2", "5678",
          "Ressources humaines", ( Profil)p.getById("DRH")));
        u.create(new Utilisateur("MOURAD", "Reda", "reda.geek@gmail.com",
        "0623456789", "user3", "1537",
          "Informatique", (Profil)p.getById("DP")));
        u.create(new Utilisateur("BOUKHLET", "Rihab", "rihab.peaceful@gmail.com",
        "0623567834", "user1",
          "1234", "Comptabilite", (Profil)p.getById("DG")));
        //Liste des utilisateurs
        t = u.getAll().iterator();
        System.out.println("\nLes différents utilisateurs sont:\n");
        while(t.hasNext()){ System.out.println(  ((Utilisateur)t.next()).toString()  );  }
        System.out.println("\n**************************************");
        // Liste des managers
        t = u.getAll().iterator();
        System.out.println("La liste des managers est :\n");
        while(t.hasNext()){
         Utilisateur util = (Utilisateur)t.next();
         if(util.getProfil().getId().equalsIgnoreCase("MN")){
                System.out.println(util.toString());
         }
         }

        }
}
```

Exercice 4.4.

Énoncé

On s'intéresse dans cet examen à la gestion d'un parc d'attractions qui propose à ses visiteurs une liste d'attractions(Jeux et loisirs). L'accès à ces attractions peut être contraint au respect d'une condition liée à l'âge et à la taille. Pour participer à une attraction il faut s'acquitter de son coût (prix en Dh fixe pour chaque attraction).

A. Les visiteurs du parc sont modélisés par une classe « Visiteur » dérivé de la classe « Personne » (d'autres classes comme la classe « Employe » dérive également de la classe « Personne »). Les attractions sont modélisées par la classe « Attraction » : Une attraction est définie par un nom, un coût (en DH) à acquitter et une condition d'accès (âge minimum et taille maximum). La liste « listeAttractionsVisiteur » de la classe « Visiteur » définit les attractions auxquelles participe un visiteur alors que la liste « listVisiteursAttraction » de la classe « Attraction » regroupe les visiteurs ayant participer à une attraction. Une instance de la classe « ParticipationAttraction » est créé pour chaque participation d'un visiteur à une attraction. Lors de cette opération, le visiteur et l'attraction correspondants sont ajouter respectivement aux listes « listVisiteursAttraction » et « listeAttractionsVisiteur ».



- La classe « Attraction » implémente l'interface « Comparable », une attraction sera considérée "plus petit" qu'un autre si son cout est inférieur. La méthode « compareTo », ainsi, redéfinit retourne la différence des couts.
- Les méthodes « ajoutVisiteur » et « ajoutAttraction » permettent d'ajouter un visiteur et une attaction respectivement aux listes « listVisiteursAttraction » et « listeAttractionsVisiteur ». Ces méthodes lèvent une exception « AccesInterditException » si la condition d'accès (relative à l'âge et à la taille) n'est pas respectée (nous considérons que cette exception est définie)

1. Définir le constructeur de la classe « Visiteur »
2. Ecrire le code des méthodes «compareTo» et «ajoutVisiteur» de « Attraction »
3. Donner le code du constructeur de la classe « ParticipationAttraction »

B. Considérons dans cette partie la modélisation d'un parc d'attractions. Aini, un objet parc (instance de la classe « ParcAttraction ») regroupe une liste d'attractions du parc, une liste de ses visiteurs et une liste des différentes participations qui leurs sont associées.

ParcAttraction
- ListAttractions : TreeSet<Attraction> ; - ListVisiteurs : HashSet<Visiteur> - ListParticipations : ArrayList<ParticipationAttraction>
+ ParcAttraction() ; + getXxxx(): … + ajouteAttraction(att : Attraction) : boolean + ajouteVisiteur(visiteur : Visiteur) : boolean + ajouteParticipation(participation: Participation) : boolean + getParticipations(annee : int, mois : int) : ArrayList<ParticipationAttraction> + getRecette(annee : int, mois : int) : float + saveParticipations(annee : int, mois : int) : void …

- La méthode « getParticipations » retourne la liste des participations ayant lieu pendant une année et un moi données alors que la méthode « getRecette » revoie la recette de ces participations. La recette relative à une participation est égale au cout de son attraction. Les méthodes getYear() et getMonth() d'un objet java.util.Date permettent de déterminer le mois et l'année lui correspondant.
- La méthode « saveParticipations » permet de mémoriser les informations des participations d'un mois et d'une année données dans un fichier binaire « participation-Annee-mois.dat ».

1. Donner le code de la méthode « getParticipations () »
2. Définir la méthode « getRecette » de la classe « Visiteur »
3. Ecrire le code de la méthode « saveParticipations »

C. Définir une classe de test nommée « TestParcAttraction » contenant une méthode « main » qui assure les taches suivantes :

- Création d'un parc d'attraction et lui ajouter certaines attractions.
- Ajouter des visiteurs au parc et définir certaines participations de ces visiteurs
- Afficher les participations du mois et de l'année en cours

Élément de correction

```
package metier;
    public abstract class Personne {
        protected String nomPrenom ; protected Boolean sexe;  // true pour masculine
        protected String email ;
        public Personne(String nomPrenom, Boolean sexe, String email) {
                this.nomPrenom = nomPrenom; this.sexe = sexe; this.email = email;}
        public String getNomPrenom() {return nomPrenom;}
        public void setNomPrenom(String nomPrenom){this.nomPrenom=nomPrenom;}
        public Boolean getSexe() {    return sexe;}
        public void setSexe(Boolean sexe) { this.sexe = sexe;}
        public String getEmail() {return email;}
        public void setEmail(String email) { this.email = email;}
        @Override
        public String toString() {
                return "Personne [nomPrenom=" + nomPrenom + ", sexe=" + sexe + ",
                        email=" + email + "]";}
        @Override
        public boolean equals(Object obj) {
                if (this == obj) return true; if (obj == null) return false;
                if (getClass() != obj.getClass()) return false;
                Personne other = (Personne) obj;
                return Objects.equals(email, other.email);}
        }
    public class Visiteur extends Personne {
        private int num ; // numéro avec valeurs incrémentales
        private static int nbV; private int age ; private int taille ;  // en centimètres
        private TreeSet<Attraction> listeAttractionsVisiteur ;
        public Visiteur(String nomPrenom, Boolean sexe, String email, int age, int taille){
                super(nomPrenom, sexe, email);this.num = ++nbV;
                this.age = age; this.taille = taille;
                listeAttractionsVisiteur= new TreeSet<Attraction>();}
        public int getAge() {   return age;}
```

```
    public void setAge(int age) { this.age = age;}
    public int getTaille() {          return taille;    }
    public void setTaille(int taille) {       this.taille = taille;}
    public int getNum() { return num;}
    public TreeSet<Attraction>getListeAttractionsVisiteur() {
            return listeAttractionsVisiteur;}
    @Override
    public String toString() {
            return "Visiteur [nomPrenom=" + nomPrenom +
                    ", age=" + age + ", taille=" + taille + "]";      }
    public boolean addAttraction (Attraction attraction)throws
                                AccesInterditException {
            if (!attraction.accesPossible(this))
              throw new AccesInterditException("AccesInterdit:"+this+", "+attraction);
            return listeAttractionsVisiteur.add(attraction);
    }
    @Override
    public int hashCode() {         return Objects.hash(num);}
    @Override
    public boolean equals(Object obj) {
            if (this == obj)return true; if (obj == null)return false;
            if (getClass() != obj.getClass()) return false;
            Visiteur other = (Visiteur) obj; return num == other.num;
    }
}
```

Classe Attraction

```
package metier;
import java.util.HashSet;
import Except.AccesInterditException;
public class Attraction implements Comparable<Object>{
    private String nom ;    private float cout ;
    private Integer ageMin ;  // peut etre null
    private Integer tailleMax ;  // peut etre null
    private HashSet<Visiteur> listVisiteursAttraction ;
    public Attraction(String nom, float cout, Integer ageMin, Integer tailleMax) {
        this.nom=nom;this.cout=cout; this.ageMin=ageMin; this.tailleMax=tailleMax;
        this.listVisiteursAttraction=new HashSet<Visiteur>();  }
    public String getNom() {return nom;}
    public void setNom(String nom) {    this.nom = nom;}
    public float getCout() {return cout;   }
    public void setCout(float cout) {this.cout = cout;     }
    public Integer getAgeMin() { return ageMin;}
    public void setAgeMin(Integer ageMin) {    this.ageMin = ageMin;}
    public Integer getTailleMax() {        return tailleMax;}
    public void setTailleMax(Integer tailleMax) {this.tailleMax = tailleMax;   }
    public HashSet<Visiteur>getListVisiteursAttraction(){return listVisiteursAttraction;}
    @Override
    public String toString() {
            return "Attraction [nom=" + nom + ", cout=" + cout + ", ageMin=" +
                    ageMin + ", tailleMa  x=" + tailleMax + "]";}
```

```
    public boolean accesPossible(Visiteur visiteur) {
            /*if (this.ageMin==null)
            if (visiteur.getTaille() < this.tailleMax) return true;
            else return false;
            if (this.tailleMax==null)
            if (visiteur.getAge() > this.ageMin)   return true;
            else return false;
            if ( (visiteur.getTaille() < this.tailleMax)
                    && (visiteur.getAge() > this.ageMin) ) return true;
            return false;*/
            return true;
    }
    public boolean addVisiteur (Visiteur visiteur) throws AccesInterditException {
            if (!this.accesPossible(visiteur)) throw new AccesInterditException
                    ("AccesInterdit : "  + this + " , "+ visiteur);
            return listVisiteursAttraction.add(visiteur);  }
    @Override
    public int compareTo(Object o) {
            if ((o != null) &&(o.getClass().equals("Attraction") ) ) {
                    Attraction other = (Attraction) o;
                    return (int)(this.cout - other.cout);}
            return 1;}
}
public class ParticipationAttraction  {
    private Visiteur visiteur; private Attraction attraction  ;
    private Date dateParticipation ;  // Date courante
    private static SimpleDateFormat formatter =
          new SimpleDateFormat ("dd/MM/yyyy", Locale.FRANCE);
    public ParticipationAttraction(Visiteur visiteur,
            Attraction attraction) throws AccesInterditException {
            this.visiteur = visiteur; this.attraction = attraction;
            this.dateParticipation= new Date();
            this.visiteur.addAttraction(this.attraction);
            this.attraction.addVisiteur(this.visiteur);}
    public ParticipationAttraction(Visiteur visiteur, Attraction attraction,
                    Date dateParticipation) throws AccesInterditException {
            this.visiteur = visiteur; this.attraction = attraction;
            this.dateParticipation= (Date) dateParticipation;
            this.visiteur.addAttraction(this.attraction);
            this.attraction.addVisiteur(this.visiteur);       }
     public Visiteur getVisiteur() {return visiteur;}
    public void setVisiteur(Visiteur visiteur) {this.visiteur = visiteur;   }
    public Attraction getAttraction() {return attraction;  }
    public void setAttraction(Attraction attraction) {this.attraction = attractio}
    public Date getDateParticipation() {  return dateParticipation;}
    public void setDateParticipation(Date dateParticipation) {
            this.dateParticipation = dateParticipation;    }
    public String getDateParticipationString() {
            return (String) formatter.format (dateParticipation.getTime());}
```

```
    @Override
    public String toString() {
            return "ParticipationAttraction [visiteur=" + visiteur.getNomPrenom() +
                    ", attraction=" + attraction.getNom() + ", dateParticipation="
                    + getDateParticipationString() + "]"; }
}
package dao
public class Connexion {
    private static Connection connection;
     static {
        try {
           Class.forName("com.mysql.jdbc.Driver");
                    connection=DriverManager.getConnection
                    ("jdbc:mysql://localhost:3306/DbAttraction","root", "" );
        } catch (ClassNotFoundException e) {  System.out.println(e);}
          catch (SQLException e) { System.out.println(e);}
     }
     public static Connection getConnection() {   return connection;  }
}
public class ParcAttractionDao {
    private static Connection ct=Connexion.getConnection();;
    public static HashSet<Visiteur> getAllVisiteur() throws SQLException {
    HashSet<Visiteur>visiteurs= new HashSet<Visiteur>();
    Statement st= ct.createStatement();
    ResultSet rs= st.executeQuery("select * from visiteur");
    while (rs.next()) {
       visiteurs.add(new Visiteur(rs.getString("nomPrenom"),   rs.getBoolean("sexe"),
                        rs.getString("email"),rs.getInt("age"),rs.getInt("taille")));}
    rs.close(); st.close();   return visiteurs; }
public static TreeSet<Attraction> getAllAttraction() throws SQLException {
    TreeSet<Attraction> attractions= new TreeSet<Attraction>();
    ct=Connexion.getConnection(); Statement st= ct.createStatement();
    ResultSet rs= st.executeQuery("select * from attraction");
    while (rs.next()) {
            attractions.add(new Attraction(rs.getString("nom"),rs.getFloat("cout"),
                        rs.getInt("ageMin"),rs.getInt("tailleMax"))); }
    rs.close(); st.close();   return attractions;      }
public static ArrayList<ParticipationAttraction> getAllParticipation() throws
    SQLException, AccesInterditException {
    ArrayList<ParticipationAttraction>participations=
                    new ArrayList<ParticipationAttraction>();
    ct=Connexion.getConnection(); Statement st= ct.createStatement();
    ResultSet rs= st.executeQuery("select  email,nomAttraction, dateParticipation "
                    + "from participationattraction ");
    while (rs.next()) {
      Visiteur v= ParcAttractionService.getVisiteurByEmail(rs.getString("email"));
      Attraction a=ParcAttractionService.getAttractionByNom(
                    rs.getString("nomAttraction"));
       java.util.Date dateParticipation=rs.getDate("dateParticipation");
      participations.add(new ParticipationAttraction(v,a,dateParticipation ) ); }
```

```
        rs.close();  st.close();  return participations;  }
        public static void addVisiteurDao(Visiteur visiteur)throws SQLException {
                PreparedStatement ps= ct.prepareStatement("INSERT INTO visiteur
                        (nomPrenom, sexe, email, age,taille )"+ " VALUES (?,?,?,?,?)");
                ps.setString(1, visiteur.getNomPrenom());
                ps.setBoolean(2, visiteur.getSexe()); ps.setString(3, visiteur.getEmail());
                ps.setInt(4, visiteur.getAge());ps.setInt(5, visiteur.getTaille());
                ps.executeUpdate();    }
        public static void addAttractionDao(Attraction attraction)throws SQLException {
                PreparedStatement ps= ct.prepareStatement("INSERT INTO attraction
                        (nom, cout, ageMin,tailleMax )"+ " VALUES (?,?,?,?)");
                ps.setString(1, attraction.getNom()); ps.setFloat(2, attraction.getCout());
                ps.setInt(3, attraction.getAgeMin());
                ps.setInt(4, attraction.getTailleMax()); ps.executeUpdate(); }
        public static void addParticipationDao(ParticipationAttraction participation)
                        throws SQLException {
                PreparedStatement ps= ct.prepareStatement("INSERT INTO
                        participationattraction (email, nom, dateParticipation)"
                        + " VALUES (?,?,?)");
                ps.setString(1, participation.getVisiteur().getEmail());
                ps.setString(2, participation.getAttraction().getNom());
                ps.setDate(3, (Date) participation.getDateParticipation());
                ps.executeUpdate();    }
}
package Service;
public class ParcAttractionService {
        private static TreeSet <Attraction> listAttractions;
        private static HashSet<Visiteur> listVisiteurs ;
        private static ArrayList<ParticipationAttraction> listParticipations;
        public static void loadData() throws SQLException, AccesInterditException {
                listAttractions=ParcAttractionDao.getAllAttraction();
                listVisiteurs= ParcAttractionDao.getAllVisiteur();
                listParticipations=ParcAttractionDao.getAllParticipation() ; }
        public static TreeSet<Attraction>getListAttractions(){return listAttractions;}
        public static HashSet<Visiteur> getListVisiteurs() {return listVisiteurs;}
        public static ArrayList<ParticipationAttraction> getListParticipations() {
                return listParticipations;}
        public static boolean ajoutAttraction(Attraction att) throws SQLException {
                ParcAttractionDao.addAttractionDao(att);
                listAttractions.add(att); return true;   }
        public static boolean ajoutVisiteur(Visiteur visiteur) throws SQLException {
                ParcAttractionDao.addVisiteurDao(visiteur);
                listVisiteurs.add(visiteur);      return true;      }
        public static boolean ajoutParticipation(ParticipationAttraction participation)
                        throws SQLException {
                ParcAttractionDao.addParticipationDao(participation);
                listParticipations.add(participation); return true;}
        public static boolean ajoutVisiteur(String nomPrenom, Boolean sexe,
                String email, int age, int taille) throws SQLException {
```

```
            if (getVisiteurByEmail(email)!=null) return false;
            Visiteur visiteur=new Visiteur(nomPrenom, sexe, email, age, taille);
            ParcAttractionDao.addVisiteurDao(visiteur);
            listVisiteurs.add(visiteur); return true; }
    public static boolean ajoutAttraction(String nom, float cout, Integer ageMin,
                Integer tailleMax ) throws SQLException {
            if (getAttractionByNom(nom)!=null) return false;
            Attraction att= new Attraction(nom, cout, ageMin, tailleMax);
            ParcAttractionDao.addAttractionDao(att);
            listAttractions.add(att); return true;   }
    public static boolean ajoutParticipation(String email, String nomAttraction)
                throws SQLException, AccesInterditException {
            Visiteur visiteur= getVisiteurByEmail(email);
            Attraction attraction=getAttractionByNom(nomAttraction);
            if (visiteur==null || attraction== null )return false;
            ParticipationAttraction participation= new
                ParticipationAttraction(visiteur, attraction);
            ParcAttractionDao.addParticipationDao(participation);
            listParticipations.add(participation); return true; }
    public static boolean ajoutParticipation(String email, String nomAttraction,Date
              dateParticipation ) throws SQLException, AccesInterditException {
            Visiteur visiteur= getVisiteurByEmail(email);
            Attraction attraction=getAttractionByNom(nomAttraction);
            if (visiteur==null || attraction== null )return false;
            ParticipationAttraction participation= new
                ParticipationAttraction(visiteur, attraction, dateParticipation );
            ParcAttractionDao.addParticipationDao(participation);
            listParticipations.add(participation); return true; }
    public static ArrayList<ParticipationAttraction> getParticipations
              (int annee ,  int mois ) { return null;}
    public static Float getRecette(int annee ,  int mois ) { return 0F; }
    public static Visiteur getVisiteurByEmail(String email) {
            for (Visiteur v:listVisiteurs){if (v.getEmail().equals(email))  return v;}
            return null; }
    public static Attraction getAttractionByNom(String nom) {
            for (Attraction a:listAttractions){if (a.getNom().equals(nom)) return a;}
            return null; }
}
```

Classe Test

```
public class Test {
    public static void main(String[] args) {
     try {
            ParcAttractionService.loadData();
            System.out.println("********** Liste des visiteurs  **********");
            for (Visiteur v : ParcAttractionService.getListVisiteurs() )
                System.out.println(v);
            System.out.println("********** Liste des attractions  **********");
            for (Attraction a : ParcAttractionService.getListAttractions() )
                System.out.println(a);
```

```
            System.out.println("********** Liste des participations  **********");
            for(ParticipationAttraction p:ParcAttractionService.getListParticipations())
                        System.out.println(p);
            /*
            for (int i =1; i<=10; i++) ParcAttractionService.ajoutVisiteur
                  ("Ensab"+i, true, "ensab"+i+"@uhp.ac.ma",i*2, i*3+10);
            for (int i =1; i<=5; i++)  ParcAttractionService.ajoutAttraction
                  ("Attraction"+i, 10.0F*i,i*5, i*5);*/
            for (int i =1; i<=5; i++) ParcAttractionService.ajoutParticipation(
                              "ensab"+i+"@uhp.ac.ma","Attraction"+i);
      } catch (Exception e) { System.out.println(e);}
      }
}
```

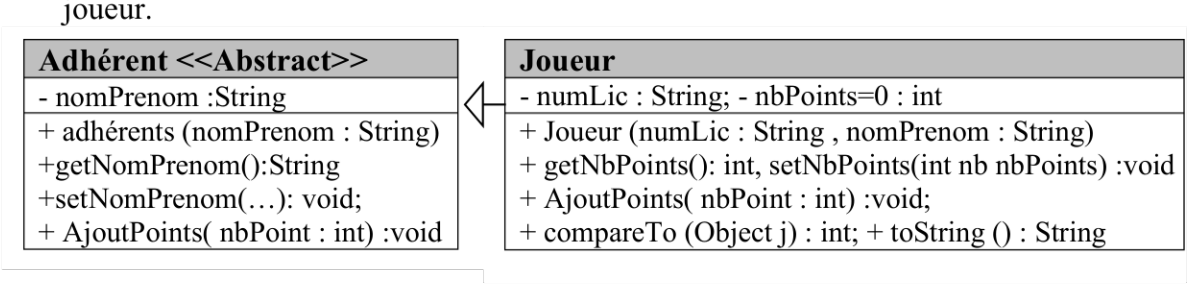
Annales des examens

Examen 1. DUT GI S3 2018-2019

Énoncé

On s'intéresse dans cet examen à la gestion de tournois de tennis. Les compétitions sont organisées en fédérations qui gèrent ses adhérents (joueurs et arbitres) en leur délivrant une licence, on parle alors de licenciés d'une fédération. Cette licence donne au joueurs le droit de participer aux tournois organisés par la fédération. Ainsi, un joueur se voit attribuer des points en fonction des résultats qu'il obtient à l'occasion de sa participation à ces tournois, en fonction des matchs qu'il y remporte.

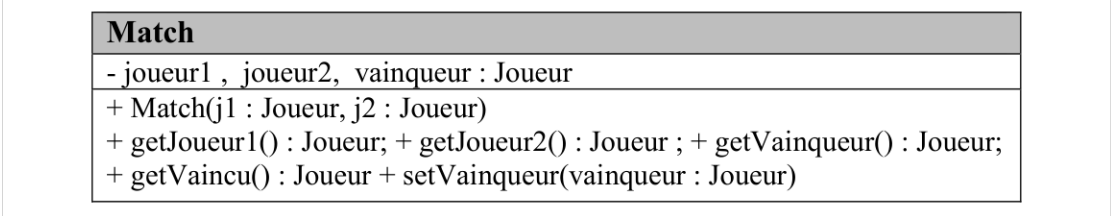
A. On considère dans un premier lieu les classes modélisant les informations d'un joueur.



La classe Joueur implémente l'interface Comparable, un joueur sera considéré "plus petit" qu'un autre si son nombre de points est inférieur .

Donner le code de la classe Joueur .

B. Un match se joue entre 2 joueurs et se conclut nécessairement par un vainqueur.

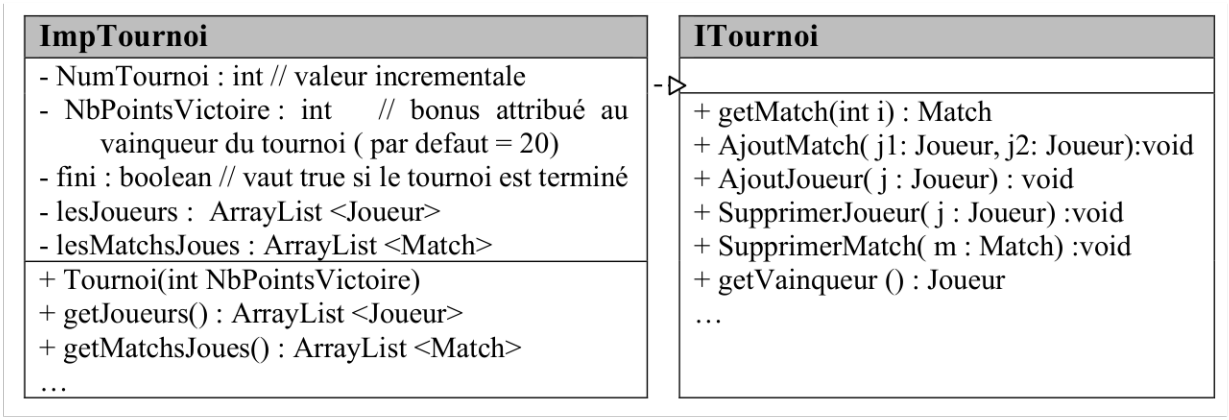


Les méthodes getVainqueur et getVaincu lèvent l'exception IllegalStateException si le match n'a pas encore été joué et donc que le vainqueur n'est pas encore connu. La

méthode setVainqueur lève une exception IllegalArgumentException si son paramètre n'est pas l'un des deux joueurs du match.

Donner le code des méthodes « getVaincu » et « setVainqueur ».

C. Un tournoi comporte un certain nombre de matchs. Sa structure est définie comme suit :

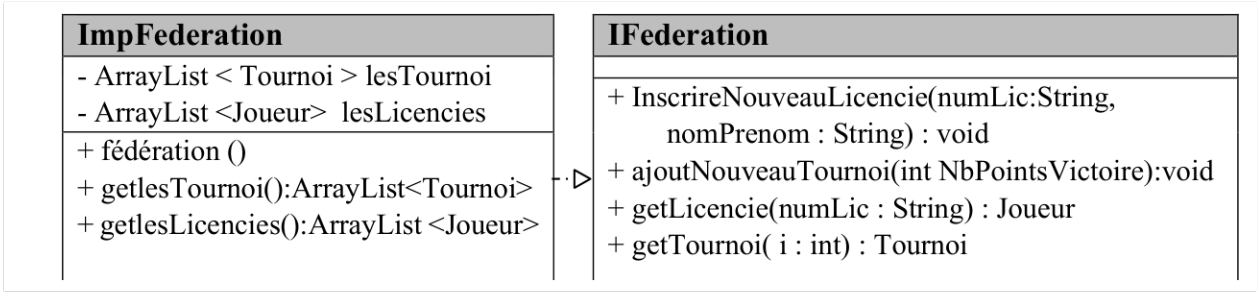


La méthode getVainqueur renvoie le joueur vainqueur du tournoi si le tournoi est fini (il s'agit du vainqueur du dernier match du tournoi) , sinon une exception TournoiNonFiniException est levée. Les méthodes d'ajout et de suppression d'un joueur ou d'un match lévent une exception TournoFiniException si le tournoi est fini.

- On suppose que les classes d'exception TournoiNonFiniException et TournoFiniException sont supposées défini. Elles contiennent un constructeur sans paramètre.
- Nous considérons que les informations de chaque tournoi sont enregistrées dans un fichier binaires(attribuer « Tournoi-num.dat » au fichier d'un tournoi dont le numéro est num. Il faut prévoir un repertoire tournoi pour regrouper l'ensemble des fichiers).

Donner le code du constructeur de la classe Tournoi et celui des méthodes « AjoutMatch » et « getVainqueur ».

D. La fédération regroupe les différents joueurs licenciés (attribut lesLicencies) et est chargée de l'attribution des points en fonction du résultat aux tournois. Avec un objet de la classe Federation, on peut inscrire un nouveau licencié (joueur) à partir d'un nom-prénom et d'un numéro de licence.



Donner le code des méthodes « InscrireNouveauLicencie » et « getLicencie

Élément de correction

```
public  abstract class Adherent {
    private String nomPr;
    public Adherent(String nomPr) {this.nomPr = nomPr;}
    public String getNomPr() {return nomPr;}
    public void setNomPr(String nomPr) {this.nomPr = nomPr;}
    @Override
    public String toString() { return "Adherent [nomPr=" + nomPr + "]";}
     public abstract void addPoint(int nbPts);
}
public  class Joueur extends Adherent implements Comparable<Object>{
     private String licence;private int nbpoint;
     private Match listMatch[];     private int nbMatch;
     public Joueur(String nomPr, String licence) {
              super(nomPr); this.licence = licence;this.listMatch= new Match [100];}
     public boolean addMatch(Match match) {
               listMatch[nbMatch++]=match;return true;   }
     public Match[] getListMatch() {
              Match [] list= new Match[nbMatch];
              for(int i=0; i<nbMatch; i++)  list[i]=this.listMatch[i];  return list;}
     public String getLicence() {   return licence; }
     public void setLicence(String licence) {this.licence = licence;}
     public int getNbpoint() {return nbpoint;}
     public void setNbpoint(int nbpoint) {this.nbpoint = nbpoint;}
     @Override
     public String toString() {
              return "Joueur [ NomPr :" + getNomPr() +", licence :" + licence + ","
                                       + " nbpoint:" + nbpoint + "]";}
     public  void addPoint(int nbPts) {this.nbpoint+=nbPts;}
     @Override
     public int compareTo(Object o) {
              Joueur o2=(Joueur) o;   // this ----- j2
              if (this.nbpoint > o2.nbpoint) return 1;
              else if (this.nbpoint<o2.nbpoint) return -1;   else return 0;   }
```

```
    @Override
    public boolean equals(Object obj) {
            if (this == null || obj == null) return false;
            if (getClass() != obj.getClass()) return false;
            Joueur other = (Joueur) obj;
            if (licence == null) {   if (other.licence != null) return false;
            } else if (!licence.equals(other.licence)) return false;
            return true; }
}
public class Match {
    private Joueur joueur1;private Joueur joueur2; private Joueur vainqueur;
    public Match(Joueur joueur1, Joueur joueur2) {
            this.joueur1 = joueur1; this.joueur2 = joueur2; }
    public Joueur getJoueur1() {  return joueur1;}
    public void setJoueur1(Joueur joueur1) { this.joueur1 = joueur1; }
    public Joueur getJoueur2() {  return joueur2; }
    public void setJoueur2(Joueur joueur2) { this.joueur2 = joueur2;}
    public Joueur getVainqueur() {return vainqueur;}
    public void setVainqueur(Joueur v) {
            if (  v.equals(joueur1) || v.equals(joueur2) )  this.vainqueur = v; }
    @Override
    public String toString() {
            String st="Match [ 1ier joueur :" + joueur1.getNomPr() +
                    ", 2eme joueur :" + joueur2.getNomPr();
            if (vainqueur!=null)    st += (", vainqueur:" + vainqueur.getNomPr() );
            st+= "]"; return  st;}
    }
public interface ITournoi {
    public Match getMatch(int i);
     public Joueur getVainqueur();}
public class Tournoi implements ITournoi{
     private int numTournoi; private int NbPointsVictoire; private boolean fini;
    private static int numInc; // assurer incrementation p/p à la classe
    private Joueur listJoueurs[]; private int nbJoueurs;
    private Match listMatchs[]; private int nbMatchs;
    public Tournoi(int nbPointsVictoire) {
            NbPointsVictoire = nbPointsVictoire; this.numTournoi=++numInc;
            listJoueurs=new Joueur[20];  listMatchs= new Match[50];  }
    @Override
    public Match getMatch(int i) {return null;}
    @Override
    public Joueur getVainqueur() {return null;}
```

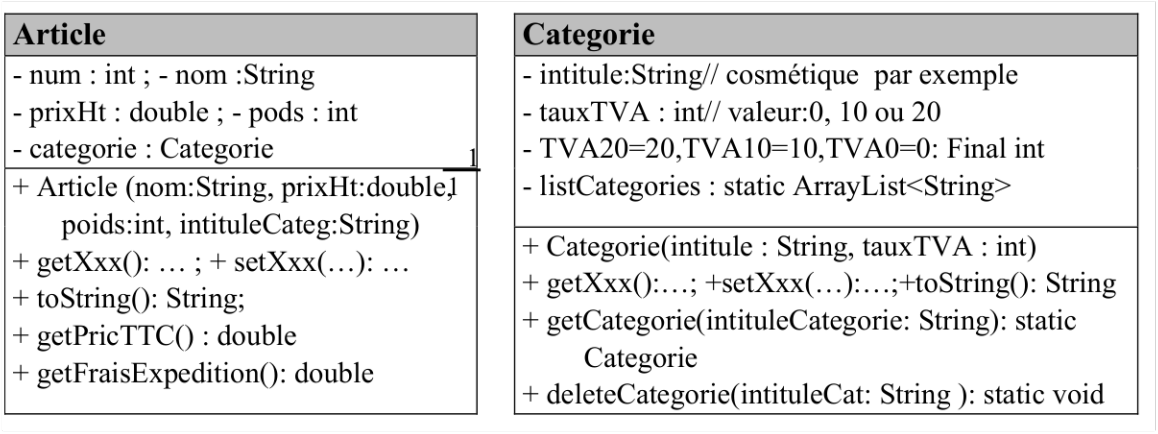
```
public class Test {
    public static void main(String[] args) {
            Joueur j1 = new Joueur("ENSA", "Lic UHP"); j1.addPoint(10);
            Joueur j2=new Joueur("FSTS", "UH1"); j2.addPoint(15);
            Joueur j3=new Joueur("ENCG", "UHP");
            //System.out.println(j2); //System.out.println( j2.compareTo(a) );
            Match m1= new Match(j1, j2); System.out.println(m1);
            m1.setVainqueur(j1); System.out.println(m1);
}}
```

Examen 2. DUT GI S3 2019-2020

Énoncé

Le contexte de ce sujet est une société de vente en ligne qui propose et assure l'expédition des articles divers.

A. On considère dans un premier lieu les classes modélisant les informations des articles mis en vente et expédiés. Un article est caractérisé par un nom, un prix HT (hors taxe) et un poids (en grammes) précisés à la construction d'un objet article (un numéro est attribué à un article par une incrémentation automatique). A chaque article est associé un taux de TVA qui dépend de sa catégorie (articles cosmétiques, articles alimentaire,…). Par conséquent, l'intitulé de la catégorie de l'article est également fourni lors de sa construction.



La méthode « getPrixTTC » retourne la prix TTC d'un article avec prixTTC = PrixHT *(1 + TauxTVA) alors que la méthode « getFraisExpedition » permet de retourner le montant correspondant aux frais d'expédition de l'article. Ce monatnt est calculé sur la base du poids de l'article (5$ si poids< 5kg, 15$ si poids< 15kg et 0,75*poids si non).

1. Définir le constructeur de la classe « Categorie » permettant de créer une nouvelle instance et d'ajoute cette dernière à la liste « listCategories ». Le taux de la TVA correspondant doit prend la valeur de l'une des 3 constantes déclarées dans la classe « Categorie » : TVA20 pour un taux de TVA de 20%, TVA10 pour un taux de TVA de 10% et TVA0 pour une exonération càd un taux de TVA de 0%. Dans

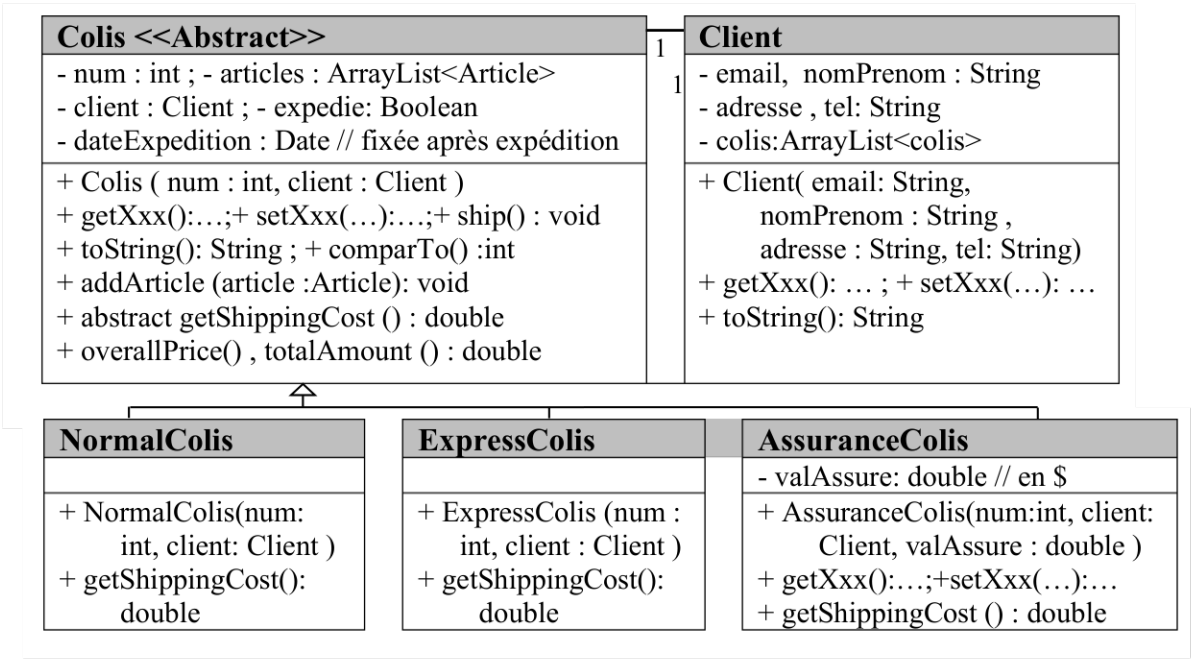
le cas où le taux est différent des 3 constantes, une exception (considérée définie) nommée « invalidTauxTVA » est levée. (2 pts)

2. Donner le code de la méthode « getCategorie » permettant de retourner l'instance de la liste « listCategories » qui correspond à l'intitulé fournie en paramètre (une valeur null est retournée si la catégorie est inexistante) ; (2 pts)

3. Ecrire le code du constructeur de la classe « Article » sachant que l'intitulé de la catégorie fournie en paramètre permet de retrouver l'objet categorie de l'article à instancier (objet existant dans la liste des catégories « listCategories »). Une nouvelle catégorie est créée (et sera automatiquement ajouter à la liste « listCategories ») lorsque l'intitulé fourni au constructeur ne correspond à aucune catégorie existante. (3 pts)

B. Un colis regroupe plusieurs articles pour leur expédition. En plus du numéro du colis et la liste des articles expédiés, un colis est défini pour un client (la classe Client est supposé défini). Nous considérons qu'il existe 3 types de colis : des colis « normaux», des colis « express » et des colis « avec assurance ». Les colis « avec assurance » sont en plus définis par la valeur maximale (un nombre réel en $) assurée est également fourni à la construction du colis « avec assurance » . Les frais d'expédition associés à un colis sont définis en fonction du type des colis :

- Les frais d'expédition d'un colis « normal » sont obtenus par la somme des frais des articles expédiés.
- Pour un colis « avec assurance », les frais d'expédition sont calculés sur la base de ceux d'un colis normal avec un surcoût obtenu comme suit : si le montant assuré du colis est inférieur à 100 $ alors le surcoût est de 10$, sinon le surcoût est de 5% du montant assuré.
- Pour un colis « express », les frais d'expédition sont le double de ceux d'un colis normal.



Un colis peut être complété jusqu'à son expédition par l'ajout de nouveaux articles grâce à la méthode « addArticle » (on suppose que la quantité de l'article est égale à 1, dans le cas contraire l'article sera ajouté plusieurs fois). L'expédition d'un colis est gérée par une méthode « ship » (l'attribut « expédie » reçoit la valeur true et la date d'expédition est , ainsi, fixée). La tentative d'ajout d'un article (via « addArticle ») provoque la levée d'une exception « ColisExpedException » (considèrée définie) si le colis est déjà expédié.

La méthode « getShippingCost » permet d'obtenir les frais d'expédition d'un colis ; la méthode « overallPrice » fournie le prix global de l'ensemble des articles, hors frais d'expédition (somme des prix TTC) et la méthode « totalAmount » retourne le montant total d'un colis (montant total = prix global + frais expédition).

Les colis de même nature sont comparables deux à deux (une exception de ClassCastException est levée dans le cas contraire) , la classe « Colis » implémente l'interface Comparable. La méthode compareTo qui définit la relation d'ordre relatif au poids permet alors de déterminer si un colis est plus lord qu'un autre.

1. Ecrire le code des constructeurs relatif aux classes Colis et AssuranceColis ;(3 pts)
2. Définir les méthodes «addArticle» et «compareTo» de la classe « Colis »; (3 pts)
3. Donner le code de la méthode«getShippingCost» de la classe«AssuranceColis».(2pts)

C. Dans cette partie nous considérons la classe ServiceColis nécessaire pour manipuler les colis de l'année en cours. Elle est représentée par le diagramme suivant :

ServiceColis
- listColis : ArrayList<Colis> ; - annee : int
+ LesColis() ; + getXxx(): … ; + setXxx(…): … + ajoutColis(numColis:int, Client: Client): void + getClientByemail(email: String):Client; + getColisByNum(num:int): Colis + loadLescolis(annee : int) : void ; + saveLesColis() : void …

La méthodes « ajouteColis » crée un nouvel colis et l'ajoute à la liste des colis. Elle lève l'exception InvalideNumColis (considèrée définie) lorsque le numéro du nouvel colis est égal à celui d'un colis existant dans la liste « listColis ». Les colis sont enregistrés dans un fichier « Colis-Annee.dat » où annee correspond à la l'année du colis.

1. Donner le code de la méthode « ajouteColis » ; (2 pts)
2. Définir la méthode « saveLesColis » permettant de sauvegarder le contenu de la liste des colis dans le fichier « Colis-Annee.dat » ; (1.5 pts)
3. Donnez le code d'une méthode main que vous jugez pertinent pour tester les services de la classe « ServiceColis ». (1.5 pts)

Élément de correction

Classe Categorie

```
import Execpt.InvalidCategourieExeption;
public class Categorie {
    private String intitule ; private int tauxTVA  ;
    private final int TVA20=20, TVA10=10, TVA0=0 ;
    private static ArrayList<Categorie>
    listCategories= new ArrayList<Categorie>();
     public Categorie(String intitule, int tauxTVA) throws InvalidCategourieExeption {
            valideIntitule(intitule);
            this.intitule = intitule;this.tauxTVA = tauxTVA;
            this.listCategories.add(this);  }
    public String getIntitule() {    return intitule; }
    public void setIntitule(String intitule) {this.intitule = intitule;}
    public int getTauxTVA() {return tauxTVA;  }
    public void setTauxTVA(int tauxTVA) {this.tauxTVA = tauxTVA;}
    @Override
```

```
    public String toString() {
            return "Categorie[intitule="+intitule+",tauxTVA="+tauxTVA+"]";}
    public static ArrayList<Categorie> getListCategories() {return listCategories;}
    public static Categorie getCategorie ( String intituleCategorie ) {
            for (Categorie ca  : listCategories ) {
                    if (ca.intitule.equals(intituleCategorie) )return ca;}
            return null;}
    public static void deleteCategorie(String intitule) {
            for (int i=0; i< listCategories.size();i++ )
                    if (listCategories.get(i).intitule.equals(intitule) ) {
                            listCategories.remove(i); return;}
    }
    public void valideIntitule(String intitule)throws InvalidCategourieExeption {
            for (Categorie c : listCategories)
            if (c.intitule.equals(intitule))
               throw new InvalidCategourieExeption("Invalide Intitulé");}
}
```

Classe Article

```
public class Article {
    private int num ;  private String nom ; private double prixHt  ;
    private int poids ; Categorie categorie ;  private static int inc=1;
    public Article(String nom, double prixHt, int poids, String intituleCategorie ) {
            Categorie c =Categorie.getCategorie(intituleCategorie);
            this.num=inc++;
            this.nom = nom;        this.prixHt = prixHt; this.poids = poids;
            this.categorie=c;      }
    public Article(String nom, double prixHt, int poids, Categorie categorie ) {
            Categorie c =categorie;this.num=inc++;
            this.nom = nom;        this.prixHt = prixHt; this.poids = poids;
            this.categorie=c;      }
    public Article(String nom, double prixHt, int poids,
            String intitule, int tauxTva) throws InvalidCategourieExeption {
            Categorie c =new Categorie(intitule, tauxTva);
            this.num=inc++;this.nom = nom;      this.prixHt = prixHt;
            this.poids = poids; this.categorie=c;  }
    public double getPricTTC () { // prixTTC = PrixHT *(1 + TauxTVA)
            return this.prixHt*(1+this.categorie.getTauxTVA());}
    public double getFraisExpedition() {
            // (5$ si poids< 5kg, 15$ si poids< 15kg et 0,75*poids si non).
            if (poids <5) return 5; else if(poids <15) return 15;
            return poids* 0.75;}
    public String getNom() {return nom; }
    public void setNom(String nom) {this.nom = nom;  }
    public double getPrixHt() {return prixHt;}
```

```
    public void setPrixHt(double prixHt) {this.prixHt = prixHt;}
    public int getPoids() {return poids;}
    public void setPoids(int poids) {this.poids = poids;}
    public Categorie getCategorie() {return categorie;}
    public void setCategorie(Categorie categorie) {this.categorie = categorie;}
    public int getNum() {return num;}
    @Override
    public String toString() {
        return "Article [num="+num+", nom="+nom+", "+ "prixHt=" + prixHt +
            ", poids=" + poids +" \n Categorie = " + categorie.getIntitule()+
            " Prix TTC ="+getPricTTC()+" Frais exp="+getFraisExpedition()+"]";}
}
```

Classe Client

```
public class Client {
    private String email ;  private String nomPrenom ; private String adresse , tel;
    private ArrayList<Colis> colis ;
    public Client(String email, String nomPrenom, String adresse, String tel) {
        this.email = email;this.nomPrenom = nomPrenom;
        this.adresse = adresse; this.tel = tel;
        this.colis= new ArrayList<Colis>();}
    public String getEmail() {return email;}
    public void setEmail(String email) {this.email = email;}
    public String getNomPrenom() {return nomPrenom;}
    public void setNomPrenom(String nomPrenom){this.nomPrenom=nomPrenom;  }
    public String getAdresse() {return adresse;}
    public void setAdresse(String adresse) {this.adresse = adresse;}
    public String getTel() {return tel;}
    public void setTel(String tel) {this.tel = tel;}
    @Override
    public String toString() {
        return "Client [email=" + email + ", nomPrenom=" + nomPrenom + ",
                adresse=" + adresse + ", tel=" + tel + "]";}
    @Override
    public int hashCode() {return Objects.hash(email);  }
    @Override
    public boolean equals(Object obj) {
        if (this == obj)return true;
        if (obj == null) return false;
        if (getClass() != obj.getClass()) return false;
        Client other = (Client) obj;
        return Objects.equals(email, other.email);    }
    public void addColis(Colis c) {this.colis.add(c);}
}
```

Classe Colis

```
public abstract class Colis {
    protected int num ;
    protected ArrayList<Article> articles ;  protected Client client ;
    protected Boolean expedie=false;  // true si c'est expédié
    protected Date dateExpedition ;  // fixée par setter après expédition
    public Colis(int num, Client client) {
            this.num = num; this.client = client;
            articles= new ArrayList<Article>(); this.client.addColis(this);       }
    public int getNum() { return num;     }
    public void setNum(int num) {this.num = num;}
    public Client getClient() {return client;}
    public void setClient(Client client) { this.client = client;}
    public Date getDateExpedition() {return dateExpedition;}
    @Override
    public String toString() {
            return "Colis [num=" + num + ", expedie=" + expedie +
                    ", dateExpedition=" + dateExpedition + "]"; }
    public Boolean isShip() {return expedie;}
    public void ship() throws ShipExeption {
            if (isShip() ) throw new ShipExeption("Colis deja Expedidée");
            this.expedie=true; this.dateExpedition= new Date();}
    public void addArticle(Article article) throws ShipExeption {
            if (isShip() ) throw new ShipExeption("Colis deja Expedidée");
            articles.add(article);}
   public abstract Double getShippingCost();
   public  Double overallPrice() {
            Double oPrice=0.0D; for (Article a   : articles) oPrice+=a.getPricTTC();
            return oPrice; }
   public  Double getShippingArticle() {
            Double fraisE=0.0D;
            for (Article a   : articles) fraisE+=a.getFraisExpedition(); return fraisE;  }
   public  Double totalAmount () {return (overallPrice()+ getShippingCost()); }
   public ArrayList<Article> getAllArticle() {   return articles; }
}
```

Classe NormalColis

```
public class NormalColis extends Colis{
    public NormalColis(int num, Client client) {super(num, client);}
     @Override
    public Double getShippingCost() {return getShippingArticle();}
}
```

Classe ExpressColis

```
public class ExpressColis extends Colis{
    public ExpressColis(int num, Client client) {super(num, client); }
```

```
    @Override
    public Double getShippingCost() { return getShippingArticle()*2;}    }
```

Classe AssuranceColis

```
public class AssuranceColis extends Colis{
    private double valAssure;
    public AssuranceColis(int num, Client client, double valAssure) {
            super(num, client); this.valAssure = valAssure;}
    @Override
    public Double getShippingCost() {
            Double surcout;
            if (valAssure< 100.0D)  surcout=10.0D; else surcout=valAssure*5/100;
            return getShippingArticle()+ surcout; }
}
```

Classe InvalidCategourieExeption

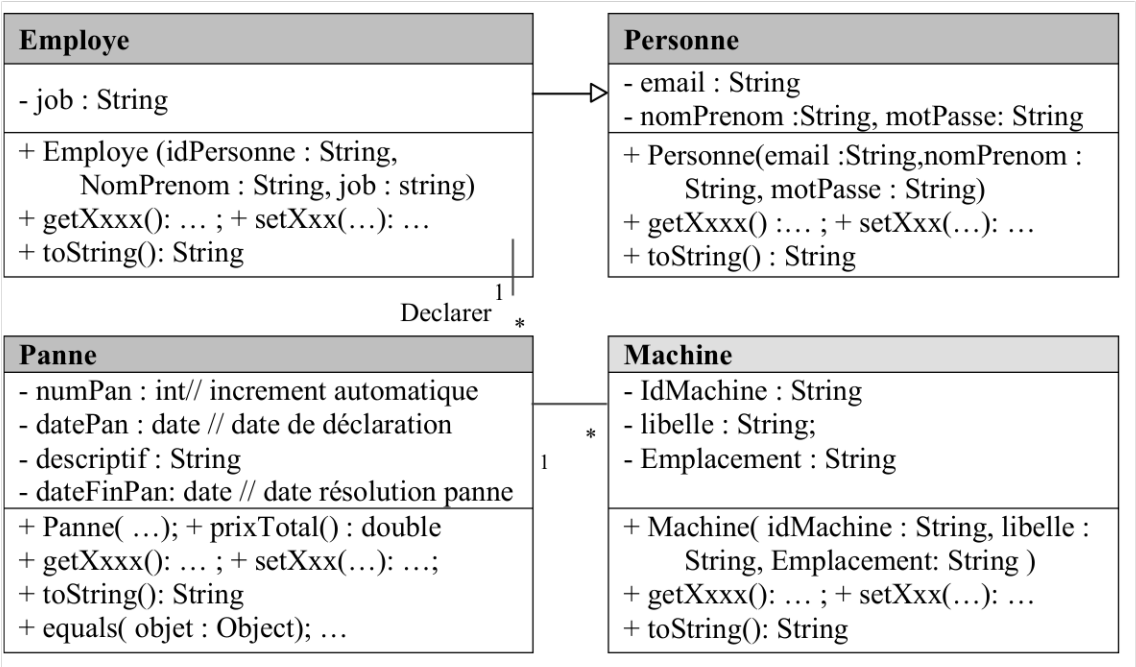
```
package Execpt;
public class InvalidCategourieExeption extends Exception{
  public InvalidCategourieExeption(String msg) {super(msg);}
}
```

Classe Test

```
public class Test {
    public static void main(String[] args)  {
        for (int i=1; i<=5; i++)
            try {
            new Categorie("intiule "+i, 20);}
            catch (InvalidCategourieExeption e){ System.out.println(e.getMessage());}
        List<Article>  listArt= new ArrayList<Article>();
        for (int i=1; i<=5; i++)  listArt.add(
        new Article("Art "+i, (double) i*10, i*100, "intiule 3") );
        ArrayList<Client> listClients= new ArrayList<Client>();
        listClients.add(new Client("ENSAB@Ensb.ma", "ensab", "Ber", "066666"));
        listClients.add(new Client("EST@Est.ma", "est", "Ber", "0363636336"));
        ArrayList<Colis> listColis= new ArrayList<Colis>();
        listColis.add(new NormalColis(11, listClients.get(0)) );
        listColis.add(new ExpressColis(11, listClients.get(0)) );
        listColis.add(new AssuranceColis(11, listClients.get(1), 1000.0D) );
        try {
              for (int i=0; i<3; i++) listColis.get(0).addArticle(listArt.get(i));
              listColis.get(0).ship(); listColis.get(0).addArticle(listArt.get(4));
              } catch (ShipExeption e) {System.out.println(e.getMessage()); }
    System.out.println(listColis.get(0).getAllArticle());
}}
```

Examen 3. Cycle ingenieur SIBD S6 2020-2021

On s'intéresse dans cet exercice à une application JAVA de gestion des déclarations des pannes dans une entreprise de production industrielle. Pour des raisons de simplification, on considère que la modélisation des pannes est représentée par le diagramme suivant :



A. Considérons, dans un premier lieu, les classes POJO du diagramme ci-dessus

1. Donner pour chacune des classes «Employe», «Machine» et «Panne» les attributs à ajouter pour prendre en considération les associations du diagramme. (2pts)
2. Ecrire le code du constructeur des classes « Employe » et « Panne » (4pts)

B. L'accès à la base de données est effectué en utilisant le Framework Hibernate. Ainsi, une classe « ServiceDAO » regroupe tous les services liés à la base de données. La structure de la classe « ServiceDAO » est donnée par le diagramme de la page 2/2 (toutes les méthodes sont considérées static) .

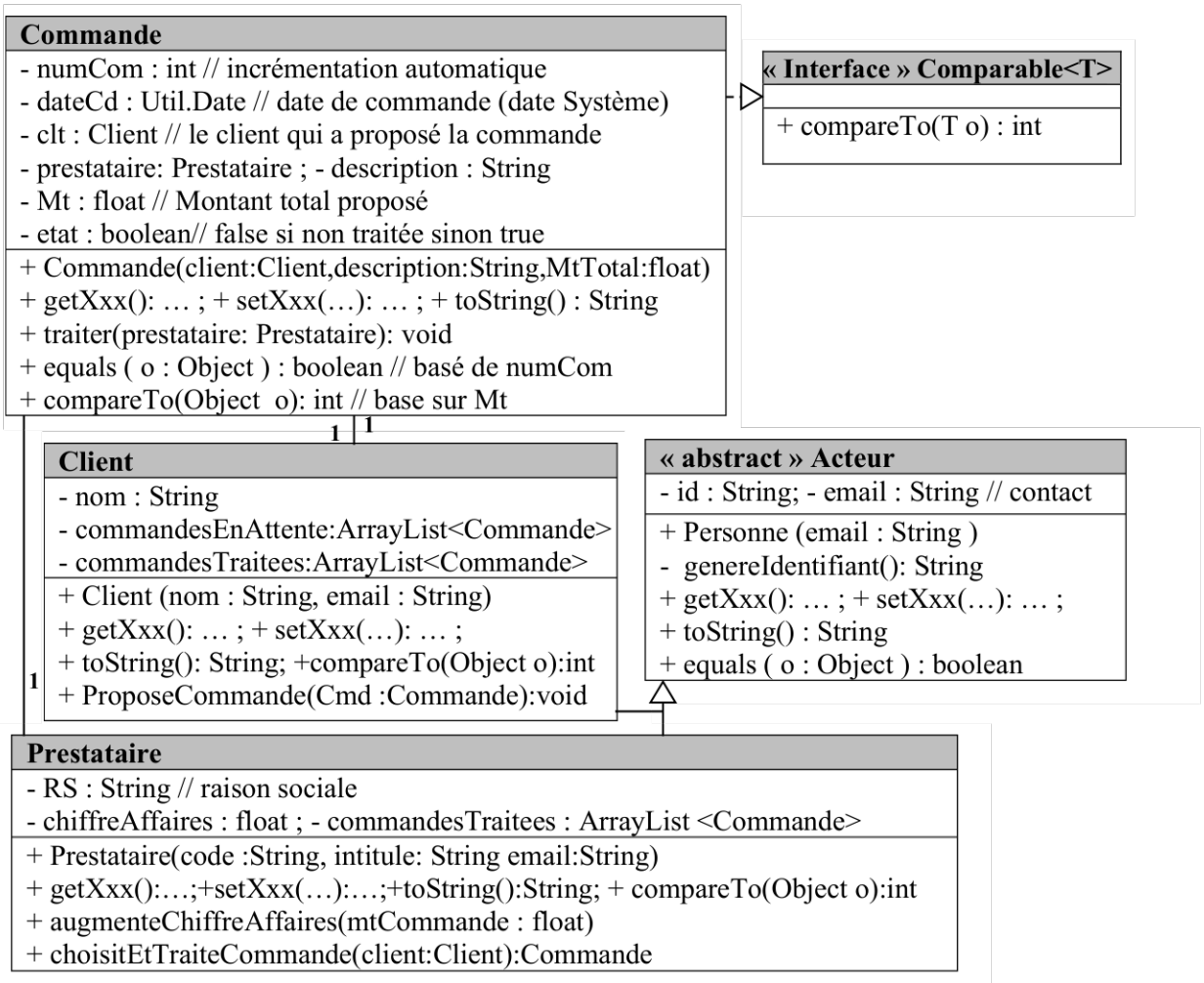
ServiceDAO
+ getAllPersonnes(): List<Personne> // //liste de tous les personnes + getAllEmployes(): List<Personne> // //liste de tous les employés + getPersonne(email :String): Personne //test exitance d'une Personne + getPersonne(email :String, motPass : String) : Personne + addEmploye(employe : Employe) : boolean + getAllMachine(): List<Machine> // //liste de tous les machines + getMachine(IdMachine : String): Machine + addMachine(machine : Machine) : boolean + getAllPannes(): List<Panne> // //liste de tous les pannes + getPanneParDate(datePan : Date): List<Panne> + getPanneDeclareParPeronne(email : String): List<Panne> + getPanneParMachine(IdMachine : String): List<Panne> + addPanne(panne: Panne) : boolean + …

1. Ecrire les fichiers de mapping correspondants aux classes « Personne » et « Panne » (définir uniquement les éléments internes à la racine <hibernate-mapping>). La stratégie d'héritage à utiliser est celle d'une table par hiérarchie de classes. (3pts)
2. Donner le code des méthodes DAO « getPanneParMachine » et « addEmploye ». On suppose que la méthode « addEmploye » déclenche une exception prédéfinie « PersonExistException ». (4pts)

Examen 4. Cycle ingénieur GI S6 2021-2022

On s'intéresse dans ce contrôle à des clients qui proposent des commandes de service à réaliser. Les prestataires de service choisissent parmi ces commandes, celle à traiter selon leur propre stratégie de choix.

A. Considérons dans un premier lieu les trois principales classes «Commande », « Client » et « Prestataire » modélisant le système étudié.



- La méthode « genereIdentifiant » de la classe « Adherent » calcul puis retourne un nouvel identifiant garanti unique. Cette identifiant est affectée à l'attribut « id » de l'adhérent lors de sa création.
- La méthode « ProposeCommande » de la classe « Prestataire » permet d'ajouter une nouvelle commande à la liste des commandes en attente(liste « commandesEnAttente »)

- L'attribut «description» de la classe «commande» définit une description textuelle du service à réaliser. L'attribut «prestataire» correspond au prestataire ayant traité la commande, il est initialisé par la valeur null. La méthode «traiter» assure la définition de l'attribut «prestataire» lorsqu'un prestataire choisie de traiter ladite commande (« etat » prend la valeur true).
- La méthode « choisitEtTraiteCommande » de la classe « Prestataire » sélectionne puis retourne l'une des commandes encore en attente du client passé en paramètre. Pour effectuer cette sélection, le prestataire s'appuie sur sa stratégie de choix. Pour des raisons de simplification on se limite à une seule stratégie de choix chez les prestataires. Il s'agit de celle permettant à ces derniers de maximiser leur chiffre d'affaires en choisissant la commande ayant le montant est le plus élevé parmi les commandes proposées. Deux scénarios sont possibles :
- Le client ne possède aucune commande en attente. Dans ce cas, aucune commande ne peut être choisie et l'exception «CommandNotFoundException» est levé.
- Une commande est choisie. Dans ce cas la commande est considérée traitée et le chiffre d'affaires est augmenté du prix de la commande. Les données relatives à ladite commandes sont ensuite mises à jour pour le prestataire et le client.

1. Définir le constructeur de la classe « Commande » (3pts)
2. Ecrire le code de la méthode « traiter » sachant qu'elle lève une exception nommée « CommandeTraiteException » si la commande a été déjà traitée (3pts)
3. Donner le code du constructeur de la classe « Clients » (3pts)
4. Définir la méthode « choisitEtTraiteCommande » (4pts)

B. Afin de gérer les clients et les prestataires de service, nous considérons une classe «Services» dont la définition est donnée par la représentation graphique ci-dessous.

Donner le code de la méthode « getTotalCommandesEnAttente» permettant de retourner la liste de l'ensemble des commandes en attente. (4pts)

Services
- clients : ArrayList <Client> - prestataires : ArrayList <Prestataire>
+ Services() + getClients : ArrayList <Client> ; getPrestataires : ArrayList <Prestataire> + getClientById(id :String) : Client + getPrestataireById(id :String) : Prestataire + getTotalCommandesEnAttente ():: ArrayList<Commande> + getCommandesEnAttenteByClient (idClient : String) : ArrayList< Commande > + getTotalCommandesTraitees ():: ArrayList<Commande> + getCommandesTraiteesByClient (idClient : String) : ArrayList< Commande > + getCommandesTraiteesByPrestataire (idPrestataire :String) : ArrayList< Commande > + getClient(client: Client) : boolean + addPrestataire(prestataire : Prestataire) : boolean …

C. Définir une classe nommée « TestService » qui contient la méthode « main ». Cette méthode assure les tests suivants (3pts):

- Créer une instance de la classe service
- Ajouter deux clients et un prestataire de service à l'instance crée
- Proposer pour le premier client deux commandes et pour le deuxième client une commande
- Indiquer que la deuxième commande du premier client est traitée par le prestataire crée.

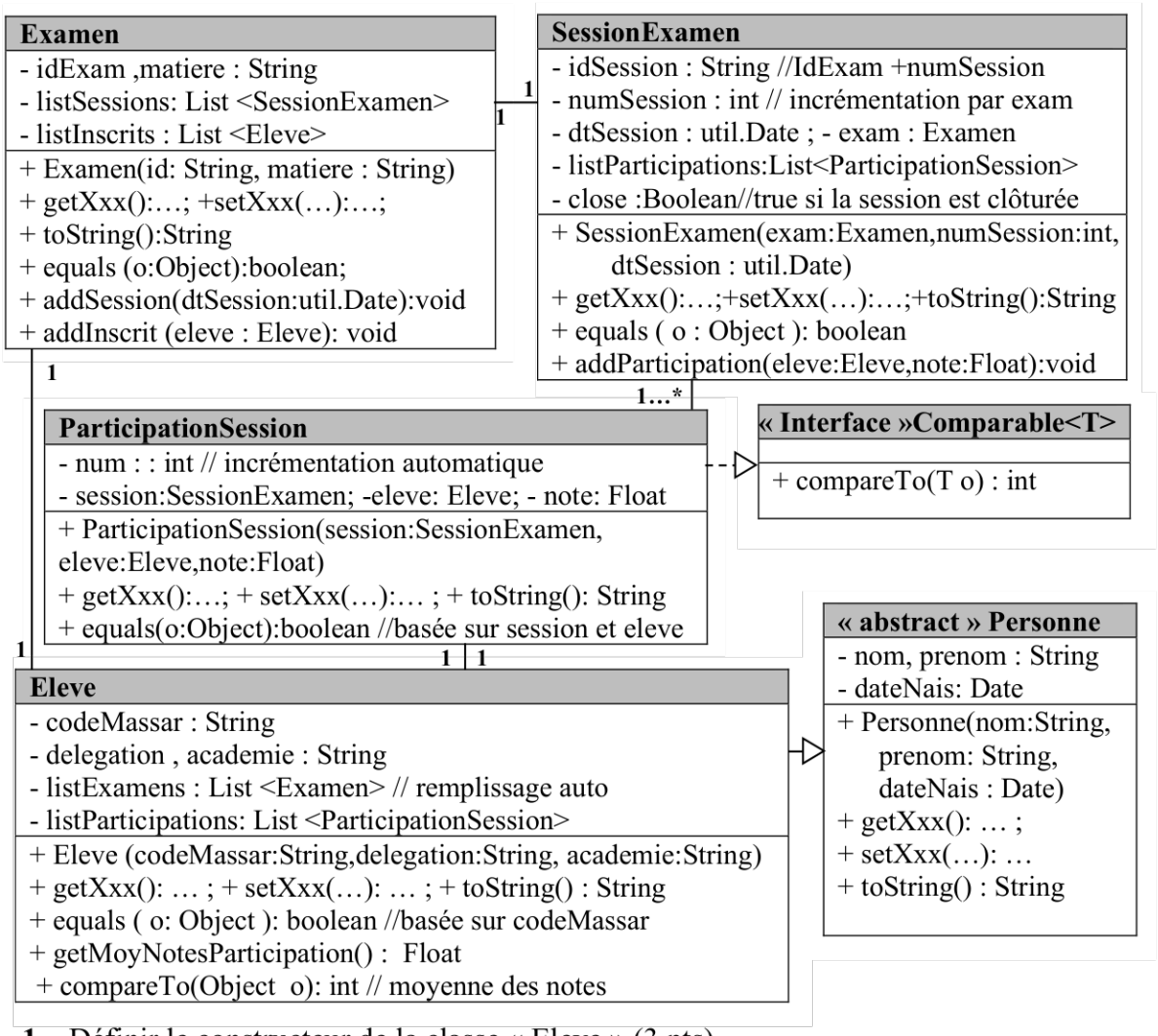
Examen 5. Cycle ingénieur GI S6 2022-2023

On s'intéresse dans ce contrôle à une gestion simplifié des examens nationaux des olympiades organisées par les différentes Académies Régionales D'éducation Et Formation (AREF) du Royaume. Les compétitions sont organisées par les académies qui gèrent les différents élèves de leurs délégations. Ainsi, un élève se voit attribuer des notes en fonction des résultats qu'il obtient à l'occasion de sa participation aux différentes sessions de ces examens.

A. Considérons dans un premier lieu le diagramme représentant les cinq principales classes «Examen », « SessionExamen», « ParticipationSession», « Eleve » et « Personne» modélisant le système étudié. Dans ce diagramme, les cardinalités sont mentionnées à titre indicatif puisqu'elles sont représentées dans les classes (objets membres et collections). Les différentes interfaces « List » du diagramme peuvent être implémentées par une classe collection de votre choix (ArrayList par exemple).

- La classe « Eleve » implémente l'interface « Comparable ». Pour comparer 2 élèves, nous utilisons la moyenne des notes de leurs participations.
- La méthode « addSession » de la classe « Examen » assure la création d'une nouvelle instance de la classe « sessionExamen » et l'ajout de cette dernière à la collection « listSessions » de la classe « Examen ». Le numéro de cette nouvelle session est obtenu par une incrémentation relative à l'examen (faire une incrémentation p/p au dernier numéro des sessions d'un examen).
- La méthode « addInscrit» de la classe « Examen » permet d'ajouter une instance « eleve » à la collection « listInscrits » de la classe « Examen ». Cette opération assure l'ajout automatique de l'instance correspondante à la collection « listExamens » de la classe « Eleve »
- La méthode « addParticipation» de la classe « sessionExamen » permet de créer une instance de la classe « ParticipationSession » et d'ajouter l'instance créée aux collections « listParticipations» des classes « sessionExamen » et « Eleve ». Cette méthode lève l'exception « CloseSessionException » si la session correspondante est fermée (attribut close est égale à « true »). La méthode « addParticipation» peut également lever l'exception « InscriptionException » lorsque l'élève reçu en paramètre n'est pas inscrit

dans l'examen relatif à la session. On suppose que les deux exceptions « CloseSessionException » et « InscriptionException » sont définies.



1. Définir le constructeur de la classe « Eleve » (3 pts)
2. Donner le code de la méthode « addSession » de la classe « Examen » (3 pts)
3. Ecrire le code de la méthode « addParticipation» de la classe « sessionExamen » (3 pts)
4. Définir la méthode « compareTo » de la classe « Eleve » (3 pts)

B. Afin de gérer les inscriptions et les participations aux différentes session des examens des olympiades, nous considérons une classe «Services» dont la définition est donnée

par la représentation graphique suivante (tous les attributs et les méthodes de cette classe sont considérées static) :

Services
- eleves : List <Eleve> - examens : List <Examen>
+ addEleve(eleve: Eleve) : boolean + addExamen(examen :Examen) : boolean + getAllEleves () : List <Eleve> ; + getAllExamens() : List<Examen> + getEleveByCodeMassar (codeMassar : String) : Eleve + getElevesByDelegation (delegation: String) : List <Eleve> + getElevesByAcademie(academie: String) : List <Eleve> + getExamenById(idExam : String) : Examen + getIscritsByExamen(idExam : String) : List<Eleve> + getSessionById(idsession : String) : SessionExamen + getCloseSessionsByExamen(idExam : String) : List<SessionExamen> + getParticipationsByExamen(idExam : String) : List<ParticipationSession> + getParticipationsBySession(idsession: String) : List<ParticipationSession> + getParticipationsByEleve(codeMassar : String) : List<ParticipationSession> + getParticipationsByMatiere(matiere : String) : List<ParticipationSession> + getMeilleurParticipationsByExamen(idExam : String) : ParticipationSession + getMeilleurEleveByExamen(idExam : String) : Eleve …

1. Donner le code de la méthode « getCloseSessionsByExamen» permettant de retourner la liste des sessions clôturées relatives à l'examen dont l'identifiant (idExam) est reçu en paramètre (2 pts)

2. Ecrire le code de la méthode « getMeilleurEleveByExamen» permettant de retourner le meilleur élève (ayant la plus grande moyenne) parmi les élèves inscrits dans l'examen dont l'identifiant (idExam) est reçu en paramètre (3 pts)

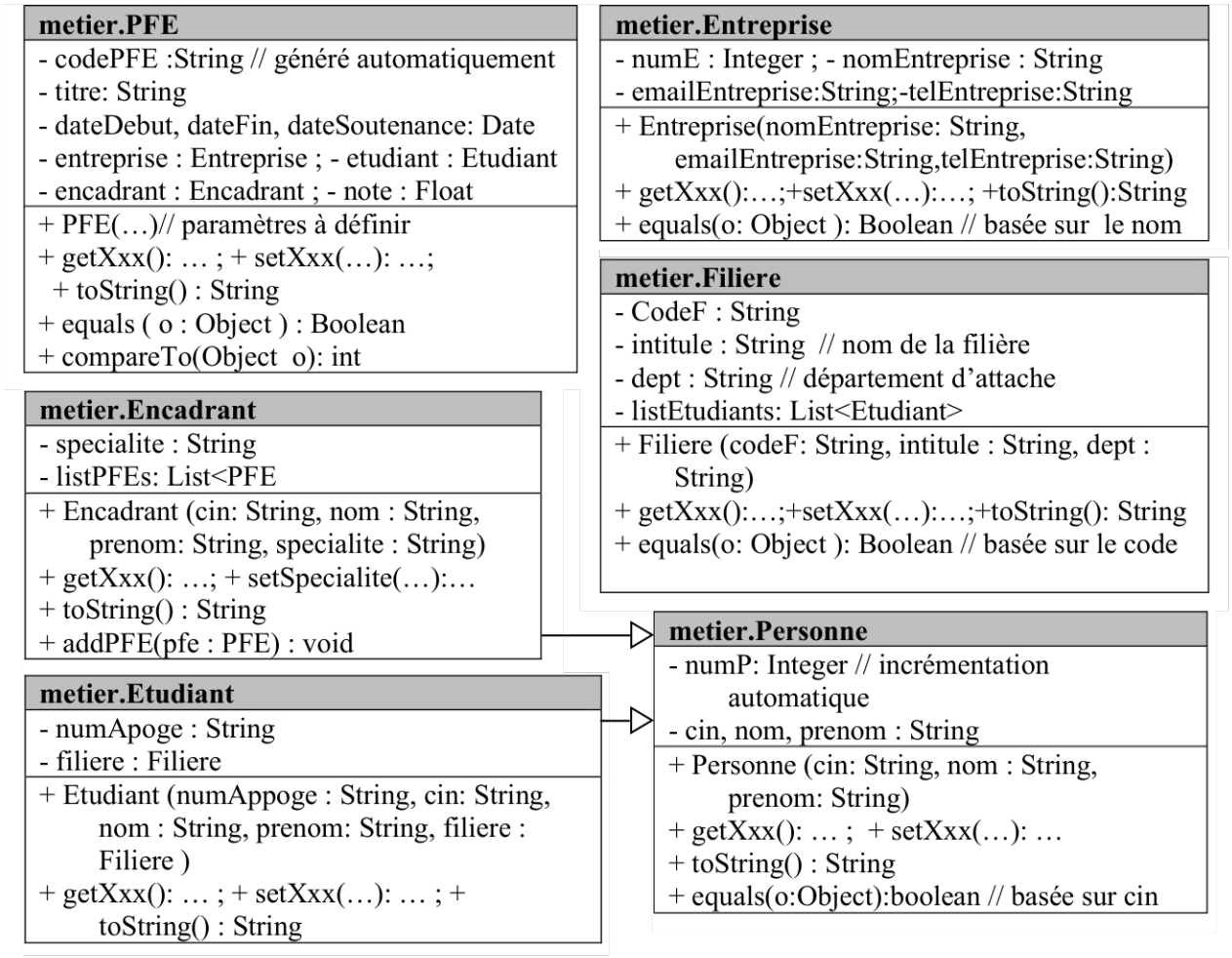
C. Définir une classe nommée « TestService » , contenant une méthode « main », qui assure les tests suivants (3 pts):

- Créer deux élèves « el1 » et « el2 » et un examen « ex » , ajouter ces instances aux collections de la classe « Service » puis inscrire les deux élèves « el1 » et « el2 » dans l'examen « ex ».
- Créer 2 sessions « s1 » et « s2 » de l' examen « ex » puis attribuer à l'élève « el1 » pour les sessions « s1 » et « s2 » les notes respectives suivantes : 11.5 et 13.4
- Afficher les participations relatives à l'examen « ex »

Examen 6. Cycle ingénieur GI S6 2023-2024

On s'intéresse dans cet examen à une application JAVA permettant de gérer les projets de fin d'étude (PFE) d'une école d'ingénieurs. Pour des raisons de simplification liée à l'examen, on considère qu'un PFE est réalisé par un seul étudiant et encadré par un seul enseignant (encadrant). Ce projet est effectué dans une entreprise.

A. On considère dans un premier lieu les classes métiers de l'application (placées dans un package« metier ») dont l'implémentation JAVA est donnée par le diagramme suivant :



- Les différents attributs sont considérés privé (« private ») alors que toutes les méthodes sont publiques (« public »).
- Lors de la création d'une nouvelle l'instance des classes « PFE » et « Etudiant », l'instance créée sera ajoutée (automatiquement) respectivement à la collection « listPFEs » de la classe « Encadrant » et à la collection « listEtudiants » de la classe « Filiere ».
- La classe « PFE » implémente l'interface « Comparable ». Le code d'un nouvel PFE est généré automatique, il sera formé par la concaténation du code de la filière , de l'année du PFE (utiliser l'année de la date du début de PFE) et du code apogée de l'étudiant (voir l'annexe pour la récupération de l'année d'une date)

1. Définir le constructeur de la classe « Etudiant» ; (2pts)
2. Définir le constructeur de la classe « encadrant» ; (2pts)
3. Définir le constructeur de la classe « PFE» ; (3pts)

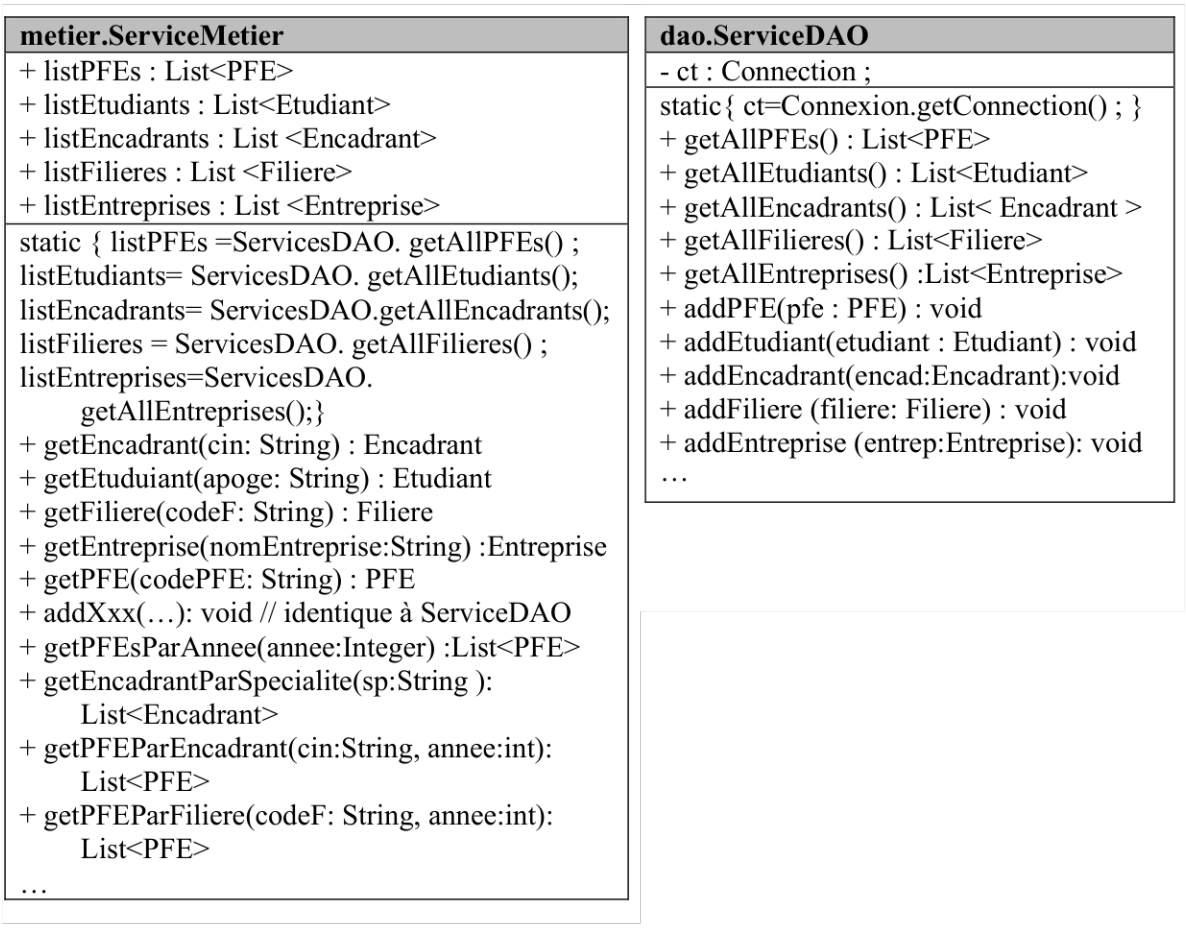
B. Dans la suite de cet examen et pour simplifier le mapping objet/relationnelle, nous considérons que la classe « Personne » est supprimée et que son contenu sera dupliqué dans les classes « Etudiant» et « Encadrant ». le diagramme ci-dessous représente la duplication correspondante.

metier.Encadrant
- cin , nom, prenom : String - specialite : String - listPFEs: List<PFE>//remplissage auto
+ Encadrant (cin: String, nom : String, prenom: String, specialite : String) + getSpecialite():…; +setSpecialite …):… + toString() : String + addPFE(pfe : PFE) : void + equals(o:Object):boolean

metier.Etudiant
- numApoge : String ; - cin: String - nom, prenom : String ; - filiere : Filiere
+ Etudiant (numAppoge : String, cin: String, nom : String, prenom: String, filiere : Filiere) + getXxx(): … ; + setXxx(…): … ; + toString() : String + equals(o:Object):boolean // basée sur cin

Les informations des PFEs sont enregistrées dans une base de données MySQL nommée « dbPFEs ». Les différentes méthodes DAO, utilisant JDBC et considérées static, sont regroupés dans une classe « ServiceDAO » d'un package « dao » (On considère une classe « Connexion » , placée dans ce même package, permettant d'obtenir une connexion unique à la base de données « dbPFEs »). Pour faciliter l'accès aux méthodes de la classe «ServiceDAO » , nous définition une classe «ServiceMetier» du package

« metier » (tous les attributs et les méthodes de cette classe sont considérés static). Ainsi, une méthode « addXxx» de la classe « ServiceDAO » est appelée par la méthode de la classe « ServiceMetier » qui lui correspond.



Le bloc static de la classe «ServiceMetier» assure le remplissage de ses collections (considérées static et publiques) lors du démarrage de l'application. De même, l'attribut « ct » (considéré static et privé) est une instance assurant la connexion à la base de données « dbPFEs ».

Schéma de la base de données « dbVentes »

- **Encadrant**(cin: varchar(10), nom : varchar(20), prenom : varchar(20), specialite : varchar(20))
- **Etudiant**(numApoge : varchar(10), cin: varchar(10), nom : varchar(20), prenom : varchar(20), codeF#:varchar(6))
- **PFE**(codePFE: varchar(20), titre: varchar(250), dateDebut : date, dateFin : date, dateSoutenance: date, numE# : int, numApogeE# : varchar(10), cinEncadrant#: varchar(10), note : Real)
- **Entreprise**(numE: int, nomEntreprise : varchar(50), emailEntreprise : varchar(20), telEntreprise: varchar(10))
- **Filiere**(CodeF: varchar(6), intitule: varchar(50), dept: varchar(20))

Les clés primaires sont soulignées alors que les clés étrangères sont suivies par # (caractère pour indication uniquement et ne fait pas partie de l'appellation des champs)

1. Donner le code de la méthode « addPFE » de la classe «ServiceMetier». Cette méthode lève l'exception «InvalidPFEException» (supposée déjà définie) lorsque le nombre de mois d'un PFE est inférieur à 4 mois (utiliser l'annexe pour la récupération du numéro du mois d'une date. Le nombre de mois d'un PFE est la différence entre les numéros des mois associés aux attributs « dateDebut » et « dateFin ») ou lorsqu'il existe un autre projet ayant le même code (3pts)
2. Définir la méthode «getAllEntreprise» de la classe «ServiceDao» ; (3pts)
3. Ecrire le code de la méthode «getEncadrantParSpecialite» permettant de retourner la liste des encadrants ayant une spécialité donnée (paramètre fournie en argument). (3pts)

C. Définir une classe nommée « TestService » , contenant une méthode « main », qui assure les tests suivants (4 pts):

- Créer deux filières (intitulé1="GI", intitule2="SIBD") et ajouter à la base de données un étudiant pour chacune de ces filières
- Ajouter à la base de données deux encadrants (nom1 ="JAVA" , nom2="JEE")
- Ajouter deux entreprises (nom1= "Oracle", nom2="OpenIA")
- Pour chacun des étudiants crée, ajouter un PFE dont l'encadrant est l'un des encadrant précédemment ajouté

- Pour la liste des PFEs de l'année en cours (utiliser l'annexe ou utiliser directement « Calendar.getInstance().get(Calendar.YEAR) pour récupérer un entier représentant cette année) .

Annexe : Exemple de récupération de l'année d'une date en java

```
import java.util.Date;
import java.text.SimpleDateFormat;

Date dt;
SimpleDateFormat formatDt;
dt= new Date();// date courante
formatDt= new SimpleDateFormat("yyyy");// yyyy : année - mm :mois – dd : jour
int anDt= Integer.parsInt(formatDt.format(dt));
```