Corrigé des exercices avancés sur les listes

Exercice 1.3.1 *utilisation de listes*

Nous allons encore une fois utiliser le type `ListeIter` vu en cours.

1. Ecrire une méthode qui calcule la somme des éléments positifs d'une liste d'entiers.
2. Ecrire une méthode qui vérifie si une liste est triée en ordre croissant.

```
class Exo23_1{
    static int sommePositif(ListeIter l){
        int somme = 0;
        ElementListe aux = l.getPremier();
        while (aux != null){
            if (aux.getValeur()>0){
                somme = somme + aux.getValeur();
            }
            aux = aux.getSuivant();
        }
        return somme;
    }
    static boolean estTriee(ListeIter l){
        ElementListe aux = l.getPremier();
        if (aux == null){
            return true;
        }
        int dernierVu = aux.getValeur();
        aux = aux.getSuivant();
        while (aux != null){
            if (aux.getValeur() < dernierVu){
                return false;
            }
            dernierVu = aux.getValeur();
            aux = aux.getSuivant();
        }
        return true;
    }
    public static void main(String[] args){
        ListeIter lal = new ListeIter();
        lal.ajouterAuDebut(4);
        lal.ajouterAuDebut(3);
```

```
        lal.ajouterAuDebut(2);
        lal.ajouterAuDebut(-1);
        Terminal.ecrireIntln(sommePositif(lal));
        Terminal.ecrireBooleanln(estTriee(lal));
        lal.ajouterAuDebut(5);
        Terminal.ecrireBooleanln(estTriee(lal));
    }
}
```

Exercice 1.3.2 *égalité de listes*

Il y a deux égalités de listes possibles : l'égalité absolue, testée par == qui teste en fait si deux expressions Java sont deux moyen de parler de la même liste, du même objet.

Le plus souvent, on s'intéresse à une égalité du contenu des listes : deux listes sont égales si

- elles ont la même longueur
- pour tout rang, l'élément de l'un situé à ce rang est égal à l'élément de l'autre liste situé à ce rang.

Ecrire une méthode qui teste si deux listes sont égales.

```
class Exo23_2{
    static boolean egales(ListeIter l1, ListeIter l2){
        ElementListe aux1, aux2;
        aux1 = l1.getPremier();
        aux2 = l2.getPremier();
        while ((aux1 != null) && (aux2 != null)){
            if (aux1.getValeur() != aux2.getValeur()){
                return false;
            }
            aux1 = aux1.getSuivant();
            aux2 = aux2.getSuivant();
        }
        return aux1 == aux2;
    }
    public static void main(String[] args){
        ListeIter la, lb, lc;
        la = new ListeIter();
        la.ajouterAuDebut(1);
        la.ajouterAuDebut(2);
        la.ajouterAuDebut(3);
        lb = new ListeIter();
        lb.ajouterAuDebut(1);
        lb.ajouterAuDebut(2);
        lb.ajouterAuDebut(3);
        lc = new ListeIter();
        lc.ajouterAuDebut(5);
        lc.ajouterAuDebut(2);
        lc.ajouterAuDebut(3);
        Terminal.ecrireBooleanln(egales(la,lb));
        Terminal.ecrireBooleanln(egales(la,lc));
    }
}
```

Le principe de cette méthode consiste à tester l'égalité des éléments de même rang, en commençant par le début. Si on trouve deux éléments différents, les listes ne sont pas égales. Sinon on avance d'un cran dans chaque liste. On s'arrête quand on est au bout d'une des deux listes (une des deux listes vaut `null`). Il reste alors à tester qu'on est au bout des deux listes, sinon, c'est qu'il y a en a une de plus longue que l'autre.

Exercice 1.3.3 *listes triées*

Cet exercice utilise la classe `ListeTriee` vue en cours.

1. Ajouter dans la classe une méthode `suppressionToutesOccurrences` qui supprime toutes les occurrences d'un élément dans une liste triée (note : ces occurrences sont forcément l'une à la suite de l'autre).
2. Ajouter une méthode qui élimine les doublons dans la liste.
3. Ajouter une méthode qui permette de fusionner deux listes triées pour obtenir une nouvelle liste triée.

```
class ListeTriee2{
    int element;
    ListeTriee2 suivant;

    public int premier(){
        return element;
    }
    public void modifiePremier(int elem){
        element = elem;
    }
    public void modifieReste(ListeTriee2 reste){
        suivant = reste;
    }
    public ListeTriee2 reste(){
        return suivant;
    }
    public ListeTriee2(int premier, ListeTriee2 reste){
        element = premier;
        suivant = reste;
    }
    public boolean contientTriee(int elem){
        ListeTriee2 ref=this;
        while(ref != null && ref.premier() <= elem){
            if(ref.premier() == elem)
                return true;
            else ref=ref.reste();
        }
        return false;
    }
    public ListeTriee2 insertionTriee(int elem){
        ListeTriee2 ref=this; //référence qui cherche elem
        ListeTriee2 pred=null; //prédécesseur, initialisé à null
```

```
        while(ref != null && ref.premier() < elem){ //recherche position
            pred = ref; //avance pred
            ref = ref.reste(); //avance ref
        }
        if(pred == null) //insertion au début
            return new ListeTriee2(elem, this);
        else{ //insertion au milieu de la liste
            ListeTriee2 nouveau = new ListeTriee2(elem, ref);
            pred.modifieReste(nouveau); //modifie le suivant du prédécesseur
            return this; //première cellule non modifiée
        }
    }
    public ListeTriee2 suppressionTriee(int elem){
        ListeTriee2 ref=this; //référence qui cherche elem
        ListeTriee2 pred=null; //prédécesseur, initialisé à null
        while(ref != null && ref.premier() < elem){
            pred = ref; //avance pred
            ref = ref.reste(); //avance ref
        }
        if(ref != null && ref.premier() == elem){
            if(pred == null)
                return ref.reste();
            else{
                pred.modifieReste(ref.reste());
                return this;
            }
        }
        else return this;
    }
    void ecrireListe(){
        ListeTriee2 parcours = this;
        while (parcours != null){
            Terminal.ecrireInt(parcours.premier());
            if (parcours.suivant != null){
                Terminal.ecrireString(",␣");
            }
            parcours = parcours.reste();
        }
    }
    ListeTriee2 suppressionToutesOccurrences(int elem){
        ListeTriee2 ref=this; //référence qui cherche elem
        ListeTriee2 pred=null; //prédécesseur, initialisé à null
        ListeTriee2 suiv;
        while(ref != null && ref.premier() < elem){
            pred = ref; //avance pred
            ref = ref.reste(); //avance ref
        }
        if(ref != null && ref.premier() == elem){
            suiv = ref;
            while (suiv != null && suiv.premier() == elem){
                suiv = suiv.reste();
            }
            if(pred == null)
```

```
                return suiv;
            else{
                pred.modifieReste(suiv);
                return this;
            }
        }
        else return this;
    }
    ListeTriee2 sansDoublon(){
        ListeTriee2 ref = this;
        while (ref.reste() != null){
            if (ref.premier() == ref.reste().premier()){
                ref.modifieReste(ref.reste().reste());
            }else{
                ref = ref.reste();
            }
        }
        return this;
    }
    ListeTriee2 fusion(ListeTriee2 l){
        ListeTriee2 res = new ListeTriee2(0,null);
        ListeTriee2 finres = res;
        ListeTriee2 ref = this;
        while (l != null && ref != null){
            if (l.premier() < ref.premier()){
                finres.modifieReste(new ListeTriee2(l.premier(),null));
                finres = finres.reste();
                l = l.reste();
            }else{
                finres.modifieReste(new ListeTriee2(ref.premier(),null));
                finres = finres.reste();
                ref = ref.reste();
            }
        }
        while (l != null){
            finres.modifieReste(new ListeTriee2(l.premier(),null));
            finres = finres.reste();
            l = l.reste();
        }
        while (ref != null){
            finres.modifieReste(new ListeTriee2(ref.premier(),null));
            finres = finres.reste();
            ref = ref.reste();
        }
        return res.reste();
    }
}
class Exo23_3{
    public static void main(String[] args){
        ListeTriee2 ex1 = new ListeTriee2(1,null);
        ListeTriee2 ex2 = new ListeTriee2(1,null);
        ex1 = ex1.insertionTriee(6).insertionTriee(3);
        ex1 = ex1.insertionTriee(6).insertionTriee(3);
```

```
        ex1 = ex1.insertionTriee(4).insertionTriee(3);
        ex1 = ex1.insertionTriee(2).insertionTriee(1);
        ex2 = ex2.insertionTriee(6).insertionTriee(5);
        ex2 = ex2.insertionTriee(6).insertionTriee(10);
        ex2 = ex2.insertionTriee(8).insertionTriee(9);
        Terminal.ecrireString("ex1:␣");
        ex1.ecrireListe();
        Terminal.sautDeLigne();
        ex1.suppressionToutesOccurrences(6);
        Terminal.ecrireString("ex1␣sans␣6:␣");
        ex1.ecrireListe();
        Terminal.sautDeLigne();
        ex1.sansDoublon();
        Terminal.ecrireString("ex1␣sans␣doublon:␣");
        ex1.ecrireListe();
        Terminal.sautDeLigne();
        Terminal.ecrireString("ex2:␣");
        ex2.ecrireListe();
        Terminal.sautDeLigne();
        Terminal.ecrireString("fusion␣ex1␣et␣ex2:␣");
        ex1.fusion(ex2).ecrireListe();
        Terminal.sautDeLigne();
    }
}
```