Corrigé des exercices avancés sur les structures récursives

Exercice 8.3.1 *utilisation de listes*

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.

```
public class ListeEntier {
    private int valeur;
    private ListeEntier suivant;
    public ListeEntier(int val, ListeEntier suiv) {
        this.valeur = val;
        this.suivant = suiv;
    }
    public ListeEntier(int val) {
        this.valeur = val;
        this.suivant = null;
    }
    public int getValeur() {
        return valeur;
    }
    public void setValeur(int val) {
        this.valeur = val;
    }
    public ListeEntier getSuivant() {
        return suivant;
    }
    public void setSuivant(ListeEntier suiv) {
        this.suivant = suiv;
    }
    public void affiche(){
        Terminal.ecrireString(valeur + "␣");
        if (suivant != null)
            suivant.affiche();
        else
            Terminal.sautDeLigne();
    }
    int sommePositif(){
        int somme = 0;
        ListeEntier aux = this;
```

```
        while (aux != null){
            if (aux.getValeur()>0){
                somme = somme + aux.getValeur();
            }
            aux = aux.getSuivant();
        }
        return somme;
    }
    boolean estTriee(){
        ListeEntier aux = this;
        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){
        ListeEntier lal = new ListeEntier(4);
        lal= new ListeEntier(3,lal);
        lal= new ListeEntier(2,lal);
        lal= new ListeEntier(-1,lal);
        Terminal.ecrireIntln(lal.sommePositif());
        Terminal.ecrireBooleanln(lal.estTriee());
        lal= new ListeEntier(5,lal);
        Terminal.ecrireIntln(lal.sommePositif());
        Terminal.ecrireBooleanln(lal.estTriee());
    }
}
```

Exercice 8.3.2 *égalité de structures récursives*

1. écrire une méthode qui teste si deux courses de la même saison ont le même classement.
2. écrire une méthode qui teste si deux courses de deux saisons différentes ont le même classement. On ne doit pas tenir compte du dossard.
3. écrire une méthode qui teste si deux courses ont le même podium (trois premiers arrivés).

```
public class Coureur{
    private String nom;
    private int dossard;
    private Coureur suivant;
    public Coureur(String n, int d, Coureur suiv) {
        this.nom = n;
```

```
        this.dossard = d;
        this.suivant = suiv;
    }
    public String getNom() {
        return nom;
    }
    public int getDossard() {
        return dossard;
    }
    public Coureur getSuivant() {
        return suivant;
    }
    public boolean memeClassement(Coureur autre){
        Coureur cl1 = this;
        Coureur cl2 = autre;
        boolean different = false;
        while (cl1 != null && cl2 != null && !different){
            if (!cl1.getNom().equals(cl2.getNom()))
                different = true;
            else{
                cl1=cl1.getSuivant();
                cl2=cl2.getSuivant();
            }
        }
        if (different) //coureurs differents a une position
            return false;
        else if (cl1 != null) // plus de coureurs dans this
            return false;
        else if (cl2 != null) // plus de coureurs dans autre
            return false;
        else // meme longueur
            return true;
    }
    public boolean memePodium(Coureur autre){
        int i = 3;
        Coureur cl1 = this;
        Coureur cl2 = autre;
        boolean different = false;
        while (cl1 != null && cl2!=null && i>0 && !different){
            if (!cl1.getNom().equals(cl2.getNom()))
                different = true;
            else{
                cl1=cl1.getSuivant();
                cl2=cl2.getSuivant();
                i = i-1;
            }
        }
        if (i!=0) // une des deux courses a moins de trois coureurs
            return false;
        else if (different)
            return false; // un des trois premiers est different
        else
            return true;
```

```
    }
    public void afficheClassement(){
        int rang = 1;
        Coureur aux = this;
        while (aux != null){
            Terminal.ecrireStringln(rang + ": " + aux.getDossard() + " " +
                                    aux.getNom());
            rang++;
            aux = aux.getSuivant();
        }
    }
    public static void main(String[] args){
        Coureur c1 = new Coureur("donald",125,null);
        c1 = new Coureur("picsou",103,c1);
        c1 = new Coureur("fifi",129,c1);
        c1 = new Coureur("loulou",155,c1);
        c1 = new Coureur("riri",157,c1);
        Coureur c2 = new Coureur("fifi",77,null);
        c2 = new Coureur("loulou",55,c2);
        c2 = new Coureur("riri",12,c2);
        c1.afficheClassement();
        Terminal.sautDeLigne();
        c2.afficheClassement();
        Terminal.ecrireBooleanln(c1.memeClassement(c2));
        Terminal.ecrireBooleanln(c1.memePodium(c2));
    }
}
```

Exercice 8.3.3 *voyage*

On veut représenter un itinéraire comme une suite de moyens de transports publics à emprunter (métro, bus, train, autocar, avion) avec éventuellement des transitions à pied. Sur cet itinéraire, il y a des lieux représentés par le nom de la ville et un complément qui peut-être le nom d'une gare, d'une station de métro, d'un arrêt de bus ou une adresse. On va représenter les lieux au moyen d'objets d'une classe des lieux et les liaisons entre lieux au moyen d'objets d'une autre classe. Une liaison comprendra le type de transport, la distance et éventuellement, les heures de départ et d'arrivée si cela a un sens pour le moyen de transport considéré. Par exemple, un train ou un avion ont des horaires précis, mais un métro ou la marche à pied n'ont pas d'horaire précis.

Un itenéraire va être une structure contenant alternativement des lieux et des liaisons (un lieu, une laison, un lieu, une liaison, un lieu, etc).

Ecrivez les classes nécessaires avec une méthode d'affichage d'un itinéraire. Ecrivez une méthode main pour prendre le cas d'une marche à pied du 3 rue Conté, 75003 Paris à la station de métro République (Paris), puis la ligne 5 jusqu'à gare du Nord et le train de 18H17 de Paris à Lille, avec affichage de cet itinéraire.

```
class Lieu{
    String ville;
    String detail;
    Liaison liaison;
    Lieu(String v, String d, Liaison l){
```

```
        ville=v;
        detail=d;
        liaison=l;
    }
    void afficher(){
        Terminal.ecrireStringln(ville + "␣" + detail);
        if (liaison !=null)
            liaison.afficher();
    }
}
class Liaison{
    String type;
    double dist;
    Lieu destination;
    Liaison(String t, double d, Lieu l){
        type=t;
        dist=d;
        destination=l;
    }
    void afficher(){
        Terminal.ecrireStringln("␣␣␣␣" + type + "␣␣" + dist + "␣km");
        destination.afficher();
    }
    public static void main(String[] args){
        Lieu lelieu;
        Liaison trajet;
        lelieu=new Lieu("Lille","gare␣SNCF",null);
        trajet=new LiaisonHoraire("train",220,lelieu,"18h17","19h23");
        lelieu=new Lieu("Paris","gare␣du␣Nord",trajet);
        trajet=new Liaison("métro␣ligne␣5",3.2,lelieu);
        lelieu=new Lieu("Paris","station␣Republique",trajet);
        trajet=new Liaison("a␣pied",1.1,lelieu);
        lelieu=new Lieu("Paris","2␣rue␣Conté",trajet);
        lelieu.afficher();
    }
}
class LiaisonHoraire extends Liaison{
    String depart, arrivee;
    LiaisonHoraire(String t, double d, Lieu l, String h1, String h2){
        super(t,d,l);
        depart=h1;
        arrivee=h2;
    }
    void afficher(){
        Terminal.ecrireString("␣␣␣␣" + type + "␣␣" + dist + "␣km␣depart␣");
        Terminal.ecrireStringln(depart + "␣arrivee␣" + arrivee);
        destination.afficher();
    }
}
```