```java
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

/**
 *
 * @author charles
 *
 * @param <E>
 */
public class LinkedBinaryTree<E> implements RBinaryTree<E>, Position<E>{
    //variable d'instance:
    private LinkedBinaryTree<E> left;
    private LinkedBinaryTree<E> right;
    private E element;
    private int size;

    //constructeur:
    public LinkedBinaryTree(E elem, LinkedBinaryTree<E> leftChild, LinkedBinaryTree<E> rightChild) {
        element = elem;
        left = leftChild;
        right = rightChild;
        if (elem == null) size = 0;// pas d'élément dans la liste ..
        else size = 1; // si il y a un element la taille est de 1
        if(left != null) size+=left.size(); // rajoute de la taille des enfants
        if(right != null) size+=right.size();
    }
    /*
     * @post: implémentation de l'interface position
     */
    @Override
    public E element() {
        return element;
    }
    /*
     * @post: implémentation de l'interface RBinaryTree, renvoie true si l'arbre est vide
     */
    @Override
    public boolean isEmpty() {
```

```
40          return (size == 0);
41      }
42      /*
43       * @post: implémentation de l'interface RBinaryTree, renvoie
   la taille de l'arbre
44       */
45      @Override
46      public int size() {
47          return size;
48      }
49      /*
50       * @post: implémentation de l'interface RBinaryTree, renvoie
   la position de root.
51       *       Mais vu que nous implémentons le fait qu'un arbre
   est soit vide, soit
52       *       il contient un noeud racine avec un fils gauche et
   un fils droite.
53       *  Dès lors, chaque noeud est root, nous utiliserons donc
   cette fonction pour obtenir
54       *  la position d'un élément.
55       */
56      @Override
57      public Position<E> root() {
58          return this;
59      }
60      /*
61       * @post : implémentation de l'interface RBinaryTree, renvoie
   true si l'arbre ne possède pas d'enfant
62       */
63      @Override
64      public boolean isLeaf() {
65          return ( (left == null) && (right == null));
66      }
67      /*
68       * @post : implémentation de l'interface RBinaryTree, renvoie
   le fils gauche
69       */
70      @Override
71      public LinkedBinaryTree<E> leftTree() {
72          return left;
73      }
74      /*
75       * @post : implémentation de l'interface RBinaryTree, renvoie
   le fils droit
```

```
 76      */
 77     @Override
 78     public LinkedBinaryTree<E> rightTree() {
 79         return right;
 80     }
 81     /*
 82      * @post : implémentation de l'interface RBinaryTree, permet
    de modifier l'élement
 83      */
 84     @Override
 85     public void setElement(E o) {
 86         element = o;
 87     }
 88     /*
 89      * @post : implémentation de l'interface RBinaryTree, permet
    de remplacer/ ajouter un fils gauche
 90      */
 91     @Override
 92     public void setLeft(RBinaryTree<E> tree) {
 93         left = (LinkedBinaryTree<E>) tree;
 94     }
 95     /*
 96      * @post : implémentation de l'interface RBinaryTree, permet
    de remplacer/ ajouter un fils droit
 97      */
 98     @Override
 99     public void setRight(RBinaryTree<E> tree) {
100         right = (LinkedBinaryTree<E>)tree;
101 
102     }
103 
104     // Itérateur d'éléments grace a un itérateur de position:
105     public class ElementIterator implements Iterator<E>{
106         Iterator<Position<E>> posIterator =
    positions().iterator();
107         public boolean hasNext(){ return posIterator.hasNext();}
108         public E next(){ return posIterator.next().element();}
109         public void remove(){posIterator.remove();}
110     }
111     public Iterator<E> iterator(){ return new ElementIterator();}
112 
113     //fonction de "tri" "non ordonné" récursive de la liste:
114     private void inorderSubtree(LinkedBinaryTree<E> p,
    List<Position<E>> snapshot){
```

```
115         if(p.leftTree() != null)
116             inorderSubtree(p.leftTree(), snapshot);
117         snapshot.add(p.root());
118         if(p.rightTree() != null)
119             inorderSubtree(p.rightTree(), snapshot);
120 
121     }
122 
123     //traversée non ordonnée de la liste:
124     public Iterable<Position<E>> inorder(){
125         List<Position<E>> snapshot = new ArrayList<>(); // pk
    rien ds le <> ?
126         if(!isEmpty()){
127             inorderSubtree(this,snapshot);
128         }
129         return snapshot; // pourtant ce n'est pas un iterateur
    qu'on renvoie ???
130     }
131     /*
132      * @post : implémentation de RBinaryTree, permt de renvoyer
    un itérateur de positions
133      */
134     @Override
135     public Iterable<Position<E>> positions() {
136         return inorder();
137     }
138 
139     public String toString(){
140         Iterator<E> coucou = iterator();
141         StringBuffer buf = new StringBuffer();
142         while(coucou.hasNext()){
143             buf.append(coucou.next() + "\n");
144         }
145         return buf.toString();
146     }
147 
148 }
149 
```