**abstract** **class** FListInteger {

// ----- BASIC CREATORS -------------------------------

**public** **static** FListInteger emptyList() {

**return** **new** EmptyList();

}

**public** **static** FListInteger add(FListInteger l, Integer n) {

**return** **new** Add(l, n);

}

// ----- OTHER CREATORS -------------------------------

**public** **static** **boolean** isEmpty(FListInteger l) {

**return** l.isEmptyMethod();

}

**abstract** **boolean** isEmptyMethod();

All methods are inherited in the classes for the basic creators.

public class BTree implements Iterable<String> {

String data;

BTree left;

BTree right;

Comparator<String> comp;

boolean isEmpty;

int active;

BTree(String data, BTree left, BTree right,

Comparator<String> comp, boolean isEmpty) {

this.data = data;

this.left = left;

this.right = right;

this.comp = comp;

this.isEmpty = isEmpty;

this.active = 0;

}

private void insert(String s) {

if (this.isEmpty) {

this.data = s;

this.left = BTree.binTree(comp);

this.right = BTree.binTree(comp);

this.isEmpty = false;

}

else {

if ((comp.compare(this.data, s) == 0)) {

return;

}

else if ((comp.compare(this.data, s)) > 0) {

left.insert(s);

}

else {

right.insert(s);

}

}

}

private ArrayList<String> flatten() {

ArrayList<String> list = new ArrayList<String>();

while (!this.isEmpty) {

this.left.flatten();

for (int i = 0; i < this.left.flatten().size(); i++) {

list.add(this.left.flatten().get(i));

}

list.add(this.data);

this.right.flatten();

for (int i = 0; i < this.right.flatten().size(); i++) {

list.add(this.right.flatten().get(i));

}

{

return list;

}

}

return new ArrayList<String>();

}

public Iterator<String> iterator() {

return new BTreeIterator();

}

private class BTreeIterator implements Iterator<String> {

int current;

BTreeIterator() {

this.current = 0;

active = active + 1;

}

public boolean hasNext() {

return flatten() != null && this.current < flatten().size();

}

public String next() {

if (!this.hasNext()) {

throw new NoSuchElementException();

}

String result = flatten().get(current);

current = current + 1;

if (!this.hasNext()) {

active = active - 1;

}

return result;

}

public void remove() {

throw new UnsupportedOperationException();

}

}

}

class StringByLength implements Comparator<String> {

public int compare(String o1, String o2) {

return o1.length() - o2.length();

}

}

class StringByLex implements Comparator<String> {

public int compare(String o1, String o2) {

return o1.compareTo(o2);

}

}