```java
import java.util.Comparator;

/**
 * Implementation of Node class
 * @author ameliado
 */

public class Node<E>
{
    private LinkedList<E> data;
    private LinkedList<Node<E>> children;
    Node<E> parent;
    private Comparator<E> comp;
    private int order;

    public Node(int theOrder, Comparator<E> theComp) {
        this.order = theOrder;
        this.comp = theComp;
        this.data = new LinkedList<>();
        this.children = new LinkedList<>();
        this.parent = null;
    }

    public Node(int theOrder, Comparator<E> theComp, Node<E>
  left, E item, Node<E> right) {
        this(theOrder, theComp);
        this.children.add(left);
        this.children.add(right);
        this.data.add(item);
        if (left != null) left.parent = this;
        if (right != null) right.parent = this;
    }

    public Node(int theOrder, Comparator<E> theComp, Node<E>
  theParent, LinkedList<E> theData, LinkedList<Node<E>>
  theChildren) {
        this.order = theOrder;
        this.comp = theComp;
        this.parent = theParent;
```

```
38          this.data = new LinkedList<>(theData);
39          this.children = new LinkedList<>(theChildren);
40      }
41
42      public boolean hasOverflow() {
43          return data.size() > order;
44      }
45
46      public boolean isLeaf() {
47          return children.isEmpty();
48      }
49
50      public Node<E> childToFollow(E item)
51      {
52          if (isLeaf()) {
53              return null;
54          }
55          for (int i = 0; i < data.size(); i++) {
56              if (comp.compare(item, data.get(i)) < 0) {
57                  return children.get(i);
58              }
59          }
60          return children.getLast();
61      }
62
63      public void leafAdd(E item)
64      {
65          int index = 0;
66          while (index < data.size() &&
  comp.compare(data.get(index), item) < 0) {
67              index++;
68          }
69          data.add(index, item);
70      }
71
72      public void split()
73      {
74  //      int midIndex = order / 2;
75  //        E midValue = data.get(midIndex);
```

```
76 //
77 //        LinkedList<E> rightData = new
   LinkedList<>(data.subList(midIndex + 1, data.size()));
78 //        data.subList(midIndex, data.size()).clear();
79 //
80 //        LinkedList<Node<E>> rightChildren = new LinkedList<>();
81 //        if (!isLeaf()) {
82 //            rightChildren = new
   LinkedList<>(children.subList(midIndex + 1, children.size()));
83 //            children.subList(midIndex + 1,
   children.size()).clear();
84 //        }
85 //
86 //        Node<E> rightNode = new Node<>(order, comp, parent,
   rightData, rightChildren);
87 //        if (parent != null) {
88 //            int parentIndex = parent.children.indexOf(this);
89 //            parent.data.add(parentIndex, midValue);
90 //            parent.children.add(parentIndex + 1, rightNode);
91 //        }
92     }
93
94     public boolean contains(E item)
95     {
96         return data.contains(item);
97     }
98 }
99
```