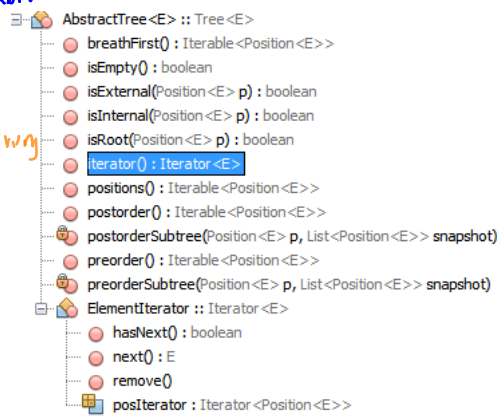
C:\Users\Rich\Documents\NetBeansProjects\Lab08\src\AbstractTree.java

```
1 /**
2  *
3  * @author Goodrich, Tamassia, Goldwasser
4  */
5
6 import java.util.ArrayList;
7 import java.util.Iterator;
8 import java.util.List;
9
10 public abstract class AbstractTree<E> implements Tree<E> {
11     @Override
12     public boolean isInternal(Position<E> p) {return numChildren(p) > 0 ; }
13     @Override
14     public boolean isExternal(Position<E> p) {return numChildren(p) == 0;}
15     @Override
16     public boolean isRoot(Position<E> p) {return p == root();}
17     @Override
18     public boolean isEmpty()    { return size() == 0;}
19
20     @Override
21     public Iterable<Position<E>> positions(){return preorder();}
22
23     /**adds positions of the subtree rooted at Position p to the given snapshot */
24     private void preorderSubtree(Position<E> p, List<Position<E>> snapshot)
25     {
26        snapshot.add(p);       // for preoder, we add positon p before exploring subtress
27        for(Position<E> c: children(p))
28           preorderSubtree(c,snapshot);
29     }
30     public Iterable<Position<E>> preorder()
31     {
32         List<Position<E>> snapshot = new ArrayList<>();
33         if(!isEmpty())
34            preorderSubtree(root(),snapshot);      // fill the snapshot recursively
35         return snapshot;
36     }
37
38     //--------------------Posorder traversal----------------
39
40     /**Adds positions of the subtree rooted at Position p to the given sanpshot*/
41     private void postorderSubtree(Position<E> p, List<Position<E>> snapshot)
42     {
43       for(Position<E> c:children(p))
44          postorderSubtree(c,snapshot);
45       snapshot.add(p);                //for postorder, we add position p after exploring subtrees
46     }
47
48     /**
49      * @return an iterable collection of positions of the tree, reported in postorder.
50      */
51     public Iterable<Position<E>> postorder()
52     {
53         List<Position<E>> snapshot = new ArrayList<>();
54         if(!isEmpty())
55            postorderSubtree(root(), snapshot); // fill the snapshot recursively
56         return snapshot;
```



```
57  }
58  //-------------------End of code for postOrder ------------------
59
60
61  //---------------  Nested iterarator class  -----------
62  /* 8.16 This class adapts the iteration produced by positions() tor return elements */
63  private class ElementIterator implements Iterator<E>
64  {
65     Iterator<Position<E>> posIterator = positions().iterator();
66     @Override
67     public boolean hasNext(){return posIterator.hasNext();}
68     @Override
69     public E next(){return posIterator.next().getElement();}
70     @Override
71     public void remove(){posIterator.remove();}
72
73  }
74  /******** End of Nested ElementIterator Class *****/
75
76  /**returns an iterator of the elements stored in the tree */
77  @Override
78  public Iterator<E> iterator(){ return new ElementIterator();}
79
80  /**
81   * Returns an iterable collection of positions of the tree in breadth-first order.
82   * code fragment 8.21
83   * @return
84   */
85  public Iterable<Position<E>> breathFirst()
86  {
87     List<Position<E>> snapshot = new ArrayList<>();
88     if(!isEmpty())
89     {
90        Queue<Position<E>> fringe = new LinkedQueue<>();
91        fringe.enqueue(root());                    // start with the root
92        while(!fringe.isEmpty())
93        {
94           Position<E> p = fringe.dequeue();       // remove from fornt of the queue
95           snapshot.add(p);                        // rerport this position
96           for(Position<E> c: children(p))
97           fringe.enqueue(c);                      // add children back to the queue.
98        }
99     }
100    return snapshot;
101 }
102
103 }
104
```

```
public int depth(Position<E> p)
{
    if(isRoot(p))
        return 0;
    else
        return 1+ depth(parent(p));
}
/**
 * Height of a tree is the maximum depth of any node.
 * @param p node position(root) of the subtree
 * @returns the height of the subtree rotted at Position p.
 */
public int height(Position<E> p)
{
    int h = 0;
    for(Position<E> c: children(p))
        h = Math.max(h, 1+ height(c));
    return h;
}
```