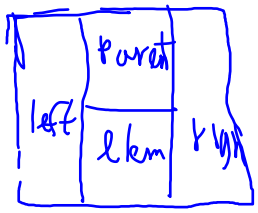
C:\Users\Rich\Documents\NetBeansProjects\Lab08\src\LinkedBinaryTree.java

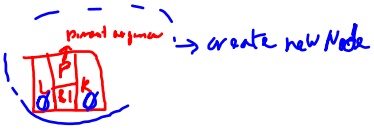
```
1 /**
2  * Data Structure and Algorithm 6th ed
3  * Code fragment 8.8-8.11
4  * @author Goldwasser, Goodrich, Tamassia
5  */
6 public class LinkedBinaryTree<E> extends AbstractBinaryTree<E> {
7
8   // nested Node class-
9   protected static class Node<E> implements Position<E>
10   {
11     private E element;          // an element stroed at this node
12     private Node<E> parent;     // a reference to the parent node(if any)
13     private Node<E> left;       // a reference to the left  child (if any)
14     private Node<E> right;      // a refence to the right child (if any)
15
16     /** Construct a node with the given element and neighbors */
17     public Node(E e, Node<E> above, Node<E> leftChild, Node<E>  rightChild)
18     {
19       element = e;
20       parent = above;
21       left = leftChild;
22       right = rightChild;
23     }
24
25     //accessor methods
26     @Override
27     public E getElement() { return element;}
28     public Node<E> getParent()  {return parent;}
29     public Node<E> getLeft()    {return left;}
30     public Node<E> getRight()   {return right;}
31
32    //update methods
33     public void setElement(E e) { element =e;}
34     public void setParent(Node<E> parentNode) { parent = parentNode;}
35     public void setLeft(Node<E> leftChild)  { left = leftChild;}
36     public void setRight(Node<E> rightChild){ right = rightChild;}
37   } // End of Nested Node class
38
39   /** Factory function to create a new node storing element e. */
40   protected Node<E> createNode(E e, Node<E> parent, Node<E> left, Node<E> right)
41   {
42     return new Node<E> (e, parent, left, right);
43   }
44
45   //LinkedBinaryTreee instance variables
46   protected Node<E> root = null;  // root of the tree
47   private int size = 0;          // nummber of nodes in the three
48
49   //constructor
50   public LinkedBinaryTree(){}        // construct an empty birnary tree
51
52   //nonpublic utility
53   /** Validates the position and returns it as a node */
54   protected Node<E> validate(Position<E> p) throws IllegalArgumentException
55   {
56     if(!(p instanceof Node))
```



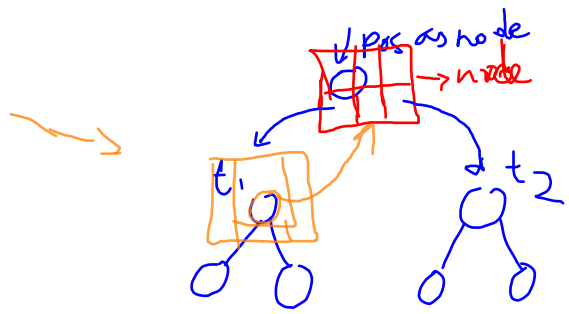
```
57          throw new IllegalArgumentException("Not valid position type");
58       Node<E> node = (Node<E>) p;           //safe cast
59       if(node.getParent()== node)           // our convention for defunct node
60          throw new IllegalArgumentException("p is no longer in the tree");
61       return node;
62    }
63
64    //accessor methods(not alread implemented in AbstractBinaryTree)
65    /**returns the number of nodes in the tree */
66    @Override
67    public int size()
68    {
69       return size;
70    }
71
72    /** returns the root position of the tree(or null if tree is empty)*/
73    public Position<E> root()
74    {
75       return root;
76    }
77
78    /**Returns the Position of p's parent(or null if p is root) */
79    @Override
80    public Position<E> parent(Position<E> p) throws IllegalArgumentException
81    {
82       Node<E> node = validate(p);
83       return node.getParent();
84    }
85
86    /**returns the position of p's left child(or null if no child exist) */
87    @Override
88    public Position<E> left(Position<E> p) throws IllegalArgumentException
89    {
90       Node<E> node = validate(p);
91       return node.getLeft();
92    }
93
94    /** Retruns the Positon of p's right child(or null if no child exists). */
95    @Override
96    public Position<E> right(Position<E> p) throws IllegalArgumentException
97    {
98       Node <E> node = validate(p);
99       return node.getRight();
100   }
101
102   //update methods supported by this class
103   /** Places element at the root of an empty tree and returns its new Position  */
104   public Position<E> addRoot(E e) throws IllegalStateException
105   {
106      if(!isEmpty()) throw new IllegalStateException("Tree is not empty");
107      root = createNode(e, null, null, null);
108      size = 1;
109      return root;
110   }
111
112   /**Creates a new left child of Position p storing element e; returns its Position. */
113   public Position<E>  addLeft(Position<E> p, E e) throws IllegalArgumentException
114   {
115      Node<E> parent = validate(p);
```

even though root was declared of type Node<E>

parent



```
116     if(parent.getLeft() != null)
117        throw new IllegalArgumentException("p alread has a left child");
118     Node<E> child = createNode(e, parent,null, null);
119     parent.setLeft(child);
120     size++;
121     return child;
122  }
123
124  /** Create a new right child of Position p storing element e; returns its Position. */
125  public Position<E> addRight(Position<E> p, E e) throws IllegalArgumentException
126  {
127     Node<E> parent = validate(p);
128     if(parent.getRight() != null)
129        throw new IllegalArgumentException("p already has a right child");
130     Node<E> child = createNode(e, parent, null, null);
131     parent.setRight(child);
132     size++;
133     return child;
134  }
135
136  /** Replace the element at Position p with e and returns the replaced element. */
137
138  public E set(Position<E> p, E e) throws IllegalArgumentException{
139     Node<E> node = validate(p);
140     E temp = node.getElement();
141     node.setElement(e);
142     return temp;
143  }
144  /** Attaches tress t1 and t2 as left and right subtrees of exteranal p. */
145  public void attach(Position<E> p, LinkedBinaryTree<E> t1, LinkedBinaryTree<E> t2)
146        throws IllegalArgumentException
147  {
148     Node <E> node = validate(p);
149     if(isInternal(p)) throw new IllegalArgumentException("p must be a leaf");
150     size+= t1.size() + t2.size();
151     if(!t1.isEmpty())
152     {
153        t1.root.setParent(node);
154        node.setLeft(t1.root);
155        t1.root= null;
156        t1.size = 0;
157     }
158     if(!t2.isEmpty())
159     {
160        t2.root.setParent(node);
161        node.setRight(t2.root);
162        t2.root = null;
163        t2.size = 0;
164     }
165  }
166  /**Removes the node at Position p and replaces it with its child, if any */
167  public E remove(Position<E> p) throws IllegalArgumentException
168  {
169     Node<E> node = validate(p);
170     if(numChildren(p)== 2)
171        throw new IllegalArgumentException("p has two children");
172     Node<E> child = (node.getLeft()!= null ? node.getLeft():node.getRight());
173     if(child != null)
174        child.setParent(node.getParent());        // child's granparent becomes its parent
```



```
175         if(node == root)
176            root = child;
177         else
178         {
179            Node<E> parent =  node.getParent();
180            if(node == parent.getLeft())
181               parent.setLeft(child);
182            else
183               parent.setRight(child);
184         }
185         size--;
186         E temp = node.getElement();
187         node.setElement(null);
188         node.setLeft(null);
189         node.setRight(null);
190         node.setParent(node);               // our convention for defunct node
191         return temp;
192      }
193   }
194
```