```java
import java.io.IOException;
import java.util.NoSuchElementException;

/*************************************************************
* Dalton Nofs                                                *
* Login ID: nofs5491                                         *
* CS-102, Summer 2017                                        *
* Programming Assignment 5                                   *
* Tree class:  Class for maintaining trees                     *
*************************************************************/
class Tree <T extends Comparable<T>>
{
    TreeNode<T> root; // root for the tree
    TreeNode<T> searchedNode; // node holder for searched node

    /*************************************************************
    * Method: Tree()                                              *
    * Purpose: default constructor for Tree class                *
    *                                                              *
    * Parameters:                N/A                            *
    * Returns: void:             N/A                            *
    *************************************************************/
    public Tree()
    {
        root=null;
    }

    /*************************************************************
    * Method: add()                                                 *
    * Purpose: adds a value to the tree                           *
    *                                                              *
    * Parameters: T:            type to add                      *
    * Returns: void:            N/A                               *
    *************************************************************/
    public void add(T target) throws IOException
    {
        root=add(target,root);
    }

    /*************************************************************
    * Method: add()    *private*                                    *
    * Purpose: recursive private func                            *
    *                                                              *
    * Parameters: T,TreeNode:    target, current node            *
    * Returns: TreeNode<T>:            TreeNode obj                *
    *************************************************************/
    private TreeNode<T> add(T target, TreeNode<T> current) throws IOException
    {
        // if null its last so just add to end
        if(current == null)
        {
            TreeNode<T> leaf = new TreeNode<T>();
            leaf.setDatum(target);
            leaf.setLeft(null);
            leaf.setRight(null);
            return leaf;
        }
        if(search(target))
            throw new IOException("Course already exists");
        if(current.getDatum().compareTo(target)<0) // cont add to right branch
           current.setRight( add(target, current.getRight()) );
        else                                       // cont add to left branch
            current.setLeft( add(target, current.getLeft()) );
        return current;
    }

    /*************************************************************
```

```
68      * Method: search()                                                *
69      * Purpose: searches tree for target                              *
70      *                                                                 *
71      * Parameters: T:         target                                  *
72      * Returns: boolean:      if found or not                        *
73      *****************************************************************/
74      public boolean search(T target)
75      {
76          return search(target,root);
77      }
78
79      /*****************************************************************
80      * Method: search()    *private*                                   *
81      * Purpose: recursive search fucn                                 *
82      *                                                                 *
83      * Parameters: T,TreeNode:    target, current node                *
84      * Returns: boolean:         if found true                        *
85      *****************************************************************/
86      private boolean search(T target, TreeNode<T> current)
87      {
88          if(current == null) {return false;} // if fallen off list
89          if(current.getDatum().compareTo(target) == 0) // base case
90          {
91              searchedNode = current;
92              return true;
93          }
94          if(current.getDatum().compareTo(target)<0) // continue search
95              return search(target,  current.getRight());
96          else
97              return search(target, current.getLeft()); // continue search
98      }
99
100     /*****************************************************************
101     * Method: remove()                                                *
102     * Purpose: searches and removes target from tree                 *
103     *                                                                 *
104     * Parameters: T:         target                                  *
105     * Returns: void:           N/A                                   *
106     *****************************************************************/
107     public void remove(T target)
108     {
109         root = remove(target,root);
110     }
111
112     /*****************************************************************
113     * Method: search()    *private*                                    *
114     * Purpose: recursive remove fucn                                 *
115     *                                                                 *
116     * Parameters: T,TreeNode:    target, current node                *
117     * Returns: TreeNode<T>:     fixed treeNode                       *
118     *****************************************************************/
119     private TreeNode<T> remove(T target, TreeNode<T> current) throws NoSuchElementException
120     {
121         if(current==null) // check to see if fell off list
122             throw new NoSuchElementException();
123         if(current.getDatum().compareTo(target)<0) // search right branch
124         {
125             current.setRight(remove(target,current.getRight()));
126             return current;
127         }
128         else if(current.getDatum().compareTo(target)>0) // search left branch
129         {
130             current.setLeft(remove(target,current.getLeft()));
131             return current;
132         }
133         if(current.getLeft()==null) return current.getRight(); //eob found
134         if(current.getRight()==null) return current.getLeft(); //eob found
135         TreeNode<T> heir=current.getLeft(); // start heir search on left
```

```
136         while(heir.getRight()!=null) // search for heir
137             heir=heir.getRight();
138         current.setDatum(heir.getDatum()); // reconfigure tree
139         current.setLeft(remove(heir.getDatum(),current.getLeft()));
140         return current;
141     }
142 
143     /*************************************************************
144     * Method: isEmpty()                                          *
145     * Purpose: is tree empty?                                    *
146     *                                                            *
147     * Parameters: void            N/A                            *
148     * Returns: boolean:         if empty 1, else 0               *
149     *************************************************************/
150     public boolean isEmpty()
151     {
152         if(root == null)
153             return true;
154         else
155             return false;
156     }
157 
158     /*************************************************************
159     * Method: removeAll()                                        *
160     * Purpose: remove all nodes                                  *
161     *                                                            *
162     * Parameters: void:           N/A                            *
163     * Returns: void:            N/A                              *
164     *************************************************************/
165     public void removeAll()
166     {
167         root = null;
168     }
169 
170     /*************************************************************
171        * Method: getSearched()                                    *
172        * Purpose: get node for last searched                      *
173        *                                                          *
174        * Parameters: void:           N/A                          *
175        * Returns: T:               data from last node searched   *
176        *************************************************************/
177     public TreeNode<T> getSearched()
178     {
179         return  searchedNode;
180     }
181 
182     /*************************************************************
183        * Method: getRoot()                                        *
184        * Purpose: get node for last searched                      *
185        *                                                          *
186        * Parameters: void:           N/A                          *
187        * Returns: TreeNode<T>:       the head node of the tree    *
188        *************************************************************/
189     public TreeNode<T> getRoot()
190     {
191         return  root;
192     }
193 }
```