```java
// Problem 8.2  Taylor Somma
// CS501

import java.io.*;
import java.util.*;

class Node
{
    public int iData;
    public double dData;
    public Node leftChild;
    public Node rightChild;

    public void displayNode()
    {
        System.out.print("{");
        System.out.print(iData);
        System.out.print(", ");
        System.out.print("dData");
        System.out.print("} ");

    }
}

class Tree
{
    private Node root;

    public Tree()
    {
        root=null;
    }

    public Node find(int key)   // from listing 8.1
    {
        Node current=root;
        while(current.iData != key)
        {
            if (key < current.iData)
                current= current.leftChild;
            else
                current = current.rightChild;
            if (current == null)
                return null;
        }
        return current;
    }

    public void insert (int id, double dd)   // from listing 8.1
    {
        Node newNode = new Node();
        newNode.iData = id;
        newNode.dData = dd;
        if (root==null)
        {
            root = newNode;
            System.out.println("Adding " + id + ", " + dd +" As Root");
            return;
        }
        else
```

```
61          {
62              Node current = root;
63              Node parent;
64              while(true)
65              {
66                  parent = current;
67                  if (id < current.iData)
68                  {
69                      current = current.leftChild;
70                      if (current == null)
71                      {
72                          parent.leftChild = newNode;
73                          return;
74                      }
75                  }
76                  else
77                  {
78                      current = current.rightChild;
79                      if (current == null)
80                      {
81                          parent.rightChild = newNode;
82                          return;
83                      }
84              }
85          }
86          }
87      }
88
89      public boolean delete(int key)   // from listing 8.1
90      {
91          Node current = root;
92          Node parent = root;
93          boolean isLeftChild = true;
94
95          while(current.iData != key)
96          {
97              parent = current;
98              if(key < current.iData)
99              {
100                 isLeftChild = true;
101                 current = current.leftChild;
102             }
103             else
104             {
105                 isLeftChild = false;
106                 current = current.rightChild;
107             }
108             if (current == null)
109             {
110                 return false;
111             }
112         }
113
114         if(current.leftChild==null && current.rightChild==null) //No Children
115         {
116             if (current==root)
117                 root = null;
118             else if(isLeftChild)
119                 parent.leftChild = null;
120             else
121                 parent.rightChild = null;
```

```
122         }
123
124         else if(current.leftChild==null)  //No left child, only right
125         {
126             if (current==root)
127                 root=current.rightChild;
128             else if(isLeftChild)
129                 parent.leftChild = current.rightChild;
130             else
131                 parent.rightChild = current.rightChild;
132         }
133
134         else if(current.rightChild == null)  // No right child, only left
135         {
136             if (current==root)
137             {
138                 root=current.leftChild;
139             }
140
141             if (isLeftChild)
142             {
143                 parent.leftChild = current.leftChild;
144             }
145             else
146             {
147                 parent.rightChild = current.leftChild;
148             }
149         }
150
151         else      //Two Children
152         {
153             Node successor = getSuccessor(current);
154
155             if (current==root)
156                 root = successor;
157             else if (isLeftChild)
158                 parent.leftChild = successor;
159             else
160                 parent.rightChild = successor;
161         successor.leftChild = current.leftChild;    //connects successor to current left
    child
162         }
163     return true;
164     }
165
166     private Node getSuccessor(Node delNode)   // from listing 8.1
167     {
168         Node successorParent = delNode;
169         Node successor = delNode;
170         Node current = delNode.rightChild;
171         while(current != null)
172         {
173             successorParent = successor;
174             successor = current;
175             current = current.leftChild;
176         }
177
178         if(successor != delNode.rightChild)
179         {
180             successorParent.leftChild = successor.rightChild;
181             successor.rightChild = delNode.rightChild;
```

```
182         }
183 
184         return successor;
185     }
186 
187     public void traverse(int traverseType)   // from listing 8.1
188     {
189         switch(traverseType)
190         {
191             case 1: System.out.print("\nPreorder traversal: ");
192             preOrder(root);
193             break;
194             case 2: System.out.print("\nInorder traversal: ");
195             inOrder(root);
196             break;
197             case 3: System.out.print("\nPostorder traversal: ");
198             postOrder(root);
199             break;
200         }
201         System.out.println();
202     }
203 
204     public void balanceTree()
205     {
206         LinkedList treeList = new LinkedList();
207         treeList = balanceHelper_1(treeList, root);
208         System.out.println("In Balance Helper");
209         printTreeList(treeList);
210         Object[] treeArray = treeList.toArray(new Integer[treeList.size()]);
211         int[] treeArrayInt = myListToArray(treeArray);
212         Tree balancedTree = new Tree();
213         int[] emptyArray = new int[0];
214         balancedTree = arrayToTree(treeArrayInt,emptyArray,balancedTree);
215         balancedTree.displayTree();
216     }
217 
218     public int[] myListToArray(Object[] oldList)
219     {
220         int[] returnArray = new int[oldList.length];
221         for (int x=0;x<oldList.length;x++)
222         {
223             //System.out.print("Converting obj: " + oldList[x]);
224             returnArray[x] = (int) oldList[x];
225         }
226         return returnArray;
227     }
228 
229     public void printTreeList(LinkedList list1)
230     {
231         Object[] treeArray = list1.toArray(new Integer[list1.size()]);
232         for (int x=0; x<list1.size();x++)
233         {
234             System.out.print(treeArray[x] + ", ");
235         }
236     }
237 
238     // To initialize must make an empty array for arrayToConvert_2
239     public Tree arrayToTree(int[] arrayToConvert_1, int[] arrayToConvert_2, Tree inTree)
240     {
241         System.out.println("in arrayToTree");
242         if (arrayToConvert_1.length > 2 && arrayToConvert_2.length==0)
```

```
243         {
244             int divSpot = arrayToConvert_1.length/2;
245             int[] arrayLeft = Arrays.copyOfRange(arrayToConvert_1,0,divSpot);
246             int[] arrayRight =
    Arrays.copyOfRange(arrayToConvert_1,divSpot+1,arrayToConvert_1.length);
247             inTree.insert(arrayToConvert_1[divSpot],0);
248             System.out.print("Inserting : " + arrayToConvert_1[divSpot]);
249             arrayToTree(arrayLeft,arrayRight, inTree);
250         }
251         else if (arrayToConvert_1.length>arrayToConvert_2.length)
252         {
253             int divSpot = arrayToConvert_1.length/2;
254             System.out.print("Inserting : " + arrayToConvert_1[divSpot]);
255             inTree.insert(arrayToConvert_1[divSpot],0);
256             int[] arrayLeft_1 = Arrays.copyOfRange(arrayToConvert_1,0,divSpot);
257             int[] arrayLeft_2 =
    Arrays.copyOfRange(arrayToConvert_1,divSpot+1,arrayToConvert_1.length);
258             int[] arrayLeft = joinArrays(arrayLeft_1,arrayLeft_2);
259             inTree.displayTree();
260             arrayToTree(arrayLeft,arrayToConvert_2,inTree);
261         }
262         // if right has more than left, take the middle element of right and add it to tree
263         else if (arrayToConvert_1.length<arrayToConvert_2.length)
264         {
265             int divSpot = arrayToConvert_2.length/2;
266             System.out.print("Inserting : " + arrayToConvert_2[divSpot]);
267             inTree.insert(arrayToConvert_2[divSpot],0);
268             int[] arrayRight_1 = Arrays.copyOfRange(arrayToConvert_2,0,divSpot);
269             int[] arrayRight_2 =
    Arrays.copyOfRange(arrayToConvert_2,divSpot+1,arrayToConvert_2.length);
270             int[] arrayRight = joinArrays(arrayRight_1,arrayRight_2);
271             inTree.displayTree();
272             arrayToTree(arrayToConvert_1,arrayRight,inTree);
273         }
274         else if (arrayToConvert_1.length==1 && arrayToConvert_2.length==1)
275         {
276             inTree.insert(arrayToConvert_1[0],0);
277             inTree.insert(arrayToConvert_2[0],0);
278             inTree.displayTree();
279             return inTree;
280         }
281         else if (arrayToConvert_1.length==arrayToConvert_2.length)
282         {
283             int divSpot = arrayToConvert_2.length/2;
284             System.out.print("Inserting : " + arrayToConvert_2[divSpot]);
285             inTree.insert(arrayToConvert_2[divSpot],0);
286             int[] arrayRight_1 = Arrays.copyOfRange(arrayToConvert_2,0,divSpot);
287             int[] arrayRight_2 =
    Arrays.copyOfRange(arrayToConvert_2,divSpot+1,arrayToConvert_2.length);
288             int[] arrayRight = joinArrays(arrayRight_1,arrayRight_2);
289             inTree.displayTree();
290             arrayToTree(arrayToConvert_1,arrayRight,inTree);
291         }
292 
293         else
294             return inTree;
295         return inTree;
296     }
297 
298     public int[] joinArrays(int[] array1,int[] array2)
299     {
```

```
300         int[] returnArray = new int[array1.length+array2.length];
301         for (int x=0;x<array1.length;x++)
302             returnArray[x]=array1[x];
303         for (int y=0;y<array2.length;y++)
304             returnArray[array1.length+y]=array2[y];
305         return returnArray;
306     }
307     public LinkedList balanceHelper_1(LinkedList returnList, Node localRoot)
308     {
309         if(localRoot!=null)
310         {
311             balanceHelper_1(returnList, localRoot.leftChild);
312             returnList.add(localRoot.iData);
313             System.out.println("Adding " + localRoot.iData + " To List");
314             balanceHelper_1(returnList, localRoot.rightChild);
315         }
316         return returnList;
317     }
318 
319     private void preOrder(Node localRoot)   // from listing 8.1
320     {
321         if (localRoot != null)
322         {
323             System.out.print("" + localRoot.iData + ' ');
324             preOrder(localRoot.leftChild);
325             preOrder(localRoot.rightChild);
326         }
327     }
328     private void postOrder(Node localRoot)   // from listing 8.1
329     {
330         if (localRoot != null)
331         {
332             postOrder(localRoot.leftChild);
333             postOrder(localRoot.rightChild);
334             System.out.print("" + localRoot.iData + ' ');
335         }
336     }
337 
338     private void inOrder(Node localRoot)   // from listing 8.1
339     {
340         if(localRoot != null)
341         {
342             inOrder(localRoot.leftChild);
343             System.out.print("" + localRoot.iData + ' ');
344             inOrder(localRoot.rightChild);
345         }
346     }
347 
348     public void displayTree()   // from listing 8.1
349     {
350         Stack globalStack = new Stack();
351         globalStack.push(root);
352         int nBlanks = 32;
353         boolean isRowEmpty = false;
354         System.out.println(
355             "------------------------------------------------------------");
356         while (isRowEmpty==false)
357         {
358             Stack localStack = new Stack();
359             isRowEmpty = true;
360 
```

```
361             for (int j=0; j<nBlanks; j++)
362             {
363                 System.out.print(' ');
364                 //System.out.println("in for loop # 1 printing blank J: " + j);
365             }
366 
367             while (globalStack.isEmpty()==false)
368             {
369                 Node temp = (Node)globalStack.pop();  //changed
370                 if(temp != null)
371                 {
372                     //System.out.println("temp!=null temp: " + temp.iData);
373                     System.out.print(temp.iData);
374                     localStack.push(temp.leftChild);
375                     localStack.push(temp.rightChild);
376 
377                     if (temp.leftChild != null || temp.rightChild != null)
378                     {
379                         //System.out.println("no left or right child");
380                         isRowEmpty = false;
381                     }
382                 }
383                 else
384                 {
385                     //System.out.println("printing blank space.. and pushing null");
386                     System.out.print("..");
387                     localStack.push(null);
388                     localStack.push(null);
389                 }
390                 for (int j=0; j<nBlanks*2-2; j++)
391                 {
392                     //System.out.println("Printing in second j for loop");
393                     System.out.print(" ");
394                 }
395             }
396                 System.out.println();
397                 nBlanks /= 2;
398                 while(localStack.isEmpty() == false)
399                 {
400                     globalStack.push(localStack.pop());
401                 }
402                 System.out.println("-----------------------------------------------------------
    ---");
403 
404         }
405     }
406 }
407 
408 public class balancedTreeTest_1
409 {
410     public static void main(String[] args)
411     {
412         System.out.println("Welcome to the jungle");
413 
414         Tree tree1 = new Tree();
415         tree1.insert(7,0);
416         tree1.insert(9,0);
417         tree1.insert(5,0);
418         tree1.displayTree();
419         tree1.traverse(1);
420         tree1.traverse(2);
```

```
421         tree1.traverse(3);
422         tree1.insert(1,0);
423         tree1.insert(2,0);
424         tree1.insert(3,0);
425         tree1.insert(4,0);
426         tree1.insert(6,0);
427         tree1.displayTree();
428         tree1.traverse(1);
429         tree1.traverse(2);
430         tree1.traverse(3);
431         tree1.balanceTree();
432     }
433 }
```