Homework Turnin

Account:	6G_06 (rgalanos@fcps.edu)
Section:	6G
Course:	TJHSST APCS 2016-17
Assignment:	07-05
Receipt ID:	7dc3a1c210a3fee87d3d69f782fc205c

Execution failed with return code 1 (general error). (Expected for JUnit when any test fails.)

Warning: Your program failed to compile :

```
BSTobject_Driver_shell.java:5: error: class BSTobject_Driver is public, should be declared in a file named BSTob
public class BSTobject_Driver
       ^
BSTobject_Driver_shell.java:63: error: duplicate class: BSTinterface
interface BSTinterface<E extends Comparable<E>>
^
BSTobject_Driver_shell.java:74: error: duplicate class: BSTobject
class BSTobject <E extends Comparable<E>> implements BSTinterface<E>
^
Note: BSTobject_Driver.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
3 errors
```

Please correct your file(s), go back, and try to submit again. If you do not correct this problem, you are likely to lose a large number of points on the assignment. Please contact your TA if you are not sure why your code is not compiling successfully.

Turnin Failed! (See above)

There were some problems with your turnin. Please look at the messages above, fix the problems, then Go Back and try your turnin again.

GradeIt has a copy of your submission, but we believe that you will want to fix the problems with your submission by resubmitting a fixed version of your code by the due date.

We have received the following file(s):

BSTobject_Driver.java (10940 bytes)

```
1.
2.  //name:    date:
3.  import java.util.*;
4.  import java.io.*;
5.
6.  public class BSTobject_Driver
7.  {
8.     public static BSTobject<String> tree = null;
9.     public static BSTobject<Widget> treeOfWidgets = null;
10.    public static int numberWidgets = 10;
11.    public static void main( String[] args )
12.    {
13.    //day one
14.      //  tree = new BSTobject<String>();
15.    //       tree = build(tree, "T");
16.    //       System.out.print(tree);
17.    //       System.out.println("Size: " + tree.size());
18.    //       System.out.println("Is empty: "+ tree.isEmpty());
19.
20.       //day two
21.       // Your assignment the second day is to finish the BSTobject class.
22.       // Specifically, prompt the user for a string, put the characters into a BST,
23.       // call toString on this tree, and print the size of the tree.
24.
25.    //       Scanner keyboard = new Scanner(System.in);
26.    //       System.out.print("Enter a string: ");
27.    //       String str = keyboard.next();
28.    //       tree = new BSTobject<String>();
29.    //       tree = build(tree, str);
30.    //       System.out.print(tree);
31.    //       System.out.println(tree.size());
32.
33.       //day two, Widgets
34.       // Next, fill your BST with 57 Widget objects from widgets.txt.  Display the tree.
35.       // Then prompt the user to enter pounds and ounces.  If the tree contains that
36.       // Widget, delete it, of course restoring the BST order. Display the new tree
37.       // and its size. If the tree does not contain that Widget, print "Not found".
38.
39.       treeOfWidgets = new BSTobject<Widget>();
40.       treeOfWidgets = build(treeOfWidgets, new File("widget.txt"));
```

```
41.       System.out.print(treeOfWidgets);
42.       System.out.println(treeOfWidgets.size());
43.
44.       Scanner keyboard = new Scanner(System.in);
45.       System.out.print("Enter pounds ");
46.       int pounds = keyboard.nextInt();
47.       System.out.print("Enter ounces ");
48.       int ounces = keyboard.nextInt();
49.       Widget w = new Widget(pounds, ounces);
50.
51.       if(treeOfWidgets.contains(w))
52.       {
53.          treeOfWidgets.delete(w);
54.          System.out.print(treeOfWidgets);
55.          System.out.println(treeOfWidgets.size());
56.       }
57.       else
58.          System.out.println("Not found");
59.
60.
61.       //day three -- AVL tree  ----------------------
62.
63.    }
64.    // build the tree for Strings, Day 1
65.    public static BSTobject<String> build(BSTobject<String> tree, String str)
66.    {
67.       for(Character c: str.toCharArray())
68.       {
69.          tree.add(c+"");
70.       }
71.       return tree;
72.    }
73.      //build the tree for Widgets, Day 2
74.    public static BSTobject<Widget> build(BSTobject<Widget> tree, File file)
75.    {
76.       Scanner infile = null;
77.       try{
78.          infile = new Scanner( file  );
79.       }
80.       catch (Exception e)
81.       {
82.          System.exit(0);
83.       }
84.       for(int i = 0; i < 10; i++)
85.       {
86.          tree.add(new Widget(infile.nextInt(), infile.nextInt()));
87.       }
88.       return tree;
89.    }
90. }
91.
92. ////////////////////////////////
93. interface BSTinterface<E extends Comparable<E>>
94. {
95.    public E add( E element );     //returns the object
96.    public boolean contains( E element );
97.    public boolean isEmpty();
98.    public E delete( E element );  //returns the object, not a Node<E>
99.    public int size();
100.   public String toString();
101. }
102. ////////////////////////////////
103.
104. class BSTobject <E extends Comparable<E>> implements BSTinterface<E>
105. {
106.    // 2 fields
107.    Node<E> root;
108.    int size;
109.    // 1 default constructor
110.    public BSTobject()
111.    {
112.       root = null;
113.       size = 0;
114.    }
115.
116.    //instance methods
117.    public E add( E obj )
118.    {
119.       root = add( root, obj );
120.       size++;
121.       return obj;
122.    }
123.    //recursive helper method
124.    private Node<E> add( Node<E> t, E obj )
125.    {
126.       if(t == null)
127.          t = new Node<E>(obj);
128.       else if(((Comparable)t.getValue()).compareTo(obj)>=0)
129.       {
130.          if(t.getLeft()==null)
131.             t.setLeft(new Node(obj));
132.          else
133.             add(t.getLeft(),obj);
134.       }
135.       else if(((Comparable)t.getValue()).compareTo(obj)<0)
136.       {
137.          if(t.getRight()==null)
138.             t.setRight(new Node(obj));
139.          else
140.             add(t.getRight(),obj);
141.       }
142.       return t;
143.    }
144.    /* implement the interface here.  Use TreeNode as an example,
145.     but root is a field. You need add, contains, isEmpty,
146.     delete, size, and toString.  */
147.    public boolean contains( E element )
148.    {
149.       Node<E> temp = root;
150.       while(temp!=null)
151.       {
152.          if(((Comparable)temp.getValue()).compareTo(element)>0)
153.             temp = temp.getLeft();
154.          else if(((Comparable)temp.getValue()).compareTo(element)<0)
155.             temp = temp.getRight();
156.          else
157.             return true;
158.       }
159.       return false;
160.    }
161.    public boolean isEmpty()
162.    {
```

```
163.       if(root == null)
164.          return true;
165.       else
166.          return false;
167.    }
168.    public E delete( E element ) //returns the object, not a Node<E>
169.    {
170.       size--;
171.       Node<E> current = root;
172.       Node<E> parent = null;
173.       while(current !=null)
174.       {
175.          int compare = ((Comparable)current.getValue()).compareTo((element));
176.          // ------->  your code goes here
177.          if(compare>0)
178.          {
179.             parent = current;
180.             current = current.getLeft();
181.          }
182.          else if(compare<0)
183.          {
184.             parent = current;
185.             current = current.getRight();
186.          }
187.          else
188.          {
189.             if(parent == null)
190.             {
191.                if(current.getLeft()==null&&current.getRight()==null);
192.                else if(current.getLeft()==null)
193.                   root = root.getRight();
194.                else if(current.getRight()==null)
195.                   root = root.getLeft();
196.                else if(current.getLeft().getRight()==null)
197.                {
198.                   current.setValue(current.getLeft().getValue());
199.                   current.setLeft(current.getLeft().getLeft());
200.                }
201.                else
202.                {
203.                   current = current.getLeft();
204.                   while(current.getRight().getRight()!=null)
205.                   {
206.                      current = current.getRight();
207.                   }
208.                   root.setValue(current.getRight().getValue());
209.                   if(current.getRight().getLeft()!=null)
210.                      current.setRight(current.getRight().getLeft());
211.                   else
212.                      current.setRight(null);
213.                }
214.             }
215.             else if(current.getLeft()==null&&current.getRight()==null)
216.             {
217.                if(parent.getLeft()==null)
218.                   parent.setRight(null);
219.                else if(((Comparable)parent.getLeft().getValue()).compareTo((Comparable)current.getValue())==0)
220.                   parent.setLeft(null);
221.                else
222.                   parent.setRight(null);
223.             }
224.             else if(current.getLeft()==null)
225.             {
226.                if(((Comparable)parent.getLeft().getValue()).compareTo((Comparable)current.getValue())==0)
227.                   parent.setLeft(current.getRight());
228.                else
229.                   parent.setRight(current.getRight());
230.             }
231.             else if(current.getRight()==null)
232.             {
233.                if(((Comparable)parent.getLeft().getValue()).compareTo((Comparable)current.getValue())==0)
234.                   parent.setLeft(current.getLeft());
235.                else
236.                   parent.setRight(current.getLeft());
237.             }
238.             else
239.             {
240.                if(current.getLeft().getRight()!=null)
241.                {
242.                   boolean bool = (((Comparable)parent.getRight().getValue()).compareTo((Comparable)current.getValue())==0);
243.                   current = current.getLeft();
244.                   while(current.getRight().getRight()!=null)
245.                   {
246.                      current = current.getRight();
247.                   }
248.                   if(current.getRight().getLeft()==null)
249.                   {
250.                      if(bool)
251.                      {
252.                         parent.getRight().setValue(current.getRight().getValue());
253.                         current.setRight(null);
254.                      }
255.                      else
256.                      {
257.                         parent.getLeft().setValue(current.getRight().getValue());
258.                         current.setRight(null);
259.                      }
260.                   }
261.                   else
262.                   {
263.                      if(bool)
264.                      {
265.                         parent.getRight().setValue(current.getRight().getValue());
266.                         current.setRight(current.getRight().getLeft());
267.                      }
268.                      else
269.                      {
270.                         parent.getLeft().setValue(current.getRight().getValue());
271.                         current.setRight(current.getRight().getLeft());
272.                      }
273.
274.                   }
275.                }
276.                else
277.                {
278.                   current.setValue(current.getLeft().getValue());
279.                   current.setLeft(current.getLeft().getLeft());
280.                }
281.             }
282.             current = null;
283.          }
284.       }
```

```
285.            //never reached
286.        return element;
287.    }
288.    public int size()
289.    {
290.        return size;
291.    }
292.    public String toString()
293.    {
294.        return toString(root, 0);
295.    }
296.    private String toString(Node<E> t, int level)
297.    {
298.        String toRet = "";
299.        if(t == null)
300.            return "";
301.        toRet += toString(t.getRight(), level + 1);
302.        for(int k = 0; k < level; k++)
303.            toRet += "\t";
304.        toRet += t.getValue() + "\n";
305.        toRet += toString(t.getLeft(), level + 1);
306.        return toRet;
307.    }
308.
309.     /*************************private inner class *************/
310.    private class Node<E>
311.    {
312.        private Object value;
313.        private Node<E> left, right;
314.
315.        public Node(Object initValue)
316.        {
317.            value = initValue;
318.            left = null;
319.            right = null;
320.        }
321.
322.        public Node(Object initValue, Node<E> initLeft, Node<E> initRight)
323.        {
324.            value = initValue;
325.            left = initLeft;
326.            right = initRight;
327.        }
328.
329.        public Object getValue()
330.        {
331.            return value;
332.        }
333.
334.        public Node<E> getLeft()
335.        {
336.            return left;
337.        }
338.
339.        public Node<E> getRight()
340.        {
341.            return right;
342.        }
343.
344.        public void setValue(Object theNewValue)
345.        {
346.            value = theNewValue;
347.        }
348.
349.        public void setLeft(Node<E> theNewLeft)
350.        {
351.            left = theNewLeft;
352.        }
353.
354.        public void setRight(Node<E> theNewRight)
355.        {
356.            right = theNewRight;
357.        }
358.
359.        // 3 fields
360.
361.        // 2 constructors, one-arg and three-arg
362.
363.
364.
365.
366.        //methods--Use TreeNode as an example. See the cheat sheet.
367.
368.    }
369. }
370.
```