Homework Turnin

Account:	6G_06 (rgalanos@fcps.edu)
Section:	6G
Course:	TJHSST APCS 2016-17
Assignment:	11-02
Receipt ID:	d4ea1c5c4b3aa660ec8caaf09bad88a2

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

Warning: Your program failed to compile :

```
Huffman_shell.java:4: error: class Huffman is public, should be declared in a file named Huffman.java
public class Huffman
       ^
Huffman_shell.java:40: error: duplicate class: HuffmanTreeNode
class HuffmanTreeNode implements Comparable<HuffmanTreeNode>
^
2 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):

Huffman.java (5401 bytes)

```java
// name:        date:
import java.util.*;
import java.io.*;
public class Huffman
{
   public static void main(String[] args) throws IOException
   {
         //Read the string
         //Make a frequency table of the letters
         //Put each letter-frequency pair into a HuffmanTreeNode. Put each
         //        node into a priority queue (or a min-heap).
         //Use the priority queue of nodes to build the Huffman tree
         //Process the string letter-by-letter and search the tree for the
         //       letter.  As you go, build the binary path, where going
         //       left is 0 and going right is 1.
         //Write the binary path to the hard drive as message.xxx.txt
         //Write the scheme to the hard drive as scheme.xxx.txt

      Scanner keyboard = new Scanner(System.in);
      System.out.println("What is the message?");
      String message = keyboard.next();

      Map<String, Integer> table = makeFrequencyTable(message);

      PriorityQueue<HuffmanTreeNode> pq = makePriorityQueue(table);

      PriorityQueue<HuffmanTreeNode> pq2 = new PriorityQueue<HuffmanTreeNode>(pq);

      HuffmanTreeNode tree = makeHuffmanTreeNode(pq);

      HashMap<String, String> scheme = findScheme(pq2, tree);

      String code = findHuffmanCode(message, scheme);

      PrintWriter messageScanner = new PrintWriter(new FileWriter(new File("message."+message+".txt")));
      messageScanner.println(code);
      messageScanner.close();

      System.setOut(new PrintStream(new FileOutputStream(new File("scheme." + message + ".txt"))));
      printScheme(scheme, tree);

   }
   public static HashMap<String, Integer> makeFrequencyTable(String s)
   {
      HashMap<String, Integer> map = new HashMap<String, Integer>();
      for(char c:s.toCharArray())
      {
         if(!map.containsKey(c+""))
            map.put(c+"", 1);
         else
            map.put(c+"", map.get(c+"")+1);
      }
      return map;
   }
   public static PriorityQueue<HuffmanTreeNode> makePriorityQueue(Map<String, Integer> m)
   {
      PriorityQueue<HuffmanTreeNode> pq = new PriorityQueue<HuffmanTreeNode>();
      Iterator it = m.keySet().iterator();
      while(it.hasNext())
      {
         String str = it.next().toString();
         pq.add(new HuffmanTreeNode(str, m.get(str)));
      }
      return pq;
   }
   public static HuffmanTreeNode makeHuffmanTreeNode(PriorityQueue<HuffmanTreeNode> p)
   {
      while(p.size()>1)
      {
         HuffmanTreeNode temp = new HuffmanTreeNode(p.remove(), p.remove());
         p.add(temp);
      }
      return p.remove();
   }
   public static HashMap<String, String> findScheme(PriorityQueue p, HuffmanTreeNode t)
   {
      HashMap<String, String> map = new HashMap<String, String>();
      while(p.size()>0)
      {
         String str = ((HuffmanTreeNode)p.remove()).getValue();
         map.put(str, find(str, t));
      }
      return map;
   }
   public static void printScheme(Map<String, String> m, HuffmanTreeNode t)
   {
      if(t==null)
         return;
      else
      {
         if(!t.getValue().equals("*"))
            System.out.println(t.getValue() + m.get(t.getValue()));
         printScheme(m, t.getLeft());
         printScheme(m, t.getRight());
      }
   }
   public static String find(String s, HuffmanTreeNode t)
   {
      if(t==null)
         return null;
      else if(t.getValue().equals(s))
         return "";
      else
      {
         String str1 = find(s, t.getLeft());
         String str2 = find(s, t.getRight());
         if(str1!=null)
            return "0" + str1;
         else if(str2!=null)
            return "1" + str2;
         else
            return null;
      }
   }
   public static String findHuffmanCode(String str, Map<String, String> m)
   {
      String s = "";
      for(int i=0;i<str.length();i++)
      {
         s+=m.get(str.charAt(i)+"");
      }
      return s;
   }
}
   /*
    * This node stores two values.
    * The compareTo method must ensure that the lowest frequency has the highest priority.
    */
class HuffmanTreeNode implements Comparable<HuffmanTreeNode>
{
   private String str;
   private int freq;
   private HuffmanTreeNode left, right;

   public HuffmanTreeNode(String s, int f)
   {
      str = s;
      freq = f;
      left = null;
      right = null;
   }

   public HuffmanTreeNode(HuffmanTreeNode initLeft, HuffmanTreeNode initRight)
   {
      str = "*";
      freq = initLeft.getFrequency() + initRight.getFrequency();
      left = initLeft;
      right = initRight;
   }

   public String getValue()
   {
      return str;
   }

   public HuffmanTreeNode getLeft()
   {
      return left;
   }

   public HuffmanTreeNode getRight()
   {
      return right;
   }

   public int getFrequency()
   {
      return freq;
   }

   public void setValue(String theNewValue)
   {
      str = theNewValue;
   }

   public void setLeft(HuffmanTreeNode theNewLeft)
   {
      left = theNewLeft;
   }

   public void setRight(HuffmanTreeNode theNewRight)
   {
      right = theNewRight;
   }

   public void setFrequency(int f)
   {
      freq = f;
   }

   public int compareTo(HuffmanTreeNode t)
   {
      return freq - t.getFrequency();
   }
   public String toString()
   {
      return str+":"+freq;
   }
}

```