```java
import java.util.HashMap;
import java.util.Map.Entry;
import java.util.PriorityQueue;
public class HuffmanRunner
{
    public static void main(String [] args)
    {
        HuffmanTree tree = new HuffmanTree("Sally sells seashells by the seashore, the shells she sell
        System.out.println(tree.encoder("Sally sells seashells, the shells she sells are seashells"));
        System.out.println(tree.decoder("01111101100010010001000101011001001100101011011010000011001000
    }
}

import java.util.HashMap;
import java.util.*;
import java.util.PriorityQueue;
/**
 * Class that creates a HuffmanTree using HuffmanNodes
 * Creates a Priority Queue and initializes the tree itself using a Hashmap
 * @author Anish Seth
 */
public class HuffmanTree
{
    private HuffmanNode root;
    private String s;
    private HashMap<String, Integer> map;
    private PriorityQueue<HuffmanNode> queue;

    /**
    * Constructor
    * Initializes map, queue, and tree
    */
    public HuffmanTree(String string)
    {
        s = string;
        root = null;
        createqueue();
        createmap();
        createtree();
    }
    public void createqueue()
    {
        queue = new PriorityQueue<HuffmanNode>();
        Object[] key = map.keySet().toArray();
        for(int i = 0; i < key.length; i++)
        {
            queue.add(new HuffmanNode(key[i].toString(), map.get(key[i])));
        }
    }
    public void createmap()
    {
        map = new HashMap<String, Integer>();
        for(int i = 0; i < s.length(); i++)
        {
            String hold = s.charAt(i);
            if(map.containsKey(hold))
            {
                int index = map.get(i);
                map.remove(i);
                map.put(hold, index + 1);
            }
            else
            {
                map.put(hold, 1);
            }
        }
    }
    public void createtree()
    {
        HuffmanNode hold;
        while(queue.size() > 1)
        {
            hold = new HuffmanNode(queue.poll(), queue.poll());
            queue.add(hold);
```

Comments? Name?

This class will not compile because encoder does not exist.

Map has not been initialized. If you do not create the map first, this program will crash.

You can't store a char in a String - it will not compile. Did you test this code at all?

```
            root = hold;
        }
    }
    public String encode()
    {
        String encode;
        for(int i = 0; i < s.length(); i++)
        {
            HuffmanNode hold = root;
            while((hold.getLeft() != null) && (hold.getRight() != null))
            {
                if(hold.getLeft().getKey().contains(s.substring(i, i + 1)))
                {
                    hold = hold.getLeft();
                    encode += "1";
                }
                else if(hold.getRight().getKey().contains(s.substring(i, i + 1)))
                {
                    hold = hold.getRight();
                    encode += "0";
                }
            }
        }
        return encode;
    }
    public String decoder(String s)
    {
        String decode = "";
        int i = 0;
        while(i < s.length())
        {
            HuffmanNode hold = root;
            while((hold.getLeft() != null) && (hold.getRight() != null))
            {
                if(s.substring(i, i + 1).equals("1"))
                    hold = hold.getLeft();
                else if(s.substring(i, i + 1).equals("0"))
                    hold = hold.getRight();
                i++;
            }
            decode += hold.getKey();
        }
        return decode;
    }
    public String toString()
    {
        return root.toString();
    }
}

/**
 * A Huffman Node class with the proper constructors, accessors, and modifiers
 * @author Anish Seth
 * @version 1-28-16
 */
public class HuffmanNode implements Comparable<HuffmanNode>
{
    private String key;
    private int value;
    private HuffmanNode left;
    private HuffmanNode right;
    /**
    * Sets key to parameter while setting value, left, and right pointers to null
    * @param k Key for the node
    */
    public HuffmanNode(String k)
    {
        key = k;
        value = 0;
        left = null;
        right = null;
    }
    /**
     * Sets key and value to parameters while setting left and right pointers to null
```

These 2 methods are identical to your previous submission that I asked you to re-do. All you've done is change a few variable names and delete the comments.

The variable key is not a good choice for a variable name. It implies that this is a map.

```
149         * @param k Key for the node
150         * @param v Value for the node
151         */
152        public HuffmanNode(String k, int v)
153        {
154            key = k;
155            value = v;
156            left = null;
157            right = null;
158        }
159        /**
160         * Sets key, value, and both pointers to parameters
161         * @param k Key for the node
162         * @param v Value for the node
163         * @param l pointer for the left node of the current node
164         * @param r pointer for the right node of the current node
165         */
166        public HuffmanNode(String k, int v, HuffmanNode l, HuffmanNode r)
167        {
168            key = k;
169            value = v;
170            left = l;
171            right = r;
172        }
173        /**
174         * Accessor that returns the node to the left of the current node
175         * @return Huffman Node to the left of the current node
176         */
177        public HuffmanNode getLeft()
178        {
179            return left;
180        }
181        /**
182         * Accessor that returns the node to the right of the current node
183         * @return Huffman Node to the right of the current node
184         */
185        public HuffmanNode getRight()
186        {
187            return right;
188        }
189        /**
190         * Accessor that returns the key of the current node
191         * @return key of the current node
192         */
193        public String getKey()
194        {
195            return key;
196        }
197        /**
198         * Accessor that returns the value of the current node
199         * @return value of the current node
200         */
201        public int getValue()
202        {
203            return value;
204        }
205        /**
206         * Modifier that sets the left pointer to the parameter
207         * @param l New node the left pointer will be set to
208         */
209        public void setLeft(HuffmanNode l)
210        {
211            left = l;
212        }
213        /**
214         * Modifier that sets the right pointer to the parameter
215         * @param l New node the right pointer will be set to
216         */
217        public void setRight(HuffmanNode r)
218        {
219            right = r;
220        }
221        /**
222         * Modifier that sets the key to the parameter
```

```
223      * @param k New String the key will be set to
224      */
225     public void setKey(String k)
226     {
227         key = k;
228     }
229     /**
230      * Modifier that sets the value to the parameter
231      * @param v New int the value will be set to
232      */
233     public void setValue(int v)
234     {
235         value = v;
236     }
237     /**
238      * Checks whether or not the current Huffman Node is a leaf
239      * @return true if it is a leaf, false otherwise
240      */
241     public boolean isLeaf()
242     {
243         return (left == null && right == null);
244     }
245     /**
246      * String representation of the current Huffman Node
247      */
248     public String toString()
249     {
250         if(isLeaf())
251             return key + ": " + value;
252         else
253         {
254             if (right == null)
255                 return key + value + "(," + left.toString() + ")";
256             else if(left == null)
257                 return key + value + "(," + right.toString() + ")";
258             return key + value + "(" + left.toString() + "," + right.toString() + ")";      }
259     }
260     /**
261      * Compares the values of the current Huffman Node and the parameter
262      * @param o Huffman Node the current node will be compared to
263      * @return If the current node is greater than the parameter, return a positive value
264      * @return If they are equal, return 0
265      * @return If the parameter is greater than the current node, return a negative value
266      */
267     public int compareTo(HuffmanNode o)
268     {
269         return value - o.getValue();
270     }
271
272 }
273
```

Anish - This needs a good deal more work. Both your tree and runner classes do not even compile. If your tree did compile, it would crash because of a pretty obvious logic error in your algorithm. Did you test this code?
Also, your tree class has almost no commenting. Given the issues with your previous submission, it's critical that you are able to demonstrate that you know how the code works. Your encode and decode methods are essentially exactly the same as your previous version. I realize that those methods can only be solved a certain number of ways, but these still appear to be copied (and again, comments would help avoid this problem!).
Given that this is your 2nd chance at this assignment and it does not even compile, I cannot give this a grade above a D+ (68%).