```java
import java.util.HashMap;
import java.util.PriorityQueue;
public class HuffmanRunner
{
    public static void main(String [] args)
    {
        HuffmanTree tree1 = new HuffmanTree("Mississippi river");
        System.out.println(tree1.encode("Mississippi river"));
        System.out.println(tree1.decode("0101001111001111001011010010010110001001000011"));
    }
}

import java.util.HashMap;
import java.util.PriorityQueue;
/**
Huffman Tree Class
*/
public class HuffmanTree
{
    /**root of the tree*/
    private HuffmanNode root;

    /**map for the tree*/
    private HashMap<String, Integer> map;

    /**queue for the tree*/
    private PriorityQueue<HuffmanNode> queue;

    /**
    constructor that takes in a string
    @param s String that is going to be turned into huffman code
    */
    public HuffmanTree(String s)
    {
        map = new HashMap<String, Integer>(); //creates map and queue
        queue = new PriorityQueue<HuffmanNode>();
        for(int i = 0; i < s.length(); i++)
        {
            if(map.containsKey(s.substring(i, i + 1))) //calls substring method to get character in th
                map.put(s.substring(i, i + 1), map.get(s.substring(i, i + 1)) + 1); //if character is
            else
            {
                map.put(s.substring(i, i + 1), 1); //adds key with a value of one if the key is not al
            }
        }
        createQueue();
        createTree();
    }

    /**
    creates the queue
    @return PriorityQueue<HuffmanNode> returns the priority queue
    */
    public PriorityQueue<HuffmanNode> createQueue()
    {
        String[] keys =  map.keySet().toArray(new String[0]); //creates a string array of the keys, ca
        for(int i = 0; i < keys.length; i++)
        {
            queue.add(new HuffmanNode(keys[i], map.get(keys[i])));
        }
        return queue;
    }

    /**
    creates the tree
    @return HuffmanNode returns the root of the tree
    */
    public HuffmanNode createTree()
    {
        while(queue.size() > 1) //goes through loop until siz of queue is one
        {
            HuffmanNode end = queue.poll();  //takes out first two nodes in the queue
            HuffmanNode end2 = queue.poll();
            HuffmanNode toAdd = new HuffmanNode(end.getValue() + end2.getValue(), end.getCount() + end
```

Description? Name?

Do map and queue need to be class fields?

Watch line lengths.

```
            root = toAdd;  //assigns root to toAdd so after last iteration root will be top of tree
            queue.add(toAdd);  //readds combination of the 2 nodes
            toAdd.setLeft(end);  //assigns left and right pointers for toAdd
            toAdd.setRight(end2);
        }
        return root;
    }

    /**
    returns the HuffmanCode of an inputted String
    @param s String that is going to be turned into HuffmanCode
    @return String returns the HuffmanCode of the inputted String
    */
    public String encode(String s)
    {
        String output = "";
        for(int i = 0; i < s.length(); i++) //iterates through charcaters in the string
        {
            HuffmanNode node = root;
            while((node.getLeft() != null) && (node.getRight() != null)) //goes until node is a leaf
            {
                if(node.getLeft().getValue().contains(s.substring(i, i + 1))) //goes to left pointer i
                {
                    node = node.getLeft();
                    output += "1";  //1 is added to output if iteration goes to left node
                }
                else if(node.getRight().getValue().contains(s.substring(i, i + 1)))
                {
                    node = node.getRight();
                    output += "0";  //0 is added to output if iteration goes to right node
                }
            }
        }
        return output;
    }

    /**
    returns the String value of an inputted HuffmanCode
    @param s inputted huffman code that is turned into a word
    @return String returns String of the huffman code
    */
    public String decode(String s)
    {
        String output = "";
        int i = 0;
        while(i < s.length()) //iterates through HuffmanCode
        {
            HuffmanNode node = root;
            while((node.getLeft() != null) && (node.getRight() != null))  //interation goes until node
            {
                if(s.substring(i, i + 1).equals("1"))  //iteration goes left if character in huffman c
                    node = node.getLeft();
                else if(s.substring(i, i + 1).equals("0"))  //iteration goes right if character in huf
                    node = node.getRight();
                i++;
            }
            output += node.getValue();  //value at node is added to output, should be a single letter
        }
        return output;
    }

    /**
    returns String representation of the tree
    @return String returns String representation of the tree
    */
    public String toString()
    {
        return root.toString();
    }

}

/**
    Node Class for Huffman Tree
```

Good.

Good.

```
*/
public class HuffmanNode implements Comparable<HuffmanNode>
{
    /**node to the left of this node*/
    private HuffmanNode left;

    /**node to the right of this node*/
    private HuffmanNode right;

    /**String value stored in this node*/
    private String value;

    /**number of times value occurs*/
    private int count;

    /**
    constructor that only takes in a String
    @param s String that is going to be assigned to value
    */
    public HuffmanNode(String s)
    {
        left = null;
        right = null;
        value = s;
        count = 0;
    }

    /**
    constructor that takes in a string and an int
    @param s String that is going to be assigned to value
    @param i int that is going to be assigned to count
    */
    public HuffmanNode(String s, int i)
    {
        left = null;
        right = null;
        value = s;
        count = i;
    }

    /**
    access for the left node
    @return HuffmanNode left node
    */
    public HuffmanNode getLeft()
    {
        return left;
    }

    /**
    accessor for the right node
    @return HuffmanNode right node
    */
    public HuffmanNode getRight()
    {
        return right;
    }

    /**
    accessor for value
    @return String return value
    */
    public String getValue()
    {
        return value;
    }

    /**
    accessor for count
    @return int returns count
    */
    public int getCount()
    {
        return count;
```

```
223     }
224
225     /**
226     modifier for left
227     @param node node that is going to be assigned to left
228     */
229     public void setLeft(HuffmanNode node)
230     {
231         left = node;
232     }
233
234     /**
235     modifier for right
236     @param node node that is going to be assigned to right
237     */
238     public void setRight(HuffmanNode node)
239     {
240         right = node;
241     }
242
243     /**
244     modifier for value
245     @param s String that is going to be assigned to value
246     */
247     public void setValue(String s)
248     {
249         value = s;
250     }
251
252     /**
253     modifier for count
254     @param i int that is going to be assigned to count
255     */
256     public void setCount(int i)
257     {
258         count = i;
259     }
260
261     /**
262     toString method
263     @return s String representation of the node
264     */
265     public String toString()
266     {
267         String s = "";
268         s += value + ": " + count;  //prints value with count
269         return s;
270     }
271
272     /**
273     compares this node with another inputted node
274     @param node node that is compared to this node
275     @return int difference between the counts in the nodes
276     */
277     public int compareTo(HuffmanNode node)
278     {
279         return count - node.getCount();
280     }
281
282 }
```

Good job. A