```java
//package HuffmanCode;

import java.util.ArrayList;
import java.util.Map;
import java.util.TreeMap;
import java.util.PriorityQueue;

/**
 * GOOGLEDOC: https://docs.google.com/document/d/12gql02Qv1YZTXcmkfTesYkv1C1h7sdvRQ6-iKjhr5zc/edit
 * Creates a Map of key / value pairs (key represents the character, value represents its frequency) c
 * Creates a HuffmanNode of each key / value pair and sorts them in a PriorityQueue
 *
 * @author ingrid
 *
 */
public class HuffmanRunner {

    public static void main(String[] args)
    {
        // phrase to use to encode
        String s = "Sally sells sea shells";
        // places k / v pair in Map
        // Key: char
        // Value: frequency
        Map<Character, Integer> map = new TreeMap<Character, Integer>();
        ArrayList<Character> listOfKeys = new ArrayList<Character>();

        for(int i = 0; i < s.length(); i++)
        {
            if(map.containsKey(s.charAt(i)))
            {
                // increments value of occurrences (by putting new entry and removing old one)
                Character tempKey = s.charAt(i);
                Integer tempVal = map.get(tempKey) + 1;

                map.remove(tempKey);
                map.put(tempKey, tempVal);
```

I'm not sure why any of this would be in a runner file.

```java
            }
            else
            {
                listOfKeys.add(s.charAt(i));
                map.put(s.charAt(i), 1);
            }
        }
        System.out.println("MAP: " + map);
        System.out.println("LIST OF KEYS: " + listOfKeys);


        // Put HuffmanNodes containing the String and the frequency in PriorityQueue
        PriorityQueue<HuffmanNode> queue = new PriorityQueue<HuffmanNode>();
        HuffmanNode tempNode;
        for(int i = 0; i < listOfKeys.size(); i++)
        {
            tempNode = new HuffmanNode(Character.toString(listOfKeys.get(i)), map.get(listOfKeys.get(i
            queue.add(tempNode);
        }

        // to test queue: remove (and print as you remove) and they should be in the correct order
        // WORKS
        /*
        while(queue.isEmpty() == false)
        {
            System.out.println(queue.poll());
        }*/
```

So, at this point queue.size() == 1?

```java

        // Test INITIALIZING
        // creates tree using the first HuffmanNode in queue as the root
        HuffmanTree tree = new HuffmanTree(queue.peek());
        tree.initializeTree(queue);
        System.out.println("INITIALIZED TREE: " + tree.getRoot());

        // Test ENCODING
        System.out.println(tree.encode("Shells"));
        System.out.println("ENCODED to DECODE: " + tree.decode(tree.encode("Shells")));
```

```java

        // Test DECODING
        // list values spell "Shells"
        Integer[] tempArray = new Integer[]{1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1};
        ArrayList<Integer> list = new ArrayList<Integer>();
        for(Integer i : tempArray)
        {
            list.add(i);
        }
        //System.out.println(list);
        System.out.println("DECODED: " + tree.decode(list));

        // list values spell "hey yes"
        Integer[] tempArray1 = new Integer[]{0,1,0,1,1,0,0,1,0,1,0,0,1,1,1,0,1,0,1,0,0,1,1};
        ArrayList<Integer> list1 = new ArrayList<Integer>();
        for(Integer i : tempArray1)
        {
            list1.add(i);
        }
        System.out.println("DECODED: " + tree.decode(list1));
```

Wouldn't using Strings be easier?

```java

        // Test HuffmanNode
        /*
        HuffmanNode node = new HuffmanNode(Character.toString('a'), 0);

        System.out.println(node.getValue());
        System.out.println(node.getCount());
        System.out.println(node);

        HuffmanNode node2 = new HuffmanNode(Character.toString('b'), 1, node, null);
        System.out.println(node2);

        HuffmanNode node3 = new HuffmanNode(Character.toString('c'), 1);

        // compareTo
        System.out.println(node.compareTo(node2));
        System.out.println(node2.compareTo(node));
        System.out.println(node3.compareTo(node2));*/

    }

}

//package HuffmanCode;

import java.util.ArrayList;
import java.util.PriorityQueue;

/**
 * How Huffman Code works:
 * Given a string, the program generates a Map of key / value pairs containing the frequency of each c
 * This data is organized in a PriorityQueue (from least to most occurrences), which is then sorted vi
 * Huffman Code: generates a tree where each leaf is a different value. To trace through the tree, use
 * (0 = left, 1 = right)
 *
 * @author ingrid
 *
 */
public class HuffmanTree {

    private HuffmanNode root;

    public HuffmanTree(HuffmanNode n)
    {
        root = n;
    }
```

This would work better if it took in a String. You end up having your code essentially do the same work twice.

```java


    /**
     * accessor
     * @return returns HuffmanNode root
     */
    public HuffmanNode getRoot()
    {
```

```java
        return root;
    }


    /**
     * returns root of initialized tree
     * @param p
     * @return root, which can be used to access the entire initialized tree
     */
    public HuffmanNode initializeTree(PriorityQueue<HuffmanNode> p)
    {
        HuffmanNode node1;
        HuffmanNode node2;
        String newKey;
        Integer newVal;

        while(p.size() >= 2)
        {
            // poll first two elements in queue and combine them into a new parent node (concatenate s
            // and set left and right pointers of parent node (newNode) to correct children
            node1 = p.poll();
            node2 = p.poll();
            newKey = node1.getValue() + node2.getValue();
            newVal = node1.getCount() + node2.getCount();

            HuffmanNode newNode = new HuffmanNode(newKey, newVal, node1, node2);

            // place the parent node back in the queue (based on their value)
            p.add(newNode);

            // set root to last remaining HuffmanNode in queue
            if(p.size() == 1)
            {
                root = newNode;
            }
        }

        return root;
    }


    /**
     * encodes String param into an ArrayList based on initialized String
     * @precondition tree has already been initialized
     * @precondition is possible to encode given the String used to initialize it
     * @param toEncode
     * @return ArrayList<Integer> of param toEncode
     */
    public ArrayList<Integer> encode(String toEncode)
    {
        ArrayList<Integer> encoded = new ArrayList<Integer>();
        String tempString;
        HuffmanNode tempNode = root;

        while(toEncode.length() != 0)
        {
            tempString = toEncode.substring(0,1);
            // check for presence of char (tempString) in left / right nodes. when present, add 0(L) o
            while(tempNode.isLeaf() == false)
            {
                if(tempNode.getLeft().getValue().contains(tempString))
                {
                    encoded.add(0);
                    tempNode = tempNode.getLeft();
                }
                else if(tempNode.getRight().getValue().contains(tempString))
                {
                    encoded.add(1);
                    tempNode = tempNode.getRight();
                }
            }
            // keep doing this until isLeaf. then create substring of toEncode w/o first char and rese
            if(tempNode.isLeaf())
            {
```

```
                toEncode = toEncode.substring(1);
                tempNode = root;
            }

        }

        return encoded;
    }

    /**
     * returns decoded message
     * @precondition tree has already been initialized
     * @precondition toDecode contains only binary values
     * @param toDecode, list of binary with 0 representing left node and 1 representing right node
     * @return decoded message
     */
    public String decode(ArrayList<Integer> toDecode)
    {
        String decoded = "";
        HuffmanNode n = root;
        Integer i;

        while(toDecode.isEmpty() == false)
        {
            // traverse through tree using values from list until gets to a leaf
            i = toDecode.remove(0);
            // 0 = left node
            if(i.equals(0))
            {
                n = n.getLeft();
                if(n.isLeaf())
                {
                    decoded += n.getValue();
                    // restart from root of tree
                    n = root;
                }
            }
            // 1 = right node
            else
            {
                n = n.getRight();
                if(n.isLeaf())
                {
                    decoded += n.getValue();
                    n = root;
                }
            }
        }
        return decoded;
    }
}

//package HuffmanCode;

/**
 * This HuffmanNode should behave like a normal TreeNode (e.g. have a left and right pointer), except
 * an Object, it will contain both a Character (named value) and an int (named count). Each should hav
 * Since you will be placing these HuffmanNodes into a PriorityQueue, they will also have to be Compar
 * be based on the value of the int. If the value is the same, the comparison should return equal (0).
 *
 * @author ingrid
 *
 */
public class HuffmanNode implements Comparable<HuffmanNode> {

    private HuffmanNode right;
    private HuffmanNode left;
    private String value;
    // count = frequency
    private int count;

    /**
     * Constructor
     * @param v
```

```java
     * @param c
     */
    public HuffmanNode(String v, int c)
    {
        value = v;
        count = c;

        left = null;
        right = null;
    }


    /**
     * Constructor for
     * @param v String value
     * @param c Frequency C
     * @param r
     * @param l
     */
    public HuffmanNode(String v, int c, HuffmanNode r, HuffmanNode l)
    {
        value = v;
        count = c;

        right = r;
        left = l;
    }

    // accessors
    /**
     * returns value of char
     * @return value of char
     */
    public String getValue()
    {
        return value;
    }


    /**
     * value of count
     * @return value of count
     */
    public int getCount()
    {
        return count;
    }


    /**
     * returns left HuffmanNode
     * @return left HuffmanNode
     */
    public HuffmanNode getLeft()
    {
        return left;
    }


    /**
     * returns right HuffmanNode
     * @return right HuffmanNode
     */
    public HuffmanNode getRight()
    {
        return right;
    }


    //modifiers
    /**
     * sets value to c
     * @param c
     * @return char that was replaced
```

```
     */
    public String setValue(String c)
    {
        String temp = value;
        value = c;

        return temp;
    }


    /**
     * sets count to i
     * @param i
     * @return previous value of count
     */
    public int setCount(int i)
    {
        int temp = count;
        count = i;

        return temp;
    }


    /**
     * sets right node to n
     * @param n
     */
    public void setRight(HuffmanNode n)
    {
        right = n;
    }

    /**
     * sets left node to n
     * @param n
     */
    public void setLeft(HuffmanNode n)
    {
        left = n;
    }

    /**
     * compare value of ints of this vs o
     * returns + int if this value is greater, 0 if values are equal, - int if this value is less
     * @param o
     * @return relationship of HuffmanNodes based on their values
     */
    public int compareTo(HuffmanNode o) {
        return count - o.getCount();
    }


    /**
     * returns whether right and left are both null
     * @return whether right and left are both null
     */
    public boolean isLeaf()
    {
        return right == null && left == null;
    }

    /**
     * returns String representation of Huffman node and its children
     * @return String representation of Huffman node and its children
     */
    public String toString()
    {
        String s = "";
        if(isLeaf())
        {
            s += value;
            return s;
        }
```

```java
        s += value;
        s += "[";
        if(left != null)
        {
            s += left.toString();
        }
        s += "]";
        s += "[";
        if(right != null)
        {
            s += right.toString();
        }
        s += "]";
        return s;
    }

}
```

Overall, this seems to work. I say seems because the way you've designed your runner, it is challenging to separate the work it does versus the function of the HuffmanTree. You veer away from the design specifications of the lab, which is completely fine; the problem is the changes you make create a muddled design. That said, it is clear you do understand the algorithm and your encode and decode methods are easy to follow.
A-