```java
import java.io.IOException;

/**
 * Implementation of HuffmanTree by
 * @author Amelia Do
 */

public class HuffmanTree {
    public HNode root;

    /**
     * builds a Huffman Tree from the given characters and
     * their corresponding frequencies.
     * @param frequencies - given characters and frequencies to build Huffman Tree from
     */
    public HuffmanTree(TreeMap<Character, Integer> frequencies)
    {
        PriorityQueue<HNode> queue = new PriorityQueue<>(new HNodeComparator());
        for (Entry<Character, Integer> entry : frequencies.entrySet()) {
            queue.add(new HNode(entry.getKey(), entry.getValue()));
        }

        while ( queue.size() > 1 ) {
            HNode left = queue.poll();
            HNode right = queue.poll();
            HNode merged = new HNode(left, right);
            queue.add(merged);
        }
        root = queue.poll();
    }

    /**
     * returns the binary encoding of the given symbol as
     * a string of '0' and '1' characters
     * @param symbol - the symbol to be encoded
```

```java
     * @return the binary encoding of the given symbol
     */
    public String encodeLoop(char symbol)
    {
        String str = "";
        HNode curr = root;

        while ( !curr.isLeaf() ) {
            if ( curr.right.contains(symbol) ) {
                curr = curr.right;
                str += 1;
            }
            else if ( curr.left.contains(symbol) ) {
                curr = curr.left;
                str += 0;
            }
        }

        if ( curr.isLeaf() && !curr.contains(symbol) ) {
            return null;
        }
        else {
            return str;
        }
    }

    /**
     * returns the binary encoding of the given symbol as
     * a string of '0' and '1' characters
     * @param symbol - the symbol to be encoded
     * @return the binary encoding of the given symbol
     */
    public String encode(char symbol)
    {
        return encode(symbol, root);
    }

    /**
     * returns the binary encoding of the given symbol as
```

```java
     * a string of '0' and '1' characters
     * @param symbol - the symbol to be encoded
     * @return the binary encoding of the given symbol
     */
    public String encode(char symbol, HNode curr)
    {
        if ( curr == null ) {
            return null;
        }

        if ( curr.isLeaf() ) {
            if ( curr.contains(symbol) ) {
                return "";
            }
            else {
                return null;
            }
        }

        String leftPath = encode(symbol, curr.left);
        if (leftPath != null) {
            return "0" + leftPath;
        }

        String rightPath = encode(symbol, curr.right);
        if (rightPath != null) {
            return "1" + rightPath;
        }

        return "";
    }

    /**
     * Returns the symbol that corresponds to the given code
     * @param code - the given code to be decoded
     * @return the symbol that corresponds to the given code
     */
    public char decode(String code)
    {
```

```
117         HNode curr = root;
118 
119         for ( int i = 0; i < code.length(); i++ ) {
120             char bit = code.charAt(i);
121 
122             if ( bit == '0' ) {
123                 curr = curr.left;
124             }
125             else if ( bit == '1' ) {
126                 curr = curr.right;
127             }
128 
129             if (curr == null) {
130                 return '\0';
131             }
132         }
133 
134         if ( curr.isLeaf() ) {
135             return curr.getSymbol();
136         }
137         else {
138             return '\0';
139         }
140     }
141 
142     /**
143      * Writes the individual bits of the binary encoding of the
    given
144      * symbol to the given bit stream
145      * @param symbol - given symbol to write the code for
146      * @param stream - given stream to be written onto
147      * @throws IOException
148      */
149     public void writeCode(char symbol, BitOutputStream stream)
    throws IOException
150     {
151         String binaryEncoding = encodeLoop(symbol);
152         if (binaryEncoding == null) {
153             return;
```

```
154         }
155         else {
156             for (char bit : binaryEncoding.toCharArray()) {
157                 stream.writeBit(bit == '1' ? 1 : 0);
158             }
159         }
160     }
161 
162 
163 
164     /**
165      * Reads from the given stream the individual bits of the
166      * binary encoding of the next symbol
167      * @param stream - given stream to be read from
168      * @return the given stream the individual bits of the
    binary encoding
169      */
170     public char readCode(BitInputStream stream) throws
    IOException
171     {
172         if ( root == null ) {
173             return '\0';
174         }
175 
176         HNode curr = root;
177         while ( !curr.isLeaf() ) {
178             int bit = stream.readBit();
179             curr = (bit == 0) ? curr.left : curr.right;
180         }
181 
182         return curr.getSymbol();
183     }
184 }
185 
186 
187 
```