Generating the Huffman Tree

```
public void genTree() {
    while(freq.size() > 1 )
    {
        int sumFreq = ((TreeNode) freq.get(index: 1)).getItem().getFreq() +
        ((TreeNode) freq.get(index: 2)).getItem().getFreq();

        TreeNode left_Right = new TreeNode(
                new LetterFreq(letter: '#', sumFreq),
                ((TreeNode) freq.get(index: 1)),
                ((TreeNode) freq.get(index: 2))

        );
        freq.remove(index: 1);
        freq.remove(index: 1);
        freq.add(freq.size()+1, left_Right);
        freq.bubbleSort();
    }
    root = ((TreeNode)freq.get(index: 1));

}
```

In order to generate the tree, I used genTree function to convey the tree generation. Functions loops through each frequency getting the top two , adding the frequencies together and adding them as left child and right child with a ‘#’ proceeds to remove the top two and add them to the new node. It will bubble sort it.Huffman.java: Line 10

Encode

Goes through the bits letter traversal through the tree looking at the left node and then the right node. Uses the final binary variable to hold the bit needed to be encoded

```
String finalBinary = "";
public void  getBits(char letter,String tempBinary, TreeNode node)
{
    if(node == null){
        return;
    }
    char currLetter =((LetterFreq)node.getItem()).getLetter();
    if(currLetter == letter)
    {
        finalBinary = tempBinary;
        return;
    }


    getBits(letter, tempBinary+'0', node.getLeft());
    getBits(letter, tempBinary+'1', node.getRight());



}
```

Encodes using the getBit function returns an encoded

```
public String encode(String decodeBits)
{
    decodeBits = decodeBits.toUpperCase();
    String encodeBits = "";

    for(int i = 0; i < decodeBits.length(); i++)
    {
        getBits(decodeBits.charAt(i), tempBinary: "", root);
        encodeBits += finalBinary;
    }

    return encodeBits;
}
```

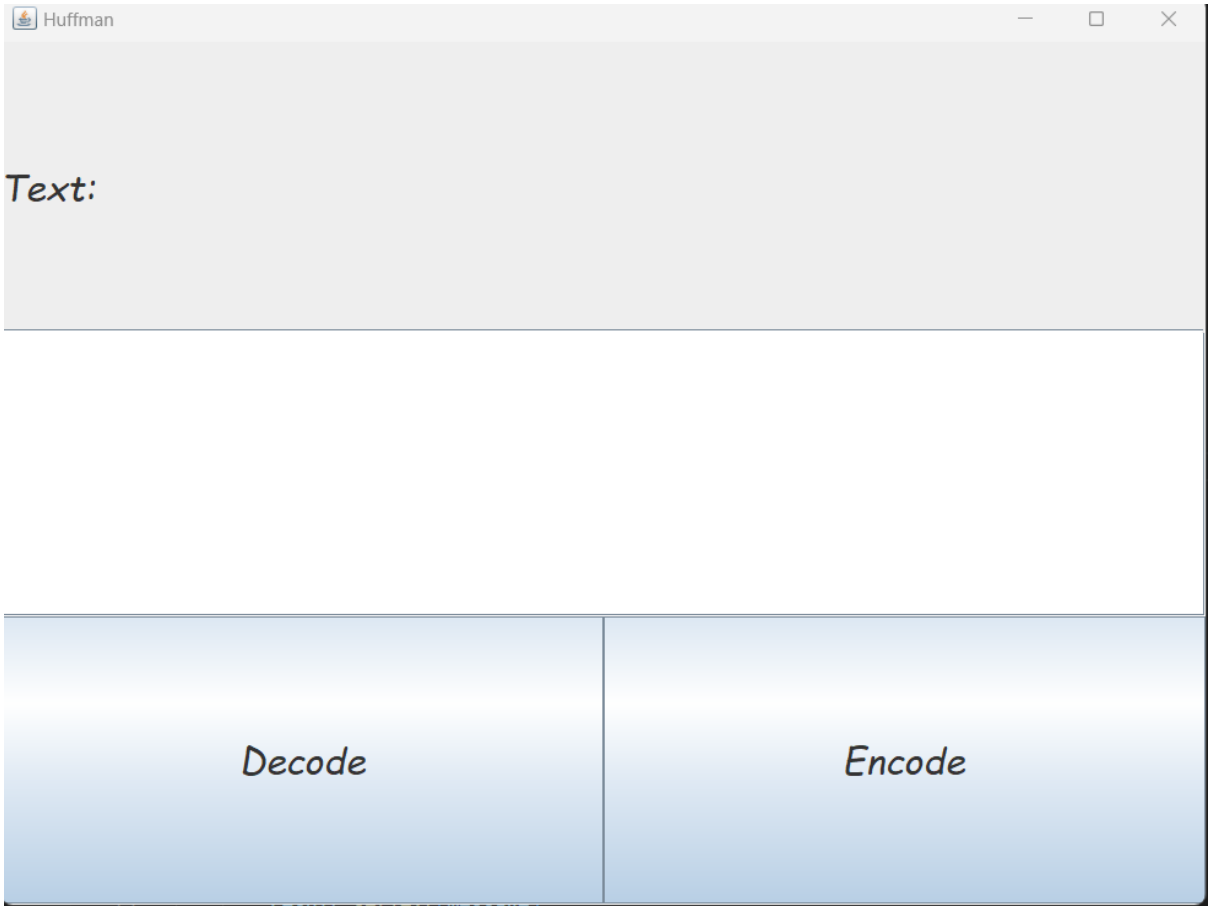
Decode

```
public String decode(String encodeBits)
{
    String decodeBits = "";
    TreeNode curr = root;

    for(int i = 0; i < encodeBits.length(); i++)
    {
        char bit = encodeBits.charAt(i);
        if(((LetterFreq)curr.getItem()).getLetter() != '#' )
        {
            decodeBits += ((LetterFreq)curr.getItem()).getLetter();
            curr = root;
        }
        if(bit == '0' )
        {
            curr = curr.getLeft();
        }else if(bit == '1')
        {
            curr = curr.getRight();
        }

    }
    decodeBits += ((LetterFreq)curr.getItem()).getLetter();

    return decodeBits;
}
```

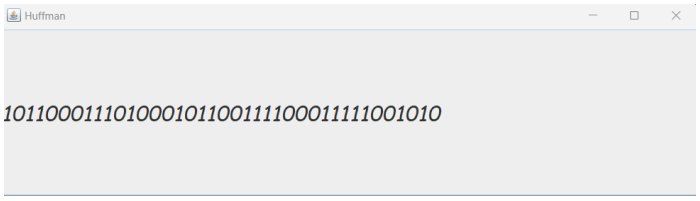
Decode method ‘decode’ loops through a bit and if the current bit is not equal to the ‘#’ it gets the letter from the tree goes left and right of the node and when done prints out the final value of the string.

Program Interface

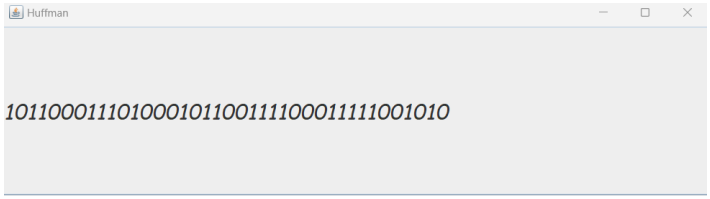
This program displays the two buttons ‘encode’ and ‘decode’ which performs their actions stated. The text label will display the either decode or encode text of the input. The user inputs the text into the textfield box.



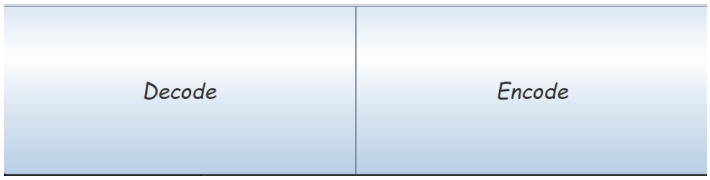
Test Case



BENJAMIN



Benjamin



INPUT	**Output**	**Valid**
BENAJMIN	10110001110100010110011110001111100101O	Y
Benjamin	10110001110100010110011110001111100101O	Y

Academic Honesty

Any work that you submit for continuous assessment or assignments must be done by you. Failure to acknowledge the source of a significant idea or approach is considered plagiarism and not allowed. Academic dishonesty will be dealt with severely. At a minimum, you will receive a mark of zero for the assignment.

Signed: Benjamin Manzambi

Date: 1/12/2022