```java
import java.util.*;

abstract class HuffmanTree implements Comparable<HuffmanTree> {
    public final int frequency; // the frequency of this tree
    public HuffmanTree(int freq) { frequency = freq; }

    // compares on the frequency
    public int compareTo(HuffmanTree tree) {
        return frequency - tree.frequency;
    }
}

class HuffmanLeaf extends HuffmanTree {
    public final char value; // the character this leaf represents

    public HuffmanLeaf(int freq, char val) {
        super(freq);
        value = val;
    }
}

class HuffmanNode extends HuffmanTree {
    public final HuffmanTree left, right; // subtrees

    public HuffmanNode(HuffmanTree l, HuffmanTree r) {
        super(l.frequency + r.frequency);
        left = l;
        right = r;
    }
}

public class HuffmanCode {
    // input is an array of frequencies, indexed by character code
    public static HuffmanTree buildTree(int[] charFreqs) {
        PriorityQueue<HuffmanTree> trees = new PriorityQueue<HuffmanTree>();
        // initially, we have a forest of leaves
        // one for each non-empty character
        for (int i = 0; i < charFreqs.length; i++)
            if (charFreqs[i] > 0)
                trees.offer(new HuffmanLeaf(charFreqs[i], (char)i));

        assert trees.size() > 0;
        // loop until there is only one tree left
        while (trees.size() > 1) {
            // two trees with least frequency
            HuffmanTree a = trees.poll();
            HuffmanTree b = trees.poll();

            // put into new node and re-insert into queue
            trees.offer(new HuffmanNode(a, b));
        }
        return trees.poll();
    }

    public static void printCodes(HuffmanTree tree, StringBuffer prefix) {
        assert tree != null;
        if (tree instanceof HuffmanLeaf) {
            HuffmanLeaf leaf = (HuffmanLeaf)tree;

            // print out character, frequency, and code for this leaf (which is just the prefix)
            System.out.println(leaf.value + "\t" + leaf.frequency + "\t" + prefix);

        } else if (tree instanceof HuffmanNode) {
            HuffmanNode node = (HuffmanNode)tree;

            // traverse left
            prefix.append('0');
            printCodes(node.left, prefix);
            prefix.deleteCharAt(prefix.length()-1);
            // traverse right
            prefix.append('1');
            printCodes(node.right, prefix);
            prefix.deleteCharAt(prefix.length()-1);
        }
    }

    public static void main(String[] args) {
        String test = "this is an example for huffman encoding";

        // we will assume that all our characters will have
        // code less than 256, for simplicity
        int[] charFreqs = new int[256];
        // read each character and record the frequencies
        for (char c : test.toCharArray())
            charFreqs[c]++;

        // build tree
        HuffmanTree tree = buildTree(charFreqs);

        // print out results
        System.out.println("SYMBOL\tWEIGHT\tHUFFMAN CODE");
        printCodes(tree, new StringBuffer());
    }
}
```

Example output:

```
SYMBOL  WEIGHT  HUFFMAN CODE
d       1       00000
t       1       00001
h       2       0001
s       2       0010
c       1       00110
x       1       00111
m       2       0100
o       2       0101
n       4       011
u       1       10000
l       1       10001
a       3       1001
r       1       10100
g       1       101010
p       1       101011
e       3       1011
i       3       1100
f       3       1101
        6       111
```

JavaScript

Translation of: Ruby
Works with: SpiderMonkey
for the `print()` function.

First, use the Binary Heap implementation from here: http://eloquentjavascript.net/appendix2.html