This code ***Ideally*** should have both compression and decompression :  
  
After compressing the message and generating Huffman codes, it then encodes the original message using the generated Huffman codes. Finally, it decodes the encoded message back to its original form using the Huffman tree.  
  
  
*Code:*  
import java.util.\*;

int frequency;

char data;

HuffmanNode left, right;

public HuffmanNode(char data, int frequency) {

this.data = data;

this.frequency = frequency;

left = right = null;

}

@Override

public int compareTo(HuffmanNode o) {

return this.frequency - o.frequency;

}

}

public class Huffman {

public static void main(String[] args) {

String text = " Peter Kadhila is the true Chief, 222042192.";

// Compression

Map<Character, Integer> frequencyMap = new HashMap<>();

for (char c : text.toCharArray()) {

frequencyMap.put(c, frequencyMap.getOrDefault(c, 0) + 1);

}

PriorityQueue<HuffmanNode> priorityQueue = new PriorityQueue<>();

for (Map.Entry<Character, Integer> entry : frequencyMap.entrySet()) {

priorityQueue.offer(new HuffmanNode(entry.getKey(), entry.getValue()));

}

HuffmanNode root = buildHuffmanTree(priorityQueue);

Map<Character, String> huffmanCodes = new HashMap<>();

generateCodes(root, "", huffmanCodes);

String encodedMessage = encodeMessage(text, huffmanCodes);

System.out.println("Encoded Message: " + encodedMessage);

// Decompression

String decodedMessage = decodeMessage(encodedMessage, root);

System.out.println("Decoded Message: " + decodedMessage);

}

public static HuffmanNode buildHuffmanTree(PriorityQueue<HuffmanNode> priorityQueue) {

while (priorityQueue.size() > 1) {

HuffmanNode left = priorityQueue.poll();

HuffmanNode right = priorityQueue.poll();

HuffmanNode parent = new HuffmanNode('$', left.frequency + right.frequency);

parent.left = left;

parent.right = right;

priorityQueue.offer(parent);

}

return priorityQueue.peek();

}

public static void generateCodes(HuffmanNode root, String code, Map<Character, String> huffmanCodes) {

if (root == null) {

return;

}

if (root.left == null && root.right == null) {

huffmanCodes.put(root.data, code);

return;

}

generateCodes(root.left, code + "0", huffmanCodes);

generateCodes(root.right, code + "1", huffmanCodes);

}

public static String encodeMessage(String text, Map<Character, String> huffmanCodes) {

StringBuilder encodedMessage = new StringBuilder();

for (char c : text.toCharArray()) {

encodedMessage.append(huffmanCodes.get(c));

}

return encodedMessage.toString();

}

public static String decodeMessage(String encodedMessage, HuffmanNode root) {

StringBuilder decodedMessage = new StringBuilder();

HuffmanNode current = root;

for (char bit : encodedMessage.toCharArray()) {

if (bit == '0') {

current = current.left;

} else if (bit == '1') {

current = current.right;

}

if (current.left == null && current.right == null) {

decodedMessage.append(current.data);

current = root;

}

}

return decodedMessage.toString();

}

}