Huffman encoding using Greedy method

```
import java.util.Comparator;

import java.util.PriorityQueue;

import java.util.Scanner;

class fibonacchi {

 // recursive function to print the

 // huffman-code through the tree traversal.

 // Here s is the huffman - code generated.

 public static void printCode(HuffmanNode root, String s)   {

  // base case; if the left and right are null

  // then its a leaf node and we print

  // the code s generated by traversing the tree.

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

   && Character.isLetter(root.c)) {

   // c is the character in the node

   System.out.println(root.c + ":" + s);

   return;

  }

  // if we go to left then add "0" to the code.

  // if we go to the right add"1" to the code.

  // recursive calls for left and

  // right sub-tree of the generated tree.

  printCode(root.left, s + "0");

  printCode(root.right, s + "1");

 }

 // main function

 public static void main(String[] args)  {

  Scanner s = new Scanner(System.in);
```

```
// number of characters.
int n = 6;
char[] charArray = { 'a', 'b', 'c', 'd', 'e', 'f' };
int[] charfreq = { 5, 9, 12, 13, 16, 45 };
// creating a priority queue q.
// makes a min-priority queue(min-heap).
PriorityQueue<HuffmanNode> q    = new PriorityQueue<HuffmanNode>(
   n, new MyComparator());
for (int i = 0; i < n; i++) {
  // creating a Huffman node object
  // and add it to the priority queue.
  HuffmanNode hn = new HuffmanNode();

  hn.c = charArray[i];
  hn.data = charfreq[i];

  hn.left = null;
  hn.right = null;

  // add functions adds
  // the huffman node to the queue.
  q.add(hn);
}

// create a root node
HuffmanNode root = null;

// Here we will extract the two minimum value
// from the heap each time until
// its size reduces to 1, extract until
// all the nodes are extracted.
```

```
while (q.size() > 1) {

  // first min extract.
  HuffmanNode x = q.peek();
  q.poll();

  // second min extract.
  HuffmanNode y = q.peek();
  q.poll();

  // new node f which is equal
  HuffmanNode f = new HuffmanNode();

  // to the sum of the frequency of the two nodes
  // assigning values to the f node.
  f.data = x.data + y.data;
  f.c = '-';

  // first extracted node as left child.
  f.left = x;

  // second extracted node as the right child.
  f.right = y;

  // marking the f node as the root node.
  root = f;

  // add this node to the priority-queue.
  q.add(f);
}
```

```
    // print the codes by traversing the tree
    printCode(root, "");
  }
}

// node class is the basic structure
// of each node present in the Huffman - tree.
class HuffmanNode {

  int data;
  char c;

  HuffmanNode left;
  HuffmanNode right;
}
// comparator class helps to compare the node
// on the basis of one of its attribute.
// Here we will be compared
// on the basis of data values of the nodes.
class MyComparator implements Comparator<HuffmanNode> {
  public int compare(HuffmanNode x, HuffmanNode y)   {
    return x.data - y.data;
  }
}
```

Output:-

```
f:0
c:100
d:101
a:1100
b:1101
e:111
```