```
// Kenny You Paragraph Processing server
// CUSTOM OPTION: Finding shortest word in a paragraph
// that doesn't start with a vowel. (This involves Good paragraph
// construction)
//
import java.util.*;
import java.net.*;
import java.text.DecimalFormat;
import java.io.*;

public class sqserver {

      public static void main(String[] args)
      {

            // Declaration
            String inputLine;
            String outputLine;

            boolean finished;

            int index;
            int numlines;
            int port;

            ServerSocket serverSocket = null;
            Socket socket = null;

            boolean listening = true; // assume serverSocket creation

            // was OK

            // get port # from command-line

            port = Integer.parseInt(args[0]);

            // try to create a server socket

            try
            {
                  serverSocket = new ServerSocket(port);
            }
            catch(IOException e)
            {
                  System.out.println(e);
                  listening = false;
            }
            if(listening) // i.e., serverSocket successfully created
            {
                  while(true) // main processing loop
                  {
                        try
                        {
                              // Listen for a connection request from a client

                              socket = serverSocket.accept();

                              // Establish the input and output streams on the 
socket

                              PrintWriter out = new 
PrintWriter(socket.getOutputStream(), true);
                              Scanner in = new Scanner(new 
InputStreamReader(socket.getInputStream()));

                              List<String> paragraph = new 
LinkedList<String>(); //Store paragraph in
                              List<String> newparagraph= new 
LinkedList<String>(); //Store paragraph for out

                              inputLine = in.nextLine(); //Get a string from 
the client (In this case the operation.)
                              numlines = in.nextInt(); //Get # of lines from 
client
                              in.nextLine(); //Go to next line (.nextInt() 
doesn't) "new line" from halpers helpful codes


                              //Store words into paragraph
                              for(int i = 0; i < numlines; i++)
                              {
                                    paragraph.add(in.nextLine());
                              }

                              //Do reverse
                              if(inputLine.equals("reverse")){
                                    newparagraph = reverse(paragraph);
                                    out.println(newparagraph.size()); //Give 
client # of lines
                                    for (int q = 0; q < newparagraph.size() ; 
q++) {
                                          out.println(newparagraph.get(q));
                                    }
                              }
                              //Do word count
                              else if(inputLine.equals("wordcount6")){
                                    outputLine = wordcount6(paragraph) + "";
                                    out.println(1);
                                    out.println(outputLine);
                              }

                              //Do longest + vowel
                              else if(inputLine.equals("longword")){
                                    newparagraph = longword(paragraph);
                                    out.println(newparagraph.size()); //Give
client # of lines
                                    for (int q = 0; q < newparagraph.size() ;
q++) {
                                          out.println(newparagraph.get(q));
                                    }
                              }

                              //Do Vowel freq
                              else if(inputLine.equals("vowelfreq")){
                                    newparagraph = vowelfreq(paragraph);
                                    out.println(newparagraph.size()); //Give
client # of lines
                                    for (int q = 0; q < newparagraph.size() ;
q++) {
                                          out.println(newparagraph.get(q));
                                    }
                              }

                              //Do custom (shortest with vowel)
                              else if(inputLine.equals("shortword")){
                                    newparagraph = shortword(paragraph);
                                    out.println(newparagraph.size()); //Give
client # of lines
                                    for (int q = 0; q < newparagraph.size() ;
q++) {
                                          out.println(newparagraph.get(q));
                                    }
                              }

                              //If nothing matches print invalid operation
                              else{
                                    out.println(1);
                                    out.println("Invaild Operation");
                              }
                              // close connection to client

                              out.close();
                              in.close();
                              socket.close();

                        }
                        catch(IOException e)
                        {
                              System.out.println(e);
                        }

                  } // end while (main processing loop)

            } // end if listening

      } // end main

      //Reverse
      public static List<String> reverse(List<String> par) {
            Stack<Character> stack = new Stack<Character>();
            List<String> newstack = new LinkedList<String>();
            String reveresed = null;
            char character;

            for (int i = 0; i < par.size(); i++) // Go through the paragraph
            {
                  reveresed = "";
                  for (int o = 0; o < par.get(i).length(); o++) // Store
characters
                  {
                        stack.push(par.get(i).charAt(o));
                  }
                  for(int y = 0; y < par.get(i).length(); y++) // When nothing
left to push into pop out
                  {
                        reveresed += stack.pop();
                  }
                  newstack.add(reveresed);
            }
            return newstack;
      }

      //Word count 6+
      public static int wordcount6(List<String> par)
      {
            String beforepuncuation;
            String[] afterpuncuation;
            int count = 0;

            for(int i = 0; i < par.size(); i++) //Go through the paragraph
lines first
            {
                  beforepuncuation = par.get(i);
                  afterpuncuation = beforepuncuation.replaceAll("[^a-zA-Z ]",
"").toLowerCase().split("\\s+"); //Remove uppercase and leave spaces where
they are
                  //Go through words now
                  for(int o = 0; o < afterpuncuation.length; o++) //Go through
each word
                  {
                        String currentword;
                        currentword = afterpuncuation[o];
                        if(currentword.length() >= 6)
                        {
                              count++;
                        }
                  }
            }

            return count;
      } //End counting 6+ words

      //Longestword + vowel
      public static List<String> longword(List<String> par)
      {
            String beforepuncuation;
            String[] afterpuncuation;
            String currentword;
            String longestVowel = "";
            List<String> finallongest = new LinkedList<String>();

            for(int i = 0; i <= par.size() - 1; i++) //Go through the
paragraph lines first
            {
                  beforepuncuation = par.get(i);
                  afterpuncuation = beforepuncuation.replaceAll("[^a-zA-Z ]",
"").toLowerCase().split("\\s+"); //Remove uppercase and leave spaces where
they are
                  //Go through words now
                  for(int o = 0; o <= afterpuncuation.length -1; o++) //Go
through each word
                  {
                        currentword = afterpuncuation[o];
                        //Check if first letter is a vowel
                        if(currentword.substring(0, 1).equals("b") ||
currentword.substring(0, 1).equals("c") ||  currentword.substring(0,
1).equals("d") ||
                                    currentword.substring(0, 1).equals("f") ||
currentword.substring(0, 1).equals("g") ||  currentword.substring(0,
1).equals("h") ||
                                    currentword.substring(0, 1).equals("j") ||
currentword.substring(0, 1).equals("k") ||  currentword.substring(0,
1).equals("l") ||
                                    currentword.substring(0, 1).equals("m") ||
currentword.substring(0, 1).equals("n") ||  currentword.substring(0,
1).equals("p") ||
                                    currentword.substring(0, 1).equals("q") ||
currentword.substring(0, 1).equals("r") ||  currentword.substring(0,
1).equals("s") ||
                                    currentword.substring(0, 1).equals("t") ||
currentword.substring(0, 1).equals("v") ||  currentword.substring(0,
1).equals("w") ||
                                    currentword.substring(0, 1).equals("x") ||
currentword.substring(0, 1).equals("y") ||  currentword.substring(0,
1).equals("z") )
                        {
                              if(longestVowel.length() < currentword.length())
//Check for length
                              {
                                    longestVowel = currentword;
                              }
                        }
                  }
            }
            int longr = longestVowel.length();
            finallongest.add(0, longestVowel);
            finallongest.add(1, longr + "");

            return finallongest;

      } //End Longest Vowel

      //Vowel freq
      public static List<String> vowelfreq(List<String> par) {

            String beforepuncuation;
            String[] afterpuncuation;
            double a = 0;
            double e = 0;
            double i = 0;
            double o = 0;
            double u = 0;
            int count = 0;
            List<String> freqtable = new LinkedList<String>();

            for(int w = 0; w <= par.size() - 1; w++) //Go through the
paragraph lines first
            {
                  beforepuncuation = par.get(w);
                  afterpuncuation = beforepuncuation.replaceAll("[^a-zA-Z ]",
"").toLowerCase().split("\\s+"); //Remove uppercase and leave spaces where
they are
                  //Go through words now
                  for(int z = 0; z < afterpuncuation.length; z++) //Go through
each word
                  {
                        String currentword;
                        currentword = afterpuncuation[z];
                        //Go through each character and check
                        for(int g = 0; g < currentword.length(); g++)
                        {
                              count ++;
                              int last = g + 1;

                              String checker = currentword.substring(g, last);
                              if (checker.equals("a")) {
                                    a++;
                              }
                              if (checker.equals("e")) {
                                    e++;
                              }
                              if (checker.equals("i")) {
                                    i++;
                              }
                              if (checker.equals("o")) {
                                    o++;
                              }
                              if (checker.equals("u")) {
                                    u++;
                              }
                        }
                  }
            }
            //Table + finding freq of each vowel
            DecimalFormat df = new DecimalFormat("#.##");
            double vowelcount = a + e + i + o + u;
            double afreq = (a/count)*100;
            double efreq = (e/count)*100;
            double ifreq = (i/count)*100;
            double ofreq = (o/count)*100;
            double ufreq = (u/count)*100;
            double totalfreq = afreq + efreq + ifreq + ofreq + ufreq;
            freqtable.add(0, "Vowel | # | Freq.");
            freqtable.add(1, "a |" + a + "|" + df.format(afreq) + "%");
            freqtable.add(2, "e |" + e + "|" + df.format(efreq)+ "%");
            freqtable.add(3, "i |" + i + "|" + df.format(ifreq)+ "%");
            freqtable.add(4, "o |" + o + "|" + df.format(ofreq)+ "%");
            freqtable.add(5, "u |" + u + "|" + df.format(ufreq)+ "%");
            freqtable.add(6, "total |" + vowelcount + "| " +
df.format(totalfreq)+ "%"); //Always has to add to 100
            return freqtable;

      } //End vowel freq

      //Shortest word + noVowel (This is the opposite of finding longest word,
this time it CANNOT start with a vowel)
      public static List<String> shortword(List<String> par)
      {
            String beforepuncuation;
            String[] afterpuncuation;
            String currentword;
            String shortest = "ooooooooooooooo"; //make a long varible so
something has to be shorter than it
            List<String> finalshortest = new LinkedList<String>();

            for(int i = 0; i <= par.size() - 1; i++) //Go through the
paragraph lines first
            {
                  beforepuncuation = par.get(i);
                  afterpuncuation = beforepuncuation.replaceAll("[^a-zA-Z ]",
"").toLowerCase().split("\\s+"); //Remove uppercase and leave spaces where
they are
                  //Go through words now
                  for(int o = 0; o < afterpuncuation.length; o++) //Go through
each word
                  {
                        currentword = afterpuncuation[o];
                        //Ok look I know this is super inefficient but I tried
2 separate solutions and neither fixed it and I was running out of time so you
get this unholy thing
                        if( currentword.substring(0, 1).equals("b") ||
currentword.substring(0, 1).equals("c") ||  currentword.substring(0,
1).equals("d") ||
                                    currentword.substring(0, 1).equals("f") ||
currentword.substring(0, 1).equals("g") ||  currentword.substring(0,
1).equals("h") ||
                                    currentword.substring(0, 1).equals("j") ||
currentword.substring(0, 1).equals("k") ||  currentword.substring(0,
1).equals("l") ||
                                    currentword.substring(0, 1).equals("m") ||
currentword.substring(0, 1).equals("n") ||  currentword.substring(0,
1).equals("p") ||
                                    currentword.substring(0, 1).equals("q") ||
currentword.substring(0, 1).equals("r") ||  currentword.substring(0,
1).equals("s") ||
                                    currentword.substring(0, 1).equals("t") ||
currentword.substring(0, 1).equals("v") ||  currentword.substring(0,
1).equals("w") ||
                                    currentword.substring(0, 1).equals("x") ||
currentword.substring(0, 1).equals("y") ||  currentword.substring(0,
1).equals("z") )
                        {
                              if(currentword.length() < shortest.length())
                              {
                                    shortest = currentword;
                              }
                        }
                  }
            }
            finalshortest.add(0, shortest);
            return finalshortest;

      } //End shortest noVowel

} // end sqserver
```