==========ClientListener.java==========

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Scanner;

import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;

public class ClientListener implements ActionListener {

	public static final String START = "start", STOP = "stop", CONNECT
= "connect", DISCONNECT = "disconnect";

	private JTextField ipAddressField;
	private JTextField portaField;
	private JLabel msgLabel;

	private boolean connected = false, transmitting = false;
	private Downloader downloader = null;

	private PrintWriter netPw;
	private Scanner scan;
	private Socket sock;
	private StatusDownloaderFrame frame;

	public ClientListener(JTextField ipAddr, JTextField porta, JLabel
msgLabel) {
		this.ipAddressField = ipAddr;
		this.portaField = porta;
		this.msgLabel = msgLabel;

	}

	private void setupConnection() throws UnknownHostException,
IOException {
		sock = new Socket(ipAddressField.getText(),
Integer.parseInt(portaField.getText()));
		OutputStream os = sock.getOutputStream();
		netPw = new PrintWriter(new OutputStreamWriter(os));
		scan = new Scanner(sock.getInputStream());
	}

	@Override
	public void actionPerformed(ActionEvent e) {
		if (frame == null)
			frame = (StatusDownloaderFrame)
SwingUtilities.getRoot((JButton) e.getSource());
```

```
            String cmd = e.getActionCommand();

            if (cmd.equals(ClientListener.CONNECT)) {
                  try {
                        setupConnection();
                        connected = true;
                  } catch (IOException e1) {
                        JOptionPane.showMessageDialog(null, "Impossibile
connettersi al server: \n" + e1.getMessage());
                        e1.printStackTrace();
                        return;
                  }

                  JOptionPane.showMessageDialog(null, "Connessione
stabilita");
            } else if (cmd.equals(ClientListener.START)) {
                  try {
                        downloader = new Downloader(msgLabel, scan);
                  } catch (IOException e1) {
                        JOptionPane.showMessageDialog(null, "Impossibile
creare il file: \n" + e1.getMessage());
                        e1.printStackTrace();
                  }
                  transmitting = true;
                  netPw.println(cmd);
                  netPw.flush();

                  Thread t = new Thread(downloader);
                  t.start();
                  JOptionPane.showMessageDialog(null, "Download avviato");
            } else if (cmd.equals(ClientListener.STOP)) {
                  netPw.println(cmd);
                  netPw.flush();
                  transmitting = false;
                  JOptionPane.showMessageDialog(null, "Download fermato");
            } else if (cmd.equals(ClientListener.DISCONNECT)) {
                  netPw.println(ClientListener.DISCONNECT);
                  netPw.flush();
                  netPw.close();
                  scan.close();
                  connected = false;
                  try {
                        sock.close();
                  } catch (IOException e1) {
                        e1.printStackTrace();
                  }

                  JOptionPane.showMessageDialog(null, "Connessione
chiusa");
            }
            frame.setButtons(connected, transmitting);
      }
}
```

==========Downloader.java==========

```
import java.awt.Color;
```

```
import java.awt.Font;
import java.io.IOException;
import java.util.Scanner;

import javax.swing.JLabel;

public class Downloader implements Runnable {

	private Scanner scan;
	private boolean running;
	private JLabel msgLabel;

	public Downloader(JLabel msgLabel, Scanner scan) throws IOException
{

		this.msgLabel = msgLabel;

		this.scan = scan;
		running = false;
	}

	@Override
	public void run() {

		if (!running) {

			running = true;
			while (running) {
				String cmd = scan.nextLine();
				Font f = msgLabel.getFont();
				String[] info = cmd.split(";");
				String color = info[0], s = info[1], text =
info[2];
				Color col = null;
				if (color.equals("3")) {
					col = Color.GREEN;
				} else if (color.equals("2")) {
					col = Color.ORANGE;
				} else if (color.equals("1")) {
					col = Color.RED;
				} else if (color.equals("0")) {
					col = Color.BLACK;
					running = false;
				}
				int size = Integer.parseInt(s);
				msgLabel.setFont(new Font(f.getName(), Font.BOLD,
size));
				msgLabel.setText(text);
				msgLabel.setForeground(col);
			}

		}
	}

	public boolean isRunning() {
		return running;
	}

}
```

==========Main.java==========

```
public class Main {

	public static void main(String[] args) {

		StatusDownloaderFrame frame = new StatusDownloaderFrame();
		frame.setVisible(true);
//		frame.pack();
		frame.setSize(800, 600);
	}
}
```

==========StatusDownloaderFrame.java==========

```
import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.event.ActionListener;

import javax.swing.*;

@SuppressWarnings("serial")
public class StatusDownloaderFrame extends JFrame {

	private JPanel north;
	private JTextField addressText;
	private JTextField portaText;
	private JLabel msgLabel;
	private JPanel addressPanel;
	private JPanel portPanel;
	private JPanel southpanel;
	private JButton connectBtn;
	private JButton disconnectBtn;
	private JButton startBtn;
	private JButton stopBtn;

	public StatusDownloaderFrame() {

		Container mainContainer = this.getContentPane();

		north = new JPanel();

		addressPanel = new JPanel(new FlowLayout());
		portPanel = new JPanel(new FlowLayout());

		msgLabel = new JLabel("STOPPED");
		msgLabel.setHorizontalAlignment(SwingConstants.CENTER);
		Font f =  msgLabel.getFont();
		msgLabel.setFont(new Font(f.getName(), Font.BOLD, 30));
```

```
            addressPanel.add(new JLabel("IP Address"),
BorderLayout.CENTER);
            addressText = new JTextField(10);
            addressText.setText("127.0.0.1");
            addressPanel.add(addressText, BorderLayout.SOUTH);

            portPanel.add(new JLabel("Port"), BorderLayout.CENTER);
            portaText = new JTextField(10);
            portaText.setText("4400");
            portPanel.add(new JPanel().add(portaText),
BorderLayout.SOUTH);

            southpanel = new JPanel();

            ActionListener list = new ClientListener(addressText,
portaText, msgLabel);

            connectBtn = new JButton("Connect");
            connectBtn.setActionCommand(ClientListener.CONNECT);
            connectBtn.addActionListener(list);
            disconnectBtn = new JButton("Disconnect");
            disconnectBtn.setActionCommand(ClientListener.DISCONNECT);
            disconnectBtn.addActionListener(list);
            startBtn = new JButton("Start");
            startBtn.setActionCommand(ClientListener.START);
            startBtn.addActionListener(list);
            stopBtn = new JButton("Stop ");
            stopBtn.setActionCommand(ClientListener.STOP);
            stopBtn.addActionListener(list);

            southpanel.add(connectBtn);
            southpanel.add(disconnectBtn);
            north.add(startBtn);
            north.add(addressPanel);
            north.add(portPanel);

            north.add(stopBtn);

            mainContainer.add(north, BorderLayout.NORTH);
            mainContainer.add(southpanel, BorderLayout.SOUTH);
            mainContainer.add(msgLabel, BorderLayout.CENTER);

            setLocation(200, 100);
            setDefaultCloseOperation(EXIT_ON_CLOSE);

            setButtons(false, false);

            setTitle("Nome Cognome 1234567");

            this.setVisible(true);
      }

      public void setButtons(boolean connected, boolean transmitting) {
            if(connected){
                  connectBtn.setEnabled(false);
                  setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);

                  if(transmitting){
                        disconnectBtn.setEnabled(false);
                        stopBtn.setEnabled(true);
```

```
                        startBtn.setEnabled(false);
                  }else{
                        stopBtn.setEnabled(false);
                        startBtn.setEnabled(true);
                        disconnectBtn.setEnabled(true);
                  }

            }else{

                  setDefaultCloseOperation(EXIT_ON_CLOSE);
                  connectBtn.setEnabled(true);
                  disconnectBtn.setEnabled(false);
                  startBtn.setEnabled(false);
                  stopBtn.setEnabled(false);
            }

      }
}
```

==========ClientThread.java==========

```
import java.io.*;
import java.net.*;
import java.util.Scanner;

public class ClientThread implements Runnable {

      private Socket sock;
      private boolean fired = false;
      private SenderThread st = null;

      Server parent;

      public ClientThread(Socket s, Server parent) {
            sock = s;

            this.parent = parent;
      }

      @Override
      public void run() {
            if (fired)
                  return;
            fired = true;
            boolean running = true;
            Scanner in = null;
            PrintWriter pw = null;
            try {
                  in = new Scanner(sock.getInputStream());
                  pw = new PrintWriter(sock.getOutputStream());
            } catch (IOException e) {
                  e.printStackTrace();
            }

            while (running) {
                  String cmd = in.nextLine();
```

```
                System.out.println("Ricevuto: "+ cmd);
                if (cmd.equals("start")) {

                      //Avvio nuovo thread per invio di 01
                      st = new SenderThread(pw);
                      Thread t = new Thread(st);
                      t.start();
                } else if (cmd.equals("stop")) {
                      st.stop();
                } else {
                      running = false;
                }
          }
          try {
                pw.close();
                in.close();
                sock.close();
          } catch (IOException e1) {
                e1.printStackTrace();
          }
     }
}
```

==========Main.java==========

```
public class Main {
     public static void main(String[] args) {

          Server serv = new Server();
          Thread avv = new Thread(serv);
          avv.start();
     }
}
```

==========SenderThread.java==========

```
import java.io.PrintWriter;

public class SenderThread implements Runnable {

     private PrintWriter pw;
     private boolean flag;
     private String[] msgs0 = { "OK", "Running", "Funzionante", "Online"
};
     private String[] msgs1 = { "Warning", "Manutenzione consigliata",
"Attenzione", "Sovraccarico" };

     private String[] msgs2 = { "Error", "Manutenzione necessaria",
"Irraggiungibile", "Out of order" };

     public SenderThread(PrintWriter pw) {
```

```
            flag = false;
            this.pw = pw;
      }

      @Override
      public void run() {
            flag = true;
            while (flag) {
                  String toSend = "";
                  double status = Math.random(), severity = 0, msg =
Math.random() * msgs0.length;

                  while(severity < 0.4){
                        severity = Math.random();
                  }

                  if (status < 0.33) {
                        toSend += "3;";
                        toSend += (int)Math.ceil(severity * 50) + ";";
                        toSend += msgs0[(int) msg];
                  } else if (status < 0.66) {
                        toSend += "2;";
                        toSend += (int)Math.ceil(severity * 50) + ";";
                        toSend += msgs1[(int) msg];
                  } else {
                        toSend += "1;";
                        toSend += (int)Math.ceil(severity * 50) + ";";
                        toSend += msgs2[(int) msg];
                  }

                  pw.println(toSend);
                  pw.flush();

                  try {
                        Thread.sleep(3000);
                  } catch (InterruptedException e) {
                        e.printStackTrace();
                  }
            }

      }

      public void stop() {
            // chiusura del pw delegata al chiamante
            flag = false;

            pw.println("0;12;STOP");
            pw.flush();
      }

}
```

==========Server.java==========

```
import java.io.IOException;
import java.net.*;

import javax.swing.JOptionPane;

public class Server implements Runnable {

	private ServerSocket lis = null;

	public void run() {

		try {
			// attendo nuove connessioni
			lis = new ServerSocket(4400);
		} catch (IOException e1) {
			e1.printStackTrace();
			JOptionPane.showMessageDialog(null, "Errore nella
creazione del ServerSocket, applicazione dismessa", null,
					0);
			System.exit(1);
		}
		System.out.println("Server Avviato");
		Socket sock = null;

		while (true) {
			try {
				// accetto nuove connessioni
				sock = lis.accept();
			} catch (IOException e) {
				break;
			}
			System.out.println("Socket creata, connessione
accettata");
			ClientThread cl = new ClientThread(sock, this);
			Thread tr = new Thread(cl);
			tr.start();
		}
	}
}
```