```
==============CLIENT============================
===============================================
===============================================



================BinaryDownloaderFrame===========



import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.ActionListener;

import javax.swing.*;

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

	private JPanel centralPanel;
	private JTextField addressText;
	private JTextField portaText;
	private JTextField matricolaText;
	private JPanel central11;
	private JPanel central12;
	private JPanel central21;
	private JPanel central22;
	private JPanel southpanel;
	private JButton connectBtn;
	private JButton disconnectBtn;
	private JButton startBtn;
	private JButton stopBtn;

	public BinaryDownloaderFrame() {

		Container mainContainer = this.getContentPane();

		centralPanel = new JPanel(new GridLayout(2, 2));

		central11 = new JPanel(new BorderLayout());
		central12 = new JPanel(new BorderLayout());
		central21 = new JPanel(new BorderLayout());
		central22 = new JPanel(new BorderLayout());

		central11.add(new JLabel("Matricola"), BorderLayout.NORTH);
		matricolaText = new JTextField(20);
		central11.add(matricolaText, BorderLayout.SOUTH);

		central12.add(new JLabel("IP Address"), BorderLayout.NORTH);
		addressText = new JTextField(20);
		central12.add(addressText, BorderLayout.SOUTH);

		central21.add(new JLabel("Porta"), BorderLayout.NORTH);
		portaText = new JTextField(20);
		central21.add(new JPanel().add(portaText), 
BorderLayout.SOUTH);

		centralPanel.add(central11);
		centralPanel.add(central12);
```

```
            centralPanel.add(central21);
            centralPanel.add(central22);

            southpanel = new JPanel();

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

            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);
            southpanel.add(startBtn);
            southpanel.add(stopBtn);

            mainContainer.add(southpanel, BorderLayout.SOUTH);
            mainContainer.add(centralPanel, BorderLayout.CENTER);

            setLocation(200, 100);

            setDefaultCloseOperation(EXIT_ON_CLOSE);

            setButtons(false, false);

            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);
            }

      }
}



=================ClientListener=======================



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.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 JTextField matricolaField;

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

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

      public ClientListener(JTextField ipAddr, JTextField porta,
JTextField matricola) {
            this.ipAddressField = ipAddr;
            this.portaField = porta;
            this.matricolaField = matricola;

      }

      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 = (BinaryDownloaderFrame)
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(matricolaField.getText(), 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===============================


import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Scanner;

public class Downloader implements Runnable {

	private PrintWriter filePw;
	private Scanner scan;
	private boolean running;
	private String rec;

	public Downloader(String fileName, Scanner scan) throws IOException
{

		rec = "";
		File file = new File(fileName+".txt");

		filePw = new PrintWriter(new FileWriter(file));
		this.scan = scan;
		running = false;
	}

	@Override
	public void run() {

		if (!running) {

			running = true;
			while (running) {
				String cmd = scan.nextLine();
				if (cmd.equals("*")) {
					filePw.println(rec);
					filePw.flush();
					String code = scan.nextLine();
					filePw.println(code);
					filePw.flush();
					filePw.close();

					code = scan.nextLine();

					running = !code.equals(ClientListener.STOP);
				} else {
					rec += cmd;
				}
			}

		}
	}

	public boolean isRunning(){
		return running;
	}
```

```
}



==================Main============================



public class Main {

      public static void main(String[] args) {

            BinaryDownloaderFrame frame = new BinaryDownloaderFrame();
            frame.setVisible(true);
            frame.pack();
      }
}



===================SERVER========================================
==============================================================
==============================================================



========================ClientThread========================


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;
      private Scanner in = null;
      private PrintWriter pw = 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;

            try {
                  in = new Scanner(sock.getInputStream());
                  pw = new PrintWriter(sock.getOutputStream());
```

```
		} catch (IOException e) {
			e.printStackTrace();
		}

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

	public boolean isClosed() {

		return sock.isClosed();
	}

	public void close() throws IOException {
		pw.close();
		in.close();
		sock.close();
	}
}


==================GUIMain====================


import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;

@SuppressWarnings("serial")
public class GUIMain extends JFrame implements ActionListener {

	private JPanel centralPanel;
	private JPanel topPanel;
	private JButton startBtn;
	private JButton stopBtn;
	private JTextField portTxt;
	private JLabel portLbl;

	private Server serv;

	public GUIMain() {

		// Widgets
		startBtn = new JButton("Start");
		stopBtn = new JButton("Stop");

		portTxt = new JTextField(10);
		portLbl = new JLabel("Port");

		// Layout
		topPanel = new JPanel();
		topPanel.add(portLbl);
		topPanel.add(portTxt);

		centralPanel = new JPanel();
		centralPanel.add(startBtn);
		centralPanel.add(stopBtn);

		Container mainCont = this.getContentPane();
		mainCont.add(centralPanel, BorderLayout.CENTER);
		mainCont.add(topPanel, BorderLayout.NORTH);

		// Listeners
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		startBtn.setActionCommand("start");
		startBtn.addActionListener(this);
		stopBtn.setActionCommand("stop");
		stopBtn.addActionListener(this);

		// Initial state
		stopBtn.setEnabled(false);

		setSize(220,120);
	}

	public static void main(String[] args) {
		GUIMain gui = new GUIMain();
		gui.setVisible(true);
	}

	@Override
	public void actionPerformed(ActionEvent e) {
		String cmd = e.getActionCommand();
		if (cmd.equals("start")) {
			try {
				serv = new Server(portTxt.getText());
			} catch (NumberFormatException ex) {
				JOptionPane.showMessageDialog(this, "Formato porta
errato", "Errore", JOptionPane.ERROR_MESSAGE);
				return;
			}
			Thread avv = new Thread(serv);
			avv.start();
```

```
                startBtn.setEnabled(false);
                stopBtn.setEnabled(true);
                setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
            }
            else if(cmd.equals("stop")){
                serv.stop();
                startBtn.setEnabled(true);
                stopBtn.setEnabled(false);
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            }
        }
}
```

```
====================SenderThread=============================
```

```
import java.io.PrintWriter;

public class SenderThread implements Runnable {

        private PrintWriter pw;
        private boolean flag;
        private String sent = "";

        public SenderThread(PrintWriter pw) {
                flag = false;
                this.pw = pw;
        }

        // thread Sender
        public void run() {
                flag = true;
                while (flag) {
                        String toSend = "";
                        double check = Math.random();
                        if (check > 0.5) {
                                toSend = "1";
                        } else {
                                toSend = "0";
                        }
                        pw.println(toSend);
                        pw.flush();
                        sent += toSend;
                        try {
                                Thread.sleep(10);
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }

        }

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

```
			pw.println("0");
			pw.flush();
			pw.println("1");
			pw.flush();
			pw.println("0");
			pw.flush();
			pw.println("*");
			pw.flush();

			sent+="010";

			String hashcode = Integer.toString(sent.hashCode());
			pw.println(hashcode);
			pw.flush();

			pw.println("stop");
			pw.flush();
		}
}




====================Server================================


import java.io.IOException;
import java.net.*;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

import javax.swing.JOptionPane;

public class Server implements Runnable {

	private ServerSocket lis = null;
	private int port;
	private boolean running;
	private Set<ClientThread> clients = null;

	public Server(String text) throws NumberFormatException {
		port = Integer.parseInt(text);
		running = false;
		clients = new HashSet<ClientThread>();
	}

	public void run() {

		try {
			// attendo nuove connessioni
			lis = new ServerSocket(port);
		} 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;
            running = true;

            while (running) {
                  try {
                        // accetto nuove connessioni
                        sock = lis.accept();
                  } catch (IOException e) {
                        break;
                  }
                  ClientThread cl = new ClientThread(sock, this);
                  Thread tr = new Thread(cl);
                  clients.add(cl);
                  tr.start();
            }
      }

      public void stop() {
            if (running) {
                  running = false;
                  try {
                        lis.close();
                  } catch (IOException e) {
                        return;
                  }
                  if (!chiudiSockets())
                        System.exit(1);
            }
      }

      private boolean chiudiSockets() {

            Iterator<ClientThread> t = clients.iterator();
            while (t.hasNext()) {
                  ClientThread clientThread = t.next();
                  if (!clientThread.isClosed()) {
                        try {
                              clientThread.close();
                        } catch (IOException e) {
                              e.printStackTrace();
                              return false;
                        }
                  }
            }
            clients = new HashSet<ClientThread>();
            return true;
      }
}
```