==========CLIENT==========
==========
==========

==========InvioEventListener==========

```
import java.awt.Window;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowEvent;
import java.io.IOException;
import java.io.ObjectOutputStream;

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

public class InvioEventListener implements ActionListener {
	/*
	 * Alla pressione del pulsante "Invio" uso lo stream di output
della socket
	 * per inviare il contenuto del text field
	 */
	private JTextField textField;
	private ObjectOutputStream oos;

	public InvioEventListener(JTextField textField, ObjectOutputStream
oos) {
		this.textField = textField;
		this.oos = oos;
	}

	@Override
	public void actionPerformed(ActionEvent e) {
		String text = textField.getText();
		if (!text.equals("")) {
			try {
				oos.writeObject(text);
				oos.flush();
			} catch (IOException e1) {

				e1.printStackTrace();
				JOptionPane.showMessageDialog(null, "Connessione
con il server persa, applicazione dismessa");
				// simulazione dell'operazione di click sulla
chiusura del frame
				// principale, chiudo il programma in maniera
controllata
				Window frame =
SwingUtilities.getWindowAncestor(textField);
				frame.dispatchEvent(new WindowEvent(frame,
WindowEvent.WINDOW_CLOSING));
			}

			textField.setText("");
		}
	}
```

```
}
```

==========MAin==========

```
public class Main {

	public static void main(String[] args) {

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


}
```

==========MainWindListener==========

```
import java.awt.event.*;
import java.io.*;
import java.net.*;

public class MainWindListener implements WindowListener {
	/*
	 * Esempio di implementazione di listner di finestra custom, in
questo caso
	 * alla chiusura della finestra prima di interrompere il programma
si avvisa
	 * il server della chiusura imminente e si chiudono socket e
streams e si
	 * interrompe il thread di polling
	 */
	private Socket sock;
	private ObjectOutputStream oos;
	private ObjectInputStream ois;

	public MainWindListener(Socket s, ObjectOutputStream oos,
ObjectInputStream ois) {
		this.sock = s;
		this.ois = ois;
		this.oos = oos;
	}

	@Override
	public void windowActivated(WindowEvent arg0) {

	}

	@Override
	public void windowClosed(WindowEvent arg0) {

	}
```

```
	@Override
	public void windowClosing(WindowEvent arg0) {

		try {
			oos.writeObject(new Integer(1));
			oos.flush();
			oos.close();
			ois.close();
			sock.close();
		} catch (IOException e) {
			System.exit(1);
		}
		System.exit(0);

	}

	@Override
	public void windowDeactivated(WindowEvent arg0) {

	}

	@Override
	public void windowDeiconified(WindowEvent arg0) {

	}

	@Override
	public void windowIconified(WindowEvent arg0) {

	}

	@Override
	public void windowOpened(WindowEvent arg0) {

	}
}
```

==========RemoteLoggingWindow==========

```
import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.event.ActionListener;
import java.awt.event.WindowListener;
import java.io.IOException;
import java.io.InputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.OutputStream;
import java.net.Socket;
import java.net.UnknownHostException;

import javax.swing.*;

@SuppressWarnings("serial")
```

```
public class RemoteLoggingWindow extends JFrame{

	private ObjectOutputStream oos;
	private ObjectInputStream ois;
	private Socket sock = null;

	private JPanel southPanel;
	private JButton invio;
	private JTextField textMessage;

	private String name;


	public RemoteLoggingWindow(){

		name = JOptionPane.showInputDialog("Username:");
		if(name == null){
			JOptionPane.showMessageDialog(null, "Necessario indicare
un username.");
			System.exit(0);
		}
		try {
			setupConnection();

			oos.writeObject(name);
			oos.flush();

		} catch (IOException e) {
			JOptionPane.showMessageDialog(null, "Impossibile
stabilire una connessione con il server");
			System.exit(1);
		}

		this.setTitle(name);
		Container mainContainer = this.getContentPane();


		southPanel = new JPanel();

		textMessage = new JTextField(50);
		invio = new JButton("Invia");

		southPanel.add(textMessage);
		southPanel.add(invio);

		mainContainer.add(southPanel, BorderLayout.CENTER);

		setLocation(200,100);

		ActionListener list= new InvioEventListener(textMessage, oos);
		invio.addActionListener(list);

		setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
		WindowListener wl = new MainWindListener(sock, oos, ois);
		this.addWindowListener(wl);

		this.setVisible(true);


	}
```

```
      private void setupConnection() throws UnknownHostException,
IOException {
            sock= new Socket("127.0.0.1",3000);
            InputStream in = sock.getInputStream();
            OutputStream os = sock.getOutputStream();
            oos = new ObjectOutputStream(os);
            ois = new ObjectInputStream(in);
      }
}
```

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

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

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

public class ClientThread implements Runnable {

      private Socket sock;
      private boolean fired = false, running = true;
      private ObjectOutputStream oos;
      private ObjectInputStream ois;

      private PrintWriter pw; //per scrittura su file
      private String nome;

      public ClientThread(Socket s) {
            sock = s;
            nome = null;
            try {

                  oos = new ObjectOutputStream(sock.getOutputStream());
                  ois = new ObjectInputStream(sock.getInputStream());

            } catch (IOException e) {
                  fired = true;
                  try {
                        ois.close();
                        oos.close();
                        sock.close();
                  } catch (IOException e1) {
                        e1.printStackTrace();
                        fired = true;
                  }
            }
      }

      @Override
      public void run() {
            running = true;
```

```
            if (fired)
                  return;
            fired = true;

            while (running) {
                  try {
                        Object o = ois.readObject();
                        if (Integer.class.isInstance(o)) {
                              running = false;

                        } else if (String.class.isInstance(o)) {
                              String s = (String)o;
                              if(nome == null){
                                    nome = s;
                                    File f = new File(s+".txt");
                                    if(!f.exists())
                                          f.createNewFile();
                                    pw = new PrintWriter(new
FileOutputStream(f,true));

                              }else{
                                    pw.println(s);
                                    pw.flush();
                                    System.out.println(s);
                              }
                        }
                  } catch (IOException e) {
                        running = false;
                  } catch (ClassNotFoundException e) {
                        running = false;
                  }
            }
            pw.close();
            try {
                  ois.close();
                  oos.close();
                  sock.close();
            } catch (IOException e1) {
                  e1.printStackTrace();
            }
      }

      public boolean isClosed() {

            return sock.isClosed();
      }

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

}



========================Main=======================
```

```
public class Main {
	@SuppressWarnings("unused")
	public static void main(String[] args) {

		ServerInterface serv = new ServerInterface();

	}

}
```

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

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

import javax.swing.JOptionPane;

public class Server implements Runnable{

	private ServerSocket lis = null;
	private boolean flag = false;
	private List<ClientThread> l = null;

	public void run(){

		if(!flag){
			flag = true;
			l=new LinkedList<ClientThread>();
			try {
				//attendo nuove connessioni
				lis = new ServerSocket(3000);
			} 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;
				}
				//Avvio il thread che gestisce la comunicazione
effettiva con il client
				ClientThread cl = new ClientThread(sock);
				Thread tr = new Thread(cl);
				tr.start();
```

```
                        l.add(cl);
                  }
            }
      }

      public void ferma(){
            if(flag){
                  flag=false;
                  try {
                        lis.close();
                  } catch (IOException e) {
                        e.printStackTrace();
                        System.exit(1);
                  }
                  if(!chiudiConnessioni()) System.exit(1);
            }
      }

      private boolean chiudiConnessioni(){

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

==========ServerInterface==========

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.*;


public class ServerInterface implements ActionListener{

      private JFrame frame;
      private JPanel pan;
      private JButton but_A;
      private JButton but_S;
      private Server serv;
```

```
	public ServerInterface(){

		frame = new JFrame("BroadcastServerServer");
		pan = new JPanel(new FlowLayout());
		but_A = new JButton("Avvia");
		but_S = new JButton("Stop");
		pan.add(but_A);
		pan.add(but_S);
		frame.add(pan);
		frame.setVisible(true);
		frame.setSize(220,100);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		but_A.addActionListener(this);
		but_S.addActionListener(this);
		but_S.setEnabled(false);
		serv = new Server();
	}

	@Override
	public void actionPerformed(ActionEvent e) {

		String x = e.getActionCommand();
		if(x.equals("Avvia")){

			but_A.setEnabled(false);
			Thread avv = new Thread(serv);
			avv.start();

	frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
			but_S.setEnabled(true);
		}
		else if(x.equals("Stop")){
			but_S.setEnabled(false);
			serv.ferma();
			frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
			but_A.setEnabled(true);
		}
		else System.exit(1);
	}
}
```