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



====================BroadcastChatWindow==================


package gui;

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.*;

import listeners.InvioEventListener;
import listeners.MainWindListener;
import threads.Polling;

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

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

	private JScrollPane centralPanel;
	private JPanel southPanel;
	private JTextArea messagesArea;
	private JButton invio;
	private JTextField textMessage;


	public BroadcastChatWindow(){

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

		this.setTitle("Messagistica");
		Container mainContainer = this.getContentPane();


		southPanel = new JPanel();

		messagesArea = new JTextArea(25,50);
```

```
            messagesArea.setEditable(false);

            centralPanel = new 
JScrollPane(messagesArea,JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,JScroll
Pane.HORIZONTAL_SCROLLBAR_AS_NEEDED);

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

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

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

            setLocation(200,100);

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

            Polling p = new Polling(messagesArea, oos, ois);
            Thread t = new Thread(p);
            t.start();

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



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


package listeners;

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 {

	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");
				Window frame =
SwingUtilities.getWindowAncestor(textField);
				frame.dispatchEvent(new WindowEvent(frame,
WindowEvent.WINDOW_CLOSING));
			}

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

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

```
package listeners;

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

import threads.Polling;

public class MainWindListener implements WindowListener{

	private Socket sock;
	private ObjectOutputStream oos;
	private ObjectInputStream ois;
	private Polling p;


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

```
        this.ois=ois;
        this.oos=oos;
        this.p=p;
    }


    @Override
    public void windowActivated(WindowEvent arg0) {

    }

    @Override
    public void windowClosed(WindowEvent arg0) {

    }

    @Override
    public void windowClosing(WindowEvent arg0) {

        p.stop();

        try{
            oos.writeObject(new Integer(1));
            oos.flush();
            if(oos!=null){
                oos.close();
            }
        }catch(IOException e){
        }
        try{
            if(ois!=null){
                ois.close();
            }
        }catch(IOException e){
        }

        try{
            if(sock!=null){
                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) {

      }
}
```

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

```
package main;


import gui.BroadcastChatWindow;

public class Main {

      public static void main(String[] args) {

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


}
```

==================Polling================================

```
package threads;

import java.awt.Window;
import java.awt.event.WindowEvent;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.Iterator;
import java.util.List;

import javax.swing.JOptionPane;
import javax.swing.JTextArea;
import javax.swing.SwingUtilities;


public class Polling implements Runnable {

      private JTextArea textArea;
      private ObjectOutputStream oos;
      private ObjectInputStream ois;
      private boolean running;
```

```
	public Polling(JTextArea textArea, ObjectOutputStream oos,
ObjectInputStream ois){
		this.textArea = textArea;
		this.oos = oos;
		this.ois=ois;
		running = false;
	}

	@SuppressWarnings("unchecked")
	@Override
	public void run() {
		running = true;
		while(running){
			Object o = null;
			try {
				oos.writeObject(new Integer(0));
				oos.flush();
				o = ois.readObject();

			} catch (IOException e) {

				JOptionPane.showMessageDialog(null, "Connessione
con il server persa, applicazione dismessa");
				Window frame =
SwingUtilities.getWindowAncestor(textArea);
				frame.dispatchEvent(new WindowEvent(frame,
WindowEvent.WINDOW_CLOSING));

			}
			catch (ClassNotFoundException e) {
				e.printStackTrace();
				JOptionPane.showMessageDialog(null, "Connessione
con il server persa, applicazione dismessa");
				Window frame =
SwingUtilities.getWindowAncestor(textArea);
				frame.dispatchEvent(new WindowEvent(frame,
WindowEvent.WINDOW_CLOSING));
			}
			List<String> l = null;
			if(List.class.isInstance(o)){

				l = (List<String>)o;
			}
			String text = textArea.getText();
			Iterator<String> it = l.iterator();
			while(it.hasNext()){
				text += it.next()+"\n\n";
			}
			textArea.setText(text);

			try {
				Thread.sleep(2000);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}
	}

	public void stop(){
		running = false;
```

```
          }

     }
```

```
====================SERVER===============================
=========================================================
=========================================================



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



package gui;

import java.awt.*;
import javax.swing.*;
import listners.ButtonListner;


public class ServerInterface {

	private JFrame frame;
	private JPanel pan;
	private JButton but_A;
	private JButton but_S;
	ButtonListner l;

	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);
		ButtonListner l=new ButtonListner(but_A,but_S,frame);
		but_A.addActionListener(l);
		but_S.addActionListener(l);
	}
}



===============ButtonListener======================



package listners;

import java.awt.event.*;
import javax.swing.*;
import thread.Server;

public class ButtonListner implements ActionListener{

	private JButton a;
	private JButton s;
	private Server serv;
	private JFrame fr;

	public ButtonListner(JButton bott, JButton bot2,JFrame frame){
		a=bott;
		s=bot2;
		s.setEnabled(false);
		serv = new Server();
		fr=frame;

	}
	@Override
	public void actionPerformed(ActionEvent e) {

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

			a.setEnabled(false);
			Thread avv = new Thread(serv);
			avv.start();
			fr.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
			s.setEnabled(true);
		}
		else if(x.equals("Stop")){
			s.setEnabled(false);
			serv.ferma();
			fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
			a.setEnabled(true);
		}
		else System.exit(1);
	}
}


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


package main;


import gui.*;

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

		ServerInterface serv = new ServerInterface();

	}

}



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



package thread;

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

public class ClientThread implements Runnable {

	private Server mainThread;
	private Socket sock;
	private boolean fired = false, running = true;
	private ObjectOutputStream oos;
	private ObjectInputStream ois;
	private volatile List<String> privateMessages;

	public ClientThread(Socket s, Server mainThread) {
		sock = s;
		privateMessages = new LinkedList<String>();
		this.mainThread = mainThread;
		try {
			InputStream is;
			OutputStream os;
			is = sock.getInputStream();
			os = sock.getOutputStream();
			oos = new ObjectOutputStream(os);
			ois = new ObjectInputStream(is);
		} catch (IOException e) {
			fired = true;
			try {
				if (ois != null) {
					ois.close();
				}
			} catch (IOException e1) {
				e1.printStackTrace();
			}
			try {
				if (oos != null) {
					oos.close();
				}
			} catch (IOException e1) {
				e1.printStackTrace();
			}
			try {
				if (sock != null) {
					sock.close();
				}
			} catch (IOException e1) {
				e1.printStackTrace();
			}
		}
	}

	@Override
	public void run() {
		running = true;
		if (fired)
			return;
		fired = true;

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

```
                        Integer cmd = (Integer) o;
                        System.out.println(cmd);
                        if (cmd == 0) // polling
                              inviaMessaggi();
                        else // chiusura client
                              running = false;
                  } else if (String.class.isInstance(o)) {
                        String msg = (String) o;
                        System.out.println(msg);
                        mainThread.newMessage(msg);
                  }
            } catch (IOException e) {
                  running = false;
            } catch (ClassNotFoundException e) {
                  running = false;
            }
      }
      try {
            if (ois != null) {
                  ois.close();
            }
      } catch (IOException e1) {
            e1.printStackTrace();
      }
      try {
            if (oos != null) {
                  oos.close();
            }
      } catch (IOException e1) {
            e1.printStackTrace();
      }
      try {
            if (sock != null) {
                  sock.close();
            }
      } catch (IOException e1) {
            e1.printStackTrace();
      }
}

private void inviaMessaggi() {
      try {
            sendObject(privateMessages);
      } catch (IOException e) {
      }
      privateMessages = new LinkedList<String>();
}

private void sendObject(Object o) throws IOException {

      oos.writeObject(o);
      oos.flush();
}

public boolean isClosed() {

      return sock.isClosed();
}

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

		public void sendMsg(String msg) {
			privateMessages.add(msg);
		}
}
```

```
==================Server=====================
```

```
package thread;

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 {
					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{
						sock = lis.accept();
					} catch (IOException e) {
						break;
					}

					ClientThread cl = new ClientThread(sock, this);
					Thread tr = new Thread(cl);
					tr.start();
					l.add(cl);
					control();
				}
			}
		}
```

```
	private void control(){
		List<ClientThread> del = new LinkedList<ClientThread>();
		Iterator<ClientThread> it = l.iterator();
		while(it.hasNext()){
			ClientThread sock = it.next();
			if(sock.isClosed()){
				del.add(sock);
			}
		}
		l.removeAll(del);
	}

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

	private boolean chiudiSockets(){

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

	public synchronized boolean newMessage(String msg){
		Iterator<ClientThread> it = l.iterator();
		while(it.hasNext()){
			ClientThread client = it.next();
			client.sendMsg(msg);
		}
		return true;
	}
}
```