PACKAGE parole

CLASSI Display, ClientListener, ReceiverThread, Principale

//DISPLAY.JAVA

package parole;

import java.awt.BorderLayout;

import java.awt.event.ActionListener;

import javax.swing.\*;

public class Display extends JFrame{

//CAMPI

private JLabel addresslabel = new JLabel("Server Address");

JTextField addressfield = new JTextField(20);

private JLabel portlabel = new JLabel("Port");

JTextField portfield = new JTextField(6);

private JButton connect = new JButton("Connect");

private JButton disconnect = new JButton("Disconnect");

JTextArea ai = new JTextArea(10, 15);

JTextArea jr = new JTextArea(10, 15);

JTextArea sz = new JTextArea(10, 15);

private JScrollPane scrollai;

private JScrollPane scrolljr;

private JScrollPane scrollsz;

private JLabel requestedstringlabel = new JLabel("Inserisci la stringa");

JTextField requestedstringfield = new JTextField(10);

private JButton get = new JButton("Get");

private JButton stop = new JButton("Stop");

public boolean connected = false;

public boolean started = false;

//STARTED=TRUE VUOL DIRE CHE è STATO PREMUTO GET

//COSTRUTTORE

public Display() {

super("Sofia Lapi 1837563");

JPanel north = new JPanel();

north.add(addresslabel);

north.add(addressfield);

north.add(portlabel);

north.add(portfield);

north.add(connect);

north.add(disconnect);

ai.setEditable(false);

jr.setEditable(false);

sz.setEditable(false);

scrollai = new JScrollPane(ai);

scrolljr = new JScrollPane(jr);

scrollsz = new JScrollPane(sz);

scrollai.setBorder(BorderFactory.createTitledBorder("A-I"));

scrolljr.setBorder(BorderFactory.createTitledBorder("J-R"));

scrollsz.setBorder(BorderFactory.createTitledBorder("S-Z"));

JPanel center = new JPanel();

center.add(scrollai);

center.add(scrolljr);

center.add(scrollsz);

JPanel south = new JPanel();

south.add(requestedstringlabel);

south.add(requestedstringfield);

south.add(get);

south.add(stop);

this.getContentPane().add(north, BorderLayout.NORTH);

this.getContentPane().add(center, BorderLayout.CENTER);

this.getContentPane().add(south, BorderLayout.SOUTH);

//ASCOLTATORI ED EVENTI

ActionListener ascoltatore = new ClientListener(this);

connect.addActionListener(ascoltatore);

connect.setActionCommand(ClientListener.CONNECT);

disconnect.addActionListener(ascoltatore);

disconnect.setActionCommand(ClientListener.DISCONNECT);

get.addActionListener(ascoltatore);

get.setActionCommand(ClientListener.GET);

stop.addActionListener(ascoltatore);

stop.setActionCommand(ClientListener.STOP);

this.setButtons();

this.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

this.pack();

this.setVisible(true);

}

//METODO GESTIONE PULSANTI

public void setButtons() {

if (!connected) {

connect.setEnabled(true);

disconnect.setEnabled(false);

get.setEnabled(false);

stop.setEnabled(false);

}else {

connect.setEnabled(false);

disconnect.setEnabled(true);

get.setEnabled(true);

stop.setEnabled(false);

if (started) {

get.setEnabled(false);

stop.setEnabled(true);

}else {

stop.setEnabled(false);

}

}

}

}

//CLIENTLISTENER.JAVA

package parole;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.IOException;

import java.io.OutputStreamWriter;

import java.io.PrintWriter;

import java.net.Socket;

import java.net.UnknownHostException;

import java.util.Scanner;

import javax.swing.JOptionPane;

public class ClientListener implements ActionListener {

//CAMPI PER EVOLUZIONE STATO

public final static String CONNECT = "Connect";

public final static String DISCONNECT = "Disconnect";

public final static String GET = "Get";

public final static String STOP = "Stop";

//CAMPO DISPLAY

private Display displayAscoltato = null;

//CAMPI PER CONNESSIONE SERVER

private Socket socket = null;

private PrintWriter pw = null;

private Scanner sc = null;

//COSTRUTTORE

public ClientListener(Display display) {

this.displayAscoltato = display;

}

//ACTION PERFORMED

@Override

public void actionPerformed(ActionEvent e) {

if (e.getActionCommand().equals(CONNECT)) {

//PARTE DI CONNESSIONE CON SERVER

try {

this.setupConnection();

} catch (IOException e1) {

JOptionPane.showMessageDialog(null, "impossibile connettersi");

}

//PARTE EVOLUZIONE GRAFICA

this.displayAscoltato.connected=true;

this.displayAscoltato.setButtons();

}else if(e.getActionCommand().equals(DISCONNECT)) {

//PARTE DI CONNESSIONE CON SERVER

try {

pw.println("disconnect");

pw.flush();

this.closeConnection();

} catch (IOException e1) {

JOptionPane.showMessageDialog(null, "impossibile disconnettersi");

}

//PARTE EVOLUZIONE GRAFICA

this.displayAscoltato.connected=false;

this.displayAscoltato.setButtons();

}else if(e.getActionCommand().equals(GET)) {

//PARTE EVOLUZIONE GRAFICA

this.displayAscoltato.started=true;

this.displayAscoltato.setButtons();

//CONTROLLO STRINGA RICHIESTA NON VUOTA

if(controlloStringaVuota()) {

JOptionPane.showMessageDialog(null, "la stringa richiesta non può essere vuota");

//ATTENTA A RISETTARE I PULSANTI

this.displayAscoltato.connected=true;

this.displayAscoltato.started=false;

this.displayAscoltato.setButtons();

}else {

//ATTENTA A CANCELLARE IL CONTENUTO DELLE AREE DI TESTO

this.displayAscoltato.ai.setText("");

this.displayAscoltato.jr.setText("");

this.displayAscoltato.sz.setText("");

//PARTE COMUNICAZIONE CON SERVER

String richiesta = this.displayAscoltato.requestedstringfield.getText();

pw.println("get:"+richiesta);

pw.flush();

//QUI THREAD DI RICEZIONE DATI

ReceiverThread rt = new ReceiverThread(sc, this.displayAscoltato);

Thread thread = new Thread(rt);

thread.start();

}

}else if(e.getActionCommand().equals(STOP)) {

//PARTE EVOLUZIONE GRAFICA

this.displayAscoltato.started=false;

this.displayAscoltato.setButtons();

//PARTE COMUNICAZIONE CON SERVER

pw.println("stop");

pw.flush();

}

}

private void setupConnection() throws IOException {

String indirizzo = this.displayAscoltato.addressfield.getText();

int port = Integer.parseInt(this.displayAscoltato.portfield.getText());

socket = new Socket(indirizzo, port);

pw = new PrintWriter(new OutputStreamWriter(socket.getOutputStream()));

sc = new Scanner(socket.getInputStream());

}

private void closeConnection() throws IOException {

sc.close();

pw.close();

socket.close();

}

private boolean controlloStringaVuota() {

boolean res;

if (this.displayAscoltato.requestedstringfield.getText().equals("")) {

res = true;

//LA STRINGA è VUOTA

}else {

res = false;

//LA STRINGA NON è VUOTA

}

return res;

}

}

//RECEIVERTHREAD.JAVA

package parole;

import java.util.Scanner;

public class ReceiverThread implements Runnable{

//CAMPI

private Scanner scanner;

private Display display;

//COSTRUTTORE

public ReceiverThread(Scanner scanner, Display display) {

this.scanner = scanner;

this.display = display;

}

//METODO RUN

@Override

public void run() {

boolean running = true;

while (running) {

String cmd = scanner.nextLine();

if(cmd.equals("END")) {

running = false;

//PRIMA DI USCIRE RICHIAMA SET BUTTONS PER

//RIABILITARE GET

display.started=false;

display.setButtons();

break;

}else if(cmd.equals("INTERRUPT")) {

running = false;

break;

}else if(cmd.equals("ERROR")) {

running = false;

break;

}else{

String attualeai = display.ai.getText();

String attualejr = display.jr.getText();

String attualesz = display.sz.getText();

if ((cmd.compareTo("i")<0 || String.valueOf(cmd.charAt(0)).equals("i"))

&& (cmd.compareTo("a")>0 || String.valueOf(cmd.charAt(0)).equals("a"))) {

display.ai.setText(attualeai + "\n" + cmd);

}else if((cmd.compareTo("q")<0 || String.valueOf(cmd.charAt(0)).equals("q"))

&& (cmd.compareTo("j")>0 || String.valueOf(cmd.charAt(0)).equals("j"))) {

display.jr.setText(attualejr + "\n" + cmd);

}else {

display.sz.setText(attualesz + "\n" + cmd);

}

}

}

}

}

//PRINCIPALE.JAVA

package parole;

public class Principale {

public static void main(String[] args) {

Display display = new Display();

}

}