**LAN UTILITY**

by

D. Shashank - 13BCE1036

Harshwardhan Agarwal - 13BCE1052

K. Gourav - 13BCE1068

S. Nikesh Kumar Reddy - 13BCE1136

*A project report submitted to the*

**SCHOOL OF COMPUTING SCIENCES AND ENGINEERING**

*In partial fulfillment of the award for degree*

*of*

**BACHELOR OF TECHNOLOGY**

*in*

**COMPUTER SCIENCE ENGINEERING**



**VIT UNIVERSITY, CHENNAI**

**Vandalur – Kelambakkam Road**

**Chennai – 600 127.**

**MAY 2015**

**ACKNOWLEDGEMENT**

**Course Code: CSE323 Batch no: 1**

This project report entitled “**LAN UTILITY”** is a bonafide work of **SHASHANK. D (13BCE1036), HARSHWARDHAN AGARWAL (13BCE1052), K. GOURAV (13BCE1068) and NIKESH. S (13BCE1136)** and have carried out the project work under the supervision and guidance of **Prof. JAGADEESH KANNAN.** We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Prof.** **JAGADEESH KANNAN,** Associate Professor, School of Computing Sciences and Engineering, for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

**TABLE OF CONTENTS**

1. Abstract
2. Introduction
3. Client – Server mechanism
4. Modules and Explanation
5. References

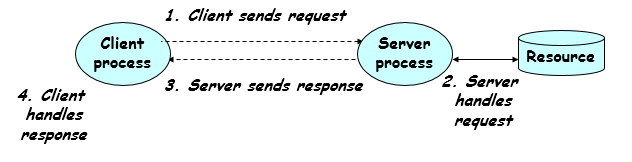
**ABSTRACT**

A LAN Utility is a software / application which is used for the establishment of connection between two or more people/systems by just connecting to the same network without any internet access. This type of utility enable people to connect and exchange information easily.

Implementation of LAN messenger now a days is a simple task which is made easier through socket programming followed by instant messaging (IM). Not only a messenger but also a lot more applications are possible through LAN. Digitalization is one major aspect in which India is very much lagging behind. This digitalization should be one of the key aspects on which both government and society should concentrate upon. This LAN utility enables digitalization in education as: for example, a lecturer wants to give a lecture, if every student and the lecturer are connected to the same network, this utility works in such a way that whatever the lecturer wants to convey through writing it on the board, if he writes it on his screen, it would reflect on every system and if any one of the students want to send some information, it would be shared with all.

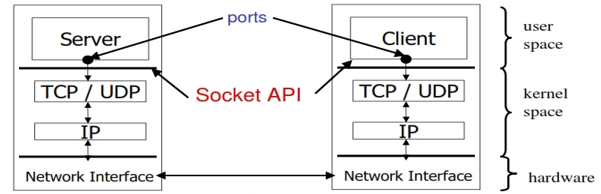
**Client – Server mechanism**

Socket Programming going hand in hand with client server mechanism gives the maximum through put to have a distinguished result.

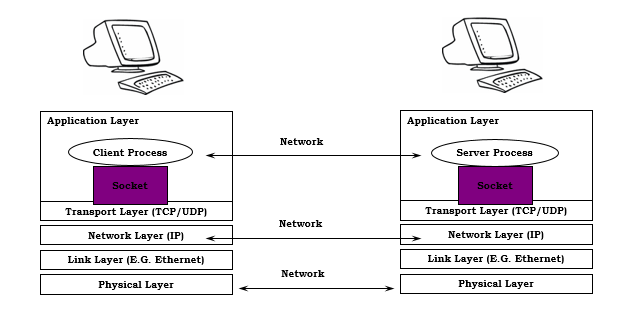


Client asks (*request*) – server provides (*response*). Typically a single server handles multiple clients but there can even be multiple servers handling multiple clients depending upon the number of clients and system resources. The server does not need to know *anything* about the client even that it exists but the client should always know *something* about the server at least where it is located. Clients and servers are processes running on network hosts i.e. they have to connect to a particular network (need not have internet access). They may be on the same host to be connected to each other for better transmission.

Sockets are used for inter process communication. Most of the inter-process communication follow a Client – Server model where client and server are two different processes. Server and Client exchange messages over the network through a common socket API.



The network interface is provided by the operating system in this way.



The IP address in the server socket address identifies the host

The (well-known) port in the server socket address identifies the service, and thus implicitly identifies the server process that performs that service.

Examples of well-known ports:

Port 7: Echo server

Port 23: Telnet server

Port 25: Mail server

Port 80: Web server

Servers are long-running processes (daemons). Created at boot-time (typically) by the init process (process 1). Run continuously until the machine is turned off. Each server waits for requests to arrive on a well-known port associated with a particular service. Other applications should choose between 1024 and 65535

To the kernel, a socket is an endpoint of communication. To an application, a socket is a file descriptor that lets the application read/write from/to the network. All Unix I/O devices, including networks, are modeled as files. Clients and servers communicate with each by reading from and writing to socket descriptors. The main distinction between regular file I/O and socket I/O is how the application “opens” the socket descriptors.

**Modules**

Messaging module:

As already explained, LAN messenger is made using TCP chat application. These are some of the steps involved in TCP protocol.

Server:

Step 1: Creating Socket

Step 2: Binding the socket with the system and port as information exchange channel.

Step 3: Listen to any connection requests being received to the server from any client.

Step 4: Accept the connection.

Step 5: Perform the actions requested by the client or ask the client to perform your desired actions. Actions here mean exchange of information (LAN Messenger)

Step 6: Close the socket.

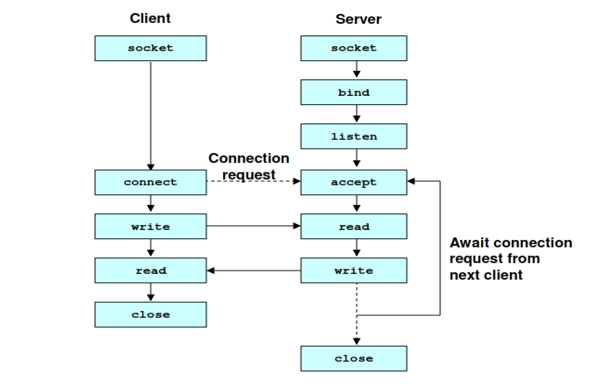
Client:

Step 1: Create the socket

Step 2: Request connection to the server.

Step 3: Exchange information with the server once the connection is established.

Step 4: Close the socket.



GUI Module:

Graphical User interface plays an important and one of the major roles in any application. Impression of the user is one basic aspect in which a developer/programmer should not override his thoughts and provide an impressive GUI for a utility to have its votes.

Utility Module:

To decide such a utility in a comprehensive manner, JAVA has been used to code the board module of the utility. Pen tip size, pseudo color configuration (214 combinations of RGB colors) and some other utilities have been added into the code and a flawless application/utility has been prepared.

The JAVA Code used is:

Message.java

package com.compnet.jlm.core.message;

import com.compnet.jlm.core.messenger.Messenger;

import java.beans.PropertyChangeListener;

import java.beans.PropertyChangeSupport;

import java.io.IOException;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\*

\* @author Nikesh

\*/

/\*

\* Message.java

\*

\* this class impliments a message object

\*/

public class Message {

// textual message

private String message = null;

// messenger associated with this message

private Messenger messenger = null;

//---------------------- GETTERS AND SETTERS -------------------------------

public String getMessage() {

return message;

}

public void setMessage(String message) {

// set the message content

this.message = message;

// firing the property change

// in-order to be listened by the property change listner components

this.propertyChangeSupport.firePropertyChange(messageProperty, 0, 1);

}

public Messenger getMessenger() {

return messenger;

}

public void setMessenger(Messenger messenger) {

this.messenger = messenger;

}

/\*

\* sendMessage Method

\* to send the current message using the messenger

\*/

public void sendMessage(String newMessage) {

try {

// send the message content using the messenger

messenger.sendMessage(newMessage);

} catch (IOException ex) {

Logger.getLogger(Message.class.getName()).log(Level.SEVERE, null, ex);

}

}

// --------------------- Property Change Listner ---------------------------

// message property name

private String messageProperty = "MESSAGE\_CHANGED";

// property change support object

private PropertyChangeSupport propertyChangeSupport = new PropertyChangeSupport(this);

/\*

\* addPropertyChangeListener Method

\* used by listners for this message for adding themself as a listner for this message

\*/

public void addPropertyChangeListener(PropertyChangeListener listener) {

propertyChangeSupport.addPropertyChangeListener(listener);

}

/\*

\* removePropertyChangeListener Method

\* used by listners for this message for removing themself as a listner for this message

\*/

public void removePropertyChangeListener(PropertyChangeListener listener) {

propertyChangeSupport.removePropertyChangeListener(listener);

}

// ------------------- MessegeProperty Getter and Setter -------------------

public String getMessegeProperty() {

return messageProperty;

}

public void setMessegeProperty(String messageProperty) {

this.messageProperty = messageProperty;

}

}

FilterTypes.java

package com.compnet.jlm.core.message.filters;

/\*\*

\*

\* @author Nikesh

\*/

/\*

\* FilterTypes.java

\*

\* this final class provides text filter types

\*/

public final class FilterTypes {

// text filter for chat messages

public static final int TEXT\_FILTER = 40000;

// control filter for whiteboard control messages

public static final int CONTROL\_FILTER = 40001;

// message type value location index

//- index where the type value is present in the recieved string message

public static final int TYPE\_LOC\_INDEX = 0;

}

MessageFilter.java

package com.compnet.jlm.core.message.filters;

import com.compnet.jlm.core.message.Message;

import java.beans.PropertyChangeEvent;

import java.beans.PropertyChangeListener;

/\*\*

\*

\* @author Nikesh

\*/

/\*

\* MessageFilter.java

\*

\* this class impliments message filter used to filter messages according to

\* the filter types

\*/

public class MessageFilter implements PropertyChangeListener {

// the raw message

private Message rawMessage = null;

// the filtered message

private Message filteredMessage = null;

// filter type variable

private int filterType = 0;

/\*

\* class constructor with input message and the filter type

\*/

public MessageFilter(Message inputMessage, int filterType) {

// set the filter type

this.setFilterType(filterType);

// set the input raw message

this.setInputMessage(inputMessage);

}

/\*

\* getter and setter for filter types

\*/

public int getFilterType() {

return filterType;

}

public final void setFilterType(int filterType) {

this.filterType = filterType;

}

/\*

\* Getter for the filter message

\*

\* Note:- NO need for filtered message setter

\*/

public Message getFilteredMessage() {

return filteredMessage;

}

/\*

\* method to set the input message as the raw message

\*/

public final void setInputMessage(Message inputMessage) {

// set the input message as the raw message

rawMessage = inputMessage;

// add this filter as a listner to the raw message object

rawMessage.addPropertyChangeListener(this);

// create the filtered message

filteredMessage = new Message();

// assign the filterd message messenger with the raw message messenger

filteredMessage.setMessenger(rawMessage.getMessenger());

}

/\*

\* method invoked on property change event

\* - ie raw message message content changed event

\*/

public void propertyChange(PropertyChangeEvent evt) {

try {

// if the message content type is equal to the assigned filter type then

if (rawMessage.getMessage().charAt(FilterTypes.TYPE\_LOC\_INDEX) == (char) filterType) {

// set the filtered message content

// - remove the content filter type value

filteredMessage.setMessage(rawMessage.getMessage().substring(1));

}

} catch (StringIndexOutOfBoundsException ex) {

// runtime error skipping` ...

}

}

}

MessageReader.java

package com.compnet.jlm.core.message.reader;

import com.compnet.jlm.core.message.Message;

import com.compnet.jlm.core.message.filters.FilterTypes;

import com.compnet.jlm.core.messenger.Messenger;

import java.io.IOException;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\*

\* @author Nikesh

\*/

/\*

\* MessageReader.java

\*

\* this class impliments a message reader

\*/

public class MessageReader {

// the message object

private Message message;

// the messenger reference object

private Messenger messengerInterface;

// flag to stop reading

private boolean stop = false;

// runnable Reader class object

private Reader reader = null;

// thread for reader runnable

private Thread thread = null;

/\*

\* privatising default constructor inorder to prevent usage

\*/

private MessageReader() {

}

/\*

\* class constructor with message object and messenger reference object as parameter

\*/

public MessageReader(Message msg, Messenger msngrInterface) {

// set the message object

this.message = msg;

// set the messenger object

this.messengerInterface = msngrInterface;

// create reader

reader = new Reader();

// create thread for the reader

thread = new Thread(reader);

}

/\*

\* method to start reading messages

\*/

public void startReading() {

// set stop flag as false

stop = false;

// start the thread for reader runnable

thread.start();

}

/\*

\* method to stop reading messages

\*/

public void stopReading() {

// set stop flag to true

stop = true;

}

/\*

\* Reader Class

\*

\* this runnable class impliments the run method to read the message

\* continiously from the messenger

\*/

private class Reader implements Runnable {

@Override

public void run() {

// read till stop equals true

while (!stop) {

try {

// read the message from messenger and set it to the message object

message.setMessage(messengerInterface.recieveMessage());

} catch (IOException ex) {

Logger.getLogger(MessageReader.class.getName()).log(Level.SEVERE, null, ex);

} catch (NullPointerException ex) {

// null pointer exception found when the other side connection is closed

// so inform the user through the message

message.setMessage((char) FilterTypes.TEXT\_FILTER + "INFO: Other side connection closed.");

// stop the message reading

stop = true;

}

}

}

}

}

Messenger.java

package com.compnet.jlm.core.messenger;

import java.io.IOException;

/\*\*

\*

\* @author Nikesh

\*/

/\*

\* Messenger.java

\*

\* This interface contains all the abstract functions a messenger must have

\*/

public interface Messenger {

/\*

\* connect method

\* to connect to a specific ip at port with id portId

\*/

public void connect(String ip, int portId) throws IOException;

/\*

\* sendMessage method

\* to send a string message to the other side

\*/

public void sendMessage(String message) throws IOException;

/\*

\* recieveMessage method

\* to recieve messege sent from the other side

\*/

public String recieveMessage() throws IOException;

/\*

\* disconnect method

\* to disconnect the connection with the other side

\*/

public void disconnect() throws IOException;

/\*

\* isConnected method

\* to check the connection status

\*/

public boolean isConnected() throws IOException;

}

MessengerAbstraction.java

package com.compnet.jlm.core.messenger;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStream;

import java.io.PrintWriter;

import java.net.ServerSocket;

import java.net.Socket;

/\*\*

\*

\* @author Nikesh

\*/

/\*

\* MessengerAbstraction.java

\*

\* this abstract class implements those methods of the Messenger

\* which have same implimentation.

\*

\* only the connect method is not implimented in here

\*/

public abstract class MessengerAbstraction implements Messenger {

// server socket object for connction if selected connection type is server

private ServerSocket serverSocket;

// a base socket object for both client and server

private Socket socket;

// inputstream object for reading reading the data/ message recieved at the socket

private InputStream inputStream;

// buffered reader object to provide a wrapper to the inputstream for reading message

private BufferedReader bufferedReader;

// print writer object to write message onto the socket

private PrintWriter printWriter = null;

// message string object

private String readMessage = null;

/\*

\* getter and setter for Messenger objects

\*

\* to be used by the classes which extends this Messenger abstraction class

\*/

public BufferedReader getBufferedReader() {

return bufferedReader;

}

public void setBufferedReader(BufferedReader bufferedReader) {

this.bufferedReader = bufferedReader;

}

public InputStream getInputStream() {

return inputStream;

}

public void setInputStream(InputStream inputStream) {

this.inputStream = inputStream;

}

public PrintWriter getPrintWriter() {

return printWriter;

}

public void setPrintWriter(PrintWriter printWriter) {

this.printWriter = printWriter;

}

public ServerSocket getServerSocket() {

return serverSocket;

}

public void setServerSocket(ServerSocket serverSocket) {

this.serverSocket = serverSocket;

}

public Socket getSocket() {

return socket;

}

public void setSocket(Socket socket) {

this.socket = socket;

}

/\*

\* Messenger Implimentation

\*/

/\*

\* send message methos to send message

\*/

public void sendMessage(String message) throws IOException {

printWriter.println(message);

}

/\*

\* recieve message to recieve the sent message from the other side

\*/

public String recieveMessage() throws IOException, NullPointerException {

// using buffered reader to read a line of the message

readMessage = bufferedReader.readLine();

// if the read data is null then throw null pointer exception

if (readMessage == null) {

throw new NullPointerException();

}

// else return the read message string

return readMessage;

}

/\*

\* disconnect method to disconnect the socket connection

\*/

public void disconnect() throws IOException {

// close socket

socket.close();

// if server socket is present then close it

if (serverSocket != null) {

serverSocket.close();

}

}

/\*

\* isConnected method to check the connection status

\*/

public boolean isConnected() throws IOException {

// return socket connection status

return socket.isConnected();

}

}

MessengerClient.java

package com.compnet.jlm.core.messenger.client;

import com.compnet.jlm.core.messenger.MessengerAbstraction;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.Socket;

/\*\*

\*

\* @author Nikesh

\*/

/\*

\* MessengerClient.java

\*

\* this class extends from the MessengerAbstraction abstract class

\* and provides the implimentation for the connect method for the client type

\*/

public class MessengerClient extends MessengerAbstraction {

/\*

\* connect method as defined in the MessengerInterface

\* used to connect to a server with ip and portId

\*/

public void connect(String ip, int portId) throws IOException {

// creating the socket

this.setSocket(new Socket(ip, portId));

// set the inputstream

this.setInputStream(this.getSocket().getInputStream());

// set the buffered reader using the input stream; for reading data

this.setBufferedReader(new BufferedReader(new InputStreamReader(this.getInputStream())));

// set the print writer; for writing data

this.setPrintWriter(new PrintWriter(this.getSocket().getOutputStream(), true));

}

}

MessengerServer.java

package com.compnet.jlm.core.messenger.server;

import com.compnet.jlm.core.messenger.MessengerAbstraction;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.ServerSocket;

/\*\*

\*

\* @author Nikesh

\*/

/\*

\* MessengerServer.java

\*

\* this class extends from the MessengerAbstraction abstract class

\* and provides the implimentation for the connect method for the server type

\*/

public class MessengerServer extends MessengerAbstraction {

/\*

\* connect method as defined in the MessengerInterface

\* used to start the server connection

\*/

public void connect(String ip, int portId) throws IOException {

// starting the server at port

this.setServerSocket(new ServerSocket(portId));

// creating the socket

this.setSocket(this.getServerSocket().accept());

// set the inputstream

this.setInputStream(this.getSocket().getInputStream());

// set the buffered reader using the input stream; for reading data

this.setBufferedReader(new BufferedReader(new InputStreamReader(this.getInputStream())));

// set the print writer; for writing data

this.setPrintWriter(new PrintWriter(this.getSocket().getOutputStream(), true));

}

}

FileTransfer.java

package com.compnet.jlm.filetransferer;

/\*\*

\*

\* @author Nikesh

\*/

public class FileTransferer {

}

JLMFrame.java

package com.compnet.jlm.gui;

import com.compnet.jlm.core.message.Message;

import com.compnet.jlm.core.message.filters.FilterTypes;

import com.compnet.jlm.core.message.filters.MessageFilter;

import com.compnet.jlm.core.message.reader.MessageReader;

import com.compnet.jlm.core.messenger.Messenger;

import com.compnet.jlm.core.messenger.client.MessengerClient;

import com.compnet.jlm.core.messenger.server.MessengerServer;

import com.compnet.jlm.gui.components.ChatTextArea;

import com.compnet.jlm.utils.Calender;

import com.compnet.jlm.whiteboard.gui.WhiteBoardFrame;

import java.awt.event.KeyEvent;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.net.InetAddress;

import java.net.SocketException;

import java.net.UnknownHostException;

import java.util.Properties;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.imageio.ImageIO;

import javax.swing.JFileChooser;

import javax.swing.JOptionPane;

/\*\*

\*

\* @author Nikesh

\*/

/\*

\* JLMFrame.java

\*

\* this class provides the gui front end for the JLM project

\*/

public class JLMFrame extends javax.swing.JFrame {

// a object reffernce of the Messenger

// used to reffernce server or client based on user selection

private Messenger messenger = null;

// white board object

private WhiteBoardFrame whiteBoard = null;

// raw message object - which will be read by the reader

private Message rawMessage = null;

// text message object - filtered text message from the raw message

private Message textMessage = null;

// control message object - filtered control message from the raw message

private Message controlMessage = null;

// text message filter to filter out text message from raw message

private MessageFilter textMessageFilter = null;

// control message filter to filter out control message from raw message

private MessageFilter controlMessageFilter = null;

// message reader for continious reading from the socket

private MessageReader messageReader = null;

// chat text area object for displaying chat messages

private ChatTextArea chatTextArea = null;

// the current user name

private String userName = null;

// server ip

private String ipString = null;

// server port id

private String portIdString = null;

// the program startup date and time used in message logging

private String chatStartUpDateAndTime = null;

/\*\*

\* Creates new form JLMFrame

\*/

public JLMFrame() {

// initialise the chat text area

// as it is reffernced by the frame textarea object ( customised generated code)

chatTextArea = new ChatTextArea();

// initilise the base components

initComponents();

// the frames default exit option is set to DO\_NOTHING

// so as to prevent exit while in connection

// exit is only done by the exitJLM method

// set the frame icon

try {

this.setIconImage(ImageIO.read(getClass().getResource("/com/compnet/jlm/gui/resources/chat\_16x16.png")));

} catch (IOException ex) {

Logger.getLogger(JLMFrame.class.getName()).log(Level.SEVERE, null, ex);

}

// initialise the white board object

whiteBoard = new WhiteBoardFrame();

}

/\*\*

\* This method is called from within the constructor to initialize the form.

\* WARNING: Do NOT modify this code. The content of this method is always

\* regenerated by the Form Editor.

\*/

@SuppressWarnings("unchecked")

// <editor-fold defaultstate="collapsed" desc="Generated Code">

private void initComponents() {

chatPanel = new javax.swing.JPanel();

jScrollPane1 = new javax.swing.JScrollPane();

frameTextArea = chatTextArea;

userInputPanel = new javax.swing.JPanel();

userInputTextField = new javax.swing.JTextField();

userInputSendButton = new javax.swing.JButton();

menuBar = new javax.swing.JMenuBar();

fileMenu = new javax.swing.JMenu();

saveChatMenuItem = new javax.swing.JMenuItem();

exitMenuItem = new javax.swing.JMenuItem();

connectionsMenu = new javax.swing.JMenu();

serverMenuItem = new javax.swing.JMenuItem();

clientMenuItem = new javax.swing.JMenuItem();

closeConnectionMenuItem = new javax.swing.JMenuItem();

whiteBoardMenu = new javax.swing.JMenu();

showBoardMenuItem = new javax.swing.JMenuItem();

helpMenu = new javax.swing.JMenu();

myIpMenuItem = new javax.swing.JMenuItem();

aboutMenuItem = new javax.swing.JMenuItem();

helpMenuItem = new javax.swing.JMenuItem();

setDefaultCloseOperation(javax.swing.WindowConstants.DO\_NOTHING\_ON\_CLOSE);

setTitle("JLM - Java LAN Messenger");

addWindowListener(new java.awt.event.WindowAdapter() {

public void windowClosing(java.awt.event.WindowEvent evt) {

windowClosingEvent(evt);

}

});

chatPanel.setBorder(javax.swing.BorderFactory.createTitledBorder("Chat"));

frameTextArea.setEditable(false);

frameTextArea.setColumns(20);

frameTextArea.setRows(5);

jScrollPane1.setViewportView(frameTextArea);

javax.swing.GroupLayout chatPanelLayout = new javax.swing.GroupLayout(chatPanel);

chatPanel.setLayout(chatPanelLayout);

chatPanelLayout.setHorizontalGroup(

chatPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(chatPanelLayout.createSequentialGroup()

.addContainerGap()

.addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT\_SIZE, 381, Short.MAX\_VALUE)

.addContainerGap())

);

chatPanelLayout.setVerticalGroup(

chatPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(chatPanelLayout.createSequentialGroup()

.addContainerGap()

.addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT\_SIZE, 98, Short.MAX\_VALUE)

.addContainerGap())

);

userInputPanel.setBorder(javax.swing.BorderFactory.createTitledBorder("Input"));

userInputTextField.setForeground(new java.awt.Color(0, 0, 204));

userInputTextField.addKeyListener(new java.awt.event.KeyAdapter() {

public void keyTyped(java.awt.event.KeyEvent evt) {

inputTextField\_KeyTypedEvent(evt);

}

});

userInputSendButton.setIcon(new javax.swing.ImageIcon(getClass().getResource("/com/compnet/jlm/gui/resources/send.png"))); // NOI18N

userInputSendButton.setText("Send");

userInputSendButton.addMouseListener(new java.awt.event.MouseAdapter() {

public void mousePressed(java.awt.event.MouseEvent evt) {

inputSend\_MousePressedEvent(evt);

}

});

javax.swing.GroupLayout userInputPanelLayout = new javax.swing.GroupLayout(userInputPanel);

userInputPanel.setLayout(userInputPanelLayout);

userInputPanelLayout.setHorizontalGroup(

userInputPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(userInputPanelLayout.createSequentialGroup()

.addContainerGap()

.addComponent(userInputTextField, javax.swing.GroupLayout.PREFERRED\_SIZE, 294, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addComponent(userInputSendButton)

.addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))

);

userInputPanelLayout.setVerticalGroup(

userInputPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(userInputPanelLayout.createSequentialGroup()

.addContainerGap()

.addGroup(userInputPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(userInputTextField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(userInputSendButton))

.addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))

);

fileMenu.setText("File");

saveChatMenuItem.setIcon(new javax.swing.ImageIcon(getClass().getResource("/com/compnet/jlm/gui/resources/save.png"))); // NOI18N

saveChatMenuItem.setText("Save Chat");

saveChatMenuItem.addMouseListener(new java.awt.event.MouseAdapter() {

public void mousePressed(java.awt.event.MouseEvent evt) {

saveChatMenuItem\_MousePressedEvent(evt);

}

});

fileMenu.add(saveChatMenuItem);

exitMenuItem.setIcon(new javax.swing.ImageIcon(getClass().getResource("/com/compnet/jlm/gui/resources/exit.png"))); // NOI18N

exitMenuItem.setText("Exit");

exitMenuItem.addMouseListener(new java.awt.event.MouseAdapter() {

public void mousePressed(java.awt.event.MouseEvent evt) {

fileExit\_MousePressedEvent(evt);

}

});

fileMenu.add(exitMenuItem);

menuBar.add(fileMenu);

connectionsMenu.setText("Connections");

serverMenuItem.setIcon(new javax.swing.ImageIcon(getClass().getResource("/com/compnet/jlm/gui/resources/server.png"))); // NOI18N

serverMenuItem.setText("Server");

serverMenuItem.addMouseListener(new java.awt.event.MouseAdapter() {

public void mousePressed(java.awt.event.MouseEvent evt) {

connectionsServer\_MousePressedEvent(evt);

}

});

connectionsMenu.add(serverMenuItem);

clientMenuItem.setIcon(new javax.swing.ImageIcon(getClass().getResource("/com/compnet/jlm/gui/resources/client.png"))); // NOI18N

clientMenuItem.setText("Client");

clientMenuItem.addMouseListener(new java.awt.event.MouseAdapter() {

public void mousePressed(java.awt.event.MouseEvent evt) {

connectionsClient\_MousePressedEvent(evt);

}

});

connectionsMenu.add(clientMenuItem);

closeConnectionMenuItem.setIcon(new javax.swing.ImageIcon(getClass().getResource("/com/compnet/jlm/gui/resources/disconnect.png"))); // NOI18N

closeConnectionMenuItem.setText("Close Connection");

closeConnectionMenuItem.addMouseListener(new java.awt.event.MouseAdapter() {

public void mousePressed(java.awt.event.MouseEvent evt) {

connectionsCloseConnection\_MousePressedEvent(evt);

}

});

connectionsMenu.add(closeConnectionMenuItem);

menuBar.add(connectionsMenu);

whiteBoardMenu.setText("WhiteBoard");

showBoardMenuItem.setIcon(new javax.swing.ImageIcon(getClass().getResource("/com/compnet/jlm/whiteboard/gui/resources/whiteBoardIcon\_16x16.png"))); // NOI18N

showBoardMenuItem.setText("Show Board");

showBoardMenuItem.addMouseListener(new java.awt.event.MouseAdapter() {

public void mousePressed(java.awt.event.MouseEvent evt) {

showBoardMenuItem\_MousePressedEvent(evt);

}

});

whiteBoardMenu.add(showBoardMenuItem);

menuBar.add(whiteBoardMenu);

helpMenu.setText("Help");

myIpMenuItem.setIcon(new javax.swing.ImageIcon(getClass().getResource("/com/compnet/jlm/gui/resources/ip.png"))); // NOI18N

myIpMenuItem.setText("My IP");

myIpMenuItem.addMouseListener(new java.awt.event.MouseAdapter() {

public void mousePressed(java.awt.event.MouseEvent evt) {

myIPMenuItem\_MousePressedEvent(evt);

}

});

helpMenu.add(myIpMenuItem);

aboutMenuItem.setIcon(new javax.swing.ImageIcon(getClass().getResource("/com/compnet/jlm/gui/resources/about.png"))); // NOI18N

aboutMenuItem.setText("About");

aboutMenuItem.addMouseListener(new java.awt.event.MouseAdapter() {

public void mousePressed(java.awt.event.MouseEvent evt) {

helpAbout\_MousePressedEvent(evt);

}

});

helpMenu.add(aboutMenuItem);

helpMenuItem.setIcon(new javax.swing.ImageIcon(getClass().getResource("/com/compnet/jlm/gui/resources/help.png"))); // NOI18N

helpMenuItem.setText("Help");

helpMenuItem.addMouseListener(new java.awt.event.MouseAdapter() {

public void mousePressed(java.awt.event.MouseEvent evt) {

helpHelp\_MousePressedEvent(evt);

}

});

helpMenu.add(helpMenuItem);

menuBar.add(helpMenu);

setJMenuBar(menuBar);

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());

getContentPane().setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(chatPanel, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addComponent(userInputPanel, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))

.addContainerGap())

);

layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addContainerGap()

.addComponent(chatPanel, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addComponent(userInputPanel, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addContainerGap())

);

pack();

}// </editor-fold>

/\*

\* exitApp method

\* to exit from the application

\*/

private void exitAPP() {

// set the messenger to null

messenger = null;

// set visibility to false

this.setVisible(false);

// dispose the gui resources

this.dispose();

// call garbage collector

System.gc();

// exit the application

System.exit(0);

}

/\*

\* exitJLM method

\* to exit from the JLM application

\*/

private void exitJLM() {

try {

// if messenger is initialised to client or server or is connected then

if (messenger != null && messenger.isConnected()) {

// ask the user, whether to terminate the live connection

int r = JOptionPane.showConfirmDialog(rootPane, "Live Connection. Terminate?", "Connection", JOptionPane.YES\_NO\_OPTION);

// on user approval

if (r == JOptionPane.YES\_OPTION) {

// disconnect the connection

messenger.disconnect();

// exit the application

exitAPP();

}

} // if no connection then exit

else {

exitAPP();

}

} catch (SocketException ex) {

JOptionPane.showMessageDialog(rootPane, "Connection Termination Failed!\nSocket Exception:\n" + ex + "\nExiting.");

Logger.getLogger(JLMFrame.class.getName()).log(Level.SEVERE, null, ex);

exitAPP();

} catch (IOException ex) {

JOptionPane.showMessageDialog(rootPane, "Connection Termination Failed!\nIO Exception:\n" + ex + "\nExiting.");

Logger.getLogger(JLMFrame.class.getName()).log(Level.SEVERE, null, ex);

exitAPP();

} catch (NullPointerException ex) {

// on null pointer exception do nothing

} catch (Exception ex) {

JOptionPane.showMessageDialog(rootPane, "Connection Termination Failed!\nUnknown Exception:\n" + ex, "Error", JOptionPane.ERROR\_MESSAGE);

Logger.getLogger(JLMFrame.class.getName()).log(Level.SEVERE, null, ex);

exitAPP();

}

}

/\*

\* fileExit\_MousePressedEvent method

\* called when user selects the exit menu from the file menu of the menu bar

\*/

private void fileExit\_MousePressedEvent(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

// exit JLM

exitJLM();

}

/\*

\* connectionsServer\_MousePressedEvent method

\* called when user selects the server type connection from

\* the connections menu of the menu bar

\*/

private void connectionsServer\_MousePressedEvent(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

// connect only if no live connection

if (messenger == null) {

// get the server port id from the user

portIdString = JOptionPane.showInputDialog(rootPane, "Enter Server Port ID to set", "3333");

try {

// if user selects cancel then throw null pointer exception

if (portIdString == null) {

throw new NullPointerException();

}

// convert the string port id to integer port id

final int portId = Integer.parseInt(portIdString);

// assign the Messenger reffernce object with MessengerServer object

messenger = new MessengerServer();

// notify the user about the connection delay

JOptionPane.showMessageDialog(rootPane, "Starting Server. Please Wait.");

// start the server using a anonymous thread object with the port id

// no ip required for the server type connection

new Thread() {

@Override

public void run() {

try {

messenger.connect(null, portId);

updateMessages();

// notify the user about the successfull connection

JOptionPane.showMessageDialog(rootPane, "Chat Server started at Port Id :" + portIdString, "Server Started", JOptionPane.INFORMATION\_MESSAGE);

} catch (IOException ex) {

JOptionPane.showMessageDialog(rootPane, "Could not start Server using Port Id: " + portIdString, "Server Start Error", JOptionPane.ERROR\_MESSAGE);

Logger.getLogger(JLMFrame.class.getName()).log(Level.SEVERE, null, ex);

}

}

}.start();

// get the user name to be used while chatting

userName = JOptionPane.showInputDialog(rootPane, "Enter your chat name", "Server");

// if cancel pressed then set the default user name

if (userName == null) {

userName = "Server";

}

// set the chat startup time - used in chat save

chatStartUpDateAndTime = Calender.getCurrentDateAndTime();

} catch (NullPointerException ex) {

// if cancel option is selected by the user then do nothing

} catch (NumberFormatException ex) {

JOptionPane.showMessageDialog(rootPane, "Please Enter a valid Port Id!", "Port ID Error", JOptionPane.ERROR\_MESSAGE);

} catch (Exception ex) {

JOptionPane.showMessageDialog(rootPane, "Could not start Server using Port Id: " + portIdString, "Server Start Error", JOptionPane.ERROR\_MESSAGE);

Logger.getLogger(JLMFrame.class.getName()).log(Level.SEVERE, null, ex);

}

} else {

// if currently connected (live connection) then notify user

JOptionPane.showMessageDialog(rootPane, "Live Connection! Close Connection to Reset!");

}

}

/\*

\* connectionsClient\_MousePressedEvent method

\* called when user selects client type connection from the connection menu

\* of the menu bar

\*/

private void connectionsClient\_MousePressedEvent(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

// connect only if no live connection

if (messenger == null) {

try {

// get the server ip from user

ipString = JOptionPane.showInputDialog(rootPane, "Enter Server IP", "localhost");

// if user selects cancel then exit by throwing null pointer exception

if (ipString == null) {

throw new NullPointerException();

}

// get the serve port id from user

portIdString = JOptionPane.showInputDialog(rootPane, "Enter Server Port ID", "3333");

// if user selects cancel then exit by throwing null pointer exception

if (portIdString == null) {

throw new NullPointerException();

}

// assign the Messenger reffernce object with MessengerClient object

messenger = new MessengerClient();

// connect the client with the server using server ip and server port id

messenger.connect(ipString, Integer.parseInt(portIdString));

updateMessages();

// notify the user about the successfull connection

JOptionPane.showMessageDialog(rootPane, "Chat Client connected to Chat Server with IP: " + ipString + " and Port Id: " + portIdString, "Client Connected", JOptionPane.INFORMATION\_MESSAGE);

// get the user name to be used while chat

userName = JOptionPane.showInputDialog(rootPane, "Enter your chat name", "Client");

// if cancel pressed then set the default user name

if (userName == null) {

userName = "Client";

}

// set the chat startup time - used in chat save

chatStartUpDateAndTime = Calender.getCurrentDateAndTime();

} catch (NullPointerException ex) {

// user cancels the input of ip or port id, do nothing

nullifyMessengerAndMessages();

} catch (NumberFormatException ex) {

nullifyMessengerAndMessages();

JOptionPane.showMessageDialog(rootPane, "Please Enter a valid Port Id!", "Port ID Error", JOptionPane.ERROR\_MESSAGE);

} catch (IOException ex) {

nullifyMessengerAndMessages();

JOptionPane.showMessageDialog(rootPane, "Could not connect to Server with IP: " + ipString + " and Port Id: " + portIdString + "\nIO Exception:\n" + ex, "Connection Error", JOptionPane.ERROR\_MESSAGE);

Logger.getLogger(JLMFrame.class.getName()).log(Level.SEVERE, null, ex);

} catch (Exception ex) {

nullifyMessengerAndMessages();

JOptionPane.showMessageDialog(rootPane, "Could not connect to Server with IP: " + ipString + " and Port Id: " + portIdString + "\nUnknown Exception:\n" + ex, "Connection Error", JOptionPane.ERROR\_MESSAGE);

Logger.getLogger(JLMFrame.class.getName()).log(Level.SEVERE, null, ex);

}

} else {

// if currently connected (live connection) then notify user

JOptionPane.showMessageDialog(rootPane, "Live Connection! Close Connection to Reset!");

}

}

/\*

\* method to nullify the messenger and messages objects

\*/

private void nullifyMessengerAndMessages() {

// assign messenger to null

messenger = null;

// assign all messages to null

rawMessage = null;

textMessage = null;

controlMessage = null;

// call garbage collector to free the messenger memory

System.gc();

}

/\*

\* method to update the messages

\* - the recieved raw message

\* - the filterd control message

\* - the filterd text message

\*/

private void updateMessages() {

// create a raw message

rawMessage = new Message();

// set the mesenger for the raw message

rawMessage.setMessenger(messenger);

// create text message filter for the raw message

textMessageFilter = new MessageFilter(rawMessage, FilterTypes.TEXT\_FILTER);

// get the filteredtext message

textMessage = textMessageFilter.getFilteredMessage();

// assign the text message to the chat text area

chatTextArea.setMessage(textMessage);

// create control message filter for the raw message

controlMessageFilter = new MessageFilter(rawMessage, FilterTypes.CONTROL\_FILTER);

// get the filtered message

controlMessage = controlMessageFilter.getFilteredMessage();

// assign the control message to the white board

whiteBoard.setControlMessage(controlMessage);

// create message reader for the raw message using the messenger

messageReader = new MessageReader(rawMessage, messenger);

// start reading the raw message using the reader

messageReader.startReading();

}

/\*

\* method called when user selects the close connection menu item from the connection menu

\*/

private void connectionsCloseConnection\_MousePressedEvent(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

try {

// if messenger is initialised and connection is present then

if (messenger != null && messenger.isConnected()) {

// disconnect the connection

messenger.disconnect();

// notify the user about the successful disconnection

JOptionPane.showMessageDialog(rootPane, "Connection Disconnected", "Connection", JOptionPane.INFORMATION\_MESSAGE);

} else {

// if messenger is null then it means no connection, so notify the user.

JOptionPane.showMessageDialog(rootPane, "No Connections Found!", "Connection", JOptionPane.ERROR\_MESSAGE);

}

} catch (SocketException ex) {

JOptionPane.showMessageDialog(rootPane, "Connection Termination Failed!\nSocket Exception:\n" + ex, "Disconnection Error", JOptionPane.ERROR\_MESSAGE);

Logger.getLogger(JLMFrame.class.getName()).log(Level.SEVERE, null, ex);

} catch (IOException ex) {

JOptionPane.showMessageDialog(rootPane, "Connection Termination Failed!\nIO Exception:\n" + ex, "Disconnection Error", JOptionPane.ERROR\_MESSAGE);

Logger.getLogger(JLMFrame.class.getName()).log(Level.SEVERE, null, ex);

} catch (NullPointerException ex) {

// on null pointer exception do nothing

} catch (Exception ex) {

JOptionPane.showMessageDialog(rootPane, "Connection Termination Failed!\nUnknown Exception:\n" + ex, "Disconnection Error", JOptionPane.ERROR\_MESSAGE);

Logger.getLogger(JLMFrame.class.getName()).log(Level.SEVERE, null, ex);

} finally {

nullifyMessengerAndMessages();

}

}

/\*

\* method called when user selects the about menu from the help menu

\*/

private void helpAbout\_MousePressedEvent(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

try {

// create a property object

Properties prop = new Properties();

//load the jlm properties file

InputStream inputStream = this.getClass().getClassLoader().getResourceAsStream("com/compnet/jlm/gui/properties/JLM.properties");

prop.load(inputStream);

// set the about info from the jlm properties

String aboutMessage = prop.getProperty("AppName") + "\n\n"

+ "Developed By: " + prop.getProperty("DevelopedBy") + "\n"

+ "Email: " + prop.getProperty("DeveloperEmail");

// show the info about the JLM

JOptionPane.showMessageDialog(rootPane, aboutMessage, "About", JOptionPane.INFORMATION\_MESSAGE, new javax.swing.ImageIcon(getClass().getResource("/com/compnet/jlm/gui/resources/chat\_32x32.png")));

} catch (FileNotFoundException ex) {

JOptionPane.showMessageDialog(rootPane, "Error! Cannot Load About Docs!\nFile Not Found Exception:\n" + ex, "Error", JOptionPane.ERROR\_MESSAGE);

Logger.getLogger(JLMFrame.class.getName()).log(Level.SEVERE, null, ex);

} catch (IOException ex) {

JOptionPane.showMessageDialog(rootPane, "Error! Cannot Load About Docs!\nIO Exception:\n" + ex, "Error", JOptionPane.ERROR\_MESSAGE);

Logger.getLogger(JLMFrame.class.getName()).log(Level.SEVERE, null, ex);

} catch (Exception ex) {

JOptionPane.showMessageDialog(rootPane, "Error! Cannot Load About Docs!\nUnknown Exception:\n" + ex, "Error", JOptionPane.ERROR\_MESSAGE);

Logger.getLogger(JLMFrame.class.getName()).log(Level.SEVERE, null, ex);

}

}

/\*

\* method called when user selects the help menu from the help menu

\*/

private void helpHelp\_MousePressedEvent(java.awt.event.MouseEvent evt) {

{

// TODO add your handling code here:

try {

// create a property object

Properties prop = new Properties();

//load the jlm properties file

InputStream inputStream = this.getClass().getClassLoader().getResourceAsStream("com/compnet/jlm/gui/properties/JLM.properties");

prop.load(inputStream);

// set the help info from the property

String helpMessage = prop.getProperty("AppHelp");

// shoe the help info

JOptionPane.showMessageDialog(rootPane, helpMessage, "Help", JOptionPane.INFORMATION\_MESSAGE, new javax.swing.ImageIcon(getClass().getResource("/com/compnet/jlm/gui/resources/chat\_32x32.png")));

} catch (FileNotFoundException ex) {

JOptionPane.showMessageDialog(rootPane, "Error! Cannot Load Help Docs!\nFile Not Found Exception:\n" + ex, "Error", JOptionPane.ERROR\_MESSAGE);

Logger.getLogger(JLMFrame.class.getName()).log(Level.SEVERE, null, ex);

} catch (IOException ex) {

JOptionPane.showMessageDialog(rootPane, "Error! Cannot Load Help Docs!\nIO Exception:\n" + ex, "Error", JOptionPane.ERROR\_MESSAGE);

Logger.getLogger(JLMFrame.class.getName()).log(Level.SEVERE, null, ex);

} catch (Exception ex) {

JOptionPane.showMessageDialog(rootPane, "Error! Cannot Load Help Docs!\nUnknown Exception:\n" + ex, "Error", JOptionPane.ERROR\_MESSAGE);

Logger.getLogger(JLMFrame.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

/\*

\* method called when user tries to close the window

\*/

private void windowClosingEvent(java.awt.event.WindowEvent evt) {

// TODO add your handling code here:

// exit JLM

exitJLM();

}

/\*

\* method called when user presses enter key or send button

\* after entering the message to be sent

\*/

private void sendMessage() {

// TODO add your handling code here:

try {

// call the send message of the messenger reffernce object

textMessage.sendMessage((char) FilterTypes.TEXT\_FILTER + userName + ": " + userInputTextField.getText());

// add the message to the user's chat window

frameTextArea.append("\n" + userName + ": " + userInputTextField.getText());

} catch (NullPointerException ex) {

JOptionPane.showMessageDialog(rootPane, "Could not Send Message!\nNo Connection!", "Send Error", JOptionPane.ERROR\_MESSAGE);

} catch (Exception ex) {

JOptionPane.showMessageDialog(rootPane, "Could not Send Message!\nUnknown Exception:\n" + ex, "Send Error", JOptionPane.ERROR\_MESSAGE);

Logger.getLogger(JLMFrame.class.getName()).log(Level.SEVERE, null, ex);

} finally {

// finally clear the chat text input field

userInputTextField.setText("");

}

}

/\*

\* method called when user presses the send button

\*/

private void inputSend\_MousePressedEvent(java.awt.event.MouseEvent evt) {

sendMessage();

}

/\*

\* method called when user types into the char input field

\*/

private void inputTextField\_KeyTypedEvent(java.awt.event.KeyEvent evt) {

// TODO add your handling code here:

// if user presses enter key then send the message

if (evt.getKeyChar() == KeyEvent.VK\_ENTER) {

// calling send message method

sendMessage();

}

}

/\*

\* method called when user selects the show board menu

\*/

private void showBoardMenuItem\_MousePressedEvent(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

// set the white board visible

whiteBoard.setVisible(true);

}

/\*

\* method called when user selects the save chat menu from file menu

\*/

private void saveChatMenuItem\_MousePressedEvent(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

// create a file chooser

JFileChooser jfc = new JFileChooser();

// get the selection result

int r = jfc.showSaveDialog(jfc);

// on approve selection

if (r == JFileChooser.APPROVE\_OPTION) {

// create file output stream object

FileOutputStream fout = null;

try {

// get the selected file

File f = jfc.getSelectedFile();

// create file output stream for the file

fout = new FileOutputStream(f);

// create the chat log header

String chatLog = "\*\*\*\*\*\*\* JLM Chat Log \*\*\*\*\*\*\*\n"

// set the program startup time

+ "Chat Started: " + chatStartUpDateAndTime + "\n"

// set the chat save time

+ "Chat Saved: " + Calender.getCurrentDateAndTime() + "\n"

// set the connection type

+ "Connected As: " + ((messenger instanceof MessengerClient) ? "Client" : "Server") + "\n"

// if connected as client then set the ip address

+ (messenger instanceof MessengerClient ? ("IP: " + ipString + "\n") : "")

// set the port id

+ "Port Id: " + portIdString + "\n"

// set the user name

+ "User Name: " + userName + "\n"

+ "\*\*\*\*\*\*\* Chat Message \*\*\*\*\*\*\*\n"

// set the chat message

+ chatTextArea.getText();

// write the created log onto the file

fout.write(chatLog.getBytes());

// flush the stream

fout.flush();

} catch (IOException ex) {

Logger.getLogger(JLMFrame.class.getName()).log(Level.SEVERE, null, ex);

} finally {

try {

// finally close the stream

fout.close();

JOptionPane.showMessageDialog(rootPane, "Chat Saved.");

} catch (IOException ex) {

Logger.getLogger(JLMFrame.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

}

/\*

\* method called when user selects the my ip menu from the help menu

\*/

private void myIPMenuItem\_MousePressedEvent(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

try {

// get the host ip and name and display it

InetAddress host = InetAddress.getLocalHost();

JOptionPane.showMessageDialog(

rootPane,

"Your IP: " + host.getHostAddress()

+ "\nHost Name: " + host.getHostName(),

"MyIP",

JOptionPane.INFORMATION\_MESSAGE,

new javax.swing.ImageIcon(getClass().getResource("/com/compnet/jlm/gui/resources/chat\_32x32.png")));

} catch (UnknownHostException ex) {

Logger.getLogger(JLMFrame.class.getName()).log(Level.SEVERE, null, ex);

}

}

/\*\*

\* @param args the command line arguments

\*/

public static void main(String args[]) {

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

new JLMFrame().setVisible(true);

}

});

}

// Variables declaration - do not modify

private javax.swing.JMenuItem aboutMenuItem;

private javax.swing.JPanel chatPanel;

private javax.swing.JMenuItem clientMenuItem;

private javax.swing.JMenuItem closeConnectionMenuItem;

private javax.swing.JMenu connectionsMenu;

private javax.swing.JMenuItem exitMenuItem;

private javax.swing.JMenu fileMenu;

private javax.swing.JTextArea frameTextArea;

private javax.swing.JMenu helpMenu;

private javax.swing.JMenuItem helpMenuItem;

private javax.swing.JScrollPane jScrollPane1;

private javax.swing.JMenuBar menuBar;

private javax.swing.JMenuItem myIpMenuItem;

private javax.swing.JMenuItem saveChatMenuItem;

private javax.swing.JMenuItem serverMenuItem;

private javax.swing.JMenuItem showBoardMenuItem;

private javax.swing.JPanel userInputPanel;

private javax.swing.JButton userInputSendButton;

private javax.swing.JTextField userInputTextField;

private javax.swing.JMenu whiteBoardMenu;

// End of variables declaration

}

ChatTextArea.java

package com.compnet.jlm.gui.components;

import com.compnet.jlm.core.message.Message;

import java.beans.PropertyChangeEvent;

import java.beans.PropertyChangeListener;

import javax.swing.JTextArea;

import javax.swing.text.DefaultCaret;

/\*\*

\*

\* @author Nikesh

\*/

/\*

\* ChatTextArea.java

\*

\* this class provides the chat text area implimentation to display the chat messages

\*/

public class ChatTextArea extends JTextArea implements PropertyChangeListener {

// message object

private Message message = null;

// getter for message object

public Message getMessage() {

return message;

}

// setter for message object

public void setMessage(Message message) {

// set the reference message

this.message = message;

// add the chat text area as alistner to the message object

// listining to message text change

this.message.addPropertyChangeListener(this);

}

/\*

\* method called on message text change to append the text message

\* to the chat text area

\*/

public void propertyChange(PropertyChangeEvent evt) {

this.append("\n" + this.message.getMessage());

}

/\*

\* constructo fo the ChatTextArea class

\*/

public ChatTextArea() {

// call super class default constructor

super();

// set auto scroll to the chat text area

// so as to always display the last added message

DefaultCaret caret = (DefaultCaret) this.getCaret();

caret.setUpdatePolicy(DefaultCaret.ALWAYS\_UPDATE);

}

}

JLMProperties

AppName = Java LAN Messenger

DevelopedBy = Team NGHS

DeveloperEmail = aboutus@gmail.com

AppHelp =\

- If you are initiating the chat then select Server mode.\n\

- If you want to join a Server select Client mode.\n\

- In Server mode you have to provide the Port Id.\n\

- You can see your IP address from the MyIP menu.\n\

- A default Port Id has been already provided.\n\

- In Client mode you have to enter the Server Ip and Server Port Id.\n\

- Use the input text field to enter your chat message.\n\

- Select the white board from the white board menu.\n\

- Save your chat using the save chat option from the file menu.\n\

- Before exiting the application close the connection.

Main.java

package com.compnet.jlm.main;

import com.compnet.jlm.gui.JLMFrame;

import com.compnet.jlm.whiteboard.gui.WhiteBoardFrame;

/\*\*

\*

\* @author Nikesh

\*/

public class Main {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

// TODO code application logic here

/\* Set the Nimbus look and feel \*/

//<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">

/\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.

\* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html

\*/

try {

for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {

if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());

break;

}

}

} catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(WhiteBoardFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(WhiteBoardFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(WhiteBoardFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(WhiteBoardFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

}

//</editor-fold>

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

// creating the main window frame for the application

// and setting its visibility to true

new JLMFrame().setVisible(true);

}

});

}

}

Calendar.java

package com.compnet.jlm.utils;

import java.text.DateFormat;

import java.text.SimpleDateFormat;

import java.util.Calendar;

/\*\*

\*

\* @author Nikesh

\*/

/\*

\* Calender.java

\*

\* this class provides a static method to get the current date and time

\*/

public class Calender {

public static String getCurrentDateAndTime() {

// set the date format

DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");

// get calender instance

Calendar cal = Calendar.getInstance();

// return the formated date and time

return dateFormat.format(cal.getTime());

}

}

WhiteBoardInterface.java

package com.compnet.jlm.whiteboard.core;

import java.awt.Color;

import java.awt.image.BufferedImage;

/\*\*

\*

\* @author Nikesh

\*/

/\*

\* this interface provides the methods that must be implimented for the whiteboard

\*/

public interface WhiteBoardInterface {

/\*

\* method to draw onto the white board at location (x,y)

\*/

public void drawOntoBoard(int x, int y);

/\*

\* method to resize the board

\*/

public void resizeBoard(int width, int height);

/\*

\* method to clear the contents of the board

\*/

public void clearBoard();

/\*

\* method to set an image onto the board

\*/

public void setBoardImage(BufferedImage boardImage);

/\*

\* method to set the chalk color

\*/

public void setChalkColor(Color chalkColor);

/\*

\* method to set the chalk size

\*/

public void setChalkSize(int chalkSize);

/\*

\* method to set the board color

\*/

public void setBoardColor(Color boardColor);

}

WhiteBoardPaner.java

package com.compnet.jlm.whiteboard.core.boardpanel;

import com.compnet.jlm.whiteboard.core.WhiteBoardInterface;

import java.awt.Color;

import java.awt.Cursor;

import java.awt.Graphics;

import java.awt.Image;

import java.awt.Point;

import java.awt.Toolkit;

import java.awt.image.BufferedImage;

import java.io.IOException;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.imageio.ImageIO;

/\*\*

\*

\* @author Nikesh

\*/

/\*

\* this class is theimplimentation of the white board interface using a jpanel

\*/

public class WhiteBoardPanel extends javax.swing.JPanel implements WhiteBoardInterface {

// the board image object onto which all drawings are made

private BufferedImage boardImage = null;

// temporary board image for board color change operations

private BufferedImage tempImage = null;

// graphics object for the board image

private Graphics boardImageGraphics = null;

// chalk color

private Color chalkColor = null;

// board color

private Color boardColor = null;

// chalk size

private int chalkSize = 0;

/\*\*

\* Creates new form WhiteBoardPanel

\*/

public WhiteBoardPanel() {

initComponents();

// set the custom cursor for the white board panel

setCursorIcon();

// set default chalk color, board color and chalk size

chalkColor = Color.BLACK;

boardColor = Color.WHITE;

chalkSize = 10;

}

/\*

\* the overridden paint method of the jpanel

\*/

@Override

public void paint(Graphics g) {

// set the board color

g.setColor(boardColor);

// clear the whole panel by filling it with the borad color

g.fillRect(0, 0, this.getWidth(), this.getHeight());

// draw the user drawn image onto the panel

g.drawImage(boardImage, 0, 0, this);

}

/\*\*

\* This method is called from within the constructor to initialize the form.

\* WARNING: Do NOT modify this code. The content of this method is always

\* regenerated by the Form Editor.

\*/

@SuppressWarnings("unchecked")

// <editor-fold defaultstate="collapsed" desc="Generated Code">

private void initComponents() {

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);

this.setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGap(0, 400, Short.MAX\_VALUE)

);

layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGap(0, 300, Short.MAX\_VALUE)

);

}// </editor-fold>

/\*

\* white board interface method implimentation

\* for drawing onto the panel (board)

\*/

@Override

public void drawOntoBoard(int x, int y) {

// pencil shape is circular so use fill oval

// drawing at location (x,y) with the chalk size

boardImageGraphics.fillOval(x - (chalkSize / 2), y - (chalkSize / 2), chalkSize, chalkSize);

// repaint the panel

this.repaint();

}

/\*

\* white board interface method implimentation

\* for resizing the board

\*/

@Override

public void resizeBoard(int width, int height) {

// resize only if board image is present

// and the new resolution is greater than the previous one

// this is done so as to prevent image clipping on panel resize by the user

// as if clipping is done then what ever the other person draws the current user wouldn't see

if ((boardImage == null) || ((width >= boardImage.getWidth()) && (height >= boardImage.getHeight()))) {

// save the curent board image to the temporary image

tempImage = boardImage;

// create a new image with new resolution and assign it to the board image

boardImage = new BufferedImage(width, height, BufferedImage.TYPE\_4BYTE\_ABGR\_PRE);

// get the graphics object for the new image

boardImageGraphics = boardImage.getGraphics();

// set the chalk color for the new graphics object

boardImageGraphics.setColor(chalkColor);

// if there is a valid temp image then

if (tempImage != null) {

// draw that image onto the new board image

boardImageGraphics.drawImage(tempImage, 0, 0, this);

}

// repaint the panel

this.repaint();

}

}

// Variables declaration - do not modify

// End of variables declaration

/\*

\* method to set the custom cursor icon for the board panel

\*/

private void setCursorIcon() {

try {

//Get the default toolkit

Toolkit toolkit = Toolkit.getDefaultToolkit();

//Load an image for the cursor

Image image = ImageIO.read(getClass().getResource("/com/compnet/jlm/whiteboard/core/resources/chalk.png"));

//Create the hotspot for the cursor

Point hotSpot = new Point(0, 0);

//Create the custom cursor

Cursor cursor = toolkit.createCustomCursor(image, hotSpot, "Chalk");

//Use the custom cursor

setCursor(cursor);

} catch (IOException ex) {

Logger.getLogger(WhiteBoardPanel.class.getName()).log(Level.SEVERE, null, ex);

}

}

/\*

\* white board interface method implimentation

\* for clearing the contents of the board

\*/

@Override

public void clearBoard() {

// assign new image with the same resolution to the board image object

boardImage = new BufferedImage(boardImage.getWidth(), boardImage.getHeight(), BufferedImage.TYPE\_INT\_ARGB\_PRE);

// get the graphics object for the new image

boardImageGraphics = boardImage.getGraphics();

// set the current chalk color for the graphics object

boardImageGraphics.setColor(chalkColor);

// repaint the panel

this.repaint();

}

/\*

\* GETTER AND SETTER for local variables

\*/

public BufferedImage getBoardImage() {

// here the refernce to the original image is not returned

// insted a new copy of the board image is made and returned

BufferedImage newBoardImage = new BufferedImage(boardImage.getWidth(), boardImage.getHeight(), BufferedImage.TYPE\_INT\_ARGB\_PRE);

Graphics newBoardImageGraphics = newBoardImage.getGraphics();

newBoardImageGraphics.setColor(boardColor);

newBoardImageGraphics.fillRect(0, 0, newBoardImage.getWidth(), newBoardImage.getHeight());

newBoardImageGraphics.drawImage(boardImage, 0, 0, this);

return newBoardImage;

}

@Override

public void setBoardImage(BufferedImage boardImage) {

this.boardImageGraphics.drawImage(boardImage, 0, 0, this);

this.repaint();

}

public Graphics getBoardImageGraphics() {

return boardImageGraphics;

}

public void setBoardImageGraphics(Graphics boardImageGraphics) {

this.boardImageGraphics = boardImageGraphics;

}

public Color getChalkColor() {

return chalkColor;

}

@Override

public void setChalkColor(Color chalkColor) {

this.chalkColor = chalkColor;

this.boardImageGraphics.setColor(this.chalkColor);

}

public int getChalkSize() {

return chalkSize;

}

@Override

public void setChalkSize(int chalkSize) {

this.chalkSize = chalkSize;

}

public Color getBoardColor() {

return boardColor;

}

@Override

public void setBoardColor(Color boardColor) {

this.boardColor = boardColor;

this.repaint();

}

}

CommandType.java

package com.compnet.jlm.whiteboard.core.controller;

/\*\*

\*

\* @author Nikesh

\*/

/\*

\* this final class provides the constants for the command message types.

\* for every method defined in the white board interface, there is a command type

\*/

public final class CommandType {

public static final int DRAW\_ONTO\_BOARD = 30000;

public static final int RESIZE\_BOARD = 30001;

public static final int CLEAR\_BOARD = 30002;

public static final int SET\_BOARD\_IMAGE = 30003;

public static final int SET\_CHALK\_COLOR = 30004;

public static final int SET\_CHALK\_SIZE = 30005;

public static final int SET\_BOARD\_COLOR = 30006;

}

CurrentBoardController.java

package com.compnet.jlm.whiteboard.core.controller;

import com.compnet.jlm.core.message.Message;

import com.compnet.jlm.whiteboard.core.boardpanel.WhiteBoardPanel;

import java.awt.Color;

import java.beans.PropertyChangeEvent;

import java.beans.PropertyChangeListener;

/\*\*

\*

\* @author Nikesh

\*/

/\*

\* this class provides method to control(draw and manipulate) the current

\* white board, according to the recieved control commad

\*/

public class CurrentBoardController implements PropertyChangeListener {

// the white board panel reffernce object

private WhiteBoardPanel whiteBoardPanel = null;

// the recieved control message reffernce object

private Message recievedMessage = null;

// the message string

private String messageString = null;

// the recieved command type

private int commandType = 0;

// current and remote chalk size and color

private int remoteChalkSize = 0;

private Color remoteChalkColor = null;

private int currentChalkSize = 0;

private Color currentChalkColor = null;

/\*

\* privatising the default constructor there by preventing usage

\*/

private CurrentBoardController() {

}

/\*

\* constructor with the white board panel and control message references as

\* arguments

\*/

public CurrentBoardController(WhiteBoardPanel whiteBoardPanel, Message recievedMessage) {

// set the white board panel refernce

this.whiteBoardPanel = whiteBoardPanel;

// set the recieved control message refernce

this.recievedMessage = recievedMessage;

// set the current class as a listner for the recieved message

this.recievedMessage.addPropertyChangeListener(this);

// initilise the current and remote chalk size and color

// with the white board defaults

currentChalkColor = whiteBoardPanel.getChalkColor();

remoteChalkColor = whiteBoardPanel.getChalkColor();

currentChalkSize = whiteBoardPanel.getChalkSize();

remoteChalkSize = whiteBoardPanel.getChalkSize();

}

/\*

\* property change event listner method

\*/

public void propertyChange(PropertyChangeEvent evt) {

// when ever the recieved message changes

//then process the recieved command message

processRecievedCommand();

}

/\*

\* method to process the recieved command message

\*/

private void processRecievedCommand() {

// get the command string

messageString = recievedMessage.getMessage();

// get the command type which is the first char in the message string

commandType = messageString.charAt(0);

switch (commandType) {

// if clear command

case CommandType.CLEAR\_BOARD: {

// clear the white board

whiteBoardPanel.clearBoard();

}

break;

// if draw command

case CommandType.DRAW\_ONTO\_BOARD: {

// get the current current chalk color and size

currentChalkColor = whiteBoardPanel.getChalkColor();

currentChalkSize = whiteBoardPanel.getChalkSize();

// set the remote chalk color and size to the current white board panel

whiteBoardPanel.setChalkColor(remoteChalkColor);

whiteBoardPanel.setChalkSize(remoteChalkSize);

// draw at the specified location

// the x and y coordiantes are present at the next two locations of the message string

whiteBoardPanel.drawOntoBoard(messageString.charAt(1), messageString.charAt(2));

// reset the white board panel's chalk color and size

// with the current current chalk color and size

whiteBoardPanel.setChalkColor(currentChalkColor);

whiteBoardPanel.setChalkSize(currentChalkSize);

}

break;

// if resize command

case CommandType.RESIZE\_BOARD: {

// resize the board panel to the specified resolution

// the width and height are present at the next two locations of the message string

whiteBoardPanel.resizeBoard(messageString.charAt(1), messageString.charAt(2));

}

break;

// if set board color command

case CommandType.SET\_BOARD\_COLOR: {

// set board color

whiteBoardPanel.setBoardColor(new Color(messageString.charAt(1), messageString.charAt(2), messageString.charAt(3)));

}

break;

//if set board image command

case CommandType.SET\_BOARD\_IMAGE: {

// ############# NOT YET IMPLIMENTED ###############

}

break;

// if set chalk color command

case CommandType.SET\_CHALK\_COLOR: {

// set remote chalk color

remoteChalkColor = new Color(messageString.charAt(1), messageString.charAt(2), messageString.charAt(3));

}

break;

// if set chalk size command

case CommandType.SET\_CHALK\_SIZE: {

try {

// set the remote chalk size

remoteChalkSize = messageString.charAt(1);

} catch (StringIndexOutOfBoundsException ex) {

// try catch to skip runtime error ...

}

}

break;

}

}

/\*

\* getter and setter for white board panel, control message

\*/

public WhiteBoardPanel getWhiteBoardPanel() {

return whiteBoardPanel;

}

public void setWhiteBoardPanel(WhiteBoardPanel whiteBoardPanel) {

this.whiteBoardPanel = whiteBoardPanel;

}

public Message getControlMessage() {

return recievedMessage;

}

public void setControlMessage(Message controlMessage) {

this.recievedMessage = controlMessage;

}

}

RemoteBoardController.java

package com.compnet.jlm.whiteboard.core.controller;

import com.compnet.jlm.core.message.Message;

import com.compnet.jlm.core.message.filters.FilterTypes;

import com.compnet.jlm.whiteboard.core.WhiteBoardInterface;

import java.awt.Color;

import java.awt.image.BufferedImage;

/\*\*

\*

\* @author Nikesh

\*/

/\*

\* this class provides methods for sending control commands to the

\* remote white board.

\*/

public class RemoteBoardController implements WhiteBoardInterface {

// the control message object

private Message controlMessage = null;

/\*

\* constructo with control message object as argument

\*/

public RemoteBoardController(Message message) {

this.controlMessage = message;

}

/\*

\* method to send draw command

\*/

public void drawOntoBoard(int x, int y) {

controlMessage.sendMessage(""

+ (char) FilterTypes.CONTROL\_FILTER

+ (char) CommandType.DRAW\_ONTO\_BOARD

+ (char) x

+ (char) y);

}

/\*

\* method to send resize board command

\*/

public void resizeBoard(int width, int height) {

controlMessage.sendMessage(""

+ (char) FilterTypes.CONTROL\_FILTER

+ (char) CommandType.RESIZE\_BOARD

+ (char) width

+ (char) height);

}

/\*

\* method to send clear board command

\*/

public void clearBoard() {

controlMessage.sendMessage(""

+ (char) FilterTypes.CONTROL\_FILTER

+ (char) CommandType.CLEAR\_BOARD);

}

/\*

\* method to send board image command ####### NOT YET IMPLIMENTED ######

\*/

public void setBoardImage(BufferedImage boardImage) {

/\*int[] rgbArray = new int[boardImage.getWidth()\*boardImage.getHeight()];

boardImage.getRGB(0, 0, boardImage.getWidth(), boardImage.getHeight(), rgbArray, 0, boardImage.getWidth());

controlMessage.sendMessage(""

+ (char) FilterTypes.CONTROL\_FILTER

+ (char) CommandType.SET\_BOARD\_IMAGE

+ boardImage.);\*/

}

/\*

\* method to send set chalk color command

\*/

public void setChalkColor(Color chalkColor) {

controlMessage.sendMessage(""

+ (char) FilterTypes.CONTROL\_FILTER

+ (char) CommandType.SET\_CHALK\_COLOR

+ (char) chalkColor.getRed()

+ (char) chalkColor.getGreen()

+ (char) chalkColor.getBlue());

}

/\*

\* method to send set chalk size command

\*/

public void setChalkSize(int chalkSize) {

controlMessage.sendMessage(""

+ (char) FilterTypes.CONTROL\_FILTER

+ (char) CommandType.SET\_CHALK\_SIZE

+ (char) chalkSize);

}

/\*

\* method to send set board color command

\*/

public void setBoardColor(Color boardColor) {

controlMessage.sendMessage(""

+ (char) FilterTypes.CONTROL\_FILTER

+ (char) CommandType.SET\_BOARD\_COLOR

+ (char) boardColor.getRed()

+ (char) boardColor.getGreen()

+ (char) boardColor.getBlue());

}

// getter and setter for control message

public Message getControlMessage() {

return controlMessage;

}

public void setControlMessage(Message controlMessage) {

this.controlMessage = controlMessage;

}

}

WhiteBoardFrame.java

package com.compnet.jlm.whiteboard.gui;

import com.compnet.jlm.core.message.Message;

import com.compnet.jlm.whiteboard.core.boardpanel.WhiteBoardPanel;

import com.compnet.jlm.whiteboard.core.controller.CurrentBoardController;

import com.compnet.jlm.whiteboard.core.controller.RemoteBoardController;

import com.compnet.jlm.whiteboard.utils.ContrastColor;

import java.awt.Color;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.InputStream;

import java.util.Properties;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.imageio.ImageIO;

import javax.swing.JColorChooser;

import javax.swing.JFileChooser;

import javax.swing.JOptionPane;

/\*\*

\*

\* @author Nikesh

\*/

/\*

\* this class provides the frame for the white board panel and implements

\* the user interface.

\*/

public class WhiteBoardFrame extends javax.swing.JFrame {

// the white board panel object

private WhiteBoardPanel whiteBoardPanel = null;

// the current board controller object to control the host white board panel

private CurrentBoardController currentBoardController = null;

// the remote board controller object to controll the remote white board panel

private RemoteBoardController remoteBoardController = null;

// control message object

private Message controlMessage = null;

// file object used while saving the drawn image

private File file = null;

// chalk and board color object

private Color chalkColor = null;

private Color boardColor = null;

// the chalk size variable

private int chalkSize = 0;

/\*\*

\* Creates new form WhiteBoardFrame

\*/

public WhiteBoardFrame() {

// initialise the white board panel

// as it is being refernce by the frame panel from within the generated code

whiteBoardPanel = new WhiteBoardPanel();

// initilise the components

initComponents();

// set the frame icon

try {

this.setIconImage(ImageIO.read(getClass().getResource("/com/compnet/jlm/whiteboard/gui/resources/whiteBoardIcon\_16x16.png")));

} catch (IOException ex) {

Logger.getLogger(WhiteBoardFrame.class.getName()).log(Level.SEVERE, null, ex);

}

// update the control panel button attribute

updateControlPanelButtonAttributes();

/\* ---------------------------------------------------------------------

\* THE SET IMAGE FUNCTION FOR THE WHITE BOARD IS NOT YET IMPLIMENTED

\* HENCE THE OPEN MENU ITEM IN THE FILE MENU IS DISSABLED FOR NOW

\* ---------------------------------------------------------------------

\*/

openMenuItem.setVisible(false);

openMenuItem.setEnabled(false);

}

/\*

\* method to update the control panel button attributes

\*/

private void updateControlPanelButtonAttributes() {

// set the chalk color selection button background color to the selected chalk color

chalkColorButton.setBackground(whiteBoardPanel.getChalkColor());

// set the chalk color selection button foreground color to the contrast of the selected chalk color

chalkColorButton.setForeground(ContrastColor.getContrastColor(whiteBoardPanel.getChalkColor()));

// set the board color selection button background color to the selected board color

boardColorButton.setBackground(whiteBoardPanel.getBoardColor());

// set the board color selection button foreground color to the contrast of the selected board color

boardColorButton.setForeground(ContrastColor.getContrastColor(whiteBoardPanel.getBoardColor()));

}

/\*\*

\* This method is called from within the constructor to initialize the form.

\* WARNING: Do NOT modify this code. The content of this method is always

\* regenerated by the Form Editor.

\*/

@SuppressWarnings("unchecked")

// <editor-fold defaultstate="collapsed" desc="Generated Code">

private void initComponents() {

framePanel = whiteBoardPanel;

controlPanel = new javax.swing.JPanel();

chalkColorButton = new javax.swing.JButton();

boardColorButton = new javax.swing.JButton();

chalkSizeSpinner = new javax.swing.JSpinner();

menuBar = new javax.swing.JMenuBar();

fileMenu = new javax.swing.JMenu();

newMenuItem = new javax.swing.JMenuItem();

openMenuItem = new javax.swing.JMenuItem();

saveMenuItem = new javax.swing.JMenuItem();

saveAsMenuItem = new javax.swing.JMenuItem();

exitMenuItem = new javax.swing.JMenuItem();

editMenu = new javax.swing.JMenu();

clearBoardMenuItem = new javax.swing.JMenuItem();

helpMenu = new javax.swing.JMenu();

aboutMenuItem = new javax.swing.JMenuItem();

helpMenuItem = new javax.swing.JMenuItem();

setTitle("WhiteBoard v0.1");

framePanel.addMouseListener(new java.awt.event.MouseAdapter() {

public void mousePressed(java.awt.event.MouseEvent evt) {

framePanel\_MousePressedEvent(evt);

}

});

framePanel.addComponentListener(new java.awt.event.ComponentAdapter() {

public void componentResized(java.awt.event.ComponentEvent evt) {

framePanel\_ComponentREsizedEvent(evt);

}

});

framePanel.addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {

public void mouseDragged(java.awt.event.MouseEvent evt) {

framePanel\_MouseDraggedEvent(evt);

}

});

javax.swing.GroupLayout framePanelLayout = new javax.swing.GroupLayout(framePanel);

framePanel.setLayout(framePanelLayout);

framePanelLayout.setHorizontalGroup(

framePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGap(0, 356, Short.MAX\_VALUE)

);

framePanelLayout.setVerticalGroup(

framePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGap(0, 0, Short.MAX\_VALUE)

);

controlPanel.setBorder(javax.swing.BorderFactory.createTitledBorder("Controls"));

chalkColorButton.setText("Chalk Color");

chalkColorButton.addMouseListener(new java.awt.event.MouseAdapter() {

public void mousePressed(java.awt.event.MouseEvent evt) {

controls\_chalkColorButton\_MousePressedEvent(evt);

}

});

boardColorButton.setText("Board Color");

boardColorButton.addMouseListener(new java.awt.event.MouseAdapter() {

public void mousePressed(java.awt.event.MouseEvent evt) {

controls\_boardColorButton\_\_MousePressedEvent(evt);

}

});

chalkSizeSpinner.setModel(new javax.swing.SpinnerNumberModel(10, 2, 100, 1));

chalkSizeSpinner.setBorder(javax.swing.BorderFactory.createTitledBorder("Chalk Size"));

chalkSizeSpinner.addChangeListener(new javax.swing.event.ChangeListener() {

public void stateChanged(javax.swing.event.ChangeEvent evt) {

controls\_chalkSizeSpinner\_StateChangedEvent(evt);

}

});

javax.swing.GroupLayout controlPanelLayout = new javax.swing.GroupLayout(controlPanel);

controlPanel.setLayout(controlPanelLayout);

controlPanelLayout.setHorizontalGroup(

controlPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(chalkColorButton)

.addComponent(chalkSizeSpinner, javax.swing.GroupLayout.PREFERRED\_SIZE, 94, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(boardColorButton)

);

controlPanelLayout.linkSize(javax.swing.SwingConstants.HORIZONTAL, new java.awt.Component[] {boardColorButton, chalkColorButton, chalkSizeSpinner});

controlPanelLayout.setVerticalGroup(

controlPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(controlPanelLayout.createSequentialGroup()

.addComponent(chalkColorButton)

.addGap(18, 18, 18)

.addComponent(chalkSizeSpinner, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(18, 18, 18)

.addComponent(boardColorButton)

.addContainerGap(85, Short.MAX\_VALUE))

);

fileMenu.setText("File");

newMenuItem.setIcon(new javax.swing.ImageIcon(getClass().getResource("/com/deepak/jlm/whiteboard/gui/resources/new.png"))); // NOI18N

newMenuItem.setText("New");

newMenuItem.addMouseListener(new java.awt.event.MouseAdapter() {

public void mousePressed(java.awt.event.MouseEvent evt) {

newMenuItem\_MousePressedEvent(evt);

}

});

fileMenu.add(newMenuItem);

openMenuItem.setIcon(new javax.swing.ImageIcon(getClass().getResource("/com/deepak/jlm/whiteboard/gui/resources/open.png"))); // NOI18N

openMenuItem.setText("Open");

openMenuItem.addMouseListener(new java.awt.event.MouseAdapter() {

public void mousePressed(java.awt.event.MouseEvent evt) {

openMenuItem\_MousePressedEvent(evt);

}

});

fileMenu.add(openMenuItem);

saveMenuItem.setIcon(new javax.swing.ImageIcon(getClass().getResource("/com/deepak/jlm/whiteboard/gui/resources/save.png"))); // NOI18N

saveMenuItem.setText("Save");

saveMenuItem.addMouseListener(new java.awt.event.MouseAdapter() {

public void mousePressed(java.awt.event.MouseEvent evt) {

saveMenuItem\_MousePressedEvent(evt);

}

});

fileMenu.add(saveMenuItem);

saveAsMenuItem.setIcon(new javax.swing.ImageIcon(getClass().getResource("/com/deepak/jlm/whiteboard/gui/resources/saveAs.png"))); // NOI18N

saveAsMenuItem.setText("Save As");

saveAsMenuItem.addMouseListener(new java.awt.event.MouseAdapter() {

public void mousePressed(java.awt.event.MouseEvent evt) {

saveAsMenuItem\_MousePressedEvent(evt);

}

});

fileMenu.add(saveAsMenuItem);

exitMenuItem.setIcon(new javax.swing.ImageIcon(getClass().getResource("/com/deepak/jlm/whiteboard/gui/resources/exit.png"))); // NOI18N

exitMenuItem.setText("Exit");

exitMenuItem.addMouseListener(new java.awt.event.MouseAdapter() {

public void mousePressed(java.awt.event.MouseEvent evt) {

exitMenuItem\_MousePressedEvent(evt);

}

});

fileMenu.add(exitMenuItem);

menuBar.add(fileMenu);

editMenu.setText("Edit");

clearBoardMenuItem.setIcon(new javax.swing.ImageIcon(getClass().getResource("/com/deepak/jlm/whiteboard/gui/resources/clear.png"))); // NOI18N

clearBoardMenuItem.setText("Clear Board");

clearBoardMenuItem.addMouseListener(new java.awt.event.MouseAdapter() {

public void mousePressed(java.awt.event.MouseEvent evt) {

clearBoardMenuItem\_MousePressedEvent(evt);

}

});

editMenu.add(clearBoardMenuItem);

menuBar.add(editMenu);

helpMenu.setText("Help");

aboutMenuItem.setIcon(new javax.swing.ImageIcon(getClass().getResource("/com/deepak/jlm/whiteboard/gui/resources/about.png"))); // NOI18N

aboutMenuItem.setText("About");

aboutMenuItem.addMouseListener(new java.awt.event.MouseAdapter() {

public void mousePressed(java.awt.event.MouseEvent evt) {

aboutMenuItem\_MousePressedEvent(evt);

}

});

helpMenu.add(aboutMenuItem);

helpMenuItem.setIcon(new javax.swing.ImageIcon(getClass().getResource("/com/deepak/jlm/whiteboard/gui/resources/help.png"))); // NOI18N

helpMenuItem.setText("Help");

helpMenuItem.addMouseListener(new java.awt.event.MouseAdapter() {

public void mousePressed(java.awt.event.MouseEvent evt) {

helpMenuItem\_MousePressedEvent(evt);

}

});

helpMenu.add(helpMenuItem);

menuBar.add(helpMenu);

setJMenuBar(menuBar);

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());

getContentPane().setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addContainerGap()

.addComponent(framePanel, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addGap(18, 18, 18)

.addComponent(controlPanel, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addContainerGap())

);

layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addComponent(controlPanel, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(0, 0, Short.MAX\_VALUE))

.addComponent(framePanel, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))

.addContainerGap())

);

pack();

}// </editor-fold>

/\*

\* method called when new menu item is selected

\*/

private void newMenuItem\_MousePressedEvent(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

// set file refernce to null

file = null;

// clear the board

whiteBoardPanel.clearBoard();

// if remote board controller is present then send clear command

if (remoteBoardController != null) {

remoteBoardController.clearBoard();

}

}

/\*

\* method called when open menu item is pressed

\*/

private void openMenuItem\_MousePressedEvent(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

// create a file chooser object

JFileChooser jfc = new JFileChooser();

// get selection result

int selection = jfc.showOpenDialog(jfc);

// if selection is of type approve

if (selection == JFileChooser.APPROVE\_OPTION) {

//get the selected file

file = jfc.getSelectedFile();

try {

// set the current white board panel image from the selected file

whiteBoardPanel.setBoardImage(ImageIO.read(file));

// if remote board controller is present then send command to set the remote board image

if (remoteBoardController != null) {

remoteBoardController.setBoardImage(ImageIO.read(file));

}

} catch (IOException ex) {

Logger.getLogger(WhiteBoardFrame.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

/\*

\* method called when save menu item is selected

\*/

private void saveMenuItem\_MousePressedEvent(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

// if no previous files saved then

if (file == null) {

// create a file chooser

JFileChooser jfc = new JFileChooser();

// get selection type

int selection = jfc.showSaveDialog(jfc);

// on approve selection

if (selection == JFileChooser.APPROVE\_OPTION) {

// get the selected file

file = jfc.getSelectedFile();

}

}

// if a previous copy of the image is already saved to a file then

if (file != null) {

try {

// write the image onto that file

ImageIO.write(whiteBoardPanel.getBoardImage(), "PNG", file);

} catch (IOException ex) {

Logger.getLogger(WhiteBoardFrame.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

/\*

\* method called when save as menu item mouse pressed

\*/

private void saveAsMenuItem\_MousePressedEvent(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

// create a file chooser

JFileChooser jfc = new JFileChooser();

// get selection result

int selection = jfc.showSaveDialog(jfc);

// if type is approve type

if (selection == JFileChooser.APPROVE\_OPTION) {

// get selected file

file = jfc.getSelectedFile();

try {

// save the board image onto the file

ImageIO.write(whiteBoardPanel.getBoardImage(), "PNG", file);

} catch (IOException ex) {

Logger.getLogger(WhiteBoardFrame.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

/\*

\* method called whenclear board menu item mouse pressed

\*/

private void clearBoardMenuItem\_MousePressedEvent(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

// clear the current white board panel

whiteBoardPanel.clearBoard();

// if remote board controller present then

if (remoteBoardController != null) {

// send command to clear the remote white board panel

remoteBoardController.clearBoard();

}

}

/\*

\* method called when about menu item moouse pressed

\*/

private void aboutMenuItem\_MousePressedEvent(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

try {

// create a property object

Properties prop = new Properties();

//load the white board properties file

InputStream inputStream = this.getClass().getClassLoader().getResourceAsStream("com/compnet/jlm/whiteboard/properties/whiteboard.properties");

prop.load(inputStream);

// set the about info from the white board properties

String aboutMessage = prop.getProperty("AppName") + "\n\n"

+ "Developed By: " + prop.getProperty("DevelopedBy") + "\n"

+ "Email: " + prop.getProperty("DeveloperEmail");

// show the info about the white board

JOptionPane.showMessageDialog(rootPane, aboutMessage, "About", JOptionPane.INFORMATION\_MESSAGE, new javax.swing.ImageIcon(getClass().getResource("/com/compnet/jlm/whiteboard/gui/resources/whiteBoardIcon\_32x32.png")));

} catch (FileNotFoundException ex) {

JOptionPane.showMessageDialog(rootPane, "Error! Cannot Load About Docs!\nFile Not Found Exception:\n" + ex, "Error", JOptionPane.ERROR\_MESSAGE);

Logger.getLogger(WhiteBoardFrame.class.getName()).log(Level.SEVERE, null, ex);

} catch (IOException ex) {

JOptionPane.showMessageDialog(rootPane, "Error! Cannot Load About Docs!\nIO Exception:\n" + ex, "Error", JOptionPane.ERROR\_MESSAGE);

Logger.getLogger(WhiteBoardFrame.class.getName()).log(Level.SEVERE, null, ex);

} catch (Exception ex) {

JOptionPane.showMessageDialog(rootPane, "Error! Cannot Load About Docs!\nUnknown Exception:\n" + ex, "Error", JOptionPane.ERROR\_MESSAGE);

Logger.getLogger(WhiteBoardFrame.class.getName()).log(Level.SEVERE, null, ex);

}

}

/\*

\* method called when help menu mouse pressed

\*/

private void helpMenuItem\_MousePressedEvent(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

try {

// create a property object

Properties prop = new Properties();

//load the white board properties file

InputStream inputStream = this.getClass().getClassLoader().getResourceAsStream("com/compnet/jlm/whiteboard/properties/whiteboard.properties");

prop.load(inputStream);

// set the help info from the property

String helpMessage = prop.getProperty("AppHelp");

// show the help info

JOptionPane.showMessageDialog(rootPane, helpMessage, "Help", JOptionPane.INFORMATION\_MESSAGE, new javax.swing.ImageIcon(getClass().getResource("/com/compnet/jlm/whiteboard/gui/resources/whiteBoardIcon\_32x32.png")));

} catch (FileNotFoundException ex) {

JOptionPane.showMessageDialog(rootPane, "Error! Cannot Load Help Docs!\nFile Not Found Exception:\n" + ex, "Error", JOptionPane.ERROR\_MESSAGE);

Logger.getLogger(WhiteBoardFrame.class.getName()).log(Level.SEVERE, null, ex);

} catch (IOException ex) {

JOptionPane.showMessageDialog(rootPane, "Error! Cannot Load Help Docs!\nIO Exception:\n" + ex, "Error", JOptionPane.ERROR\_MESSAGE);

Logger.getLogger(WhiteBoardFrame.class.getName()).log(Level.SEVERE, null, ex);

} catch (Exception ex) {

JOptionPane.showMessageDialog(rootPane, "Error! Cannot Load Help Docs!\nUnknown Exception:\n" + ex, "Error", JOptionPane.ERROR\_MESSAGE);

Logger.getLogger(WhiteBoardFrame.class.getName()).log(Level.SEVERE, null, ex);

}

}

/\*

\* method called when the chalk colour selection button mouse pressed

\*/

private void controls\_chalkColorButton\_MousePressedEvent(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

// set the chalk color from color chooser

chalkColor = JColorChooser.showDialog(this, "Select Chalk Color", Color.BLACK);

// set the white board chalk color

whiteBoardPanel.setChalkColor(chalkColor);

// if remote board controller present then send chalk color set command

if (remoteBoardController != null) {

remoteBoardController.setChalkColor(chalkColor);

}

updateControlPanelButtonAttributes();

}

/\*

\* method called when board color selection button mouse pressed

\*/

private void controls\_boardColorButton\_\_MousePressedEvent(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

// set the board color from the color chooser

boardColor = JColorChooser.showDialog(this, "Select Board Color", Color.WHITE);

// set the current white board panel board color

whiteBoardPanel.setBoardColor(boardColor);

// if remote board controller present then send the command to set the board color

if (remoteBoardController != null) {

remoteBoardController.setBoardColor(boardColor);

}

// update the control panel button attributes

updateControlPanelButtonAttributes();

}

/\*

\* method called when chalk size spinner state changed

\*/

private void controls\_chalkSizeSpinner\_StateChangedEvent(javax.swing.event.ChangeEvent evt) {

// TODO add your handling code here:

// set the chalk size from the spinner value

chalkSize = Integer.parseInt(chalkSizeSpinner.getValue().toString());

// set the current white board panel chalk size

whiteBoardPanel.setChalkSize(chalkSize);

// if remote board controller present then send remote board chalk size change command

if (remoteBoardController != null) {

remoteBoardController.setChalkSize(chalkSize);

}

}

/\*

\* method called when exit menu mouse pressed

\*/

private void exitMenuItem\_MousePressedEvent(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

// just dissappear than exitting

this.setVisible(false);

}

/\*

\* method called when frame panel mouse pressed

\*/

private void framePanel\_MousePressedEvent(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

// draw onto the white board panel at the mouse location

whiteBoardPanel.drawOntoBoard(evt.getX(), evt.getY());

// if remote board controller present then send the draw command at the mouse coordinates

if (remoteBoardController != null) {

remoteBoardController.drawOntoBoard(evt.getX(), evt.getY());

}

}

/\*

\* method called when frame panel mouse dragged

\*/

private void framePanel\_MouseDraggedEvent(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

// draw onto the white board panel at the mouse location

whiteBoardPanel.drawOntoBoard(evt.getX(), evt.getY());

// if remote board controller present then send the draw command at the mouse coordinates

if (remoteBoardController != null) {

remoteBoardController.drawOntoBoard(evt.getX(), evt.getY());

}

}

/\*

\* method called when frame panel component resized

\*/

private void framePanel\_ComponentREsizedEvent(java.awt.event.ComponentEvent evt) {

// TODO add your handling code here:

// resize the current white board panel

whiteBoardPanel.resizeBoard(this.getWidth(), this.getHeight());

// if remote board controller is present then send command to resize the remote board

if (remoteBoardController != null) {

remoteBoardController.resizeBoard(this.getWidth(), this.getHeight());

}

}

/\*

\* setter for the control message

\*/

public void setControlMessage(Message controlMessage) {

// set the control message

this.controlMessage = controlMessage;

// create remote board controller and current board controller object

// by providing the control message as the constructor parameter

remoteBoardController = new RemoteBoardController(this.controlMessage);

currentBoardController = new CurrentBoardController(whiteBoardPanel, this.controlMessage);

}

/\*\*

\* @param args the command line arguments

\*/

public static void main(String args[]) {

/\* Set the Nimbus look and feel \*/

//<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">

/\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.

\* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html

\*/

try {

for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {

if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());

break;

}

}

} catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(WhiteBoardFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(WhiteBoardFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(WhiteBoardFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(WhiteBoardFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

}

//</editor-fold>

/\* Create and display the form \*/

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

new WhiteBoardFrame().setVisible(true);

}

});

}

// Variables declaration - do not modify

private javax.swing.JMenuItem aboutMenuItem;

private javax.swing.JButton boardColorButton;

private javax.swing.JButton chalkColorButton;

private javax.swing.JSpinner chalkSizeSpinner;

private javax.swing.JMenuItem clearBoardMenuItem;

private javax.swing.JPanel controlPanel;

private javax.swing.JMenu editMenu;

private javax.swing.JMenuItem exitMenuItem;

private javax.swing.JMenu fileMenu;

private javax.swing.JPanel framePanel;

private javax.swing.JMenu helpMenu;

private javax.swing.JMenuItem helpMenuItem;

private javax.swing.JMenuBar menuBar;

private javax.swing.JMenuItem newMenuItem;

private javax.swing.JMenuItem openMenuItem;

private javax.swing.JMenuItem saveAsMenuItem;

private javax.swing.JMenuItem saveMenuItem;

// End of variables declaration

}

WhiteBoard.properties

AppName = White Board

DevelopedBy = Team NGHS

DeveloperEmail = aboutus@gmail.com

AppHelp =\

- Draw onto the board using the chalk.\n\

- Change color of the chalk and board using the controls buttons.\n\

- Change the chalk size using the chalk size spinner.\n\

- Select save option from the file menu to save the drawn image.\n\

- Select clear option from the edit menu to clear the white board.

ContrastColor.java

package com.compnet.jlm.whiteboard.utils;

import java.awt.Color;

/\*\*

\*

\* @author Nikesh

\*/

/\*

\* this class provides method to calculate the contrast color for the jbutton

\* which is used to select chalk and board colors and set their color to the

\* selected colors. the calculated contrast color is set to the text within the

\* jbutton else the text dissapears ( eg. the default text color for the jbutton

\* is black and if the user selects black color the jbutton background color is

\* set to black so the text dissappers

\*/

public class ContrastColor {

public static Color getContrastColor(Color color) {

// Counting the perceptive luminance - human eye favors green color...

double a = 1 - (0.299 \* color.getRed() + 0.587 \* color.getGreen() + 0.114 \* color.getBlue()) / 255;

if (a < 0.5) {

return Color.BLACK; // bright colors - black font

} else {

return Color.WHITE; // dark colors - white font

}

}

}