# TASK1 DESCRIPTION

This program basically is an example of socket programming where text messages are exchanged between multiple clients and server. It uses multithreading process that handles different clients and their conversation. Whole program is completed by following steps:

1. First, stage layout is created server side with heading label and text area to display all messages.
2. New thread is created for new connection. And details of connection are displayed in text area.
3. Client’s request Is accepted and details like IP address and host name is collected and displayed in text area.
4. Client’s input and output data stream in created and added into client’s list.
5. Another class is created to handle each new connection that process incoming and outgoing messages.
6. Input message is collected sent by client. It contains two main information one is message and second in client’s name to which this message must be sent.
7. Then, this message is sent to that client and both the details are added to display area.
8. Server Scene is added to stage.
9. Client layout is created which mainly has following components welcome label, client label, client text field, conversation label, text area, message label, message text field and send button.
10. Then, socket server is created with data input and output stream.
11. Adding event handler to send button to send the message to client.
12. Message includes two main information, message and client’s name separated with “#”.
13. New thread is created to read the incoming message. Again, message in separated with # having message and client’s name.
14. This information is then displayed on text area with client’s name.
15. Client scene is added to stage.

# TASK1 OUTPUT

|  |  |
| --- | --- |
| **Test Data** | **Screenshot** |
| Run the server  It should start server application and server is started. |  |
| Run client 1  Client1 connects with this server. |  |
| Client 1 is added to server and displays the confirmation message |  |
| Run client 2  Client 2 connects with this server |  |
| Client 2 is added to server and details are displayed on server side |  |
| Client 1 sends message to client 2  Message: Hello, client 1 here. |  |
| Client 2 receives message from client 1 |  |
| Client 2 sends message back to client 1  Message: Hello, client 2 here |  |
| Client 1 receives the message from client 2 |  |
| Client 1 sends message to client 2  Message: How are you? |  |
| Client 2 receives message from client 1 |  |
| Client 2 sends back message to client 1  Message: I am great. How about you ? where is client 3..? Please add client 3 as well. |  |
| Client 1 receives message from client 2 |  |
| Client 1 sends message to client 2 |  |
| Client 2 receives message from client 1 |  |
| Client 3 is joins the conversation and sends message to client 1 and client 2 both.  Client 2 and client 1 both receives message from client 3. |  |
| All the conversation displayed in server side as well |  |

# TASK1 CODE

**Server.java**

**import** java.io.\*;

**import** java.net.\*;

**import** java.util.Date;

**import** java.util.StringTokenizer;

**import** java.util.Vector;

**import** javafx.application.Application;

**import** javafx.application.Platform;

**import** javafx.geometry.Insets;

**import** javafx.geometry.Pos;

**import** javafx.scene.Scene;

**import** javafx.scene.control.Button;

**import** javafx.scene.control.Label;

**import** javafx.scene.control.ScrollPane;

**import** javafx.scene.control.TextArea;

**import** javafx.scene.control.TextField;

**import** javafx.scene.layout.FlowPane;

**import** javafx.scene.layout.HBox;

**import** javafx.scene.layout.VBox;

**import** javafx.scene.text.Font;

**import** javafx.scene.text.FontPosture;

**import** javafx.scene.text.FontWeight;

**import** javafx.scene.text.TextAlignment;

**import** javafx.stage.Stage;

**public** **class** Server **extends** Application {

// creating text area to display all the messages

TextArea textArea = **new** TextArea();

// number of clients joined in conversation

**private** **int** clientNo = 0;

// Vector to store active clients

**static** Vector<HandleAClient> *clientList* = **new** Vector<>();

@Override

**public** **void** start(Stage stage)

{

// creating flowpane to store all the components

FlowPane flowpane = **new** FlowPane();

flowpane.setAlignment(Pos.***CENTER***);

flowpane.setPadding(**new** Insets(20,20,20,20));

flowpane.setHgap(25);

flowpane.setVgap(25);

// creating vertical box

VBox vbox = **new** VBox();

vbox.setAlignment(Pos.***CENTER***);

vbox.setPadding(**new** Insets(20,20,20,20));

vbox.setSpacing(20);

// creating scene and adding flowpane to it

Scene scene = **new** Scene(flowpane, 600, 480);

// creating heading label

Label welcomeLabel = **new** Label("Server Application");

welcomeLabel.setFont(Font.*font*(***STYLESHEET\_CASPIAN***, FontWeight.***BOLD***, FontPosture.***REGULAR***, 22));

// setting properties to text area

textArea.setMinWidth(500);

textArea.setMinHeight(350);

textArea.setPadding(**new** Insets(10,10,10,10));

textArea.setEditable(**false**);

// creating scroll pane to store text area

ScrollPane container = **new** ScrollPane();

container.setMaxWidth(500);

container.setMinHeight(350);

container.setContent(textArea);

// creates new thread for new connection

**new** Thread( () ->

{

**try**

{

// Create a server socket

ServerSocket serverSocket = **new** ServerSocket(8000);

textArea.appendText("MultiThreadServer started at " + **new** Date() + '\n');

**while** (**true**)

{

// looks for a new connection request

Socket socket = serverSocket.accept();

// increase client number

clientNo++;

Platform.*runLater*( () ->

{

// display thread to client status

textArea.appendText("Assigning and Starting new thread for client " + clientNo + " at " + **new** Date() + '\n');

// looks for client's host name, and IP address

InetAddress inetAddress = socket.getInetAddress();

textArea.appendText("Client " + clientNo + "'s host name is " + inetAddress.getHostName() + "\n");

textArea.appendText("Client " + clientNo + "'s IP Address is " + inetAddress.getHostAddress() + "\n");

});

// Create data input and output streams

DataInputStream inputFromClient = **new** DataInputStream(socket.getInputStream());

DataOutputStream outputToClient = **new** DataOutputStream(socket.getOutputStream());

// Create and start a new thread for the connection

HandleAClient handleAClient = **new** HandleAClient(socket, "Client " + clientNo, inputFromClient, outputToClient);

*clientList*.add(handleAClient); // adds new client to client s list

**new** Thread(handleAClient).start();

}

}

**catch**(IOException ex)

{

System.***err***.println(ex);

}

}).start();

// adds components to flowpane

flowpane.getChildren().add(welcomeLabel);

flowpane.getChildren().add(container);

// sets scene in stage

stage.setTitle("Server Application");

stage.setScene(scene); // adding scene to insert stage

stage.setResizable(**false**);

stage.show();

}

// Define the thread class for handling new connection

**class** HandleAClient **implements** Runnable

{

// declaring default variables

**private** Socket socket;

**private** String name;

DataInputStream inputFromClient;

DataOutputStream outputToClient;

// Construct a thread

**public** HandleAClient(Socket socket, String name, DataInputStream inputFromClient, DataOutputStream outputToClient)

{

**this**.socket = socket;

**this**.name = name;

**this**.inputFromClient = inputFromClient;

**this**.outputToClient = outputToClient;

}

// Run a thread

**public** **void** run()

{

**try**

{

// using while loop to continuously serve the client

**while** (**true**)

{

// receive message from the client

String received = inputFromClient.readUTF();

// separates whole message from #

StringTokenizer stringTokenizer = **new** StringTokenizer(received, "#");

**if**(stringTokenizer.hasMoreTokens())

{

// stores message and client's name in variables

String messageToSend = stringTokenizer.nextToken();

String recipient = stringTokenizer.nextToken();

// handles individual client from client's list

**for** (HandleAClient handler : Server.*clientList*)

{

// if the recipient is found, write on its

// output stream

**if** (handler.name.equals(recipient))

{

// writes message back to specific client

handler.outputToClient.writeUTF(messageToSend+"#"+**this**.name);

**break**;

}

}

Platform.*runLater*(() ->

{

// adds message to text area

textArea.appendText("Message received from " + **this**.name + " : " + messageToSend + '\n');

});

}

}

}

**catch**(IOException ex)

{

ex.printStackTrace();

}

}

}

**public** **static** **void** main(String[] args)

{

*launch*(args);

}

}

**Client.java**

**import** java.io.DataInputStream;

**import** java.io.DataOutputStream;

**import** java.io.IOException;

**import** java.net.Socket;

**import** java.util.StringTokenizer;

**import** javafx.application.Application;

**import** javafx.geometry.Insets;

**import** javafx.geometry.Pos;

**import** javafx.scene.Scene;

**import** javafx.scene.control.Button;

**import** javafx.scene.control.Label;

**import** javafx.scene.control.ScrollPane;

**import** javafx.scene.control.TextArea;

**import** javafx.scene.control.TextField;

**import** javafx.scene.layout.FlowPane;

**import** javafx.scene.layout.HBox;

**import** javafx.scene.layout.VBox;

**import** javafx.scene.text.Font;

**import** javafx.scene.text.FontPosture;

**import** javafx.scene.text.FontWeight;

**import** javafx.scene.text.TextAlignment;

**import** javafx.stage.Stage;

**public** **class** Client **extends** Application {

// declaring input and output stream

DataOutputStream toServer = **null**;

DataInputStream fromServer = **null**;

**public** **void** start(Stage stage)

{

// creating flowpane to store all the components with default properties

FlowPane flowpane = **new** FlowPane();

flowpane.setAlignment(Pos.***CENTER***);

flowpane.setPadding(**new** Insets(20,20,20,20));

flowpane.setHgap(25);

flowpane.setVgap(25);

// creating vertical box with default properties

VBox vbox = **new** VBox();

vbox.setAlignment(Pos.***CENTER***);

vbox.setPadding(**new** Insets(20,20,20,20));

vbox.setSpacing(20);

// creating scene with default properties and adding flowpane to it

Scene scene = **new** Scene(flowpane, 550, 580);

// creating heading label with default properties

Label welcomeLabel = **new** Label("Chat Application");

welcomeLabel.setFont(Font.*font*(***STYLESHEET\_CASPIAN***, FontWeight.***BOLD***, FontPosture.***REGULAR***, 22));

// creating recipient name label with default properties

Label clientNameLabel = **new** Label("Recipient Name \n(Client 1, Client 2....Client n)");

clientNameLabel.setFont(Font.*font*(***STYLESHEET\_CASPIAN***, FontWeight.***LIGHT***, FontPosture.***REGULAR***, 15));

clientNameLabel.setMinWidth(450);

clientNameLabel.setTextAlignment(TextAlignment.***LEFT***);

// creating textfield for recipient name

TextField clientNameField = **new** TextField();

clientNameField.setMinWidth(450);

clientNameField.setPadding(**new** Insets(10,10,10,10));

// creating conversation label

Label conversationLabel = **new** Label("Conversation");

conversationLabel.setFont(Font.*font*(***STYLESHEET\_CASPIAN***, FontWeight.***LIGHT***, FontPosture.***REGULAR***, 15));

conversationLabel.setMinWidth(450);

conversationLabel.setTextAlignment(TextAlignment.***LEFT***);

// creating text area to display all the conversation details and is kept not editable

TextArea chatBox = **new** TextArea();

chatBox.setMinWidth(400);

chatBox.setPadding(**new** Insets(10,10,10,10));

chatBox.setEditable(**false**);

// creating scroll pane to set text area in it

ScrollPane container = **new** ScrollPane();

container.setMaxWidth(450);

container.setContent(chatBox);

// creating horizontal box to set components

HBox hbox = **new** HBox();

hbox.setAlignment(Pos.***CENTER***);

hbox.setSpacing(20);

hbox.minWidth(450);

// creating message label

Label chatMessageLabel = **new** Label("Message");

chatMessageLabel.setFont(Font.*font*(***STYLESHEET\_CASPIAN***, FontWeight.***LIGHT***, FontPosture.***REGULAR***, 15));

chatMessageLabel.setMinWidth(450);

chatMessageLabel.setTextAlignment(TextAlignment.***LEFT***);

// creating message text field with default properties

TextField chatMessageField = **new** TextField();

chatMessageField.setMinWidth(350);

chatMessageField.setPadding(**new** Insets(10,10,10,10));

// setting insert and back button

Button sendButton = **new** Button("Send");

sendButton.setMinWidth(80);

sendButton.setPadding(**new** Insets(10,10,10,10));

sendButton.setStyle(

"-fx-background-color: linear-gradient(#61a2b1, #2A5058);" +

"-fx-background-radius: 30;" +

"-fx-background-insets: 0;"+

"-fx-text-fill: white;"+

"-fx-effect: dropshadow( three-pass-box , rgba(0,0,0,0.6) , 5, 0.0 , 0 , 1 );"

);

// using try catch block to connect to the server and creating input and out data stream

**try**

{

// Create a socket to connect to the server

Socket socket = **new** Socket("localhost", 8000);

// Create an input stream to receive data from the server

fromServer = **new** DataInputStream(socket.getInputStream());

// Create an output stream to send data to the server

toServer = **new** DataOutputStream(socket.getOutputStream());

}

**catch** (IOException ex)

{

// writes error message in text area

chatBox.appendText(ex.toString() + '\n');

}

// adding event handler to send button to send message to server

sendButton.setOnAction(e ->

{

// using try catch block to handle error while writing message

**try**

{

// adds text message to text area

chatBox.appendText(chatMessageField.getText() + "\n");

// Send message to the server and clears the field

toServer.writeUTF(chatMessageField.getText() + "#" + clientNameField.getText());

toServer.flush();

chatMessageField.clear();

}

**catch** (IOException ex)

{

System.***err***.println(ex);

}

});

// creating read message thread

Thread readMessage = **new** Thread(**new** Runnable()

{

@Override

**public** **void** run()

{

**while** (**true**)

{

**try**

{

// reads message sent to this client

String message = fromServer.readUTF();

// separates message from # to separate message and client name

StringTokenizer stringTokenizer = **new** StringTokenizer(message, "#");

**if**(stringTokenizer.hasMoreTokens())

{

// stores message and client name in variables

String messageReceived = stringTokenizer.nextToken();

String recipient = stringTokenizer.nextToken();

chatBox.appendText(recipient + " : " + messageReceived + '\n');

}

}

**catch** (IOException e)

{

e.printStackTrace();

}

}

}

});

readMessage.start();

// adds message text field and send button to hbox

hbox.getChildren().add(chatMessageField);

hbox.getChildren().add(sendButton);

// adding all the components to flowpane

flowpane.getChildren().add(welcomeLabel);

flowpane.getChildren().add(clientNameLabel);

flowpane.getChildren().add(clientNameField);

flowpane.getChildren().add(conversationLabel);

flowpane.getChildren().add(container);

flowpane.getChildren().add(chatMessageLabel);

flowpane.getChildren().add(hbox);

// setting scene to stage with default properties

stage.setTitle("Chat Application");

stage.setScene(scene); // adding scene to insert stage

stage.setResizable(**false**);

stage.show();

}

**public** **static** **void** main(String[] args)

{

*launch*(args);

}

}