1. **Formulating the Problem**

**1.1 Problem Description**

Design and implement a server and client chat application using Java. The program should create a client and a server and connect them to communicate with each other. The GUI should be implemented using JavaFX. The server and the client should have a text area to display the incoming messages and a text field to enter a message. Use the enter key to send messages.

**1.2** **Verbalization**

What is the goal?

Build a server/ client chat using Java.

What are the givens?

The port number and server host.

What are the unknowns?

Connection between the server and client.

**1.3 Information Elicitation**

Goal Server/Client Chat Application

Givens Port Number, Server Host

Unknowns Connection

Condition Connect the client to the server if the port numbers match.

**2. Planning the Solution**

**2.1. Solution Strategy**

Create the server and client using the same port number. Wait for the server to accept the connection. Once a connection has been established, use an input and output method of the client class or the server class to send and receive methods from the server and the client.

**2.2 Goal Decomposition**

Sub-goal 1

Create the GUI.

Sub-goal 2

Create the server.

Sub-goal 3

Create the client.

Sub-goal 4

Allow the server to accept the request.

Sub-goal 5

Send the message to either the client of the server.

Sub-goal 6

Display the message .

**2.4 Data Organization and Description**

**Inputs**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Origin** | **Used in Sub-goal #** |
| socket | Client socket | ChatClient & ChatServer | 2 & 3 |
| input | Client & server input reader | ChatClient & ChatServer | 2 & 3 |
| message | Message to output to the server/client | User | 5 |
| server | Server host name | ChatClient & ChatServer | 2 & 3 |
| port | Port number | ChatClient & ChatServer | 2 & 3 |

**Output**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Origin** | **Used in Sub-goal #** |
| output | Client & server output writer | ChatClient & ChatServer | 6 |

**3. Designing the Solution**

**3.1 Structure Chart**

First Level Decomposition

Goal Refinement

**Sub-goal 1**

Create the GUI.

**Sub-goal 1.1**

Create the server stage.

**Sub-goal 1.2**

Create the client stage.

**Sub-goal 2**

Create the server.

**Sub-goal 3**

Create the client.

**Sub-goal 3.1**

Connect to the server.

**Sub-goal 4**

Wait for the server to accept the client request.

**Sub-goal 5**

Get input from the user.

**Sub-goal 6**

Send the input to the output stream.

**Sub-goal 6.1**

If the input is from the server, send the user input to the server output stream.

**Sub-goal 6.2**

If the input is from the client, send the user input to the client output stream.

**Sub-goal 7**

Get the output from the input stream.

**Sub-goal 7.1**

Display the output message from the input stream of the server on the client stage.

**Sub-goal 7.2**

Display the output message from the input stream of the client on the server stage.

Second Level Decomposition

**3.2 Module and Data Specifications**

**Name:** outputToServer - Send the input message to the server.

**Input:** Message.

**Output:** None.

**Logic:** Get the input message from the method parameter and send that message to the output stream of the client.

**Name:** inputFromServer- Get the input from the server.

**Input:** None.

**Output:** Message.

**Logic:** Read the message from the input stream of the client.

**Name:** acceptConn- Add a listener to the server socket for incoming connections .

**Input:** None.

**Output:** None.

**Logic:** Use the accept method to listen for incoming connections. If the port numbers match, make the connection.

**Name:** outputToClient - Send the input message to the client.

**Input:** Message.

**Output:** None.

**Logic:** Get the input message from the method parameter and send that message to the output stream of the server.

**Name:** inputFromClient- Get the input from the client.

**Input:** None.

**Output:** Message.

**Logic:** Read the message from the input stream of the server.

**Data:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Structure** |
| **socket** | **Socket** | **Object** |
| **input** | **BufferedReader** | **Object** |
| **output** | **PrintWriter** | **Object** |
| **message** | **String** | **Variable** |
| **server** | **String** | **Variable** |
| **port** | **Integer** | **Variable** |
| **server** | **ServerSocket** | **Object** |

**3.3 Algorithm**

Logic

1.0 Create the GUI.

1.1 Create the server stage.

1.2 Create the client stage.

2.0 Create the server.

3.0 Create the client.

4.0 Wait for the server to accept the client request.

5.0 Get input from the user.

6.0 Send the input to the output stream.

6.1 If the input is from the server, send the input to the server output stream.

6.2 If the input is from the client, send the input to the client output stream.

7.0 Get the output from the input stream.

7.1 Display the output message from the input stream of the server on the client stage.

7.2 Display the output message from the input stream of the client on the server stage.

Algorithm Description

The program starts off by displaying the client and server windows. In the background, a server object and a client object are instantiated to create the server and client. Once the

server and client are created, the server accepts of decline the clients connection request. If the server has accepted the request, the client can now communicate with the server and the server can communicate back to the client. The enter key is used as a trigger to send and receive messages between the server and the client. Once the user presses the enter key, the message from the input text field is send to the server or the client output stream by using the outputToServer or outputToClient method. To display the messages that have been send, the client or the server uses the inputFromServer or inputFromClient method to read the input stream and return the message from the input stream.

**4 Translation**

**4.1 Source Code**

//=============================================================

// Name : Nidhi Patel

// SID : 31379144

// Course : IT114

// Section : 452

// Instructor : Maura Deek

// Assignment # : Programming Assignment 5

// Date : 04/22/2016

// Description : This is the client class for a

// a client/server chat application

//=============================================================

import java.io.\*;

import java.net.\*;

public class ChatClient

{

private Socket socket;

private BufferedReader input;

private PrintWriter output;

String message,

server;

int port;

public ChatClient(String serverHost, int portNum)

{

try

{

server = serverHost;

port = portNum;

socket = new Socket(server, port);

}

catch(IOException ex)

{

System.out.println(ex);

}

}

public void outputToServer(String message)

{

try

{

output = new PrintWriter(socket.getOutputStream(), true);

output.println(message);

}

catch(IOException ex)

{

System.out.println(ex);

}

}

public String inputFromServer()

{

try

{

input = new BufferedReader(new InputStreamReader(socket.getInputStream()));

message = input.readLine();

}

catch(IOException ex)

{

System.out.println(ex);

}

return message;

}

}

//=============================================================

// Name : Nidhi Patel

// SID : 31379144

// Course : IT114

// Section : 452

// Instructor : Maura Deek

// Assignment # : Programming Assignment 5

// Date : 04/22/2016

// Description : This is the server class for a

// a client/server chat application

//==============================================================

import java.io.\*;

import java.net.\*;

public class ChatServer

{

private ServerSocket server;

private Socket socket;

private BufferedReader input;

private PrintWriter output;

String message;

int port;

public ChatServer(int portNum)

{

try

{

port = portNum;

server = new ServerSocket(port);

}

catch(IOException ex)

{

System.out.println(ex);

}

}

public void acceptConn()

{

try{

socket = server.accept();

}

catch(IOException ex)

{

System.out.println(ex);

}

}

public String inputFromClient()

{

try

{

input = new BufferedReader(new InputStreamReader(socket.getInputStream()));

message = input.readLine();

}

catch(IOException ex)

{

System.out.println(ex);

}

return message;

}

public String outputToClient(String message)

{

try

{

output = new PrintWriter(socket.getOutputStream(), true);

output.println(message);

}

catch(IOException ex)

{

System.out.println(ex);

}

return message;

}

}

//==============================================================

// Name : Nidhi Patel

// SID : 31379144

// Course : IT114

// Section : 452

// Instructor : Maura Deek

// Assignment # : Programming Assignment 5

// Date : 04/22/2016

// Description : This program is a client/server chat application

//==============================================================

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.stage.Stage;

import javafx.scene.control.TextField;

import javafx.scene.control.TextArea;

import javafx.scene.layout.GridPane;

import javafx.scene.layout.ColumnConstraints;

import javafx.geometry.Insets;

import javafx.event.EventHandler;

import javafx.scene.input.KeyEvent;

import javafx.scene.input.KeyCode;

import java.io.\*;

import java.net.\*;

public class ChatApp extends Application

{

public static void main(String[]args)

{

launch(args);

}

public void start(Stage primaryStage) throws Exception

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SERVER STAGE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Stage serverStage = new Stage();

serverStage.setTitle("Sever");

serverStage.setResizable(false);

GridPane serverLayout = new GridPane();

serverLayout.setPadding(new Insets(15));

serverLayout.setHgap(5);

serverLayout.setVgap(5);

serverLayout.getColumnConstraints().add(new ColumnConstraints(20));

serverLayout.getColumnConstraints().add(new ColumnConstraints(20));

TextArea serverDisplay = new TextArea();

serverDisplay.setEditable(false);

TextField serverInput = new TextField();

serverInput.setEditable(true);

serverLayout.add(serverDisplay,0,0,3,1);

serverLayout.add(serverInput,0,1,3,1);

Scene server = new Scene(serverLayout);

serverStage.setScene(server);

serverStage.show();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CLIENT STAGE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Stage clientStage = new Stage();

clientStage.setTitle("Client");

clientStage.setResizable(false);

GridPane clientLayout = new GridPane();

clientLayout.setPadding(new Insets(15));

clientLayout.setHgap(5);

clientLayout.setVgap(5);

clientLayout.getColumnConstraints().add(new ColumnConstraints(20));

clientLayout.getColumnConstraints().add(new ColumnConstraints(20));

TextArea clientDisplay = new TextArea();

clientDisplay.setEditable(false);

TextField clientInput = new TextField();

clientInput.setEditable(true);

clientLayout.add(clientDisplay,0,0,3,1);

clientLayout.add(clientInput,0,1,3,1);

Scene client = new Scene(clientLayout);

clientStage.setScene(client);

clientStage.show();

ChatServer newServer = new ChatServer(8000);

ChatClient newClient= new ChatClient("localhost",8000);

newServer.acceptConn();

serverInput.setOnKeyPressed(new EventHandler<KeyEvent>()

{

public void handle(KeyEvent keyPressed){

if(keyPressed.getCode() == KeyCode.ENTER){

newServer.outputToClient(serverInput.getText());

serverDisplay.appendText("\nMe: " + serverInput.getText());

clientDisplay.appendText("\nServer: " + newClient.inputFromServer());

serverInput.setText("");

}

}

});

clientInput.setOnKeyPressed(new EventHandler<KeyEvent>()

{

public void handle(KeyEvent keyPressed){

if(keyPressed.getCode() == KeyCode.ENTER){

newClient.outputToServer(clientInput.getText());

clientDisplay.appendText("\nMe: " + clientInput.getText());

serverDisplay.appendText("\nClient: " + newServer.inputFromClient());

clientInput.setText("");

}

}

});

}

}

**4.2 Program and Module Description**

**outputToServer -** Send the input message to the server.

**inputFromServer -** Get the input from the server.

**acceptConn -**  Adds a listener to the server socket to handle incoming connections.

**outputToClient-** Send the input message to the client.

**inputFromClient -** Get the input from the client.

**main -** Launch the program.

**5. Solution Testing**

**Test Case 1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User** | **Input** | **Action** | **Client Output** | **Server Output** |
| Client | Hi! | Enter | Me: Hi! | Client: Hi! |
| Server | Hi, how are you? | Enter | Me: Hi!  Server: Hi, how are you? | Client: Hi!  Me: Hi, how are you? |
| Client | I am great! How about you? | Enter | Me: Hi!  Server: Hi, how are you?  Me: I am great! How about you? | Client: Hi!  Me: Hi, how are you?  Client: I am great! How about you? |
| Server | I am fine, just busy with work. | Enter | Me: Hi!  Server: Hi, how are you?  Me: I am great! How about you?  Server: I am fine, just busy with work. | Client: Hi!  Me: Hi, how are you?  Client: I am great! How about you?  Me: I am fine, just busy with work. |

**6 Testing Output**

**Test Case 1**







