Real-time Charting Box Using Java Sockets

This project demonstrates a real-time chat application that facilitates communication between a client and a server using sockets. The graphical user interface (GUI) is created with Swing, providing an interactive experience for users to send and messages in real-time. It showcases the fundamentals of programming in Java, combined with a simple yet effective GUI, it a valuable learning resource for understanding client-server communication.

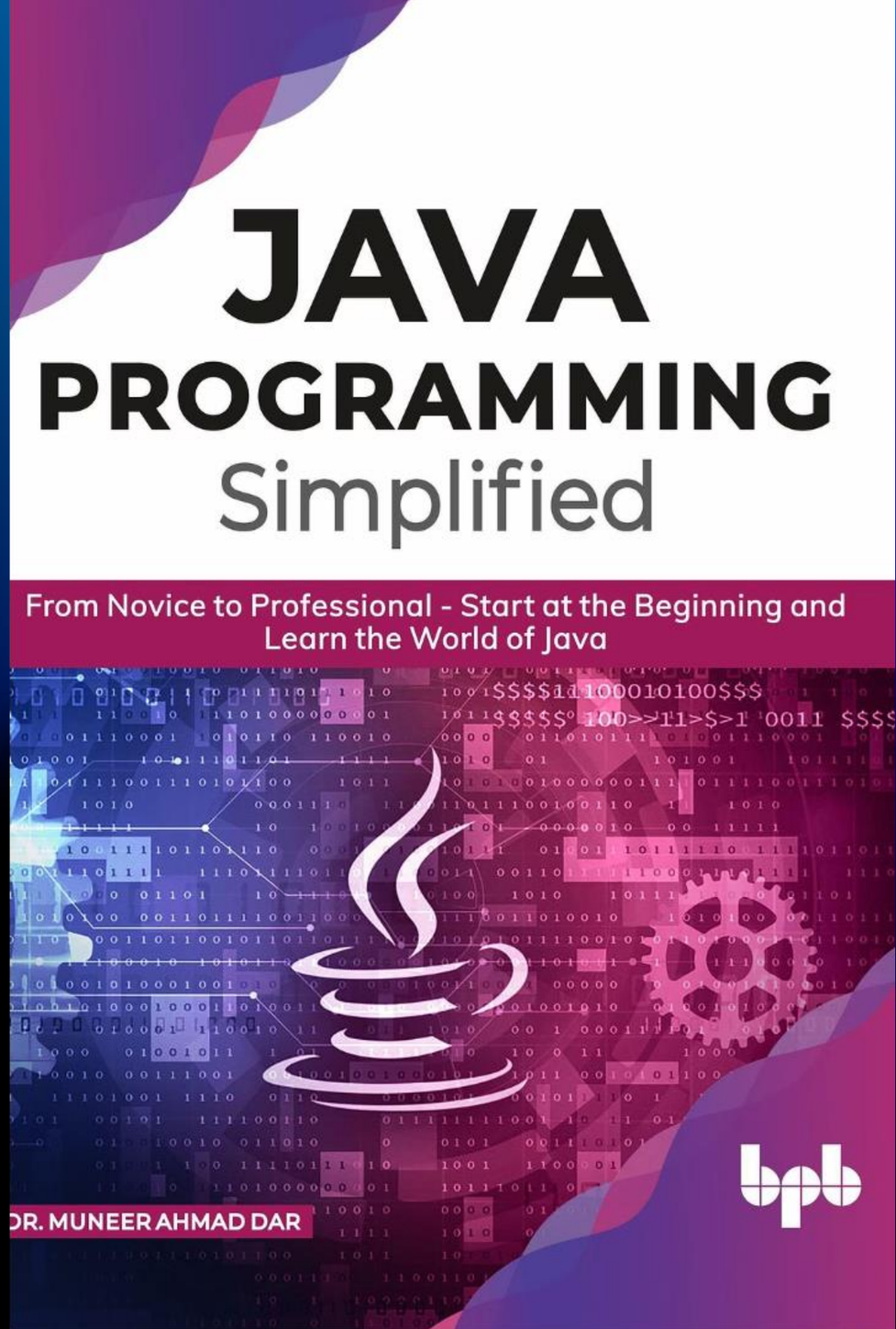
BY

Name: Bhola Yadav

USN:1CR23CS044

SECTION: K

Branch: Computer Science Engineering



Key Libraries and Imports

Swing Components

The project utilizes the javax.swing.* library to create the GUI components, such as the chat window, text input, and display areas.

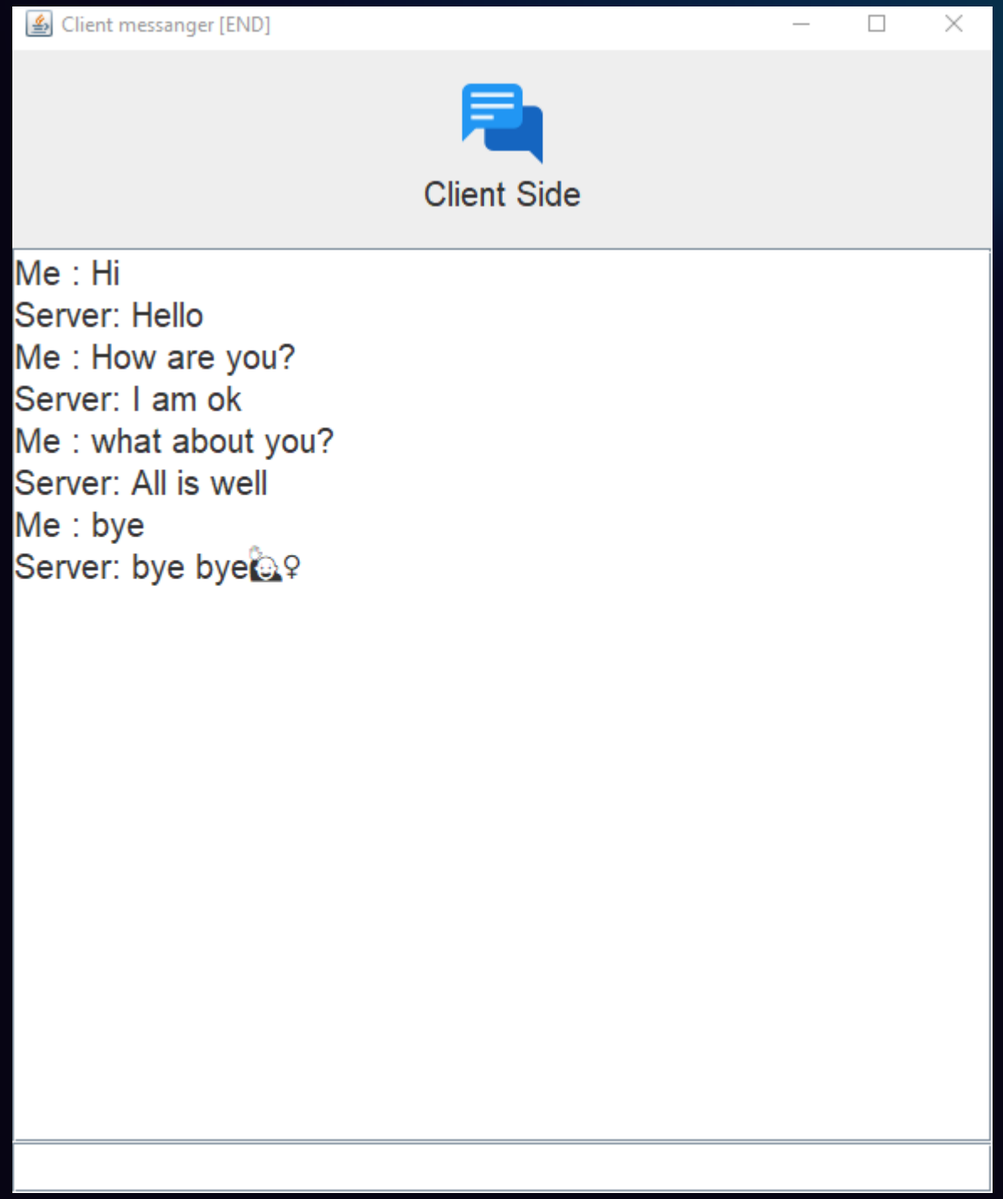
Networking Capabilities

The java.net.* library is used to enable socket communication, allowing the client and server to establish a connection and exchange messages.

Input/Output Operations

The java.io.* library is used for handling input and output operations, such as reading and writing data through the socket connection.

Client-Side Implementation



GUI Setup

The client-side implementation utilizes Swing to create a chat window with text input and display areas, providing a user-friendly interface for the application.

Socket Communication

The client establishes a connection to the server using Java sockets, enabling the exchange of messages between the user and the server.

Event Handling

The application captures user input via keyboard events and sends the messages to the server for broadcasting to other connected clients.

Client-Side Program

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.io.*;
import java.net.*;
import java.awt.BorderLayout;
import static javax.swing.SwingConstants.CENTER;
public class Client extends JFrame{
    Socket socket;
    BufferedReader br;
    PrintWriter out;
    private JLabel heading=new JLabel("Client Side");
    private JTextArea messageArea=new JTextArea();
    private JTextField messageInput=new JTextField();
    private Font font=new Font("Roboto", Font.PLAIN,20);
```

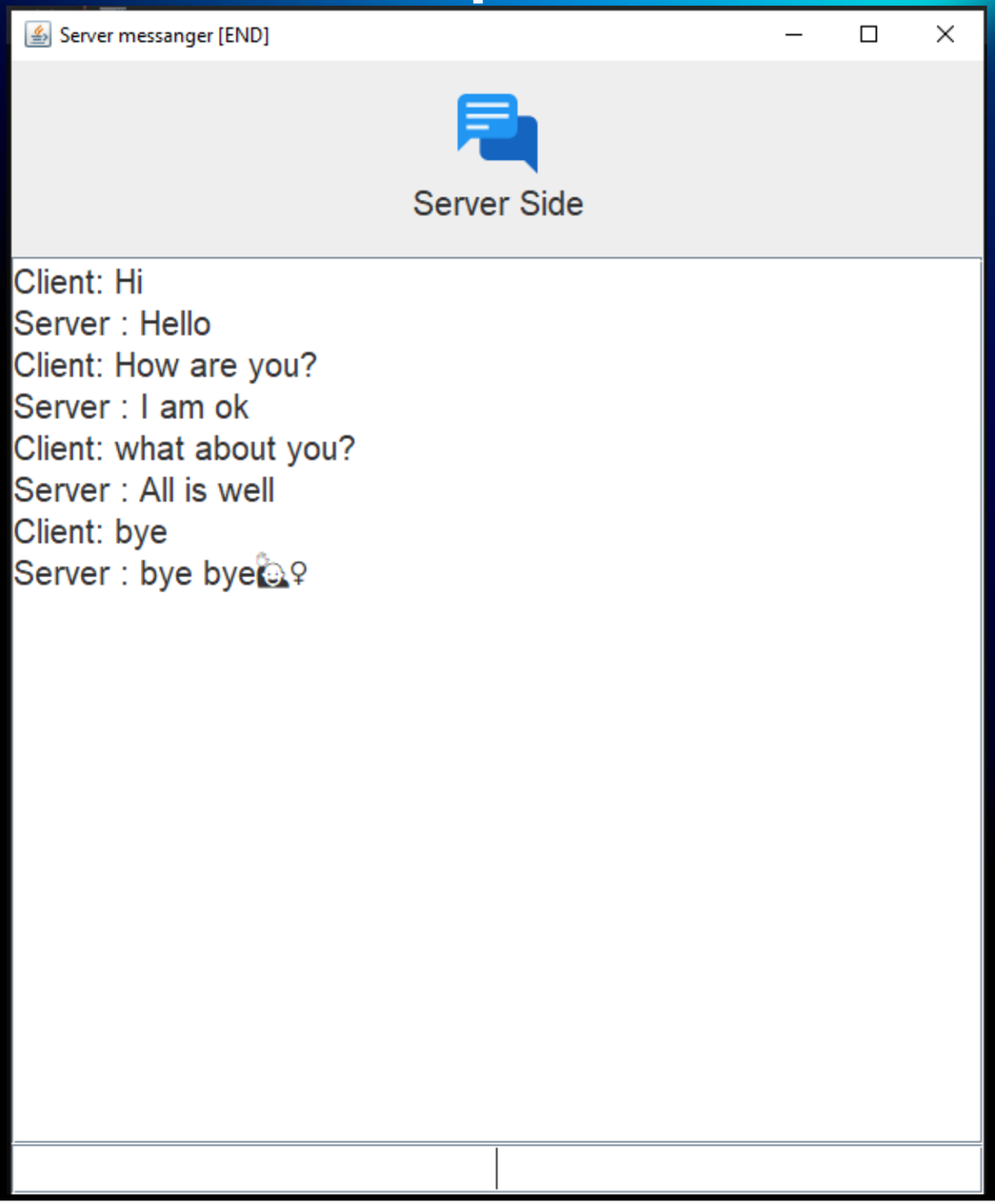
```
    //Constuctor
    public Client(){
        try{
            System.out.println("Sending Request to server");
            socket=new Socket("10.201.13.186",8080);
            System.out.println("Connection done");
            br=new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            out=new PrintWriter(socket.getOutputStream());

            createGUI();
            handelEvents();
            startReading();
//          startWriting();

        }catch (Exception e){
            e.printStackTrace();
        }
    }
```

```java
private void handelEvents() {
    messageInput.addKeyListener(new KeyListener() {
        @Override
        public void keyTyped(KeyEvent e) {

        }

        @Override
        public void keyPressed(KeyEvent e) {

        }

        @Override
        public void keyReleased(KeyEvent e) {
            //System.out.println(STR."Key released: \{e.getKeyCode()}");
            if(e.getKeyCode()==10){
                //System.out.println("You have pressed Enter Button");
                String contentToSend=messageInput.getText();
                messageArea.append(("Me : "+contentToSend+"\n"));
                out.println(contentToSend);
                out.flush();
                messageInput.setText("");
                messageInput.requestFocus();
            }

        }
    });
}
```

```java
private void createGUI(){
    //Gui Code...
    this.setTitle("Client messanger [END]");
    this.setSize(600,720);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    //Coding for component..
    heading.setFont(font);
    messageArea.setFont(font);
    messageInput.setFont(font);
    heading.setIcon(new
ImageIcon("F:\\javaproject\\SecureChamber\\src\\clogo.png"));
    heading.setHorizontalTextPosition(CENTER);
    heading.setVerticalTextPosition(SwingConstants.BOTTOM);
    heading.setHorizontalAlignment(CENTER);
    heading.setBorder(BorderFactory.createEmptyBorder(20,20,20,20));
    messageArea.setEditable(false);
    messageInput.setHorizontalAlignment(CENTER);
    //Setting Frame Layout
    this.setLayout(new BorderLayout());
    //Adding components
    this.add(heading,BorderLayout.NORTH);
    JScrollPane jScrollPane=new JScrollPane(messageArea);
    this.add(jScrollPane,BorderLayout.CENTER);
    this.add(messageInput,BorderLayout.SOUTH);
    this.setVisible(true);

}
```

```
//Start reading
    public void startReading() {
        // thread-read karkre data rahega
        Runnable r1 = () -> {
            System.out.println("reader started..");
            try {
                while (true) {
                    String msg = br.readLine();
                    if (msg.equals("exit")) {
                        System.out.println("Server terminated the chat");
                        JOptionPane.showMessageDialog(this,"Server Terminated the chat");
                        messageInput.setEnabled(false);
                        socket.close();
                        break;
                    }
                    // System.out.println("Server:" + msg);
                     messageArea.append("Server: "+msg+"\n");
                }
            }catch (Exception e){
                //e.printStackTrace();
                System.out.println("Connection Closed");
            }
        };
        new Thread(r1).start();
    }
/*---------- Main Method ----------------*/
    public static void main(String[] Args){
        System.out.println("I am a client");
        new Client();
    }
}
```

Server-Side Implementation



Socket Management

The server-side implementation listens for incoming client connections and manages multiple clients, ensuring efficient handling of the chat application.

Message Broadcasting

The server receives messages from clients and broadcasts them to all connected clients, enabling real-time communication between users.

Server-Side Program

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.io.*;
import java.net.*;
import static javax.swing.SwingConstants.CENTER;
public class Server extends JFrame {
    ServerSocket server;
    Socket socket;
    BufferedReader br;
    PrintWriter out;
    private JLabel heading=new JLabel("Server Side");
    private JTextArea messageArea=new JTextArea();
    private JTextField messageInput=new JTextField();
    private Font font=new Font("Roboto", Font.PLAIN,20);
    //Constructor.....
    public Server(){
        try{
            server=new ServerSocket(8080);
            System.out.println("Server is ready to accept connection");
            System.out.println("Waiting...");
            socket=server.accept();
            br=new BufferedReader(new InputStreamReader(socket.getInputStream()));
            out=new PrintWriter(socket.getOutputStream());
            createGUI();
            handelEvents();
            startReading();
//            startWriting();
        }catch (Exception e){
            e.printStackTrace();
        }
    }
    private void handelEvents(){
        messageInput.addKeyListener(new KeyListener() {
            @Override
            public void keyTyped(KeyEvent e) {
            }
            @Override
            public void keyPressed(KeyEvent e) {
            }
```

```
@Override
        public void keyReleased(KeyEvent e) {
            //System.out.println("Key released:"+e.getKeyCode());
            if(e.getKeyCode()==10){
                //System.out.println("You have pressed Enter Button");
                String contentToSend=messageInput.getText();
                messageArea.append(("Server : "+contentToSend+"\n"));
                out.println(contentToSend);
                out.flush();
                messageInput.setText("");
                messageInput.requestFocus();
            }
        }
    });
}
private void createGUI(){
    //Gui Code...
    this.setTitle("Server messanger [END]");
    this.setSize(600,720);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    //Coding for component..
    heading.setFont(font);
    messageArea.setFont(font);
    messageInput.setFont(font);
```

```
 heading.setIcon(new
ImageIcon("F:\\javaproject\\SecureChamber\\src\\clogo.png"));
    heading.setHorizontalTextPosition(CENTER);
    heading.setVerticalTextPosition(SwingConstants.BOTTOM);
    heading.setHorizontalAlignment(CENTER);
    heading.setBorder(BorderFactory.createEmptyBorder(20,20,20,20));
    messageArea.setEditable(false);
    messageInput.setHorizontalAlignment(CENTER);
  //Setting Frame Layout
    this.setLayout(new BorderLayout());
    //Adding components
    this.add(heading,BorderLayout.NORTH);
    JScrollPane jScrollPane=new JScrollPane(messageArea);
    this.add(jScrollPane,BorderLayout.CENTER);
    this.add(messageInput,BorderLayout.SOUTH);

    this.setVisible(true);
}

public void startReading() {
    // thread-read karkre data rahega
    Runnable r1 = () -> {
        System.out.println("reader started..");
        try{
            while (true) {
```

```
String msg = br.readLine();
                if (msg.equals("exit")) {
                    System.out.println("Client has terminated the chat");
                    JOptionPane.showMessageDialog(this,"Client Terminated the chat");
                    messageInput.setEnabled(false);
                    socket.close();
                    break;
                }
//                  System.out.println("Client:" + msg);
                messageArea.append("Client: "+msg+"\n");
            }
        }catch (Exception e){
            //e.printStackTrace();
            System.out.println("Connection closed");
        }
    };
    new Thread(r1).start();
  }

//----------------------writing mode only for Console----------------------
//    public void startWriting(){
//        Runnable r2=()->{
//            System.out.println("Writer Started....");
//            try{
//                while(!socket.isClosed()){
//                    BufferedReader br1=new BufferedReader(new InputStreamReader(System.in));
//                    String content=br1.readLine();
//                    out.println(content);
//                    out.flush();
//                    if(content.equals("exit")){
//                        socket.close();
//                        break;
//                    }
//                }
//            }catch (Exception e){
//                //e.printStackTrace();
//                System.out.println("Connection closed");
//            }
//            System.out.println("Connection closed");
//        };
//        new Thread(r2).start();
//    }


    public static void main(String[] Args) {
        System.out.println("i am a Server");
        new Server();
    }
}
```

Note : Mostly the commented line are used to just check the program as for console based

Key Features

1

Real-Time Messaging

Users can send and receive messages instantly, providing a seamless and responsive chat experience.

2

User-Friendly Interface

The Swing-based GUI offers an intuitive design, making the application easy to use and navigate.

3

Event-Driven Programming

The application efficiently handles user inputs and socket events, ensuring a smooth and responsive user experience.

4

Scalable Server

The server-side implementation is capable of managing multiple client connections simultaneously, allowing the chat application to scale as needed.

Networking Fundamentals

Socket Communication

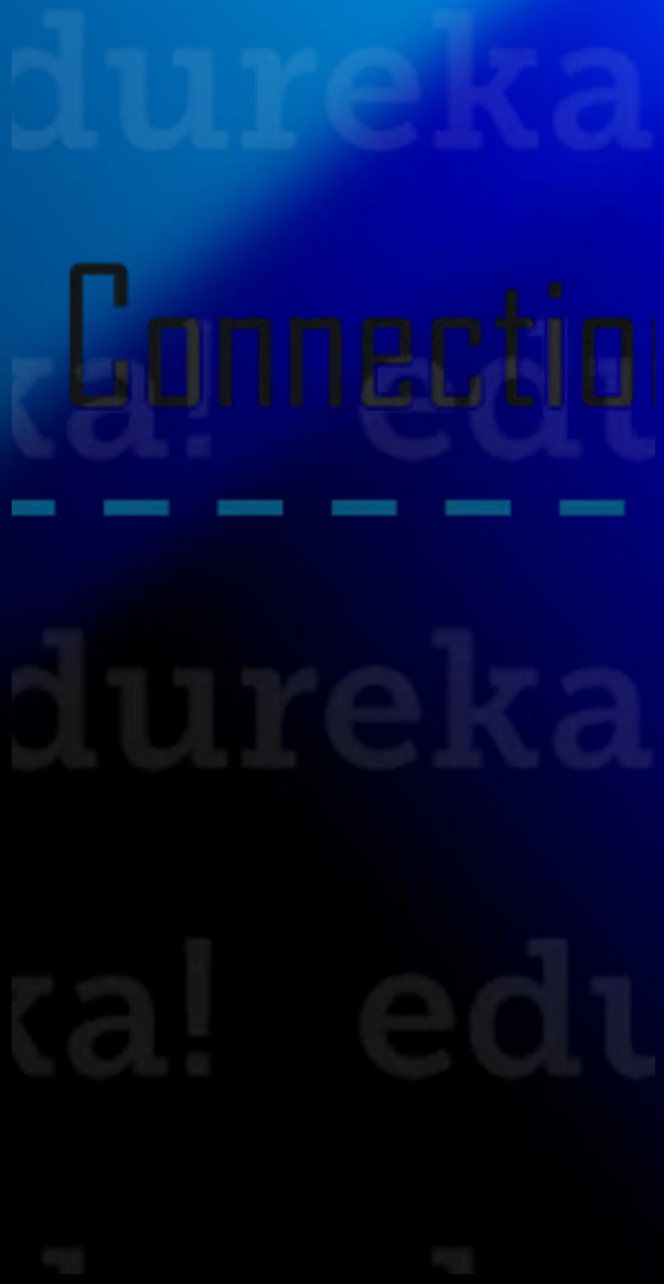
The project demonstrates the use of Java sockets to establish a connection between the client and the server, enabling the exchange of messages in real-time.

Client-Server Architecture

The application follows a client-server architecture, where the client initiates the connection and the server manages the communication with multiple clients.

Message Handling

The server-side implementation is responsible for receiving messages from clients and broadcasting them to all connected clients, ensuring seamless communication.



Learning Opportunities

Network Programming

The project provides a practical example of how to build a real-time chat application using sockets, allowing learners to understand client-server communication.

GUI Development

The use of Swing for creating the graphical user interface offers an opportunity to learn about building interactive applications in Java.

Event-Driven Programming

The efficient handling of user inputs and socket events demonstrates the principles of event-driven programming, a valuable skill for Java developers.

Conclusion

1

Fundamentals of Java

This project showcases the fundamentals of Java programming, including networking, GUI development, and event-driven programming.

2

Practical Application

The real-time chat application provides a practical example of how to apply these concepts to build a functional and interactive software solution.

3

Learning Resource

The project serves as a valuable learning resource for understanding client-server communication and building real-time applications in Java.

Thank You Everyone