COMPUTER NETWORKS PROJECT REPORT

GROUP MEMBERS:

K.PHANINDRA REDDY-BU22CSEN0101454

I.ABHISHEIK-BU22CSEN0101761

A.GOWTHAM VENKAT REDDY-BU22CSEN0101513

S.VAMSI KRISHNA-BU22CSEN0100600

TITLE:

Image and File Transfer Application using Java Sockets

AIM:

To transfer image from client to server using Socket programming.

INTRODUCTION:

The project's goal is to develop a basic Java socket application that allows text and image files to be transferred between a client and server. The client-side application and the server-side application are its two primary parts. Users can transfer a text or image file from their local machine to the server using the client application. Conversely, the files are received by the server application, which saves them locally.

Client Application:

- The client application is implemented in the Client class.
- It provides a graphical user interface (GUI) using AWT components.
- Users can enter the filename of the file they want to send using a TextField.
- If the file is an image (.jpg or .png), it reads the image file using FileInputStream and sends it to the server using DataOutputStream.
- If the file is a text file, it reads the file line by line using a BufferedReader and sends each line to the server.

Source Code:

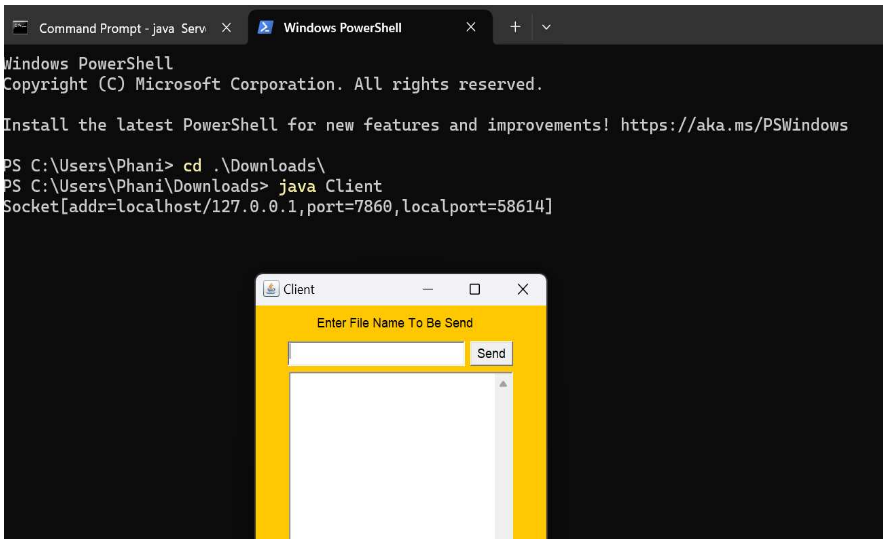
Client Program:

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;

public class Client implements ActionListener {

    private Socket s;
    private DataInputStream din;
    private DataOutputStreamdout;
    private TextFieldtf;
    private TextArea ta;
    private Label lb;
    private Button b;

    public Client() {
        Frame f = new Frame("Client");
f.setLayout(new FlowLayout());
f.setBackground(Color.orange);
tf = new TextField(15);
        ta = new TextArea(12, 20);
```

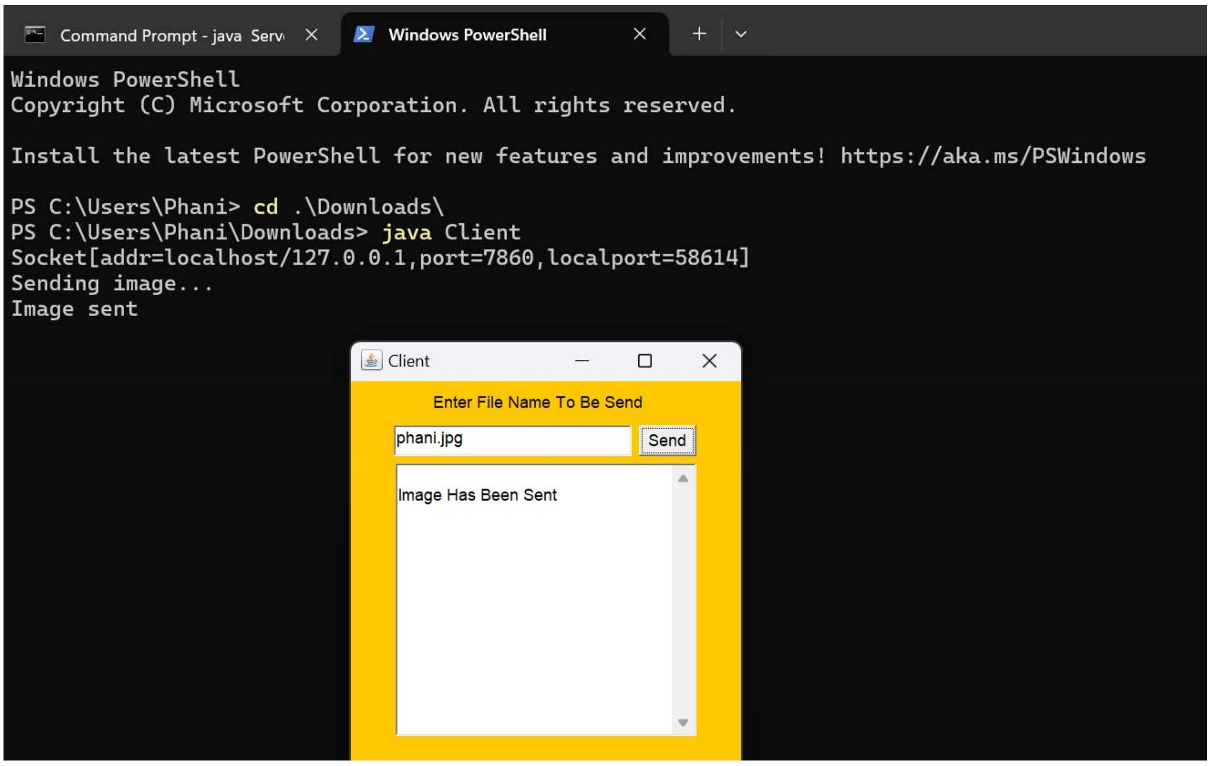
```
ta.setBackground(Color.white);
        lb = new Label("Enter File Name To Be Send");
        b = new Button("Send");
f.add(lb);
f.add(tf);
f.add(b);
f.add(ta);
b.addActionListener(this);
f.setSize(300, 400);
f.setLocation(300, 300);
f.setVisible(true);

        try {
            s = new Socket("localhost", 7860);
System.out.println(s);
            din = new DataInputStream(s.getInputStream());
dout = new DataOutputStream(s.getOutputStream());
        } catch (Exception e) {
System.out.println(e);
        }

f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
```

```
            try {
s.close();
            } catch (IOException e) {
e.printStackTrace();
            }
System.exit(0);
          }
        });
    }

    public void actionPerformed(ActionEvent ae) {
        String fileName = tf.getText();

        int flag = 0;
        String extn = "";
        for (int i = 0; i<fileName.length(); i++) {
            if (fileName.charAt(i) == '.' || flag == 1) {
                flag = 1;
                extn += fileName.charAt(i);
            }
        }

        if (extn.equals(".jpg") || extn.equals(".png")) {
            try (FileInputStream fin = new FileInputStream(fileName)) {
dout.writeUTF(fileName);
System.out.println("Sending image...");
byte[] readData = new byte[1024];
                int i;
                while ((i = fin.read(readData)) != -1) {
dout.write(readData, 0, i);
                }
System.out.println("Image sent");
ta.append("\nImage Has Been Sent");
            } catch (IOException ex) {
System.out.println("Image ::" + ex);
            }
        } else {
            try (BufferedReaderbcr = new BufferedReader(new InputStreamReader(new
FileInputStream(fileName)))) {
dout.writeUTF(fileName);
System.out.println("Sending File " + fileName);
                String s1;
ta.append("\n");
                while ((s1 = bcr.readLine()) != null) {
System.out.println("" + s1);
ta.append(s1 + "\n");
dout.writeUTF(s1);

dout.flush();

Thread.currentThread().sleep(500);

                }

            } catch (Exception e) {

System.out.println("Enter Valid File Name");

            }

        }

    }


    public static void main(String[] ar) {

        new Client();

    }

}
```

OUTPUT:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Phani> cd .\Downloads\
PS C:\Users\Phani\Downloads> java Client
Socket[addr=localhost/127.0.0.1,port=7860,localport=58614]
```



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Phani> cd .\Downloads\
PS C:\Users\Phani\Downloads> java Client
Socket[addr=localhost/127.0.0.1,port=7860,localport=58614]
Sending image...
Image sent
```

Enter File Name To Be Send

phani.jpg

Image Has Been Sent

Explanation:

Step-1:GUI Configuration

The client application begins by utilizing Frame from AWT to create a GUI window. It selects FlowLayout as the layout type and orange as the backdrop color. The frame is expanded to include elements like TextField for filename entry, TextArea for message display, Label for instructions, and Button for file transmission.

Step-2: Connector Socket

It uses Socket to create a connection to the server that is running on localhost at port 7860. DataInputStream and DataOutputStream, the input and output streams, are set up to transmit and receive data via the socket.

Step-3: Taking Action

The "Send" button click by the user initiates the actionPerformed method. The

filename that the user entered is retrieved from the TextField.
To identify if a file is a text file or an image, it looks at the file extension.

Step-4: File Transmission

In the event that the file is an image (.jpg or.png), FileInputStream is used to read the image file, which is then sent in segments to the server. If the file is text, it uses a BufferedReader to read the file line by line and sends each line to the server.

Step-5: Closing of Windows

To handle the GUI window's closing event, it adds a WindowListener.
The application ends and the socket connection to the server is ended when the window is closed.

Server Application:

- The Server class contains the implementation of the server application.
- Using a ServerSocket, it uses the port 7860 to listen for incoming connections from clients.
- To manage client communication, it establishes a distinct thread (doComms) for every incoming connection.
- The Runnable interface is implemented by the doComms class to manage communication with every client.
- After reading the filename that the client gave, it uses the file extension to identify if the file is a text or picture file.
- If it's an image, FileOutputStream is used to save the image data locally after it is received via DataInputStream.
- If the file type is text, it uses DataInputStream to get the text data line by line and PrintWriter to save it to a text file.

Source Code:

Server Program:

```
import java.io.*;
import java.net.*;

class Server
{
static int i=0;
private static int maxcon=0;

   public static void main(String args[])
   {
   try
      {
ServerSocket ss;
      Socket s;

System.out.println("Server Started");
      ss=new ServerSocket(7860);

while((i++ <maxcon) || (maxcon == 0))
         {
doComms connection;
```

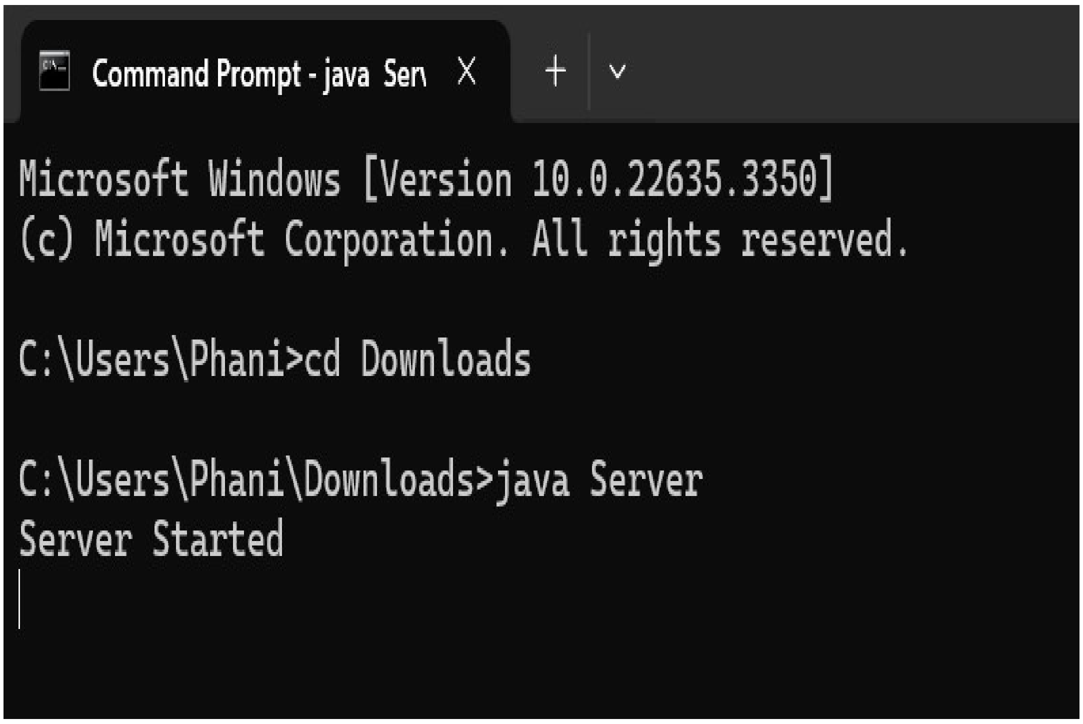
```
          s=ss.accept();
System.out.println(s);
System.out.println("Client "+i+"  Connected");
doCommsconn_c= new doComms(s);
          Thread t = new Thread(conn_c);
t.start();
          }
       } catch (IOExceptionioe) {
System.out.println("IOException on socket listen: " + ioe);
ioe.printStackTrace();
                        }


   }
}



class doComms implements Runnable
{
   private Socket s;


doComms(Socket s)
   {
this.s=s;
```

```
   }

   public void run ()
   {

      try {
      // Get input from the client
DataInputStream dis = new DataInputStream (s.getInputStream());
PrintStream out1 = new PrintStream(s.getOutputStream());

         String str,extn="";
         str=dis.readUTF();
System.out.println("\n"+str);
         int flag=0,i;

            for(i=0;i<str.length();i++)
            {

               if(str.charAt(i)=='.' || flag==1)
               {
               flag=1;
               extn+=str.charAt(i);
               }
            }



//*******reading image************//

            if(extn.equals(".jpg") || extn.equals(".png"))
              {
                File file = new File("RecievedImage"+str);
FileOutputStreamfout = new FileOutputStream(file);

                //receive and save image from client
byte[] readData = new byte[1024];
while((i = dis.read(readData)) != -1)
                {
fout.write(readData, 0, i);
                    if(flag==1)
                    {
System.out.println("Image Has Been Received");
                    flag=0;
                    }
                }
fout.flush();
fout.close();
```

```
//*********Reading Other Files*******//
                }
               else
               {
FileWriterfstream = new FileWriter("ReceivedFile"+ str);
PrintWriter out=new PrintWriter(fstream);

                   do
                   {
                   str=dis.readUTF();
System.out.println(" "+str);
out.println(str);
out.flush();
                   if(str==null)break;

}while(true);

System.out.println("One File Received");
out.close();
               }
```

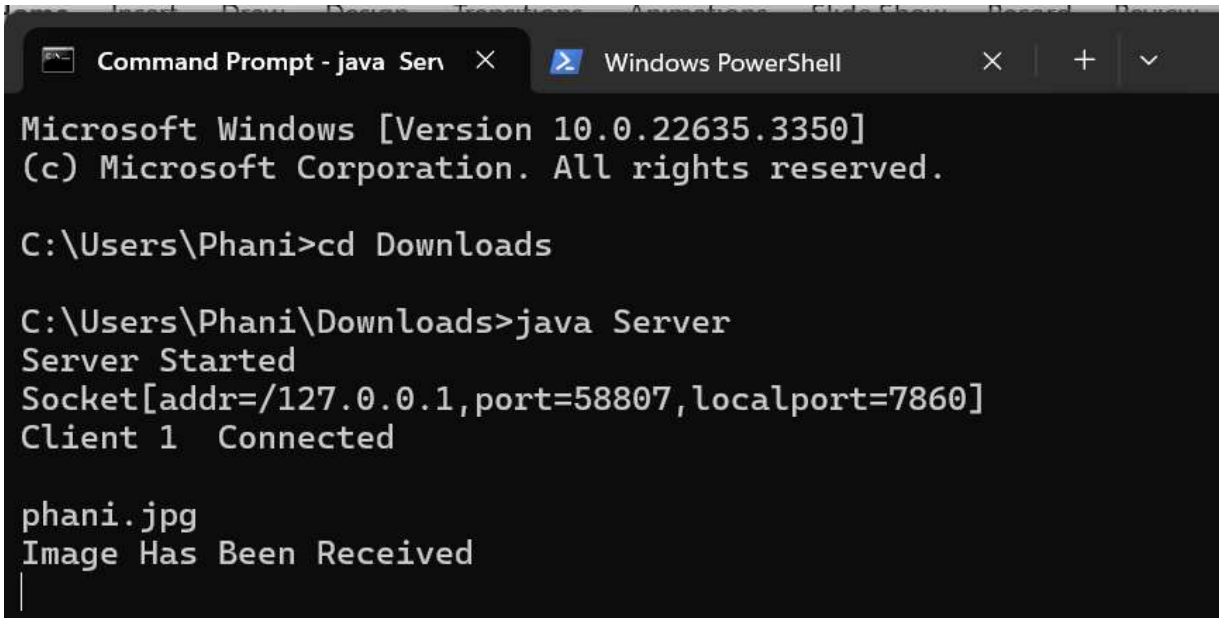
```
            } catch (IOExceptionioe) {
System.out.println("");
                              }
   }
```

OUTPUT:

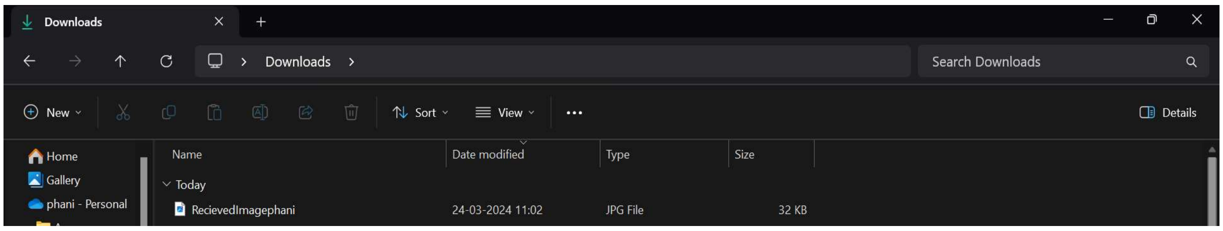


```
Microsoft Windows [Version 10.0.22635.3350]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Phani>cd Downloads

C:\Users\Phani\Downloads>java Server
Server Started
```

}



```
Microsoft Windows [Version 10.0.22635.3350]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Phani>cd Downloads

C:\Users\Phani\Downloads>java Server
Server Started
Socket[addr=/127.0.0.1,port=58807,localport=7860]
Client 1  Connected

phani.jpg
Image Has Been Received
```



Name	Date modified	Type	Size
Today			
RecievedImagephani	24-03-2024 11:02	JPG File	32 KB

Explanation:

Step-1: Configuring Sockets

In order to begin, the server program creates a ServerSocket that is listening on port 7860. It watches for connections from incoming clients.

Step-2: Client Handling

For each incoming connection, a new thread (doComms)is created to handle communication with the client.This allows the server to handle multiple clients concurrently.

Step-3: Managing Communication

The Runnable interface is implemented by the doComms class to manage communication with every client. After reading the filename that the client gave, it uses the file extension to identify if the file is a text or picture file.

Step-4: Getting Files in

If it's an image, FileOutputStream is used to save the image data locally after it is received via DataInputStream. If the file type is text, it uses DataInputStream to get the text data line by line and PrintWriter to save it to a text file.

Step-5: Error Resolution

To gracefully handle any communication problems, the server handles IOExceptions. It makes sure that even in the event of a communication mistake with a client, the server keeps listening for new connections.

CONCLUSION:

In conclusion, bi-directional communication is made possible by the client-server architecture, which enables the client to transmit files to the server and the server to receive and store them locally. The server application makes optimal use of threads to handle several client connections at once. All in all, the code offers a framework upon which to construct more complex Java file transfer programs.