To: Dr. Quan Yuan

Date: 10/04/2023

COSC 6370. NETWORKING LAB SOCKET PROGRAMMING

Saugat Pahari. Dhimant Adhikari. Nikhil Shrestha

Program Introduction: Client-Server File Transfer

The Client-Server File Transfer program demonstrates a simple file transfer mechanism between a client and a server using Java socket programming. This application allows the client to send a text file to the server, which then saves the content of the file as a new text file.

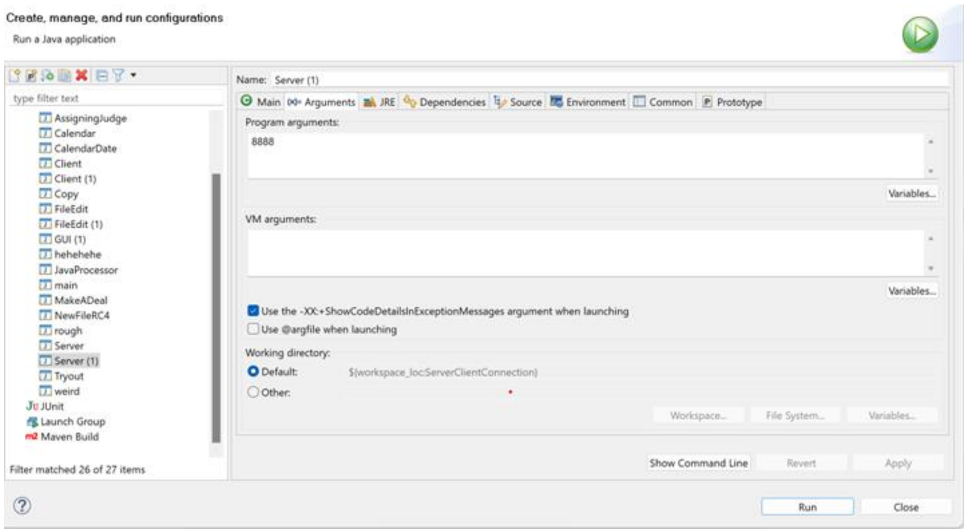
Key Features:

- **Client-Server Communication:** The program establishes a connection between a client and a server, enabling communication for file transfer.
- **File Transfer:** The client sends the contents of a specified text file to the server, which saves the content as a new text file.

Execution Steps:

1. **Server Execution:**

- Run the server class and provide a port number for the server to listen on.
- The server starts listening on the specified port 8888.



Create, manage, and run configurations
Run a Java application

Name: Server (1)

Program arguments:
```
8888
```

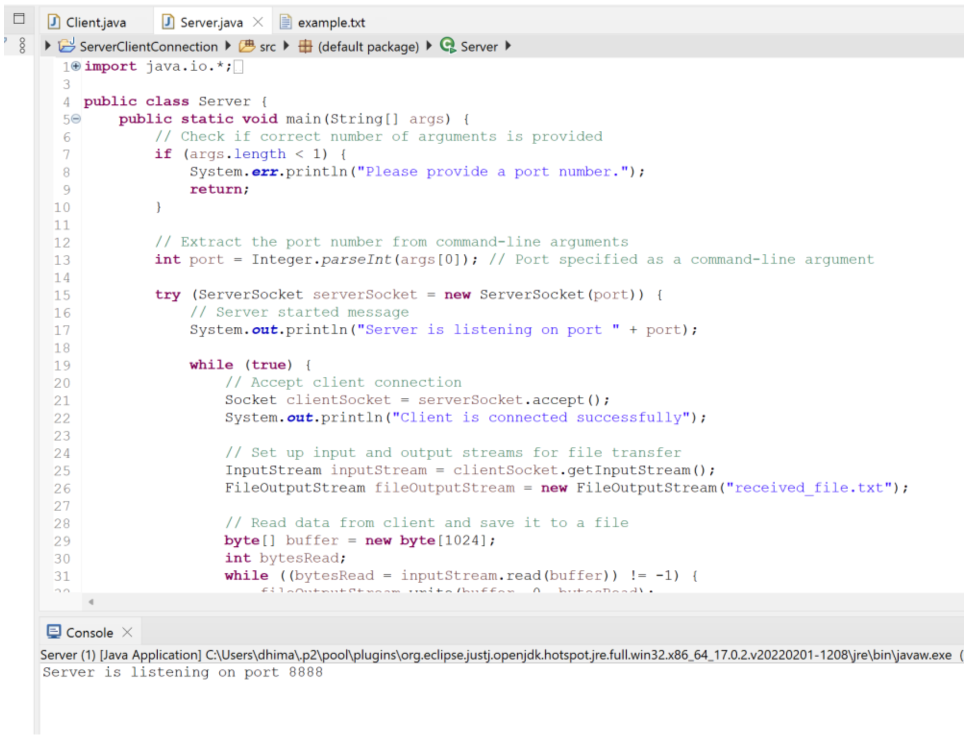
VM arguments:

[✓] Use the -XX:+ShowCodeDetailsInExceptionMessages argument when launching
[ ] Use @argfile when launching

Working directory:
[✓ Default: ${workspace_loc:ServerClientConnection}]
Other:

Filter matched 26 of 27 items

The above action provides port number 8888 for the server to listen to.

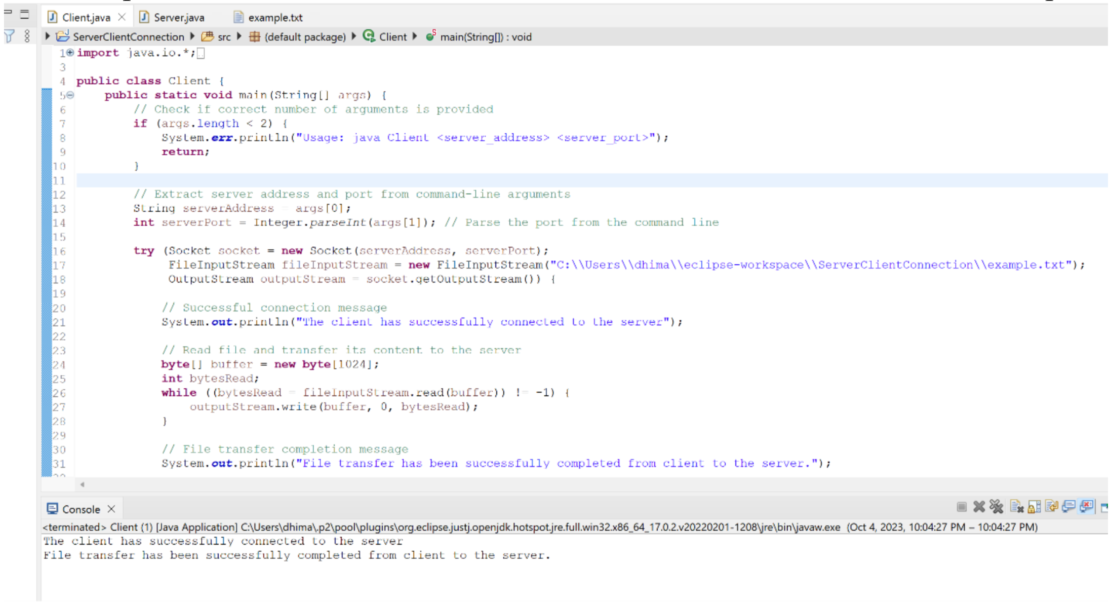


```java
import java.io.*;

public class Server {
    public static void main(String[] args) {
        // Check if correct number of arguments is provided
        if (args.length < 1) {
            System.err.println("Please provide a port number.");
            return;
        }

        // Extract the port number from command-line arguments
        int port = Integer.parseInt(args[0]); // Port specified as a command-line argument

        try (ServerSocket serverSocket = new ServerSocket(port)) {
            // Server started message
            System.out.println("Server is listening on port " + port);

            while (true) {
                // Accept client connection
                Socket clientSocket = serverSocket.accept();
                System.out.println("Client is connected successfully");

                // Set up input and output streams for file transfer
                InputStream inputStream = clientSocket.getInputStream();
                FileOutputStream fileOutputStream = new FileOutputStream("received_file.txt");

                // Read data from client and save it to a file
                byte[] buffer = new byte[1024];
                int bytesRead;
                while ((bytesRead = inputStream.read(buffer)) != -1) {
```

```
Server (1) [Java Application] C:\Users\dhima\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\javaw.exe (
Server is listening on port 8888
```

After providing the port number to the application, the server is now listening on port 8888.

2. **Client Execution:**
- Run the client class and provide the server's IP address and the desired port number.
3. **Client-Server Interaction:**
- The client attempts to connect to the server using the provided IP address and port.
- Upon successful connection, the client can send the contents of the "example.txt" file.

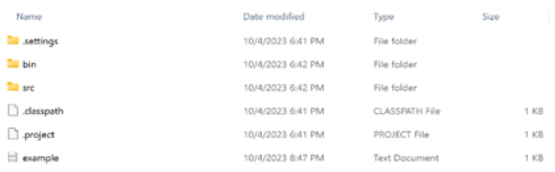


```java
import java.io.*;

public class Client {
    public static void main(String[] args) {
        // Check if correct number of arguments is provided
        if (args.length < 2) {
            System.err.println("Usage: java Client <server_address> <server_port>");
            return;
        }

        // Extract server address and port from command-line arguments
        String serverAddress = args[0];
        int serverPort = Integer.parseInt(args[1]); // Parse the port from the command line

        try (Socket socket = new Socket(serverAddress, serverPort);
             FileInputStream fileInputStream = new FileInputStream("C:\\Users\\dhima\\eclipse-workspace\\ServerClientConnection\\example.txt");
             OutputStream outputStream = socket.getOutputStream()) {

            // Successful connection message
            System.out.println("The client has successfully connected to the server");

            // Read file and transfer its content to the server
            byte[] buffer = new byte[1024];
            int bytesRead;
            while ((bytesRead = fileInputStream.read(buffer)) != -1) {
                outputStream.write(buffer, 0, bytesRead);
            }

            // File transfer completion message
            System.out.println("File transfer has been successfully completed from client to the server.");
```

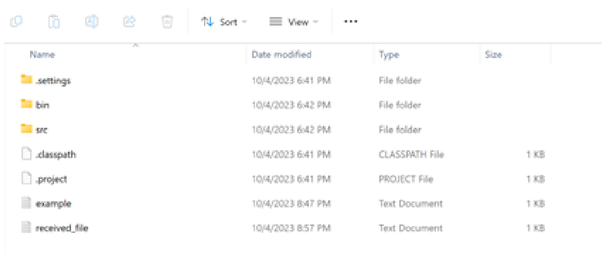
```
<terminated> Client (1) [Java Application] C:\Users\dhima\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\javaw.exe (Oct 4, 2023, 10:04:27 PM – 10:04:27 PM)
The client has successfully connected to the server
File transfer has been successfully completed from client to the server.
```

4. **File Transfer:**

- The server receives the file sent by the client and saves it as "received_file.txt".
- The contents of the client's file are transferred and saved in the server's file.

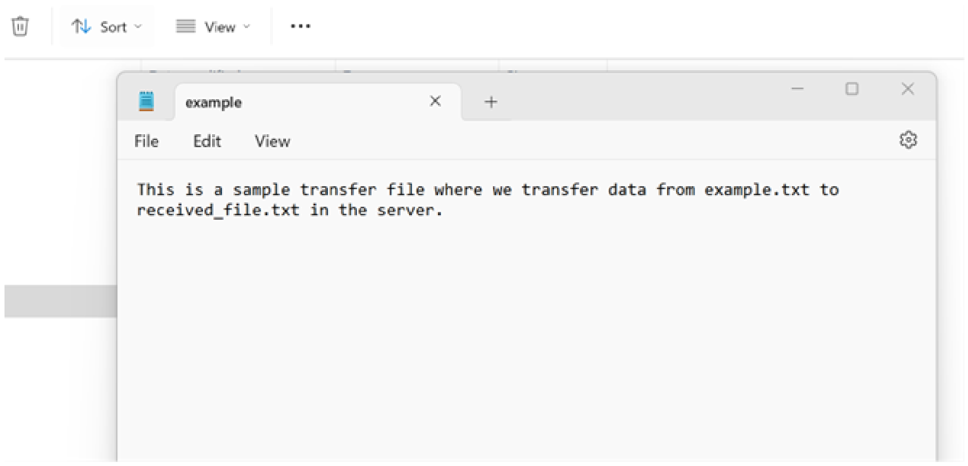


Name	Date modified	Type	Size
.settings	10/4/2023 6:41 PM	File folder	
bin	10/4/2023 6:42 PM	File folder	
src	10/4/2023 6:42 PM	File folder	
.classpath	10/4/2023 6:41 PM	CLASSPATH File	1 KB
.project	10/4/2023 6:41 PM	PROJECT File	1 KB
example	10/4/2023 8:47 PM	Text Document	1 KB

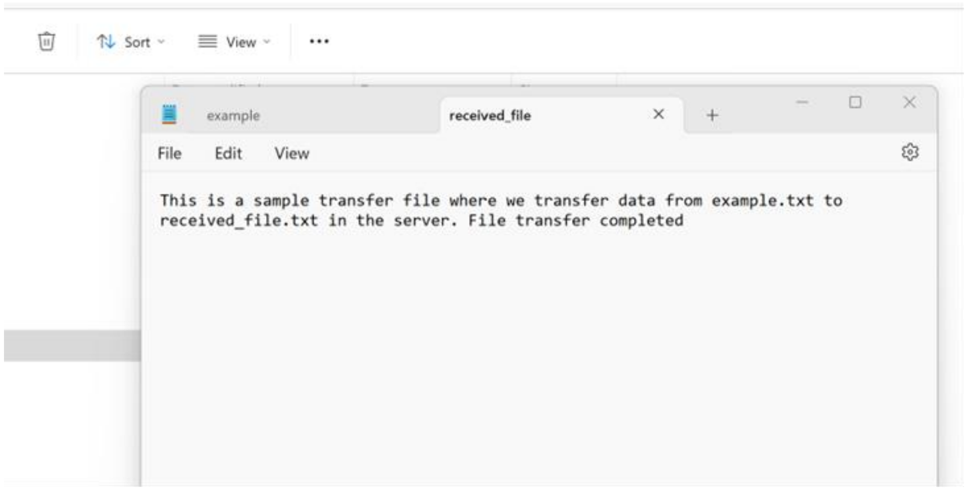


Name	Date modified	Type	Size
.settings	10/4/2023 6:41 PM	File folder	
bin	10/4/2023 6:42 PM	File folder	
src	10/4/2023 6:42 PM	File folder	
.classpath	10/4/2023 6:41 PM	CLASSPATH File	1 KB
.project	10/4/2023 6:41 PM	PROJECT File	1 KB
example	10/4/2023 8:47 PM	Text Document	1 KB
received_file	10/4/2023 8:57 PM	Text Document	1 KB

5. The received_file.txt is created at the same directory where the server application is running.



```
This is a sample transfer file where we transfer data from example.txt to
received_file.txt in the server.
```



```
This is a sample transfer file where we transfer data from example.txt to
received_file.txt in the server. File transfer completed
```

Conclusion:

This program serves as a foundational example for understanding socket programming and file transfer between a client and a server in Java.