Thank you for providing the correct code for **myserver.java**. Now, let's go through the step-by-step process of running the client-server communication program using these two files.

Step 1: Set up your development environment Make sure you have Java Development Kit (JDK) installed on your system.

Step 2: Create the project structure Create a directory for your project and name it accordingly. Inside the project directory, create two subdirectories: one for the client code and another for the server code. You can name them "client" and "server" respectively.

Step 3: Write the code for the client and server In the "client" directory, create a file named "MyClient.java" and paste the provided code for **myclient.java** into it. Similarly, in the "server" directory, create a file named "MyServer.java" and paste the provided code for **myserver.java** into it.

Step 4: Compile the code Open a terminal or command prompt and navigate to the "client" directory. Compile the "MyClient.java" file by running the following command:

Copy code

javac MyClient.java

Similarly, navigate to the "server" directory and compile the "MyServer.java" file by running:

Copy code

javac MyServer.java

This will generate the bytecode files (.class) for your programs.

Step 5: Run the server In the terminal or command prompt, while still in the "server" directory, execute the following command:

Copy code

java MyServer

This will start the server and make it listen for client connections on port 5555. The console will display the message "Server Initiated, Waiting for Client to Connect..."

Step 6: Run the client Open a new terminal or command prompt, navigate to the "client" directory, and execute the following command:

Copy code

java MyClient

This will run the client program and establish a connection with the server. The console will display the message "Connected to Server, Please type your message and hit Enter to send".

Step 7: Communicate between client and server Now, you can start exchanging messages between the client and server. In the client terminal, enter a message and press Enter. The message will be sent to the server. In response, the server will display the client's message in its terminal and prompt you to enter a server message. Enter a server message and press Enter. The server message will be sent to the client, and the client will display the server's message in its terminal. This communication will continue until either the client or the server enters "bye" as a message, which will terminate the connection.

Step 8: Terminate the connection To terminate the connection, either the client or the server needs to enter "bye" as a message. Upon termination, the console of both the client and server will display the message "Connection Terminated".

That's it! You have successfully run the client-server communication program using Java Sockets. You can repeat Step 6 and Step 7 to establish new connections and communicate between the client and server again.

During your practical exams, you can explain the steps mentioned above, emphasizing the setup, code compilation, running the server and client, communication process, and connection termination. Additionally, you can discuss the purpose of the experiment, which is to develop a distributed application using Java Sockets to enable client-server communication.

**MyClient** Explanation:

1. The **MyClient** class represents the client-side of the client-server communication.
2. The **Socket** class is used to create a client socket and establish a connection with the server. In this case, the IP address "127.0.0.1" refers to the local host, and the port number "5555" is used for communication.
3. The client uses **BufferedReader** to read input from the user through the console.
4. The **OutputStream** and **PrintWriter** are used to send data to the server.
5. The **InputStream** and **BufferedReader** are used to receive data from the server.
6. Inside the **while** loop, the client reads user input, sends it to the server using **pw.println(clientMessage)**, and waits for a response.
7. If either the client or server sends "bye", the loop breaks, terminating the connection.
8. After exiting the loop, the client closes the socket, input stream, output stream, and displays a termination message.

**MyServer** Explanation:

1. The **MyServer** class represents the server-side of the client-server communication.
2. The **ServerSocket** class is used to create a server socket and listen for client connections on port 5555.
3. Once a client connects, the **accept()** method is called, and the server establishes a socket connection with the client.
4. Similar to the client, the server uses **BufferedReader** to read input from the console.
5. The **OutputStream** and **PrintWriter** are used to send data to the client.
6. The **InputStream** and **BufferedReader** are used to receive data from the client.
7. Inside the **while** loop, the server first reads the client's message using **receive.readLine()** and displays it.
8. If the client's message is "bye", the loop breaks, terminating the connection.
9. The server then prompts for a server message using **System.out.print("Server: ")** and reads it from the console.
10. The server sends the server message to the client using **pw.println(serverMessage)**.
11. If the server message is "bye", the loop breaks, terminating the connection.
12. After exiting the loop, the server closes the socket, server socket, input stream, output stream, and displays a termination message.

Both the client and server code follow a similar structure, where they establish a connection, exchange messages, and terminate the connection based on the "bye" message. The client and server communicate by reading and writing to the input and output streams of the socket connection.

Please note that the code assumes that the client and server are running on the same machine (localhost) and using port 5555 for communication.