How Did I Choose the Classes?

When designing the chat system, we need to **think about the roles** of each part:

🖥 The Server (`NewServer`):

- It **waits** for clients to join.
- It **manages** the list of all connected clients.
- It **starts a new thread** for each client, so multiple people can talk at the same time.

👤 The Client (`Client`) :

- It **connects** to the server.
- It **sends** messages to the server.
- It **receives** messages from the server and prints them.

🎙 Handling Each Client (`NewClient`) – A Talking & Listening System:

- When a new client joins, this class **handles** their messages.
- It **forwards** their messages to all other clients.
- If a client **disconnects**, it tells everyone.

👂 Listening to Server Messages (`ServerListener`):

- This class is used **inside each client** to listen for messages from the server.
- Without it, the client wouldn't see messages from other users.

A teacher opens a classroom where students can join and talk to each other. When a student enters, they are added to the class list and can send messages that everyone hears. If a student leaves, they are removed from the list. This system ensures smooth and organized communication for everyone!

1- Server classes:

Could be separated into two classes, one for the main server and the other for the thread.

- **Main code:**

```
  import java.io.*;
a import java.net.Socket;
  import java.net.ServerSocket;
  import java.util.ArrayList;

  public class NewServer
b {
   private static ArrayList <NewClinet> clients = new ArrayList<>();
   public static void main(String[] args) throws IOException
c  {
     ServerSocket serverSocket = new ServerSocket(9090);

     while (true){
d     System.out.println("Waiting for client connection");
      Socket client = serverSocket.accept();
      System.out.println("Connected to client");
      NewClinet clientThread = new NewClinet (client,clients); // new thread
e     clients.add(clientThread);
      new Thread (clientThread).start();

     } }  }
```

This is the **main** program that runs the **server**. Think of the server as a teacher in a classroom, waiting for students (clients) to join and listen.

a. Imports (Library Magic):

- `import java.io.*;` → This brings in tools to **read and write messages**.
- `import java.net.Socket;` → This lets the server **talk** to clients.
- `import java.net.ServerSocket;` → This helps the server **wait** for clients to connect.
- `import java.util.ArrayList;` → This is like a **notebook** where the server keeps track of all students (clients).

b. Class & List of Clients:

- `public class NewServer` → This is the **server's home** where all the magic happens.
- `private static ArrayList <NewClinet> clients = new ArrayList<>();`
  - Think of this as a **classroom attendance list** 📋 where we keep track of all students (clients) who join.

c. Starting the Server:

- `public static void main(String[] args) throws IOException` → This is the **main door** 🚪 to start the server.
- `ServerSocket serverSocket = new ServerSocket(9090);`
  - The **server is now listening** at **room 9090** (like a classroom number).

d. Accepting Clients:

- `while (true) {` → This means **"forever, keep accepting students!"**
- `System.out.println("Waiting for client connection");`
  - The teacher (server) is saying **"I'm waiting for students!"**
- `Socket client = serverSocket.accept();`
  - A student **knocks** on the door 🚪, and the teacher **lets them in**.
- `System.out.println("Connected to client");`
  - The teacher says **"Hello, student! Welcome!"**

e. Adding the Client to the List:

- `NewClinet clientThread = new NewClinet(client,clients); //from NewClinet class we have created`
  - The teacher **creates a new student profile**.
- `clients.add(clientThread);`
  - The student **is added to the classroom list**.
- `new Thread(clientThread).start();`
  - The student **is now ready to talk**! 🎙️

- **Thread code:**

a
```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.logging.Level;
import java.util.logging.Logger;
```

b
```
public class Server2 implements Runnable{
  private Socket server;
  private BufferedReader in;
  private PrintWriter out;
```

c
```
  public Server2 (Socket s) throws IOException{
   server = s;
in = new BufferedReader (new InputStreamReader(server.getInputStream()));
        out = new PrintWriter(server.getOutputStream(),true);
        }
```

d
```
 @Override
 public void run(){

String serverResponse;
try {
  while(true){
  serverResponse = in.readLine();
  if (serverResponse == null) break;
  System.out.println("Server says: " + serverResponse);
  }
} catch (IOException ex) {
  ex.printStackTrace();
} finally{
  try {
    in.close();
  } catch (IOException ex) {
    ex.printStackTrace();
  }   }    }     }
```

This **listens** for messages from the client. It's like a **walkie-talkie** 🎤🎧 that listens for students speaking.

a. Imports (More Tools):

- These **tools** help us read, write, and log messages.

b. Class & Variables:

- `public class Server2 implements Runnable {` → This is a **talking machine** that listens to students.

- `private Socket server;` → This is the **connection line** between the teacher and student.
- `private BufferedReader in;` → This is a **microphone** 🎤 for hearing messages.
- `private PrintWriter out;` → This is a **speaker** 🔊 for responding.

c. Setting Up:

- `server = s;` → The **server starts talking** with this student.
- `in = new BufferedReader (new InputStreamReader(server.getInputStream()));`
    - This **listens** to what the student says.
- `out = new PrintWriter(server.getOutputStream(),true);`
    - This **sends messages** back to the student.

d. Listening to Messages:

- This part **keeps listening** 🎧
- If a student **talks**, it prints the message.
- If the student **leaves**, it **closes the connection**.

2- Client classes:

It is separated into two classes, one for the thread and the other for the main.

- **Main code:**

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
public class Client {
a       private static final String Server_IP = "localhost";
        private static final int Server_port = 9090;

public static void main(String[] args) throws IOException {
      b try(Socket socket = new Socket (Server_IP,Server_port)) {
c       Server2 servcon=new Server2(socket);
                    BufferedReader keyboard=new BufferedReader (new
InputStreamReader(System.in));
        PrintWriter out=new PrintWriter(socket.getOutputStream(),true);
        new Thread (servcon).start();
        try{
                while(true){
                        System.out.println("> ");
                      d String command=keyboard.readLine();
                        if(command.equals("quit")) break;
                        out.println(command);
                        } // end of while loop
                } catch (Exception e){
                        e.printStackTrace();
                }
                 }
                 System.exit(0);
        } }
```

<u>This is the **student** who wants to talk to the teacher.</u>

a. <u>Connecting to Server:</u>

- The **student knows** where the teacher is located (**room 9090**).

b. <u>Starting the Connection:</u>

- The **student knocks** on the teacher's door 🚪.

c. Setting Up Communication:

- The student **connects the microphone** 🎤.
- The student **can type messages** from the keyboard.

d. Sending Messages:

- The student **types something** and sends it.
- If they type `"quit"`, they **leave the classroom**.

- **Thread code:**

```
    import java.io.BufferedReader;
    import java.io.IOException;
    import java.io.InputStreamReader;
    import java.io.PrintWriter;
    import java.net.Socket;
    import java.util.ArrayList;

    class NewClinet implements Runnable {
    private Socket client;
a   private BufferedReader in;
    private PrintWriter out;
    private ArrayList<NewClinet> clients;
      public NewClinet (Socket c,ArrayList<NewClinet> clients) throws IOException{
        this.client = c;
b       this.clients=clients;
        in= new BufferedReader (new InputStreamReader(client.getInputStream()));
        out=new PrintWriter(client.getOutputStream(),true);
      }
      @Override
      public void run ()
      {
       try{
c       while (true){
          String request=in.readLine();
          if (request == null) {
              break; // Exit the loop if the client disconnects
            }
              outToAll(request);
        }
    } catch (IOException e){
          System.err.println("IO exception in new client class");
          System.err.println(e.getStackTrace());
      }
    finally{
      try {
            in.close();
d           out.close();
            client.close();
      } catch (IOException ex) {
        ex.printStackTrace();
      }
    synchronized (clients) {
          clients.remove(this); // Remove the client from the list
      }
      System.out.println("Client disconnected.");
    }
}
    private void outToAll(String substring) {
e     for (NewClinet aclient:clients) {
      aclient.out.println(substring);
} }}
```

This handles **talking between students.**

a. Class & Variables:

- This means **each client runs in a separate thread**, so many students can talk at the same time.
- `client` → This is the **student's personal connection** to the classroom.
- `in` → This **listens** to what the student says.
- `out` → This **sends messages** back to the student.
- `clients` → This is the **list of all students** in the class.

b. Setting Up the Student's Connection:

- When a new student joins, this **saves their details**.
- `in = new BufferedReader(new InputStreamReader(client.getInputStream()));`
  - This is like **a microphone** that listens to what the student says.
- `out = new PrintWriter(client.getOutputStream(), true);`
  - This is like **a speaker** that sends messages.

c. Reading Messages & Sending to All:

- This part **keeps listening** to what the student says.
- If the student **sends a message**, it gets **shared with everyone** using `outToAll(request)`.
- If the student **leaves**, the loop **stops**.

d. Removing Disconnected Clients:

- If something **goes wrong**, an error message appears.
- When a student **leaves**, we:
  1. **Close their microphone and speaker** (input and output).
  2. **Remove them from the class list**.
  3. **Announce that they have left**.

e. Sending Messages to All Students:

- This **shares** a student's message with **everyone** in the class.