

## Practical 2

### Disclaimer

#### Project 02 - Console Chat Application

Lesson 2.1 - Preparing Console Chat Application

Lesson 2.2 - ChatServer's ServerSocket and Socket

Lesson 2.3 - SessionInitial

Lesson 2.4 - ChatClient's Socket

Lesson 2.5 - Main Menu

Lesson 2.6 - Starting New Session

Lesson 2.7 - Choosing existing session

Lesson 2.8 - Connecting to existing session and start chat

# Practical 2

---

## Disclaimer

---

These examples are just to show how to use Java Streams and Java Sockets. It is not the exact way that these applications are implemented.

## Project 02 - Console Chat Application

---

This is an example application to show how far you can go to create a console chat application. The flow of the chat application is that a client can connect to the server and then the client can join a chat session where another client is waiting or the client can create a new session.

### Lesson 2.1 - Preparing Console Chat Application

1. Create a new package called `project02`
2. Create the Server Chat Application classes called `ChatServer`, `SessionHandler` and `SessionInitial`.

```
package project02;  
  
public class ChatServer {  
  
}
```

```
package project02;

public class SessionInitial {

}
```

```
package project02;

public class SessionHandler {

}
```

3. Create a Client Chat Application class called `ChatClient`.

```
package project02;

public class ChatClient {

}
```

4. Implement main method to `ChatServer` and `ChatClient` that means to start Server Chat Application will be from `ChatServer` class and Client Chat Application will be from `ChatClient` class.

```
package project02;

public class ChatServer {
    public static void main(String[] args) {

    }

}
```

```
package project02;

public class ChatClient {
    public static void main(String[] args) {

    }

}
```

## Milestone:

Commit the changes made to the `src` folder with message "Lesson 2.1 - Preparing Console Chat Application".

Hint: Add the files to the staging area first, `git add <files>` and then proceed to commit, `git commit -m "Lesson 2.1 - Preparing Console Chat Application"`.

## Lesson 2.2 - ChatServer's ServerSocket and Socket

1. For a server to start its service, it has to bind itself into the computer's port number through `ServerSocket` object. Let's bind this server to port number `9101`. When binding, if the port number in the computer is already used, it will throw a `BindException`. We could surround this using `try-catch`. Another exception is thrown by `ServerSocket` is `IOException`. Let's catch that exception as well.

```
package project02;

public class ChatServer {
    public static void main(String[] args) {
        ServerSocket serverSocket = null;
        try {
            serverSocket = new ServerSocket(9101);
        } catch (BindException e) {
            System.out.println("Unable to start server due to port is already in used.");
        } catch (IOException e) {
            System.out.println("Unexpected problem. Unable to start server.");
        }
    }
}
```

2. It's a good practice to close `ServerSocket` object once the application is done using it. Closing `ServerSocket` might throw `IOException`. We should surround this using `try-catch`.

```
package project02;

public class ChatServer {
    public static void main(String[] args) {
        ServerSocket serverSocket = null;
        try {
            serverSocket = new ServerSocket(9101);
        } catch (BindException e) {
```

```

        System.out.println("Unable to start server due to port is already in
used.");
    } catch (IOException e) {
        System.out.println("Unexpected problem. Unable to start server.");
    } finally {
        try {
            if(serverSocket != null) {
                serverSocket.close();
            }
        } catch (IOException e) {
            System.out.println("Unexpected problem. Unable to close server
socket.");
        }
    }
}
}
}

```

3. Lets add a console output to indicate that the server has started. (Make sure to import `java.util.Date`)

```

package project02;

public class ChatServer {
    public static void main(String[] args) {
        ...
        serverSocket = new ServerSocket(9101);
        System.out.println("Server started at " + new Date());
        ...
    }
}

```

4. Once the server successfully started, it will listen and accept connection to the server. Since we are going to allow multiple user, the server will create a new session everytime a client has connected, `SessionInitial` object. Since we need the `Socket` object that used for the specific client connection with the server, we need to pass the `Socket` object to the `SessionInitial` constructor. Each session should be independent from each other, hence new `Thread` is created for each session. (Note: there will be an error because we haven't create any constructor in `SessionInitial` class.)

```

package project02;

public class ChatServer {
    public static void main(String[] args) {
        ...
        serverSocket = new ServerSocket(9101);

```

```

        System.out.println("Server started at " + new Date());
        while(true) {
            Socket client = serverSocket.accept();
            new Thread(new SessionInitial(client)).start();
        }
        ...
    }
}

```

## Milestone:

Commit the changes made to the `src` folder with message "Lesson 2.2 - ChatServer's ServerSocket and Socket".

Hint: Add the files to the staging area first, `git add <files>` and then proceed to commit, `git commit -m "Lesson 2.2 - ChatServer's ServerSocket and Socket"`.

## Lesson 2.3 - SessionInitial

1. At the beginning of the session, it will create a session for the client to start choosing existing chat session or create a new chat session. Since in `ChatServer` class it creates a new `SessionInitial` object and pass Socket object to the constructor. We need to create the constructor.

```

package project02;

public class SessionInitial {
    public SessionInitial(Socket client) {

    }
}

```

2. We need to be able to use the `socket` object in this class. Lets create a Socket instance variable called `client` and make the instance variable refer to the object. Indicate in server console that a client has connected.

```

package project02;

public class SessionInitial {
    private Socket client;

    public SessionInitial(Socket client) {
        this.client = client;
        System.out.println("Client connected: " +
client.getInetAddress().getHostAddress() + " "
        + client.getInetAddress().getHostName() + " .");
    }
}

```

3. `SessionInitial` class needs to be a `Runnable` type since it is being run in a Thread.

```

package project02;

public class SessionInitial implements Runnable {
    private Socket client;

    public SessionInitial(Socket client) {
        this.client = client;
        System.out.println("Client connected: " +
client.getInetAddress().getHostAddress() + " "
        + client.getInetAddress().getHostName() + " .");
    }

    public void run() {

    }
}

```

4. Then we need input and output stream to communicate with the client from the server through the socket. Since we are going to use these stream objects in multiple methods, we are going to declare instance variables for these streams.

```

package project02;

public class SessionInitial implements Runnable {
    private Socket client;
    private DataInputStream fromClient;
    private DataOutputStream toClient;

    ...
}

```

```

public void run() {
    try {
        fromClient = new DataInputStream(client.getInputStream());
        toClient = new DataOutputStream(client.getOutputStream());
    } catch (IOException e) {
        System.out.println("Communication problem with " +
client.getInetAddress().getHostAddress()
                        + " " + client.getInetAddress().getHostName() + "
.");
    }
}
}
}

```

5. Send a welcome message to the client.

```

package project02;

public class SessionInitial implements Runnable {
    ...

    public void run() {
        try {
            fromClient = new DataInputStream(client.getInputStream());
            toClient = new DataOutputStream(client.getOutputStream());

            toClient.writeUTF("Welcome to PB Chatting.");
        } catch (IOException e) {
            System.out.println("Communication problem with "
                                + client.getInetAddress().getHostAddress()
                                + " " + client.getInetAddress().getHostName() + "
.");
        }
    }
}
}

```

## Milestone:

Commit the changes made to the `src` folder with message "Lesson 2.3 - SessionInitial".

Hint: Add the files to the staging area first, `git add <files>` and then proceed to commit, `git commit -m "Lesson 2.3 - SessionInitial"`.



## Lesson 2.4 - ChatClient's Socket

1. Create instance variables for ip address, port number and `Socket` object.

```
package project02;

public class ChatClient {
    private String ipAddress = "localhost";
    private int portNumber = 9101;
    private Socket socket;

    public static void main(String[] args) {

    }
}
```

2. Request connection, to the server based on the ip address (in this case `localhost` will point to the computer and to the server application through `portNumber`) by creating `Socket` object with ip address and port number and assign to the instance variable `socket`. Lets add the `try and catch` immediately as well.

```
package project02;

public class ChatClient {
    private static String ipAddress = "localhost";
    private static int portNumber = 9101;
    private static Socket socket;

    public static void main(String[] args) {
        try {
            socket = new Socket(ipAddress, portNumber);
        } catch (ConnectException | UnknownHostException e) {
            System.out.println("Unable to connect to server " + ipAddress + ":" +
portNumber);
        } catch (IOException e) {
            System.out.println("Problem communicating with the server.");
        } finally {
            try {
                if(socket != null) {
                    socket.close();
                }
            } catch (IOException e) {}
        }
    }
}
```

```
}
```

3. Then we need input and output stream to communicate with the server from the client through the socket.

```
package project02;

public class ChatClient {
    ...
    private static Socket socket;
    private static DataInputStream fromServer;
    private static DataOutputStream toServer;

    public static void main(String[] args) {
        ...
        socket = new Socket(ipAddress, portNumber);

        fromServer = new DataInputStream(socket.getInputStream());
        toServer = new DataOutputStream(socket.getOutputStream());
        ...
    } finally {
    } try {
        if(socket != null) {
            socket.close();
        }
        if(fromServer != null) {
            fromServer.close();
        }
        if(toServer != null) {
            toServer.close();
        }
    } catch (IOException e) {}
    }
    ...
}
```

4. Since this is a chat application, the client should be able to type something and the client application should capture that value. We need `Scanner` object.

```
package project02;

public class ChatClient {
    ...
    private static DataOutputStream toServer;
```

```

private static Scanner scanner;

public static void main(String[] args) {
    ...
    toServer = new DataOutputStream(socket.getOutputStream());

    scanner = new Scanner(System.in);
    ...
}
}

```

5. Its time to read the welcoming message from the server.

```

package project02;

public class ChatClient {
    ...
    private static DataOutputStream toServer;
    private static Scanner scanner;

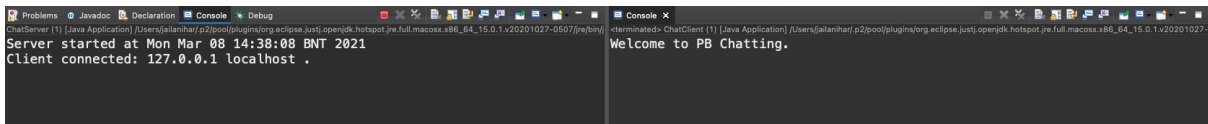
    public static void main(String[] args) {
        ...
        toServer = new DataOutputStream(socket.getOutputStream());

        scanner = new Scanner(System.in);

        System.out.println(fromServer.readUTF());
        ...
    }
}

```

6. Run ChatServer first and then run ChatClient. The ChatClient should receive the welcome message. **Don't forget to stop the ChatServer manually before continuing.**



## Milestone:

Commit the changes made to the `src` folder with message "Lesson 2.4 - ChatClient's Socket".

Hint: Add the files to the staging area first, `git add <files>` and then proceed to commit, `git commit -m "Lesson 2.4 - ChatClient's Socket"`.

## Lesson 2.5 - Main Menu

1. Its time for the main menu for the application. The idea is that chat client will receive the menu from the server for the client to choose and the client will send their choice to the server. Lets continue implementing in `SessionInitial` class.

```
package project02;

public class SessionInitial implements Runnable {

    ...

}
```

2. Create a method to send the menu to the client. (Don't forget to throw the exception.)

```
package project02;

public class SessionInitial implements Runnable {

    ...

    public void sendMainMenu() throws IOException {
        String menu = "";
        menu = menu + "Please choose an option:\n";
        menu = menu + "(1) Start a new chatting session\n";
        menu = menu + "(2) Connect to available session\n";
        toClient.writeUTF(menu);
    }

}
```

3. Lets create another method called `mainMenuLogic` that call `sendMainMenu` method and call `mainMenuLogic()` in the `run` method.

```
package project02;

public class SessionInitial implements Runnable {

    ...

}
```

```

public void run() {
    try {
        ...

        toClient.writeUTF("Welcome to PB Chatting.");
        mainMenuLogic();

    } catch (IOException e) {
        ...
    }
}

public void mainMenuLogic() throws IOException {
    sendMainMenu();
}

public void sendMainMenu() throws IOException {
    ...
}
}

```

4. Since the server is writing UTF to the stream, the client should read UTF from the stream. We need to implement this in `ChatClient` class.

```

package project02;

public class ChatClient {
    ...

    public static void main(String[] args) {
        ...
        System.out.println(fromServer.readUTF());
        mainMenuLogic();
        ...
    }

    public static void mainMenuLogic() throws IOException {
        readMainMenu();
    }

    public static void readMainMenu() throws IOException {
        System.out.println(fromServer.readUTF());
    }
}

```

5. Now its time for the client to choose an option from the menu. Lets create another method called `selectMainMenuOption()` in `ChatClient` class. (Make sure to throw `IOException`).

```
package project02;

public class ChatClient {
    ...

    public static void readMainMenu() throws IOException {
        System.out.println(fromServer.readUTF());
    }

    public static void selectMainMenuOption() throws IOException {

    }
}
```

6. Capture user input and send to the server. (Don't forget to call the `selectMainMenuOption` method in the `mainMenuLogic` method.)

```
package project02;

public class ChatClient {
    ...

    public static void mainMenuLogic() throws IOException {
        readMainMenu();
        selectMainMenuOption();
    }

    public static void selectMainMenuOption() throws IOException {
        String option = scanner.nextLine();
        toServer.writeUTF(option);
    }
}
```

7. The server need to read this option selected by the client. Lets create another method called `readOptionFromClient` in `SessionInitial` class.

```
package project02;

public class SessionInitial implements Runnable {
    ...

    public void mainMenuLogic() throws IOException {
```

```

        sendMainMenu();
        readOptionFromClient();
    }

    public void readOptionFromClient() throws IOException {
        String option = fromClient.readUTF();
    }
}

```

8. We need to make sure the user input only valid values for the menu and notify the client if the option selected is valid.

```

package project02;

public class SessionInitial implements Runnable {
    ...

    public void readOptionFromClient() throws IOException {
        String option = fromClient.readUTF();
        boolean validOptionSelected = false;
        if(option.equals("1") || option.equals("2")) {
            validOptionSelected = true;
            toClient.writeBoolean(validOptionSelected);
        } else {
            validOptionSelected = false;
            toClient.writeBoolean(validOptionSelected);
        }
    }
}

```

9. If option 1 is selected, start a new session and if option 2 is selected list down available session. Other than that it is invalid, notify client that the option is invalid and send main menu back to the client.

```

package project02;

public class SessionInitial implements Runnable {
    ...

    public void readOptionFromClient() throws IOException {
        ...
        if(option.equals("1") || option.equals("2")) {
            validOptionSelected = true;
            toClient.writeBoolean(validOptionSelected);
        }
    }
}

```

```

        if(option.equals("1")) {
            startANewSession();
        }
        if(option.equals("2")) {
            listDownAvailableSession();
        }
    } else {
        validOptionSelected = false;
        toClient.writeBoolean(validOptionSelected);
        toClient.writeUTF("Wrong option. Please select correct option.");
        mainMenuLogic();
    }
}
}
}

```

10. Lets define the two methods `startANewSession` and `listDownAvailableSession`. For now just leave them blank.

```

package project02;

public class SessionInitial implements Runnable {
    ...

    public void readOptionFromClient() throws IOException {
        ...
    }

    public void startANewSession() {

    }

    public void listDownAvailableSession() {

    }
}

```

11. In `ChatClient`, further implementation needed to receive result if the option selected is valid or not.

```

package project02;

public class ChatClient {
    ...
}

```



```

public static void selectMainMenuOption() throws IOException {
    String option = scanner.nextLine();
    toServer.writeUTF(option);

    boolean validOptionSelected = fromServer.readBoolean();
    if(validOptionSelected) {

    } else {

    }
}
}

```

12. If option 1 is selected, since its a new session the client has to wait for other client. If option 2 is selected, the client will receive list of available sessions. Other option the client should receive the notification and the main menu again. The client needs to select the option again.

```

package project02;

public class ChatClient {
    ...

    public static void selectMainMenuOption() throws IOException {
        String option = scanner.nextLine();
        toServer.writeUTF(option);

        boolean validOptionSelected = fromServer.readBoolean();
        if(validOptionSelected) {
            if(option.equals("1")) {
                waitingForOtherClient();
            }
            if(option.equals("2")) {
                listDownAvailableSession();
            }
        } else {
            System.out.println(fromServer.readUTF());
            mainMenuLogic();
        }
    }
}

```

13. Lets define the two methods `waitingForOtherClient` and `listDownAvailableSession`. For now just leave them blank.

```

package project02;

```

```

public class ChatClient {
    ...

    public static void selectMainMenuOption() throws IOException {
        ...
    }

    public static void waitingForOtherClient() {

    }

    public static void listDownAvailableSession() {

    }

}

```

14. Run it, don't forget to run `ChatServer` first then `ChatClient`. Try option 3 and 4, it should be invalid and client should be able to choose another option. But once option 1 or 2 is chosen, it should be terminated because we haven't implemented anything after that. **Don't forget to stop the ChatServer manually before continuing.**

```

Problems | Javadoc | Declaration | Console | Debug
ChatServer (1) [Java Application] /Users/jailanhar/p2/pool/plugins/org.eclipse.justi.openjdk.hotspot.jre.full.macosx.x86_64-15.0.1.v20201027-0507/jre/bin/
Server started at Mon Mar 08 14:29:16 BNT 2021
Client connected: 127.0.0.1 localhost

terminated- ChatClient (1) [Java Application] /Users/jailanhar/p2/pool/plugins/org.eclipse.justi.openjdk.hotspot.jre.full.macosx.x86_64-15.0.1.v20201027-
Welcome to PB Chatting.
Please choose an option:
(1) Start a new chatting session
(2) Connect to available session

3
Wrong option. Please select correct option.
Please choose an option:
(1) Start a new chatting session
(2) Connect to available session

4
Wrong option. Please select correct option.
Please choose an option:
(1) Start a new chatting session
(2) Connect to available session

1

```

## Milestone:

Commit the changes made to the `src` folder with message "Lesson 2.5 - Main Menu".

Hint: Add the files to the staging area first, `git add <files>` and then proceed to commit, `git commit -m "Lesson 2.5 - Main Menu"`.

## Lesson 2.6 - Starting New Session

1. The chat session between two clients will be handled by `SessionHandler` class. Each chat session should have session id to easily differentiate different sessions. Lets create two class variables; `availableSessions` to hold a list of available sessions and `numberOfSessionCreated` to indicate how many sessions have been created in the server (this will be used as session id); in `ChatServer` class. (Import from `java.util` for `List`)

```
package project02;

public class ChatServer {
    static List<SessionHandler> availableSessions;
    static int numberOfSessionCreated = 0;

    public static void main(String[] args) {
        ...
    }
}
```

2. Create a new `ArrayList` object and assign it to `availableSessions`. Since this will be accessed by multiple threads, we don't want two threads accessing the object at the same time. So we need to synchronised the list to make sure only one thread can access at a time by surrounding the new `ArrayList` object with `Collections.synchronizedList`. (Import from `java.util` for `Collections` and `ArrayList`)

```
package project02;

public class ChatServer {
    static List<SessionHandler> availableSessions;
    static int numberOfSessionCreated = 0;

    public static void main(String[] args) {
        availableSessions = Collections.synchronizedList(new
        ArrayList<SessionHandler>());
        ...
    }
}
```

3. In `SessionInitial` class, create a new `SessionHandler` object and start waiting for another client to connect to the session. Increment the `ChatServer.numberOfSessionCreated++` to make sure each session number is different. We need to pass the socket object to `SessionHandler`.

```

package project02;

public class SessionInitial implements Runnable {
    ...

    public void startANewSession() {
        ChatServer.numberOfSessionCreated++;
        SessionHandler chatSession = new
SessionHandler(ChatServer.numberOfSessionCreated, client);
    }

    ...
}

```

4. Notify the client that a new session has started and print a message in server console. We are going to generate a session name using the `getSessionName` method in `SessionHandler`.

```

package project02;

public class SessionInitial implements Runnable {
    ...

    public void startANewSession() {
        ChatServer.numberOfSessionCreated++;
        SessionHandler chatSession = new
SessionHandler(ChatServer.numberOfSessionCreated, client);
        System.out.println("New session started: " + session.getSessionName());
        toClient.writeUTF("New session started: " + session.getSessionName());
    }

    ...
}

```

5. Add the session to the `availableSession` list in `ChatServer` and start waiting for other client to connect to that session. (Don't forget to throw `IOException`)

```

package project02;

public class SessionInitial implements Runnable {
    ...

    public void startANewSession() throws IOException {
        ChatServer.numberOfSessionCreated++;

```

```

        SessionHandler chatSession = new
SessionHandler(ChatServer.numberOfSessionCreated, client);
        System.out.println("New session started: " +
chatSession.getSessionName());
        toClient.writeUTF("New session started: " +
chatSession.getSessionName());

        ChatServer.availableSessions.add(chatSession);
        chatSession.startWaitingOtherClient();
    }

    ...
}

```

6. In `SessionHandler`, let's define the `constructor` and `startWaitingOtherClient` method.

```

package project02;

public class SessionHandler {
    public SessionHandler(int sessionId, Socket client1) {

    }

    public void startWaitingOtherClient() {

    }
}

```

7. Let's declare the instance variables we need for the `SessionHandler`. Then we assigned based on the data passed through the constructor and assign `waitingForClient2` to `true`. This is to indicate it is waiting for another client.

```

package project02;

public class SessionHandler {
    private Socket client1;
    private int sessionId;
    private boolean waitingForClient2;

    public SessionHandler(int sessionId, Socket client1) {
        this.sessionId = sessionId;
        this.client1 = client1;
        waitingForClient2 = true;
    }
}

```

```

    public void startWaitingOtherClient() {

    }

}

```

8. We need to declare two more instance variables for `DataInputStream` and `DataOutputStream` object. We create those objects when we start waiting for other client. (Don't forget to implement the `try-catch` block.)

```

package project02;

public class SessionHandler {
    ...
    private boolean waitingForClient2;
    private DataInputStream fromClient1;
    private DataOutputStream toClient1;
    ...

    public void startWaitingOtherClient() {
        try {
            fromClient1 = new DataInputStream(client1.getInputStream());
            toClient1 = new DataOutputStream(client1.getOutputStream());

        } catch (IOException e) {
            System.out.println("Communication problem in " + getSessionName());
        }
    }
}

```

9. Now the server will wait for another client to connect to the session. It will try to check and notify the current client if another client has connected every 500 milliseconds.

```

package project02;

public class SessionHandler {
    ...

    public void startWaitingOtherClient() {
        try {
            fromClient1 = new DataInputStream(client1.getInputStream());
            toClient1 = new DataOutputStream(client1.getOutputStream());

            while(waitingForClient2) {
                try {

```

```

        Thread.sleep(500);
        toClient1.writeBoolean(waitingForClient2);
        toClient1.writeUTF("Waiting for other client to connect");
    } catch (InterruptedException e) {
        System.out.println(getSessionName() + " was waiting, but
interrupted.");
    }
}

} catch (IOException e) {
    System.out.println("Communication problem in " + getSessionName());
}
}
}

```

10. We still need to generate the session name. Lets define and implement the generate session name in `getSessionName` method.

```

package project02;

public class SessionHandler {
    ...

    public void startWaitingOtherClient() {
        ...
    }

    public String getSessionName() {
        return sessionId + " " + client1.getInetAddress().getHostName() + " " +
client1.getInetAddress().getHostAddress();
    }
}

```

11. In `ChatClient` class, lets read the start new session message from the server first and implement in `waitingForOtherClient()` method. (Don't forget to throw `IOException`)

```

package project02;

public class ChatClient {
    ...

    public static void waitingForOtherClient() throws IOException {
        System.out.println(fromServer.readUTF());
    }

    ...
}

```

12. Wait for another client to connect to the session by reading the boolean sent by the server. Check every 500 milliseconds.

```

package project02;

public class ChatClient {
    ...

    public static void waitingForOtherClient() throws IOException {
        System.out.println(fromServer.readUTF());
        boolean waiting = true;
        while(waiting) {
            try {
                Thread.sleep(500);
                waiting = fromServer.readBoolean();
                System.out.println(fromServer.readUTF());
            } catch (InterruptedException e) {
                System.out.println("Waiting was interrupted.");
            }
        }
    }

    ...
}

```

13. Run it, don't forget to run `ChatServer` first then `ChatClient`. Try option 1, this will create a new session. The client will now receive waiting for other client message every 500 milliseconds.



```
Problems | Javadoc | Declaration | Console | Debug
ChatServer (1) [Java Application] /Users/jalalshah/p2/pool/plugins/org.eclipse.justi.openjdk.hotspot.jre.full.macosx.x86_64_15.0.1.v20201027-0507/jre/bin/
Server started at Mon Mar 08 15:44:23 BNT 2021
Client connected: 127.0.0.1 localhost .
New session started: 1 localhost 127.0.0.1

ChatClient (1) [Java Application] /Users/jalalshah/p2/pool/plugins/org.eclipse.justi.openjdk.hotspot.jre.full.macosx.x86_64_15.0.1.v20201027-0507/jre/bin/
Welcome to PB Chatting.
Please choose an option:
(1) Start a new chatting session
(2) Connect to available session

1
New session started: 1 localhost 127.0.0.1
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
```

14. If another client created new session. **Don't forget to stop the ChatServer and the two ChatClient manually before continuing.**

```
Problems | Javadoc | Declaration | Console | Debug
ChatServer (1) [Java Application] /Users/jalalshah/p2/pool/plugins/org.eclipse.justi.openjdk.hotspot.jre.full.macosx.x86_64_15.0.1.v20201027-0507/jre/bin/
Server started at Mon Mar 08 15:44:23 BNT 2021
Client connected: 127.0.0.1 localhost .
New session started: 1 localhost 127.0.0.1
Client connected: 127.0.0.1 localhost .
New session started: 2 localhost 127.0.0.1

ChatClient (1) [Java Application] /Users/jalalshah/p2/pool/plugins/org.eclipse.justi.openjdk.hotspot.jre.full.macosx.x86_64_15.0.1.v20201027-0507/jre/bin/
(1) Start a new chatting session
(2) Connect to available session

1
New session started: 2 localhost 127.0.0.1
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
```

## ! Milestone:

Commit the changes made to the `src` folder with message "Lesson 2.6 - Starting New Session".

Hint: Add the files to the staging area first, `git add <files>` and then proceed to commit, `git commit -m "Lesson 2.6 - Starting New Session"`.

## Lesson 2.7 - Choosing existing session

1. Previously we have implemented on client creating a new session. Now we want to implement feature where client can choose existing available session. We will implement the logic at `listDownAvailableSession` in `SessionInitial` first.

```
package project02;

public class SessionInitial implements Runnable {
    ...

    public void startANewSession() throws IOException {
        ...
    }

    public void listDownAvailableSession() {

    }
}
```

2. First we need to check if there are available sessions or not. This can be achieved by checking the `availableSessions` variable from `ChatServer`.

```
package project02;

public class SessionInitial implements Runnable {
    ...

    public void listDownAvailableSession() {
        boolean noAvailableSession = ChatServer.availableSessions.isEmpty();
    }
}
```

3. Notify the client if there are available sessions or not. (Don't forget to throw `IOException`)

```
package project02;

public class SessionInitial implements Runnable {
    ...

    public void listDownAvailableSession() throws IOException {
        boolean noAvailableSession = ChatServer.availableSessions.isEmpty();
        toClient.writeBoolean(haveAvailableSession);
    }
}
```

4. If there are available sessions, send the list to the client. Else ask the client if they want to start new session.

```
package project02;

public class SessionInitial implements Runnable {
    ...

    public void listDownAvailableSession() throws IOException {
        boolean noAvailableSession = ChatServer.availableSessions.isEmpty();
        toClient.writeBoolean(noAvailableSession);
        if(!noAvailableSession) {

        } else {

        }
    }
}
```

```
}
```

5. If there are available sessions, send the number of available sessions to the client first so that the client knows how many times it should read the list. Then send the list to the client. Allow client to go back to main menu as well.

```
package project02;

public class SessionInitial implements Runnable {
    ...

    public void listDownAvailableSession() throws IOException {
        ...
        if(!noAvailableSession) {
            int numberOfAvailableSessions = ChatServer.availableSessions.size();
            toClient.writeInt(ChatServer.availableSessions.size());
            toClient.writeUTF("Please choose an option:");
            for(int i = 0; i < ChatServer.availableSessions.length; i++) {
                SessionHandler chatSession = ChatServer.availableSessions.get(i);
                toClient.writeUTF("(" + i + ") " + chatSession.getSessionName());
            }
            toClient.writeUTF("(" + ChatServer.availableSessions.size()
                               + ") Back to Main Menu");
        } else {

        }
    }
}
```

6. In `ChatClient` class, at `listDownAvailableSession` method. It needs to read and list the session. First, read the boolean from server if it have any available sessions or not. If there are available sessions, list it down. Else, client can create new session. (Don't forget to throw `IOException`)

```
package project02;

public class ChatClient {
    ...

    public static void listDownAvailableSession() {
        boolean noAvailableSession = fromServer.readBoolean();
        if(!noAvailableSession) {

        } else {

        }
    }
}
```

```

    }
}
}

```

7. List down the list.

```

package project02;

public class ChatClient {
    ...

    public static void listDownAvailableSession() throws IOException {
        boolean noAvailableSession = fromServer.readBoolean();
        if(!noAvailableSession) {
            int numberOfAvailableSession = fromServer.readInt();
            System.out.println(fromServer.readUTF());
            for(int i = 0; i < numberOfAvailableSession; i++) {
                System.out.println(fromServer.readUTF());
            }
            System.out.println(fromServer.readUTF());
        } else {

        }
    }
}

```

8. Run it, don't forget to run `ChatServer` first then run two `ChatClient`. Let the two chat clients to create new sessions. Next, run another `ChatClient` and then choose option 2 to list available sessions. There should be two listed. **Don't forget to stop the ChatServer and the three ChatClient manually before continuing.**

9. In `ChatClient`, its time to choose which session to join. Lets define another method called `selectAvailableSession`. Since we need to accomodate for going back to main menu, we need to pass the `numberOfAvailableSession` value to the method since that value is used for the main menu.

```

package project02;

public class ChatClient {

```

```

...

public static void listDownAvailableSession() throws IOException {
    boolean noAvailableSession = fromServer.readBoolean();
    if(!noAvailableSession) {
        int numberOfAvailableSession = fromServer.readInt();
        System.out.println(fromServer.readUTF());
        for(int i = 0; i < numberOfAvailableSession; i++) {
            System.out.println(fromServer.readUTF());
        }
        System.out.println(fromServer.readUTF());

        selectAvailableSession(numberOfAvailableSession);
    } else {

    }
}

public static void selectAvailableSession(int numberOfAvailableSession) {

}
}

```

10. Now we need to ask for user to select which session. Since the value is int, we need to validate if the user typed in int value as well. If it is not a number ask for another input by calling back the method.

```

package project02;

public class ChatClient {
    ...

    public static void selectAvailableSession(int numberOfAvailableSession) {
        int sessionSelected = 0;
        try {
            sessionSelected = Integer.parseInt(scanner.nextLine());

        } catch (NumberFormatException e) {
            System.out.println("Selected Session is not valid.");
            selectAvailableSession(numberOfAvailableSession);
        }
    }
}

```

11. Send the int value to the server and the server will validate. (Don't forget to throw IOException)

```
package project02;

public class ChatClient {
    ...

    public static void selectAvailableSession(int numberOfAvailableSession)
        throws IOException {
        int sessionSelected = 0;
        try {
            sessionSelected = Integer.parseInt(scanner.nextLine());
            toServer.writeInt(sessionSelected);
            boolean validSelectedSession = fromServer.readBoolean();
            System.out.println(fromServer.readUTF());
            if(validSelectedSession) {

            } else {

            }
        } catch (NumberFormatException e) {
            System.out.println("Selected Session is not valid.");
            selectAvailableSession(numberOfAvailableSession);
        }
    }
}
```

12. If it is valid, check if the user chosen one of the session or go back to main menu. If the user chosen one of the session, start the chatting session. If go back to main menu, start the main menu logic all over again.

```
package project02;

public class ChatClient {
    ...

    public static void selectAvailableSession(int numberOfAvailableSession)
        throws IOException {
        ...
        if(validSelectedSession) {
            if(sessionSelected == numberOfAvailableSession) {
                mainMenuLogic();
            } else {
                chatLogic();
            }
        } else {
        }
    }
}
```

```

    }
    ...
}
}

```

13. If it is not valid, ask for the list of available sessions again.

```

package project02;

public class ChatClient {
    ...

    public static void selectAvailableSession(int numberOfAvailableSession)
        throws IOException {
        ...
        if(validSelectedSession) {
            ...
        } else {
            listDownAvailableSession();
        }
        ...
    }
}

```

14. Lets implement the `chatLogic` method and leave it empty for now.

```

package project02;

public class ChatClient {
    ...

    public static void selectAvailableSession(int numberOfAvailableSession)
        throws IOException {
        ...
    }

    public static void chatLogic() {

    }
}

```

15. In `SessionInitial` class, handle the client choice and check if it is valid or not. Define a new method called `clientSelectedSession` and call the method in `listDownAvailableSession` method right after sending the list of available sessions.

```
package project02;

public class SessionInitial implements Runnable {
    ...

    public void listDownAvailableSession() throws IOException {
        ...
        if(!noAvailableSession) {
            ...
            toClient.writeUTF("(" + ChatServer.availableSessions.size()
                               + ") Back to Main Menu");

            clientSelectedSession();
        } else {

        }
    }

    public void clientSelectedSession() {

    }
}
```

16. In `clientSelectedSession` method, read the client choice and validate if the choice is valid. (Don't forget to throw `IOException`)

```
package project02;

public class SessionInitial implements Runnable {
    ...

    public void clientSelectedSession() throws IOException {
        boolean validSelectedSession = false;
        int selectedSession = fromClient.readInt();
        if(selectedSession >= 0 && selectedSession <
ChatServer.availableSessions.size()) {

        } else {
            if(selectedSession == ChatServer.availableSessions.size()) {

            } else {

            }
        }
    }
}
```



```

    }
}
}
}

```

17. If selected session is one of the available session, connect to the chat session and remove the chat session from the list to make it unavailable. If selected session is to go back to main menu, go back to main menu. And if invalid, send the list of available sessions again.

```

package project02;

public class SessionInitial implements Runnable {
    ...

    public void clientSelectedSession() throws IOException {
        boolean validSelectedSession = false;
        int selectedSession = fromClient.readInt();
        if(selectedSession >= 0 && selectedSession <
ChatServer.availableSessions.size()) {
            SessionHandler chatSession =
ChatServer.availableSessions.remove(selectedSession);
            validSelectedSession = true;
            toClient.writeBoolean(validSelectedSession);
            toClient.writeUTF("Connected to: " + chatSession.getSessionName());
            chatSession.connectClient2(client);
        } else {
            if(selectedSession == ChatServer.availableSessions.size()) {
                validSelectedSession = true;
                toClient.writeBoolean(validSelectedSession);
                toClient.writeUTF("Going back to main menu.");
                mainMenuLogic();
            } else {
                validSelectedSession = false;
                toClient.writeBoolean(validSelectedSession);
                toClient.writeUTF("Selected session does not exists.");
                listDownAvailableSession();
            }
        }
    }
}
}
}

```

18. Lets define the method `connectClient2` in `ServerHandler` class to add the client to an existing chat session.

```

package project02;

```

```

public class SessionHandler {
    ...

    public SessionHandler(int sessionId, Socket client1) {
        ...
    }

    public void connectClient2(Socket client) {

    }

    ...
}

```

19. Before we test, let's finish up the implementation in `listDownAvailableSession` method in `SessionInitial` class where if there are no available session, it will ask the client if they want to start new session, go back to main menu or try to list down available sessions again.

```

package project02;

public class SessionInitial implements Runnable {
    ...

    public void listDownAvailableSession() throws IOException {
        ...
        if(!noAvailableSession) {
            ...
        } else {
            toClient.writeUTF("No available session to choose.");
            toClient.writeUTF("Do you want to start new session? (y/n)");

            String decision = fromClient.readUTF();
            if(decision.equalsIgnoreCase("y")) {
                startANewSession();
            } else {
                toClient.writeUTF("Go to main menu? (y/n)");
                decision = fromClient.readUTF();
                if(decision.equalsIgnoreCase("y")) {
                    mainMenuLogic();
                } else {
                    listDownAvailableSession();
                }
            }
        }
    }
}

```

```
...  
}
```

20. Same idea for `ChatClient` class's, `listDownAvailableSession` method.

```
package project02;  
  
import java.io.DataInputStream;  
import java.io.DataOutputStream;  
import java.io.IOException;  
import java.net.ConnectException;  
import java.net.Socket;  
import java.net.UnknownHostException;  
import java.util.Scanner;  
  
public class ChatClient {  
    ...  
  
    public static void listDownAvailableSession() throws IOException {  
        ...  
        if(haveAvailableSession) {  
            ...  
        } else {  
            System.out.println(fromServer.readUTF());  
            System.out.println(fromServer.readUTF());  
            String startNewSession = scanner.nextLine();  
            toServer.writeUTF(startNewSession);  
  
            if(startNewSession.equalsIgnoreCase("y")) {  
                waitingForOtherClient();  
            } else {  
                System.out.println(fromServer.readUTF());  
                String goMainMenu = scanner.nextLine();  
                toServer.writeUTF(goMainMenu);  
  
                if(goMainMenu.equalsIgnoreCase("y")) {  
                    mainMenuLogic();  
                } else {  
                    listDownAvailableSession();  
                }  
            }  
        }  
    }  
    ...  
}
```

21. Run it, don't forget to run `ChatServer` first then run `ChatClient`. At the moment there are no available session. It should ask if the client wants to start a new session or not. If yes, it should start a new session and if no, it should ask if the client wants to go back to main menu or not.

```
Welcome to PB Chatting.
Please choose an option:
(1) Start a new chatting session
(2) Connect to available session

2
No available session to choose.
Do you want to start new session? (y/n)
y
New session started: 1 localhost 127.0.0.1
Waiting for other client to connect
Waiting for other client to connect
```

22. Lets start another `CharClient`. It should contain the new session created above. If the client choose the session, currently nothing happen. We are going to implement that next. **Don't forget to stop the ChatServer and the three ChatClient manually before continuing.**

```
Problems Javadoc Declaration Console Debug
ChatClient (1) [Java Application] /Users/jailanihar/p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_15.0.1.v20201027-0507/jre/bin/java (Mar 9, 2021, 1
Welcome to PB Chatting.
Please choose an option:
(1) Start a new chatting session
(2) Connect to available session

2
Please choose an option:
(0) 1 localhost 127.0.0.1
(1) Back to Main Menu
```

## Milestone:

Commit the changes made to the `src` folder with message "Lesson 2.7 - Choosing existing session".

Hint: Add the files to the staging area first, `git add <files>` and then proceed to commit, `git commit -m "Lesson 2.7 - Choosing existing session"`.

## Lesson 2.8 - Connecting to existing session and start chat

1. When the client choose an existing session, we access the list of sessions and retrieve the SessionHandler object. Then we call the `connectClient2` method to add the client to an existing session. That means two clients are in `SessionHandler` object.

```
package project02;

public class SessionInitial implements Runnable {
    ...

    public void clientSelectedSession() throws IOException {
        ...
        if(selectedSession >= 0 && selectedSession <
ChatServer.availableSessions.size()) {
            SessionHandler chatSession =
ChatServer.availableSessions.remove(selectedSession);
            validSelectedSession = true;
            toClient.writeBoolean(validSelectedSession);
            toClient.writeUTF("Connected to: " + chatSession.getSessionName());
            chatSession.connectClient2(client);
        } else {
            ...
        }
    }
}
```

2. Lets implement the logic for this. In `SessionHandler` class. First we need the instance variables for the second client in SessionHandler. Then we assign the socket passed to the method `connectClient2` with `client2` socket. Since there are another client connected to the chat session. Client 1 does not have to wait any longer. This will stop client 1 from waiting.

```
package project02;

public class SessionHandler {
    private Socket client1, client2;
    ...
    private DataInputStream fromClient1, fromClient2;
    private DataOutputStream toClient1, toClient2;

    ...

    public void connectClient2(Socket client2) {
        this.client2 = client2;
    }
}
```

```
        waitingForClient2 = false;
    }

    ...
}
```

3. Define a method called `startChat` and call the method in `startWaitingOtherClient`.

```
package project02;

public class SessionHandler {
    ...

    public void startWaitingOtherClient() {
        ...

        while(waitingForClient2) {
            ...
        }

        startChat();

        ...
    }

    public void startChat() {

    }

    ...
}
```

4. Send the details of client2 to client1. (Don't forget to throw IOException)

```

package project02;

public class SessionHandler {
    ...

    public void startChat() throws IOException {
        toClient1.writeUTF(client2.getInetAddress().getHostName());
        toClient1.writeUTF(client2.getInetAddress().getHostAddress());
    }

    ...
}

```

5. Create Data Stream objects for client2 socket. Then send client1 details to client2.

```

package project02;

public class SessionHandler {
    ...

    public void startChat() throws IOException {
        toClient1.writeUTF(client2.getInetAddress().getHostName());
        toClient1.writeUTF(client2.getInetAddress().getHostAddress());

        fromClient2 = new DataInputStream(client2.getInputStream());
        toClient2 = new DataOutputStream(client2.getOutputStream());

        toClient2.writeUTF(client1.getInetAddress().getHostName());
        toClient2.writeUTF(client1.getInetAddress().getHostAddress());
    }

    ...
}

```

6. Then its time to implement the chat logic where if client1 send data to the server, it will forward the data to client2 and vice versa.

```

package project02;

public class SessionHandler {
    ...

    public void startChat() throws IOException {
        toClient1.writeUTF(client2.getInetAddress().getHostName());

```

```

toClient1.writeUTF(client2.getInetAddress().getHostAddress());

fromClient2 = new DataInputStream(client2.getInputStream());
toClient2 = new DataOutputStream(client2.getOutputStream());

toClient2.writeUTF(client1.getInetAddress().getHostName());
toClient2.writeUTF(client1.getInetAddress().getHostAddress());

while(true) {
    if(fromClient1.available() > 0) {
        forwardMessageToClient2();
    }
    if(fromClient2.available() > 0) {
        forwardMessageToClient1();
    }
}

public void forwardMessageToClient1() throws IOException {
    String message = fromClient2.readUTF();
    try {
        toClient1.writeUTF(message);
    } catch(SocketException e) {

    }
}

public void forwardMessageToClient2() throws IOException {
    String message = fromClient1.readUTF();
    try {
        toClient2.writeUTF(message);
    } catch(SocketException e) {

    }
}

...
}

```

7. We need a way to identify if there are any issue with communicating with the other client. Let create two boolean instance variable with initial value of true to indicate message sent to other client is successful. If false, that means there is issue with the communication. We will use this to inform the other client.



```

package project02;

public class SessionHandler {
    ...
    private boolean successfullySentToClient1 = true;
    private boolean successfullySentToClient2 = true;
    ...
}

```

8. When sending data through a socket failed, it will throw `SocketException`. We can use this to change the respective boolean to false to indicate the communication problem. Every time we send data to client1, we also need to inform the current status of communication with client2 and vice versa.

```

package project02;

public class SessionHandler {
    ...

    public void forwardMessageToClient1() throws IOException {
        String message = fromClient2.readUTF();
        try {
            toClient1.writeUTF(message);
            toClient1.writeBoolean(successfullySentToClient2);
        } catch (SocketException e) {
            successfullySentToClient1 = false;
        }
    }

    public void forwardMessageToClient2() throws IOException {
        String message = fromClient1.readUTF();
        try {
            toClient2.writeUTF(message);
            toClient2.writeBoolean(successfullySentToClient1);
        } catch (SocketException e) {
            successfullySentToClient2 = false;
        }
    }

    ...
}

```

9. Lets implement what happens after the server knows there is an issue with the connection with one of the client. It will immediately stop the chat session as well.

```

package project02;

public class SessionHandler {
    ...

    public void startChat() throws IOException {
        ...

        while(true) {
            if(!messageSentToClient1) {
                notifyProblemToClient2();
                break;
            }
            if(!messageSentToClient2) {
                notifyProblemToClient1();
                break;
            }
            if(fromClient1.available() > 0) {
                ...
            }
        }
    }

    public void notifyProblemToClient1() throws IOException {
        toClient1.writeUTF("Other Client has disconnected.");
        toClient1.writeBoolean(successfullySentToClient2);
    }

    public void notifyProblemToClient2() throws IOException {
        toClient2.writeUTF("Other Client has disconnected.");
        toClient2.writeBoolean(successfullySentToClient1);
    }

    ...
}

```

10. In `ChatClient`, implement the chat logic for the client. First we need to make the waiting client to call the `chatLogic` method after other client has connected.

```

package project02;

public class ChatClient {
    ...

    public static void waitingForOtherClient() throws IOException {
        System.out.println(fromServer.readUTF());
        boolean waiting = true;
    }
}

```

```

while(waiting) {
    try {
        Thread.sleep(500);
        waiting = fromServer.readBoolean();
        System.out.println(fromServer.readUTF());
    } catch (InterruptedException e) {
        System.out.println("Waiting was interrupted.");
    }
}

chatLogic();
}

...
}

```

11. Implement the chat logic for the client. First read the other client details sent by the server. (Don't forget to throw IOException)

```

package project02;

public class ChatClient {
    ...

    public static void chatLogic() throws IOException {
        String otherClientHostName = fromServer.readUTF();
        String otherClientIPAddress = fromServer.readUTF();
        System.out.println("Client connect, say hi to "
            + otherClientHostName + " " + otherClientIPAddress);
    }
}

```

12. Lets have a boolean class variable called `chatActive` to indicate if the chat session is still active or not. Then when chatActive is true, let the client to send message multiple times. When chatActive is false, print message to indicate the chat session stopped.

```

package project02;

public class ChatClient {
    ...
    private static boolean chatActive = true;

    public static void chatLogic() throws IOException {
        String otherClientHostName = fromServer.readUTF();
        String otherClientIPAddress = fromServer.readUTF();
    }
}

```

```

        System.out.println("Client connect, say hi to "
                           + otherClientHostName + " " + otherClientIPAddress);

        while(chatActive) {
            String sendMessage = scanner.nextLine();
            toServer.writeUTF(sendMessage);
        }
        System.out.println("This chat session stopped. Please restart
application.");
    }
}

```

13. Due to the blocking statement of `scanner.nextLine()`. We cannot implement the read data from server in the same while loop. We need to implement it in another loop and also in a separate thread.

```

package project02;

public class ChatClient {
    ...

    public static void chatLogic() throws IOException {
        ...

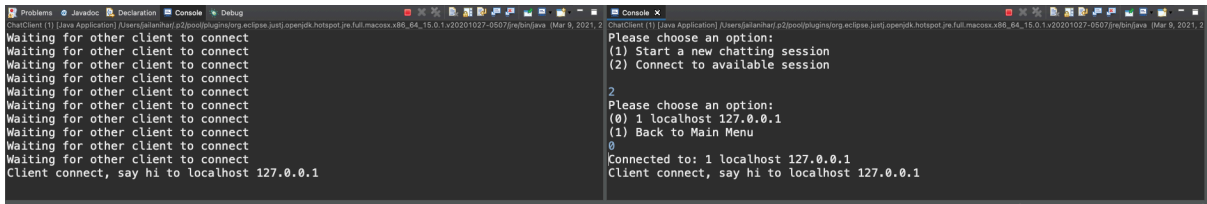
        new Thread(new Runnable() {
            @Override
            public void run() {
                while(chatActive) {
                    try {
                        if(fromServer.available() > 0) {
                            String receiveMsg = fromServer.readUTF();
                            System.out.println(">>> " + receiveMsg);
                            chatActive = fromServer.readBoolean();
                        }
                    } catch (IOException e) {
                    }
                }
            }
        }).start();

        while(chatActive) {
            String sendMessage = scanner.nextLine();
            toServer.writeUTF(sendMessage);
        }
        System.out.println("This chat session stopped. Please restart
application.");
    }
}

```

```
}
```

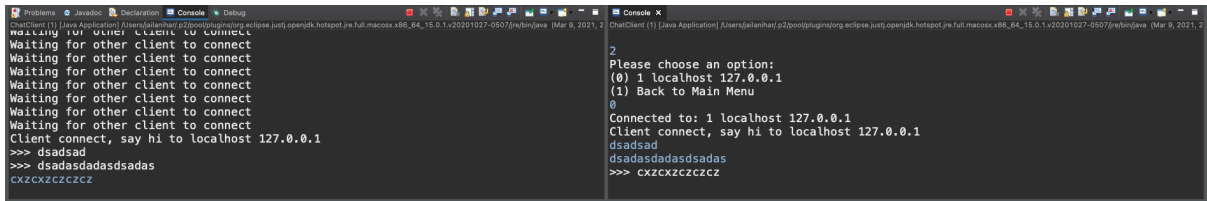
14. Run it, try the chat feature. This is when 2 clients are connected to the same chat session.



```
Problems | Javadoc | Declaration | Console | Debug
ChatClient (1) [Java Application] /Users/jalanhar/p2/pool/plugins/org.eclipse.justi.openjdk.hotspot.jre.full.macosx.x86_64_15.0.1.v20201027-0507/jre/bin/java (Mar 9, 2021, 2
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Client connect, say hi to localhost 127.0.0.1

ChatClient X
ChatClient (1) [Java Application] /Users/jalanhar/p2/pool/plugins/org.eclipse.justi.openjdk.hotspot.jre.full.macosx.x86_64_15.0.1.v20201027-0507/jre/bin/java (Mar 9, 2021, 2
Please choose an option:
(1) Start a new chatting session
(2) Connect to available session
2
Please choose an option:
(0) 1 localhost 127.0.0.1
(1) Back to Main Menu
0
Connected to: 1 localhost 127.0.0.1
Client connect, say hi to localhost 127.0.0.1
```

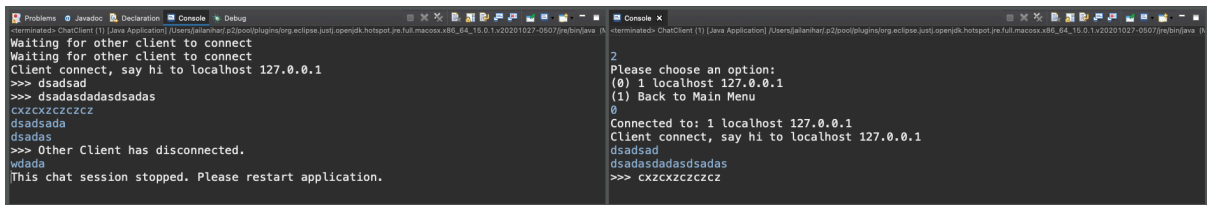
15. They can send message to each other.



```
Problems | Javadoc | Declaration | Console | Debug
ChatClient (1) [Java Application] /Users/jalanhar/p2/pool/plugins/org.eclipse.justi.openjdk.hotspot.jre.full.macosx.x86_64_15.0.1.v20201027-0507/jre/bin/java (Mar 9, 2021, 2
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Client connect, say hi to localhost 127.0.0.1
>>> dsadsad
>>> dsadasdadasdsadas
CXZCXZCZCZCZ

ChatClient X
ChatClient (1) [Java Application] /Users/jalanhar/p2/pool/plugins/org.eclipse.justi.openjdk.hotspot.jre.full.macosx.x86_64_15.0.1.v20201027-0507/jre/bin/java (Mar 9, 2021, 2
Please choose an option:
(0) 1 localhost 127.0.0.1
(1) Back to Main Menu
0
Connected to: 1 localhost 127.0.0.1
Client connect, say hi to localhost 127.0.0.1
dsadsad
dsadasdadasdsadas
>>> CXZCXZCZCZCZ
```

16. If other client has disconnected, it will notify the client itself.



```
Problems | Javadoc | Declaration | Console | Debug
ChatClient (1) [Java Application] /Users/jalanhar/p2/pool/plugins/org.eclipse.justi.openjdk.hotspot.jre.full.macosx.x86_64_15.0.1.v20201027-0507/jre/bin/java (Mar 9, 2021, 2
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Waiting for other client to connect
Client connect, say hi to localhost 127.0.0.1
>>> dsadsad
>>> dsadasdadasdsadas
dsadasda
dsadasda
>>> Other Client has disconnected.
wdada
This chat session stopped. Please restart application.

ChatClient X
ChatClient (1) [Java Application] /Users/jalanhar/p2/pool/plugins/org.eclipse.justi.openjdk.hotspot.jre.full.macosx.x86_64_15.0.1.v20201027-0507/jre/bin/java (Mar 9, 2021, 2
Please choose an option:
(0) 1 localhost 127.0.0.1
(1) Back to Main Menu
0
Connected to: 1 localhost 127.0.0.1
Client connect, say hi to localhost 127.0.0.1
dsadsad
dsadasdadasdsadas
>>> CXZCXZCZCZCZ
```

## Milestone:

Commit the changes made to the `src` folder with message "Lesson 2.8 - Connecting to existing session and start chat".

Hint: Add the files to the staging area first, `git add <files>` and then proceed to commit, `git commit -m "Lesson 2.8 - Connecting to existing session and start chat"`.