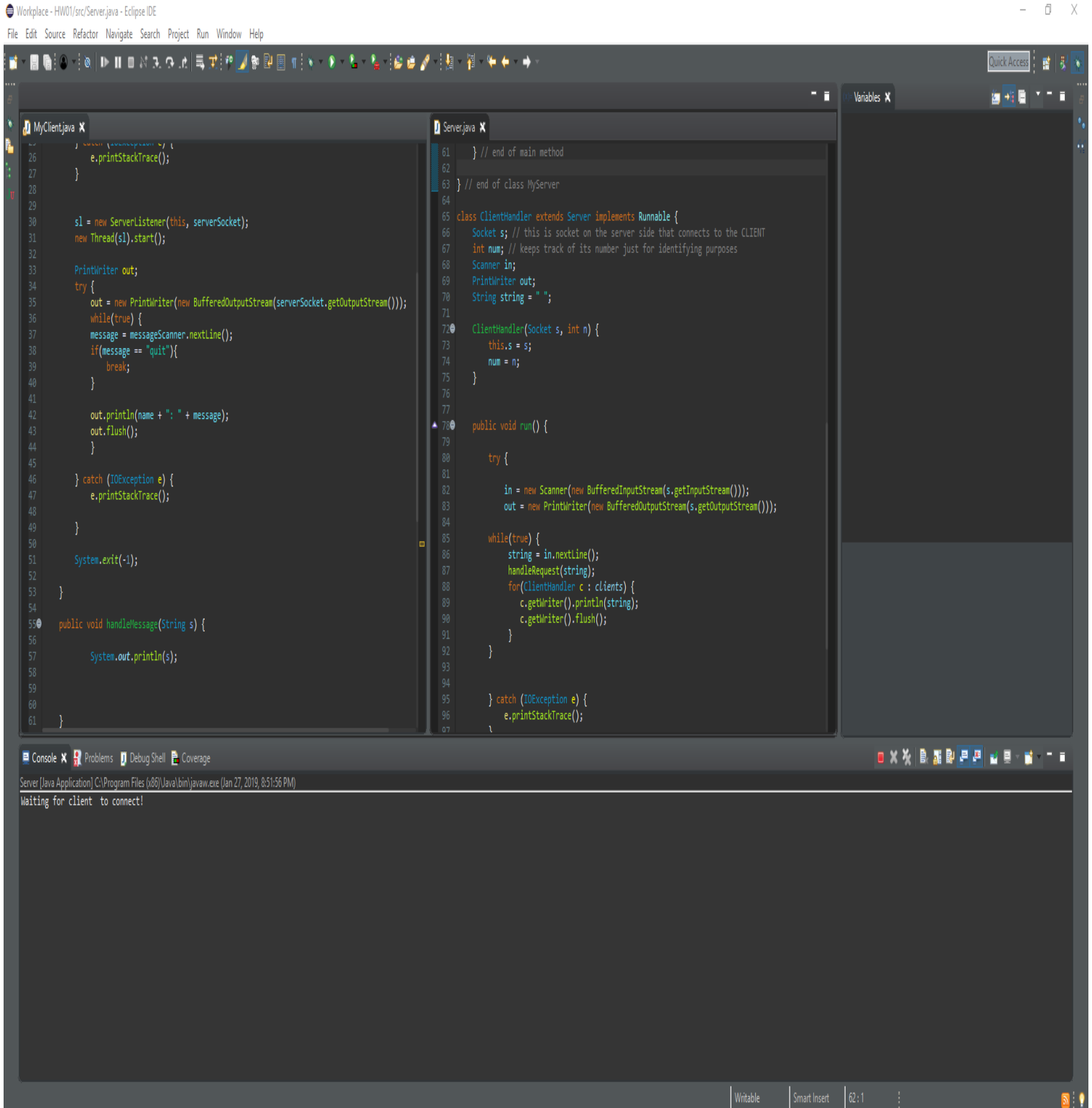
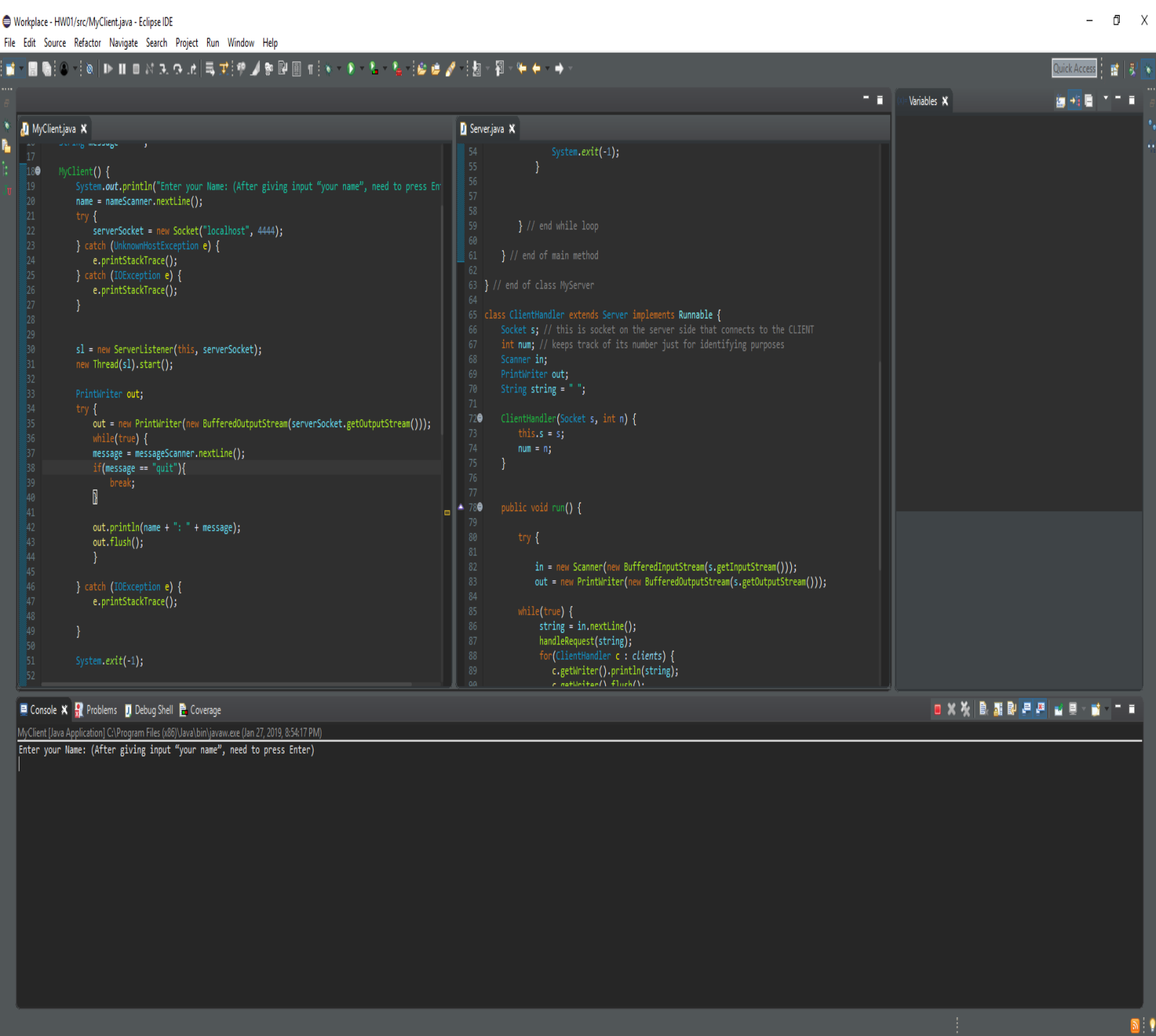


To come up with the solution, I used the server and client code that was given to us in the lab. I adjusted ListServer.java and ListClient.java. Within MyClient.java, I added one scanner that would scan for a name and one that would scan for the messages being sent. Within the Server.java, I created an ArrayList that would will populate with the clients that would be connected to the server. Once a message is received it will print out (server side) username: message (shown in screenshots below). It will then run through a for loop that will iterate through the ArrayList of clients and send the message to each client connected. I used the lab code for the way it had the server and client connect, and to learn how a client contacts with a server. As seen in the screenshots there is no GUI used and it is all in the console. Below are Screenshots of the needed output:

1) Screenshot of server waiting for a client to connect



2) Screenshot of Client requesting a username (still not connected to the server)



3) Screenshot of the server after two clients connect after inputting their username

The screenshot shows the Eclipse IDE with two Java files open: `MyClient.java` and `Server.java`. The `MyClient.java` file contains a `MyClient` class with a `main` method that prompts the user for a name and sends it to the server. The `Server.java` file contains a `MyServer` class that listens for connections and a `ClientHandler` class that processes incoming messages. The console at the bottom shows the server's output, indicating that it is waiting for clients to connect and has successfully connected to two clients.

```
MyClient.java
17
18
19 MyClient() {
20     System.out.println("Enter your Name: (After giving input \"your name\", need to press Enter)");
21     name = nameScanner.nextLine();
22     try {
23         serverSocket = new Socket("localhost", 4444);
24     } catch (UnknownHostException e) {
25         e.printStackTrace();
26     } catch (IOException e) {
27         e.printStackTrace();
28     }
29
30     sl = new ServerListener(this, serverSocket);
31     new Thread(sl).start();
32
33     PrintWriter out;
34     try {
35         out = new PrintWriter(new BufferedOutputStream(serverSocket.getOutputStream()));
36         while(true) {
37             message = messageScanner.nextLine();
38             if(message == "quit"){
39                 break;
40             }
41             out.println(name + ": " + message);
42             out.flush();
43         }
44     } catch (IOException e) {
45         e.printStackTrace();
46     }
47
48     System.exit(-1);
49
50
51
52

Server.java
54     System.exit(-1);
55 }
56
57
58 // end while loop
59
60 // end of main method
61
62 // end of class MyServer
63
64
65 class ClientHandler extends Server implements Runnable {
66     Socket s; // this is socket on the server side that connects to the CLIENT
67     int num; // keeps track of its number just for identifying purposes
68     Scanner in;
69     PrintWriter out;
70     String string = " ";
71
72     ClientHandler(Socket s, int n) {
73         this.s = s;
74         num = n;
75     }
76
77     public void run() {
78         try {
79             in = new Scanner(new BufferedInputStream(s.getInputStream()));
80             out = new PrintWriter(new BufferedOutputStream(s.getOutputStream()));
81
82             while(true) {
83                 string = in.nextLine();
84                 handleRequest(string);
85                 for(ClientHandler c : clients) {
86                     c.getWriter().println(string);
87                     c.getWriter().flush();
88                 }
89             }
90         }
91     }
92 }
```

Console Output:

```
Server [Java Application] C:\Program Files (x86)\Java\bin\java.exe (Jan 27, 2019, 8:55:37 PM)
Waiting for client to connect!
Server got connected to a client
Waiting for client to connect!
Server got connected to a client
Waiting for client to connect!
```

4) Screenshot of the server console after I sent a message

The screenshot shows the Eclipse IDE with two Java files open: `MyClient.java` and `Server.java`. The `MyClient.java` file contains a `MyClient` class with a `main` method that prompts the user for a name and a message, then sends the message to the server. The `Server.java` file contains a `MyServer` class with a `run` method that listens for client connections and handles requests. The console output shows the server's behavior: it waits for a client to connect, gets connected to a client, and then prints the message received from the client.

```
17
18 MyClient() {
19     System.out.println("Enter your Name: (After giving input 'your name', need to press Enter)");
20     name = nameScanner.nextLine();
21     try {
22         serverSocket = new Socket("localhost", 4444);
23         catch (UnknownHostException e) {
24             e.printStackTrace();
25         } catch (IOException e) {
26             e.printStackTrace();
27         }
28
29     s1 = new ServerListener(this, serverSocket);
30     new Thread(s1).start();
31
32     PrintWriter out;
33     try {
34         out = new PrintWriter(new BufferedOutputStream(serverSocket.getOutputStream()));
35         while(true) {
36             message = messageScanner.nextLine();
37             if(message == "quit"){
38                 break;
39             }
40
41             out.println(name + ": " + message);
42             out.flush();
43         }
44     } catch (IOException e) {
45         e.printStackTrace();
46     }
47
48     System.exit(-1);
49 }
50
51
52
53
54 System.exit(-1);
55 }
56
57
58 } // end while loop
59
60 } // end of main method
61
62 // end of class MyServer
63
64
65 class ClientHandler extends Server implements Runnable {
66     Socket s; // this is socket on the server side that connects to the CLIENT
67     int num; // keeps track of its number just for identifying purposes
68     Scanner in;
69     PrintWriter out;
70     String string = "";
71
72     ClientHandler(Socket s, int n) {
73         this.s = s;
74         num = n;
75     }
76
77     public void run() {
78         try {
79
80             in = new Scanner(new BufferedInputStream(s.getInputStream()));
81             out = new PrintWriter(new BufferedOutputStream(s.getOutputStream()));
82
83             while(true) {
84                 string = in.nextLine();
85                 handleRequest(string);
86                 for(ClientHandler c : clients) {
87                     c.getWriter().println(string);
88                     c.getWriter().flush();
89                 }
90             }
91         } catch (IOException e) {
92             e.printStackTrace();
93         }
94     }
95 }
```

Console Output:

```
Server [Java Application] C:\Program Files (x86)\Java\bin\javaw.exe (Jan 27, 2019, 8:55:37 PM)
Waiting for client to connect!
Server got connected to a client
Waiting for client to connect!
Server got connected to a client
Waiting for client to connect!
(server side) Abdalla: I finished the first homework!
```

5) Screenshot of the other client receiving the message

The screenshot displays the Eclipse IDE with two Java files open: `MyClient.java` and `Server.java`. The `MyClient.java` file contains a `MyClient` class that prompts the user for a name and then connects to a server at `localhost:4444`. The `Server.java` file contains a `MyServer` class that listens for incoming connections and handles them using a `ClientHandler` class. The console at the bottom shows the output of the application, indicating that the client has received a message from the server.

```
MyClient.java
17
18 MyClient() {
19     System.out.println("Enter your Name: (After giving input \"your name\", need to press Enter)");
20     name = nameScanner.nextLine();
21     try {
22         serverSocket = new Socket("localhost", 4444);
23     } catch (UnknownHostException e) {
24         e.printStackTrace();
25     } catch (IOException e) {
26         e.printStackTrace();
27     }
28
29     sl = new ServerListener(this, serverSocket);
30     new Thread(sl).start();
31
32     PrintWriter out;
33     try {
34         out = new PrintWriter(new BufferedOutputStream(serverSocket.getOutputStream()));
35         while(true) {
36             message = messageScanner.nextLine();
37             if(message == "quit"){
38                 break;
39             }
40
41             out.println(name + ": " + message);
42             out.flush();
43         }
44     } catch (IOException e) {
45         e.printStackTrace();
46     }
47
48     System.exit(-1);
49 }
50
51
52

Server.java
54     System.exit(-1);
55 }
56
57
58     } // end while loop
59
60     } // end of main method
61
62 } // end of class MyServer
63
64
65 class ClientHandler extends Thread implements Runnable {
66     Socket s; // this is socket on the server side that connects to the CLIENT
67     int num; // keeps track of its number just for identifying purposes
68     Scanner in;
69     PrintWriter out;
70     String string = " ";
71
72     ClientHandler(Socket s, int n) {
73         this.s = s;
74         num = n;
75     }
76
77     public void run() {
78         try {
79
80             in = new Scanner(new BufferedInputStream(s.getInputStream()));
81             out = new PrintWriter(new BufferedOutputStream(s.getOutputStream()));
82
83             while(true) {
84                 string = in.nextLine();
85                 handleRequest(string);
86                 for(ClientHandler c : clients) {
87                     c.getWriter().println(string);
88                     c.getWriter().flush();
89                 }
90             }
91         } catch (IOException e) {
92             e.printStackTrace();
93         }
94     }
95 }
96
```

Console Output:

```
MyClient [Java Application] C:\Program Files (x86)\Java\bin\javaw.exe (Jan 27, 2019, 8:55:44 PM)
Enter your Name: (After giving input "your name", need to press Enter)
COW S 319
Abdalla: I finished the first homework!
```

6) Screenshot of the server after the second client replies

The screenshot shows the Eclipse IDE with two Java files open: `MyClient.java` and `Server.java`. The `MyClient.java` file contains a `MyClient` class that sets up a `ServerSocket` on port 4444 and a `ServerListener` to handle incoming connections. The `Server.java` file contains a `MyServer` class that implements `Runnable` and handles client requests. The console window at the bottom shows the server's output, including messages about client connections and the received messages.

```
MyClient.java
17
18 MyClient() {
19     System.out.println("Enter your Name: (After giving input \"your name\", need to press Enter)");
20     name = nameScanner.nextLine();
21     try {
22         serverSocket = new Socket("localhost", 4444);
23     } catch (UnknownHostException e) {
24         e.printStackTrace();
25     } catch (IOException e) {
26         e.printStackTrace();
27     }
28
29     sl = new ServerListener(this, serverSocket);
30     new Thread(sl).start();
31
32     PrintWriter out;
33     try {
34         out = new PrintWriter(new BufferedOutputStream(serverSocket.getOutputStream()));
35         while(true) {
36             message = messageScanner.nextLine();
37             if(message == "quit"){
38                 break;
39             }
40
41             out.println(name + ": " + message);
42             out.flush();
43         }
44     } catch (IOException e) {
45         e.printStackTrace();
46     }
47
48     System.exit(-1);
49
50
51
52

Server.java
54     System.exit(-1);
55 }
56
57
58 // end while loop
59
60 // end of main method
61
62 // end of class MyServer
63
64
65 class ClientHandler extends Server implements Runnable {
66     Socket s; // this is socket on the server side that connects to the CLIENT
67     int num; // keeps track of its number just for identifying purposes
68     Scanner in;
69     PrintWriter out;
70     String string = " ";
71
72     ClientHandler(Socket s, int n) {
73         this.s = s;
74         num = n;
75     }
76
77
78     public void run() {
79
80         try {
81
82             in = new Scanner(new BufferedInputStream(s.getInputStream()));
83             out = new PrintWriter(new BufferedOutputStream(s.getOutputStream()));
84
85             while(true) {
86                 string = in.nextLine();
87                 handleRequest(string);
88                 for(ClientHandler c : clients) {
89                     c.getWriter().println(string);
90                     c.getWriter().flush();
91                 }
92             }
93         }
94     }
95 }
```

Console [Java Application] C:\Program Files (x86)\Java\bin\javaw.exe (Jan 27, 2019, 8:55:37 PM)

```
Waiting for client to connect!
Server got connected to a client
Waiting for client to connect!
Server got connected to a client
Waiting for client to connect!
(server side) Abdalla: I finished the first homework!
(server side) CON S 319: That's great! You will get a hundred percent!
```

7) Screenshot of the first client after the second client responds

The screenshot shows the Eclipse IDE with two Java files open: `MyClient.java` and `Server.java`. The `MyClient.java` file contains a `MyClient` class that prompts the user for a name and then sends a message to the server. The `Server.java` file contains a `MyServer` class that listens for connections and handles them using a `ClientHandler` class. The console window at the bottom shows the output of the application, including the user's input and the server's response.

```
MyClient.java
17
18 MyClient() {
19     System.out.println("Enter your Name: (After giving input 'your name', need to press Enter)");
20     name = nameScanner.nextLine();
21     try {
22         serverSocket = new Socket("localhost", 4444);
23         catch (UnknownHostException e) {
24             e.printStackTrace();
25         } catch (IOException e) {
26             e.printStackTrace();
27         }
28
29         sl = new ServerListener(this, serverSocket);
30         new Thread(sl).start();
31
32         PrintWriter out;
33         try {
34             out = new PrintWriter(new BufferedOutputStream(serverSocket.getOutputStream()));
35             while(true) {
36                 message = messageScanner.nextLine();
37                 if(message == "quit"){
38                     break;
39                 }
40
41                 out.println(name + ": " + message);
42                 out.flush();
43             }
44         } catch (IOException e) {
45             e.printStackTrace();
46         }
47     }
48
49     System.exit(-1);
50
51
52

Server.java
54     System.exit(-1);
55 }
56
57
58 // end while loop
59 } // end of main method
60
61 // end of class MyServer
62
63
64 class ClientHandler extends Server implements Runnable {
65     Socket s; // this is socket on the server side that connects to the CLIENT
66     int num; // keeps track of its number just for identifying purposes
67     Scanner in;
68     PrintWriter out;
69     String string = " ";
70
71     ClientHandler(Socket s, int n) {
72         this.s = s;
73         num = n;
74     }
75
76     public void run() {
77         try {
78             in = new Scanner(new BufferedInputStream(s.getInputStream()));
79             out = new PrintWriter(new BufferedOutputStream(s.getOutputStream()));
80
81             while(true) {
82                 string = in.nextLine();
83                 handleRequest(string);
84                 for(ClientHandler c : clients) {
85                     c.getWriter().println(string);
86                     c.getWriter().flush();
87                 }
88             }
89         } catch (IOException e) {
90             e.printStackTrace();
91         }
92     }
93 }
```

Console

```
MyClient [Java Application] C:\Program Files (x86)\Java\bin\javaw.exe (Jan 27, 2019, 8:55:39 PM)
Enter your Name: (After giving input "your name", need to press Enter)
Abdalla
I finished the first homework!
Abdalla: I finished the first homework!
COM S 319: That's great! You will get a hundred percent!
```