2023/2024(1)
EF234302 Object Oriented Programming

Lecture #12

Socket & Team Project

Misbakhul Munir IRFAN SUBAKTI
司馬伊凡
Мисбакхул Мунир Ирфан Субакти

Socket: What is that? (https://docs.oracle.com/javase/tutorial/networking/sockets)

- A socket is one endpoint of a two-way communication link between two programs running on the network.
- A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.
- The `java.net` package in the Java platform provides a class, `Socket`, that implements one side of a two-way connection between your Java program and another program on the network.

Socket: Explanation (https://docs.oracle.com/javase/tutorial/networking/sockets)

- The `Socket` class sits on top of a platform-dependent implementation, hiding the details of any particular system from your Java program. By using the `java.net.Socket` class instead of relying on native code, your Java programs can communicate over the network in a platform-independent fashion.
- Additionally, `java.net` includes the `ServerSocket` class, which implements a socket that servers can use to listen for and accept connections to clients.
- If you are trying to connect to the Web, the `URL` class and related classes (`URLConnection`, `URLEncoder`) are probably more appropriate than the socket classes. In fact, URLs are a relatively high-level connection to the Web and use sockets as part of the underlying implementation.

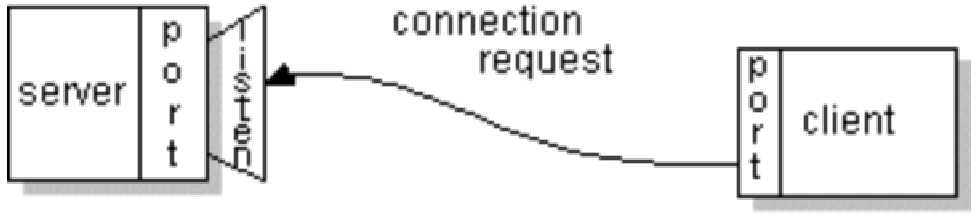
Socket: How it work? (https://www.ibm.com/docs/en/i/7.3?topic=programming-how-sockets-work)

- Sockets are commonly used for *client* and *server* interaction.
- Typical system configuration places the server on one machine, with the clients on other machines.
- The clients connect to the server, exchange information, and then disconnect.

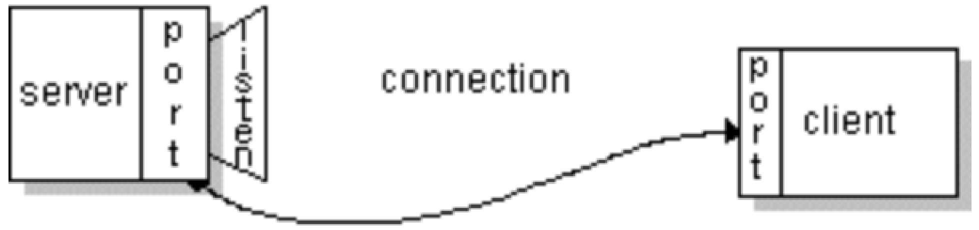
Socket: More about (https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html)

- Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.
- On the client-side: The client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.

Socket: More about (continued)



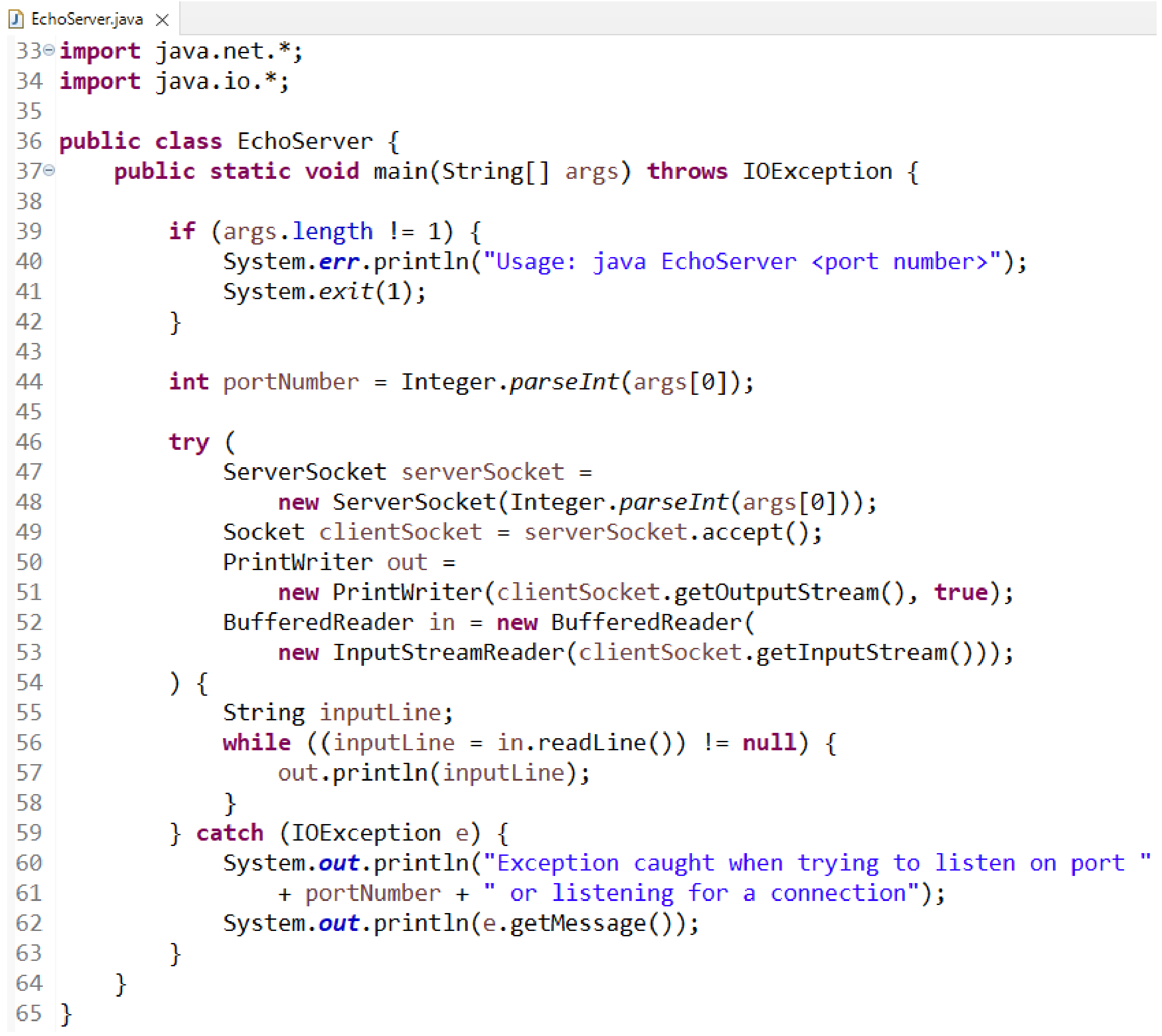
- If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.



Socket: More about (continued)

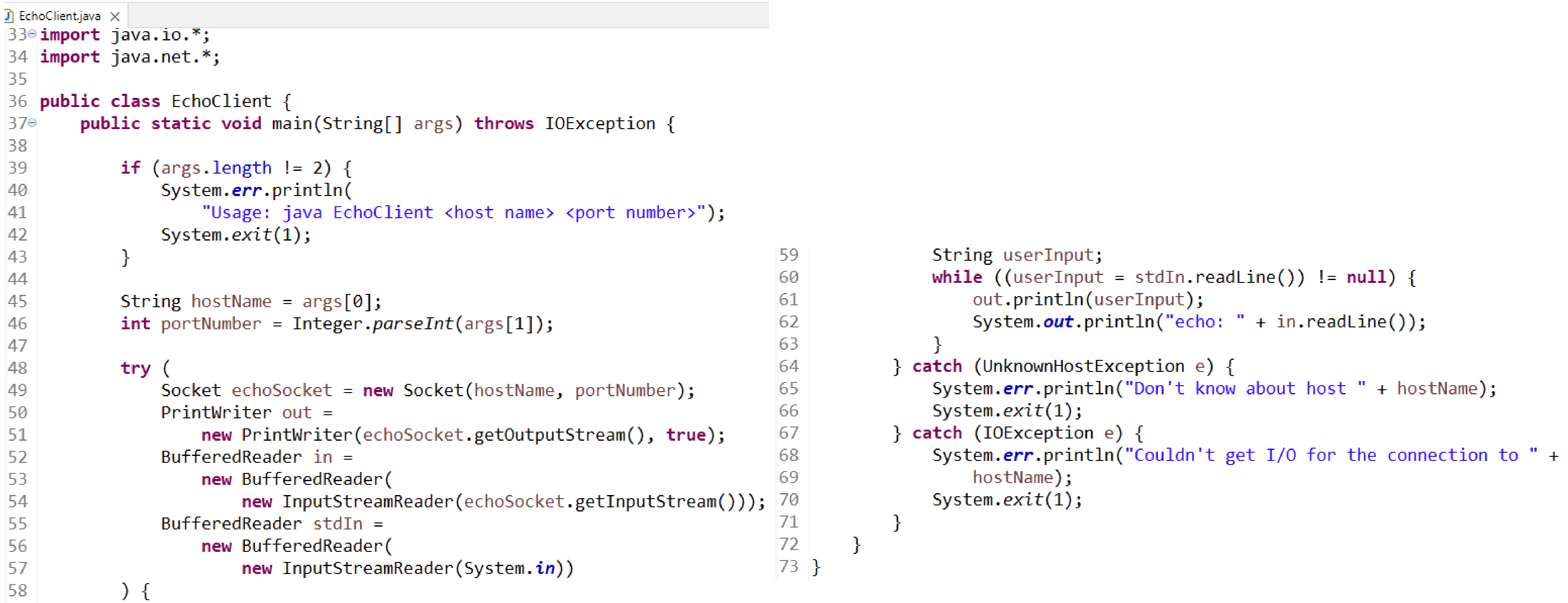
- On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server.
- The client and server can now communicate by writing to or reading from their sockets.

Socket: Reading from & writing to (https://docs.oracle.com/javase/tutorial/networking/sockets)



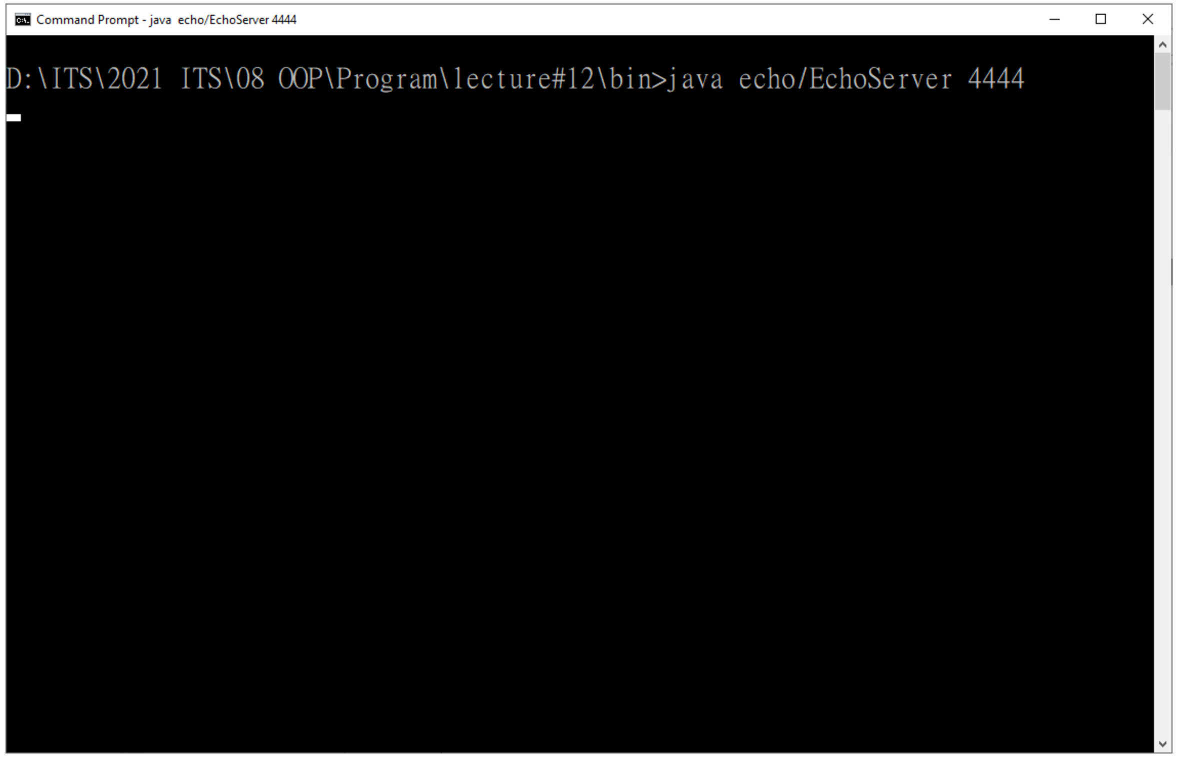
```java
import java.net.*;
import java.io.*;

public class EchoServer {
    public static void main(String[] args) throws IOException {

        if (args.length != 1) {
            System.err.println("Usage: java EchoServer <port number>");
            System.exit(1);
        }

        int portNumber = Integer.parseInt(args[0]);

        try (
            ServerSocket serverSocket =
                new ServerSocket(Integer.parseInt(args[0]));
            Socket clientSocket = serverSocket.accept();
            PrintWriter out =
                new PrintWriter(clientSocket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
        ) {
            String inputLine;
            while ((inputLine = in.readLine()) != null) {
                out.println(inputLine);
            }
        } catch (IOException e) {
            System.out.println("Exception caught when trying to listen on port "
                + portNumber + " or listening for a connection");
            System.out.println(e.getMessage());
        }
    }
}
```

Socket: Reading from & writing to (continued)

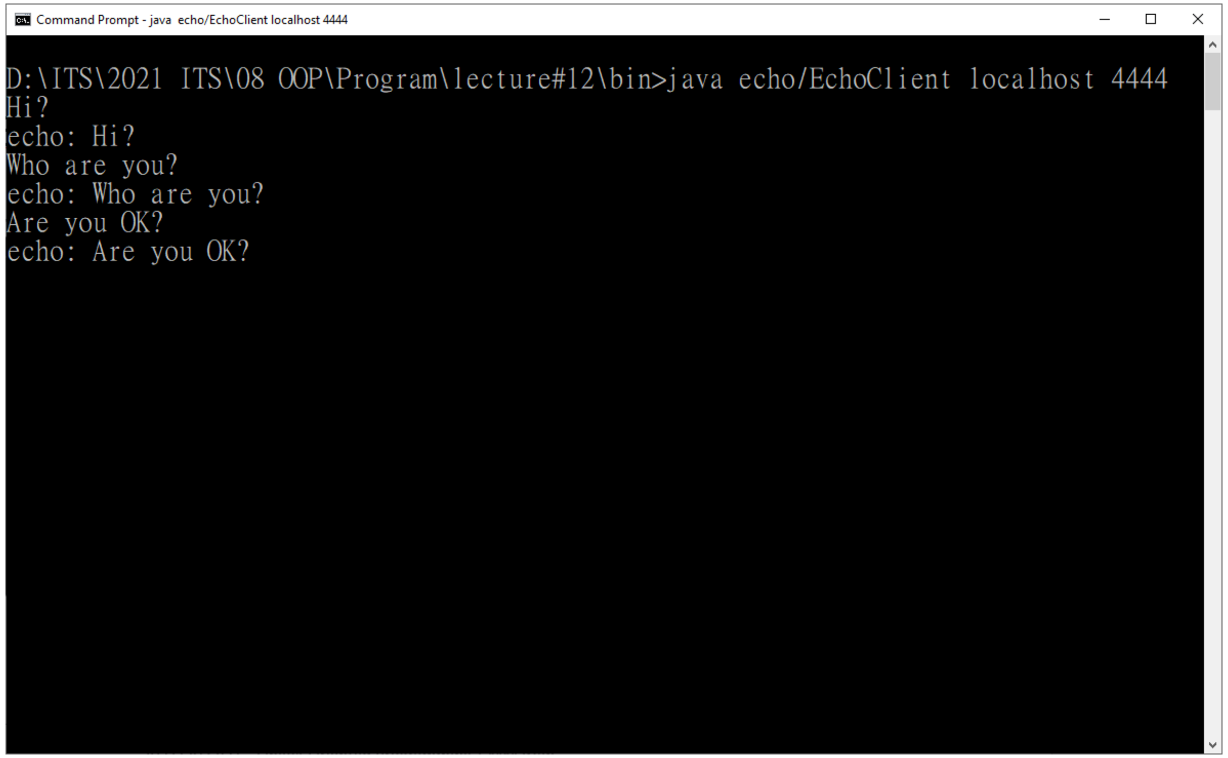

EchoClient.java

```java
import java.io.*;
import java.net.*;

public class EchoClient {
    public static void main(String[] args) throws IOException {

        if (args.length != 2) {
            System.err.println(
                "Usage: java EchoClient <host name> <port number>");
            System.exit(1);
        }

        String hostName = args[0];
        int portNumber = Integer.parseInt(args[1]);

        try (
            Socket echoSocket = new Socket(hostName, portNumber);
            PrintWriter out =
                new PrintWriter(echoSocket.getOutputStream(), true);
            BufferedReader in =
                new BufferedReader(
                    new InputStreamReader(echoSocket.getInputStream()));
            BufferedReader stdIn =
                new BufferedReader(
                    new InputStreamReader(System.in))
        ) {
            String userInput;
            while ((userInput = stdIn.readLine()) != null) {
                out.println(userInput);
                System.out.println("echo: " + in.readLine());
            }
        } catch (UnknownHostException e) {
            System.err.println("Don't know about host " + hostName);
            System.exit(1);
        } catch (IOException e) {
            System.err.println("Couldn't get I/O for the connection to " +
                hostName);
            System.exit(1);
        }
    }
}
```

`EchoServer` & `EchoClient`: Output


Command Prompt - java echo/EchoServer 4444

```
D:\ITS\2021 ITS\08 OOP\Program\lecture#12\bin>java echo/EchoServer 4444
```

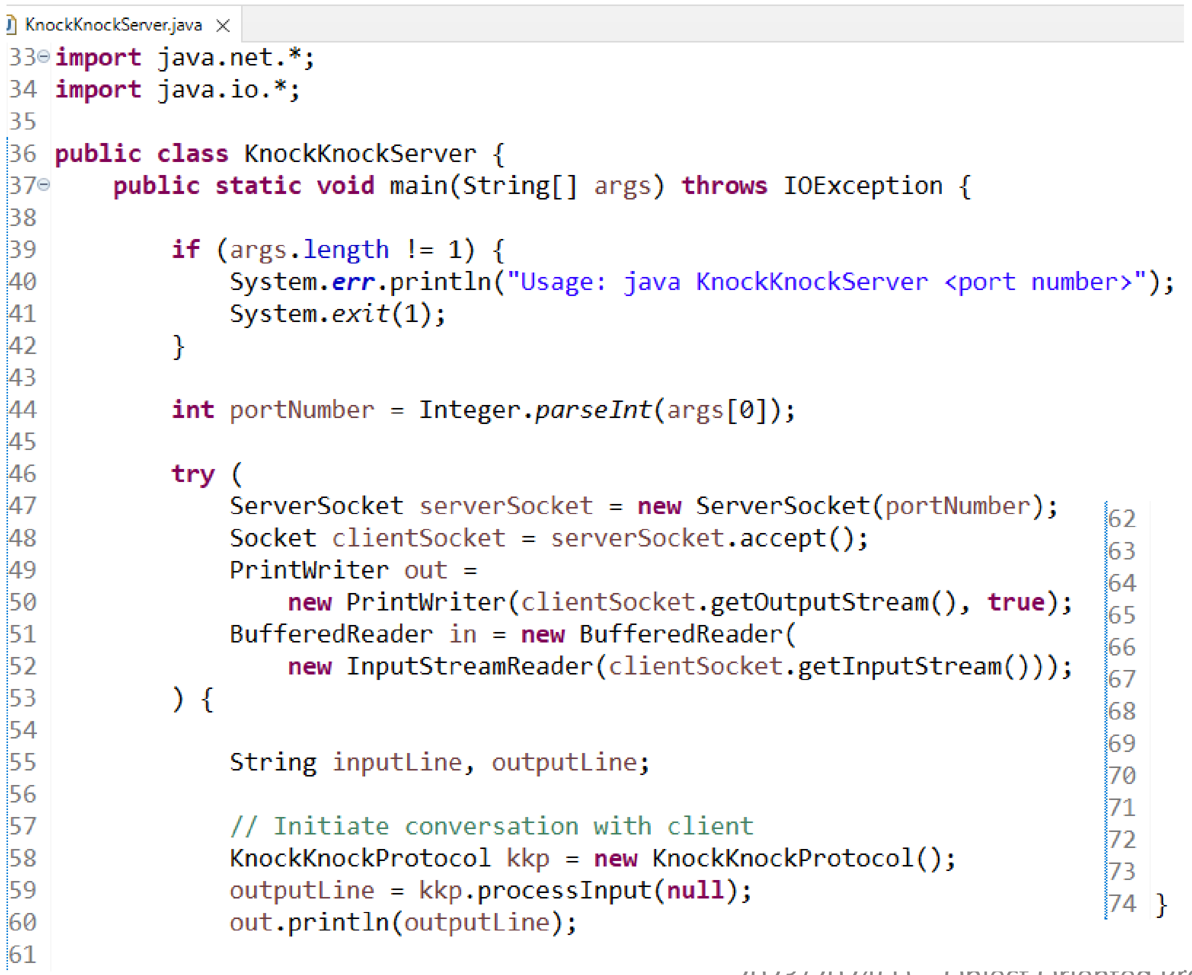

Command Prompt - java echo/EchoClient localhost 4444

```
D:\ITS\2021 ITS\08 OOP\Program\lecture#12\bin>java echo/EchoClient localhost 4444
Hi?
echo: Hi?
Who are you?
echo: Who are you?
Are you OK?
echo: Are you OK?
```

`EchoServer` & `EchoClient`: Explanation

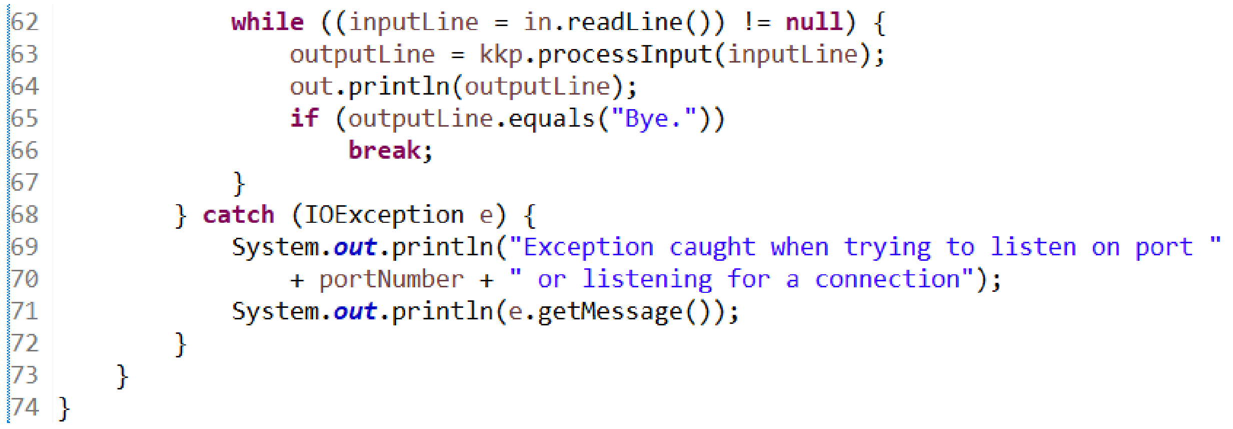
- This client program is straightforward and simple because the echo server implements a simple protocol. The client sends text to the server, and the server echoes it back. When your client programs are talking to a more complicated server such as an HTTP server, your client program will also be more complicated. However, the basics are much the same as they are in this program:
    1. Open a socket.
    2. Open an input stream and output stream to the socket.
    3. Read from and write to the stream according to the server's protocol.
    4. Close the streams.
    5. Close the socket.
- Only step 3 differs from client to client, depending on the server. The other steps remain largely the same.

Knock Knock: Server (https://docs.oracle.com/javase/tutorial/networking/sockets)



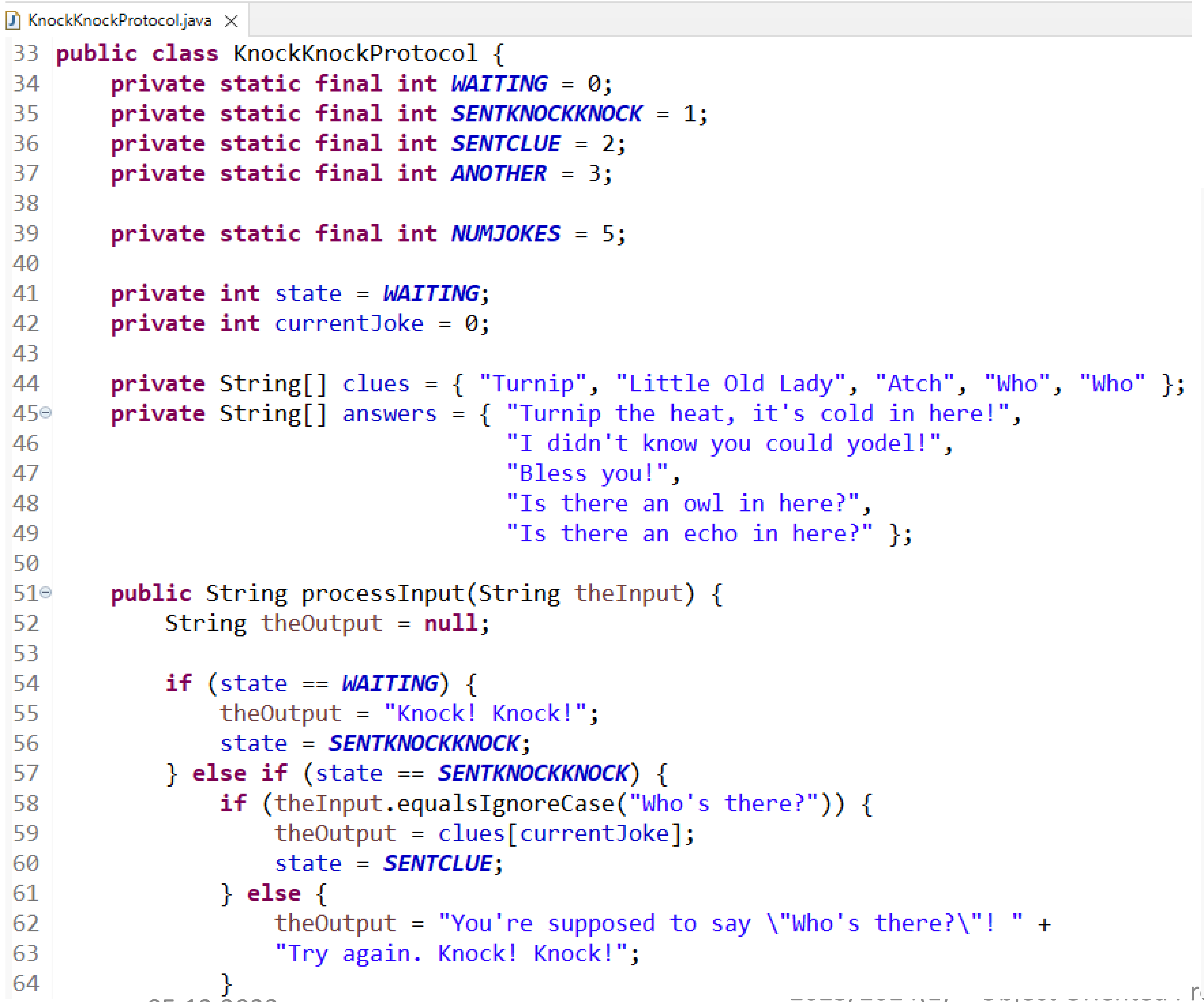
KnockKnockServer.java

```java
import java.net.*;
import java.io.*;

public class KnockKnockServer {
    public static void main(String[] args) throws IOException {

        if (args.length != 1) {
            System.err.println("Usage: java KnockKnockServer <port number>");
            System.exit(1);
        }

        int portNumber = Integer.parseInt(args[0]);

        try (
            ServerSocket serverSocket = new ServerSocket(portNumber);
            Socket clientSocket = serverSocket.accept();
            PrintWriter out =
                new PrintWriter(clientSocket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
        ) {

            String inputLine, outputLine;

            // Initiate conversation with client
            KnockKnockProtocol kkp = new KnockKnockProtocol();
            outputLine = kkp.processInput(null);
            out.println(outputLine);

```

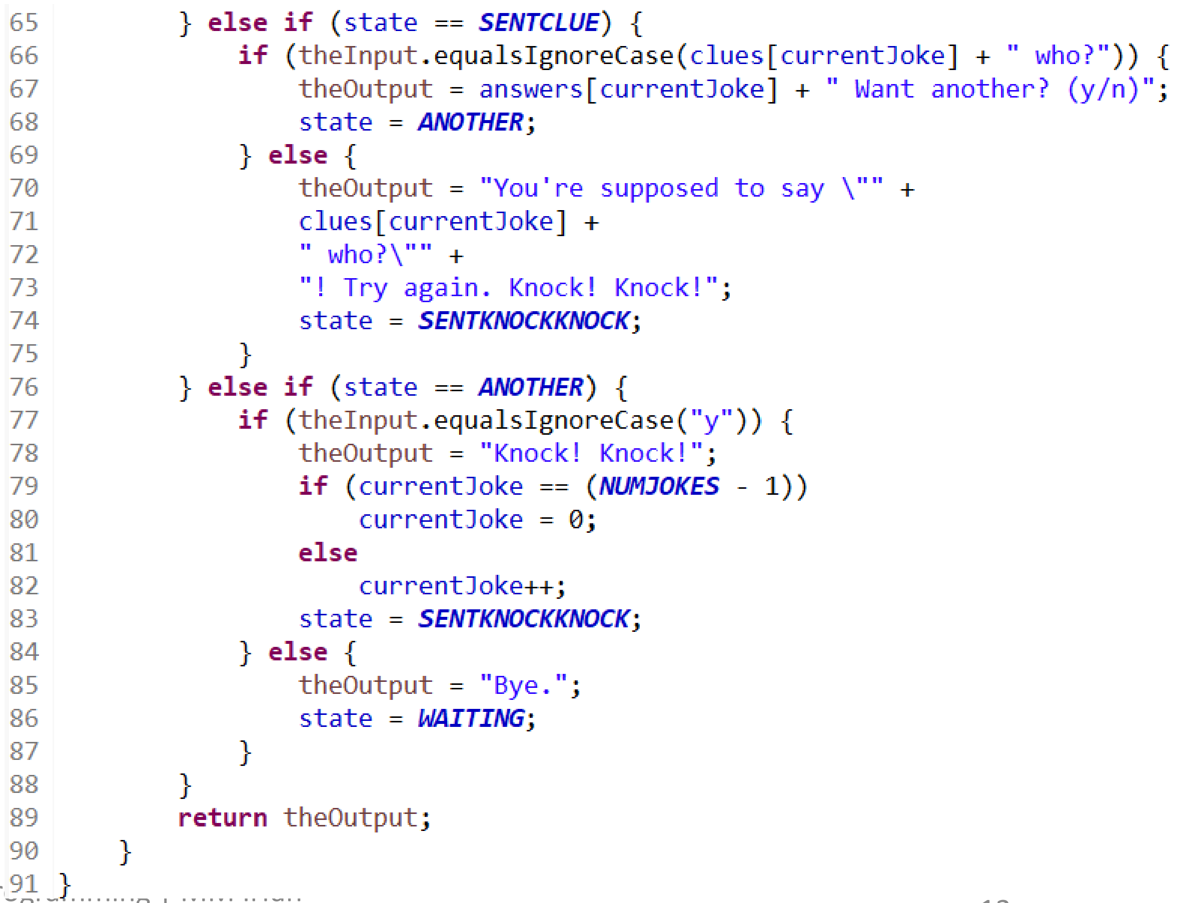


```java
            while ((inputLine = in.readLine()) != null) {
                outputLine = kkp.processInput(inputLine);
                out.println(outputLine);
                if (outputLine.equals("Bye."))
                    break;
            }
        } catch (IOException e) {
            System.out.println("Exception caught when trying to listen on port "
                + portNumber + " or listening for a connection");
            System.out.println(e.getMessage());
        }
    }
}
```

Knock Knock: Protocol (https://docs.oracle.com/javase/tutorial/networking/sockets)

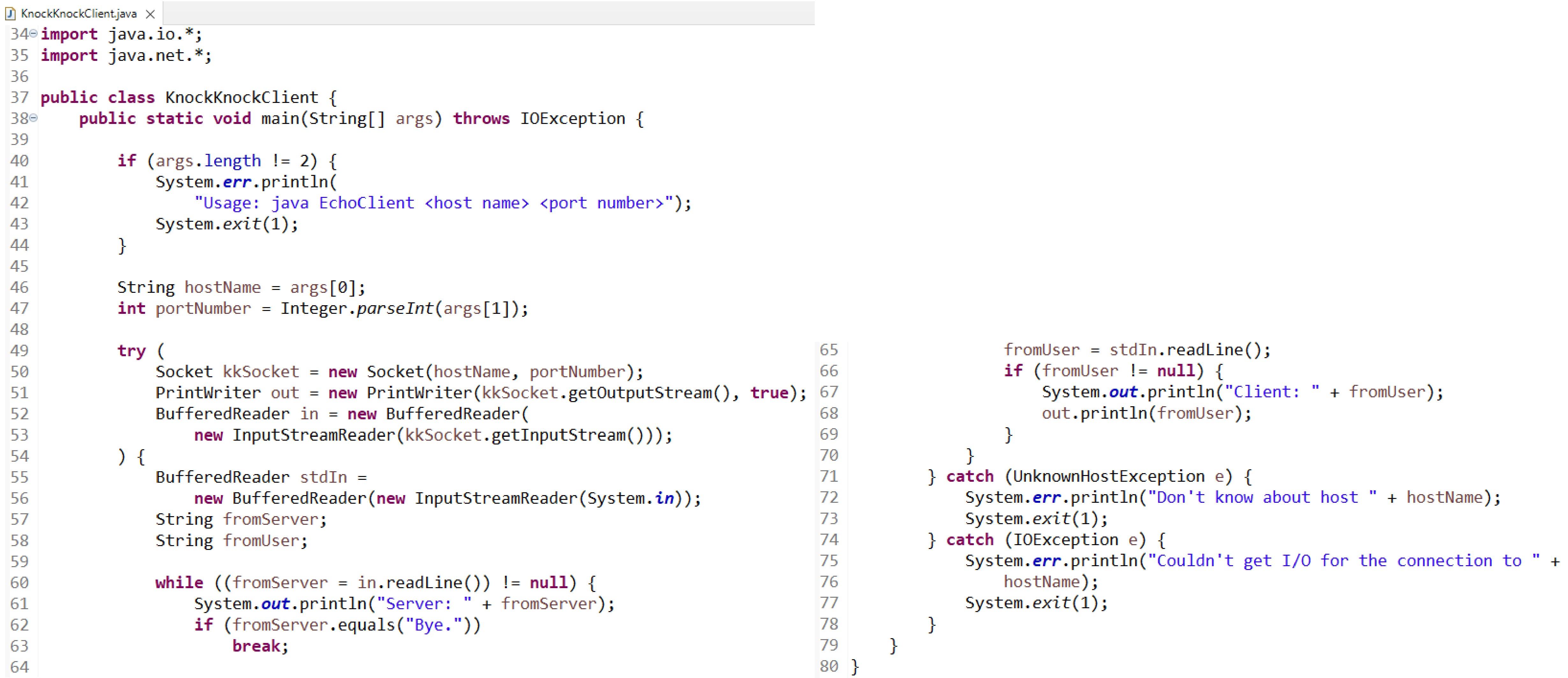

KnockKnockProtocol.java

```java
public class KnockKnockProtocol {
    private static final int WAITING = 0;
    private static final int SENTKNOCKKNOCK = 1;
    private static final int SENTCLUE = 2;
    private static final int ANOTHER = 3;

    private static final int NUMJOKES = 5;

    private int state = WAITING;
    private int currentJoke = 0;

    private String[] clues = { "Turnip", "Little Old Lady", "Atch", "Who", "Who" };
    private String[] answers = { "Turnip the heat, it's cold in here!",
                                 "I didn't know you could yodel!",
                                 "Bless you!",
                                 "Is there an owl in here?",
                                 "Is there an echo in here?" };

    public String processInput(String theInput) {
        String theOutput = null;

        if (state == WAITING) {
            theOutput = "Knock! Knock!";
            state = SENTKNOCKKNOCK;
        } else if (state == SENTKNOCKKNOCK) {
            if (theInput.equalsIgnoreCase("Who's there?")) {
                theOutput = clues[currentJoke];
                state = SENTCLUE;
            } else {
                theOutput = "You're supposed to say \"Who's there?\"! " +
                "Try again. Knock! Knock!";
            }
```



```java
        } else if (state == SENTCLUE) {
            if (theInput.equalsIgnoreCase(clues[currentJoke] + " who?")) {
                theOutput = answers[currentJoke] + " Want another? (y/n)";
                state = ANOTHER;
            } else {
                theOutput = "You're supposed to say \"" +
                clues[currentJoke] +
                " who?\"" +
                "! Try again. Knock! Knock!";
                state = SENTKNOCKKNOCK;
            }
        } else if (state == ANOTHER) {
            if (theInput.equalsIgnoreCase("y")) {
                theOutput = "Knock! Knock!";
                if (currentJoke == (NUMJOKES - 1))
                    currentJoke = 0;
                else
                    currentJoke++;
                state = SENTKNOCKKNOCK;
            } else {
                theOutput = "Bye.";
                state = WAITING;
            }
        }
        return theOutput;
    }
}
```

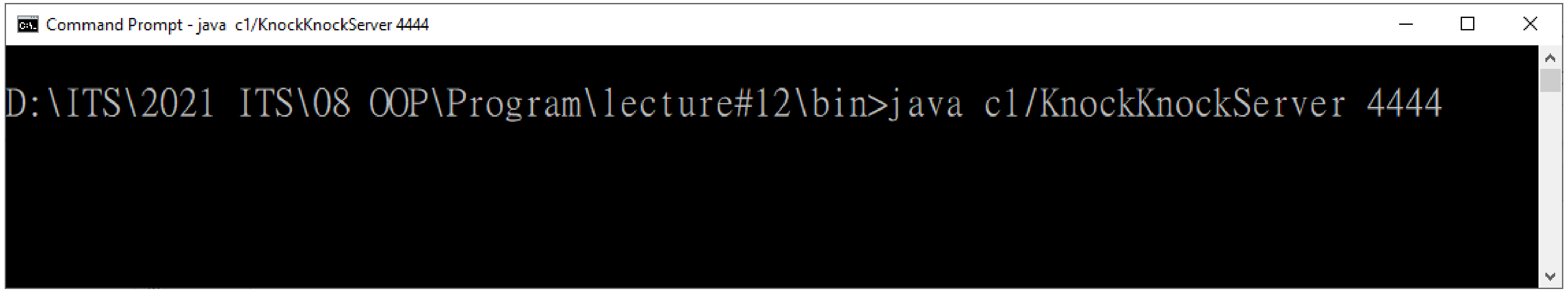
Knock Knock: Client (https://docs.oracle.com/javase/tutorial/networking/sockets)



KnockKnockClient.java

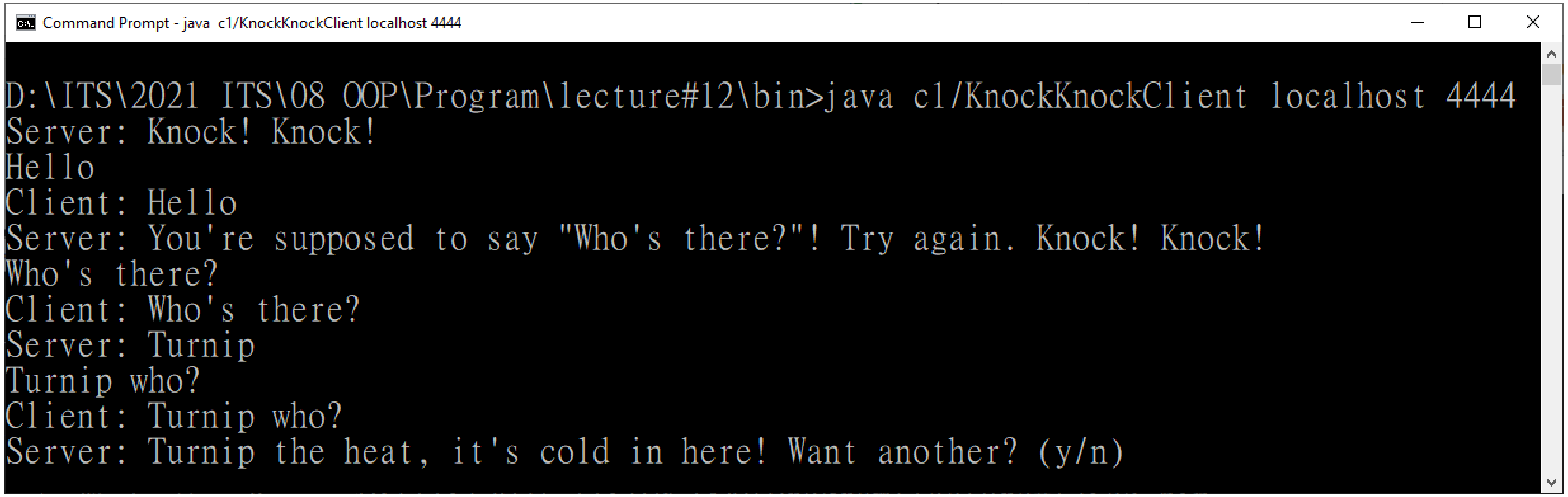
```java
import java.io.*;
import java.net.*;

public class KnockKnockClient {
    public static void main(String[] args) throws IOException {

        if (args.length != 2) {
            System.err.println(
                "Usage: java EchoClient <host name> <port number>");
            System.exit(1);
        }

        String hostName = args[0];
        int portNumber = Integer.parseInt(args[1]);

        try (
            Socket kkSocket = new Socket(hostName, portNumber);
            PrintWriter out = new PrintWriter(kkSocket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(kkSocket.getInputStream()));
        ) {
            BufferedReader stdIn =
                new BufferedReader(new InputStreamReader(System.in));
            String fromServer;
            String fromUser;

            while ((fromServer = in.readLine()) != null) {
                System.out.println("Server: " + fromServer);
                if (fromServer.equals("Bye."))
                    break;

                fromUser = stdIn.readLine();
                if (fromUser != null) {
                    System.out.println("Client: " + fromUser);
                    out.println(fromUser);
                }
            }
        } catch (UnknownHostException e) {
            System.err.println("Don't know about host " + hostName);
            System.exit(1);
        } catch (IOException e) {
            System.err.println("Couldn't get I/O for the connection to " +
                hostName);
            System.exit(1);
        }
    }
}
```

Knock Knock: Output



Command Prompt - java c1/KnockKnockServer 4444

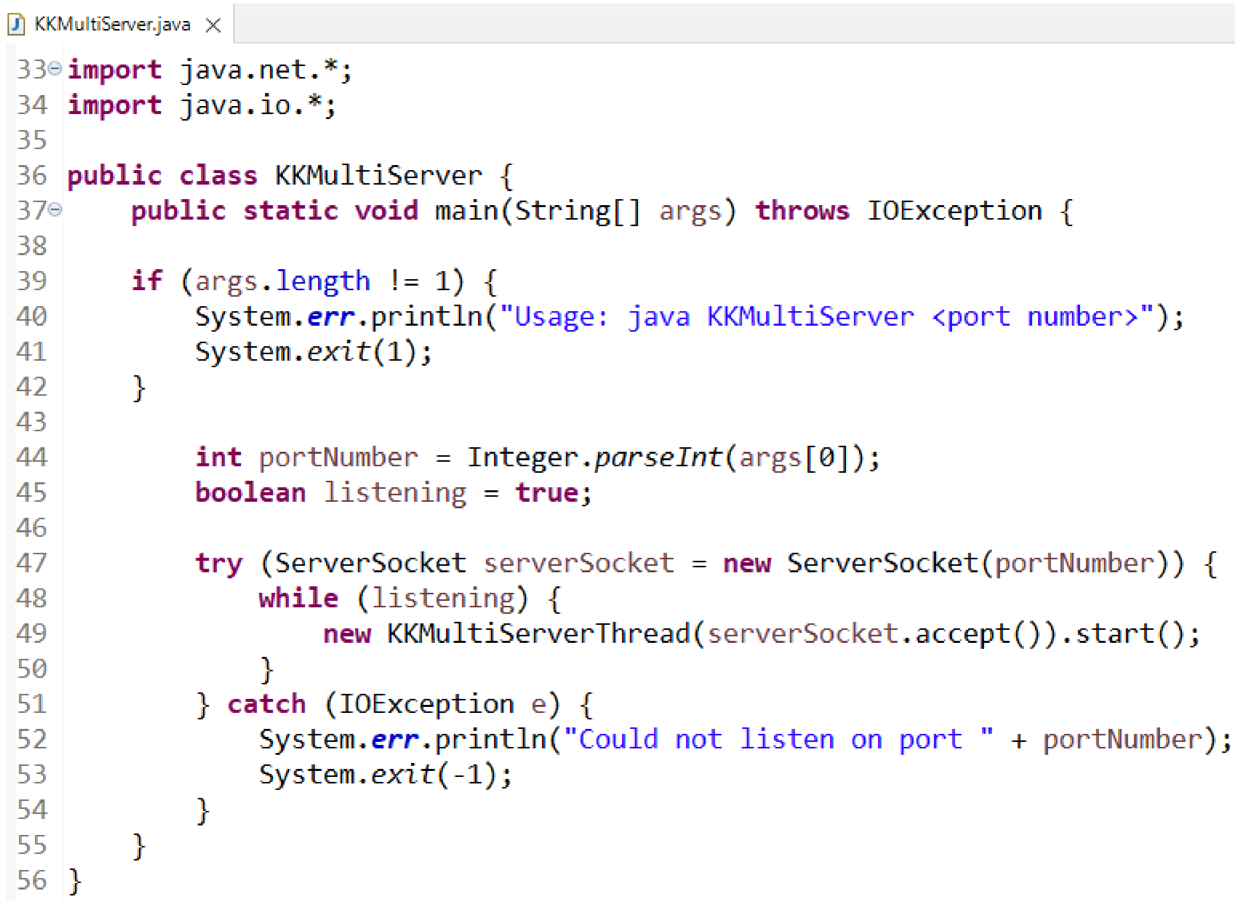
```
D:\ITS\2021 ITS\08 OOP\Program\lecture#12\bin>java c1/KnockKnockServer 4444
```



Command Prompt - java c1/KnockKnockClient localhost 4444

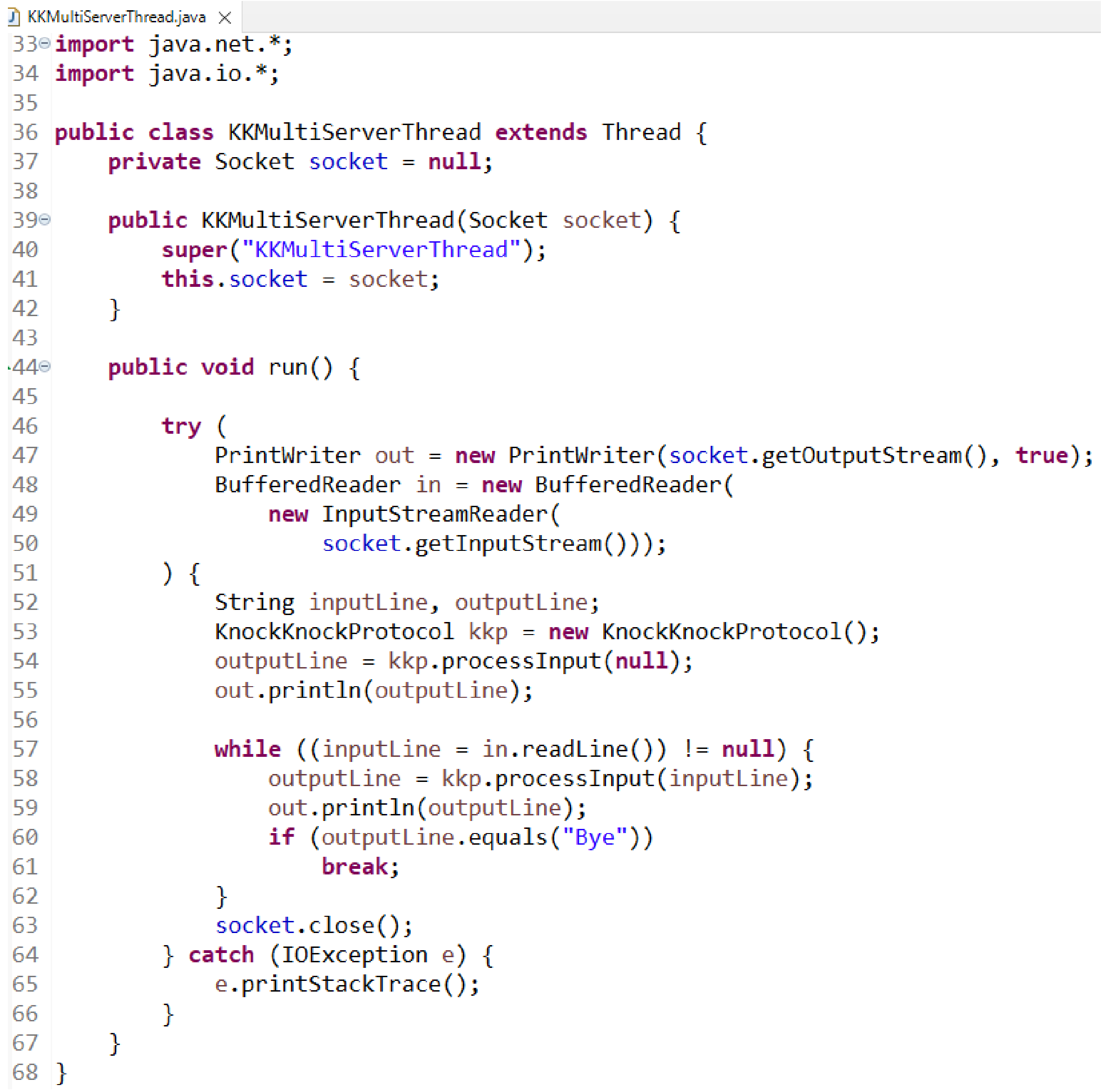
```
D:\ITS\2021 ITS\08 OOP\Program\lecture#12\bin>java c1/KnockKnockClient localhost 4444
Server: Knock! Knock!
Hello
Client: Hello
Server: You're supposed to say "Who's there?"! Try again. Knock! Knock!
Who's there?
Client: Who's there?
Server: Turnip
Turnip who?
Client: Turnip who?
Server: Turnip the heat, it's cold in here! Want another? (y/n)
```

Knock Knock: Multiple clients (https://docs.oracle.com/javase/tutorial/networking/sockets)



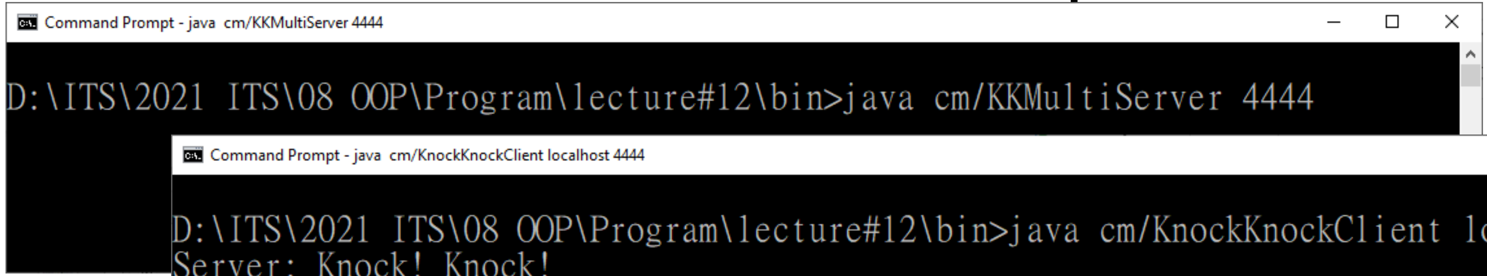
```java
import java.net.*;
import java.io.*;

public class KKMultiServer {
    public static void main(String[] args) throws IOException {

    if (args.length != 1) {
        System.err.println("Usage: java KKMultiServer <port number>");
        System.exit(1);
    }

        int portNumber = Integer.parseInt(args[0]);
        boolean listening = true;

        try (ServerSocket serverSocket = new ServerSocket(portNumber)) {
            while (listening) {
                new KKMultiServerThread(serverSocket.accept()).start();
            }
        } catch (IOException e) {
            System.err.println("Could not listen on port " + portNumber);
            System.exit(-1);
        }
    }
}
```

Knock Knock: Thread (https://docs.oracle.com/javase/tutorial/networking/sockets)



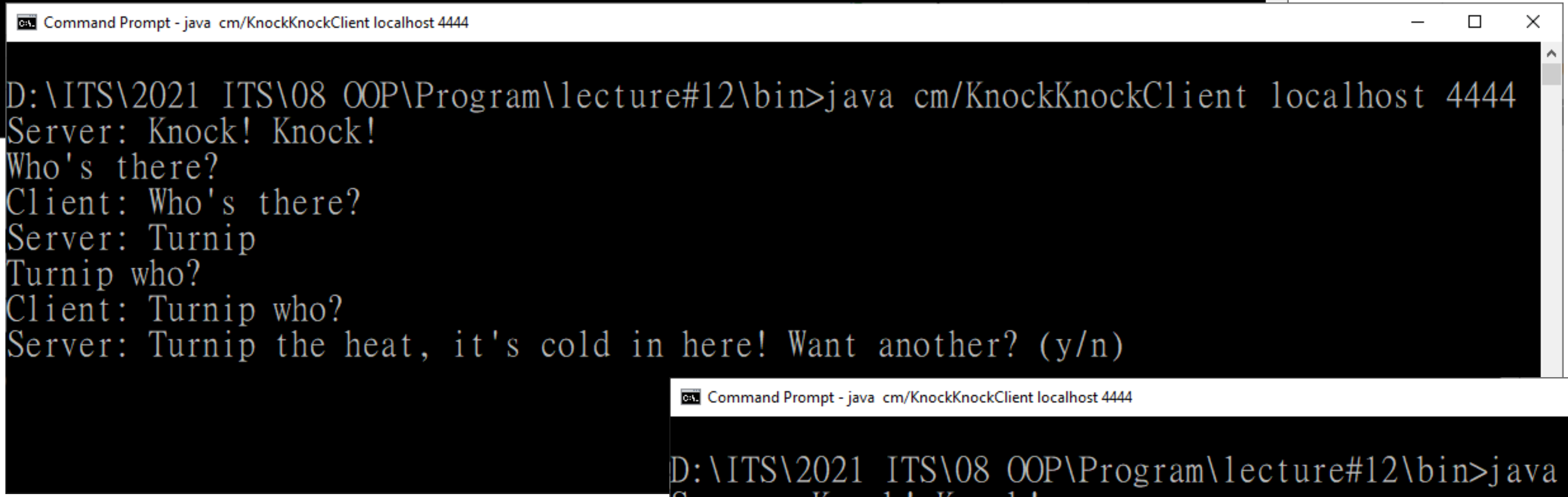
```java
import java.net.*;
import java.io.*;

public class KKMultiServerThread extends Thread {
    private Socket socket = null;

    public KKMultiServerThread(Socket socket) {
        super("KKMultiServerThread");
        this.socket = socket;
    }

    public void run() {

        try (
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(
                    socket.getInputStream()));
        ) {
            String inputLine, outputLine;
            KnockKnockProtocol kkp = new KnockKnockProtocol();
            outputLine = kkp.processInput(null);
            out.println(outputLine);

            while ((inputLine = in.readLine()) != null) {
                outputLine = kkp.processInput(inputLine);
                out.println(outputLine);
                if (outputLine.equals("Bye"))
                    break;
            }
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Knock Knock: Multiple clients (output)



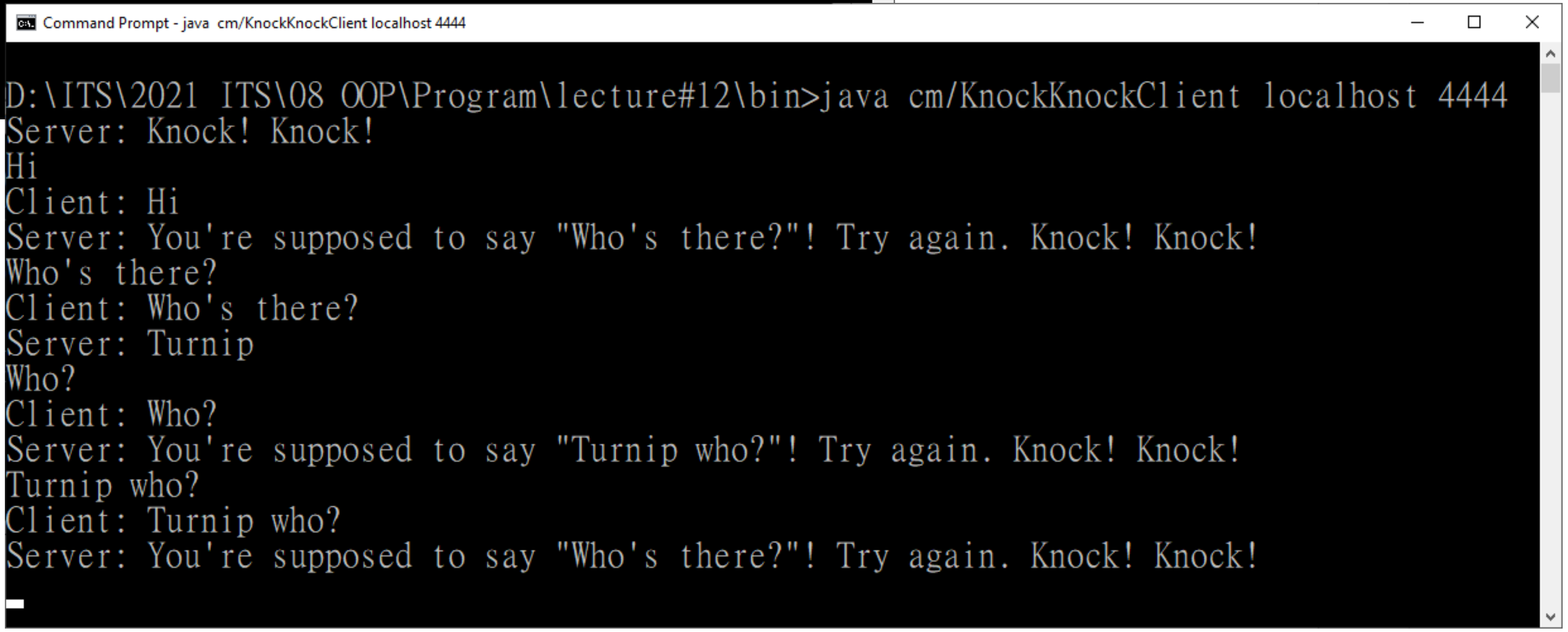
Command Prompt - java cm/KKMultiServer 4444

```
D:\ITS\2021 ITS\08 OOP\Program\lecture#12\bin>java cm/KKMultiServer 4444
```



Command Prompt - java cm/KnockKnockClient localhost 4444

```
D:\ITS\2021 ITS\08 OOP\Program\lecture#12\bin>java cm/KnockKnockClient localhost 4444
Server: Knock! Knock!
Who's there?
Client: Who's there?
Server: Turnip
Turnip who?
Client: Turnip who?
Server: Turnip the heat, it's cold in here! Want another? (y/n)
```

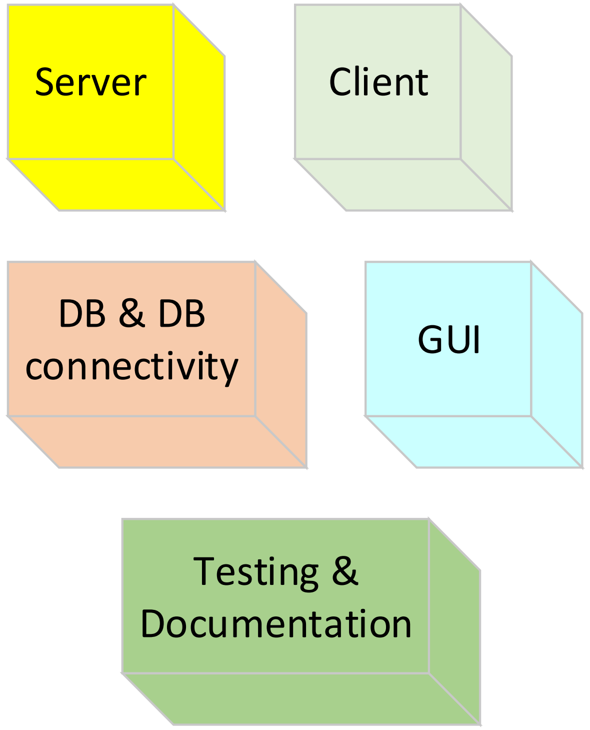


Command Prompt - java cm/KnockKnockClient localhost 4444

```
D:\ITS\2021 ITS\08 OOP\Program\lecture#12\bin>java cm/KnockKnockClient localhost 4444
Server: Knock! Knock!
Hi
Client: Hi
Server: You're supposed to say "Who's there?"! Try again. Knock! Knock!
Who's there?
Client: Who's there?
Server: Turnip
Who?
Client: Who?
Server: You're supposed to say "Turnip who?"! Try again. Knock! Knock!
Turnip who?
Client: Turnip who?
Server: You're supposed to say "Who's there?"! Try again. Knock! Knock!
```

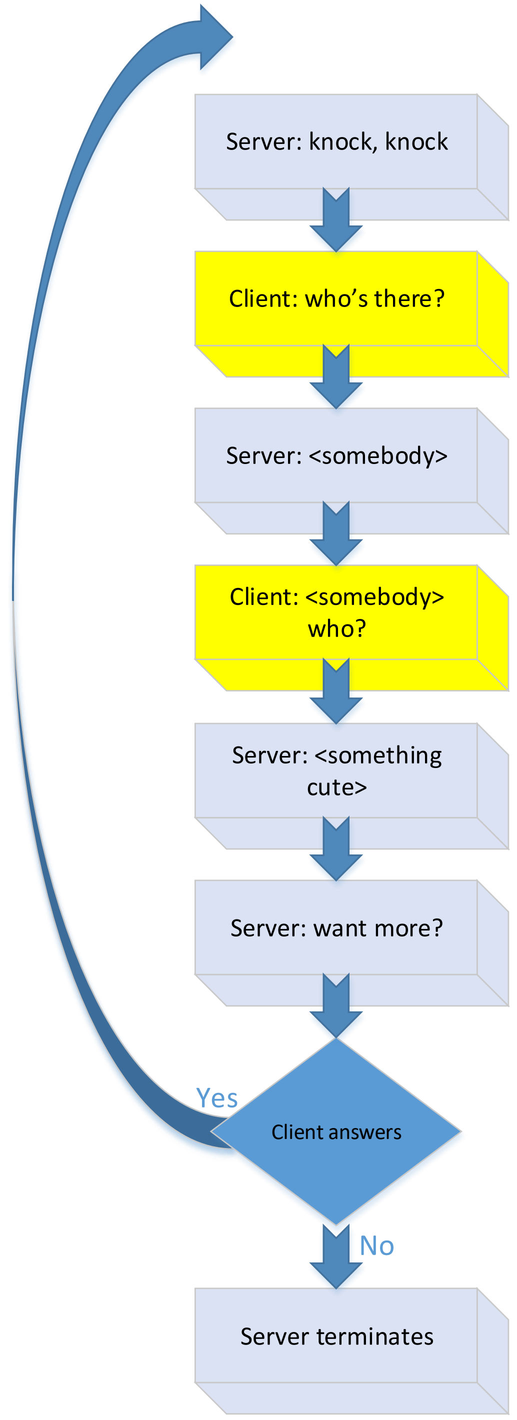
Team Project

- Topic of project
- Distribution of work → specification comes in



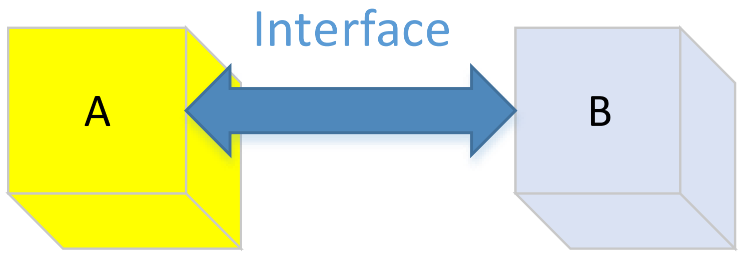
Team Project (continued)

- Specification & document interface: generic term, it’s not necessarily using Java interface
- Protocol, flow chart
- E.g., Knock-knock server-client communication protocol



Team Project (continued)

- Think about test cases



- A design test case for B, since A uses B’s functions
- B design test case for A, since B uses A’s functions

Team Project (continued)

- Clearly describe:
  - Design: *Specification* → *Design* → *Implementation*
  - Product documentation
    - User level
    - Performance criteria
    - How to install the system (installation)
    - Trouble shooting
  - Develop & use test plans
    - E.g., stubs
    - Multithreaded
    - Distributed testing

Team Project (continued)

- Need to take care carefully:
  - Server down → the client should know this
  - Client down → the server should know this
  - Exception handling
    - Connection *error* when a server *down*
    - Connection *error* when a client *down*