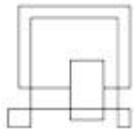
VIA University College



Software Development with UML and Java 2

Autumn 2021

Joseph Okika (jook@via.dk)

Learning Objectives

- By the end of class today, you should be able to:
  - ✓ explain the terms – client, server, host and port
  - ✓ describe the client-server model
  - ✓ write programs that use TCP Sockets in both client and server programs.
  - ✓ write programs that use UDP Sockets in both client and server programs
  - ✓ explain the convenience of Java's stream classes

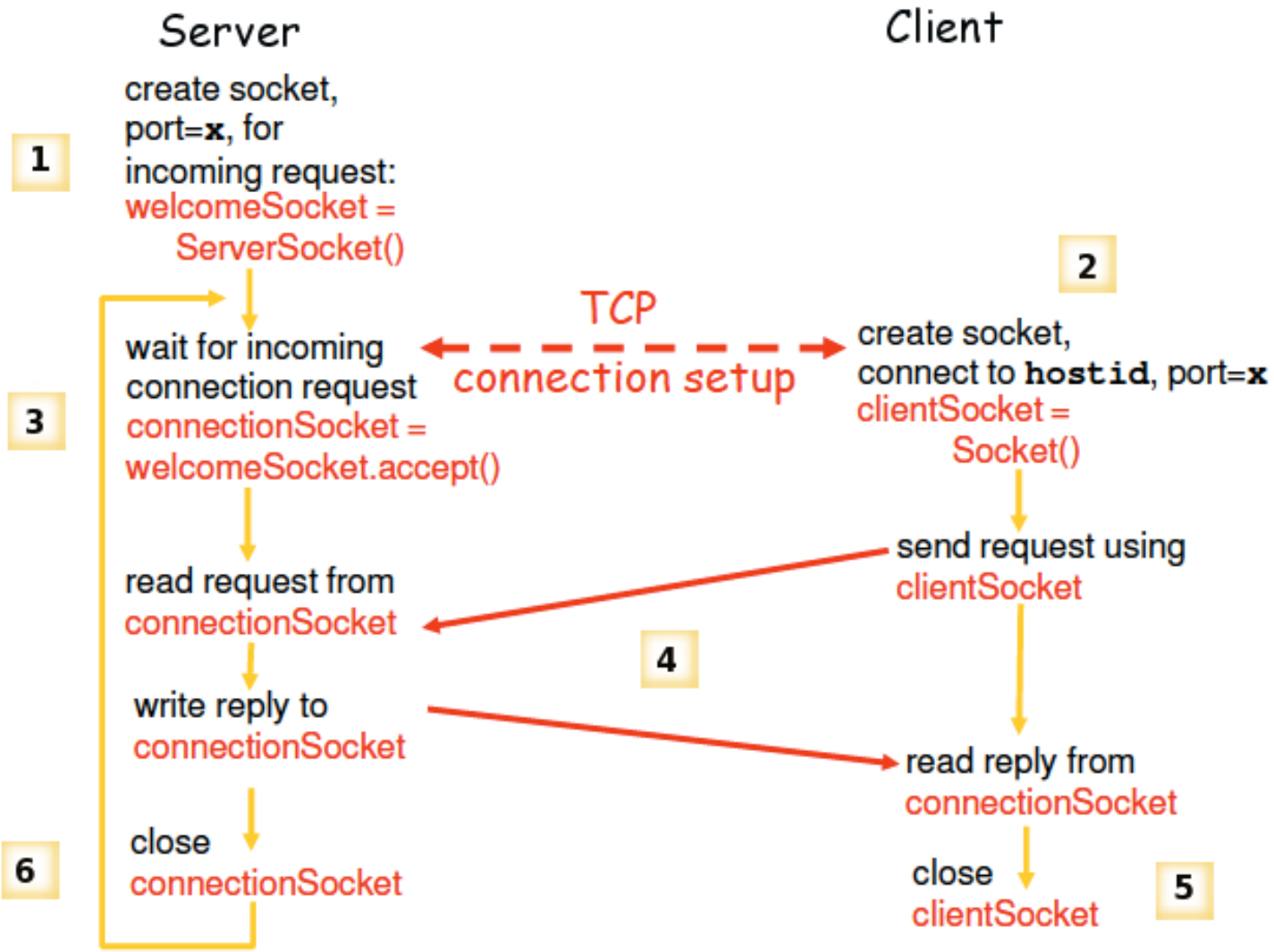
Phone calls versus Mails

- Phone call:
  - A dedicated point-to-point channel between two callers.
  - Lossless and reliable.
  - Words are received in the same order they are spoken
- Mail:
  - No point-to-point channel between sender and receiver.
  - A letter may be lost before received.
  - More letters may not be received in the same order as sent.

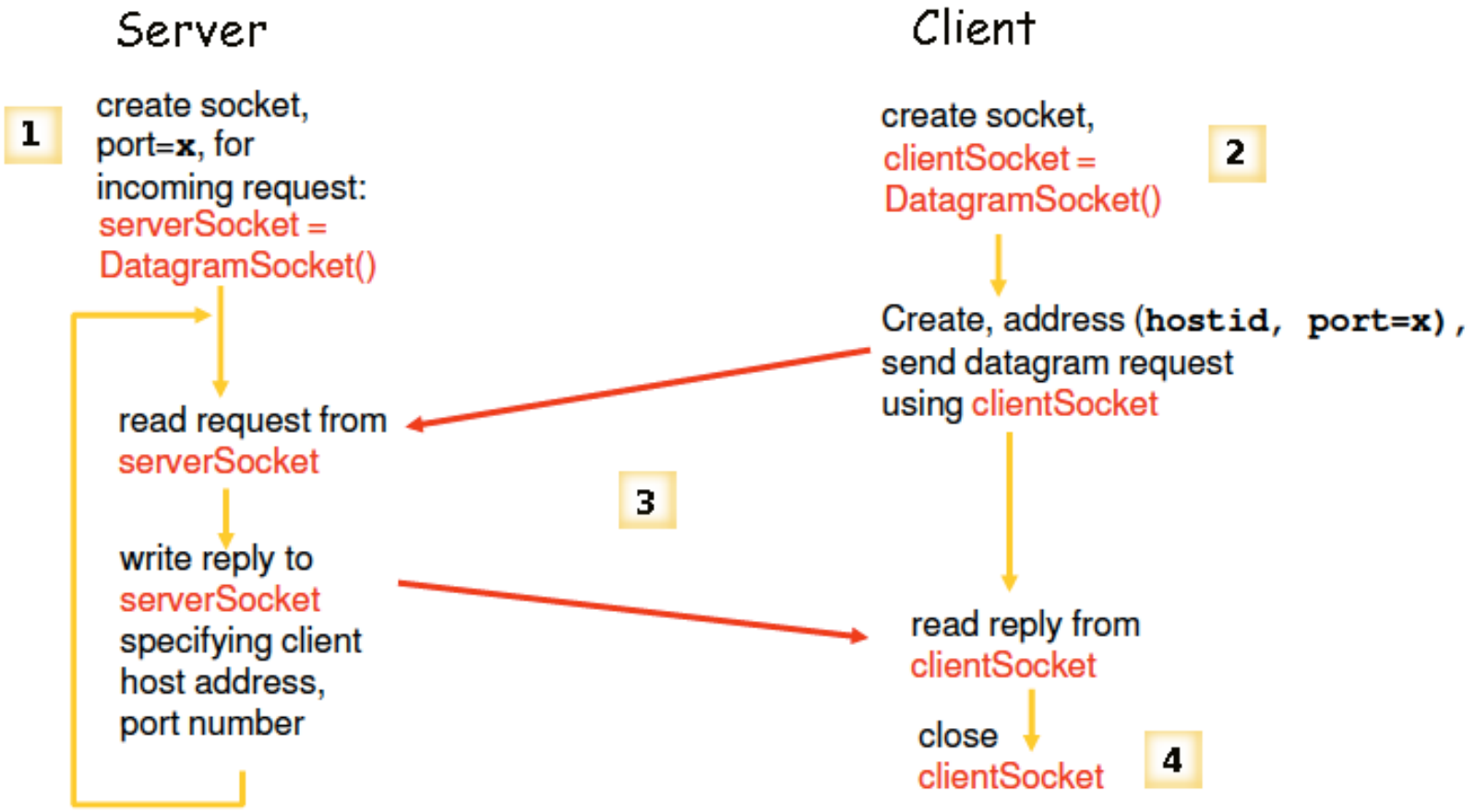
What is a socket?

- provides an interface to send data to/from the network through a port
- a stream that connects two processes running in different address spaces (usually across a network)
- creating a socket between two machines can be seen as creating an input and output streams for sending data between programs running on each machine
- lowest-level form of communication from developer's view.

TCP Sockets


Server
Client
1
create socket,
port=x, for
incoming request:
welcomeSocket =
ServerSocket()
2
TCP
connection setup
wait for incoming
connection request
connectionSocket =
welcomeSocket.accept()
create socket,
connect to hostid, port=x
clientSocket =
Socket()
3
send request using
clientSocket
read request from
connectionSocket
4
write reply to
connectionSocket
read reply from
connectionSocket
close
connectionSocket
close
clientSocket
5
6

UDP Sockets


Server
Client
1
create socket, port=x, for incoming request: serverSocket = DatagramSocket()
create socket, clientSocket = DatagramSocket()
2
Create, address (hostid, port=x), send datagram request using clientSocket
read request from serverSocket
3
write reply to serverSocket specifying client host address, port number
read reply from clientSocket
close clientSocket
4

TCP Socket vs UDP Socket

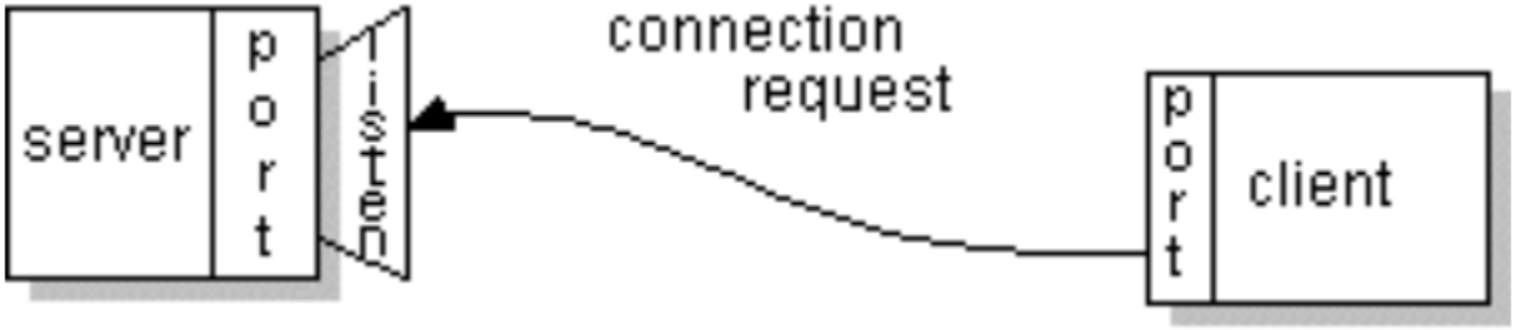
- Stream Socket:
  - A dedicated point-to-point channel between a client and server.
  - Use TCP (Transmission Control Protocol) for data transmission.
  - Lossless and reliable.
  - Sent and received in the same order.
- Datagram Socket:
  - No dedicated point-to-point channel between a client and server.
  - Use UDP (User Datagram Protocol) for data transmission.
  - May lose data and not 100% reliable.
  - Data may not be received in the same order as sent.

Example Applications

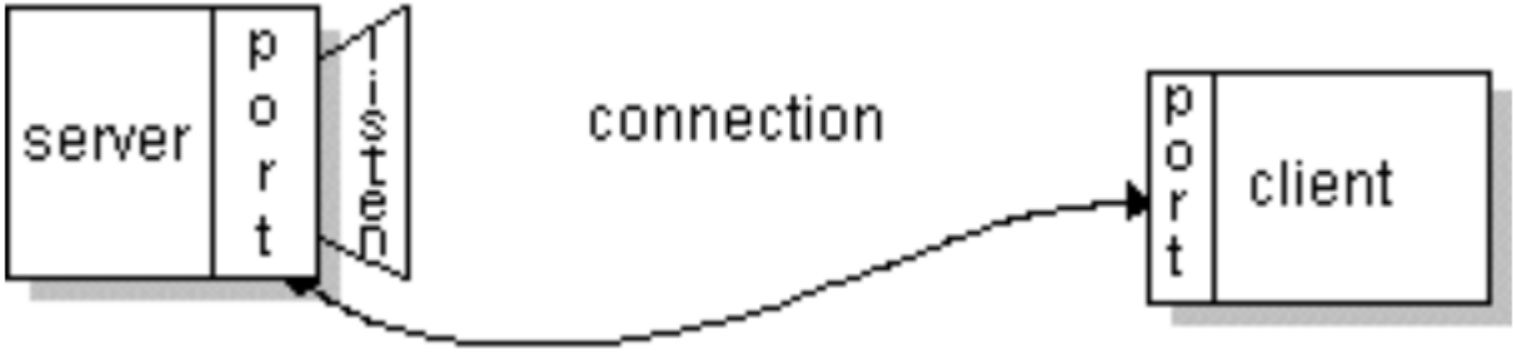
- TCP:
  - Do not accept re-ordering or loss
  - Examples
    - Transferring files
    - Downloading web pages
- UDP:
  - Require high bandwidth, can accept loss or reordering.
  - Example
    - Transmission of video/sound in real time

TCP Sockets

Step 1



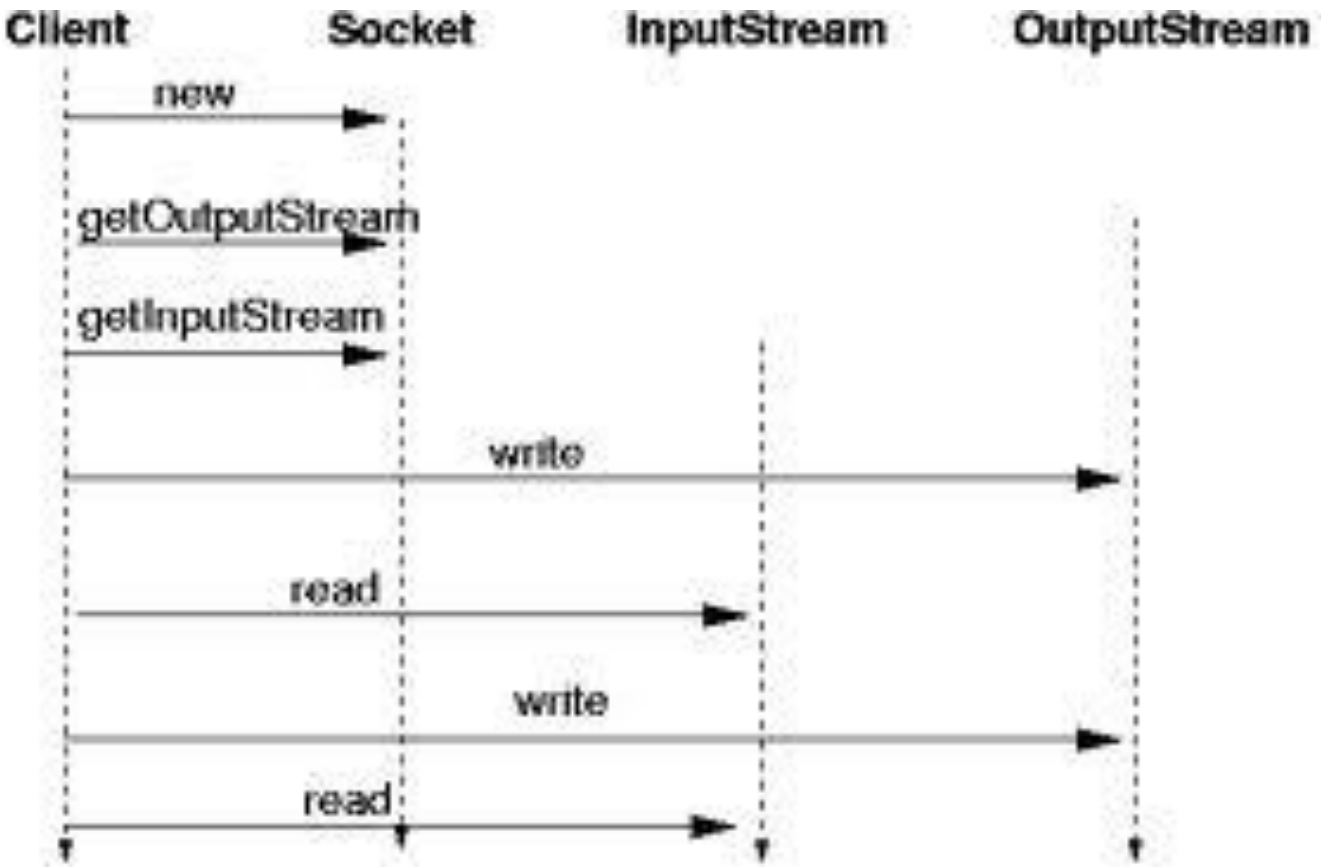
Step 2



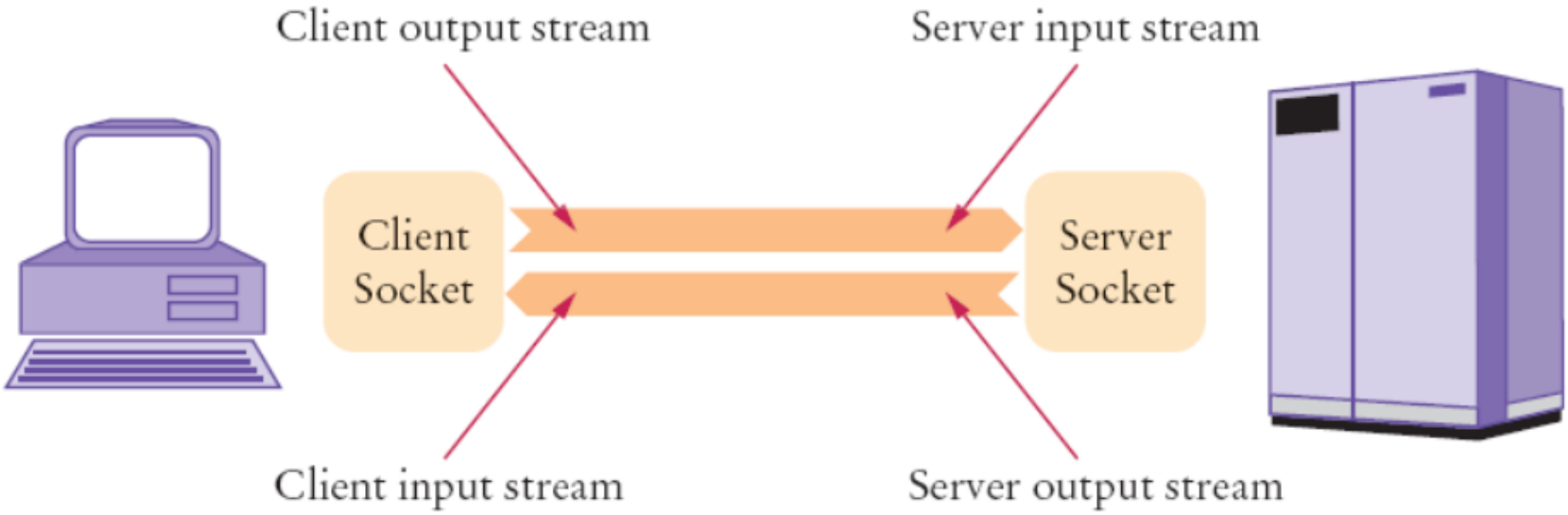
Basic Client-Server Program

- Client: write a program that rings up another program at a specified IP address and running on a specified port.
- Server: write a second program that accepts connection and establishes input/output stream to client.
- When server accepts connection, client can establish input/output stream to server.
- Client can now make request by sending data. Server sends replies to client.

Basic Socket Client



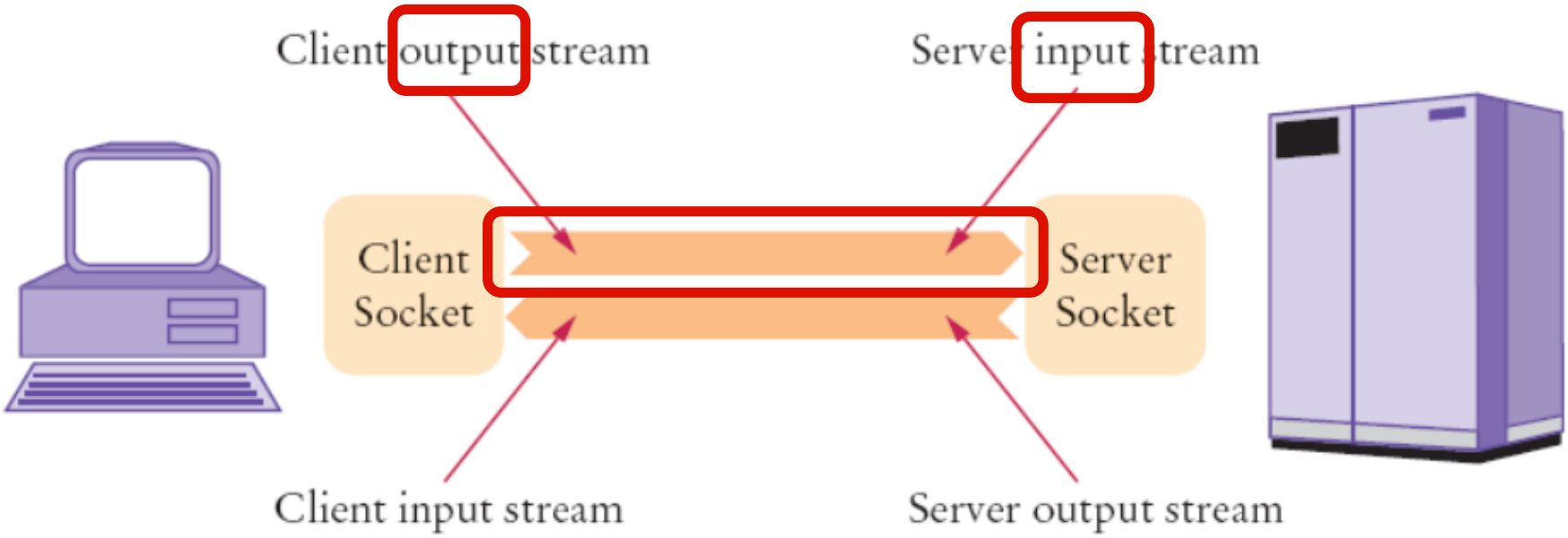
Streams



Reference: https://www.iro.umontreal.ca/~pift1025/bigjava/Ch24/ch24.html

Streams

What you send has to match what you receive (bytes, lines)



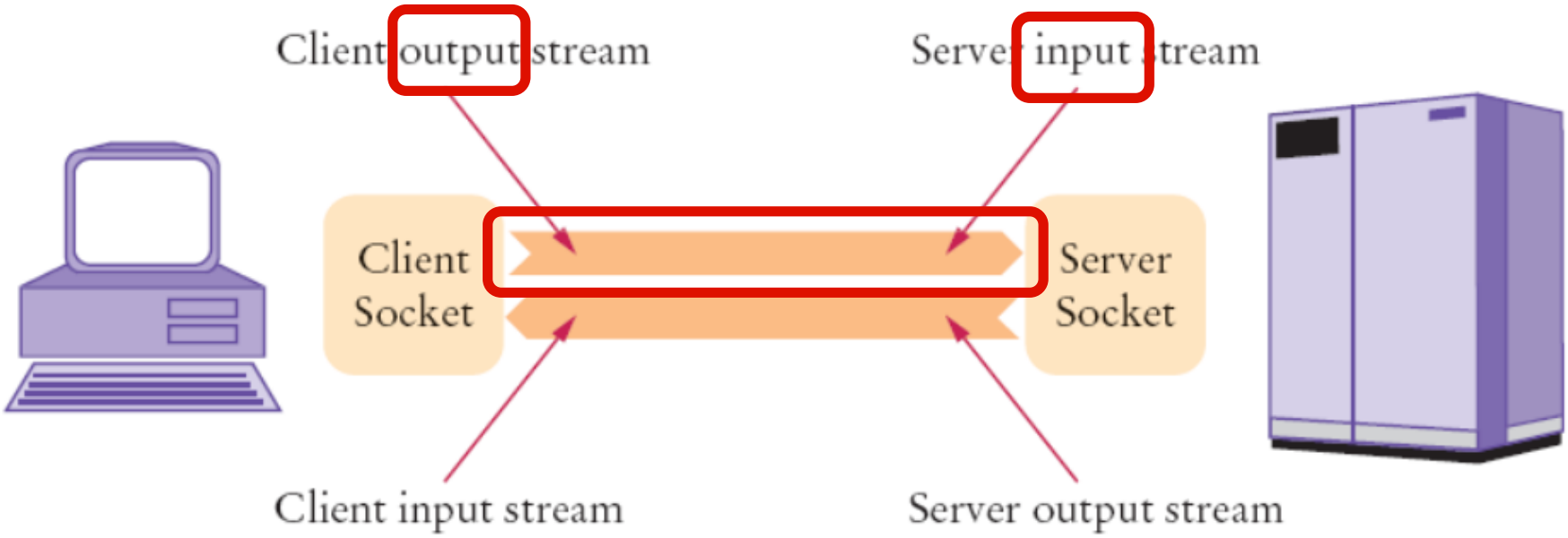
Reference: https://www.iro.umontreal.ca/~pift1025/bigjava/Ch24/ch24.html

Streams

DataOutputStream ←→ DataInputStream

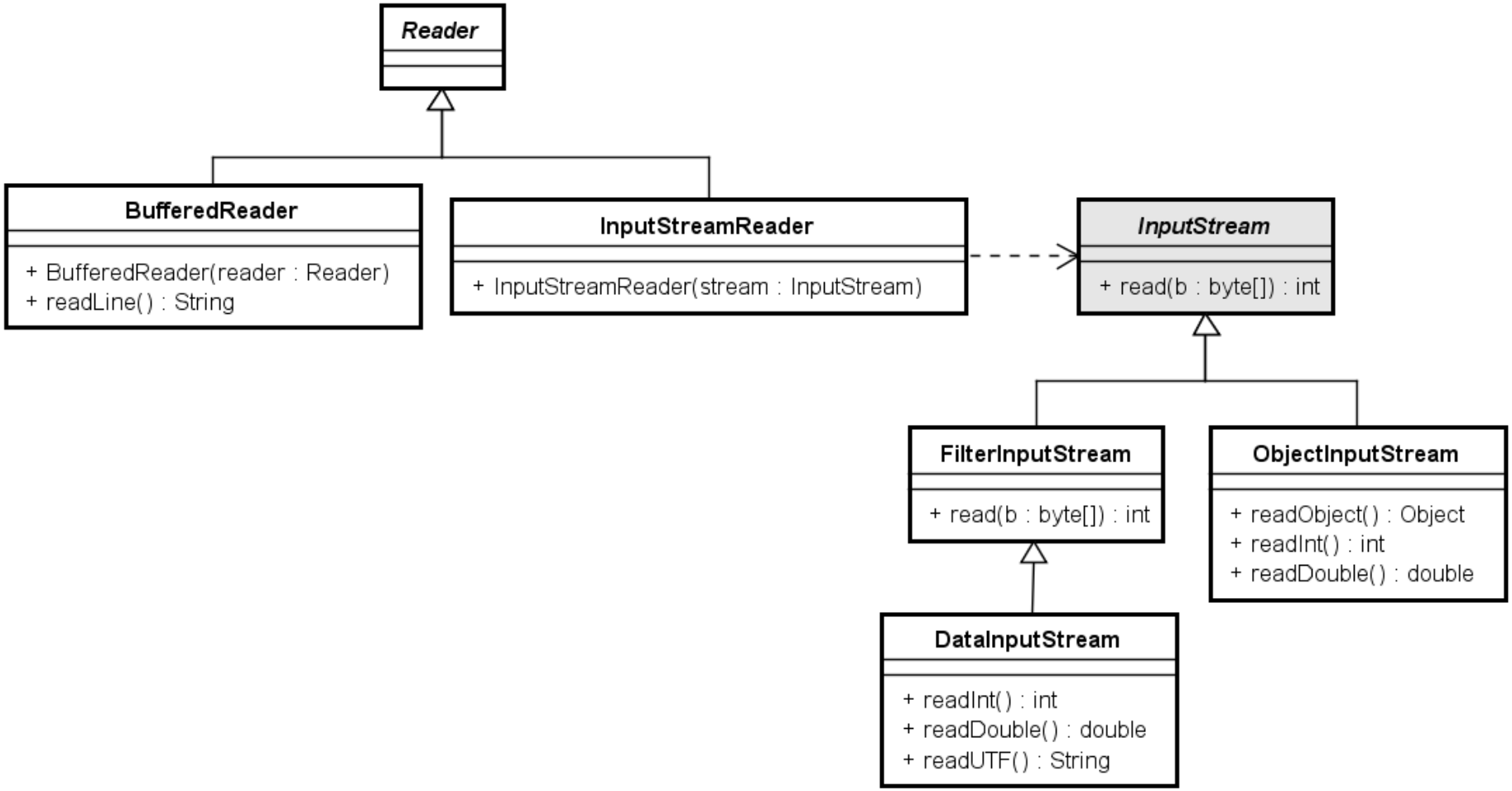
ObjectOutputStream ←→ ObjectInputStream

PrintWriter ←→ BufferedReader

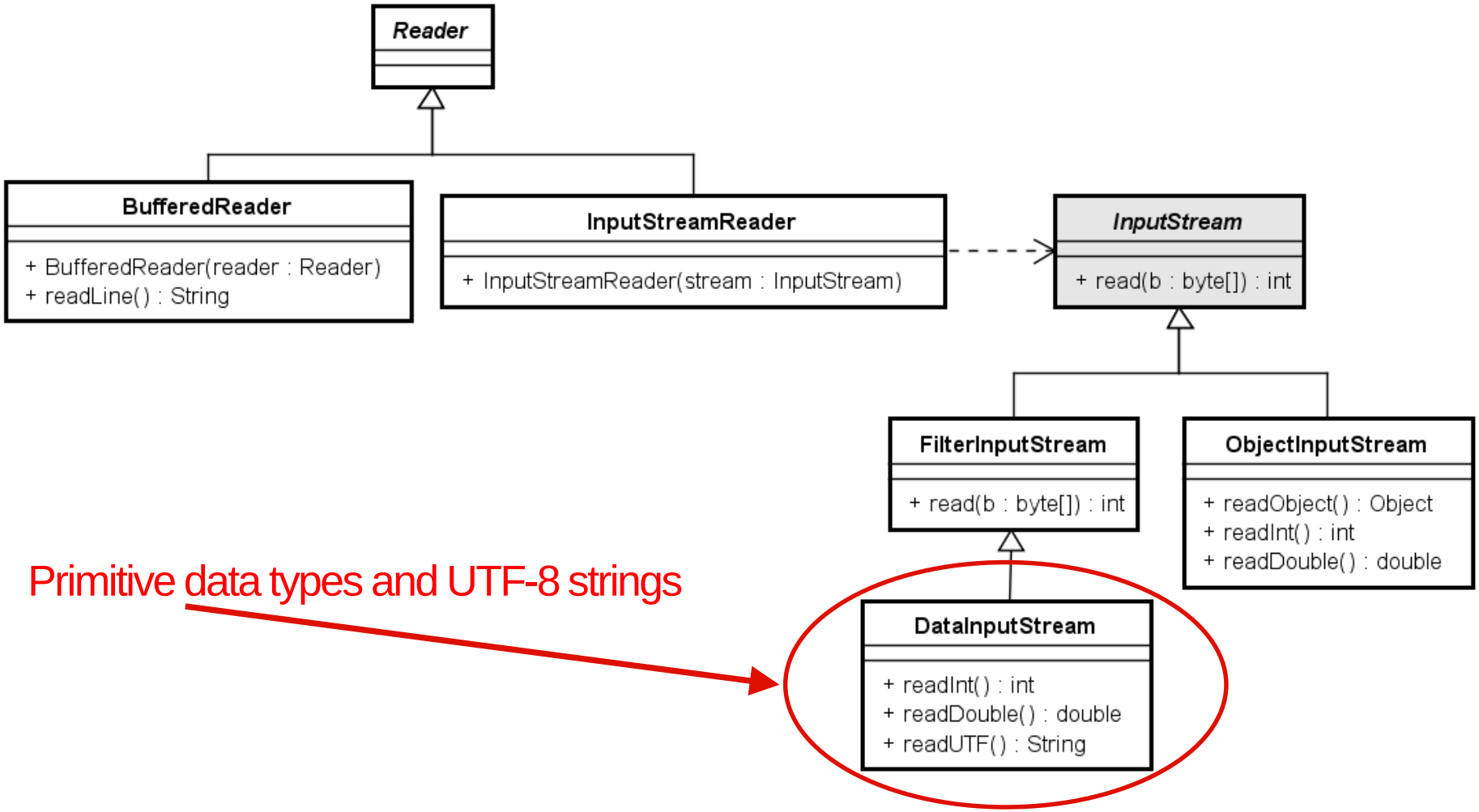


Reference: https://www.iro.umontreal.ca/~pift1025/bigjava/Ch24/ch24.html

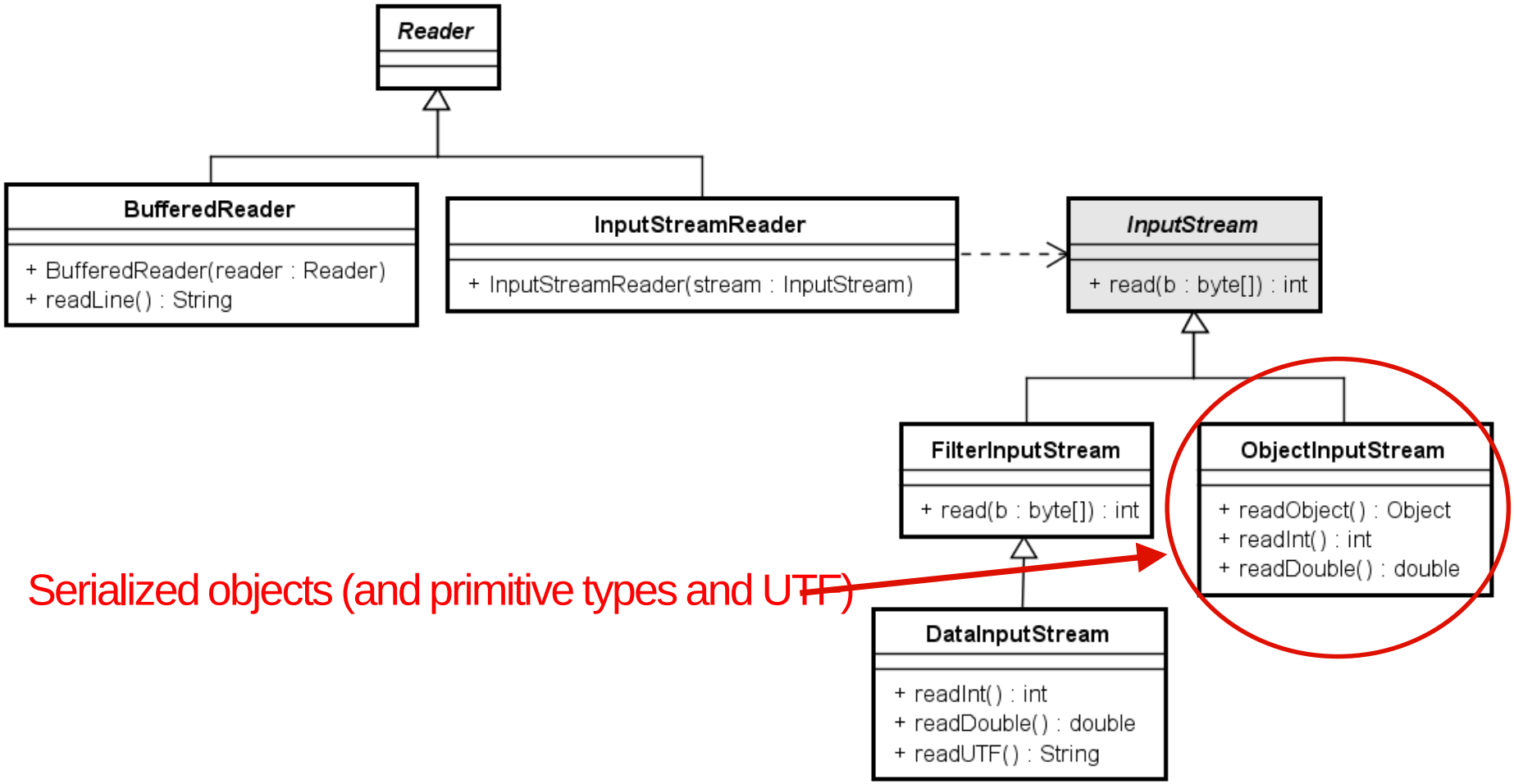
Streams - input



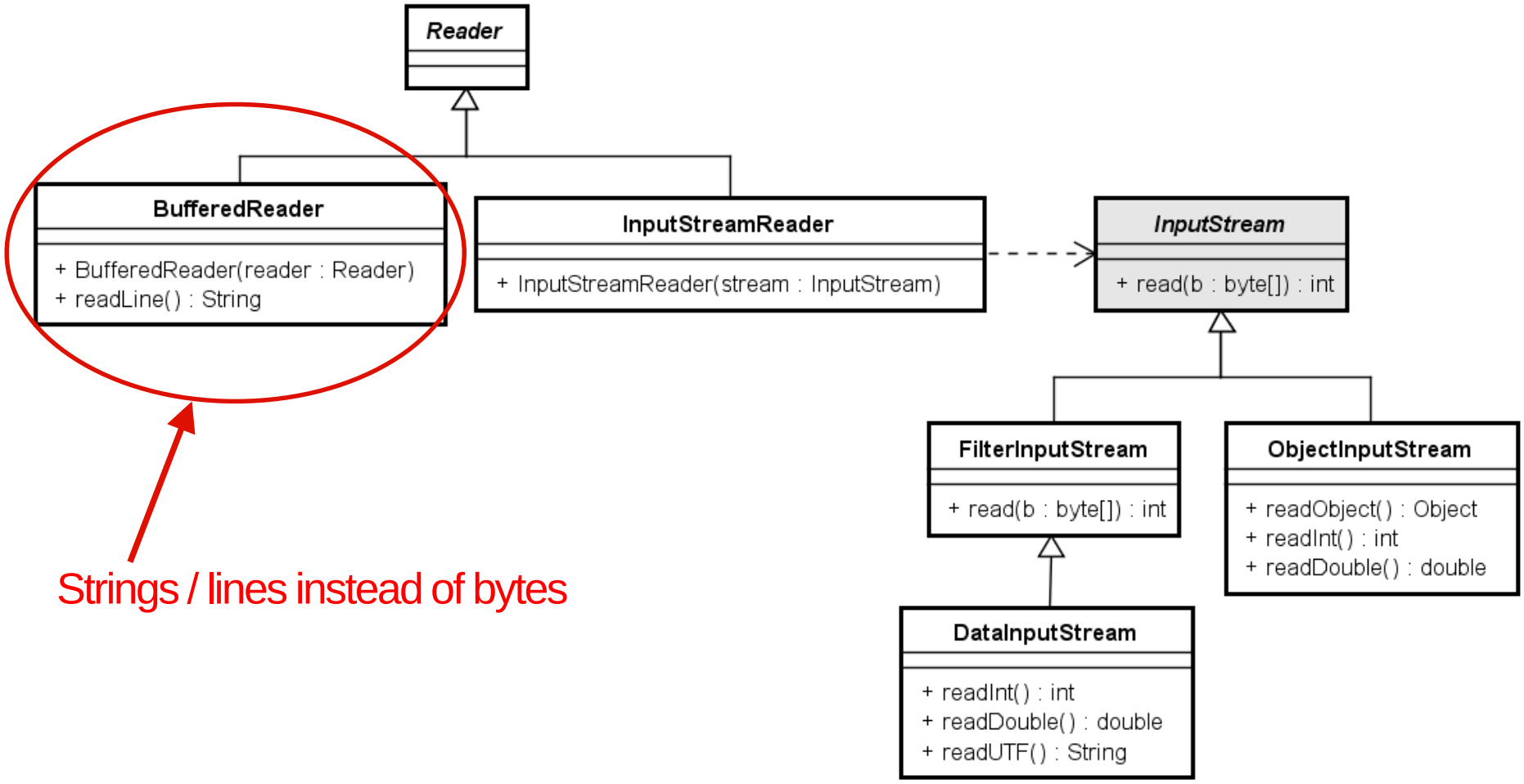
Streams - input



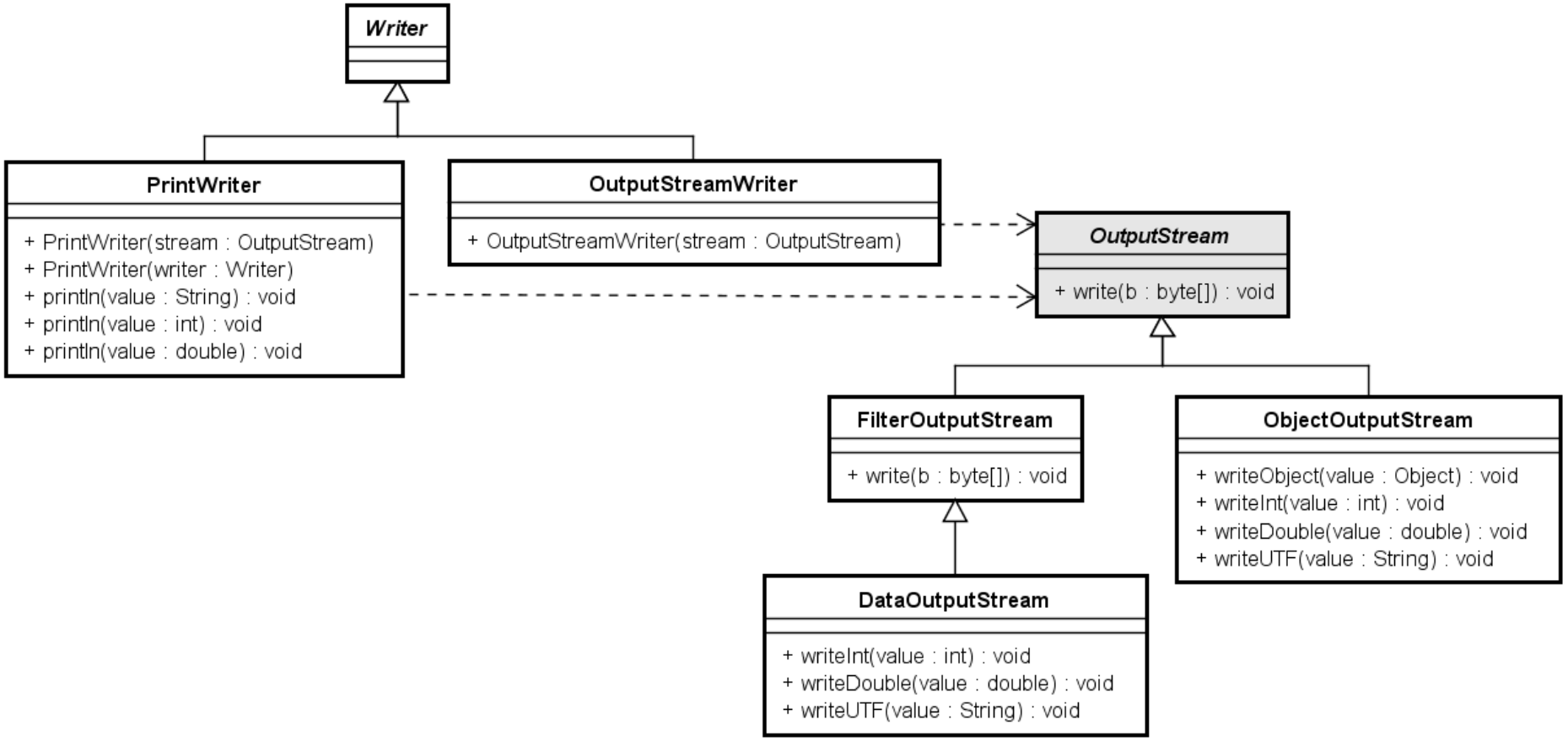
Streams - input



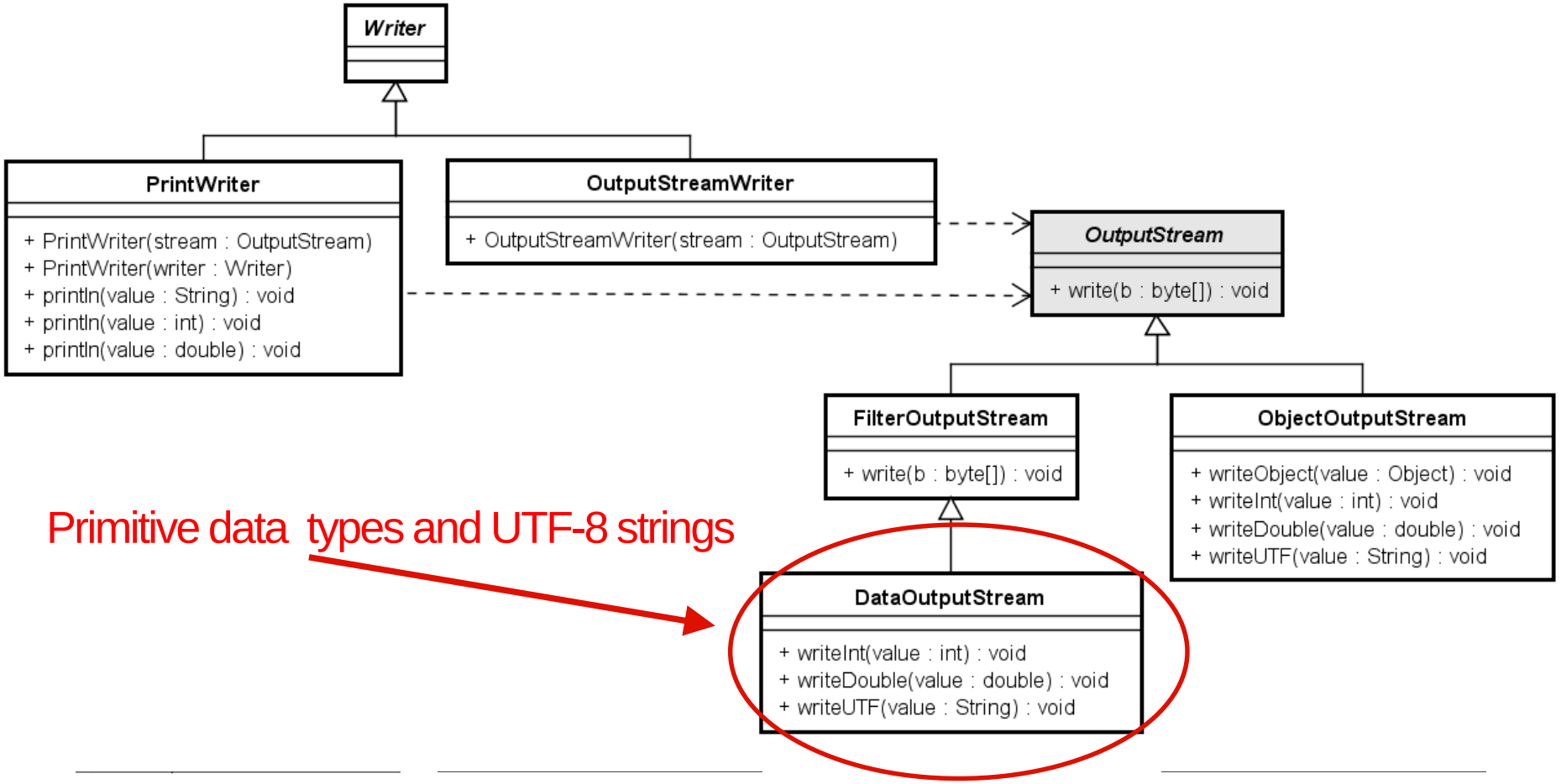
Streams - input



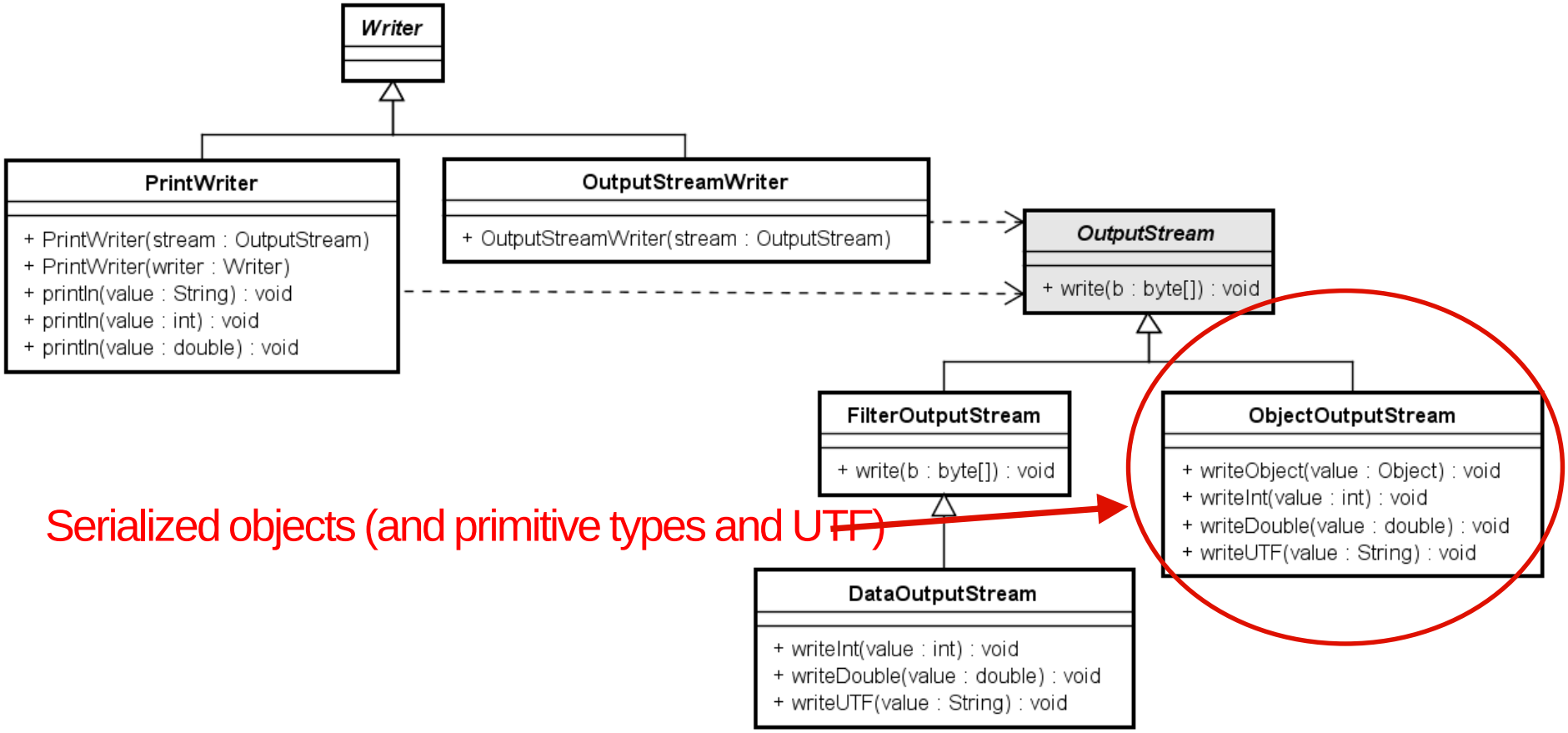
Streams - output



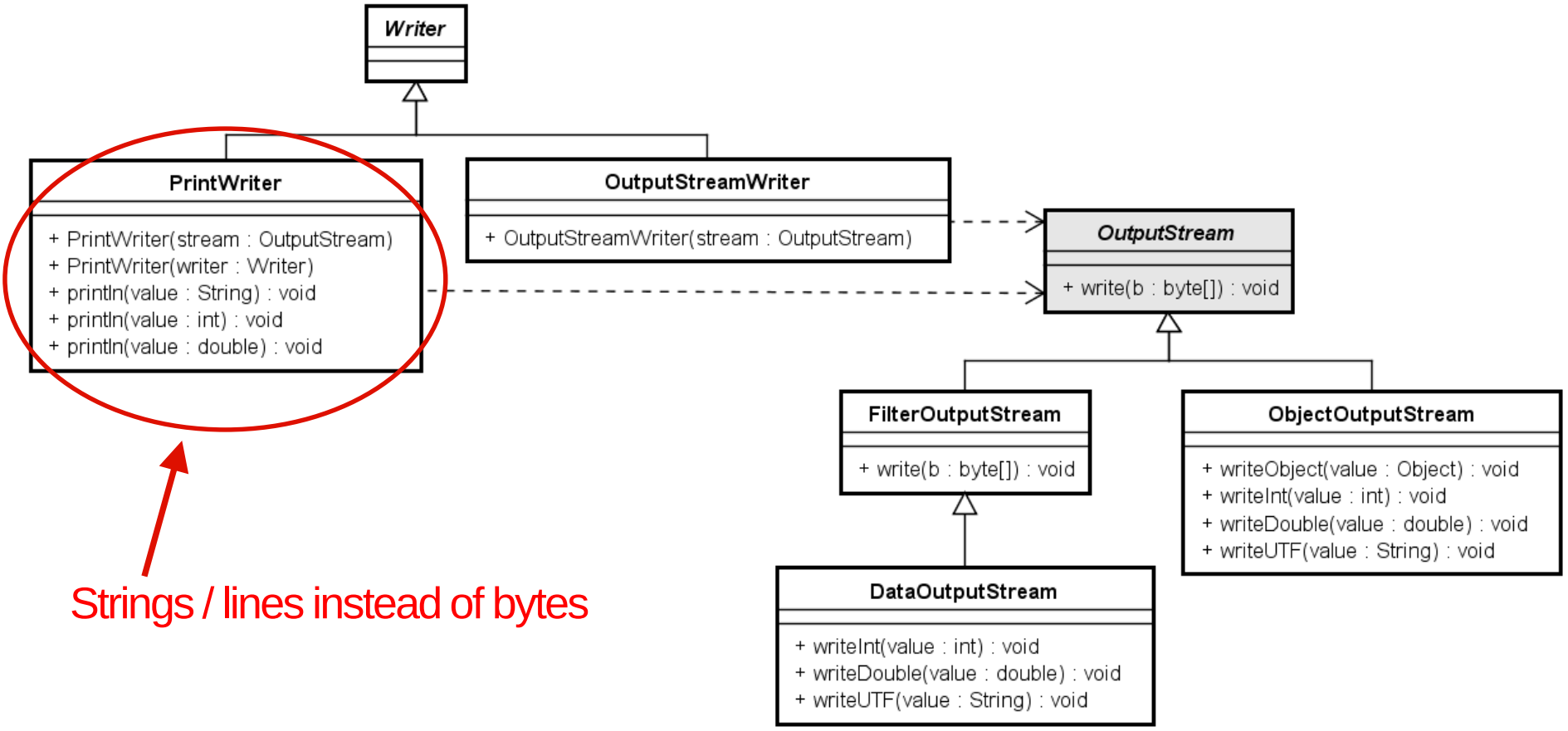
Streams - output



Streams - output



Streams - output



A Simple Server Example

- To demonstrate the principles of a client-server protocol
- The server offers a very simple service to a client
  - Wait to receive a string, and then send back the string converted to upper case.
- Steps for creating a server program
  1. Open the server socket (ServerSocket(PORT))
  2. Wait for the Client Request (socket.accept())
  3. Create I/O streams for communicating to the client (InputStream, InputStreamReader, BufferedReader, OutputStream, PrintWriter)
  4. Perform communication with the client (in.readLine(), out.println())

A Simple Client Example

- Steps for creating a client program
  1. Create a Socekt object (Socket(HOST, PORT))
  2. Create I/O streams for communicating to the client (InputStream, InputStreamReader, BufferedReader, OutputStream, PrintWriter)
  3. Perform communication with the server (in.readLine(), out.println())
  4. Close the socket when done (socket.close())

TCP Server

TCPServer – socket, stream

- Port (6789)
- a “general” socket and a server socket
- call the accept in a while loop until the connection is lost (to get the server to listen for a client requesting a connection)

```
ServerSocket listenSocket = new ServerSocket(PORT);
Socket socket = listenSocket.accept();
```

- input and output stream

```
// InputStream, InputStreamReader, BufferedReader
BufferedReader in = new BufferedReader(new InputStreamReader(
        socket.getInputStream()));

// OutputStream, PrintWriter
PrintWriter outWriter = new PrintWriter(socket.getOutputStream(), true);
```

TCPServer (1/2)

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

public class TCPServer {
    public static void main(String args[]) throws IOException {
        final int PORT = 6789;
        System.out.println("Starting Server...");

        //create a server socket at port 6789 listening for clients
        ServerSocket listenSocket = new ServerSocket(PORT);

        while (true) {
            System.out.println("Waiting for a client...");

            // Wait, on listening socket for contact by client
            Socket socket = listenSocket.accept();

            // create input stream attached to the socket
            // InputStream, InputStreamReader, BufferedReader
            BufferedReader in = new BufferedReader(new InputStreamReader(
                    socket.getInputStream()));
```

TCPServer (2/2)

```
            // create an output stream attached to the socket
            // OutputStream, PrintWriter
            PrintWriter outWriter = new PrintWriter(socket.getOutputStream(),
true);

            // read a line from a client.
            String request = in.readLine();

            System.out.println("Client> " + request);
            String reply = request.toUpperCase();
            System.out.println("Server> " + reply);

            // Send line to client.
            outWriter.println(reply);
            // loop back and wait for another client connection.
        }
    }
}
```

TCPClient (1/2)

```
import java.io.*;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Scanner;

public class TCPClient {

    public static void main(String[] args) throws UnknownHostException, IOException {
        final int PORT = 6789;
        final String HOST = "localhost";

        // create a scanner to take user input
        Scanner input = new Scanner(System.in);

        // create a client socket and connect to the server
        Socket clientSocket = new Socket(HOST, PORT);

        // create an input stream attached to the Socket
        BufferedReader in = new BufferedReader(new InputStreamReader(
                clientSocket.getInputStream()));

        // create an output stream attached to the socket
        PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
```

TCPClient (2/2)

```
 // Read a line from a user input
        System.out.print("Write a line for the server: ");
        String request = input.nextLine();
        System.out.println("Client> " + request);

        // Send line to server
        out.println(request);

        // Read line from the Server
        String reply = in.readLine();
        System.out.println("Server> " + reply);

        // Close connection
        clientSocket.close();
    }
}
```

Send/receive String objects
PrintWriter / BufferedReader

- Client sending a String

```
Socket s = new Socket(HOST, PORT);
PrintWriter outWriter = new
PrintWriter(s.getOutputStream(), true);
String data = “Don’t Worry";
outWriter.println(data);
```

- Server receiving a String

```
ServerSocket listenSocket = new ServerSocket(PORT);
Socket s = listenSocket.accept();
BufferedReader in = new BufferedReader(new InputStreamReader(
        s.getInputStream());
String data = in.readLine();
```

Send/receive String objects
DataOutputStream / DataInputStream

- Client sending a String

```
Socket s = new Socket(HOST, PORT);
DataOutputStream out = new DataOutputStream(s.getOutputStream());
String data = "Be Happy";
out.writeUTF(data);
```

- Server receiving a String

```
ServerSocket welcomeSocket = new ServerSocket(PORT);
Socket s = welcomeSocket.accept();
DataInputStream in = new DataInputStream(s.getInputStream());
String data = in.readUTF();
```

Send/receive primitive types
DataOutputStream / DataInputStream

- Client sending a double and an int

```
Socket s = new Socket(HOST, PORT);
DataOutputStream out = new
DataOutputStream(s.getOutputStream());
double data1 = 212.25;
out.writeDouble(data);
int data2 = 212;
out.writeInt(data);
```

- Server receiving a double and an int

```
ServerSocket welcomeSocket = new ServerSocket(PORT);
Socket s = welcomeSocket.accept();
DataInputStream in = new DataInputStream(s.getInputStream());
double data1 = in.readDouble();
int data2 = in.readInt();
```

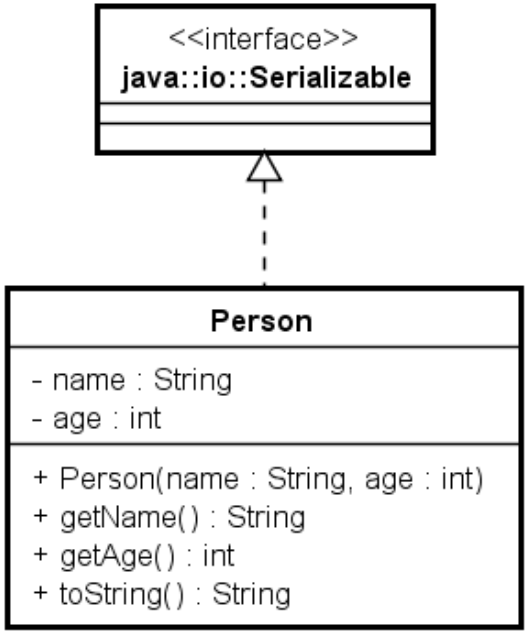
Send/receive (serialized) objects

- Client sending a Serializable object (a Person)

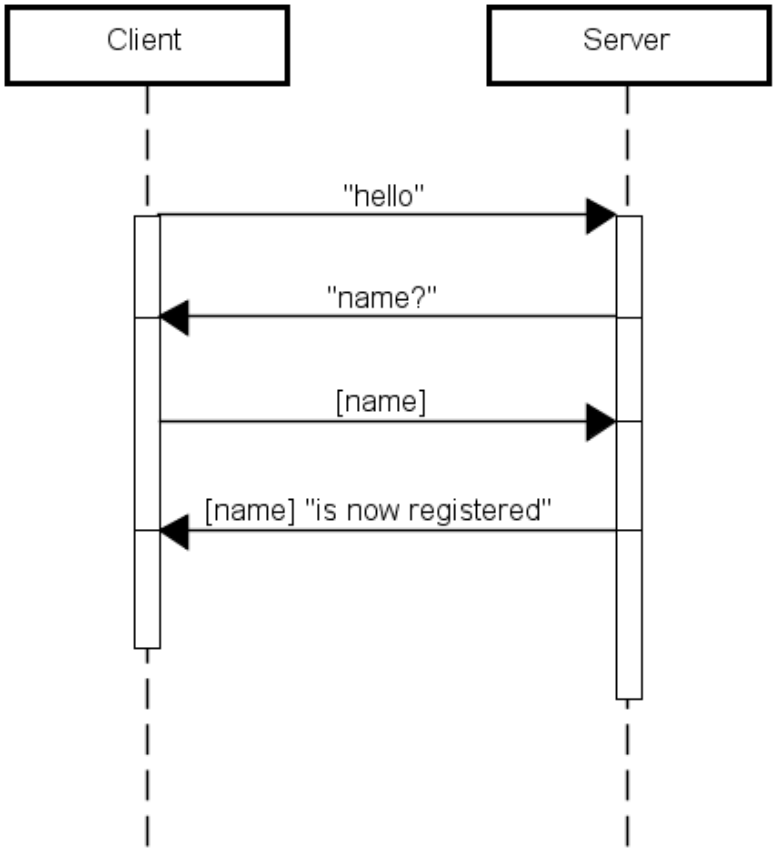
```
Socket s = new Socket(HOST, PORT);
ObjectOutputStream out = new
        ObjectOutputStream(s.getOutputStream());
Person person = new Person("Bob", 21);
out.writeObject(person);
```

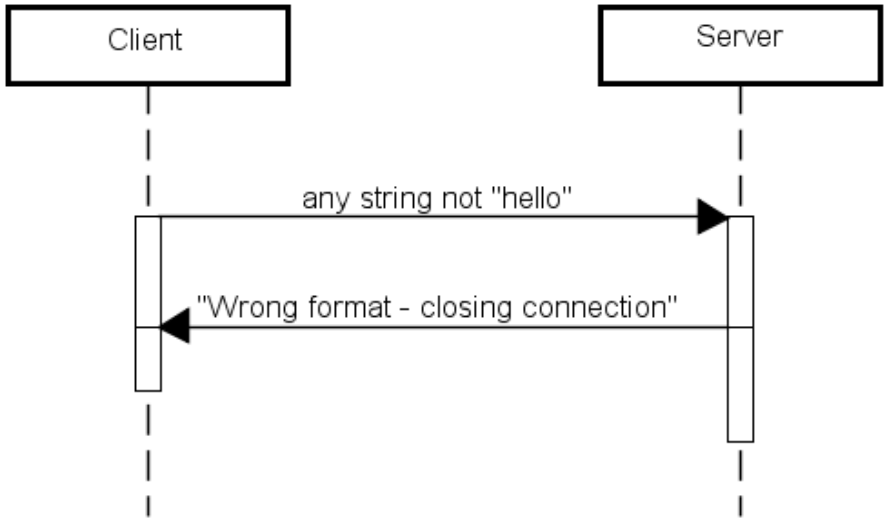
- Server receiving a Serializable object (a Person)

```
ServerSocket welcomeSocket = new ServerSocket(PORT);
Socket s = welcomeSocket.accept();
ObjectInputStream in =
        new ObjectInputStream(s.getInputStream());
Object data = in.readObject();
Person person = (Person)data;
```

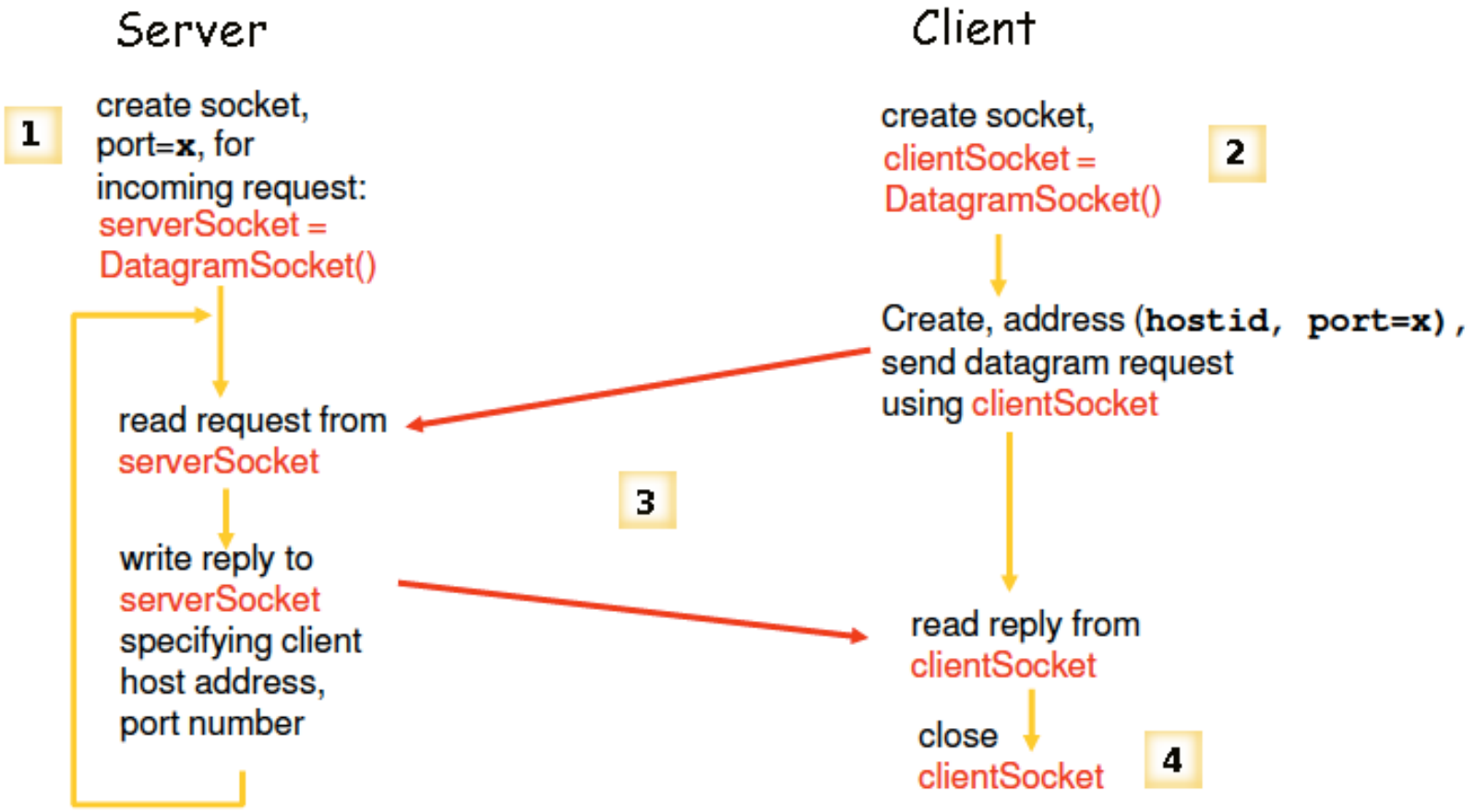


Communication flow (protocol)





UDP Sockets


Server
Client
1
create socket, port=x, for incoming request: serverSocket = DatagramSocket()
create socket, clientSocket = DatagramSocket()
2
Create, address (hostid, port=x), send datagram request using clientSocket
read request from serverSocket
3
write reply to serverSocket specifying client host address, port number
read reply from clientSocket
close clientSocket
4

UDPServer (1/2)

```java
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class UDPServer {
    public static void main(String args[]) throws IOException
    {
        final int PORT = 9876;
        System.out.println("Starting Server...");
        // Create UDP server socket at port 9876.
        DatagramSocket serverSocket = new DatagramSocket(PORT);

        while (true) {
            System.out.println("Waiting for a client...");

            // Create a space for receiving datagram
            byte[] receiveData = new byte[1024];
            DatagramPacket receivePacket = new DatagramPacket(receiveData,
                    receiveData.length);

            // Receive datagram from a client
            serverSocket.receive(receivePacket);
            String sentence = new String(receivePacket.getData()).trim();
```

UDPServer (2/2)

```
// Get the IP addr and port number of the client
            InetAddress IPAddress = receivePacket.getAddress();
            int port = receivePacket.getPort();

            System.out.println("Client:> " + sentence);
            String capitalizedSentence = sentence.toUpperCase();
            System.out.println("Server:> " + capitalizedSentence);

            byte[] sendData = new byte[1024];
            sendData = capitalizedSentence.getBytes();

            // create datagram to send to the client.
            DatagramPacket sendPacket = new DatagramPacket(sendData,
                    sendData.length, IPAddress, port);

            // Write out datagram to socket.
            serverSocket.send(sendPacket);
            // loop back and wait for another client connection
        }
    }
}
```

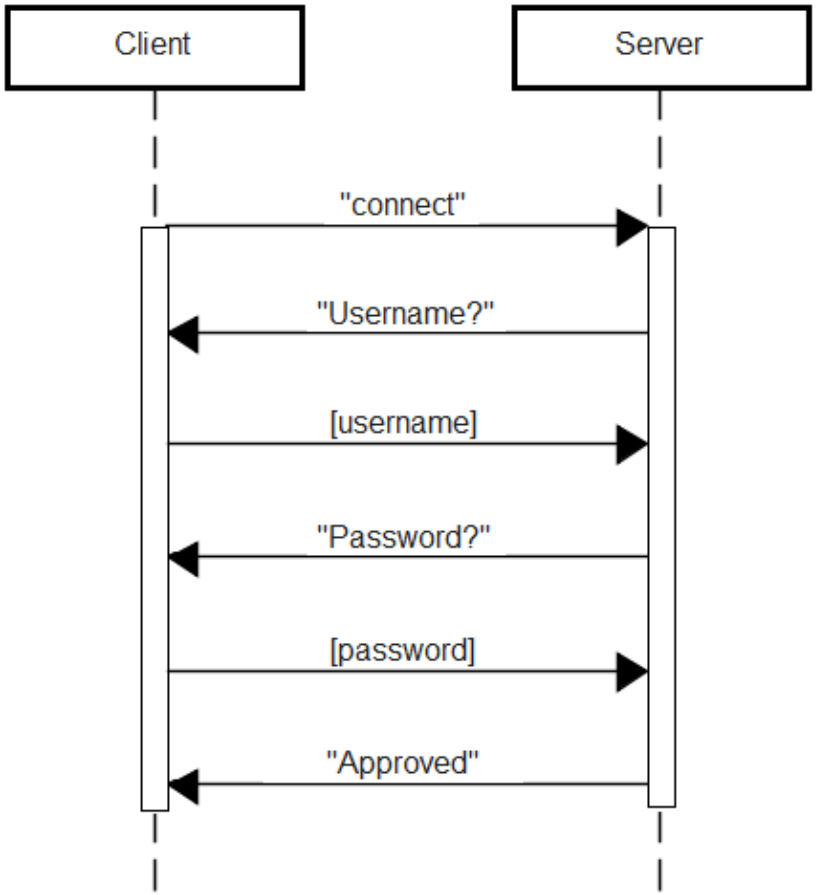
UDPClient (1/2)

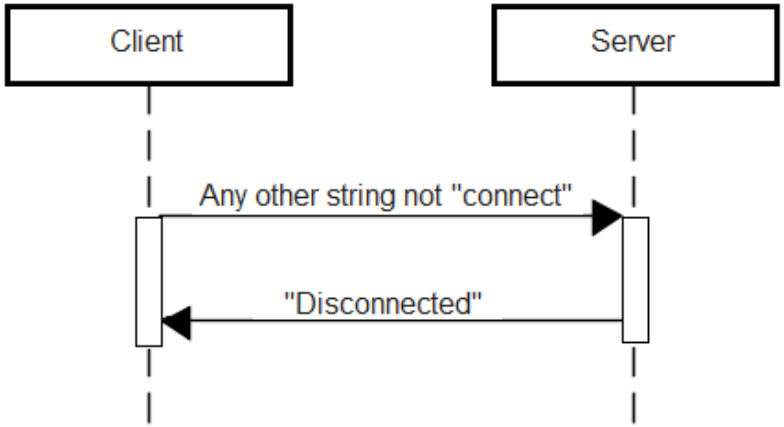
```java
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;

public class UDPClient {
    public static void main(String args[]) throws IOException
    {
        final int PORT = 9876;
        final String HOST = "localhost";

        // create an input scanner
        Scanner input = new Scanner(System.in);

        // create a client socket
        DatagramSocket clientSocket = new DatagramSocket();

        // Translate the hostname to IP using DNS
        InetAddress IPAddress = InetAddress.getByName(HOST);
        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];
```

UDPClient (2/2)

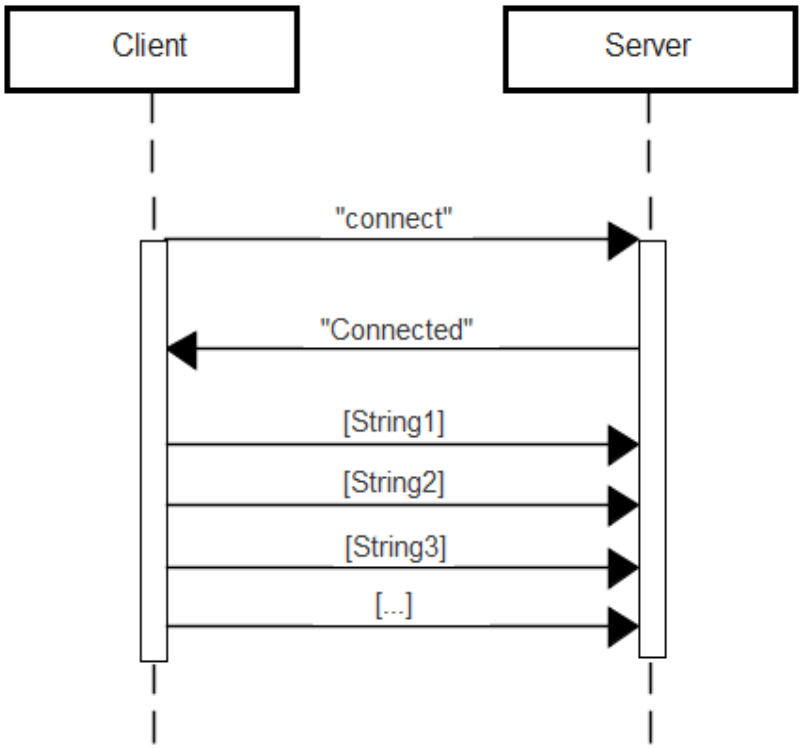
```
        // Read input from a user
        System.out.print("Write a line for the server: ");
        String sentence = input.nextLine();
        System.out.println("Client> " + sentence);
        sendData = sentence.getBytes();

        // Create a datagram with data-to-send, length, IP addr, port
        DatagramPacket sendPacket = new DatagramPacket(sendData,
                sendData.length,IPAddress, PORT);

        // Send datagram to server
        clientSocket.send(sendPacket);

        // Read datagram from server.
        DatagramPacket receivePacket = new DatagramPacket(receiveData,
                receiveData.length);
        clientSocket.receive(receivePacket);
        String modifiedStc = new String(receivePacket.getData()).trim();
        System.out.println("Server> " + modifiedStc);
        // Close connection.
        clientSocket.close();
    }
}
```

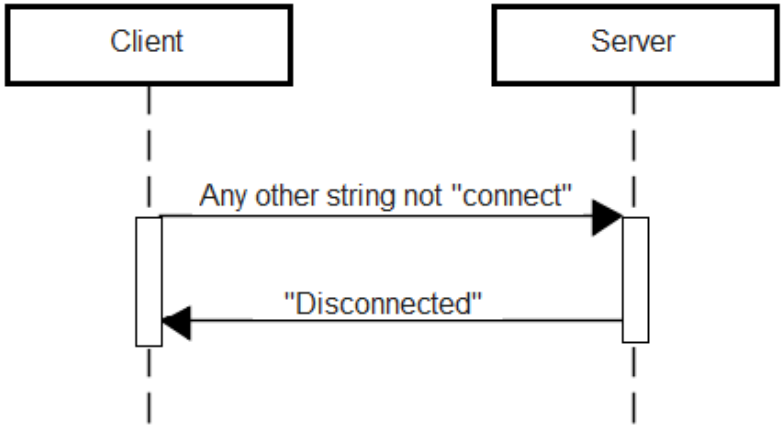
Exercises (a “login”)



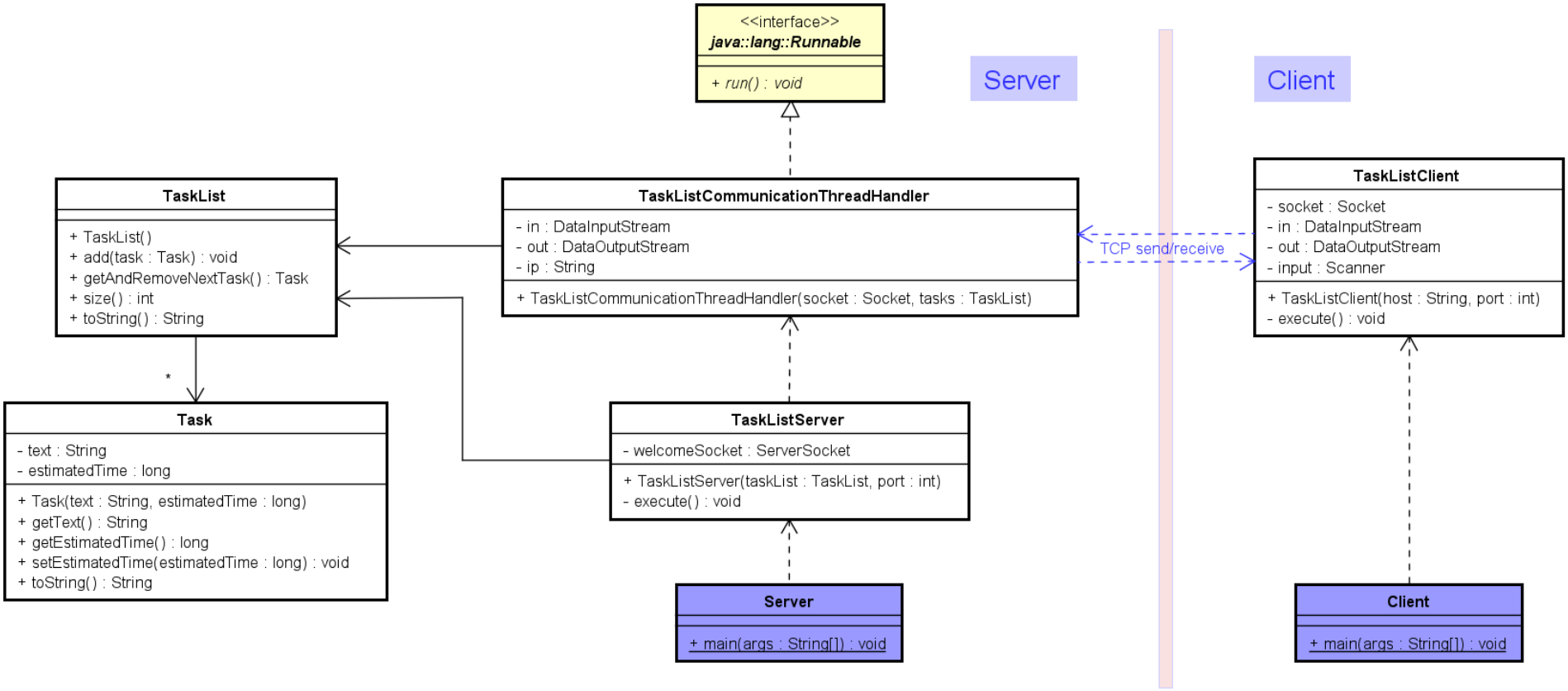


Exercises (a “chat”)





Exercises – a task list



Exercises – a task list