Java

A Simple Java Client

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

public class Client {

    protected String serverName;
    protected int serverPort;
    protected String message;

    public Client(String serverName, int serverPort, String message) {
        this.serverName = serverName;
        this.serverPort = serverPort;
        this.message = message;
    }

    public void connect() {
        try (
            Socket socket = new Socket(serverName, serverPort);
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
        ) {
            out.println(message);
            System.out.println(in.readLine());
```

Standard Java constructor; error checking could be added as an improvement

A Simple Java Client

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

public class Client {

    protected String serverName;
    protected int serverPort;
    protected String message;

    public Client(String serverName, int serverPort, String message) {
        this.serverName = serverName;
        this.serverPort = serverPort;
        this.message = message;
    }

    public void connect() {
        try (
            Socket socket = new Socket(serverName, serverPort);
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
        ) {
            out.println(message);
            System.out.println(in.readLine());
```

Classes in here are AutoClosable; close() is called automatically when exiting the try

A Simple Java Client

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

public class Client {

    protected String serverName;
    protected int serverPort;
    protected String message;

    public Client(String serverName, int serverPort, String message) {
        this.serverName = serverName;
        this.serverPort = serverPort;
        this.message = message;
    }

    public void connect() {
        try (
            Socket socket = new Socket(serverName, serverPort);
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
        ) {
            out.println(message);
            System.out.println(in.readLine());
```

Raw byte stream

A Simple Java Client

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

public class Client {

    protected String serverName;
    protected int serverPort;
    protected String message;

    public Client(String serverName, int serverPort, String message) {
        this.serverName = serverName;
        this.serverPort = serverPort;
        this.message = message;
    }

    public void connect() {
        try (
            Socket socket = new Socket(serverName, serverPort);
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
        ) {
            out.println(message);
            System.out.println(in.readLine());
```

PrintWriter provides a println() method that can be used to send a message + '\n' to the output stream

A Simple Java Client

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

public class Client {

    protected String serverName;
    protected int serverPort;
    protected String message;

    public Client(String serverName, int serverPort, String message) {
        this.serverName = serverName;
        this.serverPort = serverPort;
        this.message = message;
    }

    public void connect() {
        try (
            Socket socket = new Socket(serverName, serverPort);
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
        ) {
            out.println(message);
            System.out.println(in.readLine());
```

There is no PrintReader

A Simple Java Client

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

public class Client {

    protected String serverName;
    protected int serverPort;
    protected String message;

    public Client(String serverName, int serverPort, String message) {
        this.serverName = serverName;
        this.serverPort = serverPort;
        this.message = message;
    }

    public void connect() {
        try (
            Socket socket = new Socket(serverName, serverPort);
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
        ) {
            out.println(message);
            System.out.println(in.readLine());
```

Raw byte stream

A Simple Java Client

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

public class Client {

    protected String serverName;
    protected int serverPort;
    protected String message;

    public Client(String serverName, int serverPort, String message) {
        this.serverName = serverName;
        this.serverPort = serverPort;
        this.message = message;
    }

    public void connect() {
        try (
            Socket socket = new Socket(serverName, serverPort);
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
        ) {
            out.println(message);
            System.out.println(in.readLine());
```

Decoded character stream

A Simple Java Client

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

public class Client {

    protected String serverName;
    protected int serverPort;
    protected String message;

    public Client(String serverName, int serverPort, String message) {
        this.serverName = serverName;
        this.serverPort = serverPort;
        this.message = message;
    }

    public void connect() {
        try (
            Socket socket = new Socket(serverName, serverPort);
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
        ) {
            out.println(message);
            System.out.println(in.readLine());
```

Buffered I/O for better performance

A Simple Java Client

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

public class Client {

    protected String serverName;
    protected int serverPort;
    protected String message;

    public Client(String serverName, int serverPort, String message) {
        this.serverName = serverName;
        this.serverPort = serverPort;
        this.message = message;
    }

    public void connect() {
        try (
            Socket socket = new Socket(serverName, serverPort);
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
        ) {
            out.println(message);
            System.out.println(in.readLine());
```

Write to the server

A Simple Java Client

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

public class Client {

    protected String serverName;
    protected int serverPort;
    protected String message;

    public Client(String serverName, int serverPort, String message) {
        this.serverName = serverName;
        this.serverPort = serverPort;
        this.message = message;
    }

    public void connect() {
        try (
            Socket socket = new Socket(serverName, serverPort);
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
        ) {
            out.println(message);
            System.out.println(in.readLine());
```

Read from the server

A Simple Java Client

```
        } catch (UnknownHostException e) {
            System.err.println(e);
            System.exit(-1);
        } catch (IOException e) {
            System.err.println(e);
            System.exit(-2);
        } catch (SecurityException e) {
            System.err.println(e);
            System.exit(-3);
        } catch (IllegalArgumentException e) {
            System.err.println(e);
            System.exit(-4);
        }
    }
}
```

IP address of the host
could not be determined

A Simple Java Client

```
        } catch (UnknownHostException e) {
            System.err.println(e);
            System.exit(-1);
        } catch (IOException e) {
            System.err.println(e);
            System.exit(-2);
        } catch (SecurityException e) {
            System.err.println(e);
            System.exit(-3);
        } catch (IllegalArgumentException e) {
            System.err.println(e);
            System.exit(-4);
        }
    }
}
```

Write to stderr, not stdout!

A Simple Java Client

```
        } catch (UnknownHostException e) {
            System.err.println(e);
            System.exit(-1);
        } catch (IOException e) {
            System.err.println(e);
            System.exit(-2);
        } catch (SecurityException e) {
            System.err.println(e);
            System.exit(-3);
        } catch (IllegalArgumentException e) {
            System.err.println(e);
            System.exit(-4);
        }
    }
}
```

I/O error occurs when creating the socket (e.g., while creating the output stream)

A Simple Java Client

```
        } catch (UnknownHostException e) {
            System.err.println(e);
            System.exit(-1);
        } catch (IOException e) {
            System.err.println(e);
            System.exit(-2);
        } catch (SecurityException e) {
            System.err.println(e);
            System.exit(-3);
        } catch (IllegalArgumentException e) {
            System.err.println(e);
            System.exit(-4);
        }
    }
}
```

Java can restrict access to Sockets if a security manager exists and its checkConnect method doesn't allow the operation

A Simple Java Client

```
        } catch (UnknownHostException e) {
            System.err.println(e);
            System.exit(-1);
        } catch (IOException e) {
            System.err.println(e);
            System.exit(-2);
        } catch (SecurityException e) {
            System.err.println(e);
            System.exit(-3);
        } catch (IllegalArgumentException e) {
            System.err.println(e);
            System.exit(-4);
        }
    }
}
```

Invalid port (outside the specified range of valid port values from 0 to 65535, inclusive)

Socket Class

- Socket (String host, int port) - Creates a stream socket and connects it to the specified port number on the named host.
- UnknownHostException - if the IP address of the host could not be determined.
- IOException - if an I/O error occurs when creating the socket.
- SecurityException - if a security manager exists and its checkConnect method doesn't allow the operation.
- IllegalArgumentException - if the port parameter is outside the specified range of valid port values, which is between 0 and 65535, inclusive.

docs.oracle.com

OutputStream Class

- void flush() - Flushes this output stream and forces any buffered output bytes to be written out.
- void write (byte[] b) - Writes b.length bytes from the specified byte array to this output stream.
- void write (byte[] b, int off, int len) - Writes len bytes from the specified byte array starting at offset off to this output stream.
- IOException - if an I/O error occurs.

docs.oracle.com

PrintWriter Class

- PrintWriter (OutputStream out, boolean autoFlush) - Creates a new PrintWriter from an existing OutputStream. This convenience constructor creates the necessary intermediate OutputStreamWriter, which will convert characters into bytes using the default character encoding.

- autoFlush - A boolean; if true, the println, printf, or format methods will flush the output buffer

If you ever find that one side sends data and the other side does not receive it, especially the last couple of bytes, make sure autoflush was set (if there is no autoflush, you must flush the buffer manually by calling <name of stream>.flush(), e.g., out.flush()

docs.oracle.com

PrintWriter Class

- public void println (String x) - Prints a String and then terminates the line. This method behaves as though it invokes print(String) and then println().

docs.oracle.com

InputStream Class

- int read (byte[] b) - Reads some number of bytes from the input stream and stores them into the buffer array *b*...Returns: the total number of bytes read into the buffer, or -1 if there is no more data because the end of the stream has been reached.

- int read (byte[] b, int off, int len) - Reads up to *len* bytes of data from the input stream into an array of bytes, starting at offset *off*...Returns: the total number of bytes read into the buffer, or -1 if there is no more data because the end of the stream has been reached.

- byte[] readAllBytes() - Reads all remaining bytes from the input stream; blocks until the stream ends, or an exception occurs

docs.oracle.com

InputStream Class

- byte[] readNBytes (int len) - Reads up to a specified number of bytes from the input stream; blocks until len bytes have been read, the stream ends, or an exception occurs

- int readNBytes (byte[] b, int off, int len) - Reads the requested number of bytes from the input stream into the given byte array starting at offset *off*; blocks until len bytes have been read, the stream ends, or an exception occurs...Returns: the actual number of bytes read into the buffer

docs.oracle.com

InputStream Class

- IOException - If the first byte cannot be read for any reason other than end of file, or if the input stream has been closed, or if some other I/O error occurs.
- NullPointerException - If *b* is null.
- IndexOutOfBoundsException - If *off* is negative, *len* is negative, or *len* is greater than *b.length - off*

docs.oracle.com

InputStreamReader Class

- InputStreamReader (InputStream in) - Creates an InputStreamReader that uses the default charset.

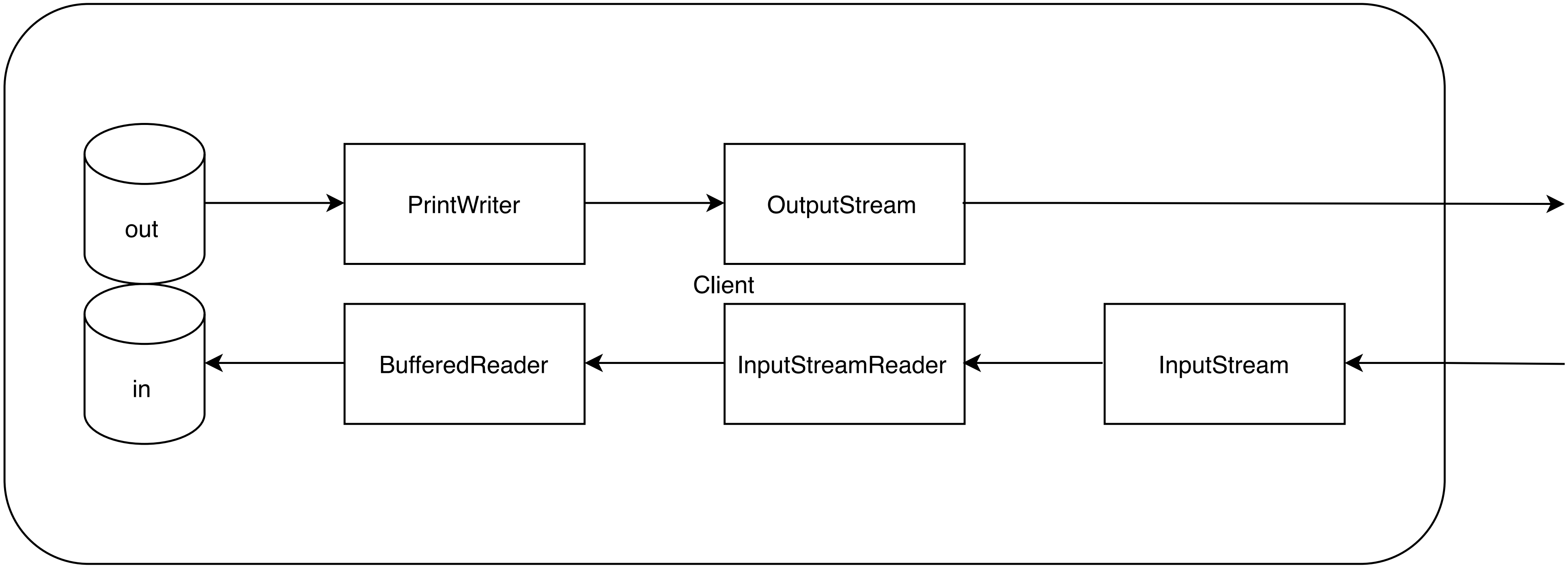
docs.oracle.com

BufferedReader Class

- BufferedReader (Reader in) - Creates a buffering character-input stream that uses a default-sized input buffer.
- public String readLine() - Reads a line of text. A line is considered to be terminated by any one of a line feed ('\n'), a carriage return ('\r'), a carriage return followed immediately by a line feed, or by reaching the end-of-file (EOF).
- IOException - If an I/O error occurs

docs.oracle.com

Client


out
PrintWriter
OutputStream
Client
in
BufferedReader
InputStreamReader
InputStream

Client Review

- Socket(serverName, serverPort) - Connects a client to serverName at serverPort; getOutputStream and getInputStream allow binary data to be sent and received, respectively, over this connection
- OutputStream() - provides methods to write binary data to and flush the output stream
- PrintWriter(OutputStream out, boolean autoFlush) - provides println(), encoding, and automatic flushing
- InputStream() - provides methods to read binary data from the input stream
- InputStreamReader(InputStream in) - decodes binary data from the InputStream using the default encoding
- BufferedReader(Reader in) - buffers decoded data; provides readLine()

A Simple Java Server

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

public class Server {

    protected int port;

    public Server(int port) {
        this.port = port;
    }

    public void serve() {
        try (
            ServerSocket serverSocket = new ServerSocket(port);
            Socket clientSocket = serverSocket.accept();
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),
true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
        ) {
```

Caller must validate the port

A Simple Java Server

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

public class Server {

    protected int port;

    public Server(int port) {
        this.port = port;
    }

    public void serve() {
        try (
            ServerSocket serverSocket = new ServerSocket(port);
            Socket clientSocket = serverSocket.accept();
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),
true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
        ) {
```

Open a socket

A Simple Java Server

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

public class Server {

    protected int port;

    public Server(int port) {
        this.port = port;
    }

    public void serve() {
        try (
            ServerSocket serverSocket = new ServerSocket(port);
            Socket clientSocket = serverSocket.accept();
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),
true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
        ) {
```

Wait for a connection

A Simple Java Server

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

public class Server {

    protected int port;

    public Server(int port) {
        this.port = port;
    }

    public void serve() {
        try (
            ServerSocket serverSocket = new ServerSocket(port);
            Socket clientSocket = serverSocket.accept();
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),
true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
        ) {
```

Raw binary

A Simple Java Server

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

public class Server {

    protected int port;

    public Server(int port) {
        this.port = port;
    }

    public void serve() {
        try (
            ServerSocket serverSocket = new ServerSocket(port);
            Socket clientSocket = serverSocket.accept();
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),
true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
        ) {
```

Provides println

A Simple Java Server

Raw binary

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

public class Server {

    protected int port;

    public Server(int port) {
        this.port = port;
    }

    public void serve() {
        try (
            ServerSocket serverSocket = new ServerSocket(port);
            Socket clientSocket = serverSocket.accept();
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),
true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
        ) {
```

A Simple Java Server

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

public class Server {

    protected int port;

    public Server(int port) {
        this.port = port;
    }

    public void serve() {
        try (
            ServerSocket serverSocket = new ServerSocket(port);
            Socket clientSocket = serverSocket.accept();
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),
true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
        ) {
```

Decoder

A Simple Java Server

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

public class Server {

    protected int port;

    public Server(int port) {
        this.port = port;
    }

    public void serve() {
        try (
            ServerSocket serverSocket = new ServerSocket(port);
            Socket clientSocket = serverSocket.accept();
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),
true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
        ) {
```

Provides readln

A Simple Java Server

```
            while (true) {
                String inputLine = in.readLine();
                if (inputLine == null) {
                    break;
                }
                System.out.println(inputLine);
                out.println(inputLine);
            }
        } catch (IOException e) {
            System.err.println(e);
            System.exit(-2);
        } catch (SecurityException e) {
            System.err.println(e);
            System.exit(-3);
        } catch (IllegalArgumentException e) {
            System.err.println(e);
            System.exit(-4);
        } catch (IllegalBlockingModeException e) {
            System.err.println(e);
            System.exit(-6);
        }
    }
}
```

ServerSocket Class

- ServerSocket (int port) - Creates a server socket, bound to the specified port.
- ServerSocket (int port, int backlog) - Creates a server socket and binds it to the specified local port number, with the specified backlog.
- IOException - if an I/O error occurs when opening the socket.
- SecurityException - if a security manager exists and its checkListen method doesn't allow the operation.
- IllegalArgumentException - if the port parameter is outside the specified range of valid port values, which is between 0 and 65535, inclusive.

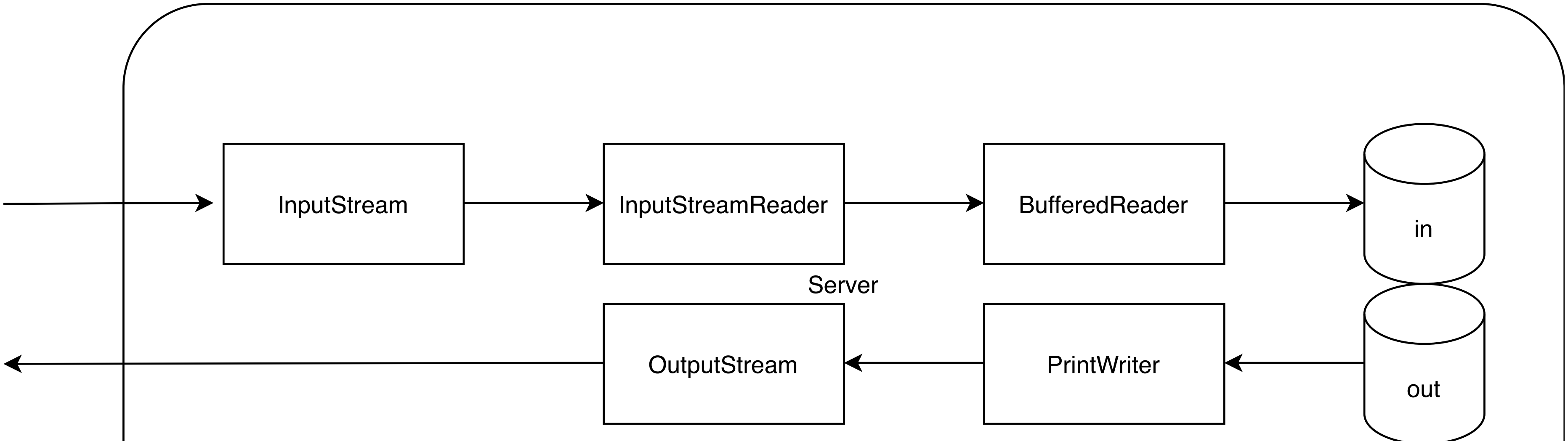
docs.oracle.com

ServerSocket Class

- Socket accept() - Listens for a connection to be made to this socket and accepts it.
- IOException - if an I/O error occurs when waiting for a connection.
- SecurityException - if a security manager exists and its checkAccept method doesn't allow the operation.
- IllegalBlockingModeException - if this socket has an associated channel, the channel is in non-blocking mode, and there is no connection ready to be accepted [*Note: This only applies when using SocketChannels, which are essentially non-blocking Sockets*]

docs.oracle.com

Server


InputStream
InputStreamReader
BufferedReader
in
Server
OutputStream
PrintWriter
out

Server Review

- ServerSocket(int port) - Creates a server socket, bound to the specified port.

- OutputStream() - provides methods to write binary data to and flush the output stream

- PrintWriter(OutputStream out, boolean autoFlush) - provides println and automatic flushing

- InputStream() - provides methods to read binary data from the input stream

- InputStreamReader(InputStream in) - decodes binary data from the InputStream using the default encoding

- BufferedReader(Reader in) - buffers decoded data; provides readLine()

Java Compression

File I/O Review

- The different stream classes in Java open up powerful combinations
- In review:
  - FileInputStream(filename) - given the name of a valid, accessible file, opens the file for reading and provides an InputStream handle
  - BufferedInputStream(fileInputStream) - similar to a BufferedReader, but designed for binary data
  - FileOutputStream(filename) - given the name of a valid, accessible file, opens the file for writing and provides an OutputStream handle
  - BufferedOutputStream(fileOutputStream) - similar to a BufferedWriter, but designed for binary data

Compression

- This approach makes it easy to plug in new functionality, such as compression:
  - GZIPInputStream(InputStream in) - Creates a new input stream with a default buffer size.
  - int read(byte[] buf, int off, int len) - Reads uncompressed data into an array of bytes.
  - GZIPOutputStream(OutputStream out) - Creates a new output stream with a default buffer size.
  - void - write(byte[] buf, int off, int len)

docs.oracle.com

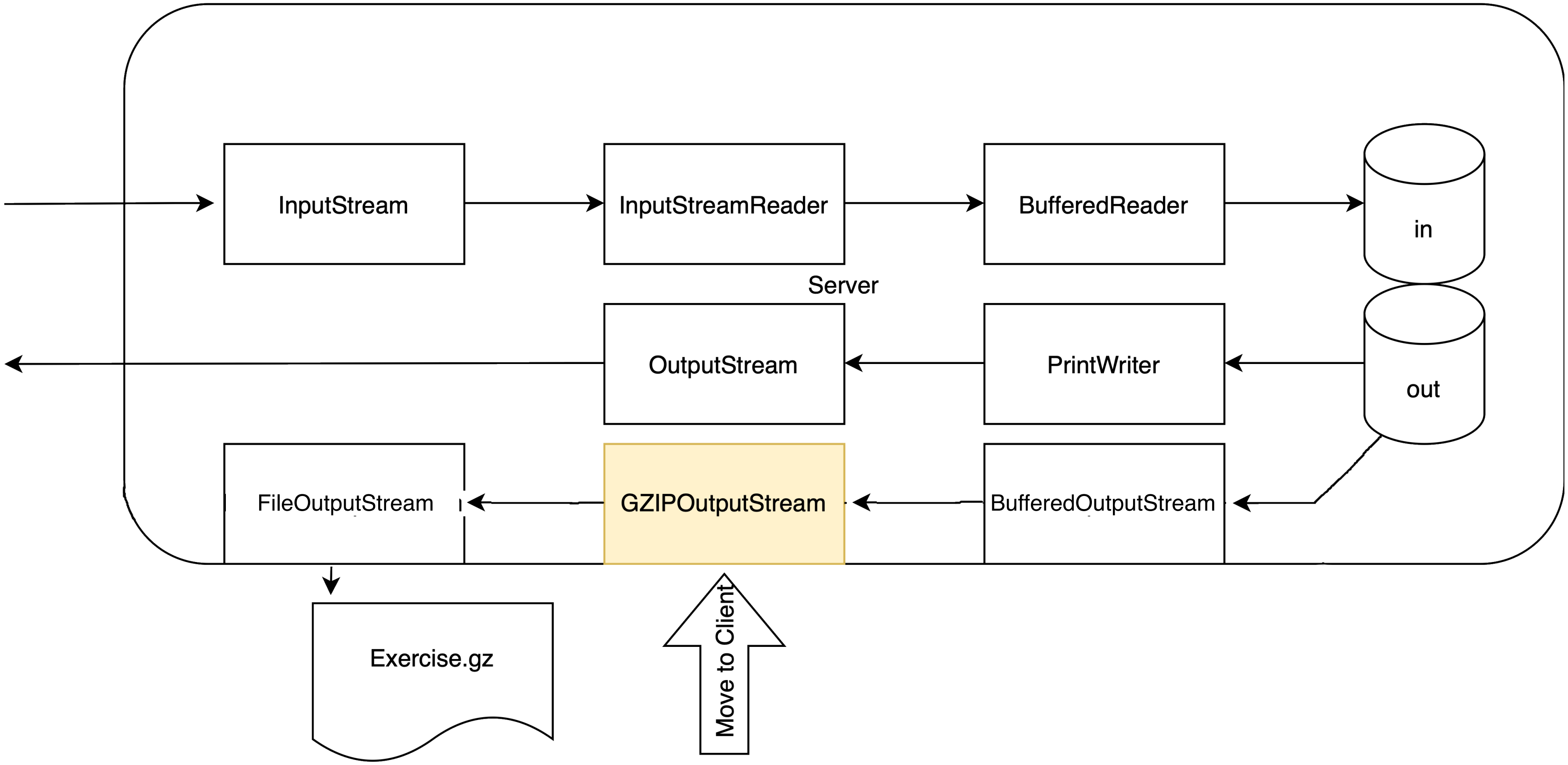
Example

```
import java.net.*;
import java.io.*;
import java.nio.channels.*;
import java.util.zip.*;

public class Server {

 protected int port;

  public Server(int port) {
    this.port = port;
  }

  public void serve() {
    try (
      ServerSocket serverSocket = new ServerSocket(port);
      Socket clientSocket = serverSocket.accept();
      PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
      BufferedReader in = new BufferedReader(
        new InputStreamReader(clientSocket.getInputStream()));
      BufferedOutputStream outFile = new BufferedOutputStream(
        new GZIPOutputStream(new FileOutputStream("Exercise.gz")));
      ) {
```

Example

```
      while (true) {
        String inputLine = in.readLine();
        if (inputLine == null) {
          break;
        }

        System.out.println(inputLine);
        out.println(inputLine);
        outFile.write(inputLine.getBytes(), 0, inputLine.length());
      }
    } catch (IOException e) {
      System.err.println(e);
      System.exit(-2);
    } catch (SecurityException e) {
      System.err.println(e);
      System.exit(-3);
    } catch (IllegalArgumentException e) {
      System.err.println(e);
      System.exit(-4);
    } catch (IllegalBlockingModeException e) {
      System.err.println(e);
      System.exit(-6);
    }
  }
}
```

Server


InputStream
InputStreamReader
BufferedReader
in
Server
OutputStream
PrintWriter
out
FileOutputStream
GZIPOutputStream
BufferedOutputStream
Exercise.gz
Move to Client

Exercises

- Copy the Java client and server (with file compression) to your VM
- Run the program and observe the traffic in Wireshark
- Confirm that anything you typed in the client is saved in Exercise.gz (you can use zcat to view the contents of this file)
- Modify the client so that compression is done on the **client** end, not the server end; no need for the client to wait for a reply
- Run the program and observe the compressed traffic in Wireshark
- Make sure the contents can still be views using zcat

Exercises

- Use javac, java provided by openjdk-11-jdk-headless
- Have to add a main to the Server and Client files

```
public static void main(String[] args) {
    Server s = new Server(12345);
    s.serve();
}

public static void main(String[] args) {
    Client c = new Client("127.0.0.01", 12345, "This is a test");
    c.connect();
}
```

Synchronous Multithreaded Server Using Sockets

A Simple Client

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

public class Client {
    protected String serverName;
    protected int serverPort;
    protected String message;

    public Client(String serverName, int serverPort, String message) {
        this.serverName = serverName;
        this.serverPort = serverPort;
        this.message = message;
    }

    public void connect() {
        String reply;

        try (
            Socket socket = new Socket(serverName, serverPort);
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
        ) {
```

Contact server at this address and port

A Simple Client

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

public class Client {
    protected String serverName;
    protected int serverPort;
    protected String message;

    public Client(String serverName, int serverPort, String message) {
        this.serverName = serverName;
        this.serverPort = serverPort;
        this.message = message;
    }

    public void connect() {
        String reply;

        try (
            Socket socket = new Socket(serverName, serverPort);
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
        ) {
```

PrintWriter to send message via println

A Simple Client

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

public class Client {
    protected String serverName;
    protected int serverPort;
    protected String message;

    public Client(String serverName, int serverPort, String message) {
        this.serverName = serverName;
        this.serverPort = serverPort;
        this.message = message;
    }

    public void connect() {
        String reply;

        try (
            Socket socket = new Socket(serverName, serverPort);
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
        ) {
```

BufferedReader to read message via readLine

A Simple Client

Send the message

```
            while (true) {
                out.println(this.message);
                if ((reply = in.readLine()) == null) {
                    break;
                }
                System.out.println(reply);
                Thread.sleep(1000);
            }
        } catch (Exception e) {
            System.err.println(e);
            System.exit(-1);
        }
    }

    public static void main(String[] args) {
        if (args.length != 3) {
            System.err.println("Need <host> <port> <message>");
            System.exit(-2);
        }
        Client c = new Client(args[0], Integer.valueOf(args[1]), args[2]);
        c.connect();
    }
}
```

A Simple Client

```
            while (true) {
                out.println(this.message);
                if ((reply = in.readLine()) == null) {
                    break;
                }
                System.out.println(reply);
                Thread.sleep(1000);
            }
        } catch (Exception e) {
            System.err.println(e);
            System.exit(-1);
        }
    }

    public static void main(String[] args) {
        if (args.length != 3) {
            System.err.println("Need <host> <port> <message>");
            System.exit(-2);
        }
        Client c = new Client(args[0], Integer.valueOf(args[1]), args[2]);
        c.connect();
    }
}
```

Read the reply from the server; stop if null

A Simple Client

```
            while (true) {
                out.println(this.message);
                if ((reply = in.readLine()) == null) {
                    break;
                }
                System.out.println(reply);
                Thread.sleep(1000);
            }
        } catch (Exception e) {
            System.err.println(e);
            System.exit(-1);
        }
    }

    public static void main(String[] args) {
        if (args.length != 3) {
            System.err.println("Need <host> <port> <message>");
            System.exit(-2);
        }
        Client c = new Client(args[0], Integer.valueOf(args[1]), args[2]);
        c.connect();
    }
}
```

Print the reply to stdout

A Synchronous, Multithreaded Server

- Our Java servers so far only handled one connection at a time
- Servers should be able to handle multiple clients concurrently
- Traditionally,
  - each connection is handled by its own thread
  - accepting a connection and reading in data are blocking (synchronous) operations

A Synchronous, Multithreaded Server

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


public class Server {
    protected final String HOST = "";
    protected int port;

    public Server(int port) {
        this.port = port;
    }
```

Called by threads

```
    void delegate(Socket clientSocket) {
        try (
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),
                true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
        ) {
            while (true) {
                String inputLine = in.readLine();
                if (inputLine == null) {
                    break;
                }
```

A Synchronous, Multithreaded Server

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


public class Server {
    protected final String HOST = "";
    protected int port;

    public Server(int port) {
        this.port = port;
    }

    void delegate(Socket clientSocket) {
        try (
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),
                true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
        ) {
            while (true) {
                String inputLine = in.readLine();
                if (inputLine == null) {
                    break;
                }
```

PrintWriter and BufferedReader like on the client

A Synchronous, Multithreaded Server

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


public class Server {
    protected final String HOST = "";
    protected int port;

    public Server(int port) {
        this.port = port;
    }

    void delegate(Socket clientSocket) {
        try (
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),
                true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
        ) {
            while (true) {
                String inputLine = in.readLine();
                if (inputLine == null) {
                    break;
                }
```

Blocks until newline is read in

A Synchronous, Multithreaded Server

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


public class Server {
    protected final String HOST = "";
    protected int port;

    public Server(int port) {
        this.port = port;
    }

    void delegate(Socket clientSocket) {
        try (
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),
                true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
        ) {
            while (true) {
                String inputLine = in.readLine();
                if (inputLine == null) {
                    break;
                }
```

Stop this loop if inputLine is null

A Synchronous, Multithreaded Server

Avoids scrambled output. Without this, we would have a race condition. Use synchronized(this) whenever using a shared resource (stdout, shared variable, shared file, etc.)

```
               synchronized(this) {
                   System.out.println(inputLine);
               }
               out.println(Thread.currentThread() + inputLine);
           }
           clientSocket.close();
       } catch (Exception e) {
           System.err.println(e);
           System.exit(-1);
       }
   }

   public void serve() {
       try (
           ServerSocket serverSocket = new ServerSocket(port);
       ) {
           while(true) {
              try {
                   Socket clientSocket = serverSocket.accept();
```

A Synchronous, Multithreaded Server

```
            synchronized(this) {
                System.out.println(inputLine);
            }
            out.println(Thread.currentThread() + inputLine);
        }
        clientSocket.close();
    } catch (Exception e) {
        System.err.println(e);
        System.exit(-1);
    }
}

public void serve() {
    try (
        ServerSocket serverSocket = new ServerSocket(port);
    ) {
        while(true) {
           try {
                Socket clientSocket = serverSocket.accept();
```

Send a reply
to the client

A Synchronous, Multithreaded Server

```
            synchronized(this) {
                System.out.println(inputLine);
            }
            out.println(Thread.currentThread() + inputLine);
        }
        clientSocket.close();
    } catch (Exception e) {
        System.err.println(e);
        System.exit(-1);
    }
}

public void serve() {
    try (
        ServerSocket serverSocket = new ServerSocket(port);
    ) {
        while(true) {
           try {
                Socket clientSocket = serverSocket.accept();
```

Don't put clientSocket into resources clause of the try, or it will be closed before the thread has a chance to run

A Synchronous, Multithreaded Server

```
            synchronized(this) {
                System.out.println(inputLine);
            }
            out.println(Thread.currentThread() + inputLine);
        }
        clientSocket.close();
    } catch (Exception e) {
        System.err.println(e);
        System.exit(-1);
    }
}

public void serve() {
    try (
        ServerSocket serverSocket = new ServerSocket(port);
    ) {
        while(true) {
           try {
                Socket clientSocket = serverSocket.accept();
```

Blocks until a connection is made

A Synchronous, Multithreaded Server

```
                    Runnable runnable = () -> this.delegate(clientSocket);
                    Thread t = new Thread(runnable);
                    t.start();
                } catch (Exception e) {
                    System.err.println(e);
                   System.exit(-2);
                }
            }
       } catch (Exception e) {
            System.err.println(e);
            System.exit(-3);
        }
    }

    public static void main(String[] args) {
        if (args.length != 1) {
            System.err.println("Need <port>");
            System.exit(-99);
        }
        Server s = new Server(Integer.valueOf(args[0]));
        s.serve();
}}
```

Every connection runs in its own thread

Sample Run

```
# java Client localhost 12345 Hi

Thread[Thread-0,5,main]Hi

Thread[Thread-0,5,main]Hi

Thread[Thread-0,5,main]Hi

…
```

```
# java Client localhost 12345 Bye

Thread[Thread-1,5,main]Bye

Thread[Thread-1,5,main]Bye

Thread[Thread-1,5,main]Bye

…
```

Exercises

- If you remove the Thread.sleep(1000) from the Client, you will notice that sometimes Hi and Hello are not perfectly interleaved on the Server
- That is because turns are not enforced
- Modify the code so that turns are strictly enforced

Exercises

- On your own:
    - Work on the questions in the *Java* section of *Practice Questions and Solutions*

Key Skills

- Write a client and server in Java
- Compress and decompress data using the GZIPOutputstream and GZIPInputStream
- Write a synchronous, multi-threaded server that can support multiple simultaneous client connections