

# *Java Socket Programming*



# *Keyboard Access in Java*

- Java provides access to the standard input stream (stdin)
- `java.lang.System.in`
  - ‘java.lang’ is the package
  - ‘System’ is the class
  - ‘in’ is an object of type `InputStream`
  - `InputStream` is defined in the `java.io` package

# *Keyboard Access in Java*

```
int cc;
While(true) {
    try {
        cc=System.in.read();
    } catch (IOException ex) {}
    System.out.write(cc);
}
```

# *java.io.InputStreamReader*

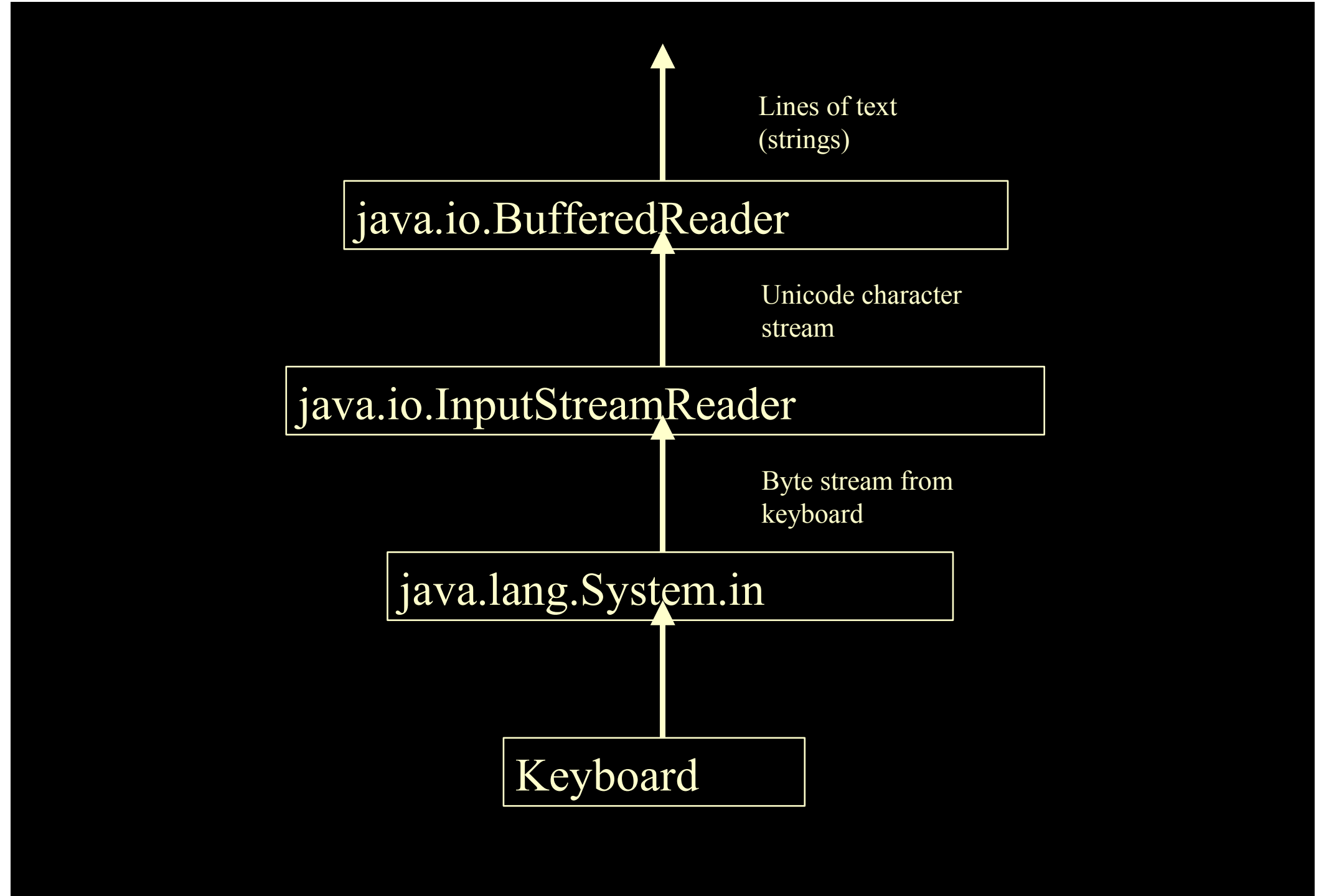
- This class provides a character interface to input stream (also converts to unicode characters)

```
int cc;  
InputStreamReader inkey;  
Inkey=new InputStreamReader(System.in);  
  
While(true) {  
    try {  
        cc = inkey.read();  
    } catch (IOException ex) {}  
    System.out.print(cc);  
}
```

# *java.io.BufferedReader*

- Provides a buffer for the input stream—can be accessed on a line by line basis

```
String s;  
InputStreamReader inkey;  
BufferedReader bufkey;  
inkey=new InputStreamReader(System.in);  
bufkey=new BufferedReader(inkey);  
  
While(true) {  
    s=bufkey.readLine();  
    System.out.println(s);  
}
```



# *File access in Java*

- `java.io.File`
  - This allows you to open a file
  - Eg. `File infile = new File("Blah.txt");`
- `java.io.FileInputStream`
  - This sets up an input stream from the file
    - `FileInputStream instream = new  
FileInputStream(infile);`
  - `FileInputStream` can open a file directly...
    - `FileInputStream instream = new  
FileInputStream("Blah.txt");`

# *File access in Java*

```
FileInputStream instream;  
instream = new FileInputStream("blah.txt")  
int cc;  
while ((cc=instream.read()) != -1) {  
    System.out.print((char)cc);  
}
```

## *File Access in Java*

- `FileInputStream` is a subclass of `InputStream`.
- Recall that the `System.in` object, used for keyboard input is also of class `InputStream`.
- This means that we can use `InputStreamReader` and `BufferedReader` just as with the keyboard

```
import java.io.*;

class uppercase {
    public static void main(String [] args) {
        FileInputStream instream;
        FileOutputStream ostream;
        BufferedReader bufinstream;
        BufferedWriter bufostream;

        if (args.length != 2) {
            System.out.println("usage: file <infile> "
                + "<outfile>");
            System.exit(1);
        }

        String infile = args[0];
        String outfile = args[1];
```

```
try {
    instream = new FileInputStream(infile);
    ostream = new FileOutputStream(outfile);
    bufinstream = new BufferedReader(new
        InputStreamReader(instream));
    bufostream = new BufferedWriter(new
        OutputStreamWriter(ostream));

    String c;
    String d;
    while ((c=bufinstream.readLine())!=null) {
        d = c.toUpperCase();
        bufostream.write(d, 0, d.length());
        bufostream.newLine();
    }
    bufinstream.close();
    bufostream.close();
} catch (IOException e) {
    System.err.println("oops");
}
}
```

```
mhall@goblin:~>more blah.txt
```

```
test1
```

```
test2
```

```
Blahblahblah
```

```
mhall@goblin:~>java uppercase blah.txt blah2.txt
```

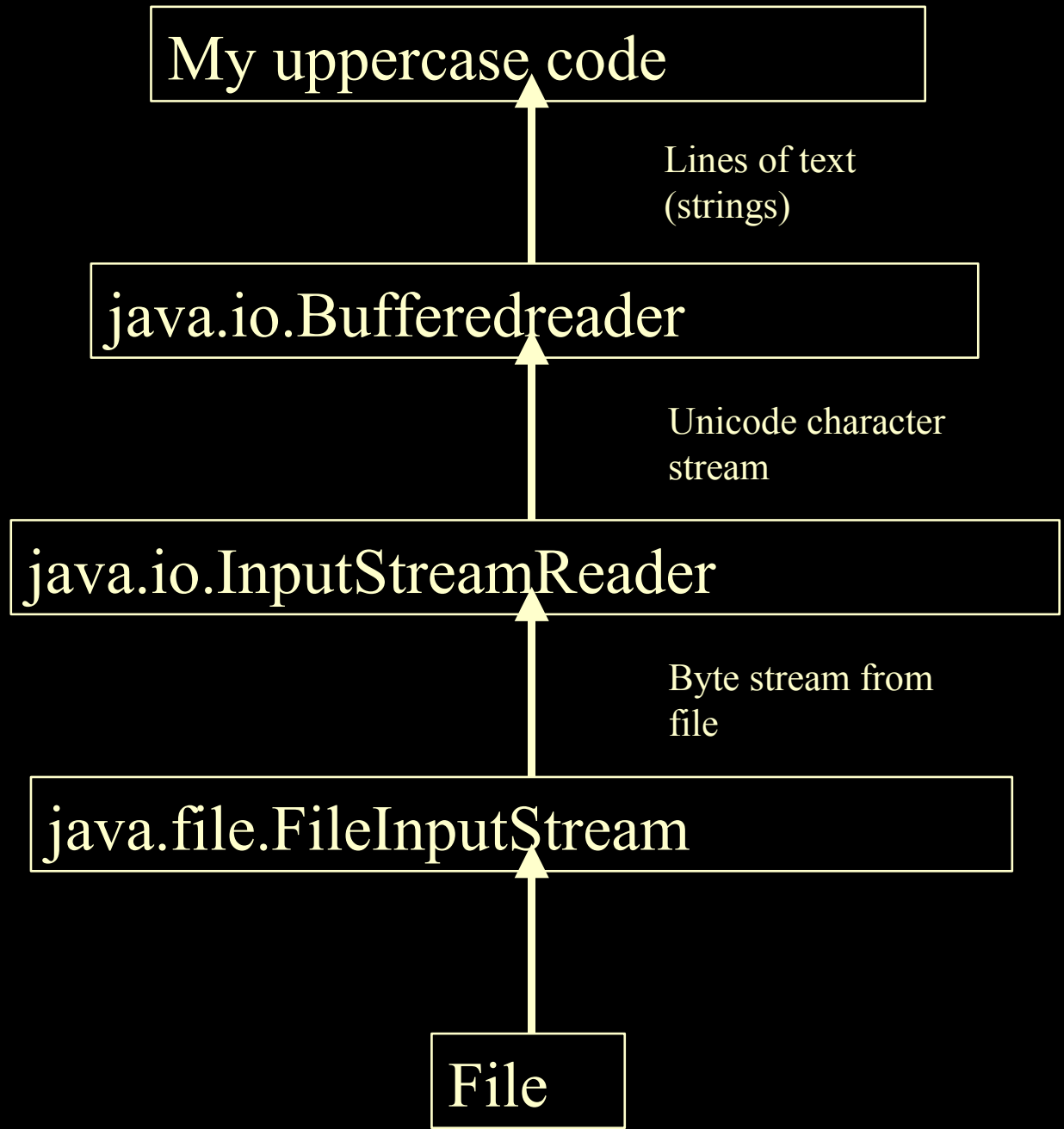
```
mhall@goblin:~>cat blah2.txt
```

```
TEST1
```

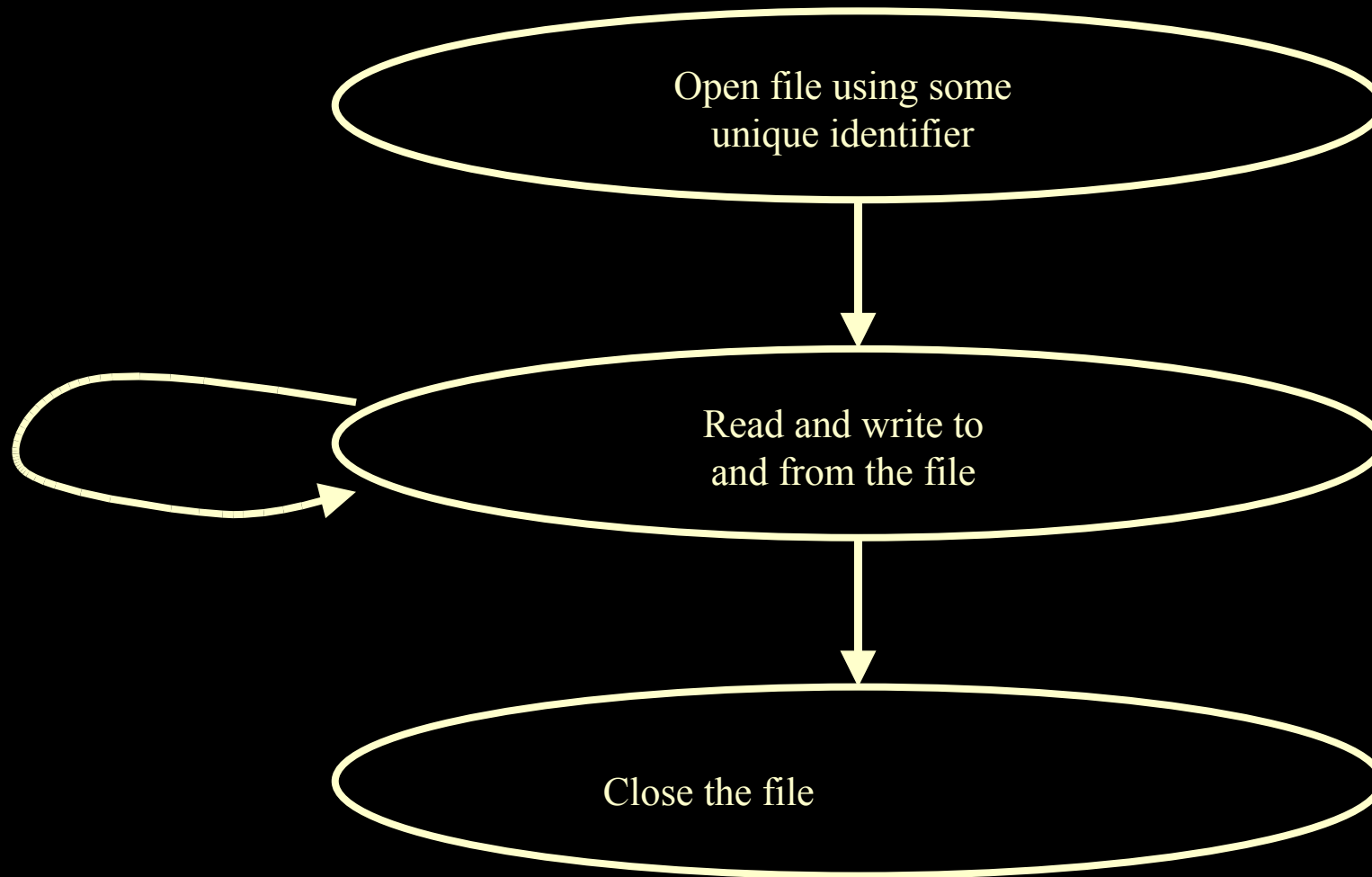
```
TEST2
```

```
BLAHBLAHBLAH
```

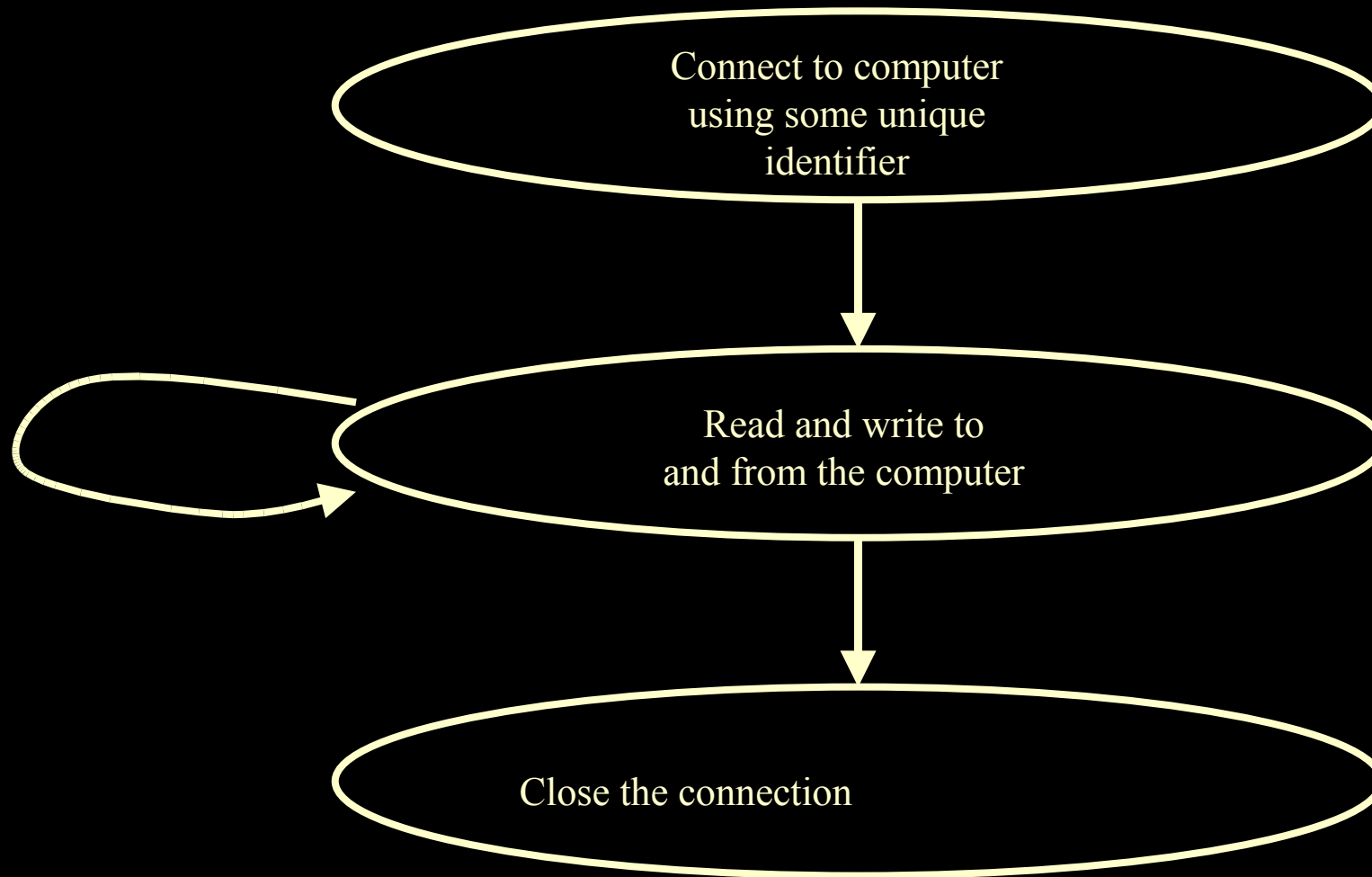
```
mhall@goblin:~>
```



# *The process of using a file*



# *Accessing a computer across a network*



# *Sockets - connecting to other computers*

- When connecting to another computer you use a 'socket'.
  - analogous to a 'file handle' used when accessing files
  - When opening a file you uniquely identify it by its file name. When connecting to a computer you uniquely identify it with its **IP number**

# *Addressing Computers*

- An **IP number** is four numbers (each between 0 and 255) separated by a '.' Eg. Goblin's IP is 130.217.208.41
  - However, because numbers are difficult to remember, the network provides a service that associates names with numbers.
    - Eg goblin.cs.waikato.ac.nz : 130.217.208.41

## *Ports — connecting to programs on other computers over a network*

- Using a unique number we can identify a computer to connect to
  - However, computers have many programs running on them
  - We identify which program to communicate with by using a **port number**
  - Common networking programs (such as telnet, ftp and WWW services) are always on the same port. These ports are called “well known”
  - Telnet is on port 23, FTP on port 21, WWW services are on port 80, etc.

# *File /etc/services*

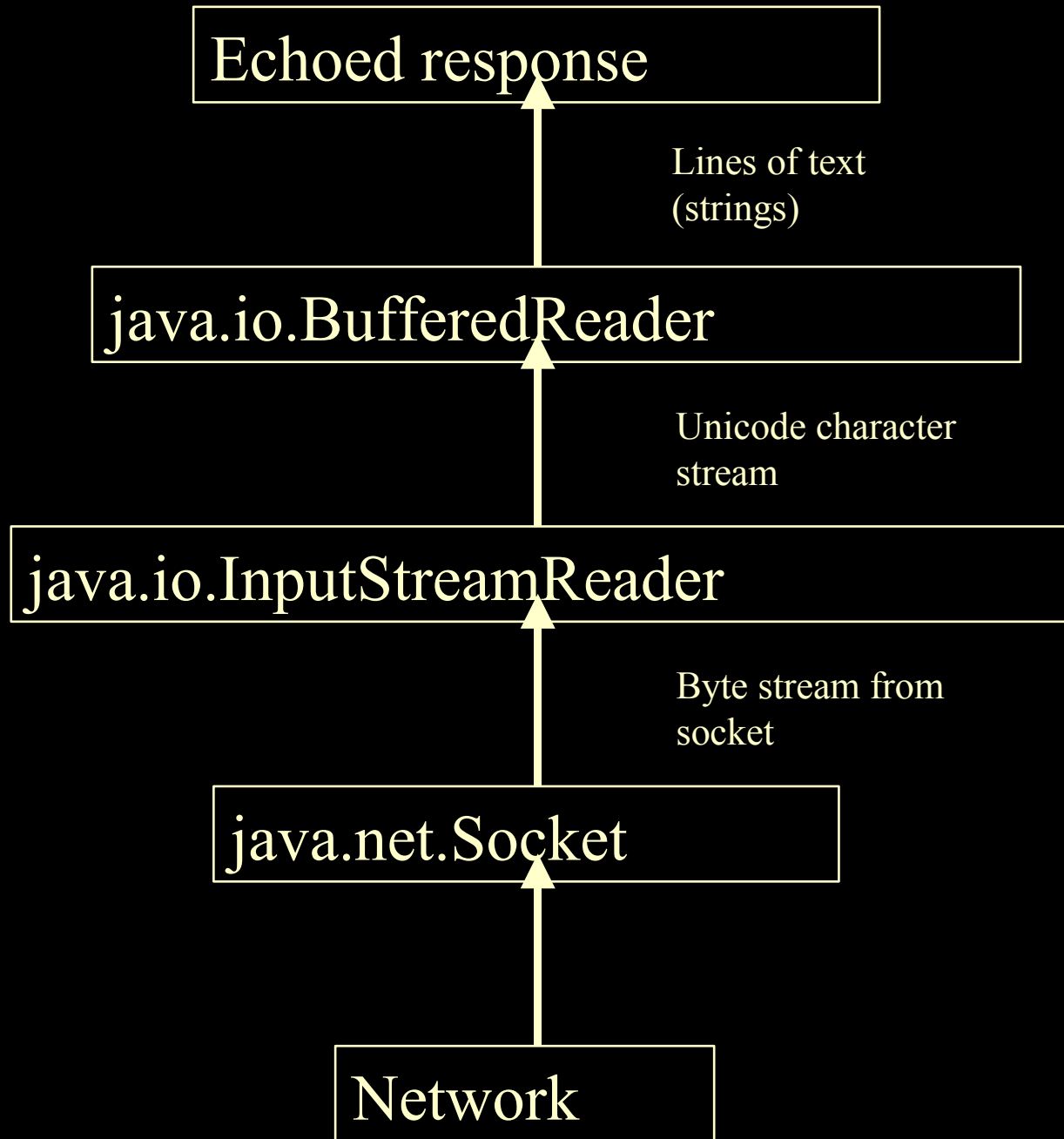
```
tcpmux      1/tcp      # TCP port service multiplexer
echo        7/tcp
echo        7/udp
discard     9/tcp      sink null
discard     9/udp      sink null
sysstat     11/tcp     users
daytime     13/tcp
daytime     13/udp
netstat     15/tcp
gotd        17/tcp     quote
msp         18/tcp     # message send protocol
msp         18/udp     # message send protocol
chargen     19/tcp     ttytst source
chargen     19/udp     ttytst source
ftp-data    20/tcp     # File Transfer [Default Data]
ftp-data    20/udp     # File Transfer [Default Data]
ftp         21/tcp     # File Transfer [Control]
ftp         21/udp     # File Transfer [Control]
ssh         22/tcp     # Secure Shell Login
ssh         22/udp     # Secure Shell Login
telnet      23/tcp
telnet      23/udp
```

# *Sockets in Java*

- Java's networking facilities are provided in the `java.net` package
  - To make a connection to another machine we must first know its IP number—this is done by the `InetAddress` class
    - `InetAddress goblinsIP = InetAddress.getByName("goblin");`
  - Now we can open a socket
    - `Socket mysocket = new Socket(goblinsIP, 23);`
  - This would connect to the telnet port on goblin

# *Sockets in Java*

- The following methods return input and output streams that you can use to read and write to the socket, just like a file
  - `mysocket.getInputStream();`
  - `Mysocket.getOutputStream();`
- Eg:
  - `InputStreamReader in = new  
InputStreamReader(mysocket.getInputStream());`
  - `OutputStreamWriter out = new  
OutputStreamWriter(mysocket.getOutputStream());`



# *Echo Client*

- There is a program (also known as a service) on most networked machines that echoes back what you write to it. This program connects to that echo service (port 7) and writes and reads to/from it

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

public class EchoClient {
    public static void main(String [] args)
        throws IOException {
        Socket echoSocket = null;
        PrintWriter out = null;
        BufferedReader in = null;
```

# *Echo Client...*

```
try {
    InetAddress goblinsIP =
        InetAddress.getByName("goblin");

    echoSocket =
        new Socket(goblinsIP, 7);
    out = new PrintWriter(echoSocket.
        getOutputStream(), true);
    in = new BufferedReader(new
        InputStreamReader(echoSocket.
            getInputStream()));
```

# *Echo Client*

```
} catch (UnknownHostException e) {  
    System.err.println("Don't know about "  
        +"host: goblin.");  
    System.exit(1);  
} catch (IOException e) {  
    System.err.println("Couldn't get I/O "  
        +"for connection to: goblin.");  
    System.exit(1);  
}
```

# *Echo Client...*

```
BufferedReader stdIn = new
    BufferedReader(new
        InputStreamReader(System.in));
String userInput;

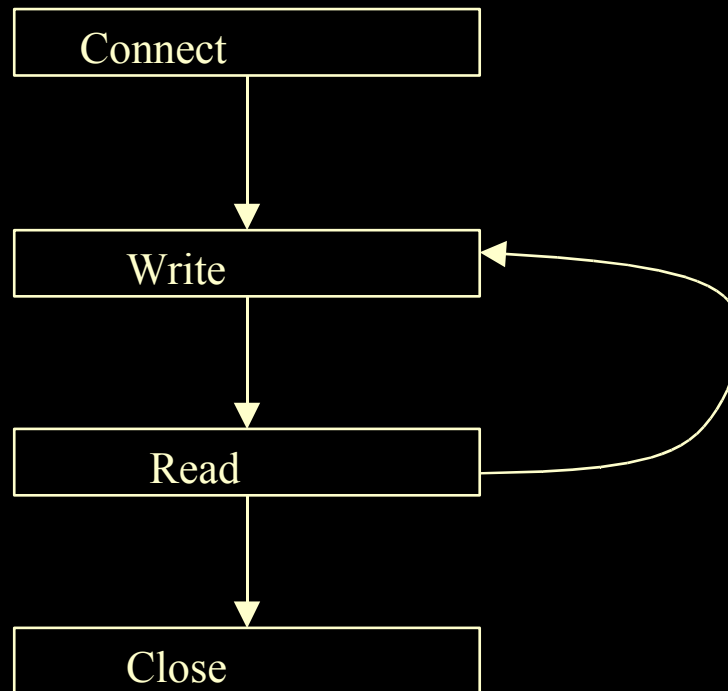
while((userInput = stdIn.readLine())
    != null) {
    out.println(userInput);
    System.out.println("echo: "
        + in.readLine());
}
out.close();
in.close();
stdIn.close();
echoSocket.close();
}
}
```

# *Echo Client in use.*

```
mhall@goblin:~> java EchoClient
Boo!
echo: Boo!
Hello there
echo: Hello there
Quit
echo: Quit
```

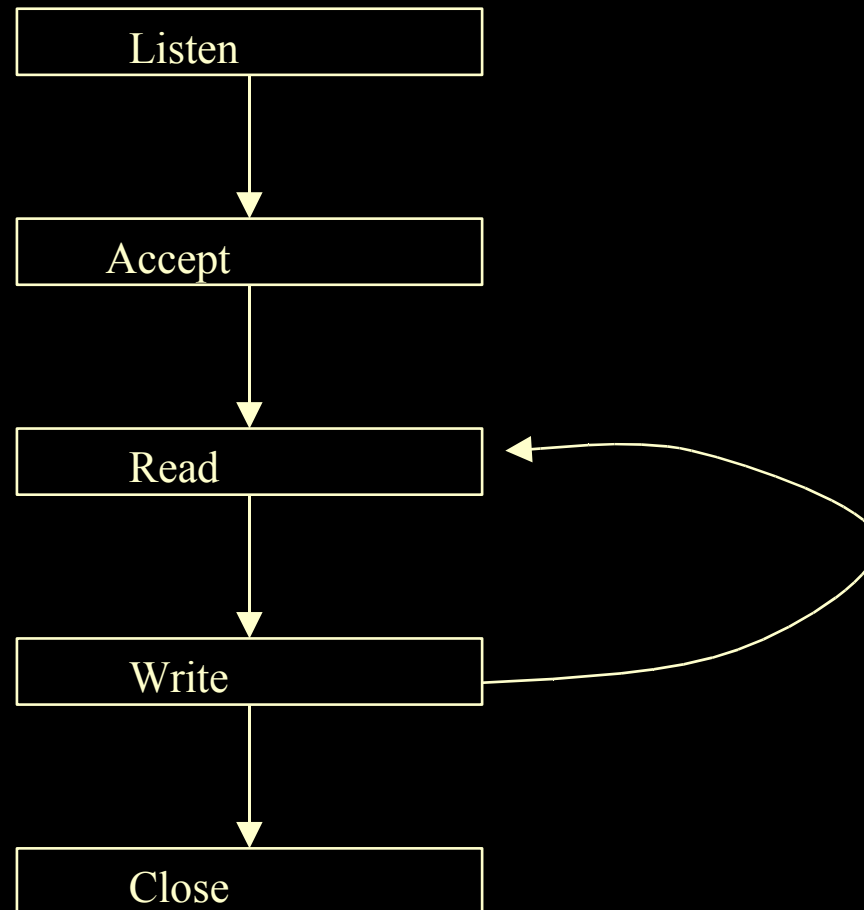
# *Clients and Servers ...*

- So far we have considered a 'client' program that connects through a socket to a service (or server)



# *Clients and Servers ...*

- Consider the actions of the server at the other end



## *Clients and Servers ...*

- Java provides the class **ServerSocket** which listens to connections. Eg:
  - `ServerSocket myserver = new ServerSocket(4420);`
  - This sets up a `ServerSocket` to listen on port 4420
  - Now wait for a connection using the “accept” method. This method returns a reference to a regular socket that can be used to read and write to the connection client
  - `Socket theClient = myserver.accept();`
  - The socket “theClient” can now be used to read and write to the connecting client in the usual way

# *Echo Server...*

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

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

        ServerSocket myserver = null;
        try {
            myserver = new ServerSocket(4444);
        } catch (IOException e) {
            System.out.println("Could not listen "
                +"on port: 4444.");
            System.exit(1);
        }
    }
}
```

# *Echo Server...*

```
Socket theclient = null;
    try {
        theclient = myserver.accept();
    } catch (IOException e) {
        System.err.println("Accept failed.");
        System.exit(1);
    }

    PrintWriter out = new
        PrintWriter(theclient.getOutputStream(),
                    true);
    BufferedReader in = new
        BufferedReader(new
            InputStreamReader(theclient.
                getInputStream()));
```

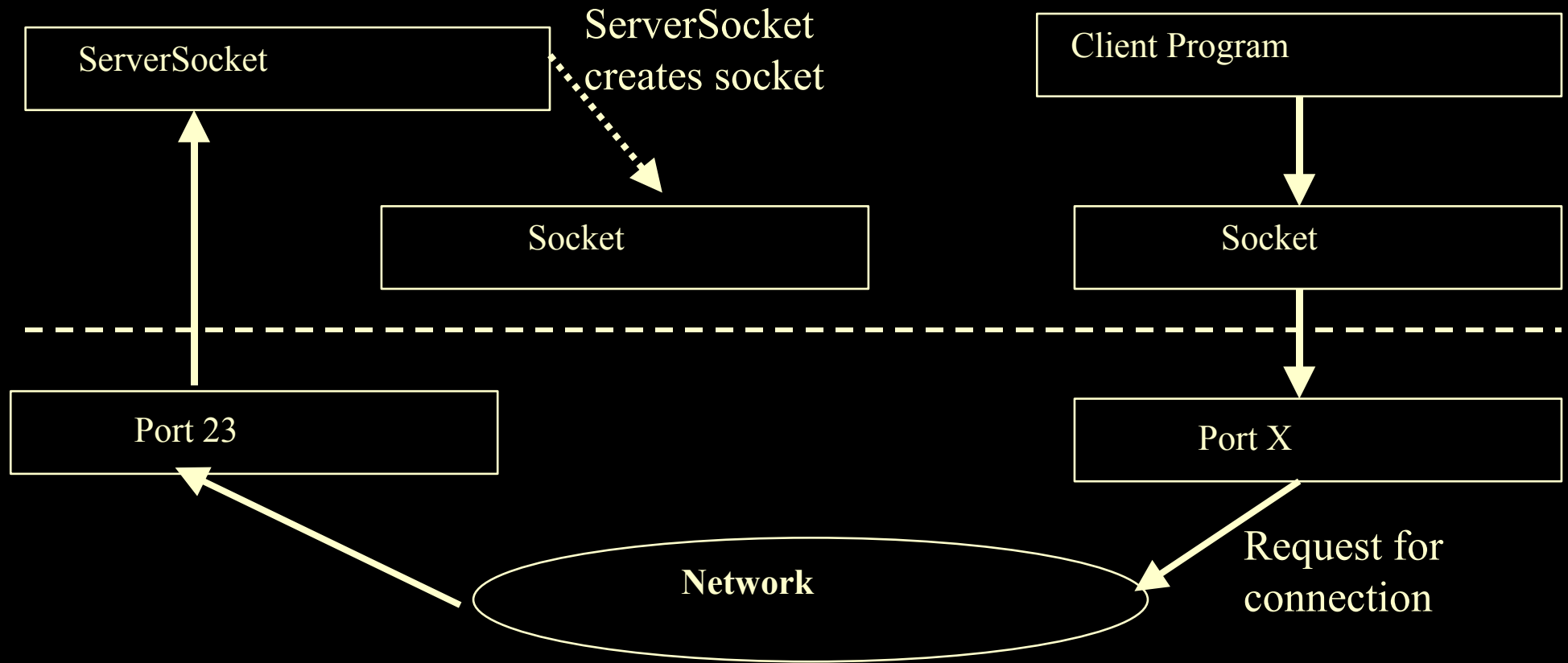
# *Echo Server...*

```
String inputLine, outputLine;

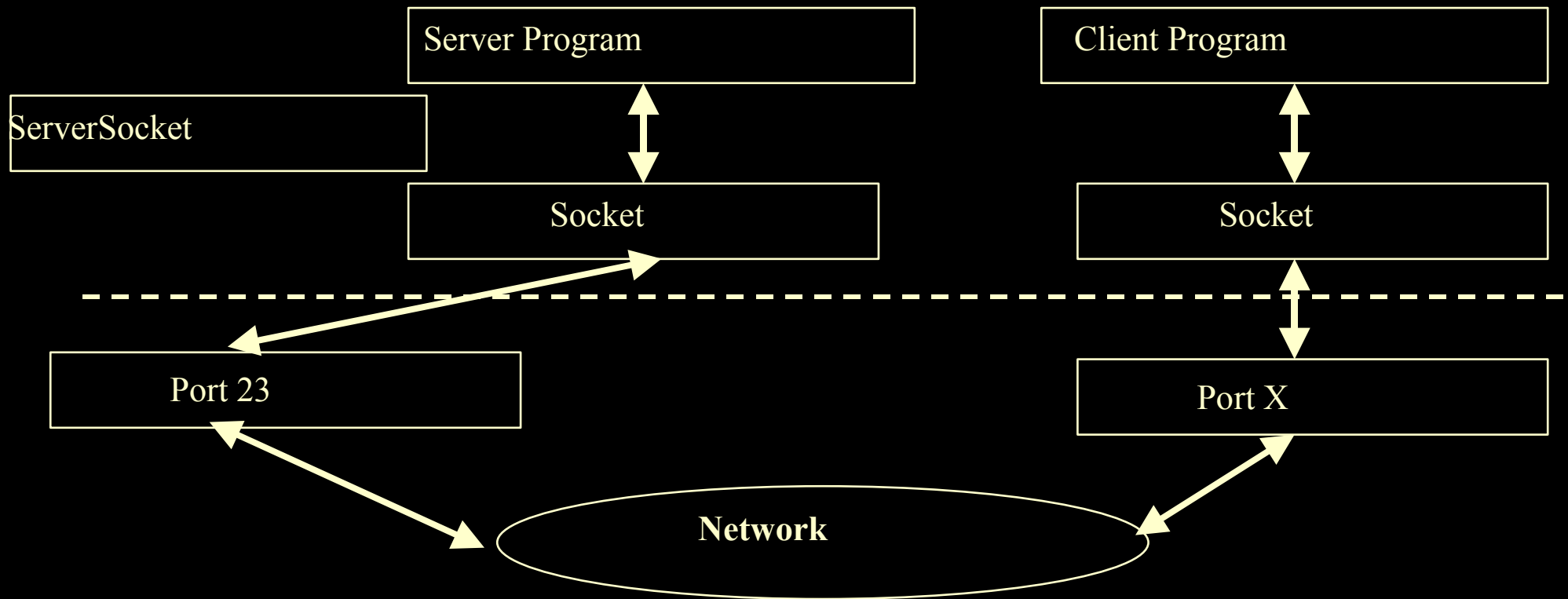
while((inputLine=in.readLine()) != null) {
    outputLine = inputLine + " :-)";
    out.println(outputLine);
    if (inputLine.equals("Bye.")) {
        break;
    }
}

out.close();
in.close();
theclient.close();
myserver.close();
}
```

# *Request for connection*



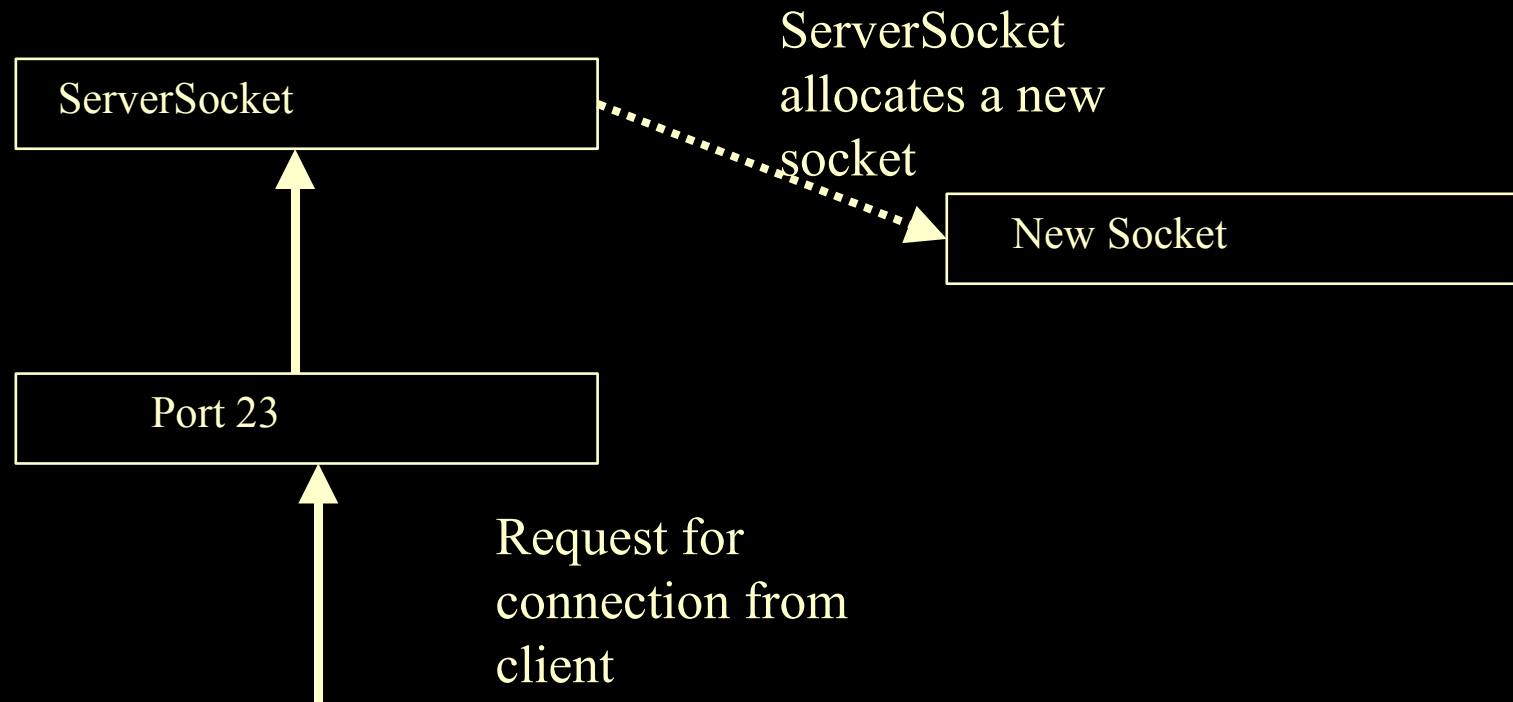
# *Server and Client Communicate*



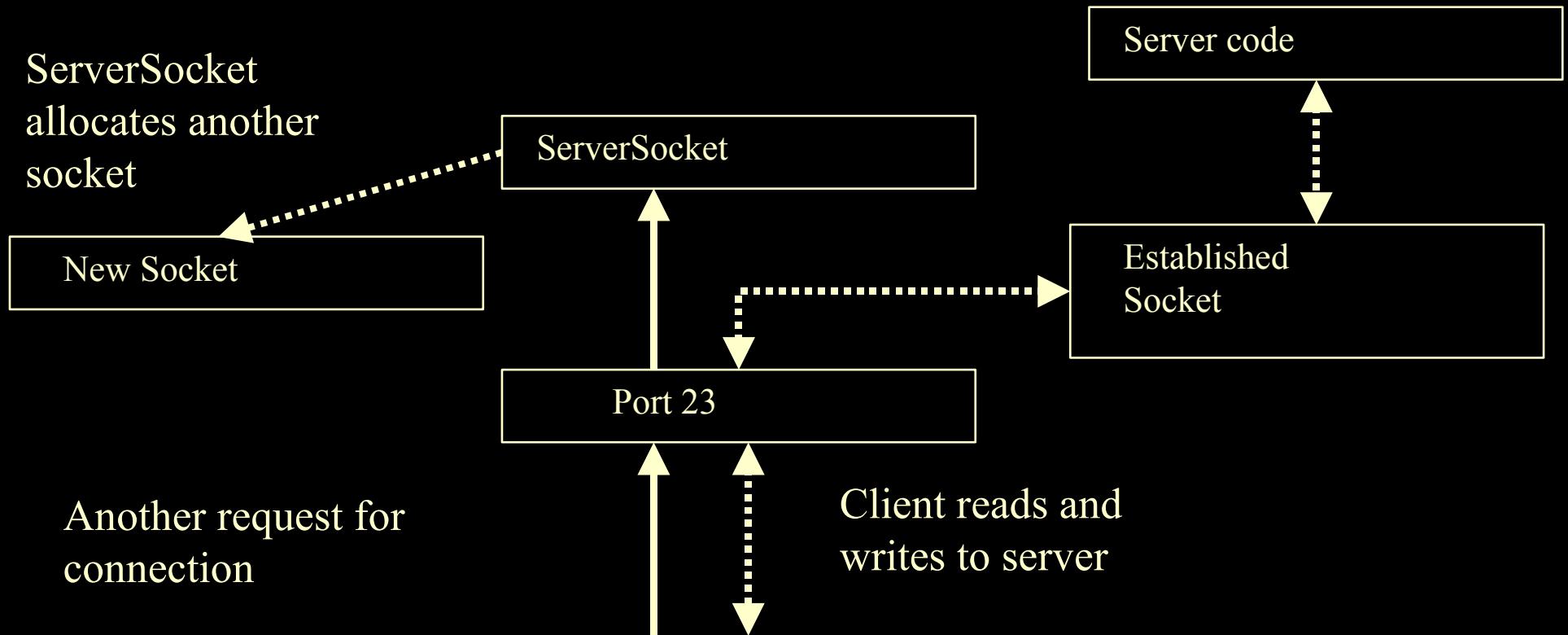
## *Multiple Connections ...*

- Consider the case where multiple clients are connected to the same service. Eg. Multiple telnet connections to goblin
- How do sockets deal with this?

# *Request for Connection*



# *Request for Another Connection*



# *Multiple Connections...*

The port records what clients have connected and diverts interaction to the appropriate socket.

A client is identified by:

- IP number and Port number!

When a socket is created on the client machine it is allocated a local port number as well. This is the port that the server uses to write to the client