Lecture Notes 12
Network Programming in Java

Anand Tripathi

CSci 4131

Internet Programming

Topics

- Basic Socket operation
- Client and Server Socket operations
- URL Connection Class to make a request to a web server

- Refer to Java Documentation
  - See java.net package

Internet Addressing

- Every internet host is assigned an IP address, which has 32 bits in IPv4 (IP version 4) and 128 bits in IPv6.
- IPv4 is commonly used; IPv6 is the new generation protocol.
- IP addresses are specified using the "dotted decimal" notation. Each byte in the 32 bit address is represented by a decimal number.
- For example: 128.101.228.45
- IP -- Internet Protocol routes IP datagrams (packets) from one host to another in the internet. It forms the foundation of all other communication protocols, such as TCP.
  - It is a "network layer" protocol.

Communication Protocol Layers

- **The following five-layer model is adopted in the textbook for presentation and discussions.**
- **It is based on TCP/IP and the OSI model.**

Application (e.g. HTTP)
Transport (TCP/UDP)
Network **(Internet Protocol)**
Data Link
Physical

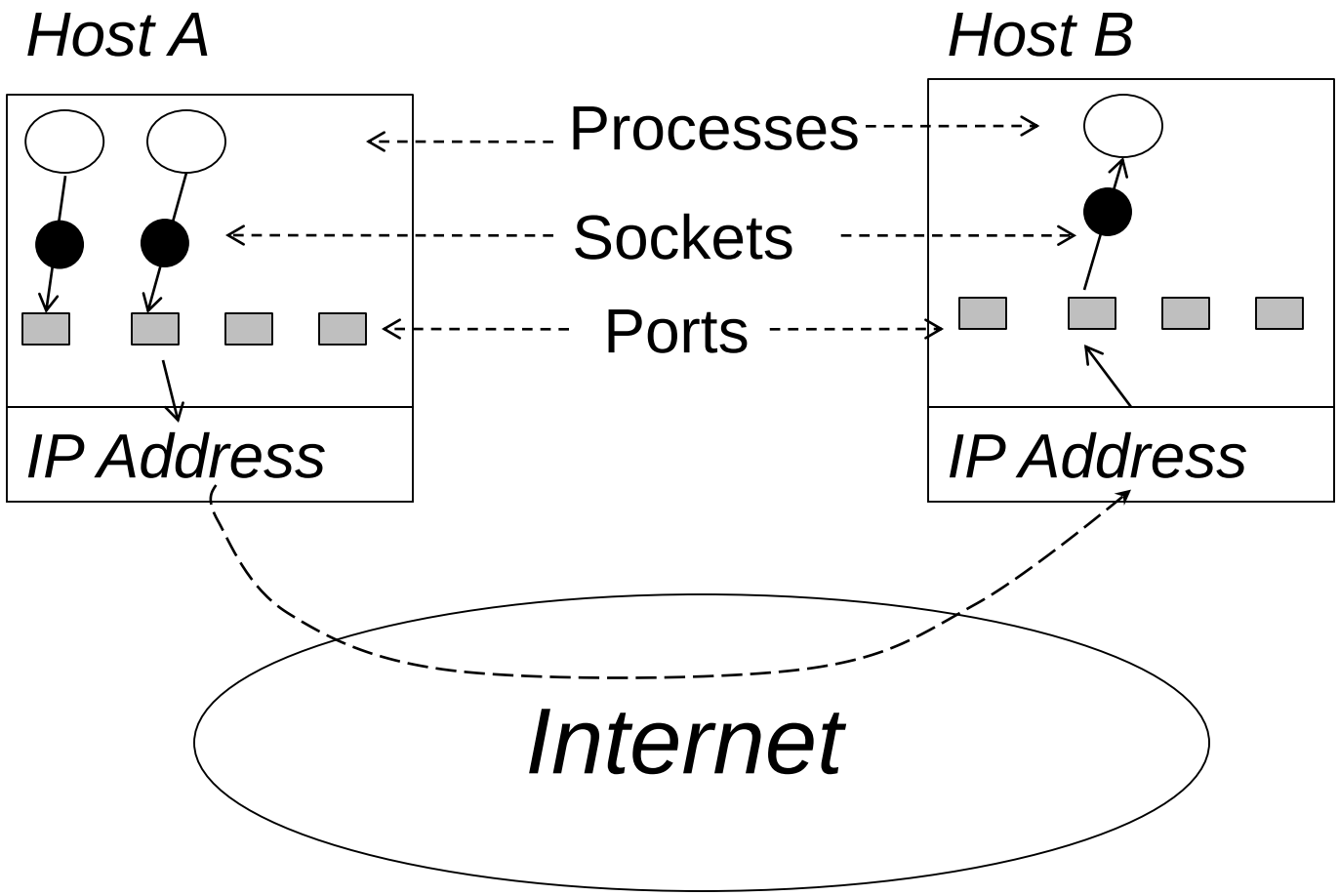
Transport Layer Protocols Transmission Control Protocol (TCP)

- TCP (Transmission Control Protocol) supports:
  - Connection-oriented protocol
  - Reliable bidirectional communication channel between two processes at any hosts in the Internet.
    - Pipes in two directions
    - No loss
    - In-order delivery of the bytes in the message stream
  - A TCP connection supports communication of any amount of data in either direction.
  - It's a connection-oriented protocol: this involves three IP packets to be exchanged to establish a connection

Sockets and Ports

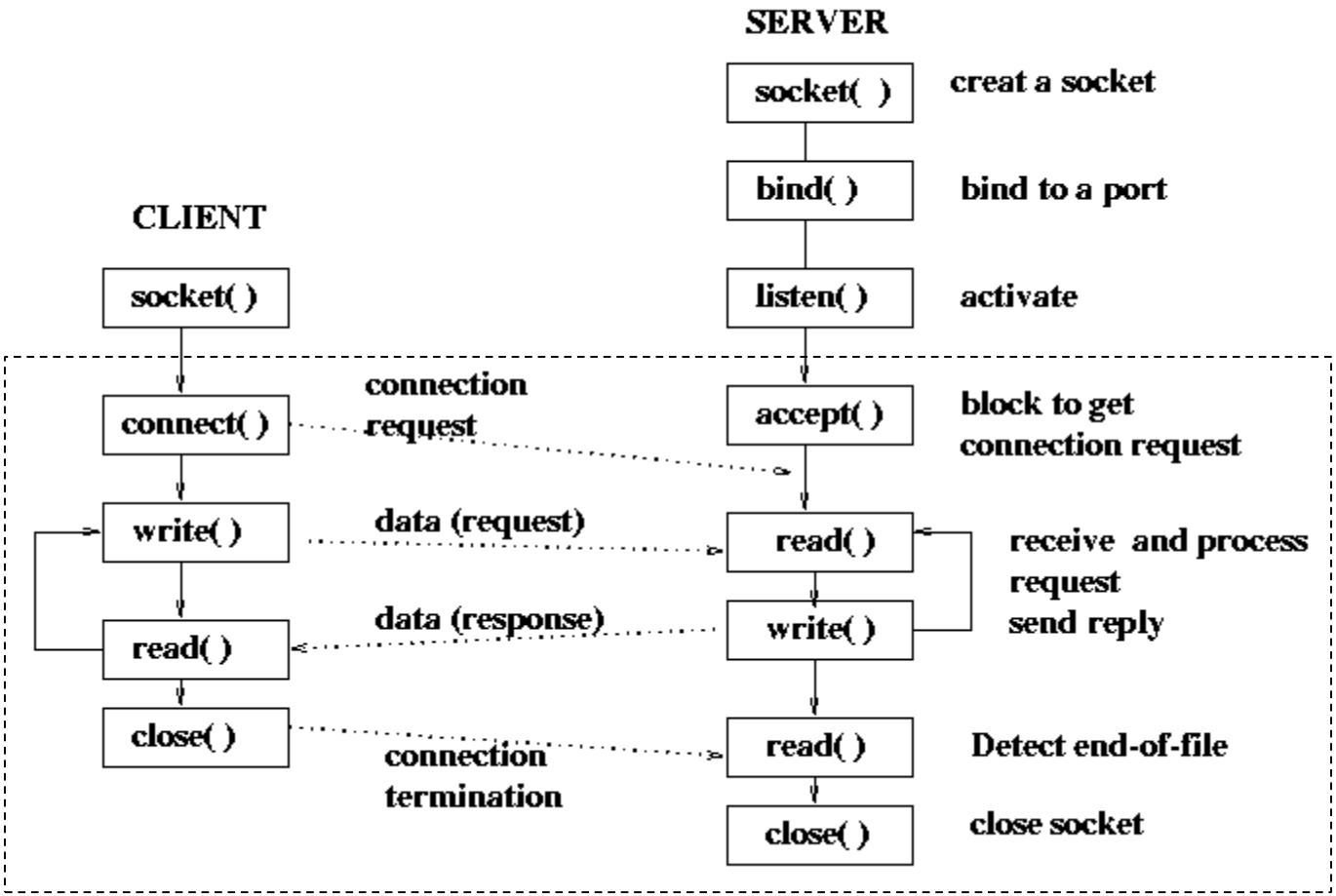
- A socket represents kernel-supported entity using which a process communicates with another process in the network.
- A socket is similar to a file descriptor in UNIX. In fact, it is referenced through an an entry in the UNIX file descriptor table of the process.
- A port is a protocol specific entity which acts as a communication end-point for messages routed to a host.
- Ports are maintained by the kernel. It is identified by a 16 bit number.

IP Address and Ports



Sockets and Ports

- On a host, different protocols (such as TCP and UDP) can have ports with the same number, but such ports represent different entities.
- A process needs to "bind" one of its sockets to a port in order to communicate through that socket.
- An <IP address, Port number> tuple uniquely identifies a communication point in the network.


SERVER
socket( )
creat a socket
bind( )
bind to a port
CLIENT
listen( )
activate
socket( )
connection
request
connect( )
accept( )
block to get
connection request
write( )
data (request)
read( )
receive and process
request
send reply
data (response)
write( )
read( )
close( )
connection
termination
read( )
Detect end-of-file
close( )
close socket

Java Classes for Network Programming

- The following classes are useful for network programming and web programming in Java.
- InetAddress
- Socket
- ServerSocket
- URLConnection
- HttpURLConnection
- URLEncoder
- URLDecoder
- HttpCookie

Example Programs

We will see several exam programs to show

- A simple client-server application
- How to write code for an HTTP client program.
- How to write a small web server or a proxy server.

Java Classes for Network Programming

- The following classes are useful for network programming in Java.

- InetAddress
  - This class represents an Internet Protocol (IP) address.
  - Several static methods of this class are used to create new a InetAddress object for some host

Java Classes for Network Programming

- Socket
  - This class is used to to create a TCP socket. This socket is created to establish a connection to some network address/port.
- ServerSocket
  - This class implements server sockets. A server socket waits for requests to come in over the network. It performs some operation based on that request, and then possibly returns a result to the requester.
- DatagramSocket
  - This class is used for connection-less communication.
- DatagramPacket

InetAddress Class

- static getByName(String)
  - Returns an InetAddress object for the IP address of a host, given the host's name.
- static getLocalHost()
  - Returns an InetAddress object for the local host.
- static getAllByName(String)
  - Returns an array of InetAddress objects, for all the IP addresses of a host, given the host's name.

- getAddress()
  - Returns the raw IP address of this InetAddress object.
- getHostAddress()
  - Returns the IP address string "%d.%d.%d.%d"
- getHostName()
  - Returns the fully qualified host name for this address.
- hashCode()
  - Returns a hashcode for this IP address.
- isMulticastAddress()
  - Utility routine to check if the InetAddress is an IP multicast address.
- toString()
  - Converts this IP address to a String

An Example of Using InetAddress Class -- NameLookup.java

- This program will lookup the IP address given the DNS name of a host, or print the DNS name if you type the IP address.
- When this program is started it will first print the local host name.

- See the code on course webpage, under

  Examples → TCP-HTTP-Examples

  NameLookup.java

Example of Using InetAddress Class

```
import java.net.*;
mport java.io.*;
public class NameLookup {
    public static void main (String args[]) {
        printLocalAddress ();
        DataInputStream in = new DataInputStream (System.in);
        try {
            String name;
            do { System.out.print ("Enter a hostname or IP address: ");
                System.out.flush (); name = in.readLine ();
                if (name != null)
                    LookupRemoteAddress (name);
            } while (name != null);
            System.out.println ("exit");
        }  catch (IOException ex) {
                System.out.println ("Input error:"); ex.printStackTrace ();
        }
}
```

```
static void printLocalAddress () {
    try {
      InetAddress myself = InetAddress.getLocalHost ();
      System.out.println ("Local machine name : " +
                                  myself.getHostName () );
      System.out.println ("Local Machine IP address : " +
                        myself.getHostAddress ( )  );
     } catch (UnknownHostException ex) {
         System.out.println (“Failed to find myself:”);
         ex.printStackTrace ();
     }
}
```

LookupRemoteAddress

```
static void LookupRemoteAddress (String name) {
    try {
       System.out.println ("Looking up " + name + "...");
       InetAddress addrs = InetAddress.getByName (name);
       System.out.println ("Remote Host name : " +
                          addrs.getHostName () );
        System.out.println ("Host IP : " +
                          addrs.getHostAddress () ) );
    } catch (UnknownHostException ex) {
          System.out.println (“Failed to lookup “ + name);
    }
}
```

Socket Class

Constructors:

- public Socket(String host, int port)
  throws UnknownHostException, IOException
  - Creates a stream socket and connects it to the specified port number on the named host.

- public Socket(InetAddress address, int port) throws IOException
  - Creates a stream socket and connects it to the specified port number at the specified IP address.

Socket Class

Constructors:

- public Socket(String host, int port,
  InetAddress localAddr,int localPort)
  throws IOException
  - Creates a socket and connects it to the:
    - Specified remote host on the specified remote port.
    - The Socket will also bind() to the local address and port supplied.
- public Socket(InetAddress address, int port,
  InetAddress localAddr, int localPort)
  throws IOException
  - Creates a socket and connects it to the:
    - Specified remote address on the specified remote port.
    - The Socket will also bind() to the local address and port supplied.

Socket Methods

- accept()
  - Wait for a connection request to arrive
  - It returns a new socket to handle the connection
- close()
  - Closes this socket.
- getInetAddress()
  - Returns the address to which the socket is connected.
- getPort()
  - Returns the remote port to which this socket is connected.
- getLocalAddress()
  - Gets the local address to which the socket is bound.
- getLocalPort()
  - Returns the local port to which this socket is bound.

Socket Methods

A socket has two streams associated with it:
InputStream and OutputStream.

- **getOutputStream()**
  - Returns an output stream for this socket.
- **getInputStream()**
  - Returns an input stream for this socket.
- **setSoTimeout(int timeout)**
  - Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.
    - A read() call on the InputStream associated with this Socket will block for only this amount of time. If the timeout expires, a java.io.InterruptedIOException is raised, though the Socket is still valid.
    - The option must be enabled prior to entering the blocking operation to have effect.
    - The timeout must be > 0. A timeout of zero is interpreted as an infinite timeout.

URL Class

- This class represents a URL.
- Allows parsing of a URL, also supports access to the resource using GET or PUT.
- Constructors:
- public URL(String spec) throws MalformedURLException
  - Creates a URL object from the String representation.
- public URL(URL context, String spec) throws MalformedURLException
  - Creates a URL by parsing the specification spec within a specified context.
  - If the context argument is not null and the spec argument is a partial URL specification, then any of the missing components are inherited from the context argument.

URL Class Methods

- getFile()
  - Returns the file name of this URL.
- getHost()
  - Returns the host name of this URL, if applicable.
- getPort()
  - Returns the port number of this URL.
- getProtocol()
  - Returns the protocol name this URL.
- getRef()
  - Returns the anchor (also known as the "reference") of this URL.

URL Class Methods

- openConnection()
  - Returns a URLConnection object that represents a connection to the remote object referred to by the URL.
- openStream()
  - Opens a connection to this URL and returns an InputStream for reading from that connection.
- set(String, String, int, String, String)
  - Sets the fields of the URL.
- toString()
  - Constructs a string representation of this URL.

URL RELATED CLASSES

- URL
  - This class represents a URL. Allows parsing of a URL, also supports access to the resource using GET or PUT.
- URLConnection (this is an abstract class)
- HttpURLConnection
  - This class is specific to HTTP protocol.
  - Defines methods to access a resource using its URL.
- URLEncoder
  - The class contains a utility method for converting a String into a MIME format called "x-www-form-urlencoded" format.
    - To convert a String, each character is examined in turn:
    - The ASCII characters 'a' through 'z', 'A' through 'Z', and '0' through '9' remain the same. The space character ' ' is converted into a plus sign '+'. All other characters are converted into the 3-character string "%xy", where xy is the two-digit hexadecimal representation of the lower 8-bits of the character. It has one static method -- "encode".

URL Class Method getContent

public final Object getContent() throws IOException

Returns the contents of this URL.

This method is a shorthand for:

openConnection().getContent()

public URLConnection openConnection() throws IOException

- Returns a URLConnection object that represents a connection to the remote object referred to by the URL.
- If there is not already an open connection, the connection is opened by calling the openConnection method of the protocol handler for this URL.

URLConnection Class

- The abstract class URLConnection is the superclass of all classes that represent a communication link between the application and a URL.
- Instances of this class can be used both to read from and to write to the resource referenced by the URL.
- In general, creating a connection to a URL is a multistep process:

- openConnection()
  - Manipulate parameters that affect the connection to the remote resource.
- connect()
  - Interact with the resource; query header fields and contents.

Methods of URLConnection Class

- The following methods are used to access the header fields and the contents after the connection is made to the remote object:
  - getContent
  - getHeaderField
  - getInputStream
  - getOutputStream
- Certain header fields can be accessed using the following methods:
  - getContentEncoding
  - getContentLength
  - getContentType
  - getDate
  - getExpiration
  - getLastModified

The getContentType method is used by the getContent method to determine the type of the remote object; subclasses may find it convenient to override the getContentType method.

URLConnection Example

```
import java.net.*;
import java.io.*;
import java.util.*;
class URLConnectionReader {
    public static void main(String[] args) throws Exception {
    URL targetURL;
    BufferedReader keyboard =  new BufferedReader(new
      InputStreamReader(System.in));

    while (true) {
      String textURL;
      System.out.print ("Enter a URL: ");
      System.out.flush ();
      if ((textURL = keyboard.readLine ()) == null)
        break;
    targetURL  = new URL( textURL );
      URLConnection connection = targetURL.openConnection();
```

URLConnection Example

```
 String date = connection.getHeaderField(“Date");
 String contentType  = connection.getHeaderField("Content-Type");

System.out.println( "The date header is: “ + date );
      System.out.println( “This document type is: “ + contentType );
   }
  }
}
```

Java Program for an HTPP Client

See the “examples” link on the course webpage for the Java programs for HTTP Client and HTTP Server

- This program will prompt the user to input an URL. It will then use "GET" method of the HTTP protocol to retrieve the document, and then print it to the standard output.

```
import java.net.*;
import java.io.*;
public class HTTPget  {
  protected String host, file;
  protected String urlString;
  protected int port;
  protected DataInputStream in;
  protected DataOutputStream out;

  public HTTPget (String textURL) throws IOException {
    urlString = textURL;
    Socket socket = null;
    parseURL (textURL);
    socket = connectToServer();
    try {
      MakeGetRequest ();
    } finally {
      socket.close ();
    }
  }
```

```
protected void parseURL (String textURL) throws MalformedURLException {
    URL url = new URL (textURL);
    host = url.getHost ();
    port = url.getPort ();
    if (port == -1)
      port = 80;
    file = url.getFile ();
  }

  protected Socket connectToServer () throws IOException {
    System.out.println ("Connecting to " + host + ":" + port + "..");
    Socket socket = new Socket (host, port);
    System.out.println ("Connected.");

    OutputStream rawOut = socket.getOutputStream ();
    InputStream rawIn = socket.getInputStream ();
    BufferedOutputStream buffOut = new BufferedOutputStream (rawOut);
    out = new DataOutputStream (buffOut);
    in = new DataInputStream (rawIn);
    return socket;
  }
```

```
protected void MakeGetRequest () throws IOException {

   System.out.println ("Sending request..");
   out.writeBytes ("GET " + urlString + " HTTP/1.1\n");
   out.writeBytes ("HOST: " + host + "\n");
   out.writeBytes ("CONNECTION: close\n\n");
   out.flush ();

   System.out.println ("Waiting for response..");
   String input;
   while ((input = in.readLine ()) != null)
     System.out.println (input);
 }
```

```
public static void main (String args[]) throws IOException {
   DataInputStream keyboard = new DataInputStream (System.in);
   while (true) {
     String textURL;
     System.out.print ("Enter a URL: ");
     System.out.flush ();
     if ((textURL = keyboard.readLine ()) == null)
       break;

     try {
       new HTTPget (textURL);
     } catch (IOException ex) {
       ex.printStackTrace ();
       continue;
     }

     System.out.println ("- OK -");
   }
   System.out.println ("exit");
  }
}
```

URL Reader Example

This program reads a URL document resource.

```
public class URLReader {
    public static void main(String[] args) throws Exception {
     URL deptHome = new URL("http://www.cs.umn.edu/");
     DataInputStream in = new DataInputStream(
                                    deptHome.openStream());

     String inputLine;
     while ((inputLine = in.readLine()) != null)
         System.out.println(inputLine);

     in.close();
    }
}
```

URL Connection Example

```
public class URLConnectionReader {
    public static void main(String[] args) throws Exception {
     URL deptHome = new URL("http://www.cs.umn.edu/");
     URLConnection csHomeConnection = deptHome.openConnection();
     DataInputStream in = new DataInputStream(
                                    csHomeConnection.getInputStream());
     String inputLine;

     while ((inputLine = in.readLine()) != null)
         System.out.println(inputLine);

     in.close();
    }
}
```

Example of a Multithreaded Server

- This server will echo back whatever data is sent by the client.
- For each client request a new thread will be created to handle that connection.
- The static main method of this class will listen on a port (given as command line argument) to wait to a connection request.
- Accepting a connection will create a new socket. This socket will be passed to a newly created thread to serve that request.
- This thread (which executes the run method) simply writes data back to the socket and also prints it on the server's standard output.

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

public class EchoServer extends Thread {
  protected Socket s;

  EchoServer (Socket s) {
    System.out.println ("New client.");
    this.s = s;
  }
```

```
public void run () {
    try {
        InputStream istream = s.getInputStream ();
        OutputStream ostream = s.getOutputStream ();
         new PrintStream(ostream).println ("Welcome to the multithreaded
              echo server.");
          byte buffer[] = new byte[16];
          int read;
          while ((read = istream.read (buffer)) >= 0) {
               ostream.write (buffer, 0, read);
               System.out.write (buffer, 0, read);
               System.out.flush();
            }
          System.out.println ("Client exit.");
      } catch (IOException ex) {
      ex.printStackTrace ();
    } finally {
      try {     s.close ();
      } catch (IOException ex) { ex.printStackTrace ();}
} }
```

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

    if (args.length != 1)
        throw new RuntimeException ("Syntax: EchoServer port-number");

    System.out.println ("Starting on port " + args[0]);
    ServerSocket server = new ServerSocket (Integer.parseInt (args[0]));

    while (true) {
      System.out.println ("Waiting for a client request");
      Socket client = server.accept ();
      System.out.println ("Received request from " + client.getInetAddress
     ());
      EchoServer c = new EchoServer (client);
      c.start ();
    }
  }
}
```

```
import java.net.*;
import java.io.*;
public class EchoClient   {
  protected String host, file;
  protected int port;
  protected DataInputStream in;
  protected DataOutputStream out;

  public static void main (String args[]) throws IOException {
    InetAddress  server  = null;
    Socket       sock = null;
    String host = args[0];
    int  port = Integer.parseInt( args[1] );

    if ( args.length != 2 ) {
       throw new RuntimeException( "hostname and port number as
     arguments" );
    }
    System.out.println ("Connecting to " + host + ":" + port + "..");
    Socket socket = new Socket (host, port);
    System.out.println ("Connected.");
```

```
    OutputStream rawOut = socket.getOutputStream ();
      InputStream rawIn = socket.getInputStream ();
      BufferedReader  buffreader = new BufferedReader( new
       InputStreamReader(rawIn) );
      PrintWriter printer = new PrintWriter(new OutputStreamWriter(rawOut));
      BufferedReader keyboard = new BufferedReader(new
                                          InputStreamReader(System.in));

      String line ;
      while ( ( line = keyboard.readLine() ) != null ) {
            printer.println( line );
            printer.flush();
      }

      while ( ( line = buffreader.readLine() ) != null ) {
            System.out.println( line );
      }
    }
  }
```