

Day 05

NETWORK PROGRAMMING IN JAVA 1

FUNDAMENTALS OF TELECOMMUNICATIONS LAB

Dr. Huy Nguyen

- Basic concepts of network
- Network programming

- Basic concepts of network
- Network programming

NETWORK PROGRAMMING IN JAVA 1

BASIC CONCEPTS OF NETWORK

Chapter Goals

- Concepts of Internet, Internet protocol, Internet address and domain name
- Definition of data transmission, data packets and transmission protocols
- Comparison between TCP and UDP
- Definition of HTTP and browser
- Usage of Telnet and basic network commands

The Internet Protocol

- Internet
 - A worldwide collection of networks, routing equipment, and computers
 - Uses a common set of protocols to define how the parties will interact with each other
- IP: Internet Protocol
 - Developed to enable different local area networks to communicate with each other
 - Has become the basis for connecting computers around the world together over the Internet

Data Transmission

- Consists of sending/receiving streams of zeros and ones along the network connection
- Two Types of Information
 - Application data
 - The information one computer wants to send to another
 - Network protocol data
 - Describes how to reach the intended computer
 - Describes how to check for errors in the transmission

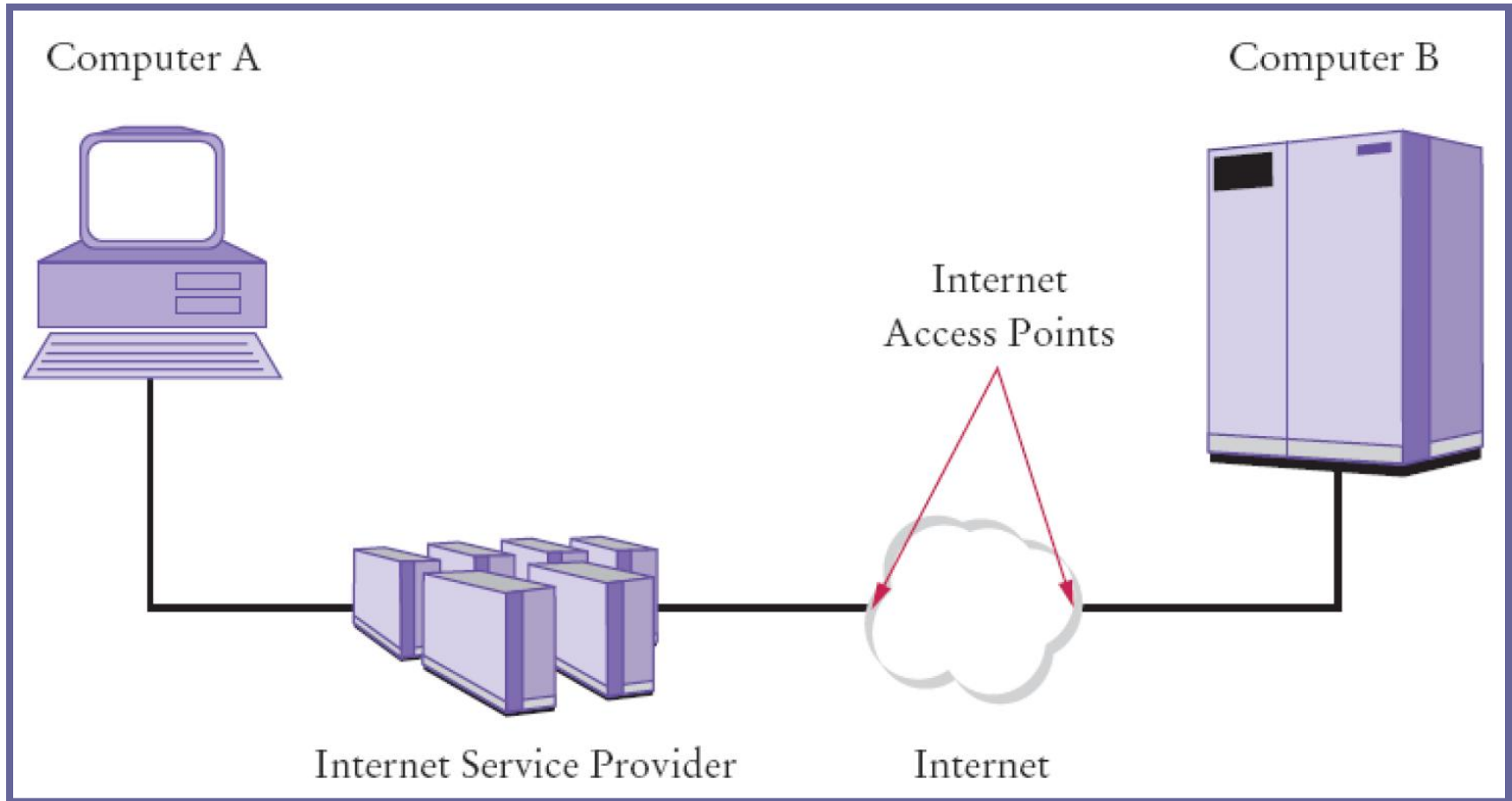
Sending Data from *A* to *B* across the Internet

- *A* is your home computer
- It is connected by phone lines to an Internet Service Provider (ISP)
- The ISP is connected to an Internet Access Point
- *B* is on an local area network at XYZ Computers

Sending Data from *A* to *B* across the Internet

- XYZ has its own Internet Access Point
- The Internet Access Points are connected by a complex collection of pathways (the Internet)
- Over these pathways a message sent from one access point can eventually reach any access point

Two Computers Communicating Across the Internet



Destination Address

- Destination Address
- Data must be marked with a destination address
- In IP, addresses are denoted by a sequence of four numbers
 - Each is one byte (a number between 0 and 255)
 - For example 130.65.86.66
 - To be able to accommodate more devices, IP addresses will be extended to sixteen bytes

Destination Address

- To send data to *B*, *A* needs to know *B*'s Internet address
 - *A* includes that address in the protocol portion when sending the data

Domain Naming Service

- In addition to an IP address, computers can have an easy-to-remember *domain name*
 - For example, `java.sun.com`
- *Domain Naming Service* (DNS): translates from domain name to IP address
- When *A* wants to request data from a domain name:
 - It asks the DNS for the numeric Internet Address
 - It includes the numeric address with the request for data

- IP breaks large chunks of data up into more manageable *packets*
- Each packet is delivered separately
- Each packet in a larger transmission may be sent by a different route
- Packets are numbered
- The recipient reassembles the data

Transmission Control Protocol

- Internet Protocol (IP) does not notify the sender if data is lost or garbled
- This is the job of a higher level protocol *Transmission Control Protocol* (TCP)
- The most commonly used Internet services use TCP with IP (TCP/IP)

- Attempt to deliver the data
- Try again if there are failures
- Notify the sender whether or not the attempt was successful

User Datagram Protocol

- A program never needs to worry about receiving data that is out of order or incorrect if it uses TCP. However, this reliability comes at a price. That price is speed. Establishing and tearing down TCP connections can take a fair amount of time, particularly for protocols such as HTTP, which tend to require many short transmissions.
- The *User Datagram Protocol* (UDP) is an alternative protocol for sending data over IP that is very quick, but not reliable. That is, when you send UDP data, you have no way of knowing whether it arrived, much less whether different pieces of data arrived in the order in which you sent them.

- Attempt to deliver the data
- Do not try again if there are failures and do not notify the sender whether or not the attempt was successful
- UDP is used in the condition of the application is responsible for reliability

Port Numbers

- One computer can offer multiple services over the Internet
 - For example, both a web server program and an email server program
- When data are sent to that computer, they need to indicate which program is to receive the data

- IP uses *port numbers* for this
 - A port number is an integer between 0 and 65,535
 - The sending program must know the port number of the receiving program
 - This number is included in the transmitted data

Contents of TCP or UDP Packet

- The Internet address of the recipient
- The port number of the recipient
- Internet address of the sender
- The port number of the sender

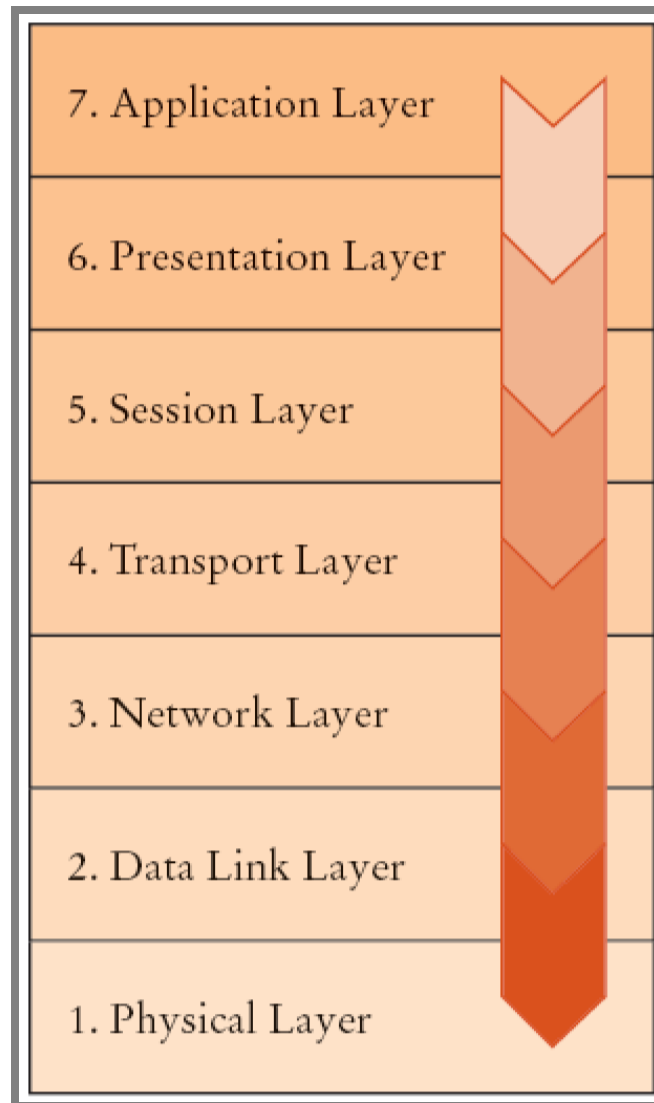
What is the difference between an IP address and a domain name?

Answer: An IP address is a numerical address, consisting of four or sixteen bytes. A domain name is an alphanumeric string that is associated with an IP address.

Why do some streaming media services not use TCP?

Answer: TCP is reliable but somewhat slow. When sending sounds or images in real time, it is acceptable if a small amount of the data is lost. But there is no point in transmitting data that is late.

The OSI Reference Model



Application Level Protocols

- TCP/IP mechanism establishes an Internet connection between two ports on two computers
- Each Internet application has its own *application protocol*
- This application protocol describes how data for that application are transmitted

Hypertext Transfer Protocol (HTTP)

- Application protocol used by the World Wide Web
- A web address is called a Uniform Resource Locator (URL)
- You type a URL into the address window of your browser
 - For example, `http://java.sun.com/index.html`

Browser Steps

1. Examines the part of the URL between the double slash and the first single slash
 - In this case: `java.sun.com`
 - This identifies the computer to which you want to connect
 - Because it contains letters, this part of the URL is a domain name, not an IP address
 - Browser sends request to a DNS server to obtain IP address for `java.sun.com`

Browser Steps

2. From the `http:` prefix, browser deduces that the protocol is HTTP
 - Uses port 80 by default
3. It establishes a TCP/IP connection to port 80 at IP address obtained in step 1

4. It deduces from the `/index.html` that you want to see the file `/index.html` and sends this request formatted as an HTTP command through the established connection

GET /index.html HTTP/1.0
a blank line

5. Web server running on computer whose IP Address was obtained above receives the request
 - It decodes the request
 - It fetches the file `/index.html`
 - It sends the file back to the browser on your computer

Browser Steps

6. The browser displays the contents of the file for you to see
 - Since this file is an HTML file, it translates the HTML codes into fonts, bullets, etc.
 - If the file contains images, it makes more GET requests through the same connection

- Telnet program allows you to
 - Type characters to send to a remote computer and
 - View the characters that the remote computer sends back
- It is a useful tool to establish test connections with servers
- You can imitate the browser connection by using a dialog box or typing at the command line

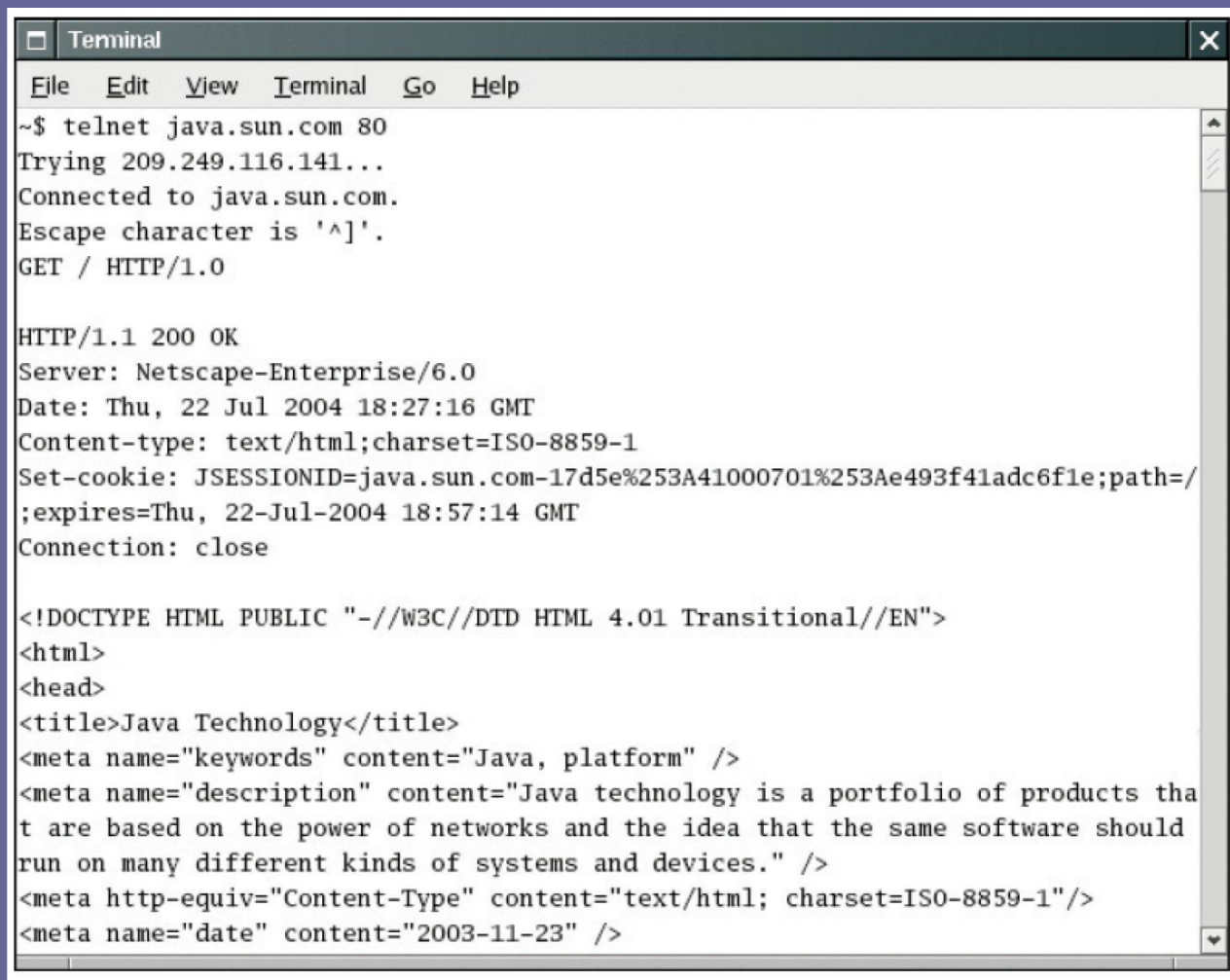
```
telnet java.sun.com 80
```


- After Telnet starts, type the following without using backspace then hit Enter twice

GET / HTTP/1.0

- The server responds to the request with the file
- Telnet is not a browser
- It does not understand HTML tags so it just displays everything it was sent

Web Server Response in Telnet



```
Terminal
File Edit View Terminal Go Help
~$ telnet java.sun.com 80
Trying 209.249.116.141...
Connected to java.sun.com.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 200 OK
Server: Netscape-Enterprise/6.0
Date: Thu, 22 Jul 2004 18:27:16 GMT
Content-type: text/html; charset=ISO-8859-1
Set-cookie: JSESSIONID=java.sun.com-17d5e%253A41000701%253Ae493f41adc6f1e; path=/
; expires=Thu, 22-Jul-2004 18:57:14 GMT
Connection: close

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Java Technology</title>
<meta name="keywords" content="Java, platform" />
<meta name="description" content="Java technology is a portfolio of products tha
t are based on the power of networks and the idea that the same software should
run on many different kinds of systems and devices." />
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
<meta name="date" content="2003-11-23" />
```

- Do not confuse HTTP with HTML
- HTML is a *document format* that describes the structure of a document
- HTTP is a protocol that describes the command set for web server requests

- Web browsers
 - Know how to display HTML documents
 - And how to issue HTTP commands
- Web servers
 - Know nothing about HTML
 - Merely understand HTTP and know how to fetch the requested items

HTTP Commands

Command	Meaning
GET	Return the requested item
HEAD	Request only the header information of an item
OPTIONS	Request communications option of an item
POST	Supply input to a server-side command and return the result
PUT	Store an item on the server
DELETE	Delete an item on the server
TRACE	Trace server communication

Application Level Protocols

- HTTP is one of many application protocols in use on the Internet
- Another commonly used protocol is the Post Office Protocol (POP)
- POP is used to download received messages from e-mail servers
- To send messages, you use another protocol: Simple Mail Transfer Protocol (SMTP)

Why don't you need to know about HTTP when you use a web browser?

Answer: The browser software translates your requests (typed URLs and mouse clicks on links) into HTTP commands that it sends to the appropriate web servers.

Why is it important that you don't make typing errors when you type HTTP commands in Telnet?

Answer: All keystrokes that you type, including the backspace key, are sent to the server. The server does not recognize a character sequence such as G W Backspace E T as a valid command.

Basic Network Commands

ping

ipconfig

tracert

netstat

net

nslookup

set

InetAddress class

```
public static InetAddress getLocalHost () throws  
    UnknownHostException
```

```
public static InetAddress getByName (String host) throws  
    UnknownHostException
```

```
public static InetAddress[] getAllByName (String host) throws  
    UnknownHostException
```

```
public byte[] getAddress()
```

```
public static getHostAddress()
```

AddrLookupApp.java

```
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.Scanner;

public class AddrLookupApp {
    public static void main (String args[]) {
        try {
            Scanner in = new Scanner(System.in);
            System.out.print("Host name or Host address: ");
            String strin = in.next();
            InetAddress host = InetAddress.getByName(strin);
            String hostName = host.getHostName();
            System.out.println("Host name: "+hostName);
            System.out.println("IP Address: "+host.getHostAddress());
        } catch (UnknownHostException e) {
            System.out.println("Address not found");
            return;
        }
    }
}
```

- Basic concepts of network
- Network programming

- Basic concepts of network
- Network programming

NETWORK PROGRAMMING IN JAVA 1

NETWORK PROGRAMMING

Chapter Goals

- To understand the concept of sockets
- To learn how to send and receive data through sockets
- To implement network clients and servers
- To communicate with web servers and server-side applications through Hypertext Transfer Protocol (HTTP)

A Client Program - Sockets

- A socket is an object that encapsulates a TCP/IP connection
- There is a *socket* on both ends of a connection
- Syntax to create a socket in a Java program

```
Socket s = new Socket(hostname, portnumber);
```


A Client Program - Sockets

- Code to connect to the HTTP port of server,

```
java.sun.com final int HTTP_PORT = 80;  
Socket s = new Socket("java.sun.com", HTTP_PORT);
```

- If it can't find the host, the `Socket` constructor throws an `UnknownHostException`

A Client Program - Input and Output Streams

- Use the input and output streams attached to the socket to communicate with the other endpoint
- Code to obtain the input and output streams

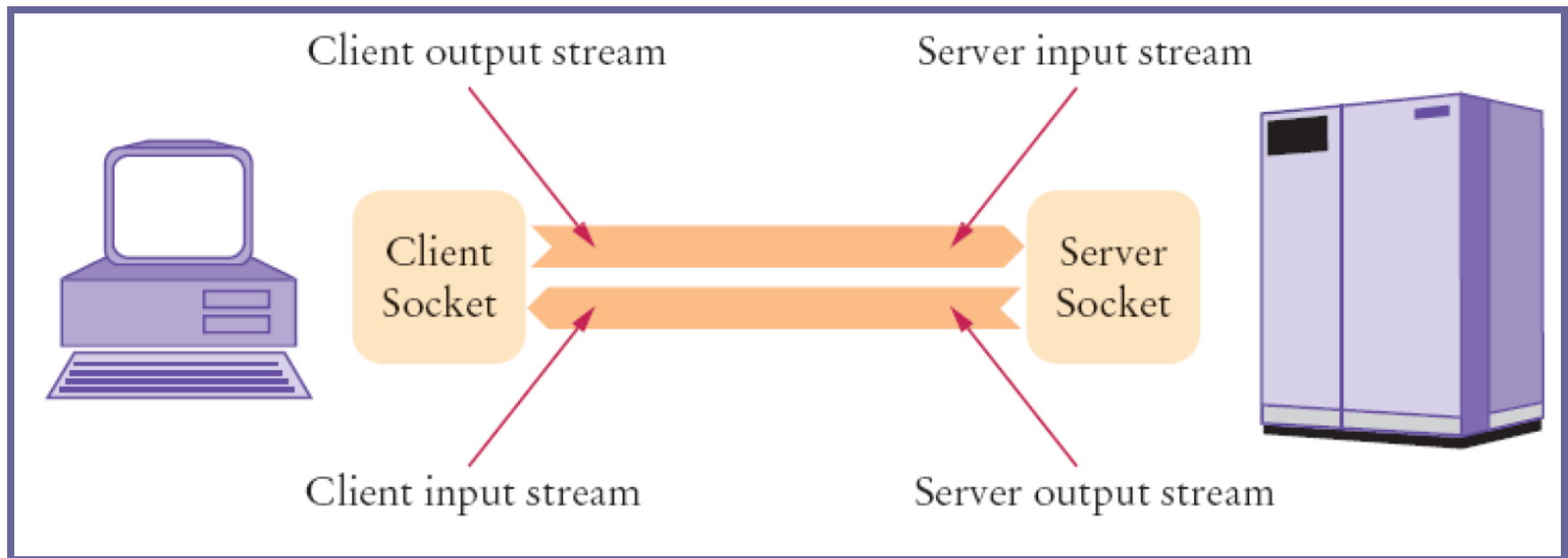
```
InputStream instream = s.getInputStream();  
OutputStream outstream = s.getOutputStream();
```

A Client Program - Input and Output Streams

- When you send data to `outstream`, the socket forwards them to the server
- The socket catches the server's response and you can read it through `instream`
- When you are done communicating with the server, close the socket

```
s.close();
```

Client and Server Sockets



A Client Program - Scanners and Writers

- `InputStream` and `OutputStream` send and receive bytes
- To send and receive text, use a scanner and a writer

```
Scanner in = new Scanner(instream);  
PrintWriter out = new PrintWriter(outstream);
```

A Client Program - Scanners and Writers

- A `PrintWriter` *buffers* the characters and only sends when the buffer is full
 - Buffering increases performance
- When sending a command, you want the whole command to be sent now
 - *Flush* the buffer manually:

```
out.print(command);  
out.flush();
```

A Client Program - WebGet

- This program lets you retrieve any item from a web server
- You specify the host and item from the command line
- For example:
The "/" denotes the root page of the web server that listens to port 80 of `java.sun.com`

```
java WebGet java.sun.com /
```

A Client Program - WebGet

- WebGet:
 - Establishes a connection to the host
 - Sends a `GET` command to the host
 - Receives input from the server until the server closes its connection


```
01: import java.io.InputStream;
02: import java.io.IOException;
03: import java.io.OutputStream;
04: import java.io.PrintWriter;
05: import java.net.Socket;
06: import java.util.Scanner;
07:
08: /**
09:  This program demonstrates how to use a socket to
10:  communicate with a web server. Supply the name of the
11:  host and the resource on the command-line, for example
12:  java WebGet java.sun.com index.html
13: */
14: public class WebGet
15: {
16:     public static void main(String[] args) throws IOException
17:     {
18:         // Get command-line arguments
```

```
19:
20:     String host;
21:     String resource;
22:
23:     if (args.length == 2)
24:     {
25:         host = args[0];
26:         resource = args[1];
27:     }
28:     else
29:     {
30:         System.out.println("Getting / from java.sun.com");
31:         host = "java.sun.com";
32:         resource = "/";
33:     }
34:
35:     // Open socket
36:
```

```
37: final int HTTP_PORT = 80;
38: Socket s = new Socket(host, HTTP_PORT);
39:
40: // Get streams
41:
42: InputStream instream = s.getInputStream();
43: OutputStream outstream = s.getOutputStream();
44:
45: // Turn streams into scanners and writers
46:
47: Scanner in = new Scanner(instream);
48: PrintWriter out = new PrintWriter(outstream);
49:
50: // Send command
51:
52: String command = "GET " + resource + " HTTP/1.0\n\n";
53: out.print(command);
54: out.flush();
55:
```

```
56:    // Read server response
57:
58:    while (in.hasNextLine())
59:    {
60:        String input = in.nextLine();
61:        System.out.println(input);
62:    }
63:
64:    // Always close the socket at the end
65:
66:    s.close();
67: }
68: }
```

Output

Getting / from java.sun.com

HTTP/1.1 200 OK

Server: Netscape-Enterprise/6.0

Date: Wed, 21 Jul 2004 23:47:27 GMT

Content-type: text/html; charset=ISO-8859-1

Connection: close

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

```
<html>
```

```
<head>
```

```
<title>
```

```
Java Technology</title>
```

```
...
```

```
</body>
```

```
</html>
```

What happens if you call `WebGet` with a nonexistent resource, such as `wombat.html` at `java.sun.com`?

Answer: The program makes a connection to the server, sends the `GET` request, and prints the error message that the server returns.

How do you open a socket to read e-mail from the POP server at e-mail.sjsu.edu?

Answer:

```
Socket s = new Socket("e-mail.sjsu.edu", 110);
```

A Server Program

- Sample server program: enables clients to manage bank accounts in a bank
- When you develop a server application, you need some application-level protocol
- The client can use this protocol to interact with the server
- A simple bank access protocol is described on the next slide

Simple Bank Access Protocol

Client Request	Server Response	Meaning
BALANCE n	n and the balance	Get the balance of account n
DEPOSIT n a	n and the new balance	Deposit amount a into account n
WITHDRAW n a	n and the new balance	Withdraw amount a from account n
QUIT	none	Quit the connection

A Server Program

- The server waits for clients to connect on a certain port
 - We choose 8888
- To listen for incoming connections, use a server socket
- To construct a server socket, provide the port number

```
ServerSocket server = new ServerSocket(8888);
```

A Server Program

- Use the `accept` method to wait for client connection and obtain a socket

```
Socket s = server.accept();  
BankService service = new BankService(s, bank);
```

A Server Program - BankService

- `BankService` carries out the service
- Implements the `Runnable` interface
- Its `run` method will be executed in each thread that serves a client connection

A Server Program - BankService

- `run` gets a scanner and writer from the socket, then executes:

```
public void doService() throws IOException
{
    while (true)
    {
        if (!in.hasNext()) return;
        String command = in.next();
        if (command.equals("QUIT"))
            return; executeCommand(command);
    }
}
```

A Server Program - `executeCommand`

- Processes a single command
- If the command is `DEPOSIT`, it carries out the deposit

```
int account = in.nextInt();  
double amount = in.nextDouble();  
bank.deposit(account, amount);
```

- `WITHDRAW` is handled in the same way

A Server Program - executeCommand

- After each command, the account number and new balance are sent to the client:

```
out.println(account + " " + bank.getBalance(account));
```

A Server Program

- `doService` returns to the `run` method if the client closed the connection or the command equals `QUIT`
- Then `run` closes the socket and exits
- How can we support multiple simultaneous clients?
 - Spawn a new thread whenever a client connects
 - Each thread is responsible for serving one client

A Server Program - Threads

- `BankService` implements `Runnable`; so, it can start a thread using `start()` (of class `Thread`)
- The thread dies when the client quits or disconnects and the `run` method exits

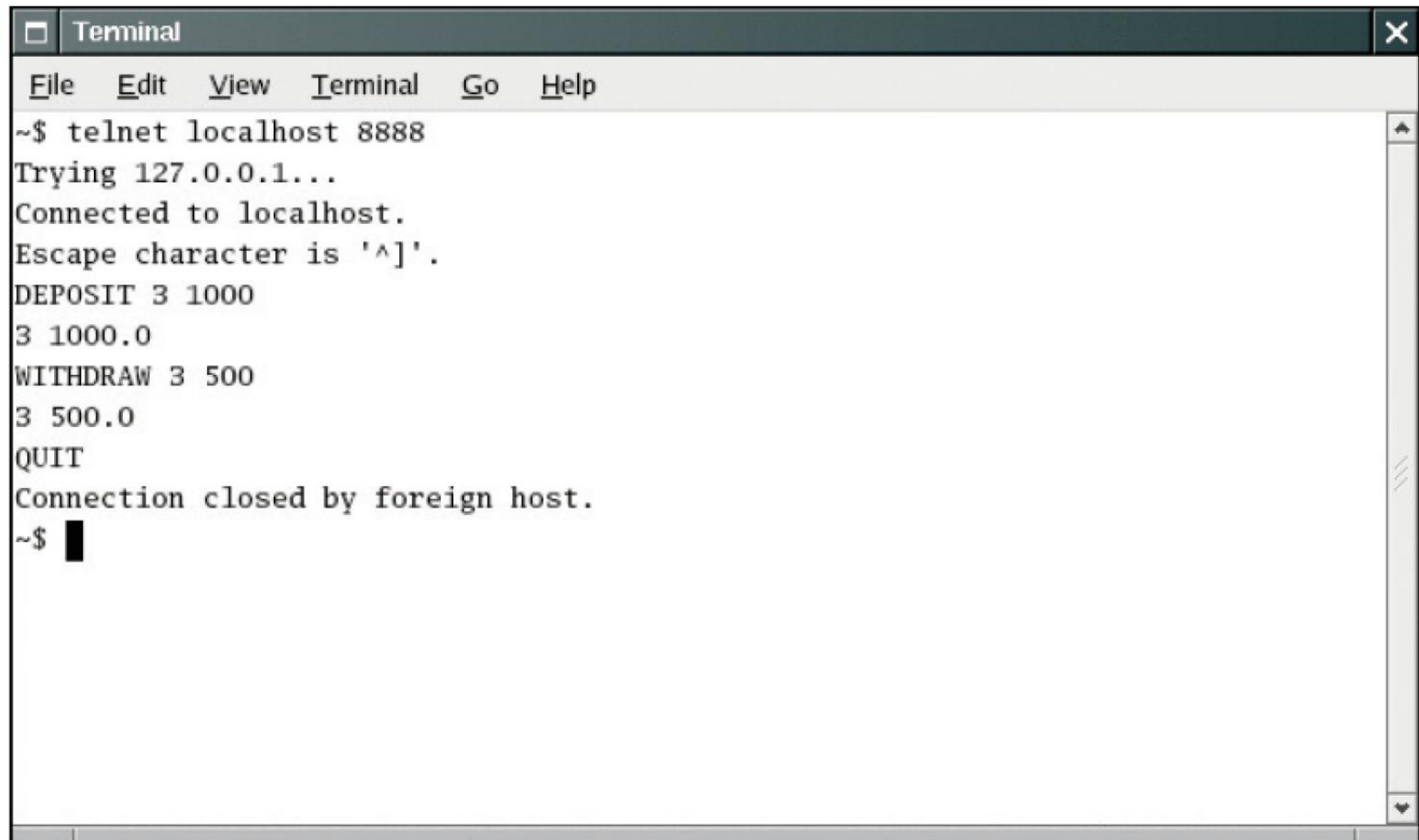
A Server Program - Threads

- In the meantime, `BankServer` loops back to accept the next connection

```
while (true)
{
    Socket s = server.accept();
    BankService service = new BankService(s, bank);
    Thread t = new Thread(service);
    t.start();
}
```

- The server program never stops
- When you are done running the server, you need to kill it

Using the Telnet Program to Connect to the Server



```
Terminal
File Edit View Terminal Go Help
~$ telnet localhost 8888
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
DEPOSIT 3 1000
3 1000.0
WITHDRAW 3 500
3 500.0
QUIT
Connection closed by foreign host.
~$
```

File BankServer.java

```
01: import java.io.IOException;
02: import java.net.ServerSocket;
03: import java.net.Socket;
04:
05: /**
06:  A server that executes the Simple Bank Access Protocol.
07:  */
08: public class BankServer
09: {
10:     public static void main(String[] args) throws IOException
11:     {
12:         final int ACCOUNTS_LENGTH = 10;
13:         Bank bank = new Bank(ACCOUNTS_LENGTH);
14:         final int SBAP_PORT = 8888;
15:         ServerSocket server = new ServerSocket(SBAP_PORT);
16:         System.out.println("Waiting for clients to connect...");
17:
```

File BankServer.java

```
18:    while (true)
19:    {
20:        Socket s = server.accept();
21:        System.out.println("Client connected.");
22:        BankService service = new BankService(s, bank);
23:        Thread t = new Thread(service);
24:        t.start();
25:    }
26: }
27: }
```

File BankService.java

```
01: import java.io.InputStream;
02: import java.io.IOException;
03: import java.io.OutputStream;
04: import java.io.PrintWriter;
05: import java.net.Socket;
06: import java.util.Scanner;
07:
08: /**
09:  Executes Simple Bank Access Protocol commands
10:  from a socket.
11: */
12: public class BankService implements Runnable
13: {
14:  /**
15:   Constructs a service object that processes commands
16:   from a socket for a bank.
17:   @param aSocket the socket
```

File BankService.java

```
18:    @param aBank the bank
19:    */
20:    public BankService(Socket aSocket, Bank aBank)
21:    {
22:        s = aSocket;
23:        bank = aBank;
24:    }
25:
26:    public void run()
27:    {
28:        try
29:        {
30:            try
31:            {
32:                in = new Scanner(s.getInputStream());
33:                out = new PrintWriter(s.getOutputStream());
34:                doService();
35:            }
36:            finally
```

File BankService.java

```
37:    {
38:        s.close();
39:    }
40: }
41: catch (IOException exception)
42: {
43:     exception.printStackTrace();
44: }
45: }
46:
47: /**
48:  Executes all commands until the QUIT command or the
49:  end of input.
50: */
51: public void doService() throws IOException
52: {
53:     while (true)
54:     {
```


File BankService.java

```
55:     if (!in.hasNext()) return;
56:     String command = in.next();
57:     if (command.equals("QUIT")) return;
58:     else executeCommand(command);
59: }
60: }
61:
62: /**
63:  Executes a single command.
64:  @param command the command to execute
65:  */
66: public void executeCommand(String command)
67: {
68:     int account = in.nextInt();
69:     if (command.equals("DEPOSIT"))
70:     {
```

```
71:     double amount = in.nextDouble();
72:     bank.deposit(account, amount);
73: }
74: else if (command.equals("WITHDRAW"))
75: {
76:     double amount = in.nextDouble();
77:     bank.withdraw(account, amount);
78: }
79: else if (!command.equals("BALANCE"))
80: {
81:     out.println("Invalid command");
82:     out.flush();
83:     return;
84: }
85: out.println(account + " " + bank.getBalance(account));
86: out.flush();
87: }
88:
89: private Socket s;
90: private Scanner in;
91: private PrintWriter out;
92: private Bank bank;
93: }
```

```
01: /**
02:  A bank consisting of multiple bank accounts.
03: */
04: public class Bank
05: {
06:     /**
07:      Constructs a bank account with a given number of
08:      // accounts.
09:      @param size the number of accounts
10:     */
11:     public Bank(int size)
12:     {
13:         accounts = new BankAccount[size];
14:         for (int i = 0; i < accounts.length; i++)
15:             accounts[i] = new BankAccount();
16:     }
```

```
17:  /**
18:     Deposits money into a bank account.
19:     @param accountNumber the account number
20:     @param amount the amount to deposit
21:  */
22:  public void deposit(int accountNumber, double amount)
23:  {
24:      BankAccount account = accounts[accountNumber];
25:      account.deposit(amount);
26:  }
27:
28:  /**
29:     Withdraws money from a bank account.
30:     @param accountNumber the account number
31:     @param amount the amount to withdraw
32:  */
```

```

33: public void withdraw(int accountNumber, double amount)
34: {
35:     BankAccount account = accounts[accountNumber];
36:     account.withdraw(amount);
37: }
38:
39: /**
40:     Gets the balance of a bank account.
41:     @param accountNumber the account number
42:     @return the account balance
43: */
44: public double getBalance(int accountNumber)
45: {
46:     BankAccount account = accounts[accountNumber];
47:     return account.getBalance();
48: }
49:
50: private BankAccount[] accounts;
51: }
52:
53:

```

File BankClient.java

```
01: import java.io.InputStream;
02: import java.io.IOException;
03: import java.io.OutputStream;
04: import java.io.PrintWriter;
05: import java.net.Socket;
06: import java.util.Scanner;
07:
08: /**
09:  * This program tests the bank server.
10:  */
11: public class BankClient
12: {
13:     public static void main(String[] args) throws IOException
14:     {
15:         final int SBAP_PORT = 8888;
16:         Socket s = new Socket("localhost", SBAP_PORT);
17:         InputStream instream = s.getInputStream();
18:         OutputStream outstream = s.getOutputStream();
```

File BankClient.java

```
19: Scanner in = new Scanner(instream);
20: PrintWriter out = new PrintWriter(outstream);
21:
22: String command = "DEPOSIT 3 1000\n";
23: System.out.print("Sending: " + command);
24: out.print(command);
25: out.flush();
26: String response = in.nextLine();
27: System.out.println("Receiving: " + response);
28:
29: command = "WITHDRAW 3 500\n";
30: System.out.print("Sending: " + command);
31: out.print(command);
32: out.flush();
33: response = in.nextLine();
34: System.out.println("Receiving: " + response);
35:
36: command = "QUIT\n";
37: System.out.print("Sending: " + command);
38: out.print(command);
39: out.flush();
40:
41: s.close();
42: }
43: }
```

Output

Sending: DEPOSIT 3 1000

Receiving: 3 1000.0

Sending: WITHDRAW 3 500

Receiving: 3 500.0

Sending: QUIT

Why didn't we choose port 80 for the bank server?

Answer: Port 80 is the standard port for HTTP. If a web server is running on the same computer, then one can't open a server socket on an open port.

Can you read data from a server socket?

Answer: No, a server socket just waits for a connection and yields a regular `Socket` object when a client has connected. You use that socket object to read the data that the client sends.

- `URLConnection` Class
 - Provides convenient support for HTTP
 - Can also handle FTP (*file transfer protocol*)
 - Takes care of socket connection for you
 - Makes it easy to communicate with a web server without giving HTTP commands

URL Connections

- Construct a `URL` object from a URL starting with the `http` or `ftp` prefix

```
URL u = new URL("http://java.sun.com/index.html");
```

- Use the `URL`'s `openConnection()` method to get the `URLConnection`

```
URLConnection connection = u.openConnection();
```

- Call the `getInputStream` method to obtain an input stream

```
InputStream instream = connection.getInputStream();
```

- You can turn the stream into a scanner in the usual way

command

request properties

blank line

- *HTTP command*
 - Such as `GET item HTTP/1.0`
- *request properties*
 - Such as `If-Modified-Since: date`
- *blank line*
 - separates the command and its request properties from the input data

URLConnection Class

- Has methods to set request properties

```
connection.setIfModifiedSince(date);
```

- Set the request properties before calling `getInputStream`
- The `URLConnection` class sends all the request properties that are set to the web server

Server Response

- Server response:

status line containing response code
response parameters
blank line

- For example:

```
HTTP/1.1 200 OK
Date: Sun, 28 Aug 2005 00:15:48 GMT
Server: Apache/1.3.3 (Unix)
Last-Modified: Thu, 23 Jun 2005 20:53:38 GMT
Content-Length: 4813
Content-Type: text/html
blank line
requested data
```


Server Response

- Normally, you don't see the response code, in this case: 200 OK
- You may have run across bad links and see a page that contains response code 404 Not Found

JAVA Retrieving Response Code and Message

- Cast the `URLConnection` object to the `HttpURLConnection` subclass
- Get the response code with `getResponseCode`
- Get the response message with `getResponseMessage`

```
HttpURLConnection httpConnection = (HttpURLConnection) connection;  
int code = httpConnection.getResponseCode(); // e.g., 404  
String message = httpConnection.getResponseMessage();  
// e.g., "Not found"
```

Retrieve Other Response Information from URLConnection

- Content length

```
int length = connection.getContentLength();
```

- Content type

```
String type = connection.getContentType();
```

```
01: import java.io.InputStream;
02: import java.io.IOException;
03: import java.io.OutputStream;
04: import java.io.PrintWriter;
05: import java.net.HttpURLConnection;
06: import java.net.URL;
07: import java.net.URLConnection;
08: import java.util.Scanner;
09:
10: /**
11:  This program demonstrates how to use an URL connection
12:  to communicate with a web server. Supply the URL on the
13:  command-line, for example
14:  java URLGet http://java.sun.com/index.html
15: */
16: public class URLGet
17: {
```

```
18: public static void main(String[] args) throws IOException
19: {
20:     // Get command-line arguments
21:
22:     String urlString;
23:     if (args.length == 1)
24:         urlString = args[0];
25:     else
26:     {
27:         urlString = "http://java.sun.com/";
28:         System.out.println("Using " + urlString);
29:     }
30:
31:     // Open connection
32:
33:     URL u = new URL(urlString);
34:     URLConnection connection = u.openConnection();
35:
```

```
36:    // Check if response code is HTTP_OK (200)
37:
38:    HttpURLConnection httpConnection
39:        = (HttpURLConnection) connection;
40:    int code = httpConnection.getResponseCode();
41:    String message = httpConnection.getResponseMessage();
42:    System.out.println(code + " " + message);
43:    if (code != HttpURLConnection.HTTP_OK)
44:        return;
45:
46:    // Read server response
47:
48:    InputStream instream = connection.getInputStream();
49:    Scanner in = new Scanner(instream);
50:
51:    while (in.hasNextLine())
52:    {
53:        String input = in.nextLine();
54:        System.out.println(input);
55:    }
56: }
57: }
```

Output

```
Using http://java.sun.com/  
200 OK <  
!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML  
4.01 Transitional//EN">  
<html>  
<head>  
<title>J  
ava Technology  
</title>  
...  
</body>  
</html>
```

Why is it better to use an `URLConnection` instead of a socket when reading data from a web server?

Answer: The `URLConnection` class understands the HTTP protocol, freeing you from assembling requests and analyzing response headers.

What happens if you use the `URLGet` program to request an image (such as `http://java.sun.com/im/logo_java.gif`)?

Answer: The bytes that encode the images are displayed on the console, but they will appear to be random gibberish.

THANK YOU FOR YOUR ATTENTION !