Java Networking

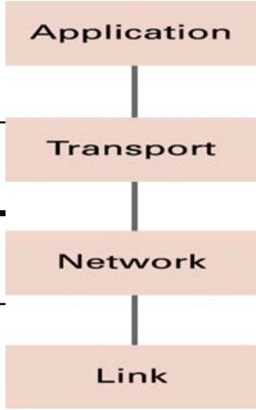
Outline

- Networking Basics
  - TCP, UDP, Ports, DNS, Client-Server Model
- TCP/IP in Java
- Sockets
- URL
  - The java classes: `URL, URLEncoder, URLConnection, HTTPURLConnection`
- Datagrams

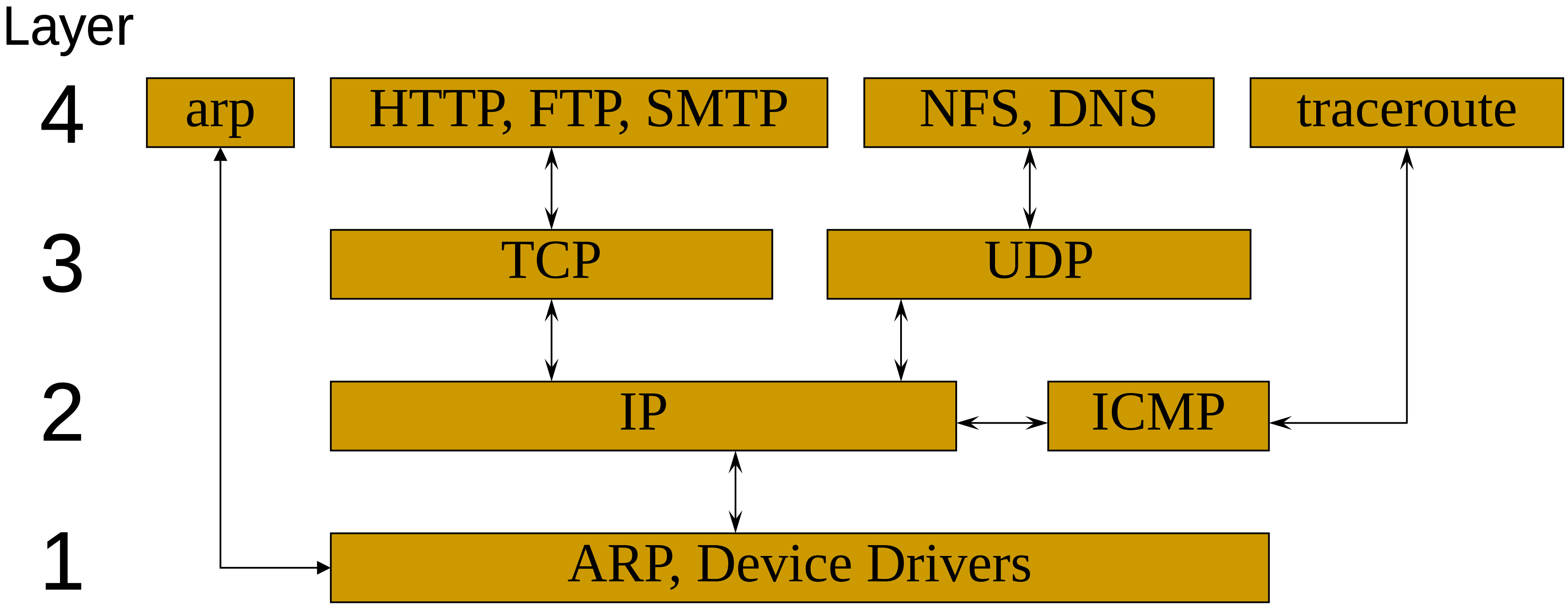
Networking Basics

- Computers running on the Internet communicate with each other using either the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP)
- TCP/IP network model

Layer	**Function**
Application	End-user application programs
Transport	Communication among programs on a net (TCP/UDP)
Network	Basic communication, addressing, and routing (IP, ICMP)
Link(Data Link)	Network hardware and device drivers(ARP, RARP)



Networking Basics



Layer

4

arp

HTTP, FTP, SMTP

NFS, DNS

traceroute

3

TCP

UDP

2

IP

ICMP

1

ARP, Device Drivers

Ethernet Header	IP Header	TCP Header	Application Data	Ethernet Trailer
ETHERNET FRAME				

TCP (Transmission Control Protocol)

- A connection-based protocol that provides a reliable flow of data between two computers.
- Provides a point-to-point channel for applications that require reliable communications.
  - The Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), and Telnet are all examples of applications that require a reliable communication channel
- Guarantees that data sent from one end of the connection actually gets to the other end and in the same order it was sent. Otherwise, an error is reported.

UDP (User Datagram Protocol)

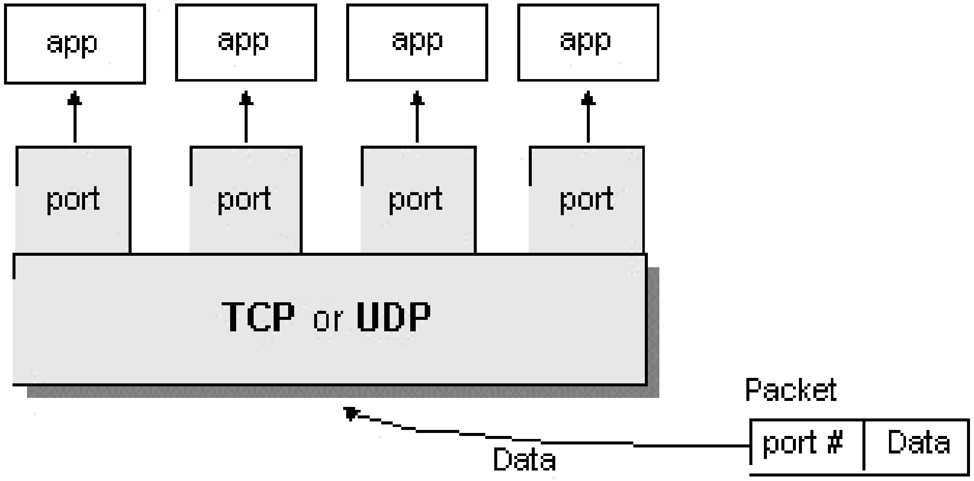
- A protocol that sends independent packets of data, called datagrams, from one computer to another with no guarantees about arrival. UDP is not connection-based like TCP and is not reliable:
  - Sender does not wait for acknowledgements
  - Arrival order is not guaranteed
  - Arrival is not guaranteed
- Used when speed is essential, even in cost of reliability
  - e.g. streaming media, games, Internet telephony, etc.

Ports

- Data transmitted over the Internet is accompanied by addressing information that identifies the computer and the port for which it is destined.
  - The computer is identified by its 32-bit IP address, which IP uses to deliver data to the right computer on the network. Ports are identified by a 16-bit number, which TCP and UDP use to deliver the data to the right application.
- Why don’t we specify the port in a Web browser?

Ports – Cont.

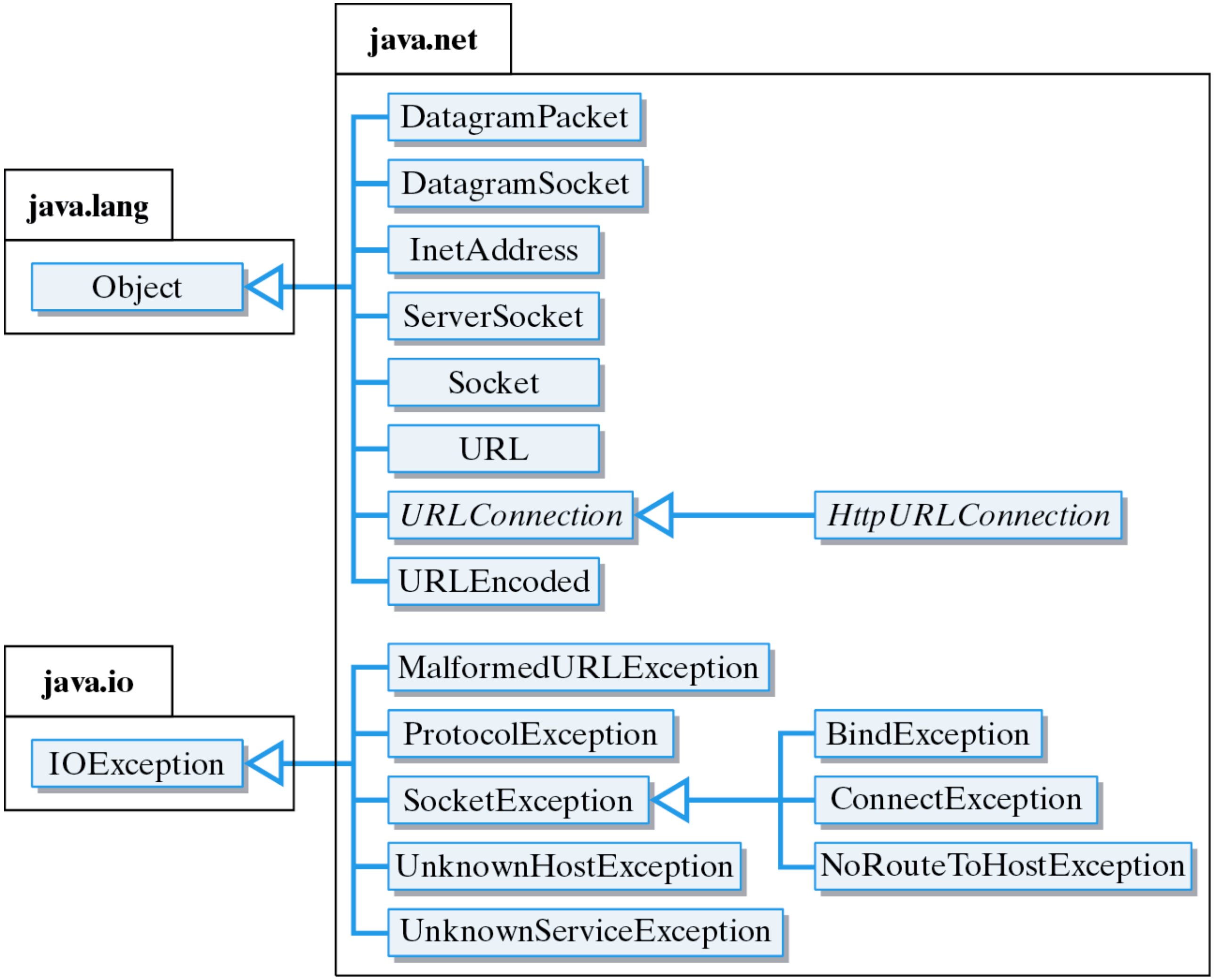
- Port numbers range from 0 to 65,535 (16-bit)
  - Ports 0 - 1023 are called *well-known ports*. They are reserved for use by well-known services:
    - 20, 21: FTP
    - 23: TELNET
    - 25: SMTP
    - 110: POP3
    - 80: HTTP



Networking Classes in the JDK

- Through the classes in java.net, Java programs can use TCP or UDP to communicate over the Internet.
  - The `URL, URLConnection, Socket,` and `ServerSocket` classes all use TCP to communicate over the network.
  - The `DatagramPacket, DatagramSocket`, and `MulticastSocket` classes are for use with UDP.

Networking Classes in the JDK



TCP/IP in Java

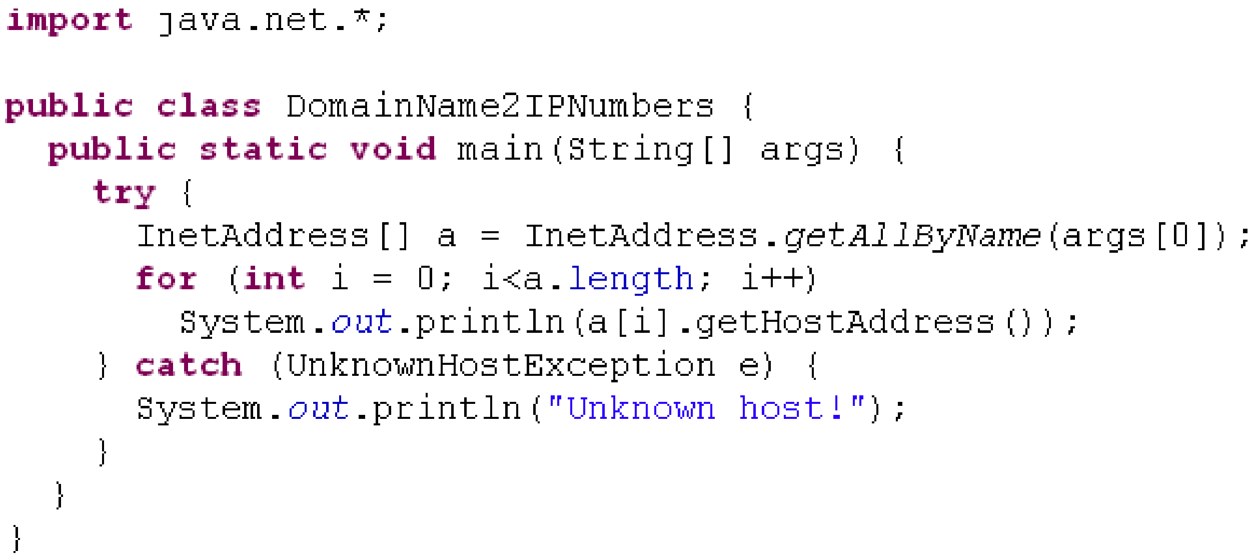
- Accessing TCP/IP from Java is straightforward. The main functionality is in the following classes:
  - `Java.net.InetAddress` : Represents an IP address (either IPv4 or IPv6) and has methods for performing DNS lookup (next slide).
  - `Java.net.Socket` : Represents a TCP socket.
  - `Java.net.ServerSocket` : Represents a server socket which is capable of waiting for requests from clients.

DNS - Domain name system

- The **Domain Name system** (DNS) associates various sorts of information with so-called domain names.
- Most importantly, it serves as the "phone book" for the Internet by translating human-readable computer hostnames, e.g. *www.example.com*, into the IP addresses, e.g. *208.77.188.166*, that networking equipment needs to deliver information.
- It also stores other information such as the list of mail exchange servers that accept email for a given domain.

DNS Lookup Example

- The following program performs a DNS lookup to find the IP numbers that are associated with a given domain name.



```
import java.net.*;

public class DomainName2IPNumbers {
  public static void main(String[] args) {
    try {
      InetAddress[] a = InetAddress.getAllByName(args[0]);
      for (int i = 0; i<a.length; i++)
        System.out.println(a[i].getHostAddress());
    } catch (UnknownHostException e) {
      System.out.println("Unknown host!");
    }
  }
}
```

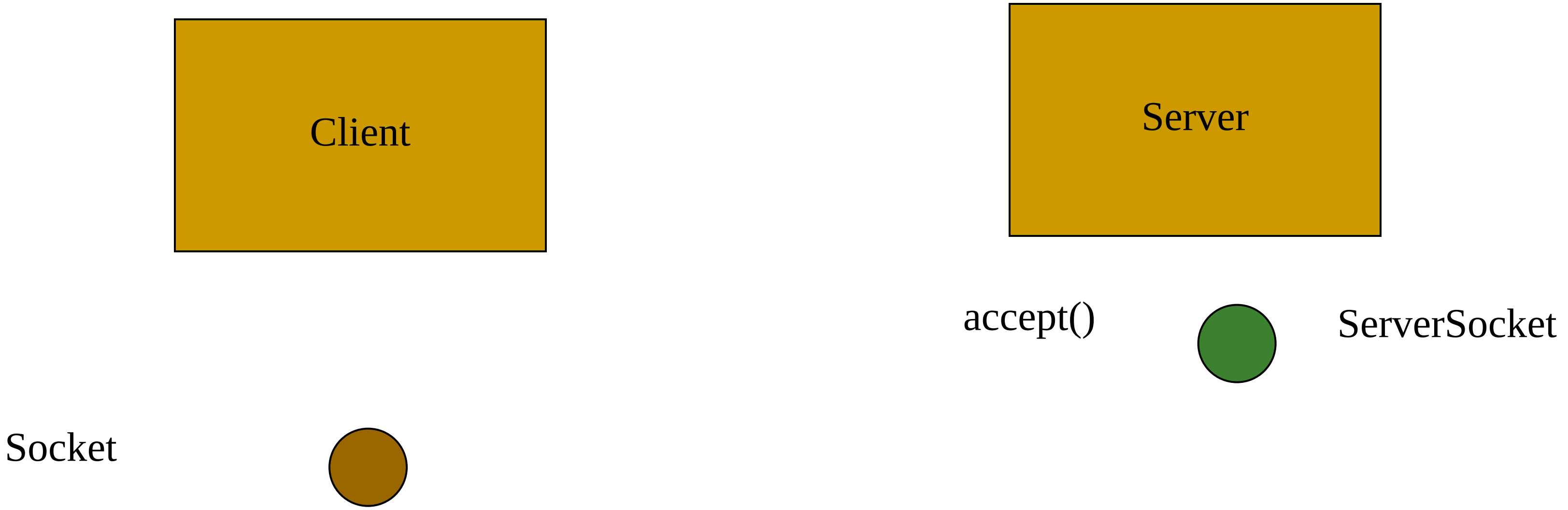
Client-Server Model

- A common paradigm for distributed applications
- Asymmetry in connection establishment:
  - Server waits for client requests at a well known address (IP+port)
  - Connection is established upon client request
- For example: Web servers and browsers

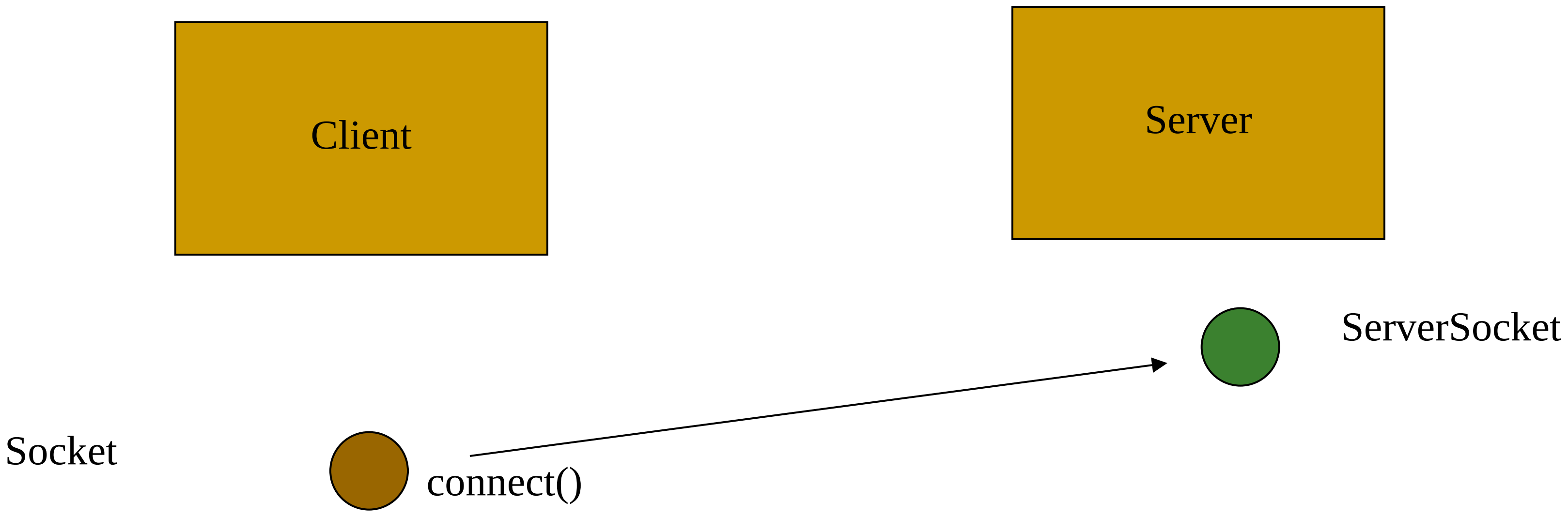
Sockets: Low-Level Networking

- A *socket* is one endpoint of a two-way communication link between two programs running on the network.
- An endpoint is a combination of an IP address and a port number.
- A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent.

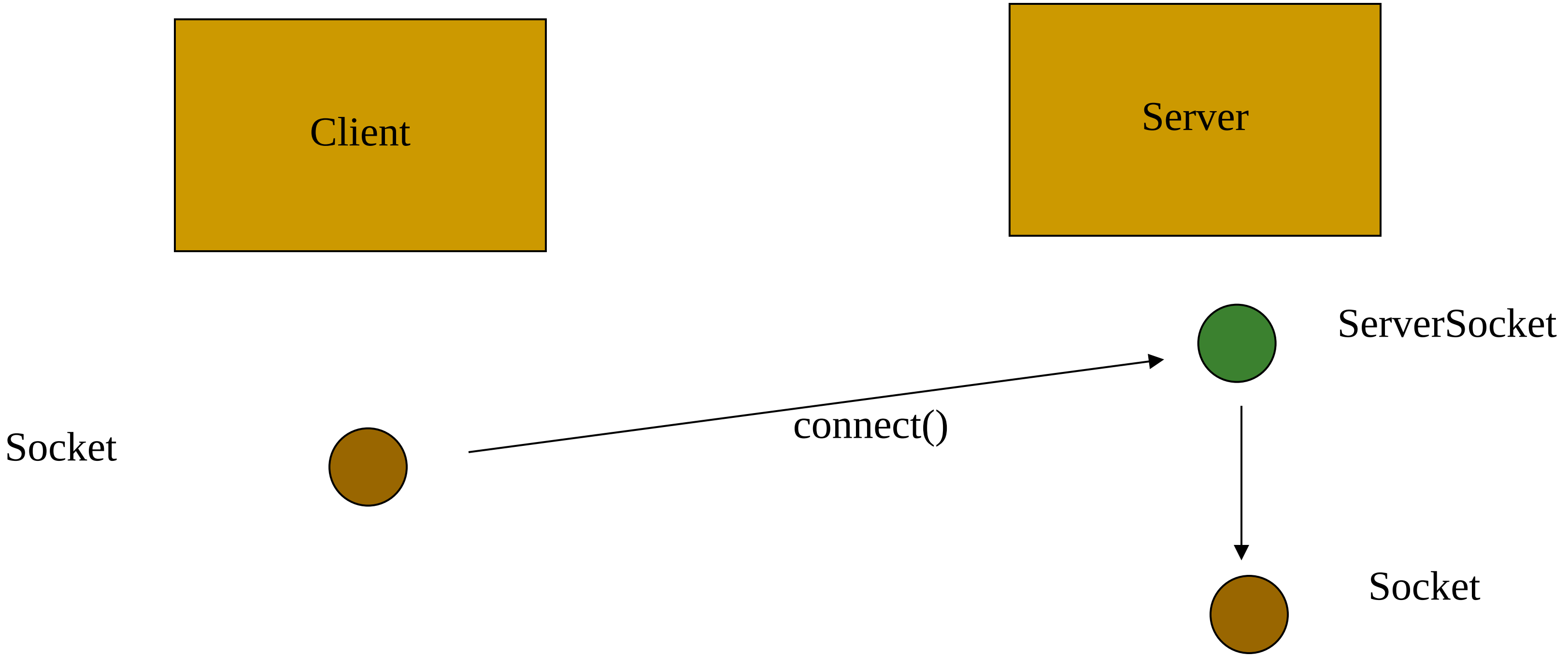
TCP Sockets



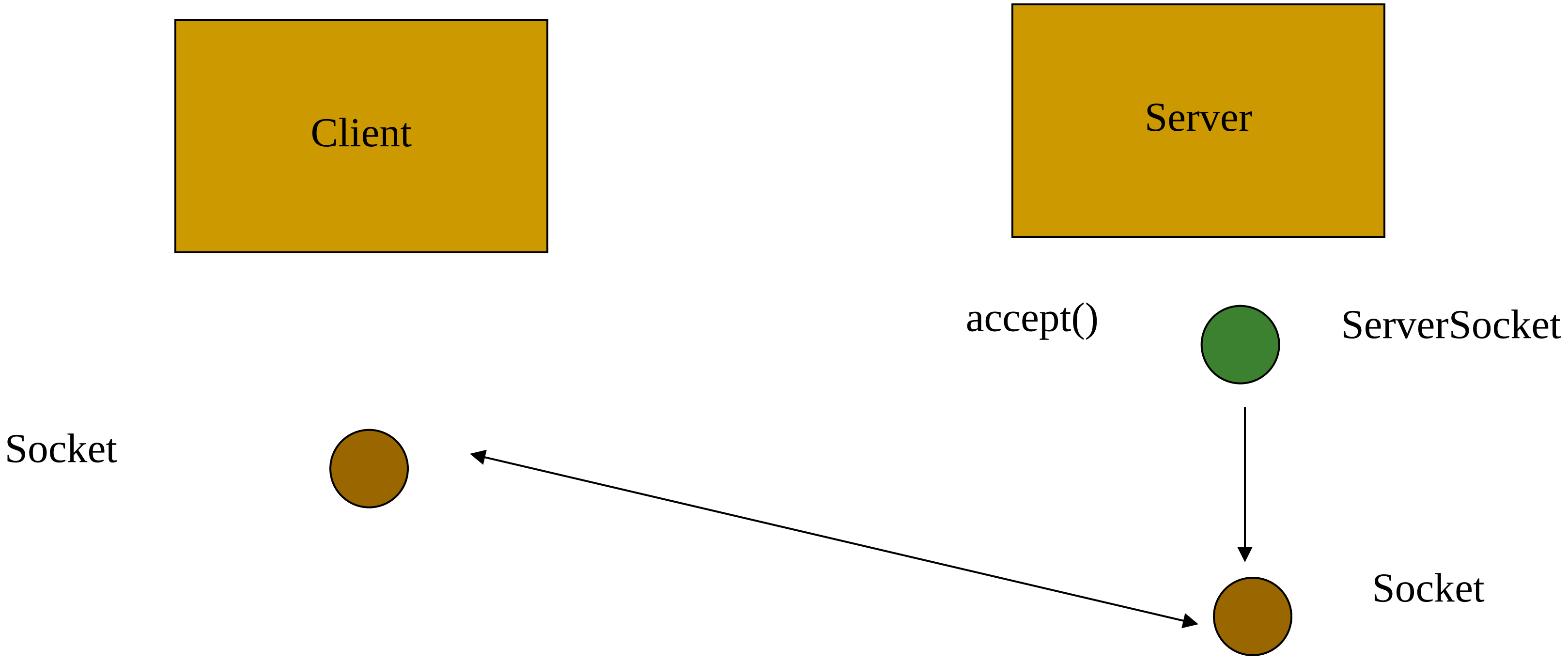
Sockets



Sockets

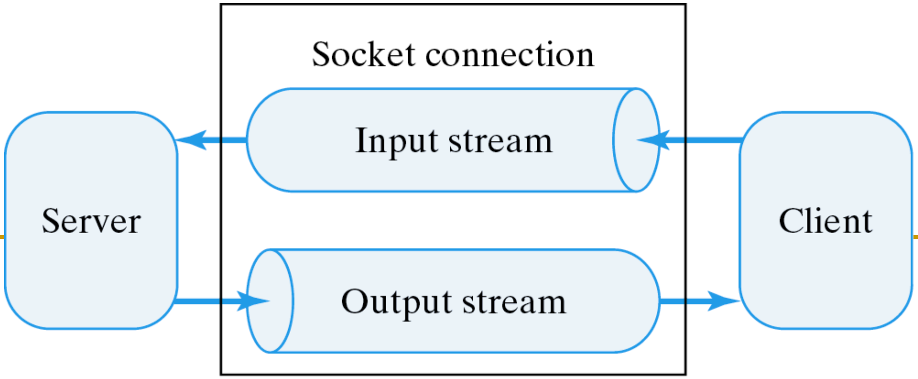


Sockets



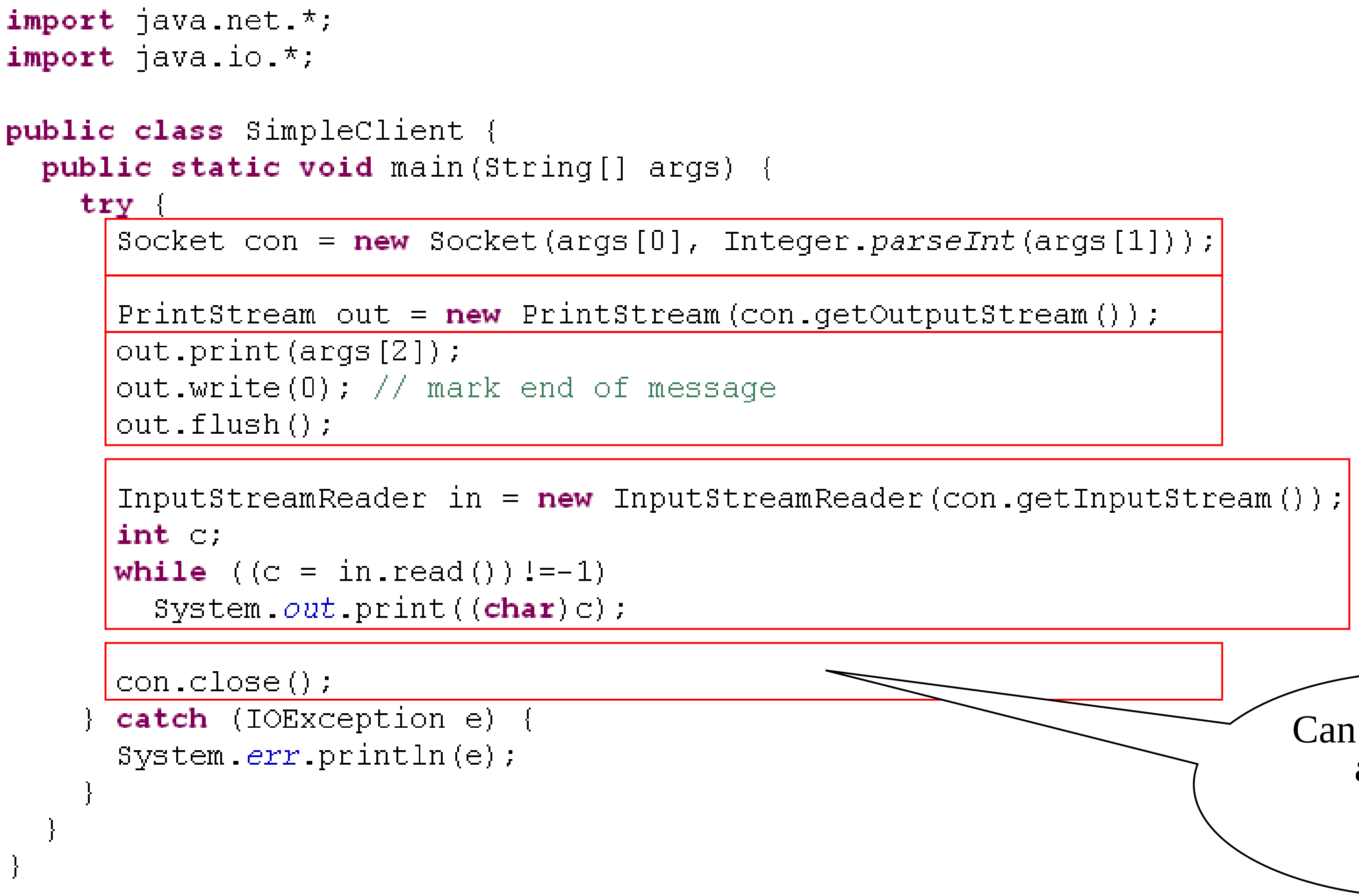
Java Sockets

- Java wraps OS sockets (over TCP) by the objects of class `java.net.Socket`
- `new Socket(String remoteHost, int remotePort)`
  Creates a TCP socket and connects it to the remote host on the remote port (hand shake)
- Write and read using streams:
  - `InputStream getInputStream()`
  - `OutputStream getOutputStream()`
- The Socket API



SimpleClient Example

- This client tries to connect to a server (coming soon…) and port given on the command line, then sends some output to the server, and finally receives some input which is printed on the screen.



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

public class SimpleClient {
  public static void main(String[] args) {
    try {
      Socket con = new Socket(args[0], Integer.parseInt(args[1]));

      PrintStream out = new PrintStream(con.getOutputStream());
      out.print(args[2]);
      out.write(0); // mark end of message
      out.flush();

      InputStreamReader in = new InputStreamReader(con.getInputStream());
      int c;
      while ((c = in.read())!=-1)
        System.out.print((char)c);

      con.close();
    } catch (IOException e) {
      System.err.println(e);
    }
  }
}
```

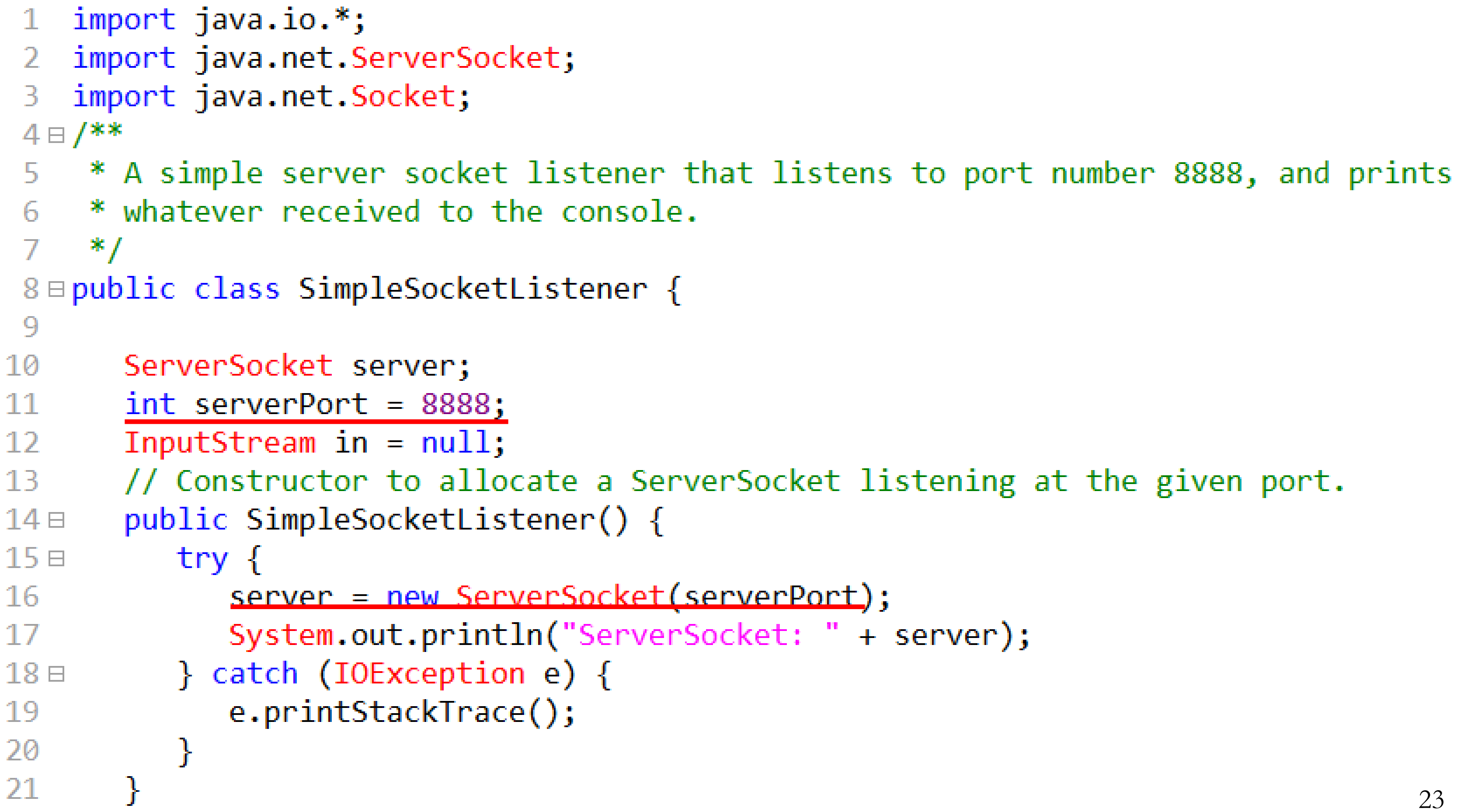
Can you think of a better place for the close action?

Class ServerSocket

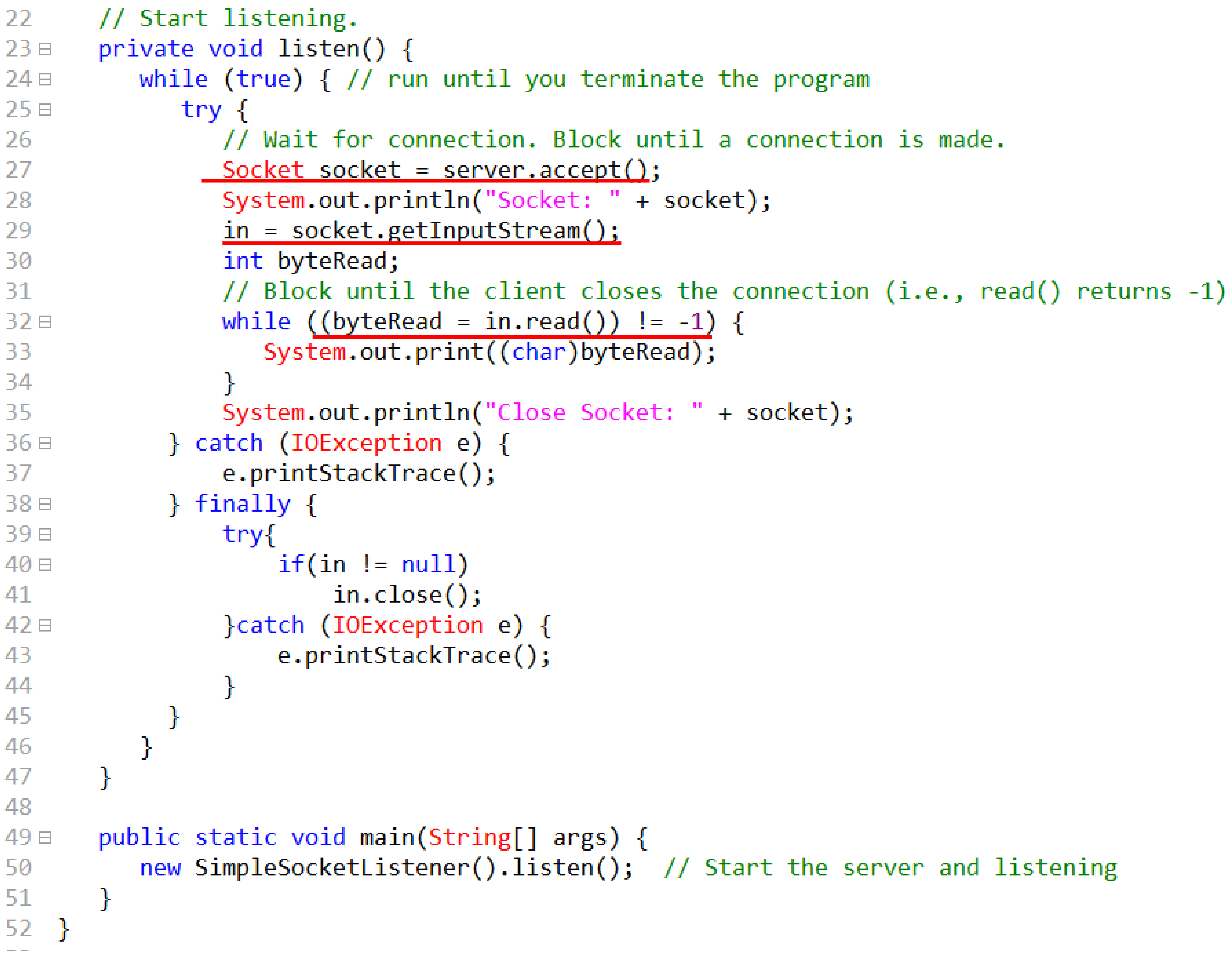
- This class implements server sockets. A server socket waits for requests to come in over the network. It performs some operation based on that request, and then possibly returns a result to the requester.
- A server socket is technically not a socket: when a client connects to a server socket, a TCP connection is made, and a (normal) socket is created for each end point.

SimpleServer Example

- The server creates a server socket that listen on the port given on the command line. It then enters infinite loop doing the following: when a connection with a client is established it reads some input from the client (terminated with a 0 byte).

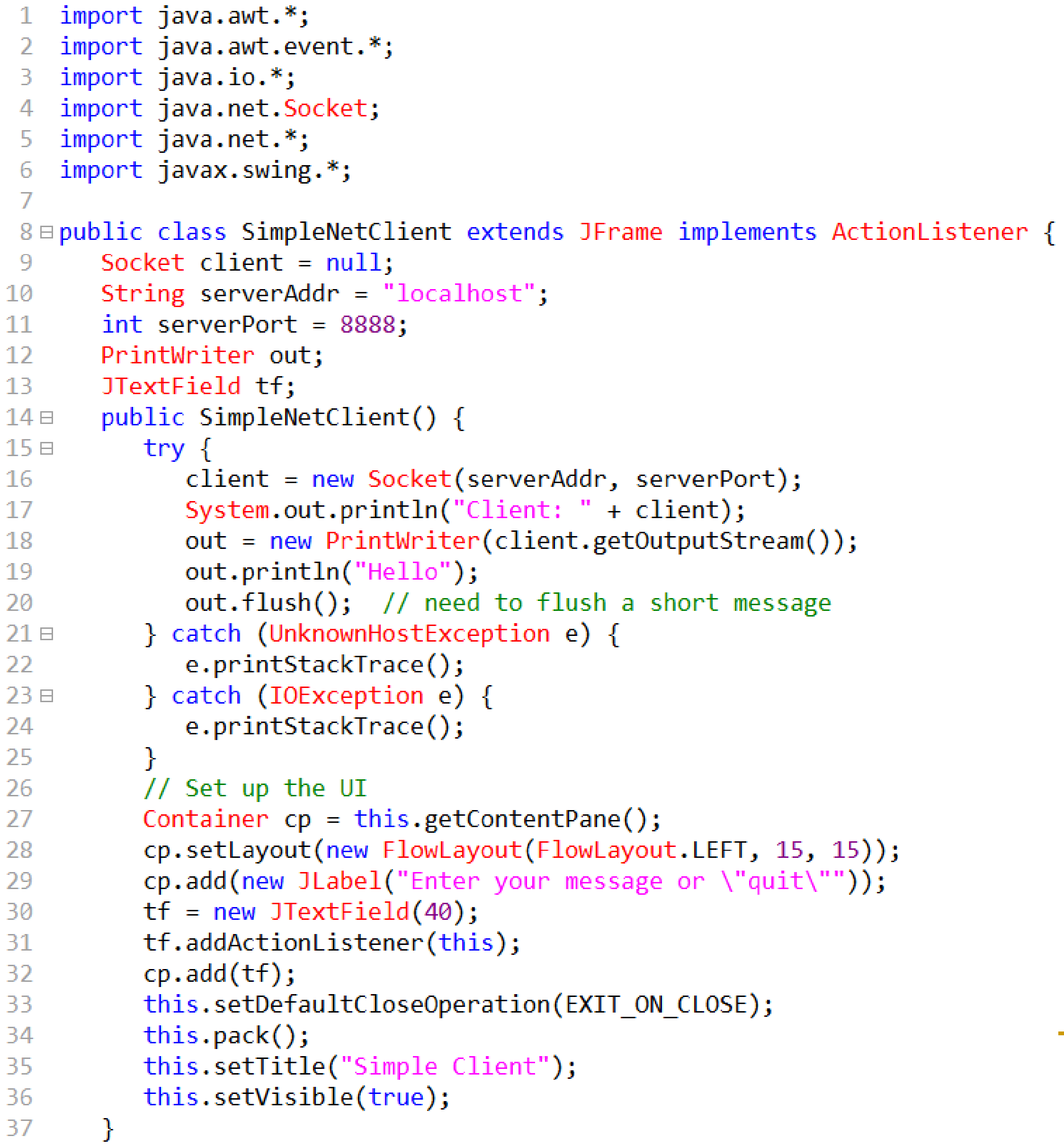


```java
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
/**
 * A simple server socket listener that listens to port number 8888, and prints
 * whatever received to the console.
 */
public class SimpleSocketListener {

    ServerSocket server;
    int serverPort = 8888;
    InputStream in = null;
    // Constructor to allocate a ServerSocket listening at the given port.
    public SimpleSocketListener() {
        try {
            server = new ServerSocket(serverPort);
            System.out.println("ServerSocket: " + server);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```



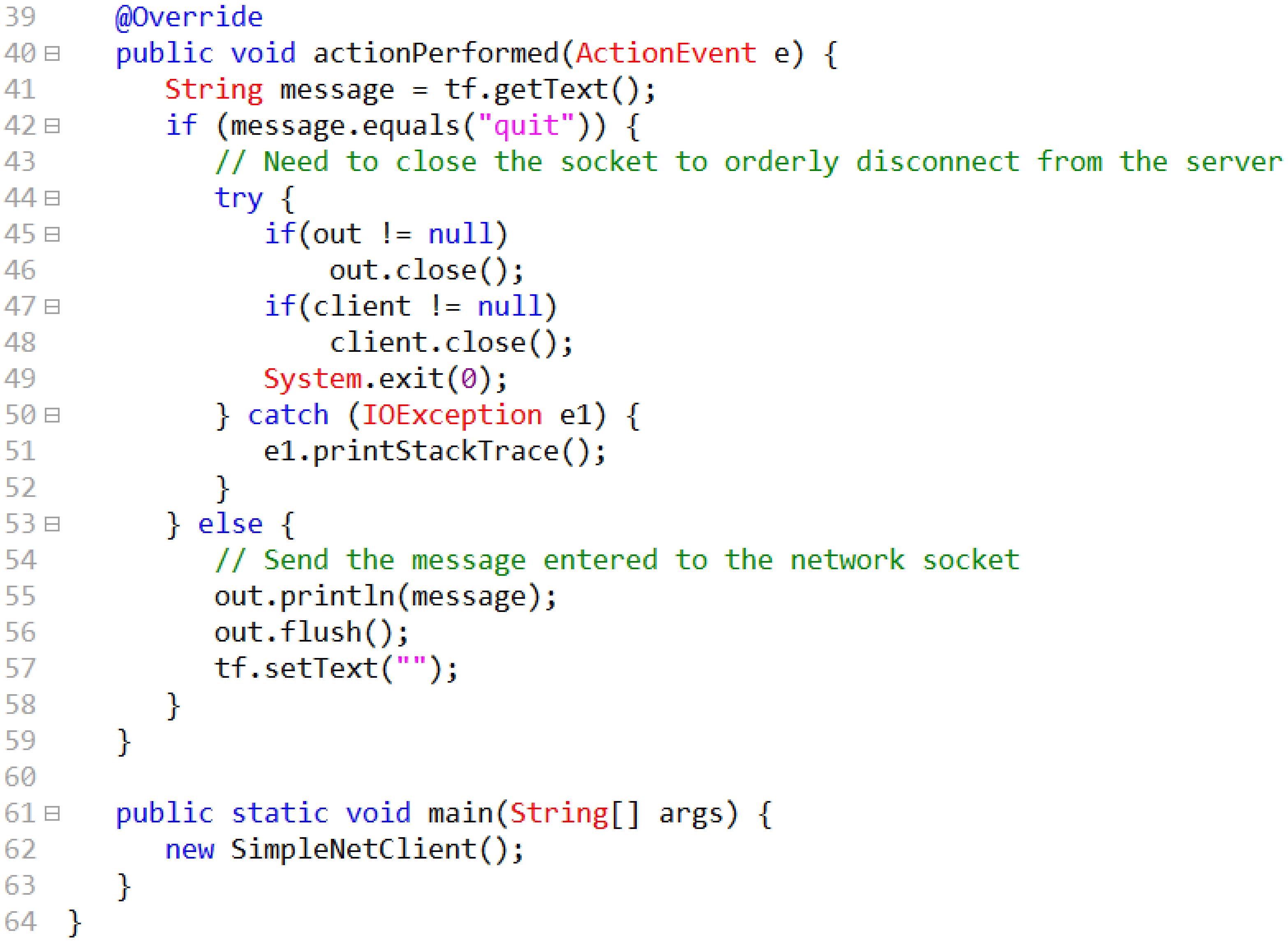
```java
   // Start listening.
   private void listen() {
      while (true) { // run until you terminate the program
         try {
            // Wait for connection. Block until a connection is made.
            Socket socket = server.accept();
            System.out.println("Socket: " + socket);
            in = socket.getInputStream();
            int byteRead;
            // Block until the client closes the connection (i.e., read() returns -1)
            while ((byteRead = in.read()) != -1) {
               System.out.print((char)byteRead);
            }
            System.out.println("Close Socket: " + socket);
         } catch (IOException e) {
            e.printStackTrace();
         } finally {
            try{
                if(in != null)
                    in.close();
            }catch (IOException e) {
                e.printStackTrace();
            }
         }
      }
   }

   public static void main(String[] args) {
      new SimpleSocketListener().listen();  // Start the server and listening
   }
}
```

SimpleNetClient Example



```java
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.Socket;
import java.net.*;
import javax.swing.*;

public class SimpleNetClient extends JFrame implements ActionListener {
   Socket client = null;
   String serverAddr = "localhost";
   int serverPort = 8888;
   PrintWriter out;
   JTextField tf;
   public SimpleNetClient() {
      try {
         client = new Socket(serverAddr, serverPort);
         System.out.println("Client: " + client);
         out = new PrintWriter(client.getOutputStream());
         out.println("Hello");
         out.flush();  // need to flush a short message
      } catch (UnknownHostException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }
      // Set up the UI
      Container cp = this.getContentPane();
      cp.setLayout(new FlowLayout(FlowLayout.LEFT, 15, 15));
      cp.add(new JLabel("Enter your message or \"quit\""));
      tf = new JTextField(40);
      tf.addActionListener(this);
      cp.add(tf);
      this.setDefaultCloseOperation(EXIT_ON_CLOSE);
      this.pack();
      this.setTitle("Simple Client");
      this.setVisible(true);
   }
```

SimpleNetClient Example

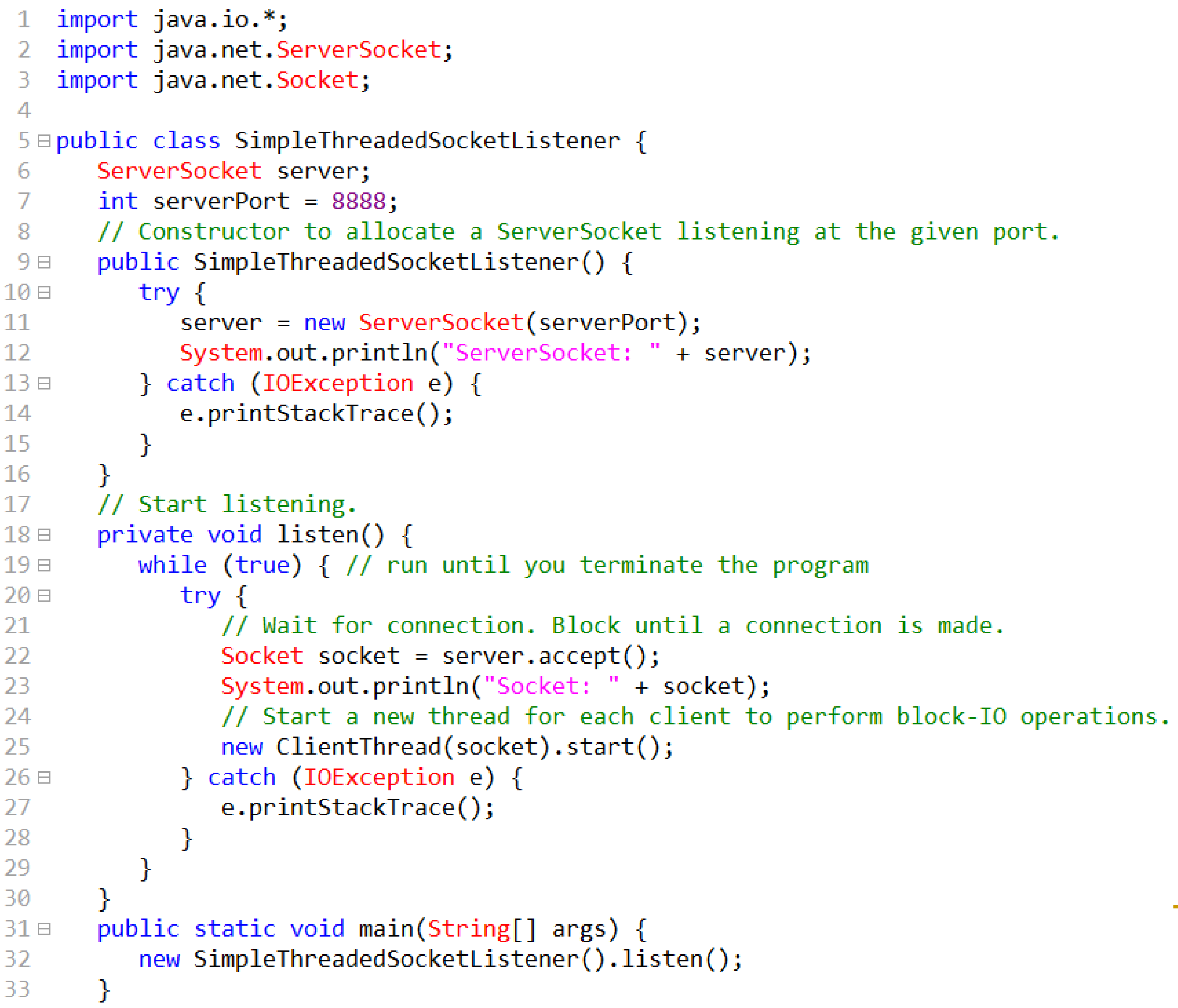


```java
    @Override
    public void actionPerformed(ActionEvent e) {
        String message = tf.getText();
        if (message.equals("quit")) {
            // Need to close the socket to orderly disconnect from the server
            try {
                if(out != null)
                    out.close();
                if(client != null)
                    client.close();
                System.exit(0);
            } catch (IOException e1) {
                e1.printStackTrace();
            }
        } else {
            // Send the message entered to the network socket
            out.println(message);
            out.flush();
            tf.setText("");
        }
    }

    public static void main(String[] args) {
        new SimpleNetClient();
    }
}
```

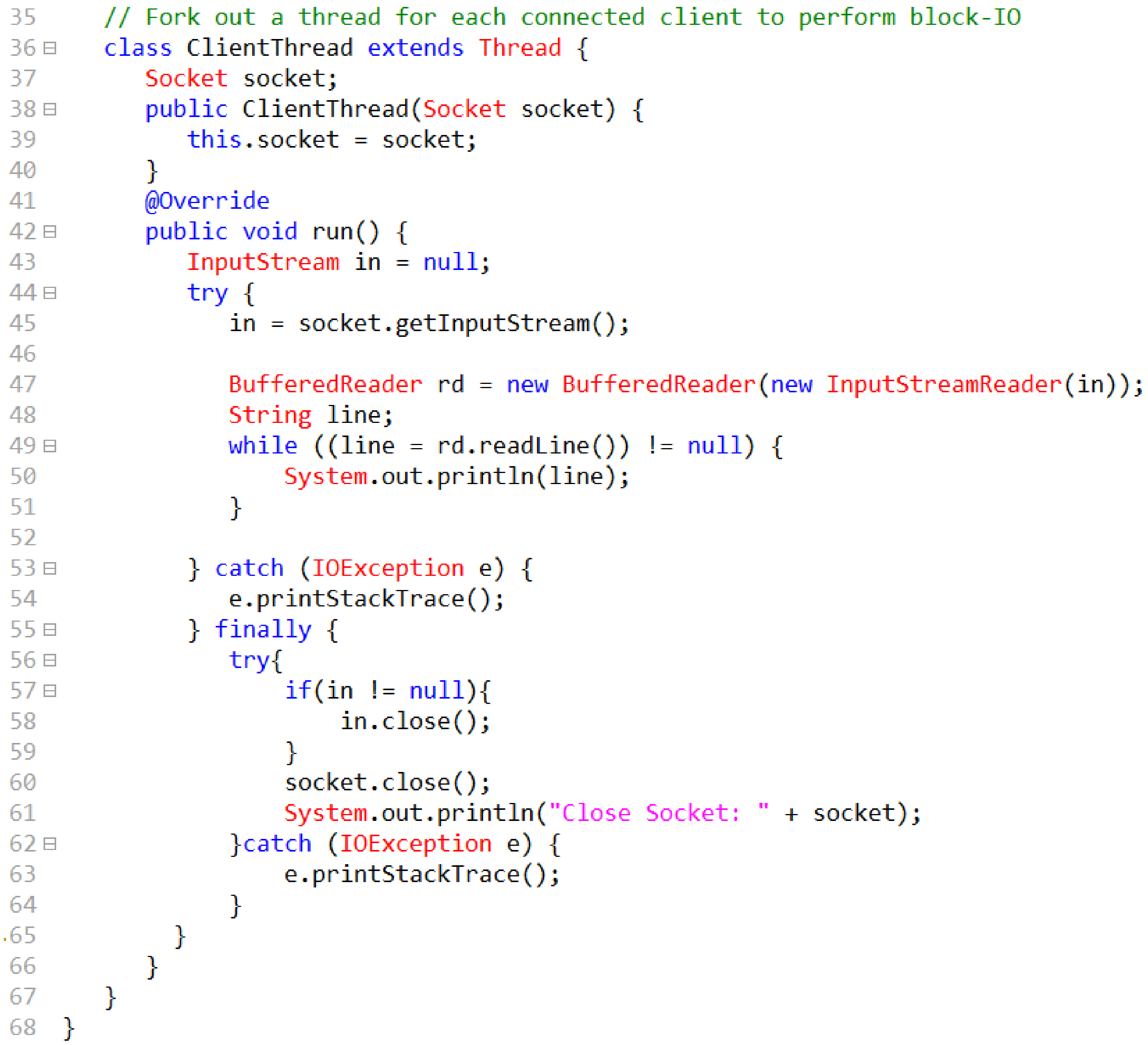
Accepting Connections

- Usually, the `accept()` method is executed within an infinite loop
  - i.e., `while(true){...}`
- The accept method returns a new socket (with a new port) for the new channel. It blocks until connection is made
- Whenever `accept()` returns, a new *thread* is launched to handle that interaction (not in our example)
- Hence, the server can handle several requests concurrently

Threaded Server Example



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

public class SimpleThreadedSocketListener {
   ServerSocket server;
   int serverPort = 8888;
   // Constructor to allocate a ServerSocket listening at the given port.
   public SimpleThreadedSocketListener() {
      try {
         server = new ServerSocket(serverPort);
         System.out.println("ServerSocket: " + server);
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
   // Start listening.
   private void listen() {
      while (true) { // run until you terminate the program
         try {
            // Wait for connection. Block until a connection is made.
            Socket socket = server.accept();
            System.out.println("Socket: " + socket);
            // Start a new thread for each client to perform block-IO operations.
            new ClientThread(socket).start();
         } catch (IOException e) {
            e.printStackTrace();
         }
      }
   }
   public static void main(String[] args) {
      new SimpleThreadedSocketListener().listen();
   }
```

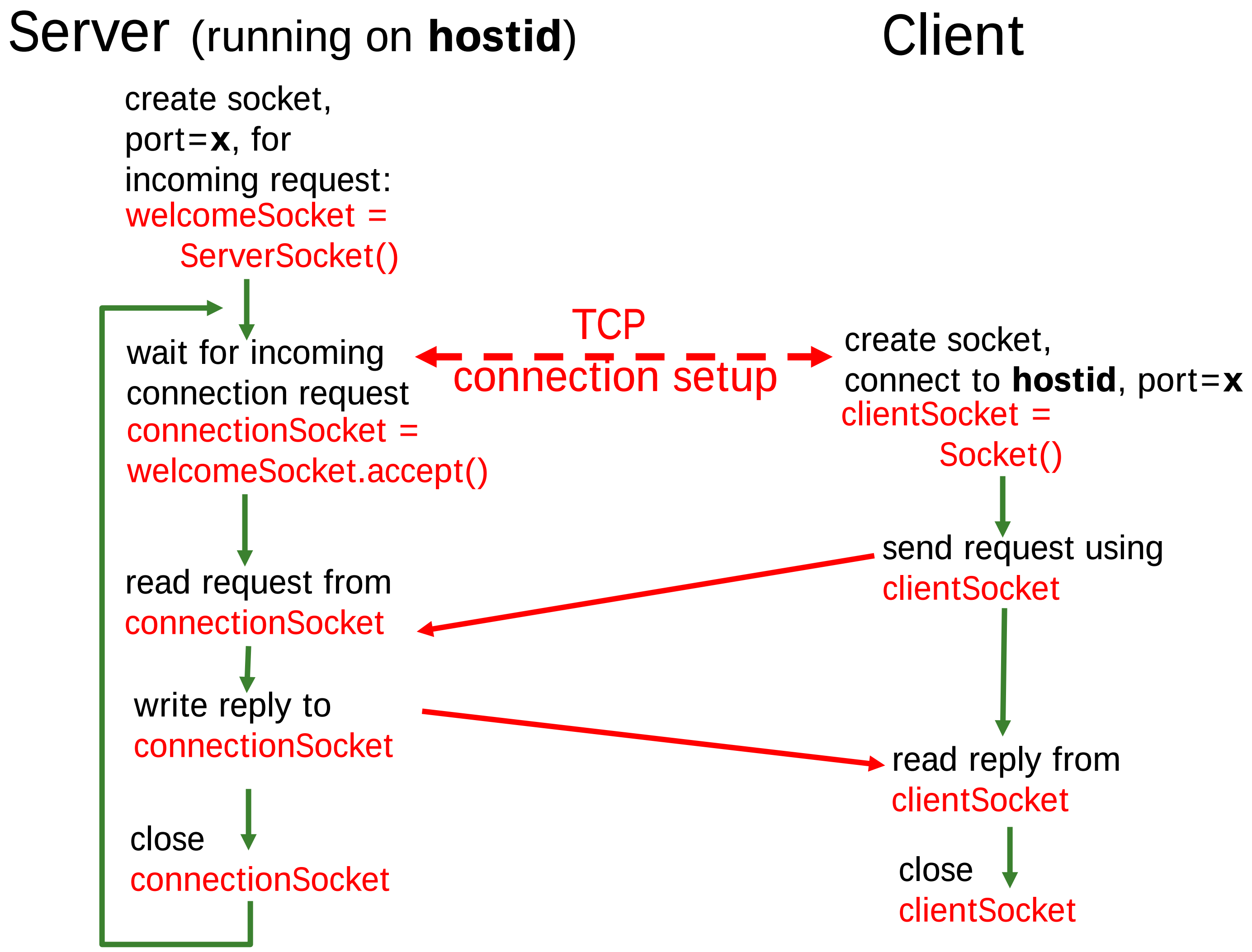


```
   // Fork out a thread for each connected client to perform block-IO
   class ClientThread extends Thread {
      Socket socket;
      public ClientThread(Socket socket) {
         this.socket = socket;
      }
      @Override
      public void run() {
         InputStream in = null;
         try {
            in = socket.getInputStream();

            BufferedReader rd = new BufferedReader(new InputStreamReader(in));
            String line;
            while ((line = rd.readLine()) != null) {
                System.out.println(line);
            }

         } catch (IOException e) {
            e.printStackTrace();
         } finally {
            try{
                if(in != null){
                    in.close();
                }
                socket.close();
                System.out.println("Close Socket: " + socket);
            }catch (IOException e) {
                e.printStackTrace();
            }
        }
      }
   }
}
```

Timeout

- You can set timeout values to the blocking `accept()` method of `ServerSocket`
- Use the method:
  `serverSocket.setSoTimeout(milliseconds)`
- If timeout is reached before the method returns, `java.net.SocketTimeoutException` is thrown.

Client/server socket interaction: TCP



Server (running on **hostid**)

Client

create socket,
port=**x**, for
incoming request:
welcomeSocket =
ServerSocket()

wait for incoming
connection request
connectionSocket =
welcomeSocket.accept()

TCP
connection setup

create socket,
connect to **hostid**, port=**x**
clientSocket =
Socket()

send request using
clientSocket

read request from
connectionSocket

write reply to
connectionSocket

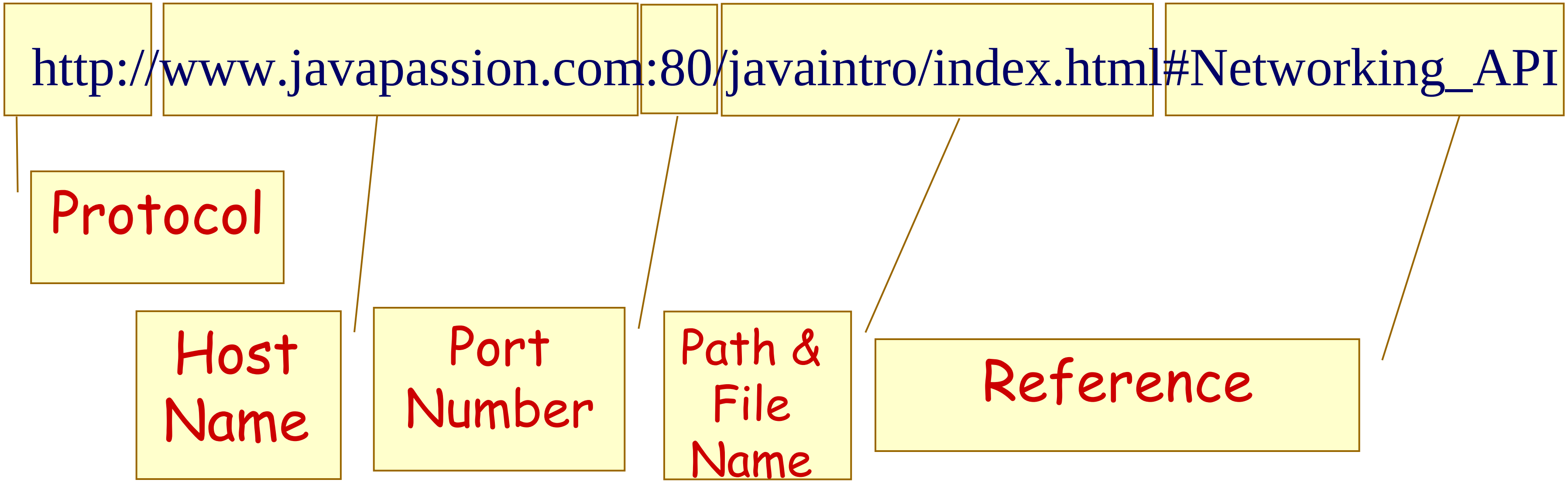
read reply from
clientSocket

close
connectionSocket

close
clientSocket

URL - *Uniform Resource Locator*

- URL is a reference (an address) to a resource on the Internet.
  - A resource can be a file, a database query and more.
- URLs are just a subset of the more general concept of Uniform Resource Identifiers (URIs) which are meant to describe all points in the information space



http://www.javapassion.com:80/javaintro/index.html#Networking_API

Protocol

Host Name

Port Number

Path & File Name

Reference

Class URL

- Class URL represents a Uniform Resource Locator, a pointer to a "resource" on the World Wide Web.
- We distinguish between:
  - Absolute URL - contains all of the information necessary to reach the resource.
  - Relative URL - contains only enough information to reach the resource relative to (or in the context of) another URL.

Class URL Cont.

- Constructing URLs:
  - `URL w3c1 = new URL("http://www.w3.org/TR/");`
  - `URL w3c2 = new URL("http","www.w3.org",80,"TR/");`
  - `URL w3c3 = new URL(w3c2, "xhtml1/");`
    - If the string is not an absolute URL, then it is considered relative to the URL
    - More constructors can be found in the URL API

URL addresses with Special characters

- Some URL addresses also contain these special characters, for example the space character.
  - Like this: http://foo.com/hello world/
- To make theses characters legal they need to be encoded before passing them to the URL constructor.

```
URL url = new URL("http://foo.com/hello%20world");
```

- One class that can help us with this is the `URI` class :

```
URI uri = new URI("http", "foo.com", "/hello world/", "");
URL url = uri.toURL();
```

- Another one is the `URLEncoder` class…

URLEncoder

- Contains a utility method encode for converting a string into an encoded format (used in URLs)
- To convert a string, each character is examined in turn:
  - Space is converted into a plus sign `+`
  - a-z, A-Z, 0-9, ., -, * and _ remain the same.
  - The bytes of all special characters are replaced by hexadecimal numbers, preceded with %
- To decode an encoded string, use `decode()` of the class `URLDecoder`
- The URLEncoder API.

URL Encoding

- URL Encoding is the process of converting string into valid URL format.
- Valid URL format means that the URL contains only what is termed "alpha | digit | safe | extra | escape" characters.
  - You can read more about the what and the whys of these terms on the World Wide Web Consortium site: http://www.w3.org/Addressing/URL/url-spec.html
  - URL Encoding Reference

URL Encoding Cont.

- URL encoding is normally performed to convert data passed via html forms, because such data may contain special character, such as "/", ".", "#", and so on, which could either:
  - Have special meanings
  - Is not a valid character for an URL
- For instance, the "#" character needs to be encoded because it has a special meaning of that of an html anchor. The <space> character also needs to be encoded because is not allowed on a valid URL format. Also, some characters, such as "~" might not transport properly across the internet.

URL Encoding Cont.

- Example: The URL encoding of “`This is a simple & short test`” is “`This+is+a+simple+%26+short+test` “
- Note that because the <space> character is very commonly used, a special code (the "+" sign) has been reserved as its URL encoding

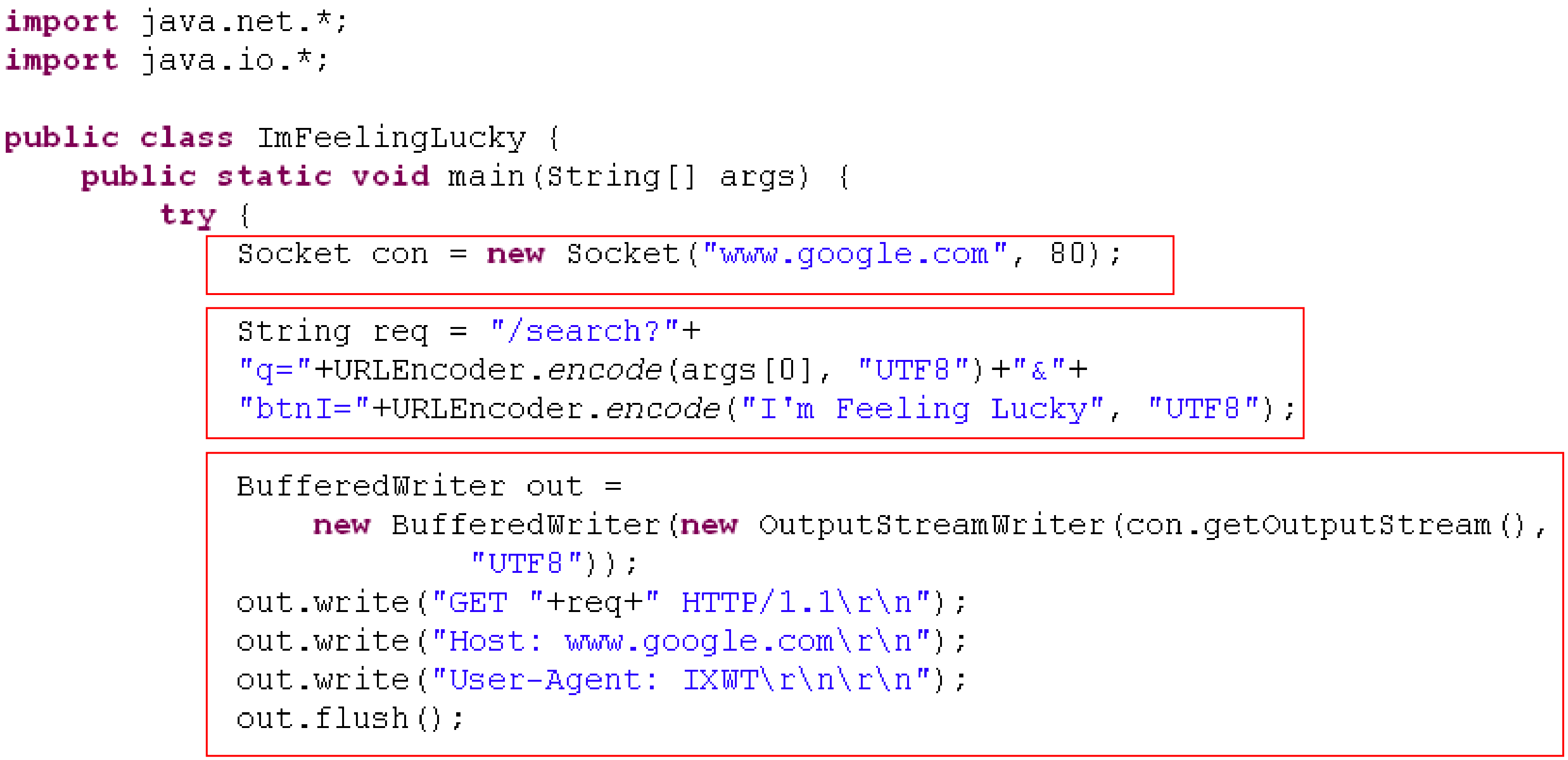
MalformedURLException

- URL constructors throws a `MalformedURLException` if the arguments to the constructor refer to a null or unknown protocol. Typically, you want to catch and handle this exception by embedding your URL constructor statements in a try/catch pair, like this:

```
try
      { URL myURL = new URL(. . .) }
catch(MalformedURLException e)
      { ... // exception handler code here ... }
```

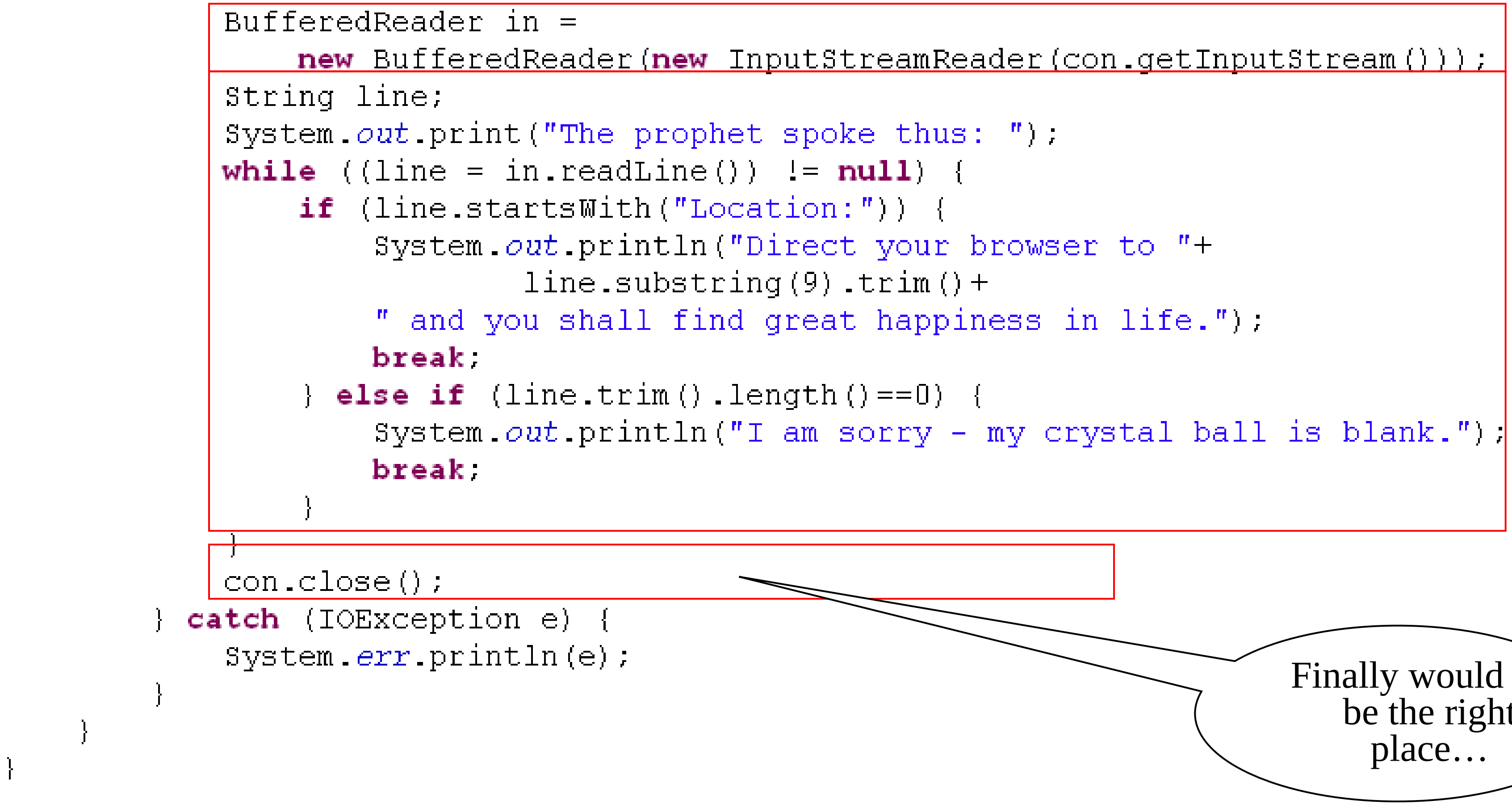
ImFeelingLucky Example

- The following program sends a request to the Google server and extracts the result.
- Google search engine accepts GET requests of a specific format. It's reply always contain a `Location` header line if the search is successful.



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

public class ImFeelingLucky {
    public static void main(String[] args) {
        try {
            Socket con = new Socket("www.google.com", 80);

            String req = "/search?"+
            "q="+URLEncoder.encode(args[0], "UTF8")+"&"+
            "btnI="+URLEncoder.encode("I'm Feeling Lucky", "UTF8");

            BufferedWriter out =
                new BufferedWriter(new OutputStreamWriter(con.getOutputStream(),
                        "UTF8"));
            out.write("GET "+req+" HTTP/1.1\r\n");
            out.write("Host: www.google.com\r\n");
            out.write("User-Agent: IXWT\r\n\r\n");
            out.flush();
```

ImFeelingLucky Example Cont.



```
            BufferedReader in =
                new BufferedReader(new InputStreamReader(con.getInputStream()));
            String line;
            System.out.print("The prophet spoke thus: ");
            while ((line = in.readLine()) != null) {
                if (line.startsWith("Location:")) {
                    System.out.println("Direct your browser to "+
                            line.substring(9).trim()+
                    " and you shall find great happiness in life.");
                    break;
                } else if (line.trim().length()==0) {
                    System.out.println("I am sorry - my crystal ball is blank.");
                    break;
                }
            }
            con.close();
        } catch (IOException e) {
            System.err.println(e);
        }
    }
}
```

Finally would be the right place…

Parsing a URL

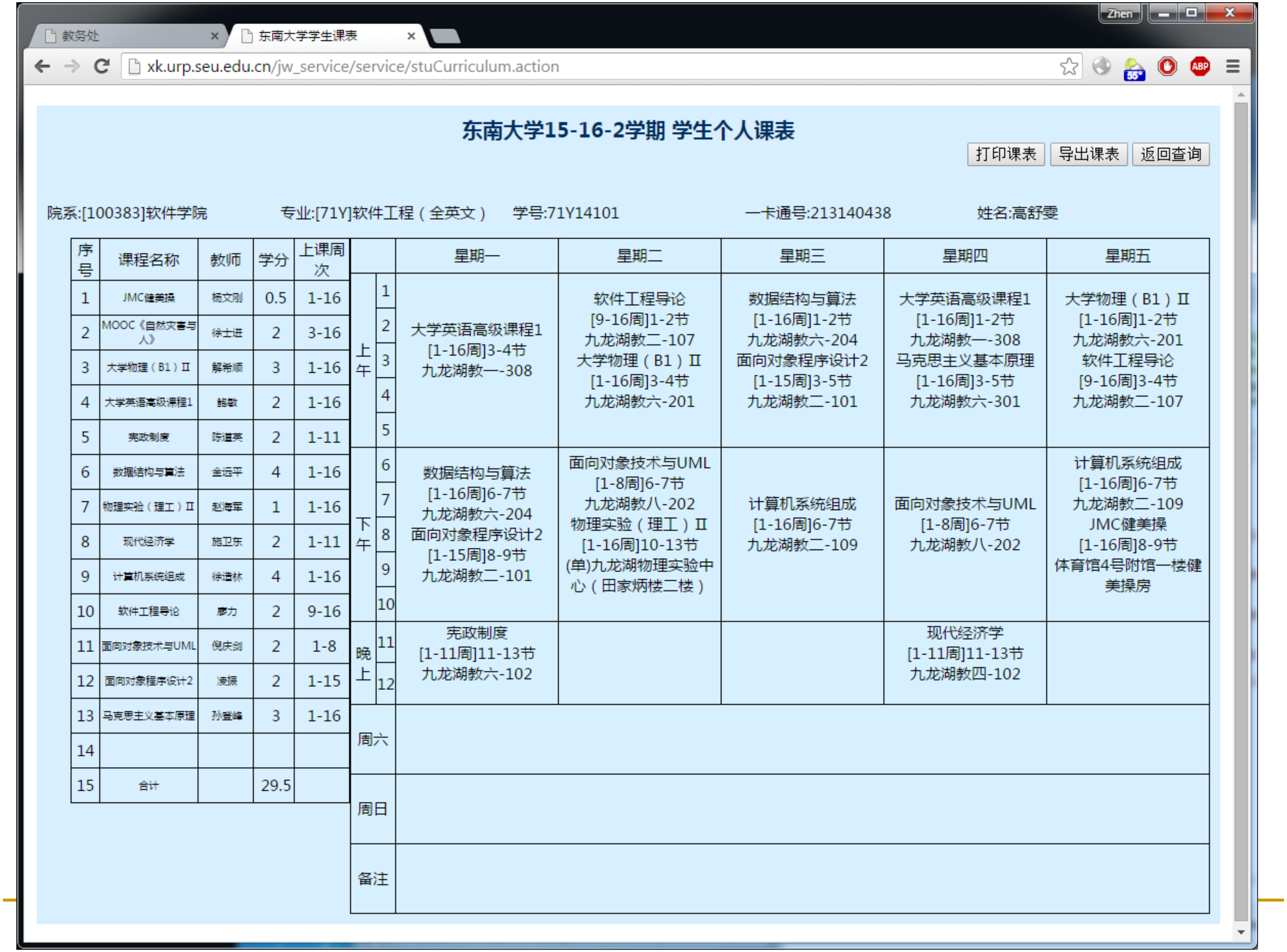
- The following methods of `URL` can be used for parsing URLs:

```
getProtocol(), getHost(),
getPort(), getPath(), getFile(),
getQuery(), getRef()
```

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

public class ParseURL {
    public static void main(String[] args) throws Exception {
        URL aURL = new URL("http://java.sun.com:80/docs/books/tutorial"
                           + "/index.html?name=networking#DOWNLOADING");
        System.out.println("protocol = " + aURL.getProtocol());
    System.out.println("authority = " + aURL.getAuthority());
        System.out.println("host = " + aURL.getHost());
        System.out.println("port = " + aURL.getPort());
        System.out.println("path = " + aURL.getPath());
        System.out.println("query = " + aURL.getQuery());
        System.out.println("filename = " + aURL.getFile());
        System.out.println("ref = " + aURL.getRef());
    }
}
```

The Output

```
protocol = http
authority = java.sun.com:80
host = java.sun.com
port = 80
path = /docs/books/tutorial/index.html
query = name=networking
filename = /docs/books/tutorial/index.html?name=networking
ref = DOWNLOADING
```

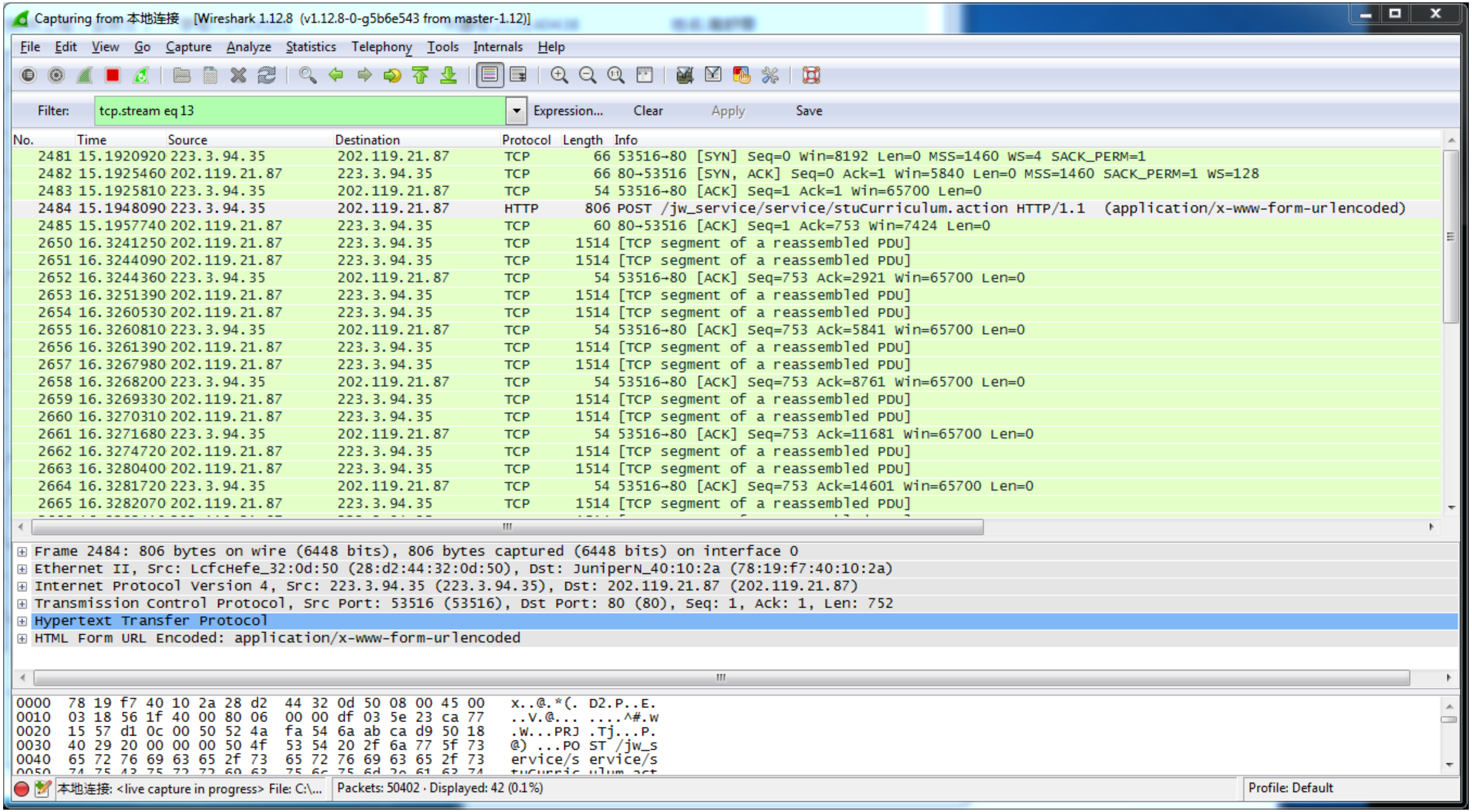


东南大学15-16-2学期 学生个人课表

院系:[100383]软件学院　专业:[71Y]软件工程（全英文）　学号:71Y14101　一卡通号:213140438　姓名:高舒雯

序号	课程名称	教师	学分	上课周次
1	JMC健美操	杨文刚	0.5	1-16
2	MOOC《自然灾害与人》	徐士进	2	3-16
3	大学物理（B1）Ⅱ	解希顺	3	1-16
4	大学英语高级课程1	鲍敏	2	1-16
5	宪政制度	陈道英	2	1-11
6	数据结构与算法	金远平	4	1-16
7	物理实验（理工）Ⅱ	赵海军	1	1-16
8	现代经济学	施卫东	2	1-11
9	计算机系统组成	徐造林	4	1-16
10	软件工程导论	廖力	2	9-16
11	面向对象技术与UML	倪庆剑	2	1-8
12	面向对象程序设计2	凌振	2	1-15
13	马克思主义基本原理	孙登峰	3	1-16
14				
15	合计		29.5	

		星期一	星期二	星期三	星期四	星期五
上午	1	大学英语高级课程1 [1-16周]3-4节 九龙湖教一-308	软件工程导论 [9-16周]1-2节 九龙湖教二-107 大学物理（B1）Ⅱ [1-16周]3-4节 九龙湖教六-201	数据结构与算法 [1-16周]1-2节 九龙湖教六-204 面向对象程序设计2 [1-15周]3-5节 九龙湖教二-101	大学英语高级课程1 [1-16周]1-2节 九龙湖教一-308 马克思主义基本原理 [1-16周]3-5节 九龙湖教六-301	大学物理（B1）Ⅱ [1-16周]1-2节 九龙湖教六-201 软件工程导论 [9-16周]3-4节 九龙湖教二-107
	2					
	3					
	4					
	5					
下午	6	数据结构与算法 [1-16周]6-7节 九龙湖教六-204 面向对象程序设计2 [1-15周]8-9节 九龙湖教二-101	面向对象技术与UML [1-8周]6-7节 九龙湖教八-202 物理实验（理工）Ⅱ [1-16周]10-13节 (单)九龙湖物理实验中心（田家炳楼二楼）	计算机系统组成 [1-16周]6-7节 九龙湖教二-109	面向对象技术与UML [1-8周]6-7节 九龙湖教八-202	计算机系统组成 [1-16周]6-7节 九龙湖教二-109 JMC健美操 [1-16周]8-9节 体育馆4号附馆一楼健美操房
	7					
	8					
	9					
	10					
晚上	11	宪政制度 [1-11周]11-13节 九龙湖教六-102			现代经济学 [1-11周]11-13节 九龙湖教四-102	
	12					
周六						
周日						
备注						



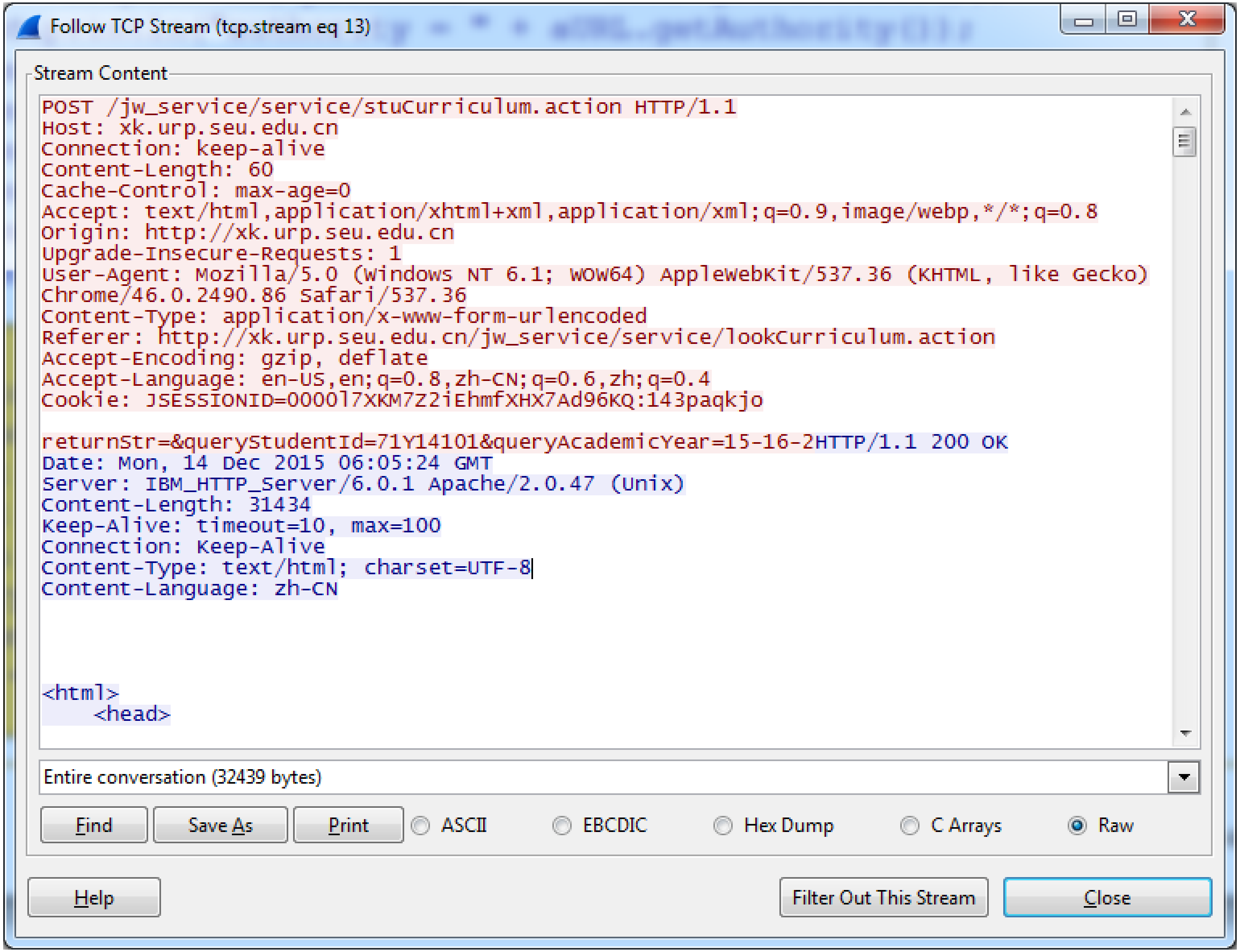
Filter: tcp.stream eq 13

```
No.    Time         Source          Destination     Protocol Length Info
  2481 15.1920920 223.3.94.35     202.119.21.87   TCP       66 53516→80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
  2482 15.1925460 202.119.21.87   223.3.94.35     TCP       66 80→53516 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 SACK_PERM=1 WS=128
  2483 15.1925810 223.3.94.35     202.119.21.87   TCP       54 53516→80 [ACK] Seq=1 Ack=1 Win=65700 Len=0
  2484 15.1948090 223.3.94.35     202.119.21.87   HTTP     806 POST /jw_service/service/stuCurriculum.action HTTP/1.1  (application/x-www-form-urlencoded)
  2485 15.1957740 202.119.21.87   223.3.94.35     TCP       60 80→53516 [ACK] Seq=1 Ack=753 Win=7424 Len=0
  2650 16.3241250 202.119.21.87   223.3.94.35     TCP     1514 [TCP segment of a reassembled PDU]
  2651 16.3244090 202.119.21.87   223.3.94.35     TCP     1514 [TCP segment of a reassembled PDU]
  2652 16.3244360 223.3.94.35     202.119.21.87   TCP       54 53516→80 [ACK] Seq=753 Ack=2921 Win=65700 Len=0
  2653 16.3251390 202.119.21.87   223.3.94.35     TCP     1514 [TCP segment of a reassembled PDU]
  2654 16.3260530 202.119.21.87   223.3.94.35     TCP     1514 [TCP segment of a reassembled PDU]
  2655 16.3260810 223.3.94.35     202.119.21.87   TCP       54 53516→80 [ACK] Seq=753 Ack=5841 Win=65700 Len=0
  2656 16.3261390 202.119.21.87   223.3.94.35     TCP     1514 [TCP segment of a reassembled PDU]
  2657 16.3267980 202.119.21.87   223.3.94.35     TCP     1514 [TCP segment of a reassembled PDU]
  2658 16.3268200 223.3.94.35     202.119.21.87   TCP       54 53516→80 [ACK] Seq=753 Ack=8761 Win=65700 Len=0
  2659 16.3269330 202.119.21.87   223.3.94.35     TCP     1514 [TCP segment of a reassembled PDU]
  2660 16.3270310 202.119.21.87   223.3.94.35     TCP     1514 [TCP segment of a reassembled PDU]
  2661 16.3271680 223.3.94.35     202.119.21.87   TCP       54 53516→80 [ACK] Seq=753 Ack=11681 Win=65700 Len=0
  2662 16.3274720 202.119.21.87   223.3.94.35     TCP     1514 [TCP segment of a reassembled PDU]
  2663 16.3280400 202.119.21.87   223.3.94.35     TCP     1514 [TCP segment of a reassembled PDU]
  2664 16.3281720 223.3.94.35     202.119.21.87   TCP       54 53516→80 [ACK] Seq=753 Ack=14601 Win=65700 Len=0
  2665 16.3282070 202.119.21.87   223.3.94.35     TCP     1514 [TCP segment of a reassembled PDU]
```

```
⊞ Frame 2484: 806 bytes on wire (6448 bits), 806 bytes captured (6448 bits) on interface 0
⊞ Ethernet II, Src: LcfcHefe_32:0d:50 (28:d2:44:32:0d:50), Dst: JuniperN_40:10:2a (78:19:f7:40:10:2a)
⊞ Internet Protocol Version 4, Src: 223.3.94.35 (223.3.94.35), Dst: 202.119.21.87 (202.119.21.87)
⊞ Transmission Control Protocol, Src Port: 53516 (53516), Dst Port: 80 (80), Seq: 1, Ack: 1, Len: 752
⊞ Hypertext Transfer Protocol
⊞ HTML Form URL Encoded: application/x-www-form-urlencoded
```

```
0000  78 19 f7 40 10 2a 28 d2  44 32 0d 50 08 00 45 00   x..@.*(. D2.P..E.
0010  03 18 56 1f 40 00 80 06  00 00 df 03 5e 23 ca 77   ..V.@... ....^#.w
0020  15 57 d1 0c 00 50 52 4a  fa 54 6a ab ca d9 50 18   .W...PRJ .Tj...P.
0030  40 29 20 00 00 00 50 4f  53 54 20 2f 6a 77 5f 73   @) ...PO ST /jw_s
0040  65 72 76 69 63 65 2f 73  65 72 76 69 63 65 2f 73   ervice/s ervice/s
0050  74 75 43 75 72 72 69 63  75 6c 75 6d 2e 61 63 74   tuCurric ulum.act
```

本地连接: <live capture in progress> File: C:\...   Packets: 50402 · Displayed: 42 (0.1%)



```
POST /jw_service/service/stuCurriculum.action HTTP/1.1
Host: xk.urp.seu.edu.cn
Connection: keep-alive
Content-Length: 60
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Origin: http://xk.urp.seu.edu.cn
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/46.0.2490.86 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Referer: http://xk.urp.seu.edu.cn/jw_service/service/lookCurriculum.action
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.8,zh-CN;q=0.6,zh;q=0.4
Cookie: JSESSIONID=0000l7XKM7Z2iEhmfXHX7Ad96KQ:143paqkjo

returnStr=&queryStudentId=71Y14101&queryAcademicYear=15-16-2HTTP/1.1 200 OK
Date: Mon, 14 Dec 2015 06:05:24 GMT
Server: IBM_HTTP_Server/6.0.1 Apache/2.0.47 (Unix)
Content-Length: 31434
Keep-Alive: timeout=10, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
Content-Language: zh-CN




<html>
    <head>
```

Entire conversation (32439 bytes)

○ ASCII ○ EBCDIC ○ Hex Dump ○ C Arrays ● Raw

URLConnection

- Represent a communications 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.
- Creating a connection to a URL:
  - The connection object is created by invoking the `openConnection` method on a URL. If the protocol of the URL is HTTP, the returned object is of class `HttpURLConnection`.
  - The setup parameters and general request properties are manipulated.
  - The actual connection to the remote object is made, using the `connect` method.
  - The remote object becomes available. The header fields and the contents of the remote object can be accessed.
- See the `URLConnection` class API for more information

URLConnection Cont.

- The life cycle of a `URLConnection` object has two parts:
  - Before actual connection establishment
    - Connection configuration (`setAllowUserInteraction`, `setDoInput` and more)
  - After actual connection establishment
    - Content retrieval
- Moving from the first phase to the second is implicit
  - A result of calling some committing methods, like `getDate()`

URLConnection Example

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

public class URLConnectionReader {
    public static void main(String[] args) throws Exception {
        URL yahoo = new URL("http://www.yahoo.com/");
        URLConnection yc = yahoo.openConnection();
        BufferedReader in = new BufferedReader(
                                new InputStreamReader(
                                yc.getInputStream()));
        String inputLine;

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

Class `HttpURLConnection`

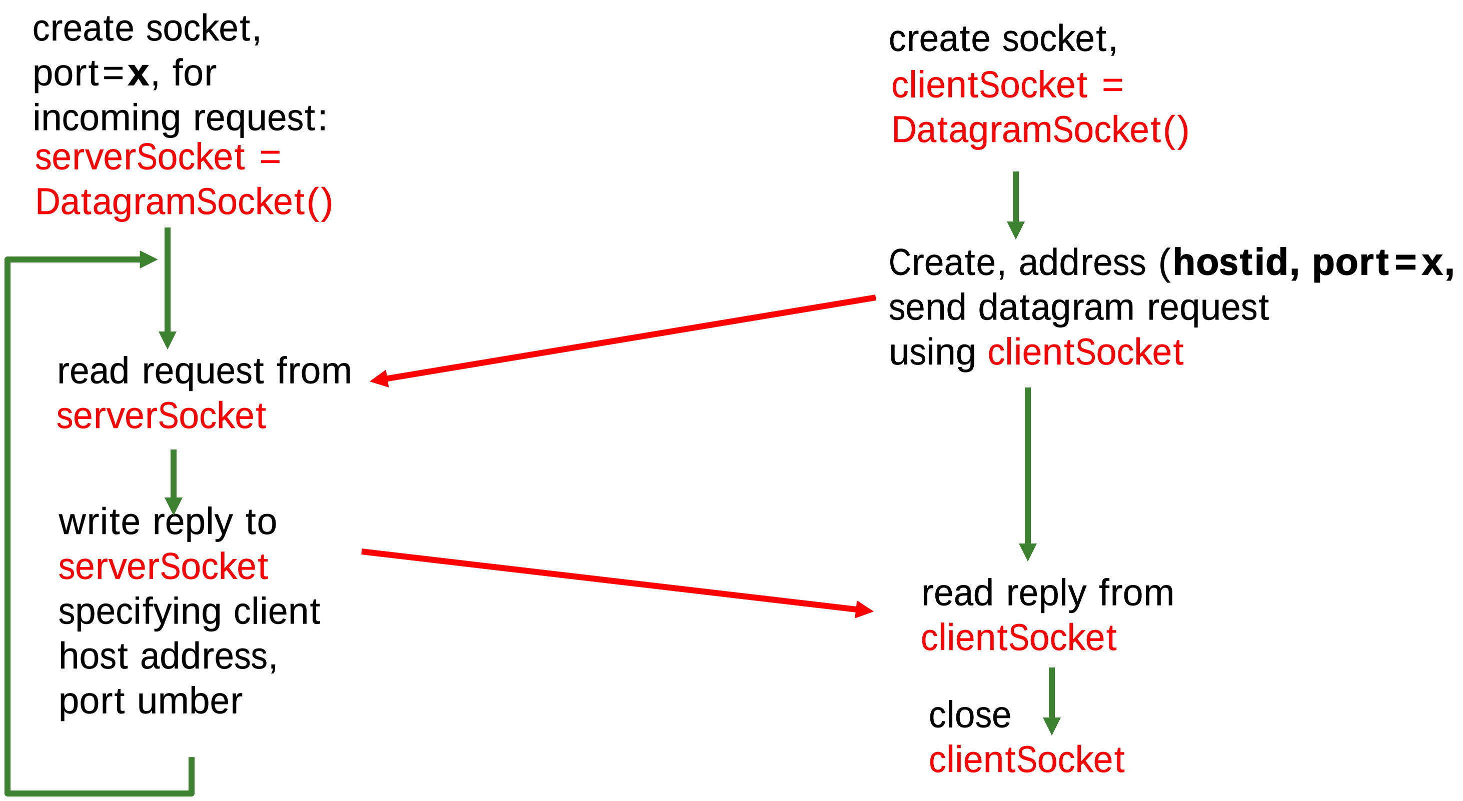
- A `URLConnection` with support for HTTP-specific features.
  - `responseMessage` field
  - `getRequestMethod()`
  - `usingProxy()`

- The HttpURLConnection API

Datagrams

- A *datagram* is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed.
- The `java.net` package contains three classes to help you write Java programs that use datagrams to send and receive packets over the network: `DatagramSocket`, `DatagramPacket`, and `MulticastSocket`

Client/server socket interaction: UDP



Server (running on **hostid**)

create socket,
port=**x**, for
incoming request:
serverSocket =
DatagramSocket()

read request from
serverSocket

write reply to
serverSocket
specifying client
host address,
port umber

Client

create socket,
clientSocket =
DatagramSocket()

Create, address (**hostid, port=x,**
send datagram request
using clientSocket

read reply from
clientSocket

close
clientSocket