320341 Programming in Java

JACOBS
UNIVERSITY

Fall Semester 2014

Lecture 12: Introduction to Network Programming

Instructor: Jürgen Schönwälder

Slides: Bendick Mahleko

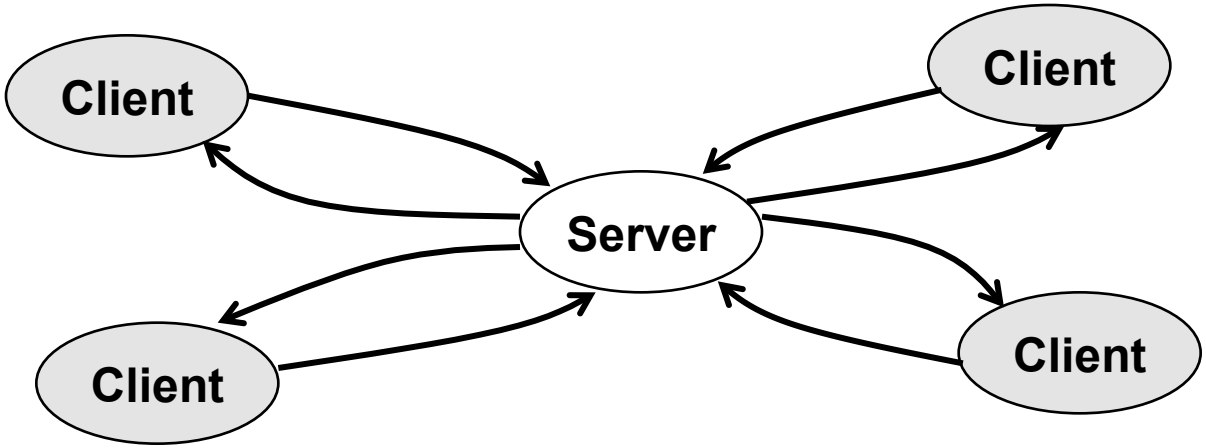
Objectives

JACOBS UNIVERSITY

This lecture introduces the following

- Basic networking concepts
- Identifying a machine
- Connecting to a server
- Implementing servers
- Sending e-mail
- Making URL connections

Basic Concepts: Client/ Server Model



- Clients request services from servers
- Synchronous: clients wait for the response before they proceed with their computation
- Asynchronous: clients proceed with computations as the response is returned by server

Basic Concepts: Client/ Server Model

JACOBS UNIVERSITY

Allows bilateral information exchange between nodes (computers)

- One acts as a **server**, another as a **client**

The **server** provides a specific service, for example

- **Web server:** serves up web pages (the *web browser* is the **client**)

- **FTP server:** serves up files (downloading via file transfer protocol)

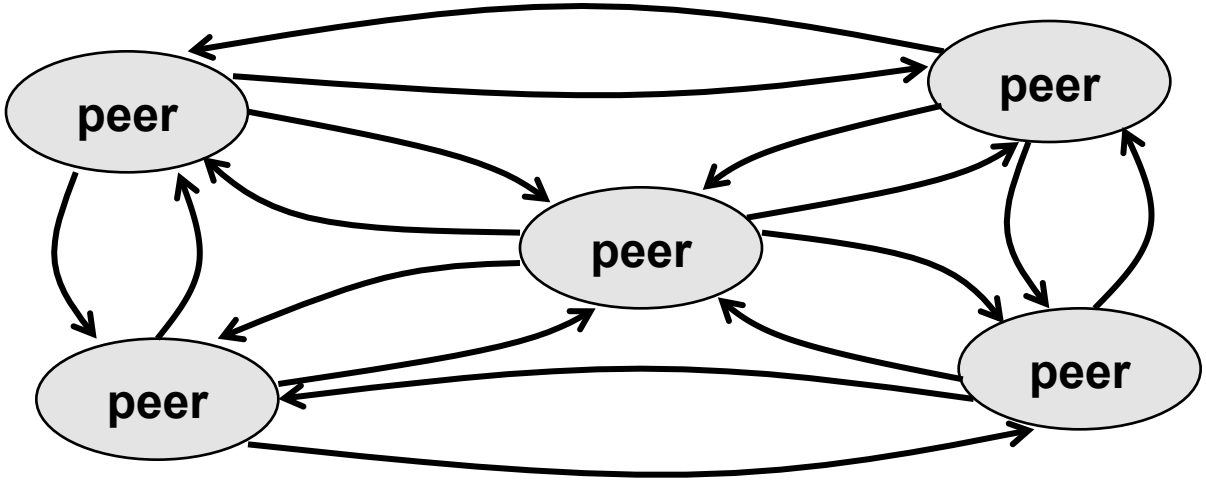
Basic Concepts: Client/ Server Model

JACOBS UNIVERSITY

Clients connect to the server to access the service

- Clients usually initiate dialog with the server
- The server "**waits**" and "**listens**" for client connections
- The machine on which server software runs is usually called the **host machine**

Basic Concepts: Peer-to-Peer (P2P)



- Every **peer** provides **client** and **server** functionality
- Ideally avoids centralized components
- Able to establish new (**overlay**) **topologies** dynamically
- Requires **control** and **coordination logic** on each node

Basic Concepts: Ports and Sockets

JACOBS UNIVERSITY

Ports and sockets are abstract concepts only and allow the programmer to make use of communication links

Port: a logical connection to a computer that's identified by a 2-byte number, thus has range 0 – 65,535

Sockets: software abstraction used to represent the "terminals" of a connection between two machines

Basic Concepts: Ports and Sockets

JACOBS UNIVERSITY

Port are classified into 3 categories:

- 0 – 1,023 are well-known ports (e.g., SMTP: 25, HTTP: 80, Telnet: 23)
- 1,024 – 49,151 are not assigned; however their use must be registered to avoid duplication
- 49,152 – 65,535 are neither assigned not registered. They are so called dynamic range and can be used by any process

For each port supplying a service, there is a server program waiting for requests

Examples

- smtp	25/tcp	Simple Mail Transfer
- smtp	25/udp	Simple Mail Transfer
- ftp	21/tcp	File Transfer [Control]
- ftp	21/udp	File Transfer [Control]
- http	80/tcp	World Wide Web HTTP
- http	80/udp	World Wide Web HTTP

.
:

Retrieved from: http://www.iana.org/assignments/port-numbers
(Last visited: 14 November 2013)

Sockets: Basic Concepts

JACOBS UNIVERSITY

- You can imagine a hypothetical “cable” running between the two machines with each end of the "cable" plugged into a socket

- The host identifier (IP address) and process identifier (port number) taken together form a socket address or simply socket

- When a client wishes to make a connection to a server, it will create a socket at its end of the communication link

- The corresponding server creates a new socket at its end that will be dedicated to communication with the particular client

Basic Concepts: Internet & IP Addresses

JACOBS UNIVERSITY

Represent machine addresses in **quad notation**

- Addresses are made up of **4 8-bit numbers**, separated by dots
  - ❑ Numbers are in the decimal range 0 - 255
  - ❑ Example: 131.122.3.219

} **IPv4**

- **IPv6** replaces **IPv4**
  - ❑ IPv6 uses **128-bit** numbers
  - ❑ Provides massively more addresses than is currently possible

Java was conceived with features designed specifically for network programming

- The features are provided in a platform-independent manner

Java Network Programming Overview

JACOBS UNIVERSITY

Java provides a rich library to support network programming

Networking abstraction

- Networking details have been abstracted away from the programmer

Handling multiple connections

- Java's built-in **multithreading** for handling multiple connections concurrently

Java Network Programming Overview

JACOBS UNIVERSITY

Networking programming model

- The programming model used is that of a file (“remote files”)
- Wrap the network connections (“sockets”) with stream objects
- Then use the same method calls as used with all other streams

Identifying a Machine

JACOBS UNIVERSITY

Machines are uniquely identified by IP (Internet Protocol) addresses

IP address object

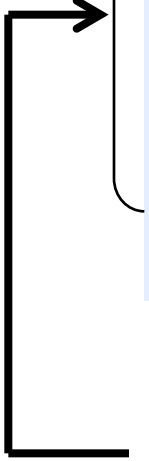
- Need to get an stream object from the IP address
- Use the static method ***InetAddress***`.getByName()` to get an object representing the IP address (package: **`java.net`**)

- The IP address is represented by an object of type **InetAddress**

Example

Finding your address

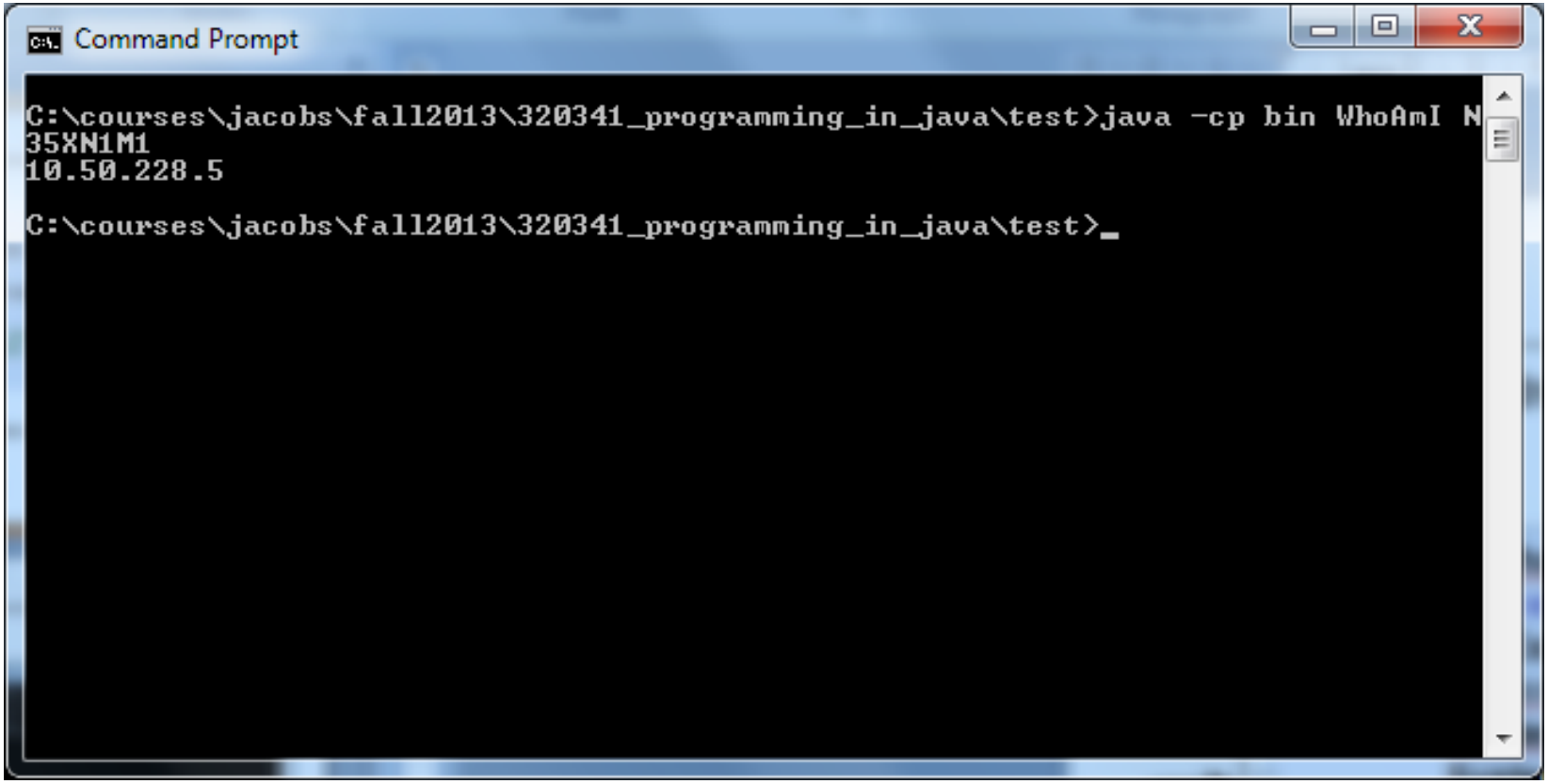
```
public class WhoAmI {
    public static void main(String[] args)
        throws Exception {
        if (args.length != 1) {
            System.err.println("Usage: WhoAmI  MachineName");
            System.exit(1);
        }
        java.net.InetAddress address = java.net.InetAddress.getByName(args[0]);
        System.out.println(address. getHostAddress());
    }
}
```



Finds out your network address when you're connected to the Internet

Example

Finding your address



Sockets in Java

JACOBS UNIVERSITY

Socket objects

1. Create a **Socket** to connect to the other machine
2. Get back an **InputStream** and **OutputStream** from the socket

- **InputStream** & **OutputStream** :

  - This allows us to treat the connections as **I/O stream** objects

Sockets in Java

JACOBS UNIVERSITY

There are 2 main stream-based socket classes (`java.net` package)

1. **Socket** – used by the client to initiate a connection

2. **ServerSocket** – used by the server to *listen to incoming connections*

Sockets in Java

JACOBS UNIVERSITY

ServerSocket

- Creates a physical “server” or listening socket on the host machine
- Returns an established socket via the `accept()` method

Socket

- Use to initiate a client connection
- The constructor requires an IP address & port number of the remote machine to connect to

Making a Connection: Server

Choose a port outside of the range 0-1023

```
public class JabberServer {
  public static void main(String[] args)  throws IOException {
   ServerSocket s = new ServerSocket(PORT); // public static final int PORT = 8080;
   try {
    Socket socket = s.accept();
    try {
       BufferedReader in = new BufferedReader(
                   new InputStreamReader(socket.getInputStream()));
       PrintWriter out = new PrintWriter(new BufferedWriter(
                new OutputStreamWriter(socket.getOutputStream())), true);
      while (true) {
          String str = in.readLine();
          if (str.equals("END")) break;
              out.println(str);
      }
    } finally {
         socket.close();
    }
   } finally {
     s.close();
   }
  }
}
```

Blocks until a connection occurs

Connection → I/O object

Always close the two sockets

Making a Connection: Client

```
public class JabberClient {
  public static void main(String[] args) throws IOException {
     InetAddress addr =  InetAddress.getByName("localhost ");
     Socket socket = new Socket(addr, JabberServer.PORT);
     try {
         BufferedReader in = new BufferedReader(
                                    new InputStreamReader(socket.getInputStream()));
         PrintWriter out = new PrintWriter(new BufferedWriter(
                  new OutputStreamWriter(socket.getOutputStream())), true);
         for (int i = 0; i < 10; i ++) {
              out.println("howdy " + i);
              String str = in.readLine(); System.out.println(str);
         }
         out.println("END");
      } finally {
           socket.close();
      }
  }
}
```

Local machine

Guard everything in a try-finally to make sure that the socket is closed

Making a Connection: Client-Server

JACOBS UNIVERSITY

An Internet connection is uniquely determined by four pieces of data:

1. ClientHost (e.g., `127.0.0.1` also the `localhost`)
2. ClientPortNumber (Allocated the next available port on its machine)
3. ServerHost (e.g., `127.0.0.1` or the `localhost`)
4. ServerPortNumber (`8080`)

Making a Connection: Client-Server

JACOBS UNIVERSITY

How to Exchange Data?

- During connection setup, the client sends a “return address” to the server
- Both the client and server know where to send data during data exchange
- Sockets produce a “dedicated” connection that persists until it is explicitly disconnected
- The dedicated connection can be disconnected inexplicitly if one side, or an intermediary link of the connection crashes

Making a Connection: Client-Server

JACOBS UNIVERSITY

Server Side

```
System.out.println("Connection accepted: "+ socket);
```

```
Connection accepted: Socket[addr=/127.0.0.1,port=1047,localport=8080]
```

The server accepted a connection from 127.0.0.1 on port 1047 while listening on its local port (8080)

Client Side

```
System.out.println("socket = " + socket);
```

```
socket = Socket[addr=localhost/127.0.0.1,port=8080,localport=1047]
```

The client made a connection to 127.0.0.1 on port 8080 using the local port 1047

Serving Multiple Clients

A server supports multiple clients simultaneously using **multithreading**

Basic Approach

- Make a single **ServerSocket** in the server
- Call the `accept()` method to wait for a new connection
- When `accept()` returns, take the resulting **Socket** object and use it to create a new thread:
  - *The new thread serves a particular client*
- Call the `accept()` method again to wait for a new client

Key principle:

The operations to serve a particular client are moved inside a thread

Serving Multiple Clients

JACOBS UNIVERSITY

Example

```
public class MultiJabberServer {
    static final int PORT = 8080;
    public static void main(String[] args)  throws IOException {
        ServerSocket s = new ServerSocket(PORT);
        System.out.println("Server Started");
        try {
            while(true) {
                Socket socket = s.accept();
                try {
                    new ServeOneJabber(socket);
                } catch(IOException e) {
                    socket.close();
                }
            }
        } finally {
            s.close();
        }
    }
}
```

Creates listener

Blocks until a connection occurs

Thread to service client requests

Close socket

Serving Multiple Clients

Example

Init reader/ writer and calling start

Echoing back

Socket cleanup

```
class ServeOneJabber implements Runnable  {
  private Socket socket;  private BufferedReader in; private PrintWriter out;
  public ServeOneJabber(Socket s)  throws IOException {
        socket = s;
        in = new BufferedReader(new InputStreamReader( socket.getInputStream()));
        out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(
                              socket.getOutputStream())), true);
        start(); // Calls run()
  }
  public void run() {
    try {    while (true) {
                  String str = in.readLine();
                  if (str.equals("END")) break;
                        out.println(str);
              }
    } catch (IOException e) { System.err.println("IO Exception");
    } finally {
            try {  socket.close(); }
             catch (IOException e) {
                  System.err.println("Socket not closed");
             }
      }
  }
}
```

Socket Programming Example

JACOBS UNIVERSITY

Sending e-mail using Simple Mail Transport Protocol (SMTP)

1. Make a socket connection to port 25 (SMTP port)
   - SMTP describes the format for e-mail messages
   - On UNIX machines SMTP is implemented using the `sendmail` daemon
   - The SMTP server must be willing to accept your request

2. Send a *mail header* (in SMTP format), followed by *email message*:
   - Lines must be terminated with `\r` followed by `\n` (SMTP specification)
   - You can supply any sender you like (fake messages can be created!)

Socket Programming Example

JACOBS UNIVERSITY

Sending e-mail steps:

1. Open a socket to your host
   - `Socket s = new Socket("mail.yourserver.com", 25);` // 25 is SMTP
   - `PrintWriter out = new PrintWriter(s.getOutputStream());`
2. Send the following information to the print stream:

HELO *sending host*
MAIL FROM: <*sender e-mail address*>
RCPT TO: <>*recipient e-mail address*>
DATA
mail message
(any number of lines)
.
QUIT

Socket Programming Example

Email Program (Core Java Vol. II, Horstmann and Cornell)

Use **JavaMail** API (standard Java extension) for sending Emails

URLs/ URIs

URI: Syntactical construct specifying the format of a string specifying a web resource

```
mailto:b.mahleko@jacobs-university.de
```

- Is not a locator. Such URIs are called **URN** (Uniform Resource Name)

URL: Special kind of URI with sufficient information to locate a web resource

- Can open a stream to a **URL**
- Works with schemes that Java library knows how to handle
- [`http`, `https`, `ftp`, local file system `file:`, and JAR files `jar:`]

URI Specification

Syntax:

```
[scheme:]schemeSpecificPart[#fragment]
```

- The schemeSpecificPart of a URI has the structure

```
[//authority:][path][?query]
```

- For server-based URIs, the authority has the form:

```
[user-info@][host][:port]
```

Examples:

- http://maps.yahoo.com/py/maps.py?Cupertino+CA
- http://docs.mycompany.com/api/java/net/Socket.html#Socket()
- ftp://username:password@ftp.yourserver.com/pub/file.txt

Locating Resources

JACOBS UNIVERSITY

URL and URLConnection classes

- Encapsulate much of the complexity of retrieving info from a remote site
- Construct a **URL** object from a **String**

```
URL url = new URL(urlString);
```

Fetch contents of a resource

- Open a Stream using the *openStream* method of the **URL** class
- Use standard I/O operations to read data

```
InputStream inStream = url.openStream();
BufferedReader in = new BufferedReader(new InputStreamReader(inStream);
```

Locating Resources

JACOBS UNIVERSITY

The URLConnection class

- To get additional information about a Web resource:
  - ❑ Use the **URLConnection** class

Basic steps:

1. Obtain a **URLConnection** object
   - ❑ Call `openConnection` method of the **URL** class

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

Locating Resources

2. Set any request properties. Use

setDoInput	Default yielding an input stream to read data
setDoOutput	Set connection to get an output stream
setIfModifiedSince	Only interested in data modified after date
setUseCaches	Used inside applets (1st check cache)
setAllowUserInteraction	Used inside applets (Applet pops-up Dialog Box)
setRequestProperty	Sets name/ value pairs for a protocol (e.g. HTTP)
setConnectTimeout	Sets connection timeout
setReadTimeout	Sets read timeout

3. Connect to the remote resource by calling the `connect` method

```
connection.connect();
```

Locating Resources

4. Query the header information if needed

```
getContentType
getContentLength
getContentEncoding
getDate
getExpiration
getLastModified
```

5. Access the resource data (use `getInputStream` to get a stream)

Locating Resources

JACOBS UNIVERSITY

Examples

- Access a password protected Web page as follows:

```
// 1. concatenate username, a colon, and a password
String input = username + ":" + password;

// 2. compute base64 encoding (bytes to ASCII characters) of the resulting string
String encoding = base64Encode(input);

// 3. call setRequestProperty method with "Authorization" value of "Basic "
plus    // encoding
Connection.setRequestProperty("Authorization", "Basic " + encoding);
```

- Access a password protected file by FTP:

```
// construct a URL of the form
ftp://username:password@ftp.yourserver.com/pub/file.txt
```

Locating Resources

JACOBS UNIVERSITY

Reading **HTTP** Headers

- Use the getHeaderFields method to get a list of headers

```
Map<String, List<String>> headerFields = connection.getHeaderFields();
```

- **HTTP** Header Fields from typical **HTTP** request

Date: Wed, 29 Aug 2004 00:15:48 GMT
Server: Apache/1.3.31 (Unix)
Last-Modified: Sun, 24 Jun 2004 20:53:38 GMT
Accept-Ranges: bytes
Content-Length: 4813
Connection: close
Content-Type: text/html

Locating Resources

JACOBS UNIVERSITY

Reading **HTTP** Headers cont …

- Convenient methods

Key Name	Method Name	Return Type
Date	`getDate`	long
Expires	`getExpiration`	long
Last-Modified	`getLastModified`	long
Content-Length	`getContentLength`	int
Content-Type	`getContentType`	String
Content-Encoding	`getContentEncoding`	String

Locating Resources

Example: accessing Web page with username and password

```
…
        URL url = new URL(urlName);
        URLConnection connection = url.openConnection();
        // set username, password if specified on command line
        if (args.length > 2)  {
          String username = args[1];
          String password = args[2];
          String input = username + ":" + password;
          String encoding = base64Encode(input);
          connection.setRequestProperty("Authorization", "Basic " + encoding);
        }
        connection.connect();
        // print header fields
        Map<String, List<String>> headers = connection.getHeaderFields();
        for (Map.Entry<String, List<String>> entry : headers.entrySet()) {
          String key = entry.getKey();
          for (String value : entry.getValue())
              System.out.println(key + ": " + value);
        }
```

Locating Resources

JACOBS UNIVERSITY

Posting form data

- Several technologies exist to enable servers to invoke programs

- Java Servlets, JavaServer Faces, Microsoft ASP (Active Server Pages), CGI (Common Gateway Interface)

- Two commands are commonly used to send information to Web server
  - GET
  - POST

Locating Resources

JACOBS UNIVERSITY

GET command

- Simply attach parameters to the end of the URL
- URL has form:

```
http://host/script?parameters
```

- Use the following scheme:
  - ❑ Replace spaces with '+'
  - ❑ Separate parameters by '&'
  - ❑ Replace nonalphanumeric characters with '%' followed by hexadecimal number
  - ❑ Encoding called URL encoding
- Ex:

```
http://maps.yahoo.com/py/maps.py?addr=1+Infinite+Loop&cz=Cupertino+CA
```

- Problem: Browsers limit the number of characters in a GET request

Locating Resources

JACOBS UNIVERSITY

Using the **POST** command

- Do not attach parameters to URL
- Get output stream from **URLConnection**:
  - ❑ Write name/ value pairs to the output stream
- Input **HTML** form to find out the parameters

```
PrintWriter out = new PrintWriter(connection.getOutputStream());

// Send data to the server:
out.print(name1 + "=" + URLEncoder.encode(value1, "UTF-8") + "&");
out.print(name2 + "=" + URLEncoder.encode(value2, "UTF-8"));

// Close the output stream.
out.close();

// Finally, call getInputStream and read the server response.
```

Reading Assignment

JACOBS UNIVERSITY

- Core Java 2 Volume II, Chapter 3. Networking by Horstmann and Cornell

- Deitel, P. & Deitel, H. (2012) Java™: How to Program, 9th Edition. Prentice Hall. Chapter 27.