Network Programming: Clients

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/java.html

© 2013 Marty Hall

For live Java-related training, see http://courses.coreservlets.com/ or email hall@coreservlets.com.

Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
  - JSF 2, PrimeFaces, servlets/JSP, Ajax, jQuery, Android development, Java 7 or 8 programming, custom mix of topics
  - Courses available in any state or country. Maryland/DC area companies can also choose afternoon/evening courses.
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Spring, Hibernate/JPA, GWT, Hadoop, HTML5, RESTful Web Services

Contact hall@coreservlets.com for details

Agenda

- **Creating sockets**
- **Implementing a generic network client**
- **Parsing data**
  - StringTokenizer
  - String.split
- **Getting user info from a mail server**
- **Retrieving files from an HTTP server**
- **Retrieving Web documents by using the URL class**

Basic Idea

Client vs. Server

- **Traditional definition**
  - Client: User of network services
  - Server: Supplier of network services
- **Problem with traditional definition**
  - If there are 2 programs exchanging data, it seems unclear
  - Some situations (e.g., X Windows) seem reversed
- **Easier way to remember distinction**
  - Server starts first. Server doesn't specify host (just port).
  - Client starts second. Client specifies host (and port).
- **Analogy: Company phone line**
  - Installing phone is like starting server
  - Extension is like port
  - Person who calls is the client: he specifies both host (general company number) and port (extension)

Client vs. Server (Continued)

- **If server has to start first, why are we covering clients before we cover servers?**
  - Clients are slightly easier.
  - We can test clients by connecting to *existing* servers that are already on the internet.
- **Point: clients created in Java need not communicate with servers written in Java.**
  - They can communicate with any server that accepts socket connections (as long as they know the proper communication protocol).
  - Exception: ObjectInputStream and ObjectOutputStream allow Java programs to send complicated data structures back and forth. Only works in Java, though.

Steps for Implementing a Client

1. **Create a Socket object**

```
Socket client = new Socket("hostname", portNumber);
```

2. **Create output stream to send data to the Socket**

```
// Last arg of true means autoflush -- flush stream
// when println is called
PrintWriter out =
  new PrintWriter(client.getOutputStream(), true);
```

3. **Create input stream to read response from server**

```
BufferedReader in =
  new BufferedReader
    (new InputStreamReader(client.getInputStream()));
```


Steps for Implementing a Client (Continued)

4. **Do I/O with the input and output Streams**
   - For the PrintWriter, use print, println, and printf, similar to System.out.print/println/printf
     - The main difference is that you can create PrintWriters for different Unicode characters sets, and you can't with PrintStream (the class of System.out).
   - For the BufferedReader, call read to get a single character or an array of chars, or call readLine to get a whole line
     - Note that readLine returns null if the connection was terminated (i.e. on EOF), but waits otherwise
   - You can use ObjectInputStream and ObjectOutputStream for Java-to-Java communication. Very powerful and simple.
5. **Close the socket when done**

```
client.close();
```

   - If you declare the socket using a try with resources, this happens automatically. Closing the socket closes the input and output streams as well.

Exceptions

- **UnknownHostException**
  - If host passed to Socket constructor is not known to DNS server.
    - Note that you may use an IP address string for the host
- **IOException**
  - Timeout
  - Connection refused by server
  - Interruption or other unexpected problem
    - Server closing connection does *not* cause an error when reading: null is returned from readLine

Helper Class: SocketUtils

- **Idea**
  - It is common to make BufferedReader and PrintWriter from a Socket, so simplify the syntax slightly
- **Code**

```
public class SocketUtils {

    public static BufferedReader getReader(Socket s) throws IOException {
        return(new BufferedReader
                        (new InputStreamReader(s.getInputStream())));
    }


    public static PrintWriter getWriter(Socket s) throws IOException {
        // Second argument of true means autoflush.
        return (new PrintWriter(s.getOutputStream(), true));
    }
}
```

A Generic Network Client

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

public abstract class NetworkClient {
  private String host;
  private int port;

  public String getHost() {
    return(host);
  }

  public int getPort() {
    return(port);
  }

  /** Register host and port. The connection won't
   *  actually be established until you call
   *  connect.
   */

  public NetworkClient(String host, int port) {
    this.host = host;
    this.port = port;
  }
```


A Generic Network Client (Continued)

```
  public void connect() {
    try(Socket client = new Socket(host, port)) {
      handleConnection(client);
    } catch(UnknownHostException uhe) {
      System.out.println("Unknown host: " + host);
      uhe.printStackTrace();
    } catch(IOException ioe) {
      System.out.println("IOException: " + ioe);
      ioe.printStackTrace();
    }
  }

  /** This is the method you will override when
   *  making a network client for your task.
   */

  protected abstract void handleConnection(Socket client)
      throws IOException;
}
```

Example Client

```
public class NetworkClientTest extends NetworkClient {
  public NetworkClientTest(String host, int port) {
    super(host, port);
  }

  protected void handleConnection(Socket client)
      throws IOException {
    PrintWriter out = SocketUtils.getWriter(client);
    BufferedReader in = SocketUtils.getReader(client);
    out.println("Generic Network Client");
    System.out.printf
      ("Generic Network Client:%n" +
       "Connected to '%s' and got '%s' in response.%n",
       getHost(), in.readLine());
  }
```


Example Client (Continued)

```
  public static void main(String[] args) {
    String host = "localhost";
    int port = 8088;
    if (args.length > 0) {
      host = args[0];
    }
    if (args.length > 1) {
      port = Integer.parseInt(args[1]);
    }
    NetworkClientTest tester =
      new NetworkClientTest(host, port);
    tester.connect();
  }
}
```

Example Client: Result

```
> java NetworkClientTest ftp.microsoft.com 21
Generic Network Client:
Made connection to ftp.microsoft.com and got
‘220 Microsoft FTP Service’ in response
>
```


Aside: String Formatting and Parsing

Formatting and Parsing Strategies

- **Idea**
  - Simple to connect to a server and create Reader/Writer
  - So, hard parts are formatting request and parsing response
- **Approach**
  - Formatting requests
    - Use printf (aka String.format)
  - Parsing response: simplest
    - Use StringTokenizer
  - Parsing response: more powerful
    - Use String.split with regular expressions
  - Parsing response: most powerful
    - Use Pattern and full regex library
      - Not covered in this tutorial

Parsing Strings Using StringTokenizer

- **Idea**
  - Build a tokenizer from an initial string
  - Retrieve tokens one at a time with `nextToken`
  - You can also see how many tokens are remaining (`countTokens`) or simply test if the number of tokens remaining is nonzero (`hasMoreTokens`)

```
StringTokenizer tok
  = new StringTokenizer(input, delimiters);
while (tok.hasMoreTokens()) {
  doSomethingWith(tok.nextToken());
}
```

StringTokenizer

- **Constructors**
  - StringTokenizer(String input, String delimiters)
  - StringTokenizer(String input, String delimiters, boolean includeDelimiters)
  - StringTokenizer(String input)
    - Default delimiter set is " `\t\n\r\f`" (whitespace)
- **Methods**
  - nextToken(), nextToken(String delimiters)
  - countTokens()
  - hasMoreTokens()
- **Also see methods in String class**
  - split, substring, indexOf, startsWith, endsWith, compareTo, …
  - Java has good support for regular expressions

Interactive Tokenizer: Example

```
import java.util.StringTokenizer;

public class TokTest {
  public static void main(String[] args) {
    if (args.length == 2) {
      String input = args[0], delimiters = args[1];
      StringTokenizer tok
        = new StringTokenizer(input, delimiters);
      while (tok.hasMoreTokens()) {
        System.out.println(tok.nextToken());
      }
    } else {
      System.out.println
        ("Usage: java TokTest string delimiters");
    }
  }
}
```

Interactive Tokenizer: Result

```
> java TokTest http://www.microsoft.com/~gates/ :/.
http
www
microsoft
com
~gates


> java TokTest "if (tok.hasMoreTokens()) {" "(){. "
if
tok
hasMoreTokens
```


Parsing Strings using the split method of String

- **Basic usage**
  - String[] tokens = mainString.split(delimiterString);
- **Differences from StringTokenizer**
  - Entire string is the delimiter (not one-char delimiters)
    - "foobar".split("ob") returns "fo" and "ar"
    - "foobar".split("bo") returns "foobar"
  - You can use regular expressions in the delimiter
    - ^, $, *, +, ., etc for beginning of String, end of String, 0 or more, 1 or more, any one character, etc.
    - See http://java.sun.com/javase/6/docs/api/java/util/regex/Pattern.html#sum
  - Unless you use "+", an empty string is returned between consecutive delimiters
    - "foobar".split("o") returns "f", "", and "bar"
    - "foobar".split("o+") returns "f" and "bar"

Importance of Regular Expressions

- **Idea**
  - String.split and other methods use regular expressions
  - So do many other languages. Knowing regex syntax is an important part of *every* programmer's repertoire.

From Randall Munroe and xkcd.com

- **Tutorials**
  - http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#sum
  - http://docs.oracle.com/javase/tutorial/essential/regex/

Interactive Tokenizer: Example

```
public class SplitTest {
  public static void main(String[] args) {
    if (args.length == 2) {
      String[] tokens = args[0].split(args[1]);
      for(String token: tokens) {
        if (token.length() != 0) {
          System.out.println(token);
        }
      }
    } else {
      System.out.println
        ("Usage: java SplitTest string delimeters");
    }
  }
}
```

Interactive Tokenizer: Result

```
> java TokTest http://www.microsoft.com/~gates/ :/.
http
www
microsoft
com
~gates

> java SplitTest http://www.microsoft.com/~gates/ :/.
http
www.microsoft.com/~gates/

> java SplitTest http://www.microsoft.com/~gates/ [:/.]+
http
www
microsoft
com
~gates
```


Problems with Blocking IO

A Client to Verify Email Addresses

- **Talking to a mail server**
  - One of the best ways to get comfortable with a network protocol is to telnet to the port a server is on and try out commands interactively
- **Example talking to apl.jhu.edu's server**

```
> telnet apl.jhu.edu 25
Trying 128.220.101.100 ...Connected … Escape character …
220 aplcenmp.apl.jhu.edu Sendmail SMI-8.6/SMI-SVR4 ready …
expn hall
250 Marty Hall <hall@aplcenmp.apl.jhu.edu>
expn root
250 Gary Gafke <…>
250 Tom Vellani <…>
quit
221 aplcenmp.apl.jhu.edu closing connection
Connection closed by foreign host
```


Address Verifier

```
public class AddressVerifier extends NetworkClient {
  private String username;

  public AddressVerifier(String username, String
  hostname,
                         int port) {
    super(hostname, port);
    this.username = username;
  }

  public static void main(String[] args) {
    if (args.length != 1) {
      usage();
    }
    MailAddress address = new MailAddress(args[0]);
    new AddressVerifier(address.getUsername(),
                        address.getHostname(), 25);
  }
```

Address Verifier (Continued)

```
  protected void handleConnection(Socket client)
      throws IOException {
    PrintWriter out = SocketUtils.getWriter(client);
    InputStream rawIn = client.getInputStream();
    byte[] response = new byte[1000];
    // Clear out mail server's welcome message.
    rawIn.read(response);
    out.println("EXPN " + username);
    // Read the response to the EXPN command.
    int numBytes = rawIn.read(response);
    // The 0 means to use normal ASCII encoding.
    System.out.write(response, 0, numBytes);
    out.println("QUIT");
  }
  ...
}
```

Main point: you can only use readLine if either
- You know how many lines of data will be sent (call readLine that many times)
- The server will close the connection when done, as with HTTP servers (call readLine until you get null)

MailAddress

```
public class MailAddress {
  private String username, hostname;

  public MailAddress(String emailAddress) {
    String[] pieces = emailAddress.split("@");
    if (pieces.length != 2) {
      System.out.println("Illegal email address");
      System.exit(-1);
    } else {
      username = pieces[0];
      hostname = pieces[1];
    }
  }

  public String getUsername() {
    return(username);
  }
  public String getHostname() {
    return(hostname);
  }
}
```

Address Verifier: Result

```
> java AddressVerifier tbl@w3.org
250 <timbl@hq.lcs.mit.edu>

> java AddressVerifier timbl@hq.lcs.mit.edu
250 Tim Berners-Lee <timbl>

> java AddressVerifier gosling@mail.javasoft.com
550 gosling... User unknown
```


Web (HTTP) Clients

Brief Aside: Using the HTTP GET Command

- **For the URL http://www.apl.jhu.edu/~hall/**

```
Unix> telnet www.apl.jhu.edu 80
Trying 128.220.101.100 ...
Connected to aplcenmp.apl.jhu.edu.
Escape character is '^]'.
GET /~hall/ HTTP/1.0

HTTP/1.1 200 OK
Date: Fri, 24 Aug 2007 18:06:47 GMT
Server: Apache/2.0.49 (Unix) mod_ssl/2.0.49 ...
Last-Modified: Tue, 07 Aug 2007 18:50:50 GMT
...
Connection: close
Content-Type: text/html; charset=ISO-8859-1


<!DOCTYPE HTML PUBLIC
          "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
...
</HTML>Connection closed by foreign host.
Unix>
```


Using HTTP 1.0 vs. HTTP 1.1

- **Advantage of 1.1**
  - You can connect to hosts that are using virtual hosting
    - I.e., sites that host multiple domain names on the same machine

- **E.g., for URL http://somehost/somepath**

HTTP 1.0	HTTP 1.1

Connect to somehost on port 80
`GET /somepath HTTP/1.0`
Blank line
	Connect to somehost on port 80
	`GET /somepath HTTP/1.1`
	`Host: somehost`
	`Connection: close`
	Blank line

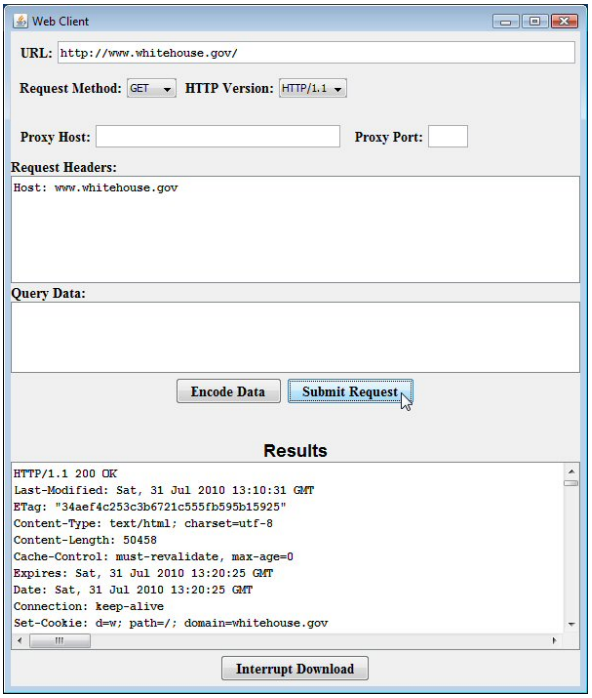
Talking to Web Servers Interactively

- **Telnet**
  - Most people think of telnet as a tool for logging into a remote server on default login port (23)
  - But, it is really more general: a tool for connecting to a remote server on any port and interactively sending commands and looking at results
- **Enabling telnet on Windows 7 or Vista**
  - Starting with Windows Vista, telnet is disabled by default
    - To enable it, see http://technet.microsoft.com/en-us/library/cc771275(WS.10).aspx
    - Or Google for "install telnet windows 7" and above page will come up #1
    - You may also need to turn on local echo – Unix telnet clients are much more convenient

Talking to Web Servers Interactively

- **Problem**
  - MS Windows telnet client works poorly for this
    - Linux, Solaris, and MacOS telnet clients work fine for this
- **Solution: WebClient**
  - Simple graphical user interface to communicate with HTTP servers
  - User can interactively specify:
    - URL with host, port, and URI
    - HTTP request headers
  - HTTP request is performed in a separate thread
  - Response document is placed in a scrollable text area
  - Download all source files for WebClient from tutorial home page

WebClient: Example



A Class to Retrieve a Given URI from a Given Host

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

public class UriRetriever extends NetworkClient {
  private String uri;

  public static void main(String[] args) {
    UriRetriever retriever =
      new UriRetriever(args[0], Integer.parseInt(args[1]),
                       args[2]);
    retriever.connect();
  }

  public UriRetriever(String host, int port,
                      String uri) {
    super(host, port);
    this.uri = uri;
  }
```

A Class to Retrieve a Given URI from a Given Host (Continued)

```
  // It is safe to use blocking IO (readLine) since
  // HTTP servers close connection when done,
  // resulting in a null value for readLine.

  protected void handleConnection(Socket client)
      throws IOException {
    PrintWriter out = SocketUtils.getWriter(client);
    BufferedReader in = SocketUtils.getReader(client);
    out.printf("GET %s HTTP/1.1\r\n", uri);
    out.printf("Host: %s\r\n", getHost());
    out.printf("Connection: close\r\n\r\n");
    String line;
    while ((line = in.readLine()) != null) {
      System.out.println(line);
    }
  }
}
```


A Class to Retrieve a Given URL

```
public class UrlRetriever {
  public static void main(String[] args) {
    checkUsage(args);
    UrlParser parser = new UrlParser(args[0]);
    UriRetriever uriClient =
      new UriRetriever(parser.getHost(), parser.getPort(),
                       parser.getUri());
    uriClient.connect();
  }

  /** Warn user if the URL was forgotten. */

  private static void checkUsage(String[] args) {
    if (args.length != 1) {
      System.out.println("Usage: UrlRetriever <URL>");
      System.exit(-1);
    }
  }
}
```

A Class to Retrieve a Given URL (Parser)

```
public class UrlParser {
  private String host;
  private int port = 80;
  private String uri;

  public UrlParser(String url) {
    StringTokenizer tok = new StringTokenizer(url);
    String protocol = tok.nextToken(":");
    checkProtocol(protocol);
    host = tok.nextToken(":/");
    try {
      uri = tok.nextToken("");
      if (uri.charAt(0) == ':') {
        tok = new StringTokenizer(uri);
        port = Integer.parseInt(tok.nextToken(":/"));
        uri = tok.nextToken("");
      }
    } catch(NoSuchElementException nsee) {
      uri = "/";
    }
  } ... // getters and setters
```


UrlRetriever in Action

- **No explicit port number**

```
Prompt> java UrlRetriever
          http://www.coreservlets.com/JSF-Tutorial
```

Note the missing slash at the end of the URL. Real URL is http://www.coreservlets.com/JSF-Tutorial/

```
HTTP/1.1 301 Moved Permanently
Date: Sat, 31 Jul 2010 13:33:44 GMT
Server: Apache
Location: http://www.coreservlets.com/JSF-Tutorial/
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>301 Moved Permanently</TITLE>
</HEAD><BODY>
<H1>Moved Permanently</H1>
The document has moved <A HREF="http://www.coreservlets.com/JSF-Tutorial/">here</A>.<P>
</BODY></HTML>
```

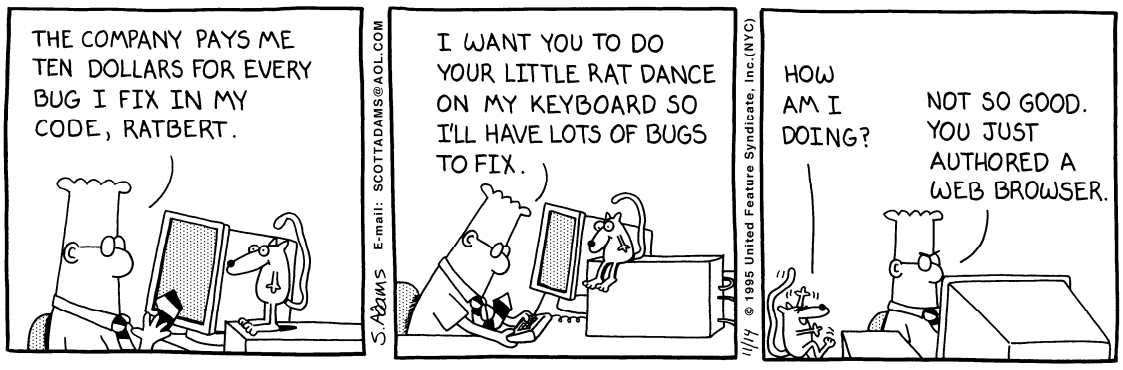
UrlRetriever in Action (Continued)

- **Explicit port number**

```
Prompt> java UrlRetriever
        http://www.google.com:80/bingSearch
HTTP/1.1 404 Not Found
Content-Type: text/html; charset=UTF-8
X-Content-Type-Options: nosniff
Date: Sat, 31 Jul 2010 13:40:09 GMT
Server: sffe
Content-Length: 1364
X-XSS-Protection: 1; mode=block
Connection: close

<html><head>
<meta http-equiv="content-type" content="text/html;charset=utf-8">
<title>404 Not Found</title>...
<body>...</body>
</html>
```


Writing a Web Browser



- **Wow! We just wrote a Web browser in 3 pages of code.**
  - Didn't format the HTML, but still not bad for 3 pages
  - But we can do even better…

Browser in 1 Page: Using URL

```
public class UrlRetriever2 {
  public static void main(String[] args) {
    try {
      URL url = new URL(args[0]);
      BufferedReader in = new BufferedReader(
                              new InputStreamReader(
                                url.openStream()));
      String line;
      while ((line = in.readLine()) != null) {
        System.out.println("> " + line);
      }
      in.close();
    } catch(MalformedURLException mue) { // URL c'tor
      System.out.println(args[0] + "is an invalid URL: "
                         + mue);
    } catch(IOException ioe) { // Stream constructors
      System.out.println("IOException: " + ioe);
    }
  }
}
```


UrlRetriever2 in Action

```
Prompt> java UrlRetriever2 http://www.yahoo.com/
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
 <html lang="en-US" class="y-fp-bg y-fp-pg-grad  bkt701">
 <!-- m2 template 0 -->
 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Yahoo!</title>
    <meta name="description" content="Welcome to Yahoo!, the world's most visited
    home page. Quickly find what you're searching for, get in touch with friends
    and stay in-the-know with the latest news and information.">
    <meta name="keywords" content="yahoo, yahoo home page, yahoo homepage,
    yahoo search, yahoo mail, yahoo messenger, yahoo games, news, finance, sport,
    entertainment">
  ...
```

Useful URL Methods

- **openConnection**
  - Yields a `URLConnection` which establishes a connection to host specified by the URL
  - Used to retrieve header lines and to supply data to the HTTP server
- **openInputStream**
  - Returns the connection's input stream for reading
- **toExernalForm**
  - Gives the string representation of the URL
- **getRef, getFile, getHost, getProtocol, getPort**
  - Returns the different components of the URL

Using the URL Methods: Example

```
import java.net.*;

public class UrlTest {
  public static void main(String[] args) {
    if (args.length == 1) {
      try {
        URL url = new URL(args[0]);
        System.out.println
          ("URL: " + url.toExternalForm() + "\n" +
           "  File:      " + url.getFile() + "\n" +
           "  Host:      " + url.getHost() + "\n" +
           "  Port:      " + url.getPort() + "\n" +
           "  Protocol:  " + url.getProtocol() + "\n" +
           "  Reference: " + url.getRef());
      } catch(MalformedURLException mue) {
        System.out.println("Bad URL.");
      }
    } else
      System.out.println("Usage: UrlTest <URL>");
  }
}
```

Using the URL Methods, Result

```
> java UrlTest http://www.irs.gov/mission/#squeezing-them-dry
URL: http://www.irs.gov/mission/#squeezing-them-dry
  File:      /mission/
  Host:      www.irs.gov
  Port:      -1
  Protocol:  http
  Reference: squeezing-them-dry
```

Note: If the port is not explicitly stated in the URL, then the standard port for the protocol is assumed and **`getPort`** returns –1

A Real Browser Using Swing

- **The `JEditorPane` class has builtin support for HTTP and HTML**

Browser in Swing: Code

```
import javax.swing.*;
import javax.swing.event.*;
...

public class Browser extends JFrame implements HyperlinkListener,
                                                ActionListener {
  private JEditorPane htmlPane;
  ...

  public Browser(String initialURL) {
    ...
    try {
        htmlPane = new JEditorPane(initialURL);
        htmlPane.setEditable(false);
        htmlPane.addHyperlinkListener(this);
        JScrollPane scrollPane = new JScrollPane(htmlPane);
        getContentPane().add(scrollPane, BorderLayout.CENTER);
    } catch(IOException ioe) {
       warnUser("Can't build HTML pane for " + initialURL
                + ": " + ioe);
    }
```


Browser in Swing (Continued)

```
    ...
    Dimension screenSize = getToolkit().getScreenSize();
    int width = screenSize.width * 8 / 10;
    int height = screenSize.height * 8 / 10;
    setBounds(width/8, height/8, width, height);
    setVisible(true);
  }

  public void actionPerformed(ActionEvent event) {
    String url;
    if (event.getSource() == urlField)
      url = urlField.getText();
    else // Clicked "home" button instead of entering URL
      url = initialURL;
    try {
      htmlPane.setPage(new URL(url));
      urlField.setText(url);
    } catch(IOException ioe) {
      warnUser("Can't follow link to " + url + ": " + ioe);
    }
  }
```

Browser in Swing (Continued)

```
...
public void hyperlinkUpdate(HyperlinkEvent event) {
  if (event.getEventType() ==
              HyperlinkEvent.EventType.ACTIVATED) {
    try {
      htmlPane.setPage(event.getURL());
      urlField.setText(event.getURL().toExternalForm());
    } catch(IOException ioe) {
       warnUser("Can't follow link to "
                + event.getURL().toExternalForm() +
                ": " + ioe);
    }
  }
}
```


Wrap-Up

Summary

- **Open a Socket**
  - new Socket("hostname-or-IP-Address", port)
- **Get a PrintWriter to send data to server**
  - new PrintWriter(client.getOutputStream(), true);
- **Get a BufferedReader to read server data**
  - new BufferedReader
    (new InputStreamReader(client.getInputStream()));
- **Notes**
  - readLine (from PrintWriter) blocks until data received or connection closed (null returned in that case)
  - HTTP servers normally close the connection after sending data, so readLine returns null at the end
  - String.split and StringTokenizer help parse strings

Questions?