ID2212 Network Programming with Java
Lecture 7

Working with Web Resources and URL Connections

Leif Lindbäck, Vladimir Vlassov
KTH/ICT/SCS
HT 2015

Outline

- Accessing Web Resources in Java
  - Locating Web resources: URL
  - Communicating with HTTP servers: URL connections
    - Downloading and uploading data on the Web
  - Retrieving Web resources pointed to by URLs
    - Images, audio, HTML documents, classes, files
    - Loading images from the Web. Tracking the loading status
- Developing a Web protocol in Java

Locating Web Resources: `java.net.URL`

- `java.net.URL` represents an URL - Uniform Resource Locator of a Web resource (service)
  - An URL object is used to locate and to grab Web resources.
    - NB! Java 2 SDK provides implementations for HTTP, FTP and SMTP (mailto).
- An URL with the http scheme:

```
http://www.it.kth.se:80/labs/se/index.html
```

protocol | host | port | resource name (path)

URL Constructors

```
URL(…)
  ├── (String locator)
  ├── (URL url, String locator)
  ├── (String protocol, String serverName, String reso
  └── (String protocol, String serverName,
       int port, String resource)
```

```
try {
    URL url1 = new
      URL("http://www.ora.com/info/java/index.html");
    // create an absolute URL from a base and relative URL
    URL url2 = new URL(url1, "bibliography.html");
    // An URL relative to the web page of an Applet
    URL url = new URL(getDocumentBase(), "bibliography.html");
} catch (MalformedURLException e) {
   e.printStackTrace();
}
```

URL Stream Handler

- A stream protocol handler that knows how to make a connection for a particular protocol type, such as http, ftp.
- A URL constructor loads a stream handler class (a subclass of **`URLStreamHandler`**) for the specified protocol
  - If `URLStreamHandlerFactory` is set, call its method `createURLStreamHandler`, otherwise:
  - Try to find `<package>.<protocol>.Handler` in the list provided by the property `java.handler.protol.pkgs`
  - Try to load the class `sun.net.www.protocol.<protocol>.Handler`
  - On failure, throw `MalFormedURLException`

Communicating with a Web Server: `java.net.URLConnection`

- The `URLConnection` class represents a communication link to the resource (a TCP socket connection).

```
URL url = new
  URL("http://www.it.kth.se/index.html");
URLConnection urlc = url.openConnection();
```

- The server is contacted only when needed:

```
URL.getContent()
URL.openConnection.getDate(), etc.
```

Input and Output Streams of URLConnection

- URL connection provides input and output streams
  - Input stream for downloading the resource contents, e.g. classes, images.

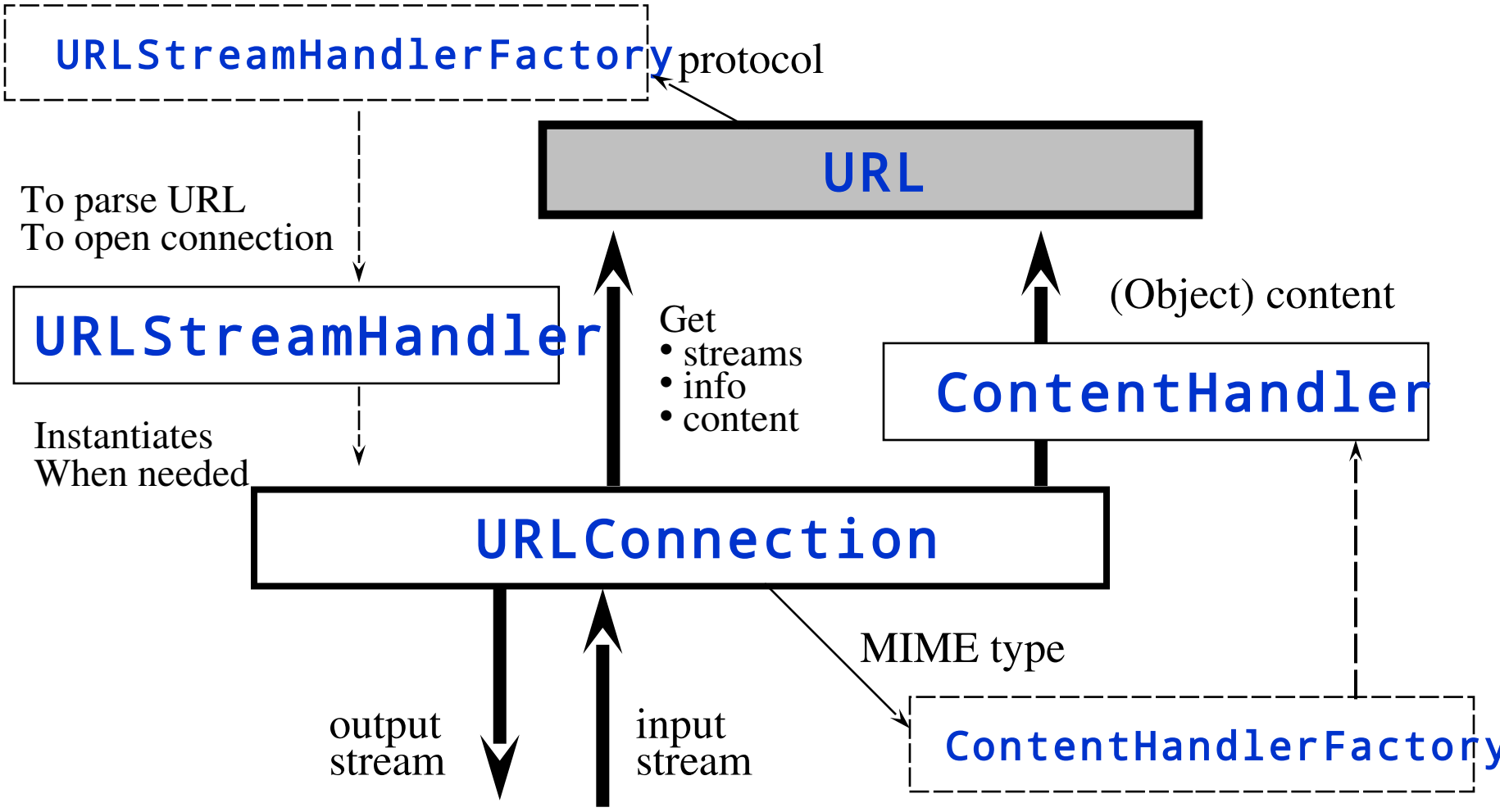
```
URLConnection urlc =url.openConnection();
InputStream in = urlc.getInputStream();
```

  - Output stream for uploading data to the server at the corresponding URL, e.g. posting query to a CGI script or a servlet

```
URLConnection urlc =url.openConnection();
OutpuStream o = urlc.getOutputStream();
```

- In the case of HTTP URL connection:
  - The input stream is returned after a GET (default) request is written to the output stream and a header is read from the input stream
  - The output stream is returned after a header of a POST request (default) is written to the output stream

Architecture of URL Related Classes



Retrieving Resource Information and Content (HTTP)

- Methods of `URLconnection` to get information about the resource and its content:

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

  - The information is obtained via the `GET` request.
  - The server sends a MIME header and resource data in reply.

Getting the Content of the Web Resource (HTTP)

`url.getContent()` – a shorthand for `url.openConnection().getContent()`

- `getContent` returns an object of the class:
  - `InputStream` - for reading the resource as byte stream;
  - `ImageProducer` - if the resource is an image file.
    - Use `instanceOf` to determine the class of the object in return.
    - To create an image use: `anyAWTcomponent.createImage(ImageProducer)`

A Content Handler

- Implements the `getContent` method that reads data from the input stream of the given URL connection and converts them into a Java object.
- The handler is associated with one or several MIME types that it can handle.
- The handler can support caching of content.

Call Stack Trace `url.getContent()`

- `URL.getContent()`
- `URLStreamHandler.openConnection(URL)`
  - The stream handler creates an URL connection
- `URLConnection.getContent()`
  - The connection connects to server
- `URLConnection.getContentHandler()`
  - The connection finds (creates) a content handler for a given MIME type such as “text/tab-separated-values”
- `ContentHandler.getContent()`
  - The handler converts resource data into a Java object

Looking for a Content Handler

- If the handler for the given MIME type is not set, try to set it:
  - If a `ContentHandlerFactory` is set, call its method `createContentHandler(String type)`, otherwise:
  - Try to load the class `sun.net.www.content.type.subtype`
  - On failure, throw `IOException`

Implementing a Content Handler

- Extends the ContentHandler class and implements for the method:

  `public Object getContent(URLConnection)`

  - Read data from the input stream of the given connection, construct and return a Java object.
- See Example 3.21: A content handler for text/tab-separated-values.
  - Reads lines from the given input stream;
  - Splits each tab-separated string into an array of strings by using a `StringTokenizer` and adds the array to a vector;
  - Returns the vector from the `getContent` method as the content of the resource pointed to by the given URL.

Examples: URL and URLConnection

- Example 3.12: Get type and content `URLContentType.java`
- Example 3.16: Content content type `GetHeader.java`
- Example 3.17: A URL Class Loader `URLClassLoader.java`
- Example 3.18: A simple applet viewer using URLClassLoader `loadApplet.java`
- Example 3.19.1: An Applet submits a POST request `Report.java`

See examples at

http://www.imit.kth.se/courses/ID2212/examples/html/

Methods for Retrieving Web Resources (HTTP)

- An image:
  - `applet.getImage(URL)` is a shorthand for `applet.getAppletContext.getImage(URL)`
  - `awtcomponent.getToolkit().getImage(URL)`
- An audio clip:
  - `applet.getAudioClip(URL)` is a shorthand for `applet.getAppletContext.getAudioClip(URL)`
- The content of a Web resource (input byte stream):
  - `url.getContent()`
- Show a hyper-text document given URL:
  - `applet.getAppletContext().showDocument(URL)`

Network Methods of the `Applet` Class

- Getting audio and image files:
  - `Image getImage(URL)`
  - `Image getImage(URL, String)`
  - `AudioClip getAudioClip(URL)`
  - `AudioClip getAudioClip(URL, String)`
  - `play(URL)`
  - `play(URL, String)`
- Locate the applet or the document in which the applet is embedded:
  - `URL getCodeBase()`
  - `URL getDocumentBase()`

Applet Context

- The applet context represents an applet's environment, e.g. a browser or an applet viewer:

```
AppletContext ac = getAppletContext();
```

- Useful methods of the AppletContext interface:

`showDocument(URL)`

  - replace the current Web with the given URL.

`showDocument(URL url, String target)`

  - Show the Web page indicated by url. The target argument indicates where to display the frame, for instance, in the current frame or in a new navigator.

`getApplet(String)`

  - returns the applet with the given name

- See Example 3.13 `ShowImage1.java`
  - The applet is at http://www.imit.kth.se/courses/ID2212/examples/html/ShowImage1.html

Named Applets

- An applet can be named in the APPLET (EMBED) tag:

```
<APPLET CODEBASE="../MyClasses" CODE=TalkApplet.class
  WIDTH=793 HEIGHT=130 NAME="Top”>
<PARAM NAME=Partner VALUE="Bottom">
</APPLET>
<HR>
<APPLET CODEBASE="../MyClasses" CODE=TalkApplet.class
  WIDTH=793 HEIGHT=130 NAME="Bottom”>
<PARAM NAME=Partner VALUE="Top">
</APPLET>
```

- Named applets embedded in the same Web page can get references to each other via the shared Applet Context:

```
TalkApplet a = (TalkApplet)(getAppletContext().
getApplet(getParameter("Partner")));
```

Interaction of Applets via the Applet Context

- Applets on the same Web page share the Applet Context (JVM) and can interact with each other via
  - Method invocation. The methods should be declared as synchronized.
  - Static class variables;
  - Piped connection.
- An applet gets reference to another applet by name.
- See Example 4.5 TalkApplet.java at
  http://www.imit.kth.se/courses/ID2212/examples/html/TalkApplet.html

Retrieving Images

- `getImage(URL)` of the Applet and Toolkit classes retrieves an image (GIF, JPEG, XBM, etc.)
  - The method immediately returns an `Image` object whether or not the image data actually has arrived:

```
Image image;
try {
    image = getImage(new
  URL( getParameter( "ImageURL" )));
} catch (MalformedURLException e) {
}
```

Loading Images

- The image starts loading as soon as the image data are actually needed for drawing, filtering or sizing:
  - `graphics.drawImage(img, x, y, ImageObserver)`
  - `image.getHeight(ImageObserver)`
  - `image.getWidth(ImageObserver)`
- To force loading:
  - `anyAWTcomponent.prepareImage(img, ImageObserver)`
- Loading of the image is performed asynchronously in a separate thread. Problem: a long latency of loading.
  - `getWidth` and `getHeight` return -1 if the size is not yet known.

Monitoring the Image Loading
`java.awt.MediaTracker`

- The class **`MediaTracker`** is used to monitor the loading status of images (media)
  - Create a `MediaTracker` object.
  - Add an image with an integer identifier (ID) to the tracker: `tracker.addImage(Image image, int ID)`
    - The ID controls the priority order in which the images are fetched.
    - The same ID can be assigned to a group of images.
  - Check the image status (ready or not) and/or wait for an image (a group of images) to finish loading.
  - Use (draw) the image when it is ready.

Waiting for Images to Finish Loading

- Block waiting: wait until loaded:
  - `waitForID(int)`
  - `waitForAll()`
- Wait until loaded, or until the specified time has passed:
  - `waitForAll(long)`
  - `waitForID(int, long)`
- Busy waiting: Checks to see if loaded (produced):
  - `checkAll()`
  - `checkID(int, long)`

Usage of MediaTracker

```
MediaTracker tracker ;
Image img ;
public void init () {
   try {
     MediaTracker tracker = new MediaTracker(this);
     Image img = getImage(getDocumentBase(),
   “front.gif“);
     tracker.addImage(img, 1, 30, 30);
   } catch (Exception e) {}
}
public void paint(Graphics g) {
   if (tracker.checkID(1)) drawImage(img, 0, 0, null);
   else g.drawString(“Picture: Loading...“, 25, 50);
}
```

Animation with `MediaTracker`

- Animation can be achieved by displaying a sequence of images
- The tracker is employed to
  - force the image to load,
  - wait until loading is complete,
  - call `drawImage` to display the image.
- See Example 3.15: `ImageBlaster.java`.
  - See the applet in action at

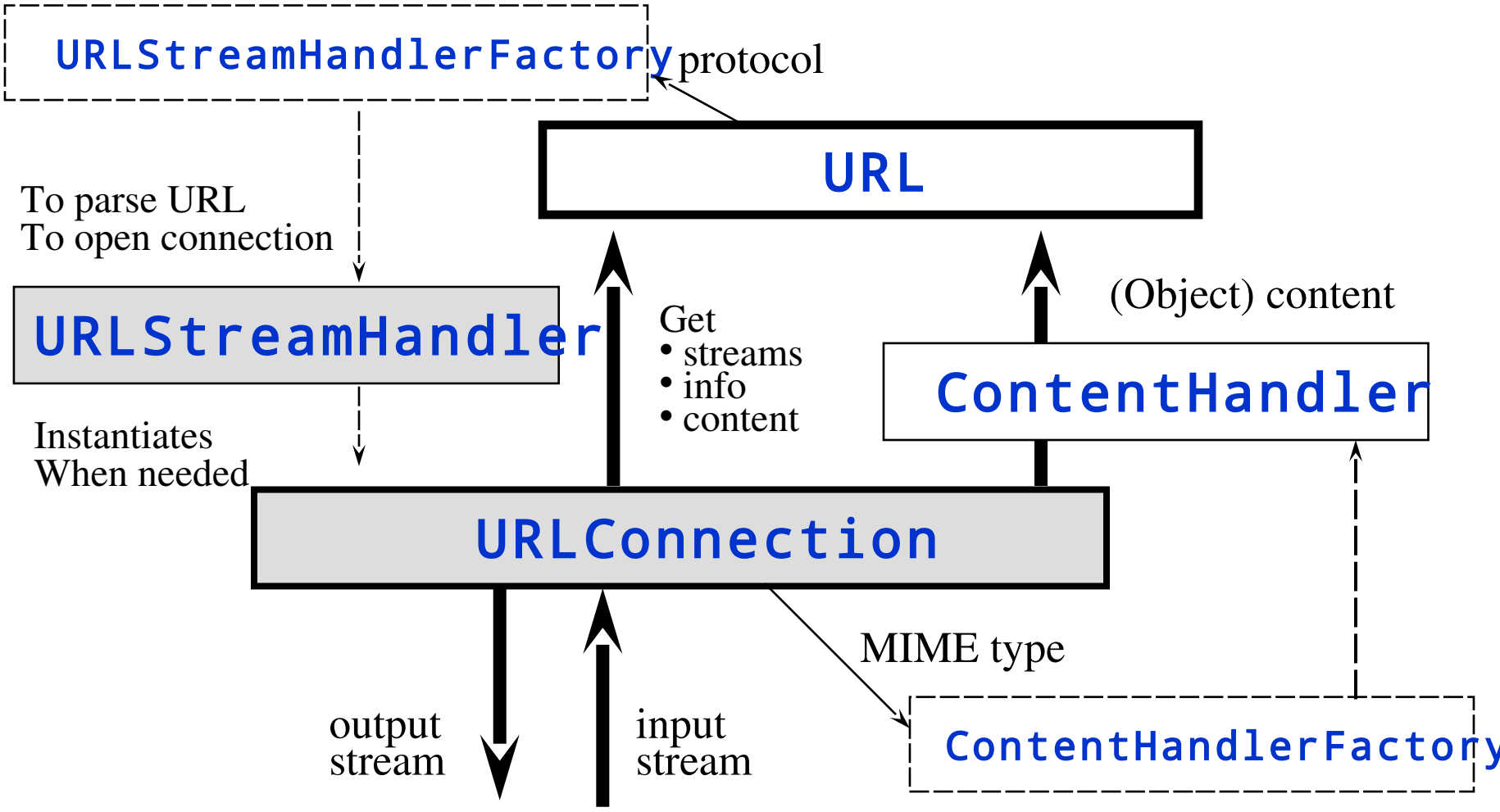
http://www.imit.kth.se/courses/ID2212/examples/html/ImageBlaster.html

  - The example is taken from the Java SDK Documentation: Class `java.awt.MediaTracker`

Developing a Custom Web Protocol

- `protocol://user:password@host:port/path`
- Steps:
  - Develop a `URLStreamHandler` subclass for the protocol
  - Develop a `URLConnection` subclass for the protocol

Review: URL Related Classes



Loading URL Stream Handler

- A URL constructor loads a stream handler class for the given protocol:
  - If the handler for the protocol is not set, try to set it:
  - If a `URLStreamHandlerFactory` is set, call its method `createURLStreamHandler`, otherwise:
  - Try to find **`<package>.<protocol>.Handler`** in the list provided by the property **`java.handler.protol.pkgs`**
  - Try to load the class **`sun.net.www.protocol.<protocol>.Handler`**
  - On failure, throw **`MalFormedURLException`**

A `URLStreamHandler` Subclass

- The `URLStreamHandler` subclass for a custom Web protocol must be able to:
  - Parse a URL specified in the `URL` constructor, i.e. implement

```
protected void parseURL(URL, String,
  int, int)
protected String toExternalForm(URL)
```

  - Create a `URLConnection` object for the URL

```
protected URLConnection
openConnection(URL)
```

Extending `URLConnection`

- A `URLConnection` subclass for the custom Web protocol must be able to:
  - Create a socket connection to the specified server

    ```
    protected boolean connected;
    abstract public void connect()
    ```

  - Provide a ready-to-read input and a ready-to-write output streams:

    ```
    getInputStream(), getOutputStream()
    ```

  - Provide information about the resource:

    ```
    getContentType(),
      getLastModified,getContentLength,
    getExpiration, getContentEncoding, getDate()
    ```

  - Provide a Java object that represents the content of the resource with the help of a ContentHandler:

    ```
    Object getContent()
    ```

Example 3.20: A Finger URL Connection

- Assume that a finger URL looks like:

```
finger:user@host
```

- Classes for the finger URL Connection to be developed:
  - `public class fingerURLStreamHandler extends java.net.URLStreamHandler`
  - `public class fingerURLConnection extends java.net.URLConnection`

Finger URL Connection

```
public class fingerURLStreamHandler
  extends java.net.URLStreamHandler
```

- Three methods to be implemented (to be overridden):

  `protected void parseURL(URL,String,int,int)`

  - Parses URL string and sets fields in the given URL object

  `protected URLConnection openConnection(URL)`

  - Returns an instance of the `fingerURLConnection` class

  `protected String toExternalForm(URL)`

Finger URL Connection (cont'd)

```
public class fingerURLConnection extends
  URLConnection
```

- Five methods to be implemented:
  - `public fingerURLConnection`
    - Constructor.
  - `public synchronized void connect()`
    - Connects to the given host on port 79 and writes a request that contains the user field of the finger URL specification.
  - `public String getContentType()`
    - Returns `"text/plain"`
  - `public synchronized InputStream getInputStream()`
    - Creates and returns `ByteArrayInputStream` (the cached resource).
  - `public Object getContent()`
    - Returns `ByteArrayInputStream` by using `getInputStream`

Finger URL Connection (cont'd)

- Implementation:
  - Example 3.20.1: `fingerURLStreamHandler`
  - Example 3.20.2: `fingerURLConnection`
- A finger test application
  - Example 3.20.3: `ProtocolTester`
- Find the examples at:

http://www.imit.kth.se/courses/ID2212/examples/html/index.html

Example 3.20.1: fingerURLStreamHandle

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

public class fingerURLStreamHandler extends java.net.URLStreamHandler {

  protected void parseURL(URL u, String spec, int start, int limit) {
    // parse only the remaining fields of the specification
    // (except of protocol a protocol part
    StringTokenizer st =
        new StringTokenizer(spec.substring(start),
        "@", false);
    String file = st.nextToken();
    String host = st.nextToken();
    String ref = null;
    int port = 79;
    setURL(u, u.getProtocol(), host, port, file, ref);
  }
  protected URLConnection openConnection(URL u) throws IOException {
    return new fingerURLConnection(u);
  }
  protected String toExternalForm(URL u) {
    return "finger:" + u.getFile() + "@" + u.getHost();
  }
}
```

Example 3.20.2:
fingerURLConnecti

```
import java.net.*;
import java.io.*;
public class fingerURLConnection extends URLConnection {
  Socket theConnection = null;
  public final static int defaultPort = 79;
  private String eol = System.getProperty("line.separator", "\n");
  public fingerURLConnection (URL u) {
    super(u);
  }
  public synchronized InputStream getInputStream() throws IOException {
    if (!connected) connect();
    InputStream is = theConnection.getInputStream();
    int bfr = 128;
    byte[] b = new byte[bfr << 4];
    int count = 0;
    int offset = 0;
    while (count >= 0) {
      count = is.read(b, offset, bfr);
      if (count == -1) break;
      offset += count;
      if (offset == b.length) {
        byte temp[] = new byte[offset << 1];
        System.arraycopy(b, 0, temp, 0, offset);
        b = temp;
        } else if (offset > b.length) return null;
    }
    if (offset < b.length) {
      byte temp[] = new byte[offset];
      System.arraycopy(b, 0, temp, 0, offset);
      b = temp;
    }
    return new ByteArrayInputStream(b);
  }
```

Example 3.20.2: `fingerURLStreamHandler` (cont'd)

```
  public String getContentType() {
    return "text/plain";
  }
  public Object getContent() throws IOException {
    return getInputStream();
  }
  public synchronized void connect() throws IOException {
    int port;
    if (!connected) {
      port = url.getPort();
      if ( port < 0) port = defaultPort;
      theConnection = new Socket(url.getHost(), port);
      PrintWriter wr = new PrintWriter(
          theConnection.getOutputStream(), true);
      wr.println(url.getFile());
      connected = true;
    }
  }
}
```