

Network Programming [CACS355]

Er.Sital Prasad Mandal

**BCA- VI
Mechi Campus
Bhadrapur, Jhapa, Nepal**

(Email : info.sitalmandal@gmail.com)


<https://networkprogam-mmc.blogspot.com/>



Text Book

- **Elliote Rusty Harold, “Java Network Programming”
O’Reilly, 2014**
- **David Reilly, Michael Reilly, “Java Networking
Programming and Distributed Computing”**

URLConnections

- 
1. Opening URLConnections
 2. Reading Data from a Server
 3. Reading the Header
 - i. Retrieving Specific Header Fields
 - ii. Retrieving Arbitrary Header Fields
 4. Caches
 - i. Web Cache for Java
 5. Configuring the Connection
 - i. protected URL url
 - ii. protected boolean connected
 - iii. protected boolean allowUserInteraction
 - iv. protected boolean doInput
 - v. protected boolean doOutput
 - vi. protected boolean ifModifiedSince
 - vii. protected boolean useCaches
 - viii. Timeouts
 6. Configuring the Client Request HTTP Header
 7. **Writing Data to a Server**
 8. Security Considerations for URLConnections
 9. Guessing MIME Media Types

URLConnections



URLConnection is an abstract class that represents an active connection to a resource specified by a URL.

*The **URLConnection** class has two different but related purposes:*

First, it provides more control over the interaction with a server (especially an HTTP server) than the URL class. A **URLConnection** can inspect the header sent by the server and respond accordingly. It can set the header fields used in the client request. Finally, a **URLConnection** can send data back to a web server with POST, PUT, and other HTTP request methods.

Second, the **URLConnection** class is part of Java's protocol handler mechanism, which also includes the **URLStreamHandler** class.

The Java **URLConnection** class represents a communication link between the URL and the application. This class can be used to read and write data to the specified resource referred by the URL.

URLConnections



Point to Note:

- The `java.net.URLConnection` class is an abstract class that handles communication with different kinds of servers like ftp servers and web servers.
- Protocol specific subclasses of `URLConnection` handle different kinds of servers.
- By default, connections to HTTP URLs use the GET method.

1. Opening URLConnections

A program that uses the `URLConnection` class directly follows this basic sequence of steps:

1. Construct a `URL` object.
2. Invoke the `URL` object's `openConnection()` method to retrieve a `URLConnection` object for that `URL`.
3. Configure the `URLConnection`.
4. Read the header fields.
5. Get an input stream and read data.
6. Get an output stream and write data.
7. Close the connection.

1. Opening URLConnections

The single constructor for the `URLConnection` class is protected:

```
protected URLConnection(URL url)
```

```
try {  
    URL u = new URL("http://www.overcomingbias.com/");  
    URLConnection uc = u.openConnection();  
    // read from the URL...  
} catch (MalformedURLException ex) {  
    System.err.println(ex);  
} catch (IOException ex) {  
    System.err.println(ex);  
}
```

The `connect()` method of `sun.net.www.protocol.http.HttpURLConnection` creates a `sun.net.www.http.HttpClient` object, which is responsible for connecting to the server:

```
public abstract void connect() throws IOException
```

2. Reading Data from a Server

The following is the minimal set of steps needed to retrieve data from a URL using a `URLConnection` object:

1. Construct a `URL` object.
2. Invoke the `URL` object's `openConnection()` method to retrieve a `URLConnection` object for that `URL`.
3. Invoke the `URLConnection`'s `getInputStream()` method.
4. Read from the input stream using the usual stream API.

Example 7-1. Download a web page with a URLConnection

```
import java.io.*;
import java.net.*;
public class SourceViewer2 {
    public static void main (String[] args) {
        if (args.length > 0) {
            try {
                // Open the URLConnection for reading
                URL u = new URL(args[0]);
                URLConnection uc = u.openConnection();
                try (InputStream raw = uc.getInputStream()) { // autoclose
                    InputStream buffer = new BufferedInputStream(raw);
                    // chain the InputStream to a Reader
                    Reader reader = new InputStreamReader(buffer);
                    int c;
                    while ((c = reader.read()) != -1) {
                        System.out.print((char) c);
                    }
                }
            } catch (MalformedURLException ex) {
                System.err.println(args[0] + " is not a parseable URL");
            } catch (IOException ex) {
                System.err.println(ex);
            }
        }
    }
}
```

Assignment



Differences between URL and URLConnection.

- URLConnection provides access to the HTTP header.
- URLConnection can configure the request parameters sent to the server.
- URLConnection can write data to the server as well as read data from the server.

3. Reading the Header

- The `getHeaderField(String name)` method returns the string value of a named header field.
- Names are case-insensitive.
- If the requested field is not present, null is returned.
- `String lm = uc.getHeaderField("Last-modified");`

HTTP servers provide a amount of information in the header that precedes each response. For example, here's a typical HTTP header returned by an Apache web server:

```
HTTP/1.1 301 Moved Permanently
Date: Sun, 21 Apr 2013 15:12:46 GMT
Server: Apache
Location: http://www.ibiblio.org/
Content-Length: 296
Connection: close
Content-Type: text/html; charset=iso-8859-1
```

i. Retrieving Specific Header Fields

The first six methods request specific, particularly common fields from the header. These are:

1. Content-type
2. Content-length
3. Content-encoding
4. Date
5. Last-modified
6. Expires

i. Retrieving Specific Header Fields

Six Convenience Methods

These return the values of six particularly common header fields:

1. `public int getLength()`
2. `public String getContentType()`
3. `public String getEncoding()`
4. `public long getExpiration()`
5. `public long getDate()`
6. `public long getLastModified()`

Example 7-4. Return the header

```
import java.io.*;
import java.net.*;
import java.util.*;
public class HeaderViewer {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            try {
                URL u = new URL(args[i]);
                URLConnection uc = u.openConnection();
                System.out.println("Content-type: " + uc.getContentType());
                if (uc.getContentEncoding() != null) {
                    System.out.println("Content-encoding: "
                        + uc.getContentEncoding());
                }
                if (uc.getDate() != 0) {
                    System.out.println("Date: " + new Date(uc.getDate()));
                }
                if (uc.getLastModified() != 0) {
                    System.out.println("Last modified: "
                        + new Date(uc.getLastModified()));
                }
                if (uc.getExpiration() != 0) {
                    System.out.println("Expiration date: "
                        + new Date(uc.getExpiration()));
                }
                if (uc.getContentLength() != -1) {
                    System.out.println("Content-length: " + uc.getContentLength());
                }
            } catch (MalformedURLException ex) {
                System.err.println(args[i] + " is not a URL I understand");
            } catch (IOException ex) {
                System.err.println(ex);
            }
            System.out.println();
        }
    }
}
```

ii. Retrieving Arbitrary Header Fields

`public String getHeaderField(String name)`

- The `getHeaderField(String name)` method returns the string value of a named header field.
- Names are case-insensitive.
- If the requested field is not present, null is returned.

For example, to get the value of the Content-type and Content-encoding header fields of a `URLConnection` object `uc`, you could write:

```
String contentType = uc.getHeaderField("content-type");  
String contentEncoding = uc.getHeaderField("content-encoding");
```

To get the Date, Content-length, or Expires headers, you'd do the same:

```
String data = uc.getHeaderField("date");  
String expires = uc.getHeaderField("expires");  
String contentLength = uc.getHeaderField("Content-length");
```

ii. Retrieving Arbitrary Header Fields

```
public String getHeaderFieldKey(int n)
```

- The keys of the header fields are returned by the `getHeaderFieldKey(int n)` method.
- The first field is 1.
- If a numbered key is not found, null is returned.
- You can use this in combination with `getHeaderField()` to loop through the complete header

For example, in order to get the sixth key of the header of the `URLConnection uc`, you would write:

```
String header6 = uc.getHeaderFieldKey(6);
```

```
public String getHeaderField(int n)
```

This method returns the value of the *nth header field*. In HTTP, the starter line containing the request method and path is header field zero and the first actual header is one.

Example 7-5. Print the entire HTTP header

```
import java.io.*;
import java.net.*;
public class AllHeaders {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            try {
                URL u = new URL(args[i]);
                URLConnection uc = u.openConnection();
                for (int j = 1; ; j++) {
                    String header = uc.getHeaderField(j);
                    if (header == null) break;
                    System.out.println(uc.getHeaderFieldKey(j) + ": " + header);
                }
            } catch (MalformedURLException ex) {
                System.err.println(args[i] + " is not a URL I understand.");
            } catch (IOException ex) {
                System.err.println(ex);
            }
            System.out.println();
        }
    }
}
```

For example, here's the output when this program is run against *<http://www.oreilly.com>*:

ii. Retrieving Arbitrary Header Fields

```
public long getHeaderFieldDate(String name, long default)
```

For example:

```
Date expires = new Date(uc.getHeaderFieldDate("expires", 0));  
long lastModified = uc.getHeaderFieldDate("last-modified", 0);  
Date now = new Date(uc.getHeaderFieldDate("date", 0));
```

You can use the methods of the `java.util.Date` class to convert the `long` to a `String`.

```
public int getHeaderFieldInt(String name, int default)
```

For example, to get the content length from a `URLConnection uc`, you would write:

```
int contentLength = uc.getHeaderFieldInt("content-length", -1);
```

In this code fragment, `getHeaderFieldInt()` returns `-1` if the `Content-length` header isn't present.

4. Caches



Web browsers have been caching pages and images for years.

If a logo is repeated on every page of a site, the browser normally loads it from the remote server only once, stores it in its cache, and reloads it from the cache whenever it's needed rather than requesting it from the remote server every time the logo is encountered.

Several HTTP headers, including Expires and Cache-control, can control caching.

4. Caches



However, HTTP headers can adjust this:

- An Expires header (primarily for HTTP 1.0) indicates that it's OK to cache this representation until the specified time.
- The Cache-control header (HTTP 1.1) offers fine-grained cache policies:
 - max-age**=[seconds]: *Number of seconds from now before the cached entry should expire*
 - s-maxage**=[seconds]: *Number of seconds from now before the cached entry should expire from a shared cache. Private caches can store the entry for longer.*
 - public**: OK to cache an authenticated response. Otherwise authenticated responses are not cached.
 - private**: Only single user caches should store the response; shared caches should not.
 - no-cache**: Not quite what it sounds like. The entry may still be cached, but the client should reverify the state of the resource with an ETag or Last-modified header on each access.
 - no-store**: Do not cache the entry no matter what.

4. Caches

- **The Last-modified header** is the date when the resource was last changed. A client can use a HEAD request to check this and only come back for a full GET if its local cached copy is older than the Last-modified date.
- **The ETag header** (entity tag HTTP 1.1) is a unique identifier for the resource that changes when the resource does. A client can use a HEAD request to check this and only come back for a full GET if its local cached copy has a different ETag.

For example, this HTTP response says that the resource may be cached for 604,800 seconds (HTTP 1.1) or one week later (HTTP 1.0). It also says it was last modified on April 20 and has an ETag, so if the local cache already has a copy more recent than that, there's no need to load the whole document now:

```
HTTP/1.1 200 OK
Date: Sun, 21 Apr 2013 15:12:46 GMT
Server: Apache
Connection: close
Content-Type: text/html; charset=ISO-8859-1
Cache-control: max-age=604800
Expires: Sun, 28 Apr 2013 15:12:46 GMT
Last-modified: Sat, 20 Apr 2013 09:55:04 GMT
ETag: "67099097696afcf1b67e"
```

i. Web Cache for Java



By default, Java does not cache anything. To install a system-wide cache of the URL class will use, you need the following:

- A concrete subclass of ResponseCache
- A concrete subclass of CacheRequest
- A concrete subclass of CacheResponse

Two abstract methods in the ResponseCache class store and retrieve data from the system's single cache:

```
public abstract CacheResponse get(URL uri, String requestMethod,  
Map<String, List<String>> requestHeaders) throws IOException
```

```
public abstract CacheRequest put(URL uri, URLConnection connection)  
throws IOException
```

i. Web Cache for Java

Example 7-7. The CacheRequest class

```
package java.net;  
  
public abstract class CacheRequest {  
    public abstract OutputStream getBody() throws IOException;  
    public abstract void abort();  
}
```

i. Web Cache for Java

Example 7-8. A concrete CacheRequest subclass

```
import java.io.*;
import java.net.*;
public class SimpleCacheRequest extends CacheRequest {
    private ByteArrayOutputStream out = new ByteArrayOutputStream();
    @Override
    public OutputStream getBody() throws IOException {
        return out;
    }
    @Override
    public void abort() {
        out.reset();
    }
    public byte[] getData() {
        if (out.size() == 0) return null;
        else return out.toByteArray();
    }
}
```


i. Web Cache for Java

Example 7-9. The CacheResponse class

```
public abstract class CacheResponse {  
    public abstract Map<String, List<String>> getHeaders() throws  
        IOException;  
    public abstract InputStream getBody() throws IOException;  
}
```

To install or change the cache, use the static `ResponseCache.setDefault()` and `ResponseCache.getDefault()` methods:

```
public static ResponseCache getDefault()  
public static void setDefault(ResponseCache responseCache)
```

Eg:

```
ResponseCache.setDefault(new MemoryCache());
```

5. Configuring the Connection

- 1) protected URL url
- 2) protected boolean connected
- 3) protected boolean allowUserInteraction
- 4) protected boolean doInput
- 5) protected boolean doOutput
- 6) protected boolean ifModifiedSince
- 7) protected boolean useCaches
- 8) Timeouts

5. Configuring the Connection

The `URLConnection` class has seven protected instance fields that define exactly how the client makes the request to the server. These are:

```
protected URL url;  
protected boolean doInput = true;  
protected boolean doOutput = false;  
protected boolean allowUserInteraction = defaultAllowUserInteraction;  
protected boolean useCaches = defaultUseCaches;  
protected long ifModifiedSince = 0;  
protected boolean connected = false;
```

5. Configuring the Connection

Because these fields are all protected, their values are accessed and modified via obviously named setter and getter methods:

```
public URL getURL()
public void setDoInput(boolean doInput)
public boolean getDoInput()
public void setDoOutput(boolean doOutput)
public boolean getDoOutput()
public void setAllowUserInteraction(boolean allowUserInteraction)
public boolean getAllowUserInteraction()
public void setUseCaches(boolean useCaches)
public boolean getUseCaches()
public void setIfModifiedSince(long ifModifiedSince)
public long getIfModifiedSince()
```

5. Configuring the Connection

There are also some getter and setter methods that define the default behavior for all instances of `URLConnection`. These are:

```
public boolean getDefaultUseCaches()  
public void setDefaultUseCaches(boolean defaultUseCaches)  
public static void setDefaultAllowUserInteraction(  
boolean defaultAllowUserInteraction)  
public static boolean getDefaultAllowUserInteraction()  
public static FileNameMap getFileNameMap()  
public static void setFileNameMap(FileNameMap map)
```

5. Configuring the Connection

protected URL url

The url field specifies the URL that this URLConnection connects to. The constructor sets it when the URLConnection is created and it should not change thereafter. You can retrieve the value by calling the `getURL()` method.

Example 7-12. Print the URL of a URLConnection to `http://www.oreilly.com/`

```
import java.io.*;
import java.net.*;
public class URLPrinter {
    public static void main(String[] args) {
        try {
            URL u = new URL("http://www.oreilly.com/");
            URLConnection uc = u.openConnection();
            System.out.println(uc.getURL());
        } catch (IOException ex) {
            System.err.println(ex);
        }
    }
}
```

O/P:

```
% java URLPrinter
```

```
http://www.oreilly.com/
```

5. Configuring the Connection

protected URL url

The url field specifies the URL that this URLConnection connects to. The constructor sets it when the URLConnection is created and it should not change thereafter. You can retrieve the value by calling the `getURL()` method.

Example 7-12. Print the URL of a URLConnection to `http://www.oreilly.com/`

```
import java.io.*;
import java.net.*;
public class URLPrinter {
    public static void main(String[] args) {
        try {
            URL u = new URL("http://www.oreilly.com/");
            URLConnection uc = u.openConnection();
            System.out.println(uc.getURL());
        } catch (IOException ex) {
            System.err.println(ex);
        }
    }
}
```

O/P:

```
% java URLPrinter
```

```
http://www.oreilly.com/
```

5. Configuring the Connection

Timeouts

Four methods query and modify the timeout values for connections; that is, how long the underlying socket will wait for a response from the remote end before throwing a `SocketTimeoutException`. These are:

```
public void setConnectTimeout(int timeout)
public int getConnectTimeout()
public void setReadTimeout(int timeout)
public int getReadTimeout()
```

For example, this code fragment requests a 30-second connect timeout and a 45-second read timeout:

```
URL u = new URL("http://www.example.org");
URLConnection uc = u.openConnection();
uc.setConnectTimeout(30000);
uc.setReadTimeout(45000);
```


Configuring the Client Request HTTP Header

An HTTP client (e.g., a browser) sends the server a request line and a header. For example, here's an HTTP header that Chrome sends:

```
Accept:text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Charset:ISO-8859-1,utf-8;q=0.7,*;q=0.3
Accept-Encoding:gzip,deflate,sdch
Accept-Language:en-US,en;q=0.8
Cache-Control:max-age=0
Connection:keep-alive
Cookie:reddit_first=%7B%22firsttime%22%3A%20%22first%22%7D
Host:lesswrong.com
User-Agent:Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_3) AppleWebKit/537.31
(KHTML, like Gecko) Chrome/26.0.1410.65 Safari/537.31
```

✓ **public void setRequestProperty(String name, String value)**

✓ `uc.setRequestProperty("Cookie", "username=elharo;
password=ACD0X9F23JJn6G; session=100678945");`

whenever the client requests a URL from that server, it includes a Cookie field in the HTTP request header that looks like this:

```
Cookie: username=elharo; password=ACD0X9F23JJn6G; session=100678945;
```

✓ **public void addRequestProperty(String name, String value)**

✓ **public String getRequestProperty(String name)**

✓ **public Map<String,List<String>> getRequestProperties()**

7. Writing Data to a Server

Posting data to a form requires these steps:

1. Decide what name-value pairs you'll send to the server-side program.
2. Write the server-side program that will accept and process the request. If it doesn't use any custom data encoding, you can test this program using a regular HTML form and a web browser.
3. Create a query string in your Java program. The string should look like this:
`name1=value1&name2=value2&name3=value3`
Pass each name and value in the query string to `URLEncoder.encode()` before adding it to the query string.
4. Open a `URLConnection` to the URL of the program that will accept the data.
5. Set `doOutput` to true by invoking `setDoOutput(true)`.
6. Write the query string onto the `URLConnection`'s `OutputStream`.
7. Close the `URLConnection`'s `OutputStream`.
8. Read the server response from the `URLConnection`'s `InputStream`.

8. Security Considerations for URLConnections

URLConnection objects are subject to all the usual security restrictions about making network connections, reading or writing files, and so forth.

Before attempting to connect a URL, you may want to know whether the connection will be allowed. For this purpose, the URLConnection class has a `getPermission()` method:

```
public Permission getPermission() throws IOException
```

This returns a `java.security.Permission` object that specifies what permission is needed to connect to the URL. It returns null if no permission is needed (e.g., there's no security manager in place). Subclasses of URLConnection return different subclasses of `java.security.Permission`.

9. Guessing MIME Media Types

The `URLConnection` class provides two static methods to help programs figure out the MIME type of some data; you can use these if the content type just isn't available or if you have reason to believe that the content type you're given isn't correct. The first of these is `URLConnection.guessContentTypeFromName()`:

```
public static String guessContentTypeFromName(String name)
```

This method tries to guess the content type of an object based upon the extension in the filename portion of the object's URL.

For instance, it omits various XML applications such as RDF (*.rdf*), XSL (*.xsl*), and so on that should have the MIME type `application/xml`. It also doesn't provide a MIME type for CSS stylesheets (*.css*).

The second MIME type guesser method is `URLConnection.guessContentTypeFromStream()`:

```
public static String guessContentTypeFromStream(InputStream in)
```

This method tries to guess the content type by looking at the first few bytes of data in the stream.

For example, an XML document that begins with a comment rather than an XML declaration would be mislabeled as an HTML file.

10. HttpURLConnection

1. The Request Method
2. Disconnecting from the Server
3. Handling Server Responses
4. Proxies
5. Streaming Mode

10. HttpURLConnection

The `java.net.HttpURLConnection` class is an abstract subclass of `URLConnection`; it provides some additional methods that are helpful when working specifically with `httpURLs`.

Cast that `URLConnection` to `HttpURLConnection` like this:

```
URL u = new URL("http://lesswrong.com/");  
URLConnection uc = u.openConnection();  
HttpURLConnection http = (HttpURLConnection) uc;
```

Or, skipping a step, like this:

```
URL u = new URL("http://lesswrong.com/");  
HttpURLConnection http = (HttpURLConnection) u.openConnection();
```

The Request Method

When a web client contacts a web server, the first thing it sends is a request line.

Typically, this line begins with GET and is followed by the path of the resource that the client wants to retrieve and the version of the HTTP protocol that the client understands.

For example:

```
GET /catalog/jfcnut/index.html HTTP/1.0
```

For example, here's how a browser asks for just the header of a document using HEAD:

```
HEAD /catalog/jfcnut/index.html HTTP/1.1
```

```
Host: www.oreilly.com
```

```
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
```

```
Connection: close
```

By default, HttpURLConnection uses the GET method.

The Request Method

When a web client contacts a web server, the first thing it sends is a request line.

Typically, this line begins with GET and is followed by the path of the resource that the client wants to retrieve and the version of the HTTP protocol that the client understands.

For example:

```
GET /catalog/jfcnut/index.html HTTP/1.0
```

For example, here's how a browser asks for just the header of a document using HEAD:

```
HEAD /catalog/jfcnut/index.html HTTP/1.1
```

```
Host: www.oreilly.com
```

```
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
```

```
Connection: close
```

By default, HttpURLConnection uses the GET method.

We can change this with the `setRequestMethod()` method:

`public void setRequestMethod(String method) throws ProtocolException`

The Request Method

The method argument should be one of these seven case-sensitive strings:

- GET
- POST
- HEAD
- PUT
- DELETE
- OPTIONS
- TRACE

Head

Example 7-15. Get the time when a URL was last changed

```
import java.io.*;
import java.net.*;
import java.util.*;
public class LastModified {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            try {
                URL u = new URL(args[i]);
                HttpURLConnection http = (HttpURLConnection) u.openConnection();
                http.setRequestMethod("HEAD");
                System.out.println(u + " was last modified at " + new Date(http.getLastModified()));
            } catch (MalformedURLException ex) {
                System.err.println(args[i] + " is not a URL I understand");
            } catch (IOException ex) {
                System.err.println(ex);
            }
        }
        System.out.println();
    }
}
```

Here's the output from one run:

```
$ java LastModified http://www.ibiblio.org/xml/
http://www.ibiblio.org/xml/ was last modified at Tue Apr 06 07:45:29 EDT 2010
```

Handling Server Responses

The first line of an HTTP server's response includes a numeric code and a message indicating what sort of response is made. For instance, the most common response is 200 OK, indicating that the requested document was found. For example:

```
HTTP/1.1 200 OK
Cache-Control:max-age=3, must-revalidate
Connection:Keep-Alive
Content-Type:text/html; charset=UTF-8
Date:Sat, 04 May 2013 14:01:16 GMT
Keep-Alive:timeout=5, max=200
Server:Apache
Transfer-Encoding:chunked
Vary:Accept-Encoding, Cookie
```

```
WP-Super-Cache:Served supercache file from PHP
<HTML>
<HEAD>
rest of document follows...
```

Handling Server Responses

Another response that you're undoubtedly all too familiar with is 404 Not Found, indicating that the URL you requested no longer points to a document. For example:

```
HTTP/1.1 404 Not Found
Date: Sat, 04 May 2013 14:05:43 GMT
Server: Apache
Last-Modified: Sat, 12 Jan 2013 00:19:15 GMT
ETag: "375933-2b9e-4d30c5cb0c6c0;4d02eaff53b80"
Accept-Ranges: bytes
Content-Length: 11166
Connection: close
Content-Type: text/html; charset=ISO-8859-1
```

```
<html>
<head>
<title>Lost ... and lost</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
```

Handling Server Responses

```
public int getResponseCode() throws IOException
```

```
public String getResponseMessage() throws IOException
```

Error conditions:

The `getErrorStream()` method returns an `InputStream` containing this page or `null` if no error was encountered or no data returned:

```
public InputStream getErrorStream()
```

Redirects

```
public static boolean getFollowRedirects()
```

```
public static void setFollowRedirects(boolean follow)
```

```
public boolean getInstanceFollowRedirects()
```

```
public void setInstanceFollowRedirects(boolean followRedirects)
```

Proxies

```
public abstract boolean usingProxy()
```

Streaming Mode: (not known size)

```
public void setChunkedStreamingMode(int chunkLength)
```

```
public void setFixedLengthStreamingMode(int contentLength)
```

```
public void setFixedLengthStreamingMode(long contentLength) // Java 7
```

Example 7-16. A SourceViewer that includes the response code and message

```
import java.io.*;
import java.net.*;
public class SourceViewer3 {
    public static void main (String[] args) {
        for (int i = 0; i < args.length; i++) {
            try {
                // Open the URLConnection for reading
                URL u = new URL(args[i]);
                HttpURLConnection uc = (HttpURLConnection) u.openConnection();
                int code = uc.getResponseCode();
                String response = uc.getResponseMessage();
                System.out.println("HTTP/1.x " + code + " " + response);
                for (int j = 1; ; j++) {
                    String header = uc.getHeaderField(j);
                    String key = uc.getHeaderFieldKey(j);
                    if (header == null || key == null) break;
                    System.out.println(uc.getHeaderFieldKey(j) + ": " + header);
                }
                System.out.println();
                try (InputStream in = new BufferedInputStream(uc.getInputStream())) {
                    // chain the InputStream to a Reader
                    Reader r = new InputStreamReader(in);
                    int c;
                    while ((c = r.read()) != -1) {
                        System.out.print((char) c);
                    }
                }
                } catch (MalformedURLException ex) {
                    System.err.println(args[0] + " is not a parseable URL");
                } catch (IOException ex) {
                    System.err.println(ex);
                }
            }
        }
    }
}
```

There's currently no method to specifically return that. In this example, you just fake it as "HTTP/1.x," like this:

Example 7-17. Download a web page with a URLConnection

```
import java.io.*;
import java.net.*;
public class SourceViewer4 {
    public static void main (String[] args) {
        try {
            URL u = new URL(args[0]);
            HttpURLConnection uc = (HttpURLConnection) u.openConnection();
            try (InputStream raw = uc.getInputStream()) {
                printFromStream(raw);
            } catch (IOException ex) {
                printFromStream(uc.getErrorStream());
            }
            } catch (MalformedURLException ex) {
                System.err.println(args[0] + " is not a parseable URL");
            } catch (IOException ex) {
                System.err.println(ex);
            }
        }
        private static void printFromStream(InputStream raw) throws IOException {
            try (InputStream buffer = new BufferedInputStream(raw)) {
                Reader reader = new InputStreamReader(buffer);
                int c;
                while ((c = reader.read()) != -1) {
                    System.out.print((char) c);
                }
            }
        }
    }
}
```