

UNIT 5

URLConnections

(5 Hours) — Network Programming (CAC355)

Disclaimer: These notes were generated automatically from the CACS355 syllabus and may contain errors or omissions. Please verify important details (syntax, method names, default values) against your textbook (Java Network Programming, E. R. Harold) and class notes before the exam.

5.1 Opening URLConnections

`java.net.URLConnection` is an abstract class that gives **more control** than `URL.openConnection()`: you can read headers, set request headers, send data and configure timeouts. The concrete subclass for HTTP is `HttpURLConnection`.

Basic steps

- Create a **URL** object.
- Call **`url.openConnection()`** — this creates the object but does **not** connect yet.
- Configure the connection (timeouts, request headers, `setDoOutput` ...) — must be done *before* connecting.
- Read the response headers and/or the data (this implicitly connects), or call **`connect()`** explicitly.

```
URL u = new URL("http://www.example.com/");
URLConnection uc = u.openConnection();
uc.setConnectTimeout(5000);
try (InputStream in = uc.getInputStream()) {
    // read data
}
```

5.2 Reading Data from the Server

`getInputStream()` returns a stream of the response **body** (headers are stripped and are read with the header methods). Wrap it in a `Reader` with the correct charset — taken from the `Content-Type` header — for text.

```

URLConnection uc = new URL("http://www.example.com/").openConnection();
String ct = uc.getContentType();           // e.g. text/html; charset=UTF-8
String enc = "ISO-8859-1";
int i = ct == null ? -1 : ct.indexOf("charset=");
if (i >= 0) enc = ct.substring(i + 8);
try (Reader r = new InputStreamReader(uc.getInputStream(), enc)) {
    int c;
    while ((c = r.read()) != -1) System.out.print((char) c);
}

```

Listing 5.1 — Reading data using the charset from the header

Binary data (images, files) should be read as bytes with `read(byte[])` and, since the server may send fewer bytes than requested, in a loop until it returns `-1`. `getContentLength()` gives the size (`-1` if unknown).

5.3 Reading Headers

Retrieving specific header fields

Method	Header
<code>getContentType()</code>	Content-Type (MIME type and charset)
<code>getContentLength()</code> / <code>getContentLengthLong()</code>	Content-Length
<code>getContentEncoding()</code>	Content-Encoding (e.g., gzip)
<code>getDate()</code>	Date – when the response was sent
<code>getExpiration()</code>	Expires
<code>getLastModified()</code>	Last-Modified
<code>getHeaderField(String name)</code>	Any named header (case-insensitive), or null
<code>getHeaderFieldInt(name, default)</code> , <code>getHeaderFieldDate(name, default)</code>	Header parsed as int / date

Retrieving arbitrary header fields

- **`getHeaderFieldKey(int n)`** — name of the *n*th header (index 0 has a null key: the status line).
- **`getHeaderField(int n)`** — value of the *n*th header.
- **`getHeaderFields()`** — a `Map<String, List<String>>` of all headers.

```

for (int n = 1; ; n++) {
    String key = uc.getHeaderFieldKey(n);
    if (key == null) break;
    System.out.println(key + ": " + uc.getHeaderField(n));
}

```

Listing 5.2 — Printing all response headers

5.4 Cache

By default Java does **not** cache responses. A cache is added by installing a subclass of `ResponseCache` with `ResponseCache.setDefault()`. It has two abstract methods:

- **CacheResponse get(URL, String requestMethod, Map requestHeaders)** — return a stored response, or null if not cached.
- **CacheRequest put(URL, URLConnection)** — called when a response arrives; return a `CacheRequest` that receives the body to store it, or null to skip caching.

CacheRequest provides `getBody()` (an `OutputStream` to write into) and `abort()`. **CacheResponse** provides `getHeaders()` and `getBody()` (an `InputStream`). The cache should respect headers such as `Cache-Control`, `Expires`, `Last-Modified` and `ETag`, and not cache things marked *no-store* or *private*.

Web cache for Java

A working cache typically stores responses in memory or on disk (keyed by URI), checks expiry from the headers, and returns the stored copy without contacting the server. A simple in-memory cache can use a `ConcurrentHashMap<URI, byte[]>`. Java's built-in HTTP client can also use `useCaches` (see 5.5) to allow or forbid caching per connection.

5.5 Configuring the Connection

`URLConnection` has protected fields with public getters and setters. Changes must be made **before** the connection is opened (otherwise `IllegalStateException`).

Field	Getter / Setter	Meaning
protected URL url	<code>getURL()</code>	The URL being connected to (final)
protected boolean connected	—	True once the connection is open (after <code>connect()</code>)
protected boolean allowUserInteraction	<code>get/setAllowUserInteraction()</code>	Allow a dialog for authentication; default false
protected boolean doInput	<code>get/setDoInput()</code>	Will the program read from the server? Default true

Field	Getter / Setter	Meaning
protected boolean doOutput	get/setDoOutput()	Will the program write (send a body)? Default false; setting true switches HTTP GET to POST
protected long ifModifiedSince	get/setIfModifiedSince(long)	Only fetch if changed after this time (ms since 1970); server may reply 304
protected boolean useCaches	get/setUseCaches()	Allow use of a cached copy; default true

*Note: The syllabus lists `ifModificationSince` and several fields as “boolean”; in the real API `ifModifiedSince` is a **long** (a time). Please check your textbook for exact wording.*

Timeouts

- **setConnectTimeout(int ms)** — maximum time to establish the connection; on expiry a `SocketTimeoutException` is thrown.
- **setReadTimeout(int ms)** — maximum time to wait for data while reading.
- The default of 0 means wait forever — always set timeouts in real programs.

5.6 Configuring the Client Request HTTP Header

Request headers are set with `setRequestProperty(String name, String value)` (replaces) or `addRequestProperty()` (adds another value). `getRequestProperty(name)` and `getRequestProperties()` read them. They must be set before connecting.

Header	Purpose
User-Agent	Identifies the client program
Accept	Content types the client accepts
Accept-Language / Accept-Encoding	Preferred language / compression
Authorization	Credentials (e.g., Basic base64(user:pw))
Cookie	Cookies to send
Content-Type / Content-Length	Describe a request body
If-Modified-Since	Conditional GET

```
uc.setRequestProperty("User-Agent", "MyClient/1.0");
uc.setRequestProperty("Accept", "text/html");
```

5.7 Security Considerations for URLConnections

- Untrusted code (applets) run under the **SecurityManager** and can normally connect only back to the host they came from; connecting elsewhere needs a `SocketPermission`.
- Local files (file: URLs) require `FilePermission`.
- Before connecting, `getPermission()` tells which `Permission` is needed.
- Never trust redirects, data or headers from the server; validate everything and use HTTPS for sensitive data.
- Do not put secrets in query strings; use timeouts to avoid denial-of-service by slow servers.

Note: The SecurityManager and applets are deprecated in modern Java, so this section is mostly of historical interest.

5.8 Guessing MIME Media Types

When a server sends no (or a wrong) Content-Type, Java can guess from the file name or the first bytes:

- **URLConnection.guessContentTypeFromName(String fileName)** — from the extension (.html → text/html, .gif → image/gif).
- **URLConnection.guessContentTypeFromStream(InputStream)** — from the magic bytes at the start; the stream must support `mark()` and `reset()`.
- **getFileNameMap() / setFileNameMap()** — the mapping between extensions and MIME types.

```
String t = URLConnection.guessContentTypeFromName("photo.jpg"); // image/jpeg
```

5.9 HttpURLConnection

`HttpURLConnection` extends `URLConnection` with HTTP-specific features. Cast the result: `(HttpURLConnection) url.openConnection()`.

The request methods

`setRequestMethod(String)` sets GET (default), POST, HEAD, PUT, DELETE, OPTIONS or TRACE. For POST/PUT call `setDoOutput(true)` and write the body to `getOutputStream()`.

```

URLConnection h = (URLConnection) new URL("http://www.example.com/login").openConne
h.setRequestMethod("POST");
h.setDoOutput(true);
h.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
try (OutputStream o = h.getOutputStream()) {
    o.write("user=ram&pw=1234".getBytes("UTF-8"));
}
System.out.println(h.getResponseCode() + " " + h.getResponseMessage());

```

Listing 5.3 — Sending a POST request

Disconnecting from the server

`disconnect()` indicates that no more requests are needed soon and may close the underlying persistent connection. Closing the input stream normally returns the connection to the keep-alive pool instead.

Handling server responses

- **`getResponseCode()`** and **`getResponseMessage()`** — status code (200, 404 ...) and text.
- **`getInputStream()`** throws `IOException` for 4xx/5xx codes; use **`getErrorStream()`** to read the error page body.
- **Redirects:** 3xx responses are followed automatically by default (`setInstanceFollowRedirects(true)`; static `setFollowRedirects`). Redirects between http and https are not followed automatically.

Proxies

`usingProxy()` tells whether the connection goes through a proxy. A proxy for one connection is given by `url.openConnection(Proxy)` (Unit 3).

Streaming mode

Normally the whole request body is **buffered in memory** so Content-Length can be sent first. For large uploads this wastes memory and delays sending. Streaming mode avoids that:

- **`setFixedLengthStreamingMode(long length)`** — you know the body size in advance.
- **`setChunkedStreamingMode(int chunkSize)`** — size unknown; the body is sent using chunked transfer encoding.
- Trade-off: automatic authentication and redirect handling cannot resend the body.

Key Takeaways

- ✓ `URLConnection` = URL + control: `openConnection()`, configure, then read headers/data.
- ✓ Header getters: `getContentType`, `getContentLength`, `getDate`, `getLastModified`, `getHeaderField`, `getHeaderFields`.
- ✓ Cache: install a `ResponseCache` with `get()` and `put()`; `useCaches` controls per-connection use.
- ✓ Configure before connecting: `doInput`, `doOutput`, `ifModifiedSince`, `useCaches`, `timeouts`, `setRequestProperty`.
- ✓ Guess MIME types with `guessContentTypeFromName` / `FromStream`.
- ✓ `HttpURLConnection`: `setRequestMethod`, `getResponseCode`, `getErrorStream`, `redirects`, `disconnect`, `streaming` modes.

— *End of Unit 5* —

bcalink.com