

# UNIT 4

## HTTP

(2 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.*

### 4.1 The Protocol

**HTTP (HyperText Transfer Protocol)** is an application-layer, text-based, **request-response** protocol used by the Web. A client (browser) sends a request to a server (default port 80; 443 for HTTPS over TLS) and the server returns a response. HTTP runs over TCP and is **stateless**: each request is independent of earlier ones.

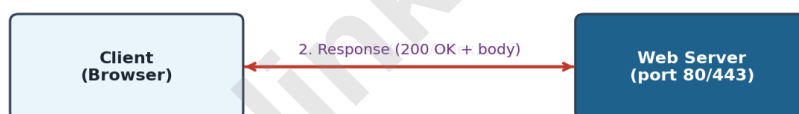


Figure 4.1 — HTTP request-response cycle

#### Request message format

```
GET /index.html HTTP/1.1      <- request line: method, path, version
Host: www.example.com        <- headers (name: value)
User-Agent: MyJavaClient/1.0
Accept: text/html

                                <- blank line ends the headers
(optional request body)
```

#### Response message format

```
HTTP/1.1 200 OK               <- status line: version, code, reason
Content-Type: text/html
Content-Length: 1234

                                <- blank line
<html> ... </html>            <- response body
```

| Status class | Meaning       | Examples                                                        |
|--------------|---------------|-----------------------------------------------------------------|
| 1xx          | Informational | 100 Continue                                                    |
| 2xx          | Success       | 200 OK, 201 Created, 204 No Content                             |
| 3xx          | Redirection   | 301 Moved Permanently, 302 Found, 304 Not Modified              |
| 4xx          | Client error  | 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found |
| 5xx          | Server error  | 500 Internal Server Error, 503 Service Unavailable              |

## Keep-Alive (persistent connections)

In **HTTP/1.0** each request used a new TCP connection, which was wasteful because opening a connection costs a handshake and slow-start. **Keep-Alive** lets the same TCP connection carry several requests and responses one after another. In **HTTP/1.1** connections are persistent by default; a client or server can send `Connection: close` to end it, and in HTTP/1.0 the client asks with `Connection: keep-alive`.

- Saves the cost of repeated TCP handshakes and reduces latency and server load.
- The response must include **Content-Length** (or use chunked encoding) so the client knows where each response ends.
- Java's `URLConnection` reuses connections automatically; system properties `http.keepAlive` (default true) and `http.maxConnections` control this behaviour.
- Reuse works best if the response body is read fully and the stream is closed.

## 4.2 HTTP Methods

| Method  | Purpose                                            | Body?      | Safe / Idempotent |
|---------|----------------------------------------------------|------------|-------------------|
| GET     | Retrieve a resource                                | No         | Safe, idempotent  |
| HEAD    | Like GET but returns only headers                  | No         | Safe, idempotent  |
| POST    | Submit data to be processed; may create a resource | Yes        | Neither           |
| PUT     | Create or replace a resource at the URL            | Yes        | Idempotent        |
| DELETE  | Remove a resource                                  | Usually no | Idempotent        |
| OPTIONS | Ask which methods the server supports              | No         | Safe, idempotent  |
| TRACE   | Echo the request back (for diagnostics)            | No         | Safe              |
| PATCH   | Partially modify a resource                        | Yes        | Not guaranteed    |

*Note: **Safe** means it does not change server state; **idempotent** means repeating it has the same effect as doing it once.*

| GET                                      | POST                                          |
|------------------------------------------|-----------------------------------------------|
| Parameters in the URL query string       | Parameters in the request body                |
| Visible in URL, browser history and logs | Not shown in the URL                          |
| Length limited by URL limits             | No practical size limit                       |
| Can be cached and bookmarked             | Normally not cached                           |
| For reading data                         | For sending or changing data, uploading forms |

### 4.3 The Request Body

Methods such as POST, PUT and PATCH send data in the **request body**, after the blank line that ends the headers. The headers **Content-Type** (format of the data) and **Content-Length** (number of bytes) describe it.

| Content-Type                      | Body format                                                          |
|-----------------------------------|----------------------------------------------------------------------|
| application/x-www-form-urlencoded | name=value pairs joined with &, URL-encoded (default for HTML forms) |
| multipart/form-data               | Several parts separated by a boundary; used for file uploads         |
| application/json                  | JSON text                                                            |
| text/plain, text/xml              | Plain or XML text                                                    |

```
POST /login HTTP/1.1
Host: www.example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 27

user=ram&password=secret123
```

*Listing 4.1 — Raw POST request*

With Java you normally send the body using `HttpURLConnection`: call `setRequestMethod("POST")`, `setDoOutput(true)`, then write bytes to `getOutputStream()` (see Unit 5).

## 4.4 Cookies

Because HTTP is stateless, servers use **cookies** to remember clients. The server sends a `Set-Cookie` header; the client stores the cookie and returns it in a `Cookie` header on later requests to the same site.

```
HTTP/1.1 200 OK
Set-Cookie: sessionId=abc123; Path=/; Max-Age=3600; HttpOnly; Secure

GET /cart HTTP/1.1
Cookie: sessionId=abc123
```

| Attribute         | Meaning                                                    |
|-------------------|------------------------------------------------------------|
| Expires / Max-Age | When the cookie expires; without it it is a session cookie |
| Domain / Path     | Where the cookie is sent back                              |
| Secure            | Sent only over HTTPS                                       |
| HttpOnly          | Not accessible to JavaScript                               |
| SameSite          | Limits sending in cross-site requests                      |

### CookieManager

`java.net.CookieManager` (Java 6+) is the built-in implementation of `CookieHandler`. Once installed with `CookieHandler.setDefault(new CookieManager())`, the HTTP classes such as `URL` and `HttpURLConnection` automatically store cookies from responses and send them with later requests. A **CookiePolicy** decides which cookies are accepted: `ACCEPT_ALL`, `ACCEPT_NONE` or `ACCEPT_ORIGINAL_SERVER` (the default).

### CookieStore

`CookieStore` is the storage used by the manager, obtained with `manager.getCookieStore()`. The default one keeps cookies in memory only. Methods: `add(URI, HttpCookie)`, `get(URI)`, `getCookies()`, `getURIs()`, `remove(URI, HttpCookie)` and `removeAll()`. `HttpCookie` represents a single cookie (`getName()`, `getValue()`, `getDomain()`, `getPath()`, `getMaxAge()`, `getSecure()`).

```
CookieManager cm = new CookieManager(null, CookiePolicy.ACCEPT_ALL);
CookieHandler.setDefault(cm);
new URL("http://www.example.com/").openStream().close(); // cookies stored
for (HttpCookie c : cm.getCookieStore().getCookies())
    System.out.println(c.getName() + " = " + c.getValue());
```

### Key Takeaways

- ✓ HTTP is a stateless request–response protocol over TCP (port 80/443).
- ✓ A message = start line + headers + blank line + optional body; status codes are grouped 1xx–5xx.
- ✓ Keep-Alive reuses one TCP connection for several requests (default in HTTP/1.1).
- ✓ Methods: GET, HEAD, POST, PUT, DELETE, OPTIONS, TRACE, PATCH; GET is safe, PUT/DELETE idempotent.
- ✓ The request body is described by Content-Type and Content-Length.
- ✓ Cookies add state; CookieManager + CookieStore manage them automatically in Java.

— End of Unit 4 —

bcalink.com