PGT 307: HYPERTEXT TRANSFER PROTOCOL (HTTP)

Mohamed Elshaikh

Faculty of Electronics Engineering Technology – UniMAP (FTKEN-UniMAP)

Objectives

- **DESIGN** and **DISPLAY** how a web client talks to a server and how data transfer from the server back to the client.
  - The Protocol
  - HTTP Methods
  - Cookies

Hypertext Transfer Protocol (HTTP)

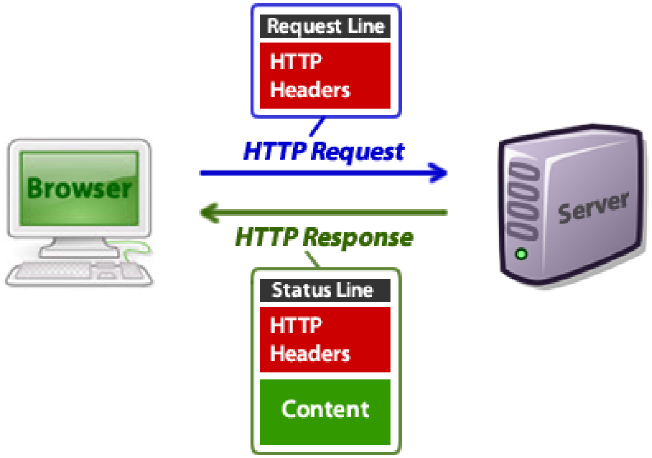
- A **standard** that defines how a web client talks to a server and how data is transferred from the server back to the client.
- Although HTTP is usually thought of as a means of transferring HTML files and the pictures embedded in them, HTTP is data format agnostic.
- It can be used to transfer TIFF pictures, Microsoft Word documents, Windows .exe files, or anything else that can be represented in bytes.
- To write programs that use HTTP, you’ll need to understand HTTP at a deeper level than the average web page designer.

The Protocol

- HTTP is the **standard protocol** for communication between web browsers and web servers.

- HTTP specifies
  - how a client and server establish a connection
  - how the client requests data from the server
  - how the server responds to that request
  - how the connection is closed
- HTTP connections use the TCP/IP protocol for data transfer.

Handling Request
(basic HTTP 1.0 procedure)

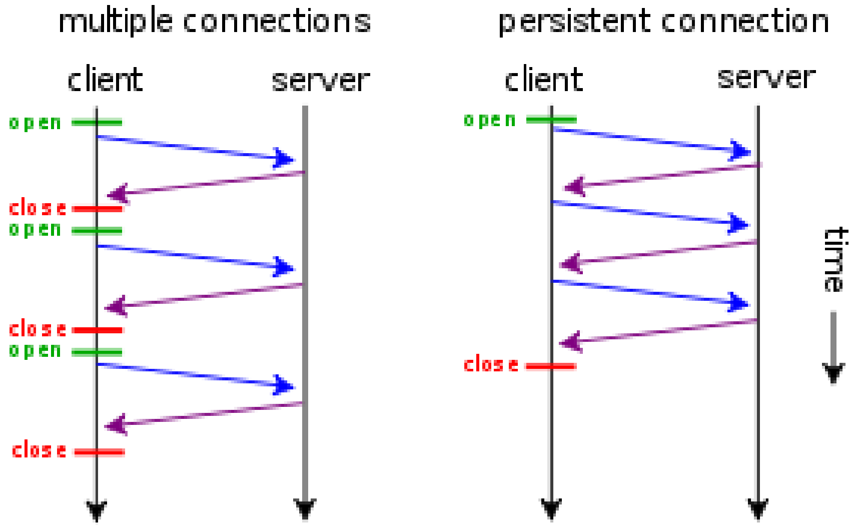


- For each request from client to server, there is a sequence of four steps:

1. The client **opens a TCP connection** to the server on **port 80**, by default; other ports may be specified in the URL.
2. The client **sends a message to the server** requesting the resource at a specified path.
   The request includes a header, and optionally (depending on the nature of the request) a blank line followed by data for the request.
3. The **server sends a response** to the client.
   The response begins with a response code, followed by a header full of metadata, a blank line, and the requested document or an error message.
4. The server closes the connection.

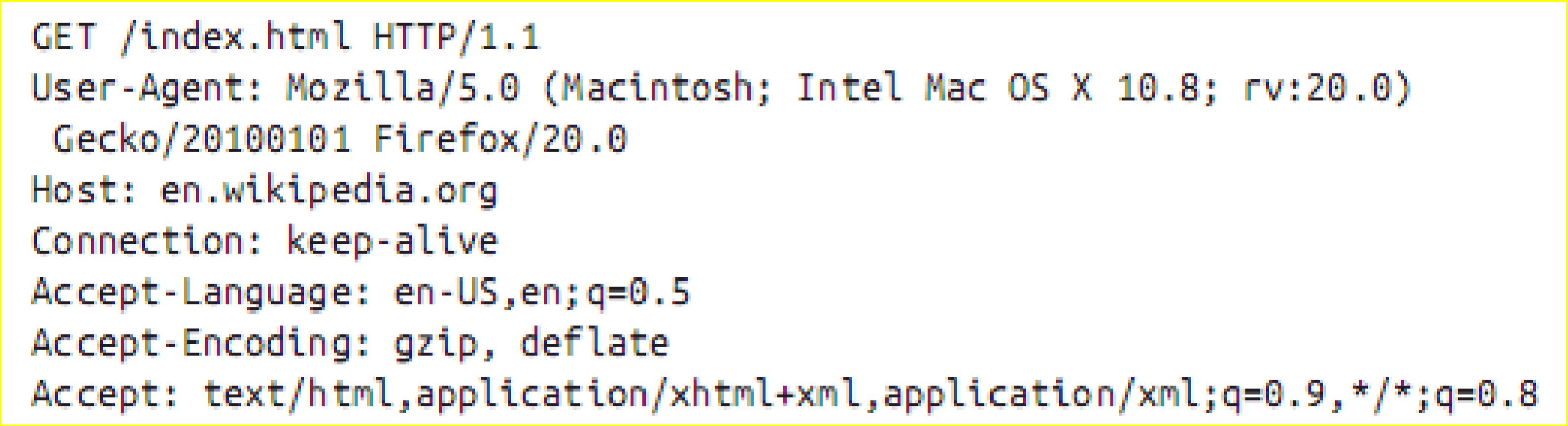
Handling Request (HTTP 1.1 →)

- In HTTP 1.1 and later, multiple requests and responses can be sent in series over a single TCP connection.
- Steps 2 and 3 can repeat multiple times in between steps 1 and 4.
- Furthermore, in HTTP 1.1, requests and responses can be sent in multiple chunks.



Basic Form

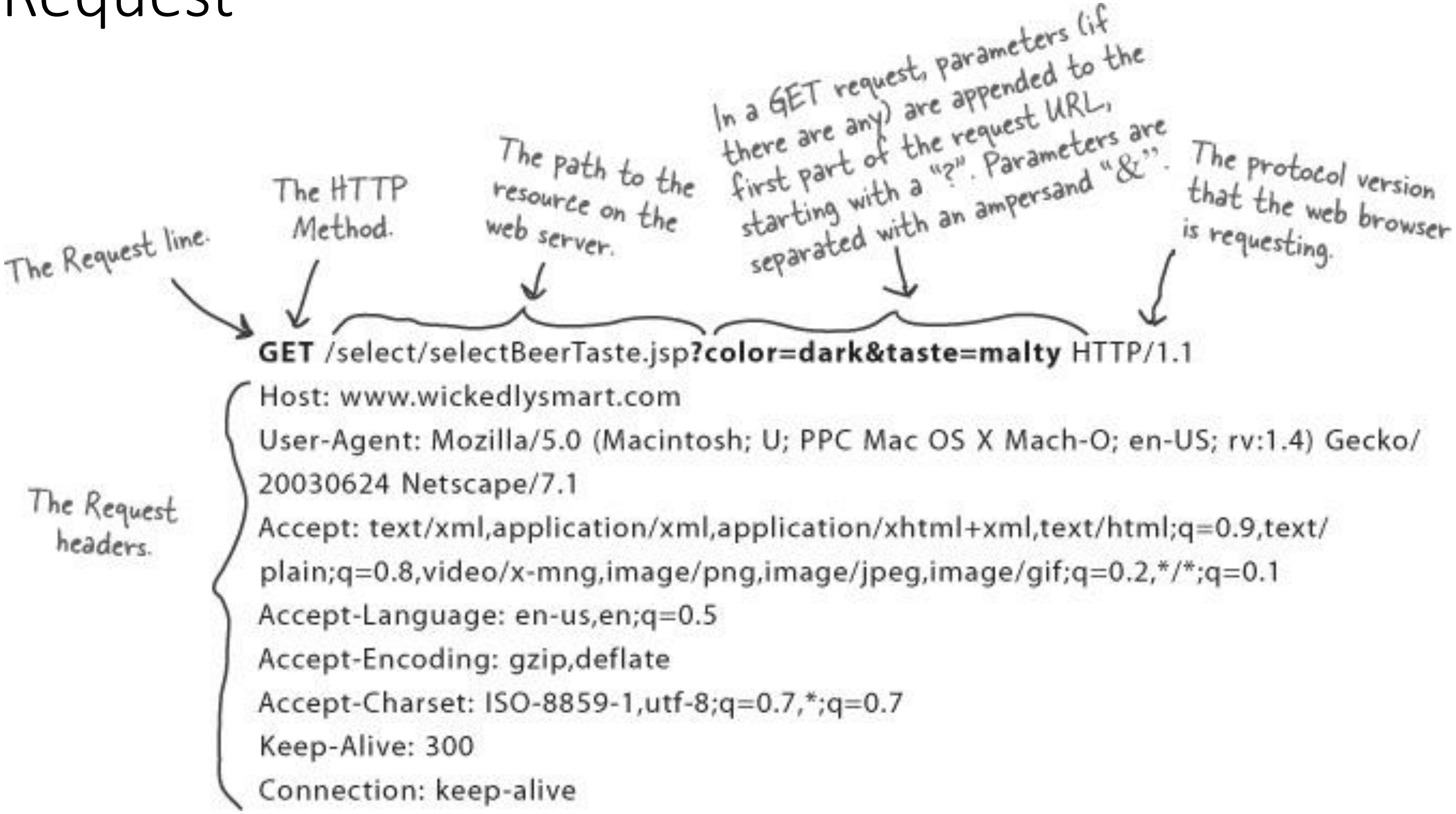
- Each request and response has the same basic form:
  - a header line,
  - an HTTP header containing metadata,
  - a blank line,
  - a message body.



```
GET /index.html HTTP/1.1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:20.0)
 Gecko/20100101 Firefox/20.0
Host: en.wikipedia.org
Connection: keep-alive
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

Example: A typical client request

Client Request



The Request line.

The HTTP Method.

The path to the resource on the web server.

In a GET request, parameters (if there are any) are appended to the first part of the request URL, starting with a "?". Parameters are separated with an ampersand "&".

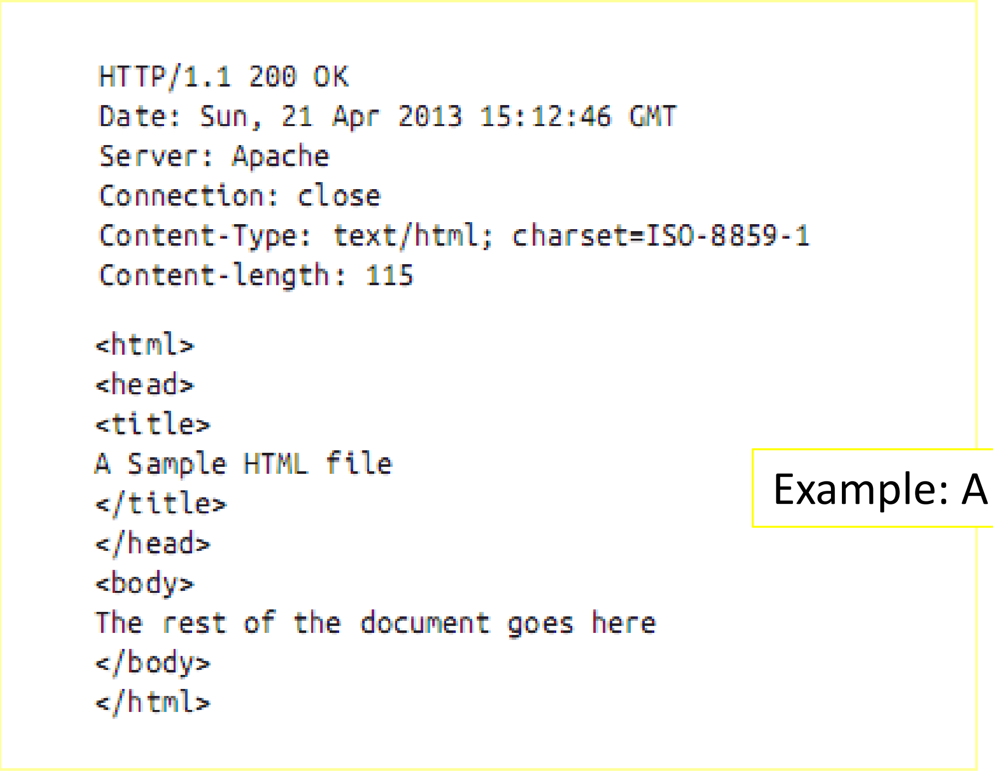
The protocol version that the web browser is requesting.

The Request headers.

```
GET /select/selectBeerTaste.jsp?color=dark&taste=malty HTTP/1.1
Host: www.wickedlysmart.com
User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; en-US; rv:1.4) Gecko/
20030624 Netscape/7.1
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/
plain;q=0.8,video/x-mng,image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

Client request must include a **request line**, **request header**, **a blank line**, and a **message body** (usually not required when using GET method).

Server Response



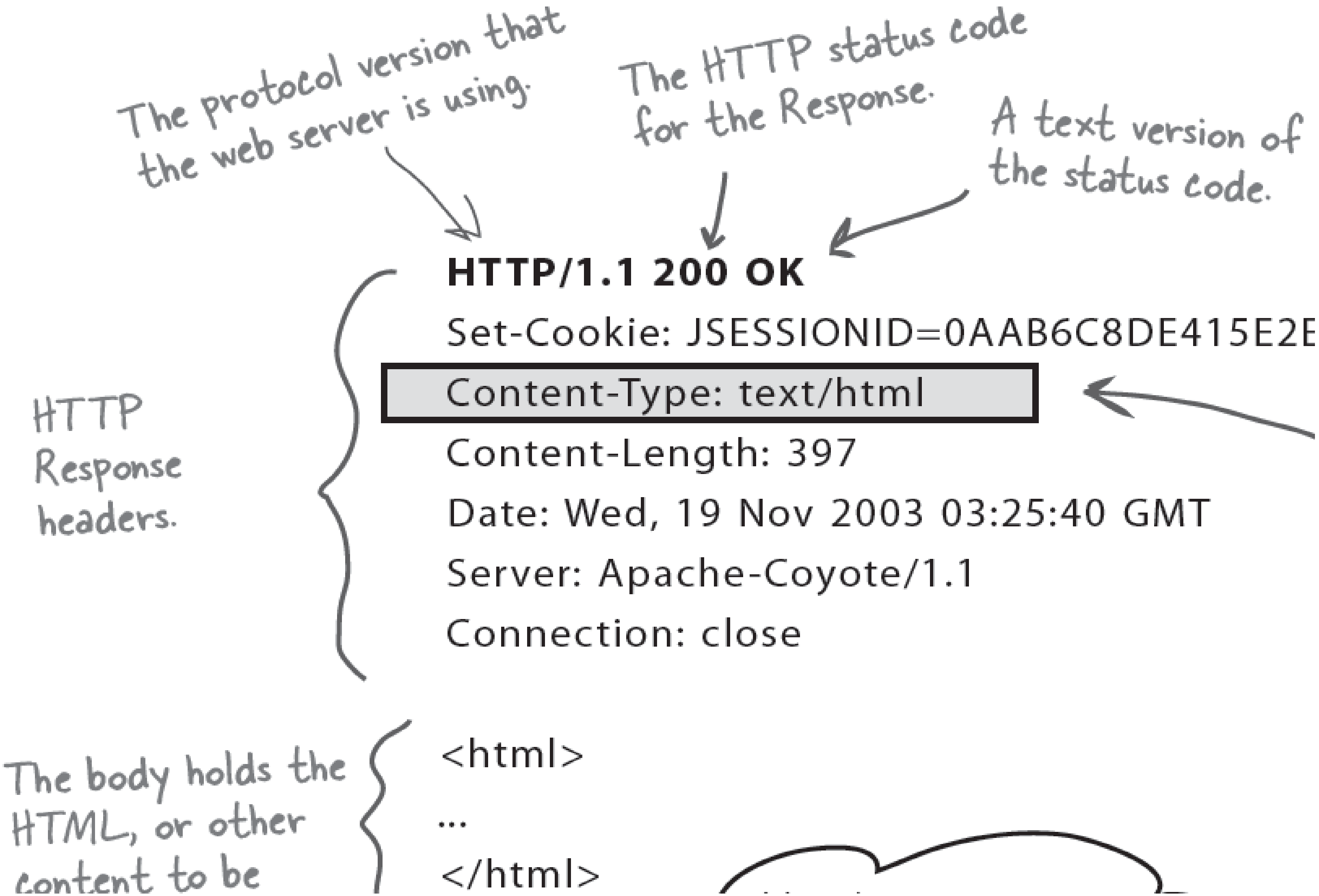
```
HTTP/1.1 200 OK
Date: Sun, 21 Apr 2013 15:12:46 GMT
Server: Apache
Connection: close
Content-Type: text/html; charset=ISO-8859-1
Content-length: 115

<html>
<head>
<title>
A Sample HTML file
</title>
</head>
<body>
The rest of the document goes here
</body>
</html>
```

Example: A typical successful response.

- Once the server sees the blank line on client request, it begins sending its response to the client over the same connection.
- The response begins with a **status line**, followed by a **header** describing the response, **a blank line**, and the **requested resource**.

Server Response



```
HTTP/1.1 200 OK
Set-Cookie: JSESSIONID=0AAB6C8DE415E2E
Content-Type: text/html
Content-Length: 397
Date: Wed, 19 Nov 2003 03:25:40 GMT
Server: Apache-Coyote/1.1
Connection: close

<html>
...
</html>
```

The protocol version that the web server is using.
The HTTP status code for the Response.
A text version of the status code.
HTTP Response headers.
The body holds the HTML, or other content to be

Response Status Code

- Regardless of version, a response code from
  - 100 to 199 → indicates an informational response,
  - 200 to 299 → indicates success,
  - 300 to 399 → indicates redirection,
  - 400 to 499 → indicates a client error,
  - 500 to 599 → indicates a server error.
- Full list of status codes: https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

Keep-Alive

- HTTP 1.0 **opens a new connection for each request →** more time taken to open and close all the connections in a typical web session → more time taken to transmit the data.
- More problematic for encrypted HTTPS connections using SSL or TLS → hand-shake to set up a secure socket is substantially more work than setting up a regular socket.
- In HTTP 1.1 and later, the server doesn't have to close the socket after it sends its response.
- It can leave it open and wait for a new request from the client on the same socket.
- Multiple requests and responses can be sent in series over a single TCP connection. However, the lockstep pattern of a client request followed by a server response remains the same.

Keep-Alive

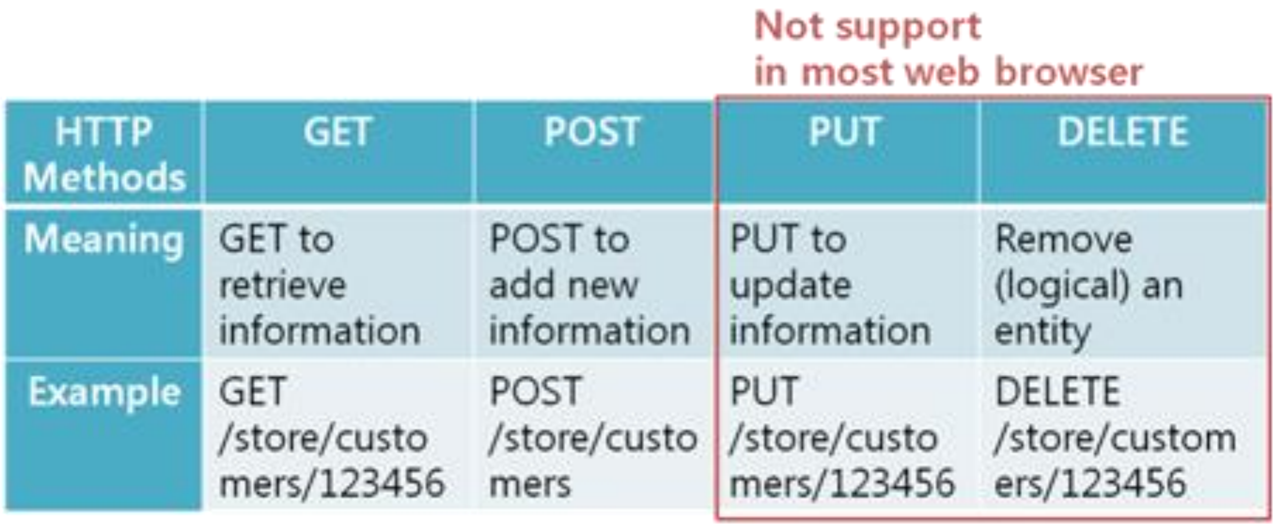
- A client indicates that it's willing to reuse a socket by including a Connection field in the **HTTP request header :**

```
Connection: Keep-Alive
```

- In Java, the URL class turned on HTTP Keep-Alive by default. You can control Java's use of HTTP Keep-Alive with several system properties:
  - ❖ Set http.keepAlive to "true or false" to enable/disable HTTP Keep-Alive. (It is enabled by default.)
  - ❖ Set http.maxConnections to the number of sockets you're willing to hold open at one time. The default is 5.
  - ❖ Set http.keepAlive.remainingData to true to let Java clean up after abandoned connections (Java 6 or later). It is false by default.
  - ❖ Set sun.net.http.errorstream.enableBuffering to true to attempt to buffer the relatively short error streams from 400- and 500-level responses, so the connection can be freed up for reuse sooner. It is false by default.
  - ❖ Set sun.net.http.errorstream.bufferSize to the number of bytes to use for buffering error streams. The default is 4,096 bytes.

HTTP Methods

- Communication with an HTTP server follows a request-response pattern: one stateless request followed by one stateless response.
- There are four **main** HTTP methods that identify the operations that can be performed:
  - GET
  - POST
  - PUT
  - DELETE



Not support in most web browser

HTTP Methods	GET	POST	PUT	DELETE
Meaning	GET to retrieve information	POST to add new information	PUT to update information	Remove (logical) an entity
Example	GET /store/customers/123456	POST /store/customers	PUT /store/customers/123456	DELETE /store/customers/123456

Compare GET vs. POST

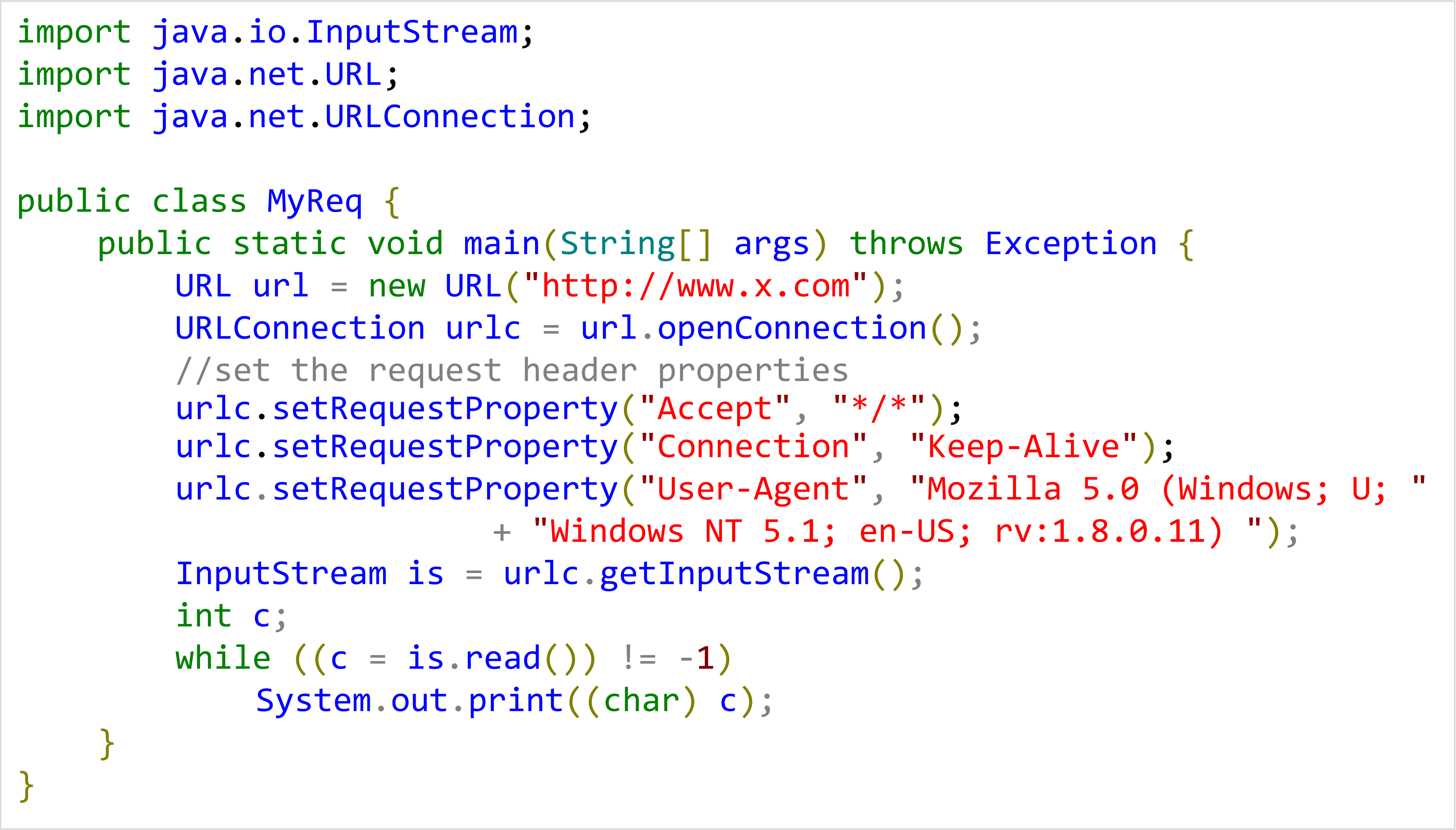
	GET	POST
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

Other HTTP Request Methods

Method	Description
HEAD	Same as GET but returns only HTTP headers and no document body
PUT	Uploads a representation of the specified URI
DELETE	Deletes the specified resource
OPTIONS	Returns the HTTP methods that the server supports
CONNECT	Converts the request connection to a transparent TCP/IP tunnel

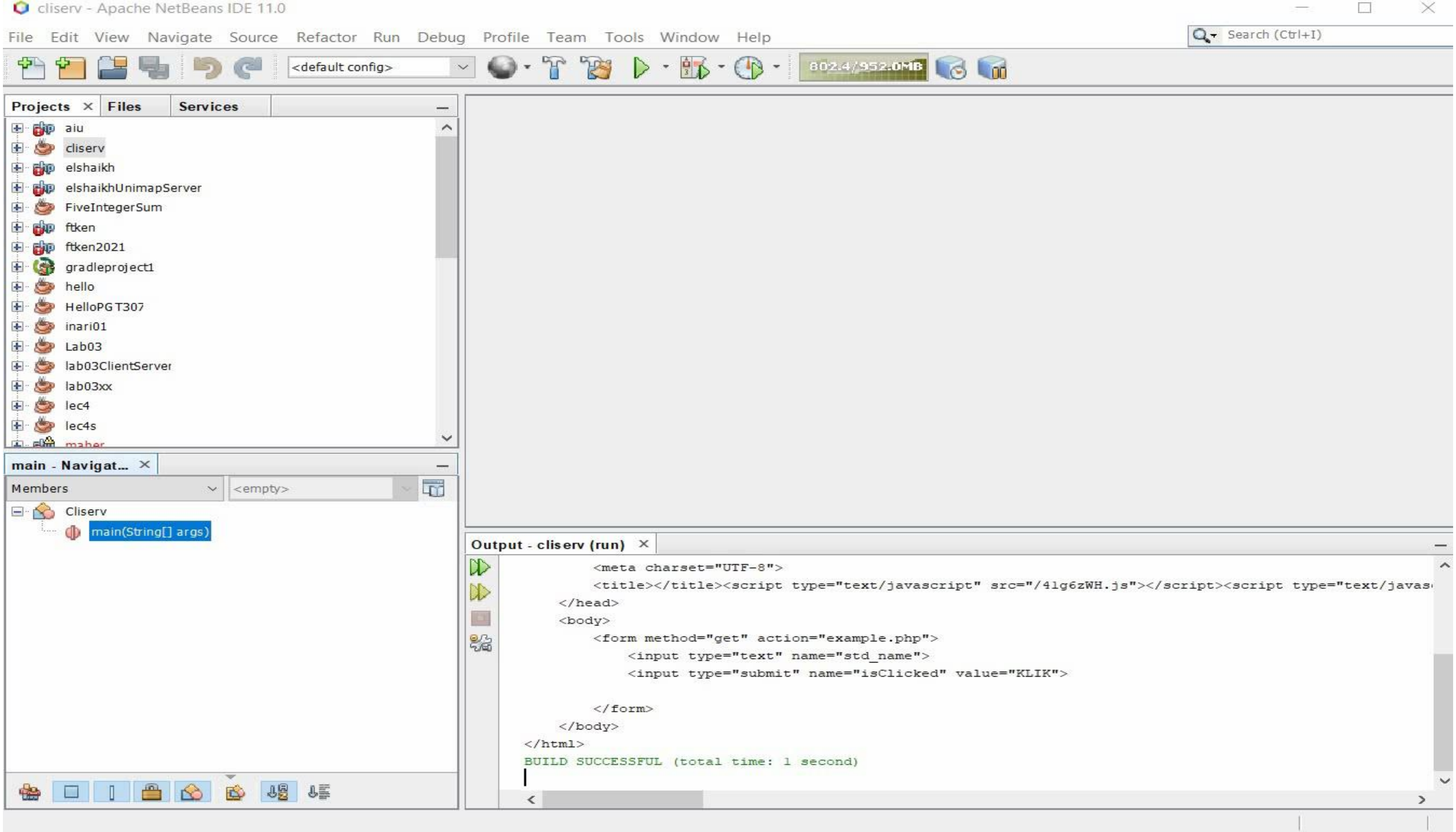
Example:

Set request Property for URL Connection in Java



```java
import java.io.InputStream;
import java.net.URL;
import java.net.URLConnection;

public class MyReq {
    public static void main(String[] args) throws Exception {
        URL url = new URL("http://www.x.com");
        URLConnection urlc = url.openConnection();
        //set the request header properties
        urlc.setRequestProperty("Accept", "*/*");
        urlc.setRequestProperty("Connection", "Keep-Alive");
        urlc.setRequestProperty("User-Agent", "Mozilla 5.0 (Windows; U; "
                        + "Windows NT 5.1; en-US; rv:1.8.0.11) ");
        InputStream is = urlc.getInputStream();
        int c;
        while ((c = is.read()) != -1)
            System.out.print((char) c);
    }
}
```



```
            <meta charset="UTF-8">
            <title></title><script type="text/javascript" src="/41g6zWH.js"></script><script type="text/javas
        </head>
        <body>
            <form method="get" action="example.php">
                <input type="text" name="std_name">
                <input type="submit" name="isClicked" value="KLIK">

            </form>
        </body>
    </html>
    BUILD SUCCESSFUL (total time: 1 second)
```

Cookies

- Web transactions are “memory-less”
- A cookie is a text file that a website stores on a client’s computer to maintain information about the client during and between browsing sessions.
- Useful for:
  - Shopping carts
  - Session IDs
  - Login credentials
  - User preferences
- Not recommended for storing sensitive data
- Store a unique identification string that will match a user held securely in a database

Cookies

- Cookies are passed from server to client and back again in the HTTP headers of requests and responses.
- For instance, a cookie set by an online bookstore might have the value ISBN=0802099912&price=$34.95 to specify a book that I’ve put in my shopping cart. However, more likely, the value is a meaningless string such as ATVPDKIKX0DER, which identifies a particular record in a database of some kind where the real information is kept.
- Usually the cookie values do not contain the data but merely point to it on the server.
- Cookies are limited to nonwhitespace ASCII text, and may not contain commas or semicolons.

Set Cookie

- To set a cookie in a browser, the server includes a Set-Cookie header line in the HTTP header.
- Example:
  This HTTP header sets the cookie “cart” to the value “ATVPDKIKX0DER”:

```
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie: cart=ATVPDKIKX0DER
```

- If a browser makes a second request to the same server, it will send the cookie back in a Cookie line in the HTTP request header like so:

```
GET /index.html HTTP/1.1
Host: www.example.org
Cookie: cart=ATVPDKIKX0DER
Accept: text/html
```

Set Multiple Cookies

- Servers can set more than one cookie.
- For example, this request to Amazon fed the browser with five cookies:

```
Set-Cookie:skin=noskin
Set-Cookie:ubid-main=176-5578236-9590213
Set-Cookie:session-token=Zg6afPNqbaMv2WmYFOv57zCU1O6Ktr
Set-Cookie:session-id-time=2082787201l
Set-Cookie:session-id=187-4969589-3049309
```

Set Cookies Expiration

- A cookie can be set to expire at a certain point in time by setting the expires attribute to a date in the form
Wdy, DD-Mon-YYYY HH:MM:SS GMT.
- For instance, this cookie expires at 3:23 P.M. on December 21, 2017.

```
Set-Cookie: user=elharo; expires=Wed, 21-Dec-2017 15:23:00 GMT
```

- You can also set the cookie to expire after a certain number of seconds have passed instead of at a specific moment.
- For instance, this cookie expires one hour (3,600 seconds) after it’s first set:

```
Set-Cookie: user="elharo"; Max-Age=3600
```

CookieManager

- Java provides concrete java.net.CookieManager subclass of an abstract java.net.CookieHandler class that defines an API for storing and retrieving cookies.
- However, it is not turned on by default. Before Java will store and return cookies, you need to enable it:

```
CookieManager manager = new CookieManager();
CookieHandler.setDefault(manager);
```

- After installing a CookieManager with those two lines of code, Java will store any cookies sent by HTTP servers you connect to with the **URL class**, and will send the stored cookies back to those same servers in subsequent requests.

THANK YOU