Network Programming

HTTP
Hypertext Transfer Protocol

HTTP

- The Hypertext Transfer Protocol (HTTP) is a protocol used mainly to access data on the World Wide Web. HTTP functions as a combination of FTP and SMTP.

HTTP

- Protocol for transfer of various data formats between server and client.
  - Plaintext
  - Hypertext
  - Images
  - Video
  - Sound
  - Meta-information also transferred

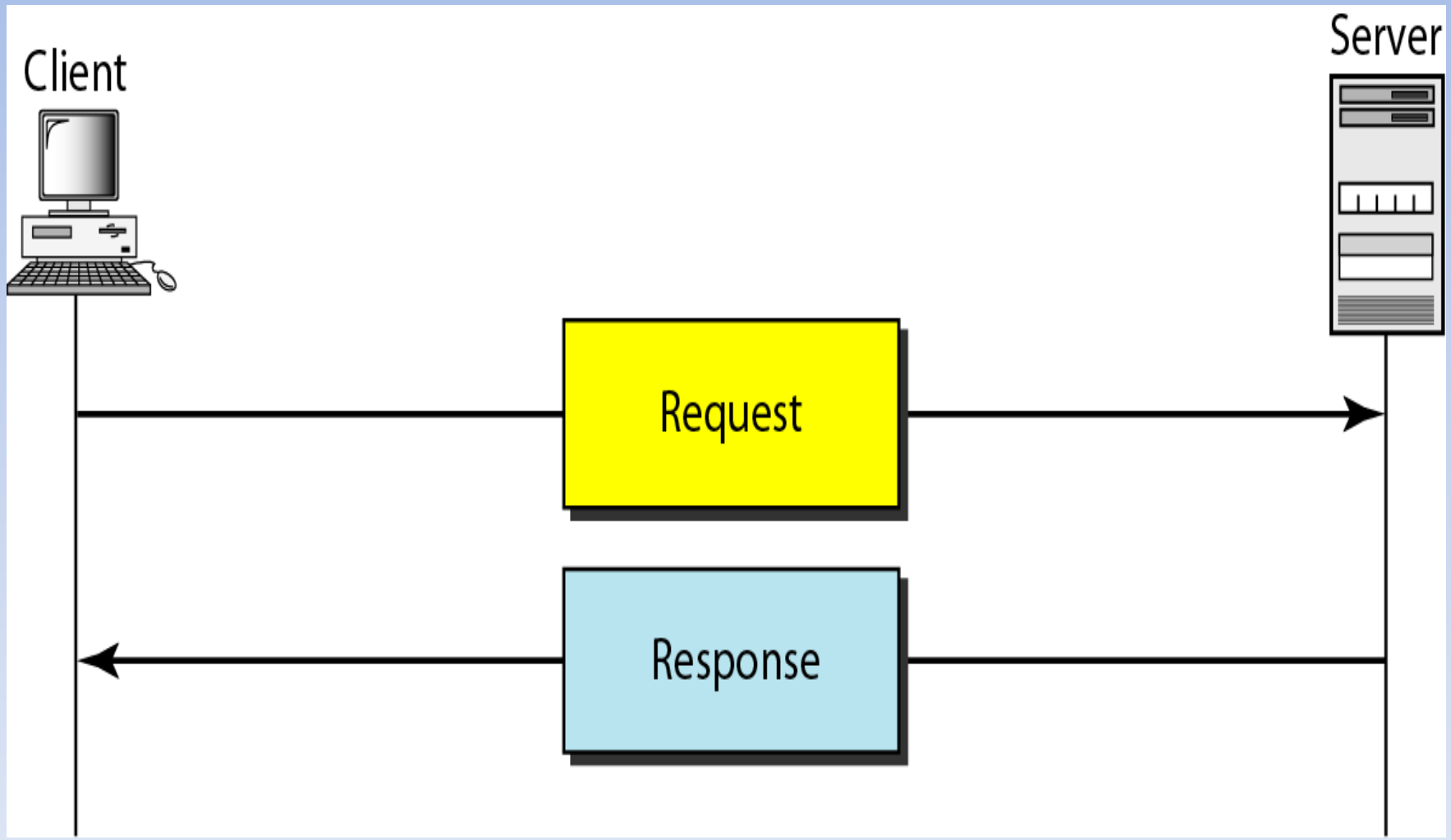
URL and HTTP

- All parts of URL, except parameters, used with http
- Scheme and host can be omitted when referenced object is on same machine as referring document
- Port can be omitted so long as referenced host is running on port listed in your /etc/services file
  - Usually port 80

URL and HTTP

- Full path used when referring to another server
  - Relative path on same server
    - Reference with relative path is a **partial URL**
- Query passes parameters to CGI
  - CGI a standard method used to generate dynamic content on web pages
- Fragment jumps to labels within a page
  - http://www.x.y/z#foo

HTTP transaction



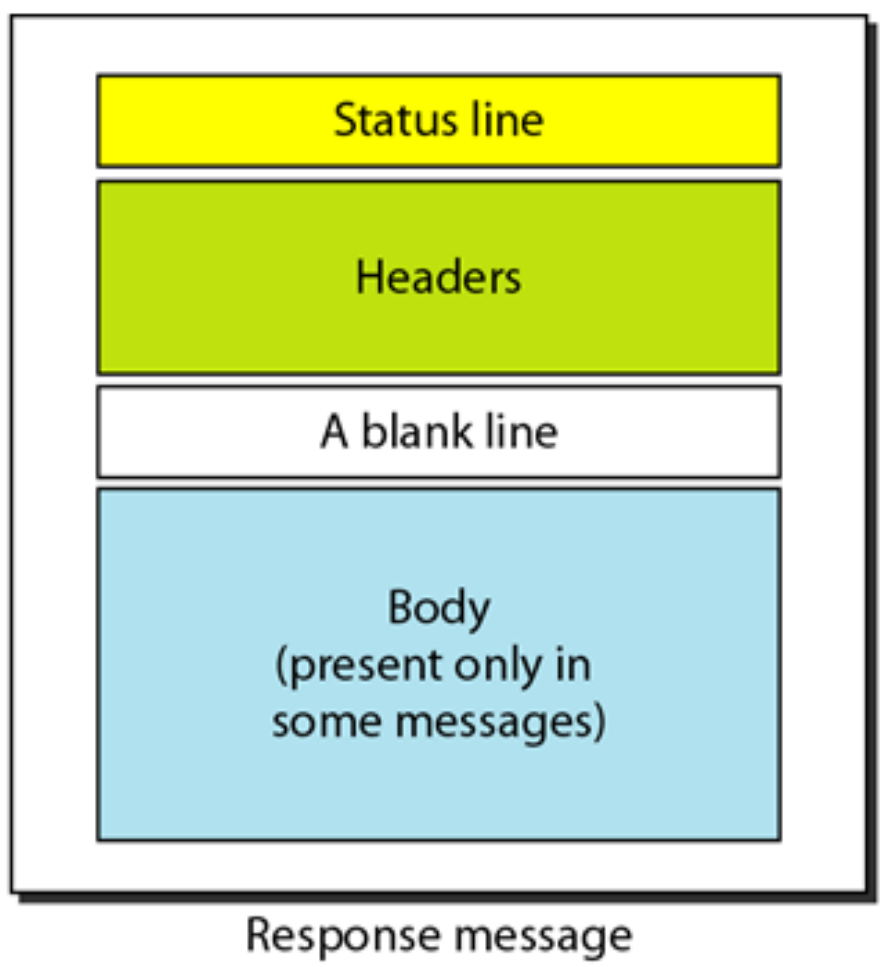
HTTP responses

- An HTTP response is made by a server to a client.
- The aim of the response is to:
  - provide the client with the resource it requested
  - or inform the client that the action it requested has been carried out
  - or else to inform the client that an error occurred in processing its request.

HTTP responses

- An HTTP response contains:
  - A status line.
  - A series of HTTP headers, or header fields.
  - A message body, which is usually needed.
- Each HTTP header is followed by a carriage return line feed (CRLF).
  - ✓ After the last of the HTTP headers, an additional CRLF is used (to give an empty line), and then the message body begins.

HTTP responses



HTTP responses

Status line

- The status line is the first line in the response message. It consists of three items:
  - The **HTTP version number**, showing the HTTP specification to which the server has tried to make the message comply.
  - A **status code**, which is a three-digit number indicating the result of the request.
  - A **reason phrase**, also known as status text, which is human-readable text that summarizes the meaning of the status code.

Status line

- An example of a response line is:

```
HTTP/1.1 200 OK
```

- In this example:
  - the HTTP version is HTTP/1.1
  - the status code is 200
  - the reason phrase is OK

HTTP headers

- The HTTP headers for a server's response contain information that a client can use to find out more about the response, and about the server that sent it.
- This information can assist the client with displaying the response to a user, with storing (or caching) the response for future use, and with making further requests to the server now or in the future.

HTTP headers

- For example, the following series of headers tell the client when the response was sent, that it was sent by CICS®, and that it is a JPEG image:

```
Date: Thu, 09 Dec 2004 12:07:48 GMT
Server: IBM_CICS_Transaction_Server/3.1.0(zOS)
Content-type: image/jpg
```

HTTP headers

- In the case of an unsuccessful request, headers can be used to tell the client what it must do to complete its request successfully.
- An empty line (that is, a CRLF alone) is placed in the response message after the series of HTTP headers, to divide the headers from the message body.

Message body

- The message body of a response may be referred to for convenience as a response body.
- Message bodies are used for most responses.
- The exceptions are where a server is responding to a client request that used the HEAD method (which asks for the headers but not the body of the response), and where a server is using certain status codes.

HTTP responses

- For a response to a successful request, the message body contains either the resource requested by the client, or some information about the status of the action requested by the client.

HTTP responses

- For a response to an unsuccessful request, the message body might provide further information about the reasons for the error, or about some action the client needs to take to complete the request successfully.

HTTP responses example

HTTP/1.1 200 OK	**Status Line**	**HTTP Response**
Date: Thu, 20 May 2004 21:12:58 GMT Connection: close	**General Headers**	
Server: Apache/1.3.27 Accept-Ranges: bytes	**Response Headers**	
Content-Type: text/html Content-Length: 170 Last-Modified: Tue, 18 May 2004 10:14:49 GMT	**Entity Headers**	
<html> <head> <title>Welcome to the Amazing Site!</title> </head> <body> This site is under construction. Please come back later. Sorry! </body> </html>	**Message Body**	

Status codes

Code	*Phrase*	*Description*
	Informational	
100	Continue	The initial part of the request has been received, and the client may continue with its request.
101	Switching	The server is complying with a client request to switch protocols defined in the upgrade header.
	Success	
200	OK	The request is successful.
201	Created	A new URL is created.
202	Accepted	The request is accepted, but it is not immediately acted upon.
204	No content	There is no content in the body.

Status codes

Code	*Phrase*	*Description*
	Redirection	
301	Moved permanently	The requested URL is no longer used by the server.
302	Moved temporarily	The requested URL has moved temporarily.
304	Not modified	The document has not been modified.
	Client Error	
400	Bad request	There is a syntax error in the request.
401	Unauthorized	The request lacks proper authorization.
403	Forbidden	Service is denied.
404	Not found	The document is not found.
405	Method not allowed	The method is not supported in this URL.
406	Not acceptable	The format requested is not acceptable.
	Server Error	
500	Internal server error	There is an error, such as a crash, at the server site.
501	Not implemented	The action requested cannot be performed.
503	Service unavailable	The service is temporarily unavailable, but may be requested in the future.

HTTP requests

- An HTTP request is made by a client, to a named host, which is located on a server.
- The aim of the request is to access a resource on the server.

HTTP requests

- To make the request, the client uses components of a URL (Uniform Resource Locator)
  - which includes the information needed to access the resource.

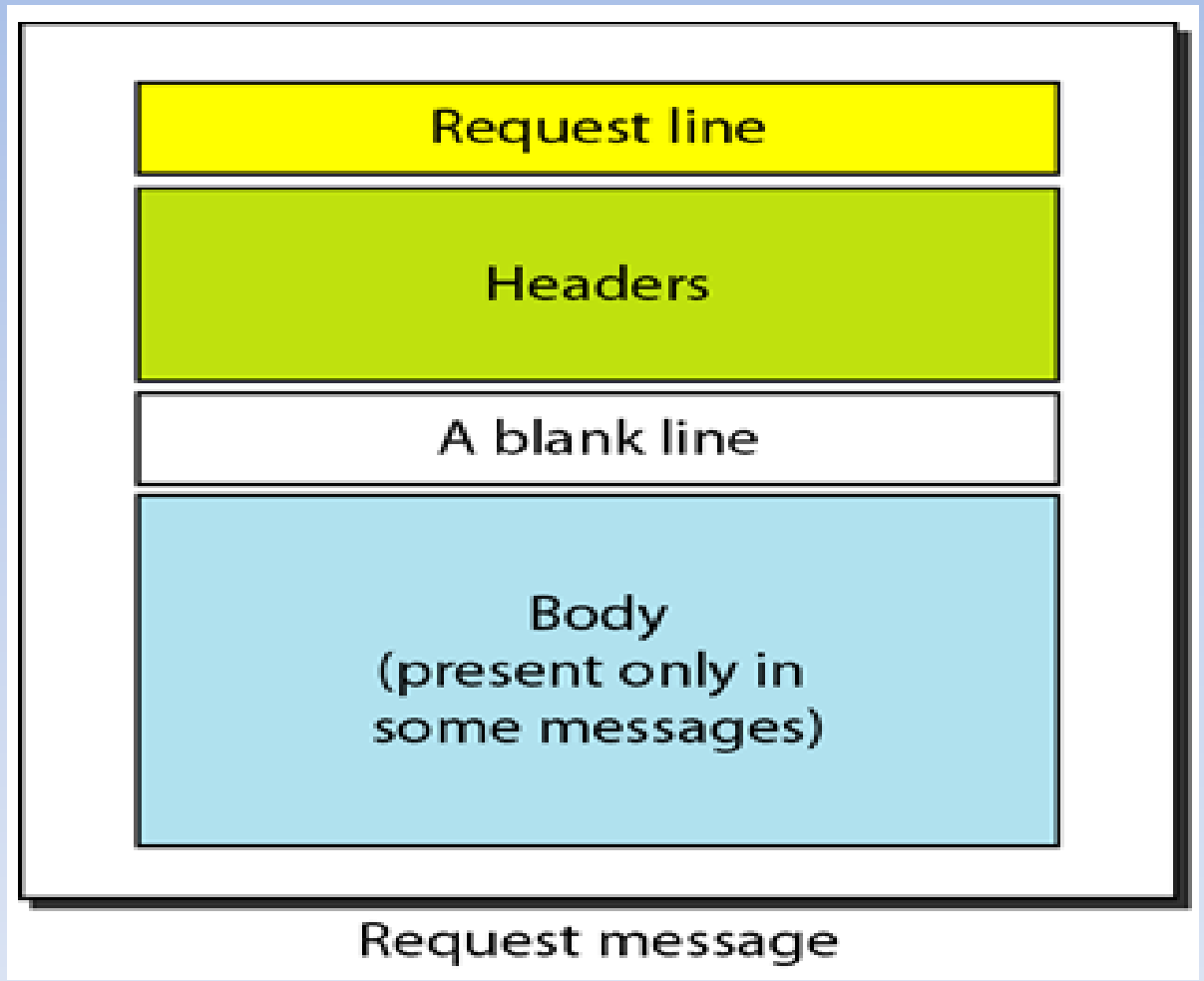
HTTP requests

- A correctly composed HTTP request contains the following elements:
  - A request line.
  - A series of HTTP headers, or header fields.
  - A message body, if needed.

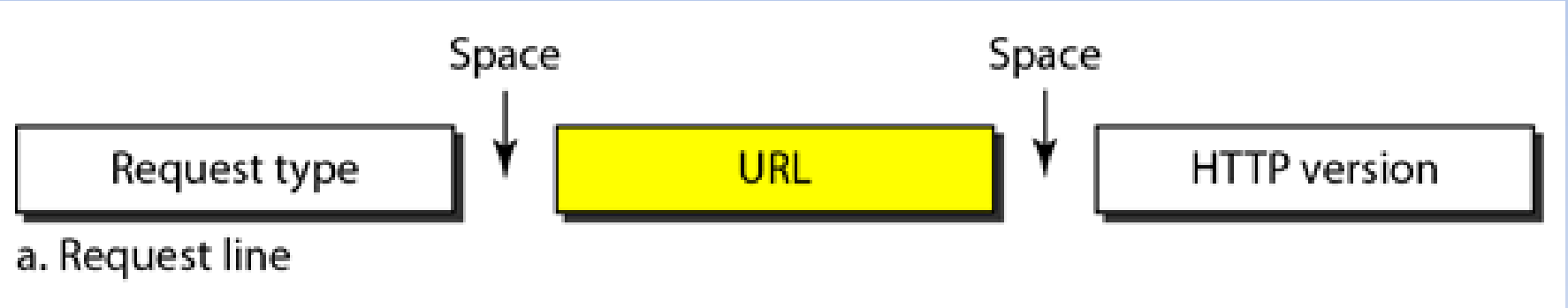
HTTP requests

- Each HTTP header is followed by a carriage return line feed (CRLF). After the last of the HTTP headers, an additional CRLF is used (to give an empty line), and then any message body begins.

HTTP requests



HTTP requests



a. Request line

Request line

- The **request line** is the first line in the request message. It consists of at least three items:
  1. A **method**. The method is a one-word command that tells the server what it should do with the resource.
     - For example, the server could be asked to send the resource to the client.

Request line

- It consists of at least three items:
  2. The path component of the URL for the request. The path identifies the resource on the server.
  3. The HTTP version number, showing the HTTP specification to which the client has tried to make the message comply.

Request line

- An example of a request line is:

```
GET /software/htp/cics/index.html HTTP/1.1
```

- In this example:
  - the method is GET
  - the path is /software/htp/cics/index.html
  - the HTTP version is HTTP/1.1

Request line

- A request line might contain some additional items:
  - A query string. This provides a string of information that the resource can use for some purpose. It follows the path, and is preceded by a question mark.

Request line

- A request line might contain some additional items:
  - The scheme and host components of the URL, in addition to the path. When the resource location is specified in this way, it is known as the **absolute URI** form. For HTTP/1.1, this form is used when a request will go through a proxy server. Also for HTTP/1.1, if the host component of the URL is not included in the request line, it must be included in the message in a Host header.

HTTP headers

- HTTP headers are written on a message to provide the recipient with information about the message, the sender, and the way in which the sender wants to communicate with the recipient.
- Each HTTP header is made up of a name and a value.

HTTP headers

- The HTTP headers for a client's request contain information that a server can use to decide how to respond to the request.

HTTP headers

- For example, the following series of headers can be used to specify that the user only wants to read the requested document in French or German, and that the document should only be sent if it has changed since the date and time when the client last obtained it:

```
Accept-Language: fr, de
If-Modified-Since: Fri, 10 Dec 2004 11:22:13 GMT
```

HTTP headers

- An empty line (that is, a CRLF alone) is placed in the request message after the series of HTTP headers, to divide the headers from the message body.

Message body

- The body content of any HTTP message can be referred to as a message body or entity body.
- Technically, the entity body is the actual content of the message.
- The message body contains the entity body, which can be in its original state, or can be encoded in some way for transport, such as by being broken into chunks (chunked transfer-coding).
- The message body of a request may be referred to for convenience as a request body.

URLConnection Class

- **URLConnection** is an abstract class whose subclasses form the **link** between the **user application** and any **resource on the web**. We can use it to read/write from/to any resource referenced by a URL object.

URLConnection Class

- There are mainly two subclasses that extend the URLConnection class:
  - **HttpURLConnection**: If we are connecting to any URL which uses “http” as its protocol, then HttpURLConnection class is used.
  - **JarURLConnection**: If however, we are trying to establish a connection to a jar file on the web, then JarURLConnection is used.

Methods of URLConnection Class

Method	Action Performed
getContent()	Retrieves the content of the URLConnection
getContentEncoding()	Returns the value of the content-encoding header field.
getContentLength()	Returns the length of the content header field
getDate()	Returns the value of date in the header field
getHeaderFields()	Returns the map containing the values of various header fields in the HTTP header
getHeaderField(int i)	Returns the value of the i^{th} index of the header

Methods of URLConnection Class

Method	Action Performed
getHeaderField(String field)	Returns the value of the field named “field” in the header
getInputStream()	Returns the input stream to this open connection been inside of OutputStream class
getOutputStream()	Returns the output stream to this connection of OutputStream class
openConnection()	Opens the connection to the specified URL.
setAllowUserInteraction()	Setting this true means a user can interact with the page. The default value is true.

Methods of URLConnection Class

Method	Action Performed
getHeaderField(String field)	Returns the value of the field named “field” in the header
getInputStream()	Returns the input stream to this open connection been inside of OutputStream class
getOutputStream()	Returns the output stream to this connection of OutputStream class
openConnection()	Opens the connection to the specified URL.
setAllowUserInteraction()	Setting this true means a user can interact with the page. The default value is true.

Methods of URLConnection Class

Method	Action Performed
setDefaultUseCaches()	Sets the default value of useCache field as the given value.
setDoInput()	Sets if the user is allowed to take input or not
setDoOutput()	Sets if the user is allowed to write on the page. The default value is false since most of the URLs don't allow to write

HttpURLConnection class

- is an abstract class directly extending from URLConnection class.
- It includes all the functionality of its parent class with additional HTTP-specific features.
- **Constructor**

HttpURLConnection(URL u): Constructs the httpurlconnection to specified URL

HttpURLConnection class (Methods)

Method	Action performed
disconnect()	Indicated that requests to the server are highly unlikely in the future.
getErrorStream()	Gets the error stream if the server cannot be connected or some error occurred. It can contain information about how to fix the error from the server.
getFollowRedirects()	Returns true or false depending on automatic redirection or not.

HttpURLConnection class (Methods)

Method	Action performed
getHeaderField()	Returns the nth header field, or null if it does not exist. It overrides the getHeaderField method of URLConnection class.
getInstanceFollowRedirects()	Returns true or false depending on whether automatic instance redirection is set or not.
getPermission()	Retrieves the permission required to connect to a destination host and port.
getResponseCode()	Used to retrieve the response status from server.
getResponseMessage()	Retrieves the response message.

HttpURLConnection class (Methods)

Method	Action performed
getRequestMethod()	Returns the request method.
setInstanceFollowRedirects()	Sets whether response code requests be redirected automatically by this instance of HTTP URL connection. It overrides the more generic setFollowRedirects()
setRequestMethod()	Used to set the request method. Default is GET
setFixedLengthStreamingMode()	Used to set the length of content written on outputstream if it is known in advance.

HttpURLConnection class (Methods)

Method	Action performed
setChunkedStreamingMode()	Used when the content length is not known. Instead of creating a buffer of fixed length and writing it to a server, content is broken into chunks and then written. Not all servers support this mode.
usingProxy()	Returns true if connection is established using a proxy, else false

Java HttpURLConnection Example

```
import java.io.*;
import java.net.*;
public class HttpURLConnectionDemo{
public static void main(String[] args){
try{
URL url=new URL("http://www.javatpoint.com/java-tutorial");
HttpURLConnection huc=(HttpURLConnection)url.openConnection();
for(int i=1;i<=8;i++){
System.out.println(huc.getHeaderFieldKey(i)+" = "+huc.getHeaderField(i));
}
huc.disconnect();
}catch(Exception e){System.out.println(e);}
}
}
```

Output

```
Date = Thu, 22 Jul 2021 18:08:17 GMT
Server = Apache
Location = https://www.javatpoint.com/java-tutorial
Cache-Control = max-age=2592000
Expires = Sat, 21 Aug 2021 18:08:17 GMT
Content-Length = 248
Keep-Alive =timeout=5, max=1500
Connection = Keep-Alive
```

HTTP request methods

- The most commonly used HTTP request methods are
  - GET
  - HEAD
  - POST
  - PUT
  - PATCH
  - and DELETE.

HTTP request methods (GET)

- GET request is used to read/retrieve data from a web server.
- GET returns an HTTP status code of 200 (OK) if the data is successfully retrieved from the server.

HTTP request methods (HEAD)

- **HEAD** Same as GET, but transfers the status line and header section only.

HTTP request methods (POST)

- POST request is used to send data (file, form data, etc.) to the server.
- On successful creation, it returns an HTTP status code of 201.

HTTP request methods (PUT)

- A PUT request is used to modify the data on the server.
- It replaces the entire content at a particular location with data that is passed in the body payload.
- If there are no resources that match the request, it will generate one.

HTTP request methods (PATCH)

- PATCH is similar to PUT request, but the only difference is, it modifies a part of the data.
- It will only replace the content that you want to update.

HTTP request methods (DELETE)

- A DELETE request is used to delete the data on the server at a specified location.