The **URLConnection**and **HttpURLConnection** classes for developing Java network applications through various examples.

Example #1: Reading data from the server

The following code snippet downloads HTML source code of a webpage and save it to a file:

```
import java.net.*;
import java.io.*;
import java.util.*;

public class DownloadWebPage {

    public static void main(String[] args) {
        if (args.length < 2) {
            System.out.println("Syntax: <url> <file>");
            return;
        }

        String url = args[0];
        String filePath = args[1];

        try {

            URL urlObj = new URL(url);
            URLConnection urlCon = urlObj.openConnection();

            InputStream inputStream = urlCon.getInputStream();
            BufferedInputStream reader = new BufferedInputStream(inputStream);

            BufferedOutputStream writer = new BufferedOutputStream(new
FileOutputStream(filePath));

            byte[] buffer = new byte[4096];
            int bytesRead = -1;

            while ((bytesRead = reader.read(buffer)) != -1) {
                writer.write(buffer, 0, bytesRead);
            }

            writer.close();
            reader.close();

            System.out.println("Web page saved");

        } catch (MalformedURLException e) {
            System.out.println("The specified URL is malformed: " +
e.getMessage());
        } catch (IOException e) {
            System.out.println("An I/O error occurs: " + e.getMessage());
        }
    }
}
```

The above code opens a connection from the specified URL, gets an input stream and an output stream. Then it reads data from the input stream and writes the data to a specified file.

To run this program from command line use this syntax:
```
java DownloadWebPage <url> <file>
```

For example, download the Google's homepage:
```
java DownloadWebPage https://google.com Google.html
```

Example #2: Check HTTP Response Code

The above program will fail silently if the server returns a response code other than 200 - the HTTP response code indicates the server returns the document without any problem. For example, if you run the above program using this command:
```
java DownloadWebPage http://facebook.com Facebook.html
```

The program terminates normally but the file Facebook.html is empty (0 byte). So let use the following code to check the server's response code before reading and writing data:

```
URL urlObj = new URL(url);
HttpURLConnection httpCon = (HttpURLConnection) urlObj.openConnection();

int responseCode = httpCon.getResponseCode();

if (responseCode != HttpURLConnection.HTTP_OK) {
    System.out.println("Server returned response code " + responseCode + ". Download failed.");
    System.exit(0);
}
```

This code will terminate the program if the response code is not 200 (HTTP_OK). Update, recompile and run the program again:
```
java DownloadWebPage http://facebook.com Facebook.html
```

And you should see the following output:
```
Server returned response code 301. Download failed.
```

The HTTP status code 301 indicates that the requested document is moved permanently. That means we should use `https://` instead of `http://`. Run the program again with this command:
```
java DownloadWebPage https://facebook.com Facebook.html
```

Example #3: Set Client's HTTP Request Header Fields

Use the `setRequestProperty(String key, String value)` method of the `URLConnection` class to set header fields for the request. The client's header fields provide additional information about the client and how the client expects response from the server.
Here's an example:
As you can see, this code snippet specifies 4 header fields for the request:

```
URL urlObj = new URL(url);
URLConnection urlCon = urlObj.openConnection();

urlCon.setRequestProperty("User-Agent", "Java Client; Mac OS");
urlCon.setRequestProperty("Accept", "text/html");
urlCon.setRequestProperty("Accept-Language", "en-US");
urlCon.setRequestProperty("Connection", "close");
```

- User-Agent: information about the client such as browser type, operating system, architecture, etc.

- Accept: the content type understood by the client

- Accept-Language: the language understood by the client

- Connection: type of the connection. In this case, the connection is closed after a request-response roundtrip finished.

Example #4: Read all Server's Header Fields

We can use the `getHeaderFields()` method of the `URLConnection` class to read all header fields sent from the server. Here's an example:

```
String url = "https://google.com";
URL urlObj = new URL(url);
URLConnection urlCon = urlObj.openConnection();

Map<String, List<String>> map = urlCon.getHeaderFields();


for (String key : map.keySet()) {
    System.out.println(key + ":");

    List<String> values = map.get(key);

    for (String aValue : values) {
        System.out.println("\t" + aValue);
    }
}
```

Run this code and you should see the output looks like this (from google.com):
This information varies, depending on each server.

```
 Transfer-Encoding:
        chunked
null:
        HTTP/1.1 200 OK
Server:
        gws
Date:
        Sat, 09 Dec 2017 15:38:16 GMT
Accept-Ranges:
        none
Cache-Control:
        private, max-age=0
```

```
Content-Type:
        text/html; charset=ISO-8859-1
```

Example #5: Read Common Header Fields

Using descriptive get header fields methods, the following code snippet reads and prints the values of common header fields such as response code, response message, content type, content encoding, content length, and so on:

```
String url = "https://facebook.com";

URL urlObj = new URL(url);
HttpURLConnection httpCon = (HttpURLConnection) urlObj.openConnection();

int responseCode = httpCon.getResponseCode();
String responseMessage = httpCon.getResponseMessage();
String contentType = httpCon.getContentType();
String contentEncoding = httpCon.getContentEncoding();
int contentLength = httpCon.getContentLength();

long date = httpCon.getDate();
long expiration = httpCon.getExpiration();
long lastModified = httpCon.getLastModified();

System.out.println("Response Code: " + responseCode);
System.out.println("Response Message: " + responseMessage);
System.out.println("Content Type: " + contentType);
System.out.println("Content Encoding: " + contentEncoding);
System.out.println("Content Length: " + contentLength);
System.out.println("Date: " + new Date(date));
System.out.println("Expiration: " + new Date(expiration));
System.out.println("Last Modified: " + new Date(lastModified));
```

The output would be:

```
Response Code: 200
Response Message: OK
Content Type: text/html; charset=UTF-8
Content Encoding: null
Content Length: -1
Date: Sun Dec 10 10:52:31 ICT 2017
Expiration: Sat Jan 01 07:00:00 ICT 2000
Last Modified: Thu Jan 01 07:00:00 ICT 1970
```

Example #6: Set HTTP Request Method

By default the HTTP request method is GET. You can call the `setRequestMethod(String method)` to set the request method which is one of GET, POST, HEAD, PUT, DELETE, TRACE, and OPTIONS.
For example, the following code sets the request method to HEAD:

```
URL urlObj = new URL(url);
HttpURLConnection httpCon = (HttpURLConnection) urlObj.openConnection();
```

```
httpCon.setRequestMethod("HEAD");
```

When processing a HEAD request, the server returns a response without the body content. Only the header fields are returned. Hence the method name “HEAD”.
Also note that when you set the request method to POST, you must also enable output for the connection, as POST method is about send data to the server. For example:

```
httpCon.setDoOutput(true);
httpCon.setRequestMethod("POST");
```

Example #7: Send HTTP POST Request

To send an HTTP POST request along with parameters, you need to constructor the parameters in the following form:

```
param1=value1&param2=value2&param3=value3
```

The following code snippet demonstrates how to send a login request to Twitter via HTTP POST:

```
String url = "https://twitter.com/sessions";
String email = "yourname@gmail.com";
String password = "yourpass";


URL urlObj = new URL(url);
HttpURLConnection httpCon = (HttpURLConnection) urlObj.openConnection();

httpCon.setDoOutput(true);
httpCon.setRequestMethod("POST");

String parameters = "username=" + email;
parameters += "password=" + password;


OutputStreamWriter writer = new OutputStreamWriter(
    httpCon.getOutputStream());
writer.write(parameters);
writer.flush();
```

API References:

- URLConnection Javadoc
- HttpURLConnection Javadoc