UNIT 5
URL CONNECTION

LH - 5HRS

PRESENTED BY: ER. **SHARAT MAHARJAN**

NETWORK PROGRAMMING

PRIME COLLEGE, NAYABAZAAR

CONTENTS (LH - 5HRS)

5.1 Opening URL Connections

5.2 Reading Data from Server

5.3 Reading Header: Retrieving Specific Header Fields and Retrieving Arbitrary Header Fields

5.4 Cache: Web Cache for Java

5.5 Configuring the Connection: protected URL url, protected boolean connected, protected boolean allowUserInteraction, protected boolean doInput, protected boolean doOutput, protected boolean ifModificationSince, protected boolean useCaches and Timeouts

5.6 Configuring the Client Request HTTP Header

5.7 Security Considerations for URLConnection

5.8 Guessing MIME Media Types

5.9 HTTPURLConnection: The Request Methods, Disconnecting from the Server, Handling Server Responses, Proxies and Streaming Mode

- **URLConnection** is an **abstract class that represents an active connection to a resource specified by a URL**.
- The **URLConnection class has two different but related purposes**:
- **First**, it **provides more control over the interaction with a server** (especially an HTTP server) **than the URL class**. A URLConnection **can inspect the header** sent by the server and respond accordingly. It can **set the header fields used in the client request**. Finally, a URLConnection **can send data back to a web server** with POST, PUT, and other HTTP request methods.
- **Second**, the URLConnection class is part of Java's protocol handler mechanism, which also includes the URLStreamHandler class.
- The Java URLConnection class **represents a communication link between the URL and the application**. This class can be **used to read and write data to the specified resource referred by the URL**.

5.1 Opening URL Connections

A program that uses the URLConnection class directly follows this basic sequence of steps:

1. **Construct** a **URL object**.
2. **Invoke the URL object's openConnection() method to retrieve a URLConnection object** for that URL.
3. **Configure** the URLConnection.
4. **Read** the **header fields**.
5. **Get an input stream and read data**.
6. **Get an output stream and write data**.
7. **Close the connection**.

- The **single constructor** for the URLConnection class is protected:

protected URLConnection(URL url)

```
try {
        URL u = new URL("http://www.overcomingbias.com/");
        URLConnection uc = u.openConnection();
        // read from the URL...
} catch (MalformedURLException ex) {
        System.err.println(ex);
} catch (IOException ex) {
        System.err.println(ex);
}
```

Example 1: Demonstrate URLConnection

```
import java.io.*;
import java.net.*;
public class Example1 {
public static void main(String[] args) {
 try {
  URL u = new URL("https://www.prime.edu.np/");
  URLConnection uc = u.openConnection();
  System.out.println("Content:"+uc.getContentType());
 } catch (MalformedURLException ex) {
  System.err.println("It is not a parseable URL");
 } catch (IOException ex) {
  System.err.println(ex);}}}
```

5.2 Reading Data from Server

- The following is the minimal set of steps needed to retrieve data from a URL using a URLConnection object:

1. **Construct** a URL object.

2. **Invoke** the URL object's **openConnection()** method **to retrieve a URLConnection object** for that URL.

3. **Invoke** the URLConnection's **getInputStream()** method.

4. **Read from the input stream** using the usual stream API.

Example 2: Download a web page with a URLConnection

```
import java.io.*;
import java.net.*;
public class Example2 {
public static void main(String[] args) {
 try {
  URL u = new URL("https://www.prime.edu.np/");
  URLConnection uc = u.openConnection();
  BufferedReader in_data = new BufferedReader(new
                                        InputStreamReader(uc.getInputStream()));
  String entry;
  while((entry=in_data.readLine())!=null) {
  System.out.println(entry);
  }
 } catch (MalformedURLException ex) {
  System.err.println("It is not a parseable URL");
 } catch (IOException ex) {
  System.err.println(ex);}}}
```

Differences between URL and URLConnection.

- URLConnection provides access to the HTTP header.
- URLConnection can configure the request parameters sent to the server.
- URLConnection can write data to the server as well as read data from the server.

5.3 Reading Header

- HTTP servers **provide amount of information in the header that precedes each response**.
- For example, here's a **typical HTTP header returned by an Apache** web server:

```
HTTP/1.1 301 Moved Permanently
Date: Sun, 21 Apr 2013 15:12:46 GMT
Server: Apache
Location: http://www.ibiblio.org/
Content-Length: 296
Connection: close
Content-Type: text/html; charset=iso-8859-1
```

i. Retrieving Specific Header Fields

- Six Convenience Methods
- These return the values of six particularly common header fields:

1. public int **getContentLength()**
2. public String **getContentType()**
3. public String **getContentEncoding()**
4. public long **getExpiration()**
5. public long **getDate()**
6. public long **getLastModified()**

Example 3: Return the header

```
import java.net.*;
import java.util.Date;
import java.io.*;
public class Example3 {
public static void main(String[] args) {
 try {
  URL u = new URL("https://www.prime.edu.np/");
  URLConnection uc = u.openConnection();
  System.out.println("Content-type: " + uc.getContentType());
  if (uc.getContentEncoding() != null) {
   System.out.println("Content-encoding: "+ uc.getContentEncoding());}
  if (uc.getDate() != -1) {
   System.out.println("Date: " + new Date(uc.getDate()));}
  if (uc.getLastModified() != -1) {
   System.out.println("Last modified: "+ new Date(uc.getLastModified()));}
  if (uc.getExpiration() != -1) {
   System.out.println("Expiration date: "+ new Date(uc.getExpiration()));}
  if (uc.getContentLength() != -1) {
   System.out.println("Content-length: " + uc.getContentLength());}
  } catch (MalformedURLException ex) {
   System.err.println("It is not a valid URL.");
  } catch (IOException ex) {
   System.err.println(ex);}
   System.out.println();}}
```

Retrieving Arbitrary Header Fields

public String getHeaderField(String name)

- The getHeaderField(String name) method returns the string value of a named header field.

✓Names are case-insensitive.

✓If the requested field is not present, null is returned.

- For example, to get the value of the **Content-type and Content-encoding header fields** of a URLConnection object uc, you could write:

```
String contentType = uc.getHeaderField("content-type");
String contentEncoding = uc.getHeaderField("content-encoding"));
```

- To get the Date, Content-length, or Expires headers, you'd do the same:

```
String data = uc.getHeaderField("date");
String expires = uc.getHeaderField("expires");
String contentLength = uc.getHeaderField("Content-length");
```

Retrieving Arbitrary Header Fields

public String getHeaderFieldKey(int n) //key:value pair

- The keys of the header fields are returned by the getHeaderFieldKey(int n) method.
- The first field is 1.
- If a numbered key is not found, null is returned.
- You can use this in combination with getHeaderField() to loop through the complete header

For example, in order to get the sixth key of the header of the URLConnection uc, you would write:

String header6 = uc.getHeaderFieldKey(6);

public String getHeaderField(int n)

- This **method returns the value of the nth header field**. In HTTP, **the starter line containing the request method and path is header field zero and the first actual header is one.**

Example 4: Print the entire HTTP header

```
import java.net.*;
import java.io.*;
public class Example4 {
public static void main(String[] args) {
 try {
  URL u = new URL("https://www.prime.edu.np/");
  URLConnection uc = u.openConnection();
  int i=1;
  while(uc.getHeaderField(i)!=null) {//value check in key:value pair
  System.out.println(uc.getHeaderFieldKey(i)+":"+uc.getHeaderField(i));
  i++;
  }} catch (MalformedURLException ex) {
   System.err.println("It is not a valid URL.");
  } catch (IOException ex) {
   System.err.println(ex);}
  System.out.println();}}
```

5.4 Cache: Web Cache for Java

- Web browsers have been caching pages and images for years.
- If a logo is repeated on every page of a site, the browser normally loads it from the remote server only once, stores it in its cache, and reloads it from the cache whenever it's needed rather than requesting it from the remote server every time the logo is encountered.
- Several HTTP headers, including Expires and Cache-control, can control caching.

However, HTTP headers can adjust this:

- The Cache-control header (HTTP 1.1) offers fine-grained cache policies:

max-age=[seconds]: Number of seconds from now before the cached entry should expire

no-cache: Not quite what it sounds like. The entry may still be cached, but the client should reverify the state of the resource with an ETag or Last-modified header on each access.

no-store: Do not cache the entry no matter what.

For example, this HTTP response says that the resource may be **cached for 604,800 seconds** (HTTP 1.1). It also says it was **last modified on April 20** and has an **ETag, so if the local cache already has a copy more recent than that, there's no need to load the whole document now:**

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
Cache-control: max-age=604800
Expires: Sun, 28 Apr 2013 15:12:46 GMT
Last-modified: Sat, 20 Apr 2013 09:55:04 GMT
ETag: "67099097696afcf1b67e"

By default, Java does not cache anything. To install a system-wide cache of the URL class, you need the following:

- A concrete subclass of ResponseCache
- A concrete subclass of CacheRequest
- A concrete subclass of CacheResponse

- Two **abstract methods** in the **ResponseCache class** store and retrieve data from the system's single cache:

public abstract CacheRequest put(URI uri, URLConnection connection) throws IOException

public abstract CacheResponse get(URI uri, String requestMethod, Map<String, List<String>> requestHeaders) throws IOException

- **The CacheRequest class**

```
package java.net;
public abstract class CacheRequest {
        public abstract OutputStream getBody() throws IOException;
        public abstract void abort();
}
```

- **The CacheResponse class**

```
public abstract class CacheResponse {
        public abstract Map<String, List<String>> getHeaders() throws
        IOException;
        public abstract InputStream getBody() throws IOException;
}
```

5.5 Configuring the Connection

The URLConnection class has **seven protected fields** that define exactly how the client makes the request to the server. These are:

- ✓protected URL **url**;
- ✓protected boolean **doInput** = true;
- ✓protected boolean **doOutput** = false;
- ✓protected boolean **allowUserInteraction** = defaultAllowUserInteraction;
- ✓protected boolean **useCaches** = defaultUseCaches;
- ✓protected long **ifModifiedSince** = 0;
- ✓protected boolean **connected** = false;

- Their values are accessed and modified via setter and getter methods:

✓public URL **getURL()**

✓public void **setDoInput**(boolean doInput)

✓public boolean **getDoInput**()

✓public void **setDoOutput**(boolean doOutput)

✓public boolean **getDoOutput**()

✓public void **setAllowUserInteraction**(boolean allowUserInteraction)

✓public boolean **getAllowUserInteraction**()

✓public void **setUseCaches**(boolean useCaches)

✓public boolean **getUseCaches**()

✓public void **setIfModifiedSince**(long ifModifiedSince)

✓public long **getIfModifiedSince**()

protected URL url

The url field specifies the URL that this URLConnection connects to. The constructor sets it when the URLConnection is created and it should not change thereafter. You can retrieve the value by calling the getURL() method.

Example 5: Print the URL of a URLConnection to http://www.oreilly.com/

```
import java.io.*;
import java.net.*;
public class URLPrinter {
        public static void main(String[] args) {
                try {
                URL u = new URL("http://www.oreilly.com/");
                URLConnection uc = u.openConnection();
                System.out.println(uc.getURL());
                } catch (IOException ex) {
                        System.err.println(ex);}}}
```

5.6 Configuring the Client Request HTTP Header

- An HTTP client (e.g., a browser) sends the server a request line and a header. For example, here's an HTTP header that Chrome sends:

  Accept:text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
  Accept-Charset:ISO-8859-1,utf-8;q=0.7,*;q=0.3
  Accept-Encoding:gzip,deflate,sdch
  Accept-Language:en-US,en;q=0.8
  Cache-Control:max-age=0
  Connection:keep-alive
  Cookie:reddit_first=%7B%22firsttime%22%3A%20%22first%22%7D
  Host:lesswrong.com
  User-Agent:Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_3) AppleWebKit/537.31
  (KHTML, like Gecko) Chrome/26.0.1410.65 Safari/537.31

public void setRequestProperty(String name, String value)

uc.setRequestProperty("Cookie", "username=elharo; password=ACD0X9F23JJJn6G; session=100678945");

- Whenever the client requests a URL from that server, it includes a Cookie field in the HTTP request header that looks like this:

Cookie: username=elharo; password=ACD0X9F23JJJn6G; session=100678945;

5.7 Security Considerations for URLConnection

- URLConnection objects are subject to all the usual security restrictions about making network connections, reading or writing files, and so forth.
- Before attempting to connect a URL, you may want to know whether the connection will be allowed. For this purpose, the URLConnection class has a getPermission() method:

public Permission getPermission() throws IOException

- This returns a java.security.Permission object that specifies what permission is needed to connect to the URL. It returns null if no permission is needed (e.g., there's no security manager in place). Subclasses of URLConnection return different subclasses of java.security.Permission.

5.8 Guessing MIME Media Types

- The **URLConnection class** provides **two static methods to help programs figure out the MIME type of some data**; you can use these if the content type just isn't available or if you have reason to believe that the content type you're given isn't correct.
- The first of these is

URLConnection.guessContentTypeFromName():

public static String guessContentTypeFromName(String name)

- This method tries to guess the content type of an object based upon the extension in the filename portion of the object's URL.
- The second MIME type guesser method is

URLConnection.guessContentTypeFromStream():

public static String guessContentTypeFromStream(InputStream in)

- This method tries to guess the content type by looking at the first few bytes of data in the stream.

5.9 HTTPURLConnection

- The java.net.HttpURLConnection class is an abstract subclass of URLConnection; it provides some additional methods that are helpful when working specifically with httpURLs.
- Cast that URLConnection to HttpURLConnection like this:

```
URL u = new URL("http://lesswrong.com/");
URLConnection uc = u.openConnection();
HttpURLConnection http = (HttpURLConnection) uc;
```

Or, skipping a step, like this:

```
URL u = new URL("http://lesswrong.com/");
HttpURLConnection http = (HttpURLConnection) u.openConnection();
```

The Request Method

- When a web client contacts a web server, the first thing it sends is a request line.
- Typically, this line begins with GET and is followed by the path of the resource that the client wants to retrieve and the version of the HTTP protocol that the client understands.

For example:

GET /catalog/jfcnut/index.html HTTP/1.0

- For example, here's how a browser **asks for just the header** of a document using HEAD:

HEAD /catalog/jfcnut/index.html HTTP/1.1

Host: www.oreilly.com

Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2

Connection: close

- By default, HttpURLConnection uses the GET method.
- We can change this with the setRequestMethod() method:

public void setRequestMethod(String method) throws ProtocolException

The method argument should be one of these seven case-sensitive strings:

- GET
- POST
- HEAD
- PUT
- DELETE
- OPTIONS
- TRACE

Handling Server Responses

- The first line of an HTTP server's response includes a numeric code and a message indicating what sort of response is made. For instance, the most common response is 200 OK, indicating that the requested document was found.
  For example:

HTTP/1.1 200 OK

Cache-Control:max-age=3, must-revalidate

Connection:Keep-Alive

Content-Type:text/html; charset=UTF-8

Date:Sat, 04 May 2013 14:01:16 GMT

Keep-Alive:timeout=5, max=200

Server:Apache

Transfer-Encoding:chunked

Vary:Accept-Encoding,Cookie

WP-Super-Cache:Served supercache file from PHP

<HTML>

<HEAD>

rest of document follows...

public int getResponseCode() throws IOException

public String getResponseMessage() throws IOException

Error conditions:

- The getErrorStream() method returns an InputStream containing this page or null if no error was encountered or no data returned:

public InputStream getErrorStream()

Redirects

public static boolean getFollowRedirects()

public static void setFollowRedirects(boolean follow)

public boolean getInstanceFollowRedirects()

public void setInstanceFollowRedirects(boolean followRedirects)

Proxies

public abstract boolean usingProxy()

Streaming Mode: (not known size)

public void setChunkedStreamingMode(int chunkLength)

public void setFixedLengthStreamingMode(int contentLength)

public void setFixedLengthStreamingMode(long contentLength) // Java 7

Program to print all response from server.

```
import java.io.*;
import java.net.*;
public class Example6 {
 public static void main(String[] args) {
  try {
   // Open the URLConnection for reading
   URL u = new URL("http://www.prime.edu.np");
   HttpURLConnection uc = (HttpURLConnection) u.openConnection();
   int code = uc.getResponseCode();//200
   String response = uc.getResponseMessage();//OK
   System.out.println("HTTP/1.x " + code + " " + response);
   int i=1;
   while(uc.getHeaderField(i)!=null) {//value check in key:value pair
    System.out.println(uc.getHeaderFieldKey(i)+":"+uc.getHeaderField(i));
    i++;
   }
   System.out.println();
   BufferedReader in_data = new BufferedReader(new InputStreamReader(uc.getInputStream()));
   String entry;
   while((entry=in_data.readLine())!=null) {
    System.out.println(entry);
   }} catch (MalformedURLException ex) {
       System.err.println("It is not a parseable URL");
    } catch (IOException ex) {
       System.err.println(ex);}}}
```

THANK YOU FOR YOUR ATTENTION