

UNIT 3

URLs and URIs

(5 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.

3.1 URIs, URLs and Relative URLs

A **URI** (Uniform Resource Identifier) is a string that identifies a resource. A **URL** (Uniform Resource Locator) is a URI that also tells *where* the resource is and *how* to access it (e.g., `http://...`). A **URN** (Uniform Resource Name) names a resource without giving its location (e.g., `urn:isbn:0596007213`).

General syntax: **scheme:scheme-specific-part#fragment**. For hierarchical URIs:

```
scheme://userinfo@host:port/path?query#fragment  
https://user:pw@www.example.com:8443/docs/a.html?id=5#top
```

Part	Example	Meaning
scheme / protocol	https	How to access the resource
userinfo	user:pw	Optional login information
host	www.example.com	Server name or IP
port	8443	Optional; default depends on scheme (http 80, https 443)
path	/docs/a.html	Location on the server
query	id=5	Parameters, after ?
fragment / ref	top	Position inside the resource, after #

Relative URLs

A **relative URL** lacks some parts and is interpreted against a **base URL**. If the document at `http://www.example.com/a/b/c.html` contains a link `d.html`, it resolves to `http://www.example.com/a/b/d.html`. A link `../x.html` resolves to `http://www.example.com/a/x.html`, and `/y.html` resolves to `http://www.example.com/y.html`.

3.2 The URL Class

`java.net.URL` is an **immutable** class representing a URL and offering ways to retrieve data from it. It depends on protocol handlers; Java supports `http`, `https`, `ftp`, `file` and `jar` by default. A URL for an unsupported protocol throws **MalformedURLException**.

Creating new URLs

Constructor	Use
<code>URL(String spec)</code>	Absolute URL from a string
<code>URL(String protocol, String host, String file)</code>	Built from parts (default port)
<code>URL(String protocol, String host, int port, String file)</code>	Built from parts with port
<code>URL(URL base, String relative)</code>	Resolves a relative URL against a base

Retrieving data from a URL

Method	Returns
<code>InputStream openStream()</code>	Stream of the raw data (no headers)
<code>URLConnection openConnection()</code>	Connection object giving control over headers and options (Unit 5)
<code>Object getContent()</code>	Content converted to an object (text stream, image, etc.) by a content handler

```
URL u = new URL("http://www.example.com/index.html");
try (InputStream in = u.openStream();
    BufferedReader r = new BufferedReader(new InputStreamReader(in))) {
    String line;
    while ((line = r.readLine()) != null) System.out.println(line);
}
```

Listing 3.1 — Downloading a page with `openStream()`

Splitting a URL into pieces

Method	Returns (for <code>https://user@www.ex.com:8443/a/b.html?x=1#top</code>)
<code>getProtocol()</code>	<code>https</code>
<code>getHost()</code>	<code>www.ex.com</code>
<code>getPort()</code>	<code>8443</code> (–1 if not given)

Method	Returns (for <code>https://user@www.ex.com:8443/a/b.html?x=1#top</code>)
<code>getDefaultPort()</code>	443 for https
<code>getFile()</code>	<code>/a/b.html?x=1</code> (path + query)
<code>getPath()</code>	<code>/a/b.html</code>
<code>getQuery()</code>	<code>x=1</code>
<code>getRef()</code>	<code>top</code>
<code>getUserInfo()</code> / <code>getAuthority()</code>	<code>user</code> / <code>user@www.ex.com:8443</code>

Equality, comparison and conversion

- **`equals()`** compares protocol, host, port, file and fragment; to compare hosts it **may perform DNS lookups**, which can be slow and give surprising results.
- **`sameFile(URL)`** is like `equals()` but ignores the fragment.
- **`toString()`, `toExternalForm()`** return the URL as a string; **`toURI()`** converts to a URI.

3.3 The URI Class

`java.net.URI` is a pure **parsing and syntax** class: it never opens connections and does not need a protocol handler, so it can represent any scheme. It throws the checked **`URISyntaxException`** when the text is not a valid URI. It is also immutable.

Constructing a URI

- **`new URI(String)`** — parse a complete string.
- **`new URI(scheme, host, path, fragment)`, `new URI(scheme, userInfo, host, port, path, query, fragment)`** — build from parts (illegal characters are quoted automatically).
- **`URI.create(String)`** — static factory that throws an unchecked `IllegalArgumentException`.

The parts of a URI

A URI is either **opaque** (e.g., `mailto:a@b.com`, no slash after the scheme) or **hierarchical**. Getters include `getScheme()`, `getSchemeSpecificPart()`, `getAuthority()`, `getUserInfo()`, `getHost()`, `getPort()`, `getPath()`, `getQuery()`, `getFragment()`. Each has a `getRawXxx()` version that returns the text without decoding %-escapes. `isAbsolute()` and `isOpaque()` test the kind.

Resolving relative URIs

- **resolve(URI) / resolve(String)** — resolves the given URI against this base URI.
- **relativize(URI)** — the reverse: makes a relative URI from an absolute one.
- **normalize()** — removes “.” and “..” segments.

```
URI base = new URI("http://www.example.com/a/b/c.html");
URI r = base.resolve("../d.html");
System.out.println(r); // http://www.example.com/a/d.html
```

Equality, comparison and string representation

URI.equals() compares the parts case-sensitively where required (scheme and host are case-insensitive) and does **not** do DNS lookups, unlike URL. URI also implements **Comparable**. toString() returns the original (or constructed) string, toASCIIString() returns a pure-ASCII form, and toURL() converts to a URL.

Point	URL	URI
Purpose	Locate and retrieve resource	Identify and parse
Network access	Yes (openStream, openConnection)	No
Needs protocol handler	Yes	No
equals()	May do DNS lookup	Purely syntactic
Checked exception	MalformedURLException	URISyntaxException

3.4 x-www-form-urlencoded: URLEncoder and URLDecoder

URLs may safely contain only letters, digits and a few symbols. Other characters must be **percent-encoded**. The **x-www-form-urlencoded** format is used for HTML form data and query strings: spaces become +, unsafe bytes become %HH, and name=value pairs are joined with &.

- **URLEncoder.encode(String s, String charset)** — encodes text. Letters, digits and . - * _ are unchanged.
- **URLDecoder.decode(String s, String charset)** — decodes it.
- Encode each *name* and *value* separately, never the whole URL (or the &, = and / would be encoded too).
- Always specify the charset, normally "UTF-8".

```
String q = "name=" + URLEncoder.encode("Ram Sharma", "UTF-8")
        + "&city=" + URLEncoder.encode("Kathmandu & Patan", "UTF-8");
// name=Ram+Sharma&city=Kathmandu+%26+Patan
String back = URLDecoder.decode("Ram+Sharma", "UTF-8");    // Ram Sharma
```

3.5 Proxies

A **proxy server** sits between a client and the Internet, forwarding requests. It is used for security, filtering, caching and anonymity.

System properties

Property	Meaning
http.proxyHost / http.proxyPort	Proxy for HTTP
https.proxyHost / https.proxyPort	Proxy for HTTPS
ftp.proxyHost / ftp.proxyPort	Proxy for FTP
socksProxyHost / socksProxyPort	SOCKS proxy
http.nonProxyHosts	Hosts that bypass the proxy, separated by

They can be set with `-Dhttp.proxyHost=proxy.x.com`
`-Dhttp.proxyPort=8080` on the command line or with `System.setProperty()`.

The Proxy class

`java.net.Proxy` represents a proxy explicitly, with a **type** (`Proxy.Type.HTTP`, `SOCKS` or `DIRECT`) and a socket address. It is passed to `url.openConnection(Proxy)` to use a proxy for one connection only. `Proxy.NO_PROXY` forces a direct connection.

```
SocketAddress addr = new InetSocketAddress("proxy.example.com", 8080);
Proxy p = new Proxy(Proxy.Type.HTTP, addr);
URLConnection c = new URL("http://www.example.com").openConnection(p);
```

The ProxySelector class

`ProxySelector` is an abstract class that decides **which proxy to use for each URI**. Subclass it and implement `select(URI)` (returns a List of Proxy) and `connectFailed(URI, SocketAddress, IOException)` (called when a proxy fails). Install it with `ProxySelector.setDefault(selector)`.

3.6 Communicating with Server-Side Programs Through GET

Server-side programs (CGI, servlets, PHP, JSP) receive input through the HTTP request. With the **GET** method the input is placed in the query string of the URL: `http://host/search?q=java&page=2`. The client program builds the query string (with `URLEncoder`), creates the URL and reads the response with `openStream()`.

```
String query = "q=" + URLEncoder.encode("java sockets", "UTF-8");
URL u = new URL("http://www.example.com/search?" + query);
try (Scanner s = new Scanner(u.openStream())) {
    while (s.hasNextLine()) System.out.println(s.nextLine());
}
```

*Note: GET should be used only for **safe, idempotent** requests (reading data). URLs have length limits and appear in logs, so do not put passwords or large data in GET queries; use POST (Unit 4).*

3.7 Accessing Password-Protected Sites

When a site uses HTTP Basic (or similar) authentication, the server replies **401 Unauthorized** with a `WWW-Authenticate` header. Java can supply credentials automatically through the **Authenticator** mechanism.

The Authenticator class

`java.net.Authenticator` is an abstract class. Subclass it and override `getPasswordAuthentication()`, then register it with `Authenticator.setDefault(myAuthenticator)`. Inside the method you can call `getRequestingHost()`, `getRequestingPort()`, `getRequestingPrompt()`, `getRequestingScheme()`, `getRequestingURL()` to learn who is asking.

The PasswordAuthentication class

`PasswordAuthentication` is a simple immutable holder for a **user name** (`String`) and a **password** (`char[]`). Methods: `getUserName()` and `getPassword()`.

The JPasswordField class

For a GUI program, `javax.swing.JPasswordField` is a text field that hides typed characters; `getPassword()` returns a `char[]` which can be given to `PasswordAuthentication`. Using `char[]` (rather than `String`) lets the program overwrite the password in memory after use.

```

class MyAuth extends Authenticator {
    protected PasswordAuthentication getPasswordAuthentication() {
        return new PasswordAuthentication("user", "secret".toCharArray());
    }
}
// in main:
Authenticator.setDefault(new MyAuth());
InputStream in = new URL("http://www.example.com/private/").openStream();

```

Listing 3.2 — Supplying credentials with Authenticator

Key Takeaways

- ✓ URI identifies, URL locates, URN names; a relative URL is resolved against a base.
- ✓ URL: `openStream()`, `openConnection()`, `getContent()`, and part getters (`getHost`, `getPath`, `getQuery...`). `equals()` may use DNS.
- ✓ URI: syntax-only, no network access; `resolve()`, `relativize()`, `normalize()`; throws `URISyntaxException`.
- ✓ `URLEncoder/URLDecoder` handle x-www-form-urlencoded data; encode each parameter, use UTF-8.
- ✓ Proxies: system properties, `Proxy` class for one connection, `ProxySelector` for a policy.
- ✓ GET sends parameters in the URL; `Authenticator` + `PasswordAuthentication` supply credentials; `JPasswordField` for GUI input.

— *End of Unit 3* —