



PGT307: URL & URIs

Mohamed Elshaikh

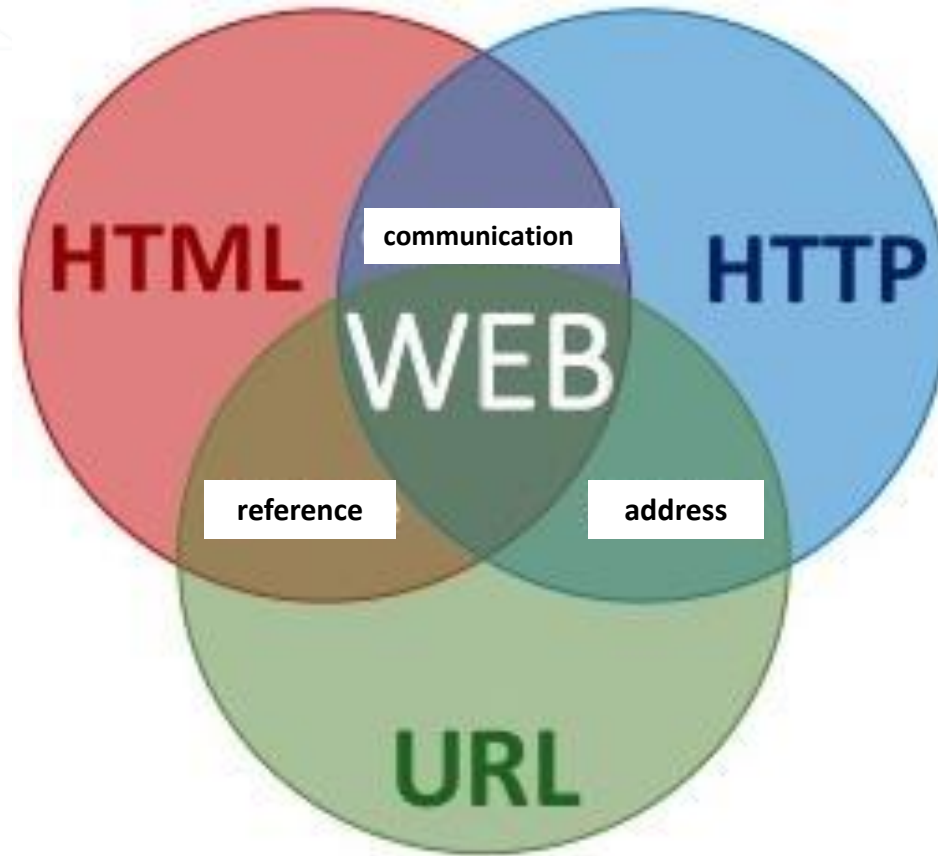
Faculty of Electronics Engineering Technology – UniMAP (FTKEN-UniMAP)

Objectives

- ❖ **ANALYZE** and **SHOW** a powerful abstraction for downloading information and files from network servers and the URL class enables to connect and download files and documents from a network server.
 - ➔ URLs and URIs class
 - ➔ Proxies
 - ➔ Communicating with Server
 - ➔ Accessing Password
 - ➔ URL Connections

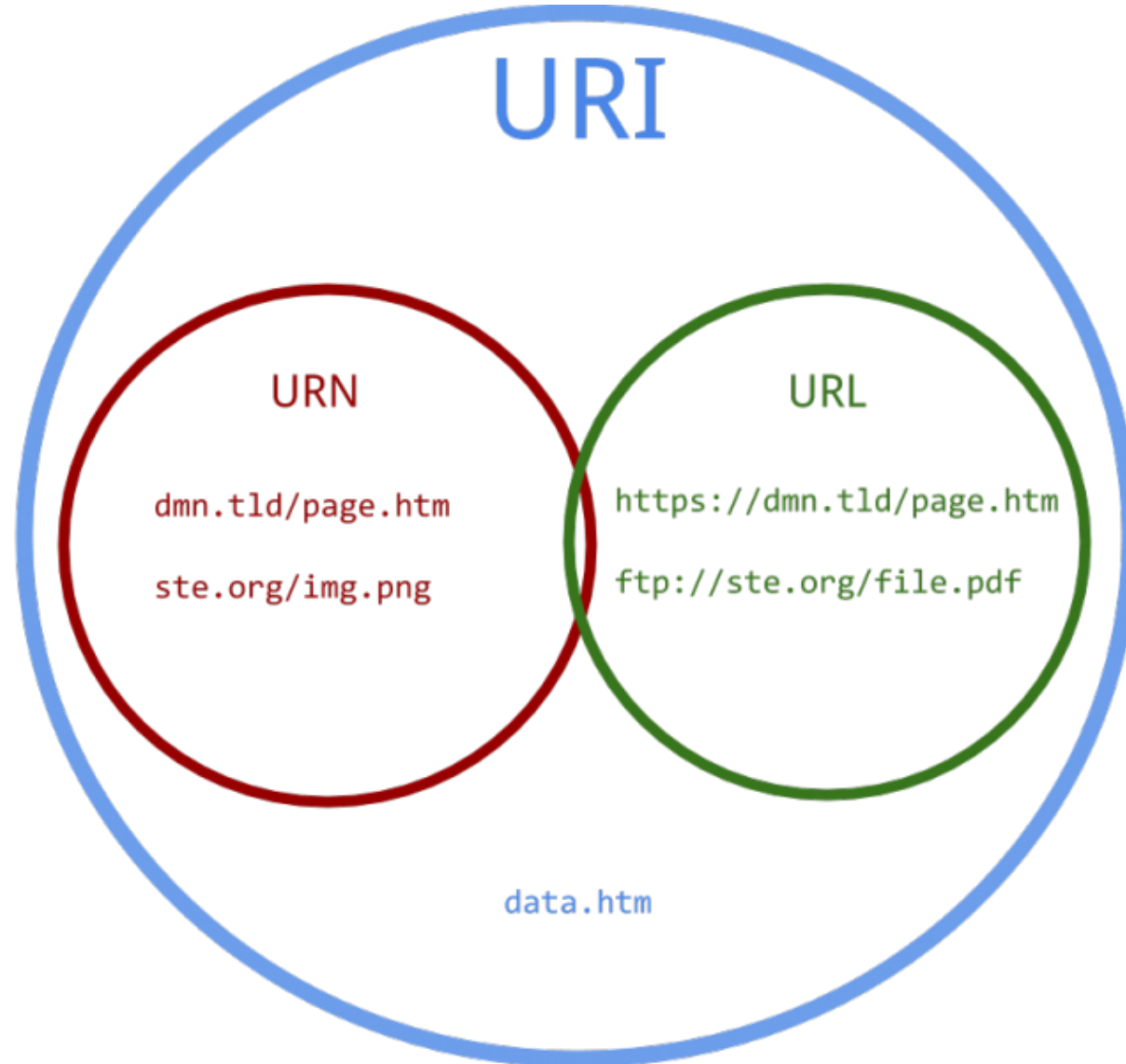
The Web

- ◎ HTML includes a way to specify links to other documents identified by **URLs**.



- ❖ The **URL class** is the simplest way for a Java program to locate and retrieve data from the network.
- ❖ You do not need to worry about the details of the protocol being used, or how to communicate with the server.

URI, URL, URN



URIs

- ❖ **Uniform Resource Identifier (URI)**
- ❖ A string of characters in a particular syntax that identifies a **resource**.
- ❖ The resource identified may be **a file on a server**; but it may also be an email address, a news message, etc.
- ❖ All you ever receive from a server is a representation of a resource which comes **in the form of bytes**.
- ❖ However a single resource may have different representations. Eg: html, pdf, txt, jpeg

URI Syntax

- ❖ URIs are composed of a scheme and a scheme-specific part, separated by a colon, like this:

scheme:scheme-specific-part

- ❖ Current scheme includes:

Scheme	Description
data	Base64-encoded data included directly in a link; see RFC 2397
file	A file on a local disk
ftp	An FTP server
http	A World Wide Web server using the Hypertext Transfer Protocol
mailto	An email address
magnet	A resource available for download via peer-to-peer networks such as BitTorrent
telnet	A connection to a Telnet-based service
urn	A Uniform Resource Name

URI Syntax

- ❖ There is no specific syntax that applies to the scheme-specific parts of all URIs.
- ❖ However, many have a hierarchical form, like this:

//authority/path?query

The structure of URIs



URI, URL, URN

<http://thinkzarahatke.com/author/amty.html#posts>

URI

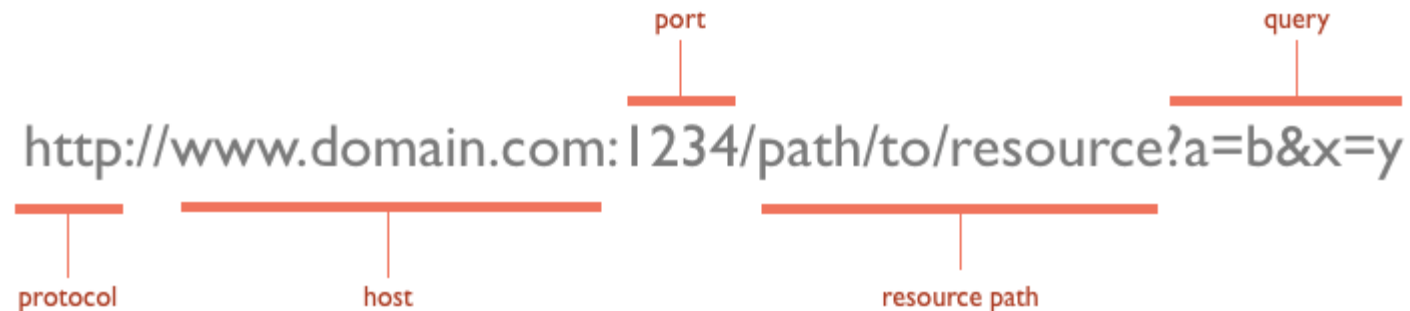
- © **java.net.URI** class only identifies resources and the **java.net.URL** class can both identify and retrieve resources

URLs

- ❖ A **URL (Uniform Resource Locator)** is a URI that, as well as identifying a resource, **provides a specific network location** for the resource that a client can use to retrieve a representation of that resource.
- ❖ By contrast, a generic URI may tell you what a resource is, but not actually tell you where or how to get that resource.
- ❖ Syntax:

protocol://userInfo@host:port/path?query#fragment

- ❖ Example:


http://www.domain.com:1234/path/to/resource?a=b&x=y

The diagram shows the URL `http://www.domain.com:1234/path/to/resource?a=b&x=y` with red brackets and labels identifying its parts: `http` is labeled 'protocol', `www.domain.com` is labeled 'host', `:1234` is labeled 'port', `/path/to/resource` is labeled 'resource path', and `?a=b&x=y` is labeled 'query'.

Absolute vs Relative URLs

- ❖ Rather than including the full (absolute) URL for each page, you can use a **relative URL**.
- ❖ A relative URL indicates where the resource is in relation to the current page.
- ❖ For example to link from the index page to the chapter 3 page:

Absolute link:

```
<a href="http://theweb.net/chapter3/index.html">Chapter 3</a>
```

Relative link:

```
<a href="chapter3/index.html">Chapter 3</a>
```

The URL class

- ❖ The **java.net.URL** class

- ➔ an abstraction of a Uniform Resource Locator

- ➔ Example: `http://www.lolcats.com/` or `ftp://ftp.redhat.com/pub/`

- ❖ URLs are **immutable**.

- ➔ *After* a URL object has been *constructed*, its **fields do not change**.

- ➔ This has the **side effect** of making them **thread safe**.

- ❖ **Constructors:**

```
public URL(String url) throws MalformedURLException
public URL(String protocol, String hostname, String file) throws MalformedURLException
public URL(String protocol, String host, int port, String file) throws MalformedURLException
public URL(URL base, String relative) throws MalformedURLException
```

- ❖ All these constructors throw a **MalformedURLException** if you try to create a URL for an **unsupported protocol** or if the URL is syntactically incorrect.

Example :

Create a URL object

```
try {  
    URL u = new URL("http://www.audubon.org/");  
} catch (MalformedURLException ex) {  
    System.err.println(ex);  
}
```

Retrieving Data From URL

- ❖ The URL class has several methods that retrieve data from a URL:

```
public InputStream openStream() throws IOException
public URLConnection openConnection() throws IOException
public URLConnection openConnection(Proxy proxy) throws IOException
public Object getContent() throws IOException
public Object getContent(Class[] classes) throws IOException
```

- ❖ `openStream()` → (most commonly used) returns an `InputStream` from which you can read the data.
- ❖ `openConnection()` → more control over the download process; gives a `URLConnection` which you can configure, and then get an `InputStream` from it
- ❖ `getContent()` → ask the URL for its content which may give you a more complete object such as `String` or an `Image`.

Example :

openStream method

```
//lets read the data from the main page of lolcats.com
try {
    URL u = new URL("http://www.lolcats.com");
    InputStream in = u.openStream();

    int c;
    while ((c = in.read()) != -1) {
        System.out.write(c);
    }
    in.close();
} catch (IOException ex) {
    System.err.println(ex);
}
```

Splitting a URL into Pieces

- ❖ Read-only access to the parts of a URL is provided by 9 public methods in the URL class:
- ❖ `getFile()` → returns a String that contains the path portion of a URL
- ❖ `getHost()` → returns a String containing the hostname of the URL
- ❖ `getPort()` → returns the port number specified in the URL as an int (return -1 if none)
- ❖ `getProtocol()` → returns a String containing the scheme of the URL
- ❖ `getRef()` → returns the fragment identifier part of the URL (returns null if none)
- ❖ `getQuery()` → returns the query string of the URL (null if none)
- ❖ `getPath()` → returns a String containing the path and file portion of a URL without query string
- ❖ `getUserInfo()` → returns user info such as username and password
- ❖ `getAuthority()` → returns the authority that resolves the resource

Example : The parts of a URL

```
import java.net.*;
public class URLSplitter { //the url is given in args parameter during execution
    public static void main(String args[]) {
        try {
            URL u = new URL("http://www.lolcats.com");
            System.out.println("The URL is " + u);
            System.out.println("The scheme is " + u.getProtocol());
            System.out.println("The user info is " + u.getUserInfo());
            String host = u.getHost();
            if (host != null) {
                int atSign = host.indexOf('@');
                if (atSign != -1) host = host.substring(atSign+1);
                System.out.println("The host is " + host);
            } else {
                System.out.println("The host is null.");
            }
            System.out.println("The port is " + u.getPort());
            System.out.println("The path is " + u.getPath());
            System.out.println("The ref is " + u.getRef());
            System.out.println("The query string is " + u.getQuery());
        } catch (MalformedURLException ex) {
            System.err.println(ex.getMessage());
        }
        System.out.println();
    }
}
```


Example :

Output

```
The URL is http://www.lolcats.com
The scheme is http
The user info is null
The host is www.lolcats.com
The port is -1
The path is
The ref is null
The query string is null
```

The URI class

- ❖ A URI
 - ➔ a generalization of a URL
 - ➔ includes Uniform Resource Locators AND Uniform Resource Names (URNs).
- ❖ Most URIs used in practice are URLs,
 - ➔ but most specifications and standards such as XML are defined in terms of URIs.
- ❖ In Java, URIs are represented by the **java.net.URI** class.
- ❖ Differs from the `java.net.URL` class in three ways:
 1. Provides no methods to retrieve a representation of the resource identified by its URI.
 2. More conformant to the relevant specifications than the URL class
 3. A URI object can represent a relative URI. The URL class absolutizes all URIs before storing them

URI class vs URL class

- ❖ **URL** object is a representation of an application layer protocol for **network retrieval**, whereas a **URI** object is **purely for string parsing and manipulation**.
- ❖ The URI class has no network retrieval capabilities.
- ❖ **URL class** → when you want to download the content at a URL
- ❖ **URI class** → when you want to use the URL for identification rather than retrieval.
- ❖ When you need to do both, you may convert from a URI to a URL with the **toURL()** method, and from a URL to a URI using the **toURI()** method.

Constructing a URI

❖ URI constructors:

```
public URI(String uri) throws URISyntaxException
public URI(String scheme, String schemeSpecificPart, String fragment)
    throws URISyntaxException
public URI(String scheme, String host, String path, String fragment)
    throws URISyntaxException
public URI(String scheme, String authority, String path, String query, String fragment)
    throws URISyntaxException
public URI(String scheme, String userInfo, String host, int port, String path,
    String query, String fragment) throws URISyntaxException
```

❖ If the string argument does not follow URI syntax rules—for example, if the URI begins with a colon—this constructor throws a `URISyntaxException`.

❖ Example:

```
URI absolute = new URI("http", "://www.ibiblio.org", null);
URI relative = new URI(null, "/javafaq/index.shtml", "today");
```

URI methods

- ❖ If the scheme is omitted, the URI reference is relative. If the fragment identifier is omitted, the URI reference is a pure URI.
- ❖ The URI class has getter methods that return these three parts of each URI object:
- ❖ `getRawFoo()` → return the encoded forms of the parts of the URI
- ❖ `getFoo()` → first decode any percent-escaped characters and then return the decoded part

```
public String getScheme()  
public String getSchemeSpecificPart()  
public String getRawSchemeSpecificPart()  
public String getFragment()  
public String getRawFragment()  
public String getAuthority()  
public String getHost()  
public String getPath()  
public String getPort()  
public String getQuery()  
public String getUserInfo()  
public URI parseServerAuthority() throws URISyntaxException
```

Proxies

- ❖ Java programs based on the **URL class can work through most common proxy servers and protocols.**
- ❖ This is one reason you might want to choose to use the URL class rather than rolling your own HTTP or other client on top of raw sockets.

The Proxy Class

- ❖ Allows more fine-grained control of proxy servers from within a Java program.
- ❖ Allow to choose different proxy servers for different remote hosts.
- ❖ The proxies themselves are represented by instances of the **java.net.Proxy** class.
- ❖ Three kinds of proxies represented by three constants in the Proxy.Type enum:
 - Proxy.Type.DIRECT
 - Proxy.Type.HTTP
 - Proxy.Type.SOCKS
- ❖ Example: create a Proxy object representing an HTTP proxy server on port 80 of proxy.example.com:

```
SocketAddress address = new InetSocketAddress("proxy.example.com", 80);  
Proxy proxy = new Proxy(Proxy.Type.HTTP, address);
```

Communicating with Server-Side Programs Through GET

- ❖ The URL class makes it easy for Java applets and applications to communicate with server-side programs such as CGIs, servlets, PHP pages, and others that use the GET method.
- ❖ All you need to know is what combination of names and values the program expects to receive.
- ❖ Then you can construct a URL with a query string that provides the requisite names and values.
- ❖ All names and values must be x-www-form-urlencoded—as by the `URLEncoder.encode()` method.

Example :

Server-side Program (HTML form)

```
<form action="search" method="GET">  
  <input size="45" name="q" value="" >  
  <input value="Search" type="submit">  
</form>
```

- The Open Directory interface is a simple form with one input field named search
- Input typed in this field is sent to a program at <http://search.dmoz.org/cgi-bin/search>, which does the actual search.
- to submit a search request to the Open Directory, you just need to append **q=searchTerm** to <http://www.dmoz.org/search>
- Eg: Type this URL directly into the browser address
<http://www.dmoz.org/search?q=java>

WE CAN DO THIS IN JAVA... →

Example : Client-side Program (Java)

```
import java.io.*;
import java.net.*;
public class DMoz {
    public static void main(String[] args) {
        String target = "";
        for (int i = 0; i < args.length; i++) {
            target += args[i] + " ";
        }
        target = target.trim();
        QueryString query = new QueryString("/");  
//QueryString class must be written. See e-book page 153
        query.add("q", target);
        try {
            URL u = new URL("http://www.dmoz.org/search/q?" + query);
            try (InputStream in = new BufferedInputStream(u.openStream())) {
                InputStreamReader theHTML = new InputStreamReader(in);
                int c;
                while ((c = theHTML.read()) != -1) {
                    System.out.print((char) c);
                }
            } catch (MalformedURLException ex) {
                System.err.println(ex);
            } catch (IOException ex) {
                System.err.println(ex);
            }
        }
    }
}
```

Output:

Result of the query will be printed on the console.



THANK YOU