# Network Programming
# [CACS355]


## Er.Sital Prasad Mandal

**BCA- VI**
**Mechi Campus**
**Bhadrapur, Jhapa, Nepal**

(Email : info.sitalmandal@gmail.com)

# Text Book

- Elliotte Rusty Harold, “Java Network Programming” O’Reilly, 2014
- David Reilly, Michael Reilly, “ Java Netwoking Programming and DistributedComputing”

# Unit - 3

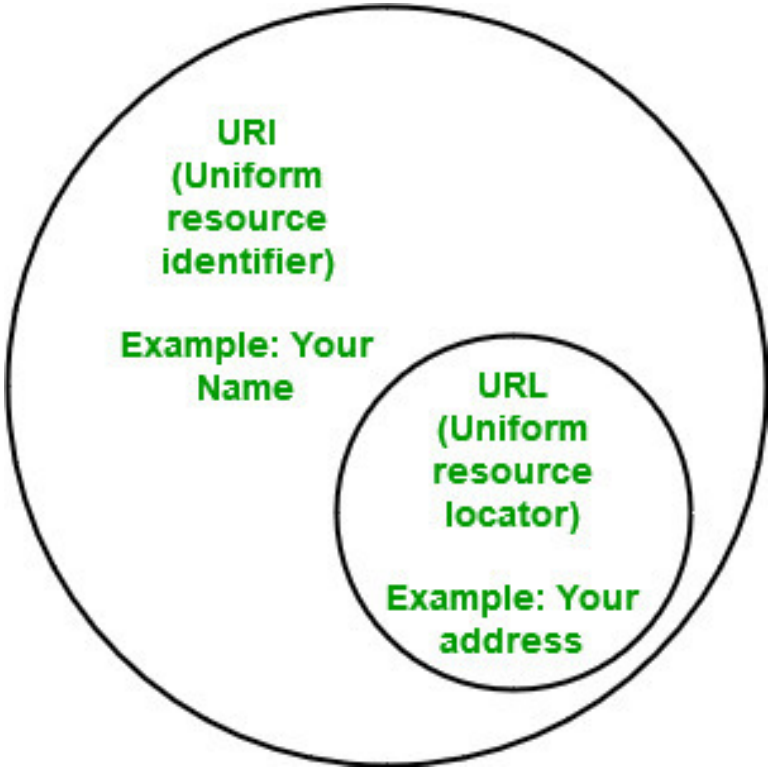
## 3.1 URLs and URIs

- URIs
- URLs
- Relative URLs

# Unit - 3

## URLs and URIs

| URL | URI |
|---|---|
| URL is used to describe the identity of an item. | URI provides a technique for defining the identity of an item. |
| URL links a web page, a component of a web page or a program on a web page with the help of accessing methods like protocols. | URI is used to differentiate one resource from other regardless of the method used. |
| URL provides the details about what type of protocol is to be used. | URI doesn't contains the protocol specification. |
| URL is a type of URI. | URI is the superset of URL. |



# URIs

- A Uniform Resource Identifier (URI) is 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, a book, a person's name, an Internet host, the current stock price of Oracle, or something else.

# URIs

- syntax

*scheme:scheme-specific-part*

- *Data*
- *File*
- *ftp*
- *http*
- *mailto*
- *Magnet*
- *telnet*
- *urn*

**hierarchical form:**

*//authority/path?query*

For instance, the URI *http://www.ietf.org/rfc/rfc3986.txt has the scheme http,* the authority *www.ietf.org, and the path /rfc/rfc3986.txt .*

*This* means the server at www.ietf.org is responsible for mapping the path */rfc/rfc3986.txt to* a resource. This URI does not have a query part.

The URI *ttp://www.powells.com/cgibin/biblio?inkey=62-1565928709-0 has the scheme http, the authority www.powells.com, the path /cgi-bin/biblio, and the query inkey=62-1565928709-0.*

# URIs

The scheme part is composed of lowercase letters, digits, and the plus sign, period, and hyphen.

- ASCII
- Digits
- Delimiters i.e. | ? $
- Punctuation characters: / $@ ~ -

# URLs

A URL 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.

A generic URI may tell you what a resource is, but not actually tell you where or how to get that resource.

In Java, it's the difference between the java.net.URI class that only identifies resources and

the java.net.URL class that can both identify and retrieve resources.

The syntax of a URL is:
*protocol://userInfo@host:port/path?query#fragment*

http://www.cafeaulait.org/javafaq.html#xtocid1902914

In a URL, the protocol part can be *file, ftp, http, https, magnet, telnet, or* various other strings

# Relative URLs

**Relative URLs**

A URL may inherit the protocol, hostname, and path of its parent document.

URLs that aren't complete but inherit pieces from their parent are called *relative URLs.*

A completely specified URL is called an *absolute URL.*

*Eg:* *http://www.ibiblio.org/javafaq/javatutorial.html*

<a href="javafaq.html">

The browser cuts *javatutorial.html off the end of ttp://www.ibiblio.org/javafaq/javatutorial.html to get http://www.ibiblio.org/javafaq/. Then it attaches javafaq.html onto* the end of *http://www.ibiblio.org/javafaq/ to get http://www.ibiblio.org/javafaq/javafaq.html. Finally, it loads that document.*

# Relative URLs

If the relative link begins with a /, then it is relative to the document root instead of relative to the current file.
Thus, if we click on link
*http://www.ibiblio.org/javafaq/javatutorial.html:*

<a href="/projects/ipv6/">

the browser would throw away */javafaq/javatutorial.html and attach /projects/ipv6/ to* the end of *http://www.ibiblio.org to get http://www.ibiblio.org/projects/ipv6/.*

***Relative URLs Important—***

- they save a little typing.
- allow a single document tree to be servedby multiple protocols: for instance, both HTTP and FTP. HTTP might be used for direct surfing, while FTP could be used for mirroring the site.
- allow entire trees of documents to be moved or copied from one site to another without breaking all the internal links.

# The URL Class

## 3.2 The URL Class

1. Creating New URLs
2. Retrieving Data from a URL
3. Splitting a URL into Pieces
4. Equality and Comparison
5. Conversion

# The URL Class

The java.net.URL class is an abstraction of a Uniform Resource Locator such as *http://www.lolcats.com/ or ftp://ftp.redhat.com/pub/.*

*It extends java.lang.Object, and it is a* final class that cannot be subclassed.

# The URL Class

## Creating New URLs

Instances of java.net.URL. The constructors differ in the information they require:

1. public URL(String url) throws MalformedURLException
2. public URL(String protocol, String hostname, String file) throws MalformedURLException
3. public URL(String protocol, String host, int port, String file) throws MalformedURLException
4. public URL(URL base, String relative) throws MalformedURLException

# The URL Class

Constructing a URL from a string

The simplest URL constructor just takes an absolute URL in string form as its single argument:

```
public URL(String url) throws MalformedURLException
```

**Example:**

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

*Example :Which protocols does a virtual machine support?*

```
import java.net.*;
public class ProtocolTester {
public static void main(String[] args) {
// hypertext transfer protocol
testProtocol("http://www.adc.org");
// secure http
testProtocol("https://www.amazon.com/exec/obidos/order2/");
// file transfer protocol
testProtocol("ftp://ibiblio.org/pub/languages/java/javafaq/");
// Simple Mail Transfer Protocol
testProtocol("mailto:elharo@ibiblio.org");
// telnet
testProtocol("telnet://dibner.poly.edu/");
// local file access
testProtocol("file:///etc/passwd");
// gopher
testProtocol("gopher://gopher.anc.org.za/");
// Lightweight Directory Access Protocol
testProtocol(
"ldap://ldap.itd.umich.edu/o=University%20of%20Michigan,c=US?postalAddress");
// JAR
testProtocol(
"jar:http://cafeaulait.org/books/javaio/ioexamples/javaio.jar!"
+ "/com/macfaq/io/StreamCopier.class");
// NFS, Network File System
testProtocol("nfs://utopia.poly.edu/usr/tmp/");
// a custom protocol for JDBC
testProtocol("jdbc:mysql://luna.ibiblio.org:3306/NEWS");
// rmi, a custom protocol for remote method invocation
testProtocol("rmi://ibiblio.org/RenderEngine");
// custom protocols for HotJava
testProtocol("doc:/UsersGuide/release.html");
testProtocol("netdoc:/UsersGuide/release.html");
testProtocol("systemresource://www.adc.org/+/index.html");
testProtocol("verbatim:http://www.adc.org/");
}
```

```
private static void testProtocol(String url) {
try {
URL u = new URL(url);
System.out.println(u.getProtocol() + " is supported");
} catch (MalformedURLException ex) {
String protocol = url.substring(0, url.indexOf(':'));
System.out.println(protocol + " is not supported");
}
}
}
```

The results of this program depend on which virtual machine runs it. Here are the results from Java 7 on Mac OS X:

```
http is supported
https is supported
ftp is supported
mailto is supported
telnet is not supported
file is supported
gopher is not supported
ldap is not supported
jar is supported
nfs is not supported
jdbc is not supported
rmi is not supported
doc is not supported
netdoc is supported
systemresource is not supported
verbatim is not supported
```

# The URL Class

Constructing a URL from its component parts

You can also build a URL by specifying the protocol, the hostname, and the file:

```
public URL(String protocol, String hostname, String file)
throws MalformedURLException
```

```
try {
URL u = new URL("http", "www.eff.org", "/blueribbon.html#intro");
} catch (MalformedURLException ex)
{
throw new RuntimeException("shouldn't happen; all VMs recognize http");
}
```

# The URL Class

```
public URL(String protocol, String host, int port, String file)
throws MalformedURLException
```

*http://fourier.dur.ac.uk:8000/~dma3mjh/jsci/, specifying port 8000 explicitly:*

```
try {
URL u = new URL("http", "fourier.dur.ac.uk", 8000, "/~dma3mjh/jsci/");
} catch (MalformedURLException ex) {
throw new RuntimeException("shouldn't happen; all VMs recognize
http");
}
```

# The URL Class

Constructing relative URLs

```
public URL(URL base, String relative) throws MalformedURLException
```

The constructor computes the new URL as *http://www.ibiblio.org/javafaq/mailinglists.html. For example:*

```
try {
URL u1 = new URL("http://www.ibiblio.org/javafaq/index.html");
URL u2 = new URL (u1, "mailinglists.html");
} catch (MalformedURLException ex) {
System.err.println(ex);
}
```

# The URL Class

Retrieving Data from a URL

1. `public InputStream openStream() throws IOException`
2. `public URLConnection openConnection() throws IOException`
3. `public URLConnection openConnection(Proxy proxy) throws IOException`
4. `public Object getContent() throws IOException`
5. `public Object getContent(Class[] classes) throws IOException`

# The URL Class

public final InputStream openStream() throws IOException

```
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);
}
```

# The URL Class

For example: java-6

```
InputStream in = null
try {
URL u = new URL("http://www.lolcats.com");
in = u.openStream();
int c;
while ((c = in.read()) != -1) System.out.write(c);
} catch (IOException ex) {
System.err.println(ex);
} finally {
try {
if (in != null) {
in.close();
}
} catch (IOException ex) {
// ignore
}
}
```

# The URL Class

Java 7 makes this somewhat cleaner by using a nested try-with-resources statement:

```
try {
URL u = new URL("http://www.lolcats.com");
try (InputStream in = u.openStream()) {
int c;
while ((c = in.read()) != -1) System.out.write(c);
}
} catch (IOException ex) {
System.err.println(ex);
}
```

*Example 5-2. Download a web page*

```
import java.io.*;
import java.net.*;
public class SourceViewer {
public static void main (String[] args) {
if (args.length > 0) {
InputStream in = null;
try {
// Open the URL for reading
URL u = new URL(args[0]);
in = u.openStream();
// buffer the input to increase performance
in = new BufferedInputStream(in);
// chain the InputStream to a Reader
Reader r = new InputStreamReader(in);
int c;
while ((c = r.read()) != -1) {
System.out.print((char) c);
}
} catch (MalformedURLException ex) {
System.err.println(args[0] + " is not a parseable URL");
} catch (IOException ex) {
System.err.println(ex);
} finally {
if (in != null) {
try {
in.close();
} catch (IOException e) {
// ignore
}
}
}
}
 }
}
```

And here are the first few lines of output when SourceViewer downloads *https://www.oreilly.com:*

```
&lt;!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"&gt;
&lt;html xmlns="http://www.w3.org/1999/xhtml" lang="en-US"
xml:lang="en-US"&gt;
&lt;head&gt;
&lt;title&gt;oreilly.com -- Welcome to O'Reilly Media, Inc. --
computer books,
software conferences, online publishing&lt;/title&gt;
&lt;meta name="keywords" content="O'Reilly, oreilly, computer
books, technical
books, UNIX, unix, Perl, Java, Linux, Internet, Web, C, C++,
Windows, Windows
NT, Security, Sys Admin, System Administration, Oracle, PL/SQL,
online books,
books online, computer book online, e-books, ebooks, Perl
Conference, Open Source
Conference, Java Conference, open source, free software, XML,
Mac OS X, .Net, dot
net, C#, PHP, CGI, VB, VB Script, Java Script, javascript,
Windows 2000, XP,
```

# The URL Class

public final Object getContent() throws IOException

```
URL u = new URL("http://mesola.obspm.fr/");
Object o = u.getContent();
// cast the Object to the appropriate type
// work with the Object...
```

*Example 5-3. Download an object*

```
import java.io.*;
import java.net.*;
public class ContentGetter {
public static void main (String[] args) {
if (args.length > 0) {
// Open the URL for reading
try {
URL u = new URL(args[0]);
Object o = u.getContent();
System.out.println("I got a " + o.getClass().getName());
} catch (MalformedURLException ex) {
System.err.println(args[0] + " is not a parseable URL");
} catch (IOException ex) {
System.err.println(ex);
}
}
}
}
```

Here's the result of trying to get the content of *https://www.oreilly.com:*

```
% java ContentGetter http://www.oreilly.com/ I got a
sun.net.www.protocol.http.HttpURLConnection$HttpInp
utStream
```

# The URL Class

## Splitting a URL into Pieces

URLs are composed of five pieces:

- The scheme, also known as the protocol
- The authority
- The path
- The fragment identifier, also known as the section or ref
- The query string

# The URL Class

## public String getProtocol()

The getProtocol() method returns a String containing the scheme of the URL (e.g.,
“http”, “https”, or “file”). For example, this code fragment prints *https:*

```
URL u = new URL("https://xkcd.com/727/");
System.out.println(u.getProtocol());
```

## public String getHost()

The getHost() method returns a String containing the hostname of the URL. For
example, this code fragment prints *xkcd.com:*

```
URL u = new URL("https://xkcd.com/727/");
System.out.println(u.getHost());
```

# The URL Class

## public int getPort()

For example, if
the URL is *http://www.userfriendly.org/, getPort() returns -1; if the URL is http://www.userfriendly.org:80/, getPort() returns 80. The following code prints -1 for the* port number because it isn't specified in the URL:

```
URL u = new URL("http://www.ncsa.illinois.edu/AboutUs/");
System.out.println("The port part of " + u + " is " + u.getPort());
```

*Example 5-4. The parts of a URL*

```
import java.net.*;
public class URLSplitter {
public static void main(String args[]) {
for (int i = 0; i < args.length; i++) {
try {
URL u = new URL(args[i]);
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(args[i] + " is not a URL I understand.");
}
System.out.println();
}
}
}
```

Here's the result of running this against several of the URL examples in this chapter:

```
% java URLSplitter
ftp://mp3:mp3@138.247.121.61:21000/c%3a/ \
http://www.oreilly.com \
http://www.ibiblio.org/nywc/compositions.phtml?category=Piano
\
http://admin@www.blackstar.com:8080/ \
The URL is ftp://mp3:mp3@138.247.121.61:21000/c%3a/
The scheme is ftp
The user info is mp3:mp3
The host is 138.247.121.61
The port is 21000
The path is /c%3a/
The ref is null
The query string is null
```

# The URL Class

## Equality and Comparison

The URL class contains the usual equals() and hashCode() methods.

Two URLs are considered equal if and only if both URLs point to the same resource on the same host, port, and path, with the same fragment identifier and query string.

The equals() method actually tries to resolve the host with DNS so that, for example, it can tell that *http://www.ibiblio.org/ and http://ibiblio.org/ are the same.*

*On the other hand, equals() does not go so far as to actually compare the resources identified by two URLs. For example, http://www.oreilly.com/ is not equal to http://www.oreilly.com/index.html; and http://www.oreilly.com:80 is not equal to http://www.oreilly.com/.*

## *Example 5-5. Are http://www.ibiblio.org and http://ibiblio.org the same?*

```
import java.net.*;
public class URLEquality {
public static void main (String[] args) {
try {
URL www = new URL ("http://www.ibiblio.org/");
URL ibiblio = new URL("http://ibiblio.org/");
if (ibiblio.equals(www)) {
System.out.println(ibiblio + " is the same as " + www);
} else {
System.out.println(ibiblio + " is not the same as " +
www);
}
} catch (MalformedURLException ex) {
System.err.println(ex);
}
}
}
```

O/P:

http://ibiblio.org/ is the same as http://www.ibiblio.org/

# The URL Class

## Conversion

URL has three methods that convert an instance to another form:

1. toString()
2. toExternalForm()
3. toURI()

java.net.URL has a toString() method. The String produced by toString() is always an absolute URL, such as *http://www.cafeaulait.org/javatutorial.html. It's uncommon to call toString() explicitly. Print statements call to*String() implicitly. Outside of print statements, it's more proper to use toExternalForm() instead:

```
public String toExternalForm()
```

The toExternalForm() method converts a URL object to a string that can be used in an HTML link or a web browser's Open URL dialog. The toExternalForm() method returns a human-readable String representing the URL. It is identical to the toString() method. In fact, all the toString() method does is return toExternalForm().

Finally, the toURI() method converts a URL object to an equivalent URI object:

```
public URI toURI() throws URISyntaxException
```

# The URI Class

## 3.3 The URI Class

1. Constructing a URI
2. The Parts of the URI
3. Resolving Relative URIs
4. Equality and Comparison
5. String Representations

# KEY DIFFERENCES: URL & URI Class

**KEY DIFFERENCES:**

- ✓ URL is a subset of URI that specifies where a resource exists and the mechanism for retrieving it, while URI is a superset of URL that identifies a resource.
- ✓ The main aim of URL is to get the location or address of a resource whereas the main aim of URI is to find a resource.
- ✓ URL is used to locate only web pages, on the other hand, URI in used in HTML, XML and other files.
- ✓ URL contains components such as protocol, domain, path, hash, query string, etc. while, URI contains components like scheme, authority, path, query, etc.
- ✓ Example of URL is : https://google.com while example of URI is :urn:isbn:0-486-27557-4.

# 3.3 The URI Class

A URI is a generalization of a URL that includes not only Uniform Resource Locators but also 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. This class differs from the java.net.URL class in three important ways:

1. The URI class is purely about identification of resources and parsing of URIs. It provides no methods to retrieve a representation of the resource identified by its URI.
2. The URI class is 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.

# 3.3 The URI Class

## Constructing a URI

1. **public URI(String uri) throws URISyntaxException**
2. **public URI(String scheme, String schemeSpecificPart, String fragment) throws URISyntaxException**
3. **public URI(String scheme, String host, String path, String fragment) throws URISyntaxException**
4. **public URI(String scheme, String authority, String path, String query,** String fragment) **throws URISyntaxException**
5. **public URI(String scheme, String userInfo, String host, int port,** String path, String query, String fragment) **throws URISyntaxException**

# 3.3 The URI Class

The *first constructor* creates a new URI object from any convenient string. For example:

```
URI voice = new URI("tel:+1-800-9988-9938");
URI web = new URI("http://www.xml.com/pub/a/2003/09/17/stax.html#id=_hbc");
URI book = new URI("urn:isbn:1-565-92870-9");
```

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

The *second constructor* that takes a scheme specific part is mostly used for nonhierarchical URIs. The scheme is the URI's protocol, such as http, urn, tel, and so forth. of ASCII letters and digits and the three characters +, -, and .. It must begin with a letter. Passing null for this argument omits the scheme, thus creating a relative URI. For example:

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

# 3.3 The URI Class

The *third constructor* is used for hierarchical URIs such as http and ftp URLs. The host and path together (separated by a /) form the scheme-specific part for this URI. For example:

```
URI today= new URI("http", "www.ibiblio.org", "/javafaq/index.html", "today");
```

This produces the URI *http://www.ibiblio.org/javafaq/index.html#today.*

*URI has to be absolute but the path doesn't start with /—then it throws a URISyntaxException.*

The fourth constructor is basically the same as the third, with the addition of a query string. For example:

```
URI today = new URI("http", "www.ibiblio.org", "/javafaq/index.html“, "referrer=cnet&date=2014-02-23", "today");
```

# 3.3 The URI Class

The *fifth constructor* is the master hierarchical URI constructor that the previous two invoke. It divides the authority into separate user info, host, and port parts, each of which has its own syntax rules. For example:

```
URI styles = new URI("ftp", "anonymous:elharo@ibiblio.org“, "ftp.oreilly.com", 21, "/pub/stylesheet", null, null);
```

However, the resulting URI still has to follow all the usual rules for URIs; and again null can be passed for any argument to omit it from the result.

*If you’re sure your URIs are legal and do not violate any of the rules, you can use the static factory URI.create() method instead.*

For example, this invocation creates a URI for anonymous FTP access using an email address as password:

```
URI styles = URI.create("ftp://anonymous:elharo%40ibiblio.org@ftp.oreilly.com:21/pub/stylesheet");
```

If the URI does prove to be malformed, then an IllegalArgumentException is thrown by this method.

# 3.3 The URI Class

## The Parts of the URI

A URI reference has up to three parts: a scheme, a scheme-specific part, and a fragment identifier. The general format is:

*scheme:scheme-specific-part:fragment*

The URI class has getter methods that return these three parts of each URI object. The getRawFoo() methods return the encoded forms of the parts of the URI, while the equivalent getFoo() methods first decode any percentescaped characters and then return the decoded part:

```
public String getScheme()
public String getSchemeSpecificPart()
public String getRawSchemeSpecificPart()
public String getFragment()
public String getRawFragment()
```

# 3.3 The URI Class

A URI that has a scheme is an *absolute URI. A URI without a scheme is relative. The* isAbsolute() method returns true if the URI is absolute, false if it's relative:

```
public boolean isAbsolute()
```

The isOpaque() method returns false if the URI is hierarchical, true if it's not hierarchical— that is, if it's opaque:

```
public boolean isOpaque()
```

If the URI is opaque, all you can get is the scheme, scheme-specific part, and fragment identifier. However, if the URI is hierarchical, there are getter methods for all the different parts of a hierarchical URI:

```
public String getAuthority()
public String getFragment()
public String getHost()
public String getPath()
public String getPort()
public String getQuery()
public String getUserInfo()
```

# 3.3 The URI Class

These methods all return the decoded parts; in other words, percent escapes, such as %3C, are changed into the characters they represent, such as <. If you want the raw, encoded parts of the URI, there are five parallel getRaw_Foo_() methods:

```
public String getRawAuthority()
public String getRawFragment()
public String getRawPath()
public String getRawQuery()
public String getRawUserInfo()
```

We can call parseServerAuthority() to force the authority to be reparsed:

```
public URI parseServerAuthority() throws URISyntaxException
```

*Example 5-6. The parts of a URI*

```
import java.net.*;
public class URISplitter {
public static void main(String args[]) {
for (int i = 0; i < args.length; i++) {
try {
URI u = new URI(args[i]);
System.out.println("The URI is " + u);
if (u.isOpaque()) {
System.out.println("This is an opaque URI.");
System.out.println("The scheme is " + u.getScheme());
System.out.println("The scheme specific part is "
+ u.getSchemeSpecificPart());
System.out.println("The fragment ID is " +
u.getFragment());
} else {
System.out.println("This is a hierarchical URI.");
System.out.println("The scheme is " + u.getScheme());
try {
u = u.parseServerAuthority();
System.out.println("The host is " + u.getHost());
System.out.println("The user info is " +
u.getUserInfo());
System.out.println("The port is " + u.getPort());
} catch (URISyntaxException ex) {
// Must be a registry based authority
System.out.println("The authority is " +
u.getAuthority());
}
System.out.println("The path is " + u.getPath());
System.out.println("The query string is " +
u.getQuery());
System.out.println("The fragment ID is " +
u.getFragment());
}
} catch (URISyntaxException ex) {
System.err.println(args[i] + " does not seem to be a
URI.");
}
System.out.println();
}
}
}
```

```
% java URISplitter tel:+1-800-9988-9938
http://www.xml.com/pub/a/2003/09/17/stax.html#id=_h
bc
urn:isbn:1-565-92870-9
The URI is tel:+1-800-9988-9938
This is an opaque URI.
The scheme is tel
The scheme specific part is +1-800-9988-9938
The fragment ID is null
```

# 3.3 The URI Class

**Resolving Relative URIs**

The URI class has three methods for converting back and forth between relative and
absolute URIs:

```
public URI resolve(URI uri)
public URI resolve(String uri)
public URI relativize(URI uri)
```

The resolve() methods compare the uri argument to this URI and use it to construct
a new URI object that wraps an absolute URI. For example, consider these three lines of
code:

```
URI absolute = new URI("http://www.example.com/");
URI relative = new URI("images/logo.png");
URI resolved = absolute.resolve(relative);
```

After they've executed, resolved contains the absolute URI
*http://www.example.com/images/logo.png.*

# 3.3 The URI Class

the resolve() method resolves as much of the URI as it can and returns a new relative URI object as a result.

For example, take these three statements:

```
URI top = new URI("javafaq/books/");
URI resolved = top.resolve("jnp3/examples/07/index.html");
```

After they've executed, resolved now contains the relative URI *avafaq/books/jnp3/examples/07/index.html with no scheme or authority.*

The relativize() method creates a new URI object from the uri argument that is relative to the invoking URI. The argument is not changed. For example:

```
URI absolute = new URI("http://www.example.com/images/logo.png");
URI top = new URI("http://www.example.com/");
URI relative = top.relativize(absolute);
```

The URI object relative now contains the relative URI *images/logo.png.*

# 3.3 The URI Class

## Equality and Comparison

The hashCode() method is consistent with equals. Equal URIs do have the same hash code and unequal URIs are fairly unlikely to share the same hash code.

URI implements Comparable, and thus URIs can be ordered. The ordering is based on string comparison of the individual parts, in this sequence:

1. If the schemes are different, the schemes are compared, without considering case.
2. Otherwise, if the schemes are the same, a hierarchical URI is considered to be less than an opaque URI with the same scheme.
3. If both URIs are opaque URIs, they're ordered according to their scheme-specific parts.
4. If both the scheme and the opaque scheme-specific parts are equal, the URIs are compared by their fragments.

# 3.3 The URI Class

5. If both URIs are hierarchical, they're ordered according to their authority components, which are themselves ordered according to user info, host, and port, in that order. Hosts are case insensitive.
6. If the schemes and the authorities are equal, the path is used to distinguish them.
7. If the paths are also equal, the query strings are compared.
8. If the query strings are equal, the fragments are compared.

Comparing a URI to anything except another URI causes a ClassCastException.

# 3.3 The URI Class

## String Representations

Two methods convert URI objects to strings, toString() and toASCIIString():

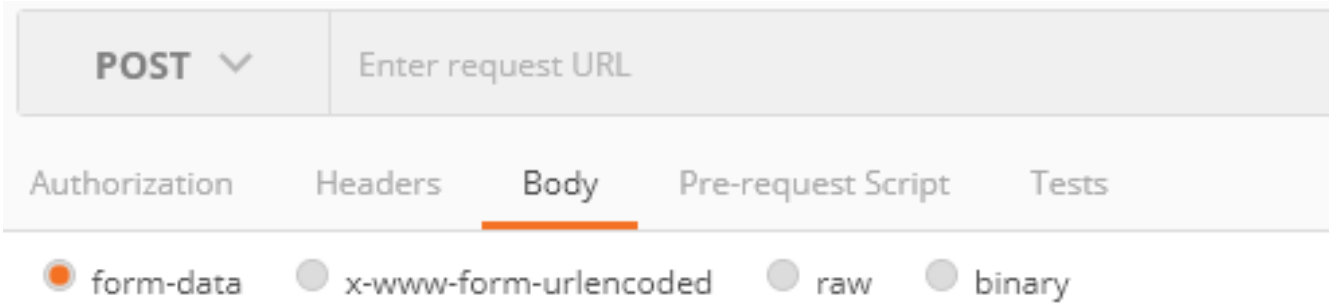
```
public String toString()
public String toASCIIString()
```

The toString() method returns an *unencoded string form of the URI (i.e., characters* like é and \ are not percent escaped).

The toASCIIString() method returns an *encoded string form of the URI. Characters* like é and \ are always percent escaped whether or not they were originally escaped.

# x-www-form-urlencoded

| value | description |
| --- | --- |
| application/x-www-form-urlencoded | Encode all characters before sending(default) |
| multipart/form-data | Do not encode characters.<br>This value must be used when using a form that contains a file upload control. |
| text/plain | Spaces are converted to a "+" plus sign, but no special characters are encoded. |



# x-www-form-urlencoded

1. URLEncoder
2. URLDecoder

# x-www-form-urlencoded

In the case of x-www-form-urlencoded, all name-value pairs are sent as one big query string where an alphanumeric and reserved character is url encoded i.e. replaced by % and their hex value e.g. space is replaced by %20. The length of this string is not specified by HTTP specification and depends upon server implementation.

To solve these problems, characters used in URLs must come from a fixed subset of ASCII, specifically:

- The capital letters A–Z
- The lowercase letters a–z
- The digits 0–9
- The punctuation characters - _ . ! ~ * ' (and ,)

The characters : / & ? @ # ; $ + = and % may also be used, but only for their specified purposes. If these characters occur as part of a path or query string, they and all other characters should be encoded.

# x-www-form-urlencoded

## URLEncoder

To URL encode a string, pass the string and the character set name to the URLEncoder.encode() method. For example:

```
String encoded = URLEncoder.encode("This*string*has*asterisks", "UTF-8");
```

URLEncoder.encode() returns a copy of the input string with a few changes. Any nonalphanumeric characters are converted into % sequences (except the space, underscore, hyphen, period, and asterisk characters). It also encodes all non-ASCII characters. The space is converted into a plus sign.

Although this method allows you to specify the character set, the only such character set you should ever pick is UTF-8.

*Example 5-7. x-www-form-urlencoded strings*

```
import java.io.*;
import java.net.*;
public class EncoderTest {
public static void main(String[] args) {
try {
System.out.println(URLEncoder.encode("This string has spaces",
"UTF-8"));
System.out.println(URLEncoder.encode("This*string*has*asterisks",
"UTF-8"));
System.out.println(URLEncoder.encode("This%string%has%percent
%signs",
"UTF-8"));
System.out.println(URLEncoder.encode("This+string+has+pluses",
"UTF-8"));
System.out.println(URLEncoder.encode("This/string/has/slashes",
"UTF-8"));
System.out.println(URLEncoder.encode("This\"string\"has\"quote\"mar
ks",
"UTF-8"));
System.out.println(URLEncoder.encode("This:string:has:colons",
"UTF-8"));
System.out.println(URLEncoder.encode("This~string~has~tildes",
"UTF-8"));
System.out.println(URLEncoder.encode("This(string)has(parentheses
)",
"UTF-8"));
System.out.println(URLEncoder.encode("This.string.has.periods",
"UTF-8"));
System.out.println(URLEncoder.encode("This=string=has=equals=sig
ns",
"UTF-8"));
System.out.println(URLEncoder.encode("This&string&has&ampersan
ds",
"UTF-8"));
System.out.println(URLEncoder.encode("Thiséstringéhasé
non-ASCII characters", "UTF-8"));
} catch (UnsupportedEncodingException ex) {
throw new RuntimeException("Broken VM does not support
UTF-8");
}
}
}
```

```
% javac -encoding UTF8 EncoderTest
% java EncoderTest
This+string+has+spaces
This*string*has*asterisks
This%25string%25has%25percent%25signs
This%2Bstring%2Bhas%2Bpluses
This%2Fstring%2Fhas%2Fslashes
This%22string%22has%22quote%22marks
This%3Astring%3Ahas%3Acolons
This%7Estring%7Ehas%7Etildes
This%28string%29has%28parentheses%29
This.string.has.periods
This%3Dstring%3Dhas%3Dequals%3Dsigns
This%26string%26has%26ampersands
This%C3%A9string%C3%A9has%C3%A9non-ASCII+characters
```

# x-www-form-urlencoded

For example, suppose you want to encode this URL for a Google search:

https://www.google.com/search?hl=en&as_q=Java&as_epq=I/O

This code fragment encodes it:

```
String query = URLEncoder.encode(
"https://www.google.com/search?hl=en&as_q=Java&as_epq=I/O", "UTF-8");

System.out.println(query);
```

Unfortunately, the output is:

```
https%3A%2F%2Fwww.google.com%2Fsearch%3Fhl%3Den%26as_q%3DJava
%26as_epq%3DI%2FO
```

# x-www-form-urlencoded

For example, suppose you want to encode this URL for a Google search:

https://www.google.com/search?hl=en&as_q=Java&as_epq=I/O

This code fragment encodes it:

```
String query = URLEncoder.encode(
"https://www.google.com/search?hl=en&as_q=Java&as_epq=I/O", "UTF-8");

System.out.println(query);
```

Unfortunately, the output is:

```
https%3A%2F%2Fwww.google.com%2Fsearch%3Fhl%3Den%26as_q%3DJava%26as_epq%3DI%2FO
```

The problem is that URLEncoder.encode() encodes blindly. It can't distinguish between special characters used as part of the URL or query string, like / and =, and characters that need to be encoded.

# x-www-form-urlencoded

Consequently, URLs need to be encoded a piece at a time like this:

```
String url = "https://www.google.com/search?";

url += URLEncoder.encode("hl", "UTF-8");
url += "=";
url += URLEncoder.encode("en", "UTF-8");
url += "&";
url += URLEncoder.encode("as_q", "UTF-8");
url += "=";
url += URLEncoder.encode("Java", "UTF-8");
url += "&";
url += URLEncoder.encode("as_epq", "UTF-8");
url += "=";
url += URLEncoder.encode("I/O", "UTF-8");
System.out.println(url);
```

The output of this is what you actually want:
https://www.google.com/search?hl=en&as_q=Java&as_epq=I/O

*Example 5-8. The QueryString class*

```
import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;
public class QueryString {
private StringBuilder query = new StringBuilder();
public QueryString() {
}
public synchronized void add(String name, String value) {
query.append('&');
encode(name, value);
}
private synchronized void encode(String name, String value) {
try {
query.append(URLEncoder.encode(name, "UTF-8"));
query.append('=');
query.append(URLEncoder.encode(value, "UTF-8"));
} catch (UnsupportedEncodingException ex) {
throw new RuntimeException("Broken VM does not support UTF-8");
}
}
public synchronized String getQuery() {
return query.toString();
}
@Override
public String toString() {
return getQuery();
}
}
```

Using this class, we can now encode the previous example:

```
QueryString qs = new QueryString();
qs.add("hl", "en");
qs.add("as_q", "Java");
qs.add("as_epq", "I/O");
String url = "http://www.google.com/search?" + qs;
System.out.println(url);
```

# x-www-form-urlencoded

## URLDecoder

The corresponding URLDecoder class has a static decode() method that decodes strings encoded in x-www-form-url-encoded format. That is, it converts all plus signs to spaces and all percent escapes to their corresponding character:

```
public static String decode(String s, String encoding)
throws UnsupportedEncodingException
```

We can pass an entire URL to it rather than splitting it into pieces first. For example:

```
String input = "https://www.google.com/" +
"search?hl=en&as_q=Java&as_epq=I%2FO";
String output = URLDecoder.decode(input, "UTF-8");
System.out.println(output);
```

# Proxies

**What is an agent ?**

The local client sends a request to the Proxy Server, and then the Proxy Server sends a request to the remote Server. The remote server sends the result to the proxy, and the proxy returns to the client.

**Why use a proxy?**

1. Security considerations:Use proxy to shield client side details from remote server
2. Access control: The proxy can have a filtering function to block sites that do not want the client to visit.
3.Performance considerations: When multiple clients request a unified resource, proxy can cache that resource instead of requesting the remote server every time.

# Proxies

Java programmers can use the URL class to do most of the communication with the proxy server or protocol.

**System Properties**

```
System.setProperty("http.proxyHost", "192.168.254.254");
System.setProperty("http.proxyPort", "9000");
System.setProperty("http.nonProxyHosts", "java.oreilly.com|xml.oreilly.com");
```

# Proxies

If you use http proxy, use http.proxyHost to set the proxy host and http.proxyPort to set the proxy port. Hosts that do not want to be proxied can use http.nonProxyHosts.

In addition to using System.setProperty (), you can also configure parameters when the program starts

```
java -D http.proxyHost=192.168.254.254  -D http.nonProxyHosts=*.oreilly.com
```

For ftp protocol, you can use ft p.proxyHost, ft p.proxyPort, ftp.nonProxyHosts.

Java only supports these 2 for application layer protocol proxy. If you want to do Socket proxy, socksProxyHost and socksProxyPort for all TCP connections in the transport layer, but there is no nonS ocksProxyHosts

# Proxies

**ProxyClass**

ProxyClass can be "fine-grained" (fine-grained) control. Choose different proxy servers for different remote servers. Different Proxy instances represent different proxies.

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

There are 3 agents:

- Proxy.Type.DIRECT
- Proxy.Type.HTTP
- Proxy.Type.SOCKS

DIRECT is directly accessed, no proxy is required.

# Proxies

**ProxySelector Class**

There is only one ProxySelector instance per JVM, used to locate different proxy servers for different connections. The default ps only detects some system properties (such as those mentioned above) and URL protocol and then decides how to connect to the remote server.

To implement 2 methods

public abstract List<Proxy> select(URI uri)

The system connects to the remote server according to the returned List <Proxy>.

public void connectFailed(URI uri, SocketAddress address, IOException ex)

*Example 5-9. A ProxySelector that remembers what it can connect to*

```
import java.io.*;
import java.net.*;
import java.util.*;
public class LocalProxySelector extends ProxySelector {
private List<URI> failed = new ArrayList<URI>();
public List<Proxy> select(URI uri) {
List<Proxy> result = new ArrayList<Proxy>();
if (failed.contains(uri)
|| !"http".equalsIgnoreCase(uri.getScheme())) {
result.add(Proxy.NO_PROXY);
} else {
SocketAddress proxyAddress
= new InetSocketAddress( "proxy.example.com", 8000);
Proxy proxy = new Proxy(Proxy.Type.HTTP, proxyAddress);
result.add(proxy);
}
return result;
}
public void connectFailed(URI uri, SocketAddress address, IOException ex) {
failed.add(uri);
}
}
```

To change the Proxy Selector, pass the new selector to the static ProxySelector.setDefault() method, like so:

ProxySelector selector = **new LocalProxySelector():**
ProxySelector.setDefault(selector)

# Communicating with Server-Side Programs Through GET

The URL class makes it easy for Java applets and applications to communicate with serverside programs such as CGIs, servlets, PHP pages, and others that use the GET method.

## The HTML for the form looks like this:

```
<form class="center mb1em" action="search" method="GET">
<input style="*vertical-align:middle;" size="45" name="q" value="" class="qN">
<input style="*vertical-align:middle; *padding-top:1px;" value="Search"
class="btn" type="submit">
<a href="search?type=advanced"><span class="advN">advanced</span></a>
</form>
```

There are only two input fields in this form: the Submit button and a text field named *q. Thus, to submit a search request to the Open Directory, you just need to append* q=*searchTerm to* [*http://www.dmoz.org/search*](http://www.dmoz.org/search)*.*

For example, to search for “java”, We would open a connection to the URL *http://www.dmoz.org/search/?q=java and read the* resulting input stream

*Example 5-10. Do an Open Directory search*

```
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();
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);
}
}
}
```

# Accessing Password-Protected Sites

Many popular sites require a username and password for access. Some sites, such as the W3C member pages, implement this through HTTP authentication.

Java's URL class can access sites that use HTTP authentication.

Implementing cookie authentication is hard short of implementing a complete web browser with full HTML forms and cookie support.

**The Authenticator Class:**

The java.net package includes an Authenticator class you can use to provide a username and password for sites that protect themselves using HTTP authentication:

**public abstract class Authenticator extends Object**

# The Authenticator Class:


*Figure 5-2. An authentication dialog*

To make the URL class use the subclass, install it as the default authenticator by passing it to the static Authenticator.setDefault() method:

```
public static void setDefault(Authenticator a)
```

For example, if we written an Authenticator subclass named DialogAuthenticator, we did install it like this:

```
Authenticator.setDefault(new DialogAuthenticator());
```

# The Authenticator Class:

It will ask the DialogAuthenticator using the static Authenticator.requestPasswordAuthentication() method:

```
public static PasswordAuthentication requestPasswordAuthentication(
InetAddress address, int port, String protocol, String prompt, String scheme) throws
SecurityException
```

We collect the username and password from the user or some other source and return it as an instance of the java.net.PasswordAuthentication class:

```
protected PasswordAuthentication getPasswordAuthentication()
```

# The Authenticator Class:

We can get more details about the request by invoking any of these methods inherited from the Authenticator superclass:

```
protected final InetAddress getRequestingSite()
protected final int getRequestingPort()
protected final String getRequestingProtocol()
protected final String getRequestingPrompt()
protected final String getRequestingScheme()
protected final String getRequestingHost()
protected final String getRequestingURL()
protected Authenticator.RequestorType getRequestorType()
```

# The PasswordAuthentication Class

PasswordAuthentication is a very simple final class that supports two read-only properties: username and password.

The username is a String. The password is a char array so that the password can be erased when it's no longer needed.

Both username and password are set in the constructor:

```
public PasswordAuthentication(String userName, char[] password)
```

Each is accessed via a getter method:

```
public String getUserName()
public char[] getPassword()
```

# The JPasswordField Class

One useful tool for asking users for their passwords in a more or less secure fashion is the JPasswordField component from Swing:

```
public class JPasswordField extends JTextField
```

This lightweight component behaves almost exactly like a text field.

JPasswordField also stores the passwords as a char array so that when we are done with the password. It provides the getPassword() method to return this:

```
public char[] getPassword()
```

*Example 5-12. A program to download password-protected web pages*

*Example 5-11. A GUI authenticator*