Table of Contents

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

Chapter 7. URLs and URIs

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, the format of the data being retrieved, or how to communicate with the server; you simply tell Java the URL and it gets the data for you. Although Java can only handle a few protocols and content types out of the box, in later chapters you'll learn how to write and install new content and protocol handlers that extend Java's capabilities to include new protocols and new kinds of data. You'll also learn how to open sockets and communicate directly with different kinds of servers. But that's later; for now, let's see how much can be done with a minimum of work.

7.1. The URL Class

The `java.net.URL` class is an abstraction of a Uniform Resource Locator such as http://www.hamsterdance.com/ or ftp://ftp.redhat.com/pub/. It extends `java.lang.Object`, and it is a final class that cannot be subclassed. Rather than relying on inheritance to configure instances for different kinds of URLs, it uses the strategy design pattern. Protocol handlers are the strategies, and the `URL` class itself forms the context through which the different strategies are selected:

```
public final class URL extends Object implements Serializable
```

Although storing a URL as a string would be trivial, it is helpful to think of URLs as objects with fields that include the scheme (a.k.a. the protocol), hostname, port, path, query string, and fragment identifier (a.k.a. the ref), each of which may be set independently. Indeed, this is almost exactly how the `java.net.URL` class is organized, though the details vary a little between different versions of Java.

The fields of `java.net.URL` are only visible to other members of the `java.net` package; classes that aren't in `java.net` can't access a `URL`'s fields directly. However, you can set

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

these fields using the `URL` constructors and retrieve their values using the various getter methods (`getHost( )`, `getPort()`, and so on). URLs are effectively immutable. After a `URL` object has been constructed, its fields do not change. This has the side effect of making them thread-safe.

7.1.1. Creating New URLs

Unlike the `InetAddress` objects in Chapter 6, you can construct instances of `java.net.URL`. There are six constructors, differing in the information they require. Which constructor you use depends on the information you have and the form it's in. All these constructors throw a `MalformedURLException` if you try to create a `URL` for an unsupported protocol and may throw a `MalformedURLException` if the URL is syntactically incorrect.

Exactly which protocols are supported is implementation-dependent. The only protocols that have been available in all major virtual machines are http and file, and the latter is notoriously flaky. Java 1.5 also requires virtual machines to support https, jar, and ftp; many virtual machines prior to Java 1.5 support these three as well. Most virtual machines also support ftp, mailto, and gopher as well as some custom protocols like doc, netdoc, systemresource, and verbatim used internally by Java. The Netscape virtual machine supports the http, file, ftp, mailto, telnet, ldap, and gopher protocols. The Microsoft virtual machine supports http, file, ftp, https, mailto, gopher, doc, and systemresource, but not telnet, netdoc, jar, or verbatim. Of course, support for all these protocols is limited in applets by the security policy. For example, just because an untrusted applet can construct a `URL` object from a file URL does not mean that the applet can actually read the file the URL refers to. Just because an untrusted applet can construct a `URL` object from an HTTP URL that points to a third-party web site does not mean that the applet can connect to that site.

If the protocol you need isn't supported by a particular VM, you may be able to install a protocol handler for that scheme. This is subject to a number of security checks in applets and is really practical only for applications. Other than verifying that it recognizes the URL scheme, Java does not make any checks about the correctness of the URLs it constructs. The programmer is responsible for making sure that URLs created are valid. For instance, Java does not check that the hostname in an HTTP URL does not contain spaces or that the query string is x-www-form-URL-encoded. It does not check that a mailto URL actually contains an email address. Java does not check the URL to make sure that it points at an existing host or that it meets any other requirements for URLs. You can create URLs for hosts that don't exist and for hosts that do exist but that you won't be allowed to connect to.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

7.1.1.1. 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
```

Like all constructors, this may only be called after the `new` operator, and like all `URL` constructors, it can throw a `MalformedURLException`. The following code constructs a `URL` object from a `String`, catching the exception that might be thrown:

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

Example 7-1 is a simple program for determining which protocols a virtual machine supports. It attempts to construct a `URL` object for each of 14 protocols (8 standard protocols, 3 custom protocols for various Java APIs, and 4 undocumented protocols used internally by HotJava). If the constructor succeeds, you know the protocol is supported. Otherwise, a `MalformedURLException` is thrown and you know the protocol is not supported.

Example 7-1. ProtocolTester

```
/* 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://metalab.unc.edu/pub/languages/java/javafaq/");

    // Simple Mail Transfer Protocol
    testProtocol("mailto:elharo@metalab.unc.edu");

    // telnet
    testProtocol("telnet://dibner.poly.edu/");
```


Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

```
      // 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.metalab.unc.edu:3306/NEWS");

      // rmi, a custom protocol for remote method invocation
      testProtocol("rmi://metalab.unc.edu/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 1.4.1 on Mac OS X 10.2, which turns out to support all the protocols except Telnet, LDAP, RMI, NFS, and JDBC:

```
% java ProtocolTester
http is supported
https is supported
ftp is supported
mailto is supported
```


Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

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

Results using Sun's Linux 1.4.2 virtual machine were identical. Other 1.4 virtual machines derived from the Sun code will show similar results. Java 1.2 and later are likely to be the same except for maybe HTTPS, which was only recently added to the standard distribution. VMs that are not derived from the Sun codebase may vary somewhat in which protocols they support. For example, here are the results of running `ProtocolTester` with the open source Kaffe VM 1.1.1:

```
% java ProtocolTester
http is supported
https is not supported
ftp is supported
mailto is not 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 not supported
systemresource is not supported
verbatim is not supported
```

The nonsupport of RMI and JDBC is actually a little deceptive; in fact, the JDK does support these protocols. However, that support is through various parts of the `java.rmi` and `java.sql` packages, respectively. These protocols are not accessible through the `URL` class like the other supported protocols (although I have no idea why Sun chose to wrap up RMI and JDBC parameters in URL clothing if it wasn't intending to interface with these via Java's quite sophisticated mechanism for handling URLs).

7.1.1.2. Constructing a URL from its component parts

The second constructor builds a `URL` from three strings specifying the protocol, the hostname, and the file:

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

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

This constructor sets the port to -1 so the default port for the protocol will be used. The `file` argument should begin with a slash and include a path, a filename, and optionally a fragment identifier. Forgetting the initial slash is a common mistake, and one that is not easy to spot. Like all `URL` constructors, it can throw a `MalformedURLException`. For example:

```
try {
  URL u = new URL("http", "www.eff.org", "/blueribbon.html#intro");
}
catch (MalformedURLException ex)  {
  // All VMs should recognize http
}
```

This creates a `URL` object that points to http://www.eff.org/blueribbon.html#intro, using the default port for the HTTP protocol (port 80). The file specification includes a reference to a named anchor. The code catches the exception that would be thrown if the virtual machine did not support the HTTP protocol. However, this shouldn't happen in practice.

For the rare occasions when the default port isn't correct, the next constructor lets you specify the port explicitly as an `int`:

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

The other arguments are the same as for the `URL(String protocol, String host, String file)` constructor and carry the same caveats. For example:

```
try {
  URL u = new URL("http", "fourier.dur.ac.uk", 8000, "/~dma3mjh/jsci/");
}
catch (MalformedURLException ex)  {
  System.err.println(ex);
}
```

This code creates a `URL` object that points to http://fourier.dur.ac.uk:8000/~dma3mjh/jsci/, specifying port 8000 explicitly.

Example 7-2 is an alternative protocol tester that can run as an applet, making it useful for testing support of browser virtual machines. It uses the three-argument constructor rather than the one-argument constructor in Example 7-1. It also stores the schemes to be tested in an array and uses the same host and file for each scheme. This produces seriously malformed URLs like *mailto://www.peacefire.org/bypass/SurfWatch/*, once again demonstrating that all Java checks for at object construction is whether it recognizes the scheme, not whether the URL is appropriate.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

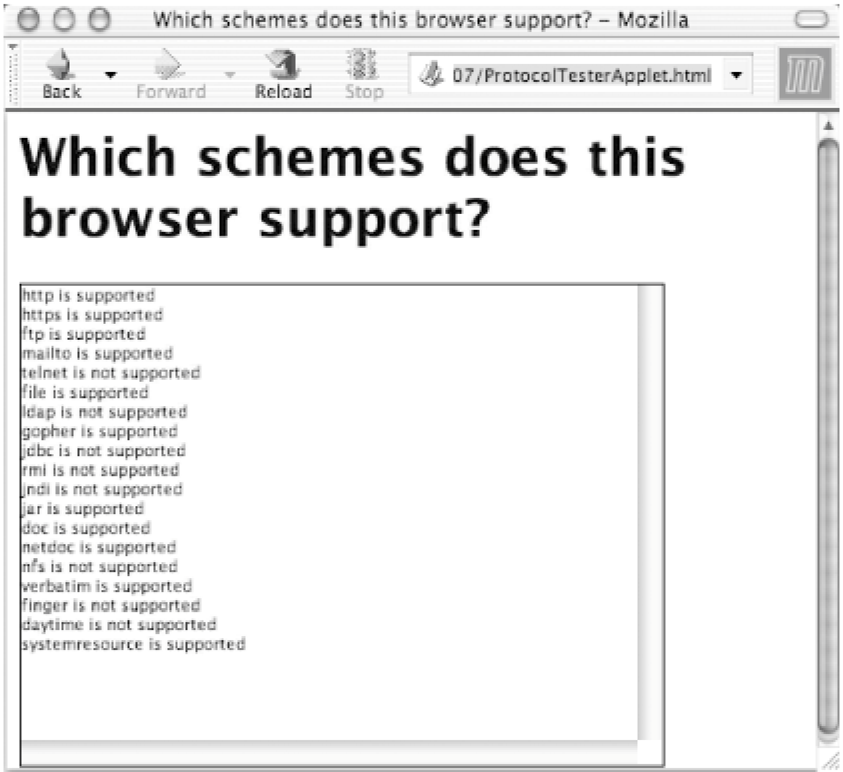
Example 7-2. A protocol tester applet

```
import java.net.*;
import java.applet.*;
import java.awt.*;

public class ProtocolTesterApplet extends Applet {

  TextArea results = new TextArea( );

  public void init( ) {
    this.setLayout(new BorderLayout( ));
    this.add("Center", results);
  }

  public void start( ) {

    String host = "www.peacefire.org";
    String file = "/bypass/SurfWatch/";

    String[] schemes = {"http",   "https",   "ftp",  "mailto",
                        "telnet", "file",    "ldap", "gopher",
                        "jdbc",   "rmi",     "jndi", "jar",
                        "doc",    "netdoc",  "nfs",  "verbatim",
                        "finger", "daytime", "systemresource"};

    for (int i = 0; i < schemes.length; i++) {
      try {
        URL u = new URL(schemes[i], host, file);
        results.append(schemes[i] + " is supported\r\n");
      }
      catch (MalformedURLException ex) {
        results.append(schemes[i] + " is not supported\r\n");
      }
    }

  }

}
```

Figure 7-1 shows the results of Example 7-2 in Mozilla 1.4 with Java 1.4 installed. This browser supports HTTP, HTTPS, FTP, mailto, file, gopher, doc, netdoc, verbatim, systemresource, and jar but not HTTPS, ldap, Telnet, jdbc, rmi, jndi, finger or daytime.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

Figure 7-1. The ProtocolTesterApplet running in Mozilla 1.4



7.1.1.3. Constructing relative URLs

This constructor builds an absolute `URL` from a relative `URL` and a base `URL`:

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

For instance, you may be parsing an HTML document at http://www.ibiblio.org/javafaq/index.html and encounter a link to a file called *mailinglists.html* with no further qualifying information. In this case, you use the URL to the document that contains the link to provide the missing information. 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 filename is removed from the path of `u1` and the new filename *mailinglists.html* is appended to make `u2`. This constructor is particularly useful when you want to loop through

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

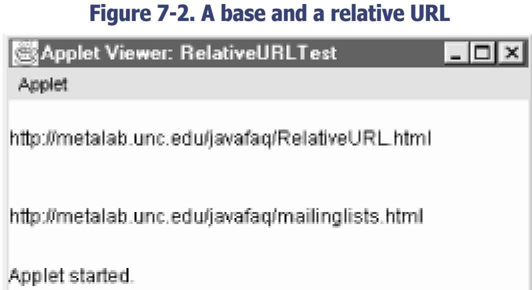
a list of files that are all in the same directory. You can create a URL for the first file and then use this initial URL to create `URL` objects for the other files by substituting their filenames. You also use this constructor when you want to create a `URL` relative to the applet's document base or code base, which you retrieve using the `getDocumentBase()` or `getCodeBase()` methods of the `java.applet.Applet` class. Example 7-3 is a very simple applet that uses `getDocumentBase( )` to create a new `URL` object:

Example 7-3. A URL relative to the web page

```
import java.net.*;
import java.applet.*;
import java.awt.*;

public class RelativeURLTest extends Applet {

  public void init ( ) {

    try {
      URL base = this.getDocumentBase( );
      URL relative = new URL(base, "mailinglists.html");
      this.setLayout(new GridLayout(2,1));
      this.add(new Label(base.toString( )));
      this.add(new Label(relative.toString( )));
    }
    catch (MalformedURLException ex) {
      this.add(new Label("This shouldn't happen!"));
    }

  }

}
```

Of course, the output from this applet depends on the document base. In the run shown in Figure 7-2, the original `URL` (the document base) refers to the file *RelativeURL.html*; the constructor creates a new `URL` that points to the *mailinglists.html* file in the same directory.

Figure 7-2. A base and a relative URL



Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

When using this constructor with getDocumentBase(), you frequently put the call to getDocumentBase( ) inside the constructor, like this:

```
URL relative = new URL(this.getDocumentBase( ), "mailinglists.html");
```

7.1.1.4. Specifying a URLStreamHandler // Java 1.2

Two constructors allow you to specify the protocol handler used for the URL. The first constructor builds a relative URL from a base URL and a relative part. The second builds the URL from its component pieces:

```
public URL(URL base, String relative, URLStreamHandler handler) // 1.2
 throws MalformedURLException
public URL(String protocol, String host, int port, String file, // 1.2
 URLStreamHandler handler) throws MalformedURLException
```

All URL objects have URLStreamHandler objects to do their work for them. These two constructors change from the default URLStreamHandler subclass for a particular protocol to one of your own choosing. This is useful for working with URLs whose schemes aren't supported in a particular virtual machine as well as for adding functionality that the default stream handler doesn't provide, such as asking the user for a username and password. For example:

```
URL u = new URL("finger", "utopia.poly.edu", 79, "/marcus",
 new com.macfaq.net.www.protocol.finger.Handler( ));
```

The com.macfaq.net.www.protocol.finger.Handler class used here will be developed in Chapter 16.

While the other four constructors raise no security issues in and of themselves, these two do because class loader security is closely tied to the various URLStreamHandler classes. Consequently, untrusted applets are not allowed to specify a URLSreamHandler. Trusted applets can do so if they have the NetPermission specifyStreamHandler. However, for reasons that will become apparent in Chapter 16, this is a security hole big enough to drive the Microsoft money train through. Consequently, you should not request this permission or expect it to be granted if you do request it.

7.1.1.5. Other sources of URL objects

Besides the constructors discussed here, a number of other methods in the Java class library return URL objects. You've already seen getDocumentBase( ) from

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

`java.applet.Applet`. The other common source is `getCodeBase( )`, also from `java.applet.Applet`. This works just like `getDocumentBase( )`, except it returns the `URL` of the applet itself instead of the URL of the page that contains the applet. Both `getDocumentBase( )` and `getCodeBase( )` come from the `java.applet.AppletStub` interface, which `java.applet.Applet` implements. You're unlikely to implement this interface yourself unless you're building a web browser or applet viewer.

In Java 1.2 and later, the `java.io.File` class has a `toURL( )` method that returns a file URL matching the given file. The exact format of the URL returned by this method is platform-dependent. For example, on Windows it may return something like *file:/D:/JAVA/JNP3/07/ToURLTest.java*. On Linux and other Unixes, you're likely to see *file:/home/elharo/books/JNP3/07/ToURLTest.java*. In practice, *file* URLs are heavily platform- and program-dependent. Java file URLs often cannot be interchanged with the URLs used by web browsers and other programs, or even with Java programs running on different platforms.

Class loaders are used not only to load classes but also to load resources such as images and audio files. The static `ClassLoader.getSystemResource(String name)` method returns a `URL` from which a single resource can be read. The `ClassLoader.getSystemResources(String name)` method returns an `Enumeration` containing a list of `URL`s from which the named resource can be read. Finally, the instance method `getResource(String name)` searches the path used by the referenced class loader for a URL to the named resource. The URLs returned by these methods may be file URLs, HTTP URLs, or some other scheme. The name of the resource is a slash-separated list of Java identifiers, such as */com/macfaq/sounds/swale.au* or *com/macfaq/images/headshot.jpg*. The Java virtual machine will attempt to find the requested resource in the class path—potentially including parts of the class path on the web server that an applet was loaded from—or inside a JAR archive.

Java 1.4 adds the `URI` class, which we'll discuss soon. URIs can be converted into URLs using the `toURL( )` method, provided Java has the relevant protocol handler installed.

There are a few other methods that return `URL` objects here and there throughout the class library, but most are simple getter methods that return only a URL you probably already know because you used it to construct the object in the first place; for instance, the `getPage( )` method of `java.swing.JEditorPane` and the `getURL( )` method of `java.net.URLConnection`.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

7.1.2. 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

For example, given the URL http://www.ibiblio.org/javafaq/books/jnp/index.html?isbn=1565922069#toc, the scheme is *http*, the authority is *www.ibiblio.org*, the path is */javafaq/books/jnp/index.html*, the fragment identifier is *toc*, and the query string is *isbn=1565922069*. However, not all URLs have all these pieces. For instance, the URL http://www.faqs.org/rfcs/rfc2396.html has a scheme, an authority, and a path, but no fragment identifier or query string.

The authority may further be divided into the user info, the host, and the port. For example, in the URL http://admin@www.blackstar.com:8080/, the authority is admin@www.blackstar.com:8080. This has the user info *admin*, the host *www.blackstar.com*, and the port *8080*.

Read-only access to these parts of a URL is provided by five public methods: `getFile( )`, `getHost()`, `getPort( )`, `getProtocol( )`, and `getRef( )`. Java 1.3 adds four more methods: `getQuery( )`, `getPath( )`, `getUserInfo( )`, and `getAuthority( )`.

7.1.2.1. public String getProtocol( )

The `getProtocol( )` method returns a `String` containing the scheme of the URL, e.g., "http", "https", or "file". For example:

```
URL page = this.getCodeBase( );
System.out.println("This applet was downloaded via "
 + page.getProtocol( ));
```

7.1.2.2. public String getHost( )

The `getHost( )` method returns a `String` containing the hostname of the URL. For example:

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

```
URL page = this.getCodeBase( );
System.out.println("This applet was downloaded from " + page.getHost( ));
```

The most recent virtual machines get this method right but some older ones, including Sun's JDK 1.3.0, may return a host string that is not necessarily a valid hostname or address. In particular, URLs that incorporate usernames, like ftp://anonymous:anonymous@wuarchive.wustl.edu/, sometimes include the user info in the host. For example, consider this code fragment:

```
URL u = new URL("ftp://anonymous:anonymous@wuarchive.wustl.edu/");
String host = u.getHost( );
```

Java 1.3 sets `host` to `anonymous:anonymous@wuarchive.wustl.edu`, not simply `wuarchive.wustl.edu`. Java 1.4 would return `wuarchive.wustl.edu` instead.

7.1.2.3. public int getPort( )

The `getPort( )` method returns the port number specified in the URL as an `int`. If no port was specified in the `URL`, `getPort( )` returns -1 to signify that the URL does not specify the port explicitly, and will use the default port for the protocol. 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.uiuc.edu/demoweb/html-primer.html");
System.out.println("The port part of " + u + " is " + u.getPort( ));
```

7.1.2.4. public int getDefaultPort( )

The `getDefaultPort( )` method returns the default port used for this `URL`'s protocol when none is specified in the URL. If no default port is defined for the protocol, `getDefaultPort( )` returns -1. For example, if the URL is http://www.userfriendly.org/, `getDefaultPort( )` returns 80; if the URL is ftp://ftp.userfriendly.org:8000/, `getDefaultPort( )` returns 21.

7.1.2.5. public String getFile( )

The `getFile( )` method returns a `String` that contains the path portion of a URL; remember that Java does not break a URL into separate path and file parts. Everything from the first slash (/) after the hostname until the character preceding the # sign that begins a fragment identifier is considered to be part of the file. For example:

```
URL page = this.getDocumentBase( );
System.out.println("This page's path is " + page.getFile( ));
```

If the URL does not have a file part, Java 1.2 and earlier append a slash to the URL and return the slash as the filename. For example, if the URL is http://www.slashdot.org (rather than something like http://www.slashdot.org/, `getFile()` returns `/`. Java 1.3 and later simply set the file to the empty string.

7.1.2.6. public String getPath( ) // Java 1.3

The `getPath( )` method, available only in Java 1.3 and later, is a near synonym for `getFile ( )`; that is, it returns a `String` containing the path and file portion of a URL. However, unlike `getFile( )`, it does not include the query string in the `String` it returns, just the path.

> Note that the `getPath( )` method does not return only the directory path and `getFile( )` does not return only the filename, as you might expect. Both `getPath()` and `getFile( )` return the full path and filename. The only difference is that `getFile()` also returns the query string and `getPath( )` does not.

7.1.2.7. public String getRef( )

The `getRef( )` method returns the fragment identifier part of the URL. If the URL doesn't have a fragment identifier, the method returns `null`. In the following code, `getRef( )` returns the string `xtocid1902914`:

```
URL u = new URL(
 "http://www.ibiblio.org/javafaq/javafaq.html#xtocid1902914");
System.out.println("The fragment ID of " + u + " is " + u.getRef( ));
```


Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

7.1.2.8. public String getQuery( ) // Java 1.3

The `getQuery( )` method returns the query string of the URL. If the URL doesn't have a query string, the method returns `null`. In the following code, `getQuery()` returns the string `category=Piano`:

```
URL u = new URL(
 "http://www.ibiblio.org/nywc/compositions.phtml?category=Piano");
System.out.println("The query string of " + u + " is " + u.getQuery( ));
```

In Java 1.2 and earlier, you need to extract the query string from the value returned by `getFile( )` instead.

7.1.2.9. public String getUserInfo( ) // Java 1.3

Some URLs include usernames and occasionally even password information. This information comes after the scheme and before the host; an @ symbol delimits it. For instance, in the URL http://elharo@java.oreilly.com/, the user info is *elharo*. Some URLs also include passwords in the user info. For instance, in the URL *ftp://mp3:secret@ftp.example.com/c%3a/stuff/mp3/*, the user info is *mp3:secret*. However, most of the time including a password in a URL is a security risk. If the URL doesn't have any user info, `getUserInfo()` returns `null`. Mailto URLs may not behave like you expect. In a URL like mailto:elharo@metalab.unc.edu, elharo@metalab.unc.edu is the path, not the user info and the host. That's because the URL specifies the remote recipient of the message rather than the username and host that's sending the message.

7.1.2.10. public String getAuthority( ) // Java 1.3

Between the scheme and the path of a URL, you'll find the authority. The term *authority* is taken from the Uniform Resource Identifier specification (RFC 2396), where this part of the URI indicates the authority that resolves the resource. In the most general case, the authority includes the user info, the host, and the port. For example, in the URL ftp://mp3:mp3@138.247.121.61:21000/c%3a/, the authority is *mp3:mp3@138.247.121.61:21000*. However, not all URLs have all parts. For instance, in the URL http://conferences.oreilly.com/java/speakers/, the authority is simply the hostname *conferences.oreilly.com*. The `getAuthority( )` method returns the authority as it exists in the URL, with or without the user info and port.

Example 7-4 uses all eight methods to split URLs entered on the command line into their component parts. This program requires Java 1.3 or later.

Example 7-4. 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( ));
      }  // end try
      catch (MalformedURLException ex) {
        System.err.println(args[i] + " is not a URL I understand.");
      }
      System.out.println( );
    }  // end for

  }  // end main

}  // end URLSplitter
```

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

```
% java URLSplitter    \
             http://www.ncsa.uiuc.edu/demoweb/html-primer.html#A1.3.3.3 \
             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 http://www.ncsa.uiuc.edu/demoweb/html-primer.html#A1.3.3.3
The scheme is http
The user info is null
The host is www.ncsa.uiuc.edu
```


Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

```
The port is -1
The path is /demoweb/html-primer.html
The ref is A1.3.3.3
The query string is null

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

The URL is http://www.ibiblio.org/nywc/compositions.phtml?category=Piano
The scheme is http
The user info is null
The host is www.ibiblio.org
The port is -1
The path is /nywc/compositions.phtml
The ref is null
The query string is category=Piano

The URL is http://admin@www.blackstar.com:8080/
The scheme is http
The user info is admin
The host is www.blackstar.com
The port is 8080
The path is /
The ref is null
The query string is null
```

7.1.3. Retrieving Data from a URL

Naked URLs aren't very exciting. What's interesting is the data contained in the documents they point to. 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 // 1.5
public Object getContent( ) throws IOException
public Object getContent(Class[] classes)  throws IOException // 1.3
```

These methods differ in that they return the data at the URL as an instance of different classes.

7.1.3.1. public final InputStream openStream( ) throws IOException

The `openStream( )` method connects to the resource referenced by the `URL`, performs any necessary handshaking between the client and the server, and returns an `InputStream` from which data can be read. The data you get from this `InputStream` is the raw (i.e., uninterpreted) contents of the file the `URL` references: ASCII if you're reading an ASCII text file, raw HTML if you're reading an HTML file, binary image data if you're reading an image file, and so forth. It does not include any of the HTTP headers or any other protocol-related information. You can read from this `InputStream` as you would read from any other `InputStream`. For example:

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

This code fragment catches an `IOException`, which also catches the `MalformedURLException` that the `URL` constructor can throw, since `MalformedURLException` subclasses `IOException`.

Example 7-5 reads a URL from the command line, opens an `InputStream` from that URL, chains the resulting `InputStream` to an `InputStreamReader` using the default encoding, and then uses `InputStreamReader`'s `read( )` method to read successive characters from the file, each of which is printed on `System.out`. That is, it prints the raw data located at the URL: if the URL references an HTML file, the program's output is raw HTML.

Example 7-5. Download a web page

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

public class SourceViewer {

  public static void main (String[] args) {

    if  (args.length > 0) {
      try {
        //Open the URL for reading
        URL u = new URL(args[0]);
```


Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

```
          InputStream 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);
        }

      } //  end if

    } // end main

  }  // end SourceViewer
```

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

```
% java SourceViewer http://www.oreilly.com
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en-US" xml:lang="en-US">
<head>
<title>oreilly.com -- Welcome to O'Reilly Media, Inc. -- computer books,
software conferences, online publishing</title>
<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,
bioinformatics, web services, p2p" />
<meta name="description" content="O'Reilly is a leader in technical and computer book
documentation, online content, and conferences for UNIX, Perl, Java, Linux, Internet,
Mac OS X, C, C++, Windows, Windows NT, Security, Sys Admin, System Administration,
Oracle, Design and Graphics, Online Books, e-books, ebooks, Perl Conference, Java
Conference, P2P Conference" />
```

There are quite a few more lines in that web page; if you want to see them, you can fire up your web browser.

The shakiest part of this program is that it blithely assumes that the remote URL is text, which is not necessarily true. It could well be a GIF or JPEG image, an MP3 sound file, or something else entirely. Even if it is text, the document encoding may not be the same as the default encoding of the client system. The remote host and local client may not have the same default character set. As a general rule, for pages that use a character set radically different from

ASCII, the HTML will include a `META` tag in the header specifying the character set in use. For instance, this `META` tag specifies the Big-5 encoding for Chinese:

```
<meta http-equiv="Content-Type" content="text/html; charset=big5">
```

An XML document will likely have an XML declaration instead:

```
<?xml version="1.0" encoding="Big5"?>
```

In practice, there's no easy way to get at this information other than by parsing the file and looking for a header like this one, and even that approach is limited. Many HTML files hand-coded in Latin alphabets don't have such a `META` tag. Since Windows, the Mac, and most Unixes have somewhat different interpretations of the characters from 128 to 255, the extended characters in these documents do not translate correctly on platforms other than the one on which they were created.

And as if this isn't confusing enough, the HTTP header that precedes the actual document is likely to have its own encoding information, which may completely contradict what the document itself says. You can't read this header using the `URL` class, but you can with the `URLConnection` object returned by the `openConnection( )` method. Encoding detection and declaration is one of the thornier parts of the architecture of the Web.

7.1.3.2. public URLConnection openConnection( ) throws IOException

The `openConnection( )` method opens a socket to the specified URL and returns a `URLConnection` object. A `URLConnection` represents an open connection to a network resource. If the call fails, `openConnection( )` throws an `IOException`. For example:

```
try {
  URL u = new URL("http://www.jennicam.org/");
  try {
    URLConnection uc = u.openConnection( );
    InputStream in = uc.getInputStream( );
    // read from the connection...
  } // end try
  catch (IOException ex) {
    System.err.println(ex);
  }
} // end try
catch (MalformedURLException ex) {
  System.err.println(ex);
}
```

Use this method when you want to communicate directly with the server. The `URLConnection` gives you access to everything sent by the server: in addition to the

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

document itself in its raw form (e.g., HTML, plain text, binary image data), you can access all the metadata specified by the protocol. For example, if the scheme is HTTP, the `URLConnection` lets you access the HTTP headers as well as the raw HTML. The `URLConnection` class also lets you write data to as well as read from a URL—for instance, in order to send email to a mailto URL or post form data. The `URLConnection` class will be the primary subject of Chapter 15.

Java 1.5 adds one overloaded variant of this method that specifies the proxy server to pass the connection through:

```
public URLConnection openConnection(Proxy proxy) throws IOException
```

This overrides any proxy server set with the usual `socksProxyHost`, `socksProxyPort`, `http.proxyHost`, `http.proxyPort`, `http.nonProxyHosts`, and similar system properties. If the protocol handler does not support proxies, the argument is ignored and the connection is made directly if possible.

7.1.3.3. public final Object getContent( ) throws IOException

The `getContent( )` method is the third way to download data referenced by a URL. The `getContent( )` method retrieves the data referenced by the URL and tries to make it into some type of object. If the URL refers to some kind of text object such as an ASCII or HTML file, the object returned is usually some sort of `InputStream`. If the URL refers to an image such as a GIF or a JPEG file, `getContent( )` usually returns a `java.awt.ImageProducer` (more specifically, an instance of a class that implements the `ImageProducer` interface). What unifies these two disparate classes is that they are not the thing itself but a means by which a program can construct the thing:

```
try {
  URL u = new URL("http://mesola.obspm.fr/");
  Object o = u.getContent( );
  // cast the Object to the appropriate type
  // work with the Object...
}
catch (Exception ex) {
  System.err.println(ex);
}
```

`getContent( )` operates by looking at the `Content-type` field in the MIME header of the data it gets from the server. If the server does not use MIME headers or sends an unfamiliar `Content-type`, `getContent( )` returns some sort of `InputStream` with which the data can be read. An `IOException` is thrown if the object can't be retrieved. Example 7-6 demonstrates this.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

Example 7-6. Download an object

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

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]);
        try {
          Object o = u.getContent( );
          System.out.println("I got a " + o.getClass( ).getName( ));
        } // end try
        catch (IOException ex) {
          System.err.println(ex);
        }
      } // end try
      catch (MalformedURLException ex) {
        System.err.println(args[0] + " is not a parseable URL");
      }
    } //  end if

  } // end main

}  // end ContentGetter
```

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

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

The exact class may vary from one version of Java to the next (in earlier versions, it's been `java.io.PushbackInputStream` or `sun.net.www.http.KeepAliveStream`) but it should be some form of `InputStream`.

Here's what you get when you try to load a header image from that page:

```
% java ContentGetter http://www.oreilly.com/graphics_new/animation.gif
I got a sun.awt.image.URLImageSource
```

Here's what happens when you try to load a Java applet using `getContent( )`:

```
% java ContentGetter http://www.cafeaulait.org/RelativeURLTest.class
I got a sun.net.www.protocol.http.HttpURLConnection$HttpInputStream
```


Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

Here's what happens when you try to load an audio file using `getContent( )`:

```
% java ContentGetter http://www.cafeaulait.org/course/week9/spacemusic.au
I got a sun.applet.AppletAudioClip
```

The last result is the most unusual because it is as close as the Java core API gets to a class that represents a sound file. It's not just an interface through which you can load the sound data.

This example demonstrates the biggest problems with using `getContent( )`: it's hard to predict what kind of object you'll get. You could get some kind of `InputStream` or an `ImageProducer` or perhaps an `AudioClip`; it's easy to check using the `instanceof` operator. This information should be enough to let you read a text file or display an image.

7.1.3.4. public final Object getContent(Class[] classes) throws IOException // Java 1.3

Starting in Java 1.3, it is possible for a content handler to provide different views of an object. This overloaded variant of the `getContent( )` method lets you choose what class you'd like the content to be returned as. The method attempts to return the URL's content in the order used in the array. For instance, if you prefer an HTML file to be returned as a `String`, but your second choice is a `Reader` and your third choice is an `InputStream`, write:

```
URL u = new URL("http://www.nwu.org");
Class[] types = new Class[3];
types[0] = String.class;
types[1] = Reader.class;
types[2] = InputStream.class;
Object o = u.getContent(types);
```

You then have to test for the type of the returned object using `instanceof`. For example:

```
if (o instanceof String) {
  System.out.println(o);
}
else if (o instanceof Reader) {
  int c;
  Reader r = (Reader) o;
  while ((c = r.read( )) != -1) System.out.print((char) c);
}
else if (o instanceof InputStream) {
  int c;
  InputStream in = (InputStream) o;
  while ((c = in.read( )) != -1) System.out.write(c);
}
else {
```


Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

```
    System.out.println("Error: unexpected type " + o.getClass( ));
  }
```

7.1.4. Utility Methods

The URL class contains a couple of utility methods that perform common operations on URLs. The sameFile( ) method determines whether two URLs point to the same document. 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.

7.1.4.1. public boolean sameFile(URL other)

The sameFile( ) method tests whether two URL objects point to the same file. If they do, sameFile( ) returns true; otherwise, it returns false. The test that sameFile( ) performs is quite shallow; all it does is compare the corresponding fields for equality. It detects whether the two hostnames are really just aliases for each other. For instance, it can tell that http://www.ibiblio.org/ and http://metalab.unc.edu/ are the same file. However, it cannot tell that http://www.ibiblio.org:80/ and http://metalab.unc.edu/ are the same file or that http://www.cafeconleche.org/ and http://www.cafeconleche.org/index.html are the same file. sameFile( ) is smart enough to ignore the fragment identifier part of a URL, however. Here's a fragment of code that uses sameFile( ) to compare two URLs:

```
try {
  URL u1 = new URL("http://www.ncsa.uiuc.edu/HTMLPrimer.html#GS");
  URL u2 = new URL("http://www.ncsa.uiuc.edu/HTMLPrimer.html#HD");
  if (u1.sameFile(u2)) {
    System.out.println(u1 + " is the same file as \n" + u2);
  }
  else {
    System.out.println(u1 + " is not the same file as \n" + u2);
  }
}
catch (MalformedURLException ex) {
  System.err.println(ex);
}
```

The output is:

```
http://www.ncsa.uiuc.edu/HTMLPrimer.html#GS is the same file as
http://www.ncsa.uiuc.edu/HTMLPrimer.html#HD
```

The sameFile( ) method is similar to the equals( ) method of the URL class. The main difference between sameFile( ) and equals( ) is that equals( ) considers the fragment identifier (if any), whereas sameFile( ) does not. The two URLs shown here do

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

not compare equal although they are the same file. Also, any object may be passed to `equals( )`; only `URL` objects can be passed to `sameFile( )`.

7.1.4.2. public String toExternalForm( )

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( )`. Therefore, this method is currently redundant and rarely used.

7.1.4.3. public URI toURI( ) throws URISyntaxException // Java 1.5

Java 1.5 adds a `toURI( )` method that converts a `URL` object to an equivalent `URI` object. We'll take up the `URI` class shortly. In the meantime, the main thing you need to know is that the `URI` class provides much more accurate, specification-conformant behavior than the `URL` class. For operations like absolutization and encoding, you should prefer the `URI` class where you have the option. In Java 1.4 and later, the `URL` class should be used primarily for the actual downloading of content from the remote server.

7.1.5. The Object Methods

`URL` inherits from `java.lang.Object`, so it has access to all the methods of the Object class. It overrides three to provide more specialized behavior: `equals( )`, `hashCode( )`, and `toString( )`.

7.1.5.1. public String toString( )

Like all good classes, `java.net.URL` has a `toString( )` method. Example 7-1 through Example 7-5 implicitly called this method when `URL`s were passed to `System.out.println( )`. As those examples demonstrated, 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 `toString( )` implicitly. Outside of print statements, it's more proper to use `toExternalForm( )` instead. If you do call `toString( )`, the syntax is simple:

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

```
URL codeBase = this.getCodeBase( );
String appletURL = codeBase.toString( );
```

7.1.5.2. public boolean equals(Object o)

An object is equal to a URL only if it is also a URL, both URLs point to the same file as determined by the sameFile( ) method, and both URLs have the same fragment identifier (or both URLs don't have fragment identifiers). Since equals( ) depends on sameFile( ), equals( ) has the same limitations as sameFile( ). 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/. Whether this makes sense depends on whether you think of a URL as a string or as a reference to a particular Internet resource.

Example 7-7 creates URL objects for http://www.ibiblio.org/ and http://metalab.unc.edu/ and tells you if they're the same using the equals() method.

Example 7-7. Are http://www.ibiblio.org and http://www.metalab.unc.edu the same?

```
import java.net.*;

public class URLEquality {

  public static void main (String[] args) {

    try {
      URL ibiblio = new URL ("http://www.ibiblio.org/");
      URL metalab = new URL("http://metalab.unc.edu/");
      if (ibiblio.equals(metalab)) {
        System.out.println(ibiblio + " is the same as " + metalab);
      }
      else {
        System.out.println(ibiblio + " is not the same as " + metalab);
      }
    }
    catch (MalformedURLException ex) {
      System.err.println(ex);
    }

  }

}
```

When you run this program, you discover:

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

```
% java URLEquality
http://www.ibiblio.org/ is the same as http://metalab.unc.edu/
```

7.1.5.3. public int hashCode( )

The `hashCode( )` method returns an `int` that is used when `URL` objects are used as keys in hash tables. Thus, it is called by the various methods of `java.util.Hashtable`. You rarely need to call this method directly, if ever. Hash codes for two different `URL` objects are unlikely to be the same, but it is certainly possible; there are far more conceivable URLs than there are four-byte integers.

7.1.6. Methods for Protocol Handlers

The last method in the `URL` class I'll just mention briefly here for the sake of completeness: `setURLStreamHandlerFactory( )`. It's primarily used by protocol handlers that are responsible for new schemes, not by programmers who just want to retrieve data from a URL. We'll discuss it in more detail in Chapter 16.

7.1.6.1. public static synchronized void setURLStreamHandlerFactory(URLStreamHandlerFactory factory)

This method sets the `URLStreamHandlerFactory` for the application and throws a generic `Error` if the factory has already been set. A `URLStreamHandler` is responsible for parsing the URL and then constructing the appropriate `URLConnection` object to handle the connection to the server. Most of the time this happens behind the scenes.

7.2. The URLEncoder and URLDecoder Classes

One of the challenges faced by the designers of the Web was dealing with the differences between operating systems. These differences can cause problems with URLs: for example, some operating systems allow spaces in filenames; some don't. Most operating systems won't complain about a # sign in a filename; but in a URL, a # sign indicates that the filename has ended, and a fragment identifier follows. Other special characters, nonalphanumeric characters, and so on, all of which may have a special meaning inside a URL or on another operating system, present similar problems. To solve these problems, characters used in URLs must come from a fixed subset of ASCII, specifically:

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

- 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 filename, they and all other characters should be encoded.

The encoding is very simple. Any characters that are not ASCII numerals, letters, or the punctuation marks specified earlier are converted into bytes and each byte is written as a percent sign followed by two hexadecimal digits. Spaces are a special case because they're so common. Besides being encoded as %20, they can be encoded as a plus sign (+). The plus sign itself is encoded as %2B. The / # = & and ? characters should be encoded when they are used as part of a name, and not as a separator between parts of the URL.

> This scheme doesn't work well in heterogeneous environments with multiple character sets. For example, on a U.S. Windows system, é is encoded as %E9. On a U.S. Mac, it's encoded as %8E. The existence of variations is a distinct shortcoming of the current URI specification that should be addressed in the future through Internationalized Resource Identifiers (IRIs).

The `URL` class does not perform encoding or decoding automatically. You can construct `URL` objects that use illegal ASCII and non-ASCII characters and/or percent escapes. Such characters and escapes are not automatically encoded or decoded when output by methods such as `getPath()` and `toExternalForm( )`. You are responsible for making sure all such characters are properly encoded in the strings used to construct a `URL` object.

Luckily, Java provides a `URLEncoder` class to encode strings in this format. Java 1.2 adds a `URLDecoder` class that can decode strings in this format. Neither of these classes will be instantiated.

```
public class URLDecoder extends Object
public class URLEncoder extends Object
```


Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

7.2.1. URLEncoder

In Java 1.3 and earlier, the `java.net.URLEncoder` class contains a single static method called `encode( )` that encodes a `String` according to these rules:

```
public static String encode(String s)
```

This method always uses the default encoding of the platform on which it runs, so it will produce different results on different systems. As a result, Java 1.4 deprecates this method and replaces it with a method that requires you to specify the encoding:

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

`Both variants` change any nonalphanumeric characters into % sequences (except the space, underscore, hyphen, period, and asterisk characters). Both also encode all non-ASCII characters. The space is converted into a plus sign. These methods are a little over-aggressive; they also convert tildes, single quotes, exclamation points, and parentheses to percent escapes, even though they don't absolutely have to. However, this change isn't forbidden by the URL specification, so web browsers deal reasonably with these excessively encoded URLs.

Both variants return a new `String`, suitably encoded. The Java 1.3 `encode( )` method uses the platform's default encoding to calculate percent escapes. This encoding is typically ISO-8859-1 on U.S. Unix systems, Cp1252 on U.S. Windows systems, MacRoman on U.S. Macs, and so on in other locales. Because both encoding and decoding are platform- and locale-specific, this method is annoyingly non-interoperable, which is precisely why it has been deprecated in Java 1.4 in favor of the variant that requires you to specify an encoding. However, if you just pick the platform default encoding, your program will be as platform- and locale-locked as the Java 1.3 version. Instead, you should always pick UTF-8, never anything else. UTF-8 is compatible with the new IRI specification, the `URI` class, modern web browsers, and more other software than any other encoding you could choose.

Example 7-8 is a program that uses `URLEncoder.encode( )` to print various encoded strings. Java 1.4 or later is required to compile and run it.

Example 7-8. x-www-form-urlencoded strings

```
import java.net.URLEncoder;
import java.io.UnsupportedEncodingException;
```

```
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\"marks",
                                           "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=signs",
                                           "UTF-8"));
      System.out.println(URLEncoder.encode("This&string&has&ampersands",
                                           "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");
    }

  }

}
```

Here is the output. Note that the code needs to be saved in something other than ASCII, and the encoding chosen should be passed as an argument to the compiler to account for the non-ASCII characters in the source code.

```
% 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
```

Notice in particular that this method encodes the forward slash, the ampersand, the equals sign, and the colon. It does not attempt to determine how these characters are being used in a URL. Consequently, you have to encode URLs piece by piece rather than encoding an entire URL in one method call. This is an important point, because the most common use of `URLEncoder` is in preparing query strings for communicating with server-side programs that use `GET`. For example, suppose you want to encode this query string used for an AltaVista search:

```
pg=q&kl=XX&stype=stext&q=+"Java+I/O"&search.x=38&search.y=3
```

This code fragment encodes it:

```
String query = URLEncoder.encode(
 "pg=q&kl=XX&stype=stext&q=+\"Java+I/O\"&search.x=38&search.y=3");
System.out.println(query);
```

Unfortunately, the output is:

```
pg%3Dq%26kl%3DXX%26stype%3Dstext%26q%3D%2B%22Java%2BI%2FO%22%26search
.x%3D38%26search.y%3D3
```

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 `=` in the previous string, and characters that need to be encoded. Consequently, URLs need to be encoded a piece at a time like this:

```
String query = URLEncoder.encode("pg");
query += "=";
query += URLEncoder.encode("q");
query += "&";
query += URLEncoder.encode("kl");
query += "=";
query += URLEncoder.encode("XX");
query += "&";
query += URLEncoder.encode("stype");
query += "=";
query += URLEncoder.encode("stext");
query += "&";
query += URLEncoder.encode("q");
query += "=";
query += URLEncoder.encode("\"Java I/O\"");
query += "&";
query += URLEncoder.encode("search.x");
query += "=";
query += URLEncoder.encode("38");
```

```
query += "&";
query += URLEncoder.encode("search.y");
query += "=";
query += URLEncoder.encode("3");
System.out.println(query);
```

The output of this is what you actually want:

```
pg=q&kl=XX&stype=stext&q=%2B%22Java+I%2FO%22&search.x=38&search.y=3
```

Example 7-9 is a `QueryString` class that uses the `URLEncoder` to encode successive name and value pairs in a Java object, which will be used for sending data to server-side programs. When you create a `QueryString`, you can supply the first name-value pair to the constructor as individual strings. To add further pairs, call the `add( )` method, which also takes two strings as arguments and encodes them. The `getQuery( )` method returns the accumulated list of encoded name-value pairs.

Example 7-9. -The QueryString class

```
package com.macfaq.net;

import java.net.URLEncoder;
import java.io.UnsupportedEncodingException;

public class QueryString {

  private StringBuffer query = new StringBuffer( );

  public QueryString(String name, String value) {
    encode(name, value);
  }

  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 String getQuery( ) {
```


Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

```
      return query.toString( );
    }

    public String toString( ) {
      return getQuery( );
    }

  }
```

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

```
QueryString qs = new QueryString("pg", "q");
qs.add("kl", "XX");
qs.add("stype", "stext");
qs.add("q", "+\"Java I/O\"");
qs.add("search.x", "38");
qs.add("search.y", "3");
String url = "http://www.altavista.com/cgi-bin/query?" + qs;
System.out.println(url);
```

7.2.2. URLDecoder

The corresponding `URLDecoder` class has two static methods that decode strings encoded in x-www-form-url-encoded format. That is, they convert all plus signs to spaces and all percent escapes to their corresponding character:

```
public static String decode(String s) throws Exception
public static String decode(String s, String encoding)  // Java 1.4
  throws UnsupportedEncodingException
```

The first variant is used in Java 1.3 and 1.2. The second variant is used in Java 1.4 and later. If you have any doubt about which encoding to use, pick UTF-8. It's more likely to be correct than anything else.

An `IllegalArgumentException` may be thrown if the string contains a percent sign that isn't followed by two hexadecimal digits or decodes into an illegal sequence. Then again it may not be. This is implementation-dependent, and what happens when an illegal sequence is detected and does not throw an `IllegalArgumentException` is undefined. In Sun's JDK 1.4, no exception is thrown and extra bytes with no apparent meaning are added to the undecodable string. This is truly brain-damaged, and possibly a security hole.

Since this method does not touch non-escaped characters, you can pass an entire URL to it rather than splitting it into pieces first. For example:

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

```
String input = "http://www.altavista.com/cgi-bin/" +
"query?pg=q&kl=XX&stype=stext&q=%2B%22Java+I%2FO%22&search.x=38&search.y=3";
 try {
  String output = URLDecoder.decode(input, "UTF-8");
  System.out.println(output);
 }
```

7.3. The URI Class

A URI is an abstraction 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 1.4 and later, URIs are represented by the `java.net.URI` class. This class differs from the `java.net.URL` class in three important ways:

- 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.
- The `URI` class is more conformant to the relevant specifications than the `URL` class.
- A `URI` object can represent a relative URI. The `URL` class absolutizes all URIs before storing them.

In brief, a `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. The `URL` class has some string parsing methods, such as `getFile( )` and `getRef( )`, but many of these are broken and don't always behave exactly as the relevant specifications say they should. Assuming you're using Java 1.4 or later and therefore have a choice, you should use the `URL` class when you want to download the content of a URL and the `URI` class when you want to use the URI for identification rather than retrieval, for instance, to represent an XML namespace URI. In some cases when you need to do both, you may convert from a `URI` to a `URL` with the `toURL( )` method, and in Java 1.5 you can also convert from a `URL` to a `URI` using the `toURI( )` method of the `URL` class.

7.3.1. Constructing a URI

URIs are built from strings. Unlike the `URL` class, the `URI` class does not depend on an underlying protocol handler. As long as the URI is syntactically correct, Java does not need to understand its protocol in order to create a representative URI object. Thus, unlike the `URL` class, the `URI` class can be used for new and experimental URI schemes.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

7.3.1.1. public URI(String uri) throws URISyntaxException

This is the basic constructor that 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`. This is a checked exception, so you need to either catch it or declare that the method where the constructor is invoked can throw it. However, one syntactic rule is not checked. In contradiction to the URI specification, the characters used in the URI are not limited to ASCII. They can include other Unicode characters, such as ø and é. Syntactically, there are very few restrictions on URIs, especially once the need to encode non-ASCII characters is removed and relative URIs are allowed. Almost any string can be interpreted as a URI.

7.3.1.2. public URI(String scheme, String schemeSpecificPart, String fragment) throws URISyntaxException

This constructor is mostly used for nonhierarchical URIs. The scheme is the URI's protocol, such as http, urn, tel, and so forth. It must be composed exclusively of ASCII letters and digits and the three punctuation 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");
```

The scheme-specific part depends on the syntax of the URI scheme; it's one thing for an http URL, another for a mailto URL, and something else again for a tel URI. Because the `URI` class encodes illegal characters with percent escapes, there's effectively no syntax error you can make in this part.

Finally, the third argument contains the fragment identifier, if any. Again, characters that are forbidden in a fragment identifier are escaped automatically. Passing null for this argument simply omits the fragment identifier.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

7.3.1.3. public URI(String scheme, String host, String path, String fragment) throws URISyntaxException

This 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");
```

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

If the constructor cannot form a legal hierarchical URI from the supplied pieces—for instance, if there is a scheme so the URI has to be absolute but the path doesn't start with /—then it throws a `URISyntaxException`.

7.3.1.4. public URI(String scheme, String authority, String path, String query, String fragment) throws URISyntaxException

This constructor is basically the same as the previous one, with the addition of a query string component. For example:

```
URI today= new URI("http", "www.ibiblio.org", "/javafaq/index.html",
                   "referrer=cnet&date=2004-08-23",  "today");
```

As usual, any unescapable syntax errors cause a `URISyntaxException` to be thrown and null can be passed to omit any of the arguments.

7.3.1.5. public URI(String scheme, String userInfo, String host, int port, String path, String query, String fragment) throws URISyntaxException

This 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@metalab.unc.edu",
  "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.

7.3.1.6. public static URI create(String uri)

This is not a constructor, but rather a static factory method. Unlike the constructors, it does not throw a `URISyntaxException`. If you're sure your URIs are legal and do not violate any of the rules, you can use this method. For example, this invocation creates a `URI` for anonymous FTP access using an email address as password:

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

If the URI does prove to be malformed, this method throws an `IllegalArgumentException`. This is a runtime exception, so you don't have to explicitly declare it or catch it.

7.3.2. 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
```

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. The `getRaw`*`Foo`*`( )` methods return the encoded forms of the parts of the URI, while the equivalent `get`*`Foo`*`()` methods 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( )
```

There's no `getRawScheme( )` method because the URI specification requires that all scheme names be composed exclusively of URI-legal ASCII characters and does not allow percent escapes in scheme names.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

These methods all return null if the particular `URI` object does not have the relevant component: for example, a relative URI without a scheme or an http URI without a fragment identifier.

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 details of the scheme-specific part vary depending on the type of the scheme. For example, in a tel URL, the scheme-specific part has the syntax of a telephone number. However, in many useful URIs, including the very common file and http URLs, the scheme-specific part has a particular hierarchical format divided into an authority, a path, and a query string. The authority is further divided into user info, host, and port. 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( )
```

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( )
```

Remember the `URI` class differs from the URI specification in that non-ASCII characters such as é and ü are never percent-escaped in the first place, and thus will still be present in the strings returned by the `getRaw`*`Foo`*`()` methods unless the strings originally used to construct the URI object were encoded.

There are no `getRawPort( )` and `getRawHost( )` methods because these components are always guaranteed to be made up of ASCII characters, at least for now. Internationalized domain names are coming, and may require this decision to be rethought in future versions of Java.

In the event that the specific URI does not contain this information—for instance, the URI *http://www.example.com* has no user info, path, port, or query string—the relevant methods return null. `getPort( )` is the single exception. Since it's declared to return an `int`, it can't return `null`. Instead, it returns -1 to indicate an omitted port.

For various technical reasons that don't have a lot of practical impact, Java can't always initially detect syntax errors in the authority component. The immediate symptom of this failing is normally an inability to return the individual parts of the authority: port, host, and user info. In this event, you can call `parseServerAuthority()` to force the authority to be reparsed:

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

The original `URI` does not change (`URI` objects are immutable), but the `URI` returned will have separate authority parts for user info, host, and port. If the authority cannot be parsed, a `URISyntaxException` is thrown.

Example 7-10 uses these methods to split URIs entered on the command line into their component parts. It's similar to Example 7-4 but works with any syntactically correct URI, not just the ones Java has a protocol handler for.

Example 7-10. 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( ));
```


Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

```
            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.getUserInfo( ));
              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( ));
          } // end else
        }  // end try
        catch (URISyntaxException ex) {
          System.err.println(args[i] + " does not seem to be a URI.");
        }
        System.out.println( );
      }  // end for

    }  // end main

  }  // end URISplitter
```

Here's the result of running this against three of the URI examples in this section:

```
% java URISplitter tel:+1-800-9988-9938 \
           http://www.xml.com/pub/a/2003/09/17/stax.html#id=_hbc \
           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

The URI is http://www.xml.com/pub/a/2003/09/17/stax.html#id=_hbc
This is a hierarchical URI.
The scheme is http
The host is null
The user info is null
The port is -1
The path is /pub/a/2003/09/17/stax.html
The query string is null
The fragment ID is id=_hbc

The URI is urn:isbn:1-565-92870-9
This is an opaque URI.
The scheme is urn
```


Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

```
The scheme specific part is isbn:1-565-92870-9
The fragment ID is null
```

7.3.3. Resolving Relative URIs

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

7.3.3.1. public URI resolve(URI uri)

This method compares the `uri` argument to this `URI` and uses 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*.

If the invoking `URI` does not contain an absolute URI itself, 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 relative = new URI("jnp3/examples/07/index.html");
URI resolved = top.resolve(relative);
```

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

7.3.3.2. public URI resolve(String uri)

This is a convenience method that simply converts the string argument to a URI and then resolves it against the invoking URI, returning a new URI object as the result. That is, it's equivalent to `resolve(new URI(str))`. Using this method, the previous two samples can be rewritten as:

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


Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

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

7.3.3.3. public URI relativize(URI uri)

It's also possible to reverse this procedure; that is, to go from an absolute URI to a relative one. 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*.

7.3.4. Utility Methods

The `URI` class has the usual batch of utility methods: `equals()`, `hashCode( )`, `toString( )`, and `compareTo( )`.

7.3.4.1. public boolean equals(Object o)

URIs are tested for equality pretty much as you'd expect. It's not a direct string comparison. Equal URIs must both either be hierarchical or opaque. The scheme and authority parts are compared without considering case. That is, *http* and *HTTP* are the same scheme, and *www.example.com* is the same authority as *www.EXAMPLE.com*. The rest of the URI is case-sensitive, except for hexadecimal digits used to escape illegal characters. Escapes are *not* decoded before comparing. *http://www.example.com/A* and *http://www.example.com/%41* are unequal URIs.

7.3.4.2. public int hashCode( )

The `hashCode( )` method is a usual `hashCode( )` method, nothing special. Equal URIs do have the same hash code and unequal URIs are fairly unlikely to share the same hash code.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

7.3.4.3. public int compareTo(Object o)

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

- If the schemes are different, the schemes are compared, without considering case.
- Otherwise, if the schemes are the same, a hierarchical URI is considered to be less than an opaque URI with the same scheme.
- If both URIs are opaque URIs, they're ordered according to their scheme-specific parts.
- If both the scheme and the opaque scheme-specific parts are equal, the URIs are compared by their fragments.
- 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.
- If the schemes and the authorities are equal, the path is used to distinguish them.
- If the paths are also equal, the query strings are compared.
- If the query strings are equal, the fragments are compared.

URIs are not comparable to any type except themselves. Comparing a `URI` to anything except another `URI` causes a `ClassCastException`.

7.3.4.4. public String toString( )

The `toString( )` method returns an *unencoded* string form of the `URI`. That is, characters like é and \ are not percent-escaped unless they were percent-escaped in the strings used to construct this `URI`. Therefore, the result of calling this method is not guaranteed to be a syntactically correct URI. This form is sometimes useful for display to human beings, but not for retrieval.

7.3.4.5. public String toASCIIString( )

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. This is the string form of the URI you should use most of the time. Even if the form returned by `toString( )` is more legible for humans, they may still copy and paste it into areas that are not expecting an illegal URI. `toASCIIString( )` always returns a syntactically correct URI.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

7.4. Proxies

Many systems access the Web and sometimes other non-HTTP parts of the Internet through *proxy servers*. A proxy server receives a request for a remote server from a local client. The proxy server makes the request to the remote server and forwards the result back to the local client. Sometimes this is done for security reasons, such as to prevent remote hosts from learning private details about the local network configuration. Other times it's done to prevent users from accessing forbidden sites by filtering outgoing requests and limiting which sites can be viewed. For instance, an elementary school might want to block access to http://www.playboy.com. And still other times it's done purely for performance, to allow multiple users to retrieve the same popular documents from a local cache rather than making repeated downloads from the remote server.

Java programs based on the `URL` class can work through most common proxy servers and protocols. Indeed, 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.

7.4.1. System Properties

For basic operations, all you have to do is set a few system properties to point to the addresses of your local proxy servers. If you are using a pure HTTP proxy, set `http.proxyHost` to the domain name or the IP address of your proxy server and `http.proxyPort` to the port of the proxy server (the default is 80). There are several ways to do this, including calling `System.setProperty()` from within your Java code or using the -D options when launching the program. This example sets the proxy server to 192.168.254.254 and the port to 9000:

```
% java -Dhttp.proxyHost=192.168.254.254  -Dhttp.proxyPort=9000
               com.domain.Program
```

If you want to exclude a host from being proxied and connect directly instead, set the `http.nonProxyHosts` system property to its hostname or IP address. To exclude multiple hosts, separate their names by vertical bars. For example, this code fragment proxies everything except *java.oreilly.com* and *xml.oreilly.com*:

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


Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

You can also use an asterisk as a wildcard to indicate that all the hosts within a particular domain or subdomain should not be proxied. For example, to proxy everything except hosts in the *oreilly.com* domain:

```
% java -Dhttp.proxyHost=192.168.254.254  -Dhttp.nonProxyHosts=*.oreilly.com
               com.domain.Program
```

If you are using an FTP proxy server, set the `ftp.proxyHost`, `ftp.proxyPort`, and `ftp.nonProxyHosts` properties in the same way.

Java does not support any other application layer proxies, but if you're using a transport layer SOCKS proxy for all TCP connections, you can identify it with the `socksProxyHost` and `socksProxyPort` system properties. Java does not provide an option for nonproxying with SOCKS. It's an all-or-nothing decision.

7.4.2. The Proxy Class

Java 1.5 allows more fine-grained control of proxy servers from within a Java program. Specifically, this allows you to choose different proxy servers for different remote hosts. The proxies themselves are represented by instances of the `java.net.Proxy` class. There are still only three kinds of proxies, HTTP, SOCKS, and direct connections (no proxy at all), represented by three constants in the `Proxy.Type` enum:

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

Besides its type, the other important piece of information about a proxy is its address and port, given as a `SocketAddress` object. For example, this code fragment creates 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);
```

Although there are only three kinds of proxy objects, there can be many proxies of the same type for different proxy servers on different hosts.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

7.4.3. The ProxySelector Class

Each running Java 1.5 virtual machine has a single `java.net.ProxySelector` object it uses to locate the proxy server for different connections. The default `ProxySelector` merely inspects the various system properties and the URL's protocol to decide how to connect to different hosts. However, you can install your own subclass of `ProxySelector` in place of the default selector and use it to choose different proxies based on protocol, host, path, time of day, or other criteria.

The key to this class is the abstract `select( )` method:

```
public abstract List<Proxy> select(URI uri)
```

Java passes this method a `URI` object (not a `URL` object) representing the host to which a connection is needed. For a connection made with the URL class, this object typically has the form *http://www.example.com/* or *ftp://ftp.example.com/pub/files/*, or some such. For a pure TCP connection made with the Socket class, this URI will have the form *socket://host:port*:, for instance, *socket://www.example.com:80*. The `ProxySelector` object then chooses the right proxies for this type of object and returns them in a `List<Proxy>`.

The second abstract method in this class you must implement is `connectFailed( )`:

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

This is a callback method used to warn a program that the proxy server isn't actually making the connection. Example 7-11 demonstrates with a `ProxySelector` that attempts to use the proxy server at *proxy.example.com* for all HTTP connections unless the proxy server has previously failed to resolve a connection to a particular URL. In that case, it suggests a direct connection instead.

Example 7-11. A ProxySelector that remembers what it can connect to

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

public class LocalProxySelector extends ProxySelector {

  private List failed = new ArrayList( );

  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);
      }

    }
```

As I already said, each running virtual machine has exactly one `ProxySelector`. To change the `ProxySelector`, pass the new selector to the static `ProxySelector.setDefault( )` method, like so:

```
ProxySelector selector = new LocalProxySelector( ):
ProxySelector.setDefault(selector);
```

From this point forward, all connections opened by that virtual machine will ask the `ProxySelector` for the right proxy to use. You normally shouldn't use this in code running in a shared environment. For instance, you wouldn't change the `ProxySelector` in a servlet because that would change the `ProxySelector` for all servlets running in the same container.

7.5. 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. (Server-side programs that use the `POST` method require the `URLConnection` class and are discussed in Chapter 15.) All you need to know is what combination of names and values the program expects to receive, and cook up a URL with a query string that provides the requisite names and values. All names and values must be x-www-form-url-encoded—as by the `URLEncoder.encode()` method, discussed earlier in this chapter.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

There are a number of ways to determine the exact syntax for a query string that talks to a particular program. If you've written the server-side program yourself, you already know the name-value pairs it expects. If you've installed a third-party program on your own server, the documentation for that program should tell you what it expects.

On the other hand, if you're talking to a program on a third-party server, matters are a little trickier. You can always ask people at the remote server to provide you with the specifications for talking to their site. However, even if they don't mind doing this, there's probably no single person whose job description includes "telling third-party hackers with whom we have no business relationship exactly how to access our servers." Thus, unless you happen upon a particularly friendly or bored individual who has nothing better to do with their time except write long emails detailing exactly how to access their server, you're going to have to do a little reverse engineering.

> This is beginning to change. A number of web sites have realized the value of opening up their systems to third party developers and have begin publishing developers' kits that provide detailed information on how to construct URLs to access their services. Sites like Safari and Amazon that offer RESTful, URL-based interfaces are easily accessed through the `URL` class. SOAP-based services like eBay's and Google's are much more difficult to work with.

Many programs are designed to process form input. If this is the case, it's straightforward to figure out what input the program expects. The method the form uses should be the value of the `METHOD` attribute of the `FORM` element. This value should be either `GET`, in which case you use the process described here, or `POST`, in which case you use the process described in Chapter 15. The part of the URL that precedes the query string is given by the value of the `ACTION` attribute of the `FORM` element. Note that this may be a relative URL, in which case you'll need to determine the corresponding absolute URL. Finally, the name-value pairs are simply the `NAME` attributes of the `INPUT` elements, except for any `INPUT` elements whose `TYPE` attribute has the value submit.

For example, consider this HTML form for the local search engine on my Cafe con Leche site. You can see that it uses the `GET` method. The program that processes the form is accessed via the URL http://www.google.com/search. It has four separate name-value pairs, three of which have default values:

```
<form name="search" action="http://www.google.com/search" method="get">
  <input name="q" />
```


Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

```
  <input type="hidden" value="cafeconleche.org" name="domains" />
  <input type="hidden" name="sitesearch" value="cafeconleche.org" />
  <input type="hidden" name="sitesearch2" value="cafeconleche.org" />
   <br />
   <input type="image" height="22" width="55"
      src="images/search_blue.gif" alt="search" border="0"
      name="search-image" />
</form>
```

The type of the `INPUT` field doesn't matter—for instance, it doesn't matter if it's a set of checkboxes, a pop-up list, or a text field—only the name of each `INPUT` field and the value you give it is significant. The single exception is a submit input that tells the web browser when to send the data but does not give the server any extra information. In some cases, you may find hidden `INPUT` fields that must have particular required default values. This form has three hidden `INPUT` fields.

In some cases, the program you're talking to may not be able to handle arbitrary text strings for values of particular inputs. However, since the form is meant to be read and filled in by human beings, it should provide sufficient clues to figure out what input is expected; for instance, that a particular field is supposed to be a two-letter state abbreviation or a phone number.

A program that doesn't respond to a form is much harder to reverse engineer. For example, at http://www.ibiblio.org/nywc/bios.phtml, you'll find a lot of links to PHP pages that talk to a database to retrieve a list of musical works by a particular composer. However, there's no form anywhere that corresponds to this program. It's all done by hardcoded URLs. In this case, the best you can do is look at as many of those URLs as possible and see whether you can guess what the server expects. If the designer hasn't tried to be too devious, this information isn't hard to figure out. For example, these URLs are all found on that page:

```
http://www.ibiblio.org/nywc/compositionsbycomposer.phtml?last=Anderson
    &first=Beth&middle=
http://www.ibiblio.org/nywc/compositionsbycomposer.phtml?last=Austin
    &first=Dorothea&middle=
http://www.ibiblio.org/nywc/compositionsbycomposer.phtml?last=Bliss
    &first=Marilyn&middle=
http://www.ibiblio.org/nywc/compositionsbycomposer.phtml?last=Hart
    &first=Jane&middle=Smith
```

Looking at these, you can guess that this particular program expects three inputs named first, middle, and last, with values that consist of the first, middle, and last names of a composer, respectively. Sometimes the inputs may not have such obvious names. In this case, you have to do some experimenting, first copying some existing values and then tweaking them to see what values are and aren't accepted. You don't need to do this in a Java program. You can simply edit the URL in the Address or Location bar of your web browser window.

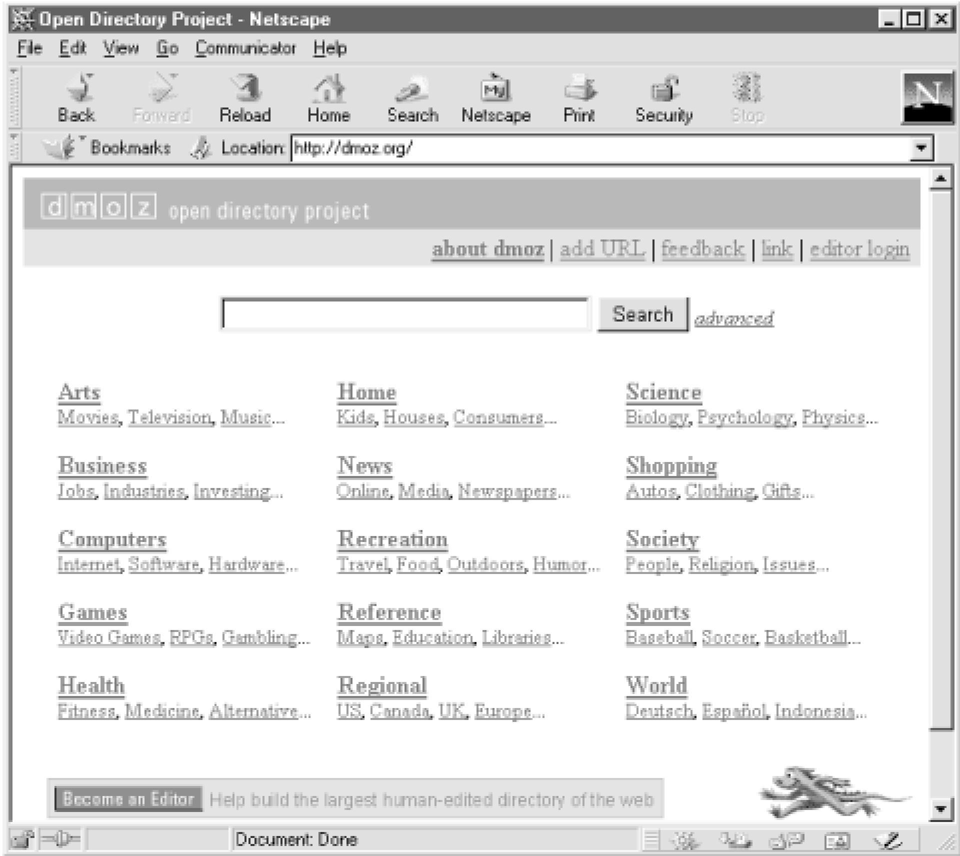
The likelihood that other hackers may experiment with your own server-side programs in such a fashion is a good reason to make them extremely robust against unexpected input.

Regardless of how you determine the set of name-value pairs the server expects, communicating with it once you know them is simple. All you have to do is create a query string that includes the necessary name-value pairs, then form a URL that includes that query string. Send the query string to the server and read its response using the same methods you use to connect to a server and retrieve a static HTML page. There's no special protocol to follow once the URL is constructed. (There is a special protocol to follow for the `POST` method, however, which is why discussion of that method will have to wait until Chapter 15.)

To demonstrate this procedure, let's write a very simple command-line program to look up topics in the Netscape Open Directory (http://dmoz.org/). This site is shown in Figure 7-3 and it has the advantage of being really simple.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

Figure 7-3. The basic user interface for the Open Directory



The basic Open Directory interface is a simple form with one input field named `search`; input typed in this field is sent to a CGI program at http://search.dmoz.org/cgi-bin/search, which does the actual search. The HTML for the form looks like this:

```
<form accept-charset="UTF-8"
      action="http://search.dmoz.org/cgi-bin/search" method="GET">
          <input size=30 name=search>

<input type=submit value="Search">
<a href="http://search.dmoz.org/cgi-bin/search?a.x=0">
<small><i>advanced</i></small></a>
</form>
```

There are only two input fields in this form: the Submit button and a text field named Search. Thus, to submit a search request to the Open Directory, you just need to collect the search string, encode it in a query string, and send it to http://search.dmoz.org/cgi-bin/search. For example, to search for "java", you would open a connection to the URL http://search.dmoz.org/cgi-bin/search?search=java and read the resulting input stream. Example 7-12 does exactly this.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

Example 7-12. Do an Open Directory search

```
import com.macfaq.net.*;

import java.net.*;
import java.io.*;

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("search", target);
    try {
      URL u = new URL("http://search.dmoz.org/cgi-bin/search?" + query);
      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);
    }

  }

}
```

Of course, a lot more effort could be expended on parsing and displaying the results. But notice how simple the code was to talk to this server. Aside from the funky-looking URL and the slightly greater likelihood that some pieces of it need to be x-www-form-url-encoded, talking to a server-side program that uses `GET` is no harder than retrieving any other HTML page.

7.6. Accessing Password-Protected Sites

Many popular sites, such as *The Wall Street Journal*, require a username and password for access. Some sites, such as the W3C member pages, implement this correctly through HTTP authentication. Others, such as the Java Developer Connection, implement it incorrectly

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

through cookies and HTML forms. Java's URL class can access sites that use HTTP authentication, although you'll of course need to tell it what username and password to use. Java does not provide support for sites that use nonstandard, cookie-based authentication, in part because Java doesn't really support cookies in Java 1.4 and earlier, in part because this requires parsing and submitting HTML forms, and, lastly, because cookies are completely contrary to the architecture of the Web. (Java 1.5 does add some cookie support, which we'll discuss in the next chapter. However, it does not treat authentication cookies differently than any other cookies.) You can provide this support yourself using the URLConnection class to read and write the HTTP headers where cookies are set and returned. However, doing so is decidedly nontrivial and often requires custom code for each site you want to connect to. It's really hard to do short of implementing a complete web browser with full HTML forms and cookie support. Accessing sites protected by standard, HTTP authentication is much easier.

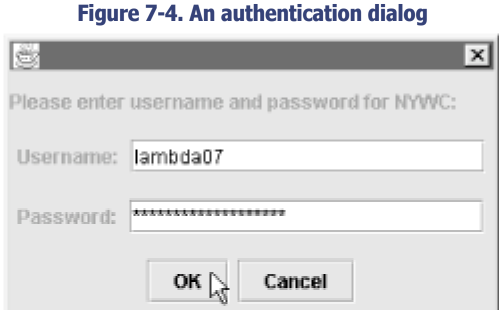
7.6.1. The Authenticator Class

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 // Java 1.2
```

Since Authenticator is an abstract class, you must subclass it. Different subclasses may retrieve the information in different ways. For example, a character mode program might just ask the user to type the username and password on System.in. A GUI program would likely put up a dialog box like the one shown in Figure 7-4. An automated robot might read the username out of an encrypted file.

Figure 7-4. 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)
```


Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

For example, if you've written an `Authenticator` subclass named `DialogAuthenticator`, you'd install it like this:

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

You only need to do this once. From this point forward, when the `URL` class needs a username and password, 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
```

The `address` argument is the host for which authentication is required. The `port` argument is the port on that host, and the `protocol` argument is the application layer protocol by which the site is being accessed. The HTTP server provides the `prompt`. It's typically the name of the realm for which authentication is required. (Some large web servers such as *www.ibiblio.org* have multiple realms, each of which requires different usernames and passwords.) The `scheme` is the authentication scheme being used. (Here the word *scheme* is not being used as a synonym for *protocol*. Rather it is an HTTP authentication scheme, typically basic.)

Untrusted applets are not allowed to ask the user for a name and password. Trusted applets can do so, but only if they possess the `requestPasswordAuthentication` `NetPermission`. Otherwise, `Authenticator.requestPasswordAuthentication( )` throws a `SecurityException`.

The `Authenticator` subclass must override the `getPasswordAuthentication( )` method. Inside this method, you 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( )
```

If you don't want to authenticate this request, return `null`, and Java will tell the server it doesn't know how to authenticate the connection. If you submit an incorrect username or password, Java will call `getPasswordAuthentication( )` again to give you another chance to provide the right data. You normally have five tries to get the username and password correct; after that, `openStream( )` throws a `ProtocolException`.

Usernames and passwords are cached within the same virtual machine session. Once you set the correct password for a realm, you shouldn't be asked for it again unless you've explicitly deleted the password by zeroing out the `char` array that contains it.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

You 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( )  // Java 1.4
```

These methods either return the information as given in the last call to `requestPasswordAuthentication( )` or return `null` if that information is not available. (`getRequestingPort( )` returns -1 if the port isn't available.) The last method, `getRequestingHost( )`, is only available in Java 1.4 and later; in earlier releases you can call `getRequestingSite( ).getHostName( )` instead.

Java 1.5 adds two more methods to this class:

```
protected final String getRequestingURL( )  // Java 1.5
protected Authenticator.RequestorType getRequestorType( )
```

The `getRequestingURL( )` method returns the complete URL for which authentication has been requested—an important detail if a site uses different names and passwords for different files. The `getRequestorType( )` method returns one of the two named constants `Authenticator.RequestorType.PROXY` or `Authenticator.RequestorType.SERVER` to indicate whether the server or the proxy server is requesting the authentication.

7.6.2. 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. A `String` would have to wait to be garbage collected before it could be erased, and even then it might still exist somewhere in memory on the local system, possibly even on disk if the block of memory that contained it had been swapped out to virtual memory at one point. Both username and password are set in the constructor:

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

Each is accessed via a `getter` method:

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

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

7.6.3. 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. However, anything the user types into it is echoed as an asterisk. This way, the password is safe from anyone looking over the user's shoulder at what's being typed on the screen.

`JPasswordField` also stores the passwords as a `char` array so that when you're done with the password you can overwrite it with zeros. It provides the `getPassword( )` method to return this:

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

Otherwise, you mostly use the methods it inherits from the `JTextField` superclass. Example 7-13 demonstrates a Swing-based `Authenticator` subclass that brings up a dialog to ask the user for his username and password. Most of this code handles the GUI. A `JPasswordField` collects the password and a simple `JTextField` retrieves the username. Figure 7-4 showed the rather simple dialog box this produces.

Example 7-13. A GUI authenticator

```
package com.macfaq.net;

import java.net.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class DialogAuthenticator extends Authenticator {

  private JDialog passwordDialog;
  private JLabel mainLabel
   = new JLabel("Please enter username and password: ");
  private JLabel userLabel = new JLabel("Username: ");
  private JLabel passwordLabel = new JLabel("Password: ");
```


Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

```
    private JTextField usernameField = new JTextField(20);
    private JPasswordField passwordField = new JPasswordField(20);
    private JButton okButton = new JButton("OK");
    private JButton cancelButton = new JButton("Cancel");

    public DialogAuthenticator( ) {
      this("", new JFrame( ));
    }

    public DialogAuthenticator(String username) {
      this(username, new JFrame( ));
    }

    public DialogAuthenticator(JFrame parent) {
      this("", parent);
    }

    public DialogAuthenticator(String username, JFrame parent) {

      this.passwordDialog = new JDialog(parent, true);
      Container pane = passwordDialog.getContentPane( );
      pane.setLayout(new GridLayout(4, 1));
      pane.add(mainLabel);
      JPanel p2 = new JPanel( );
      p2.add(userLabel);
      p2.add(usernameField);
      usernameField.setText(username);
      pane.add(p2);
      JPanel p3 = new JPanel( );
      p3.add(passwordLabel);
      p3.add(passwordField);
      pane.add(p3);
      JPanel p4 = new JPanel( );
      p4.add(okButton);
      p4.add(cancelButton);
      pane.add(p4);
      passwordDialog.pack( );

      ActionListener al = new OKResponse( );
      okButton.addActionListener(al);
      usernameField.addActionListener(al);
      passwordField.addActionListener(al);
      cancelButton.addActionListener(new CancelResponse( ));

    }

    private void show( ) {

      String prompt = this.getRequestingPrompt( );
      if (prompt == null) {
        String site     = this.getRequestingSite( ).getHostName( );
        String protocol = this.getRequestingProtocol( );
        int    port     = this.getRequestingPort( );
        if (site != null & protocol != null) {
          prompt = protocol + "://" + site;
          if (port > 0) prompt += ":" + port;
        }
        else {
```


Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

```
          prompt = "";
        }

      }

      mainLabel.setText("Please enter username and password for "
       + prompt + ": ");
      passwordDialog.pack( );
      passwordDialog.show( );

    }

    PasswordAuthentication response = null;

    class OKResponse implements ActionListener {

      public void actionPerformed(ActionEvent e) {

        passwordDialog.hide( );
        // The password is returned as an array of
        // chars for security reasons.
        char[] password = passwordField.getPassword( );
        String username = usernameField.getText( );
        // Erase the password in case this is used again.
        passwordField.setText("");
        response = new PasswordAuthentication(username, password);

      }

    }

    class CancelResponse implements ActionListener {

      public void actionPerformed(ActionEvent e) {

        passwordDialog.hide( );
        // Erase the password in case this is used again.
        passwordField.setText("");
        response = null;

      }

    }

    public PasswordAuthentication getPasswordAuthentication( ) {

      this.show( );
      return this.response;

    }

  }
```

Example 7-14 is a revised `SourceViewer` program that asks the user for a name and password using the `DialogAuthenticator` class.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

Example 7-14. A program to download password-protected web pages

```
import java.net.*;
import java.io.*;
import com.macfaq.net.DialogAuthenticator;

public class SecureSourceViewer {

  public static void main (String args[]) {

    Authenticator.setDefault(new DialogAuthenticator( ));

    for (int i = 0; i < args.length; i++) {

      try {
        //Open the URL for reading
        URL u = new URL(args[i]);
        InputStream 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);
      }

      // print a blank line to separate pages
      System.out.println( );

    } //  end for

    // Since we used the AWT, we have to explicitly exit.
    System.exit(0);

  } // end main

}  // end SecureSourceViewer
```


Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004