Chapter 3
URLs and URIs

Contents:

- **URIs**: URLs and Relative URLs
- **The URL Class:** Creating New URLs, Retrieving Data From a URL, Splitting a URL into Pieces, Equality & Comparison and Conversion
- **The URI Class**: Constructing a URI, The Parts of the URI, Resolving Relative URIs, Equality & Comparison and String Representation
- **x-www-form-urlencoded:** URL Encoder and URL Decoder
- **Proxies**: System Properties, The ProxyClass and The ProxySelector Class
- Communicating with Server-Side Programs Through GET
- **Accessing Password-Protected Sites**: The Authenticator Class, The Password Authentication Class and The JPasswordField Class

URLs:

- URL is an abbreviation for Uniform Resource Locator.
- An URL is a form of string that helps to find a resource on the World Wide Web (WWW).
- The URL class is the simplest way for a java program to locate and retrieve data from the network.
- We 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, simply tell java the URL and it gets the data.
- URL has two components:
    1. The protocol required to access the resource.
    2. The location of the resource.
- Example : http://www.example.com/index.html

URL Class:

URL Class represents a Uniform Resourse Locator, a pointer to a “resource” on the world wide web.

Absolute URL:

Contains all of the information necessary to reach the resource.

Example: http://www.example.com/xyz.html

Relative URL:

A relative URL usually contains the path and, if present, the resource; it does not include the scheme or server.

Concatenating the absolute and relative URLs effectively specifies the target’s “complete URL.”

Example: /xyz.html

Components of URL:

1. **Protocol**: In this case, http is the protocol.
2. **Hostname**:
   - Name of the machine on which the resource lives.
   - In this case, www.studytonight.com is the Host name.
3. **Port Number**:
   - It is an optional attribute.
4. **File Name or directory name**: In this case, index.html is the file name.

Example Of Base, Realtive, Resolved URL:

```
// Define a base URL
URL baseURL = new
URL("https://www.example.com/");

// Define a relative URL
String relativeURL = "foo/bar.html";
```

```
// Resolve the relative URL against the base URL to get the absolute URL
URL resolvedURL = new URL(baseURL,relativeURL);

// Print the results
System.out.println("Base URL: " + baseURL);
System.out.println("Relative URL: " + relativeURL);
System.out.println("Resolved URL: " + resolvedURL);
```

Creating URL:

(a) creates a URL with string url representation

- URL url1 = new URL("https://www.google.com/search?q=computer+engineer&sclient=gws-wiz");

(b) creates a URL with a protocol, hostname,and path

- URL url2 = new URL("http", "www.google.com","/contact/");

(b) creates a URL with a protocol, hostname, port and path

- URL url2 = new URL("http", "www.google.com",8008,"/contact/");

(b) creates a URL with a url and string relative

- URL u1 = new URL("http://www.ibiblio.org/javafaq/index.html");
- URL u2 = new URL (u1, "mailinglists.html");

Retrieving Data from a URL:

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

openStream():

- The openStream() method connects to the resource referenced by the URL, performs any necessary handshaking between the client and the server, and return an InputStream form which data can be read.

- Clearly, InputStream returns the raw data, that is, characters are read as ASCII, images in binary, and HTML as raw html without headers.
- Java program can read from a URL using the openStream() method.
- **Syntax:**

  Public InputStream openStream() throws IOException

openConnection():

- The openConnection() method opens a socket to the specified URL and return a URLConnection object.
- Unlike openStream, it also decodes the headers, meta tags of HTML
- unlike openStream, openConnection lets you perform wrtie, or query operations on server
- **Syntax:**

  public URLConnection openConnection()throws IOException

getContent():

- method retrives the data referenced by the URL and tries to make it into some type of object.
- The type of object depends upon the content it is asked to fetch
- for e.g. if the url refers to HTML/ASCII then the Object type will be InputStream, similarly, if the url refers to image such as GIF/JPG/JPEG then this method will return ImageProcuder
- **Syntax**:

  public Object getContent() throws IOException

Example: openStream():

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

public class OpenStreamDemo {
    public static void main(String[] args) {
        try {
            URL url = new URL("https://www.facebook.com/");
```

```
                InputStream stream = url.openStream();
                int i;
                System.out.println("running");
                while ((i = stream.read()) != -1) {
                    System.out.print((char) i);
                }
            } catch (IOException e) {
                System.out.println(e);
            }
        }
}
```

Example: openConnection():

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

public class OpenConnectionDemo {
    public static void main(String[] args) {
        try {
```

```
            URL url = new
URL("https://www.youtube.com/");
            URLConnection urlcon =
url.openConnection();
            InputStream stream =
urlcon.getInputStream();
            int i;
            System.out.println("running");
            while ((i = stream.read()) != -1) {
                System.out.print((char) i);
            }
        } catch (IOException e) {
            System.out.println(e);
        }
    }
}
```

Example: getcontent():

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

```
public class GetContentDemo {
	public static void main(String[] args) {
		try {
			URL url = new
URL("https://unsplash.com/photos/wRdSItfeMLs");
			Object o = url.getContent();
			System.out.print(o.getClass().getName());
		} catch (IOException e) {
			System.out.println(e);
		}
	}
}
```

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

E.g.

http://www.ibiblio.org:8080/javafaq/books/jnp/index.html? isbn=1565922069#toc

scheme = http,

authority = www.ibiblio.org:8080,

path = /javafaq/books/jnp/index.html, fragment identifier = toc,

query string = isbn=1565922069

Method to access pieces:

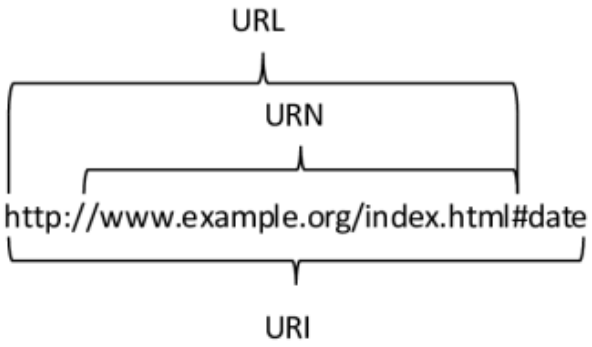
The methods of URL class are given below:

- Public String getProtocol(): It return the protocol of the URL.
- Public String getHost(): It return the host name of the URL.
- Public String getPort(): It return the port number of the URL.
- Public String getFile(): It return the file name of the URL.
- Public String getAuthority(): It returns the authority of the URL.
- Public String getRef(): It return the anchor or reference of the URL.
- Public String getQuery(): It return the query string of the URL.
- Public String getpath(): It returns the path specified for a specific resource

```
import java.net.*;

public class MethodToAccess {
	public static void main(String[] args) {
		try {
			URL url = new
URL("https://www.youtube.com/results?search_query=abc");

			System.out.println("Protocol: " + url.getProtocol());
			System.out.println("Host Name: " + url.getHost());
			System.out.println("Port Number: " + url.getPort());
			System.out.println("Default Port Number: " +
url.getDefaultPort());
			System.out.println("Query String: " +
url.getQuery());
			System.out.println("Path: " + url.getPath());
			System.out.println("File: " + url.getFile());

		} catch (Exception e) {
			System.out.println(e);
		}
	}
}
```

URI:

- A URI or Uniform Resource Identifier is a string identifier that refers to a resource on the internet.
- It is a string of characters that is used to identify any resource on the internet using location, name, or both.
- A URI has two subsets; URL (Uniform Resource Locator) and URN (Uniform Resource Number). If it contains only a name, it means it is not a URL. Instead of directly URI, we mostly see the URL and URN in the real world.



Examples of URL and URN:

URL: ftp://ftp.is.co.za/rfc/rfc1808.txt

URL: http://www.ietf.org/rfc/rfc2396.txt

URL: ldap://[2001:db8::7]/c=GB?objectClass?one URL: mailto:John.Doe@example.com URL: news:comp.infosystems.www.servers.unix

URL: telnet://192.0.2.16:80/

URN (not URL):
urn:oasis:names:specification:docbook:dtd:xml:4.1.2

URI Syntax :

scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]

1. **scheme :** represents a protocol that a browser must need to use to request the resource.

   abc://admin:admin@example.org:1234/path/data
   ?key=value&key2=value2#fragid1

2. **authority** : Authority component is made of several components- authentication section, a host and optional port number preceded by a ‘:’. Authentication section includes username and password.

   abc://**admin:admin@example.org:1234**/path/data
   ?key=value&key2=value2#fragid1

3. **path** : The path represents a string containing the address within the server to the resource.
   abc://admin:admin@example.org:1234/**path/data**
   ?key=value&key2=value2#fragid1

4. **query** : Query represent a nonhierarchical data usually the query used for searching of a particular resource. They are separated by a '?' from the preceding part.

   abc://admin:admin@example.org:1234/path/data
   ?key=value&key2=value2#fragid1

5. **fragment** : Fragments are used to identify secondary resources as headings or subheadings within a page etc.

   abc://admin:admin@example.org:1234/path/data
   ?key=value&key2=value2**#fragid1**

URI Constructors:

1.URI(String str):

Syntax :public URI(String str)

throws URISyntaxException

Parameters :

str : String to be parsed into URI

2.URI(String scheme, String ssp, String fragment) :

Syntax :public URI(String scheme, String ssp, String fragment)

throws URISyntaxException

Parameters :

scheme : scheme name

ssp : scheme-specific part

fragment : fragment part

3.URI(String scheme, String userInfo, String host, int port, String path,

String query, String fragment):

Syntax :public URI(String scheme, String userInfo, String host, int port,

String path, String query, String fragment)

Parameters :

scheme : string representing scheme

userInfo : userinfo of URI

host : host component of URI

port : listening port number

path : path of URI

query : String representing the query part

fragment :optional fragment

4.URI(String scheme, String host, String path, String fragment):

Syntax :public URI(String scheme, String host, String path, String fragment)

Parameters :

scheme : string representing scheme

host : host component of URI

path : path of URI

fragment :optional fragment

5.URI(String scheme, String authority, String path, String query, String fragment):

Syntax :public URI(String scheme, String authority, String path,

String query, String fragment)

Parameters :

scheme : string representing scheme

authority : authority

path : path of URI

query : String representing the query part

Constructing URI:

```
// Constructor to create a new URI
// by parsing the string
URI uriBase = new URI(uribase);

// create() method
URI uri = URI.create(str);
```

Resolving Relative URIs:

```
String uribase = "https://www.example.org/";
String urirelative = "languages/../java";
String str = "https://www.google.co.in/?gws_rd=ssl#"+""
      + "q=networking+in+java"+""
      +"&spf=1496918039682";

// Constructor to create a new URI
// by parsing the string
URI uriBase = new URI(uribase);

// create() method
URI uri = URI.create(str);

// toString() method
System.out.println("Base URI = " + uriBase.toString());

URI uriRelative = new URI(urirelative);
System.out.println("Relative URI = " + uriRelative.toString());

// resolve() method
URI uriResolved = uriBase.resolve(uriRelative);
```

```
System.out.println("Resolved URI = " + uriResolved.toString());
```

X-WWW-FORM-URLENCODED:

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

  The capital letters A-Z

  The Lowercase letters a-z

  The digits 0-9

  The punctuation characters - _ . ! ~ * ‘ (and ,)

- The characters: / & ? @ # ; $ + = and % may also be used, but only for their specified purposes.
- If these characters occur as part of a filename, they and all other characters should be encoded.
- The encoding is very simple.

URL Encoder:

- URL encoder takes a URL as input and encodes it by replacing certain characters with a sequence of characters that represent them in the ASCII character set.
- This is done to ensure that the URL can be safely transmitted over the internet without causing issues such as data corruption or misinterpretation.
- For example, a URL containing a space character would be encoded as "%20" using URL encoder.
- In java 1.3 and earlier, the java.net.URLEncoder class contains a single static method called encoder() that encodes a String according to these rules:

  public static String encode(String s)
- this method always uses the defult encoding of the platform on which it runs, so it produce different results on different systems.
- As a result, java 1.4 deprecates this method and replace it with a method that require you to specify the encoding:

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

Example:

```
import java.net.URLEncoder;

public class UrlEncoderDecoderExample {
  public static void main(String[] args) {
    try {
      // URL to be encoded
      String url = "https://www.example.com/search?q=query string";

      // Encode the URL
      String encodedUrl = URLEncoder.encode(url, "UTF-8");

      // Print the encoded URL
      System.out.println(encodedUrl); // Output:
https%3A%2F%2Fwww.example.com%2Fsearch%3Fq%3Dquery+strin
g

    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

URL Decoder:

URL decoder takes an encoded URL as input and decodes it by converting any encoded characters back to their original form.

The corresponding URLDecoder class has two static methods that decode strings encoded in x-www-form-encoded format.

public static String decode(String s)

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

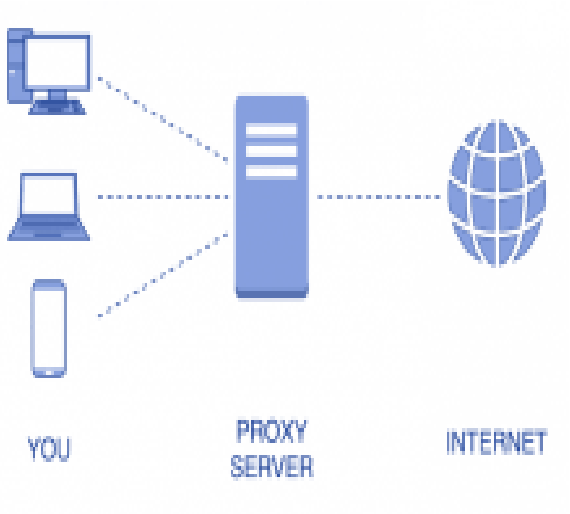
Example:

```
package chapter3;

import java.net.URLEncoder;
import java.net.URLDecoder;

public class UrlEncoderDecoderExample {
  public static void main(String[] args) {
    try {
      // URL to be encoded
      String url = "https://www.example.com/search?q=query string";

      // Encode the URL
      String encodedUrl = URLEncoder.encode(url, "UTF-8");
```

```
        // Print the encoded URL
        System.out.println(encodedUrl); // Output:
https%3A%2F%2Fwww.example.com%2Fsearch%3Fq%3Dquery+strin
g

        // Decode the URL
        String decodedUrl = URLDecoder.decode(encodedUrl, "UTF-8");

        // Print the decoded URL
        System.out.println(decodedUrl); // Output:
https://www.example.com/search?q=query string

    } catch (Exception e) {
        e.printStackTrace();
    }
  }
}
```

Proxies:

- A proxy is an intermediary server or software application that acts as a gateway between a local network and the internet.
- When a user requests a resource such as a web page, the request is sent to the proxy server instead of the destination server.
- The proxy server then retrieves the resource on behalf of the user and forwards it back to the user.



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 server.
- If you are using a HTTP proxy, you have to set host address and port of proxy server, to do so, 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 defult is 80).

https.proxyHost: The hostname or IP address for the HTTP proxy server.

Example: System.setProperty("http.proxyHost", "192.168.254.252");

http.proxyPort: - The port number for the HTTP proxy server. This system property is used in conjunction with

Example: System.setProperty("http.proxyPort", "8080");

http.nonProxyHosts:

Is a Java system property that specifies a list of hostnames or IP addresses that should not be proxied when making HTTP requests.

his property is useful when you have a proxy server configured, but you need to connect to certain hosts or addresses directly without going through the proxy.

System.setProperty("http.nonProxyHosts", "java.oracal.com| xml.oracal.com");

Here, whenever accessing java.oracal.com or xml.oracal.com the request won't go via proxy server.

The ProxyClass:

- The Proxy class in Java is a way to represent a proxy server in a platform-independent way.
- It allows you to configure and use a proxy server for making network connections.
- It allow you to choose different proxy server for different remote host.

- The Proxy class has two static methods that allow you to create instances of the Proxy class:

1.Proxy.Type proxyType = Proxy.Type.HTTP;

This sets the type of the proxy server you want to use.

There are three types of proxy servers: HTTP, SOCKS, and DIRECT. In this example, we are setting the type to HTTP.

2.SocketAddress proxyAddress = new InetSocketAddress("192.168.1.1", 8080);

This sets the address and port number of the proxy server. In this example, we are setting the address to "192.168.1.1" and the port number to 8080.

```
// Create a Proxy object with the desired type and address
Proxy proxy = new Proxy(Proxy.Type.HTTP, new InetSocketAddress("192.168.1.1", 8080));

// Create a URL object for the target URL
URL url = new URL("http://www.example.com");

// Open a connection to the target URL using the Proxy object
```

```
HttpURLConnection conn = (HttpURLConnection) url.openConnection(proxy);
```

The ProxySelector Class:

- The ProxySelector class is part of the Java networking API and represents a strategy for selecting a proxy server to use for a network connection.
- It provides an interface for implementing custom proxy selection algorithms based on various criteria such as the type of connection, the destination host, and the user's preferences.

The ProxySelector class has two main methods:

1.**List<Proxy> select(URI uri)**

This method is called by the URLConnection class when it needs to establish a connection to a remote server. The method takes a URI object representing the target server and returns a list of Proxy objects representing the available proxy servers that can be used to connect to the target server.

2.**void connectFailed(URI uri, SocketAddress sa, IOException ioe)**

This method is called by the URLConnection class when a connection attempt through a proxy server has failed.

Accessing Password-Protected Site:

- Java URL class can access sites that use HTTP authentication, through you will of course need to tell it what username and password to use.
- Cookie-based authentication is more challenging, not least because this varies a lot from one site to another.

Authenticator Class: Authenticator()

- 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

- **Methods**

```
Authenticator.setDefault(new DialogAuthenticator());
// Sets the authenticator to be used when a HTTP server requires authentication.
```

PasswordAuthentication Class:

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

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

- Each is **accessed** via a getter method:

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

Example:

```
String userName = “user";
char[] password = { ‘p', ‘a', ‘s', ‘s' };
PasswordAuthentication auth = new PasswordAuthentication(userName, password);
System.out.println("UserName: " + auth.getUserName());
System.out.println( "Password: "   + passwordAuthentication.getPassword());
```

JPasswordField Class:

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

Example:

```
import javax.swing.*;

public class JPasswordFieldExample {

    public static void main (String[]args) {
        JPasswordField passwordField = new
JPasswordField();
        JLabel passwordLabel = new JLabel("Password: ");
```

```
        JFrame f = new JFrame("Password Field Example");
        passwordLabel.setBounds(20, 100, 80, 30);
        passwordField.setBounds(100, 100, 100, 30);
        f.add(passwordField);
        f.add(passwordLabel);
        f.setLayout(null);
        f.setSize(300, 300);
        f.setVisible(true);

    }

}
```