The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases and might use technology no longer available.
See Java Language Changes for a summary of updated language features in Java SE 9 and subsequent releases.
See JDK Release Notes for information about new features, enhancements, and removed or deprecated options for all JDK releases.

Reading from and Writing to a URLConnection

The `URLConnection` class contains many methods that let you communicate with the URL over the network. `URLConnection` is an HTTP-centric class; that is, many of its methods are useful only when you are working with HTTP URLs. However, most URL protocols allow you to read from and write to the connection. This section describes both functions.

Reading from a URLConnection

The following program performs the same function as the `URLReader` program shown in Reading Directly from a URL.

However, rather than getting an input stream directly from the URL, this program explicitly retrieves a `URLConnection` object and gets an input stream from the connection. The connection is opened implicitly by calling `getInputStream`. Then, like `URLReader`, this program creates a `BufferedReader` on the input stream and reads from it. The bold statements highlight the differences between this example and the previous:

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

public class URLConnectionReader {
    public static void main(String[] args) throws Exception {
        URL oracle = new URL("http://www.oracle.com/");
        URLConnection yc = oracle.openConnection();
        BufferedReader in = new BufferedReader(new InputStreamReader(
                                    yc.getInputStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);
        in.close();
    }
}
```

The output from this program is identical to the output from the program that opens a stream directly from the URL. You can use either way to read from a URL. However, reading from a `URLConnection` instead of reading directly from a URL might be more useful. This is because you can use the `URLConnection` object for other tasks (like writing to the URL) at the same time.

Again, if the program hangs or you see an error message, you may have to set the proxy host so that the program can find the Oracle server.

Writing to a URLConnection

Many HTML pages contain *forms* — text fields and other GUI objects that let you enter data to send to the server. After you type in the required information and initiate the query by clicking a button, your Web browser writes the data to the URL over the network. At the other end the server receives the data, processes it, and then sends you a response, usually in the form of a new HTML page.

Many of these HTML forms use the HTTP POST METHOD to send data to the server. Thus writing to a URL is often called *posting to a URL*. The server recognizes the POST request and reads the data sent from the client.

For a Java program to interact with a server-side process it simply must be able to write to a URL, thus providing data to the server. It can do this by following these steps:

The servlet running in a container reads from its InputStream, reverses the string, and writes it to its OutputStream. The servlet requires input of the form `string=string_to_reverse`, where `string_to_reverse` is the string whose characters you want displayed in reverse order.

Here's an example program that runs the `ReverseServlet` over the network through a `URLConnection`:

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

public class Reverse {
    public static void main(String[] args) throws Exception {

        if (args.length != 2) {
            System.err.println("Usage:  java Reverse "
                + "http://<location of your servlet/script>"
                + " string_to_reverse");
            System.exit(1);
        }

        String stringToReverse = URLEncoder.encode(args[1], "UTF-8");

        URL url = new URL(args[0]);
        URLConnection connection = url.openConnection();
        connection.setDoOutput(true);

        OutputStreamWriter out = new OutputStreamWriter(
                                       connection.getOutputStream());
        out.write("string=" + stringToReverse);
        out.close();

        BufferedReader in = new BufferedReader(
                                   new InputStreamReader(
                                   connection.getInputStream()));
        String decodedString;
        while ((decodedString = in.readLine()) != null) {
            System.out.println(decodedString);
        }
        in.close();
    }
}
```

Let's examine the program and see how it works. First, the program processes its command-line arguments:

```
if (args.length != 2) {
    System.err.println("Usage:  java Reverse "
        + "http://<location of your servlet/script>"
        + " string_to_reverse");
    System.exit(1);
}

String stringToReverse = URLEncoder.encode(args[1], "UTF-8");
```

These statements ensure that the user provides two and only two command-line arguments to the program. The command-line arguments are the location of the `ReverseServlet` and the string that will be reversed. It may contain spaces or other non-alphanumeric characters. These characters must be encoded because the string is processed on its way to the server. The `URLEncoder` class methods encode the characters.

Next, the program creates the `URL` object, and sets the connection so that it can write to it:

Next, the program writes the required information to the output stream and closes the stream:

```
out.write("string=" + stringToReverse);
out.close();
```

This code writes to the output stream using the `write` method. So you can see that writing data to a URL is as easy as writing data to a stream. The data written to the output stream on the client side is the input for the servlet on the server side. The `Reverse` program constructs the input in the form required by the script by prepending `string=` to the encoded string to be reversed.

The servlet reads the information you write, performs a reverse operation on the string value, and then sends this back to you. You now need to read the string the server has sent back. The `Reverse` program does it like this:

```
BufferedReader in = new BufferedReader(
                            new InputStreamReader(
                            connection.getInputStream()));
String decodedString;
while ((decodedString = in.readLine()) != null) {
    System.out.println(decodedString);
}
in.close();
```

If your `ReverseServlet` is located at `http://example.com/servlet/ReverseServlet`, then when you run the `Reverse` program using

```
http://example.com/servlet/ReverseServlet "Reverse Me"
```

as the argument (including the double quote marks), you should see this output:

```
Reverse Me
 reversed is:
eM esreveR
```

Previous page: Connecting to a URL
Next page: All About Sockets