Code Assignment 01 - Xurl Java Code

Exercise 0 - Testing with Telnet

For testing-purposes, you can try to use the Unix command 'telnet' to establish a TCP connection to any port on a server. For example:

```
telnet www.enseignement.polytechnique.fr 80
```

will connect you to the WWW-server at Ecole Polytechnique.

You can then type commands ("GET / HTTP/1.1" ...) on the keyboard and inspect the exact replies that the WWW-server generates. This is a good exercise to do, and a good tool for when debugging later. And, it works for all ASCII-based protocols running over TCP - such as, for example, IRC, SMTP, POP and IMAP. One of your teachers is occasionally seen reading his email using telnet to port 110 ... Unless you suffer from the same disorder, it is advisable to have first a look at the `telnet` man page.

Exercise 1 - Parsing an URL

As you will have to parse an URL, you must first write a small equivalent of the java.net.URL class. This class must be named `MyURL`, and in the default package.

Your `MyURL` class MUST implement the following constructor:

- `MyURL(String url)` that parses the given String and stores the corresponding elements of the URL (i.e., the protocol, hostname, ...)

In any case of unrecognized syntax in `url` argument the constructor must throw an **`IllegalArgumentException`**, carrying a descriptive message indicating what went wrong.

The recognized URL syntax is of the form:

```
<protocol>://<hostname>[:<port>]/<path>
```

For simplification, consider that:

- `<protocol>` and `<hostname>` are non-empty strings, which do not contain '/' nor ':',
- `<port>` is any integer, and
- `<path>` is any (even empty) string.

Thus, this `MyURL` class **shall not** check wheter:

- `<protocol>` is a recognized protocol,

- `<hostname>` corresponds to an existing server,
- `<port>` is a valid port number.

Note also that an url such as `http://www.google.com`" MUST NOT be recognized as a valid URL, as a trailing '/' is expected.

Your `MyURL` class MUST, at least, implement the following 4 methods `getProtocol()`, `getHost()`, `getPort()` and `getPath()`, performing the same operations as those specified for the class `java.net.URL`.

Note that, for our implementation, the value returned by `getPath()` always contains a leading '/', and is never the empty String.

For instance, for the input url "http://www.google.com/",
`getProtocol()` returns "http",
`getHost()` returns "www.google.com",
`getPort()` returns -1, as an integer,
`getPath()` returns "/".

A couple of hints might not go amiss, to help with programming:

- To extract useful tokens from <url>, you might look at the method split, of the class String.
- You may also consider using the powerful Pattern class for this.

You MUST submit your file **`MyURL.java`** through the upload form here. You must then check that your code is validated and, if necessary, you may submit again, until the deadline. You will then have to do the same for the following exercices.

Exercise 2:
Implement 'Xurl', satisfying the following functional specification. Typing:

java Xurl <url>

will download the file, indicated by way of <url>, into the local directory on your computer. If the file has moved to a different location, then Xurl will automatically download the file from its new location.

If the URL doesn't exist, or if there is another error, a graceful message should be displayed to the user.

<url> is a classic URL, of the following form:

http://<hostname>[:<port>]/<path>

For example:

http://www.example.com/foobar.html
http://www.example.com:8080/foo/bar/42.pdf

In case the path is ended by a / (including the case of / only), use a conventional filename, say 'index'.

Exercise 2a

Before hitting the keyboard and firing up your favorite text editor to write a program, draw a state machine — a state machine as detailed as you can.
Your program first reads the command line argument <url>. What can happen? Well, it can be that no argument is given, or that it doesn't satisfy the format given above — or, that it is a valid looking (read: it respects the format) URL. Depending of which of these happens, your program will enter a different state — in which it can either send or receive a message, or do something else as a result of internal processing.
Draw up your state-machine — annotate and/or modify it as you think through the exercise, and eventually implement Xurl. There is a "Homework" part at the end of this note, which you are required to complete, and for which keeping your state-machine diagram up-to-date is going to be a huge help for you.
Remember whilst developing this state machine that it is unfortunate if your program gets stuck in a state "until the end of the universe" — all states (except for a state "End", which terminates the program) must have transitions leading out of it. Also think about this: how do you avoid getting stuck in a given state "until the end of the universe", given that you are connecting to a server which may or may not behave as expected? This must be reflected in your state-machine, and in your program.

Exercise 2b

Implement Xurl according to the specification that you have developed — and test that it functions correctly, for example against the servers at Ecole Polytechnique. For now, don't worry about 3xx status code. In such a cas the document shall not be retrieved from its new location and no file is to be written.

A couple of hints might not go amiss, to help with programming:

- To open a client TCP connection in Java, you have to create a 'connected' Socket object (i.e., an object of the class Socket).Understanding the semantics of this object, choosing an appropriate constructor and setup calls is an essential part of the requirements for network programming — look at the Java API documentation, if you do not already know what the different constructors are. You should start to think about what is meant by binding, connecting and closing of a connection (and, know that this will be discussed in detail in Lecture 02, and definitely be part of the exam).

- Once you have a 'connected' Socket object, you can respectively send/read to/from the server through its OutputStream/InputStream. It is convenient to wrap these streams in, respectively, a PrintStream and a BufferedReader (using an intermediate InputStreamReader also).

Exercise 2c (optional)

Now, full implementaton of HTTP features is required.

~~The DSI at Polytechnique thinks that Exercise 1c is waaaaay too easy for us. Fortunately, after thinking long and hard about how to make life more challenging, they came up with Kuzh, our all-time beloved proxy.~~

In some (strange) networks, the administator requires the http connection go through a proxy. Using http proxy is also a common way to hide our real IP address recorded by the server. Try typing "http proxy list" in Google should bring you lots of http proxy freely available.

So, obviously, the next step is to extend your implementation to support a proxy. Fortunately, given that you started out drawing a clear state diagram, this is really not hard to do.

1. Rather than establish the TCP connection with the <hostname>, you will want to establish the TCP connection with the proxy
2. You will then send the request with a slight modification to the first line, of the form:

    GET <url> HTTP/1.1

This, since the proxy will have to make the entire query (connect to the server with the given <hostname>, then fetch the right filename) all over again.

So, your program should be able to take the following argument for when connecting by way of a proxy:

java Xurl <url> [proxyName [proxyPort]]

Of course, the previous format of arguments (without proxy) should work also:

java Xurl <url>

Thus, you should be able to accept both of:

java Xurl http://www.google.com/ kuzh 8080

java Xurl http://enseignement.polytechnique.fr/

Again, before starting to code, think about the impact that this will have on your state-machine, and update the state-machine accordingly to accommodate this new option. You should think about what the minimum required number of additional states are for accommodating a proxy option.

Remember: the more states, and the more transitions, the more code will you have to write, debug (and, if you end up doing software for a living) maintain and support.