Overview

In this part, you will implement another web server – HTTPAsk. This is a web server that uses the client you did in Task 1. When you send an HTTP request to HTTPAsk, you provide a hostname, a port number, and optionally a string as parameters for the request.

When HTTPAsk receives the HTTP request, it will call the method TCPClient.askServer, (which you wrote for Task 1, remember?), and return the output as an HTTP response. This may seem confusing, but it is really very simple: instead of running TCPAsk from the command line, you will build a web server that runs TCPAsk for you, and presents the result as a web page (an HTTP response).

What you will learn here is to:

- Read and analyse an HTTP GET request, and extract a query string from it.
- Launch an application from your server, where the application provides the data that the server returns in response to the HTTP get.

Task

Your job is to implement a class called HTTPAsk. It's "main" method implements the server. It takes one argument: the port number. So if you want your server to run on port 8888, you would start it in the following way:

```
$ java HTTPAsk 8888
```

You will be provided with a file with a template for the HTTPAsk class. There is not much content in it though, but your task is to fill in the missing Java code *in this file*.

The requirements for HTTPAsk are the same as for HTTPEcho in Task 2: it should run in an infinite loop, but does not need to be multithreaded.

Instructions and Tips

At this point, you should have some code that you can re-use. From Task 1, you should be able to import the TCPClient class and use it without any modifications. From Task 2, you have the skeleton for an HTTP server, and you have code to generate an HTTP response.

What you need to work on in this task is mainly to process the HTTP request. You should extract the hostname, port, and string parameters from it. Your HTTP server only needs to recognise GET requests. In a GET request, parameters are sent in the URI component (the second component) of the first line of the request. The URI component comes from the URL that was used to navigate to the server.

For instance, assume that you give the following URL to your web browser:

```
http://hostname.domain/ask?hostname=time.nist.gov&port=13
```

This would result in the following GET request:

```
GET /ask?hostname=time.nist.gov&port=13 HTTP/1.1
Host: hostname.domain
```

The "/ask" part of the URI tells the HTTPAsk server to set up a TCP connection to a remote server (specified by the "hostname" parameter) and a port ("port" parameter), and return the result. Think of this as a command to the server!

You need to write the code that validates the request, "picks apart" the URI component of the request and extracts the parameters and their values. This involves a certain amount of string processing. There are (at least) two ways you could do this. One way is to use the URL class (Länkar till en externa sida.), which has many different methods for processing URL strings. Another way is to take care of the string processing yourself – you will probably find that the String "split" method (Länkar till en externa sida.) is very handy for this. It is up to you!

The query parameters your HTTPAsk server should recognize are shown in the following table:

Parameter	**Meaning**	**Example**
hostname	Domain name for host	hostname=whois.iis.se
port	Port number	port=43
string	Data to send to server	string=kth.se

Resources

For this task, you will be provided with the following:

- HTTPAsk.java – Skeleton declaration for the HTTPAsk class.

The file are available in a zip archive called "task3.zip". In this zip archive, there is a directory called "task3" with the file in it. So, when you unzip the archive, the "task3" directory will be created, and in this directory you will find the Java file. *This is important, because you are expected to submit your solutions in a zip archive with exactly the same structure!*

You can find the zip archive here ladda ner.