

Task 1: TCPAsk

Due 28 Feb by 17:00 **Points** 0 **Submitting** a file upload **File types** zip
Available 28 Jan at 17:00 - 28 Feb at 17:00 about 1 month

This assignment was locked 28 Feb at 17:00.

Overview

In this task, you will learn how to:

- Create TCP sockets, and use them to send and receive data
- Design the client side of client-server communication
- Deal with errors that can happen during the communication

The task is a programming assignment to implement a simple TCP client, as a Java class. The class is called `TCPClient`, and works in a straight-forward manner:

1. Open a TCP connection to a server at a given host address and port number.
2. Send data to the server.
3. Take the data that the server sends back in response, and return that as the result.

The TCPClient Class

To use the `TCPClient` class, an instance should first be created. The constructor has no parameters:

```
public TCPClient()
```

TCP provides the communication service of bidirectional transfer of streams of bytes. Hence, the `TCPClient` class has a method to send bytes to a server and receive bytes in return. This method is called `askServer`, and its specification is as follows:

```
public byte[] askServer(String hostname, int port, byte[] bytesToServer) throws IOException
```

The `hostname` parameter is the domain name of the server to which the client should connect. The `port` parameter is the TCP port number on the server. The `bytesToServer` parameter is a byte array with the data to send to the server. So, for instance, to connect to KTH's web server and send the data in the byte array `"webdata"`, a program could do the following:

```
byte[] webdata = new byte[] ...  
  
TCPClient webClient = new TCPClient();  
byte[] response = webClient.askServer("www.kth.se", 80, webdata);
```

Task

Your task is straight-forward: implement the TCPClient class.

To help you with development and testing, you will also get an application program that uses the TCPClient class. The application is called TCPAsk, which is where the name of this assignment comes from.

Instructions and Tips

Socket I/O and Wrappers

You should use the [Socket class](https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html) [_](https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html) to create the client's socket. In Java, [InputStream](https://docs.oracle.com/javase/7/docs/api/java/io/InputStream.html) [_](https://docs.oracle.com/javase/7/docs/api/java/io/InputStream.html) and [OutputStream](https://docs.oracle.com/javase/7/docs/api/java/io/OutputStream.html) [_](https://docs.oracle.com/javase/7/docs/api/java/io/OutputStream.html) are the basic classes for I/O, which read and write "raw" binary data in the form of byte arrays. The slides from [Kurose-Ross 5e](https://canvas.kth.se/courses/31590/files/5060007/download?download_frd=1) [↓](https://canvas.kth.se/courses/31590/files/5060007/download?download_frd=1) suggest to use additional "wrappers" around the socket's InputStream and OutputStream. Those wrappers add data processing and buffering to the basic InputStream and OutputStream, and automatically convert between text and binary data. Using wrappers like that is common in Java I/O. However, in this assigned we want to transmit binary data transparently without additional data processing, so we have no use for wrappers. Furthermore, it would not be a general solution; it would only work for text-based applications, and not for binary transfers.

It is requirement in this assignment that you use InputStream and OutputStream classes for "raw" binary I/O with byte arrays. You are not allowed to use wrappers such as InputStreamReader/OutputStreamWriter and BufferedReader/BufferedWriter. See the slides from the [Socket Programming Lecture](https://canvas.kth.se/courses/31590/files/5153647/download?download_frd=1) [↓](https://canvas.kth.se/courses/31590/files/5153647/download?download_frd=1) for more information, and for examples of how to use the [Socket class](https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html) [_](https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html) for byte I/O.

Receiver Buffers

TCPClient's askServer method waits until all data has been received from the server before it returns. In other words, askServer reads data from the TCP connection until the connection is closed. Then the question is, how does askServer store all that data internally? Clearly askServer needs to have a receive buffer where it stores the data it receives from the server. A simple (but incorrect) solution would be to use a byte array of fixed size for storing the data, and make that byte array so large that it is highly unlikely that it will overflow. This is not a good solution, for several reasons. First, it means there is an assumption built into the program about how much data the server could send. To design a program in that way that is not good programming. Second, the program would occupy a lot of memory resources on the computer, resources that the program does not use, so it is not efficient usage of the computer.

A better solution is to store the data from the server in a byte array that grows dynamically (and automatically) when more data is received. Java has a class called `ByteArrayOutputStream` that does exactly this. Use this class to implement the receive buffer for `askServer`. You probably need two levels of buffer. One smaller buffer (byte array) of fixed size that you use in the read operations on the socket, and a `ByteArrayOutputStream` object where you store the data received so far.

TCPAsk

TCPAsk is a Java application that uses the `TCPClient` class to communicate with a server. TCPAsk has two mandatory parameters given on the command line: the host and the port to which TCPAsk should connect. TCPAsk also takes an optional string as parameter. This string is appended with a newline character (linefeed '\n') before being converted to a byte array and sent as data to the server. So TCPAsk connects to the server, sends the optional data over the connection to the server (if there is any), decodes the data received from the server into text, and prints the text as program output.

For example, assume we want to contact the "daytime" server at "time.nist.gov". The Daytime protocol is a standardised protocol for asking a Daytime server about the current date and time. Daytime runs at TCP port 13 by default. So we could use TCPAsk in the this way:

```
$ java TCPAsk time.nist.gov 13
time.nist.gov:13 says:

59604 22-01-25 18:03:08 00 0 0 626.1 UTC(NIST) *
```

The following example uses another Internet protocol, the "whois" protocol, which is for making queries about resources on the Internet. The whois protocol uses port 43. Here we use the third optional argument to TCPAsk to pass a string to the whois server asking for information about the domain name "kth.se".

```
$ java TCPAsk whois.iis.se 43 kth.se
whois.iis.se:43 says:
...
```

(The output is quite lengthy, so try this for yourself and check the output!)

Note: TCPAsk is a text application, meaning that it sends encoded text to the server, and decodes the result from the server into text. This makes it easier to use from the command line program, but it also means that we cannot use TCPAsk to communicate with servers that use other data formats than text.

Testing

You will need servers to test against. Here are some suggestions.

Protocol	Server name	Port	Arguments (data sent to server)	Comment
				Public server at NIST

Daytime	time.nist.gov	13	None	Public server at NIST, of Standards and Tech government agency. N has limitations for how query it (at most once See https://tf.nist.gov (https://tf.nist.gov/tf-cg
Daytime	java.lab.ssvl.kth.se	13	None	KTH server.
Whois	whois.iis.se	43	String (a domain name, an IP address or an AS number)	Public server at the Th Foundation (Internetst
Whois	whois.internic.net	43	String (a domain name, an IP address or an AS number)	Public server at ICANN Corporation For Assign Numbers.


Use public servers with care. Abuse may be detected and reported.

Resources

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

- TCPAsk.java – the TCPAsk Java program
- tcpclient/TCPClient.java – Skeleton declaration of the tcpclient.TCPClient class.

The files are available in a zip archive called "task1.zip". In this zip archive, the files are stored in a directory called "task1". So, when you unzip the archive, the "task1" directory will be created, and in this directory you will find the two Java files. *This is important, because you are expected to submit your files in a zip archive with exactly the same structure!*

[You can find the zip archive here: task1.zip.](https://canvas.kth.se/courses/31590/files/5150004/download?download_frd=1) 
(https://canvas.kth.se/courses/31590/files/5150004/download?download_frd=1)

Submission

Your submission should be a zip file named "task1.zip", containing the same files and directory structure as in the template. In other words, you should return a zip archive like the one you received

with templates, with the same file names and the same directory structure.

Submit by uploading your "task1.zip" file below. You can submit as many times as you like before the due date.

Submit in the Correct Format

Make sure to verify that your submission follow the required format! You can run the unzip command

line program to check your submission. The output should be something like this (depending on your platform):

```
$ unzip -l task1.zip
Archive: task1.zip
  Length      Date    Time    Name
-----
   288  01-26-2018  10:02  task1/tcpclient/TCPClient.java
  1254  02-05-2018  13:03  task1/TCPAsk.java
-----
  1542                  2 files
```

If you are using zip as a command line tool, the easiest way to make sure that you get the right format for the zip archive is to *go to the directory where the "task1" directory is*. (In other words, don't position yourself inside the task1 directory. Instead, you should be one step above.)

Then run the following:

```
zip task1.zip task1/tcpclient/TCPClient.java task1/TCPAsk.java
```

Mac users that have problems with extraneous files in the archive can try the following:

```
COPYFILE_DISABLE=1 zip task1.zip task1/tcpclient/TCPClient.java task1/TCPAsk.java
```

We use automatic tools for the grading. If your submission does not follow this format, the tools will not recognise your submission, and it cannot be graded. Before you upload your submission, make sure that it follows the format requirements. You can do that by repeating what the submissions tools do, by running the following commands *at the command line*:

```
$ unzip task1.zip
$ cd task1
$ javac TCPAsk.java
$ java TCPAsk time.nist.gov 13
```

Make sure that this sequence of command can be executed with your submission *exactly as written*. Otherwise, you are very likely to fail.

Evaluation

The first part of the evaluation is to verify that your submission has the correct format, as described above. If it does not have the correct format, no further evaluations are made, and the result will be "fail".

As part of the evaluation of your submission, a number of tests are conducted to check the functionality of your submission.

