In this task, we will extend the TCPClient class from Task 1 so that it can handle different kinds of servers.

In Task 1, the client connects to the server, sends some (optional) data to the server, and reads data from the server until the server closes the connection. So the assumption is that the server closes the connection when all data is sent. Some application protocols work that way, but not all. There are application protocols that expect the client to close the connection first, for instance. The rules for how and when a connection is closed depend on the application protocol.

The idea with the TCPClient class is that it should be a general-purpose TCP client that can be used to communicate with servers for different application protocols. The problem is that there is not a single model for closing connections, it depends on the application protocol. In this assignment, you will extend the TCPClient class from Task 1 with options to support different ways of communicating with servers when it comes to closing connections.

In Task 1, the askServer method returns when the server closes the connection. Here we will add two other conditions for askServer to return:

1. Timeout. When askServer has not received any data from the server during a period of time, it closes the connection and returns.
2. Data limit. When askServer has received a certain amount of bytes from the server, it closes the connection and returns.

Furthermore, we will add the possibility for TCPClient to shut down the connection first.

Extending TCPClient

We add a constructor for TCPClient that takes three parameters:

```
public TCPClient(boolean shutdown, Integer timeout, Integer limit)
```

If the `shutdown` boolean parameter is true, TCPClient will shut down the connection in the outgoing direction (but only in that direction) after having sent the (optional) data to the server. Otherwise, TCPClient will not shut down the connection.

The `timeout` parameter is the maximum time in milliseconds that the client should wait for data before returning. If there is no upper limit for how long the client should wait, `timeout` is null.

The `limit` parameter is the maximum amount of data (in bytes) that the client should receive before returning. If there is no upper limit for how much data the client should receive, `limit` is null.

The conditions for when askServer should return can be summarised as follow. The askServer methods returns when one of the following conditions is met:

1. The server closes the connection.
2. The `timeout` parameter is not null, and the client has not received any data during the last `timeout` milliseconds.
3. The `limit` parameter is not null, and the client has received `limit` bytes of data from the receiver.

Note that the `limit` parameter is an absolute limit. The askServer method must not return more data than that to the caller.

Task

Implement the TCPClient class with support for closing the connection, timeouts and data limits.

TCPAsk

For Task 2, there is an extended version of TCPAsk with the shutdown, timeout and limit parameters.

```
$ java TCPAsk [--shutdown] [--timeout <milliseconds>] [--limit <bytes>]  <hostname> <port> [<data for server>]
```

Explanation of the notation: names within angle brackets "<...>" are symbolic names representing parameter values. Parameters within square brackets "[...]" are optional.

For example, to call the whois server at "iis.se" asking for information about "kth.se" using a timeout of 300 milliseconds, TCPAsk would be executed in the following way:

```
$ java TCPAsk --timeout 300 whois.iis.se 43 kth.se
```

Testing

You will need servers to test against. Here are some suggestions. Test your implementation by running TCPAsk with different settings of the options that control how connections are managed.

Protocol	Server name	Port	Arguments (data sent to server)	Comment
Daytime	time.nist.gov	13	None	Public server at NIST, the National Institute of Standards and Technology, a US government agency. Note that this server has limitations for how frequently you can query it (at most once every four seconds). See https://tf.nist.gov/tf-cgi/servers.cgi.
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 The Swedish Internet Foundation (Internetstiftelsen).
Whois	whois.internic.net	43	String (a domain name, an IP address or an AS number)	Public server at ICANN, the Internet Corporation For Assigned Names and Numbers.
Echo	java.lab.ssvl.kth.se	7	String	KTH server. Returns whatever data it receives. *Note: server does not close connection.*
Discard	java.lab.ssvl.kth.se	9	String	KTH server. Drops all data it receives. *Note: server does not close connection.*
Chargen	java.lab.ssvl.kth.se	19	None	KTH server. Generates stream of characters. *Note: server does not close connection.*

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

Resources

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

- TCPAsk.java – the TCPAsk Java program with options for controlling connections.
- tcpclient/TCPClient.java – Skeleton declaration of the tcpclient.TCPClient class.

The files are available in a zip archive called "task2.zip".

You can find the zip archive here: task2.zip

Submission

The rules are the same as for Task 2: Your submission should be a zip file named "task2.zip", containing the same files and directory structure as in the template.

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". This check will be made for each submission you make, with a short delay.

As part of the evaluation of your submission, a number of tests are conducted to check the functionality of your submission. These tests will be performed only after the due date.

Note: to participate in the pre-runs of grading tests that take place before the due date, you have to use the special "pre-run" assignment created specifically for that purpose. You can find it in the Socket Programming Project module. Here is a direct link: Task 2: TCPAsk Connections pre-run.