# Assignment 1: USI Twitter

***Due date:*** *Friday, October 19, 2018 at 22:00*

> *This is an individual assignment. You may discuss it with others, but your code and documentation must be written on your own. You might receive a bonus for an outstanding solution.*

Implement a new messaging system called *USI Twitter*. In *USI Twitter*, clients send tweets (messages) containing hashtags. A hashtag is a word that start with the hash character "#". Clients can subscribe to various hashtags that they're interested in. A client must only receive tweets containing hashtags that the client subscribed to.

At the center of this messaging system is the server. Clients connect and communicate with the server using a protocol you will design. The server is responsible for keeping track of each client's subscriptions, as well as routing tweets to clients based on the hashtags. When the server receives a tweet, it has to parse all the hashtags within the tweet. Then, the server sends the tweet to all clients that have subscribed to one of the hashtags within the tweet. Note that a client should never receive its own tweets.

Specifically, you must implement two independent Java applications. One, called *TwitterClient*, implements the client component. The other one, called *TwitterServer*, implements the server component. You must design your own protocol for the communication between the client and the server. This protocol must use TCP sockets.

The *TwitterClient* applications takes two command-line arguments: the hostname and TCP port of the *TwitterServer*. The *TwitterClient* must work as an interactive shell, reading commands from the standard input, with one command per line. Upon receiving a tweet from the server, the client should immediately print it to standard output, one tweet per line. The *TwitterClient* must support the following commands:

- *SUBSCRIBE #hashtag*

  the client instructs the server to send to the client, from now on, all the tweets that contain the given *#hashtag* sent by other clients.

- *UNSUBSCRIBE #hashtag*

  the client instructs the server to stop sending tweets that contain the given *#hashtag*.

- *TWEET message*

  the client sends the tweet to the server, which in turn will send the tweet to other subscribed clients. *message* is a sequence of any character except for newlines ('\n' or '\r').

The *TwitterServer* application takes one command-line argument: the TCP port on which the server should listen. The server prints every tweet it receives to standard output.

## Requirements

To grade your implementation, we will run automated tests that use the client and server program you submit. To ensure you receive full points, the programs must have the specific input/output that we describe. Here are the most important requirements:

- Your code must compile.
- When each client starts, it should set-up a single TCP session with the server that it will use for all communication.

- The client program must support the two commands described above. The client should read these commands from standard input.
- The client should print all the tweets it receives to standard output (one tweet per line). It must not print anything other than the tweets it receives (i.e., you should not print debugging information in your final submission).
- The client/server should support UTF-8 encoded tweets (default encoding for most terminal applications), including all printable characters, excluding unprintable characters and newlines.
- The server must print all the tweets it receives to standard output, one per line. It must not print anything else.

# Example

We will show you an example of running three clients connected to a server. We show what you'd see in the terminal window for each program. The dark text is what we typed (i.e. standard input to the program), while the light text is what the program printed to standard output.

We first launch the server, listening on port 1234. Then we launch three clients, and enter a subscription commands.

Server

```
$ ./server 1234
```

Client A

```
$ ./client 127.0.0.1 1234
SUBSCRIBE #love
```

Client B

```
$ ./client 127.0.0.1 1234
SUBSCRIBE #love
```

Client C

```
$ ./client 127.0.0.1 1234
SUBSCRIBE #trump
```

Then we send a tweet from client C,

Client C

```
TWEET this has a hashtag at the end #love
```

which should be received by both clients A and B:

Client A

```
this has a hashtag at the end #love
```

Client B

```
this has a hashtag at the end #love
```

In the middle of the session, a client can add a new subscription:

Client B

```
SUBSCRIBE #trump
```

So now this tweet:

Client A

```
TWEET this #love tweet has two hashtags #trump
```

Should be received by both clients B and C (but not A):

Client B

```
this #love tweet has two hashtags #trump
```

Client C

```
this #love tweet has two hashtags #trump
```

Client B unsubscribes from a hashtag:

Client B

```
UNSUBSCRIBE #trump
```

So client B will not receive this tweet (but client C should):

Client A

```
TWEET only client C receives this #trump
```

This is what the terminal windows of each program look like at the end of this session.

Server

```
$ ./server 1234
this has a hashtag at the end #love
this #love tweet has two hashtags #trump
only client C receives this #trump
```

Client A

```
$ ./client 127.0.0.1 1234
SUBSCRIBE #love
this has a hashtag at the end #love
TWEET this #love tweet has two hashtags #trump
TWEET only client C receives this #trump
```

Client B

```
$ ./client 127.0.0.1 1234
SUBSCRIBE #love
this has a hashtag at the end #love
SUBSCRIBE #trump
this #love tweet has two hashtags #trump
UNSUBSCRIBE #trump
```

Client C

```
$ ./client 127.0.0.1 1234
SUBSCRIBE #trump
TWEET this has a hashtag at the end #love
this #love tweet has two hashtags #trump
only client C receives this #trump
```

# Submission Instructions

Submit all your source files. Add comments to your code to explain sections of the code that might not be clear. You may use an integrated development environment (IDE) of your choice. However, *do not submit any IDE-specific file,* such as project description files, and *make absolutely sure that the files you submit can be compiled and tested with a simple invocation of the standard* javac *compiler and the standard* java *virtual machine.*

In addition to the source files, submit a text file called *README* containing a brief description of your implementation, including a description of the communication protocol you developed, and also a list of all the limitations and errors you are aware of were unable to fix.

Package all the files you need to submit in an archive file named

`a01-<lastname>-<firstname>`

and submit that file through the iCorsi system.