The Twins Service

The Twins protocol provides clients with access to a service allowing them to register their name and date of birth in an online database and learn the names of other people who share the same birthday. For the purposes of this service, a "twin" is any other person sharing the same day and month of birth (the year is ignored). **Note**: nobody is their own twin.

Server states

While interacting with a client, a Twins server session can be in one of these states:

1. **NEW**: waiting for a "hello" message from the client
2. **RECEIVE_NAME**: waiting for the client to send a name
3. **RECEIVE_DATE**: waiting for the client to send a date
4. **RECEIVE_REQ**: waiting for the client to send a request

Message formats

All messages are plain text and are terminated by newlines; the terminating newline is not treated as part of the message. This allows Java implementations to use `java.io.BufferedReader`.

Leading and trailing whitespace in all messages should be accepted and discarded. Letter case is significant in a <*name*> (see below) but **not** in any other message. So, for example, the names `pele` and `Pele` are distinct but the following "hello" messages are all equivalent:

```
Hello
  hello
hELLo
```

A <*date*> is a date string in colon-separated format *day* : *month* : *year*, where each field is a string representation of an integer (as accepted by `Integer.parseInt` in Java), subject to the following constraints:

$1900 \leq year \leq 2019$

$1 \leq month \leq 12$

$1 \leq day \leq n$, where n is either 31, 30 or 29, depending on *month*

(leap-year determination is **not** required)

For example, `1:1:1900` and `29:02:02019` are valid <*date*>s, but `31:9:1993` and `18:12:1899` and `1/1/1900` are not.

A <*name*> is any non-empty string, excluding `BEGIN TWINS` and `END TWINS`.

A <*twins list*> is a multi-line message of the form:

BEGIN TWINS
<*name*>$_1$
…
<*name*>$_n$
END TWINS

where the (possibly empty) list of <*name*>s is a list of all the client's twins. The order of the list is not specified. The begin and end parts must be sent, even if the list is empty.

The Twins Protocol

The sequences of messages required by the protocol are determined by the server session state, as follows (comments in square brackets describe the effect on server state):

NEW:

Client: `Hello`
Server: `What is your name?`
[Server state → **RECEIVE_NAME**]

RECEIVE_NAME:

Client: *<name>*

Then alternatives chosen by server:
if <name> is already registered:

Server: *<twins list>*
[Server state → **RECEIVE_REQ**]

or, if <name> is not already registered:

Server: `When were you born?`
[Server state → **RECEIVE_DATE**]

RECEIVE_DATE:

Client: *<date>*
[Server adds client to database]
Server: *<twins list>*
[Server state → **RECEIVE_REQ**]

RECEIVE_REQ:

Alternatives chosen by client:

Client: `Quit`
[Server closes connection]

or

Client: `Refresh`
Server: *<twins list>*
[Server state → **RECEIVE_REQ**]

or

Client: `Delete me`
[Server deletes client from database and closes connection]

At any point, if the client violates the protocol (ie the server is waiting for a message and the client sends a message which breaks the rules) the server should send the appropriate error message and close the connection:

violation	error message
invalid name:	`Error 1` [*if server was expecting a name*]
invalid date:	`Error 2` [*if server was expecting a date*]
name in use:	`Error 3` [*only possible for multi-threaded servers*]
all other cases:	`Error 0`

Notes about concurrency

1. If the server is single-threaded, it will not be possible for more than one client to connect at a time. In this case, the refresh option is not much use to the client (since the database cannot be updated by any other client until the currently connected client quits) but refresh must still be implemented.

2. If the server is multi-threaded, several connections can be active at the same time. In this case implementations must take care to synchronize access to the database to ensure that clients do not interfere with one another. This is tricky. `Error 3` should be sent to any client who tries to use a name which is already being used by some other currently connected client. (But note that preventing this behaviour is not sufficient to prevent interference: what happens if one client adds or deletes a name while a twins list is being output to another, differently named client?)