

## Example Valid Message Sequences

Any deviation, however small, will mean that your implementation is incorrect. For example, when responding to a client's initial "hello" message, the following server responses may look very similar to the human eye, but only 1 and 2 are correct:

1. What is your name?
2.     what is YOUR name?
3. What's your name?
4. What is your name
5. What   is   your   name?

Valid Message Sequences examples:

Client: Hello

Server: What is your name?

Client: Alice

Server: When were you born?

Client: 15:4:1999

Server: BEGIN TWINS

Server:    ENDTWINS

Client:    deleteme

[connection closed]

Client: hello

Server: What is your name?

Client: Bob

Server: When were you born?

Client: 15:4:1999

Server: BEGIN TWINS

Server:    ENDTWINS

Client:           Quit

[connection closed]

Client: HELLO

Server: What is your name?

Client: Gregory Peck

Server: When were you born?

Client: 15:4:1923

Server: BEGIN TWINS

Server: Bob

Server: END TWINS Client:

refresh

Server: BEGINTWINS

Server: Bob

Server: END TWINS

## Study the server code

Before proceeding, you should have a basic understanding of the prototype code. The key parts to study are the **start()** method and the **session()** method.

### **start**

```
try (ServerSocket serverSocket = new ServerSocket(port)) { while (true) {  
    Socket conn = serverSocket.accept(); session(conn);  
}  
}
```

This code creates a `ServerSocket` and then enters a never-ending loop. In the loop, the call **serverSocket.accept()** waits for an incoming connection request. When a connection is established, `accept` creates a `Socket`, which implements a bi-directional connection between the server and the client. This `Socket` is then passed to the `session` method for processing. When `session` returns, we go back round the loop to wait for another connection.

### **session**

```
writer = new OutputStreamWriter(connection.getOutputStream());  
reader = new BufferedReader(...)
```

These two lines obtain the input and output streams from the connection, for communicating with the client. The following lines of code use those streams to receive and send messages. When the session is finished, the connection is closed.