Problem:
  low level
  lots of set-up

Better:
  Appl.
  Developer
  model
  • hide
    connection
    setup
  • support
    concurrent
    clients
    (some clients
     may be slow or block forever)

msg

Server

msg

Client A

Client B
Design a client/server app framework: 

Asynchronous model

What should client developers do?

- define message types [classes ⇒ object streams]
  ⇒ GVTF for user (sitting on top)

- API design choice
  {  
  ⇒ 1 active loop to read incoming messages in app.  
  ⇒ Message receive();
  ⇒ 2 react as we're told about messages (event-driven)
  ⇒ void messageReceived(Message m);

- active behavior of client (including sending messages)  
  (optional, but allow & support Runnable client objects)
- send(Message m) — sends m to the server (nonblocking)
- way to process messages received

API design choice

1. active loop to read incoming messages in app
   ```java
   Message receive();
   void messageReceived(Message m);
   ```

2. react as we're told about messages (event-driven)

   ① Client sends m
   Client waits for reply or Client sends m₁, sends m₂
   ③ [AbstractClient/]
   ② Client waits for reply 1, reply 2

   Meanwhile, client handles incoming messages from server

If both are available, using both simultaneously would be awkward: Don't know whether blocking call or msg handler should get the next msg.
Server side:
Server developer will provide:
  • mechanism for handling each client

- Shared state
  - Should be thread safe

- ClientHandler (run in their own threads)

- Need a way to create ClientHandler objects (factory method)

- Same way of handling messages received
- Sending messages
- Active? (Runnable?)
- Server Active computation (or just passive?)
Client

1. does it run? — does it actively do things

   => whether or not it's runnable

2. how does it receive messages?

   Active client — explicitly calls `recv()`

   Reactive client — implements `messageReceived(M msg)`