EE 422C
Socket Programming

Lecture 23

Last Lecture

- Multithreading, Synchronization

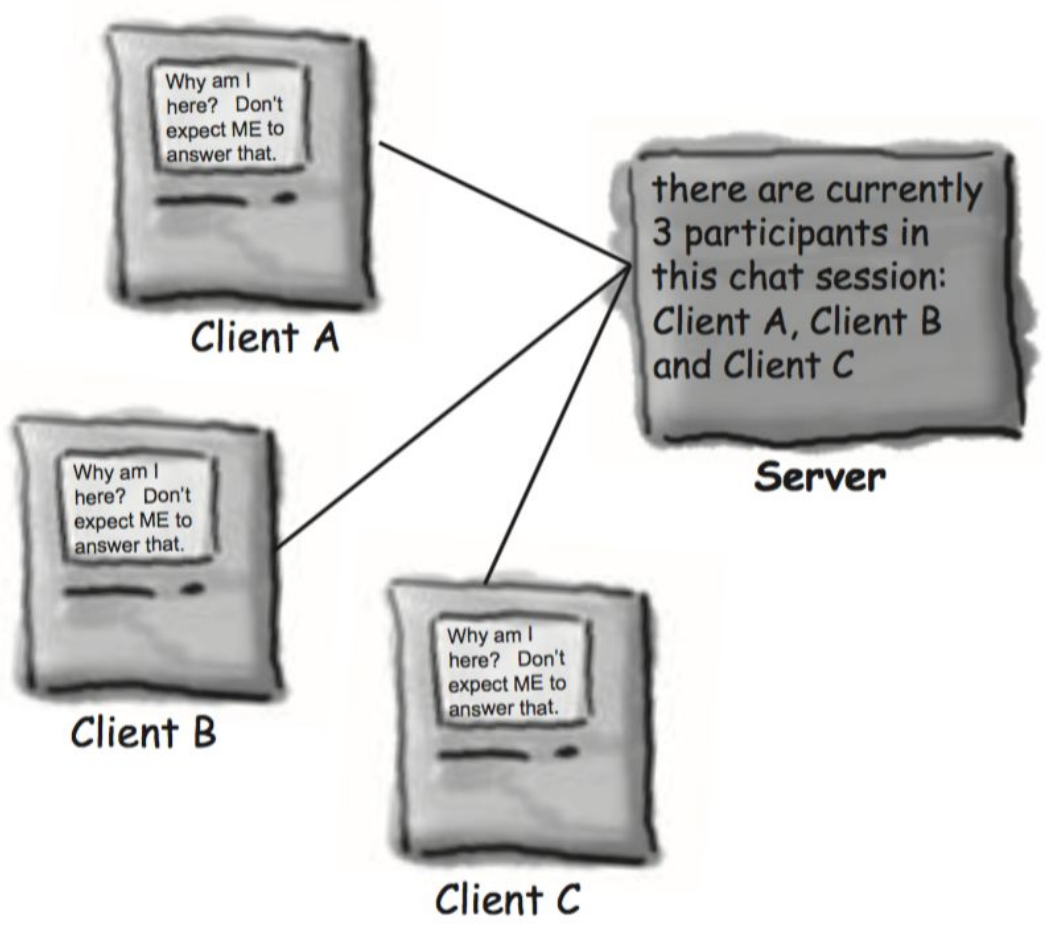
- Today:
  - Socket Programming (an app for multi-threading)

Scenario: Chat Program

Chat Program Overview

The Client has to know about the Server.

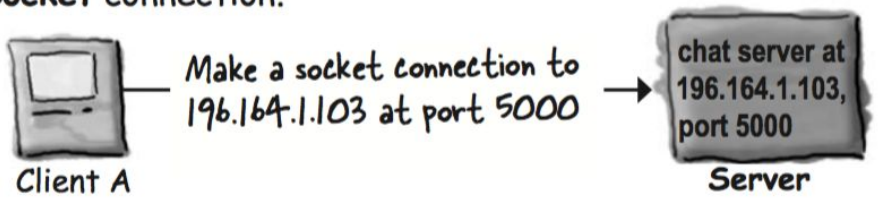
The Server has to know about ALL the Clients.



Network Connection - Overview

❶ Connect

Client connects to the server by establishing a **Socket** connection.

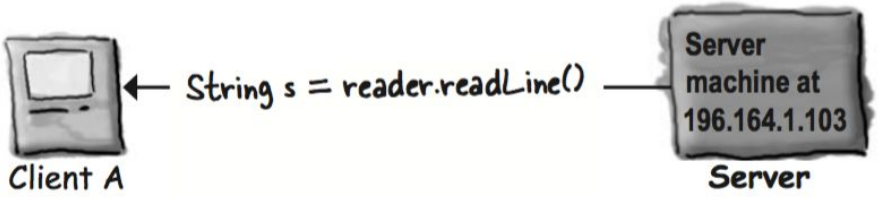


❷ Send

Client **sends** a message to the server

❸ Receive

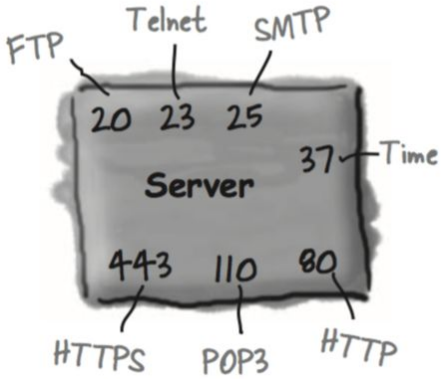
Client **gets** a message from the server



Socket Programming - Concept

1. Socket (java.net.Socket class) is an object that represents a network connection between two machines.
2. Client wants to make a Socket connection
   - who it is (Server's IP address), and
   - which service (Server's port number).

3. Port number is a 16-bit number that identifies a specific program on the server. The port numbers from 0 to 1023 are reserved for well-known services, pick your port number from 1024 to 65535.

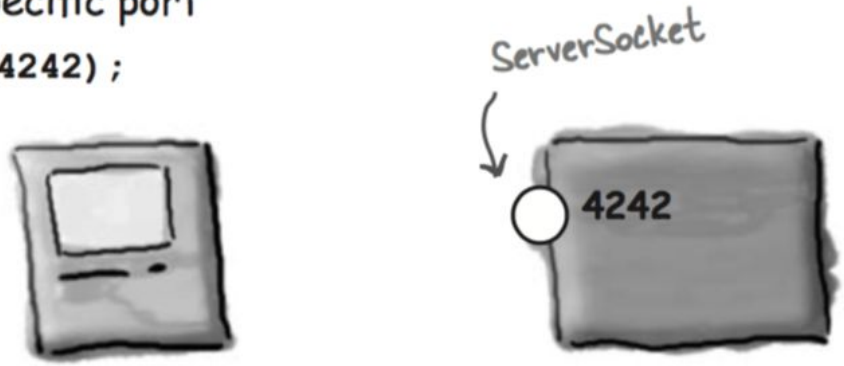


Socket Programming - Connect

1. Server application makes a ServerSocket, on a specific port

```
ServerSocket serverSock = new ServerSocket(4242);
```

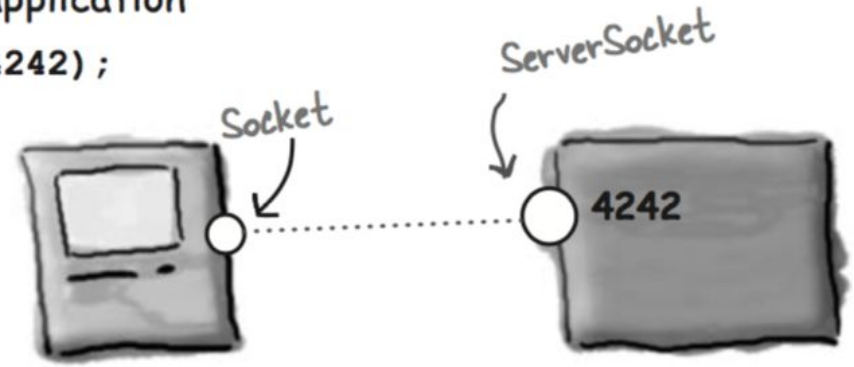
This starts the server application listening for client requests coming in for port 4242.



2. Client makes a Socket connection to the server application

```
Socket sock = new Socket("190.165.1.103", 4242);
```

Client knows the IP address and port number (published or given to him by whomever configures the server app to be on that port)

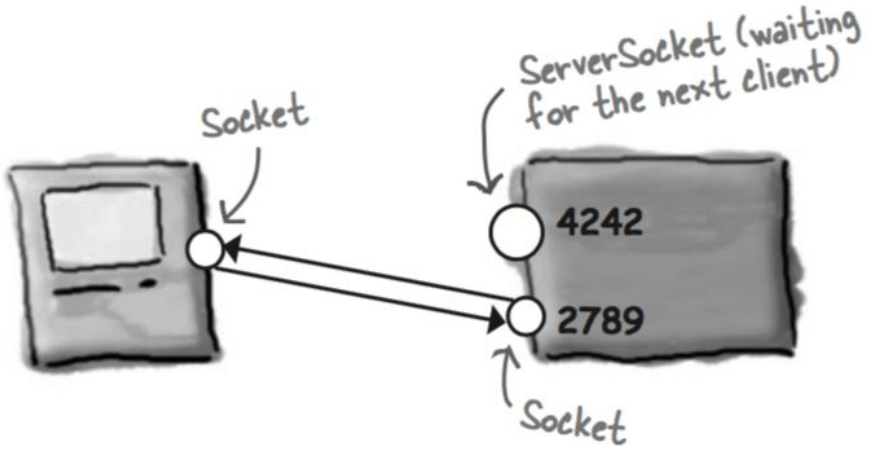


Socket Programming - Connect (cont.)

3 Server makes a new Socket to communicate with this client

```
Socket sock = serverSock.accept();
```

The accept() method blocks (just sits there) while it's waiting for a client Socket connection. When a client finally tries to connect, the method returns a plain old Socket (on a *different* port) that knows how to communicate with the client (i.e., knows the *client's* IP address and port number). The Socket is on a different port than the ServerSocket, so that the ServerSocket can go back to waiting for other clients.




```
ServerSocket serverSock = new ServerSocket(4242);
while(true) {
   Socket clientSocket = serverSock.accept();
    Thread t = new Thread(new ClientHandler(clientSocket));
    t.start();
    System.out.println("got a connection");
}
```

Server

```
Socket clientSock = new Socket("190.165.1.103",4242);
```

Client

Connection - iClicker (1)

When starting up a Java program with sockets

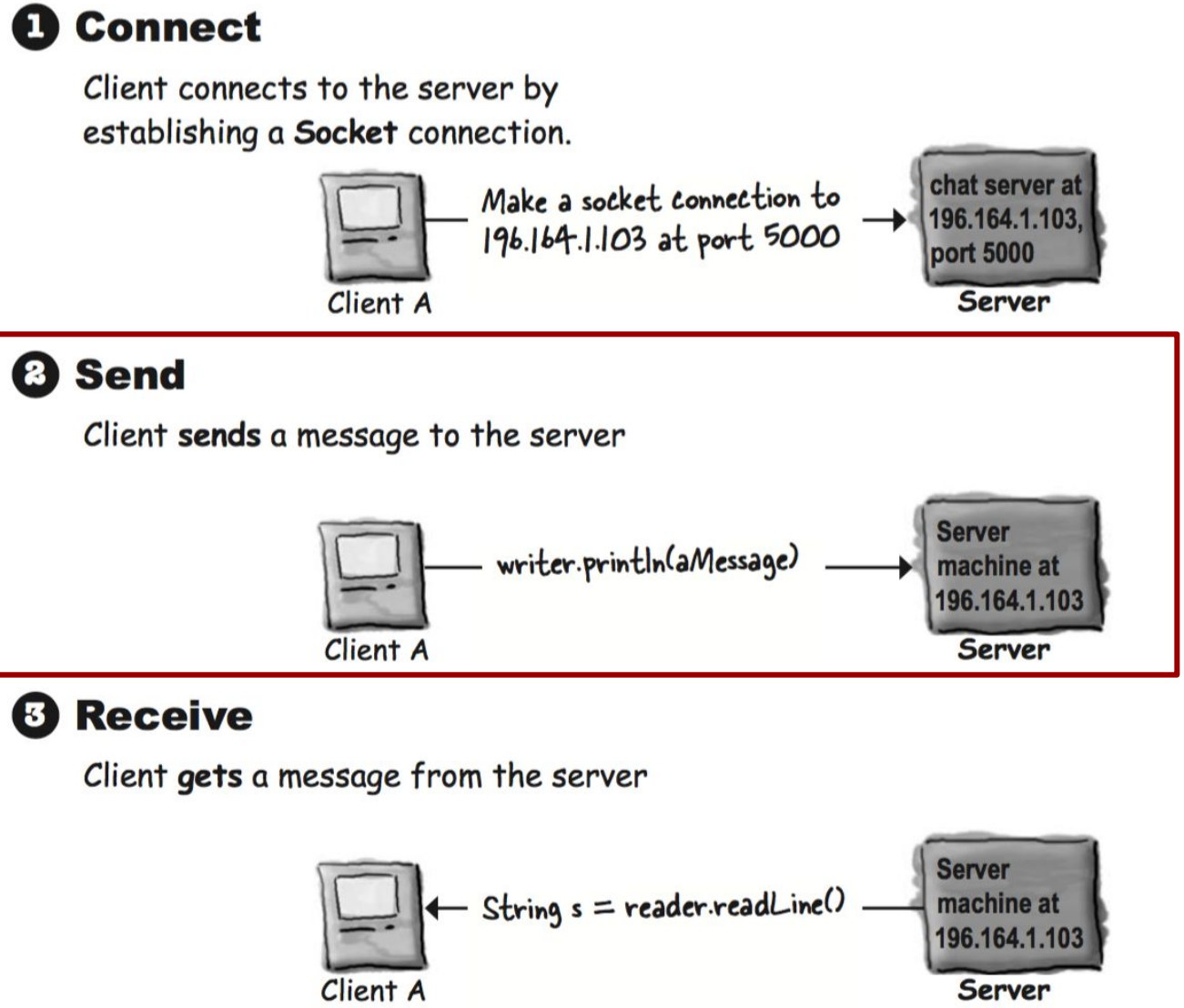
A. Start the Server up first

B. Start the Client up first

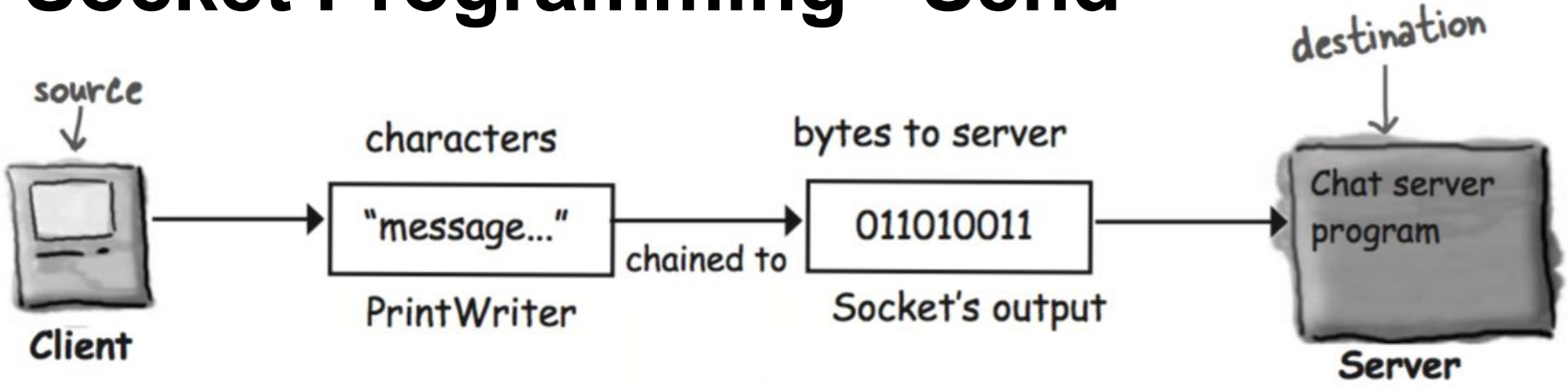
C. Either way is fine

D. Start them up simultaneously with parallel threads

Socket Programming - Send



Socket Programming - Send



1. Make a Socket connection to the server

```
Socket chatSocket = new Socket("127.0.0.1", 5000);
```

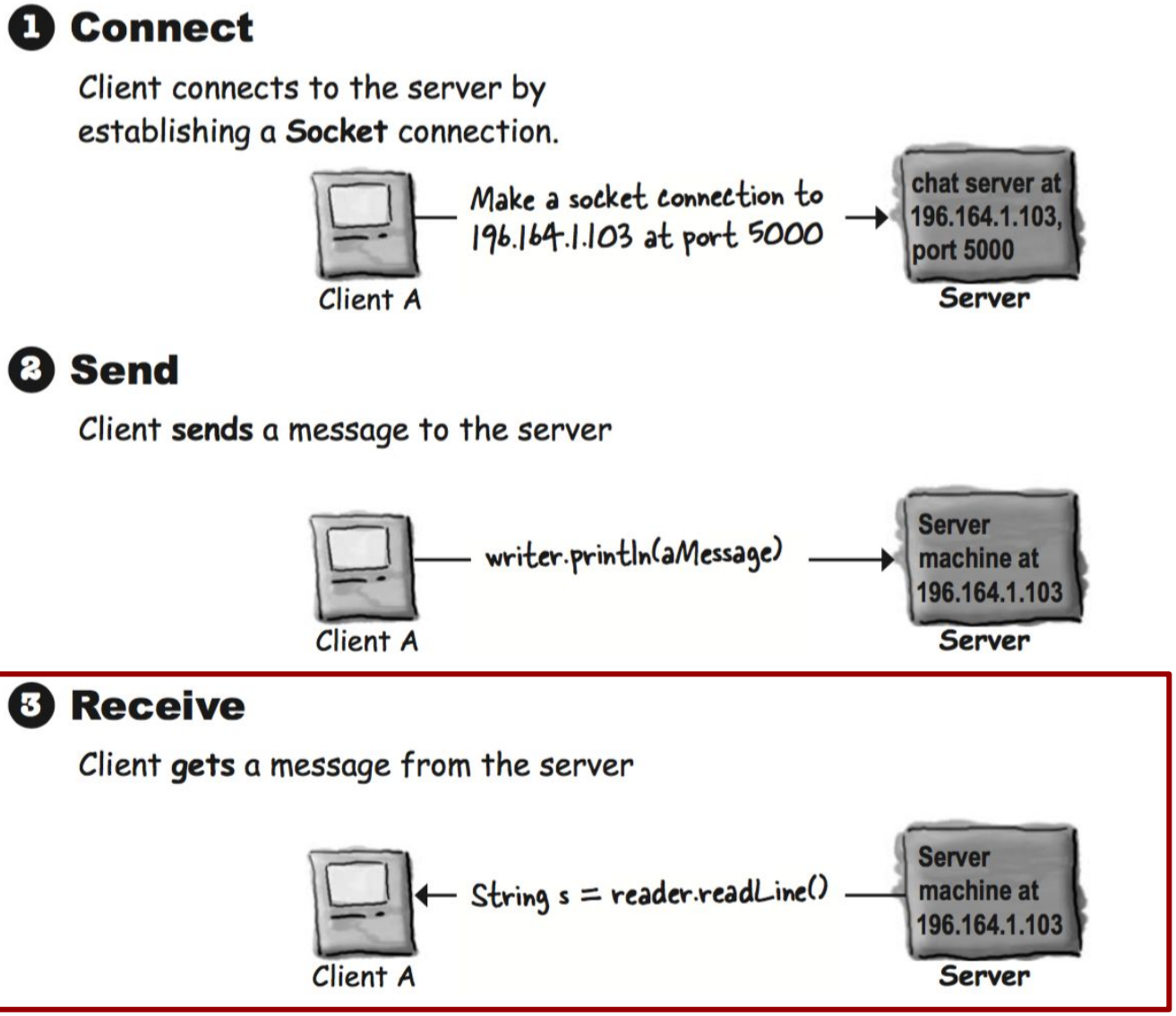
2. Make a PrintWriter chained to the Socket's low-level (connection) output stream

```
PrintWriter writer = new PrintWriter(chatSocket.getOutputStream());
```

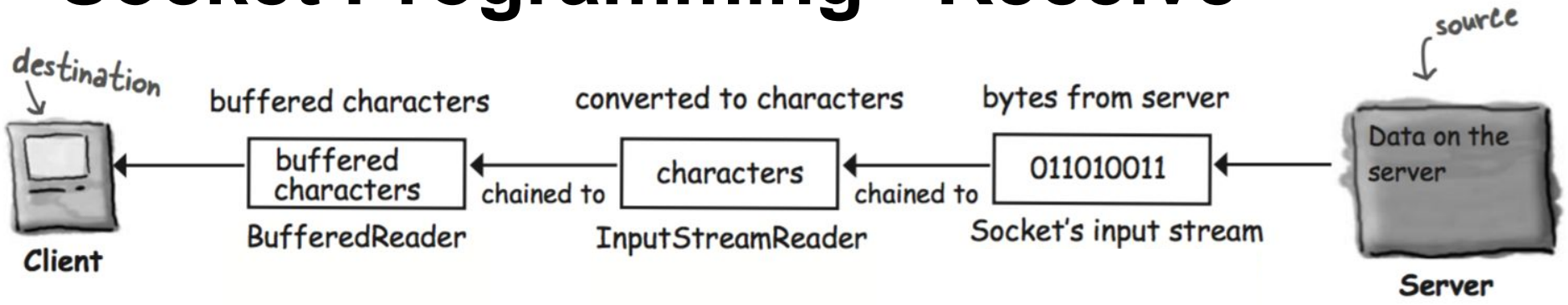
3. Write (print) something

```
writer.println("message to send");
writer.print("another message");
```

Socket Programming - Receive



Socket Programming - Receive



1 Make a Socket connection to the server

```
Socket chatSocket = new Socket("127.0.0.1", 5000);
```

2 Make an InputStreamReader chained to the Socket's low-level (connection) input stream

```
InputStreamReader stream = new InputStreamReader(chatSocket.getInputStream());
```

3 Make a BufferedReader and read!

```
BufferedReader reader = new BufferedReader(stream);
String message = reader.readLine();
```

Socket Programming - iClicker (2)

How does a client know the host's IP address when requesting a socket connection?

A. By querying the host directly

B. By looking up the host's IP address in some database, like a DNS, or some means outside the program.

C. Other

Socket Programming - Example

Example: Chat Server and Chat Client

Example - Client

```
public class ChatClient {
     private BufferedReader reader;
     private PrintWriter writer;
     private JTextArea incoming;
     private JTextField outgoing;
     ...
     private void setUpNetworking() throws Exception {
          Socket sock = new Socket("127.0.0.1", 5000);
          InputStreamReader streamReader = new InputStreamReader(sock.getInputStream());
          reader = new BufferedReader(streamReader);
          writer = new PrintWriter(sock.getOutputStream());
          Thread readerThread = new Thread(new IncomingReader());
          readerThread.start();
     }
}
class IncomingReader implements Runnable {
     public void run() {
          String message;
          while ((message = reader.readLine()) != null) {
               incoming.append(message + "\n");
          }}
```

Example - Server

```
public class ChatServer {
     private ArrayList<PrintWriter> clientOutputStreams;

     private void setUpNetworking() throws Exception {
          clientOutputStreams = new ArrayList<PrintWriter>();

          ServerSocket serverSock = new ServerSocket(5000);
          while (true) {
               Socket clientSocket = serverSock.accept();
               PrintWriter writer = new PrintWriter(clientSocket.getOutputStream());
               Thread t = new Thread(new ClientHandler(clientSocket));
               t.start();
               clientOutputStreams.add(writer);
               System.out.println("got a connection");
          }
     }
```

Example - Server (Cont.)

```
class ClientHandler implements Runnable {
     private BufferedReader reader;

     public ClientHandler(Socket clientSocket) throws IOException {
          Socket sock = clientSocket;
          reader = new BufferedReader(new InputStreamReader(sock.getInputStream()));
     }
     public void run() {
          String message;
          while ((message = reader.readLine()) != null) {
               notifyClients(message);
          }
     }
}
     private void notifyClients(String message) {
          for (PrintWriter writer : clientOutputStreams) {
               writer.println(message);
               writer.flush();
          }
     }
```

Example - Client (Cont.)

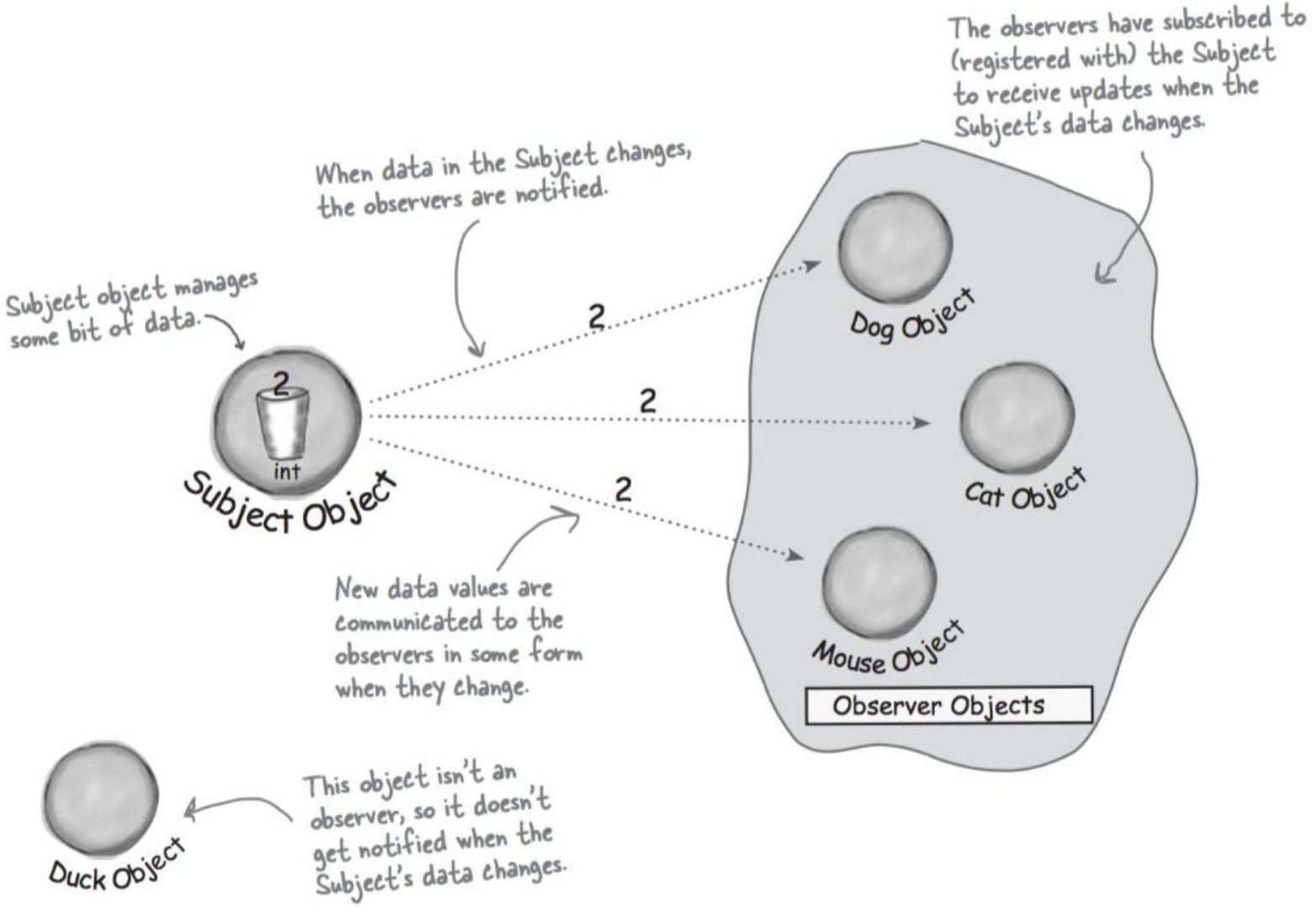
```
public class ChatClient {
     ...
     private void initView() {
     ...
          outgoing = new JTextField(20);
          JButton sendButton = new JButton("Send");
          sendButton.addActionListener(new SendButtonListener());

class SendButtonListener implements ActionListener {
          public void actionPerformed(ActionEvent ev) {
               writer.println(outgoing.getText());
               writer.flush();
          }
     }
```

Example - Chat Program

Demo: Chat Server and Chat Client

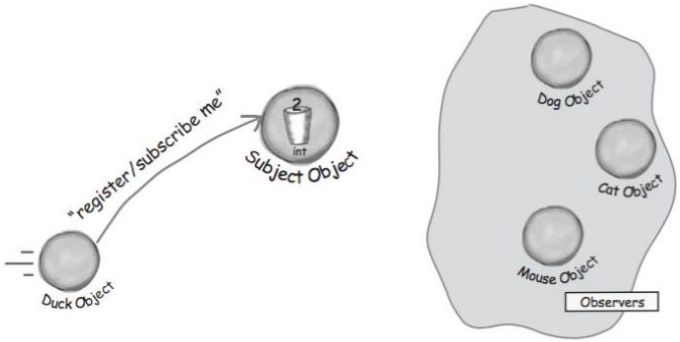
Recall: Observer Design Pattern



Observer Design Pattern

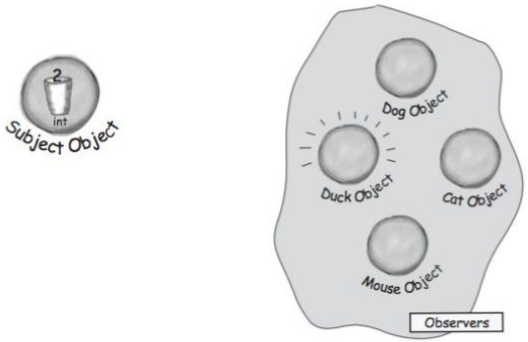
A Duck object comes along and tells the Subject that it wants to become an observer.

Duck really wants in on the action; those ints Subject is sending out whenever its state changes look pretty interesting...



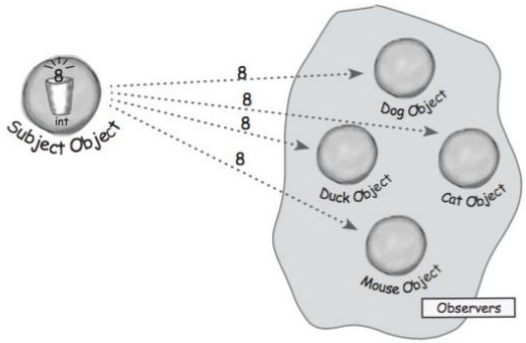
The Duck object is now an official observer.

Duck is psyched... he's on the list and is waiting with great anticipation for the next notification so he can get an int.



The Subject gets a new data value!

Now Duck and all the rest of the observers get a notification that the Subject has changed.



Example - Chat Program

Demo: Chat Program with Observer Design Pattern

Socket Programming - iClicker (3)

Which one is wrong?

A. Socket is an object that represents a network connection between different machines.

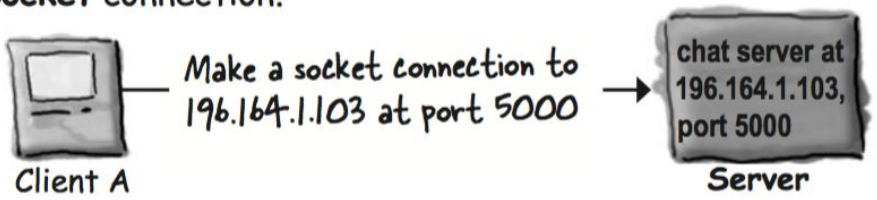
B. A ServerSocket can be used to communicate with multiple clients.

C. Given a ServerSocket with port number 5000 (`new ServerSocket(5000)`), it will return a Socket with the same port number when invoking `serverSock.accept()`.

Socket Programming - Summary

1 Connect

Client connects to the server by establishing a **Socket** connection.



2 Send

Client **sends** a message to the server

3 Receive

Client **gets** a message from the server