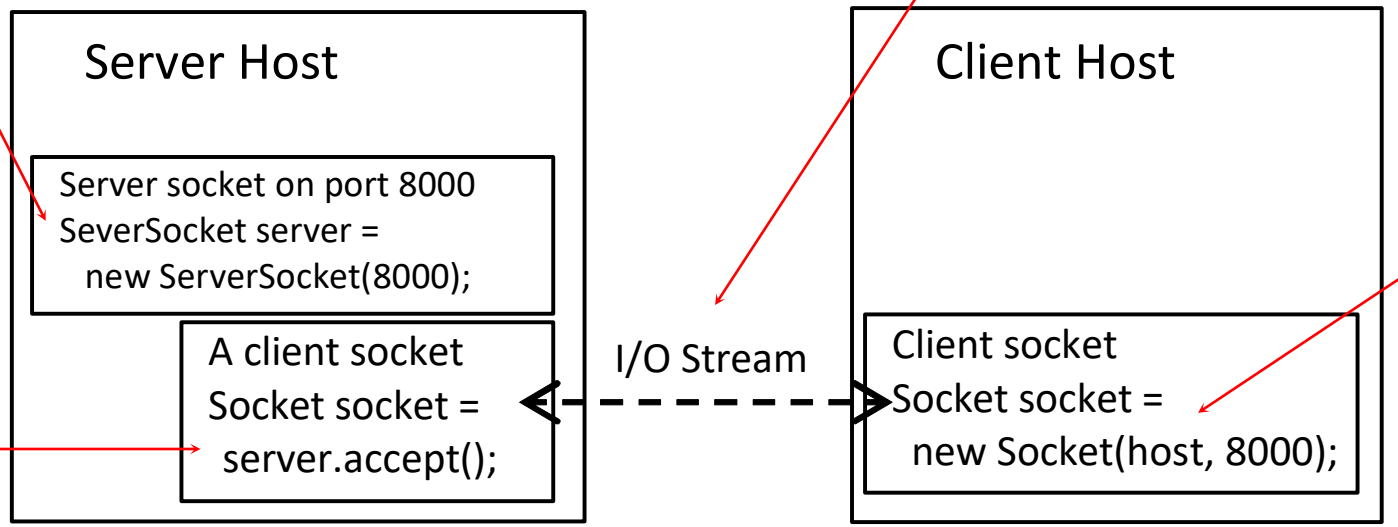
Networking

Shing-Chi Cheung
Computer Science and Engineering
HKUST

Client/Server Communications

The server must be running when a client starts. The server waits for a connection request from a client. To establish a server, you need to create a server socket and attach it to a port, which is where the server listens for connections.

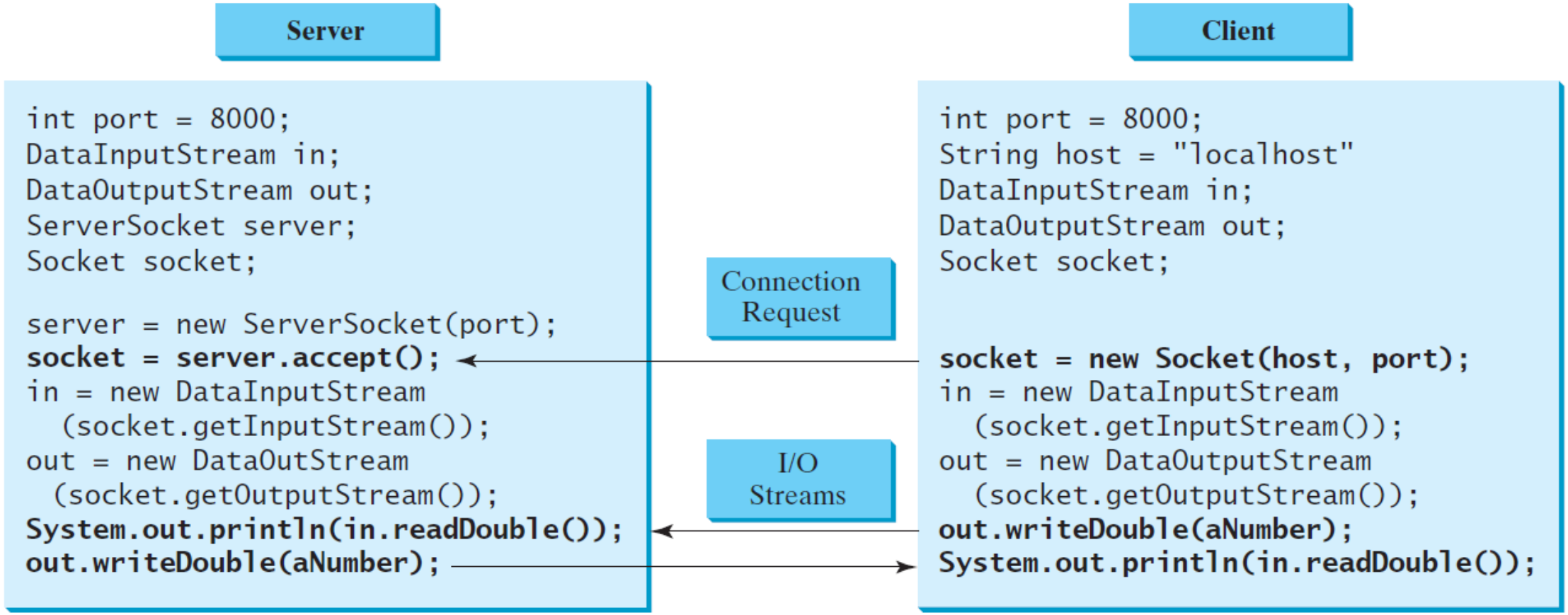
After the server accepts the connection, communication between server and client is conducted the same as for I/O streams.



After a server socket is created, the server can use this statement to listen for connections.

The client issues this statement to request a connection to a server.

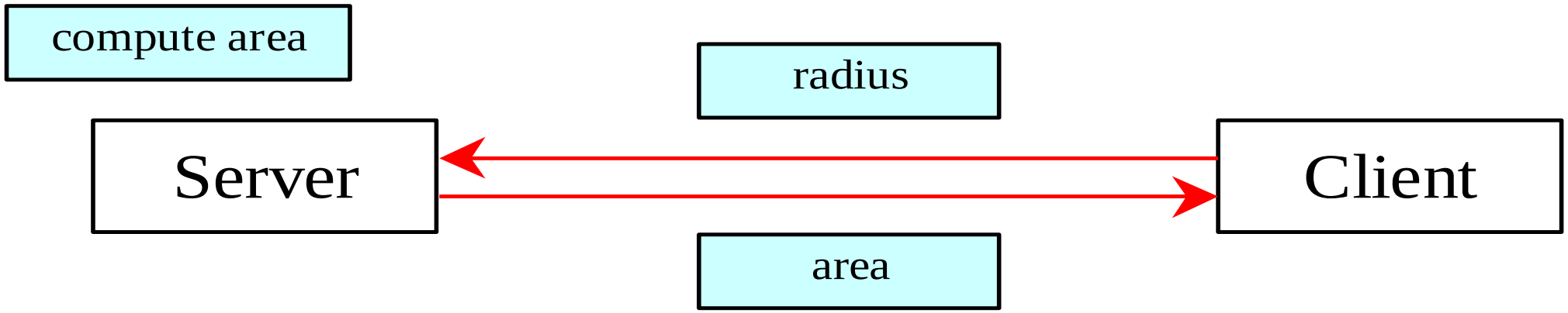
Data Transmission through Sockets



InputStream input = socket.getInputStream();

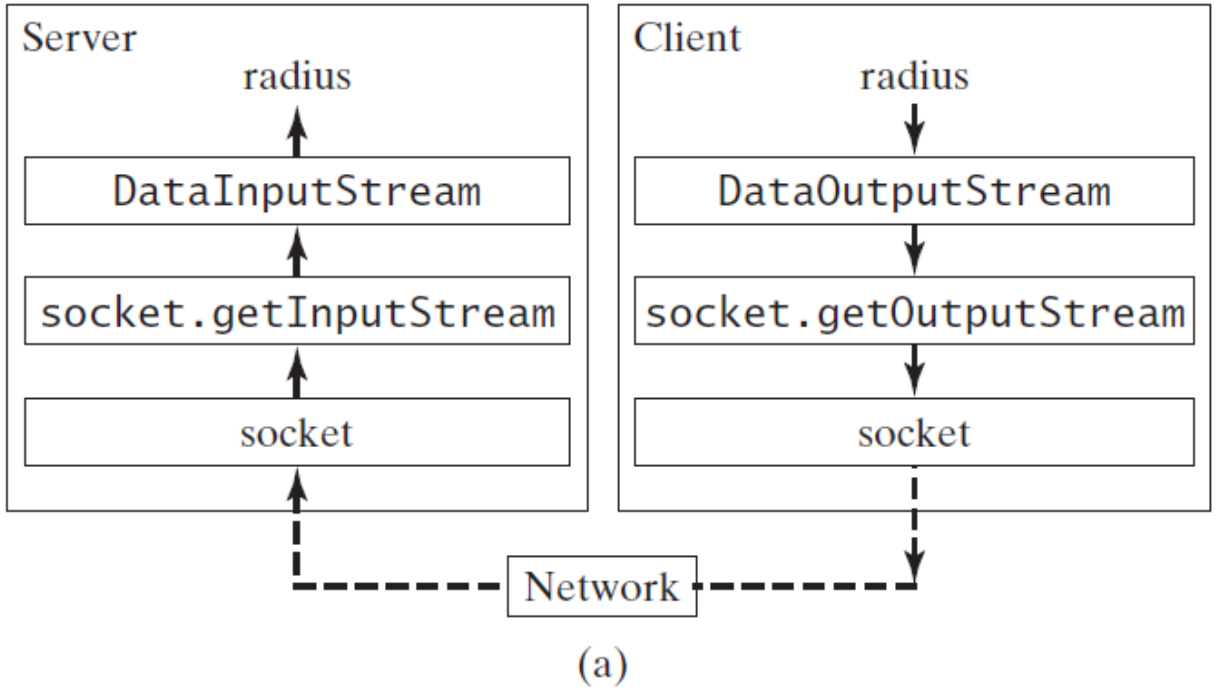
OutputStream output = socket.getOutputStream();

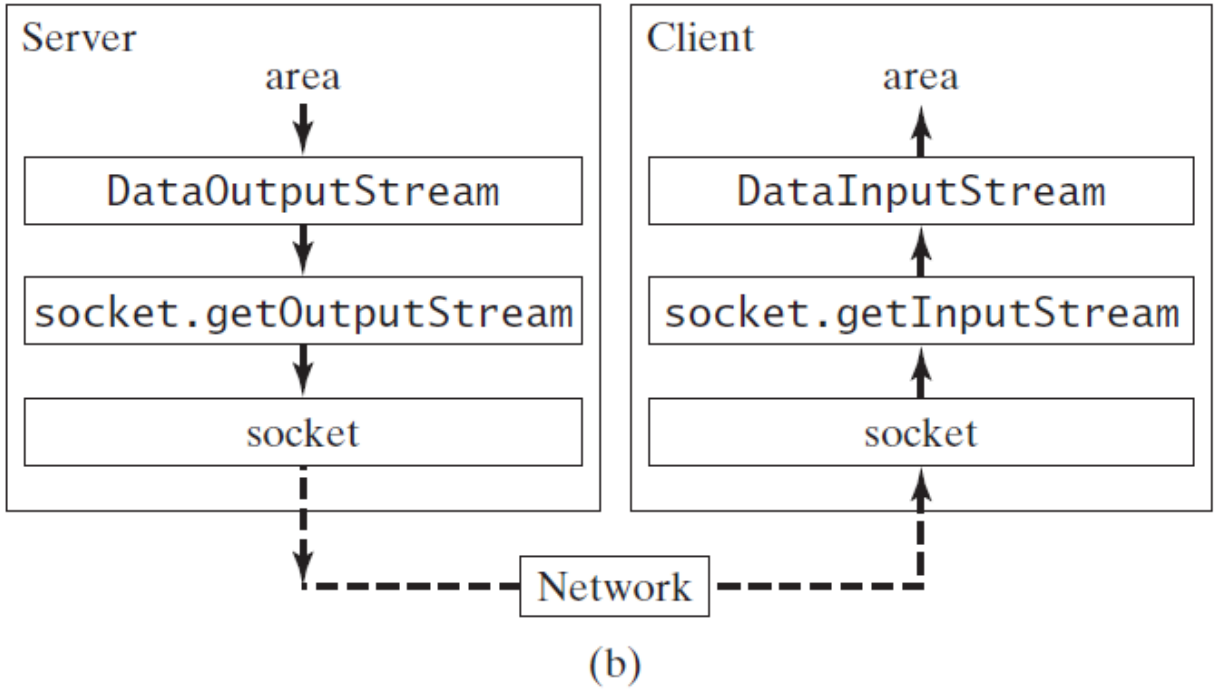
A Client/Server Example



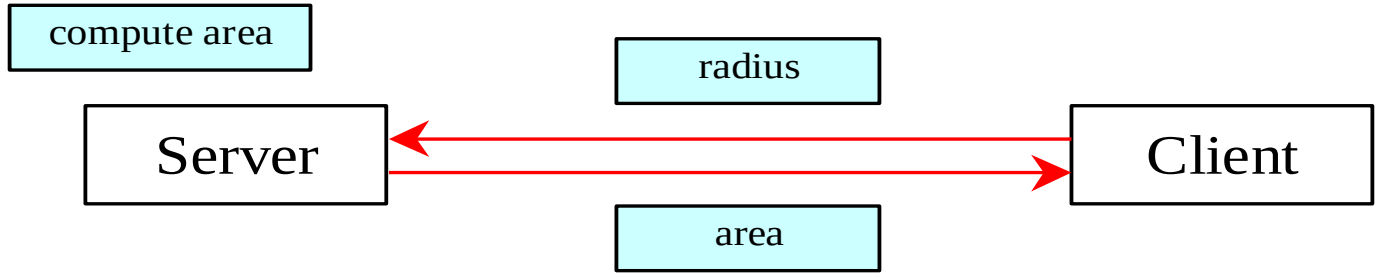
- Problem: Write a client to send data to a server. The server receives the data, uses it to produce a result, and then sends the result back to the client. The client displays the result on the console. In this example, the data sent from the client is the radius of a circle, and the result produced by the server is the area of the circle.

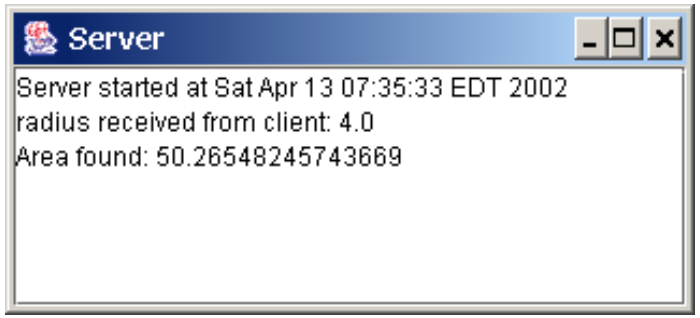
A Client/Server Example, cont.

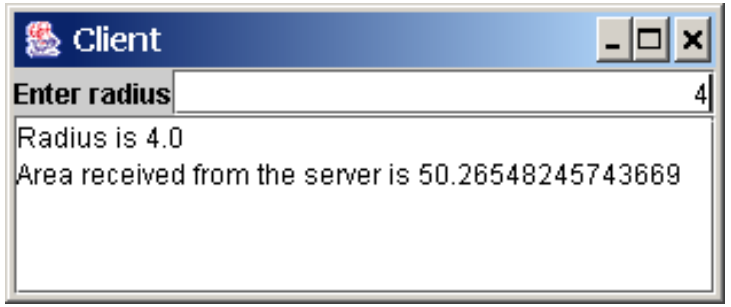




A Client/Server Example, cont.







Server Code

Client Code

Start Server

Start Client

Note: Start the server, then the client.

The InetAddress Class

Occasionally, we would like to know who is connecting to the server. We can use the InetAddress class to find the client's host name and IP address. The InetAddress class models an IP address. We can use the statement shown below to create an instance of InetAddress for the client on a socket.

```
InetAddress inetAddress = socket.getInetAddress();
```

Next, you can display the client's host name and IP address, as follows:

```
System.out.println("Client's host name is " +
  inetAddress.getHostName());
System.out.println("Client's IP Address is " +
  inetAddress.getHostAddress());
```

Serving Multiple Clients

Multiple clients are quite often connected to a single server at the same time. Typically, a server runs constantly on a server computer, and clients from all over the Internet may want to connect to it. You can use threads to handle the server's multiple clients simultaneously. Simply create a thread for each connection. Here is how the server handles the establishment of a connection:

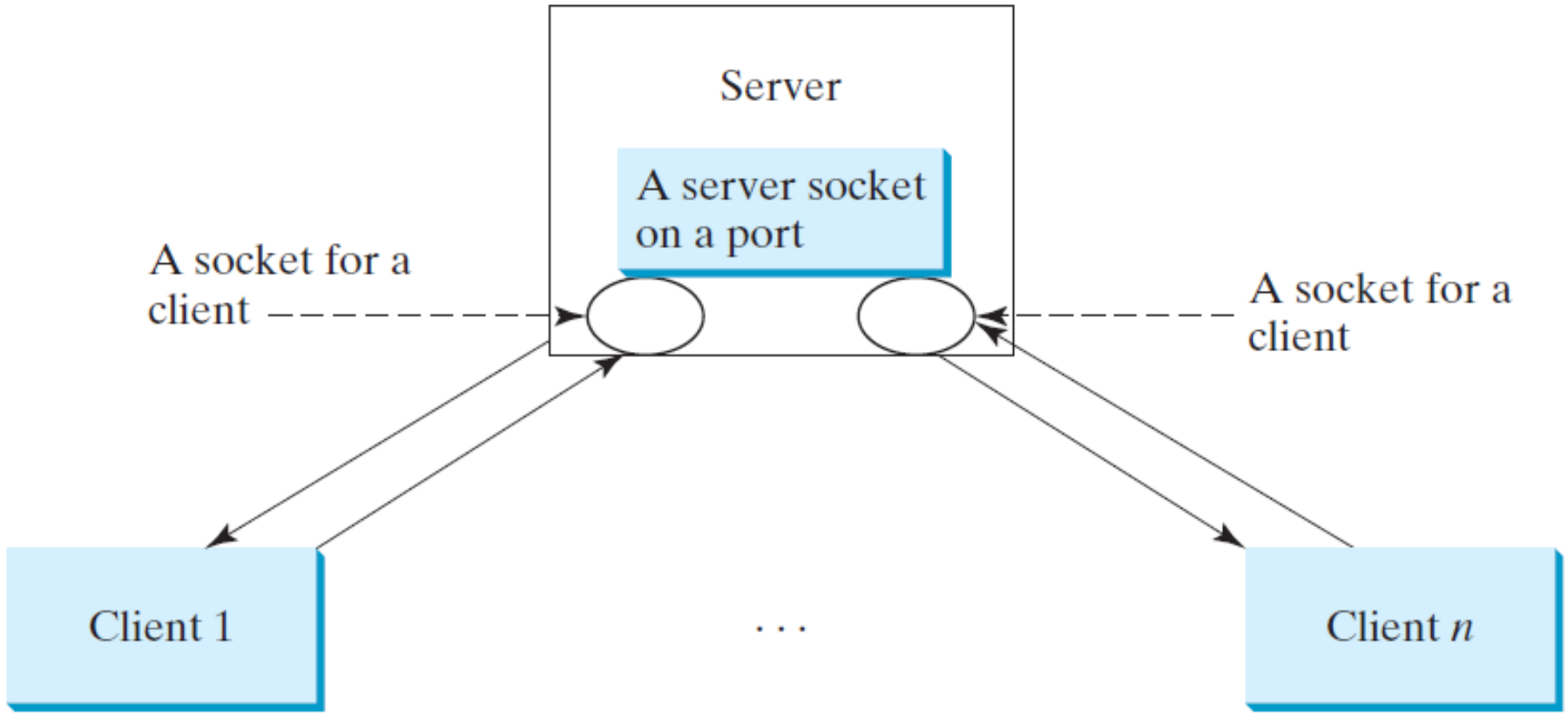
```
while (true) {
  Socket socket = serverSocket.accept();
  Thread thread = new ThreadClass(socket);
  thread.start();
}
```

Serving Multiple Clients

```
while (true) {
  Socket socket = serverSocket.accept();
  Thread thread = new ThreadClass(socket);
  thread.start();
}
```

The server socket can have many connections. Each iteration of the while loop creates a new connection. Whenever a connection is established, a new thread is created to handle communication between the server and the new client; and this allows multiple connections to run at the same time.

Example: Serving Multiple Clients



Server for Multiple Clients

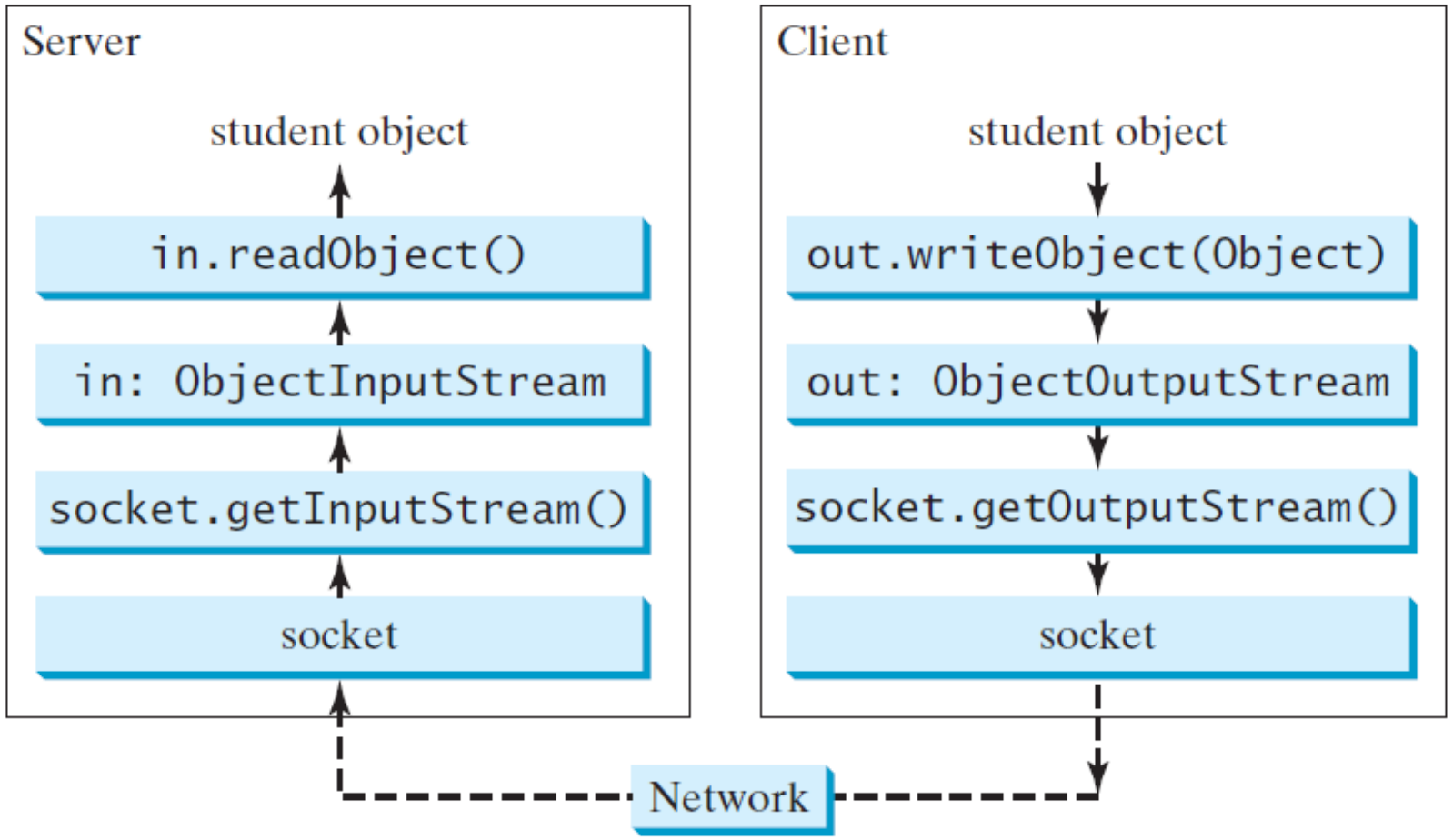
Start Server

Start Client

Note: Start the server first, then start multiple clients.

Example: Passing Objects in Network Programs

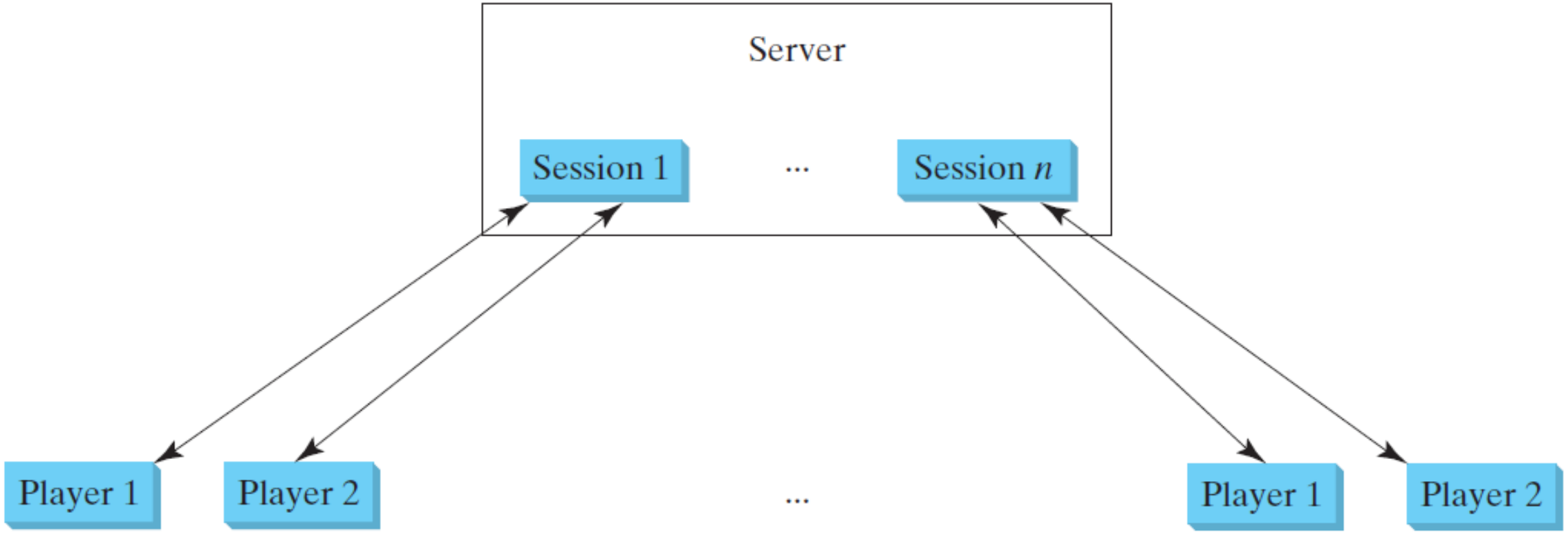
Write a program that collects student information from a client and send them to a server. Passing student information in an object.



Student Class | Student Sever | Student Client | Start Server | Start Client

Note: Start the server first, then the client.

Case Studies: Distributed TicTacToe Games



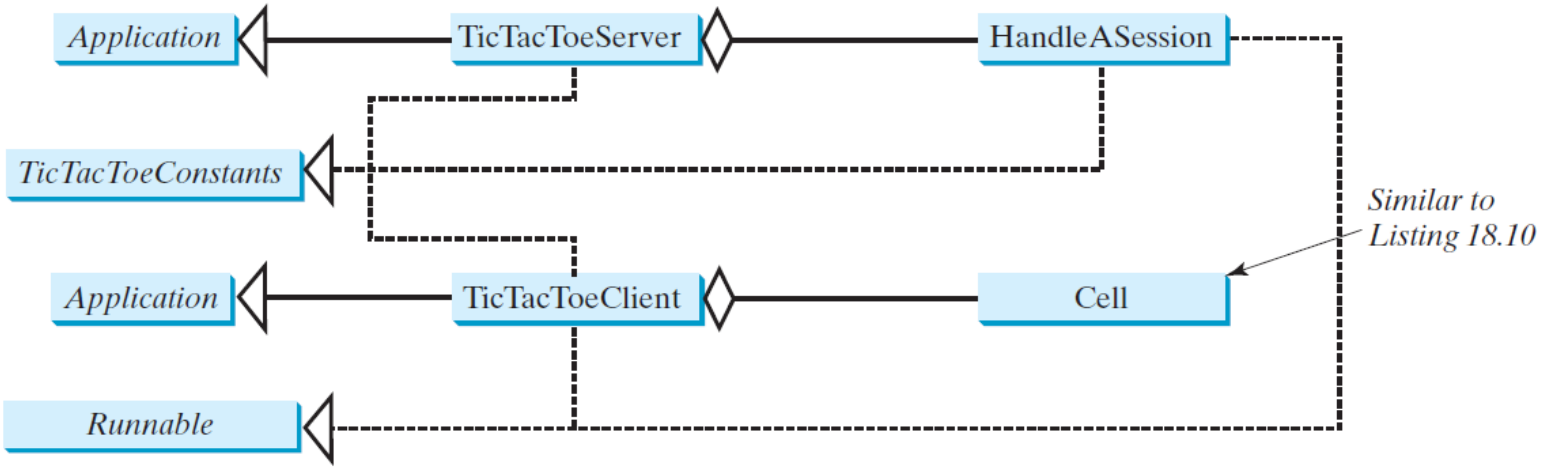
TicTacToeServer

TicTacToeClient

Run Server

Run Client

Distributed TicTacToe, cont.



TicTacToeServer

```
start(primaryStage: Stage):
  void
```

«interface» *TicTacToeConstants*

```
+PLAYER1 = 1: int
+PLAYER2 = 2: int
+PLAYER1_WON = 1: int
+PLAYER2_WON = 2: int
+DRAW = 3: int
+CONTINUE = 4: int
```

HandleASession

```
-player1: Socket
-player2: Socket
-cell: char[][]
-continueToPlay: boolean

+run(): void
-isWon(): boolean
-isFull(): boolean
-sendMove(out:
  DataOutputStream, row: int,
  column: int): void
```

TicTacToeClient

```
-myTurn: boolean
-myToken: char
-otherToken: char
-cell: Cell[][]
-continueToPlay: boolean
-rowSelected: int
-columnSelected: int
-fromServer: DataInputStream
-toServer: DataOutputStream
-waiting: boolean

+run(): void
-connectToServer(): void
-receiveMove(): void
-sendMove(): void
-receiveInfoFromServer(): void
-waitForPlayerAction(): void
```

Distributed TicTacToe Game