Chapter 31 Networking

Objectives

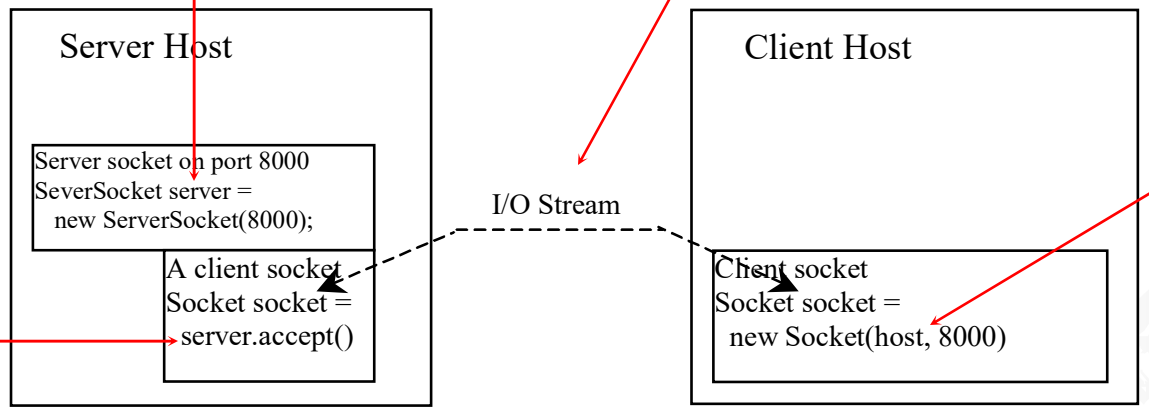
- To explain terms: TCP, IP, domain name, domain name server, stream-based communications, and packet-based communications (§31.2).
- To create servers using server sockets (§31.2.1) and clients using client sockets (§31.2.2).
- To implement Java networking programs using stream sockets (§31.2.3).
- To develop an example of a client/server application (§31.2.4).
- To obtain Internet addresses using the **InetAddress** class (§31.3).
- To develop servers for multiple clients (§31.4).
- To send and receive objects on a network (§31.5).
- To develop an interactive tic-tac-toe game played on the Internet (§31.6).

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.

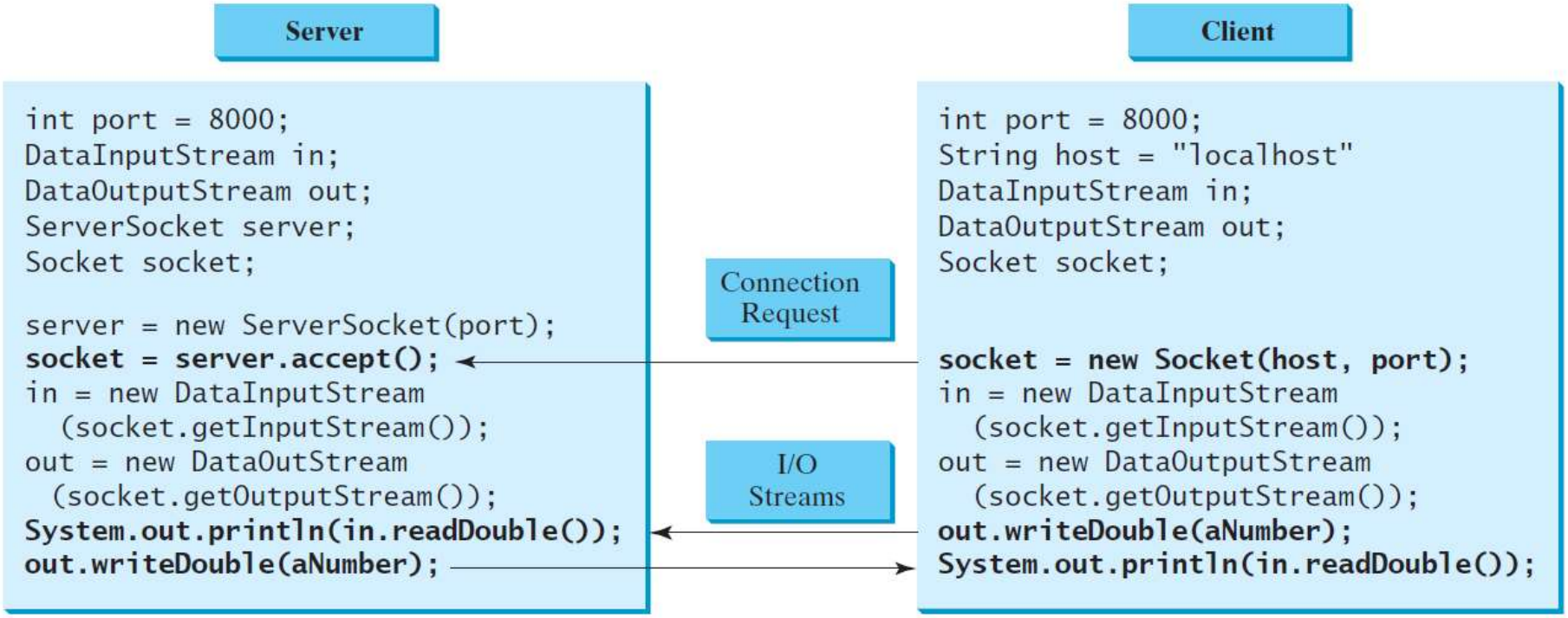
After the server accepts the connection, communication between server and client is conducted the same as for I/O streams.

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

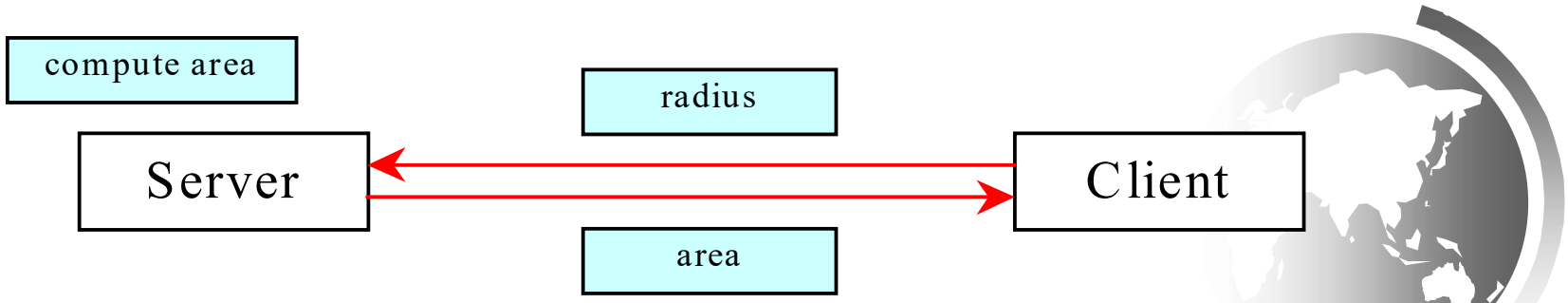


InputStream input = socket.getInputStream();

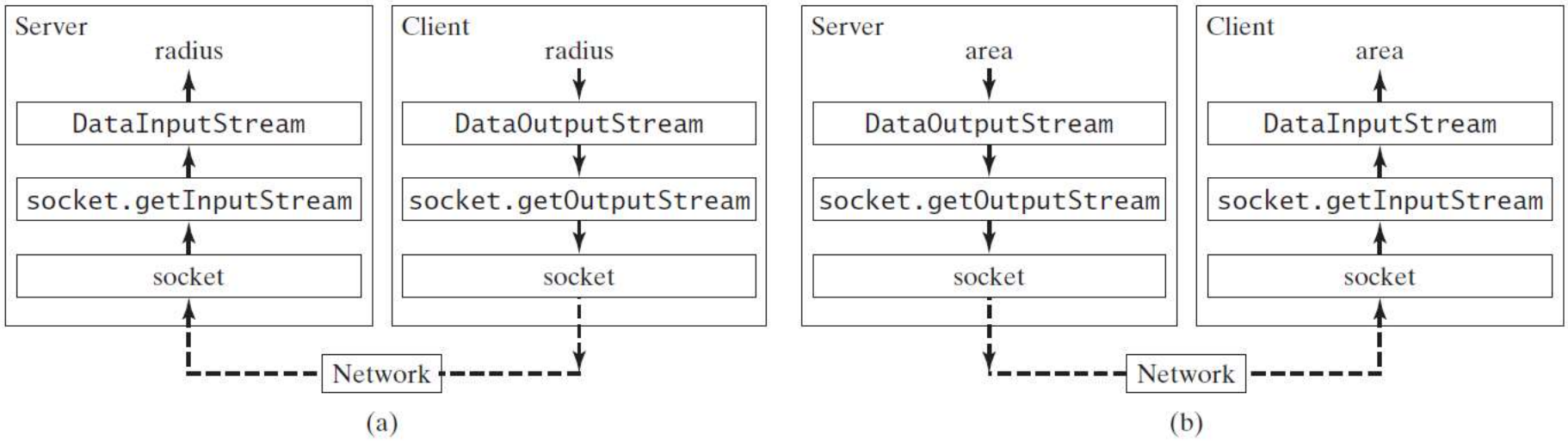
OutputStream output = socket.getOutputStream();

A Client/Server Example

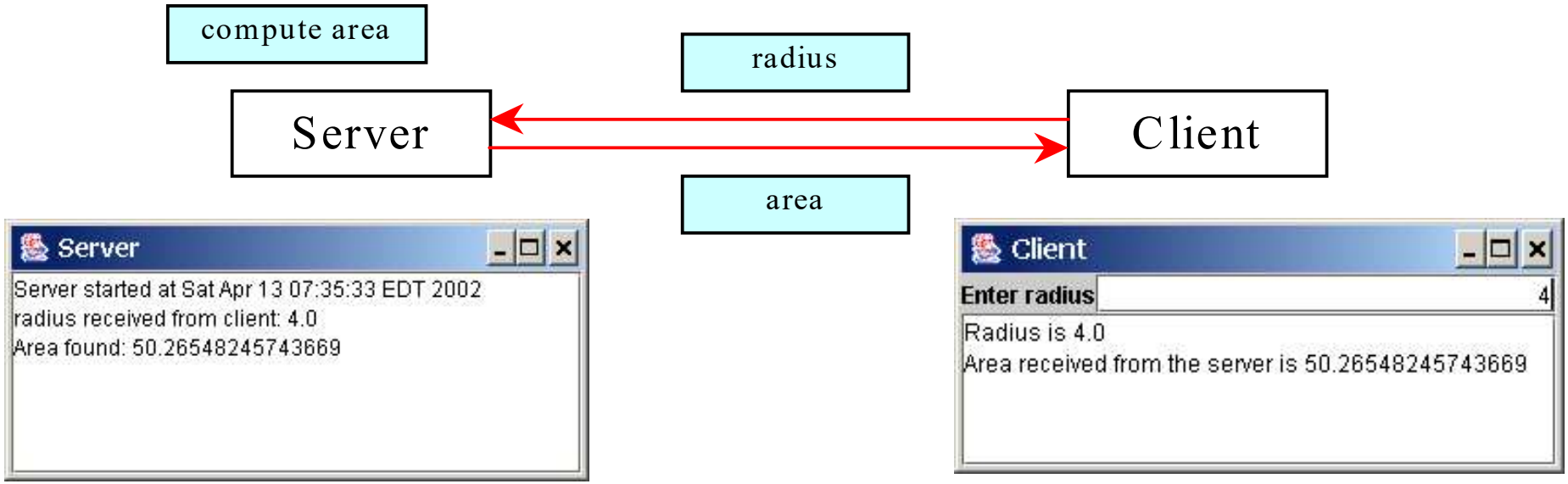
- Problem: Write a client to send data to a server. The server receives the data, uses it to produce a result, and then sends the result back to the client. The client displays the result on the console. In this example, the data sent from the client is the radius of a circle, and the result produced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



A Client/Server Example, cont.



A Client/Server Example, cont.



Server

Start Server

Client

Start Client

Note: Start the server, then the client.

The InetAddress Class

Occasionally, you would like to know who is connecting to the server. You can use the InetAddress class to find the client's host name and IP address. The InetAddress class models an IP address. You 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());
```

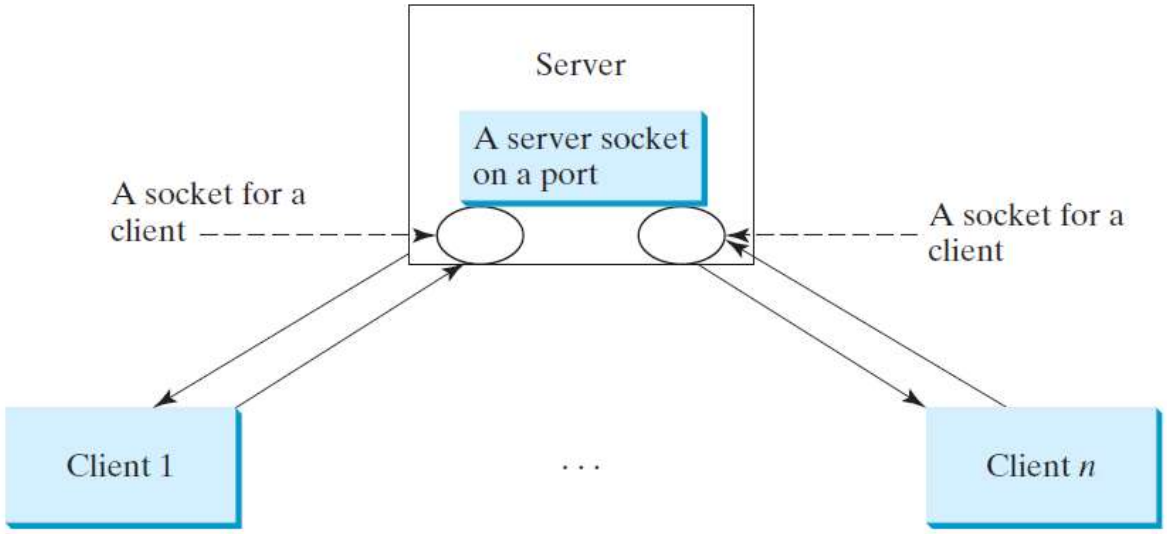
Serving Multiple Clients

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();
}
```

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



MultiThreadServer

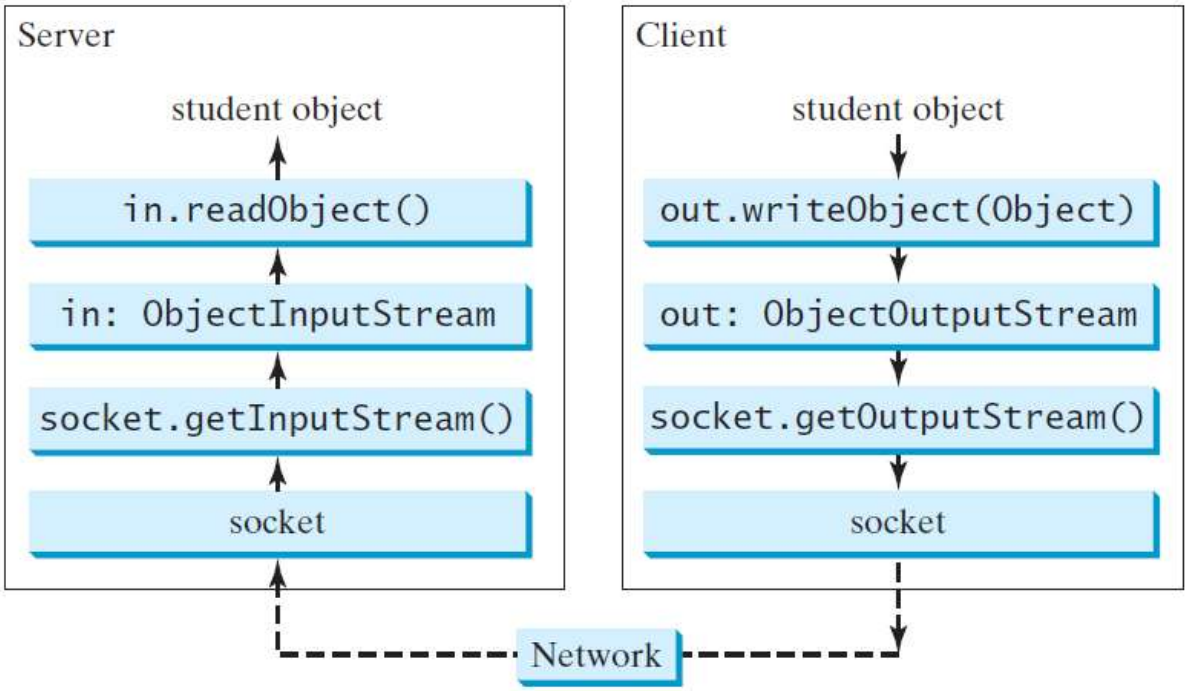
Start Server

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.



StudentAddres Class

StudentServer

StudentClient

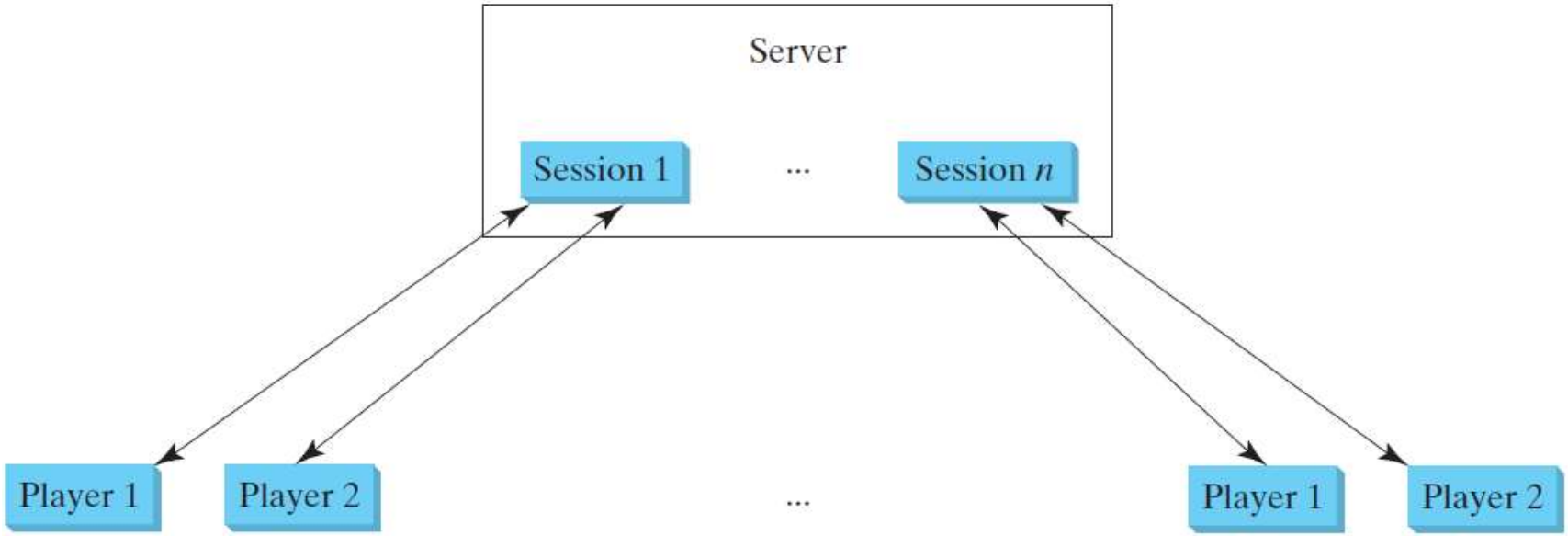
Start Server

Start Client

Note: Start the server first, then the client.

Optional

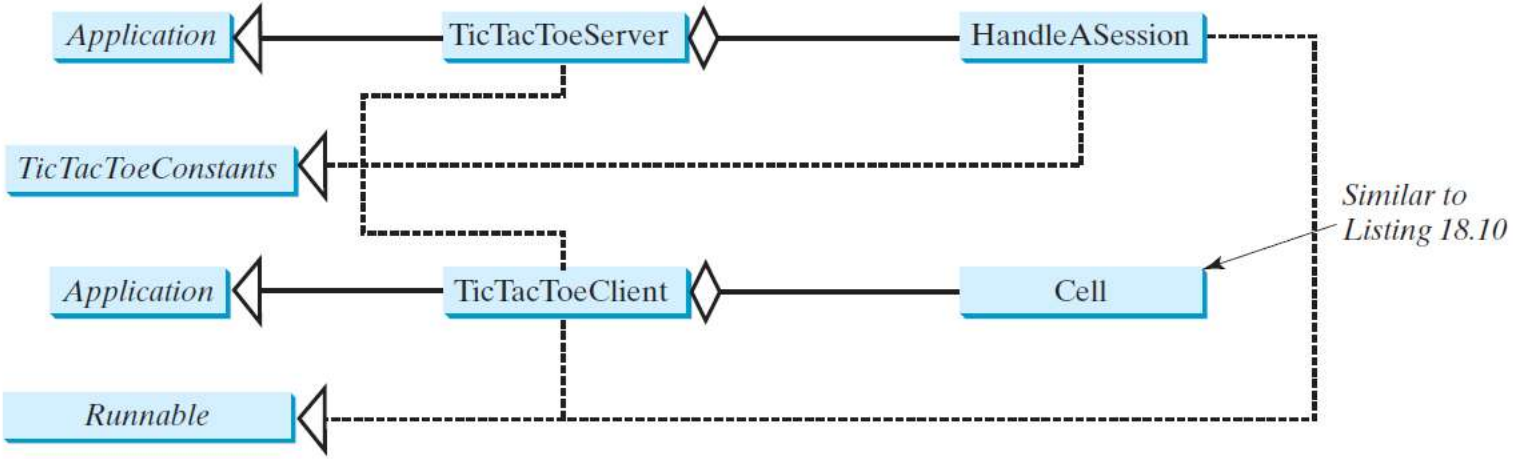
Case Studies: Distributed TicTacToe Games



TicTacToeServer Run

TicTacToeClient Run

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
```

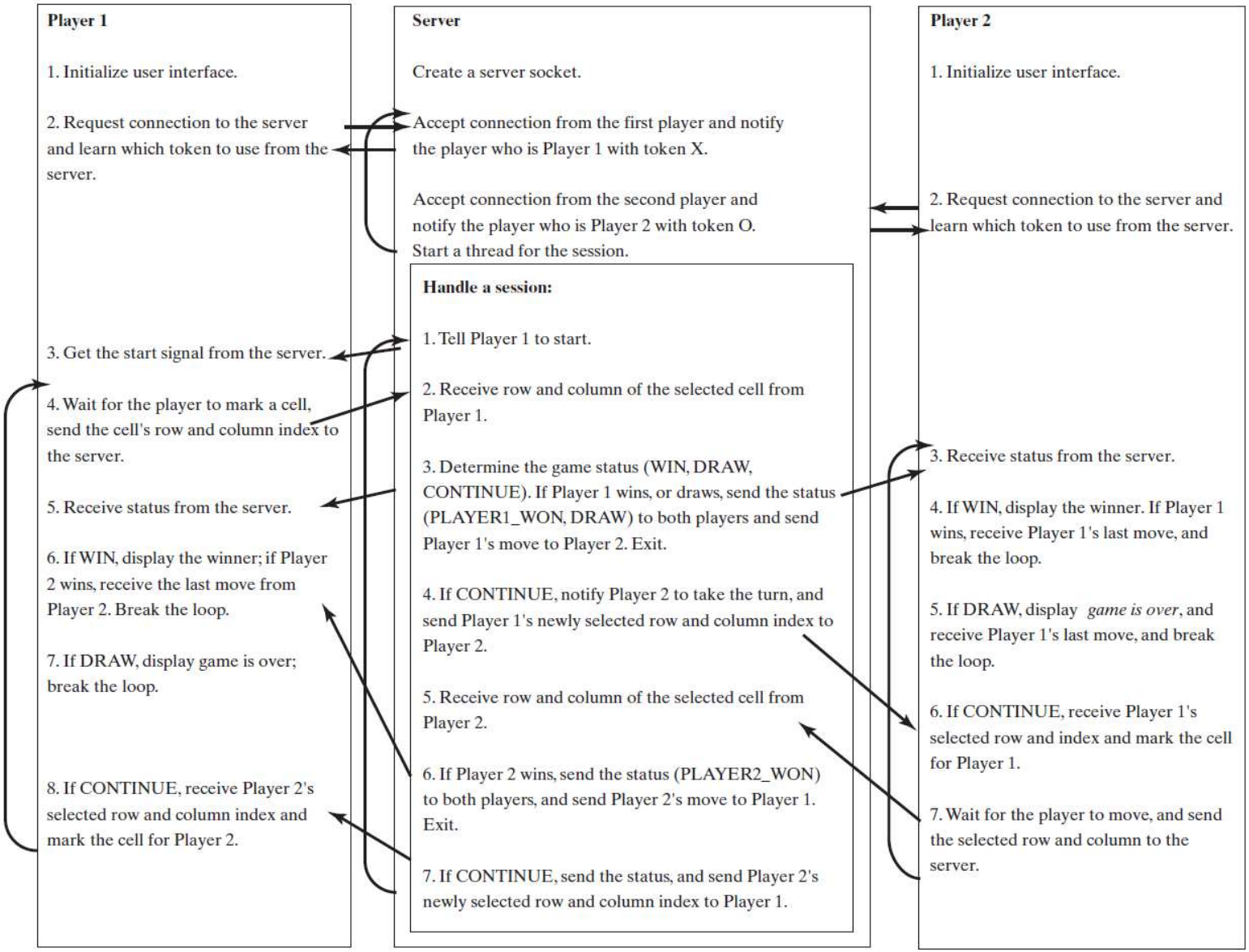
HandleASession

```
-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



Companion Website

Stream Socket vs. Datagram Socket

Stream socket

- A dedicated point-to-point channel between a client and server.
- Use TCP (Transmission Control Protocol) for data transmission.
- Lossless and reliable.
- Sent and received in the same order.

Datagram socket

- No dedicated point-to-point channel between a client and server.
- Use UDP (User Datagram Protocol) for data transmission.
- May lose data and not 100% reliable.
- Data may not received in the same order as sent.

Companion Website

DatagramPacket

The DatagramPacket class represents a datagram packet. Datagram packets are used to implement a connectionless packet delivery service. Each message is routed from one machine to another based solely on information contained within the packet.

java.net.DatagramPacket	
length: int	A JavaBeans property to specify the length of buffer.
address: InetAddress	A JavaBeans property to specify the address of the machine where the package is sent or received.
port: int	A JavaBeans property to specify the port of the machine where the package is sent or received.
+DatagramPacket(buf: byte[], length: int, host: InetAddress, port: int)	Constructs a datagram packet in a byte array buf of the specified length with the host and the port for which the packet is sent. This constructor is often used to construct a packet for delivery from a client.
+DatagramPacket(buf: byte[], length: int)	Constructs a datagram packet in a byte array buf of the specified length.
+getData(): byte[]	Returns the data from the package.
+setData(buf: byte[]): void	Sets the data in the package.

Companion Website

DatagramSocket

DatagramSocket

The DatagramSocket class represents a socket for sending and receiving datagram packets. A datagram socket is the sending or receiving point for a packet delivery service. Each packet sent or received on a datagram socket is individually addressed and routed. Multiple packets sent from one machine to another may be routed differently, and may arrive in any order.

Create a server DatagramSocket

To create a server DatagramSocket, use the constructor DatagramSocket(int port), which binds the socket with the specified port on the local host machine.

Create a client DatagramSocket

To create a client DatagramSocket, use the constructor DatagramSocket(), which binds the socket with any available port on the local host machine.

Companion Website

Sending and Receiving a DatagramSocket

Sending

To send data, you need to create a packet, fill in the contents, specify the Internet address and port number for the receiver, and invoke the send(packet) method on a DatagramSocket.

Receiving

To receive data, create an empty packet and invoke the receive(packet) method on a DatagramSocket.

Companion Website

Datagram Programming

Datagram programming is different from stream socket programming in the sense that there is no concept of a ServerSocket for datagrams. Both client and server use DatagramSocket to send and receive packets.

Designate one a server

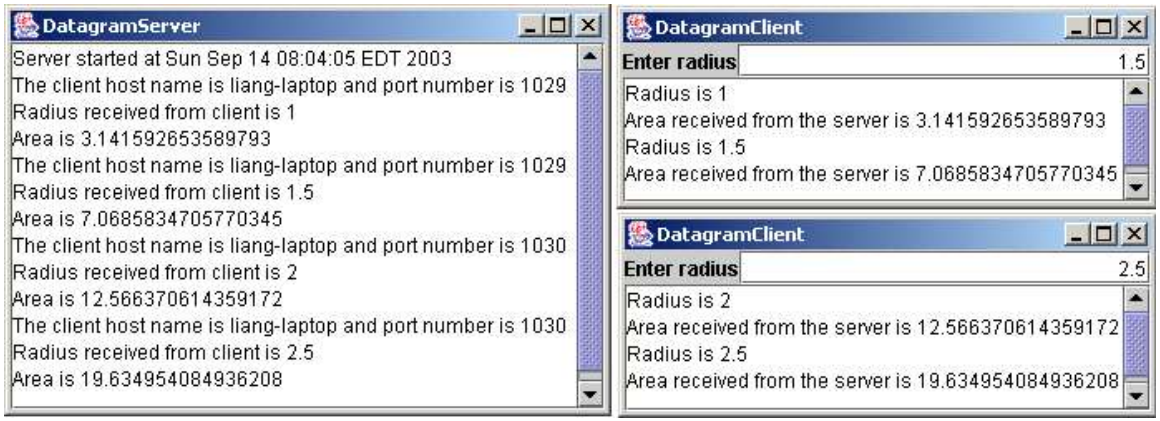
```
DatagramServer
  DatagramSocket socket;
  socket = new DatagramSocket(8000);

  byte[] buf = new byte[256];

  DatagramPacket receivePacket = new
     DatagramPacket(buf, bef.length)

  socket.receive(receivePacket);

  get data from buf or receivePacket.getData();

  DatagramPacket sendPacket = new
    DatagramPacket(buf, bef.length)

  fill in the contents in buf;
  socket.send(sendPacket);
```

```
DatagramClient
  DatagramSocket socket;
  socket = new DatagramSocket();

  byte[] buf = new byte[256];

  InetAddress address = new
     InetAddress(serverName);

  DatagramPacket sendPacket = new
     DatagramPacket(buf,  bef.length, address, 8000)

  fill in the contents in buf;
  socket.send(sendPacket);

  DatagramPacket receivePacket = new
     DatagramPacket(buf,  bef.length)

  socket.receive(receivePacket);

  get data from buf or receivePacket.getData();
```

Companion Website

Example: A Client/Server Example

Section 31.2 presents a client program and a server program using socket streams. The client sends radius to a server. The server receives the data, uses them to find the area, and then sends the area to the client. Rewrite the program using datagram sockets.



DatagramServer

Run

DatagramClient

Run

Note: Start the server, then the client.