# Networking



Shing-Chi Cheung Computer Science and Engineering HKUST

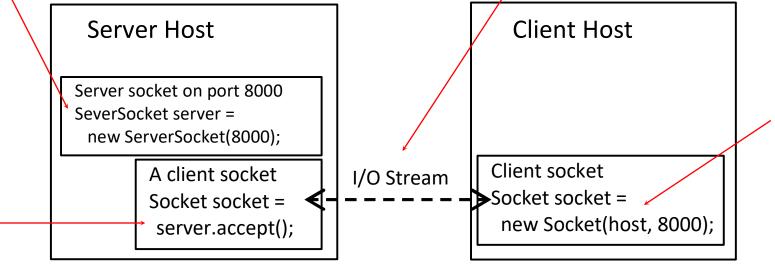
Sep-18

## 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.

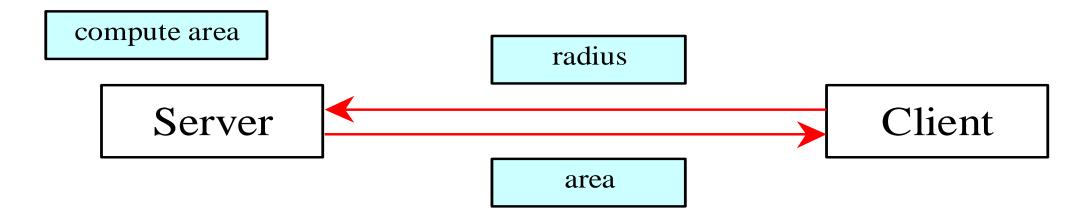
## Data Transmission through Sockets

Server Client int port = 8000; int port = 8000; DataInputStream in: String host = "localhost" DataOutputStream out; DataInputStream in; ServerSocket server: DataOutputStream out; Socket socket: Socket socket: Connection Request server = new ServerSocket(port); socket = server.accept(); ← socket = new Socket(host, port); in = new DataInputStream in = new DataInputStream (socket.getInputStream()); (socket.getInputStream()); out = new DataOutStream out = new DataOutputStream I/O (socket.getOutputStream()); (socket.getOutputStream()); Streams System.out.println(in.readDouble()); out.writeDouble(aNumber); out.writeDouble(aNumber); -System.out.println(in.readDouble());

InputStream input = socket.getInputStream();

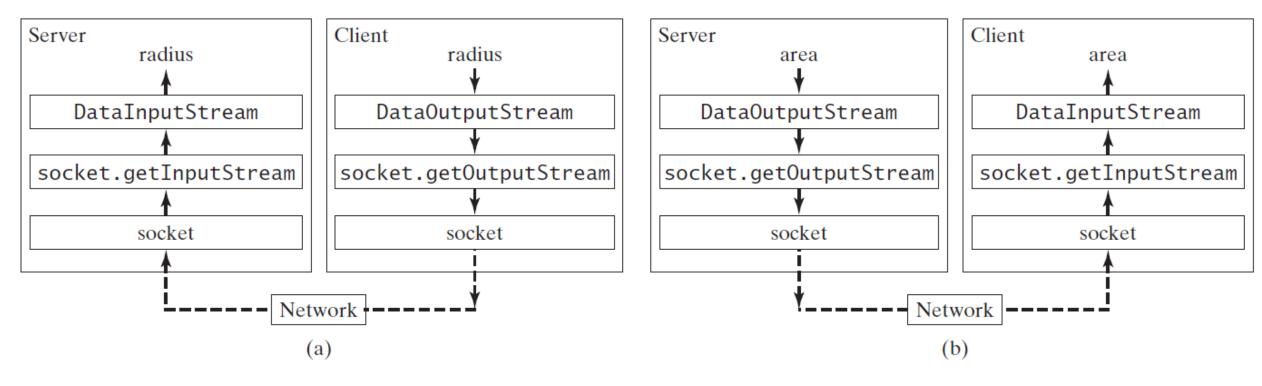
OutputStream output = socket.getOutputStream();

## A Client/Server Example

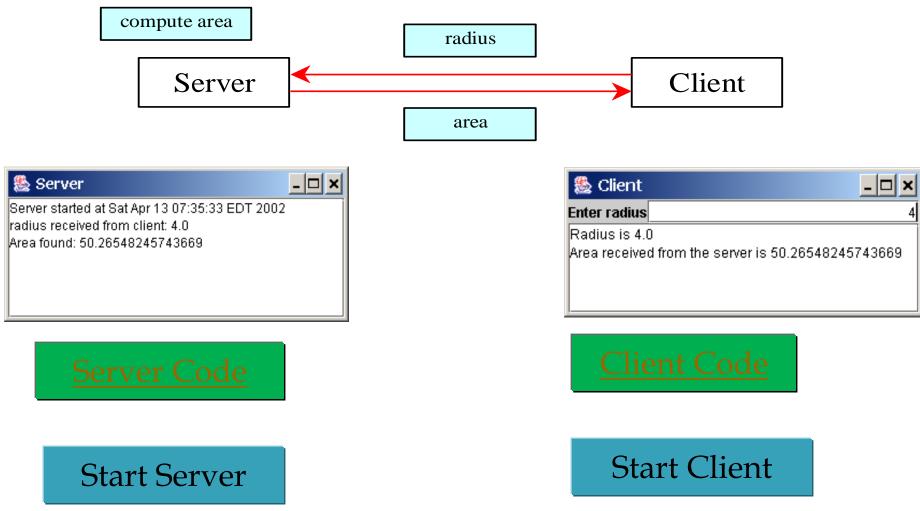


■ Problem: Write a client to send data to a server. The server receives the data, uses it to produce a result, and then sends the result back to the client. The client displays the result on the console. In this example, the data sent from the client is the radius of a circle, and the result produced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

## A Client/Server Example, cont.



## A Client/Server Example, cont.



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:

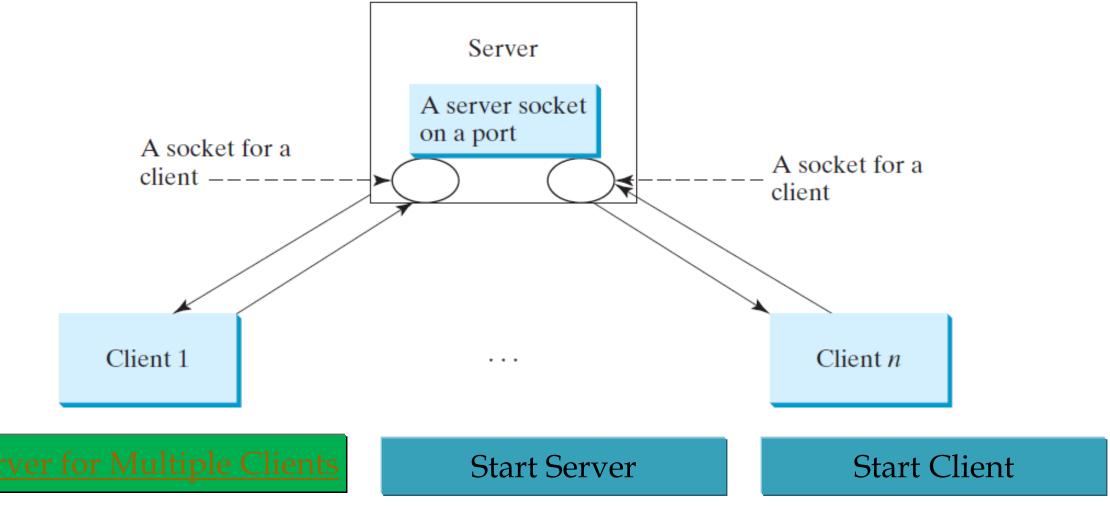
```
while (true) {
   Socket socket = serverSocket.accept();
   Thread thread = new ThreadClass(socket);
   thread.start();
}
```

## 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 <u>while</u> 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.

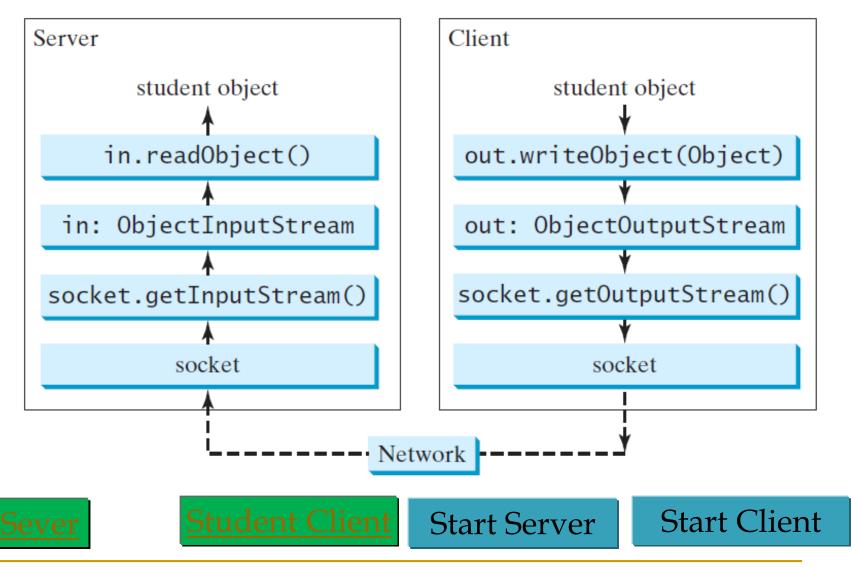
# **Example: Serving Multiple Clients**



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

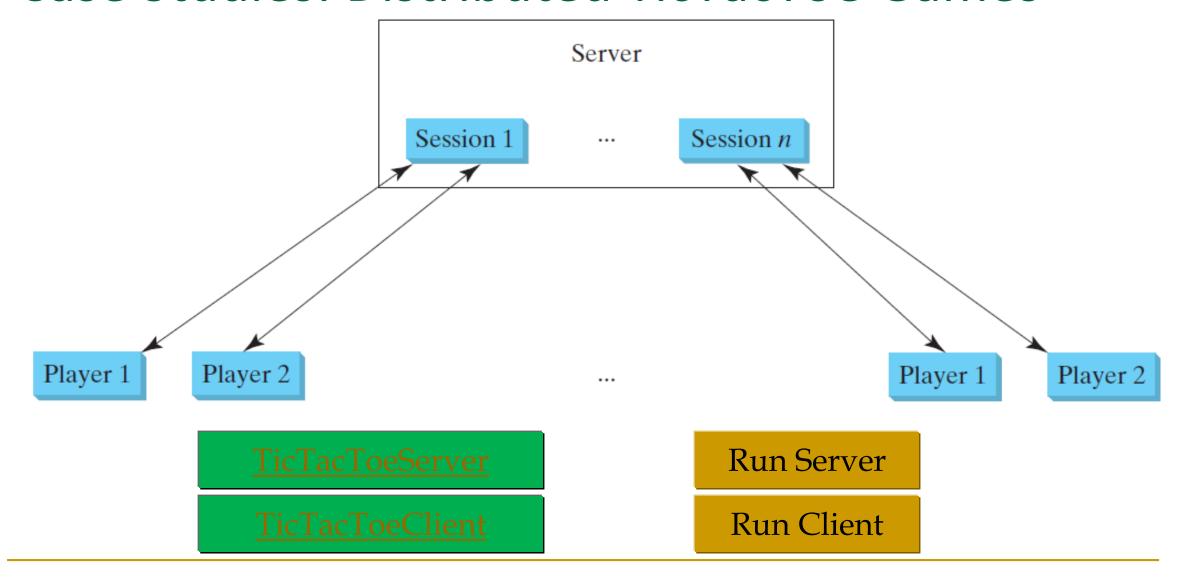
### **Example: Passing Objects in Network Programs**

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

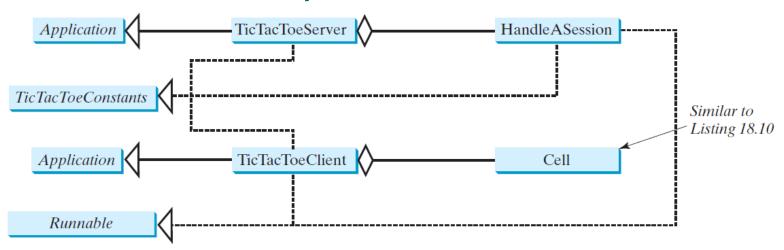


Note: Start the server first, then the client.

## Case Studies: Distributed TicTacToe Games



## 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

### **Handle A Session**

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

-continueroriay: boorean

+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

mark the cell for Player 2.

#### Player 1 Server Player 2 1. Initialize user interface. 1. Initialize user interface. Create a server socket. 2. Request connection to the server Accept connection from the first player and notify and learn which token to use from the the player who is Player 1 with token X. server. Accept connection from the second player and 2. Request connection to the server and notify the player who is Player 2 with token O. learn which token to use from the server. Start a thread for the session. Handle a session: 1. Tell Player 1 to start. 3. Get the start signal from the server. 2. Receive row and column of the selected cell from 4. Wait for the player to mark a cell, Player 1. send the cell's row and column index to 3. Receive status from the server. the server. 3. Determine the game status (WIN, DRAW, CONTINUE). If Player 1 wins, or draws, send the status -5. Receive status from the server. 4. If WIN, display the winner. If Player 1 (PLAYER1\_WON, DRAW) to both players and send wins, receive Player 1's last move, and Player 1's move to Player 2. Exit. 6. If WIN, display the winner; if Player break the loop. 2 wins, receive the last move from 4. If CONTINUE, notify Player 2 to take the turn, and Player 2. Break the loop. 5. If DRAW, display game is over, and send Player 1's newly selected row and column index to receive Player 1's last move, and break Player 2. 7. If DRAW, display game is over; the loop. break the loop. 5. Receive row and column of the selected cell from 6. If CONTINUE, receive Player 1's Player 2. selected row and index and mark the cell for Player 1. 6. If Player 2 wins, send the status (PLAYER2\_WON) 8. If CONTINUE, receive Player 2's to both players, and send Player 2's move to Player 1. selected row and column index and 7. Wait for the player to move, and send

7. If CONTINUE, send the status, and send Player 2's newly selected row and column index to Player 1.

Exit.

the selected row and column to the

server.