Lecture 11: Networking in Java (Ch 33)

Adapted by Fangzhen Lin for COMP3021 from Y. Danial Liang's PowerPoints for Introduction to Java Programming, Comprehensive Version, 9/E, Pearson, 2013.

Objectives

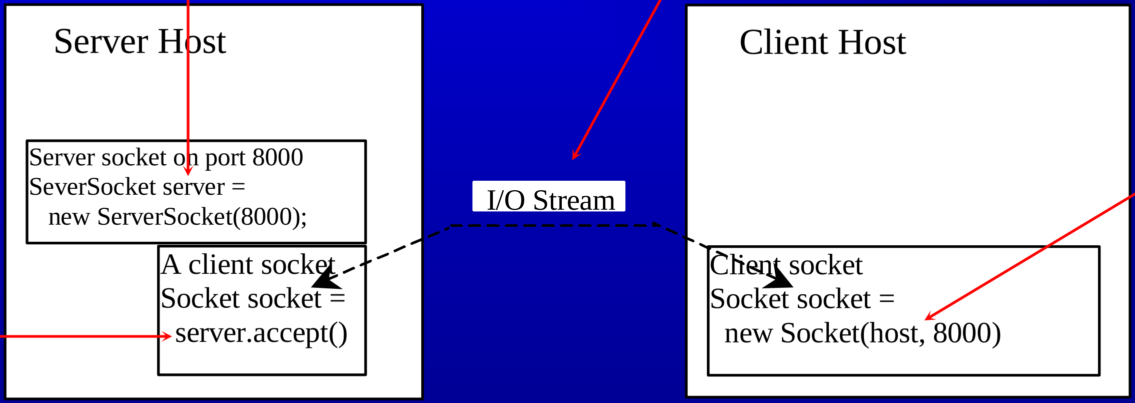
- To explain terms: TCP, IP, domain name, domain name server, stream-based communications, and packet-based communications (§ 33.2).
- To create servers using server sockets (§ 33.2.1) and clients using client sockets (§ 33.2.2).
- To implement Java networking programs using stream sockets (§ 33.2.3).
- To develop an example of a client/server application (§ 33.2.4).
- To obtain Internet addresses using the **InetAddress** class (§ 33.3).
- To develop servers for multiple clients (§ 33.4).
- To develop applets that communicate with the server (§ 33.5).
- To send and receive objects on a network (§ 33.6).
- To develop an interactive tic-tac-toe game played on the Internet (§ 33.7).

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.

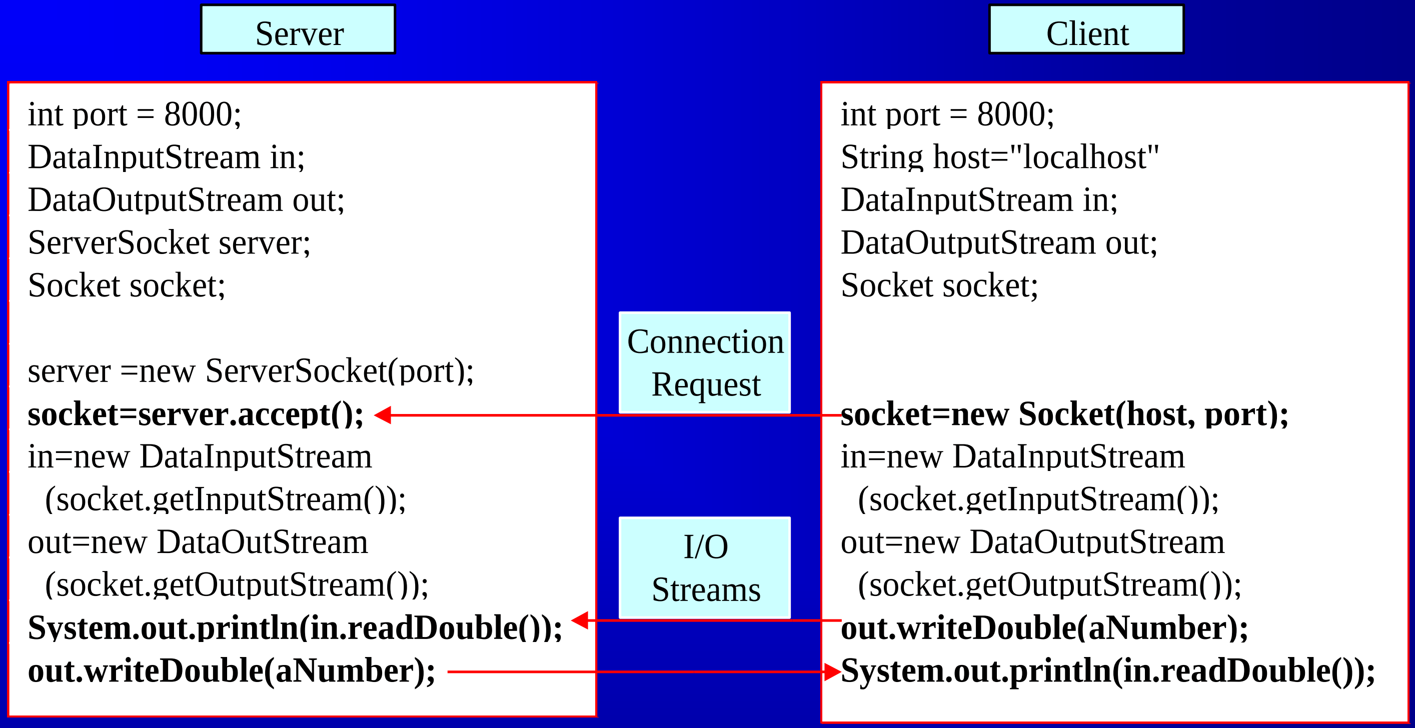
After the server accepts the connection, communication between server and client is conducted the same as for I/O streams.

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:

```
int port = 8000;
DataInputStream in;
DataOutputStream out;
ServerSocket server;
Socket socket;

server =new ServerSocket(port);
socket=server.accept();
in=new DataInputStream
  (socket.getInputStream());
out=new DataOutStream
  (socket.getOutputStream());
System.out.println(in.readDouble());
out.writeDouble(aNumber);
```

Client:

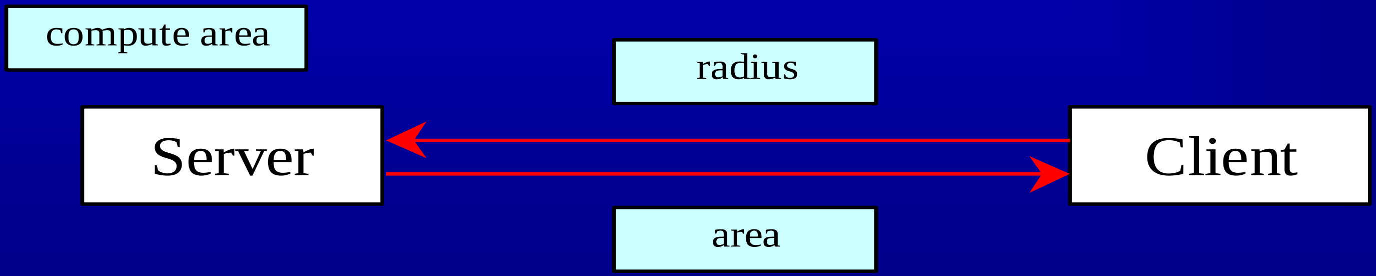
```
int port = 8000;
String host="localhost"
DataInputStream in;
DataOutputStream out;
Socket socket;


socket=new Socket(host, port);
in=new DataInputStream
  (socket.getInputStream());
out=new DataOutputStream
  (socket.getOutputStream());
out.writeDouble(aNumber);
System.out.println(in.readDouble());
```

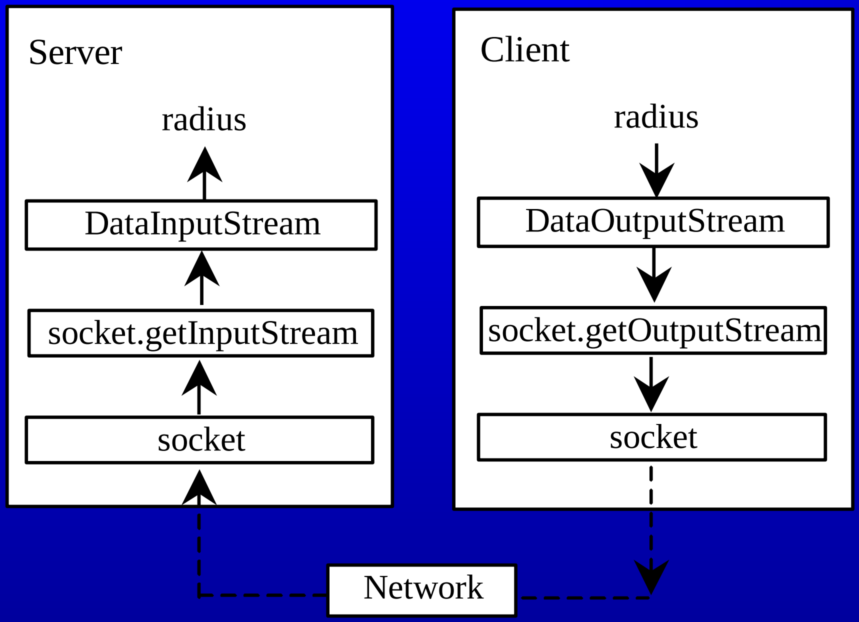
```
InputStream input = socket.getInputStream();
OutputStream output = socket.getOutputStream();
```

A Client/Server Example

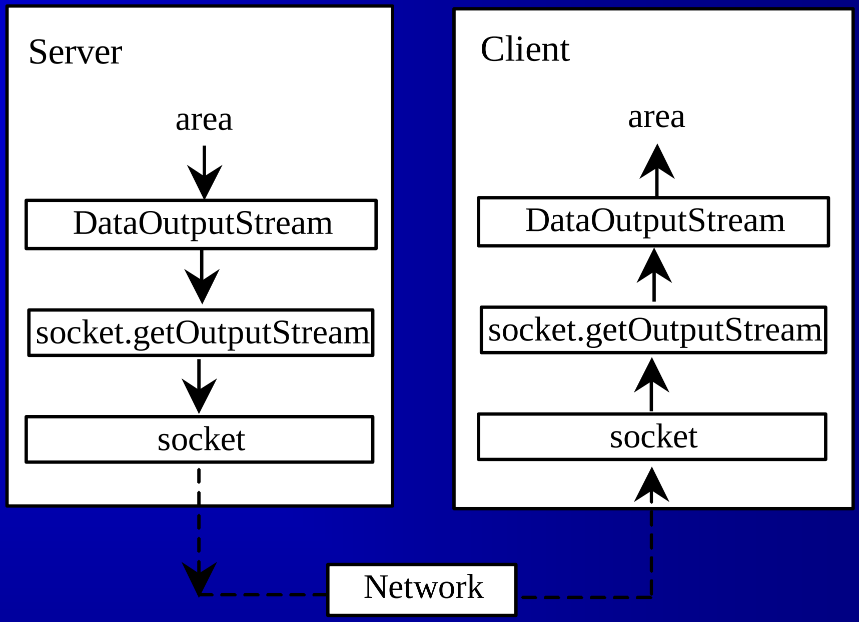
- Problem: Write a client to send data to a server. The server receives the data, uses it to produce a result, and then sends the result back to the client. The client displays the result on the console. In this example, the data sent from the client is the radius of a circle, and the result produced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



A Client/Server Example, cont.

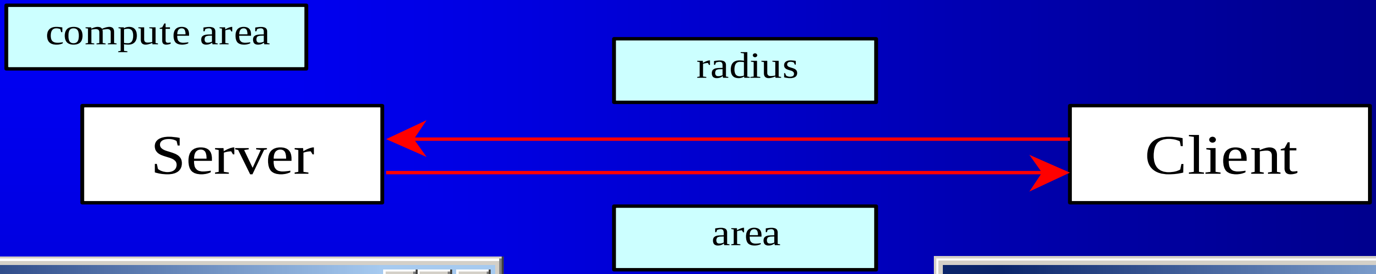


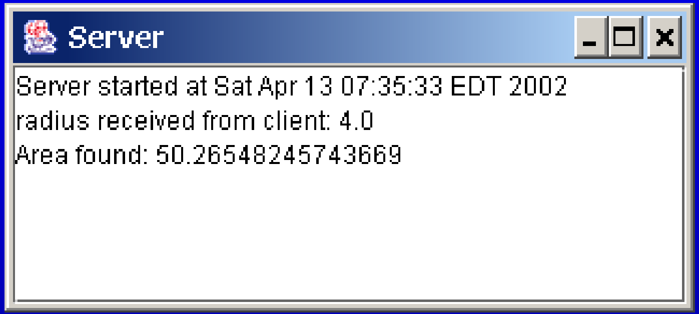
(A)



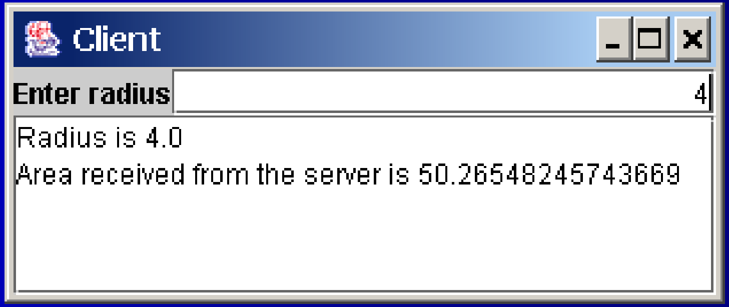
(B)

A Client/Server Example, cont.





```
Server started at Sat Apr 13 07:35:33 EDT 2002
radius received from client: 4.0
Area found: 50.26548245743669
```



Enter radius 4

```
Radius is 4.0
Area received from the server is 50.26548245743669
```

Server Code

Client Code

Start Server

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();
}
```

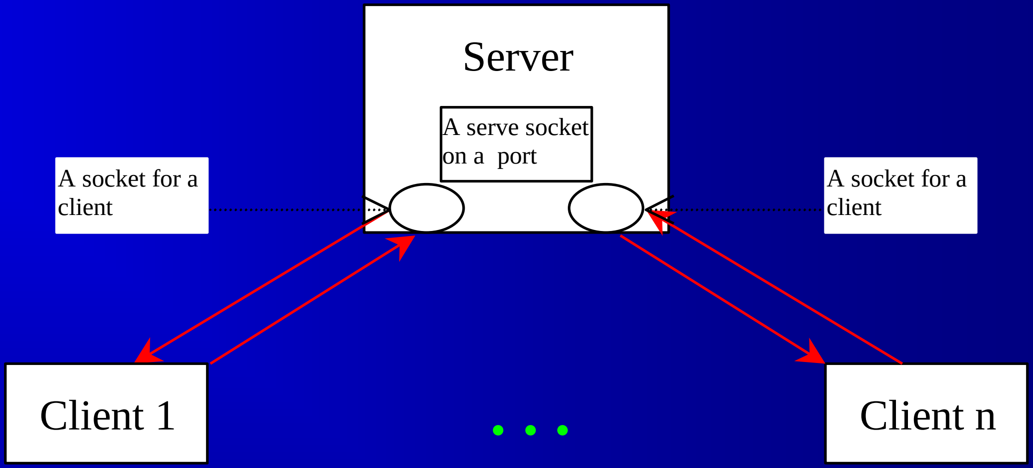
The server socket can have many connections. Each iteration of the while loop creates a new connection. Whenever a connection is established, a new thread is created to handle communication between the server and the new client; and this allows multiple connections to run at the same time.

Example: Serving Multiple Clients

Server for Multiple Clients

Start Server

Start Client



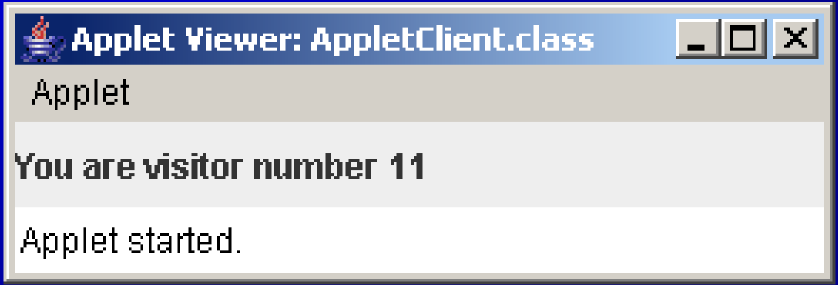
Note: Start the server first, then start multiple clients.

Applet Clients

Due to security constraints, applets can only connect to the host from which they were loaded. Therefore, the HTML file must be located on the machine on which the server is running.

Example: Creating Applet Clients

Write an applet that shows the number of visits made to a Web page. The count should be stored in a file on the server side. Every time the page is visited or reloaded, the applet sends a request to the server, and the server increases the count and sends it to the applet. The applet then displays the new count in a message, such as **You are visitor number 11**.

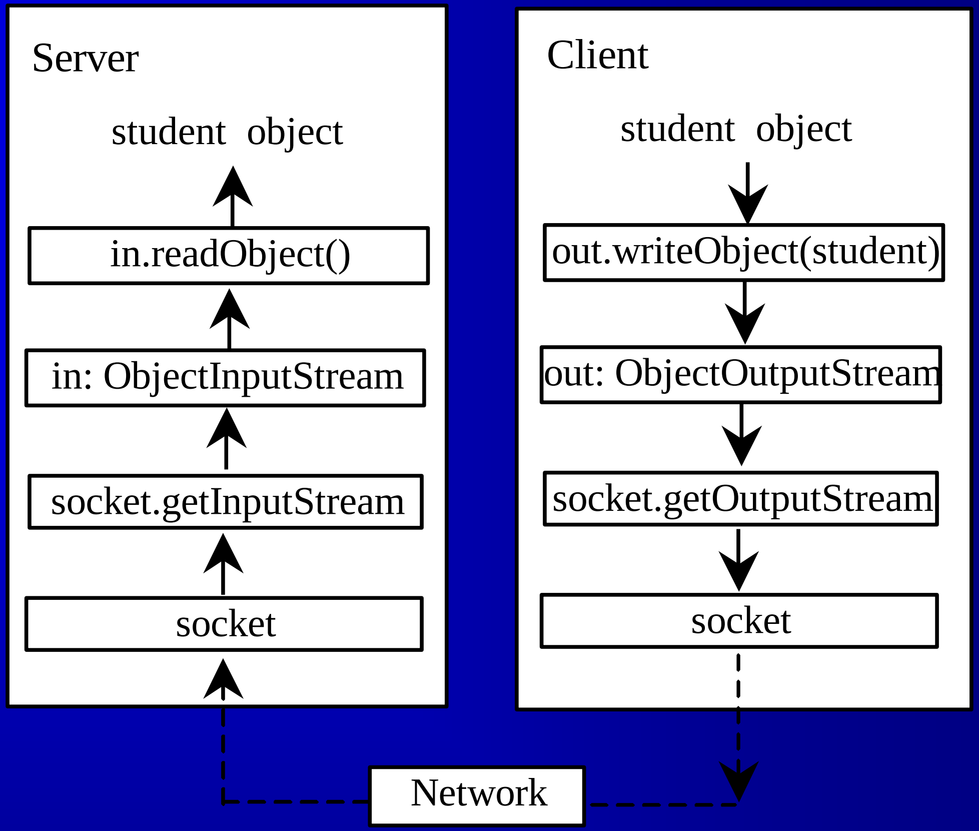


```
You are visitor number 11
Applet started.
```

CountServer AppletClient Start Server Start Client

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

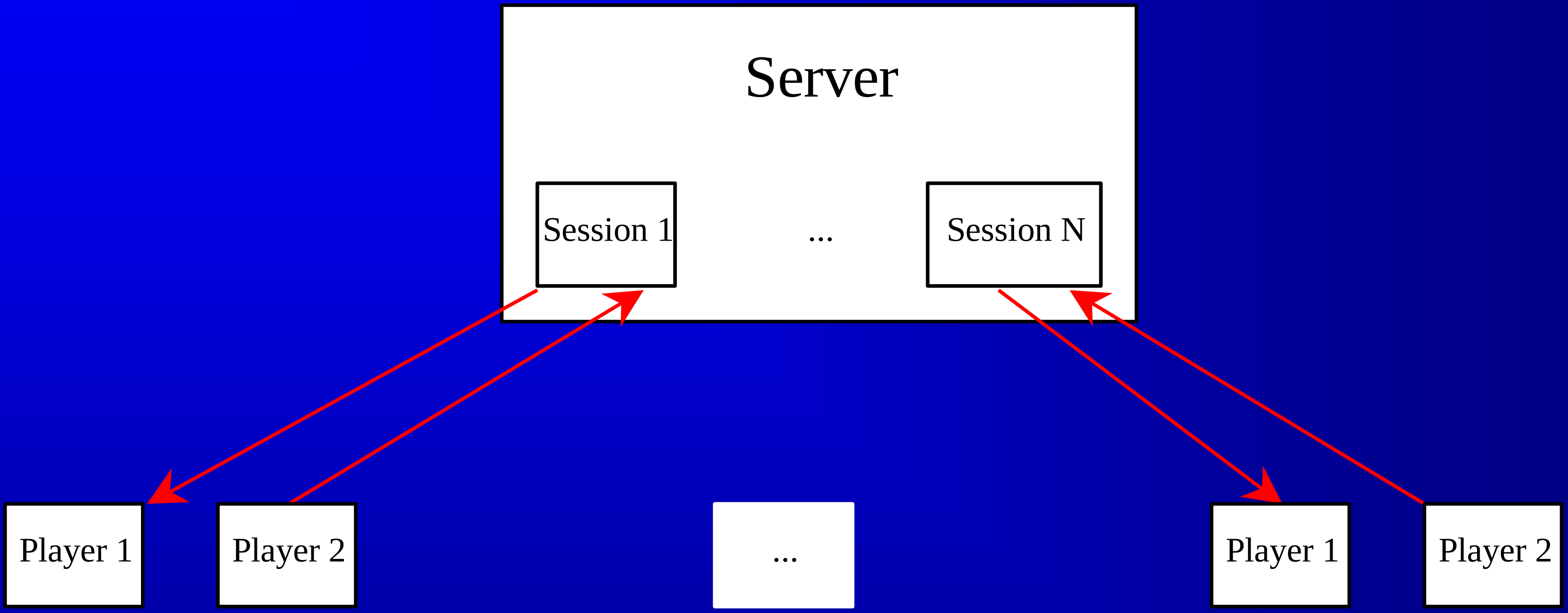
Student Client

Start Server

Start Client

Note: Start the server first, then the client.

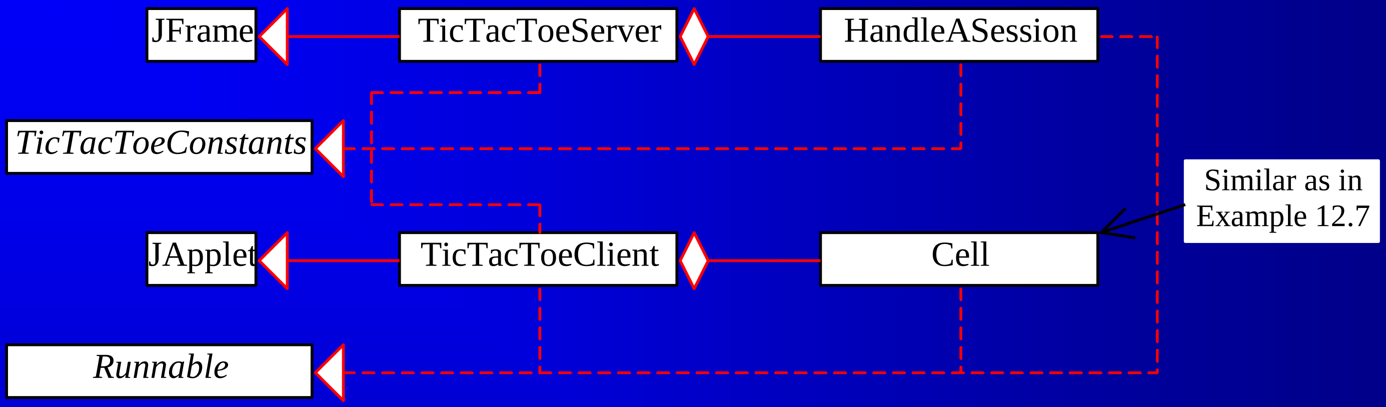
Case Studies: Distributed TicTacToe Games



TicTacToeServer

TicTacToeClient

Distributed TicTacToe, cont.



TicTacToeServer

+main(args: String[]): void

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

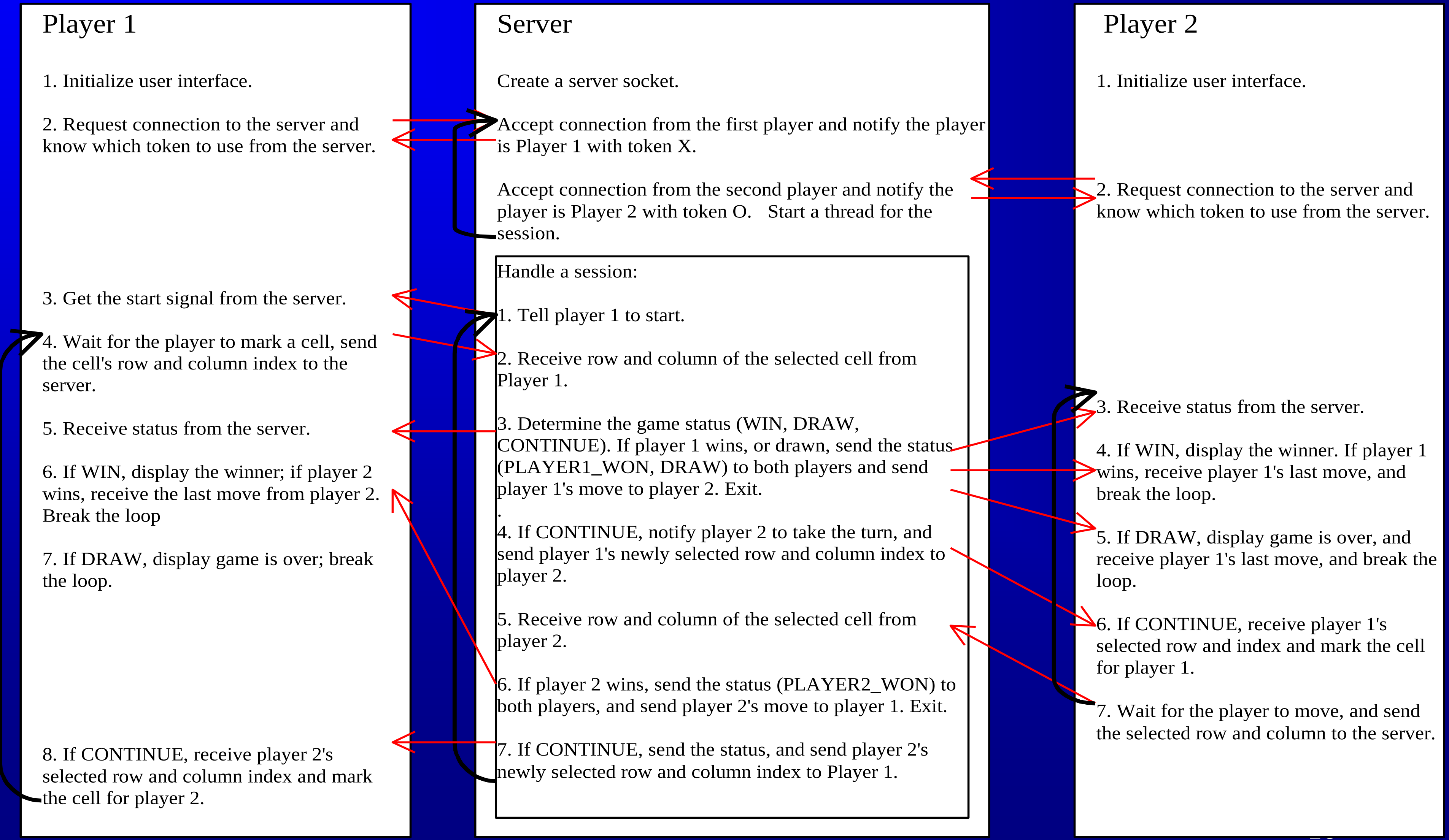
+run(): void
-isWon(): boolean
-isFull(): boolean
-sendMove(out: DataOuputStream, row: int, column: int): void

TicTacToeClient

-myTurn: boolean
-myToken: char
-otherToken: char
-cell: Cell[][]
-continueToPlay: boolean
-rowSelected: int
-columnSelected: int
-isFromServer: DataInputStream
-osToServer: DataOutputStream
-waiting: boolean

+run(): void
-connectToServer(): void
-recieveMove(): void
-sendMove(): void
-receiveInfoFromServer(): void
-waitForPlayerAction(): void

Distributed TicTacToe Game



Player 1

1. Initialize user interface.

2. Request connection to the server and know which token to use from the server.

3. Get the start signal from the server.

4. Wait for the player to mark a cell, send the cell's row and column index to the server.

5. Receive status from the server.

6. If WIN, display the winner; if player 2 wins, receive the last move from player 2. Break the loop

7. If DRAW, display game is over; break the loop.

8. If CONTINUE, receive player 2's selected row and column index and mark the cell for player 2.

Server

Create a server socket.

Accept connection from the first player and notify the player is Player 1 with token X.

Accept connection from the second player and notify the player is Player 2 with token O. Start a thread for the session.

Handle a session:

1. Tell player 1 to start.

2. Receive row and column of the selected cell from Player 1.

3. Determine the game status (WIN, DRAW, CONTINUE). If player 1 wins, or drawn, send the status (PLAYER1_WON, DRAW) to both players and send player 1's move to player 2. Exit.

.

4. If CONTINUE, notify player 2 to take the turn, and send player 1's newly selected row and column index to player 2.

5. Receive row and column of the selected cell from player 2.

6. If player 2 wins, send the status (PLAYER2_WON) to both players, and send player 2's move to player 1. Exit.

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

Player 2

1. Initialize user interface.

2. Request connection to the server and know which token to use from the server.

3. Receive status from the server.

4. If WIN, display the winner. If player 1 wins, receive player 1's last move, and break the loop.

5. If DRAW, display game is over, and receive player 1's last move, and break the loop.

6. If CONTINUE, receive player 1's selected row and index and mark the cell for player 1.

7. Wait for the player to move, and send the selected row and column to the server.

```
public interface TicTacToeConstants {
    public static int PLAYER1 = 1;
    public static int PLAYER2 = 2;
    public static int PLAYER1_WON = 1;
    public static int PLAYER2_WON = 2;
    public static int DRAW = 3;
    public static int CONTINUE = 4;
}
```

JEditorPane

Swing provides a GUI component named javax.swing.JEditorPane that can be used to display plain text, HTML, and RTF files automatically. So you don’t have to write code to explicit read data from the files. JEditorPane is a subclass of JTextComponent. Thus it inherits all the behavior and properties of JTextComponent.

To display the content of a file, use the setPage(URL) method as follows:

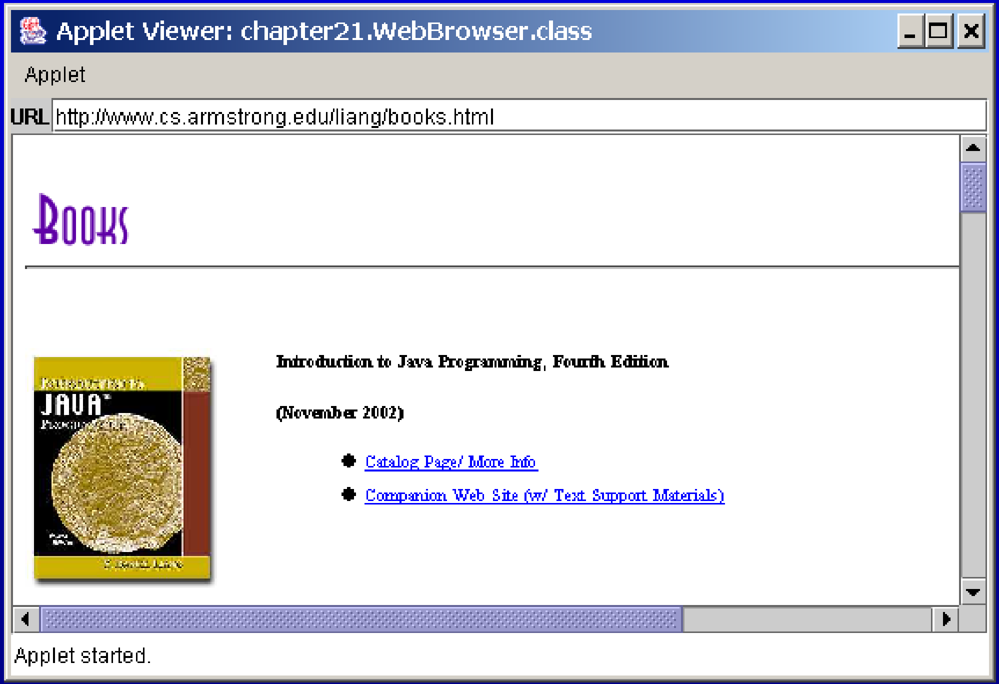
```
public void setPage(URL url) throws IOException
```

JEditorPane generates javax.swing.event.HyperlinkEvent when a hyperlink in the editor pane is clicked. Through this event, you can get the URL of the hyperlink and display it using the setPage(url) method.

Example: Creating a Web Browser

Viewing HTML Files Using the JEditorPane.

`JEditorPane` can be used to display HTML files.



URL http://www.cs.armstrong.edu/liang/books.html

Introduction to Java Programming, Fourth Edition

(November 2002)

- Catalog Page/ More Info
- Companion Web Site (w/ Text Support Materials)

Applet started.

WebBrowser

Companion Website

eam 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.