Chapter 4

Network Programming

Computer Science and Engineering Department@SoEEC

Advanced Programming(CSE 3312)

Key Objectives:

At the end of this chapter, you will be able to:

- understand about the Java network classes and interfaces
- know about the different types of sockets and their functionality
- create servers using server sockets and clients using client sockets
- develop a client/server applications.

4.1. Introduction

- Java provides:-
  - Stream-based communications
    - a process establishes a connection to another process. While the connection is in place, data flows between the processes
  - Connection-based protocol - Uses TCP
  - Packet-based communications – use UDP
    - Individual packets transmitted
- Client-server relationship
  - Client requests some action to be performed
  - Server performs the action and responds to client
  - Request-response model
    - Common implementation: Web browsers and Web servers

4.2. Java Networking

- Java Networking is a concept of connecting two or more computing devices together so that we can share resources.
- Java Network programming provides facility to share data between different computing devices.

Advantage of Java Networking

- ✓ sharing resources
- ✓ centralize software management

Java Networking terminology

- IP Address
- Protocol
- Port number
- MAC Address, connection-oriented and connection-less protocol

4.2.1. Java Socket Programming

- A socket, in network terminology, is an end-point for communication between two processes on the network.
- Java **Socket programming** is used for communication between the applications running on different JRE.
- Java Socket programming can be connection-oriented or connection-less.
- **Socket** and **ServerSocket** classes are used for connection-oriented socket programming and **DatagramSocket** and **DatagramPacket** classes are used for connection-less socket programming.
- The client in socket programming must know two information:

    1. IP Address of Server, and
    2. Port Number

4.3. Client/Server Computing

- Java provides the **ServerSocket** class for creating a server socket and the **Socket** class for creating a client socket. Two programs on the Internet communicate through a server socket and a client socket using I/O streams.
- Networking is tightly integrated in Java. The Java API provides the classes for creating sockets to facilitate program communications over the Internet. Sockets are the endpoints of logical connections between two hosts and can be used to send and receive data. Java treats socket communications much as it treats I/O operations; thus, programs can read from or write to sockets as easily as they can read from or write to files.

.... Client/Server Computing

- Network programming usually involves a server and one or more clients. The client sends requests to the server, and the server responds. The client begins by attempting to establish a connection to the server. The server can accept or deny the connection. Once a connection is established, the client and the server communicate through sockets.
- The server must be running when a client attempts to connect to the server. The server waits for a connection request from the client. The statements needed to create sockets on a server and on a client are shown in Figure 4.1.

4.3.1. Server Sockets

- 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. The port identifies the TCP service on the socket. Port numbers range from **0** to **65536**, but port numbers 0 to 1024 are reserved for privileged services.
- For instance, the email server runs on port 25, and the Web server usually runs on port 80. You can choose any port number that is not currently used by other programs.
- The following statement creates a server socket **serverSocket:**

```
ServerSocket serverSocket = new ServerSocket(port);
```

... Server Sockets

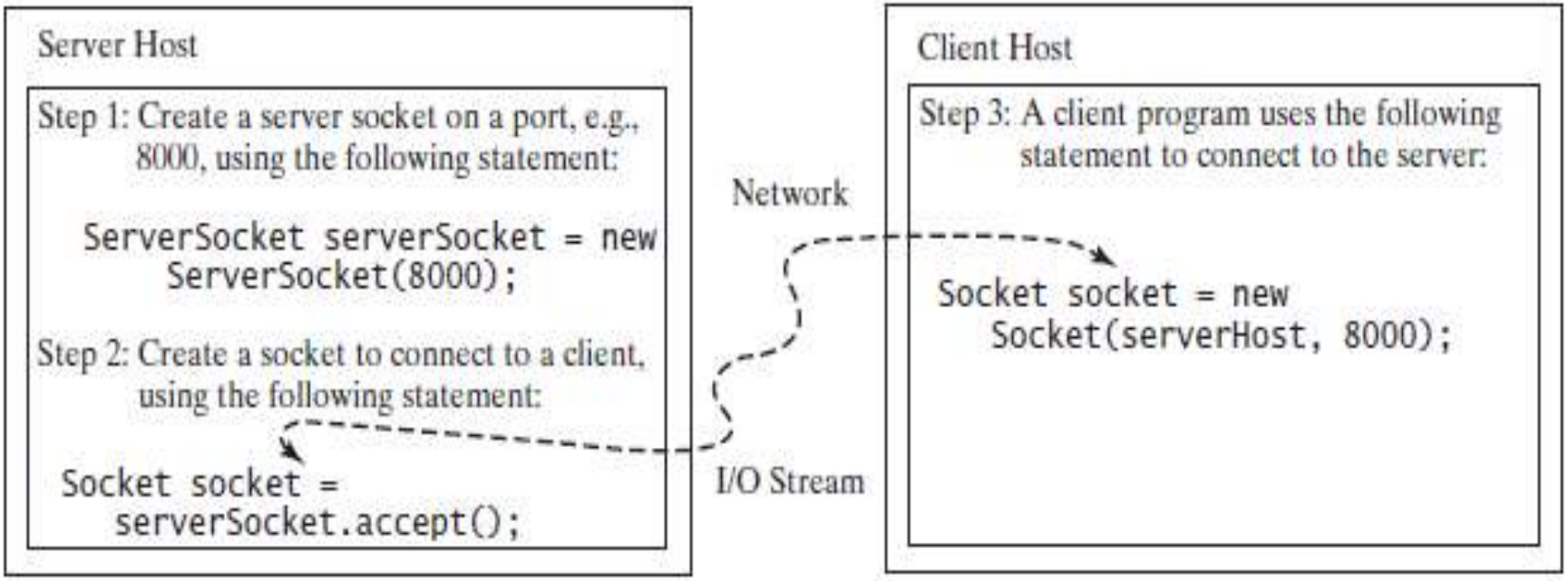


Figure 4.1: The server creates a server socket and, once a connection to a client is established, connects to the client with a client socket.

... Server Sockets

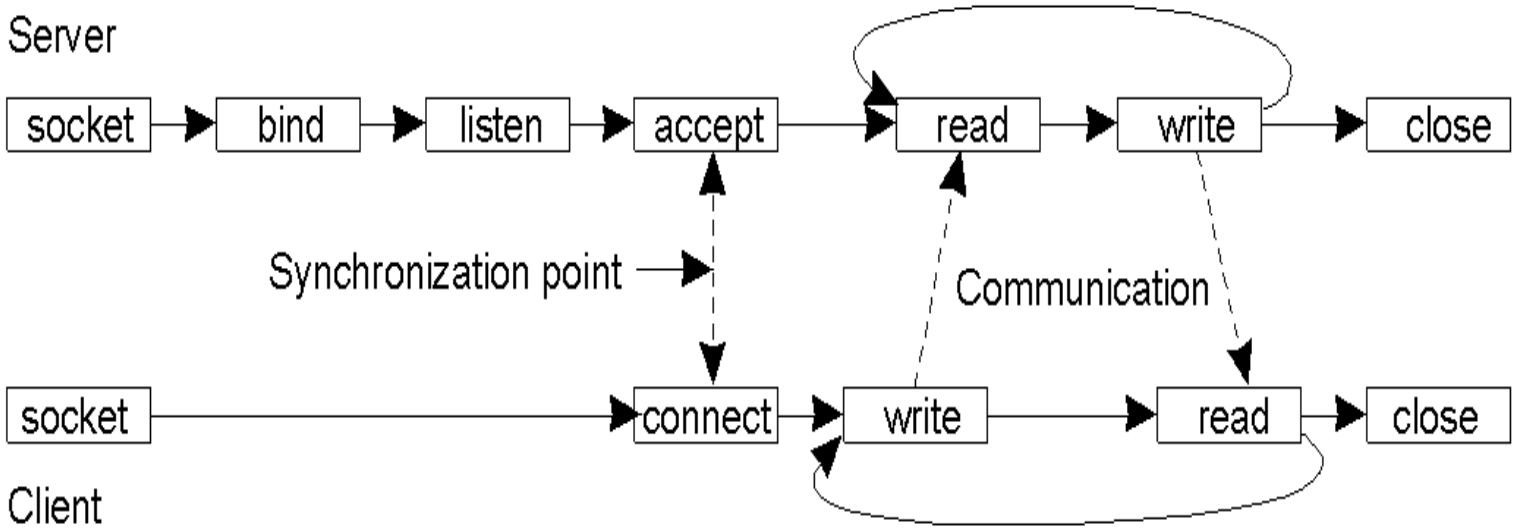


Fig 4.2. connection-oriented communication pattern using sockets

4.3.2. Client Sockets

- After a server socket is created, the server can use the following statement to listen for connections:

```
Socket socket = serverSocket.accept();
```

- This statement waits until a client connects to the server socket. The client issues the following statement to request a connection to a server:

```
Socket socket = new Socket(serverName, port);
```

- This statement opens a socket so that the client program can communicate with the server. **serverName** is the server's Internet host name or IP address.

... Client Sockets

- The following statement creates a socket on the client machine to connect to the host 130.254.204.33 at port 8000:

```
Socket socket = new Socket("130.254.204.33", 8000)
```

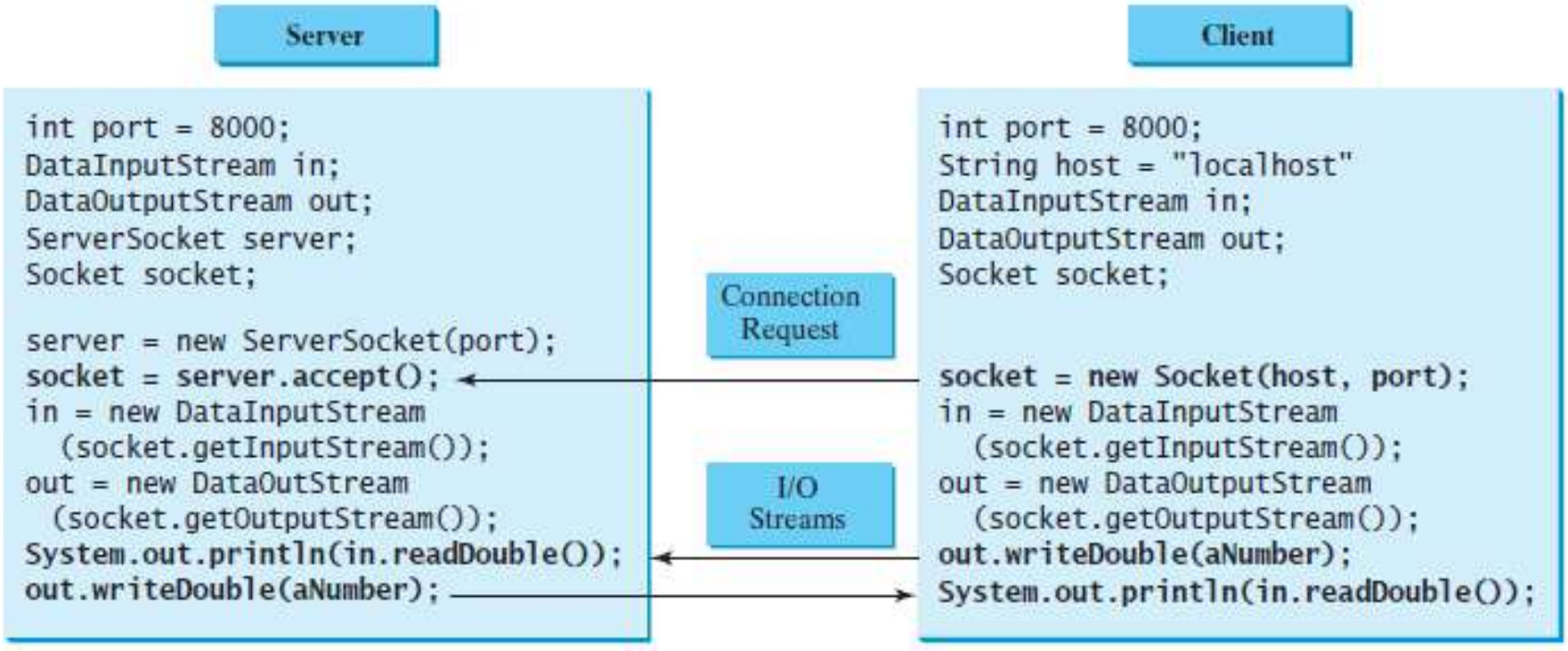
Alternatively, you can use the domain name to create a socket, as follows:

```
Socket socket = new Socket("www.facebook.com", 8000);
```

- When you create a socket with a host name, the JVM asks the DNS to translate the host name into the IP address.

4.4. Data Transmission through Sockets

- After the server accepts the connection, communication between the server and the client is conducted in the same way as for I/O streams. The statements needed to create the streams and to exchange data between them are shown in Example below.



The server and client exchange data through I/O streams on top of the socket.

….Data Transmission through Sockets

- To get an input stream and an output stream, use the **getInputStream()** and **getOutputStream()** methods on a socket object. For example, the following statements create an **InputStream** stream called **input** and an **OutputStream** stream called **output** from a socket:

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

- The **InputStream** and **OutputStream** streams are used to read or write bytes. You can use **DataInputStream**, **DataOutputStream**, **BufferedReader**, and **PrintWriter** to wrap on the **InputStream** and **OutputStream** to read or write data, such as **int, double,or String**.

….Data Transmission through Sockets

- The following statements, for instance, create the **DataInputStream** stream **input** and **the DataOutputStream output** to read and write primitive data values:

```
DataInputStream input = new DataInputStream(socket.getInputStream());

DataOutputStream output = new DataOutputStream(socket.getOutputStream());
```

4.5. The InetAddress Class

- The server program can use the **InetAddress** class to obtain the information about the IP address and host name for the client.
- 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 following statement in the server program to get an instance of **InetAddress** on a socket that connects to the client.

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

….The InetAddress Class

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());
```

4.6. Serving Multiple Clients

- A server can serve multiple clients. The connection to each client is handled by one thread.
- Multiple clients are quite often connected to a single server at the same time. Typically, a server runs continuously on a server computer, and clients from all over the Internet can 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(); // Connect to a client
 Thread thread = new ThreadClass(socket);
 thread.start();}
```

... Serving Multiple Clients

- 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. A sample run of the server with two clients is shown in Figure below:

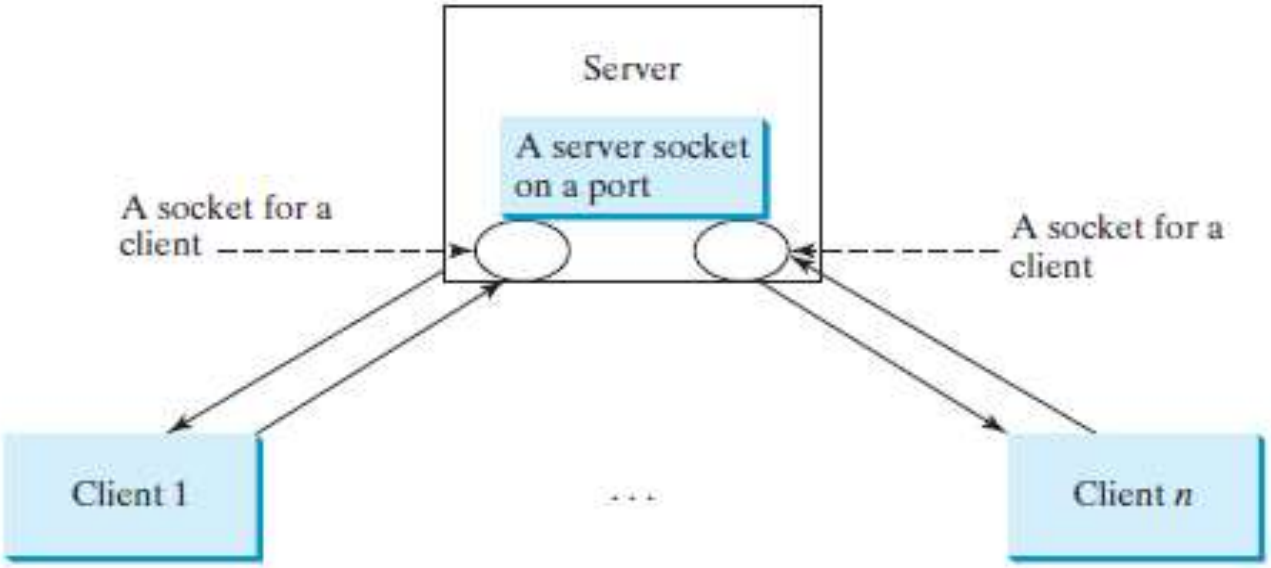


Figure 4.3: Multithreading enables a server to handle multiple independent clients.

4.7. Connectionless Client/Server Interaction with Datagrams

- Connectionless transmission with datagrams
  - No connection maintained with other computer
  - Break message into separate pieces and send as packets
- Datagram packet
  - used as short lived *envelopes* for datagram messages
  - Used to assemble messages before they are dispatched onto the network,
  - or dismantle messages after they have been received
- The DatagramPacket class
  - represents a datagram packet.
  - Has the following attributes:
    - Destination/source address
    - Destination/source port number
    - Data bytes constituting the message
    - Length of message data bytes

4.8. DatagramSocket and DatagramPacket

- Java **DatagramSocket** and **DatagramPacket** classes are used for connection-less socket programming.
- **DatagramSocket class**
- **Java DatagramSocket** class represents a connection-less socket for sending and receiving datagram packets. A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.
- **Commonly used Constructors of DatagramSocket class**
  - **DatagramSocket() throws SocketException:** it creates a datagram socket and binds it with the available Port Number on the localhost machine.
  - **DatagramSocket(int port) throws SocketException:** it creates a datagram socket and binds it with the given Port Number.
  - **DatagramSocket(int port, InetAddress address) throws SocketException:** it creates a datagram socket and binds it with the specified port number and host address.

..... DatagramSocket and DatagramPacket

- **Java DatagramPacket class**

Java DatagramPacket is a message that can be sent or received. If you send multiple packet, it may arrive in any order. Additionally, packet delivery is not guaranteed.

Commonly used Constructors of DatagramPacket class

- **DatagramPacket(byte[] buf, int length):** it creates a datagram packet. This constructor is used to receive the packets of lengh length.
- **DatagramPacket(byte[] buf, int length, InetAddress address, int port):** it creates a datagram packet. This constructor is used to send the packets to the specified port number on the specified host.

ExampleSender ExampleReceiver


THANKS