UNIVERSITY OF BIRMINGHAM

Java Socket Programming

FSAD/SWW2 Week 11

Ana Stroescu

Contents

- Sockets in Java
- TCP Sockets
  - TCP Sockets in Java
  - TCP Client-Server Socket Interaction
  - Simple client-server example: Greet Server
    - Server Program
    - Client Program
    - Greet Server – sample output
    - Continuous Communication
- UDP Sockets
  - UDP Sockets in Java
  - UDP Client-Server Socket Interaction
    - Server Program
    - Client Program
- Additional reading

UNIVERSITY OF BIRMINGHAM

Sockets in Java

- Java provides three types of sockets:

1 & 2) Server Sockets & Client Sockets – TCP

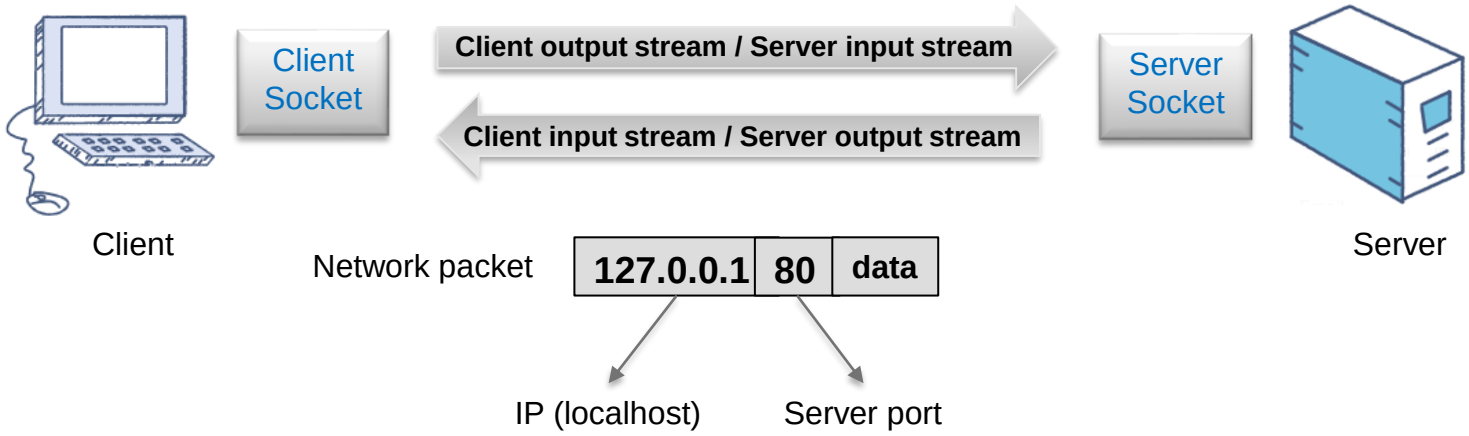
- Use the TCP protocol, so they provide a connection-oriented service;
- Stream-based sockets which establish a connection between processes;
- While the connection is active, the data is transferred in continuous streams.

3) Datagram Sockets – UDP

- Use the UDP protocol;
- The transmission of packets follows a connection-less service;
- With datagram sockets, individual packets of information are transmitted;

TCP Sockets

- A TCP/IP socket enables a Java program running on a client machine to open a connection with a web server;
- There is a socket on both the client and server sides;
- Client server communication is carried out through input and output streams.



Note: for the sake of simplicity, we are going to run both client and server on the same machine, with the localhost IP address 127.0.0.1 and the default HTTP port 80.

TCP Sockets in Java

- Java provides a collection of classes and interfaces that take care of low-level communication details between the client and the server.
- These are mostly contained in the `java.net` package.

```
import java.net.*;
```

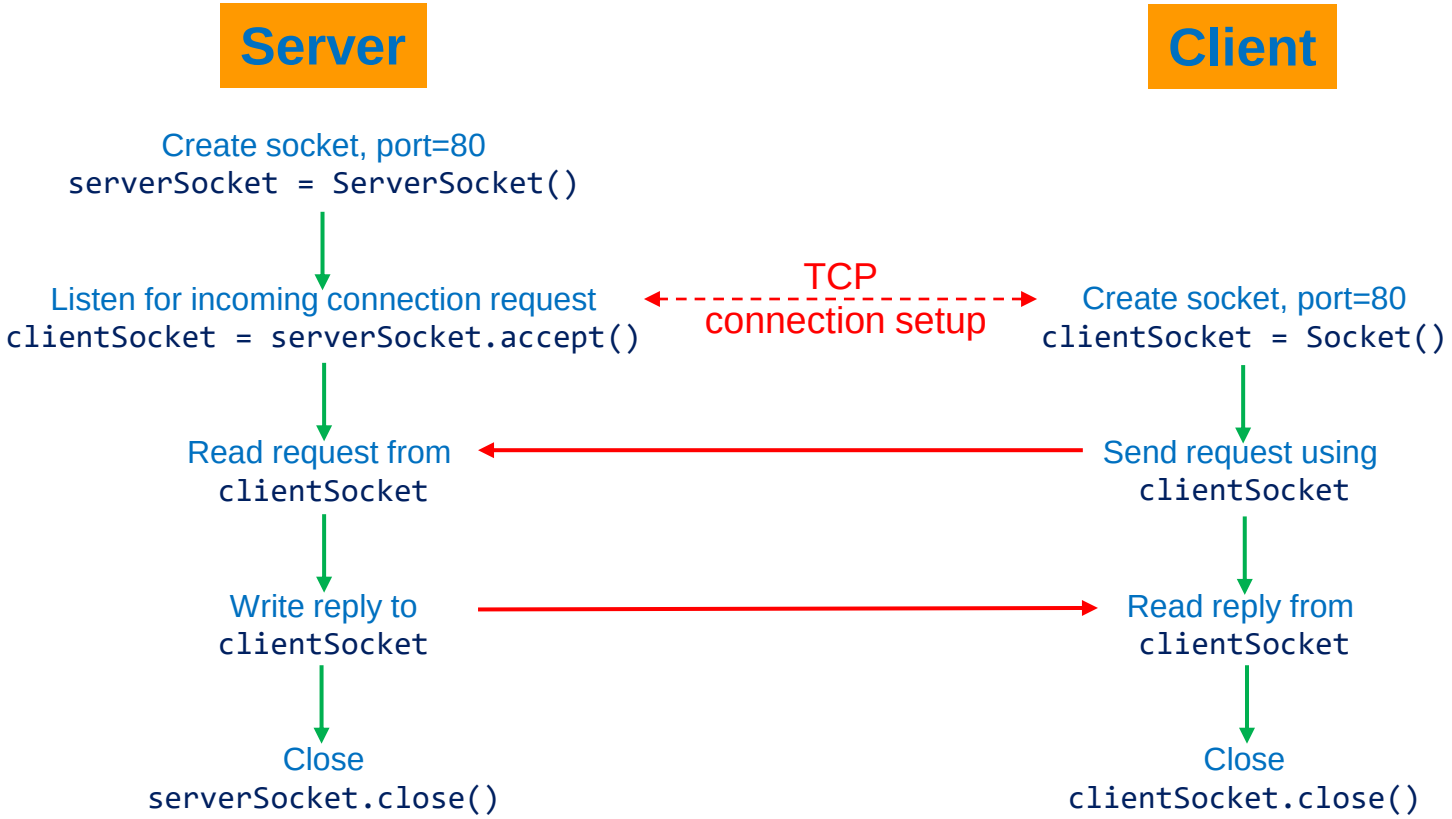
`Socket(), ServerSocket()`

- For the input and output streams to write to and read from while communicating we need the `java.io` package.

```
import java.io.*;
```

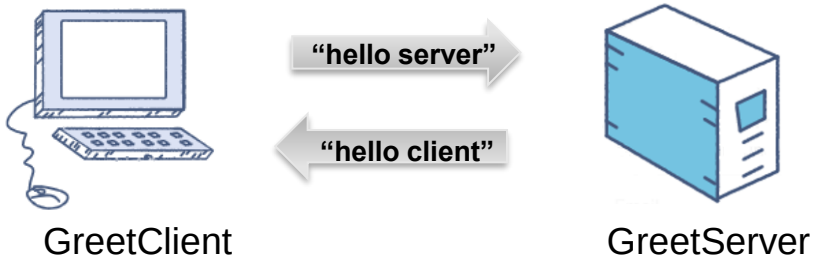
`PrintWriter(), BufferedReader()`

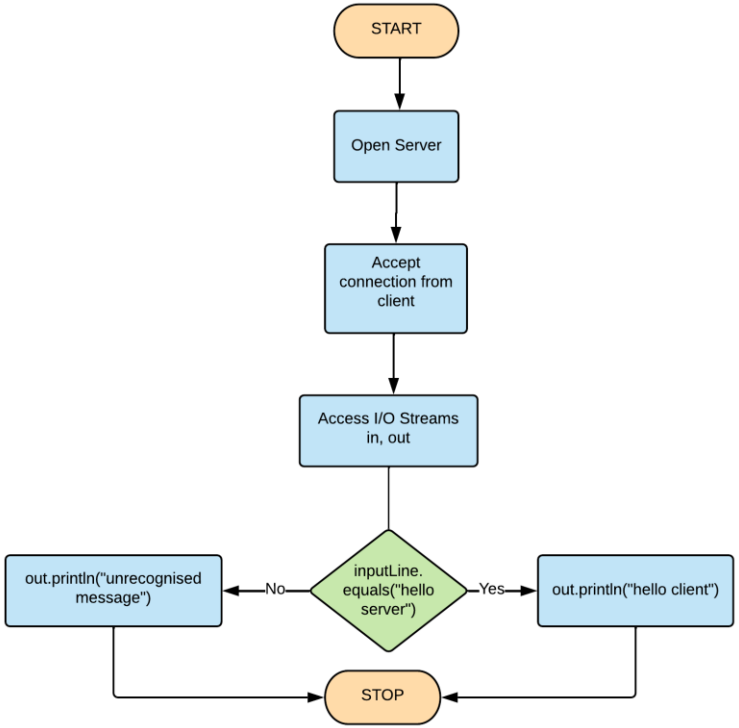
TCP Client-Server socket interaction



Simple client-sever example: Greet Server

- In this basic example, the client greets the server and the server responds.





- We are going to use two separate classes for the client and the server: `TCPGreetClient.java` and `TCPGreetServer.java`
- The source code is available on Canvas.

Server program

- Any basic TCP server program follows this structure:

- Step 1: Import `java.net.*` and `java.io.*`
- Step 2: Open the server socket specifying the port number

```
ServerSocket serverSocket = new ServerSocket(80);
```

- Step 3: Accept client connection

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

- Step 4: I/O streams to read/write from/to the client

```
BufferedReader inFromClient = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
PrintWriter  outToClient = new PrintWriter(clientSocket.getOutputStream(),true);
```

- Step 5: Read message from client

```
clientMessage = inFromClient.readLine();
```

- Step 6: Write message to client

```
outToClient.println("hello client");
```

- Step 7: Close streams and socket

```
inFromClient.close(); outToClient.close(); serverSocket.close();
```

TCPGreetServer.java

1: Import

```
import java.io.*;
import java.net.*;

class TCPGreetServer {

   public static void main(String args[]) throws IOException {
      String clientMessage;

      ServerSocket serverSocket = new ServerSocket(80);
      System.out.println("Server is running" );

      Socket clientSocket = serverSocket.accept();

      BufferedReader inFromClient = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
      PrintWriter  outToClient = new PrintWriter(clientSocket.getOutputStream(),true);

      clientMessage=inFromClient.readLine();

      if ("hello server".equals(clientMessage)) {
         outToClient.println("hello client");
      }
      else {
         outToClient.println("unrecognised message");
      }

      inFromClient.close();
      outToClient.close();
      serverSocket.close();
   }
}
```

2: Open the server socket

3: Accept client connection

4: I/O Streams

5: Read message from client

6: Send message to client

7: Close I/O streams and socket

Client program

- Any basic TCP client program follows this structure:

- Step 1: Import `java.net.*` and `java.io.*`
- Step 2: Open a connection to the server specifying IP (localhost) and port (80)

```
Socket clientSocket = new Socket("127.0.0.1", 80);
```

- Step 3: I/O Streams to read/write data

```
PrintWriter outToServer = new PrintWriter(clientSocket.getOutputStream(), true);
BufferedReader inFromServer = new BufferedReader (new InputStreamReader(clientSocket.getInputStream()));
BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in)); //(optional)
```

- Step 4 (optional): Read message from the user

```
message = inFromUser.readLine();
```

- Step 5: Send message to the server

```
outToServer.println(message);
```

- Step 6: Receive message from the server

```
serverMessage = inFromServer.readLine();
```

- Step 7: Close I/O streams and socket

```
inFromServer.close(); inFromUser.close(); outToServer.close(); clientSocket.close();
```

TCPGreetClient.java

1: Import

```
import java.io.*;
import java.net.*;

public class TCPGreetClient {

   public static void main(String args[]) throws IOException {
      String message, serverMessage;

      Socket clientSocket = new Socket("127.0.0.1", 80);
      System.out.println("Client is running");

      PrintWriter outToServer = new PrintWriter(clientSocket.getOutputStream(), true);
      BufferedReader inFromServer = new BufferedReader (new InputStreamReader(clientSocket.getInputStream()));
      BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));

      System.out.println("CLIENT MESSAGE: ");
      message = inFromUser.readLine();

      outToServer.println(message);

      serverMessage = inFromServer.readLine();
      System.out.println("SERVER MESSAGE: " + serverMessage);

      inFromServer.close();
      inFromUser.close();
      outToServer.close();
      clientSocket.close();
   }
}
```

2: Open the client socket

Connect to the server on localhost IP address and port 80

3: I/O Streams

4: Read message from user

5: Send message to the server

6: Read server response

7: Close I/O streams and socket

Greet Server - sample output

You need to run the server first, otherwise the client will not be able to connect.

```
Server is running
```

Console output for **`TCPGreetServer.java`**

```
Client is running
CLIENT MESSAGE:
hello server
SERVER MESSAGE: hello client
```

```
Client is running
CLIENT MESSAGE:
hellooo
SERVER MESSAGE: unrecognised message
```

Console output for **`TCPGreetClient.java`**

Continuous communication

- Notice that in the previous application, the server closes the connection after it receives one single message from the client.
- If we want to implement a client-server model with back and forth communication, we need to deal with continuity.
- This is implemented by just adding a while loop in the server program, to continuously monitor the input stream from the client:

Option 1:

```
while(true) {
    clientMessage = inFromClient.readLine();
.
.
}
```

Option 2:

```
while((clientMessage = inFromClient.readLine())!= null) {
.
.
}
```

Communication ends when the client stops the connection

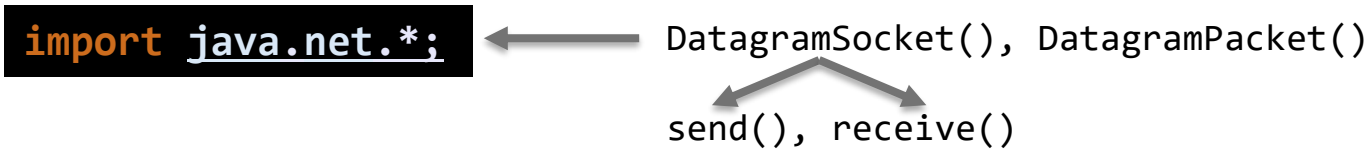
Option 3:

```
while(!(clientMessage = inFromClient.readLine()).equals("exit")) {
.
.
}
```

Communication ends when the client types an exit message

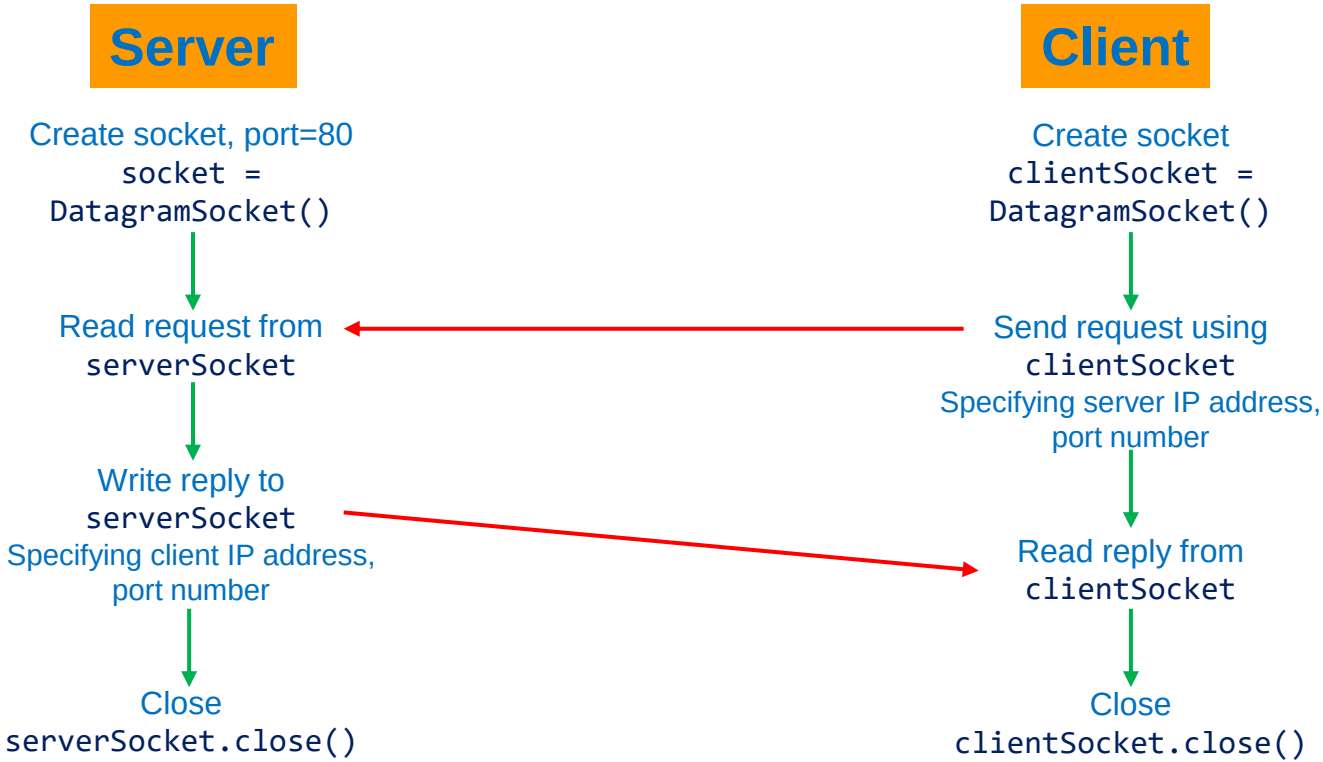
UDP Sockets in Java

- The UDP protocol is connectionless, which means there are no mechanisms for starting, maintaining, ending and controlling the flow of a communication.
- UDP transmission is lightweight and fast.
- Messages sent with UDP are called datagrams.
- In Java, network communication via UDP is achieved with Datagram Sockets, which can be used to send and receive packets over the network.
- A single message is encapsulated in a `DatagramPacket` which is sent/received through a `DatagramSocket`, using the `send()/receive()` method.



Note: there is no need for I/O streams attached to the sockets.

UDP Client-Server socket interaction



Greet Server Application using UDP

Server program

- **Step 1:** Import `java.net.*` and `java.io.*`
- **Step 2:** Create send/receive buffers

```
byte[] sendData = new byte[256];
byte[] receiveData = new byte[256];
```

- **Step 3:** Create a Datagram Socket, specifying the port number

```
DatagramSocket socket = new DatagramSocket(80);
```

- **Step 4:** Create a receiving Datagram Packet

```
DatagramPacket packet = new DatagramPacket(receiveData, receiveData.length);
```

- **Step 5:** Receive a packet from the client

```
serverSocket.receive(packet);
```

- **Step 6:** Get the address and the port of the client

```
InetAddress address = packet.getAddress();
int port = packet.getPort();
```

- **Step 7:** Convert the Datagram Packet into a sending one, include the client's IP and port

```
packet = new DatagramPacket(sendData, sendData.length, address, port);
```

- **Step 8:** Send the packet to the client

```
serverSocket.send(packet);
```

- **Step 9:** Close the socket connection

```
serverSocket.close();
```

UDPGreetServer.java

1: Import

```
import java.net.*;
import java.io.*;

public class UDPGreetServer {
   public static void main(String args[]) throws IOException{
      String clientMessage, sendMessage;
      byte[] sendData = new byte[256];
      byte[] receiveData = new byte[256];

      DatagramSocket serverSocket = new DatagramSocket(80);
      System.out.println("Server is running" );

      DatagramPacket packet = new DatagramPacket(receiveData, receiveData.length);

      serverSocket.receive(packet);
      clientMessage = new String (packet.getData(), 0, packet.getLength());

      InetAddress address = packet.getAddress();
      int port = packet.getPort();
      if ("hello server".equals(clientMessage)) {
         sendMessage = "hello client";
      }
      else {
         sendMessage = "unrecognised message";
      }

      sendData = sendMessage.getBytes();
      packet = new DatagramPacket(sendData, sendData.length, address, port);
      serverSocket.send(packet);

      serverSocket.close();
   }
}
```

2: Create send/receive buffers

3: Create a DatagramSocket on port 80

4: Create a receiving DatagramPacket

5: Receive a packet from the client

Create a String using a part of the buffer, from byte 0 to packet length

6: Get the client's address and port

7: Convert the DatagramPacket into a sending one

Convert String to bytes

8: Send the packet to the client

9: Close the socket

Client program

- **Step 1:** Import `java.net.*` and `java.io.*`
- **Step 2:** Create send/receive buffers

```
byte[] sendData = new byte[256];
byte[] receiveData = new byte[256];
```

- **Step 3:** Create a Datagram Socket

```
DatagramSocket clientSocket = new DatagramSocket();
```

- **Step 4:** Get the IP address of the server

```
InetAddress address = InetAddress.getByName("localhost");
```

- **Step 5:** Create a sending Datagram Packet, include the server's IP and the port

```
DatagramPacket packet = new DatagramPacket(sendData, sendData.length, address, 80);
```

- **Step 6:** Send the packet to the server

```
clientSocket.send(packet);
```

- **Step 7:** Convert the Datagram Packet into a receiving one

```
packet = new DatagramPacket(receiveData, receiveData.length);
```

- **Step 8:** Receive a packet from the server

```
clientSocket.receive(packet);
```

- **Step 9:** Close the socket connection

```
clientSocket.close();
```

UDPGreetClient.java

1: Import

```
import java.net.*;
import java.io.*;

public class UDPGreetClient {
   public static void main(String args[]) throws IOException{
      String message, serverMessage;
      BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
```

2: Create send/receive buffers

```
      byte[] sendData = new byte[256];
      byte[] receiveData = new byte[256];

      DatagramSocket clientSocket = new DatagramSocket();
      System.out.println("Client is running" );
```

3: Create a DatagramSocket

4: Get the IP of the server

```
      InetAddress address = InetAddress.getByName("localhost");

      System.out.println("CLIENT MESSAGE: ");
      message = inFromUser.readLine();
      sendData = message.getBytes();
```

Read message from the user and convert it to bytes

5: Create a sending DatagramPacket

```
      DatagramPacket packet = new DatagramPacket(sendData, sendData.length, address, 80);

      clientSocket.send(packet);
```

6: Send the packet to the server

```
      packet = new DatagramPacket(receiveData, receiveData.length);

      clientSocket.receive(packet);
      serverMessage = new String (packet.getData(),0, packet.getLength());
      System.out.println("SERVER MESSAGE: " + serverMessage);

      clientSocket.close();
   }
}
```

7: Convert the DatagramPacket into a receiving one

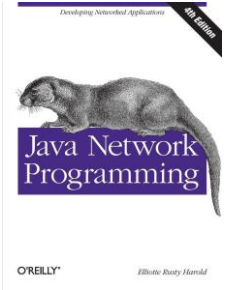
8: Receive a packet from the server

Create a String using a part of the buffer, from byte 0 to packet length

9: Close the socket

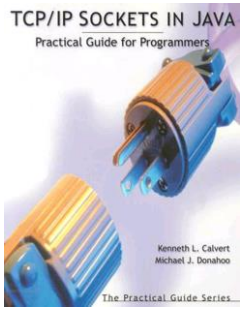
Additional reading

- **Java Network Programming,** by Elliotte Rusty Harold, O'Reilly Media, 4th edition



E-book

- **TCP/IP Sockets in Java: practical guide for programmers,** by Kenneth L. Calvert and Michael J. Donahoo



E-book

- Java.net documentation
- Java tutorial on sockets