CST8288 Java Network Programming – Basics

The materials in this handout are heavily based on Deitel and Deitel (2015). This includes the code samples.

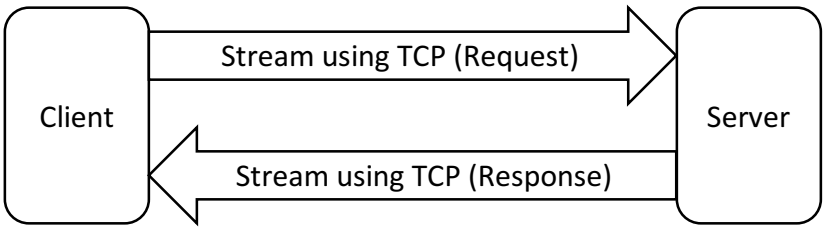
Networking - Basics

- Writing Java code to exchange information between two different Java programs, typically a server and a client across a computer network.
- Server – The software that listens for client requests and responds with data and / or services. Can also be used to refer to a physical machine.
- Client – The software that connects to a server across a network and requests data and / or services. Can also be used to refer to a physical machine.

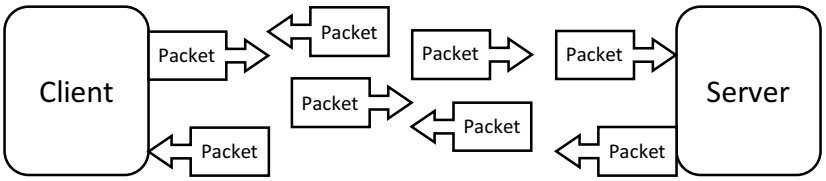
Note: Client / Server are relative terms, a single laptop computer can be both a physical client and server at the same time provided that there is a client program, and a server program running at the same time.

Java Networking Overview (Deitel and Deitel 2015)

- Classes and interfaces are inside java.net
- Java uses socket-based communications
    - Writing networking code is almost as simple as using File-IO
    - Textbook will look at stream sockets and datagram sockets
- Stream Socket
    - One process (program) connects via a connection to another process
    - While connection is active data flows in streams from one process to the other
    - This is known as a connection-oriented service
    - TCP (Transmission Control Protocol) is the protocol used here
    - Data sent is guaranteed to be received in the correct order



- Datagram Socket
    - Packets of information are transmitted rather than streams
    - UDP (User Datagram Protocol) is the protocol used here
    - This is known as a connectionless service
    - Data packets are not guaranteed to be received in any particular order, and packets can be lost or duplicated. (Extra programming work if you need to handle these issues).



Packets are sent using UDP protocol

- Most Java networking applications will likely use Stream Sockets

Demo – Simple Client and Server using Stream Sockets (Deitel and Deitel 2015)

Server Steps

1) Create a **ServerSocket**
   ServerSocket server = new **ServerSocket**(portNumber, queueLength);
2) Wait for a Connection
   Socket connection = server.accept();
3) Get Socket's I/O Streams
   Note: Textbook wraps these into ObjectInputStream and ObjectOutputStream to use Serialization
   ObjectInputStream input = new ObjectInputStream(connection.getInputStream());
   ObjectOutputStream output = new ObjectOutputStream(connection.getOutputStream());
4) Perform any needed processing
5) Close the Connection
   input.close(), output.close() and server.close() !!!! IMPORTANT !!!!

Client Steps

1) Create a **Socket** to Connect to the Server (Client does not use a ServerSocket object)
   Socket connection = new **Socket**(serverAddress, portNumber);
   (Can throw exceptions: IOException or UnknownHostException)
2) Get the Socket's I/O Streams
   Note: Textbook wraps these into ObjectInputStream and ObjectOutputStream to use Serialization
   ObjectInputStream input = new ObjectInputStream(connection.getInputStream());
   ObjectOutputStream output = new ObjectOutputStream(connection.getOutputStream());
3) Perform any needed processing
4) Close the Connection
   input.close(), output.close(), connection.close(); !!!! ALSO IMPORTANT !!!!

(Code will be demonstrated in Eclipse and made available as a zip archive)

Demo – Very Simple Client and Server using Datagrams (Deitel and Deitel 2015)

Server Overview / Code Highlights

1) Use a **DatagramSocket**
   DatagramSocket socket = new **DatagramSocket**(*portNumber*); // book uses port number of 5000
2) Use an infinite loop to continually check for incoming packets from client using a **DatagramPacket**
   while(true){
     try {
       byte[] data = new byte[100]; // packet
       DatagramPacket receivePacket = new **DatagramPacket**(data, data.length);
       Socket.receive(receivePacket); // wait for a packet to come in, then dump it into receivePacket object
3) Echo the packet back to the client
   DatagramPacket sendPacket = new **DatagramPacket**(
    receivePacket.getData(), receivePacket.getLength(), receivePacket.getAddress(), receivePacket.getPort());
   socket.send(sendPacket);
4) Make sure the socket is closed when done with it… Added by Stan…

Client Overview / Code Highlights

1) Use a **DatagramSocket**
   DatagramSocket socket = new **DatagramSocket**(); // note no portNumber here in client
2) Use a **DatagramPacket** based on converting String into a byte array
   byte[] data = message.getBytes(); // message is a String object, getBytes() converts the String to byte array
   DatagramPacket sendPacket = new **DatagramPacket**(
     data, data.length, InetAddress.getLocalHost(), *portNumberOfServer*); // book uses port 5000
3) Use an infinite loop to continually check for incoming packets from server using a DatagramPacket
   while(true){
     try{
       byte[] data = new byte[100]; // packet
       DatagramPacket receivePacket = new **DatagramPacket**(data, data.length);
       Socket.receive(receivePacket); // wait for a packet to come in, then dump it into receivePacket object
4) Convert the data received back into a String for display
   String packetData = new String(receivePacket.getData(), 0, receivePacket.getLength());
5) Make sure the socket is closed when done with it… Added by Stan…

(Code will be demonstrated in Eclipse and made available as a zip archive)

Required Reading

Deitel, P. and Deitel, H. (2015). Java ™ How to Program: Early Objects 10th Ed. Pearson Education, Inc. USA.
Chapter 28 Networking (Available as a PDF through the publisher's on-line content)