**Socket Programming in Java**

This experiment describes a very basic one-way Client and Server setup where a Client connects, sends messages to server and the server shows them using socket connection. There’s a lot of low-level stuff that needs to happen for these things to work but the Java API networking package (java.net) takes care of all of that, making network programming very easy for programmers.

**Client Side Programming**

**Establish a Socket Connection**

To connect to other machine we need a socket connection. A socket connection means the two machines have information about each other’s network location (IP Address) and TCP port.The java.net.Socket class represents a Socket. To open a socket:

Socket socket = new Socket(“127.0.0.1”, 5000)

* First argument – **IP address of Server**. ( 127.0.0.1  is the IP address of localhost, where code will run on single stand-alone machine).
* Second argument – **TCP Port**. (Just a number representing which application to run on a server. For example, HTTP runs on port 80. Port number can be from 0 to 65535)

**Example Program – Simple Communication between Client and Server**

**Java Implementation - Client**

|  |
| --- |
| *// A Java program for a Client*  *import java.net.\*;*  *import java.io.\*;*    *public class Client*  *{*  *// initialize socket and input output streams*  *private Socket socket            = null;*  *private DataInputStream  input   = null;*  *private DataOutputStream out     = null;*    *// constructor to put ip address and port*  *public Client(String address, int port)*  *{*  *// establish a connection*  *try*  *{*  *socket = new Socket(address, port);*  *System.out.println("Connected");*    *// takes input from terminal*  *input  = new DataInputStream(System.in);*    *// sends output to the socket*  *out    = new DataOutputStream(socket.getOutputStream());*  *}*  *catch(UnknownHostException u)*  *{*  *System.out.println(u);*  *}*  *catch(IOException i)*  *{*  *System.out.println(i);*  *}*    *// string to read message from input*  *String line = "";*    *// keep reading until "Over" is input*  *while (!line.equals("Over"))*  *{*  *try*  *{*  *line = input.readLine();*  *out.writeUTF(line);*  *}*  *catch(IOException i)*  *{*  *System.out.println(i);*  *}*  *}*    *// close the connection*  *try*  *{*  *input.close();*  *out.close();*  *socket.close();*  *}*  *catch(IOException i)*  *{*  *System.out.println(i);*  *}*  *}*    *public static void main(String args[])*  *{*  *Client client = new Client("127.0.0.1", 5000);*  *}*  *}* |
| **Java Implementation - Server**  *// A Java program for a Server*  *import java.net.\*;*  *import java.io.\*;*    *public class Server*  *{*  *//initialize socket and input stream*  *private Socket          socket   = null;*  *private ServerSocket    server   = null;*  *private DataInputStream in       =  null;*    *// constructor with port*  *public Server(int port)*  *{*  *// starts server and waits for a connection*  *try*  *{*  *server = new ServerSocket(port);*  *System.out.println("Server started");*    *System.out.println("Waiting for a client ...");*    *socket = server.accept();*  *System.out.println("Client accepted");*    *// takes input from the client socket*  *in = new DataInputStream(*  *new BufferedInputStream(socket.getInputStream()));*    *String line = "";*    *// reads message from client until "Over" is sent*  *while (!line.equals("Over"))*  *{*  *try*  *{*  *line = in.readUTF();*  *System.out.println(line);*    *}*  *catch(IOException i)*  *{*  *System.out.println(i);*  *}*  *}*  *System.out.println("Closing connection");*    *// close connection*  *socket.close();*  *in.close();*  *}*  *catch(IOException i)*  *{*  *System.out.println(i);*  *}*  *}*    *public static void main(String args[])*  *{*  *Server server = new Server(5000);*  *}*  *}* |

**Important Points**

Server application makes a ServerSocket on a specific port which is 5000. This starts our Server listening for client requests coming in for port 5000.

Then Server makes a new Socket to communicate with the client.

socket = server.accept()

The accept() method blocks(just sits there) until a client connects to the server.

Then we take input from the socket using getInputStream() method. Our Server keeps receiving messages until the Client sends “Over”.

After we’re done we close the connection by closing the socket and the input stream.

To run the Client and Server application on your machine, compile both of them. Then first run the server application and then run the Client application.

**To run on Terminal or Command Prompt**

Open two windows one for Server and another for Client

1. First run the Server application as ,

$ java Server

Server started  
Waiting for a client …

2. Then run the Client application on another terminal as,

$ java Client

It will show – Connected and the server accepts the client and shows,

Client accepted

3. Then you can start typing messages in the Client window. Here is a sample input to the Client

Hello

I made my first socket connection

Over

Which the Server simultaneously receives and shows,

Hello

I made my first socket connection

Over

Closing connection

Notice that sending “Over” closes the connection between the Client and the Server just like said before.

**UDP implementation – Echo client and Server**

import java.net.\*;

import java.util.\*;

public class ClientEcho

{

  public static void main( String args[] ) throws Exception

  {

    InetAddress add = InetAddress.getByName("snrao");

    DatagramSocket dsock = new DatagramSocket( );

    String message1 = "This is client calling";

    byte arr[] = message1.getBytes( );

    DatagramPacket dpack = new DatagramPacket(arr, arr.length, add, 7);

    dsock.send(dpack);                                   // send the packet

    Date sendTime = new Date();                          // note the time of sending the message

    dsock.receive(dpack);                                // receive the packet

    String message2 = new String(dpack.getData( ));

    Date receiveTime = new Date( );   // note the time of receiving the message

    System.out.println((receiveTime.getTime( ) - sendTime.getTime( )) + " milliseconds echo time for " + message2);

  }

}

import java.net.\*;

import java.util.\*;

public class ServerEcho

{

public static void main( String args[]) throws Exception

{

DatagramSocket dsock = new DatagramSocket(7);

byte arr1[] = new byte[150];

DatagramPacket dpack = new DatagramPacket(arr1, arr1.length );

while(true)

{

dsock.receive(dpack);

byte arr2[] = dpack.getData();

int packSize = dpack.getLength();

String s2 = new String(arr2, 0, packSize);

System.out.println( new Date( ) + " " + dpack.getAddress( ) + " : " + dpack.getPort( ) + " "+ s2);

dsock.send(dpack);

}

}

}

**List of possible applications**

* TCP/UDP Day Time Server
* TCP/UDP Echo Server
* TCP/UDP FTP
* TCP/UDP Chat Application

Explore ways of execution other than local host.

Implement the same application in “C”