Simple Client/Server:

**EchoServer:**

Simple server that just *echoes back* whatever the client sends.

- **Create a server socket:**
  ```java
  ServerSocket s = new ServerSocket(PORT);
  ```

- **Accept a client connection:**
  ```java
  Socket socket = s.accept();
  ```

- **Get socket input stream:**
  ```java
  InputStream in = socket.getInputStream();
  ```

- **Get socket output stream:**
  ```java
  OutputStream out = socket.getOutputStream();
  ```

- **Read:**
  ```java
  String str = in.readLine();
  ```

- **Write:**
  ```java
  out.println(str);
  ```

**Program Listing:**

```java
public class EchoServer {
    public static final int PORT = 10101;
    public static void main (String[] args) throws IOException {
        ServerSocket s = new ServerSocket(PORT);
        System.out.println("Started: " + s);
        try {
            Socket socket = s.accept();
```
try {
    System.out.println("Connection accepted: "+ socket);
    BufferedReader in = new BufferedReader(
        new InputStreamReader(socket.getInputStream()));
    PrintWriter out = new PrintWriter(new BufferedWriter(
        new OutputStreamWriter(socket.getOutputStream())),true);
    while (true) {
        String str = in.readLine();
        if (str.equals("END")) break;
        System.out.println("Echoing: "+ str);
        out.println(str);
    }
} finally {
    socket.close();
}} finally {
    s.close();
} }

**EchoClient**:

Simple client to sends lines to the server and reads lines that the server sends back.

Create a client socket connected to the server:

```java
Socket socket = new Socket(addr, PORT);
```

Get socket input stream:

```java
in = socket.getInputStream();
```

Get socket output stream:

```java
out = socket.getOutputStream();
```

Write: `out.println(str);`

Read: `String str = in.readLine();`

**Program Listing:**

```java
public class EchoClient {
    public static void main(String[] args) throws IOException {
        InetAddress addr = InetAddress.getByName(args[0]);
        System.out.println("addr = " + addr);
```
Socket socket = new Socket(addr, EchoServer.PORT);
try {
    System.out.println("socket = "+socket);
    BufferedReader in = new BufferedReader(
        new InputStreamReader(socket.getInputStream()));
    PrintWriter out = new PrintWriter(new BufferedWriter(
        new OutputStreamWriter(socket.getOutputStream())),true);
    for(int i = 0; i < 10; i++) {
        out.println("howdy "+i);
        String str = in.readLine();
        System.out.println(str);
    }
    out.println("END");
} finally {
    socket.close();
}  }}

How to run:
% java    EchoServer
% java    EchoClient    <hostname>

Multi-Threaded Client/Server:

MultiEchoServer:

A server that uses *multi_threading* to handle any number of clients.

For each client connection accepted in socket, create a new *thread* to handle it:

    new   ServeOneEcho (socket);

The thread will then get *in/out* streams from socket to read/write

Program Listing:

    class  EchoServOne  extends  Thread {
        private Socket soc;
        private BufferedReader in;
        private PrintWriter out;

        public  EchoServOne (Socket s) throws IOException {
            sock = s;
in = new BufferedReader(
  new InputStreamReader(soc.getInputStream()));
out = new PrintWriter (new BufferedWriter(
  new OutputStreamWriter(soc.getOutputStream())),true);
start ();  // Calls run()
}

public void run() {
  try {
    while (true) {
      String str = in.readLine();
      if (str.equals("END")) break;
      System.out.println("Echoing: " + str);
      out.println(str);
    }
    System.out.println("closing...");
  }
  catch (IOException e) {
    System.err.println("IO Exception");
  }
  finally {
    try {
      socket.close();
    }
    catch (IOException e) {
      System.err.println("Socket not closed");
    }
  }
}

public class MultiEchoServer {
  public static void main (String[] args) throws IOException {
    ServerSocket s = new ServerSocket (0);
    String ServerPort = s.getLocalPort() + "\n";
    System.out.println("Server Started at Port: " + ServerPort);
    try {
      while (true) {
        Socket socket = s.accept();
        try {
          new EchoServOne (socket);
        }
        catch (IOException e) {
          socket.close();
        }
      }
    }
    finally {
      s.close();
    }
  }
}
MultiEchoClient:

Tests the MultiEchoServer by starting up multiple clients.

Create many clients up to MAX_THREADS and sleep 5000 ms (5 seconds) after the creation of each client.

while (true) {
    if (EchoClientThread.threadCount() < MAX_THREADS)
        new EchoClientThread(addr);
    Thread.currentThread().sleep (5000);
}

Each client sends 5 messages to the server and exits.

for (int i = 1; i <= 5; i++) {
    out.println("Client " + id + ": " + i);
    String str = in.readLine();
    System.out.println (str);
}
out.println("END");

Program Listing:

```java
class EchoClientThread extends Thread {
    private Socket socket;
    private BufferedReader in;
    private PrintWriter out;
    private static int counter = 0;
    private int id = ++counter;
    private static int threadcount = 0;
    public static int threadCount() {
        return threadcount;
    }
    public EchoClientThread (InetAddress addr, int PORT) {
        System.out.println ("Making client " + id);
        threadcount++;
        try {
            socket = new Socket (addr, PORT);
```
try {
    in = new BufferedReader(
            new InputStreamReader(socket.getInputStream()));
    out = new PrintWriter( new BufferedWriter(
            new OutputStreamWriter(socket.getOutputStream())),true);
    start();
} catch (IOException e) {
    try {
        socket.close();
    } catch (IOException e2) {
        System.err.println("Socket not closed");
    }
}

public void run() {
    try {
        for (int i = 1; i <=5; i++) {
            out.println("Client " + id + ": " + i);
            String str = in.readLine();
            System.out.println(str);
            try {
                sleep(5000);
            } catch(InterruptedException e) {}
        }
        out.println("END");
    } catch (IOException e) { 
        System.err.println("IO Exception");
    }
} finally {
    try {
        socket.close();
    } catch (IOException e) {
    }
    threadcount--; // Ending this thread
}

public class MultiEchoClient {
    static final int MAX_THREADS = 5;
    public static void main (String[] args)
            throws IOException, InterruptedException {
        InetAddress addr = InetAddress.getByName(args[0]);
        int Sport = Integer.parseInt(args[1]);
        while(true) { (IOException e) {
            System.err.println("Socket failed");
        }
        catch (IOException e) {
            System.err.println("Socket failed");
        }
        try {
            in = new BufferedReader(
                    new InputStreamReader(socket.getInputStream()));
            out = new PrintWriter( new BufferedWriter(
                    new OutputStreamWriter(socket.getOutputStream())),true);
            start();
        } catch (IOException e) { 
            try {
                socket.close();
            } catch (IOException e2) {
                System.err.println("Socket not closed");
            }
        }
        public void run() {
            try {
                for (int i = 1; i <=5; i++) {
                    out.println("Client " + id + ": " + i);
                    String str = in.readLine();
                    System.out.println(str);
                    try {
                        sleep(5000);
                    } catch(InterruptedException e) {}
                }
                out.println("END");
            } catch (IOException e) { 
                System.err.println("IO Exception");
            }
        } finally {
            try {
                socket.close();
            } catch (IOException e) { 
            }
            threadcount--; // Ending this thread
        }
    }
}
if (EchoClientThread.threadCount() < MAX_THREADS) {
    new EchoClientThread(addr, Sport);
    Thread.currentThread().sleep(5000);
}

How to run:
% java MultiEchoServer
   Server Started at Port: xxxx
% java MultiEchoClient localhost xxxx
   (Type CTRL-C to interrupt).

Some Useful Details

Detecting socket close:
String str = in.readLine();
if (str==null)  // the client closed the connection

Printing local and remote end points of connected socket:
socket = s.accept();
System.out.println("Connection accepted, complete information:
"+ socket);
System.out.println("Local Port: " + socket.getLocalPort());
System.out.println("Local IP Address: " +
socket.getLocalAddress());
System.out.println("Remote Port: " + socket.getPort());
System.out.println("Remote IP Address: " +
socket.getInetAddress());
System.out.println("Remote Address: " +
socket.getRemoteSocketAddress());
System.out.println("Local Address: " +
socket.getLocalSocketAddress());