Creating a simple Chat Client/Server Solution

Here is an example of how to extend a very simple client-server demo program into a fully functioning (but simple) Chat Client/Server package. There are five stages involved:

- Step 1: A simple server that will accept a single client connection and display everything the client says on the screen. If the client user types ".bye", the client and the server will both quit.
- Step 2: A server as before, but this time it will remain 'open' for additional connection once a client has quit. The server can handle at most one connection at a time.
- Step 3: A server as before, but this time it can handle multiple clients simultaneously. The output from all connected clients will appear on the server's screen.
- Step 4: A server as before, but this time it sends all text received from any of the connected clients to all clients. This means that the server has to receive and send, and the client has to send as well as receive
- Step 5: Wrapping the client from step 4 into a very simple GUI interface but not changing the functionality of either server or client. The client is implemented as an Applet, but a Frame would have worked just as well (for a stand-alone program).

Step 1: Simple, one-time Server

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

public class ChatServer
{  private Socket          socket   = null;
   private ServerSocket    server   = null;
   private DataInputStream streamIn =  null;

   public ChatServer(int port)
   {  try
      {  System.out.println("Binding to port " + port + ", please wait  ...");
         server = new ServerSocket(port);
         System.out.println("Server started: " + server);
         System.out.println("Waiting for a client ...");
         socket = server.accept();
         System.out.println("Client accepted: " + socket);
         open();
         boolean done = false;
         while (!done)
         {  try
            {  String line = streamIn.readUTF();
               System.out.println(line);
               done = line.equals(".bye");
            }
            catch(IOException ioe)
            {  done = true;
            }
         }
         close();
      }
      catch(IOException ioe)
      {  System.out.println(ioe);
      }
   }
   public void open() throws IOException
   {  streamIn = new DataInputStream(new BufferedInputStream(socket.getInputStream()));
```

```
   }
   public void close() throws IOException
   {  if (socket != null)    socket.close();
      if (streamIn != null)  streamIn.close();
   }
   public static void main(String args[])
   {  ChatServer server = null;
      if (args.length != 1)
         System.out.println("Usage: java ChatServer port");
      else
         server = new ChatServer(Integer.parseInt(args[0]));
   }
}
```

The Simple Client corresponding to the previous server (and to step 2 and step 3 servers as well):

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

public class ChatClient
{  private Socket socket              = null;
   private DataInputStream  console   = null;
   private DataOutputStream streamOut = null;

   public ChatClient(String serverName, int serverPort)
   {  System.out.println("Establishing connection. Please wait ...");
      try
      {  socket = new Socket(serverName, serverPort);
         System.out.println("Connected: " + socket);
         start();
      }
      catch(UnknownHostException uhe)
      {  System.out.println("Host unknown: " + uhe.getMessage());
      }
      catch(IOException ioe)
      {  System.out.println("Unexpected exception: " + ioe.getMessage());
      }
      String line = "";
      while (!line.equals(".bye"))
      {  try
         {  line = console.readLine();
            streamOut.writeUTF(line);
            streamOut.flush();
         }
         catch(IOException ioe)
         {  System.out.println("Sending error: " + ioe.getMessage());
         }
      }
   }
   public void start() throws IOException
   {  console   = new DataInputStream(System.in);
      streamOut = new DataOutputStream(socket.getOutputStream());
   }
   public void stop()
   {  try
      {  if (console   != null)  console.close();
         if (streamOut != null)  streamOut.close();
         if (socket    != null)  socket.close();
      }
      catch(IOException ioe)
      {  System.out.println("Error closing ...");
      }
   }
```

```
   public static void main(String args[])
   {  ChatClient client = null;
      if (args.length != 2)
         System.out.println("Usage: java ChatClient host port");
      else
         client = new ChatClient(args[0], Integer.parseInt(args[1]));
   }
}
```

Step 2: Many Time Server, One Client (Client is as before)

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

public class ChatServer implements Runnable
{  private Socket       socket = null;
   private ServerSocket server = null;
   private Thread       thread = null;
   private DataInputStream  streamIn  =  null;

   public ChatServer(int port)
   {  try
      {  System.out.println("Binding to port " + port + ", please wait  ...");
         server = new ServerSocket(port);
         System.out.println("Server started: " + server);
         start();
      }
      catch(IOException ioe)
      {  System.out.println(ioe);
      }
   }
   public void run()
   {  while (thread != null)
      {   try
         {  System.out.println("Waiting for a client ...");
            socket = server.accept();
            System.out.println("Client accepted: " + socket);
            open();
            boolean done = false;
            while (!done)
            {  try
               {  String line = streamIn.readUTF();
                  System.out.println(line);
                  done = line.equals(".bye");
               }
               catch(IOException ioe)
               {  done = true;  }
            }
            close();
         }
         catch(IOException ie)
         {  System.out.println("Acceptance Error: " + ie);  }
      }
   }
   public void start()
   {  if (thread == null)
      {  thread = new Thread(this);
         thread.start();
      }
   }
   public void stop()
   {  if (thread != null)
```

```
      {  thread.stop();
         thread = null;
      }
   }
   public void open() throws IOException
   {  streamIn = new DataInputStream(new
                       BufferedInputStream(socket.getInputStream()));
   }
   public void close() throws IOException
   {  if (socket != null)    socket.close();
      if (streamIn != null)  streamIn.close();
   }
   public static void main(String args[])
   {  ChatServer server = null;
      if (args.length != 1)
         System.out.println("Usage: java ChatServer port");
      else
         server = new ChatServer(Integer.parseInt(args[0]));
   }
}
```

Step 3: Multi-Server handling Multi-Client (Client as before)

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

public class ChatServer implements Runnable
{  private ServerSocket     server = null;
   private Thread           thread = null;
   private ChatServerThread client = null;

   public ChatServer(int port)
   {  try
      {  System.out.println("Binding to port " + port + ", please wait  ...");
         server = new ServerSocket(port);
         System.out.println("Server started: " + server);
         start();
      }
      catch(IOException ioe)
      {  System.out.println(ioe); }
   }
   public void run()
   {  while (thread != null)
      {  try
         {  System.out.println("Waiting for a client ...");
            addThread(server.accept());
         }
         catch(IOException ie)
         {  System.out.println("Acceptance Error: " + ie); }
      }
   }
   public void addThread(Socket socket)
   {  System.out.println("Client accepted: " + socket);
      client = new ChatServerThread(this, socket);
      try
      {  client.open();
         client.start();
      }
      catch(IOException ioe)
      {  System.out.println("Error opening thread: " + ioe); }
   }
   public void start()                  { /* no change */ }
   public void stop()                   { /* no change */ }
```

```
   public static void main(String args[]){ /* no change */ }
}

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

public class ChatServerThread extends Thread
{  private Socket          socket   = null;
   private ChatServer      server   = null;
   private int             ID       = -1;
   private DataInputStream streamIn =  null;

   public ChatServerThread(ChatServer _server, Socket _socket)
   {  server = _server;  socket = _socket;  ID = socket.getPort();
   }
   public void run()
   {  System.out.println("Server Thread " + ID + " running.");
      while (true)
      {  try
         {  System.out.println(streamIn.readUTF());
         }
         catch(IOException ioe) {  }
      }
   }
   public void open() throws IOException
   {  streamIn = new DataInputStream(new BufferedInputStream(socket.getInputStream()));
   }
   public void close() throws IOException
   {  if (socket != null)    socket.close();
      if (streamIn != null)  streamIn.close();
   }
}
```

Step 4: A Simple but functional, text based chat server/client

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

public class ChatServer implements Runnable
{  private ChatServerThread clients[] = new ChatServerThread[50];
   private ServerSocket server = null;
   private Thread       thread = null;
   private int clientCount = 0;

   public ChatServer(int port)
   {  try
      {  System.out.println("Binding to port " + port + ", please wait  ...");
         server = new ServerSocket(port);
         System.out.println("Server started: " + server);
         start(); }
      catch(IOException ioe)
      {  System.out.println("Can not bind to port " + port + ": " + ioe.getMessage()); }
   }
   public void run()
   {  while (thread != null)
      {  try
         {  System.out.println("Waiting for a client ...");
            addThread(server.accept()); }
         catch(IOException ioe)
         {  System.out.println("Server accept error: " + ioe); stop(); }
      }
   }
```

```
   public void start()  { /* as before */ }
   public void stop()   { /* as before */ }
   private int findClient(int ID)
   {  for (int i = 0; i < clientCount; i++)
         if (clients[i].getID() == ID)
            return i;
      return -1;
   }
   public synchronized void handle(int ID, String input)
   {  if (input.equals(".bye"))
      {  clients[findClient(ID)].send(".bye");
         remove(ID); }
      else
         for (int i = 0; i < clientCount; i++)
            clients[i].send(ID + ": " + input);
   }
   public synchronized void remove(int ID)
   {  int pos = findClient(ID);
      if (pos >= 0)
      {  ChatServerThread toTerminate = clients[pos];
         System.out.println("Removing client thread " + ID + " at " + pos);
         if (pos < clientCount-1)
            for (int i = pos+1; i < clientCount; i++)
               clients[i-1] = clients[i];
         clientCount--;
         try
         {  toTerminate.close(); }
         catch(IOException ioe)
         {  System.out.println("Error closing thread: " + ioe); }
         toTerminate.stop(); }
   }
   private void addThread(Socket socket)
   {  if (clientCount < clients.length)
      {  System.out.println("Client accepted: " + socket);
         clients[clientCount] = new ChatServerThread(this, socket);
         try
         {  clients[clientCount].open();
            clients[clientCount].start();
            clientCount++; }
         catch(IOException ioe)
         {  System.out.println("Error opening thread: " + ioe); } }
      else
         System.out.println("Client refused: maximum " + clients.length + " reached.");
   }
   public static void main(String args[]) { /* as before */ }
}

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

public class ChatServerThread extends Thread
{  private ChatServer       server    = null;
   private Socket           socket    = null;
   private int              ID        = -1;
   private DataInputStream  streamIn  =  null;
   private DataOutputStream streamOut = null;

   public ChatServerThread(ChatServer _server, Socket _socket)
   {  super();
      server = _server;
      socket = _socket;
      ID     = socket.getPort();
   }
   public void send(String msg)
   {   try
```

```
            {  streamOut.writeUTF(msg);
               streamOut.flush();
            }
            catch(IOException ioe)
            {  System.out.println(ID + " ERROR sending: " + ioe.getMessage());
               server.remove(ID);
               stop();
            }
         }
         public int getID()
         {  return ID;
         }
         public void run()
         {  System.out.println("Server Thread " + ID + " running.");
            while (true)
            {  try
               {  server.handle(ID, streamIn.readUTF());
               }
               catch(IOException ioe)
               {  System.out.println(ID + " ERROR reading: " + ioe.getMessage());
                  server.remove(ID);
                  stop();
               }
            }
         }
         public void open() throws IOException
         {  streamIn = new DataInputStream(new
                           BufferedInputStream(socket.getInputStream()));
            streamOut = new DataOutputStream(new
                           BufferedOutputStream(socket.getOutputStream()));
         }
         public void close() throws IOException
         {  if (socket != null)    socket.close();
            if (streamIn != null)  streamIn.close();
            if (streamOut != null) streamOut.close();
         }
      }

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

      public class ChatClient implements Runnable
      {  private Socket socket              = null;
         private Thread thread              = null;
         private DataInputStream  console   = null;
         private DataOutputStream streamOut = null;
         private ChatClientThread client    = null;

         public ChatClient(String serverName, int serverPort)
         {  System.out.println("Establishing connection. Please wait ...");
            try
            {  socket = new Socket(serverName, serverPort);
               System.out.println("Connected: " + socket);
               start();
            }
            catch(UnknownHostException uhe)
            {  System.out.println("Host unknown: " + uhe.getMessage()); }
            catch(IOException ioe)
            {  System.out.println("Unexpected exception: " + ioe.getMessage()); }
         }
         public void run()
         {  while (thread != null)
            {  try
               {  streamOut.writeUTF(console.readLine());
                  streamOut.flush();
```

```
            }
            catch(IOException ioe)
            {  System.out.println("Sending error: " + ioe.getMessage());
               stop();
            }
         }
      }
      public void handle(String msg)
      {  if (msg.equals(".bye"))
         {  System.out.println("Good bye. Press RETURN to exit ...");
            stop();
         }
         else
            System.out.println(msg);
      }
      public void start() throws IOException
      {  console   = new DataInputStream(System.in);
         streamOut = new DataOutputStream(socket.getOutputStream());
         if (thread == null)
         {  client = new ChatClientThread(this, socket);
            thread = new Thread(this);
            thread.start();
         }
      }
      public void stop()
      {  if (thread != null)
         {  thread.stop();
            thread = null;
         }
         try
         {  if (console   != null)  console.close();
            if (streamOut != null)  streamOut.close();
            if (socket    != null)  socket.close();
         }
         catch(IOException ioe)
         {  System.out.println("Error closing ..."); }
         client.close();
         client.stop();
      }
      public static void main(String args[])
      {  ChatClient client = null;
         if (args.length != 2)
            System.out.println("Usage: java ChatClient host port");
         else
            client = new ChatClient(args[0], Integer.parseInt(args[1]));
      }
   }

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

   public class ChatClientThread extends Thread
   {  private Socket           socket   = null;
      private ChatClient       client   = null;
      private DataInputStream  streamIn = null;

      public ChatClientThread(ChatClient _client, Socket _socket)
      {  client   = _client;
         socket   = _socket;
         open();
         start();
      }
      public void open()
      {  try
         {  streamIn  = new DataInputStream(socket.getInputStream());
```

```
      }
      catch(IOException ioe)
      {  System.out.println("Error getting input stream: " + ioe);
         client.stop();
      }
   }
   public void close()
   {  try
      {  if (streamIn != null) streamIn.close();
      }
      catch(IOException ioe)
      {  System.out.println("Error closing input stream: " + ioe);
      }
   }
   public void run()
   {  while (true)
      {  try
         {  client.handle(streamIn.readUTF());
         }
         catch(IOException ioe)
         {  System.out.println("Listening error: " + ioe.getMessage());
            client.stop();
         }
      }
   }
}
```

Stage 5: Chat client moved to very simple GUI interface

```
import java.net.*;
import java.io.*;
import java.applet.*;
import java.awt.*;
public class ChatClient extends Applet
{  private Socket socket              = null;
   private DataInputStream  console   = null;
   private DataOutputStream streamOut = null;
   private ChatClientThread client    = null;
   private TextArea  display = new TextArea();
   private TextField input   = new TextField();
   private Button    send    = new Button("Send"), connect = new Button("Connect"),
                     quit    = new Button("Bye");
   private String    serverName = "localhost";
   private int       serverPort = 4444;

   public void init()
   {  Panel keys = new Panel(); keys.setLayout(new GridLayout(1,2));
      keys.add(quit); keys.add(connect);
      Panel south = new Panel(); south.setLayout(new BorderLayout());
      south.add("West", keys); south.add("Center", input);  south.add("East", send);
      Label title = new Label("Simple Chat Client Applet", Label.CENTER);
      title.setFont(new Font("Helvetica", Font.BOLD, 14));
      setLayout(new BorderLayout());
      add("North", title); add("Center", display);  add("South",  south);
      quit.disable(); send.disable(); getParameters(); }
   public boolean action(Event e, Object o)
   {  if (e.target == quit)
      {  input.setText(".bye");
         send();  quit.disable(); send.disable(); connect.enable(); }
      else if (e.target == connect)
      {  connect(serverName, serverPort); }
      else if (e.target == send)
```

```
            {  send(); input.requestFocus(); }
            return true; }
         public void connect(String serverName, int serverPort)
         {  println("Establishing connection. Please wait ...");
            try
            {  socket = new Socket(serverName, serverPort);
               println("Connected: " + socket);
               open(); send.enable(); connect.disable(); quit.enable(); }
            catch(UnknownHostException uhe)
            {  println("Host unknown: " + uhe.getMessage()); }
            catch(IOException ioe)
            {  println("Unexpected exception: " + ioe.getMessage()); } }
         private void send()
         {  try
            {  streamOut.writeUTF(input.getText()); streamOut.flush(); input.setText(""); }
            catch(IOException ioe)
            {  println("Sending error: " + ioe.getMessage()); close(); } }
         public void handle(String msg)
         {  if (msg.equals(".bye"))
            {  println("Good bye. Press RETURN to exit ...");  close(); }
            else println(msg); }
         public void open()
         {  try
            {  streamOut = new DataOutputStream(socket.getOutputStream());
               client = new ChatClientThread(this, socket); }
            catch(IOException ioe)
            {  println("Error opening output stream: " + ioe); } }
         public void close()
         {  try
            {  if (streamOut != null)  streamOut.close();
               if (socket    != null)  socket.close(); }
            catch(IOException ioe)
            {  println("Error closing ..."); }
            client.close();  client.stop(); }
         private void println(String msg)
         {  display.appendText(msg + "\n"); }
         public void getParameters()
         {  serverName = getParameter("host");
            serverPort = Integer.parseInt(getParameter("port")); }
      }
```