New and improved SimpleChatClient

Way back near the beginning of this chapter, we built the SimpleChatClient that could *send* outgoing messages to the server but couldn't receive anything. Remember? That's how we got onto this whole thread topic in the first place, because we needed a way to do two things at once: send messages *to* the server (interacting with the GUI) while simultaneously reading incoming messages *from* the server, displaying them in the scrolling text area.

Yes, there really IS an end to this chapter. But not yet...

```
import java.io.*;
import java.net.*;
import java.util.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class SimpleChatClient {

    JTextArea incoming;
    JTextField outgoing;
    BufferedReader reader;
    PrintWriter writer;
    Socket sock;

    public static void main(String[] args) {
       SimpleChatClient client = new SimpleChatClient();
       client.go();
    }

    public void go() {

        JFrame frame = new JFrame("Ludicrously Simple Chat Client");
        JPanel mainPanel = new JPanel();
        incoming = new JTextArea(15,50);
        incoming.setLineWrap(true);
        incoming.setWrapStyleWord(true);
        incoming.setEditable(false);
        JScrollPane qScroller = new JScrollPane(incoming);
        qScroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
        qScroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
        outgoing = new JTextField(20);
        JButton sendButton = new JButton("Send");
        sendButton.addActionListener(new SendButtonListener());
        mainPanel.add(qScroller);
        mainPanel.add(outgoing);
        mainPanel.add(sendButton);
        setUpNetworking();

        Thread readerThread = new Thread(new IncomingReader());
        readerThread.start();

        frame.getContentPane().add(BorderLayout.CENTER, mainPanel);
        frame.setSize(400,500);
        frame.setVisible(true);

     } // close go
```

This is mostly GUI code you've seen before. Nothing special except the highlighted part where we start the new 'reader' thread.

We're starting a new thread, using a new inner class as the Runnable (job) for the thread. The thread's job is to read from the server's socket stream, displaying any incoming messages in the scrolling text area.

```
private void setUpNetworking() {

     try {
        sock = new Socket("127.0.0.1", 5000);
        InputStreamReader streamReader = new InputStreamReader(sock.getInputStream());
        reader = new BufferedReader(streamReader);
        writer = new PrintWriter(sock.getOutputStream());
        System.out.println("networking established");
     } catch(IOException ex) {
        ex.printStackTrace();
     }
  } // close setUpNetworking
```

We're using the socket to get the input and output streams. We were already using the output stream to send to the server, but now we're using the input stream so that the new 'reader' thread can get messages from the server.

```
public class SendButtonListener implements ActionListener {
   public void actionPerformed(ActionEvent ev) {
       try {
          writer.println(outgoing.getText());
          writer.flush();

       } catch(Exception ex) {
          ex.printStackTrace();
       }
       outgoing.setText("");
       outgoing.requestFocus();
   }
 }  // close inner class
```

Nothing new here. When the user clicks the send button, this method sends the contents of the text field to the server.

```
public class IncomingReader implements Runnable {
   public void run() {
         String message;
         try {

           while ((message = reader.readLine()) != null) {
              System.out.println("read " + message);
              incoming.append(message + "\n");

            } // close while
         } catch(Exception ex) {ex.printStackTrace();}
     } // close run
 } // close inner class
} // close outer class
```

This is what the thread does!!

In the run() method, it stays in a loop (as long as what it gets from the server is not null), reading a line at a time and adding each line to the scrolling text area (along with a new line character).

Ready-bake Code

The really really simple Chat Server

You can use this server code for both versions of the Chat Client. Every possible disclaimer ever disclaimed is in effect here. To keep the code stripped down to the bare essentials, we took out a lot of parts that you'd need to make this a real server. In other words, it works, but there are at least a hundred ways to break it. If you want a Really Good Sharpen Your Pencil for after you've finished this book, come back and make this server code more robust.

Another possible Sharpen Your Pencil, that you could do right now, is to annotate this code yourself. You'll understand it much better if you work out what's happening than if we explained it to you. Then again, this is Ready-bake code, so you really don't have to understand it at all. It's here just to support the two versions of the Chat Client.

To run the chat client, you need two terminals. First, launch this server from one terminal, then launch the client from another terminal

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

public class VerySimpleChatServer {

    ArrayList clientOutputStreams;

    public class ClientHandler implements Runnable {
         BufferedReader reader;
         Socket sock;

         public ClientHandler(Socket clientSocket) {
           try {
             sock = clientSocket;
             InputStreamReader isReader = new InputStreamReader(sock.getInputStream());
             reader = new BufferedReader(isReader);

           } catch(Exception ex) {ex.printStackTrace();}
          } // close constructor

        public void run() {
           String message;
           try {
             while ((message = reader.readLine()) != null) {
                System.out.println("read " + message);
                tellEveryone(message);

              } // close while
           } catch(Exception ex) {ex.printStackTrace();}
       } // close run
    } // close inner class
```

```
    public static void main (String[] args) {
         new VerySimpleChatServer().go();
    }

    public void go() {
       clientOutputStreams = new ArrayList();
       try {
         ServerSocket serverSock = new ServerSocket(5000);

         while(true) {
            Socket clientSocket = serverSock.accept();
            PrintWriter writer = new PrintWriter(clientSocket.getOutputStream());
            clientOutputStreams.add(writer);

            Thread t = new Thread(new ClientHandler(clientSocket));
            t.start();
            System.out.println("got a connection");
         }

       } catch(Exception ex) {
         ex.printStackTrace();
       }
   } // close go

   public void tellEveryone(String message) {

      Iterator it = clientOutputStreams.iterator();
      while(it.hasNext()) {
         try {
            PrintWriter writer = (PrintWriter) it.next();
            writer.println(message);
            writer.flush();
         }  catch(Exception ex) {
              ex.printStackTrace();
         }

      } // end while

   } // close tellEveryone
} // close class
```