Internet Programming 300130

WORKSHOP-7

(5 Marks, due by Week 11)

Exercise 7.1 (1 mark)

Short Answers

(Consult Lecture 7)

1. Describe briefly the main technical steps involved in setting up a network server in Java via TCP/IP.

2. Describe briefly the main technical steps involved in setting up a network client in Java via TCP/IP.

Exercise 7.2 (2 marks)

Experience A Simple Chat Server

1. The following program **SChatServer.java** can communicate with multiple clients concurrently:

```
// file: SChatServer.java
import java.net.*;
import java.io.*;
import java.util.*;

class SChatServer {
    public SChatServer (int port) throws IOException {
      ServerSocket server=new ServerSocket(port);
      while(true) {
          Socket client=server.accept();
          new SChatter(client).start();
      }
    }

    public static void main(String[] args) throws IOException {
      new SChatServer(Integer.parseInt(args[0]));
    }
}

class SChatter extends Thread {
    private static int count=0;
    private static Vector chatters=new Vector();
```

```
        private String id="#"+Integer.toString(++count);
        private Socket s;
        private BufferedReader in;
        private PrintWriter out;

        public SChatter(Socket socket) throws IOException {
          s=socket;
          in=new BufferedReader(new
                 InputStreamReader(s.getInputStream()));
          out=new PrintWriter(s.getOutputStream(),true);
          broadcast("** user "+id+" has just joined in");
        }

        public void run() {
          try {
              chatters.add(this);
              while(true) { broadcast(id+": "+in.readLine()); }
          } catch(IOException e) {}
          finally { try{ s.close(); } catch(IOException e) {} }
        }

        private void broadcast(String msg) {
          synchronized(chatters) {
              for(int i=0;i<chatters.size();i++) {
                  SChatter c=(SChatter) chatters.get(i);
                  synchronized(c.out) { c.out.println(msg); }
              }
          }
        }
    }
```

Read, compile and execute **SChatServer** on **student** machine at a port **5xxx**, where **xxx** is the last 3 digits of your student ID. This is to avoid multiple students using the same port number.

2. Compile a simple client **TextClient.java**. Open another 2 windows on **student** machine, and then connect to the server via

```
java TextClient student 5xxx
```

You can now chat with one another, albeit with yourself.

3. Recall that the following trivial client LameClient.java is provided in the lecture

```
// file: LameClient.java
import java.io.*;
import java.net.*;

public class LameClient {
    public static void main(String[] args) throws IOException {
      String remoteHost=args.length>0? args[0]:"127.0.0.1";
      Socket socket=new Socket(remoteHost, 5678);
      System.out.println(socket);
      try {
          BufferedReader in=new BufferedReader(new
                           InputStreamReader(
                            socket.getInputStream()));
          PrintWriter out=new PrintWriter(
                         socket.getOutputStream(),true);
          out.println("client sends greetings");
          String str=in.readLine(); // blocks for a message
```

```
                    System.out.println(str);
                    out.println("END");
                } finally { socket.close(); }
            }
        }
```

This lame client says just "*client sends greetings*" and "*END*", then immediately terminates itself. However it does illustrate the basic ingredients of a client. This Java program is hardwired to the port **5678**, so you may have to change this number in order to avoid conflict with other fellow students using the same port number.

5. To test interactions between **SChatServer**, **TextClient** and **LameClient**, open 3 windows on **student** machine.

- in the 1st window run

```
java SChatServer 5678
```

- then in the 2nd window run

```
java TextClient student 5678
```

- then in the 3rd window run

```
java LameClient
```

We will see the messages sent by **LameClient** are displayed in the 2nd window. If you are having problems, it may be due to the fact that someone else is also using the port **5678**. Then just use another port number (you will also have to change the port number in **LameClient.java** accordingly).

Exercise 7.3 (2 marks)

Analysing the Simple Chat Server

Read and understand the source code **SChatServer.java**. Answer the following questions:

1. Explain the purposes of the classes: SChatServer class and SChatter class.

2. Explain the purposes of the following two lines

```
        Socket client=server.accept();
        new SChatter(client).start();
```

in **SChatServer.java**.

NOTE:

1. The simple chat server, **SChatServer**, can be set up by compiling and running the Java program SChatServer.java. The server can't be run twice on the same machine, unless you change the port number for your own use and testing.

2. For testing, do **not** run **SChatServer** at *background* so that you won't forget to terminate the server after the testing.

3. With **ShatCServer**, every chat client will be automatically given an integer ID as the nickname for the chatter. A client leaves by press CTRL-C.

4. The messages coming from different chatters will take the form

```
#name: the actual message line then follows
```

5. If all other chatters are getting continuous **null** message when one chatter quits (program terminates), then you can modify the line in SChatServer.java containing

```
while(true) { broadcast(id+": "+in.readLine()); }
```

so as to fix the problem.