Homework Assignment 2
Java Sockets
ECE 155B – Spring 2012
Due May 2, 2012

In the first assignment you explored Java's "write-once-run-anywhere" feature by implementing your Healthcare Services Application as a Java Applet running in a browser. A secure program for a Healthcare Service is unlikely to run as an Applet in a browser. Therefore, in this assignment, you will develop a Java Application that has almost the same user interface as the first assignment but, instead of an Applet, it will be a Java Application.

Requirements

In this assignment you will revise and extend your Healthcare Service Application to use a Java Client Application (instead of a Java Applet) communicating with a Java Server Application using Java Sockets. In addition to the *Doctor*, you will introduce a *Patient* at this stage of the project. The Patient will act as a client, and the Doctor as a server. In general, there can be multiple Doctors for a patient and multiple Patients of a Doctor (i.e., use *multithreading* for both the Doctor and the Patient). The demo for this assignment will have two Doctors and two Patients, as shown in Figure 1.

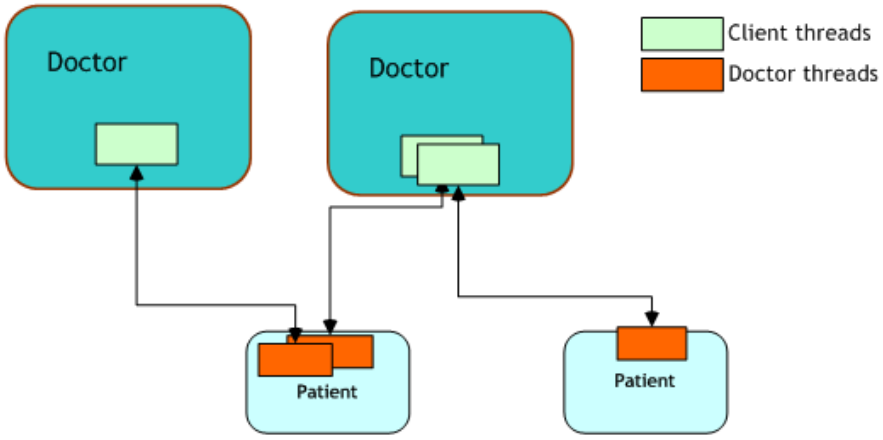

Figure 1. Demo Scenario.

The Doctor Application must have a Graphical User Interface (GUI), and provide the same functionality as your Java Applet. You might consider using *JTabbedPane* in the GUI for the Doctor to display patient-specific information and data, and another tab to display doctor-specific information, in order to have a simple GUI.

The Doctor has a calendar that keeps track of future appointments. A Patient connects to the Doctor Application and requests an appointment date. While a Patient is doing that, other Patients cannot request appointments. You will be asked to demonstrate this feature during the demo.

In addition, after examining the Patient, the Doctor will associate Notes with the specific appointment with the Patient. At the end of the appointment, the Doctor will also prepare a prescription for the Patient. The data structures to use are shown on the next page.

Doctor Data Structure

- Doctor first name, last name, etc.
- **Patients** (for each patient)
• Information about patient
  ○ First name, last name
  ○ Birth date
  ○ Phone number
  ○ Address, etc.
• **Appointments** *(for each appointment)*
  ○ Date/time of appointment
  ○ Doctor Name
  ○ **Notes** associated with the appointment *(for each note)*
  ▪ Author
  ▪ Date
  ▪ Content, etc.
  ○ Prescription
  ▪ List of medicines
  ▪ Prescription date
  ▪ Expiration date, etc.

You will be provided with a list of medicine information (in an XML-formatted file) from which the Doctor can make a selection. At the beginning, the Doctor application will load medicine information from that file.

A Patient requests an appointment from the Doctor Application for a particular day. This communication will be done using Java Sockets. The Patient saves the appointment information in a Patient file for future reference. The Patient saves the following information to that file.

Patient data structure

Patient
• Doctor List
  ○ Name of doctor, URL:port information
• Appointments
  ○ Date/time of appointment
  ○ Doctor name

Socket communication requires, in addition to TCP/IP, a common application-layer protocol between communicating clients and servers. You will need to design the communication methods before you start coding. The client needs to know how to ask for information, and the server needs to know how to analyze a client's request and respond in an appropriate way.

Design Approach

In a project like this, it is better first to design a system where a client simply connects and the server greets the client or replies with the same message (echo server). Then, using multiple threads, you can allow more clients to connect to the server and to send multiple messages to the server over their respective open connections.

Design Considerations

When transferring data between the Java client application and the Java server application, you will pass XML-formatted messages between the client and the server. You might need a common Message object and special XML parsers to do this. You will need *three different XML parsers:*

1 Messages exchanged between *Doctor* and *Patient*
2 *Doctor* (to keep doctor-specific information)
3 *Patient* (to keep patient-specific information).

For this project, the client needs to send well-formatted messages that the server will be able to understand. Consider the following:

1 ***Authenticate Patient***
The Patient sends name, social security number.

2 ***Authenticate Patient Reply***
The Doctor positively acknowledges the Patient if there is such a Patient record; otherwise, the communication ends with a ***terminate message***.

3 ***Request Time Availability of Doctor***
The Patient requests time availability of the Doctor for a specific date.

4 ***Request Time Availability Reply***
The Doctor replies with the times that he/she is available on that date, if any. If the Doctor has no times available, the Doctor replies with "No Times Available" and the Patient must reissue his/her request for another date. If there is another Patient currently communicating with the Doctor, you need to inform the client to send his/her request later.

5 ***Request Appointment***
The Patient requests an appointment for a specific date/time with the Doctor.

6 ***Request Appointment Reply***
The Doctor sends an acknowledgement to the Patient indicating that the appointment at the particular date/time has been made.

7 ***More as you wish.***

You will define your own protocol, i.e., how the client asks for information and how the server understands and responds to the client. It is not meaningful for the client to send messages that the server does not understand, and vice versa. Therefore, before implementing your protocol, make sure you have thought about every little detail.

Report (10%)

In addition to the programming portion of this assignment, you will write a small report. The report should be clear, well organized and concise. It needs to include the following:
1) An overview of how your system operates
2) Details of your design
3) A class diagram sketch outlining the hierarchy of your class structure
4) Sample input and output, e.g., the content of your XML elements before and after an operation, such as add or update.

Demonstration (10%)

You will be performing project demonstrations for this assignment and all others. You will be responsible for scheduling a 5-15 minute timeslot for your demonstration with the TA. Because time is limited, you should run your application on the PCs on which you will do your demo before the demo. In particular, make sure that those PCs are set up correctly. Demonstrations will be on Fridays, and will fall between specific times. You will be required to demonstrate your application's functionality, and answer several questions covering the technology and your particular implementation. Demonstrations for the assignments will be scheduled through email, or during office hours/discussion.

Submission (10%)

You will submit a compressed file *(tar)*. You will need to type "*ant submission*" to create your submission file. If there is anything missing in the submission file, edit the *build.xml* file to include the missing information. Make sure you modify the *User* property in the *build.xml* file. The *ant* call will create the submission file with the username you provide.

The compressed file should include a folder with all project content necessary to build/run the program (.java, README, Report). The projects are to be sent to ece155b.s12@gmail.com with subject line "Project 02: FirstName LastName".