HST 952 Fall 2002

Project

The project builds upon the database you designed in Homeworks 8 and 9. It also will utilize many of the skills you have developed in previous segments of this course.

You will develop a part of the laboratory information system. The functionality you will add is the ability to view a patient's test results and perform rudimentary decision support using the database. You do not need to add functionality to place orders so you should not have to implement classes you do not need.

1. If your schema does not yet support it, change the schema so that patient test orders and results are stored as test codes instead of names. A set of codes is provided in a table on a later page. You should also add data about patients from the Sample Patient file. This has a little more data than was given in Homework 9. **(5 points)**
2. Create Java classes for entities you had created in your LIS E-R diagram to support viewing of test results. Possible classes may be Test, Patient, Result, etc. The actual set of classes will depend on your E-R and database design. The classes must contain variables for each of the attributes of an entity and must have get/set functions for each of the attributes. Your E-R diagram should guide you in designing the classes. The relationships can be expressed as attributes of a class. For example, the relationship that a Patient has many tests can be expressed as follows:

```
public class Patient {
        private String patient_id;
        private String first_name;
        private Vector tests;

        // constructor
        // see below for explanation of db parameter
        public Patient(LIS_Database db) {

        }

        // load the orders from the database
        public void getTests() {
                Tests t = db.getTestsForPatient(this);
                // add them to our tests vector
        }

        public void addTest(Test t) {
                tests.add(t);
        }
}
```

   (25 points)

3. Design a class called LIS_Database that has public functions for the following
   a. Managing the connection to the database

b. Handling all input and output from the database.

The purpose of this class is to hide from the other classes of your application, such as Patient and Test, all the ugly details of how to deal with the database and how the storage is managed. The decoupling of classes from the database allows you to switch databases or make changes in your database design with minimal effects on the rest of your application.

An example stub of the class is provided. You may need different method signatures than the ones in the example below. You will probably need to add other public and private functions.

```
public class LIS_Database {
        private Connection conn;
        private String connection_url; // for the database

        // opens a connection to the database
        public void open() {
        }
        // closes the connection to the database
        public void close() {
        }

        public Patient getPatient(String id) {
        }
        public void addPatient(Patient pat) {
        }
        // add result for a test
        public void addResult(Patient pat, Test test, float r) {
        }
}
```

(15 points)

Write Java programs described below that utilizes the above classes. Not all of the functionality required can be handled by SQL so you may have to implement java algorithms for some of them.

4. Find all patients whose serum potassium value was above normal and whose next serum potassium value (measured between 7 and 14 days after the first one) has not decreased. Send the following alert to the physician who ordered the test.

   Patient <Insert Patient's Name Here> potassium not improving.
   <Test name> <Test result> <Date/time for second test>
   <Test name> <Test result> <Date/time for first test>

   The alert must be sent by e-mail. See the sample program later in this document as an example of how to send e-mail from a java program. Change the e-mail address to your own for testing purposes.

   (30 points)

5. Dr. Bean Counter would like to view all the tests for the month in an organized manner. He would like a program that sorts tests by analytes and by sample types.

The table below provides the Analytes and Sample Types for each of the tests. Write a program that takes as a command-line parameter the format of the report (i.e., organized by analyte or sample type). The output should consist of Patient name, test name, test result, and analyte or sample type.

Hint: you could store the table below as a new table in your database.

(25 points)

Test name	**Test code**	**Analyte**	**Sample type**
Serum sodium	01	Sodium	Serum
Serum potassium	02	Potassium	Serum
Total cholesterol	03	Cholesterol	Serum
Serum HCG	04	HCG	Serum
Hemoglobin	05	Hemoglobin	Whole blood
INR	06	Prothrombin	Whole blood
Blood urea nitrogen	07	Urea	Whole blood
Red blood cell (RBC) count	08	RBC	Whole blood
White blood cell (WBC) count	09	WBC	Whole blood
Urine Sodium	10	Sodium	Urine

Please mail all your database meta, java source code, and output of the programs to ta952@dsg.harvard.edu

Hint: Good documentation of software makes us happy ☺

SAMPLE CODE FOR SENDING E-MAIL

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

public class SendAlertEmail {

    public static void main (String[] args) {

        Socket socketSmtpServer = null;
        DataOutputStream dos    = null;
        BufferedReader dis      = null;
        // put your smtp server name between the quotes e.g. "smtp.harvard.edu"
        String smtpServerName = " ";
        // put your login name on that server between the quotes below
        String strMyName = " ";
        String strBuf; // stores messages from the SMTP server
        int portNumToListenTo = 25;

        /* generally, SMTP servers listen for connections on port 25, so
```

```
   to establish a connection to an SMTP host, we need to create
   a TCP/IP socket that is connected to port 25 of the SMTP host */
try {
    /** Establish a TCP/IP connection with the SMTP server you'll
          be using to send your e-mail */
    socketSmtpServer = new Socket(smtpServerName, portNumToListenTo);
    dos = new DataOutputStream(socketSmtpServer.getOutputStream());
    dis    = new BufferedReader(new
          InputStreamReader(socketSmtpServer.getInputStream()));
    /** Done establishing the TCP/IP connection */

    /** Log in to the SMTP server */
    strBuf = dis.readLine();
    System.out.println(strBuf);
    // The HELO command identifies you to the SMTP server
    dos.writeBytes("HELLO " + strMyName + "\n");
    strBuf = dis.readLine();
    System.out.println(strBuf);
    // The RSET command resets the state of the SMTP server
    dos.writeBytes("RSET\n");
    strBuf = dis.readLine();
    System.out.println(strBuf);
    /** Done logging in to server */

    /** Provide return and delivery addresses for your e-mail alert */
    // replace you@youraddress.com with your valid e-mail address
    // this will not work if you use an invalid e-mail address
    dos.writeBytes("MAIL FROM:<you@youraddress.com>\n");
    strBuf = dis.readLine();
    System.out.println(strBuf);
    // replace you@youraddress.com with your valid e-mail address
    dos.writeBytes("RCPT TO:<you@youraddress.com>\n");
    strBuf = dis.readLine();
    System.out.println(strBuf);
    /** Done providing return and delivery addresses */

    /** Construct data section of the e-mail alert including headers */
    dos.writeBytes ("DATA\n");
    strBuf = dis.readLine();
    System.out.println(strBuf);

    // create message headers
    /* Replace To: message with the appropriate physician's
       first name and last name and put your e-mail address
       in the angle brackets */
    dos.writeBytes ("To: Physicianfirstname Physicianlastname "
                      + "<physician@physiciansaddress.com>\n");
    /* Replace From: message with your first name and last name
       and put your e-mail address in the angle brackets */
    dos.writeBytes ("From: Yourfirstname Yourlastname "
                      + "<you@youraddress.com>\n");
    dos.writeBytes ("Subject: Medical alert about your patient!\n");
    dos.writeBytes("Content-Type: text/plain; "
                     + "charset=\"us-ascii\"\n\n");

    // create message body
```

```
            dos.writeBytes("Your patient: <name of patient here> has elevated "
                           + "potassium levels!\n");

            // indicate end of e-mail message body
            dos.writeBytes("\n.\n");

            // read line from SMTP server after indicating end of message body
            strBuf = dis.readLine();
            System.out.println(strBuf);
            /** Done constructing data section of the e-mail alert */

            /** wrap up e-mail session with SMTP server */
            dos.writeBytes("QUIT\n");
            strBuf = dis.readLine();
            System.out.println(strBuf);
            dos.close();
            dis.close();
            socketSmtpServer.close();
            System.out.println("Done with e-mail session");
            /** Done with e-mail session */
        }
        catch (UnknownHostException e) {
            System.out.println("Host " + smtpServerName + " unknown");
        }
         catch (IOException e) {
            System.out.println(e);
        }
    }
}
```