**The University of West Florida  
Department of Computer Science**

**Advanced Computer Programming Assignment**

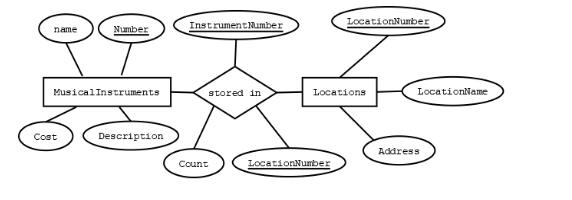
**John W. Coffey**

**Program Description**

We will write a three-tier architecture application with a client, server and backend database. The system will enable the user to issue queries to find information regarding musical instruments, prices, availabilities, and locations.

**The Server and Database**

The database schema looks like this:



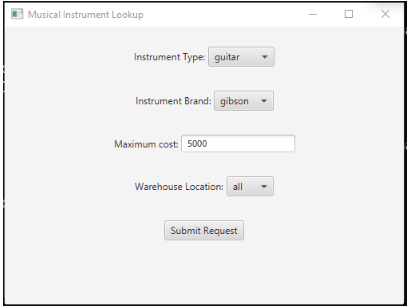
The database data will be standard and can be downloaded from the project assignment page. For testing purposes, your program should drop the tables in the database at the beginning of the server run, create the tables, and populate the tables with the data supplied with the project. The server should then wait for clients to connect. The program will run on localhost as the network tic tac toe game did.

You should use these methods to create the tables, add the test data, and confirm that everything worked:

public ResultSet createInstruments(Statement stat) throws Exception  
  {  
         stat.execute("CREATE TABLE Instruments (instName CHAR(12),instNumber INTEGER,cost DOUBLE,descrip CHAR(20))");  
         stat.execute("INSERT INTO Instruments VALUES ('guitar',1,100.0,'yamaha')");  
         stat.execute("INSERT INTO Instruments VALUES ('guitar',2,500.0,'gibson')");  
         stat.execute("INSERT INTO Instruments VALUES ('bass',3,250.0,'fender')");  
         stat.execute("INSERT INTO Instruments VALUES ('keyboard',4,600.0,'roland')");  
         stat.execute("INSERT INTO Instruments VALUES ('keyboard',5,500.0,'alesis')");  
         ResultSet result = stat.executeQuery("SELECT \* FROM Instruments");  
         return result;  
  }  
  
  public ResultSet createLocations(Statement stat) throws Exception  
  {  
         stat.execute("CREATE TABLE Locations (locName CHAR(12),locNumber INTEGER,address CHAR(50))");  
         stat.execute("INSERT INTO Locations VALUES ('PNS',1,'Pensacola Florida')");  
         stat.execute("INSERT INTO Locations VALUES ('CLT',2,'Charlotte North Carolina')");  
         stat.execute("INSERT INTO Locations VALUES ('DFW',3,'Dallas Fort Worth Texas')");  
         ResultSet result = stat.executeQuery("SELECT \* FROM Locations");  
         return result;  
  }  
  
  public ResultSet createInventory(Statement stat) throws Exception  
  {  
         stat.execute("CREATE TABLE Inventory (iNumber INTEGER,lNumber INTEGER,quantity INTEGER)");  
         stat.execute("INSERT INTO Inventory VALUES (1,1,15)");  
         stat.execute("INSERT INTO Inventory VALUES (1,2,27)");  
         stat.execute("INSERT INTO Inventory VALUES (1,3,20)");  
         stat.execute("INSERT INTO Inventory VALUES (2,1,10)");  
         stat.execute("INSERT INTO Inventory VALUES (2,2,10)");  
         stat.execute("INSERT INTO Inventory VALUES (2,3,35)");  
         stat.execute("INSERT INTO Inventory VALUES (3,1,45)");  
         stat.execute("INSERT INTO Inventory VALUES (3,2,10)");  
         stat.execute("INSERT INTO Inventory VALUES (3,3,17)");  
         stat.execute("INSERT INTO Inventory VALUES (4,1,28)");  
         stat.execute("INSERT INTO Inventory VALUES (4,2,10)");  
         stat.execute("INSERT INTO Inventory VALUES (4,3,16)");          
         ResultSet result = stat.executeQuery("SELECT \* FROM Inventory");  
         return result;  
  }

**The Client Program**

The client program will run after the server has been started. It should be able to handle a failure to connect to the server without exceptions or abnormal termination. The client should present a user interface that allows the user to enter searches for musical instruments, communicate the searches to the server, and receive results back indicating availability and locations of the instrument. We will use ChoiceBoxes to provide pull-down menus to the user. A basic client interface might look something like this:



Note that the instrument brand ONLY populates after an instrument type has been chosen, since different types of instruments have different brand names (think Ludwig drums and Fender guitars). The client interface must be built in such a way that it always sends all the information the server needs. For instance, if no maximum cost is specified, the server should identify all instruments that meet the other criteria, regardless of cost.

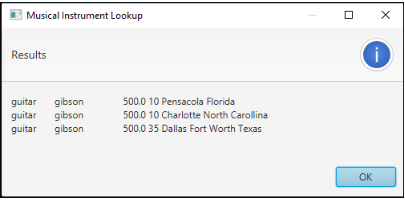
The client produces string parameters to be sent to the server:

* musical instrument type (all, guitar, bass, drums, keyboard, etc.),
* brand (Ludwig, Gibson, Fender, etc, conditioned by selection of instrument type),
* maximum cost (which can be unspecified)
* warehouse where gear is located.

The server receives the parameters (some of which can be 'all' or 'null') from the client. The server hands the parameters to a thread that builds the SQL query for the request, issues the query to the backend, gets the result as a ResultSet, builds an ArrayList of Strings representing the result, and ships the strings back to the client. The client pops up a child window with the results.

The database has three tables - instruments, warehouses, and inventory in the various warehouses, so the SQL queries have to do joins on multiple relations and have highly variable WHERE clauses. A typical query might be: all Gibson guitars under $500 located in the PNS warehouse (I used airport codes for the locations :-) )

The results displayed on the client might look like this:



This result indicates Gibson guitars in three locations, each being $500.00 with 10 guitars in Pensacola, 10 in Charlotte, N.C., and 35 guitars in DFW.

**Deliverables**

You will submit all the usual things:  
1. A UML class diagram

2. A User's manual for your program  
3. Your source code in Java  

Please review the policy on Academic dishonesty. This is an individual project.

**Submission:**

1. Compile and run your program one last time before submitting it.

2. Create a "zip"  file (using WinZip or similar program) to hold your project files.

The name of your zip file should be: your first initial, your last name, a dash (-), the letter 'p', the project number, followed by the .zip extension. For example, if your name is John Coffey, and you are submitting project 5, your zip file should be called jcoffey-p5.zip.

3. Login to UWF's Canvas system. Select our course.

4. Select the appropriate Assignment. Click the Submit Assignment button and upload your zip file..

5. Check the to insure that the file was uploaded. You MUST do this with enough time before the deadline to account for slow server response and differences in your computer's time and the server's time