Using JTables with a SQL Database

Tables are always great ways to display lots of data at once. Me and my group decided to use a table to display all the patients in our database. This pdf will show you how I implemented them. It assumes that you already know how to interface Java with your SQL database using the JDBC driver and associated packages.

1. Setting up the JTable Model

A JTable runs on a table model. By default the constructor takes in arrays of Objects that act as it's data and it's column names. However, static data isn't useful when interfacing with dynamic data in a database. In order to complete this task you'll want to make your own table model. This model tells the table what data to display, it's much more flexible than the default because you write it to do what you want.

First things first you'll want to make the table model. This model extends AbstractTableModel and all of it's methods. A simple class can be written with the header:

```
public class TableModel extends AbstractTableModel
```

Now, if you're using eclipse, all the needed implementation methods will be imported automatically. These methods can be found in the AbstractTableModel api entry for Java SDK 6.

This model will be what actually pulls the data down from the database, meaning this model has a very specific task. Only one table can use this model, other table's need their own models to pull their own data.

You'll want to call some variables that the table model will read as instance fields:

```
private Object[][] data;


private String columnNames[] = {"First Name", "Last Name", "Date of
Birth", "Social Security Number"};
```

The data array is a 2d array holding the actual table contents and the columnNames array holds the column names. Now, a 2d array isn't very flexible when pulling down a varying amount of data from the database. In order to surmount this, I used ArrayLists for each column. For example, the "First Name" column's pull down would look like this.

```
try {
	Statement stmt = con.createStatement();
	ResultSet rs = stmt.executeQuery("SELECT * FROM `patients`");
	while(rs.next()){
		patientFirstNames.add(rs.getString("P_FIRSTNAME"));
	}

} catch (SQLException e) {
	// TODO Auto-generated catch block
	e.printStackTrace();
}
```

Once again this assumes that you know how to utilize the JDBC and all it's components.

You'll want to initialize your 2D data array here.

```
data = new Object[patientFirstNames.size()][4];
```

Where the second number is the number of columns that your table will have.

Now that you have this ArrayList of data you want to put it into the first column of the data array. This is just a simple for loop.

```
for(int i = 0; i<patientFirstNames.size();i++){
	data[i][0] = patientFirstNames.get(i);
}
```

Do this for the other columns also. You will have to add all the data to their corresponding ArrayLists in the previous try statement and then add them to the data array. The rest of the necessary methods are very simple, their names will tell you what to do.

2. **Initializing the table with the new model.**

Now that we have this model, we need to add it to our JTable. The easy way would just be to say

```
JTable tblDoctorPatientList = new JTable(new TableModel());
```

The way I did it though, and this allows for other classes to modify the table model and simulate live updating is to create a myModel() method such as this:

```
/**
 * The myModel method establishes the TableModel that powers the class.
 */
public TableModel myModel() throws Exception{
	TableModel model = new TableModel(mac);
	return model;
}
```

Then all you have to do is call this method in the new initialization of the table:

```
tblDoctorPatientList = new JTable(myModel());
```

Put this table in a scrollpane and you've got yourself a great new JTable interfaced with SQL!