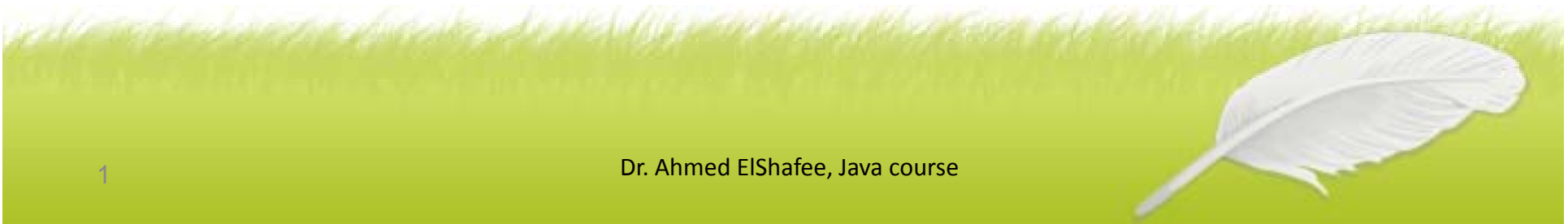




Lecture (16)

Database II

Dr. Ahmed ElShafee



Agenda

Databases and Java Forms

- In this section, you'll create a form with buttons and text fields.
- The buttons will be used to scroll forwards and backwards through the records in a database table.
- We'll also add buttons to perform other common database tasks.
- The form you'll design will look something like this:

1 Helen James

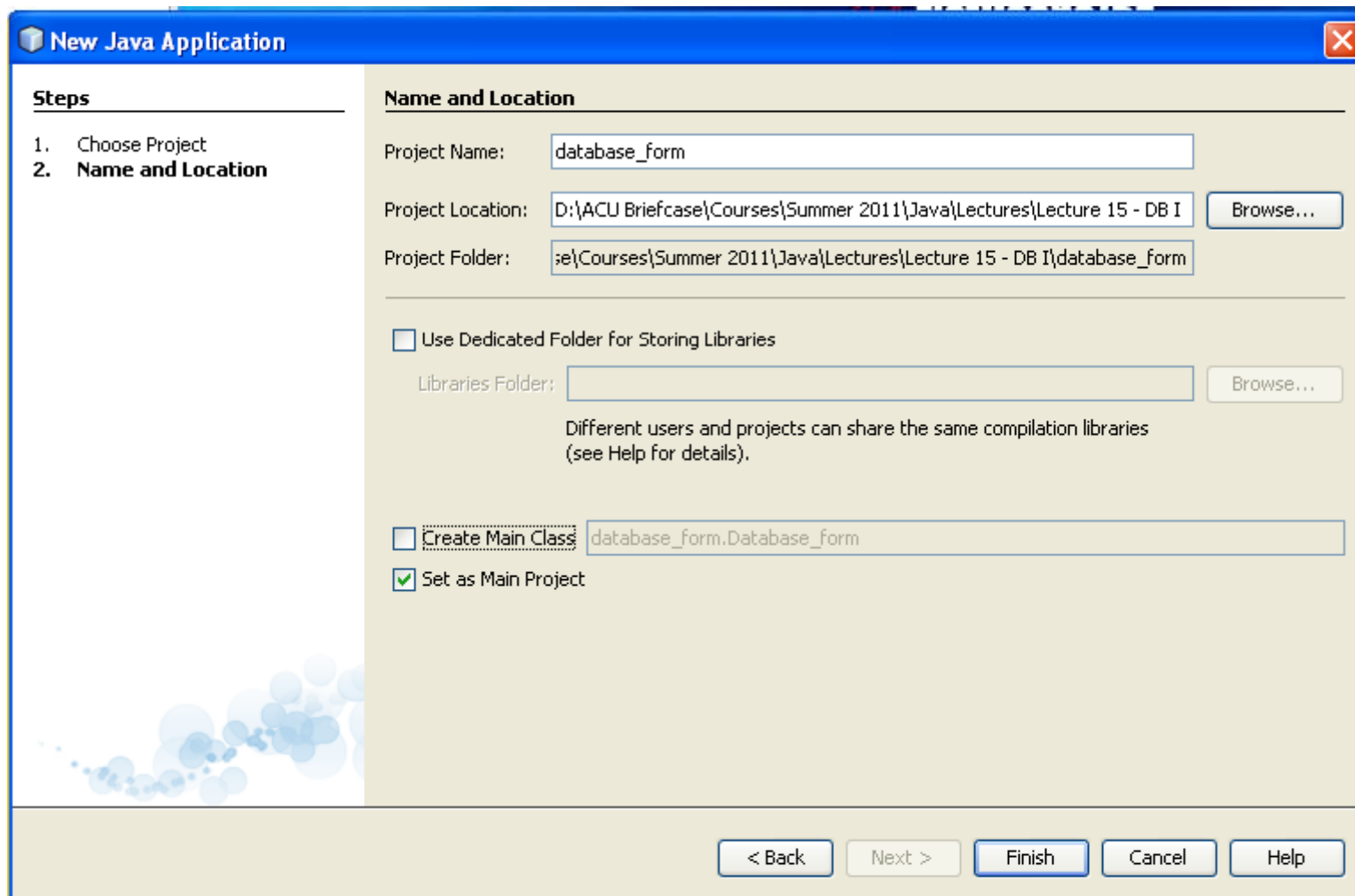
Job Title: IT Manager

First Previous Next Last

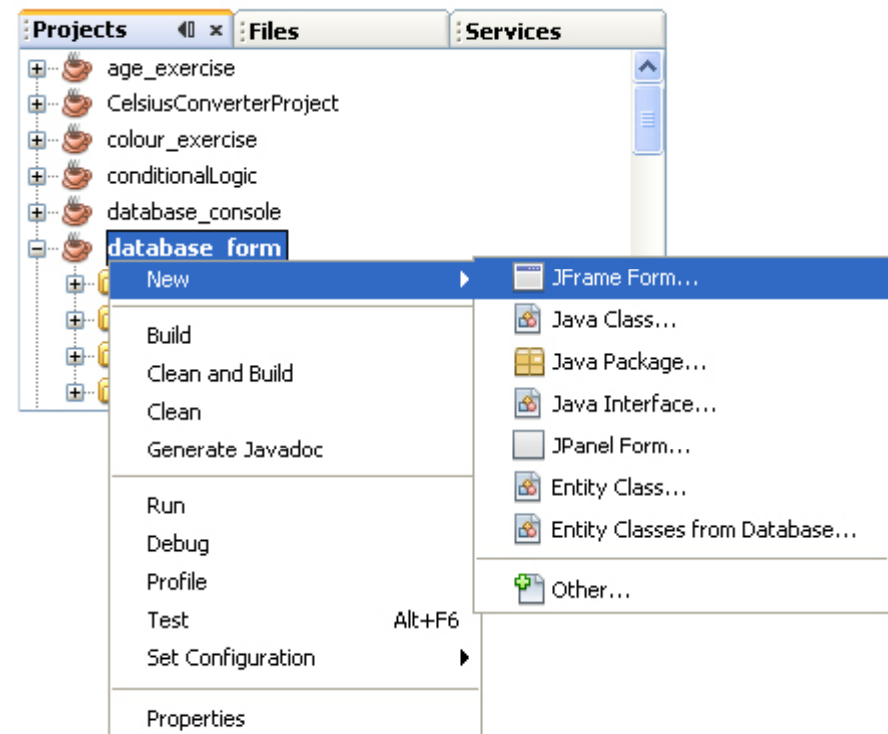
Update Record Delete Record New Record

Save New Record Cancel New Record

-
- Start a new project for this by clicking **File > New Project** from the NetBeans menu.
 - When the dialogue box appears, select **Java > Java Application**.
 - On step one of the dialogue box, type **database_form** as the Project Name.
 - Uncheck the box at the bottom for **Create Main Class**.
 - Click the Finish button to create an empty project.



- In the Project area on the left locate your **database_form project, and right click** the entry.
- From the menu that appears select **New > JFrame Form:**



- When the dialogue box appears, type **Workers** for the Class name, and **Employees** as the package name.
- When you click **Finish**, you should see a **blank form** appear in the main NetBeans window.

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

Class Name:

Project:

Location:

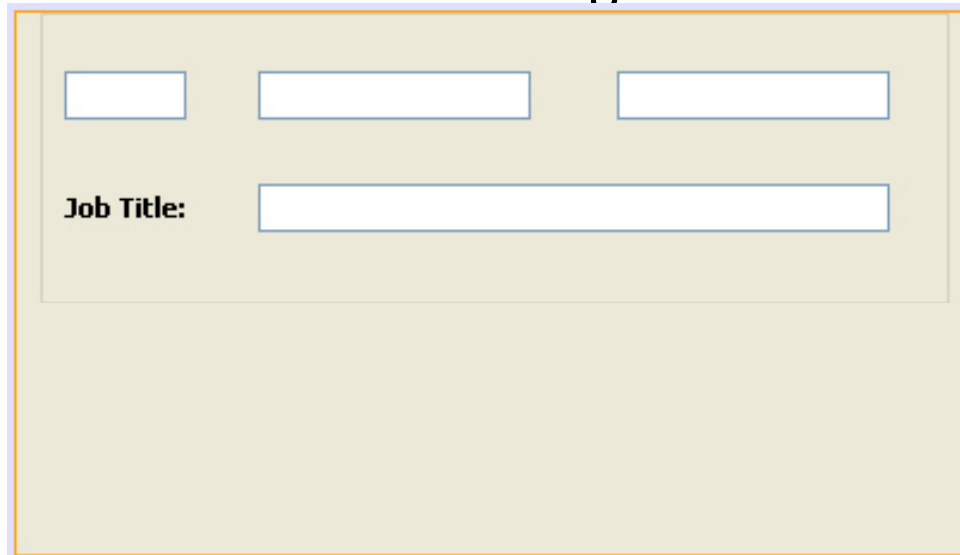
Package:

Created File:

< Back Next > **Finish** Cancel Help

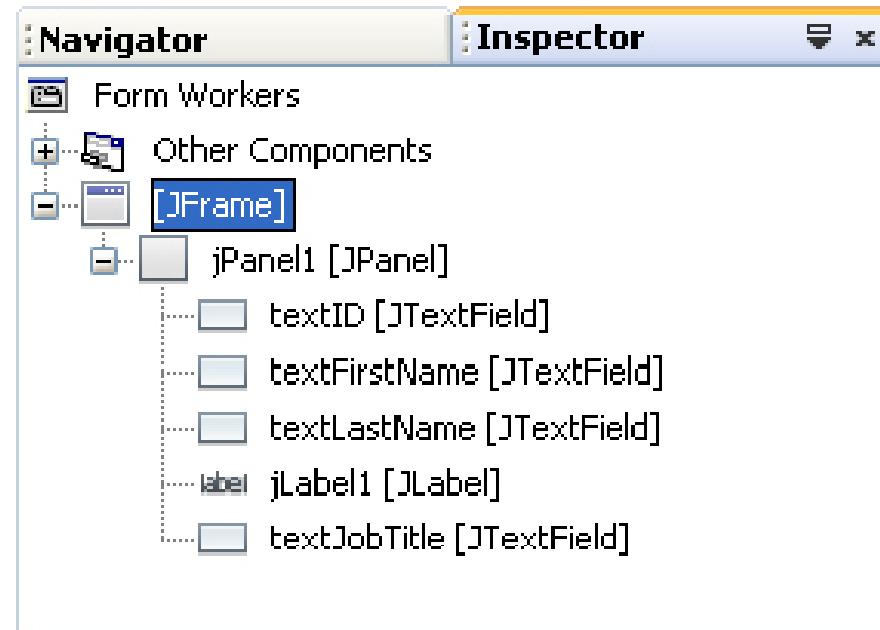
-
- Add a Panel to your form. Then place four Text Fields on the panel.
 - Delete the default text for the Text Fields, leaving them all blank.
 - Change the default variable names for the Text Fields to the following:
 - **textID**
 - **textFirstName**
 - **textLastName**
 - **textJobTitle**

-
- Add a label to your panel.
 - Position it just to the left of the job title Text Field.
 - Enter “Job Title” as the text for the label.
 - Arrange the Text Fields and the Label
 - so that your form looks something like this:



The image shows a light beige rectangular panel with a thin orange border. Inside the panel, there are four text input fields. At the top, three small, empty rectangular text boxes are arranged horizontally. Below them, on the left side, is the text "Job Title:" in a bold, black font. To the right of this label is a single, larger, empty rectangular text box.

- Now have a look at the Inspector area to the left of NetBeans. (If you can't see it, click **Window > Inspector** from the NetBeans menu.)
- It should match :



-
- What we want to do now is to have the first record from the database table appear in the text fields when the form first loads.
 - To do that, we can call a method from the form's Constructor.
 - First, though, we can add **MySql Jconnector JAR** file to the project, just like last time.
 - This will prevent any "Driver Not Found" errors. So, in the Projects area, right click the Libraries entry for your project. From the menu that appears, select **Add JAR/Folder**.
 - When the dialogue box appears, locate the **mysql-connector-java-5.1.15-bin.jar** file.
 - Then click Open to add it to your project.

-
- In the main NetBeans window, click the Source button at the top to get to your code.
 - Now add the following import statements near the top:

```
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
import java.sql.Statement;  
import java.sql.ResultSet;  
import javax.swing.JOptionPane;
```

-
- Inside of the Class, add the following variable declarations:

Connection con;

Statement stmt;

ResultSet rs;

- Just below the Workers Constructor, add the following method:

```
public void DoConnect( ) {  
    }  
}
```

- Now add a call to this method from the Constructor:

```
public Workers() {  
    initComponents();  
    DoConnect();  
}
```

-
- Your code window will then look like this:

```
package Employees;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSet;
import javax.swing.JOptionPane;

public class Workers extends javax.swing.JFrame {

    Connection con=null;
    Statement stmt=null;
    ResultSet rs=null;

    public Workers() {
        initComponents();
        DoConnect();
    }

    public void DoConnect() {
```

- When the form loads, our **DoConnect** method will be called.
- We can add code here to connect to the database, and display the first record in the text fields.

```
public void DoConnect( ) {
    try{
        String uName = "root";
        String uPass = "root";
        String host = "jdbc:mysql://localhost:3306/Employees";
        con = DriverManager.getConnection( host, uName, uPass );
        stmt =con.createStatement( ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
        String SQL = "SELECT * FROM Workers";
        rs = stmt.executeQuery( SQL );
        //while()
        rs.next( );
        {
            int id_col = rs.getInt("ID");
            String id_string=Integer.toString(id_col);
            String first_name = rs.getString("First_Name");
            String last_name = rs.getString("Last_Name");
            String job = rs.getString("Job_Title");
            jTextField1.setText(id_string);
            jTextField2.setText(first_name);
            jTextField3.setText(last_name);
            jTextField4.setText(job);
        }
    }
    catch(SQLException err){
        JOptionPane.showMessageDialog(Workers.this,err.getMessage());
    }
}
```

Run,...

The image shows a screenshot of a Windows-style application window titled "DataBase Form". The window has a blue title bar with standard minimize, maximize, and close buttons. The main content area is light gray and contains four text input fields. The first field contains the number "4". The second and third fields are side-by-side and contain the names "Ibrahim" and "Gamal" respectively. The fourth field is labeled "JobTitle" and contains the text "Programmer".

4	Ibrahim	Gamal
JobTitle	Programmer	

-
- **Database Scrolling Buttons**
 - What we'll do now is to add four buttons to the form.
 - The buttons will enable us to move forward through the records, move back, move to the last record, and move to the first record.
 - add a new panel to your form. Enlarge it and then add four buttons to the panel.
 - Change the variable names of the buttons to the following:
btnNext, btnPrevious, btnLast, btnFirst
 - Change the text on each button to **Next, Previous, Last, First.**

-
- Form now look like that

The image shows a web form with a light beige background and a thin blue border. At the top, there are three small, empty rectangular input fields arranged horizontally. Below these, on the left, is the text 'Job Title:' followed by a single, wide, empty rectangular input field. At the bottom of the form, there are four buttons arranged horizontally, each with a light beige background and a thin blue border. The buttons are labeled 'First', 'Previous', 'Next', and 'Last' from left to right.

-
- You need to do two things with the Next button:
 - first, check if there is a next record to move to;
 - and second, if there is a next record, display it in the Text Fields.

```
try {  
if ( rs.next( ) ) {  
}  
else {  
rs.previous( );  
JOptionPane.showMessageDialog(Workers.this, "End of File");  
}  
}
```

```
catch (SQLException err) {  
    JOptionPane.showMessageDialog(Workers.this,  
        err.getMessage());  
}
```

- In the curly brackets for the IF Statement we can add the code to display the record in the Text Fields

```
private void btnNextActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
  
    try {  
  
        if ( rs.next( ) )  
        {  
            int id_col = rs.getInt("ID");  
            String id = Integer.toString(id_col);  
            String first = rs.getString("First_Name");  
            String last = rs.getString("Last_Name");  
            String job = rs.getString("Job_Title");  
            jTextField1.setText(id);  
            jTextField2.setText(first);  
            jTextField3.setText(last);  
            jTextField4.setText(job);  
        }  
        else {  
            rs.previous( );  
            JOptionPane.showMessageDialog(Workers.this, "End of File");  
        }  
    }  
    catch (SQLException err) {  
        JOptionPane.showMessageDialog(Workers.this, err.getMessage());  
    }  
}
```

Move Backward through the Database

- The code for the Previous button is similar to the **Next** button.
- But instead of using **rs.Next**, you use **rs.Previous**.
- Return to the Design window and double click your **Previous button** to create a code stub.

```
private void btnPreviousActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    try {  
  
        if ( rs.previous( ) )  
        {  
            int id_col = rs.getInt("ID");  
            String id = Integer.toString(id_col);  
            String first = rs.getString("First_Name");  
            String last = rs.getString("Last_Name");  
            String job = rs.getString("Job_Title");  
            jTextField1.setText(id);  
            jTextField2.setText(first);  
            jTextField3.setText(last);  
            jTextField4.setText(job);  
        }  
        else {  
            rs.next( );  
            JOptionPane.showMessageDialog(Workers.this, "End of File");  
        }  
    }  
    catch (SQLException err) {  
        JOptionPane.showMessageDialog(Workers.this, err.getMessage());  
    }  
}
```

Moving to the First and Last Records

- Moving to the first and last records of your database is a lot easier.

```
private void btnLastActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    try {  
        rs.last();  
        int id_col = rs.getInt("ID");  
        String id = Integer.toString(id_col);  
        String first = rs.getString("First_Name");  
        String last = rs.getString("Last_Name");  
        String job = rs.getString("Job_Title");  
        jTextField1.setText(id);  
        jTextField2.setText(first);  
        jTextField3.setText(last);  
        jTextField4.setText(job);  
    }  
    catch (SQLException err) {  
        JOptionPane.showMessageDialog(Workers.this, err.getMessage());  
    }  
}
```

-
- For moving to the first record

```
private void btnFirstActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    try {  
        rs.first();  
        int id_col = rs.getInt("ID");  
        String id = Integer.toString(id_col);  
        String first = rs.getString("First_Name");  
        String last = rs.getString("Last_Name");  
        String job = rs.getString("Job_Title");  
        jTextField1.setText(id);  
        jTextField2.setText(first);  
        jTextField3.setText(last);  
        jTextField4.setText(job);  
    }  
    catch (SQLException err) {  
        JOptionPane.showMessageDialog(Workers.this, err.getMessage());  
    }  
}
```

Updating a Record

- The ResultSet has Update methods that allow you to update records not only in the ResultSet itself, but in the underlying database.
- Now add a new panel to the form. Add a new button to the panel. Change the default variable name to **btnUpdateRecord**.
- Change the text on the button to **Update Record**.

-

The image shows a web form with a light beige background and a blue border. At the top, there are three empty input fields. Below them is a label "Job Title:" followed by a single wide input field. The bottom section contains several buttons: "First", "Previous", "Next", and "Last" in a row; "Update Record", "Delete Record", and "New Record" in a second row; and "Save New Record" and "Cancel New Record" in a third row.

-
- Add the following buttons

Button Variable Name: btnNewRecord

Button Text: New Record

Button Variable Name: btnDeleteRecord

Button Text: Delete Record

Button Variable Name: btnSaveRecord

Button Text: Save New Record

-
- Double click your Update button to create a code stub.
 - The first thing to do is get the text from the Text Fields:

```
String first = textFirstName.getText( );
```

```
String last = textLastName.getText( );
```

```
String job = textJobTitle.getText( );
```

```
String ID = textID.getText( );
```

- If we want to update an ID field, however, we need to convert the String to an Integer:

```
int newID = Integer.parseInt( ID );
```

-
- Now that we have all the data from the Text Fields, we can call the relevant update methods of the ResultSet object:

rs.updateString("First_Name", first);

- The one above uses **updateString**.
- But you need the field type from your database table here.
- We have three strings (First_Name, Last_Name, Job_Title) and one integer value (ID).
- So we need three **updateString** methods and one **updateInt**.
- In between the round brackets of the update methods, you need the name of a column from your database (though this can be its Index value instead).
- After a comma you type the replacement data.

-
- The update methods just update the ResultSet, however. To commit the changes to the database, you issue an **updateRow** command:

rs.updateRow();

- You need try catch statement, just in case something goes wrong.

- code

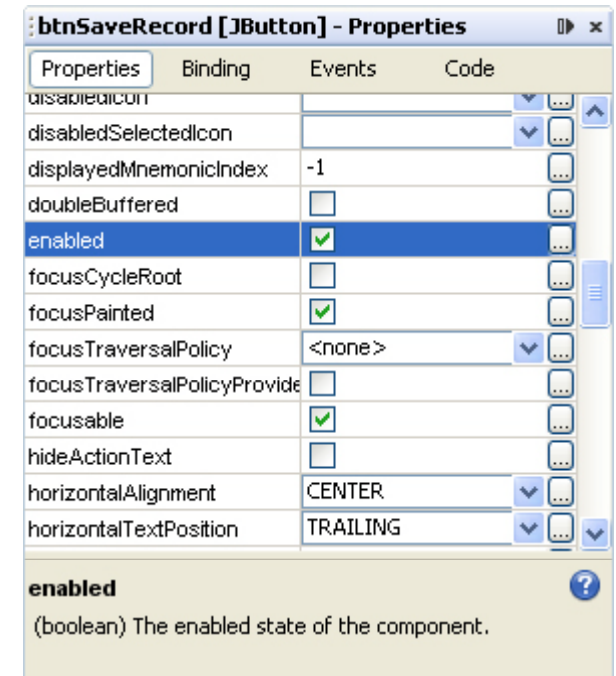
```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    String first = jTextField2.getText( );  
    String last = jTextField3.getText( );  
    String job = jTextField4.getText( );  
    String ID = jTextField1.getText( );  
    int newID = Integer.parseInt( ID );  
    try {  
        rs.updateInt( "ID", newID );  
        rs.updateString( "First_Name", first );  
        rs.updateString( "last_Name", last );  
        rs.updateString( "Job_Title", job );  
        rs.updateRow( );  
        JOptionPane.showMessageDialog(Workers.this, "Updated");  
    }  
    catch (SQLException err) {  
        System.out.println(err.getMessage( ));  
    }  
}
```

-
- Run your program and try it out. Change some data in a Text Field
 - Then click your Update button.
 - Scroll past the record then go back.
 - The change should still be there. Now close down your program and run it again.
 - You should find that the changes are permanent.

Adding a New Record

- We have three buttons that refer to new records: **New Record**, **Save New Record**, and **Cancel New Record**
- The New Record button will only clear the Text Fields,
- and ready them for new data to be entered.
- We can also disable some other buttons, including the **New Record button**.
- Another thing we can do is to make a note of which record is currently loaded.
- If a user changes his or her mind, we can enable all the buttons again by clicking the **Cancel button**.
- Clicking the Save **New Record button** will do the real work of saving the data to the database.

- Click on your Save New Record button to select it.
- In the Properties area on the right, locate the Enabled property:
- Uncheck the box to the right of **enabled**. The **Save New Record** will be **disabled**.
- Do the same for the **Cancel New Record** button.
- The **Cancel New Record** will be **disabled**.
- When your form loads, it will look like this:



- Even if you had code for these two buttons, nothing would happen if you clicked on either of them.
- When the New Record button is clicked, we can disable the following buttons:
 - **First** **Previous**
 - **Next** **Last**
 - **Update Record**
 - **Delete Record**
 - **New Record**



The screenshot shows a Java Swing window with a blue title bar and standard window controls (minimize, maximize, close). The window contains a form with the following elements:

- Three text input fields at the top: the first contains '1', the second contains 'Helen', and the third contains 'James'.
- A label 'Job Title:' followed by a text input field containing 'IT Manager'.
- A row of four buttons: 'First', 'Previous', 'Next', and 'Last'.
- A row of three buttons: 'Update Record', 'Delete Record', and 'New Record'.
- A row of two buttons: 'Save New Record' and 'Cancel New Record'.

-
- The **Save** and **Cancel** buttons, however, can be enabled.
 - If the user clicks **Cancel**, we can switch the buttons back on again.
 - Double click your **New Record** button to create a code stub.
 - Add the following lines of code:

```
private void btnNewRecordActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    btnFirst.setEnabled( false );  
    btnPrevious.setEnabled( false) ;  
    btnNext.setEnabled( false );  
    btnLast.setEnabled( false );  
    btnUpdateRecord.setEnabled( false );  
    btnDelete.setEnabled( false );  
    btnNewRecord.setEnabled( false );  
    btnSaveRecord.setEnabled( true );  
    btnCancelNewRecord.setEnabled( true );  
}
```

-
- So seven of the buttons get turned off using the **setEnabled** property.
 - Two of the buttons get turned on.
 - We can do the reverse for the **Cancel** button.
 - Switch back to Design view.
 - Double click your **Cancel New Record** button to create a code stub.
 - Add the following:

```
private void btnCancelNewRecordActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    btnFirst.setEnabled( true );  
    btnPrevious.setEnabled( true ) ;  
    btnNext.setEnabled( true );  
    btnLast.setEnabled( true );  
    btnUpdateRecord.setEnabled( true );  
    btnDelete.setEnabled( true );  
    btnNewRecord.setEnabled( true );  
    btnSaveRecord.setEnabled( false );  
    btnCancelNewRecord.setEnabled( false );  
}
```


-
- Now run your program and test it out.
 - Click the New Record button and the form will look like this:



The screenshot shows a Windows application window with a blue title bar and standard window controls (minimize, maximize, close). The window contains a data entry form with the following elements:

- Three text input fields at the top: the first contains '1', the second contains 'Helen', and the third contains 'James'.
- A label 'Job Title:' followed by a text input field containing 'IT Manager'.
- A row of four buttons: 'First', 'Previous', 'Next', and 'Last'.
- A row of three buttons: 'Update Record', 'Delete Record', and 'New Record'.
- A row of two buttons at the bottom: 'Save New Record' and 'Cancel New Record'.

-
- Click the Cancel New Record button and the form will look like this:



The screenshot shows a window with a blue title bar and standard Windows window controls (minimize, maximize, close). The form contains the following elements:

- Three input fields at the top: the first contains '1', the second contains 'Helen', and the third contains 'James'.
- A label 'Job Title:' followed by a single input field containing 'IT Manager'.
- A row of four buttons: 'First', 'Previous', 'Next', and 'Last'.
- A row of three buttons: 'Update Record', 'Delete Record', and 'New Record'.
- A row of two buttons: 'Save New Record' and 'Cancel New Record'.

-
- Another thing we need to do is to record which row is currently loaded.
 - In other words, which row number is currently loaded in the Text Fields.
 - We need to do this because the Text Fields are going to be cleared.
 - If the Cancel button is clicked, then we can reload the data that was erased.
 - Add the following Integer variable to the top of your code, just below your Connection, Statement, and ResultSet lines:

int curRow = 0;

```
public class Workers extends javax.swing.JFrame {

    Connection con;
    Statement stmt;
    ResultSet rs;
    int curRow = 0;

    public Workers() {
        initComponents();
        DoConnect();
    }
}
```

- Now go back to your New Record code.
- To get which row the Cursor is currently pointing to there is a method called **getRow**.
- This allows you to store the row number that the Cursor is currently on:

```
curRow = rs.getRow( );
```

-
- We'll use this row number in the Cancel New Record code.
 - The only other thing we need to do for the New Record button is to clear the Text Fields. This is quite simple:

```
textFirstName.setText("");
```

```
textLastName.setText("");
```

```
textJobTitle.setText("");
```

```
textID.setText("");
```

- Because we've used a method of the **ResultSet**, we need to wrap everything up in a **try ... catch block**.

```
private void btnNewRecordActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    try{  
        curRow = rs.getRow();  
        btnFirst.setEnabled( false );  
        btnPrevious.setEnabled( false) ;  
        btnNext.setEnabled( false );  
        btnLast.setEnabled( false );  
        btnUpdateRecord.setEnabled( false );  
        btnDelete.setEnabled( false );  
        btnNewRecord.setEnabled( false );  
        btnSaveRecord.setEnabled( true );  
        btnCancelNewRecord.setEnabled( true );  
        jTextField1.setText( "" );  
        jTextField2.setText( "" );  
        jTextField3.setText( "" );  
        jTextField4.setText( "" );  
  
    }  
    catch(SQLException err){}
```

-
- For the Cancel button, we need to get the row that was previously loaded and put the data back in the Text Fields.
 - To move the Cursor back to the row it was previously pointing to, we can use the **absolute method**:

rs.absolute(curRow);

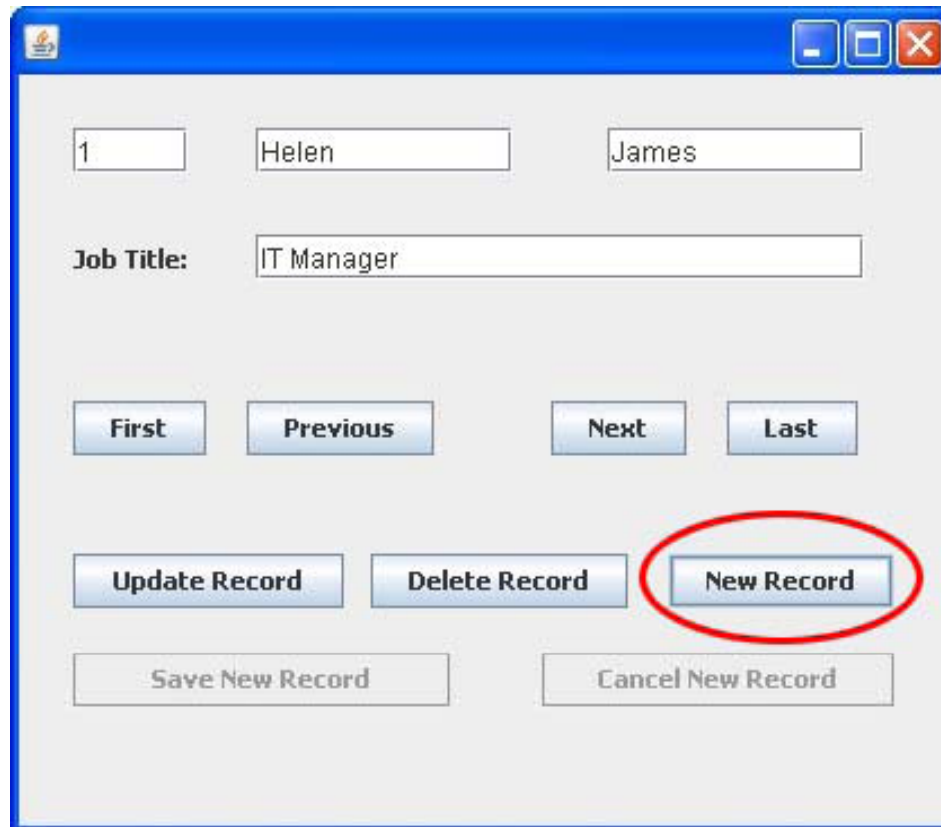
- The absolute method moves the Cursor to a fixed position in the ResultSet.
- We want to move it the value that we stored in the variable **curRow**.
- Now that Cursor is pointing at the correct row, we can load the data into the Text Fields:

```
textFirstName.setText( rs.getString("First_Name") );  
textLastName.setText( rs.getString("Last_Name") );  
    textJobTitle.setText( rs.getString("Job_Title") );  
textID.setText( Integer.toString( rs.getInt("ID" ) ) );
```

- code

```
private void btnCancelNewRecordActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    try{  
        rs.absolute( curRow );  
        btnFirst.setEnabled( true );  
        btnPrevious.setEnabled( true ) ;  
        btnNext.setEnabled( true );  
        btnLast.setEnabled( true );  
        btnUpdateRecord.setEnabled( true );  
        btnDelete.setEnabled( true );  
        btnNewRecord.setEnabled( true );  
        btnSaveRecord.setEnabled( false );  
        btnCancelNewRecord.setEnabled( false );  
        jTextField1.setText( Integer.toString(rs.getInt("ID")) );  
        jTextField2.setText(rs.getString("First_Name"));  
        jTextField3.setText(rs.getString("Last_Name"));  
        jTextField4.setText(rs.getString("Job_Title"));  
    }  
    catch(SQLException err){}
```

-
- When you've finished adding the code for the **New and Cancel buttons**, run your program and try it out.
 - Before clicking the **New Record** button, the form will look like this



The screenshot shows a Windows-style application window with a blue title bar. The window contains a data entry form with the following elements:

- Three text input fields at the top: the first contains '1', the second contains 'Helen', and the third contains 'James'.
- A label 'Job Title:' followed by a text input field containing 'IT Manager'.
- A row of four buttons: 'First', 'Previous', 'Next', and 'Last'.
- A row of three buttons: 'Update Record', 'Delete Record', and 'New Record'. The 'New Record' button is circled in red.
- A row of two buttons: 'Save New Record' and 'Cancel New Record'.

-
- Click the New Record button to see the Text Fields cleared:



A screenshot of a software window with a blue title bar and standard Windows window controls (minimize, maximize, close). The window contains a form with the following elements:

- Three empty text input fields arranged horizontally at the top.
- A label "Job Title:" followed by a single wide text input field.
- Four buttons labeled "First", "Previous", "Next", and "Last" arranged horizontally.
- Three buttons labeled "Update Record", "Delete Record", and "New Record" arranged horizontally.
- Two buttons labeled "Save New Record" and "Cancel New Record" arranged horizontally at the bottom.

-
- Clicking the Cancel button will reload the data:

A screenshot of a data entry form window. The window has a blue title bar with standard Windows window controls (minimize, maximize, close). The form contains the following elements:

- Three input fields in the top row: the first contains the number '1', the second contains the name 'Helen', and the third contains the name 'James'.
- A label 'Job Title:' followed by a single input field containing the text 'IT Manager'.
- A row of four buttons: 'First', 'Previous', 'Next', and 'Last'.
- A row of three buttons: 'Update Record', 'Delete Record', and 'New Record'.
- A row of two buttons: 'Save New Record' and 'Cancel New Record'.

Saving a New Record

- Before you can save a new record, you have to move the Cursor to something called the **Insert Row**. This creates a blank record in the **ResultSet**.
- You then add the data to the **ResultSet**:
- After adding the data to the **ResultSet**, the final line inserts a new row.
- However, to commit any changes to the database what we'll do is to close our Statement object and our **ResultSet** object. We can then reload everything.
- If we don't do this, there's a danger that the new record won't get added, either to the **ResultSet** or the database. (This is due to the type of Driver we've used.)

-
- The code to reload everything is the same as the code you wrote when the form first loads
 - You're not doing anything different, here: just selecting all the records again and putting the first one in the Text Fields.

- Code

```
private void btnSaveRecordActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    try{  
        rs.moveToInsertRow( );  
  
        rs.updateInt("ID", Integer.parseInt(jTextField1.getText()));  
        rs.updateString("First_Name", jTextField2.getText());  
        rs.updateString("Last_Name", jTextField3.getText());  
        rs.updateString("Job_Title", jTextField4.getText());  
        rs.insertRow( );  
  
        stmt.close( );  
        rs.close( );  
  
        stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,  
        ResultSet.CONCUR_UPDATABLE);  
        String sql = "SELECT * FROM Workers";  
        rs = stmt.executeQuery(sql);  
        rs.next( );  
        int id_col = rs.getInt("ID");  
        String id = Integer.toString(id_col);  
        String first2 = rs.getString("First_Name");  
        String last2 = rs.getString("Last_Name");  
        String job2 = rs.getString("Job_Title");  
        jTextField1.setText(id);  
        jTextField2.setText(first2);  
        jTextField3.setText(last2);  
        jTextField4.setText(job2);  
    }  
    catch(SQLException err){}
```

Deleting a Record

- Deleting a row can be straightforward: Just use **deleteRow** method of the ResultSet:

rs.deleteRow();

- the **Driver** we are using, the **ClientDriver**, leaves a blank row in place of the data that was deleted.
- If you try to move to that row using your **Next** or **Previous** buttons, the **ID Text Field** will have a 0 in it, and all the others will be blank.

-
- To solve this problem we'll first delete a row then, again, close the **Statement** object and the **ResultSet** objects.
 - We can then reload all the data in the **Text Fields**.
 - That way, we won't have any blank rows.

```
private void btnDeleteActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    try{  
        rs.deleteRow( );  
        stmt.close( );  
        rs.close( );  
  
        stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,  
        ResultSet.CONCUR_UPDATABLE);  
        String sql = "SELECT * FROM Workers";  
        rs = stmt.executeQuery(sql);  
        rs.next( );  
        int id_col = rs.getInt("ID");  
        String id = Integer.toString(id_col);  
        String first2 = rs.getString("First_Name");  
        String last2 = rs.getString("Last_Name");  
        String job2 = rs.getString("Job_Title");  
        jTextField1.setText(id);  
        jTextField2.setText(first2);  
        jTextField3.setText(last2);  
        jTextField4.setText(job2);  
    }  
    catch(SQLException err){}
```



Thanks.

