MIS 768: Advanced Software Concepts

Spring 2024

Database Applications (2)

Purpose

- Use scrollable ResultSet in applications
- Create database applications in JavaFX
- Practice Data Access Object Pattern

1. Preparation

(1) Please make sure the **MySQL** server is up and running.

(2) Launch Eclipse. Create a new package to hold our source file. Name the package as **edu.unlv.mis768.labwork18.**

(3) Download **18_lab_files.zip** from WebCampus. Extract the zip file and then import the .java files into the package.

2. Database ResultSet

(4) Please open **CoffeeBrowser.fxml** in **SceneBuilder** to see the layout of the application.
It will display the content of a **ResultSet** (i.e., the database query result) one record at a time. The four buttons can be used to switch records.

(5) Open **CoffeeBrowser.java** in the editor. In this class, the two fields are a **Connection** object and a **ResultSet** object. The **getDBConnection**() will establish the database connection and execute the query to generate a result set.

(6) Thus, in the **start()** method, the **getDBConnection()** method will be called. Then the **ResultSet** object will be passed to the controller class.

```
20    @Override
21    public void start(Stage primaryStage) throws Exception {
22
23        // Call the method to execute the query and generate the ResultSet
24        getDBConnection();
25
26        // the FXMLLoader object to load the UI
27        FXMLLoader loader = new FXMLLoader();
28        //specify the file location
29        loader.setLocation(getClass().getResource("CoffeeBrowser.fxml"));
30        // load the UI
31        Parent parent = loader.load();
32
33        // access the controller via the loader
34        // in the FXML file, the controller class is specified
35        CoffeeBrowserController controller = loader.getController();
36
37        // call the method in the controller class
38        // Pass the ResultSet object to the Scene
39        controller.initData(result);
40        // set the title of the window
41        primaryStage.setTitle("Coffee Information");
42
43        // set the scene
44        Scene scene = new Scene(parent);
45        // set the scene for the stage
46        primaryStage.setScene(scene);
47
48        // show the stage
49        primaryStage.show();
50    }
```

(7) The **stop**() method will be called when the application ends. We thus should close the database connection here.

```
84    /**
85     * Override the default stop() method to close the database connection
86     */
87    @Override
88    public void stop(){
89        try {
90            conn.close();
91        } catch (SQLException e) {
92            System.out.println("ERROR: "+e.getMessage());
93        }
94    }
```

(8) Please open **CoffeeBrowserController.java**. The **initData**() method will be called by the application when this scene is loaded.

It gets the **ResultSet** object and assigns it to the variable.

Also, it displays the total numbers of record in this **ResultSet** object.

Then, it displays the first record.

```
41    /**
42     * This method accepts a ResultSet object to be displayed.
43     * @param The ResultSet generated ordered in the previous screen
44     * @throws SQLException
45     */
46    public void initData(ResultSet re) throws SQLException {
47        // assign the passed ResultSet object to the result variable in this class
48        result = re;
49
50        try {
51            // Display the number of rows
52            result.last();                  // Move to the last row
53            int numRows = result.getRow();  // Get the current row number
54            result.first();                 // Move back to the first row
55            totalRowLabel.setText("Total Rows: "+numRows);
56
57            //Update the labels to show the content of the current row; i.e., the first row
58            showRowContent(result);
59        }
60        catch (Exception ex){
61            System.out.println("ERROR at initData(): " + ex.getMessage());
62        }
63    }
```

(9) The **showRowContent()** is used for displaying the content of a row in the **ResultSet** object.

We use the **getString()** or **getDouble()** method to retrieve the data and set the value to the text of the label.

```
65    /**
66     * This function displays the content of the current row in a result set.
67     * Three labels are used to display the content of the ProdNum, Description, and price columns
68     * Another label is used to display the row number
69     * @param result A ResultSet
70     */
71    public void showRowContent(ResultSet result) {
72        try {
73            //Display the content of the current row
74            numLabel.setText(result.getString("ProdNum"));
75            descLabel.setText(result.getString("Description"));
76            priceLabel.setText(Double.toString(result.getDouble("Price")));
77
78            //Display the current row number
79            currentRowLabel.setText("Current Row: "+result.getRow());
80
81        } catch(Exception ex){
82            System.out.println("ERROR: " + ex.getMessage());
83        }
84    }
```

(10) The **firstButtonListener**() method move the cursor to the first record and display the content of the record by calling the **showRowContent()** method.

```
85      /**
86       * ActionListener for the firstButton. Move the record cursor to the first record
87       */
88      public void firstButtonListener()  {
89          try{
90              //Move to the first row
91              result.first();
92
93              //Update the labels to show the content of the current row
94              showRowContent(result);
95
96          } catch (Exception ex){
97              System.out.println("ERROR: " + ex.getMessage());
98          }
99      }
```

(11) Following this logic, we can now implement **lastButtonListener**(), **previousButtonListener**(), and **nextButtonListener**() method.

(12) Please run **CoffeeBrowser.java** to test the program.

3. Data Access Object Pattern

(13) Open **Cutomer.java**.

There are fields corresponding to the columns in the **Customer** table in **CoffeeDB**. The constructor and getter/setters are defined and implemented.

(14) Open **CustomerDAO.java**

This interface defines the required data access operations, including data retrieval, insertion, update, and deletion. In addition to getting all the customers, the class also defines a method to query customer by state.

(15) Open the partially completed **CutomerDAOImpl.java**

This class implements the **CustomerDAO** interface. In this class, the **getAllCustomers**() method is not complete. Please complete the method:

```
14      public List<Customer> getAllCustomers() {
15          // Create a array list for the data.
16          List<Customer> customerList =  new ArrayList<Customer>();
17
18          try {
19              Connection conn = CoffeeDBUtil.getDBConnection();
20              Statement stmt = conn.createStatement(
21                      ResultSet.TYPE_SCROLL_INSENSITIVE,
22                      ResultSet.CONCUR_READ_ONLY);
23
24
25              String sql = "SELECT * from " + CoffeeDBConstants.CUSTOMER_TABLE_NAME;
26              //Execute the query.
27              ResultSet result = stmt.executeQuery(sql);
28
29
30
31              //Get the number of rows.
32              result.last();                  // Move to last row
33              int numRows = result.getRow(); // Get row number
34              result.first();                 // Move to first row
35
36              for (int row = 0; row < numRows; row++) {
37                  // create a new object and fill the field with the values from the result set.
38                  Customer aCustomer = new Customer(result.getString("CustomerNumber"),
39                                                    result.getString("Name"),
40                                                    result.getString("Address"),
41                                                    result.getString("City"),
42                                                    result.getString("State"),
43                                                    result.getString("Zip"));
44
45                  //Add the object to the list
46                  customerList.add(aCustomer);
47
48
49                 // Go to the next row in the ResultSet.
50                 result.next();
51              }
52              // close the database connection
```

(16) Please also implement **getCustomersByState()**

(17) Other methods this DAO provides include **insertCustomer().**

We can use it in applications.

4. DOA Application (1)

(18) Please open **CustomerInserter.fxml** in **SceneBuiler**.

It allows the user to enter the data and click the Save button to save the data.

(19) Please open **CustomerInserterController.java.**

In **saveButtonListener()** method, when the user enters the customer number, the application should create a customer object, and then use it and the DAO object to insert the record. Please complete the code.

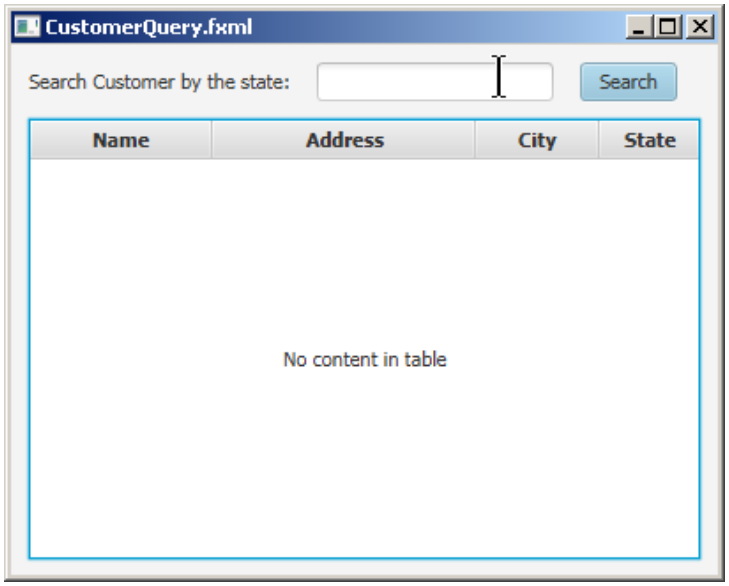
```
56          // if the customer number is good
57          else {
58              String name=nameTextField.getText();
59              String address=addreessTextField.getText();
60              String city=cityTextField.getText();
61              String state=stateTextField.getText();
62              String zip=zipTextField.getText();
63
64              // Create a customer object, and fill the fields with the values
65              Customer someone = new Customer(num,name,address,city,state,zip);
66
67              // Create a DAO object
68              CustomerDAO cDao = new CustomerDAOImpl();
69
70              // use the DAO object, insertCustomer() method to insert the record
71              if (cDao.insertCustomer(someone)) {
72                  // Message to the user, showing the end of execution
73                  // use an Alert object to show an error message.
74                  Alert confirm = new Alert(AlertType.INFORMATION);
75                  confirm.setTitle("Customer Inserter");
76                  confirm.setContentText("The record was inserted.");
77
78                  confirm.showAndWait();
79              }
80              else {
81                  // Message to the user, showing the failure of execution
82                  // use an Alert object to show an error message.
83                  Alert alert = new Alert(AlertType.WARNING);
84                  alert.setTitle("Customer Inserter");
85                  alert.setContentText("The record was not inserted. \\nPlease try again.");
86
87                  alert.showAndWait();
88              }
```

(20) Please run **CustomerInserter.java** to test the program.

5. DOA Application (2)

(21) Please open **CustomerQuery.fxml** in **SceneBuilder.**

It allows the user to enter a state and displays customer in the state.



(22) Please open **CustomerQueryController.java.**

In **queryButtonListern**() method, we need to to instantiate the **CustomerDAOImpl** object and then use its **getCustomersByState**() method to get the list of the customer.

```
36    public void queryButtonListern() {
37        // Instantiate an DAO object
38        CustomerDAOImpl cDAO = new CustomerDAOImpl();
39        // Declare the List for the result data
40        List<Customer> customerList =  new ArrayList<Customer>();
41        // call the getCustomersByState() method of the DAO class to get the result.
42        // use the TextField as the query string
43        customerList =  cDAO.getCustomersByState(stateTextField.getText());
44        // Convert the ArrayList into an ObservableList
```

(23) However, we need to convert the **ArrayList** to a **ObservableList** before we can display it in the TableView.

```
36    public void queryButtonListern() {
37        // Instantiate an DAO object
38        CustomerDAOImpl cDAO = new CustomerDAOImpl();
39        // Declare the List for the result data
40        List<Customer> customerList =  new ArrayList<Customer>();
41        // call the getCustomersByState() method of the DAO class to get the result.
42        // use the TextField as the query string
43        customerList =  cDAO.getCustomersByState(stateTextField.getText());
44        // Convert the ArrayList into an ObservableList
45        ObservableList<Customer> observableCustomers =
46                FXCollections.observableArrayList(customerList);
47        // Display at the TableView
48        tableView.setItems(observableCustomers);
49
50    }
```

(24) Please run **CustomerQuery.java** to test the program.