# Assignment Module 3 Description

*By Michael Floerchinger*

06/04/2017

Java III Programming

Assignment #3

Project Change Description

Replace the “Model” component of your application with a new model. The new model will make use of JDBC to connect to an

underlying relational database. MySQL is a simple and free choice for your database. You must design a model that supports

the functionality of the previous “Model” component. The new “Model” component will interface with the “View” component in

the same manner as the previous “View” component. In addition, your code will make use of exception handling to deal with

SQLExceptions. Additionally, your View will make use of a JTable to display the results of your queries upon the Model.

Deliverables:

There are two deliverables that must be submitted. These deliverables are as follows:

1. Your source code in a .jar or .zip file format.

2. Captured images of your application in operation.

To execute place both the jar executable driver file into the same directory:

InventoryProgramMainApp.jar

mysql-connector-java-5.1.42-bin.jar

Run the following to launch:

# java -jar InventoryProgramMainApp.jar

# Deliverables

## **Images of Application Execution and Functionality**

### **Overview**

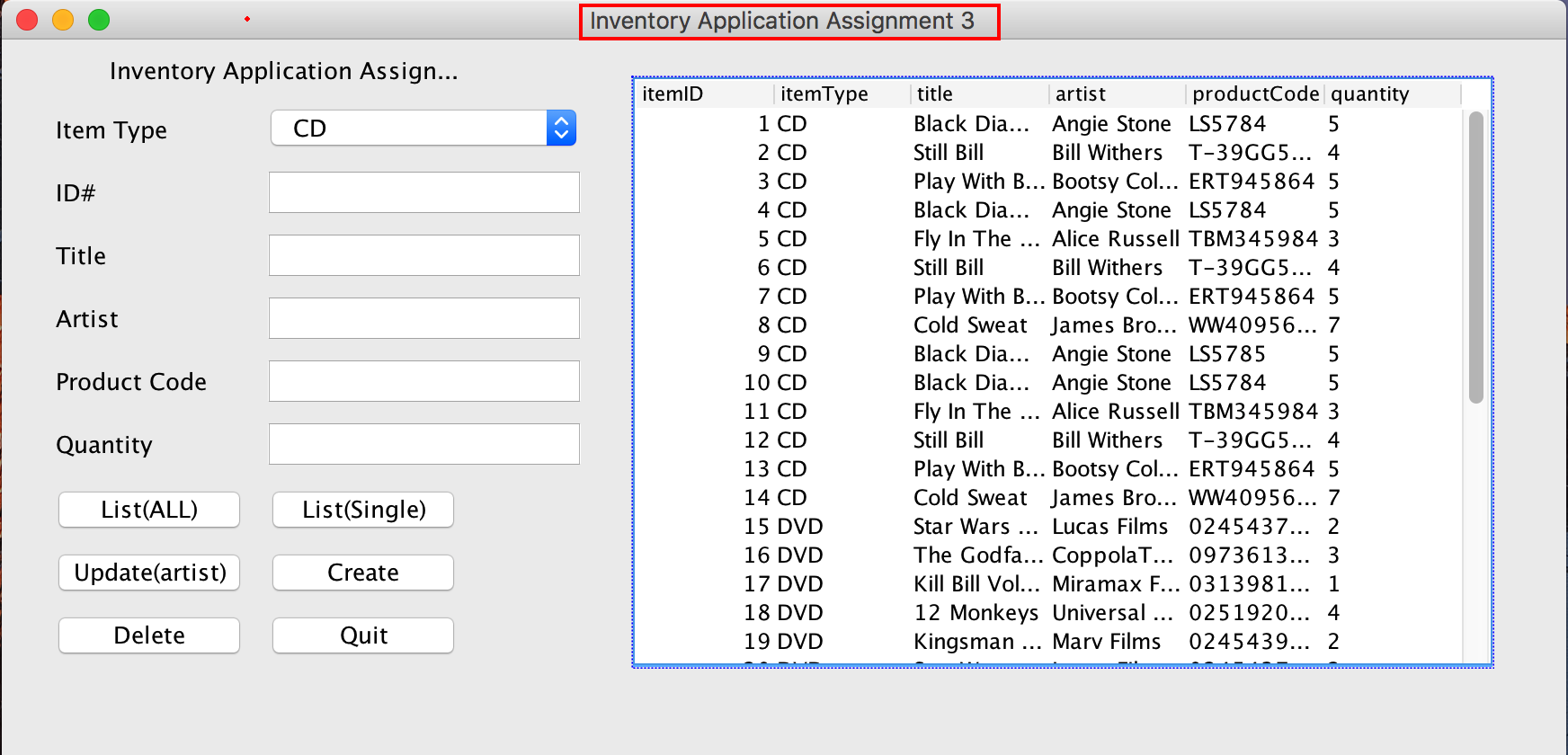
Application Assignment 3 demonstrates the replacement and implementation of a database layer. This aims to replace only the previously implemented properties files with a MySQL database and the relevant JDBC connectors to communicate with the database. All other existing functions previously implemented were to remain the same.

The database implementation is based on the lessons provided in the text book, and uses different features from most of the examples to implement the database layer and JDBC functions within this application. The database extends the *AbstractTableModel* library to provide all the methods for retrieving and arranging the data as necessary for the *ResultSet* and the *ResultSetMetaData* libraries.

The display view uses a *JTable* to display the results of a button initiated search query.

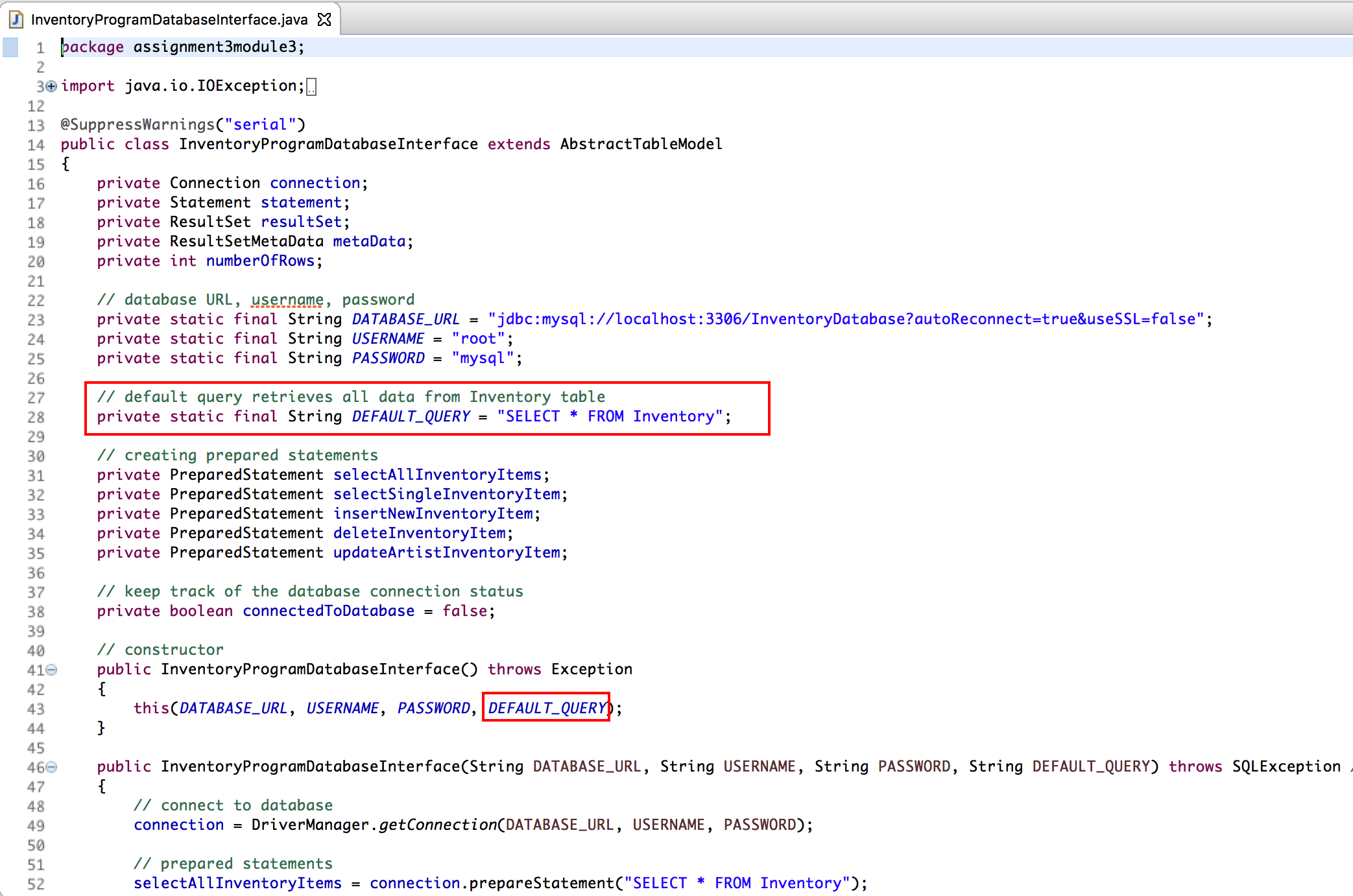
The application also still uses the Observer Pattern implementation, however improvements have been applied to ensure the view is notified of any changed. The implementation required the view to “get” data relevant to its request, similar to a buffet implementation where the diner selects only the items of immediate dining interest.

1. ***Application Startup***



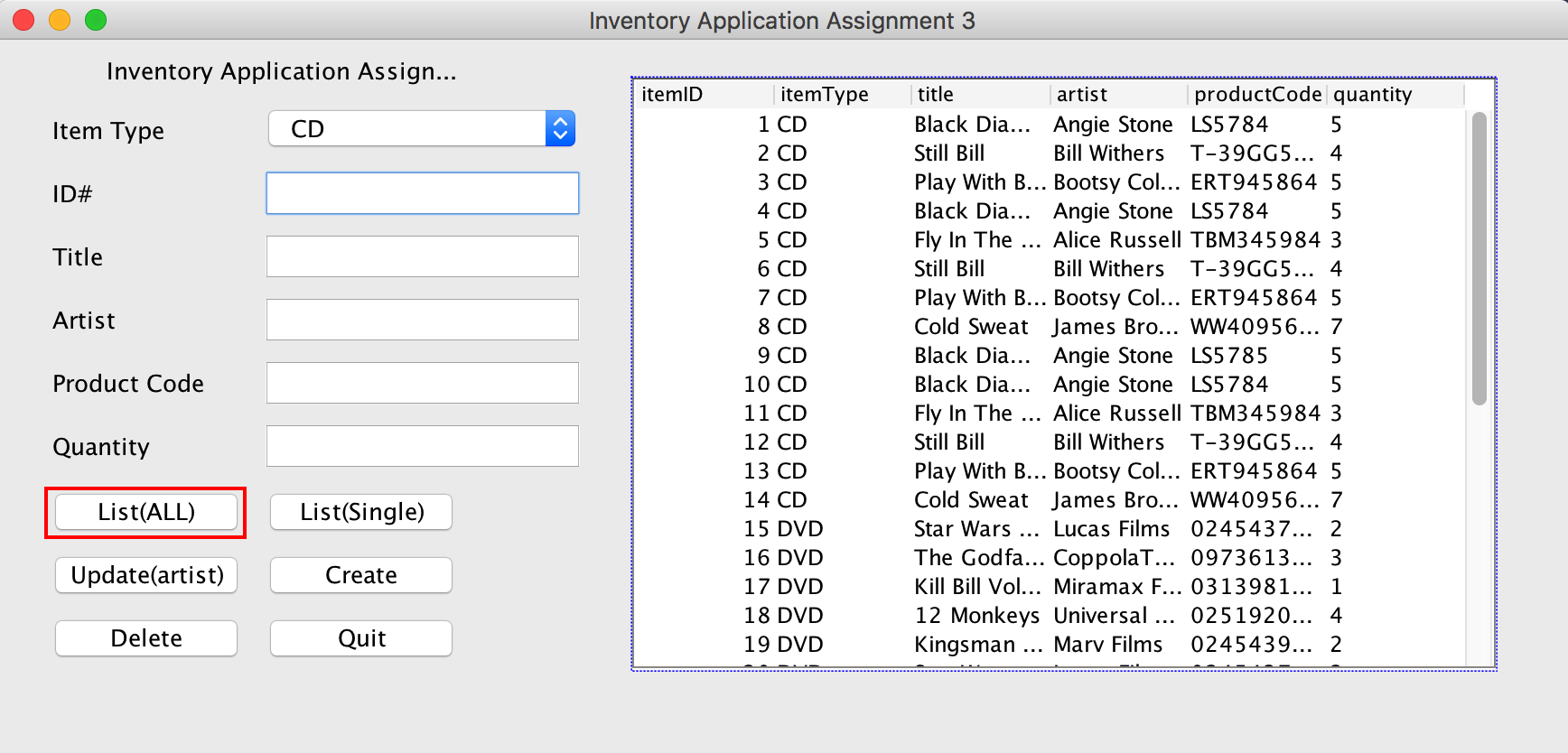
Application Assignment on startup. The application starts to present a GUI representation of the inventory program assignment for module 1 and module 3. The application starts to present the current inventory items. This display is achieved through the database constructor, which includes a show all “select \* from Inventory”.

1. ***Constructor Displaying Initial View***

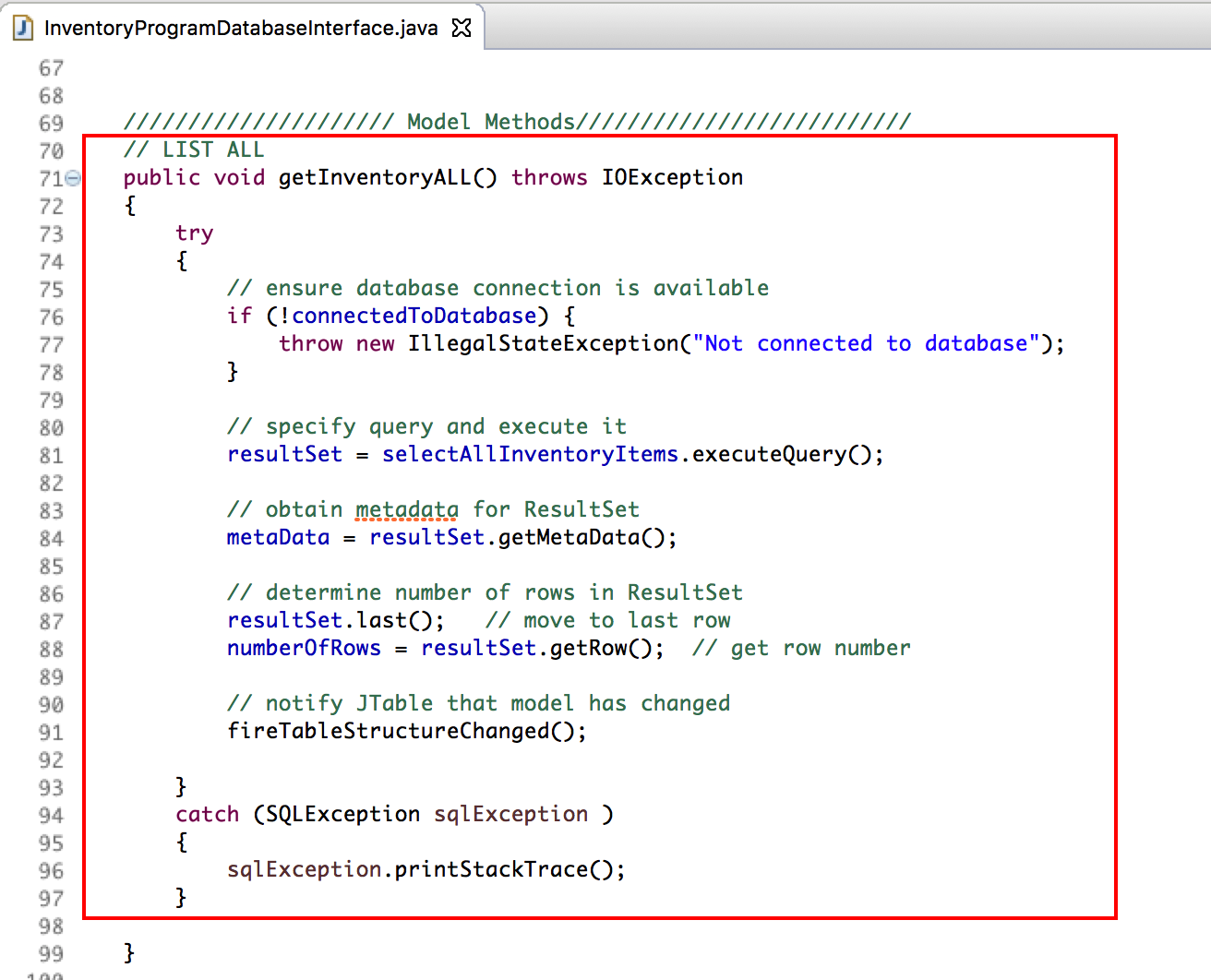


Screenshot shows initial *DEFAULT\_QUERY* on startup. All other database calls, especially those that receive user input were implemented using PreparedStatements

1. ***List ALL Inventory Item***



Pressing the “*List(ALL)*” button executes the button listener within the View component of the MVC architecture. This calls the equivalent method within the “control” layer, which in-turn calls the relevant method in the “model” layer.

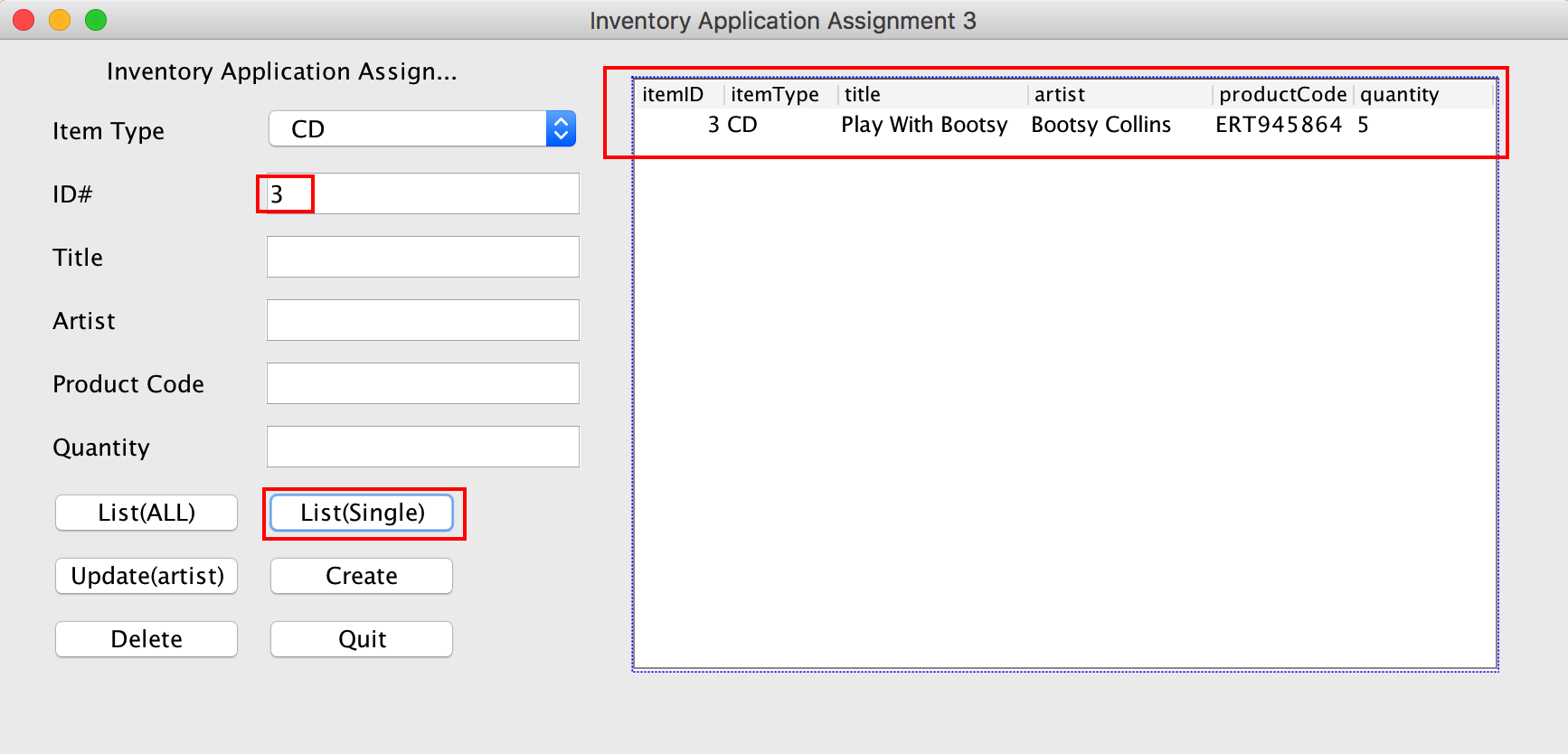


This calls the “*getInventoryAll()*” method on the database layer. The method checks there is a current connection to the database before executing the prepared statement to retrieve all items in the Inventory database.

The data is retrieved and passed through to the resultSet object. This provides methods for preparing and formatting the data for the JTable.

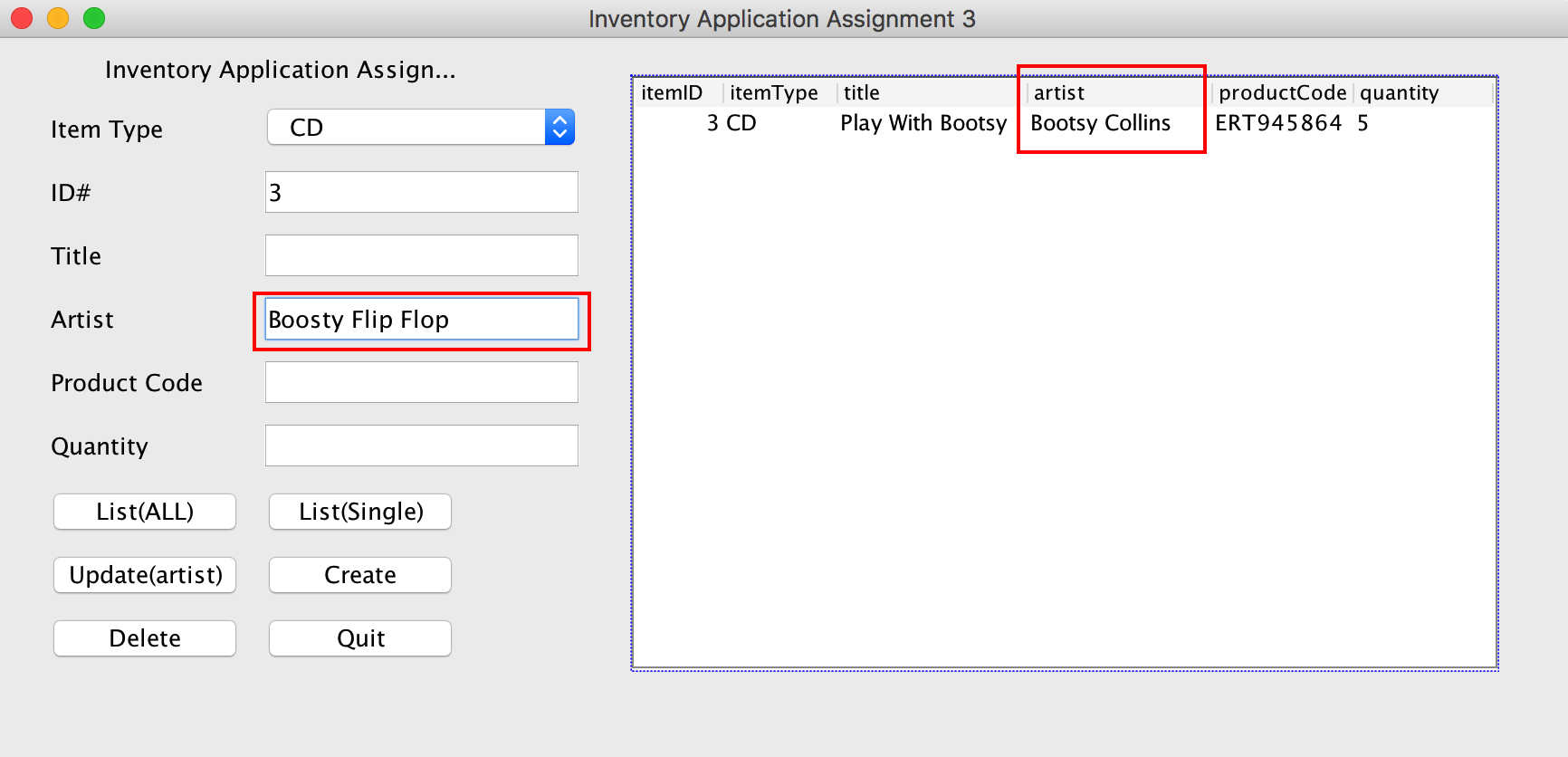
After all the information has been prepared, the method finishes by executing the “fireTableStructureChanged()” method from the AbstractTableModel methods. This in itself is an implementation of the Observer Pattern, however this implementation of the MVC implements a dedicated Observer pattern for notifying the “view” of any data changes.

1. ***List(Single) Inventory Item***

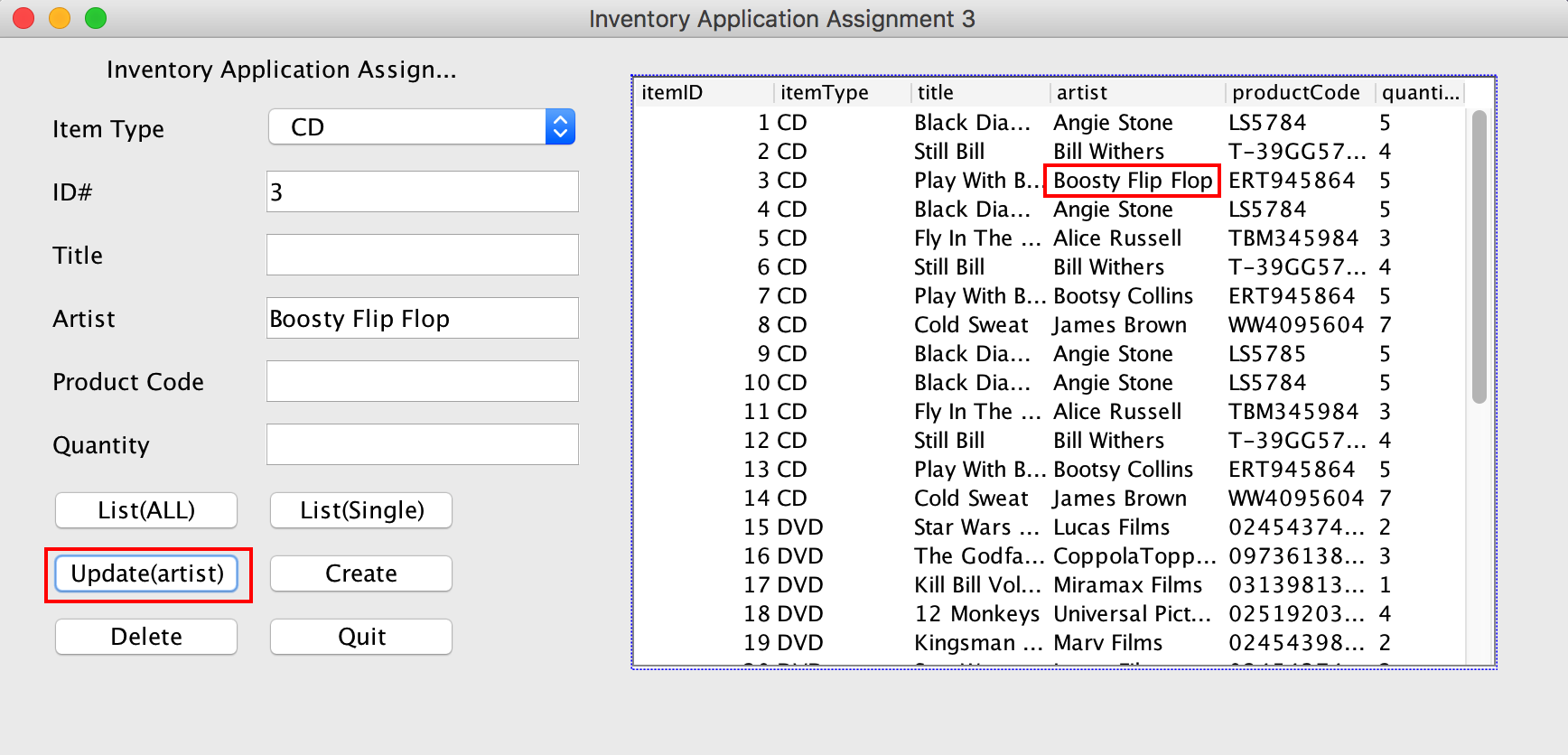
******

Selecting the “List(Single)” button executes the required methods for retrieving only a single element from the database. Input the desired “*itemID*” into the field and click the list single button. This retrieves only a single row item.

1. ***Update Record, (Artist Only)***

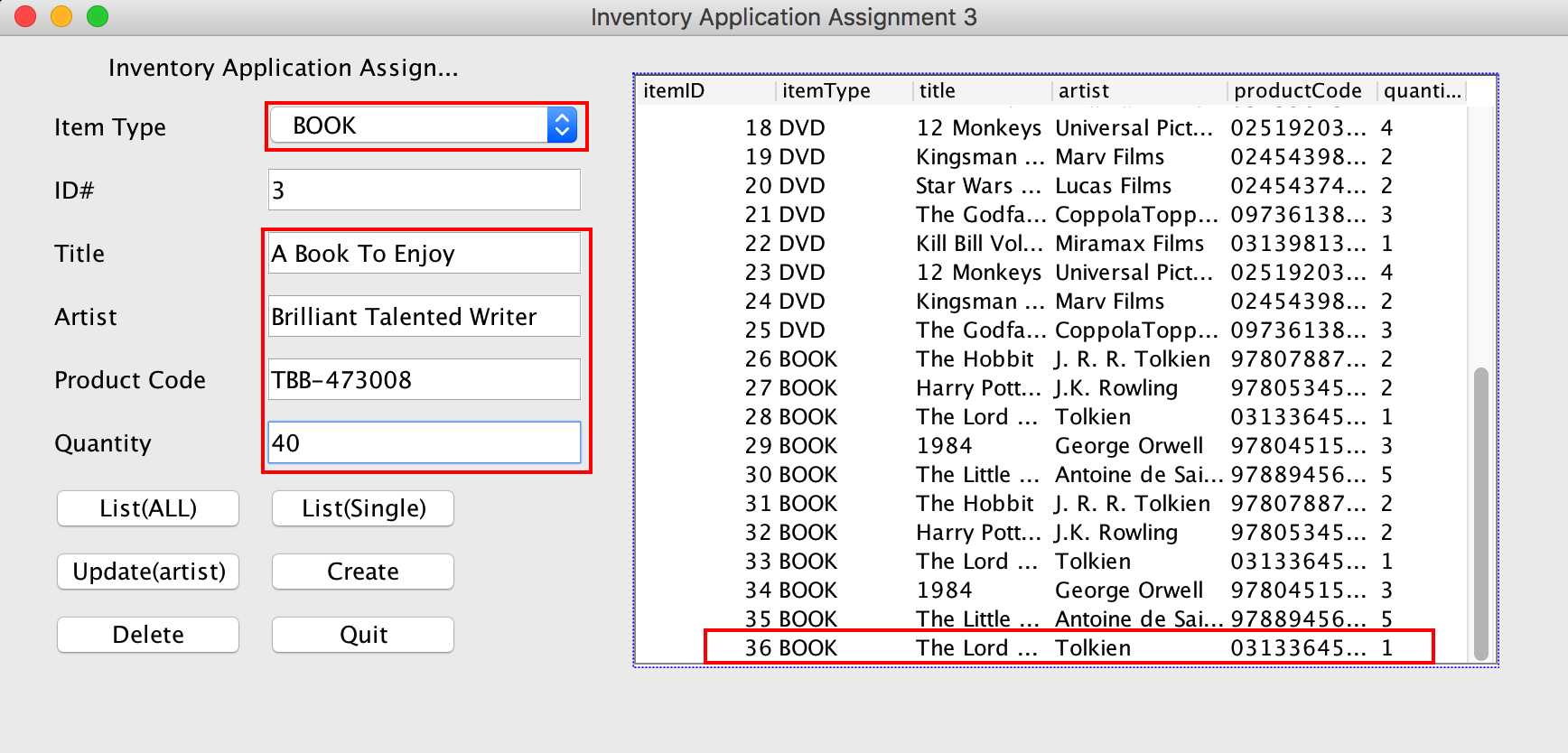


For this application only the update artist name was originally implemented in the first assignment. Enter the new artist name. The screenshot shows the original artist name and the desired change to be written to the database once the update button is executed.

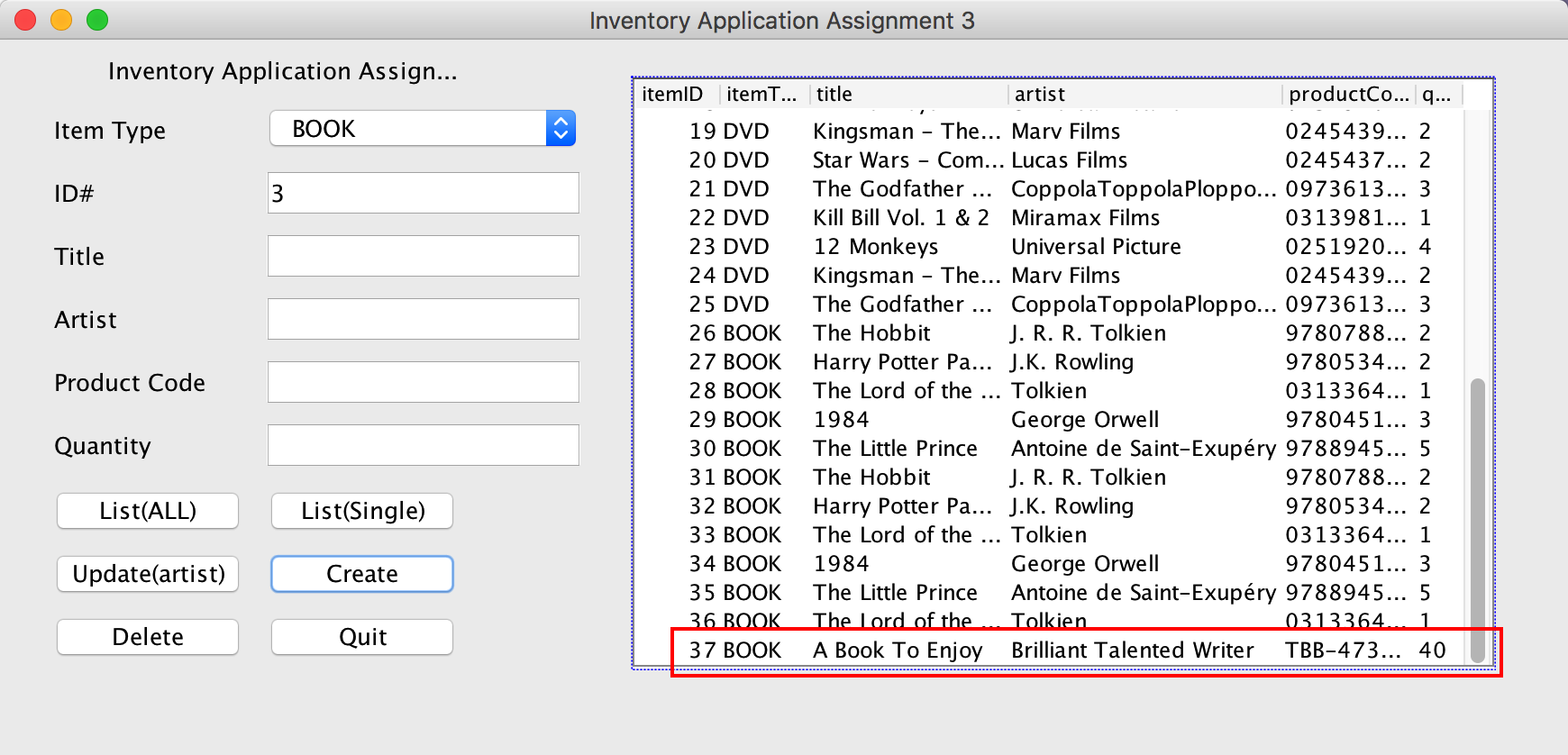


The above screenshot shows the updated artist name as it is redisplayed with all the inventory items.

1. ***Create New Inventory Item***

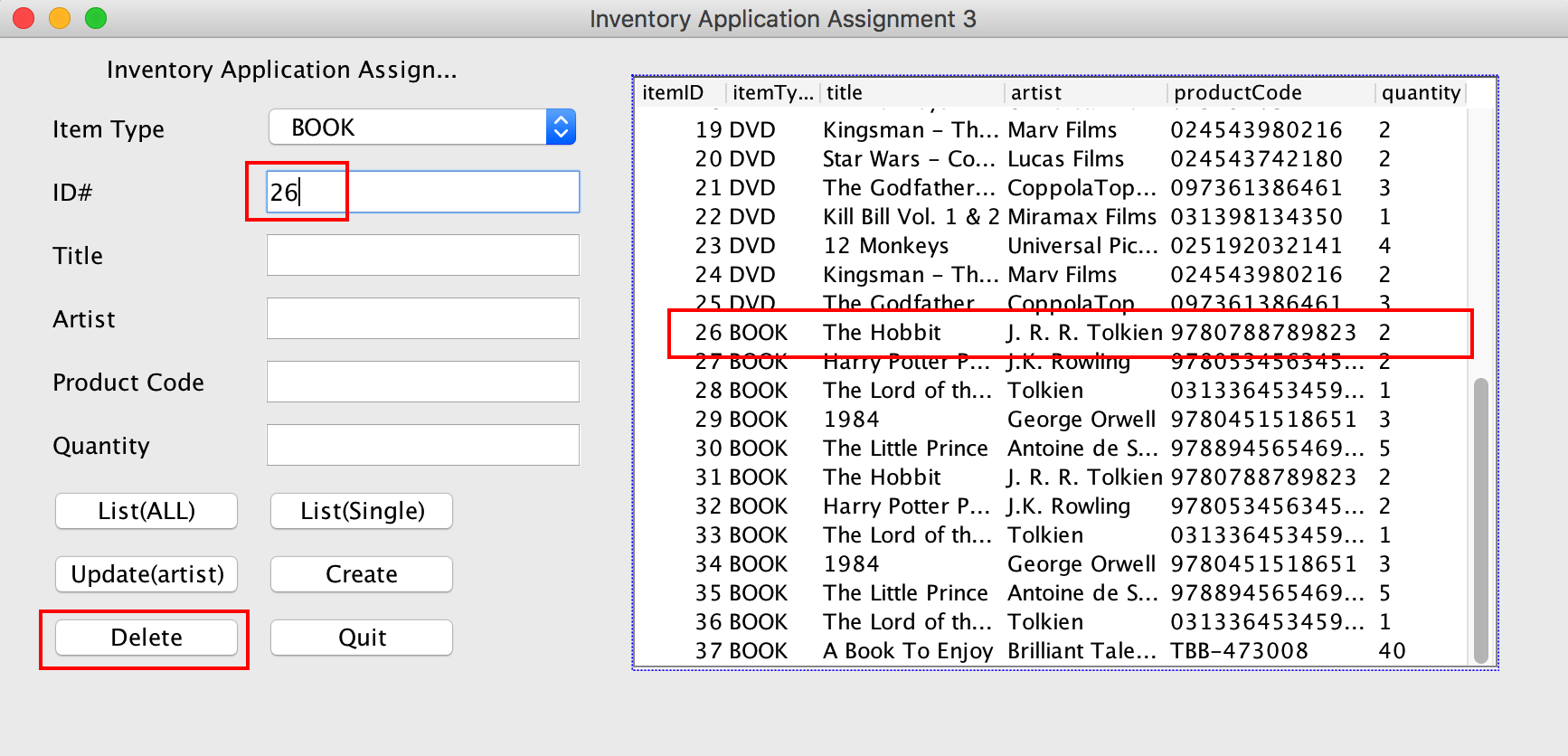


The above screenshot shows creating a new inventory item. The comboBox provides for selecting inventory type item, CD, DVD, BOOK. User enters the details of the record to be added. The right side screen shows the current record numbers for all the inventory items. Selecting the drop down itemType selector automatically selects the next item number available for a particular itemType. Clicking the “Create” button creates the next record.

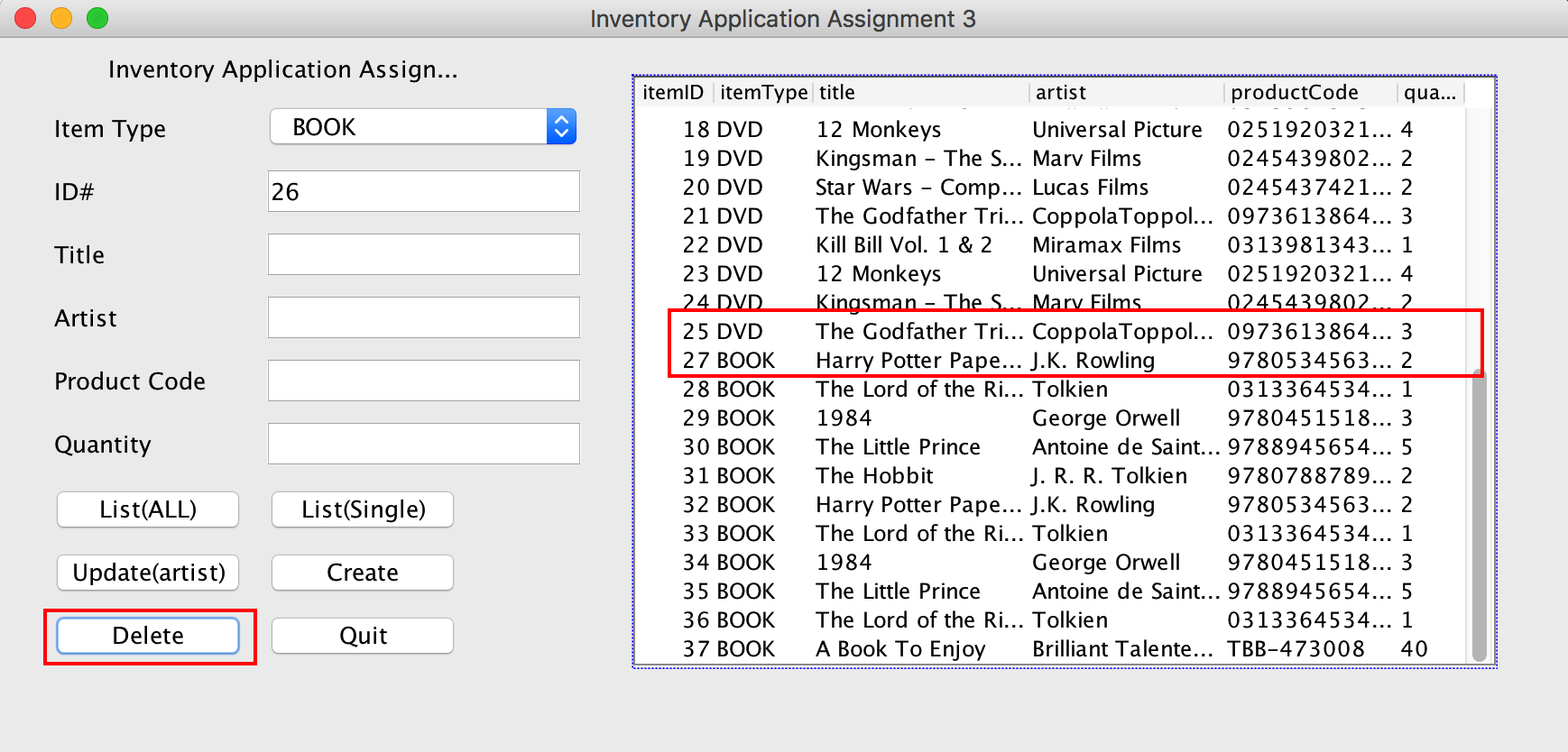


The above screenshot shows the new BOOK inventory item, using the next consecutive itemType number, 37. Also the input fields were cleared allowing for the next record to be created.

1. ***Delete Inventory Item***

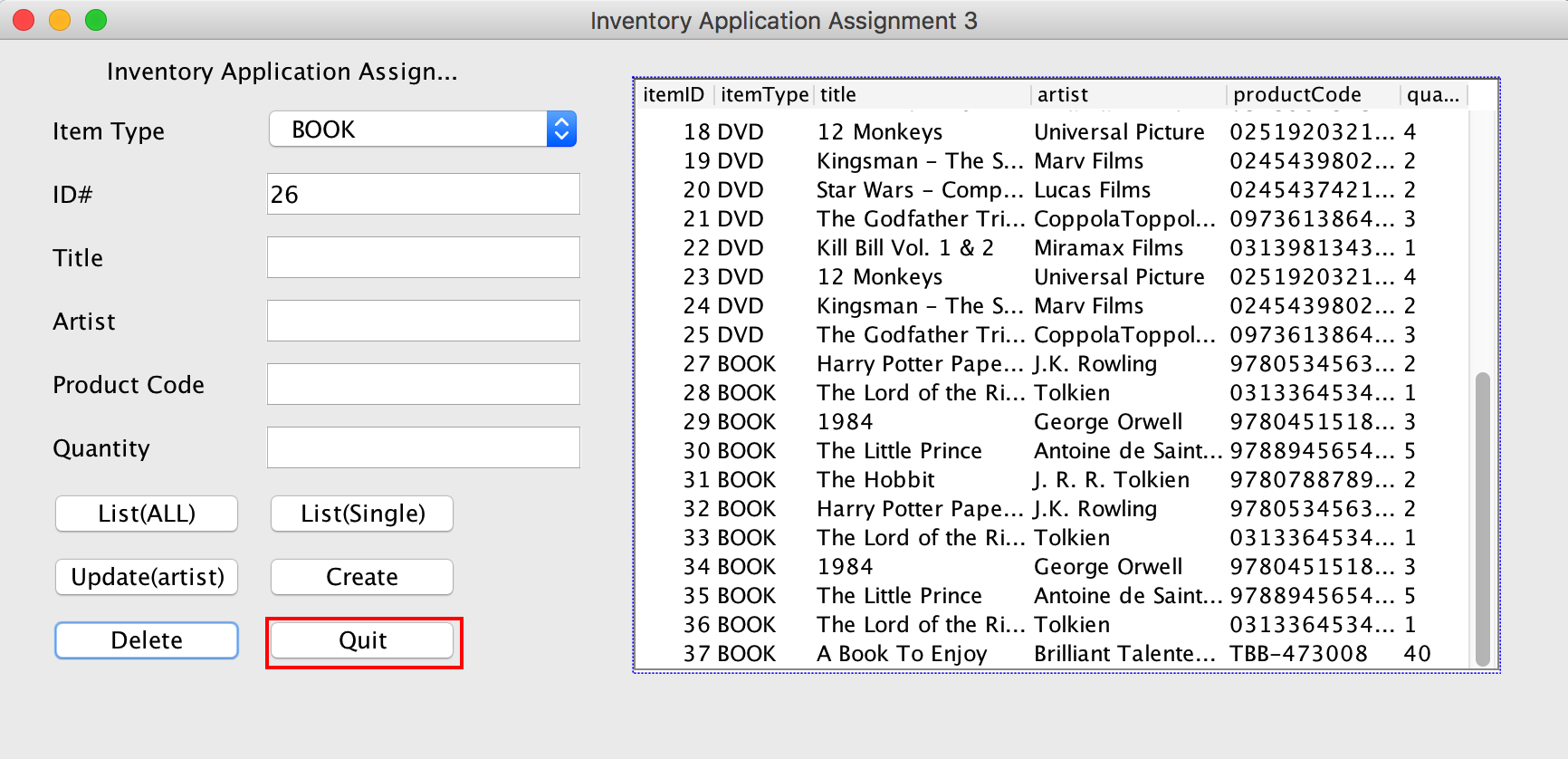


The “Delete” buttons enables the deletion of items from the inventory. By selecting an unwanted item number from the view, a user is able to delete a particular ID#, by submitting that number into the ID# field and clicking the “Delete” button.



Screenshot above shows item 26 deleted and no longer available.

1. ***Quit Application***



To exit the application, click the “Quit” button.