|  |  |
| --- | --- |
| **Database I – ITEC 340** | |
| **P03 – Java** | **Catalog System** |

This assignment will assess your ability to design and develop a Java interface for an information system.

**Assignment**

Develop a Java, text-based interface for the catalog system.

**Design**

Follow the model-view-controller (MVC) design pattern to develop an interface for the catalog information system. The key is to separate the interface from the model. Follow the instructions below.

<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>



Your Java application includes two key classes: TextIO and DataSource. All user interaction is encapsulated in the TextIO class and the DataSource encapsulates all interaction with the database. No data processing may be done in either of these classes. The control class is a middle layer between the interface and the data source.

Why is this a good design? The interface could easily be replaced by swapping out the TextIO class with a graphical user interface (GUI) or a web interface. No changes would be made to the control class or the data source. Similarly, if we decided to port the system to a different database, such as SQL Server, only the DataSource class would require changes.

**CatalogDriver**

Develop a driver class with one method, the *main* method that starts your program. The driver will instantiate the interface class, the data source class, and call the run catalog method.

**TextIO**

The TextIO class is *responsible* for **all** user interaction (input/output). This is the only class in the entire program that has a print statement or that uses a scanner.

The TextIO class has an instance variable that stores a scanner – the one and *only* scanner in the entire application.

runCatalog method – displays a startup message to the user (e.g., Catalog System running) and accepts input until the user quits the program. The run method uses a switch statement to validate the command. If the command is invalid, the run method displays a meaningful error message and waits for the next command.

If the command is valid, a helper method processes the command parameters. If the parameters are invalid, display a meaningful error message and wait for the next command. If the parameters are valid, call the appropriate method in the control class to process the command.

The run method should be short: a loop and a switch statement. Develop helper methods to process the input.

The control class calls the display method (shown below) in the TextIO class. The control class must pass a nicely formatted string to the display method. The interface class *only* manages input and output. The TextIO class does not process the response from the control class.

Develop helper methods in the TextIO class as needed.

/\*\*

\* display() -- displays the given string to the screen.

\*

\* @param msg - string to be displayed.

\*/

public void display(String msg)

{

System.out.print(msg);

}

**DataSource**

The DataSource class is *responsible* for **all** interaction with the database. The DataSource class is the only class that uses JDBC.

Setting up and tearing down database connections is expensive. Consequently, production systems do not create a database connection for each query. Database applications create a connection pool and reuse a set of database connections. Our application is single-threaded. Therefore, the data source class will create one database connection and use that connection for every query. Conse­quently, the data source class must implement a method named *close* to close the connection to the database. The application calls *close* when the user quits the application.

The DataSource class has an instance variable that stores a database connection – the one and *only* database connection in the entire application.

**CatalogCtrl**

The Catalog Controller class contains the business logic to execute user commands. Methods in the controller class call methods in the DataSource, process the results, and call the display method in the TextIO class. This creates clear separation between the interface and the database. The TextIO class is *only* responsible for user interaction, the DataSource class is *only* responsible for interacting with the database. Any additional processing is performed in the controller class.

**Interface**

You will develop a command line interface. Each 3-character command is followed by parameters separated by one or more spaces.

The relevance of parsing text strings in Java may not be obvious for a class on databases. Text processing is a very common task when working with data. Your TextIO class should have a few general methods that handle all of the parsing.

help – display a list of commands available to the user. Only list commands that have been implemented.

help

quit – quit the application.

quit

cua – create user account

cua bob smith bsmith@gmail.com

gui – get user ID

gui bsmith@gmail.com

atc – add to catalog

atc 5021 123

rfc – remove from catalog

rfc 5021 123

rrb – rate and review book

rrb 5021 123 3 “Great book”

ubs – update book status

ubs 5021 123 “Want to Read”

aaf – add a friend

aaf 123 125

Get Book Information (gbi) takes one option: -title, -author, or -id.

gbi – get book information

gbi -title “Clear and Present Danger”

gbi -author “Tom Clancy”

gbi -id 5021

**Getting Started**

The key step for this assignment is establishing a connection to the database. You will not be able to test anything until the DataSource is connected to the database.

I recommend that you start with the DataSource class. Implement the DataSource constructor to establishe the database connection with the instance variable, implement the *close* method, and implement a simple test method – model a method after one of the example programs. Start the test driver with a call to the DataSource test method to verify that the DataSource is connected to the database.

**Submit Your Assignment**

Submit Java source code for all classes including your test driver. You may submit .tar or .zip files. I am unable to open .rar files or any other type of archive file.