P02: Java Interface Part I

Design and develop a Java program that provides a command line user interface for our Flight system. Extend the program you developed for the Java Interface exercise. Your P02 solution must meet all requirements stated in the Java Interface exercise. Use the MVC design pattern and the singleton pattern for the DataSource and TextIO class.

Follow the single responsibility design pattern when designing classes and methods. Keep methods small -- easy to read, easy to test, and easy to maintain.

Use incremental development and test-driven development. The key to successful software development is to write small amounts of code and test all code thoroughly before you code any additional functionality.

The best programmers *fail fast* by testing early and often. Do not waste time going down a dead end. Take a minute to read about High Velocity Leadership, a valuable book about the Mars Pathfinder mission.

Assignment
Develop a Java interface that provides the following functionality.

Command Line Interface
Develop a command line interface (CLI). Each 3-character command is followed by parameters separated by one or more spaces.

For this assignment, implement the following commands:

help -- display a list of commands available to the user. Only list the commands that are implemented.

quit -- quit the application.

lst airports -- list all airports in the system.

fnd <departure airport> <destination airport> -- list all direct flights between the origin and destination.

prt ticket #

List Airports
lst airports

The interface accepts the command *lst* followed by the parameter "airport".

Use a JDBC *statement* and a *materialized view* to display a list of airports. For each airport, show the airport code and the city and state where the airport is located. Update the materialized view on commit.

For example: ORD Chicago IL

Find Flights
fnd <departure airport code> <destination airport code>

Given two airport codes (departure and destination) return information for all *direct* flights between the origin and destination.

Use a JDBC *prepared statement* and a *logical view* to display the following information for each flight.

Flight #, origin, departure time, destination, arrival time, number of open seats

All output messages are displayed by the TextIO class. For simplicity, the DataSource class will call TextIO directly to display error messages. The DataSource class may store an instance of TextIO as a private instance variable initialized by the constructor.

Print Ticket
prt ticket #

Ticket #		**First**	**Last**	
123		James	Smith	
Date	**Leave**	**From**	**To**	**Arrive**
7/7/23	8:35 AM	ROA	CLT	9:48 AM
7/7/23	11:54 AM	CLT	SFO	5:02 PM

The interface accepts the command *prt* followed by parameter "ticket" and the ticket number.

Given a ticket number, print the ticket as shown below.

The ticket has a header line with the ticket number and the first and last name of the passenger.

For each leg of the ticket, print the departure date, departure time, departure airport code, destination airport code, and arrival time.

The output must be readable. Use tabs between columns on each line.

Running Your Program
I will run your program from the Unix command line. Test your program from the command line. Do **NOT** submit any packages. Packages create problems running the code. If I cannot run your code from the command line you will lose a minimum of 50%.

Your program **must** accept two command line arguments, the username and password for the database. You and your partner should be able to run your program from your own database. Providing the username and password on the command line makes it easy to run the program with different Oracle accounts.

Input/Output
Your program may *only* interact with the user through the TextIO class. The one and only exception is the DataSource class which may print error messages directly to the monitor using an SOP statement. This is not how a product system would work. We are simplifying the project to focus on the most relevant details.

Input Validation
Verify that each command has the correct number of arguments. Do **not** invest time validating the input values. You may assume you have intelligent users who do not make mistakes. Focus on the database and apply software engineering best practices.

DataSource
The DataSource class may **NOT** expose any details of the data storage mechanism. The DataSource will provide one method for each operation. The client will call methods in the DataSource class with simple parameters and the DataSource will return a Data Access Object -- create a class or classes to return data from each DataSource method. For example, create a class named Airports that contains an airport code, city, and state, and create a class named AirportsDAO that contains an array of Airports.

Database Interaction
The Java interface may NOT directly access any database table. Why?

Shutting Down
Your program must close the database connection. When the user enters "*quit*", the program should close the connection. To be safe, the main method should also attempt to close the database connection. Use the Java *finally* clause to close the database connection if a database operation fails.

Keep It Simple
The best designs are simple. If your design feels complicated, stop and rethink the design. You will lose points if I cannot understand your design with a reasonable amount of time and effort. Develop and submit a simple application. If you want to extend your application with additional functionality, create and modify a copy of your application and submit a video showing the additional functionality. No extra credit will be awarded for additional functionality, but I will review and comment. You have a lot to do before the end of the class. Use your time wisely.

Testing
Submit a simple test driver named TestDrv that executes three test cases, one for each command above. For each command, the output should print the command and print pass or fail. For example:

lst airports -- pass

lst -- test case with at least three airports
fnd -- test case with two flights
prt -- ticket with two legs

You are encouraged to do additional testing. Submit your test driver.

Submit Your Assignment
Submit all Java source code files and the scripts to create and load the database schema. Do **NOT** submit the JDBC driver.

Submit all files (**.java and* *.sql) in a zip or tar file. I will **NOT** accept rar or any other archive file. Use zip or tar.