Program: Create a class that allows one to work (indirectly) with the database table SPFLI

The table SPFLI contains information on flights offered by different airlines. In our organization, people frequently need to incorporate data from this table into their programs. To assist in this process, we will create a class that people can instantiate in their programs as an easy way of accessing this data.

So as not to disrupt the work of other students, your first task is to **copy ZC001SPFLI to your own ZCxxxSPFLI database table.** This should be an exact copy in terms of fields, types, data, etc.

Follow these directions to make a copy of ZC001SPFLI:

1. **Go to transaction SE11**
2. **Make sure Database Table is selected and type SPFLI into the corresponding field**
3. **Click the copy icon**
4. **In the bottom input field, type your new table name ZCxxxSPFLI**
5. **Select the check icon**
6. **Select local object**
7. **Select the Activate button**

**This should be done prior to beginning work.** Create two programs—one that copies all of the data from SPFLI to your table and one that clears out all the records from your ZCxxxSPFLI table. **These two programs should be included as part of your lab submission but are not a part of the main assignment. These programs do not need to be Object Oriented.**

In this program we will design and implement a class that is local to our program so as to make sure the design is effective. In a future lab we will install this class as a global class in the ABAP Dictionary.

Use transaction SE11 to note the various fields in ZCxxxSPFLI. You may wish to create a screen capture of the fields in this table and the data in the table to have for future reference as you write your program. (This does not need to be turned in for grading.)

Your class will have one attribute—an internal table that replicates *all* of the data found in ZCxxxSPFLI.

Your class will have multiple methods that are described below. You are responsible to deduce any necessary parameter passing based on the specification listed. Be particularly careful to note which methods access the attribute internal table and which reference the database table. You are free to create local data objects in your methods as needed for them to accomplish their defined tasks. Only pass to or from a method those things that are necessary and specifically defined.

**Except where specifically noted, instructions such as “take in,” “output,” and “return” refer to parameter passing. User input (via a data entry screen) and writing to the display are specifically described as such. Note that it may be indicated that a method returns more than one piece of data. In this situation, it should be noted that this is a reference to using the CHANGING or EXPORTING method of parameter passing since RETURNING can only return a single item. It should be used when appropriate.**

**Some elements of program behavior are left as a design decision for the developer. Be sure to consider each method completely.**

**All references to the database or table in the below are referencing ZCxxxSPFLI, NOT SPFLI.**

* constructor—this method, called automatically at object instantiation, should load the internal table with data from the database. It should follow best practCices for error handling. This method writes nothing to the screen (presuming that all the data was correctly loaded).
* showAllFlights—this method will write out, as a table to the user display, all the data in the table (except for MANDT). Make the table look nice and place headers at the top of each column. After displaying the table, write the text “x flights displayed” where x is the number of flights in the table.
* showFromCountryFlights—this method should perform the same operation as showAllFlights, but it should only display those flights that are from a designated country which is passed in as a parameter. In the event the country passed in has no flights, an appropriate message should be written to output.
* showCityFlights—this method should perform the same operation as showAllFlights, but it should only display those flights that are from a designated city which is passed in as a parameter and flights that are to that same city. The display should first show a table for all the flights to the city, the number of flights displayed, then the flights from the city, and the number of those flights displayed.
* getConnidData—this method will take in a connection ID and return back to the caller a work area populated with the information from that connection ID.
* numFlightsTo—this method will take in an airport code (JFK, FRA, etc.) and return the *number* of connections that travel to that airport.
* getConnid—this method will take in two airport codes—one representing the departure airport and the other representing the arrival airport—and *returns* a valid connection ID of that particular flight sequence. If the airport combination is invalid, the method returns 0.
* getFlightTime—this method will take in a connection ID and return the number of minutes in flight time. This method writes nothing to output.
* AddConnection—this method takes in a workarea filled with information on a new connection to be added to the database table. The method should add this workarea to ZNxxxSPFLI. If it is able to successfully add the new record, it should also add it to your internal table.
* ChangeCityTo—this method takes in a connection ID, a city, and a country. It changes the destination city to the new city and country for the specified connection ID in both the database and the internal table.

It is recommended that you implement each method and get it working before adding the next.

**Create a short program to test your class.**

**It is not necessary for this program to take in user input.** Your program should begin by instantiating an object based on the class. It should then progressively test each method, writing the results of each “test” to the screen. Add necessary labels or headings to your output so that the user can deduce which methods are being tested, what their result is, etc. Test both successful and “failed” results where appropriate.

As an example, your program should product output similar to the following:

Testing getConnidData with a connid of 0017:

(Output shown)

Testing getConnidData with a connid of 0001:

(Output shown)

Note that in some instances the methods write output to the screen, but in other instances you will need to display the return value from the method or present the whole table. Make sure that your program demonstrates that your methods fulfill the specification. Make your program output such that one can examine just the output and know that everything is correctly working.

To submit your work, upload to D2L a Microsoft Word document containing the following:

* The name of your program. (The program may be run tested for grading.) Make sure the program is activated.
* A copy of your ABAP source code. (Copy the code in its entirety and paste it into the Word document as text.)
* A screen shot of your application’s output shown running in the SAP GUI.

**Upload to the Lab 5 dropbox by the due date posted in D2L.**