Introducing Object-Based Concepts in a COBOL Course

Gerald P. Marquis

ABSTRACT: *While making decisions on curriculum content, many colleges have to resolve the conflict between teaching the traditional skills required by future employers of their students, and teaching new technology. These decisions are further constrained by the limited number of courses available for the major. One such conflict is between teaching COBOL and teaching Object-Oriented (O-O) techniques. A review of the current literature, both practitioner and academic, clearly indicates that the teaching of the O-O paradigm needs to be included within MIS curricula. Two approaches can be taken: to integrate the teaching of O-O concepts into several courses spread across the curriculum; or to teach a single course covering the entire O-O model. It is suggested that the more practical approach is to distribute the teaching of O-O concepts across several courses. In particular, the elements of encapsulation and modularity can be addressed in a beginning course covering the COBOL language. The paper begins with a literature review and an explanation of the major elements of the O-O model. The paper then describes a student project that was developed to introduce two of these concepts in a course in COBOL.*

KEYWORDS: *Object-Based Model, Encapsulation, Modularity, COBOL, IS Curriculum*

INTRODUCTION

There is a conflict that many Information Systems educators face regularly when contemplating curriculum design and course content. This clash arises when curriculum designers have to balance the needs of their students' future employers with the IS department's desire to include the latest technology trends in our IS curriculum. In particular, how do we balance the need to introduce the concepts of the object-oriented (O-O) paradigm and the need to teach COBOL as an entry level skill for future employment? This paper addresses this issue and relates how the author has, in part, combined these two desired outcomes into one course.

The next section presents some background comments and objectives to be considered in the design of a curriculum for the IS major. Section three presents a summary of some of the major concepts found in the O-O paradigm. The fourth section is a discussion of what O-O concepts can be taught in a COBOL course and a general approach on how to teach these concepts. Section five is a description of a student project that has been used to introduce the ideas of encapsulation and modularity within a COBOL course.

BACKGROUND

As of 1991 there were an estimated seventy billion lines of COBOL code in production throughout the United States in various firms. [1] This is a clear indication that a working knowledge of the COBOL language is a skill that many potential employers are seeking when interviewing students for entry level positions. This is particularly true in major metropolitan areas that have a large installed base of systems running on mainframe computers. Thus, for many schools and colleges it is imperative to include a course in the COBOL language in their curriculum.

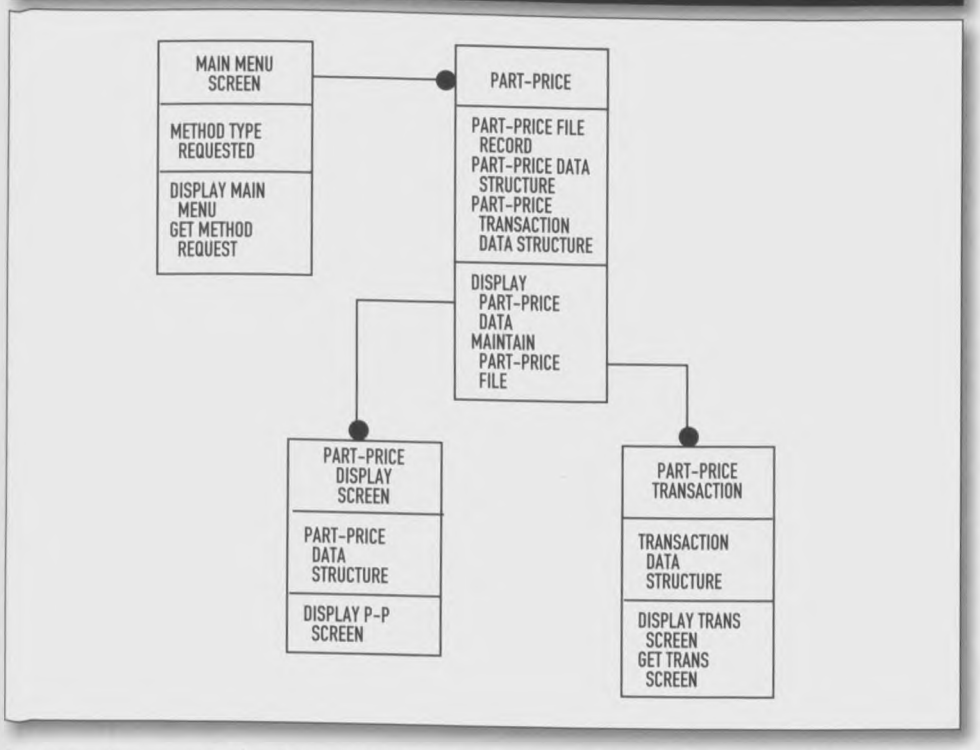
At the same time, it is known that object-oriented (O-O) technology is one of the "hot" current trends. Many firms are also looking for graduates that have an understanding of this methodology. Students that also have a firm foundation in O-O concepts will be in a better position for securing their first positions after graduation.

A further indication of the need to include O-O topics in the IS curriculum is documented by looking at the current CIS model curricula as presented by the DPMA and the ACM/IEEE-CS. As pointed out by Gotwals and Smith, both of these "curriculum development groups expect up-to-date CIS curricula to begin to include coverage of object-oriented aspects of programming, analysis, and design." [2]

It follows that the O-O paradigm should be included within our curricula. The question is how to best accomplish this within the limited number of courses available in the typical IS major, while trying to address as many topics as possible in our exploding discipline. The two approaches that can be taken are, either to integrate the teaching of O-O concepts into several courses spread across the curriculum, or to teach it all in a single course. Cain argues that O-O terminology "cannot be neatly packaged into one CIS course." [3] He supports the distributed approach over courses like systems analysis, systems design, COBOL programming, and database. In contrast, Rajkumar states that "a possibly better approach is to teach object-orientation concepts in a separate course." [4] To this end, he has described this single course's content to include basic concepts, programming in C++, object-oriented analysis (OOA), object-oriented design (OOD), and issues in managing O-O technology.

To compound the problem, there is evidence that object-oriented technology is harder to learn than the process-oriented technology currently, i.e., structured analysis, structured design, functional decomposition,

Figure 1: Object diagram for the PART-PRICE system



being taught in most schools. As Fichman and Kemerer point out, "To be successful with O-O...an adopter must absorb a new lexicon, new development methods, and new development tools." [5] The O-O approach will take more time to teach to students who are more familiar with the process-oriented world of registering for classes or arranging for an inter-library loan of reference materials. This longer learning curve is supported by a DATAPRO examination of O-O use in industry that "indicates that OOD [object-oriented development] analysis and design is more costly than conventional analysis and design." [6] The article goes on to explain that this additional cost is in time and effort during analysis and design. This suggests that, as Cain presented, O-O concepts might best be taught in small doses across several courses rather than trying to cover the whole topic in a single course.

Against the background of many years of teaching, the author advocates the distributed approach. Experience has shown that it is only by repetition over a period of time, and across several courses, that the typical student will internalize and understand the more complex topics of our discipline. Each of these repeated exposures should be in greater depth and detail.

SOME BASICS OF THE O-O PARADIGM

To understand the O-O paradigm one must start with the concept of an object. Embley et al. describe an object as "a person, place, or thing... [that] may be physical or conceptual... an object is a single entity or notion." [7] An object can be an employee, a part, a department, a screen, or a window. An object class "describes a group of objects with similar properties (attributes), common behavior (operations), common relationships to other objects, and common semantics". [8] Thus, an object class is a set of common objects and an object is an instance of an object class.

Attributes are data values held by objects in an object class. The generic name of an attribute is generally included in an object class representation. For example, the object class STUDENT may have an attribute AGE, but the object Jane will have an attribute 21.

Operations and methods both refer to actions that can be applied to object instances within a class. "An operation is a function or transformation that may be applied to or by objects in a class." and "A method is the implementation of an operation for a class." [9] The terms method and operation are often used interchangeably in the literature.

An object class is modeled as a rectangle divided into three horizontal sections. The top section contains the name of the object class. The middle section contains the names of the attributes of the objects represented by the class. The bottom section contains the names of the operations that can be performed on the objects of the class. Examples of object class diagrams can be found in Figure 1.

To operationalize these concepts, a framework is needed to organize the relationships between object classes. Grady Booch provides such a conceptual framework by identifying four major elements of the object model: Abstraction, Encapsulation, Modularity, and Hierarchy. [10] To review, "An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects" [11] and encapsulation is the "packaging of related data and procedures into objects" that uses a "technique...called information hiding." [12] Modularity refers to "the property of a system that has been decomposed into a set of cohesive and loosely coupled modules." [13] And finally, the element of hierarchy is defined as "a ranking or ordering of abstractions." [14]

The question is, what concepts can be introduced in which courses to lay a foundation for the understanding of the object model for the student? In particular, which O-O elements can be introduced in a first course in COBOL so that professors can begin the task of guiding the student in the understanding of the O-O model?

WHAT CAN BE TAUGHT IN A COBOL COURSE

Clearly, COBOL is not an object-oriented language since it does not possess all of the last four elements enumerated above, and no attempt is being made here to represent COBOL as an O-O language. In particular, COBOL does not provide the inheritance, generalization, or specialization relationships intrinsic in the hierarchical structure between abstractions. However, Booch's elements of encapsulation and modularity can be addressed in the COBOL language.

Teaching the Encapsulation Concept

The concept of encapsulation can be demonstrated in COBOL by creating subprograms for each object class; for example, by developing one subprogram for each file. Each subprogram contains the file description for an object class and code for all of the methods (operations) that will affect the contents of that file. Or, there could be a subprogram for each individual screen object and its related methods. Each subprogram contains separate routines to accomplish each operation required on the object class. The details of this code are contained wholly within each subprogram and thus each subprogram operates as a "black box" within the system.

The subprograms are executed with a CALL statement, which, when appropriate, passes a field designating the method to be executed in the subprogram; or passes data to be used by the subprogram; or passes both. This is the COBOL implementation of message

passing from a client object to another object to cause a method to be executed. This technique implements the information hiding feature embodied in encapsulation. The calling program does not have a copy of the file description or the screen format; therefore, it has no knowledge of the structure of the object class. And, the calling program does not have knowledge of the algorithms used to implement the various methods included in the subprogram.

Teaching the Modularity Concept

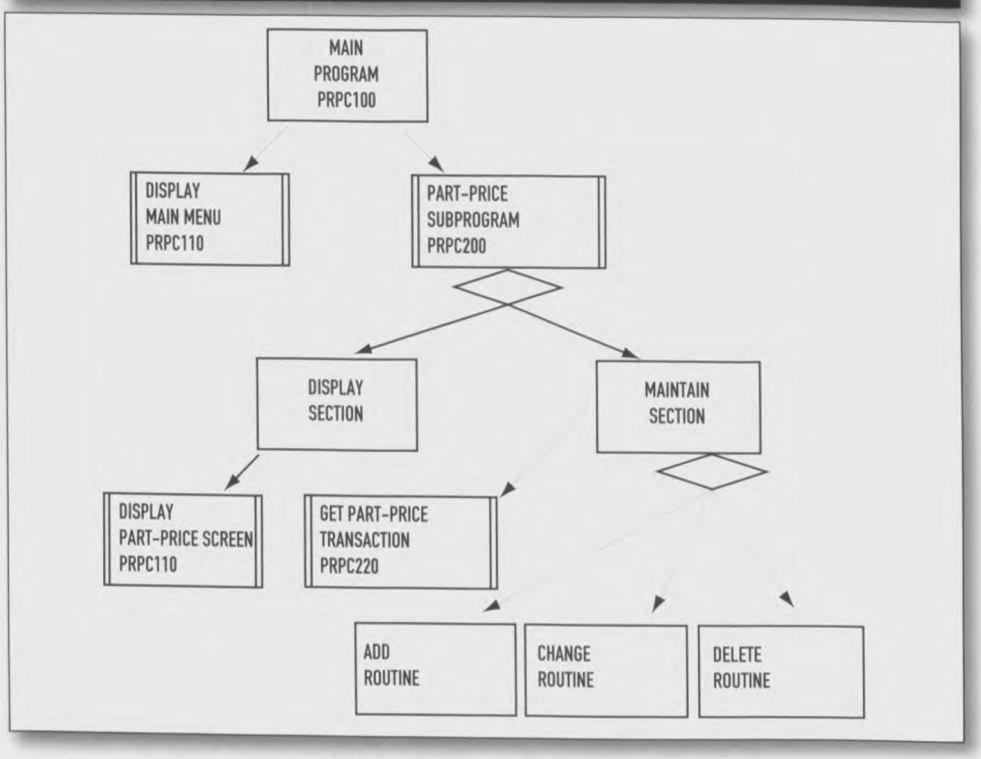
The concept of highly cohesive and loosely coupled modules is not new to programming and should be a part of every language course. Modularity can be initially taught in the COBOL language using small, single functioned paragraphs in the PROCEDURE DIVISION. Most students do not have difficulty understanding cohesion and coupling after completion of several projects, and they will understand the value of reusable paragraphs within a program. The O-O view of a module as a collection of functions is just an adjustment of size.

In the object model, the module can be represented as a subprogram that contains the code to execute each method. An easy way to isolate the various functions within the module (subprogram) is to place the code required for each method (function) into separate SECTIONs within the PROCEDURE DIVISION. For example, in the subprogram PRPC200, all code pertaining to the maintenance of the part-price file is contained in the M1000-MAINTENANCE SECTION of the subprogram. The subprogram contains one data structure for the object class that this module manipulates in the DATA DIVISION, and that data structure is available to all of the SECTIONs in the module. By structuring the subprogram in this manner we have created a single module encompassing both data and functions (methods) pertaining to a single object class. It is functionally cohesive, containing all the operations that act upon the object, and it is loosely coupled since the client object needs only to CALL the subprogram with the requested operation (method).

DESCRIPTION OF THE COBOL PROJECT

To introduce these two concepts, a student project was developed for a course in COBOL. Since this is generally the student's first programming course, the project is kept quite elementary. The project is centered around a single class, an indexed part-price file. This object class is named PART-PRICE in Figure 1. Objects (instances) of this class can be thought of as part-price records.

Figure 2: Structure diagram for the PART-PRICE system



The PART-PRICE module (object class) has two methods: (1) display of the fields of an object (an instance) given the part number; and (2) maintenance of objects in the price class. Maintenance includes the creation of new objects (adds), the modification of the non-key fields of an object (changes), and removal of objects from the part-price class (deletes).

Three other object classes are introduced to handle the screen input and output. These include screens to process the main menu (PRPC110), to display a part-price object (PRPC210), and to accept a maintenance transaction (PRPC220). Each of these are implemented as separate subprograms. The object model in Figure 1 shows the relationships between the four object classes. Each object class in the diagram includes the class name, the attributes it uses, and the methods it employs.

To operationalize this object model one additional module is needed, a main program (PRPC100), that controls the top level processing of the system and provides the entry point to begin execution. The system structure diagram is illustrated in Figure 2. The modules that operate on object classes are shown as subprograms (double side bars). The two methods in the part-price subprogram (PRPR200) are shown as separate SECTIONs of this subprogram selected by the decision indicated in the diamond. The add, change and delete routines within the maintain method are again invoked as a result of a decision depending upon the transaction type.

In total, the student project entails one main program and four subprograms. Each of these are written and compiled separately and dynamically linked at run time. The link to execute a subprogram is achieved with a CALL statement in the form: CALL "subprogram-name" USING data-field(s).

This is the COBOL equivalent to a client object operating on another object via a message with either a method request, or passing data to the object, or both. A copy of an indexed part-price master file is provided to the students so they do not have to deal with the intricacies of the original creation of an indexed file. A copy of the source code for all programs is in the appendix to this paper.

The Main Program (PRPC100)

The Main program for this project provides the entry point and controls the iteration until the user terminates the program. This program calls the Display Main Menu subprogram to receive the user's choice of operations to be performed. This is followed by a call to the Part-Price subprogram with the method received from the Display Main Menu subprogram.

The Display Main Menu Subprogram (PRPC110)

The Display Main Menu subprogram displays a menu and accepts the user's choice of operations to be performed. The choices are to display a price record on the screen, to perform price file maintenance, and to quit. The user request (1, 2, or 3) is translated into a

method code ("DISP ", "MAINT", or "QUIT ") using an EVALUATE statement. This code is then passed back to the main program. Clearly, the client program (main program) has no knowledge of the internal logic of the server object (display menu subprogram) from which it is using resources.

The Part-Price Subprogram (PRPC200)

The Part-Price subprogram is organized around the indexed part-price file object class. The DATA DIVISION contains the data structure for each instance (record) in the class. The PROCEDURE DIVISION is structured in three SECTIONs, a control section and one section each for the display method and the maintain method.

The first SECTION is a case structure implemented with an EVALUATE statement. This routine tests the value of the operation request passed to the subprogram through the USING clause of the CALL statement. Depending on the value passed ("DISP " or "MAINT"), this segment determines which other SECTION of the subprogram will be executed for the requested operation, the DISPLAY SECTION or the MAINTAIN SECTION.

The DISPLAY SECTION contains the code to first display a screen requesting a part number. Next it reads the part-price file for the requested part. Third, this routine moves the file data to data fields to be passed to the next subprogram. Forth, it calls the subprogram Display Part-Price Screen to display the information for the requested part passed to it through the USING clause of the Call statement. Finally, at the end of each iteration, there is a prompt message asking whether the user wants to display another part from the part-price class. The answer to this prompt then either repeats the actions or terminates the method and returns control to the client object (the Main program).

The MAINTAIN SECTION first calls the Get Part-Price Transaction subprogram to obtain a transaction through the USING clause. Once the transaction has been returned from the called subprogram, the method executes the appropriate code depending upon the transaction type (method request), add, change, or delete. After completing this process, this method allows the user to continue the maintenance function or terminate, returning control to the client object (the Main program).

The Display Part-Price Screen Subprogram (PRPC210)

The Display Part-Price Screen subprogram contains all the code required to display a single instance of the Part-Price class. The Part-Price data is passed to this routine through the USING clause of the CALL statement. This code segment simply displays the data for the requested part number.

The Get Part-Price Transaction Subprogram (PRPC220)

The Get Part-Price Transaction subprogram first displays a data entry screen for all fields and accepts the data entered by the user. After the data are accepted, a prompt for the type of transaction is displayed; add, change, or delete. Finally, there is a prompt to allow the user to correct the data before they are passed back to the calling subprogram.

CONCLUSION

By including the project described in this paper in a course in the COBOL language, students are introduced to two object-based concepts early in their academic career. The object model elements developed in this project are encapsulation and modularity. Although COBOL is not now an O-O language, the ideas of information hiding and modularity can be demonstrated through the use of called subprograms. With early introduction to some of the concepts embodied in the object model, the student will have a firm foundation to build upon in future courses that use the O-O paradigm.

ACKNOWLEDGEMENTS

The author would like to thank the anonymous reviewers for their helpful comments for improving the manuscript.

REFERENCES

[1] Appleby, Doris, "Classic Languages, Part 2: COBOL," *BYTE*, October 1991, pp. 129-132.

[2] Gotwals, John K., and Mark W. Smith, "Bringing Object-Oriented Programming into the Undergraduate Computer Information Systems Curriculum," *Journal of Information Systems Education*, Vol. 5, No. 3, Fall 1993, pp. 2-8.

[3] Cain, William P., Object-Oriented Programming and the CIS Curriculum," *Journal of Information Systems Education*, Vol. 3, No. 1, Spring 1991, pp. 2-7.

[4] Rajkumar, T. M., "An Object-Oriented Information Systems Course," *Journal of Information Systems Education*, Vol. 5, No. 3, Fall 1993, pp. 9-16.

[5] Fichman, Robert G., and Chris F. Kemerer, "Adoption of Software Engineering Process Innovations: The Case of Object Orientation," *Sloan Management Review*, Vol. 34, No. 2, Winter 1993, pp. 7-22.

[6] Teti, Frank, "Object-Oriented Development: A Managerial Emphasis," *Datapro: Managing Information Technology*, McGraw-Hill, Datapro Information Services Group, Delran, NJ, 1993.

[7] Embley, David W., Barry D. Kurtz, and Scott N. Woodfield, *Object-Oriented Systems Analysis: A Model-Driven Approach*, Yourdon Press, Englewood Cliffs, NJ, 1992, p. 18.

[8] Rumbaugh, James, et al., *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991, p. 22.

[9] Ibid., p. 25.

[10] Booch, Grady C., *Object Oriented Design with Application*, Benjamin/Cummings, Redwood City, CA, 1991, p. 38.

[11] Ibid., p. 39.

[12] Harmon, Paul and David A. Taylor, *Objects in Action: Commercial Application of Object-Oriented Technologies*, Addison-Wesley, Reading, MA, 1993, p. 3.

[13] Booch, op. cit., p. 52.

[14] Booch, op. cit., p. 54.

APPENDIX

```
IDENTIFICATION DIVISION
PROGRAM-ID. PRPC100.
AUTHOR.
*
*  MAIN PROGRAM FOR THE PART-PRICE SYSTEM
*
*
ENVIRONMENT DIVISION.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 OPERATION-REQUEST              PIC X(05).

PROCEDURE DIVISION.

1000-MAIN.
   PERFORM 2000-PROCESS-ITERATION
     UNTIL OPERATION-REQUEST = "QUIT ".
   STOP RUN.

2000-PROCESS-ITERATION.
   CALL "PRPC110" USING OPERATION-REQUEST.
   IF OPERATION-REQUEST = "QUIT "
     NEXT SENTENCE
   ELSE
     CALL "PRPC200" USING OPERATION-REQUEST.

------------------------------------------------
IDENTIFICATION DIVISION.

PROGRAM-ID. PRPC110.
AUTHOR.
*
*  THIS SUBPROGRAM CONTROLS THE MAIN MENU OBJECT
*
*
ENVIRONMENT DIVISION.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 CONTROL-FIELDS.
   05 COMMAND          PIC X VALUE SPACE.
      88 DISPLAY-P-FILE          VALUE "1".
      88 MAINT-P-FILE            VALUE "2".
      88 QUIT                    VALUE "9".
      88 VALID-CHOICE            VALUE "1" "2" "9".

01 SCREEN-DATE.
   05 SD-MO             PIC X(02).
   05 SD-SLASH-1        PIC X(01) VALUE "/".
   05 SD-DA             PIC X(02).
   05 SD-SLASH-2        PIC X(01)   VALUE "/".
   05 SD-YR             PIC X(02).

01 WORK-DATE.
   05 WD-YR             PIC X(02).
   05 WD-MO             PIC X(02).
   05 WD-DA             PIC X(02).

LINKAGE SECTION.

01 PASS-DATA.
   05 PD-MENU-CHOICE  PIC X(05).

PROCEDURE DIVISION USING PASS-DATA.
```

```
1000-MAIN.
   ACCEPT WORK-DATE FROM DATE.
   MOVE WD-MO  TO  SD-MO.
   MOVE WD-DA  TO  SD-DA.
   MOVE WD-YR  TO  SD-YR.
   MOVE SPACE  TO  COMMAND.
   PERFORM 2000-PROCESS-USER-COMMANDS
      UNTIL VALID-CHOICE.
   EVALUATE TRUE
      WHEN COMMAND = "1"
         MOVE "DISP "    TO  PD-MENU-CHOICE
      WHEN COMMAND = "2"
         MOVE "MAINT"    TO  PD-MENU-CHOICE
      WHEN COMMAND = "9"
         MOVE "QUIT "    TO  PD-MENU-CHOICE
   END-EVALUATE.

1001-EXIT-PROGRAM.
   EXIT PROGRAM.

2000-PROCESS-USER-COMMANDS.
   PERFORM 3000-DISPLAY-MENU.
   PERFORM 3100-GET-COMMAND.

3000-DISPLAY-MENU.
   DISPLAY SPACES                  LINE  1 COL  1.
   DISPLAY SCREEN-DATE             LINE  1 COL  1.
   DISPLAY "WALCABA CORPORATION"LINE  1 COL 31.
   DISPLAY "PF 1.0"                LINE  1 COL 73.
   DISPLAY "PRICE FILE MENU"       LINE  3 COL 33.
   DISPLAY "1  DISPLAY PRICE RECORD ON SCREEN"
                                   LINE  5 COL 25.
   DISPLAY "2  PRICE FILE MAINTENANCE"
                                   LINE  7 COL 25.
   DISPLAY "9  EXIT TO SYSTEM"     LINE  9 COL 25.

3100-GET-COMMAND.
   DISPLAY "ENTER COMMAND CODE" LINE 13 COL 30.
   ACCEPT COMMAND                  LINE 13 COL 55.

-----------------------------------------------
IDENTIFICATION DIVISION.

PROGRAM-ID. PRPC200.
AUTHOR.
*
*SUBPROGRAM TO PROCESS THE PART-PRICE FILE OBJECT
*
ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

   SELECT PRICE-FILE
      ASSIGN TO "C:\PRICE.LST"
         ORGANIZATION IS INDEXED
         ACCESS IS RANDOM
         RECORD KEY IS PR-PART-NBR.

DATA DIVISION.

FILE SECTION.

FD PRICE-FILE
   LABEL RECORDS ARE STANDARD
   DATA RECORD IS PRICE-RECORD.

01 PRICE-RECORD.
   05 PR-PART-NBR              PIC X(05).
   05 PR-PROD-LINE             PIC X(02).
   05 PR-PART-DESC             PIC X(30).
   05 PR-WHOLESALE-PRICE       PIC 9(04)V99.
   05 PR-RETAIL-PRICE          PIC 9(04)V99.
WORKING-STORAGE SECTION.

01 DUMMY              PIC X(01).

01 CONTROL-FIELDS.
   05 ANSWER          PIC X(01).
      88 QUIT                    VALUE "N" "n".
      88 DATA-OK                 VALUE "Y" "y".
   05 MAINT-ANSWER    PIC X(01).
      88 MORE-UPDATES      VALUE "Y" "y".
      88 STOP-UPDATES      VALUE "N" "n".
      88 VALID-ANSWER      VALUE "Y" "y" "N" "n".
   05 ERROR-FLAG      PIC X(01).

*
*  DATE CONVERSION FIELDS
*
01 WORK-DATE.
   05 WD-YR          PIC X(02).
   05 WD-MO          PIC X(02).
   05 WD-DA          PIC X(02).

01  SCREEN-DATE.
   05 SD-MO          PIC X(02).
   05 SD-SLASH-1     PIC X(01)   VALUE "/".
   05 SD-DA          PIC X(02).
   05 SD-SLASH-2     PIC X(01)   VALUE "/".
   05 SD-YR          PIC X(02).

*
*  DATA FIELDS FOR THE DISPLAY ROUTINE
*
01 DISPLAY-FIELDS.
   05 DF-WHOLESALE-PRICE      PIC Z,ZZ9.99.
   05 DF-RETAIL-PRICE         PIC Z,ZZ9.99.
*
*  DATA FIELD  FOR THE RECORD DISPLAY ROUTINE
*
01  REQ-PART-NBR               PIC X(05).
*
*  DATA FIELDS FOR THE PART DISPLAY ROUTINE
*
01 PART-DATA.
   05 PD-PART-NBR              PIC X(05).
   05 PD-PROD-LINE             PIC X(02).
   05 PD-PART-DESC             PIC X(30).
   05 PD-WHOLESALE-PRICE       PIC 9(04)V99.
   05 PD-RETAIL-PRICE          PIC 9(04)V99.
*
*  DATA FIELDS FOR THE FILE MAINTENANCE ROUTINE
*
01 TRANS-DATA.
   05 TD-TRANS-TYPE       PIC X(01).
      88 NEW-RECORD       VALUE "A" "a".
      88 CHANGE-RECORD    VALUE "C" "c".
      88 DELETE-RECORD    VALUE "D" "d".
      88 VALID-TRAN    VALUE "A" "a" "C" "c" "D" "d".
   05 TD-PART-NBR         PIC X(05).
   05 TD-PROD-LINE        PIC X(02).
   05 TD-PART-DESC        PIC X(30).
   05 TD-WHOLESALE-PRICE  PIC 9(04)V99.
   05 TD-RETAIL-PRICE     PIC 9(04)V99.

LINKAGE SECTION.

01 METHOD-REQUEST              PIC X(05).

PROCEDURE DIVISION USING METHOD-REQUEST.

1000-MAIN.
   EVALUATE TRUE
      WHEN METHOD-REQUEST = "DISP "
         PERFORM D1000-DISPLAY
      WHEN METHOD-REQUEST = "MAINT"
         PERFORM M1000-MAINTENANCE
   END-EVALUATE.

1001-EXIT-PROGRAM.
   EXIT PROGRAM.

D1000-DISPLAY SECTION.

D1000-DISPLAY-MAIN-CONTROL.
   PERFORM D2000-INITIALIZATION.
   PERFORM D3000-PROCESS-USER-REQUEST
      UNTIL QUIT.
   PERFORM D6000-TERMINATION.
   GO TO D9000-EXIT.

D2000-INITIALIZATION.
   OPEN I-O          PRICE-FILE.
   MOVE "Y" TO ANSWER.
   ACCEPT WORK-DATE FROM DATE.
   MOVE WD-MO  TO  SD-MO.
   MOVE WD-DA  TO  SD-DA.
   MOVE WD-YR  TO  SD-YR.

D3000-PROCESS-USER-REQUEST.
   PERFORM D4000-DISPLAY-REQUEST-SCREEN.
   PERFORM D8000-READ-PRICE-FILE.
   IF PRICE-RECORD NOT EQUAL TO SPACES
      PERFORM D5000-DISPLAY-PRICE-RECORD.

D4000-DISPLAY-REQUEST-SCREEN.
   DISPLAY SPACES              LINE  1 COL  1.
   DISPLAY SCREEN-DATE         LINE  1 COL  1.
   DISPLAY "WALCABA CORPORATION"LINE  1 COL 31.
   DISPLAY "PF 1.2"            LINE  1 COL 73.
   DISPLAY "PRICE FILE SYSTEM, DISPLAY A PRICE
     RECORD"                   LINE  3 COL 20.
   DISPLAY "ENTER PART NUMBER: "LINE  5 COL 05.
   ACCEPT PR-PART-NBR          LINE  5 COL 25.

D5000-DISPLAY-PRICE-RECORD.
   MOVE PRICE-RECORD   TO  PART-DATA.
   CALL "PRPC210" USING PART-DATA.
   DISPLAY "DISPLAY ANOTHER PART? (Y/N): "
                               LINE 11 COL  5.
   ACCEPT ANSWER               LINE 11 COL 40.

D6000-TERMINATION.
   CLOSE PRICE-FILE.

D8000-READ-PRICE-FILE.
   READ PRICE-FILE
      INVALID KEY
   MOVE SPACES TO PRICE-RECORD
   DISPLAY "RECORD NOT FOUND "  LINE 13 COL  5
   DISPLAY "HIT ENTER TO CONTINUE" LINE 14 COL  5
   ACCEPT  DUMMY.

D9000-EXIT.
   EXIT.

M1000-MAINTENANCE SECTION.

M1000-MAINTAIN-MAIN-CONTROL.
   PERFORM M2000-INITIALIZATION.
   PERFORM M3000-PROCESS-MAINT-REQUEST
      UNTIL STOP-UPDATES.
   PERFORM M6000-TERMINATION.
   GO TO M9000-EXIT.

M2000-INITIALIZATION.
   OPEN I-O   PRICE-FILE.
   MOVE "N"    TO ERROR-FLAG.
   MOVE SPACE  TO  MAINT-ANSWER.

   ACCEPT WORK-DATE FROM DATE.
   MOVE WD-MO  TO  SD-MO.
   MOVE WD-DA  TO  SD-DA.
   MOVE WD-YR  TO  SD-YR.

M3000-PROCESS-MAINT-REQUEST.
   CALL "PRPC220" USING TRANS-DATA.
   PERFORM M4000-PROCESS-TRANSACTION.
   MOVE SPACE TO MAINT-ANSWER.
   PERFORM M5000-GET-CONTINUE-RESPONSE
      UNTIL VALID-ANSWER.

M4000-PROCESS-TRANSACTION.
   EVALUATE TRUE
      WHEN NEW-RECORD
         PERFORM M4100-WRITE-NEW-RECORD
      WHEN CHANGE-RECORD
         PERFORM M4200-CHANGE-RECORD
      WHEN DELETE-RECORD
         PERFORM M4300-DELETE-RECORD
      WHEN OTHER
         DISPLAY "INVALID UPDATE CODE"
                                  LINE 12 COL  5
         DISPLAY "HIT ENTER TO CONTINUE""
                                  LINE 13 COL  5
         ACCEPT DUMMY
   END-EVALUATE.

M4100-WRITE-NEW-RECORD.
   MOVE TD-PART-NBR TO  PR-PART-NBR.
   MOVE TD-PROD-LINETO  PR-PROD-LINE.
   MOVE TD-PART-DESCTO  PR-PART-DESC.
   MOVE TD-WHOLESALE-PRICE TO  PR-WHOLESALE-
   PRICE.
   MOVE TD-RETAIL-PRICE    TO  PR-RETAIL-PRICE
   WRITE PRICE-RECORD
      INVALID KEY
         DISPLAY SPACES          LINE 15 COL 5
         DISPLAY "RECORD ALREADY EXISTS"
         DISPLAY "NEW DATA NOT ENTERED INTO
         FILE"
         DISPLAY "HIT ENTER TO CONTINUE""
         ACCEPT DUMMY.

M4200-CHANGE-RECORD.
   MOVE TD-PART-NBR TO  PR-PART-NBR.
   MOVE "N" TO ERROR-FLAG.
   READ PRICE-FILE
```

```
      INVALID KEY MOVE "Y" TO ERROR-FLAG.
   IF ERROR-FLAG = "Y"
      DISPLAY "RECORD DOES NOT EXIST IN FILE"
      DISPLAY "HIT ENTER TO CONTINUE "
      ACCEPT DUMMY
   ELSE
      PERFORM M4210-APPLY-CHANGES
      REWRITE PRICE-RECORD
         INVALID KEY DISPLAY "INTERNAL LOGIC ERROR"
         DISPLAY "HIT ENTER TO CONTINUE "
         ACCEPT DUMMY.

M4210-APPLY-CHANGES.
   IF TD-PROD-LINE    NOT EQUAL SPACES
      MOVE TD-PROD-LINETO  PR-PROD-LINE.
   IF TD-PART-DESC    NOT EQUAL SPACES
      MOVE TD-PART-DESCTO  PR-PART-DESC.
   IF TD-WHOLESALE-PRICE NOT EQUAL ZERO
      MOVE TD-WHOLESALE-PRICE TO PR-WHOLESALE-PRICE.
   IF TD-RETAIL-PRICE NOT EQUAL ZERO
      MOVE TD-RETAIL-PRICE    TO  PR-RETAIL-PRICE.

M4300-DELETE-RECORD.
   MOVE TD-PART-NBR TO  PR-PART-NBR.
   DELETE PRICE-FILE RECORD
      INVALID KEY DISPLAY "RECORD NOT PRESENT IN FILE"
         DISPLAY "HIT ENTER TO CONTINUE "
         ACCEPT DUMMY.

M5000-GET-CONTINUE-RESPONSE.
   DISPLAY SPACES                LINE  1 COL  1.
   DISPLAY SCREEN-DATE           LINE  1 COL  1.
   DISPLAY "WALCABA CORPORATION" LINE  1 COL 31.
   DISPLAY "PF 1.1"              LINE  1 COL 73.
   DISPLAY "PRICE FILE TRANSACTION INPUT"
                                 LINE  3 COL 27.
   DISPLAY "MAINTAIN ANOTHER PRICE RECORD? (Y/N)"
                                 LINE  5 COL  5.
   ACCEPT MAINT-ANSWER.

M6000-TERMINATION.
   CLOSE PRICE-FILE.

M9000-EXIT.
   EXIT.
```

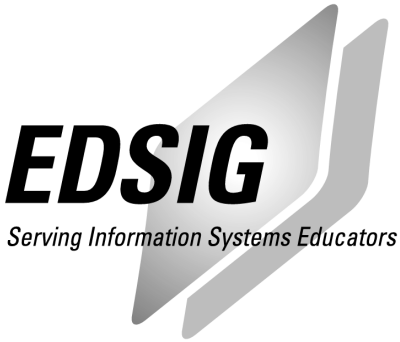
```
------------------------------------------------
IDENTIFICATION DIVISION.

PROGRAM-ID. PRPC210.
AUTHOR.
*
*  SUBPROGRAM TO CONTROL THE PART DISPLAY
SCREEN OBJECT
*
ENVIRONMENT DIVISION.

DATA DIVISION.

WORKING-STORAGE SECTION.
*
*  DATE CONVERSION FIELDS
*
01 WORK-DATE.
   05 WD-YR              PIC X(02).
   05 WD-MO              PIC X(02).
   05 WD-DA              PIC X(02).

01 SCREEN-DATE.
   05 SD-MO              PIC X(02).
   05 SD-SLASH-1         PIC X(01)   VALUE "/".
   05 SD-DA              PIC X(02).
   05 SD-SLASH-2         PIC X(01)   VALUE "/".
   05 SD-YR              PIC X(02).
*
*  DATA FIELDS FOR THE DISPLAY ROUTINE
*
01  DISPLAY-FIELDS.
   05 DF-WHOLESALE-PRICE          PIC Z,ZZ9.99.
   05 DF-RETAIL-PRICE             PIC Z,ZZ9.99.

LINKAGE SECTION.

01 PART-DATA.
   05 PR-PART-NBR                 PIC X(05).
   05 PR-PROD-LINE                PIC X(02).
   05 PR-PART-DESC                PIC X(30).
   05 PR-WHOLESALE-PRICE          PIC 9(04)V99.
   05 PR-RETAIL-PRICE             PIC 9(04)V99.
```

```
PROCEDURE DIVISION USING PART-DATA.
1000-MAIN.
   PERFORM 2000-INITIALIZATION.
   PERFORM 3000-DISPLAY-PRICE-RECORD.

1001-EXIT-PROGRAM.
   EXIT PROGRAM.

2000-INITIALIZATION.
   ACCEPT WORK-DATE FROM DATE.
   MOVE WD-MO  TO  SD-MO.
   MOVE WD-DA  TO  SD-DA.
   MOVE WD-YR  TO  SD-YR.

3000-DISPLAY-PRICE-RECORD.
   DISPLAY SPACES                LINE  1 COL  1.
   DISPLAY SCREEN-DATE           LINE  1 COL  1.
   DISPLAY "WALCABA  CORPORATION" LINE  1 COL 31.
   DISPLAY "PF 1.2"              LINE  1 COL 73.
   DISPLAY "PRICE FILE SYSTEM, DISPLAY A PRICE
   RECORD"                       LINE  3 COL 20.
   DISPLAY "PART NUMBER: "       LINE  5 COL  5.
   DISPLAY "PRODUCT LINE CODE: "LINE  5 COL 35.
   DISPLAY "PART DESCRIPTION: " LINE  7 COL  5.
   DISPLAY "WHOLESALE PRICE: "  LINE  9 COL  5.
   DISPLAY "RETAIL PRICE:"      LINE  9 COL 45.
   MOVE PR-WHOLESALE-PRICE TO  DF-WHOLESALE-PRICE.
   MOVE PR-RETAIL-PRICE TO  DF-RETAIL-PRICE.
   DISPLAY PR-PART-NBR           LINE  5 COL 27.
   DISPLAY PR-PROD-LINE          LINE  5 COL 57.
   DISPLAY PR-PART-DESC          LINE  7 COL 27.
   DISPLAY DF-WHOLESALE-PRICE    LINE  9 COL 27.
   DISPLAY DF-RETAIL-PRICE       LINE  9 COL 60.

------------------------------------------------
IDENTIFICATION DIVISION.

PROGRAM-ID. PRPC220.
AUTHOR.
*
*  THIS SUBPROGRAM CONTROLS THE TRANSACTION
   SCREEN OBJECT
*
ENVIRONMENT DIVISION.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 CONTROL-FIELDS.
   05 ANSWER             PIC X(01).
      88 QUIT            VALUE "N" "n".
      88 DATA-OK         VALUE "Y" "y".
   05 DUMMY PIC X.
*
*  DATE WORK FIELDS
*
01 SCREEN-DATE.
   05 SD-MO              PIC X(02).
   05 SD-SLASH-1         PIC X(01)   VALUE "/".
   05 SD-DA              PIC X(02).
   05 SD-SLASH-2         PIC X(01)   VALUE "/".
   05 SD-YR              PIC X(02).

01  WORK-DATE.
   05 WD-YR              PIC X(02).
   05 WD-MO              PIC X(02).
   05 WD-DA              PIC X(02).
*
*  DATA FIELDS FOR THE DISPLAY ROUTINE
*
01 DISPLAY-FIELDS.
   05 DF-WHOLESALE-PRICE          PIC Z,ZZ9.99.
   05 DF-RETAIL-PRICE             PIC Z,ZZ9.99.

LINKAGE SECTION.
*
*  DATA FIELDS FOR THE FILE MAINTENANCE ROUTINE
*
01 TRANS-DATA.
   05 TD-TRANS-TYPE               PIC X(01).
      88 NEW-RECORD               VALUE "A" "a".
      88 CHANGE-RECORD            VALUE "C" "c".
      88 DELETE-RECORD            VALUE "D" "d".
      88 VALID-TRAN       VALUE "A" "a" "C" "c" "D"
"d".
   05 TD-PART-NBR                 PIC X(05).
   05 TD-PROD-LINE                PIC X(02).
   05 TD-PART-DESC                PIC X(30).
   05 TD-WHOLESALE-PRICE          PIC 9(04)V99.
   05 TD-RETAIL-PRICE             PIC 9(04)V99.

PROCEDURE DIVISION USING TRANS-DATA.

1000-MAIN.
   PERFORM 2000-INITIALIZATION.
   PERFORM 3000-GET-MAINT-REQUEST.

1001-EXIT-PROGRAM.
   EXIT PROGRAM.

2000-INITIALIZATION.
   ACCEPT WORK-DATE FROM DATE.
   MOVE WD-MO  TO  SD-MO.
   MOVE WD-DA  TO  SD-DA.
   MOVE WD-YR  TO  SD-YR.

3000-GET-MAINT-REQUEST.
   MOVE "N" TO ANSWER.
   MOVE SPACES TO TRANS-DATA.
   PERFORM 3100-GET-TRANS
      UNTIL DATA-OK.

3100-GET-TRANS.
   DISPLAY SPACES                LINE  1 COL  1.
   DISPLAY SCREEN-DATE           LINE  1 COL  1.
   DISPLAY "WALCABA CORPORATION"LINE  1 COL 31.
   DISPLAY "PF 1.1"              LINE  1 COL 73.
   DISPLAY "PRICE FILE TRANSACTION INPUT"
                                 LINE  3 COL 27.
   DISPLAY "PART NUMBER: "       LINE  5 COL  5.
   DISPLAY "PRODUCT LINE CODE: "LINE  5 COL 35.
   DISPLAY "PART DESCRIPTION: " LINE  7 COL  5.
   DISPLAY "WHOLESALE PRICE: "  LINE  9 COL  5
   DISPLAY "RETAIL PRICE:"      LINE  9 COL 45.
   ACCEPT TD-PART-NBR           LINE  5 COL 27.
   ACCEPT TD-PROD-LINE          LINE  5 COL 57.
   ACCEPT TD-PART-DESC          LINE  7 COL 27.
   ACCEPT DF-WHOLESALE-PRICE    LINE  9 COL 27.
   ACCEPT DF-RETAIL-PRICE       LINE  9 COL 60.
   MOVE DF-WHOLESALE-PRICE TO TD-WHOLESALE-PRICE.
   MOVE DF-RETAIL-PRICE    TO  TD-RETAIL-PRICE.
   DISPLAY "ENTER TRANSACTION TYPE (A/C/D): "
                                 LINE 11 COL  5.
   ACCEPT TD-TRANS-TYPE          LINE 11 COL 40.

   DISPLAY "IS THIS DATA CORRECT? (Y/N)? "
                                 LINE 13 COL  5.
   ACCEPT ANSWER                 LINE 13 COL 35.

IF NOT VALID-TRAN
   DISPLAY "INVALID ACTION CODE, RE-ENTER"
      LINE 15 COL 10
   DISPLAY "HIT ENTER TO CONTINUE "
      LINE 16 COL 10
   ACCEPT DUMMY
   MOVE "N" TO ANSWER.
```


Gerald P. Marquis
Area of Information Systems and Quantitative Sciences
College of Business Administration
Texas Tech University
Lubbock, TX 79404-2101
(806) 797-0540
e-mail: odmrq@coba2.ttu.edu

Gerald P. Marquis has over 15 years of industrial experience and over 15 years of teaching experience at the undergraduate and graduate levels. He is a Ph.D. candidate at Texas Tech University.

Information Systems & Computing
Academic Professionals



STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the Journal of Information Systems Education have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

ISSN 1055-3096