CBA · NAU
College of Business Administration
Northern Arizona University
Box 15066
Flagstaff AZ 86011

Something Lost – Something Gained: From COBOL to Java to C# in Intermediate Programming Courses

Working Paper Series 03-05— May 2003

John D. Haney
Professor of CIS
College of Business Administration
Northern Arizona University
Box 15066
Flagstaff, AZ 86011-5066
(928) 523-7352
john.haney@nau.edu

Something Lost – Something Gained: From COBOL to Java to C# in Intermediate Programming Courses

CBA · NAU
College of Business Administration
Northern Arizona University
Box 15066
Flagstaff AZ 86011

John D. Haney

INTRODUCTION

For many years COBOL has been the primary programming language in business-oriented programming courses for both the intermediate and advanced courses. At the core of any business information system are the creation, processing, and retrieval of data. This is the fundamental model of Input-Process-Output. An additional component is that of data storage, either in flat sequential files or in databases. The type of programs that work with data fit into three general categories: data creation, data processing, and data retrieval. Data retrieval falls into two key classifications: reports (hard copy) and screen displays (soft copy). For many years, the primary language used in the development of business information systems has been COBOL, which was specifically designed to manipulate data. Therefore, the statements and syntax of the language lend themselves easily and intuitively to the processes of data processing. So the use of COBOL in the intermediate and advanced programming courses was a natural decision.

COBOL for many years was command-line oriented; thus, the User Interfaces were not graphical. With the advent of Visual Basic, for business information systems, many user interfaces were developed in VB that dovetailed with legacy COBOL systems. This made the use of VB in the introductory programming course and COBOL in the remaining programming courses an easy decision. With the passage of time a graphical user interface (GUI) was incorporated into COBOL, such as MicroFocus COBOL. This added the dimension of a GUI into the intermediate and advanced courses. However, what remained at the core of the intermediate and advanced courses was a strong focus on data processing, either in sequential files or with databases.

Java has brought a different focus. First, object-orientation came with Java, since Java is a completely object-oriented language. This has been a positive change. Object-orientation must be included in the curriculum, and the increasing use of Java in the workplace makes it a natural language to include in the curriculum. However, Java comes with some costs. The primary cost is the loss of focus on data processing in the intermediate programming courses. It is not that data processing is not possible in Java but the procedures are not as straightforward as they are in COBOL. The added focus of objects in Java diminishes the time available for data processing. A secondary cost of using Java in the intermediate programming course is the graphical user interface. A GUI is available in Java, but the use of the Advanced Windowing Toolkit (AWT) and Swing is not as intuitive as the development environment in Visual Basic or even the Dialog system in MicroFocus COBOL. These costs take time away from data processing to the extent that data processing has been almost eliminated from the intermediate programming course. This has been a substantial cost because data processing is the core of all business information systems.

So, the resulting challenge is to provide content in intermediate programming courses that consists of data processing, object-orientation, and graphical user interfaces. The solution was found in Microsoft's new .NET development environment. The programming languages in the .NET suite are all object-oriented; the graphical user interface is a continuance of the Visual Basic interface; and the data file usage is essentially a combination of Visual Basic and Java in an intuitive context.

A solution to this challenge is presented by comparing the course content of intermediate programming courses using COBOL, then Java, and currently C#. Several comparisons could be made, however, for brevity the statements for sequential data file processing in each language are compared.

INTERMEDIATE PROGRAMMING COURSES USING COBOL

Since intermediate programming courses are an extension of introductory programming courses, there is an assumption of knowledge that includes the fundamentals of programming such as: data definitions, assignment of data values, arithmetic operations, logical operations, decision making, looping, and the use of arrays. Along with

the fundamentals of programming there is also a supposition that introductory programming courses include a certain amount of development of graphical user interfaces, and the use of objects.

The content of intermediate programming courses focused very heavily on data file processing when COBOL was used. This included flat files with sequential processing, and indexed-sequential files with random processing. Examples of the type of programs that were required of the students to write are as follows: 1) a file creation program, where data is input from the console and then written to a sequential output file; 2) a retrieval type of program, where data is read from a sequential input file, and then a report is generated; 3) a data validation program, that is similar to the first project with the addition of data validation of the data entered. If the validation criteria are met then the record is written to the output file; 4) a project that uses a graphical user interface; 5) a project where the data is read into a table (array) and then a report is generated with sub-totals; 6) a complete update to an indexed-sequential file where records are added, changed, deleted, or queried from the file; 7) and a program where a sub-program is created and called from a main program.

As evidenced from the above list of projects, there was a very strong emphasis on data file processing, in addition to the use of graphical user interfaces and the use of arrays.

Intermediate Programming Courses Using Java

The content of intermediate programming courses changed dramatically when COBOL was dropped and Java was added. Because of the nature of Java, object-orientation became the primary focus, along with the some use of graphical user interfaces, and some data file usage. Depending on the instructor, there was a trade-off between a focus on graphical user interfaces and data file processing. Examples of the type of programs that the students were required to write are as follows: 1) a review program, that provides an overview of objects: classes, data members, and data methods, with a focus on parameter passing; 2) arithmetic processing that uses objects; 3) the use of a graphical user interface; 4) a project that focuses on inheritance; 5) a project that emphasizes polymorphism; 6) a project that uses arrays; 7) sequential data file usage; 8) and a project that interacts with a database.

The above list of projects shows a strong emphasis on object-orientation, the use of arrays, some data file and database processing, and some graphical user interfaces.

Intermediate Programming Courses Using C#

The change of the programming language of intermediate programming courses from Java to C# combines the focus of data processing when COBOL was used, and object-orientation when using Java, in addition to the use of graphical user interfaces. The intent was to bring back a heavy emphasis on data file processing and the inclusion of graphical user interfaces, without the loss of object-orientation. Examples of the type of programs that are required of the students to write are as follows: 1) a review program, that provides an overview of objects: classes, data members, and data methods, with a focus on parameter passing; 2) a project that focuses on inheritance; 3) a program that emphasizes polymorphism; 4) a project that uses a graphical user interface for data entry and validation, and then writes to a sequential output data file; 5) a project with a graphical user interface for user interaction, that reads from a fixed length record sequential file and writes to a comma delimited output sequential file. The output is conditionally written based on selection criteria; 6) a project that interacts with a relational database, that retrieves data from the database and displays the data one row at a time into objects (textboxes) on a graphical user interface. This program makes use of arrays by placing the contents of the selected dataset from the database into the array before displaying the data on the GUI; 7) a project that interacts with a relational database, that retrieves data from the database and displays the entire dataset into a data grid on a graphical user interface. This program allows for making changes to the dataset via the data grid; 8) and a project that provides for a full update to a database table by adding rows to the database table, changing or deleting rows, and querying a single row.

The above list of projects indicates a strong emphasis on data file processing, in addition to a substantial use of graphical user interfaces, the use of arrays, and retention of object-orientation.

COMPARISON OF I/O STATEMENTS USING COBOL, JAVA, AND C#

Following is a comparison of the input and output statements of COBOL, Java, and C#, by showing the differences of opening a file for both input and output, reading a record from an input file, writing a record to an output file, and closing a file.

Opening a File for Input and for Output in COBOL

First the data file on disk is assigned to a file-reference that becomes the internal reference that refers to the external data file. This is accomplished with the SELECT and FD statements.

```
SELECT FILE-NAME Assign to Disk Organization is Line Sequential.
FD        FILE-NAME Value of File-ID is “C:/somewhere/something.dta”.
01        RECORD-NAME  PIC X(--).
```

The statements for opening an input file and an output file in COBOL are simple and straightforward. The verb OPEN describes the action, followed by the file-reference, and then the open mode, either input or output.

```
OPEN Input FILE-NAME.
OPEN Output FILE-NAME.
```

Opening a File for Input and for Output in Java

Opening a file for input and output in Java is much more complicated than in COBOL. Three classes are involved in opening a file for input – InputStream, File, and BufferedReader. First, an instance of the File class is created that references the location and name of the file on disk. The name of the File instance is given the name *inputFile*. Then the InputStream class is wrapped around the File object, and given the name *input*. Finally, an instance of the BufferedReader class is wrapped around InputStream, which is given the name *br*. The code for opening the same sequential file that was opened in COBOL above is as follows:

```
// open the input data file
File inputFile = new File (“C:/somewhere/something.dta”);
InputStream input = new FileInputStream (inputFile);
BufferedReader br = new BufferedReader(new InputStreamReader(input));
```

The process for opening a file for output is similar. First, an instance of the Writer class is created, which is named *output*. Next, the instance of the Writer class is wrapped around the FileWriter class, which references the location and name of the file on disk. Then the BufferedWriter class is wrapped around the Writer class and given an instance name of *bw*. The InputStream object is then wrapped around the File object. Finally, the PrintWriter class is wrapped around the BufferedWriter class, and given the name *bw*. The code for opening a sequential file in Java is as follows:

```
// open the output data file
Writer output;
output = new FileWriter (“C:/somewhere/something.dta”);
BufferedWriter bw = new BufferedWriter( output );
PrintWriter pw = new PrintWriter(bw);
```

The open statements must reside within a try block.

Opening a File for Input and for Output in C#

Opening a file for input and output in C# is a much more simplified process than in Java. First, variables must be defined for the StreamReader and/or StreamWriter depending on whether the file is being opened for input or output.

```
private StreamReader input;
private StreamWriter output;
```

C# does not require that the open statement reside within a try block, although it is recommended. Even within a try block the statement to open a sequential data file in C# for input is very straightforward.

```
try  {  input = new StreamReader (“C:/somewhere/something.dta”, false);  }
```

The statement to open a sequential data file for output is very similar to the statement that opens a file for input.

```
try  {  output = new StreamWriter (“C:/somewhere/something.dta”, false);  }
```

Reading a Record from a File in COBOL

Reading a record from a file in COBOL specifies the file-reference and an action that should be taken when the end of the file is reached.

```
READ    FILE-NAME at end Move “end” to END-FLAG.
```

Reading a Record from a File in Java

Reading a record from a sequential file in Java is much more straightforward than opening the file in Java, but more complicated than reading a record in COBOL. The readLine method of the BufferedReader class reads a record from the data file into a String variable, in this case named *inputLine*. The while loop is used to control for the end of file condition, so the while loop must be embedded within a try block.

```
// read from the data file until no more data

while (( inputLine = br.readLine() ) != null)          {              } // end of loop
```

Reading a Record from a File in C#

Reading a record from a sequential file in C# is identical to the process in Java, with the exception that the StreamReader class is used instead of the BufferedReader class.

```
// read from the data file until no more data

while (( inputLine = StrteamReader.readLine() ) != null)      {          } // end of loop
```

Writing a Record to a File in COBOL

Writing a record to a file in COBOL specifies the name of the record.

```
WRITE RECORD-NAME.
```

Writing a Record to a File in Java

Writing a record to a sequential file in Java is also fairly straightforward. The println method of the PrintWriter class writes a record to the data file. In this example, a comma-delimited file is created, so commas are placed between each of the fields.

```
//  write the record

pw.println (ID + "," + FirstName + "," + LastName);
```

Writing a Record to a File in C#

Writing a record to a sequential file in C# is very similar to writing a record in Java. The only difference is that the writeLine method of the StreamWriter class is used instead of the println method of the PrintWriter class, which is wrapped around the BufferedWriter class.

```
//  write the record

output.WriteLine (ID + "," + FirstName + "," + LastName);
```

Closing a File in COBOL

Closing a file in COBOL specifies the file-reference.

```
CLOSE FILE-NAME.
```

Closing a File in Java

To close a file, either the input or output, in Java the instances of the objects should be closed in reverse order from how they were wrapped, closing the outer class first and the inner class last.

```
// close the input streams

br.close( );

input.close( );


// close the output streams

pw.close();
```

```
bw.close( );
output.close( );
```

Closing a File in C#

The process to close a file in C# is very similar to the process of closing a file in COBOL. It merely entails closing either the StreamReader or StreamWriter classes.

```
// close the input file
input.close( );


// close the output file
output.close( );
```

SUMMARY AND CONCLUSIONS

The decision pertaining to which programming language to use in intermediate programming courses has become problematic. For many years COBOL was the primary language for developing business information systems, so the decision as to which programming language to use was an easy one. With the widespread use of Java, including the development of web based information systems, COBOL has been dropped and Java added to many intermediate programming courses. This decision has produced both costs and benefits. The primary benefit has been the addition of object-orientation to the curriculum. The main cost has been the diminished emphasis on data file processing. Another cost has been a more awkward graphical user interface. By changing from Java to C#, the emphasis on objects has been retained, and the focus on data file processing has been greatly enhanced with a much greater weight given to graphical user interfaces.

The challenge regarding which language to use in intermediate business programming courses has been met by using C# .NET. This development tool provides an environment that combines an easy to use graphical user interface, a fully object-oriented language, and data processing statements that are fairly intuitive. What was lost by going from COBOL to Java in intermediate programming courses has been regained by going to C#.

REFERENCES

Bradley, Julia Case & Millspaugh, Anita C. (1999). Programming in Visual Basic 6.0. San Francisco, CA.: McGraw-Hill-Irwin.

Bradley, Julia Case & Millspaugh, Anita C. (2003). Programming in Visual Basic .NET. San Francisco, CA.: McGraw-Hill-Irwin.

Bradley, Julia Case & Millspaugh, Anita C. (2003). Advanced Programming using Visual Basic .NET. San Francisco, CA.: McGraw-Hill-Irwin.

Deitel, H.M., Deitel, P.J., ListField, J.A., Nieto, T.R., Yaeger, C.H. & Zlatkina, M. (2003). C#" A Programmer's Introduction. Upper Saddle River, N.J.: Prentice Hall.

Farrell, Joyce. (2002). Visual C# .NET. Canada: Course Technology.

Grauer, Robert T., Villar, Carol Vasquez & Buss, Arthur R. (1998). COBOL: From Micro to Mainframe. Upper Saddle River, N.J.: Prentice Hall.

Lorents, Alden C. (2000). Elements of Dialog System Using Micro Focus Net Express. Orinda, CA. Object-Z Publishing.

Price, Wilson. (1997). Elements of Object-Oriented COBOL. Orinda, CA. Object-Z Publishing.

Sharp, John & Jagger, Jon. (2002). Visual C# .NET. Redmond, WA.: Microsoft.

Staggered, Andrew C. Jr. (1999). Java for Computer Information Systems. Upper Saddle River, N.J.: Prentice Hall.

Stern, Nancy & Stern, Robert A. (2000). Structured COBOL Programming. New York, N.Y.: Wiley.