

<http://www.cs.arizona.edu/classes/cs460/fall08>

## Program #1: Text to Binary File Conversion

*Due Date: September 9<sup>th</sup>, 2008, at the beginning of class*

**Overview:** In the not-to-distant future, you will be writing a program to create an index on a binary file. I could just give you the binary file, or merge the two assignments, but after a long summer, creating the binary file from a generically-formatted text file makes for a nice “shake off the rust” assignment, plus it provides a gentle (?) introduction to binary file processing.

A binary file is just a file that contains information in the same format in which the information is held in memory. (In a standard text file, all information is stored as ASCII or UNICODE characters.) As a result, binary files are faster and easier for a program to read and write than are text files. As long as the file doesn't need to be read by people or ported to a different type of system, binary files are often the best way to store program information.

For this program, we have lines of data items that we need to store, and each line's items need to be stored as a group (in a database, such a group is called a *record*). Making this happen in Java requires a bit of effort; Java wasn't originally designed for this sort of task. By comparison, “systems” languages like C and C++ can interact more directly with the operating system and provide more convenient file I/O. On the class web page you'll find a sample Java binary file I/O program, which should help you figure out what needs to be done.

**Assignment:** Linked to the class web page is a file named `2008_FE_guide.csv`. This is a file consisting of over 1200 lines of data on automobile fuel economy for the 2008 model year, courtesy of the U.S. Department of Energy and the U.S. Environmental Protection Agency.

Using Java, write a program that:

1. Creates a binary version of the text file
2. Closes that binary file
3. Re-opens the same binary file (for read access)
4. Reads and prints to the screen the content of the first five records of data and the last five records of data in the binary file, along with some statistics (see the details in the Output section, below).

**Data:** The data file name and location are given in the “Assignment” section, above. Your program can read the file directly out of that directory; just use the complete path name of the file when you open the file. There's no reason to waste disk space by making a copy of the file in your account.

Each of the 1200+ lines in the file contain 31 fields (pieces) of information, separated by commas (thus the “.csv” file extension: Comma Separated Values). Here is a sample line:

```
TWO SEATERS,ASTON MARTIN,V8 VANTAGE,4.3,8,Auto(S6),R,12,13,20,15,16.1945,27.4018,19.8474,P,G,,,,,,,,,3002, ,4MODE ,4,1,6-Aug-07,DERIVED
```

The first line of the file has abbreviated field descriptions. Additional explanation of these fields is available from the file `2008_FE_guide.txt`, also available from the class web page. I've provided the URLs of the sources of this information below, and linked them to the class web page. Note that the line of descriptions is not to be stored in the binary file; we only want data stored in that file, not descriptions of the data (a.k.a. *metadata*).

(Continued...)

**Output:** Your program needs to create the binary file, of course. The required on-screen output is the (ASCII) content of the first five and last five lines of processed data, along with the total number of data records processed (don't count that first description line of the `.csv` file), and the Manufacturer and Car Line of both the car with the highest COMB MPG (GUIDE) and the car with the lowest COMB MPG (GUIDE). You are welcome and encouraged to add additional output if you have extra time and ambition (e.g., time required for the processing, mean fleet MPGs, etc.). The format of the output is up to you, but the output should be easy to read

**Hand In:** You are required to submit your completed program file(s) using the `turnin` facility on `lectura`. The submission folder is `cs460p1`. Instructions are available from the document of `turnin` instructions linked to the class web page. Name your main program source file `Prog1.java` so that we don't have to guess which files to compile, but feel free to split up your code over additional files if you feel that doing so is appropriate.

### Want to Learn More?

- <http://www.fueleconomy.gov/feg/download.shtml> — The page with links to the fuel economy data.
- <http://www.fueleconomy.gov/feg/epadata/Readme.txt> — The incomplete and sloppily-formatted descriptions of the data fields.

### Other Requirements and Hints:

- Don't "hard-code" values in your program if you can avoid it. For example, don't assume a certain number of records in the input file or the binary file. Your program should automatically adapt to simple changes, such as more or fewer lines in a file or changes to the file names or locations. For example, we may test your program with a file of just 3 lines. We expect that your program will handle this situation gracefully.
- Once in a while, a student will think that "create a binary file" means "convert all the data into the characters '0' and '1'." The binary I/O functions in Java will read/write the data in binary format automatically.
- Wondering how to efficiently get the last five records from a binary file? Remember: The Java API is your friend!
- Comment your code according to the style guidelines *as you write the code* (not an hour before class!). The requirements and some examples are available from: <http://www.cs.arizona.edu/~mccann/style.html>
- You can make debugging easier by using only a line or two of data from the `.csv` file for your initial testing. Try running the program on the complete file only when you can process that reduced data file successfully.
- Finally: Start early! File processing is often tricky. Also, after the long, relaxing summer, your programming skills are probably a little stale.