<u>Should you set up a Separate File (and/or Record) Class</u>

Kaminski / cs3310

Why would you choose to set up a separate file class (and possibly a separate record class) for a particular file? A class is a mechanism for "burying the details" for storing data about an object and the methods for manipulating that data – hiding such details from the rest of the program outside that class. It's also a way of organizing code into categories – labeling the category with a descriptive class name. It also facilitates method-naming, so the same descriptive names can be re-used repeatedly if 2 methods do the same function (e.g., ReadARec), but they operate on different kinds of objects (i.e., of different classes), and do so in perhaps somewhat different ways (e.g., different fields in different files' records, binary vs. text files, using StreamReader which returns a string vs. FileStream which returns a byteArray, fixed-length records vs. comma-separated values, etc.). A developer would generally consider setting up a separate file and/or record class for a particular file if:

- this needed to be done as a strictly ("letter-of-the-law") OOP application
- this was a large application where as much as possible needed "hiding" (in a class) and "labeling" (i.e., the class & method names)
- there were several files used in the one program, both of which had similar functionality, so separate classes would categorize their methods– e.g., InputFile.ReadARec and MasterFile.ReadARec
- individual fields in the file's record needed to be accessed from elsewhere in the program
- there was a certain amount of manipulation that needed to be done on the record's fields (vs. just moving aLine around) - so a class is a way to "bury these details" from the outside world, putting this manipulation in the attribute properties. This is generally the case
  - o when certain inputting/outputting commands are used, like C#'s FileStream which reads/writes byteArrays, which generally need to be converted to/from strings for use in the rest of the program – and since FileStream has no ReadLine or WriteLine methods (just Read and Write), the <CR><LF> may have to be stripped off or added, as appropriate
  - o when a random access file is used where the records are not already fixed-length records with fixed-length fields, and so conversion (padding/truncating) is needed before the records are written out (and further, FileStream is needed so as to use its Seek method, so there's some byteArray ← → string transformations handling)
  - o when a binary file is used, which often requires converting fields to other data types
  - o when a generic binary file is used which must be formatted to suit another language's or OS's standard format (e.g., the endian byte-order issue, string storage format issue, etc.) (and further, FileStream is generally needed, so there's the byteArray ← → string transformations handling)
- several different methods are used on the file's records - so a class is a way to collect those methods together
- several programs or other classes' methods need to access the file's methods - so a class could be shared, thus giving both programs access to the file's methods
- there might be changes to the nature of the data source in the future – i.e., instead of reading from the file, data might come from: a database, command-line user input, a GUI text box (or other GUI object) , a sensor, the network, etc. - so any changes would be done in the class, not in the Main program or calling module(s)

In conclusion, if an application pretty much only does these 4 things with a file,

1) open the file
2) reads (or writes) a record/line from (to) the file inside a loop
3) where the loop controller watches for EOF if it's an input file
4) then after the loop, closes the file

then setting up a class for that file may not be warranted.