Sequential Stream Processing

Setup and UserApp are CONTROLLERS. They don't do any (much) real work. Instead, they call public service methods in other classes to actually DO THE WORK, like:

- RawData or UiInput for input requests
- UiOutput for logging
- DataStorage and CodeIndex to insert or lookup appropriate data

Setup and UserApp both do basic sequential processing of their respective input streams. As controllers, they don't know where their stream of input is coming from – e.g., a file, an interactive user, a database, a bar-code scanner, a stream of input arriving from scraping the web, a series of touch-screen interactions from the user, etc.

Both of these processing the input (records) in PHYSICALLY SEQUENTIAL order. It is NOT the same as KEY SEQUENTIAL processing, in which the records would need to be accessible IN ORDER by some key – e.g., the file is sorted on id. This is NOT the case for RawData file or for TransData file.

RawData file and TransData file are both input streams of records, to Setup and UserApp respectively. You will NEVER HAVE THE ENTIRE FILE's DATA IN MEMORY AT ONCE - there is no need for this. Both Setup and UserApp need only a SINGLE INPUT RECORD's data in memory at one time. So the SequentialStreamProcessing algorithm (design pattern) is used. That is,

```
Get stream ready (e.g., file opened in class's constructor for OOP)
Loop til nothing more arriving from stream
            (i.e., a public boolean method/variable in the class which indicates "done")
{     1.   call a method in the input stream class to INPUT a SINGLE data set (record)
                              which READs & splits the record/line into fields
                    and does whatever other cleanup is needed
                              (though it does NOT anticipate what some other class will need,
                    like
                              DataStorage or CodeIndex, in terms of padding/trimming or
                              data type conversion)
                    AND also set the DONE variable/method to TRUE, as appropriate
   2    call a method in the output handler class to       PROCESS that SINGLE data set
                    (using public GETTERs from the input class as parameters to supply the
                              appropriate data to the HANDLER class)
                    [For A2, call appropriate method in BOTH DataStorage AND CodeIndex]

}
FinishUp with stream (file closed in class's finishUp method, for OOP)
```

Implementation NOTES:

- *The above shows a read/process loop structure. In your actual coding you might use a process/read loop structure with a priming pre-while read – depending on what "read" method you're using and how you're detecting EOF.*
- *There is never more than a single RawData record or a single TransData record in memory at once. So only a single object is needed for storing a RawData record or a TransData record. New records are stored in the SAME STORAGE SPACE, replacing the prior record since you only ever need 1 record (and its fields) at once. This speeds up the processing for large files because it avoids constructing/destructing a new objects for EVERY RECORD in the file. [Yes, this increases run-time efficiency at the expense of a pure OOP approach].*
- *You could have a separate class for data RECORDS rather than combining all FILE and all RECORD handling in a single class.*