CS3310 – Kaminski - Spring 2015 **CountriesOfTheWorld App 1.0**
Asgn 2 Project Specs

################################ OVERVIEW ################################

A2 is similar to A1 except that we now have:

1) The actual country data stored in a random access file, CountryData (which is indexed by CodeIndex, which is still implemented as a BST)
2) The overall project structure has changed:
   a. Setup, UserApp & PrettyPrint are now actual PROGRAMS rather than just callable methods.
   b. TestDriver PROGRAM calls these PROGRAMS to run them rather than the single project main calling those 3 public METHODS.
   c. The data FILES are now hidden inside their own OOP classses rather than being accessed directly from the main programs – so 4 new classes need to be added:
      RawData (for RawData),
      UiInput (for TransData),
      UiOutput (for Log),
      DataStorage (for the random access actual CountryData)
      [IndexBackup file is still hidden in CodeIndex class].

################################ 4 PROGRAMS ################################

EACH PROGRAM IN A SEPARATE FILE

[A program is a stand-alone, separately executable chunk of code which has its own main method, which is where the code starts executing.]
[BUT, a program can also be executed by another program – by calling its main method.]

1) **TestDriver** – the overall CONTROLLER which runs the other 3 programs multiple times. A2DemoSpecs will specify the exact order to run them, the number of times to run each, and the fileNameSuffix to pass in (i.e., for RawData?.csv and TransData?.txt). This program does NOT access any of the OOP classes.

2) **PrettyPrint** – displays the BOTH CountryData AND IndexBackup files to the Log file. It is not an OOP. It does NOT use any other class. It reads/writes to those files DIRECTLY (as if your CS1110 intern were doing this program).

3) **Setup** – the procedural CONTROLLER program which builds the EXTERNAL DataStorage (i.e., CountryData.txt file) and INTERNAL CodeIndex using the data in RawData (file).

4) **UserApp** – the procedural CONTROLLER program which processes transaction requests from the UiInput class (which comes from TransData?.txt file) using DataStorage and CodeIndex to get the appropriate data to send to UiOutput class (for writing to the Log file). The program contains a big switch statement (based on transCode: IN, DC, DI, SC, SI, AC, AI) to call the appropriate handler method in either (or both) DataStorage class or CodeIndex class. NOTE:
   IN needs to call both dataStorage.insert AND codeIndex.insert
   (with the appropriate parameters sent in)
   DC needs to call codeIndex.delete (using the code supplied)
   which returns the DRP
   AND THEN call dataStorage.delete (using that DRP as the id)
   DI needs to call dataStorage.delete (using the id supplied)
   which returns the code in the record
   AND THEN call codeIndex.delete (using that code)
   SC and AC display the ACTUAL DATA for A2, and NOT JUST THE DRP (id)

NOTES FOR SETUP & USERAPP

- *Any work on the objects (RawData file, Log file, CountryData file, TransData file, CodeIndex) is done TOTALLY within their classes rather than in Setup or UserApp itself. Setupand UserApp are just the "bosses" which requests public services (i.e., they calls public methods/constructors/getters/setters) from those classes.*
- *The controller code in Setup and in UserApp uses the INPUT STREAM PROCESSING "design pattern" for batch processing - i.e., loop til done {input 1 item, deal with it}*
- *Any status messages or responses to userRequests (from TransData) are sent to the UiOutput class's displayThis method (for writing to the Log file).*

########################## 5 SHARED OOP CLASSES ##########################

EACH CLASS IN A SEPARATE FILE

[These instantiable (OOP) classes store data (internally or externally) and provide public services They don't "DO ANYTHING" on their own (pretty much) – they only DO something when one of the controllers (Setup or UserApp) puts in a request for some service (i.e., calls a public method/constructor/getter/setter).]

1) **RawData** class – only used by Setup
This class handles EVERYTHING to do with RawData.csv file/records/fields including:
   - open the file (in the **constructor**)
   - close the file (in **finishUp** method)
   - read a record/line (in i**nput1Country** method) – which:
     changes done to true/false depending on EOF,
     cleans up that record/line,
     splits the line into fields,
     sets the instance variables for each relevant field
   - reports back whether it's EOF is true (in **done** method)
   - returns the value of a field (in relevant field's **getter**)

2) **DataStorage** class – used by both Setup and UserApp
This class handles EVERYTHING to do with the CountryData.txt file/records/fields including
   - open & close the file (in the **constructor** & **finishUp** methods)

- insert data record in the appropriate location in the file (in **insert** method) having first assembled the correct fields in the correct order with appropriate truncating/padding. All fields will be sent in as strings.
  NOTE: Probably need overloaded INSERT method – when it's called
  –by Setup, do NOT give a reassurance message to uiOut
  -by UserApp, DO give a reassurance message to uiOut
- delete the specified data record in the file (in **delete** method). The id will be sent in to specify the primary key.
- retrieve the specified data record (in **select** method). The id will be sent in to specify the primary key.
- retrieve All the data records in physical sequence (in **selectAll** method).

3) ***CodeIndex*** class - used by both Setup and UserApp
This class handles EVERYTHING to do with the internal code index, implemented as a BST. The class should be usable directly from A1's CodeIndex.java file, if you did that right. There will already be public methods for:

constructor, finishUp, insert, delete, retrieve, retrieveAll

NOTE: Probably need overloaded INSERT method – when it's called
–by Setup, do NOT give a reassurance message to uiOut
-by UserApp, DO give a reassurance message to uiOut

NOTE ON PASSING IN dataStorage OBJECT
*CodeIndex's retrieve and retrieveAll methods need to be able to access dataStorage's retrieve method (supplying it the DRP which is id, in the CountryData file). So the dataStorage object must be **passed IN** to those methods in this classes so they can call **dataStorage.retrieve** method.*

4) ***UiIn*** class – used by UserApp
This class handles EVERYTHING to do with TransData?.txt file. This includes:
- open & close the file (in **constructor& finishUp** methods)
- read a line (in **input1Request** method) from the transData file
  changes done to true/false depending on EOF,
  splits the line into the appropriate fields (as appropriate),
  sets the instance variables for each relevant field,
  echos the line to the log file
  (by calling UiOut's displayThis method)
- reports back whether it's EOF is true (in **done** method – when called)
- returns the value of a field (in relevant field's **getter** – when called)

5) ***UiOut*** class – used by Setup and UserApp (not PrettyPrint)
This class handles EVERYTHING to do with Log.txt file. This includes:
- open & close the file (in **constructor & finishUp** methods)
- write a line (in **displayThis** method) – assuming the caller has already constructed the exact line that needs to be printed

NOTES FOR PASSING IN uiOut OBJECT
- *The uiOut object is declared in Setup's main and again in UserApp's main. So these two can call uiOut.displayThis when they need to write a status message to the Log file.*
- *Certain methods in RawData, DataStorage, CodeIndex and UiIn classes need to be able to write to the Log file. So the uiOut object needs to be **passed IN** to the appropriate methods in these classes so they can call **uiOut.displayThis** method.*

############################## INPUT DATA FILES ##############################

RawData?.csv - record description

NOTE: .csv file** → **variable-length FIELDS → variable-length RECORDS

NOTE: Char fields are enclosed in single quotes – **your program code REMOVES THEM**

- ~~Extra characters for SQL-compatibility: INSERT INTO `Country` VALUES (~~
  ~~NOT USED IN THIS PROJECT~~
- code - 3 capital letters [uniquely identifies a country]
- id - 1- 3 digits [uniquely identifies a country]
- name - all chars (may contain spaces or special characters) [uniquely identifies a country]
- continent - one of: Africa, Antarctica, Asia, Europe, North America, Oceania, South America
- ~~region – NOT USED IN THIS PROJECT~~
- area (physical size of the country) - a positive integer
- ~~yearOfIndep – NOT USED IN THIS PROJECT~~
- population - a positive integer or 0 [which could be a very large integer]
- lifeExpectancy - a positive float with 1 decimal place
- ~~Rest of fields – NOT USED IN THIS PROJECT~~
- ~~Extra characters for SQL-compatibility:); - NOT USED IN THIS PROJECT~~
- <CR><LF>

^^

TransData?.txt description

One transaction per line (ends with <CR><LF>), starts with 2-char transCode, for example:

```
SC USA                                          (i.e., Select by code)
SI 44                                           (i.e., Select by id)
AC                                              (i.e., Select All by code)
AI                                              (i.e., Select All by id)
DC FRA                                          (i.e., Delete by code)
DI 44                                           (i.e., Delete by id)
IN WMU,240,West Mich Uni,Europe,123,4567,88.9  (i.e., Insert)
      (i.e., code,id,name,continent,area, population, lifeExpectancy)
```

########################## OUTPUT DATA FILES ##########################

CountryData.txt - record description

NOTE: direct address files → fixed-length RECORDS → fixed -length FIELDS

- code - 3 chars
- id – LEFT-FILL WITH 0's to make it 3 digits
- name – RIGHT-PAD WITH SPACES OR RIGHT-TRUNCATE to 18 chars
- continent - RIGHT-PAD WITH SPACES OR RIGHT-TRUNCATE to 13 chars
- area – LEFT-FILL WITH 0's to make it 8 digits (no embedded commas)
- population – LEFT-FILL WITH 0's to make it 10 digits (no embedded commas)
- lifeExpectancy – 4 chars (e.g., 78.4)
- <CR><LF>

NOTE: Who's doing the field's padding/truncating? Setters in DataStorage class

^^

Log.txt – description

NOTES:

- *You **MUST** use my **exact format/wording/spacing/alignment** as shown below ! ! ! !*
- *3 kinds of things written to the Log file (via uiOut.displayThis method):*
  - ***1)*** *Status messages- SAME AS A1*
  - ***2)*** *Transactions (equests echoed – SAME AS A1*
    *& responses written – **MUST WRITE ACTUAL DATA (7 FIELDS)***
  - ***3)*** *Output from PrettyPrint*
- *Data shown below are not necessarily accurate based on the actual RawData files. I'm just showing you the appropriate display format.*
- *Fill in the . . . portion of the AN and AI transaction responses and PrettyPrint.*
- *Transaction responses do NOT show storage location (i.e., SUBSCRIPT or RRN), they just show the data itself. BUT PrettyPrint DOES show storage location and the data*
- *Use appropriate formatters to display the data so it aligns:*
  - *RIGHT-justify numeric fields with embedded commas as appropriate*
  - *LEFT-justify char field*

Log – STATUS MESSAGES – appear AT THE APPROPRIATE TIMES

```
Use same messages as A1 but ADD FILE OPEN & CLOSE messages for
       CountryData file
```

NOTE: you MUST place code right after (except for closing Log file) the event it's reporting

Log – TRANSACTION PROCESSING

NOTE: SC/DC/IN do NOT have to print out # nodes visited for A2

```
SC CHN
   CHN 094 China             Asia          9,572,900 1,277,558,000 71.4
SI 94
   CHN 094 China             Asia          9,572,900 1,277,558,000 71.4
SC WMU
   invalid code
SI 599
   invalid id
AC
   CDE ID- NAME-------------- CONTINENT---- ------AREA ---POPULATION LIFE
   AFG 001 Afganistan         Asia             652,090    22,720,000 45.9
   . . .
   ZWE 204 Zimbabwe           Africa           390,757    11,669,000 37.8
AI
   CDE ID- NAME-------------- CONTINENT---- ------AREA ---POPULATION LIFE
   AFG 001 Afganistan         Asia             652,090    22,720,000 45.9
   . . .
   CHN 094 China              Asia           9,572,900 1,277,558,000 71.4
   . . .
   UMI 239 United States Mino Oceania              16             0  0.0
DC FRA
   ok, deleted (from code index)
   ok, deleted (from data storage)
DI 13
   ok, deleted (from data storage)
   ok, deleted (from code index)
IN SCT,240,Scotland,Europe,12345,98765,72.6
   ok, inserted (in data storage)
   ok, inserted (in code index)
```

Log – PRETTY PRINT results look like this:

```
N: 240
DATA STORAGE
LOC/ CDE ID- NAME-------------- CONTINENT---- ------AREA ---POPULATION LIFE
001/ AFG 001 Afganistan         Asia             652,090    22,720,000 45.9
. . .
094/ CHN 094 China              Asia           9,572,900 1,277,558,000 71.4
. . .
240/ SCT 240 Scotland           Europe            12,345        98,765 72.6

CODE INDEX
```

SAME FORMAT AS Asgn1

########################## COUNTRY DATA FILE's STRUCTURE ##########################

Direct address with ID as the primary key (PK).

More on this in class.