OVERVIEW

A2 project is an improved version of A1 in which the CountryData file is now:

- a binary file (`.bin`) *(rather than a csv text file (`.csv or .txt`))*
  - so numeric field storage for numeric data
- different fields stored & different storage for those fields *(rather than what's used in A1)*
- a hash file structure *(rather than direct address)* for random access functionality
- country code as the primary key *(rather than id)*
- a header record *(which A1 did not)*
- *other changes to accommodate this*

SOME CHANGES

- PrettyPrint must accommodate the differently formatted CountryData file
- Very minor changes to Setup program
  - Different fields to `get` from rawData object to send to countryData object
    but they're still ALL STRINGS (as far as rawData & Setup are concerned)
- Transaction options have changed:
  - QI, LI, DI are now just "dummy stubs"
  - IN is still an option
  - QC (queryByCode) and DC (deleteByCode) are new options
  - LC is not an option (*hash files don't allow key-sequential access)*
- Very minor changes to UserApp program
  - QC, DC cases added to the big switch statement (keep the other 4 cases)
- No changes to UI class in terms of Log file handling
- Minor changes to UI class for TransData.csv file handling
  - DC and QC will have a 3-char code as their 2nd fields (if that affects your code)
- Lots of changes to CountryData CLASS & CountryData FILE
- Different format-string for printing (used in PrettyPring and in QC results sent to Log)

NOTE: If A1 had errors in the output or in not following A1 specs or it had the wrong project structure (e.g., methods not in their proper class, public/private mis-labeling, non-descriptive method/object/ variable naming, etc.) or inaccurate top-comments, FIX THEM (as you can lose points AGAIN for A2)

CountryData FILE / RECORDS

A binary file

- numeric fields stored as int/short/long/byte/float/double's . . .as specified below
- alphanumeric fields stored as fixed-length char-arrays (left-justified & either truncated or space-padded on right, as needed, based on specifications below)
- no <CR><LF> after each records
- no field-separators (no commas)

Header record (with fields in this order):

nHomeRec – a short integer
nCollRec – a short integer
nextEmpty (in Coll area) – a short integer
MAX_N_HOME_LOC – a short integer

so SIZE_OF_HEADER_REC = 4 * SIZE_OF_SHORT;

Data records (with fields in this order)

- `id` – 16-bit short integer
- `countryCode` – 3 char (always 3 char on input file, so…)
- `name` – 17 char (left-justified & (truncated or space-filled on right))
- `continent` – 13 char (left-justified & (truncated or space-filled on right))
- ~~`region` – *DON'T USE THIS FIELD which is in RawData though not used in A1*~~
- `surfaceArea` – 32-bit integer
- `yearOfIndep` – 16-bit short integer
- `population` – 64-bit long integer
- `lifeExp` – 32-bit float (not a double – save space)
- **`link`** - a 16-bit short integer ***[DUE TO COLL RES ALGOR]***

so SIZE_OF_DATA_REC =
(SIZE_OF_CHAR * (3 + 17 + 13)) +
(SIZE_OF_INT * 1) + (SIZE_OF_SHORT * 3) + (SIZE_OF_LONG * 1) +
(SIZE_OF_FLOAT * 1);

NOTE: there are NO commas's, and NO <CR><LF>'s
NOTE: char data may be Strings when in memory, but when stored in the file, just the actual char's are stored, with NO preceding length/byte-count
NO C-style null-terminators

CountryData HEADER RECORD

The 4 values in HeaderRec are stored in MEMORY during the run of a program, which makes it faster to increment/decrement the 2 counters and nextEmpty.

nextEmpty is initialized to MAX_N_HOME_LOC + nCollRec + 1
after nCollRec is initialized to 0
[so nextEmpty starts out at RRN 21]
[but because of tombstoning (see DELETE section at the end of this doc)
nextEmtpy does not stay = to MAX_N_HOME_LOC + nCollRec + 1]

The headerRec is only written to the FILE by CountryData's finishUp method at the end of a program-run, just before closing the file (for both Setup and UserApp).

The header record data is read into MEMORY from the file in CountryData's constructor

- but only when it's called by UserApp
- and NOT when called by Setup since the file doesn't exist yet at that point

MAX_N_HOME_LOC must be s a named "constant" in CountryData class

- Setup's call to CountryData class's constructor specifies its value as 20 for now
- UserApp finds out its value because the HeaderRec was read into memory by CountryData's constructor (only when called by UserApp, not Setup)

NOTE: do NOT use 20 (constant) anywhere else in the project, use MAX_N_HOME_LOC

nHomeRec & nCollRec

- incremented/decremented (in memory) in CountryData's insert/delete methods

Log File

[Note: data below NOT accurate for data in RawData & TransData files – I'm just illustrating the required formatting]

Status messages – same as A1 for opening/closing files and program-ending

Transaction generates 1) request, 2) resulting record or response,
3) count for records read during search (for developer's benefit)

```
QC,RUS
005 RUS Russian Federatio Europe        17,075,400  1991   146,934,000 67.2
  search path:  1
QC,CHI
026 CHN China             Asia           9,572,900 -1523 1,277,558,000 71.4
  search path:  4
QC,GRL
015 GRL Greenland         North America  2,166,090  0000        56,000 68.1
  search path:  3
DC,FRA
OK, France deleted
  search path:  5
DC,WMU
** ERROR:  no country with code WMU
  search path:  4
QC,FRA
** ERROR:  no country with code FRA
  search path:  1
IN . . .
OK, Australia inserted
  search path:  2
LC
** SORRY: list all by code is an invalid option
LI
** SORRY: list all by id not yet operational
QI,15
** SORRY: query by id not yet operational
DI,15
** SORRY: delete by id not yet operational
```

PrettyPrint (with the . . . part fully filled in, of course)

```
HEADER REC:  20 Rec in Home Area,  6 Rec in Collision Area, MAX_N_HOME_LOC: 20

HOME AREA ******************************************************************************
RRN> ID CODE NAME              CONTINENT     ------SIZE  YEAR ---POPULATION L.EX LINK
001> 043 USA United States of  North America  9,363,520  1776   278,357,000 77.1   -1
002> EMPTY
003> 026 CHN China             Asia           9,572,900 -1523 1,277,558,000 71.4   25
004> EMPTY
005> TOMBSTONE                                                                    23
006> 005 RUS Russian Federatio Europe        17,075,400  1991   146,934,000 67.2   35
. . .
COLLISION AREA *************************************************************************
021> TOMBSTONE                                                                    -1
022> 015 GRL Greenland         North America  2,166,090  0000        56,000 68.1   -1
023> TOMBSTONE                                                                    22
. . .
```

NOTE:

- *PrettyPrint shows: RRNs & Links,*
  but QC transactions do NOT show these when they find a record
- *PrettyPrint shows: EMPTY and TOMBSTONE labels*
  But QC and DC transactions do not

Hashing

hashFunction

MUST BE a callable, private method named **hashFunction** inside CountryData class which any other method in this class can call, as needed:

- input: code & MAX_N_HOME_LOC
- output: homeRRN
- processing: (the algorithm)
  1. convert 3 char's in country code to caps, if they're not
  2. convert 3 cap char's to their 3 ASCII codes
     (giving 3 numbers, all between 65 to 90, inclusive)
  3. multiply those 3 numbers together
     *(giving numbers between 65*65*65=274,625 to 90*90*90=729,000)*
  4. use division-remainder algorithm (that is,
     divide step #3 result by MAX_N_HOME_LOC and use the remainder)
     (giving a number between 0 & MAX_N_HOME_LOC – 1, inclusive)
  5. add 1 to remainder
     (giving a number between 1 & MAX_N_HOME_LOC, inclusive)
  6. return that as homeRRN

NOTE: nothing else should go in this method – no reading/writing records – no seeking or byteOffset calculations, no . . .

Collision resolution algorithm: "**Chaining** with **separate** overflow". That is,
There are 2 SEPARATE areas of the file

- HOME area – in locations 1 through MAX_N_HOME_LOC
  (i.e., RRNs 1-20)
- COLLISION area - in locations MAX_N_HOME_LOC + 1 through nextEmpty-1
  (i.e., RRNx 21 to ???)

The collisions in a particular synonym family form a chain.

Chains

- A chain is an UNORDERED linked list for a SINGLE synonym family
- Data in home area is NOT part of chain - chaining is for collisionns only
- Since there's only 1 headPtr of a chain, it's stored in link field of home data node.
- There'll be a total of MAX_N_HOME_LOC chains (i.e., one per synonym family), some of which could be empty
- Use -1 to indication "points nowhere" (i.e., "null ptr").
- Inserting in a linked list requires:
  1. Physically store node in nextEmpty location in collision area
  2. Insert node in data structure on FRONT of the chain (since it's the cheapest spot to find, and there's no reason to keep ordered lists)
  3. Increment "next empty" (i.e., nCollRec to be used in the calculation in #1 above)

NOTES on RRNs

- *RRNs for the data records start at 1 (not 0) in both CountryData.bin and*
- *The headerRec does NOT have an RRN*

FAQ on PHYSICAL storage

Q: How are locations being "given out" in the home area?
A: Based on the calculated homeRRN from HashFunction
Q: How are locations "given out" in the collision area?
A: Based on the next empty location (i.e.,in MAX_N_HOME_LOC + nCollRec + 1)
Q: Is it necessary to test if a location is empty or not before writing a record there?
A: In the home area, Yes. In the collision area, No.
Q: At the end of Setup, will there be any empty locations in the file?
A: In the home area - Yes, probably.
In the collision area? - Yes, at the end, since it's theoretically infinite.

############## query/deleteByCode & insert (in CountryData class) **###############**

- Both query & delete MUST use the proper "hash search". That is, use the hashFunction to find homeRRN. If the target record (with the matching country code) is not in that location, proceed to search the collision chain.
- When to stop the search?
  - TOMBSTONES (a "no-match" case) indicate to keep searching
  - An un-used (empty) location in the HOME area indicates to stop searching
  - The end of the chain in the home area or the collision area indicates to stop searching

DELETE algorithm

Use STATIC DELETE – i.e., "TOMBSTONE" a record to "delete" it, whether it's in the home area or in the collision area. This means that synonym search paths (and chains) grow longer with an insert, but they do not get shorter with a delete.

NOTE: *static delete means that garbage collection would be done (by some other program) every week? every month?... We are NOT using, dynamic space management (i.e., physically move records in the synonym family when deleting a record).*

TOMBSTONE 0's in all numeric fields
All spaces in alphanumeric fields

SEARCH PATH

- When determining the search path, include both GOOD records and TOMBSTONEd records if they are records read in to find the target record

INSERT

- GOOD records and TOMBSTONEd locations are treated the same, for inserting.
- If the homeRRN location is EMPTY, use that location
- If the homeRRN location contains a GOOD record OR a TOMBSTONE,
  Do NOT use it – instead, proceed to using the COLLISION area/algorithm
- Collision locations are given out based on nextEmpty.So you do NOT re-use a TOMBSTONEd location.

NOTES

- *If you use Linear Search of the file (i.e., physical linear search) or Linear Search of the collision area, you will get 0 points for this asgn !!!! But YES, you do a logical linear search of the chain for the target's synonym family.*
- *Just because there's A record in homeRRN doesn't mean it's THE record you were searching for.*
- *Allow for successful and unsuccessful searches.*

############################## DUMMY STUB ##############################

deleteById, queryById and listAllById are "dummy stub" methods. That is,

- The methods EXIST
- They are callable
- Their body just contains code to call ui.displayThis with the appropriate SORRY message