CS3310 – Kaminski - Spr 2014
Asgn 3 Project Specs

World Data App 2.0
CountryDataTable is a Hash file on ID
with a Hash Table index on CODE

DESIGN NOTES

UserApp NEEDS to be able to do: SelectById & SelectByCode
& DeleteById & DeleteByCode
& InsertCountry
[so we want very fast random access on both ID and CODE]
but does NOT need: SelectAllById NOR SelectAllByCode
[so we do NOT need key-sequential access on either field]

OBSERVATIONS:

- Both id and code uniquely identify countries, so either one could be used for the PK (PrimaryKey), the other is a CK (CandidateKey).
  - **The designer specified id as the PK and code as the CK**
- id values are not a nice compact set of small numbers with a known key-distribution with relatively few gaps
  [so we can NOT use the preferred Direct Address file structure]
- Since we need access by 2 different key fields
  - The file structure for the MAIN DATA FILE ITSELF is based on the id
    [a hash file structure would be the best choice]
  - We need an INDEX for the code field
    [a hash structure would be the best choice]
    [internal vs. external ? The designer chose internal because of the activity rate and index size]

PROJECT OVERVIEW

A3 PROJECT SUMMARY

A3 is a modification of A2. (Assume A3 is like A2 unless noted in these specs or in class). The core difference is that CountryDataTable is now:

- a hash file (on id) rather than a direct address file
- indexed on country code using an internal hash table structure
- the internal index must be
  - saved from memory to a file (at end of Setup and at end of UserApp)
  - restored to memory from the file (at the start of UserApp)

Program Code Structure:

CountryDataTable is now handled in 2 physically separate classes (each in its own physically separate file):

- CountryData – for the hash file containing the actual data itself (which used to be CountryDataTable in A2)
- CountryIndex – for the internal hash table (array) for the metadata index

INCREMENTAL DEVELOPMENT

Several simplifications will be done for YOUR A3, including:

- Several transaction handlers are currently just "dummy stubs"
  - deleteById and deleteByCode
  - insertCodeInIndex
- **IndexBackup.csv** file
  - is just a csv text file for now, rather than a binary file
  - which YOU BUILD MANUALLY (rather than Setup building it)
    USING A TEXT EDITOR (based on your worksheet)
    including the <CR><LF>'s at the end of each record
  - is used by Setup to create the index (using RestoreIndex method, which is normally only used by UserApp), rather than creating it properly using RawData file and insertCodeInIndex method

CHANGES TO OTHER MODULES

Main Changes specified in DemoSpecs - fairly similar to A2

Setup (pseudocode)

```
Declare 4 objects:   rawData, theLog, countryData, countryIndex
// TEMPORARILY BUILD OF INDEX TIL InsertCodeInIndex WORKING
    countryIndex.restoreIndex            // REMOVE THIS LATER
int rrn;
rawData.input1Country
Loop til rawData.Done
{   rrn = countryData.insertCountry(rawData.getId, . . .)
    // countryIndex.insertCodeInIndex(rawData.getCode, rrn)
    rawData.input1Country
}
Call finishUp for each of the 4 objects
```

UserApp `switch` statement allows for SI, SC, DI, DC, IN (but not SA)
with appropriate calls to methods in CountryData class (for SI & DI),
OR CountryIndex class (for SC & DC)
OR both (for IN)
[YES, even though deleteById and deleteByCode and InsertCodeInIndex are "dummy stubs", they can still be called. See note below]

RawData Nothing new here

TransData Transaction options now include SI, SC, DI, DC, IN
[BUT THERE WILL BE NO IN transactions in my test file
because countryIndex.insertCodeInIndex is a dummy stub.
But you still need a working countryData.insertCountry method since
Setup will call it]

TheLog

- 2 additional status messages
  `FILE STATUS > IndexBackup FILE opened`
  `FILE STATUS > IndexBackup FILE closed`
- Snapshot displays
  - CountryData.bin file
    N's value at the top
    [RRN] then the 7 fields (just like A2 did) for good records
    [RRN] EMTPY for empty locations
  - IndexBackup.csv file

```
CODE INDEX> MAX_N_HOME_LOC: 20, nHome: 17, nColl: 9

[SUB] CODE | DRP |LINK |
[000]  MEX | 012 | -01 |
[001]  CHN | 003 | 033 |
[002]  EMPTY
. . .                                    (with this part filled in, of course)
[027]  OMN | 033 | -01 |
[028]  DEU | 013 | 032 |
```

- Transaction processing
  - Different transaction types, so their echo looks slightly different
  - Sorry messages for dummy stubs:

```
SORRY, deleteById not yet working
SORRY, deleteByCode not yet working
SORRY, insertCodeInIndex not yet working
```

  - Responses for SUCCESSFUL SI & SC requests
    Display all 7 fields as in A2
    [SC requests display the ACTUAL DATA, NOT the RRN]
  - Responses for UNSUCCESSFUL SI & SC requests

```
SORRY, no country with that id
SORRY, no country with that id
```

  - # nodes/records visited is displayed (in theLog) for BOTH successful & unsuccessful SI's & SC's
    - For SI requests `3 data records read`
    - For SC requests `4 index nodes visited`
      `1 data records read`

CountryData

- Needs lots of changes from A2
- All handling of the MAIN DATA FILE happens in here
- There MUST BE a private method called HashFunction which
  - returns homeRRN
  - when supplied with id and MAX_N_LOC (as parameters)

CountryIndex

- This is NEW for A3 – all handling of the CodeIndex happens in here
- There MUST BE a private method called HashFunction which
  - returns homeSubscript
  - when supplied with code and MAX_N_HOME_LOC (as parameters)

######################## WHAT'S A DUMMY STUB? ########################

The method exists and is callable (with appropriate parameters).
The method body is (pretty much) just a "Sorry not yet working" message (to theLog).

Dummy stubs include: countryData.deleteById
countryIndex.deleteByCode
countryIndex.insertCodeInIndex

######################## HASHING CONCEPTS ########################

- **Hashing needs 2 algorithms**:
  1. A **hash function** – where the key field (and MaxN value) are supplied, and the homeAddress is returned
  2. A **collision resolution algorithm** – which determines where collisions are stored and how they'll be accessed. [A record is a collision if it can NOT be stored in its homeAddress because another record is already there].

  All records which hash to the same homeAddress are called a "synonym family". The first record hashing to a particular homeAddress is stored in that location. Subsequent records in that synonym family (whether during the initial loading in SetUp or during transaction processing in UserApp) are collisions and must be stored elsewhere.

- **Static hashing** is used for both the hash TABLE (the code index) & for the hash FILE (for the actual data)
  - Static hashing (*vs. dynamic hashing*) means that the storage size is determined by compile time (*rather than at run-time, and thus would be increased, as needed*)
  - "Storage size" refers to the number of storage locations:
    - in the entire table/file for "EMBEDDED overflow" (collisions)
    - in the home area for "SEPARATE overflow" (collisions)
  - CountryData FILE uses **MAX_N_LOC = 30** (meaning RRNs of 1-30)
  - CountryIndex TABLE (array in memory) uses **MAX_N_HOME_LOC = 20** (meaning subscripts of 0-19)
  - Hard-code these two values as NAMED CONSTANTS in their appropriate class (*i.e., do NOT ever use the actual numbers/values of 30 or 20 anywhere else – use the constant **names*** instead)

- The TWO DIFFERENT **hash functions** for these two storage structures:

  - CountryData FILE uses "Division-Remainder", then 0 → MAX_N_LOC:
    1. divide id by MAX_N_LOC, and use the remainder not the quotient (HINT: use % operator)
    2. if remainder is 0, change it to MAX_N_LOC
    3. RETURN this homeAddress which is between 1 & 30, inclusive

  - CountryIndex TABLE uses this function:
    1. convert 3 char's in code to their 3 ASCII codes *(giving 3 numbers, all between 65 to 90, inclusive)*
    2. multiply those 3 numbers together *(giving numbers between 65*65*65=274,625 and 90*90*90=729,000)*
    3. use division-remainder algorithm (i.e., divide step #2 result by MAX_N_HOME_LOC and use the remainder) *(giving a number between 0 & MAX_N_HOME_LOC – 1, inclusive)*
    4. RETURN homeAddress which is between 0 & 19, inclusive

- REMINDER ON **STARTING & ENDING LOCATIONS**

  - CountryData is a FILE (stored in a RELATIVE FILE)
    - by convention (as in asgn 2), RRNs start at 1
    - so MAX_N_LOC of 30 means the last location really is RRN 30

  - CountryIndex is a TABLE in MEMORY (stored in an ARRAY)
    - Typically languages default SUBSCRIPTS start at 0
    - so MAX_N_HOME_LOC of 20 means last location really is [19]

- The **collision resolution algorithms** are different for these two:
  - CountryData uses "**Linear** with wrap-around with **Embedded** Overflow"
    - home area & collision area share locations: 1 to MAX_N_LOC
    - linear with wrap-around means: add one **mod** MAX_N_LOC

  - CountryIndex uses "**Chaining** with **Separate** Overflow"
    - "Separate Overflow" means
      - home area: locations 0 to MAX_N_HOME_LOC - 1
      - collision area: locations MAX_N_HOME_LOC → ????
    - a "Chain" is a linked lists (**of just a single synonym family**)
      - The collisions form a chain
      - The HP of the chain is stored in the link field of the record in its home location
      - The record in its home location is NOT part of the chain
    - NOTE: There will be MAX_N_HOME_LOC distinct chains, 1 per synonym family (though some may be empty)
    - NOTE: The links of the linked list (& the HPs) are subscripts
    - NOTE: Use -1 for a link which "points nowhere" (including for HPs which point nowhere)

################### CountryIndex STORAGE ISSUES ###################

Physical storage (in memory):
- 3 numbers: MAX_N_HOME_LOC, nHome (a counter), nColl (a counter)
- array of nodes (or 3 parallel arrays) where a node contains:
  1) key value - country Code
     (stored as a string or a 3-char-array)
  2) DRP - the RRN of the actual record in the CountryData FILE
     (this is NOT necessarily the record's homeAddress in the file – it's the FINAL ACTUAL STORAGE LOCATION which might be a homeAddress or a collision address)
     (NOTE: DRP is NOT the country's ID, it is its RRN)
  3) Link (for the linked list)
     a. For nodes in the HOME area, this is the HP (headPtr) for the collision chain for THIS synonym family,
        (a -1 if it's an empty chain)
     b. For nodes in the COLLISION area, this is the link to the next node in the chain
        (a -1 for end-of-chain)

Chains (linked lists) – 1 per synonym family
- HP is stored in the link field of the node in the HOME area
  (though the code & its DRP in the home area are NOT part of the chain)
- New collision nodes are always inserted on the FRONT of the chain,
  NOT THE BACK (and the chains are NOT an ordered lists)
- Inserting in a linked list require these 3 steps:
  1. Physically store the node in nextEmpty location in collision area - that is, at MAX_N_HOME_LOC + nColl
  2. Hang the node in linked list (at the FRONT)
     a. this new node's link = HP
     b. HP = this new node's storage location (see step 1)
  3. Increment nColl
- How do you know where the nextEmpty is?
  - It was initially MAX_N_HOME_LOC
    (that is, 20, since HOME area uses 0-19)
  - When you did the physical storing & incrementing above, you added 1 to nColl – so nextEmpty is:
    MAX_N_HOME_LOC + nColl

Initialization of physical storage?
- Is this necessary?
  (Yes, for the HOME area, but not for the COLLISION area)
- How are locations being "given out"?
  (Based on the calculated homeAddress from the HashFunction)
- Will it be necessary to test whether a location is empty or not?
  (Yes, for the HOME area, but not for the COLLISION area)
- At the end of Setup, will there be any empty holes in the array?
  (Yes, in the HOME area, but not in the COLLISION area)

- Do nHome and nColl need initializing? (Yes, since they're counters)
  - And nColl is used to calculate nextEmpty in the COLLISION area