CS3310 – Kaminski - Fall 2014 **CountriesOfTheWorld App 1.5**
Asgn 3 Project Specs

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

A3 is a modification of A2. [This assumes that you already have the correct program structure for A1/A2 which follows the design specs – so fix things in that regard before proceeding with A3 changes]. Assume A3 requirements are the same as A2's, unless changes are specified here or in class. The 4 main changes:

- dataTable is still an **external** table (a file) – BUT, it now uses
  1) a **hash file** structure (instead of direct address)
  2) on **CODE as the key** (instead of id)
  3) AND it's **a binary file** (instead of a text file)
- nameIndex is still an **internal** index, with a BST structure on name as the key. BUT, the **DRP** is no longer id – it's just a DRP, **which is the RRN pointing to the actual storage locatio**n of the record in the file (whether that is in:
  - the home area of the dataTable file
  - OR the collision area of the dataTable file

######################## WHAT NEEDS CHANGING ? ########################

Programs

- TestDriver – minor changes (to be described in the A3DemoSpecs)
- PrettyPrintUtility – (which will be written by someone else which you MUST use)
  - reads/prints CountryData file - BUT
    - it's now a BINARY file
    - where the designer is dictating EXACTLY how a record looks
    - and every data record now includes a link field at the end of it
    - and headerRec has different data in it (as compared with A2)
  - it still reads/prints Backup file – BUT
    - it's now a BINARY file
    - which now must **follow exact specs** since someone else is writing the program to read this file
- Setup – minor change (assuming you did it right in A1 & A2) due to the DRP issue:
  Core algorithm inside the loop is now:
  - rawData.input1Country()
  - storageLocation = dataTable.insertCountry(
    rawData.getId(), rawData.getCode(), . . . rawData.getLifeExp());
  - nameIndex.insertCountry(rawData.getName(), storageLocation);

  [although the actual code may use a process/read loop structure rather than this read/process loop structure]
- UserApp – change big switch statement due to different TranCodes
  AN, SN, DN – still valid (they call methods in NameIndex class)
  IN – still valid (calls dataTable.insertCountry then
  nameIndex.insertCountry – but see changes in Setup)
  AI, SI, DI – now Invalid codes
  SC, DC – new valid codes (these call methods in DataTable class)
  AC – NOT A VALID CODE

Instantiable Classes

- RawData and UI – no changes
- NameIndex – minor changes:
  - calls dataTable.getDataRec(specifying DRP for the RRN)
    rather than calling dataTable.selectById(DRP)
  - don't ever refer to DRP as id, since it's NOT, it's the DRP (which is != id)
  - finishUp method's writing of the index to the Backup file MUST follow the exact specs for the file since someone else is writing the PrettyPrintUtility
    - and it's a BINARY file, not a text file
    - where numeric fields are all shorts
    - and name is truncated to 15 so it fits in the char array
    - and no field-separators, nor record-separators are used
- BSTNode – no changes
- DataTable – major changes:
  - file/records MUST follow exact specs for headerRec and dataRecs since someone else is writing PrettyPrintUtility
  - it's a BINARY file, not a text file
    - so fields are different & hence dataRecord storage is different
    - and there's a new link field
  - so sizeOfDataRec and sizeOfHeaderRec, used in byteOffset calculations, are different (in the one-off calculations at the top of the class)
  - Fix insertCountry method
    - returns actualStorageLocation, the RRN where record is stored, whether it's in the home area or collision area
    - uses NEW file structure (HASH) & new key (CODE)
  - Remove methods for invalid transaction handlers:
    - selectById
    - deleteById
    - selectAllById
  - Add a method, getDataRec(int RRN), which nameIndex calls
  - Add methods for:
    - selectByCode – this MUST use hashSearch algorithm
      BIG LOSS OF POINTS FOR LINEAR SEARCH
    - deleteByCode – which is a dummy stub
  - NO selectAllByCode – if the app needed this, we shouldn't have chosen a hash file structure, since it doesn't support key-sequential access
- DataRecord - optional – may do this work in DataTable

Data Files

- A2RawData.csv and A2TransData?.txt – same format, but different:
  fileNameSuffix, transCodes & data contents
- CountryData.bin - it's a BINARY file, not a text file
- Backup.bin– must follow EXACT specs since someone else is writing PrettyPrintUtility (and it's a BINARY file, not a text file)
- Log.txt – results from PrettyPrintUtility look a bit different:
  - dataTable results
    - headerRec has different fields
    - records have a LINK field at the end of each record
  - nameIndex results follow revised results shown in A2's FAQ

################################ Backup.bin file ################################

HeaderRec:
- n (a short)
- rootPtr (a short)

DataRec's:
- leftChPtr (a short)
- name (a char array of size 15)
- drp (a short)
- rightChPtr (a short)

NO field-separators are needed (e.g., commas) – so don't use them.
NO record-separators are needed (i.e., <CR><LF>) – so don't use them.

############################ HASH FILE on CODE ############################
[see HashingConcept notes] [see DataTableIssues notes]

Hash 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 single number between 0 & MAX_N_HOME_LOC – 1, inclusive)
4. if remainder is 0, change it to MAX_N_HOME_LOC to get homeAddress
   (giving a number between 1 & MAX_N_HOME_LOC, inclusive)
5. RETURN homeAddress

NOTE: This MUST be a single callable method named hashFunction.
It may NOT be just in-line code because it's so little code.
There must NOT be multiple copies in the project.

MAX_N_HOME_LOC (a named constant): 20 for now – meaning:
- home area uses RRNs of 1 through 20 in the file
- collision area uses RRNs 21 through infinity (theoretically) in the file

Collision resolution algorithm: "**Chaining** with **Separate** Overflow"
- "Separate Overflow" means collision area: locations MAX_N_HOME_LOC + 1 → ????
- a "Chain" is a linked lists (**of just a single synonym family**)
  - Collisions form a chain, with new inserts going onto the FRONT of the chain.
  - HP of the chain is stored in the link field of the record in its home location
  - BUT the record in its home location is NOT part of the chain

NOTE: Each data record must now include a link field as the last field in the record
- For home area records, that's a HP
- For collision area records, that's a linked list "link"

NOTE: There'll 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 RRNs
NOTE: Use -1 for a link which "points nowhere" (including for HPs which point nowhere)

########################## CountryData.bin records ##########################
[see DataTableIssues notes]

Header record:
- MAX_N_HOME_LOC - a short
- nHome - a short – this is a counter
- nColl – a short – this is a counter

Data records:

code	– 3 char charArray
id	– a short
name	– 15 char charArray (truncated or space-filled on right)
continent	– 13 char charArray (truncated or space-filled on right)
size	– an int
population	– a long
lifeExp	– a float
link	- a short **<<<< NOTE: NEW FIELD FOR CHAIN**

NOTE: Truncating and space-filling of charArray fields is done to ensure all records are of the exact same size (i.e., fixed-length records, necessary for hash files). That isn't needed for numeric fields since all shorts are 2 bytes, ints are 4 bytes, longs are 8 bytes, floats are 4 bytes, doubles are 8 bytes.

#################### PrettyPrintUtility output in Log file ####################

NOTE: data below is not accurage – I'm just demonstrating the output formatting

```
DATA STORAGE
MAX_N_HOME_LOC: 20,  nHome: 17,  nColl: 9

LOC/ CDE ID- NAME---------- CONTINENT---- ------AREA ---POPULATION LIFE LINK
001/ CHN 094 China          Asia          9,572,900 1,277,558,000 71.4  -1
. . .
020/ . . .
021/ . . .
. . .
029/ AFG 001 Afganistan     Asia            652,090    22,720,000 45.9 024


NAME INDEX
N: 25,  RootPtr: 0
LOC/ LCh Name---------- DRP RCh
000/ 001 China          001 003
. . .
024/ . . .
```