Asgn 1 – Spring 2015
CS3310 – Kaminski

Code Index (countryCode as key)
(Binary Search Tree (BST))

A1/A2/A3 PROJECT OVERVIEW: Asgn 1 (A1) provides a temporary program development environment to design, build and test the CodeIndex class which will be used later in A2/A3 application. And the 3 procedural/controller `main`-ish methods which use the codeIndex object, will be transformed in A2/A3 into actual stand-alone programs (with their own actual `main` methods) in A2/A3.

CodeIndex in A1 is an internal index based on countryCode as the key (e.g., USA, MEX, CAN) which will be used (later, in A2/A3) to access the main CountryData file (i.e., the actual data records for each country) which will itself be organized as a random access file based on countryId as primary key.

A1 OVERALL PROGRAM STRUCTURE: A1 is a single program which includes 3 pseudo-programs which are procedural controllers (`Setup, UserApp, PrettyPrint`) – each is in its own procedural class. They share `CodeIndex` class (an OOP class). Each of these 4 classes is stored in a physically separate file (i.e., its own `.java` file) to make reuse (in A2/A3) easier. `CodeIndex` class file will also contain a `BstNode` class (an OOP class), only "viewable" from CodeIndex.

`setupMain, userAppMain` and `prettyPrintMain` are called from the overall program's `main` controller (the "test driver") of the program. The DemoSpecs will provide the pseudocode for this.

There'll be a single codeIndex object (declared by each of the 3 pseudo-programs) and many bstNode objects.

A BATCH-PROCESSING APPS: For simplicity, these 3 pseudo-programs just use file-in, file-out for a quick interface which facilitates testing, rather than an interactive, GUI user interface. We'll use an OOP design for A2/A3 which will make it easier to convert this project to a use a nicer interface in the future (e.g., A7, not done this semester).

********** **3 PSEUDO-PROGRAMS** **********

1. **Setup** creates CodeIndex based on data in the RawData file. Since it's an INTERNAL index, it's saved to an EXTERNAL IndexBackup file after being built (at the end of Setup's execution).
2. **UserApp** uses the CodeIndex to carry out the transaction requests (from the TransData file) including I/D/S/A type transactions (for Insert, Delete, Search, and All requests). Transaction processing is done on the INTERNAL index – so before any transaction processing, the EXTERNAL IndexBackup file is re-loaded to the INTERNAL CodeIndex storage – and then re-saved to the EXTERNAL IndexBackup file at the end since the index may have changed with Insert/Delete transactions.
3. **PrettyPrint** is a utility which reads/prints the IndexBackup file, showing it (nicely) on the Log file. NOTE: It does NOT display the internal array!!!

IMPORTANT NOTE: setupMain, userAppMain, prettyPrintMain (nor their private methods) do NOT ACTUALLY DEAL WITH CodeIndex directly of course. They only:
a) declare a codeIndex object (which runs its constructor, possibly overloaded),
b) then call appropriate CodeIndex public service methods, as needed,
c) then call CodeIndex's finishUp method to "tidy up" at the end (to, in effect "destruct" the object).

IMPORTANT NOTE: The overall program `main` is merely the "test driver" of the 3 pseudo-programs. It doesn't declare any objects or open any files. The 3 pseudo-programs each open/close the Log file (ONCE) themselves.

********** **DATA FILES** **********

`RawData?.csv`
- INPUT file for setupMain (so this method or its private methods) is responsible for opening/closing this file
- where ? is the fileNameSuffix string, supplied to setupMain by the main controller
- a "csv" file is a Comma-Separated-Values text file (so it's readable by NotePad just like a .txt file). More in class on this.
- See actual file for format

`TransData?.txt`
- INPUT file for userAppMain (so this method or its private methods) is responsible for opening/closing this file
- where ? is the fileNameSuffix string, supplied to userAppMain by the main controller
- transCode is the 1st char in the record: `I, D, S, A`
- sample records: `I CAN 02  D CHN  S USA  A`

`Log.txt`
- OUTPUT file for setupMain, userAppMain, prettyPrintMain
- Since this is a single file used by all 3 pseudo-programs, open it in append mode in userAppMain and prettyPrintMain (or their private methods), and in truncate mode in setupMain.
- See what needs to be written to this file and the exact format below

`IndexBackup.txt`
- OUTPUT file from finishUp method in CodeIndex class
  - This overwrites any previously existing file with this name, so open it in truncate mode.
- INPUT file for constructor (#2) method in CodeIndex class

- This file is opened/closed by methods in CodeIndex class (in constructor #2 & finishUp) and NOT in Setup, UserApp, PrettyPrint classes.
- What gets written to the file:
    - The HeaderRecord: rootPtr, n, nextEmptyPtr
    - The nextEmptyPtr BST nodes (including good nodes & tombstones)
        - LChPtr, KV, DRP, RChPtr

********** **LOG FILE FORMAT** **********

NOTES:

- *You **MUST** use my **exact format/wording/spacing/alignment** as shown below ! ! ! !*
- *Data shown below are NOT necessarily accurate based on the actual RawData file. I'm just showing you the appropriate display format.*
- *The . . . portion of the A transaction responses and the PrettyPrint results must be filled in, of course.*
- *Transaction responses just display DRP since we don't yet have the actual main CountryData file built yet in A1*
- *Should the STORAGE LOCATIONS (SUBSCRIPTS) BE DISPLAYED?*
    - *NO when responding A transaction requests since the USER does NOT care about such things*
    - *YES when showing PrettyPrint results since the DEVELOPER DOES care to know such things.*
- *Should the reassurance message be displayed when codeIndex.insert is called?*
    - *NO when Setup calls it (multiple times)*
    - *YES when UserApp calls it for an I transaction*
- *Should TOMBSTONES be displayed?*
    - *NO when responding to A transactions since the USER does NOT care about such things*
    - *YES when showing PrettyPrint results since the DEVELOPER DOES care*
- *PrettyPrint must use appropriate formatters to display the data so it aligns when printed out in Courier (fixed-width) font:*
- *3 kinds of things written to the Log file:*
    1. *Status messages*
    2. *Transaction processing (transaction requests are echo'd, then responses written)*
    3. *Output from PrettyPrint*

Status messages appear AT THE APPROPRIATE TIMES when the EVENT HAPPENS

NOTE: you MUST place code appropriately, that is:

- FILE OPENED messages generate in the line of code JUST AFTER opening the file
- FILE CLOSED messages generate in the line of code JUST BEFORE closing the file
- CODE STARTED messages generate AT THE TOP of the appropriate method
- CODE ENDING messages generate AT THE BOTTOM of the appropriate method

```
STATUS >>> started Setup
STATUS >>> ended Setup finished - 26 countries processed
STATUS >>> started UserApp
STATUS >>> ended UserApp - 14 transactions processed
STATUS >>> started PrettyPrint
STATUS >>> ended PrettyPrint - 26 countries processed

STATUS >>> opened RawDataAZ.csv file
STATUS >>> closed RawDataAZ.csv file
STATUS >>> opened TransData1.txt file
STATUS >>> closed TransData1.txt file
STATUS >>> opened Log.txt file
STATUS >>> closed Log.txt file
STATUS >>> opened IndexBackup.txt file
STATUS >>> closed IndexBackup.txt file
```

Transaction processing results

```
S USA >>> DRP is 06       /// visited 2 nodes
S WMU >>> invalid code    /// visited 5 nodes
D DEU >>> ok, deleted     /// visited 6 nodes
D CHN >>> ok, deleted     /// visited 2 nodes
D WMU >>> invalid code    /// visited 5 nodes
I CAN 02 >>> ok, inserted /// visited 3 nodes
. . .
A
ATA 39
BEL 27
EGY 15
. . .
ZWE 25
//////
```

PrettyPrint results

```
RootPtr: 00,  N: 25,  NextEmptyPtr: 27

LOC      LCH  KEY  DRP  RCH
000 >>>  001  MEX  012  003
001 >>>  005  CHN  -01  002
. . .
025 >>>  -01  ATA  039  -01
026 >>>  -01  CAN  002  -01
//////////////////////////
```

********** **CODE INDEX** **********

IMPLEMENTATION OF CodeIndex: a Binary Search Tree (BST).

(NOTE: Any mention/use of BST is private to this class or bstNode class – and is completely hidden from the 3 controller pseudo-programs).

IMPLEMENTATION OF the BST: an array of bstNode objects for the data

PLUS the rootPtr

and n (for number of good BST nodes, not including tombstones)

and nextEmptyPtr (since you're doing "memory management" manually and using static delete).

Indexes need these 2 fields: KV & DRP, i.e.,

KeyValue (i.e., countryCode) and

DataRecordPtr (i.e., countryId – because of how we'll implement CountryData file)

BST nodes need these 3 fields: LChPtr, KV, RChPtr, i.e.,

LeftChildPtr & RightChildPtr = array subscripts, pointing to some node in the array

(or -1 for "points nowhere")

KeyValue (i.e., countryCode – used to decide "head left" or "head right")

NOTE: "Pointers" include lChPtr, rChPtr, rootPtr and next EmptyPtr. They aren't true pointers; they all just "point to" somewhere in the array of BstNodes – so they're subscript values.

NOTE: Using manual space management (of array storage) and "static delete"

- *n and nextEmptyPtr are initialized (in the constructor?)*
- *n and nextEmptyPtr are incremented in insert (used by Setup and UserApp)*
- *n is NOT decremented in delete since we're using "static delete" with tombstones*
- *insert uses the nextEmpty location – it does NOT USE TOMBSTONED locations*

********** **BST ALGORITHMS** **********

[FAQ: You don't have to use "MY" algorithms, specifically. You may use iterative or recursive algorithms. But you MUST tailor ANY algorithms to deal with MY SPECIFIC REQUIREMENTS FOR HOW THE BST IS IMPLEMENTED].

- Search uses BST search algorithm
  - Doing a LINEAR search will result in TAKING LOTS OR POINTS OFF
- Delete uses a STATIC DELETE BST algorithm (NOT DYNAMIC DELETE)– i.e.,
  - Search for correct node
  - Tombstone it (i.e., mark it as deleted, but do NOT remove it from the tree – so keep the KV/LChPtr/RChPtr as is, but change DRP to -1)
- Insert uses BST Insert algorithm
  - This is used by both Setup and UserApp
- All uses BT Inorder Traversal recursive algorithm
  - Doing any kind of SORT will result in TAKING LOTS OR POINTS OFF

  [FAQ: Note that a BST is in logical sequence (i.e., logically sorted), but it's NOT in physical sequence (i.e., physically sorted). So if you looked at the array (or the IndexBackup file), you would NOT see the codes in alphabetical order. HOWEVER, if you use the BT InOrder Traversal algorithm, then you can visit the nodes in alphabetical order, and so "use" the DRPs in the order needed

IMPORTANT NOTE: PrettyPrint has no idea that the CodeIndex nodes in the IndexBackup file are in the form of a BST. After reading/printing the HeaderRecord data (rootPtr and n), it just uses a FOR loop to go from 0 to n-1.