CS3310 – Kaminski - Spring 2015 World Data App 1.8 – partial
Asgn 5 Project Specs *CodeIndex implemented as external B+ Tree providing SC & AC functionality*

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

This asgn is NOT a modification of an earlier project, although conceptually it's related. The focus here is using CodeIndex (a B+ tree file) to access the main DataStorage file (direct address on id). However, only PARTS OF the overall project are implemented for Asgn5.

- No Setup program – CountryData and CodeIndex files are provided and/or created with a utility program.
- No PrettyPrint program needed since you're not creating these data files (per se).
- No TestDriver program.
- No separate UI classes for TransData file and Log file.
- No IndexBackup file needed since CodeIndex is an EXTERNAL structure (a FILE).
- **Only SC (Select by Code) and AC (Select All by Code) functionality are provided.**
- No IN/DC/DI/SI/AI transactions (so you don't even need dummy stubs for these).

Since there's no Setup program and no insert methods for CodeIndex or DataStorage, the 2 files are provided (sort of) so you can test is your SC and AC handling.

- The B+ tree CodeIndex BINARY file will be created with a simple ASCII-to-BINARY conversion utility program using an ASCII TEXT file, created manually in NotePad.
- A simple 4-field CountryData ASCII CSV file created was created in NotePad which is DIRECT ADDRESS based on id.

######################## 4 REQUIRED CODE MODULES ########################

1. **AtoBConversionUtility PROGRAM**
   - Converts **`AsciiCodeIndex.txt`** file into **`CodeIndex.bin`** file using the record specifications described later in these specs
   - This is just a utility program, OOP is optional. Don't use CodeIndex class though since AtoB conversion is not really part of the full real project once it's working (e.g., in Asgn10), so don't clutter up CodeIndex class.

2. **UserApp PROGRAM**
   - Handles **`TransData8.txt`** file for input and **`Log.txt`** file for output (which may be done directly by UserApp program rather than using separate UI classes).
   - Uses the Input Stream Processing Algorithm for TransData file - i.e., loop til EOF doing {read1Trans, processThatTrans}
   - Calls CodeIndex public service classes, selectByCode or selectAllByCode methods, as appropriate, based on transCode.
   - Who calls dataStorage.readOneRecord(rrn)? Either pass dataStorage object in to the codeIndex select methods OR have those 2 methods pass back the DRP(s) for the desired RRN(s), which UserApp would then send to dataStorage for record accessing and printing.
   - Who prints to the Log file? Pass that object in for printing by dataStorage's method (which might require passing it to codeIndex which in turn passes it to dataStorage).
   - The program itself has no idea HOW the codeIndex object is implemented.

3. **CodeIndex (OOP) CLASS** in a physically separate code file
   - Handles everything to do with the **`CodeIndex.bin`** file
   - ***CAUTION:* If you use AsciiCodeIndex.txt file, you'll lose 30 points!**
   - Index implemented as a B+ tree (an EXTERNAL index)
   - Open/close file ONLY ONCE (in the constructor & finishUp) when running UserApp. HeaderRec data is read into memory after opening the file (in the constructor).
   - The entire index is NOT LOADED INTO MEMORY – this is an EXTERNAL index – all processing is done using the data on the FILE. Call readOneNode if you need to see what's on the file.
   - There must NEVER be more than a single node in memory at once – so this class only needs storage for a SINGLE NODE
   - selectByCode and selectAllByCode are CONTROLLERS – they do NOT access the file directly themselves – instead, they call readOneNode as needed (specifying the appropriate RRN).
   - Only readOneNode method can access the CodeIndex file. The method body read in an ENTIRE NODE (field-by-field? byte-by-byte?) into memory, and NOT just a single code or its drp or tp.
   - The 2 select methods also call searchOneNode at the appropriate time.
   - ***CAUTION:* readOneNode does NOT call searchOneNode, and searchOneNode does NOT call readOneNode – (if you do, you'll lose 20 points). The top-down CONTROLLER methods call those 2 methods as needed.**
   - SIZE_OF_HEADER_REC and SIZE_OF_NODE "constants" (needed for byteOffset calculations) are calculated ONLY ONCE as a function of m (read from the headerRec).
   - ***CAUTION:* No hardcoding of M's value (7) anywhere in the class. M is read in from the headerRec. Any looping is based on some function of M (like M-1). If you mention 7, you'll lose 10 points.**
   - If you use fixed-length arrays for kv's and ptr's (drpOrTp), use something like MAX_M = 10.
   - ***CAUTION:* Any linear searching of CodeIndex file results in 0 points for this asgn.** However, you'll probably do linear searching of a NODE in MEMORY (in searchOneNode).

4. **DataStorage (OOP) CLASS** in a physically separate code file
   - Handles everything to do with the **`CountryData.csv`** file.
   - This a simple 4-field ASCII, CSV file which has FIXED-LENGTH records with a contiguous set of id's, 1 through 239, with records in sequence by id. So readOneRecord can use DIRECT ADDRESS.
   - The file is must NOT be loaded into memory. You never have more than ONE RECORD in memory at once. Hence the class only needs internal storage for one record.

- Open/close file only once (in constructor & finishUp) when running UserApp.
- **CAUTION: Any linear searching of CountryData file results in 0 points for this asgn.**

################################ 5 DATA FILES ################################

1. **AsciiCodeIndex.txt**
2. **CodeIndex.bin**
3. **CountryData.csv**
4. **TransData8.txt**
5. **Log.txt**

########################## AsciiCodeIndex.txt FILE ##########################

It's a BPlus tree (not a plain B Tree).

As an ASCII text file (created manually with NotePad) - it contains:

- <CR><LF>'s after each record
- a space between fields within a record (see file itself).

1st record: HeaderRec contains these fields in this order:
m, rootPtr, nextEmptyRRN, firstLeafPtr, nKV

All subsequent records are BPlus tree NODES containing:
1st) leafOrNonLeaf (a single char, L or N)
2nd) m pairs, where each pair contains: countryCode and drpOrTp
(a leaf node contains a DRP, a nonLeaf node contains a TP)
3rd) nextLeafPtr

Non-full nodes:
- have their good-data pairs left-justified within the node
- have their "empty" pairs on the right end containing ^^^ for countryCode (since "^^^" > "ZZZ" in terms of ASCII code order) and 000 for drpOrTp

########################## CodeIndex.bin FILE ##########################

It's a BPlus tree (not a plain B Tree).

As a BINARY file, there are NO <CR><LF>'s after records and NO spaces between fields

1st record: HeaderRec contains these fields as SHORT integers, in this order:
m, rootPtr, nextEmptyRRN, firstLeafPtr, nKV

All subsequent records are BPlus tree NODES containing:
1st) leafOrNonLeaf (L or N) – a single CHAR
2nd) nextLeafPtr – a SHORT integer (which is 0 for NonLeaf nodes)
3rd) array of m code's (the keyValue)– all strings or char arrays
- You'll want to use built-in String comparison methods rather than manual char-by-char comparisons, so either STORE them on the file as strings OR convert the charArrays into strings in readOneNode when storing them internally

4th) array of m ptr's (drp or tp, depending on L or N) - all SHORT integers
(NOTE: a leaf node contains a DRP, a nonLeaf node contains a TP)

Non-full nodes have their good-data pairs left-justified within the node, with the "empty" pairs on the right end. Empty pairs contain ^^^ for countryCode (since "^^^" > "ZZZ" in terms of ASCII code order) and 0 for drpOrTp

[NOTE: C#'s String.Compare doesn't use the strict ASCII code order, but String.CompareOrdinal does].

########################## CountryData.csv FILE ##########################

CSV file with FIXED-LENGTH RECORDS containing id, code, name, lifeExpectancy <CR><LF>.

```
016,USA,United States,77.1
```

########################## TransData8.txt FILE ##########################

2 types of transactions of the following format:

```
SC FRA
AC
```

################################ Log FILE ################################

The data below for counters is not accurate - I'm just showing what the format looks like.

The number of key-comparisons includes both = and <.
The number of I/O's includes both the number of INDEX NODES read (for CodeIndex)
PLUS the number of DATA RECORDS read (for DataStorage)
– i.e., 1 since it's direct address for successful searches

```
SC FRA
>> 5 I/O's, 13 key-comparisons >> FRA France        78.8 006
SC WMU
>> 3 I/O's, 9 key-comparisons >> CODE NOT FOUND
AC
ABW Aruba          78.4 211
. . .
USA United States 77.1 016
. . .
ZWE Zimbabwe       37.8 161
++++++++++++++++++++++++++ END OF DATA (239 countries)
```