Asgn 4 - CS3310 Spring 2015
Kaminski

using a **Priority Queue**
implemented as a **Max Heap**
[an **Internal Data Structure**]

OVERVIEW: This project is a single batch processing program with a separate OOP *class (in a separate code file)* for the PriorityQueue (PQ) storage and handling. The main program *(or its methods)* is the overall controller does the reading and managing of getting the request processed. Tasks include:

- build a PQ using designated data from the InputStream
- add items to the PQ using designated data from the InputStream
- remove a designated number of items from the PQ
- empty the PQ completely
- display a snapshot of the physical implementation (an array) of the MaxHeap (just a built-in utility for the developer).

********** THE 3 DATA FILES **********

INPUT FILE: `TaskList.csv` contains a sequence of requests. This is the main input file that the main controller uses to guide the execution of the whole program.

INPUT FILE: `InputStream.csv` includes name, continent, region, population.
No quotes on the strings.
Only name & population are stored in the PQ.
Population is used as the priority value.
Continent & region are just used for control purposes.
Program only reads through this file once. Data records are in the order needed for the proper handling of the specific order of the requests in my TaskList file.

OUTPUT FILE: `Log.txt`. Requests are echoed, then the appropriate response.

NOTE on input file processing – both input files are streams of data, so you must use the input stream processing design pattern (algorithm) – that is,
Loop { read 1 record, completely deal with it }
Do NOT read either the file into memory (e.g., an array), then process it from the array!

********** 5 Types of TASKS **********

`buildPQ,continent,`*nameOfContinent*
// read & store countries in the PQ which match the specified nameOfContinent
// - when a non-matching continent is encountered, suspend inputting the stream
// USES: `pq.add` (called repeatedly, 1 call per country to be inserted)
// RESPONSE to Log:
```
>> OK, 14 countries stored in PQ
```

`add,region,`*nameOfRegion*
// read & store countries in the PQ which match the specified nameOfRegion
// - when a non-matching region is encountered, suspend inputting the stream
// USES: `pq.add` (called repeatedly, 1 call per country to be inserted)
// RESPONSE to Log:
```
>> OK, 9 countries added to PQ
```

`remove,`*number*
// delete designated number (N) of countries from PQ (i.e., the N biggest ones in decreasing size order)
// USES: `pq.remove` (called repeatedly, N times)
// RESPONSE to Log: // NOTE: all N countries are printed
```
01 > United States                  / 278,357,000
02 > Russian Federation             / 146,934,000
. . .
```
// FORMATTING NOTE: Longest name is 28 char's & biggest population is 9 digits.
// Right-justify populations and include commas

`empty`
// delete all countries from the PQ (in decreasing size order)
// USES: `pq.empty` (called only once – the method does its own looping)
// RESPONSE to Log: // NOTE: N's current value, and then all N countries are printed
```
N is 28
. . .
27 > Falkland Islands               /       2,000
28 > Vatican City                   /       1,000
```

`arraySnapshot`
// show physical array storage to aid developer. [Note: these will NOT be in sorted order – it's a HEAP]
// USES: `pq.snapshot` (only once – the method does its own looping)
// RESPONSE to Log: // NOTE: N's current value, and then all N countries are printed
```
N is 14
[00] United States                  / 278,357,000
. . .
[13] Belgium                        /  10,239,000
```

********** THE PRIORITY QUEUE **********

- All storage/handling of the PQ / heap MUST be done in the PriorityQueue OOP class.
- The PQ MUST be implemented as a Max Heap (which is a binary tree (BT)).
- The heap MUST use a Linear implimentation of a BT (not Linked with explicit childPtrs).
- Linear: array of heapNodes (or 2 parallel arrays) containing:
  name & priorityValue (which is population).
- A heap also needs N, but does not need RootPtr since that's always 0.
- 4 public service methods:
  - constructor – uses heapInitialize
  - add – uses heapInsert (which uses walkup)
  - remove – uses heapDelete (which uses walkdown)
  - empty – repeatedly calls heapDelete
  - snapshot – uses a for loop (using N, not MAX_N) to show array(s)
- 2 private methods:
  - walkup & walkdown