Asgn 4 - CS3310 - Fall 2015 Black Friday Customer Servicing App
Dr. Kaminski Priority Queue (PQ) (using a BinHeap)

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

This app handles the servicing of GoodBuy store's customers for Black Friday. Rather than a purely FIFO (FirstInFirstOut) queue based solely on a customer's arrival time, a **priority queue (PQ)** is used which takes into account arrival order as well as several preferential categories (employee or not, VIP status, age, loyalty status, etc. – as specified in the **priority rules**) to determine each person's **priority value (PV)** – and hence each person's turn to be served.

For efficiency sake, the PQ must be implemented as a ***binary heap****, an internal data structure.*

Batch processing is used during the development stage (i.e., A4) - events are read from `CustomerEvents.csv` file, and output is written to `Log.txt` file (including status messages, event/result logging and developer/testing feedback). There are 2 other events that occur which aren't in the events file: StoreOpens (which uses `LineAt6Am.csv` file) and StoreCloses. The program is kept running all day and finishes serving ALL customers that day, so there's no need to save the data in the PQ to a backup file.

############################## 4 TYPES OF EVENTS ##############################

1. **StoreOpens** – `LineAt6Am.csv` file contains the queue of people waiting outside when store opens. At opening, they're all put into the PQ, based on their PV.
2. **CustomerArrives** (after store is opened) – customer added to PQ based on their PV.
3. **CustomerServed** - next customer in PQ is removed from PQ and served.
4. **StoreCloses** – everyone still in PQ is served (based on their PV, of course).

############################### 3+ CODE FILES ################################

1. **CustomerServicingApp** program - the overall controller – `main`
2. **CustomerPQ** class
3. **Heap** class

###################### CustomerServicingApp PROGRAM ######################

```
open event & log files
declare pq object
assume StoreOpens event occurs → automatically call appropriate pq method to handle it
loop til no more customer events happening
{       read an event
        handle that event (switch/case to call appropriate pq method to handle it)
}
assume StoreCloses event occurs → automatically call appropriate pq method to handle it
close event & log files
finish up with pq object
```

############################ CustomerPQ CLASS ############################

public service methods:

- **`createCustPQ`** – from data in `LineAt6Am.csv` file (a FIFO queue).
  - opens/closes file at appropriate time
  - uses stream-processing ("design pattern") algorithm for processing file:
    i.e., repeatedly calls to heap.insert (1 per customer) approach
    (do NOT use special HeapCreate algorithm – you'd get different results !!!)
- **`addCustToPQ`** – uses heap.insert
- **`serveNextCustInPQ`** – uses heap.delete
- **`serveRestOfCustInPQ`**
  - repeatedly calls serveNextCustInPq until heap.isEmpty

NOTE: Each of these 4 methods

- *generates appropriate status message(s) – see Log.txt*
- *logs event/result(s) – see Log.txt*
- *provides developer/testing feedback – see Log.txt*

private method: findPriorityValue *[see Priority Rules section below]*

- returns priorityValue, given customer characteristics sent in as parameters

############################### Heap CLASS ###############################

data storage for binary heap (which is the PQ) which uses "linear implementation of a BT":

- N
- array of heapNodes, where a heapNode contains: name & priorityValue
  [or use 2 parallel arrays for name & priorityValue]
  [static array of constant MAX_SIZE = 200 OR dynamic array]

public methods: insert delete isEmpty

private methods: walkUp walkDown

RULE FOR TIES (on priorityValue)

- *During walkup: if parent = child then do NOT swap*
- *During walkDown, if parent = child then DO swap*
  AND when swapping, if leftChild = rightChild then swap with leftChild

This won't be perfectly "fair" since comparisons only go up/down 1 branch of tree – but at least everyone will get the same results this way.

################################ PRIORITY RULES ################################

Priority order determined by who has **LOWEST priority value** (hence, a minHeap is used).

- **nextInLine** number given out, starting with 101 → **initialPriorityValue**
  - Initialize nextInLine counter with constant START_VALUE = **101**.
  - NOTE: use the SAME " nextNumberGenerator" for BOTH
    - initial store opening (handling customers in LineAt6am file)
    - AND for each new customer arriving later (in CustomerEvents file)

    [i.e., do NOT RE-SET counter for new people in Events file – just keep incrementing]

- **points SUBTRACTED** from initialPriorityValue using these rules → **actualPriorityValue**
  - employee → 10 points
  - owner → 50 points

    (yes, owner gets OWNER points PLUS EMPLOYEE points)
  - VIP card → 5 points
  - super VIP card → 8 points

    (yes, super VIP gets BOTH VIP points PLUS SUPER VIP points)
  - loyalty card → 4 points
  - brought 1+ child along → 2 points
  - senior status (age >= 65) → 5 points
  - elderly status (age >= 80) → 5 points

    (yes, 80+ year old people get BOTH the SENIOR & ELDERLY points)

############################### LineAt6Am.csv FILE ###############################

Record Format:
`name,employeeStatus,vipStatus,loyaltyCard,haveChild,age <CR><LF>`

where

name	will always be present (and may have embedded space(s))
employeeStatus	may be empty OR say `employee` OR `owner`
vipStatus	may be empty OR say `vip` OR `superVip`
loyaltyCard	may be empty OR say `loyalty`
haveChild	may be empty OR say `child`
age	will always be present (and be a positive integer)

["empty" means that there'll be 2 contiguous commas with no value in between]

################################ Events.csv FILE ################################

2 types of records:

`CustomerArrives:  name,employeeStatus, . . .` *[same format at LineAt6Am record]*
`CustomerServed`

################################ Log.txt FILE ################################

NOTE: The data below is **NOT ACCURATE** with respect to actual A4 data in the 2 input files. It's is just to illustrate what needs to be printed out, when and its format.

NOTE: >> indicates output used by the developer for testing (AND BY GRADER).

NOTE: Numbers after names in parentheses are that customer's priorityValue

NOTE: Code that generates output to Log file should be inserted in your code AS CLOSE AS POSSIBLE to where to where the actual event described occurred, so as to make it is useful as a testing/debugging aid.

```
>> Program starting.
STORE OPENS
>> Will now insert customers from LineAt6Am into PQ.
ADDED:  Mary Smith (76)
ADDED:  John Doe (132)
. . .
ADDED:  Jim O'Leary (17)

>> Finished putting customers from LineAt6Am into PQ.
>> Dump of current heap (array) - 21 nodes:
>> SUB  PV   NAME
>> 00   017  Jim O'Leary
. . .
>> 20   103  Linda VanderCook

>> Will now process CustomerEvents data.
SERVED:  Maria Garcia (78)
SERVED:  Rajesh Patel (80)
ADDED:   Lottie Zipnowski (71)
SERVED:  Lottie Zipnowski (71)
. . .

>> Finished processing CustomerEvents data.
>> Dump of current heap (array) - 16 nodes:
>> SUB  PV   NAME
>> 00   051  Mohammed AlSabir
. . .
>> 15   123  Ravi Ganesh

STORE CLOSES
>> Will now automatically serve 16 remaining customers
SERVED:  John Doe (132)
. . .
SERVED:  Maleea Brown (142)

>> Heap is now empty - 0 nodes
>> Program ending
```