CS3310 Fall 2014 A4 – Crowd Organizer App
Kaminski (Priority Queue implemented as a MinHeap)

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

This project organizes the servicing of BetterBuy's customers for Black Friday, prioritizing people based on several criteria to determine the order in which they will be served. Rather than a regular queue (FIFO - FirstInFirstOut) based on the customer's place in line, a **priority queue** (BIFO - BestInFirstOut) is used which takes into account arrival order along with several preference categories (employee, VIP status, age, etc.) to determine the actual priority – and hence that person's actual turn to be served. *[The priority queue (**PrQ**) will be implemented as a **min-heap** (minimum heap), which is an internal data structure].*

Batch processing is used during development – i.e., events are read from `Events.txt` file - and output is written to `Log.txt` file (including status messages, event result logging and developer testing feedback). The program will be running all day and finish process ALL customers that day, so there'll be no remaining internal data which would need saving to a backup file.

############################## 4 types of EVENTS ##############################

1. **Store opens** at 6am – this is the line of people waiting when the store opens (stored in `LineAt6Am.csv` file) which will be put into the PrQ, based on their priority value. *[See Priority Rules].*
2. **Store closes** at 11pm – anyone still in the PrQ is served (in priority order, of course).
3. A **new customer** enters – he/she is added to the PrQ based on his/her priority value. *[See Priority Rules].*
4. The **next customer** in the PrQ is **served** – and hence he/she is removed from the PrQ.

############################## CODE STRUCTURE ##############################

CrowdOrganizerApp – the controller program: *[pseudocode]*

```
set up prQ object (of CustomerPrQ class type)
open event & log files
loop til no more events
{       read an event
        handle the event – the cases:
                //(a commentLine):     ignore line
                OpenStore:             call prQ.arrangeCustomerQ
                CloseStore:            call prQ.serveRemainingCustomers
                NewCustomer:           call prQ.addCustomerToQ
                ServeACustomer:        call prQ.serveACustomer
}
close event and log files
finish up with prQ object
```

CustomerPrQ class contains:

- 4 public service methods (& a public constructor?):
  1. **`arrangeCustomerQ`** – builds the PrQ from data in the `LineAt6Am.csv` file using data in the order that it appears in that file. It also generates the appropriate status message(s)
     - uses repeated calls to heapInsert method (which uses regular HeapInsert algorithm - *do NOT use the special HeapCreate algorithm, which results in a different heap)*
     - opens/closes the file, of course
     - uses the stream-processing ("design pattern") algorithm for processing the file, i.e., loop til EOF: {read1Customer, heapInsert (of Customer in PrQ)}
  2. **`serveRemainingCustomers`** – handles each customer in the PrQ in appropriate priority order & does appropriate status message(s)
     - loop until heapIsEmpty
     - uses repeated calls to heapDelete method
     - removing the node from the heap
  3. **`addCustomerToQ`** – uses heapInsert method & does appropriate status message(s)
  4. **`serveACustomer`** – uses heapDelete method & does appropriate status message(s)

- private data storage for the priority queue, which is implemented as a heap, so
  - "linear implementation of a BT" – that is, N plus an array of heapNodes where a heapNode contains name & priorityValue *[or use 2 parallel arrays for name & priorityValue]*
- private methods for dealing with the heap
  - heapInsert which uses walkUp method
  - heapDelete which uses walkDown method
- private method for
  - determinePriorityValue
- other private methods, getters, setters, data storage and nested class(es) as needed

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

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

- **nextInLine** numbers are given out **starting with 101** (a counter) → **initialPriorityValue**
  NOTE: consecutive numbers are given out during BOTH the
  - initial store opening
  - AND for each new customer entering later *[Do NOT re-set counter for new people – just keep incrementing]*

- **jumpTheQPoints** are **subtracted** from initialPriorityValue based on the following rules giving the final actual **priorityValue**:
    - employee → 25 points
    - owner → 80 points
    - VIP card → 5 points
    - super VIP card → 10 points
    - senior status (age >= 65) → 15 points
    - elderly status (age >= 80) → 15 points

      (yes, 80+ year old people get BOTH the senior & the elderly points)

- In the case of ties (i.e., = priorityValues), the person joining the queue earlier would stay ahead. To enact this (though not perfectly*), use these rules:
    - During walkup, if parent and child are =, then do NOT swap
    - During walkdown, if parent and child are =, 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 the tree. HOWEVER, since everyone in class is doing this the same way with the same rules/heap/tree/data/dataOrder, everyone will get the same results.

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

2 kinds of records:

1) DataRecord Format: name,employeeStatus,vipStatus,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`
   age will always be present (and be a positive integer)
2) a comment line starts with // in the first 2 columns (so NOT csv with 4 fields)

for example:

```
// a comment - FYI: Mary's priorityValue is 101-5-15-15 → 66
Mary Smith,,vip,81
John Doe,,,25
Maria Garcia,employee,,64
Rajesh Patel,,superVip,57
```

############################## Events.txt FILE ##############################

5 types of event records – first char can be treated as the transactionCode (/, O, C, N, S)

```
// this is just a comment
OpenStore
CloseStore
NewCustomer,Lottie Zipnowski,owner,,41
ServeACustomer
```

There'll be multiple N and S transactions, but only a single O and a single C transaction.
There'll ALWAYS be an O transaction before ANY C/N/S transactions.
There'll ALWAYS be a final C transaction as the last record.

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

NOTE: The data below is just to show the format for the output – it is not necessarily accurate with respect to actual data in the 2 input files.
NOTE: >>> status messages are for the developer (used during testing)
NOTE: The numbers in the parentheses are the priorityValue's for that person.

```
>>> program starting
STORE IS OPENING
>>> initial heap built containing 30 nodes

SERVING:  Mary Smith (66)
SERVING:  Maria Garcia (78)
ADDING:   Lottie Zipnowski (51)
ADDING:   Maleea Brown (132)
SERVING:  Lottie Zipnowski (51)
SERVING:  Rajesh Patel (94)
. . .

STORE IS CLOSING
>>> heap currently has 16 nodes remaining
SERVING:  John Doe (102)
. . .
SERVING:  Maleea Brown (132)
>>> heap is now empty
>>> program terminating
```

NOTE: Status messages must be generated in the appropriate method, RIGHT NEXT TO the code that actually does the work which the messages describes
NOTE: Mary Smith's priorityValue is 101 – 5 – 15 – 15 → 66
John Doe's is 102
Maria Garcia's is 103 – 25 → 78
Rajesh Patel's is 104 – 10 → 94
(and there were 26 others in that initial 6am line)
Lottie Zipnowski's is 131 – 80 → 51
Maleea Brown's is 132