**public class Process implements Comparable<Process>**

In my design I have one Object called Process. When this object is created with the constructor Process(int currentPID) a random priority from 0-9999 is assigned and a processID is assigned based on the currentPID. The currentPID is tracked in main. It implements comparable in order to make it possible to compare each process to each other based on priority. It has getters for the processID & priority and a setter for priority. In addition to that it has a method called increased priority which increases the priority of the process by 50 (limit of 9999). This was created to simulate that a process was waiting for a long time and will get increased priority.

**Initialization in Main**

20 Processes are created in main with a for loop. All of them are stored inside an ArrayList of Processes. After that buildMaxHeap is called to create my heap data structure which satisfies the maxHeap property. From there a method called startMenu(ArrayList of Processes) is called.

As mentioned above currentPID is located here and is incremented anytime a new process is created.

In addition to that a\_HeapSize is a static variable in main. This is done to keep track of how big the heap is. Note heap is not always the same as the arraySize.

**startMenu()**

This method prints a menu and takes a user input. The options were:

1. Show current ArrayList of Processes.

2. View and remove the first priority process.

3. Increase priority of a specified process.

4. Insert a new process with a random priority index.

5. Sort Processes by priority and displays list of Processes

Q. Quit.

Option 1 just displays the heap.

Option 2 simulates a process being popped out of the priorityqueue. HeapMaximum(A) is called so the user can see the process that is about to be removed. Because the ArrayList is already in maxHeap I know that the first element is the highest priority one. That’s the element that needs be removed and it is removed by calling HeapExtractMax.

Option 3 simulates a process that has been waiting for a while so it should get more priority. The user selects the process by index and that process will have its priority increased by 50 (limit 9999). After that buildMaxHeap is called to re-maintain the maxHeap property.

Option 4 simulates adding in a new process into the priority queue. A new process is first created with a random priority. Then maxHeapInsert is called to insert in the process starting from the leaf and bubbles up until that process is less than its parent. This is done to maintain the maxHeap.

Option 5 creates a copy of the ArrayList of Processes and a heapSort is performed on it. This ArrayList is then printed. The user will then see the processes sorted by priority from lowest to highest. A copy was created because heapSort is an in place sorting algorithm. Once heapsort is done the original array would have been modify and would no longer maintain maxHeap. In order to keep the original priority queue safe heapSort was done on a cloned ArrayList.

**private static void printProcess(Process p)**

This method just prints a single Process in the form of

Index PID Priority.

**private static void printArrayList(ArrayList<Process> A)**

Similar to above, but prints all Processes in an ArrayList of Processes.

**Problems encountered during implementation**

I kept getting index out of bounds errors because the pseudocode was misleading. Index 1 in the pseudocode meant index 0 in java. It also caused me to realize 2i + 1 is the left child and 2i + 2 is the right child due to the index being 0 and not 1. At some point I also realized the parent was (i-1)/2

I made a mistake of swapping priorities initially. It was later when I created the menu I realized I was supposed to swap processes not priorities. I learned creating the menu first before implementing other methods is better because it helps me understand the requirements better. This also made me realized I needed to make Processes comparable so I could compare their priorities.

My printArrayList() method prints all elements from a\_HeapSize to 0. However when I called heapSort a\_heapSize ends up being 0. I added a line to reset the a\_HeapSize back to the A.Size()-1 so my print method would print properly at the end.

**Lessons Learned**

I learned that A.HeapSize is the size of our abstract heap data structure. It is not always the size of the ArrayList. The ArrayList is just there to represent our heap. So it’s always possible for the heap to be much smaller than the ArrayList.

At some point in coding I realized that heapSort was not necessary for priority queue. However the methods it uses like buildMaxHeap() and maxHeapify() were important. Because heapSort does not need to be done the time complexity of the priority queue using heapSort’s methods is actually significantly better than heapSort itself.

I think I could have better designed this by making heap a separate class and have priorityQueue inherit a few methods that the heap uses.