Lab Report 03

Nathan Bickel

Problem

The problem was to create a number of helper classes (and an interface) to allow a generic queue of processes to be scheduled in a driver.

Solution

The solution involved a number of files, which are detailed in the outline below:

I. ProcessSchedulerSimulator is the driver that uses the files below to schedule the processes.
II. Process is a class that goes in the queue to be scheduled.
   A. Attributes
      1. A string called processName gives the process a name to be used in the driver.
      2. A double called completionTime stores the time remaining in the process.
   B. Constructors/Methods
      1. The constructor takes in a String and double and assigns them to processName and completionTime (respectively).
      2. getProcessName() returns processName, and getCompletionTime() returns completionTime.
      3. setProcessName(String processName) assigns the parameter to the attribute processName if the parameter isn't null, or "none" if it is. setCompletionTime(double completionTime) assigns the parameter to the attribute processName if the parameter is non-negative, and 0.0 if it is.
      4. toString() returns a String in the format "Process Name: <Name> Completion Time: <Completion Time>".
III. QueueI is an interface that the queue class implements, and it takes a generic type (called T). It has the following method signatures, which means that any class implementing the interface will be required to implement functionality:
   A. Method signatures (functionality in classes that implement the queue must implement functionality for these methods)
      1. enqueue(T data) returns nothing.

        2. dequeue() returns a type T.
        3. peek() returns a type T.
        4. print() returns nothing.

IV. LLQueue implements QueueI and is used in the driver to store processes. It takes a generic type (called T), which is used to store instances of Process.
    A. Attributes/Internal Classes/
        1. A private class called ListNode is written to allow data to be stored associated with links.
        2. A ListNode called head stores the data at the beginning of the linked list.
        3. A ListNode called tail stores the data at the end of the linked list.
    B. Constructors/Methods
        1. The default constructor is used so none were written.
        2. enqueue(T data) first creates a new ListNode with the data pointing to null. Then, if there is data in the list, it appends it to the end and moves tail to point to it. Otherwise, it sets head and tail to both point to the new node.
        3. dequeue() first creates a T and sets it to null. If there is data in the list, this T is pointed to the data at the head and then the head is moved forward. If there isn't data, nothing happens (so it will stay null). Then, this T is returned.
        4. peek() returns the data at head if there is data in the list, or nothing if there isn't.
        5. print() iterates through the list and prints the data in every node on a new line.

V. ProcessScheduler is the back-end to the driver. It creates a queue with type Process and adds functionality to this queue.
    A. Attributes
        1. A QueueI with type Process called processQueue stores the queue of instances of Process.
        2. A Process called currentProcess stores the instance of Process currently running.
    B. Constructors/Methods
        1. The constructor takes no input. It instantiates a new LLQueue of type Process and assigns it to processQueue, and sets currentProcess equal to null.

2. getCurrentProcess() dequeues a process from processQueue and sets it equal to currentProcess if and only if currentProcess is null or has a completionTime less than 0. Then, it returns currentProcess.
3. addProcess(Process process) enqueues the parameter process into processQueue, using its method to do so.
4. runNextProcess() returns nothing and dequeues a process from processQueue, using its method to do so.
5. cancelNextProcess() returns nothing and dequeues a process from processQueue, using its method to do so.
6. printProcessQueue() returns nothing and prints processQueue, using its method to do so.

Implementation Problems Encountered

The only problem I had was interpreting the instructions for the ProcessScheduler. My output matched the example output, but I wasn't quite sure if I had implemented the correct functionality in each method. However, I asked in class and was told my solution was correct. Other than that, everything was fairly straightforward.

Lab Report Questions

1. A queue is a first-in, first-out data structure, so it is similar to a line. When an element comes in, it is put at the back of the queue, and elements are removed from the front of a queue. Thus, the first element that enters should be processed first, the n^{th} element that enters should be processed before the $(n+1)^{th}$ element but after the $(n-1)^{th}$ element (assuming n is neither the first nor last element), and the last element that enters should be processed last.
2. While a queue is a first-in, first-out data structure, a stack is a last-in, first-out data structure. A queue is more analogous to a stack of paper—the last paper that is put down is the first one to be picked up again. A stack would usually be more appropriate for storing processes where one process being done depends on another process happening (like methods being called in Java), whereas a queue would usually be more appropriate for storing processes that are independent of one another.