GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Department of Computing & Mathematics

CP2SD – Data Structures & Algorithms
Lab 4: Queues

Like stacks, queues are ubiquitous data structures in computer science and are applied anytime first-in-first-out (FIFO) functionality is required. The FIFO property of queues makes them ideal structures for scheduling the processing of elements. In this lab we will look at a custom implementation of a hybrid queue that can act as a FIFO queue and also as a priority queue. Recall from lectures, that a priority queue sorts the underlying queue each time a new element is added and returns the next "best" element when the ***poll()*** method is called.

- Download the Zip archive ***lab4Queues.zip*** from Moodle and extract the files into a folder on your hard disk. As the classes involved are all packaged, ensure that you create the correct directory structure and study the source code before you compile.

- Examine the class *Queue.java*. Pay particular attention to the definition of the class name and inheritance constraints:

  public class Queue<E extends Element> {

  This definition of the class queue allows the class to be parameterized with an *Element* type or any subclass of *Element*. Note that all calls to the queue are delegated to the underlying *LinkedList*. The queue implementation takes a boolean value to its constructor that specifies whether the queue should implement FIFO functionality or behave as a priority queue.

- Edit the class *QueueTest.java*. In the constructor, instantiate *Queue* and pass the queue as a parameter to the *init()* method.

  Queue<Element> queue = new Queue<Element>(true);
  init(queue);

  The *init()* method will populate the queue with a number of elements.

The implementation of *Queue* provides two methods for adding elements: ***add()*** and ***offer()***. The ***add()*** method places an element at the tail of the queue and, if the class has been constructed as a priority queue, may also sort the underlying *LinkedList*. The ***offer()*** method is the preferred mechanism for adding to a queue, as it can refuse to add a new element (for whatever reason) and inform the caller by returning a boolean *false*.

In our implementation, a call to ***offer()*** will cause the delegate method ***contains()*** to check whether the element already exists in the queue. It is critical to understand that the ***equals()*** method, inherited from *java.lang.Object*, is used to determine object equality. In the *Element* class, the ***equals()*** method has been over-ridden to calculate equality based on the element name (as opposed to the default implementation that calculates equality based on Object ID). Thus, the call to ***contains()*** will return *true* if two elements have the same name.

- Try to add a new element to the queue with a name is already used by a queued element:
  System.out.println(queue.offer(new Element("Kingsley Shacklebolt", 11)));

  The call to ***offer()*** will return a false, meaning that the queue refused to add the new element.

- Use a ***while*** loop to ***poll*** each element from the head (front) of the queue and print out the element name by calling the ***getElementName()*** method. You should see the elements being removed from the queue in the order in which they were added, i.e. the FIFO property is being upheld. Using ***while*** and ***poll*** like this is the standard way to process a queue.

- Change the boolean parameter to the constructor of *Queue* from *false* to ***true***. This will cause the queue to act as a priority queue. Each time the ***add()*** method is called, if the queue has been parameterized as a priority queue, ***Collections.sort()*** is called to sort the queue based on element value. You should see the queued items being processed in ascending order of element value instead of using the FIFO rule.

Questions & Exercises

- Note that *Element* could also have been implemented as an interface. What would be the consequences for the queue of using an interface definition for *Element* instead of a concrete class?
- Change the queue implementation used in *QueueTest* from ***gmit.Queue*** to ***java.util.LinkedList*** as follows and execute the programme again:

  Queue<Element> queue = new LinkedList<Element>();

- The java..util library also provides a robust implementation of a priority queue. Change the queue to a priority queue as follows and execute the programme again:

  Queue<Element> queue = new PriorityQueue<Element>();