**INF 260 Program #6**

**Due Monday December 2nd, 2013**

**Queue Data Structure**

We examined the Stack data structure in class and in the course slides. It was a form of restricted list that finds many useful applications in computing/programming. For this assignment, I’d like you to implement a similar structure known as a Queue.

A Queue is known as a “fair” data structure due to its “First In First Out” behavior. It works the way you would hope a line at the movies or checking out at the grocery store would work – first one in line is the first one serviced. The later you show up the longer you wait.

A Queue is another form of restricted list. This time, however, it has 2 ends – the front and the back. All items inserted into the Queue enter at the back and all items removed from the Queue exit at the front.

The operations on a Queue do not truly have standardized names. Often the insertion operation is called an Enqueue( ) and the deletion operation is the Dequeue( ) but in some texts/papers they may simply be called insert and remove or something similar.

**Queue Implementation**

A queue is easily implemented using an array in Java. It is important to note however that we need to keep track of two things – the front and the rear of the queue. IT is also important to note that the behavior of a queue with deletions from the front and insertions at the back means that the items in the array have a sort of “head chasing a tail” behavior. That is, suppose we inserted 10 items into the queue. Now we delete say three of them. Then those three cells of the array are now empty right? And if we add more? Well those get added to the rear so the empty three cells remain empty. IF we delete a couple more items then more cells empty at the front.

Get the picture? The front of the queue begins to sort of chase after the rear. So if we added say 10 items to the queue and have deleted 4 of them then cells 0-3 would be empty and 4-9 would be full….. This is a problem. One can imagine a queue implemented as an array of say 100 cells. Suppose we added 100 items but deleted 98. Then the last to cells of the array would be full but the first 98 would be empty! And yet, when we try to enqueue a new element we will find that we have reached the end of the array! But we only have 2 elements currently in the queue!

We have two possible solutions, one is to shuffle the elements back down to the front of the array and continue or we could try to treat the array as if it wrapped back around on itself. You will eventually learn this “circular array” approach in a later course but for now we will use the shuffle down the cells approach.

**The Queue Class**

The queue class will need to have an array to hold the elements and two ints to mark the front and rear positions of the queue.

**int [ ] elts;**

**int front, rear;**

We will also need to implement the following ***public*** methods:

A **default constructor** that assumes that the queue will require an array of 10 elements. It also should initialize the front and rear to 0.

A **second constructor** which takes an integer that is used to set the size of the elts array. The front and rear indicators should also be initialized to 0.

**int getfront( )** - a method that returns the value currently at the front of the queue.

**boolean isempty( )** - method that returns true if the queue is empty else it returns false. Be careful on this one. It differs from the Stack we did in class. Now the Queue is empty whenever front == rear. **WHEN the queue is truly empty, reset the front and rear pointers to 0.**

**int dequeue( )** - method that removes and returns the front element of the queue and updates the front position by incrementing it.

**void enqueue( int x )** - method that inserts the given element x into the queue at the rear and updates the rear position by incrementing it. If the rear position then exceeds the end of the array this method should use the copyOf( ) method of the Arrays class as we discussed with Stacks to shuffle things up. See the note below on how to do this. Note that if the array is truly full, then you should double its current size as we did with the Stack example done in class and in the slides. Regardless of how the array is copied you should ***BE SURE*** to reset the front and rear pointers to their correct new values.

**Note on copyOf( )**

For our queue, we will need to use a different version of the copyOf( ) command. In particular we will want to make the copy from only the full cells of the array NOT the empty cells at the beginning. This means we want to copy only the cells of the elts array from the front position to rear – 1.

So we will need a second version of copyOf( ). This one take 4 arguments. The first two are the same as we have seen before. The array and the new length of the array. The next two arguments are the start and end indices of the array that we wish to copy.

So, assuming that your array is called elts and it is NOT truly full (that is, the queue has some open spots in it) then you would need a call such as

**elts = Arrays.copyOf( elts, elts.length, front, rear – 1);**

IF the elts array truly is full, that is, front == 0 and rear now equals elts.length, then you will need to double the size of the array elts. Do this with

**elts = Arrays.copyOf( elts, elts.length \* 2, front, rear-1);**

**Testing the Queue Class**

You should create a class called ***prog6.java*** which will test the queue. The program should contain the following code in the main( ) method.

**Queue Q = new Queue( 20 );**

**// Load the queue with 10 random integers**

**for (int x = 0; x < 10; x++)**

**Q.enqueue( x );**

**// test if the queue enqueue worked via the dequeue**

**System.out.println(“Testing Q – output should be 0-9”);**

**while ( !Q.isempty( ) )**

**System.out.println( Q.dequeue( ) );**

**// test if queue grows automatically when truly full**

**System.out.println(“\nTesting for automatic growth of queue when truly full \n”);**

**System.out.println(“Should be 0 – 34”);**

**for (int x = 0; x < 35; x++)**

**Q.enqueue(x);**

**while (!Q.isempty( ))**

**System.out.println(Q.dequeue( ) );**

**System.out.println(“\nTesting if queue copy works when not truly full”);**

**// insert 10 elements**

**for ( int x = 0; x < 10; x++)**

**Q.enqueue(x);**

**// now delete 8 of them**

**for ( int x = 0; x < 8; x++)**

**Q.dequeue( );**

**// now queue should have 2 elements at positions 8-9**

**// so we will try to add 12 elements.**

**for ( int x = 10; x < 22; x++)**

**Q.enqueue(x);**

**System.out.println(“Testing copyOf not truly full output should be 8-21\n”);**

**while (!Q.isempty( ))**

**System.out.println(Q.dequeue( ) );**