# Objectives

Practice Queue and Exception handling

# Problem

In this assignment you will be adding more methods to the Queue class created during the lecture. The methods implemented during the lecture:

* Constructor
* Dequeue
* Enqueue
* toString
* restore
* getMax

# you will be adding the following methods to the Queue class:

1. getMin
2. reverse
3. getAverage
4. isSorted
5. a method of your choice. This method has 10 points
6. driver class

# public Integer getMin()

This method finds the minimum element in the queue and returns it. Pay attention that you cannot loop through the list to find the minimum. This is the queue data structure and you can only take the element from the front. And can add it to the end. This method is very similar to the getMax method.

# Public void reverse ()

This method reverses the queue. For example, if the queue is 2 3 4 5, the method reverse will change it to 5 4 3 2. Remember that Queue is a data structure that you can take the element from the front of the queue. You cannot loop through the queue. Here is the algorithm: first all the elements must be pushed to the stack, then pop all the elements from the stack and rebuild the queue

Declare an extra storage of stack

Boolean b = false

While(!b)

{

Try

{

dequeue an element

Push the element to the stack

}

Catch (Exception e)

{

Set b to true

}

//Pop the elements from the stack and rebuild the queue.

Set b to false

While(!b)

{

Try

{

pop an element from the stack

enqueue the element that was popped

}

Catch (Exception e)

{

Set b to true

}

}

# Public double getAverage()

This method finds the average of all the elements in the queue. Here is the algorithm:

Declare sum and average, and count. Make sure to initialize the variables

Declare a Queue called copy//to restore the original Queue

Boolean b = false

While(!b)

{

Try

{

Dequeue an element from the queue and store it in a variable

Enqueue the element to the Queue copy

Add the dequeued element to the sum

Increment the count

}

Catch (Exception e)

{

b = true;

}

}end while

Call the restore method

Return the average

End method

# Public Boolean isSorted()

this method returns true if the list is sorted and false otherwise. Again, you can take an element from the beginning of the queue. You must restore the queue in its original state. Here is the algorithm

Declare and instantiate a Queue: Queue q = new Queue ();

Boolean b = false;

Boolean sorted = true;

While(!b)

{

Try

{

Integer n1 = this.dequeue();

Integer n2 = this. dequeue()

q.enqueue(n1);

q.enqueue(n2);

if(n1 > n2)

sorted = false;

}

Catch (Exception e)

{

b = true;

}

} end while

Restore(q)

Return sorted

# Here is the main method that your queue class must work with.

class Driver  
{  
 public static void main(String[] args)  
 {  
 Queue m = new Queue();  
 m.enqueue(10);  
 m.enqueue(12);  
 m.enqueue(15);  
 m.enqueue(7);  
 m.enqueue(100);  
 m.enqueue(22);  
 System.out.println("The queue is : " + m);  
 m.reverseOrder();  
 System.out.println("The queue in the reverse order is: "+ m );  
 m.reverseOrder();  
 System.out.printf("Average = %.2f\n", m.getAverage());  
 System.out.println(m);  
 System.out.println("Max = " + m.getMax());  
 System.out.println("Max = " + m.getMin());  
 System.out.println("The list is sorted: "+ m.isSorted());  
   
   
 }  
}

sample output:

The queue is : 10 12 15 7 100 22

The queue in the reverse order is: 22 100 7 15 12 10

Queue is back to its original state: 10 12 15 7 100 22

Average = 27.67

Max = 100

Max = 7

The list is sorted: false