CSIS 10B Lab 4 Queues and Lists

Goal

In this lab you will work with two new ADTs: Queues, which represent a "line" of service requesters, and Lists, which are the most general purpose linear data structure, used to represent all kinds of collections where order is important (unlike the Bag, where the order of items is not a concern).

Part A: Queues

Resources

- Chapter 10

Java Files

- *LQueue.java (a Linked Queue)*
- *Queue.java (Queue Interface)*
- *StoreSim.java*

Introduction

This laboratory focuses on another constrained linear data structure, the queue. The elements in a queue are ordered from least recently added (the front) to most recently added (the rear). Insertions are performed at the rear of the queue, and deletions are performed at the front. (Compare this to the Stack, in which both insertions and deletions happen at the top of the stack.) You use the enqueue operation to insert elements and the dequeue operation to remove elements. The Queue interface we will be using for this lab is defined below.

```
public interface Queue
{
    // Queue manipulation operations
    public void enqueue ( Object newElement );  // Enqueue element at rear
    public Object dequeue ( );                  // Dequeue element from front
    public void clear ( );                      // Remove all elements from queue

    // Queue status operations
    public Object getFront();             // get item at front without removing
    public boolean isEmpty ( );           // Is Queue empty?
    public boolean isFull ( );            // Is Queue full?
    public int size();                    // returns number of items in Queue
    public String toString( );            // Outputs a String representation of Q
 }
```

Pre-Lab Visualization

1) To make sure you understand how a Queue works, show the contents of the Queue at each step while performing the following operations on **Queue<Character> testQ**. Just list the contents from left to right, where left is the front of the queue:

Operation	Resulting Queue items (front on left)
testQ.enqueue('a');	
testQ.enqueue('b');	
testQ.enqueue('c');	
testQ.dequeue();	
testQ.dequeue();	

2) Show the contents of the Queue at each step while performing the following operations on Queue testQ. Just list the contents from left to right, where left is the front of the queue:

Operation	Resulting Queue items (front on left)
testQ.enqueue('a');	
testQ.enqueue('b');	
testQ.enqueue(testQ.dequeue());	
testQ.enqueue(testQ.getFront());	
testQ.dequeue();	

Solve the StoreSim problem

[See end of lecture 4a slides for an animation of the data involved]
In this exercise, you use a queue to simulate the flow of customers through a checkout line in a store. In order to create this Simulation, you must model both the passage of time and the flow of customers through the line. You can model time using a loop in which each pass corresponds to a set time interval-one minute, for example. You can model the flow of customers using a queue in which each element corresponds to a customer in the line.

In order to complete the simulation, you need to know the rate at which customers join the line, as well as the rate at which they are served and leave the line. Suppose the checkout line has the following properties.

- One customer is served and leaves the line every minute (assuming there is at least one customer waiting to be served during that minute).
- Between zero and two customers join the line every minute, where there is a 50% chance that no customers arrive, a 25% chance that one customer arrives, and a 25% chance that two customers arrive.

You can simulate the flow of customers through the line during a time period *n* minutes long using the following algorithm:

```
Initialize the queue to empty.
for ( minute = 0 ; minute < n ; minute++ )
    If the queue is not empty, then remove the customer at the front of the queue.
    Compute a random number k between 0 and 3.
    If k is 1, then add one customer to the line. If k is 2, then add two customers
    to the line. Otherwise (if k is 0 or 3), do not add any customers to the line.
```

Step 1: Using the program shell given in the file *StoreSim.java* as a basis, create a program that uses an LQueue (a linked Queue) to implement the model described above. Your program should update the following information during each simulated minute, that is, during each pass through the loop:

- The total number of customers served
- The combined length of time these customers spent waiting in line
- The maximum length of time any of these customers spent waiting in line

In order to compute how long a customer waited to be served, you need to store the "minute" that the customer was added to the queue as part of the queue element corresponding to that customer.

Step 2: Use your program to simulate the flow of customers through the line and complete the following table. Note that the average wait is the combined waiting time divided by the total number of customers served.

Total Sim Time in minutes	Total number of customers served	Average wait	Longest wait
30			
60			
120			
480			

Part B Lists

Resources

- Chapter 12

In this part of the lab, you will develop two different List applications, a DNA counting method and a simple ASCII slide show program, that use Lists.

Java Files

- *AList.java, LList.java*
- *CountBases.java*
- *SlideShow.java*
- *ListInterface.java*

Introduction

This laboratory focuses on the most general purpose of data structures: the list. The elements in a list are ordered by index, similar to an array. However, lists provide extra tools such as the ability to insert, remove and search for items within the list. The List interface we will be using for this lab is defined below (note the use of generic type specifiers).

```
public interface ListInterface<T>
{
   public void add(T newEntry); // adds newEntry to end of list
   public boolean add(int newPosition, T newEntry); // inserts newEntry at newPosition
   public T remove(int givenPosition); // removes item at givenPosition and returns it
   public void clear(); // removes all items from list
   public boolean replace(int givenPosition, T newEntry);
                     // replaces item at givenPosition with newEntry

   public T getEntry(int givenPosition);  // returns item at givenPosition
   public boolean contains(T anEntry);  // returns true if anEntry is in list
   public int getLength(); // returns number of entries in list
   public boolean isEmpty(); // returns true if list isEmpty
   public Object[] toArray();   // returns an Object array containing all items in list
} // end ListInterface
```

Pre-Lab Visualization

1) To make sure you understand how a List works, show the contents of the List at each step while performing the following operations on List myList. Just list the contents from left to right:

Operation	Resulting List items (item 1 on left)
myList.add("A");	
myList.add("B");	
myList.add("C");	
myList.add("D");	
myList.add(1, "one");	
myList.add(1, "two");	
myList.add(1, "three");	
myList.add(1, "four");	

2) Show the contents of the List at each step while performing the following operations on List myList. Just list the contents from left to right:

Operation	Resulting List items (item 1 on left)
myList.add("alpha");	
myList.add(1, "beta");	
myList.add("gamma");	
myList.add(2, "delta");	
myList.add(4, "alpha");	
myList.remove(2);	
myList.remove(2);	
myList.replace(3, "delta");	

Define the method CountBases

The genetic information encoded in a strand of deoxyribonucleic acid (DNA) is stored in the purine and pyrimidine bases (adenine, guanine, cytosine, and thymine) that form the strand. Biologists are keenly interested in the bases in a DNA sequence because these bases determine what the sequence docs.

By convention, DNA sequences are represented using lists containing the letters 'A', 'G','C', and 'T' (for adenine, guanine, cytosine, and thymine, respectively). The following method computes one property of a DNA sequence-the number of times each base occurs in the sequence.

void countBases (AList dnaSequence)

Input:
dnaSequence: contains the bases in a DNA sequence encoded using the characters 'A', 'G','C', and 'T'

Output:
aCount, cCount, tCount, gCount: the number of times the corresponding base appears in the DNA sequence.

Step 1: Implement this method and add it to the program in the file TestDNA.java. Your implementation should manipulate the DNA sequence using the operations in the List ADT. An incomplete definition for this method is given in the file TestDNAjava.

Step 2: The program in the file TestDNAjava reads a DNA sequence from the keyboard, calls the countBases ( ) method, and outputs the resulting base counts. Test your method by running it on DNA sequences of different lengths and various combinations of bases.

SlideShow Application

List elements need not be one of Java's built-in data types. The following code fragment, for example, defines the programmer-defined class `Slide` (which is included at the bottom of the SlideShow.java file).

By using generic angle brackets <>, we can make a list of slides. Thus a slide show presentation can be represented as a list of slides.

```
class Slide
{
        // constants
        static final int SLIDE HEIGHT 10;              // Slide dimensions
        static final int SLIDE WIDTH 36;

        // Data members
        private String [ ] image =                     // Slide image
                new String [SLIDE_HEIGHT] ;
        private long pause;                            // Seconds to pause

        public boolean read ( Scanner  inputFile )
        // Read a slide from the file. Returns false at EOF.
        {

        }
```

```
    public void display ( )                                  // Display a slide and pause.
    {
    }
}
```

Step 1: Using the program shell given in the file SlideShow.java as a basis, create a program that reads a list of slides from a file (slides.txt) and displays the resulting slide show from beginning to end. Your program should pause for the specified length of time after displaying each slide. It then should clear the screen (by scrolling, if necessary) before displaying the next slide.

Assume that the file containing the slide show consists of repetitions of the following slide descriptor (see file slides.txt in your lab4b folder)

```
    Time
    Row 1
    Row 2
    …
    Row 10
```

where Time is the length of time to pause after displaying a slide (in seconds) and Rows 1 to 10 form a slide image (each row is 35 characters long).

You can pause while displaying a slide by putting the current thread object to sleep for the desired number of milliseconds, as shown in SleepDemo.

Step 2: Test your program using the slide show in the file slides.txt.