**CSC 300 Project Fall 2019**

**Deque – The Double Ended Queue**

Submit via e-mail. All parts of assignment must be submitted in a single e-mail with multiple attachments. E-mail address is:

[csc300csudh@gmail.com](mailto:csc300csudh@gmail.com)

Each program is to be submitted in a separate file with the file name being the class name with extension .java. I only need the source file.

Note: If you want to use a LinkedList or ArrayList as opposed to DoublyLinkedList to implement Deque and in the Simulation where DLList is identified you may do so, but you then lose the opportunity for extra credit points (up to 8 extra points since this project is worth 20% of your grade.).

If you want to implement Deque and the simulation without the DLList, you can still submit a DLList implementation for up to 20 extra credit points. It will be tested separately.

If you are submitting as a group, only one person needs to submit. Put the names of all (up to three) team members in the e-mail message.

**The problem:**

1. **Implement a Doubly-Linked List (40)**

To do this problem you will need to download LList.java and Node.java. Call the modified files DLList.java and DLNode.java. You will also need ListInterface.java, but you don’t need to modify this unless you want to make the changes.

A doubly linked list is a chain of nodes, each of which has three components: data, next, and previous. The data component holds a reference to some data, the next component holds the address of the next node in the list, and the previous component holds the address of the previous node in the list. See Figure below:



Create DLNode by modifying Node so that it also includes previous.

Create a class DLList<E> that implements the ListInterface <E> using a doubly linked chain of nodes (DLNode). Include all the LList<E> methods.

Include additional methods:

• E previous()

if current == null, reports an error and terminates the application; otherwise returns the data of the current node and sets current equal to current.previous

Updated:

E data = (E) current.getData();

current = current.getPrevious();

return data;

• boolean hasPrevious()

returns true if current variable’s previous is not null.

Modify the following to make them work from back to from if the index is greater than or equal to half the length, thereby improving efficiency:

void add(int index, E x)

E get (int index)

E remove(int index)

E set (int index, E x)

Remember to write a test driver to test your methods before moving forward. You could modify the TestLLists.java file to run these tests.

1. **Implement a generic Deque class (20)**

Overview

A deque is an ordered list of elements is an ordered list of elements such that insertions and deletions can take place from either end of the list.

The following is the interface definition for a Deque. This file may be found under the PROJECT/PROJECT DEFINITION tab. You will need to implement this interface.

public interface DequeInterface <E>

{

public void addFront(E item);

//Inserts at from of the dequeue (item 0)

public void addBack(E item);

//Inserts at end of the dequeue

public E removeFront();

//Remove from the front (item 0) of the deque and return the reference to the object removed

//Return null if empty

public E removeBack();

//Remove from the back of the deque and return the reference to the object removed

//Return null if empty

public E peekFront();

//Return reference to item at the front of the deque

//Return null if empty

public E peekBack();

//Return reference to item at the back of the deque

//Return null if empty

public int size();

//Returns number of elements in the deque

public boolean isEmpty();

//Returns true if the deque has no elements, otherwise false

Our goal is a dynamic implementation of Deque<E> so that each method executes in a more constant time. That is, each method should require the same amount of time no matter the size of the Deque. Several possible implementations and their drawbacks include:

**Implement a generic Deque class using a doubly linked list.**

To implement Deque use must implement all of the methods of the DequeInterface interface. (You can see how we implemented Stack using ArrayList as an example.)

There is one instance variable for a Generic DLList. (call it dLList).

There should be a simple default constructor for Deque that creates the new DLList with reference variable dLList.

1. **Implement the following simulation using a Deque queue and two Doubly-Linked Lists. (80)**

General Description:

When visiting the Senate Washington DC, you may want to go to the visitor’s gallery. There are 90 seats in the gallery. Spectators enter in groups of 35, but only when a block of 35 seats become available. If there are less than 35 waiting you can bring them in, as long as there are at least 35 spots in the gallery. Each person may stay as long as they like (as we describe below). VIPs get to go the front of the line.

Write a program using a Deque to simulate the waiting line for the Senate gallery. Assume that the gallery is initially empty and that 100 people are waiting on the line at time 0, including 4 VIPs. (Note: you must use Deque for the queues)

After the doors are opened 3 regular persons arrive every minute, and one VIP every 5 minutes.

20% of all persons stay in the gallery for 5 minutes, 60% for 10 minutes, and 20% for 20 minutes.

Simulate simMinutes.

Calculate the average wait-time for regular people, and the average wait time for VIPs including those who got into the gallery and are still in the gallery and those who have already left.

Notes on the implementation.

Implement the following classes:

**Visitor (20)**

Visitor includes the following instance variables

private int timeOntoQueue;

private int timeOutOfQueue;

private int totalTimeInQueue;

private int timeInGallery;

private int timeRemainingInGallery;

A private static Random variable randy initialized with a seed of 2.

A single constructor which accepts the time the Visitor got online

The constructor calls setTimeOnQueue with the time the Visitor got online

The constructor calls setTimeInGallery with no parameters

Visitor has the following methods:

* setTimeOntoQueue(int timeOntoQueue) which sets the time timeOntoQueue
* setTimeOutOfQueue(int timeOutOfQueue) which sets the timeOutOfQueue and calculates and sets the totalTimeInQueue
* setTimeInGallery() which uses the random variable to set the time as described above
* decrementTimeRemainingInGallery() to decrement the time by one minute
* The two accessors getTotalTimeInQueue() and getTimeRemainingInGallery()

**VIPVisitor (5)**

VIPVisitor extends Visitor

It has a **static** vipCounter initialized to 0

It has a private String vipID

The single constructor accepts the timeOntoQueue and calls super. It then call setVIPID()

The two new methods are:

* setVIPID() which increments vipCounter and creates a vipID which is a concatenation of “VIP” and the value of vipCounter
* getVIPID() returns the String with the vipID

**RegularVisitor (5)**

RegularVisitor extends Visitor

It has a **static** regularCounter initialized to 0

It has a private String regularID

The single constructor accepts the timeOntoQueue and calls super. It then call setRegularID()

The two new methods are:

* setRegularID() which increments regularCounter and creates a regularID which is a concatenation of “Regular” and the value of regularCounter
* getRegularID() returns the String with the regularID

**SenateGallery (40)**

This class is the heart of the simulation model.

There are 4 instance variables:

A Deque of type Visitor called waitingLine

A DLList of type Visitor called visitorInGallery

A DLList of type Visitor called visitorFinished

A String variable called date

There are two SenateGallery constructors.

The default constructor calls setDate with the empty String

The constructor with a single parameter which is the String containing the date

There are setDate and getDate mutator/accessor methods

There are the following additional methods:

* initializeSenateGallery() does the following:

Assume this is time 0

Create 96 regularVisitors and add them to the waitingLine

Create 4 VIPVisitors and add them to the front of the waitingLine (just add them at the beginning as they arrive)

Remove the first seventy Visitors at any time from the front of the waitingLine, set their timeOutOfQueue as 0, and add them to visitorInGallery

* runSenateGallerySimulation(int simMinutes)

//Now we run the simulation minute by minute starting from 1 until the time equals one less than the simMinutes

For each minute starting at 1

Create three RegularVisitors with the current time and add to the waitingLine

Every five minutescreate one VIPVisitor with the current time and add to the waitingLine

//Now remove people from the Gallery

For each Visitor currently in the gallery (The DLL visitorInGallery)

Decrement the timeRemainingInGallery using the getTimeRemainingInGallery() method

If the timeRemainingInGallery is zero

Remove the Visitor from visitorInGallery

Place the Visitor in the visitorFinished DLList

//Now move people into the Gallery (if possible)

While there are at least 35 seats available (i.e. 55 or less in the Visitor gallery) and if there is anyone on the waitingLine

Calculate number to bring in (up to 35)

Remove Visitor from the front of waitingLine

setTimeOutOfQueue for the Visitor to the current minute

Add the Visitor to visitorInGallery

* outputStatistics()

Print out the date

Print out the Total who finished viewing

Print out the Total who are still in gallery

Total who are still waiting

//Now you will need to use polymorphism

Calculate the average wait time before entering the Gallery for all VIPVisitors who are either viewing or finished viewing as a double. Print this out in a sentence.

Calculate the average wait time before entering the Gallery for all RegularVisitors who are either viewing or finished viewing as a double. Print this out in a sentence.

**CreateGallerySimulation class (10)**

This class has the main method.

It requests the date of the simulation as a String

It creates the SenateGallery object with the date as the parameter..

Ask for the number of minutes to simulate.

It then calls the initializeSenateGallery() method, the runSenateGallerySimulation(int sim Minutes) method, and the outputStatistics() method of the SenateGallery object.

**Sample Input and output**

Please enter the Month Day and Year of the simulation: November 29, 2019

Please enter the length of the simulation in minutes: 480

The date of the visit to the Senate Gallery is November 29, 2019

Total who finished viewing is 1595

Total who are still in gallery is 37

Total who are still waiting is 0

The average wait tome for VIP Visitors who are in the gallery or finished viewing is 0.10 seconds

The average wait tome for Regular Visitors who are in the gallery or finished viewing is 0.38 seconds