Queues, Deques and Priority Queues

- References:
  - Text book : Chapter 10 and Chapter 11

Part I: Specification (chapter 10)

1. ADT Queue

- Definition : A queue is an ordered list in which all insertion take place at one end and all deletions take place at opposite end. It is also known as First-In-First-Out (FIFO) lists.

- Applications :

  Waiting lists (banks, gas stations, phone queues, process queue in kernel, customer services etc)

- ADT Queue Operations (interface)

```
public interface QueueInterface<T>
{

  /** Task: Adds a new entry to the back of the queue.
   *  @param newEntry  an object to be added */
  public void enqueue(T newEntry);

  /** Task: Removes and returns the entry at the front of the queue.
   *  @return either the object at the front of the queue or, if the
   *          queue is empty before the operation, null */
  public T dequeue();

  /** Task: Retrieves the entry at the front of the queue.
   *  @return either the object at the front of the queue or, if the
   *          queue is empty, null */
  public T getFront();
```

```
  /** Task: Detects whether the queue is empty.
   *  @return true if the queue is empty, or false otherwise */
  public boolean isEmpty();

  /** Task: Removes all entries from the queue. */
  public void clear();
} // end QueueInterface
```

- Example usage of stack

```
QueueInterface<String> myQueue = new LinkedQueue<String>();
myQueue.enqueue("Jim");
myQueue.enqueue("Jess");
myQueue.enqueue("Jill");
myQueue.enqueue("Jane");
myQueue.enqueue("Joe");

String front = myQueue.getFront(); // returns "Jim"
System.out.println(front + " is at the front of the queue.");

front = myQueue.dequeue();         // removes and returns "Jim"
System.out.println(front + " is removed from the queue.");

myQueue.enqueue("Jerry");

front = myQueue.getFront();        // returns "Jess"
System.out.println(front + " is at the front of the queue.");

front = myQueue.dequeue();         // removes and returns "Jess"
System.out.println(front + " is removed from the queue.");
```

2. Application: Introduction to simulation

- A technique for modeling the behavior of both natural and human-made system.

- Example : Highway Traffic & Bank Customer Traffic

- Purpose of simulation: To generate statistics that summarize the performance of the simulated system

- Example : Simple bank simulation

  Assume a bank with one teller (and one waiting line) to serve customers

  Goal : measure the average waiting time of customers

  Usually, customer's arrival time and the duration to process customer transaction are generated according to certain mathematical distributions.

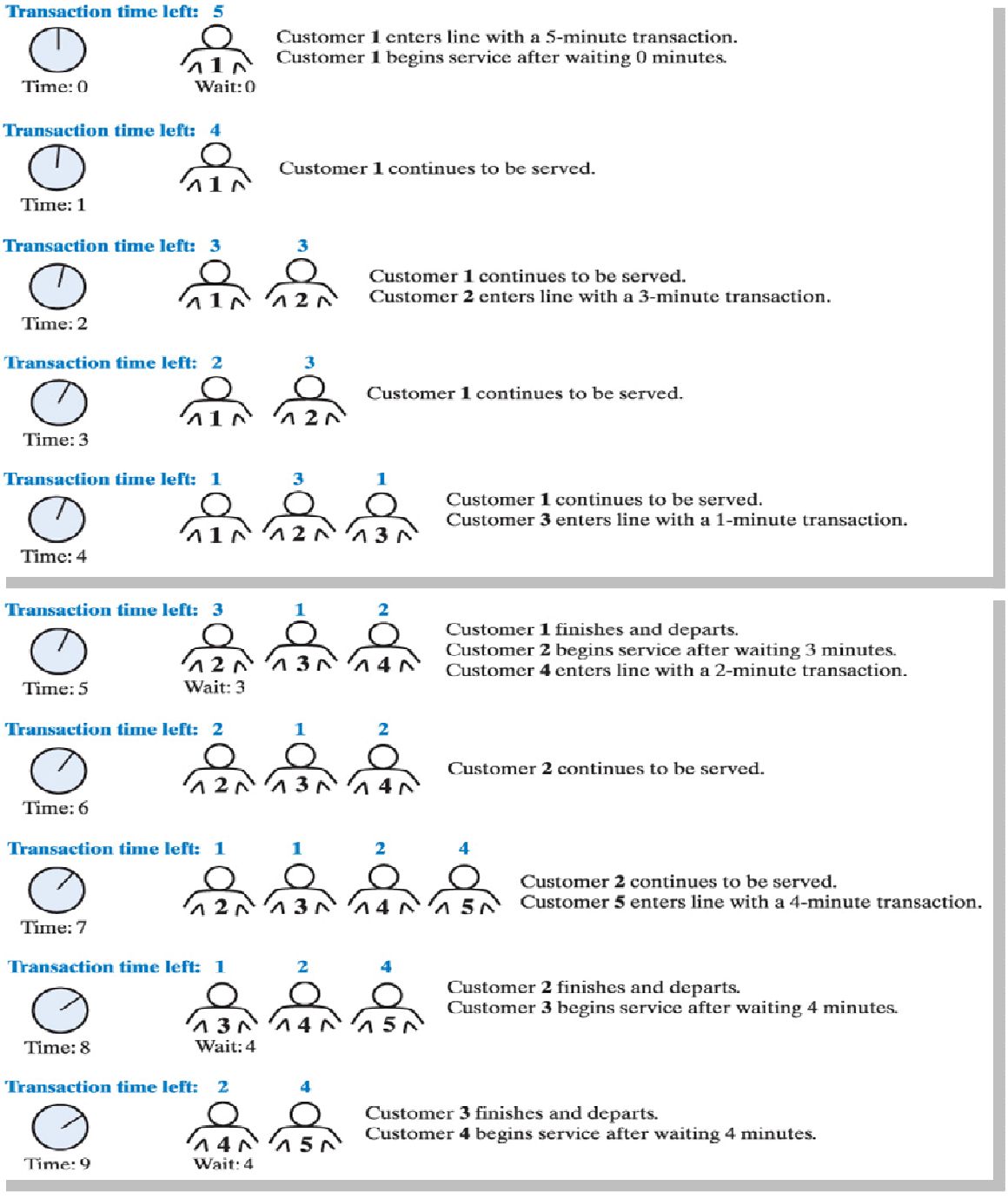
  For simplicity, assume arrival events (arrival time and transaction time) are given in a file and are ordered according to arrival time

ID	Arrival T.	Transaction T.
c1	0	5
c2	2	3
c3	4	1
c4	5	2
c5	7	4

  Algorithm outline:

  - The simulation will start at time 0
  - Loop until no more customer:
    - if a new customer arrive, s/he enter the waiting queue
    - if the teller is free and there are waiting customers, teller serves the first customer in the queue

- if teller is not free, customers continue to wait in queue
- Increment current time by 1
- Go to loop



Analysis : Total waiting Time = leave queue time – enter queue time

c1	0-0	= 0
c2	5-2	= 3
c3	8-4	= 4
c4	9-5	= 4
c5	11-7	= 4

Average = 15/5 = 3

- Time-driven : Increase current time counter by 1 each time, then determine whether or not there are events to serve. If yes, serve all events, update the system state accordingly. See example above.

- Event-driven : Determine the next event(s) which can be accomplished, set the current time counter to the event time, serve event(s), update the system state accordingly. Note : current time counter always increases.

- Java program for above problem (time-driven simulation)

  Objects: WaitLine and Customer

  WaitLine: Simulate customers entering and leaving waiting line
  Display number served, total/average waiting time,
  and number left in line

  Customer: customer id, arrival time and transaction time

```
/** Simulates a waiting line. */
public class WaitLine
{
      private QueueInterface<Customer> line;
      private int numberOfArrivals;
      private int numberServed;
      private int totalTimeWaited;

      public WaitLine()
      {
            line = new LinkedQueue<Customer>();
            reset();
      } // end default constructor

      /** Task: Simulates a waiting line with one serving agent.
       *  @param duration  the number of simulated minutes
       *  @param arrivalProbability  a real number between 0 and 1, and the
       *                             probability that a customer arrives at
       *                             a given time
       *  @param maxTransactionTime  the longest transaction time for a
       *                             customer */
      public void simulate(int duration, double arrivalProbability,
                           int maxTransactionTime)
      {
            int transactionTimeLeft = 0;  // current customer remaining transaction time

            for (int clock = 0; clock < duration; clock++)  // main loop
            {
                  // use random number to generate new customer
                  // this may be replaced by reading from file
                  if (Math.random() < arrivalProbability)
                  {     // if there is new customer
                        numberOfArrivals++;
                        int transactionTime = (int)(Math.random() * maxTransactionTime +1);
                        Customer nextArrival = new Customer(clock, transactionTime,
                                                            numberOfArrivals);
                        line.enqueue(nextArrival);
                        System.out.println("Customer " + numberOfArrivals
                                           + " enters line at time " + clock
                                           + ". Transaction time is " + transactionTime);
                  } // end if
```

```
            if (transactionTimeLeft > 0)  // still serving current customer
               transactionTimeLeft--;
            else if (!line.isEmpty())   // serve next customer in line
            {
               Customer nextCustomer = line.dequeue();
               transactionTimeLeft = nextCustomer.getTransactionTime() - 1;
               int timeWaited = clock - nextCustomer.getArrivalTime();
               totalTimeWaited = totalTimeWaited + timeWaited;
               numberServed++;
               System.out.println("Customer " + nextCustomer.getCustomerNumber()
                           + " begins service at time " + clock
                           + ". Time waited is " + timeWaited);
            } // end if
         } // end for
      } // end simulate

      /** Task: Displays summary results of the simulation. */
      public void displayResults()
      {
         System.out.println();
         System.out.println("Number served = " + numberServed);
         System.out.println("Total time waited = " + totalTimeWaited);
         double averageTimeWaited = ((double)totalTimeWaited) / numberServed;
         System.out.println("Average time waited = " + averageTimeWaited);
         int leftInLine = numberOfArrivals - numberServed;
         System.out.println("Number left in line = " + leftInLine);
      } // end displayResults

      /** Task: Initializes the simulation. */
      public final void reset()
      {
         line.clear();
         numberOfArrivals = 0;
         numberServed = 0;
         totalTimeWaited = 0;
      } // end reset
} // end WaitLine
```

Sample usage:

```
WaitLine customerLine = new WaitLine();
customerLine.simulate(20, 0.5, 5);
customerLine.displayResults();
```

3. Java class library: java.util.Queue (Interface)

http://download.oracle.com/javase/6/docs/api/java/util/Queue.html

public interface Queue<E> extends Collection<E>

extend from interface Collection → all methods from Collection, including add(), isEmpty(), clear(), size(), iterator() etc

additional methods:

	Throws exception	*Returns special value*
Insert	`add(e)`	`offer(e)`
Remove	`remove()`	`poll()`
Examine	`element()`	`peek()`

Example usage: LinkedList implement Queue

```
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Queue;
```

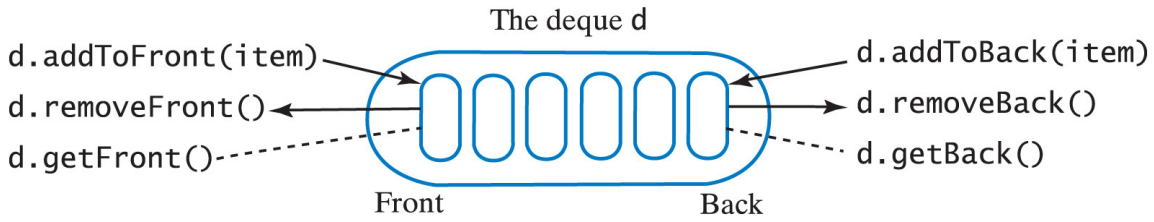
```
public class QueueExample {

   public static void main(String[] args) {
      Queue<String> qe=new LinkedList<String>();
      qe.add("b");
      qe.add("a");
      qe.add("c");
      qe.add("e");
      qe.add("d");

      Iterator<String>  it=qe.iterator(); // will cover iterator later

      System.out.println("Initial Size of Queue :"+qe.size());

      while(it.hasNext())
      {
         String iteratorValue=(String)it.next();
         System.out.println("Queue Next Value :"+iteratorValue);
      }

      // get value and does not remove element from queue
      System.out.println("Queue peek :"+qe.peek());

      // get first value and remove that object from queue
      System.out.println("Queue poll :"+qe.poll());

      System.out.println("Final Size of Queue :"+qe.size());
   }
}
```

Output:

```
Initial Size of Queue :5
Queue Next Value :b
Queue Next Value :a
Queue Next Value :c
Queue Next Value :e
Queue Next Value :d
Queue peek :b
Queue poll :b
Final Size of Queue :4
```

4. ADT Deque

- Deque (double-ended queue) allows add, remove & retrieve at both ends (like Stack + Queue operations)



- ADT Deque Operations (interface):

```
public interface DequeInterface<T>
{
  public void addToFront(T newEntry);
  public void addToBack(T newEntry);
  public T removeFront();
  public T removeBack();
  public T getFront();
  public T getBack();
  public boolean isEmpty();
  public void clear();
} // end DequeInterface
```

- Sample usage:

```
DequeInterface<String> myDeque = new LinkedDeque<String>();
myDeque.addToFront("Jim");
myDeque.addToBack("Jess");
myDeque.addToFront("Jill");
myDeque.addToBack("Jane");          // order: Jill Jim Jess Jane
String name = myDeque.getFront();   // Jill
myDeque.addToBack(name);            // Jill Jim Jess Jane Jill
myDeque.removeFront();              // remove Jill from front
myDeque.addToFront(myDeque.removeBack());  // Jill Jim Jess Jane
```

- Java class library: java.util.Deque (interface)

  New in Java 1.6:

  http://download-llnw.oracle.com/javase/6/docs/api/java/util/Deque.html

  public interface Deque<E> extends Queue<E>

	First Element (Head)		Last Element (Tail)	
	Throws exception	*Special value*	*Throws exception*	*Special value*
Insert	addFirst(e)	offerFirst(e)	addLast(e)	offerLast(e)
Remove	removeFirst()	pollFirst()	removeLast()	pollLast()
Examine	getFirst()	peekFirst()	getLast()	peekLast()

  Several Known Implementing Classes:

  ArrayDeque, LinkedList

- Sample usage:

```
import java.util.ArrayDeque;
import java.util.Iterator;

public class DequeExample
{
  public static void main(String as[])
  {
      ArrayDeque<String>  adObj = new ArrayDeque<String>();

      //Insertion by using various methods
      adObj.add("Oracle");
      adObj.addFirst("DB2");
      adObj.offerFirst("MySQL");   //returns boolean - true or false
      adObj.offerLast("Postgres");   //returns boolean - true or false
```

```
        //Retrievals
        System.out.println("Retrieving First Element :" +
        adObj.peekFirst());
        System.out.println("Retrieving Last Element  :"+
        adObj.peekLast());

        //Removals
        System.out.println("Removing First  Element  :"+
        adObj.pollFirst());
        System.out.println("Removing Last  Element   :"+
        adObj.pollLast());


        //Reverse traversal
        System.out.println("Remaining Elements :");
        Iterator<String> it = adObj.descendingIterator();
        while(it.hasNext())
        {
        System.out.println(it.next());
        }
    }
```

- Output:

```
Retrieving First Element :MySQL
Retrieving Last Element  :Postgres
Removing First  Element  :MySQL
Removing Last  Element   :Postgres
Remaining Elements :
Oracle
DB2
```

5. ADT Priority Queue

- Organizes objects according to priorities - Contrast to regular queue in order of arrival

- Priority queue examples – a hospital ER; process priorities in kernel; Deposit transaction in bank waiting line etc

- Priority can be specified by an integer- Must define whether high number is high priority or low number (say 0) is high priority

- Objects can specify priority - must have a **compareTo** method

- ADT Priority Queue Operations (interface):

```
public interface PriorityQueueInterface<T extends Comparable<? super T>>
{
    /** Task: Adds a new entry to the priority queue.
     *  @param newEntry  an object */
    public void add(T newEntry);

    /** Task: Removes and returns the item with the highest priority.
     *  @return either the object with the highest priority or, if the
     *          priority queue is empty before the operation, null */
    public T remove();

    /** Task: Retrieves the item with the highest priority.
     *  @return either the object with the highest priority or, if the
     *          priority queue is empty, null */
    public T peek();

    /** Task: Detects whether the priority queue is empty.
     *  @return true if the priority queue is empty, or false otherwise */
    public boolean isEmpty();

    /** Task: Gets the size of the priority queue.
     *  @return the number of entries currently in the priority queue */
```

```
    public int getSize();

    /** Task: Removes all entries from the priority queue */
    public void clear();
} // end PriorityQueueInterface
```

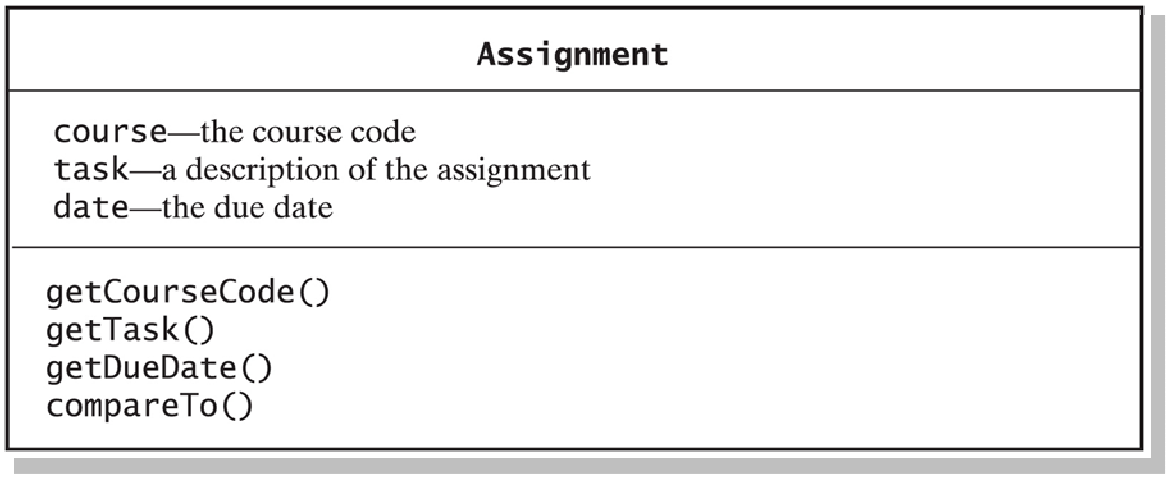
- Sample usage:

  Assume strings have "reversed" alphabetical order priority,
  i.e. "World" has higher priority than "Hello"

```
PriorityQueueInterface<String> myPriorityQueue =
new LinkedPriorityQueue<String>();
myPriorityQueue.add("Jane");
myPriorityQueue.add("Jim");
myPriorityQueue.add("Jill");
String name = myPriorityQueue.remove();  // remove: Jim
myPriorityQueue.add(name);       // add: Jim
myPriorityQueue.add("Jess");     // add: Jess
String name = myPriorityQueue.peek();  // still get Jim
```

- Application: Tracking course assignments

  Assignment class:



  Earliest due date → Highest priority

```
// java.sql.Date provide date comparison,
// return negative value if "this" date is before "other" date
//
// compareTo method in Assignment class:
// want to return positive value if "this" date is early than "other"
// date. So, need to negate to the result of the Date compareTo()

public int compareTo(Assignment other)
{
 return -date.compareTo(other.date);
} // end compareTo
```

AssignmentLog class:

```
import java.sql.Date;
public class AssignmentLog
{
 private PriorityQueueInterface<Assignment> log;

 public AssignmentLog()
 {
   log = new PriorityQueue<Assignment>();
 } // end constructor

 public void addProject(Assignment newAssignment)
 {
   log.add(newAssignment);
 } // end addProject

 public void addProject(String courseCode, String task, Date dueDate)
 {
   Assignment newAssignment =
          new Assignment(courseCode, task, dueDate);
   addProject(newAssignment);
 } // end addProject

 public Assignment getNextProject()
 {
   return log.peek();
 } // end getNextProject
```

```
  public Assignment removeNextProject()
  {
    return log.remove();
  } // end removeNextProject
} // end AssignmentLog
```

Usage:

```
AssignmentLog myHomework = new AssignmentLog();
myHomework.addProject("CSC211", "Pg 50, Ex 2", Date.valueOf("2007-10-21"));
Assignment pg75Ex8 = new
    Assignment("CSC215", "Pg 75, Ex 8", Date.valueOf("2007-10-14"));
myHomework.addProject(pg75Ex8);
// . . .
System.out.println("The following assignment is due next:");
System.out.println(myHomework.getNextProject());
// . . .
```

- Java class library: java.util.PriorityQueue

  http://download.oracle.com/javase/6/docs/api/java/util/PriorityQueue.html

  Several Constructors + operations:

  Note: default "natural ordering" → smallest value first

boolean	add(E o) Adds the specified element to this queue.
void	clear() Removes all elements from the priority queue.
Comparator<? super E>	comparator() Returns the comparator used to order this collection, or null if this collection is sorted according to its elements natural ordering (using Comparable).
Iterator<E>	iterator() Returns an iterator over the elements in this queue.
boolean	offer(E o) Insert the specified element into this priority queue.
E	peek() Retrieves, but does not remove, the head of this queue, returning null if this queue is empty.

E	poll() Retrieves and removes the head of this queue, or null if this queue is empty.
int	size() Returns the number of elements in this collection.

Sample usage:

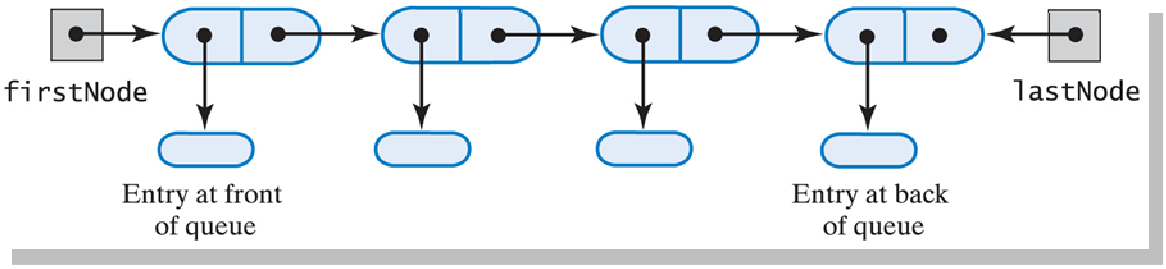
```
import java.util.*;
class PriorityQueueDemo {

   public static void main(String[] args){

      PriorityQueue<String> stringQueue = new PriorityQueue<String>();

      stringQueue.offer("ab");
      stringQueue.offer("abcd");
      stringQueue.offer("abc");
      stringQueue.offer("a");
      stringQueue.offer("abcde");

      //iterator may or may not
      //show the PriorityQueue's order
      System.out.print("Display PQ using iterator:");
      Iterator<String> it = stringQueue.iterator();
      while(it.hasNext())
            System.out.print(it.next()+" ");

      System.out.println("\nDisplay PQ using toString():"+stringQueue);

      System.out.print("Display PQ using poll():");
      while(stringQueue.size() > 0)
            System.out.print(stringQueue.poll()+" ");
      System.out.println();
   }
}
```

Output: // default compareTo is alphabetical order

```
Display PQ using iterator:a ab abc abcd abcde
Display PQ using toString():[a, ab, abc, abcd, abcde]
Display PQ using poll():a ab abc abcd abcde
```

Part II: Implementation (chapter 11)

6. A Singly Linked List Implementation of a Queue



- Using singly linked list : use 2 references, firstNode and lastNode. Insert into back, Delete from the front

```
public class LinkedQueue < T > implements QueueInterface < T >
{
   private Node firstNode; // references node at front of queue
   private Node lastNode;  // references node at back of queue

   public LinkedQueue ()
   {
      firstNode = null;
      lastNode = null;
   } // end default constructor

   public void enqueue (T newEntry)
   {  // allocate new node
      Node newNode = new Node (newEntry, null);
      if (isEmpty ())  // for empty list
         firstNode = newNode;
      else  // for non-empty list
         lastNode.setNextNode (newNode);
      lastNode = newNode;   // update lastNode
   } // end enqueue

   public T getFront ()
   {
      T front = null;
      if (!isEmpty ())
         front = firstNode.getData ();
      return front;
```

```
    } // end getFront


    public T dequeue ()
    {
       T front = null;
       if (!isEmpty ())
       {
          front = firstNode.getData ();
          firstNode = firstNode.getNextNode ();
          if (firstNode == null)  // special case: remove last node
             lastNode = null;
       } // end if
       return front;
    } // end dequeue


    public boolean isEmpty ()
    {
       return (firstNode == null) && (lastNode == null);
    } // end isEmpty


    public void clear ()
    {
       firstNode = null;
       lastNode = null;
    } // end clear


    private class Node
    {
       private T data; // entry in queue
       private Node next; // link to next node

      // Constructors and the methods getData, setData,
      // getNextNode, and setNextNode are here.

    } // end Node
} // end LinkedQueue
```

7. An Array-Based Implementation of a Queue

- A naive strategy

```
#define maxSize 100;
QueueElement queue[maxSize];
int backIndex = frontIndex = -1;

// backIndex+1= next empty cell,
// frontIndex+1 = the first data to be removed
```

Operations :

To check whether or not it is full/empty :
frontIndex = backIndex → empty
backIndex = maxSize -1 → full

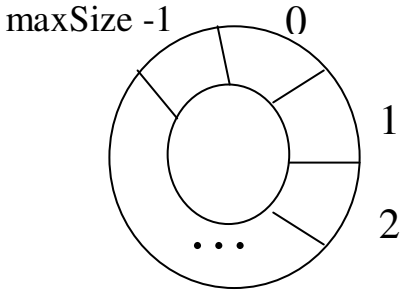
To insert : if not full(backIndex), backIndex = backIndex + 1, insert data to location backIndex.

To delete : if not empty(frontIndex,backIndex), frontIndex = frontIndex+1; delete item at location frontIndex.

Problem : when the backIndex and frontIndex get to the end of queue, i.e. backIndex or frontIndex = maxSize-1. We need to move the entire queue so that the first element is at queue[0] again (during queueFull()). This is too time consuming. To prevent this, we use *circular array*.

- Use circular array to implement queue

  View the array as circular, i.e. after index maxSize – 1 is index 0



  Use a counter to count # of objects in queue

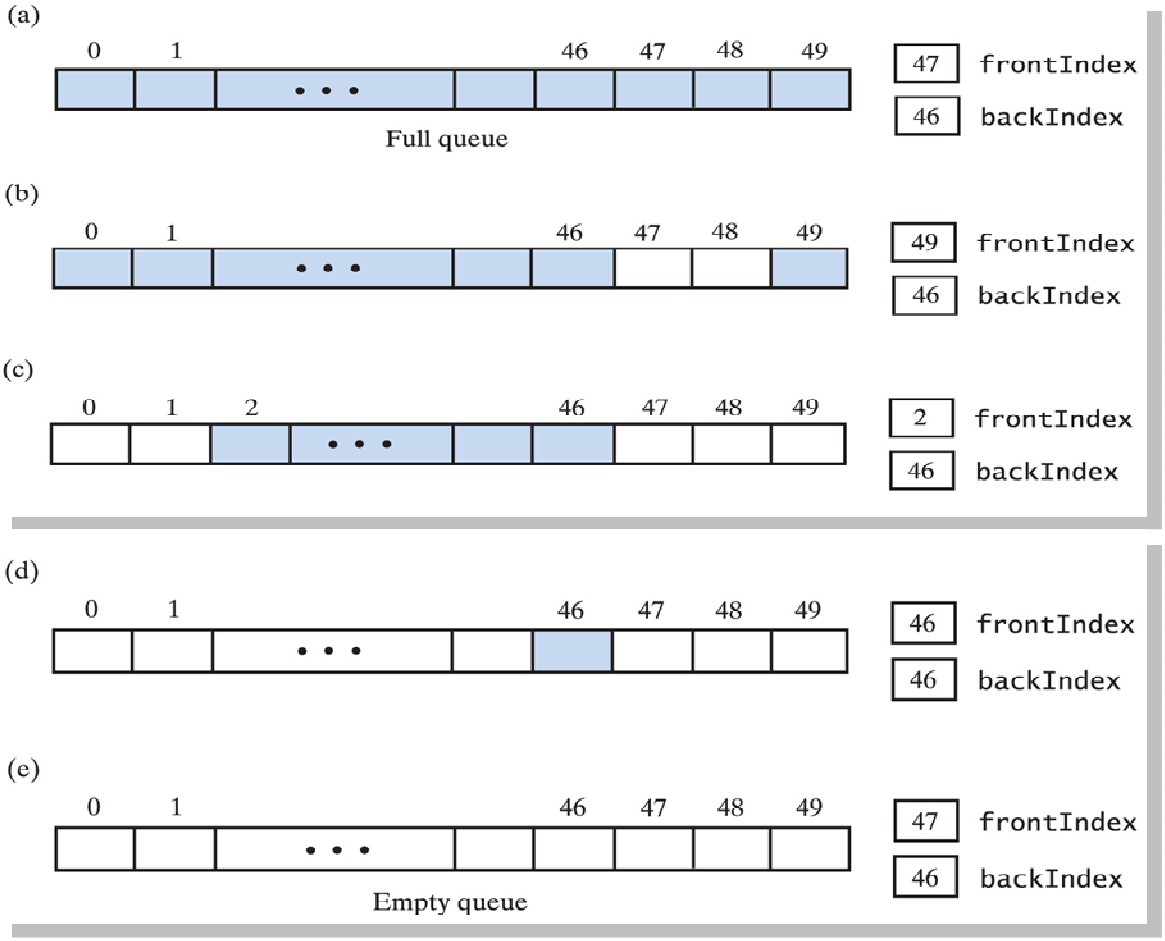
  - Initial values : frontIndex = 0, backIndex = Max_Size-1, count = 0
    - i.e. frontIndex: first entry in the queue; backIndex : last entry in the queue

  - Use modulo "%" operation to move the pointer to next space
    - (take care of wrap-around effect)

    - backIndex = (backIndex+1) % maxSize
    - frontIndex= (frontIndex+1) % maxSize

  - To insert : if ! isFull(), backIndex = (backIndex+1) % maxSize, insert data to location back, count++

  - To delete : if !isEmpty(), remove data from location frontIndex, frontIndex= (frontIndex+1)% maxSize, count--



(a) Array is full, (b) delete 2 entries, (c) remove 3 more entries
(d) after remove all but one entry , (e) remove last entry

Note:

- With circular array: frontIndex == backIndex + 1 →both when queue is empty and when full. Use counter to check empty or full.

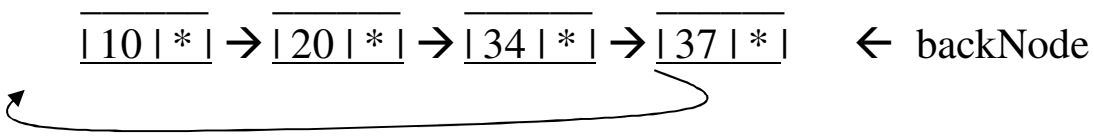
- To avoid using a counter, one array location is unused!

  Skip this topic: A Circular Array with One Unused Location

- When an array is full, you need to copy entries from old array to new array (double size) → need to set frontIndex and backIndex properly

8. Outline: A Circular Singly Linked List Implementation of a Queue

- Using circular linked list with one backNode reference



Outline Insertion : assume newNode is allocated

```
if (backNode != null) {
        newNode.next = backNode.next
        backNode.next = newNode
        backNode = newNode
} else { // no node in queue
        newNode.next = newNode
        backNode = newNode
}
```

Outline Deletion : assume non-empty, 1st node = backNode.next

```
tempNode = backNode.next;  // may want to return deleted node
if (backNode.next = backNode)  // only one node
      backNode = null
else
      backNode.next = (backNode.next).next;
```

9. Outlint: A Doubly Linked Implementation of a Deque

- Each node has next, prev reference data
- The class Deque has firstNode and lastNode data fields

```
public void addToBack (T newEntry)
{
   DLNode newNode = new DLNode (lastNode, newEntry, null);
   if (isEmpty ())  // special case : empty list
      firstNode = newNode;
   else                  // general case
      lastNode.setNextNode (newNode);
   lastNode = newNode;
} // end addToBack
```

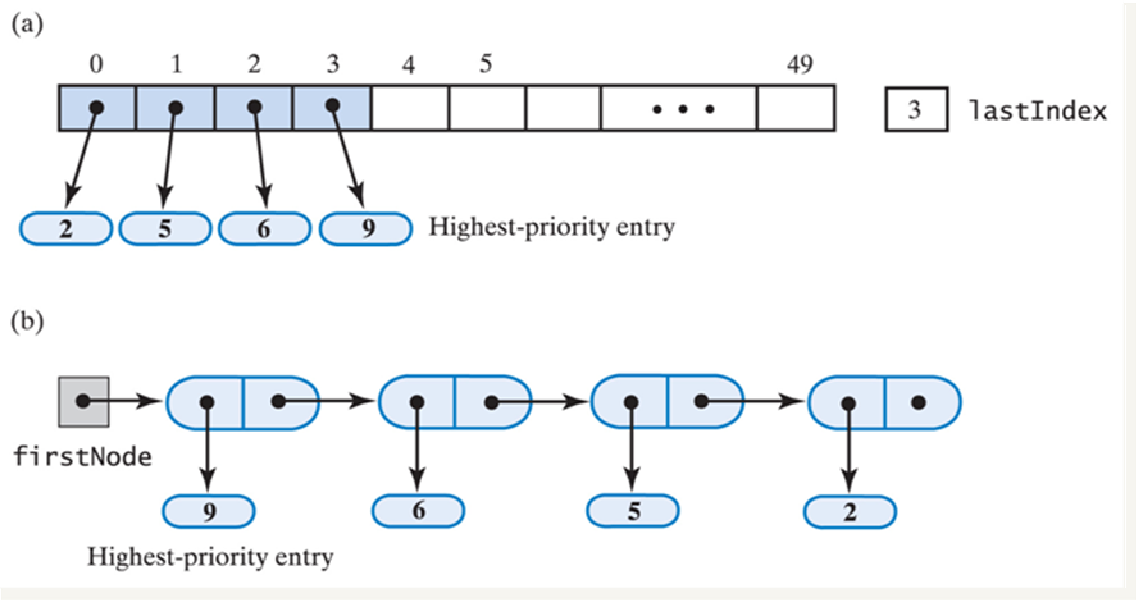
```
public void addToFront (T newEntry)
{
   DLNode newNode = new DLNode (null, newEntry, firstNode);
   if (isEmpty ())   // special case
      lastNode = newNode;
   else                  // general case
      firstNode.setPreviousNode (newNode);
   firstNode = newNode;
} // end addToFront
```

```
public T removeFront ()
  {
     T front = null;
     if (!isEmpty ())
     {
        front = firstNode.getData ();
        firstNode = firstNode.getNextNode ();  // update firstNode
        if (firstNode == null) // special case: removed last node
           lastNode = null;
        else                          // general case
           firstNode.setPreviousNode (null);
     } // end if
     return front;
  } // end removeFront



  public T removeBack ()
  {
     T back = null;
     if (!isEmpty ())
     {
        back = lastNode.getData ();  // update lastNode
        lastNode = lastNode.getPreviousNode ();
        if (lastNode == null)  // special case: removed last node
           firstNode = null;
        else                          // general case
           lastNode.setNextNode (null);
     } // end if
     return back;
  } // end removeBack
```

<< How to use array to implement a Deque ?? >>

10: Simple Implementation of a Priority Queue

- Maintain sorted order in Array or Linked-list
- Insert: insert new entry in proper location
- Delete/retrieve: array – last entry; linked list – first entry



- We will look at more efficient way to implement priority queue later