Queue - a **F**irst object
In
First
Out

All insertions to the <u>rear</u> of the list
All deletions to the <u>front</u> of the list

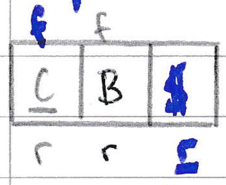
Great for storing pending work, scheduling algorithms, any application where data is processed in order of arrival

q: ~~DC9~~, ~~F100~~, F100 ↗
e. DC9 ⟶
q. dequeue();

MD80, <u>F100</u>
↳ q.enqueue(q.front());

q. dequeue();
printQueue(q);
↳ ~~F100, F100, DC9,~~
F100, DC9, MD80, F100

<u>Queue class</u>
Implemation code with circular array of size 3 — max size



<u>Client code</u>

size : 1 → 2 → 3 → 2
front : 0 → 0 → 0 → 1
rear : 2 → 0 → 1 → <u>2</u>

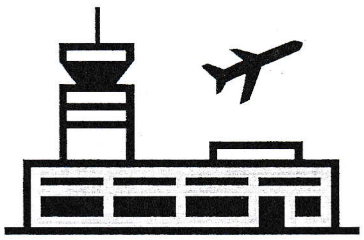
```
Queue q = new Queue();
q.enqueue("C");
q.enqueue("B");
q.enqueue("$");
q.dequeue();
```

"C" is still there. Just lost pointer to RAM

The Plane Queue Program

A Sample Client Program Which <u>Uses</u> the Queue Class

```
public class Plane
{
        public static void main(String[] args)
                     throws CloneNotSupportedException
        {
                Queue q = new Queue();
                Object e;

                q.enqueue("DC9");
                q.enqueue("F100");
                q.enqueue("F100");
                e = q.front();
                q.enqueue(e);
                q.dequeue();
                q.enqueue("MD80");
                q.enqueue( q.front() );
                q.dequeue();
                printQueue(q);
        }

        private static void printQueue (Queue q)
                   throws CloneNotSupportedException
        {
                if (q.isEmpty())
                {
                  System.out.println(
                      "Your Queue is Currently Empty!");
                  return;
                }

                Queue   tempQ  = (Queue) q.clone();

                while (! tempQ.isEmpty())
                {
                        System.out.println(tempQ.front());
                        tempQ.dequeue();
                }
        }
}
```



The Queue Class *(with a circular array implementation)*

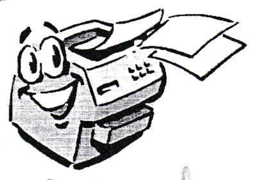
```
public class Queue implements QueueInterface {
  private final int MAX_QUEUE = 50; // maximum size of queue
  private Object[] items;
  private int front, back, count;

  public Queue() {
    items = new Object[MAX_QUEUE];
    front = 0;
    back = MAX_QUEUE-1;
    count = 0;
  }  // end default constructor

  public boolean isEmpty() {
    return count == 0;
  }  // end isEmpty

  public boolean isFull() {
    return count == MAX_QUEUE;
  }  // end isFull

  public void enqueue(Object newItem) {
    if (!isFull()) {
      back = (back+1) % (MAX_QUEUE);
      items[back] = newItem;
      ++count;
    } else {
throw new QueueException("QueueException on enqueue: " + "Queue full");
    }  // end if
  }  // end enqueue

  public Object dequeue() throws QueueException {
    if (!isEmpty()) {
      // queue is not empty; remove front
      Object queueFront = items[front];
      front = (front+1) % (MAX_QUEUE);
      --count;
      return queueFront;
    }
    else {
throw new QueueException("QueueException on dequeue: "+ "Queue empty");
    }   // end if
  }  // end dequeue

public void dequeueAll() {
    items = new Object[MAX_QUEUE];
    front = 0;
    back = MAX_QUEUE-1;
    count = 0;
  }  // end dequeueAll

  public Object front() throws QueueException {
    if (!isEmpty()) {
      // queue is not empty; retrieve front
      return items[front];
    } else {
throw new QueueException("Queue exception on front: " + "Queue empty");
    }  // end if}  // end front} // end Queue
```

Use integer indices to keep track of the size, front index and rear ind

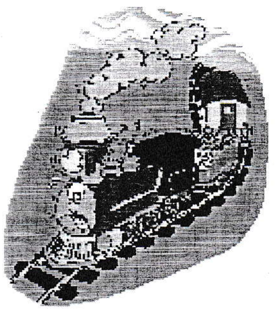
enqueue = (rear + 1) % size

dequeue = (front + 1) % size

← circular array effect moves index to front



Queue Practice



1. The class method below is supposed to count the number of negative integers in a queue. The method has a bug, and does not always work correctly. Identify the bug.

```
private static int countNegs (Queue q)
{
        int first, num, count = 0;
        if (q.isEmpty())
                return 0;

        first = ((Integer) q.front() ).intValue();
        q.dequeue();
        q.enqueue(first);
        if (first < 0)
                count++;

        while (q.front() != first)
        {
                num = ( (Integer) q.front() ).intValue();
                q.dequeue();
                q.enqueue(num);
                if (num < 0)
                  count++;
        }
        return count;
}
```

-5, 0, -2, -1, 4, $\boxed{-5}$, 0, -2, ...

reinsert what was deleted

Bug: All integers are not unique

2. <u>Problem</u>. Consider a network of railroad cars numbered 1, 2, and 3 on the right track that are to be permuted and moved along the left track. A car may be moved directly onto the left track, or it may be shunted onto the siding (which acts like a queue) to be removed at a later time and placed onto the left track. Find all possible permutations of cars that can be obtained. Any permutations not possible?

See pink sheet