

QUEUES

=====

A `_queue_` is also a crippled list. You may read or remove only the item at the front of the queue, and you may add an item only to the back of the queue. The main operations: you may "enqueue" an item at the back of the queue; you may "dequeue" the item at the front; you may examine the "front" item. Don't be fooled by the diagram; a queue can grow arbitrarily long.

```

===          ===          ===          === -front()-> b
ab. -dequeue()-> b.. -enqueue(c)-> bc. -enqueue(d)-> bcd
===          |          ===          === -dequeue() x 3--> ===
              v          ...
              a          null <-front()-- ===

```

Sample Application: Printer queues. When you submit a job to be printed at a selected printer, your job goes into a queue. When the printer finishes printing a job, it dequeues the next job and prints it.

```

public interface Queue {
    public int size();
    public boolean isEmpty();
    public void enqueue(Object item);
    public Object dequeue();
    public Object front();
}

```

In any reasonable implementation, all these methods run in $O(1)$ time. A queue is easily implemented as a singly-linked list with a tail pointer.

DEQUES

=====

A `_deque_` (pronounced "deck") is a Double-Ended `QUEue`. You can insert and remove items at both ends. You can easily build a fast deque using a doubly-linked list. You just have to add `removeFront()` and `removeBack()` methods, and deny applications direct access to list nodes. Obviously, deques are less powerful than lists whose list nodes are accessible.

Postscript: A Faster Hash Code (not examinable)

Here's another hash code for Strings, attributed to one P. J. Weinberger, which has been thoroughly tested and performs well in practice. It is faster than the one above, because it relies on bit operations (which are very fast) rather than the `%` operator (which is slow by comparison). You will learn about bit operations in CS 61C. Please don't ask me to explain them to you.

```

static int hashCode(String key) {
    int code = 0;

    for (int i = 0; i < key.length(); i++) {
        code = (code << 4) + key.charAt(i);
        code = (code & 0xffffffff) ^ ((code & 0xf0000000) >> 24);
    }

    return code;
}

```