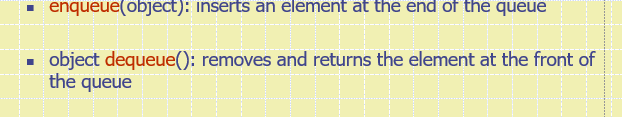
queue这个ADT存储任意object

加入和删除遵循first-in first-out FIFO 先进先出方案

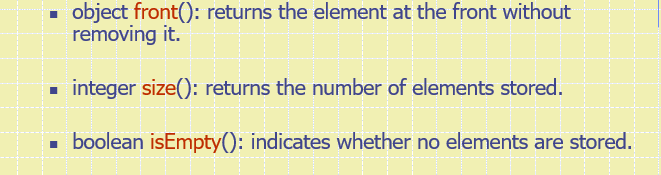
insert在queue后面，removal在queue前面

主要操作

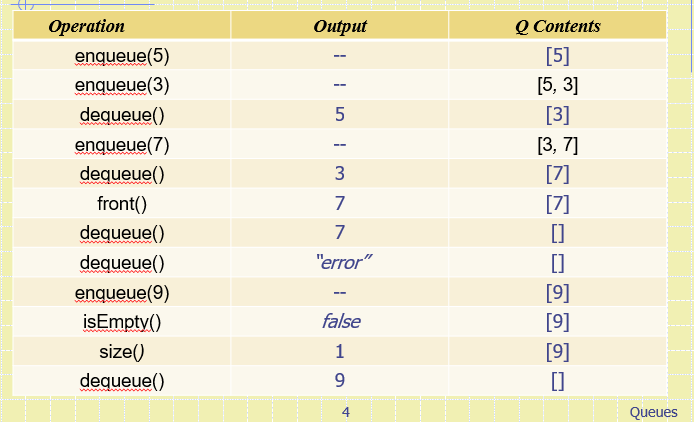


enqueue(object)，把一个element插入queue的末尾

dequeue():把开头移去



如果dequeue或者front对一个空QUEUE,会throw一个 EmptyQueueException



建立在array上的queue

一个queue可以用一个array Q来构建，能承载的数据是固定的

以循环方式来使用大小为N的array

我们可以让Q[0] 作为queue的开头，然而这不是很方便因为每次DQ都需要移动所有元素

（因为开头element被删除了，我们又要Q（0）做开头，就要把Q(1)移到Q(0),Q2移到Q1）

换句话来说，每个dq操作都有complexity of O(n)

所以，我们用两个变量来跟踪front首 和rear尾

f 第一个element的index //首

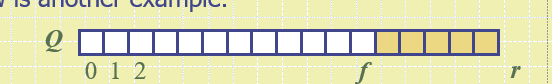
r 最后的elemeent的INDEX+1，//指向尾后面的空index

一开始,我们让个F=R=0，证明F是empty的（F=R就是empty queue），

我们让r始终为空，INERTION就是向Q(r)插入然后r+1

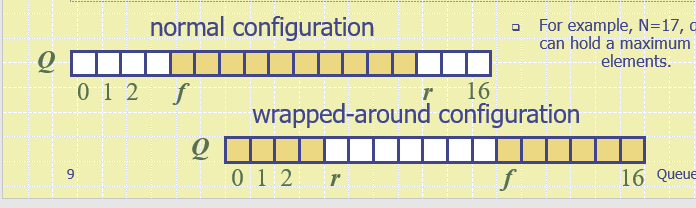
这就让enqueue,dequeue和front的compllexity都变成了O（1）

这样又会导致新问题，当我们重复enq,dq一个元素N次，我们最终会得到f=r=N,这意味着QUEUE是empty

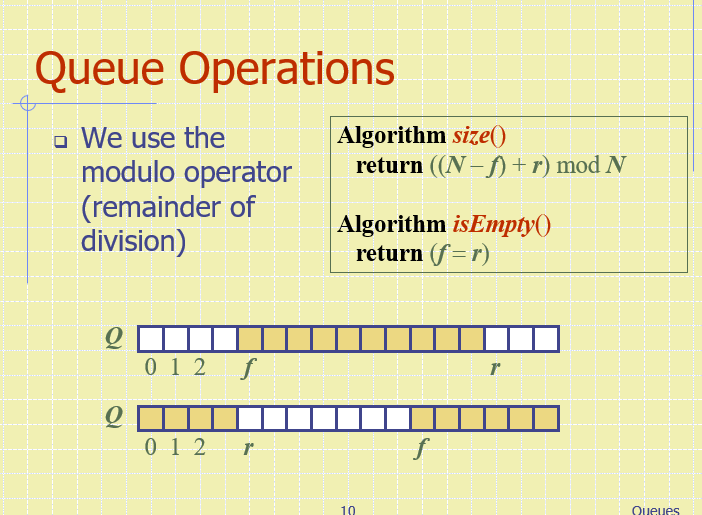
然而，，这时F与R都到了尾端，而ARRAY还有EMPTY ELEMENT， INSERTION会导致 array-out-of-bounds error

所以我们可以将这个array看成一个循环array，当到了末尾Q[N-1,]，又重新回到Q[0]

注意index r必须保持为空，所以一个queue最多Hold n-1个元素



queue的操作

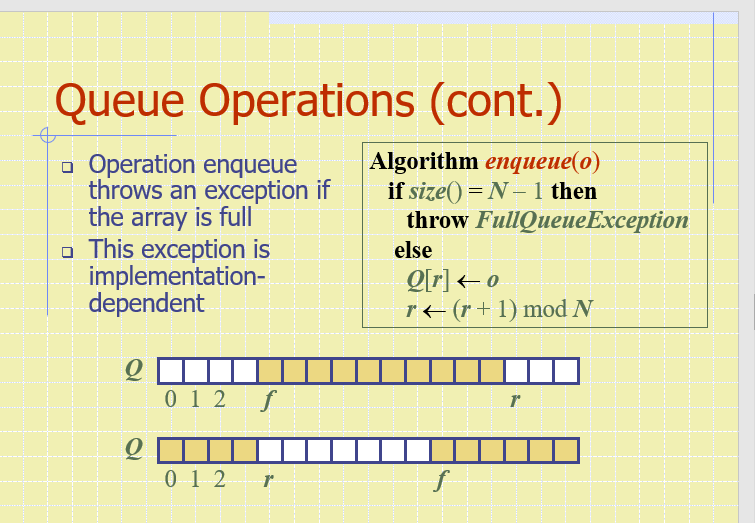


Q的操作，循环queue我们会用mod

N-f+r, modN

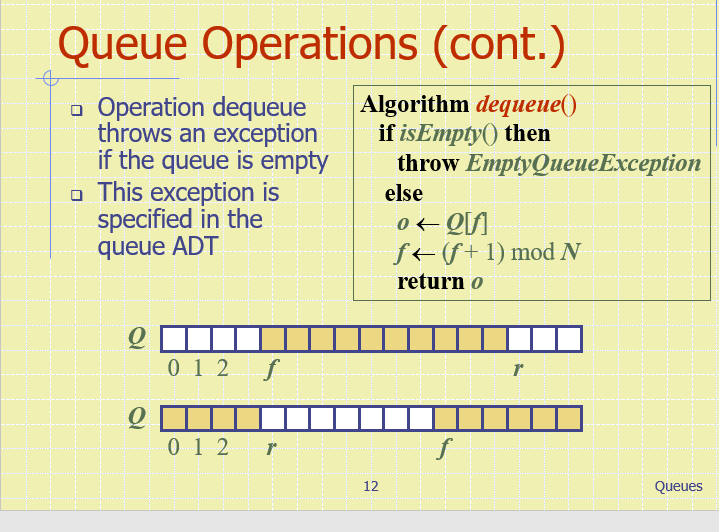
如果r在f后面还没循环，是第一张图，N-F+r modN相当于queue的实际大小（r-f）加上一个整个array的长度 modN, NmodN是0

如果r在f前面循环了，就是第二张图，N-f+r相当于queue的实际大小，直接modN没事



enq，如果SIZE已经满了，那么throw fullQException

不然把o赋给Q（r），然后r=(r+1)modN //r如果在N内，那么就往后加1，如果在N外，就会从左往右加,格子外的情况与DQ相同



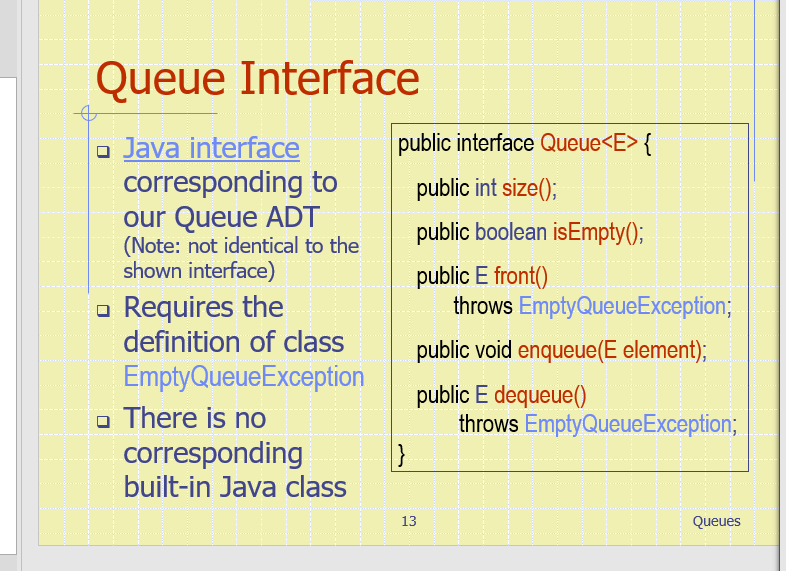
DQ,

EMPTY就throw emptyQueueException

不然把Q[f]//第一个数暂时赋予到o上， f=f+1modN//如果f+1在格子内，f右移一格，如果f再隔子外，那么重头开始到第一格//因为一次加1，格子外只能是f=N-1,+1以后MODN 等于0等于从头开始(Index(0))

然后return这个o

Queue需要在Java内重新定义



可以增长大小的基于array的Q

在enq的时候，如果array full了，我们可以用更大的arrray替换他，与stack相同



Double-ended queue

首位都支持insertion和deletion

