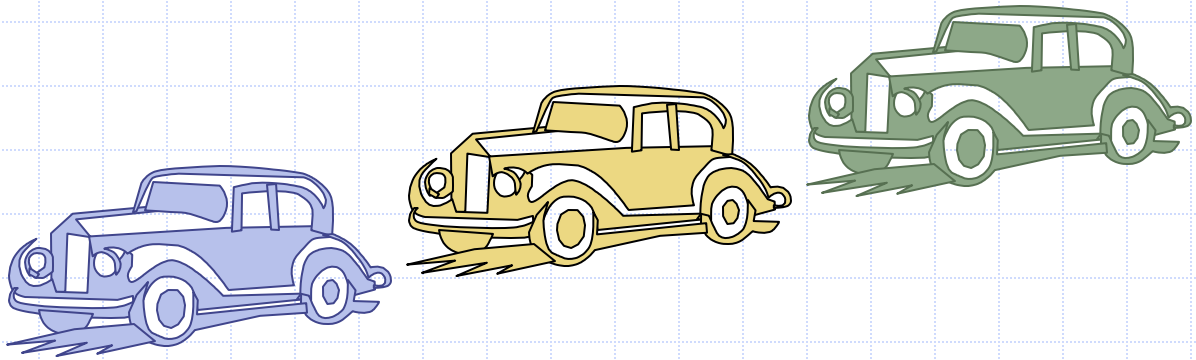
Presentation for use with the textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

Queues



Reading

M. T. Goodrich, R. Tamassia and M. H. Goldwasser, *Data Structures and Algorithms in Java*, 6th Edition, 2014.

- **Chapter 6. Stacks and Queues**

The Queue ADT

- The Queue ADT stores arbitrary objects
- Insertions and deletions follow the *first-in first-out* scheme
- Insertions are at the rear of the queue and removals are at the front of the queue
- Main queue operations:
  - enqueue(object): inserts an element at the end of the queue
  - object dequeue(): removes and returns the element at the front of the queue
- Auxiliary queue operations:
  - object first(): returns the element at the front without removing it
  - integer size(): returns the number of elements stored
  - boolean isEmpty(): indicates whether no elements are stored
- Boundary cases:
  - Attempting the execution of dequeue or first on an empty queue returns null

Example

Operation	*Output*	*Q*
enqueue(5)	–	(5)
enqueue(3)	–	(5, 3)
dequeue()	*5*	(3)
enqueue(7)	–	(3, 7)
dequeue()	*3*	(7)
first()	*7*	(7)
dequeue()	*7*	()
dequeue()	*null*	()
isEmpty()	*true*	()
enqueue(9)	–	(9)
enqueue(7)	–	(9, 7)
size()	*2*	(9, 7)
enqueue(3)	–	(9, 7, 3)
enqueue(5)	–	(9, 7, 3, 5)
dequeue()	*9*	(7, 3, 5)

队列的特性

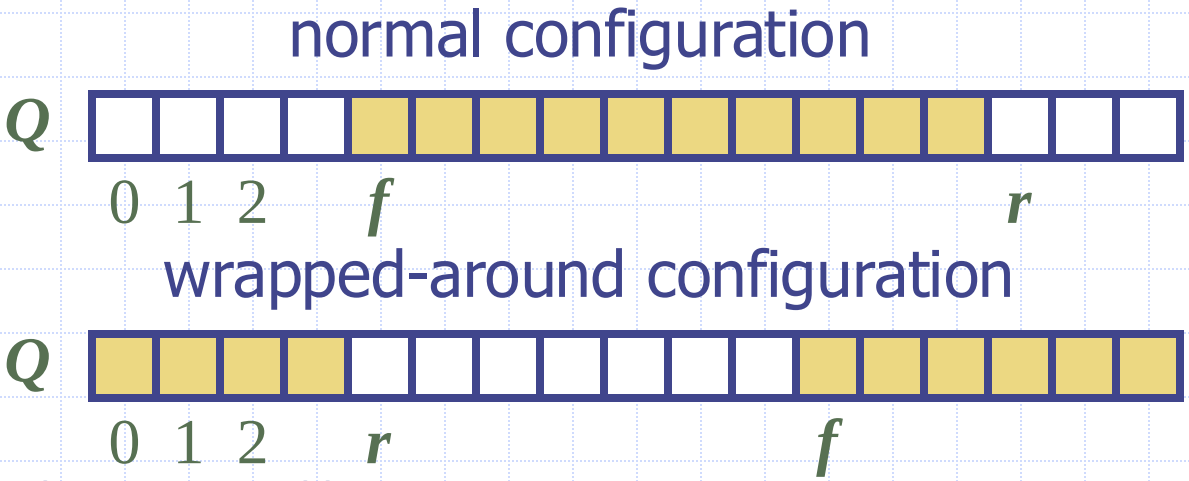
1. **先进先出 (FIFO, First-In First-Out)**
   - enqueue(x)：元素插入队列的**尾部**。
   - dequeue()：移除并返回**队列前端** (最早进入的元素)。
   - first()：返回**队列前端**的元素但不移除。
2. **空队列的行为**
   - 当 dequeue() 作用于**空队列**时，返回 null。
   - isEmpty() 返回 true 表示队列为空。
3. **示例行为**
   - **插入** 5、3 后，队列顺序是 (5，3)。
   - **取出 (dequeue)** 5 后，队列变为 (3)，再插入 7 变为 (3，7)。
   - **继续取出** 3 和 7，队列变为空 ()。
   - **新的元素加入**：9, 7, 3, 5 按顺序进入队列，最终队列 (9，7，3，5)。
   - **取出 (dequeue) 9**，剩下 (7，3，5)。

Applications of Queues

- Direct applications
  - Waiting lists
  - Access to shared resources (e.g., printer)
  - Multiprogramming
- Indirect applications
  - Auxiliary data structure for algorithms
  - Component of other data structures

Array-based Queue

- Use an array of size N in a circular fashion
- Two variables keep track of the front and size
  - f index of the front element f（front）：队列头部元素的索引
  - sz number of stored elements sz（size）：队列中当前存储的元素数量
- *When the queue has fewer than N elements*, array location $r = (f + sz) \bmod N$ is the first empty slot past the rear of the queue

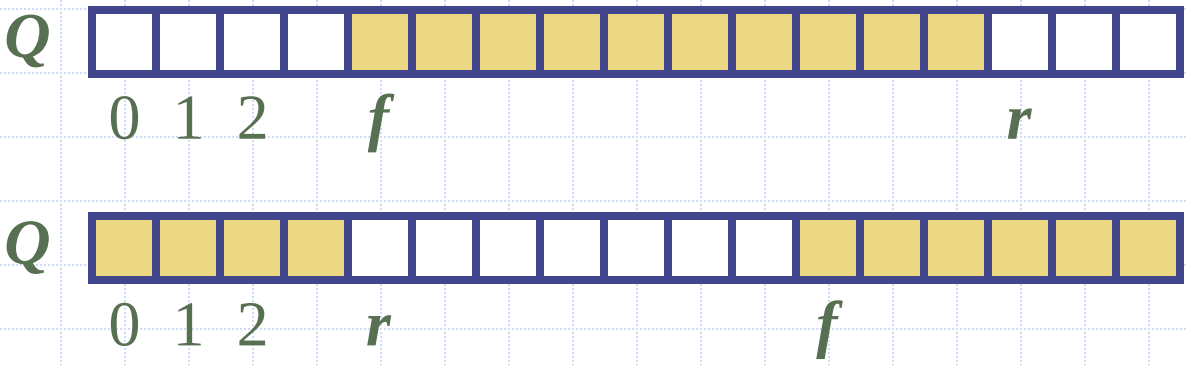


Queue Operations

- We use the modulo operator (remainder of division)

Algorithm *size*()
  return *sz*

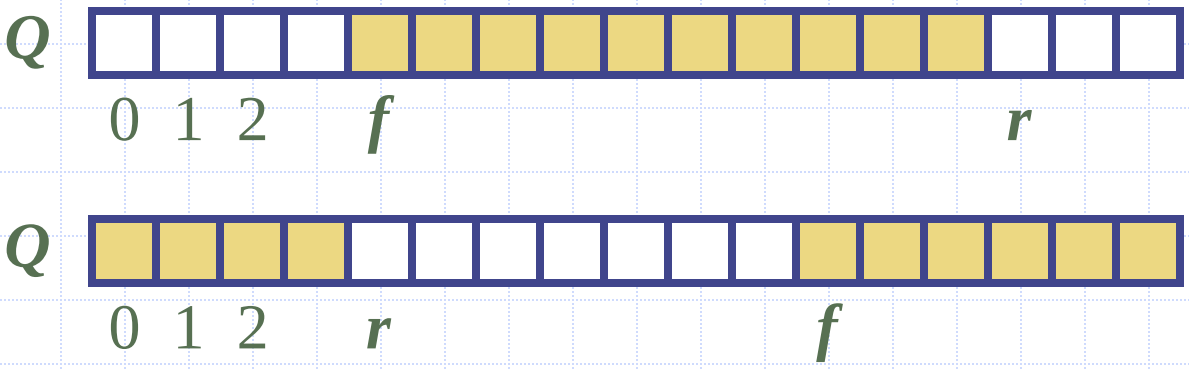
Algorithm *isEmpty*()
  return (*sz* == 0)



Queue Operations (cont.)

- Operation enqueue throws an exception if the array is full
- This exception is implementation-dependent

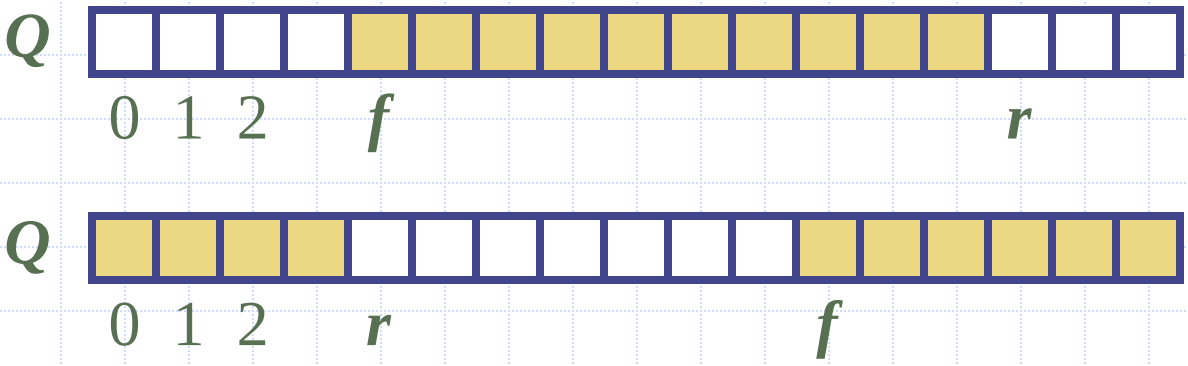
Algorithm *enqueue*(*o*)
  if *size*() = *N* **then**
    throw *IllegalStateException*
  else
    $r \leftarrow (f + sz) \bmod N$
    $Q[r] \leftarrow o$
    $sz \leftarrow (sz + 1)$



Queue Operations (cont.)

- Note that operation dequeue returns null if the queue is empty

Algorithm ***dequeue***()
  if *isEmpty*() **then**
    return *null*
  else
    $o \leftarrow Q[f]$
    $f \leftarrow (f + 1) \bmod N$
    $sz \leftarrow (sz - 1)$
    return *o*



Queue Interface in Java

- Java interface corresponding to our Queue ADT
- Assumes that first() and dequeue() return null if queue is empty

```
public interface Queue<E> {
   int size();                 返回队列中元素个数
   boolean isEmpty();          判断队列是否为空
   E first();                  返回队首元素（不移除）
   void enqueue(E e);          入队（插入元素）
   E dequeue();                出队（移除并返回队首元素）
}
```

Comparison to java.util.Queue

- Our Queue methods and corresponding methods of java.util.Queue:

Our Queue ADT	Interface java.util.Queue	
	throws exceptions	returns special value
enqueue(e)	add(e)	offer(e)
dequeue( )	remove( )	poll( )
first( )	element( )	peek( )
size( )	size( )	
isEmpty( )	isEmpty( )	

自定义 Queue ADT vs. java.util.Queue

我们的队列 ADT	java.util.Queue 接口（抛出异常）	java.util.Queue 接口（返回特殊值）
enqueue(e)	add(e)(抛出异常)	offer(e)(返回 false)
dequeue()	remove()(抛出异常)	poll()(返回 null)
first()	element()(抛出异常)	peek()(返回 null)
size()	size()(相同)	size()(相同)
isEmpty()	isEmpty()(相同)	isEmpty()(相同)

Application: Round Robin Schedulers

- We can implement a round robin scheduler using a queue Q by repeatedly performing the following steps:
  1. e = Q.dequeue()
  2. Service element e
  3. Q.enqueue(e)