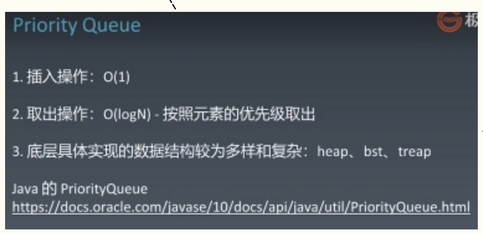
Vector vs
ArrayList
前者是线程安全的

Stack & Queue 关键点

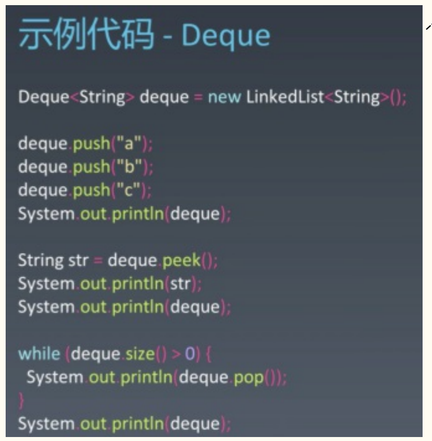
- Stack: 先入后出; 添加、删除皆为 O(1)
- Queue: 先入先出; 添加、删除皆为 O(1)
- Stack & Queue & Deque 查询皆为 O(n),
  因为无序 所以查找需要遍历
- Deque: Double-End Queue
  队列前后都可以插入删除

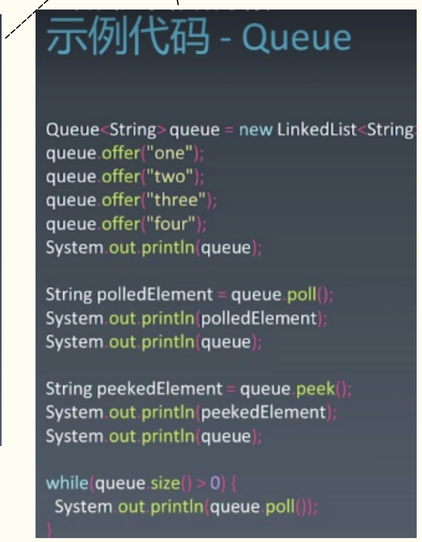
内部保持了数据的有序性

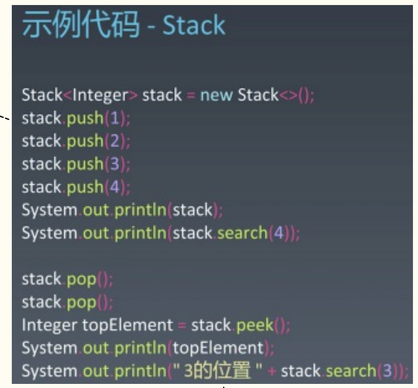


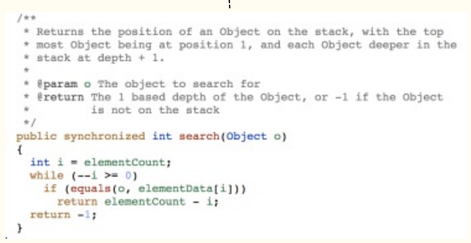
Stack & Queue 工程实现

使用数组模拟，
在数组上加一些 **API**









```
/**
 * Returns the position of an Object on the stack, with the top
 * most Object being at position 1, and each Object deeper in the
 * stack at depth + 1.
 *
 * @param o The object to search for
 * @return The 1 based depth of the Object, or -1 if the Object
 *         is not on the stack
 */
public synchronized int search(Object o)
{
  int i = elementCount;
  while (--i >= 0)
    if (equals(o, elementData[i]))
      return elementCount - i;
  return -1;
}
```

Stack

最近相关性 —> 用 栈 来解决
现实情况用栈这种逻辑来解决，就能用栈来解决

工程里面有很多 从外向内、从内向外的扩散

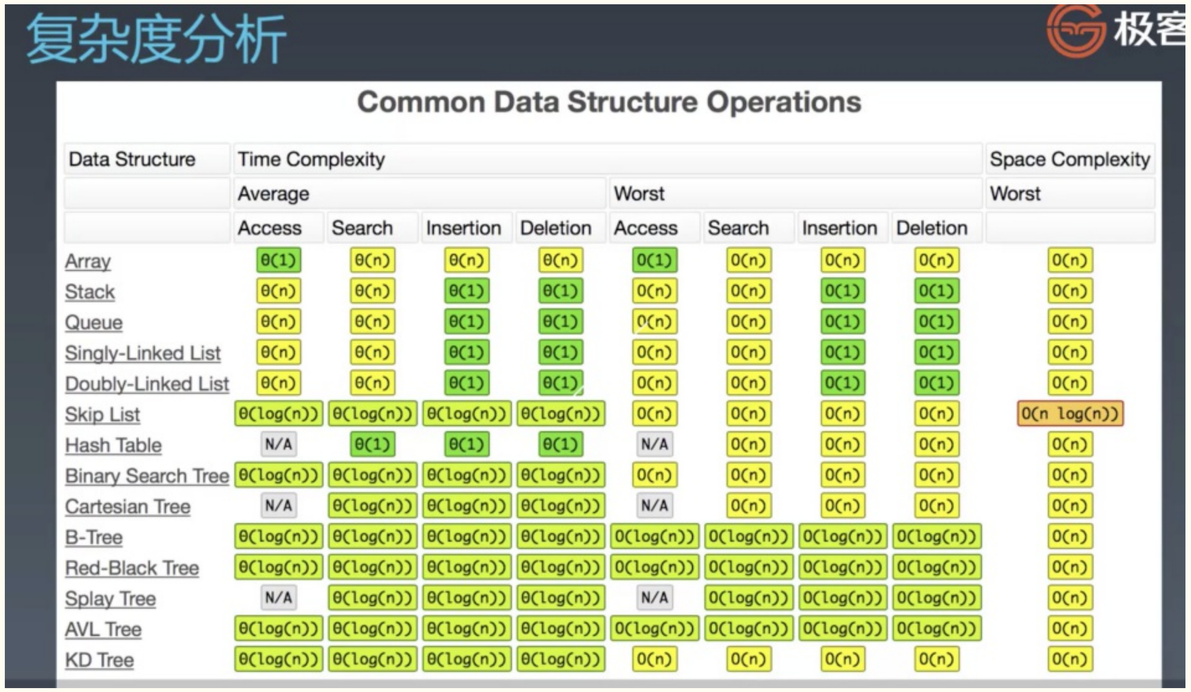
Queue

先来后到，公平性

只用队列来做栈 或 只用栈来做队列
那么你需要两个队列 或者 两个栈

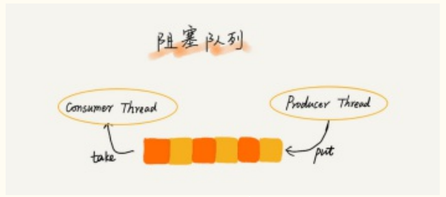
栈实现队列, 需要 A B 两个栈，出栈时 将 A 的元素全部转移至 B 栈，然后弹出 B 栈即可

复杂度分析



Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)



阻塞队列

e.g 生产者 - 消费者模型

并发队列

线程安全的队列

Queue 实现

对于大部分资源有限的场景，当没有空闲资源时，可以通过使用队列来实现请求排队

- 线程池已满
  - 拒绝请求
  - 阻塞并将请求排队

- 数组实现: 顺序队列
  - 有界队列，响应灵敏，但是要合理利用资源设置好队列大小
- 链表实现: 链式队列
  - 无界队列，但是可能会导致过多的请求排队等待，响应时间长

队列需要两个指针
head 和 tail

- 解决数据搬移 -> 循环队列
  - 入队时检查是否有空闲空间, 有则做数据搬移，再入队 O(n+1)
  - 确定好队空和队满的判定条件
    - 队满判断 tail ==
      队空判断 head == tail
    - 队空 tail == head
      队满 (tail + 1) % n = head