最后，迭代器的 remove 方法如第 23 行至第 32 行所示。该方法主要是错误检测(这就是为什么我们避免 ArrayListIterator 中的错误检测的原因)。在第 30 行上的具体的 remove 模仿 ArrayListIterator 中的逻辑。不过在这里 current 是保持不变的，因为 current 正在观察的节点不受前面节点被删除的影响(在 ArrayListIterator 中，项被移动，要求更新 current)。

3.6 栈 ADT

3.6.1 栈模型

栈(stack)是限制插入和删除只能在一个位置上进行的表，该位置是表的末端，叫做栈的顶(top)。对栈的基本操作有 push(进栈)和 pop(出栈)，前者相当于插入，后者则是删除最后插入的元素。最后插入的元素可以通过使用 top 例程在执行 pop 之前进行考查。对空栈进行的 pop 或 top 一般被认为是栈 ADT 中的一个错误。另一方面，当运行 push 时空间用尽是一个实现限制，但不是 ADT 错误。

栈有时又叫做 LIFO(后进先出)表。在图 3-33 中描述的模型只象征着 push 是输入操作而 pop 和 top 是输出操作。普通的清空栈的操作和判断是否空栈的测试都是栈的操作指令系统的一部分，但是，我们对栈所能够做的，基本上也就是 push 和 pop 操作。

图 3-34 表示在进行若干操作后的一个抽象的栈。一般的模型是，存在某个元素位于栈顶，而该元素是唯一的可见元素。

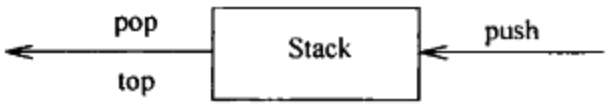

图 3-33 栈模型：通过 push 向栈输入，通过 pop 和 top 从栈中输出

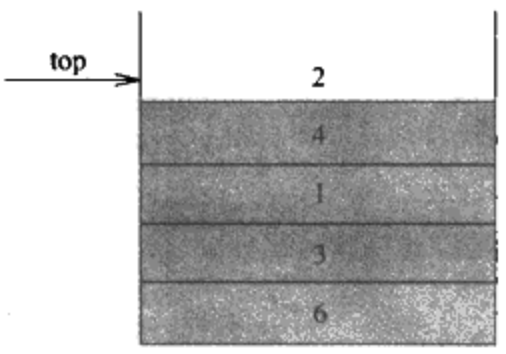

图 3-34 栈模型：只有栈顶元素是可访问的

3.6.2 栈的实现

由于栈是一个表，因此任何实现表的方法都能实现栈。显然，ArrayList 和 LinkedList 都支持栈操作；99%的时间它们都是最合理的选择。偶尔设计一种特殊目的的实现可能会更快(例如，如果被放到栈上的项属于基本类型)。因为栈操作是常数时间操作，所以，除非在非常独特的环境下，这是不可能产生任何明显的改进的。对于这些特殊的时机，我们将给出两个流行的实现方法，一种方法使用链式结构，而另一种方法则使用数组，二者均简化了在 ArrayList 和 LinkedList 中的逻辑，因此我们不提供代码。

栈的链表实现

栈的第一种实现方法是使用单链表。通过在表的顶端插入来实现 push，通过删除表顶端元素实现 pop。top 操作只是考查表顶端元素并返回它的值。有时 pop 操作和 top 操作合二为一。

栈的数组实现

另一种实现方法避免了链而且可能是更流行的解决方案。由于模仿 ArrayList 的 add 操作，因此相应的实现方法非常简单。与每个栈相关联的操作是 theArray 和 topOfStack，对于空栈它是 -1(这就是空栈初始化的做法)。为将某个元素 x 推入栈中，我们使 topOfStack 增 1 然后置 theArray[topOfStack] = x。为了弹出栈元素，我们置返回值为 theArray[topOfStack]然后使