第3章　表、栈和队列

本章讨论最简单和最基本的三种数据结构。实际上，每一个有意义的程序都将显式地至少使用一种这样的数据结构，而栈则在程序中总是要被间接地用到，不管我们在程序中是否做了声明。在本章中，我们将重点

- 介绍抽象数据类型的概念。
- 阐述如何有效地执行表的操作。
- 介绍栈 ADT 及其在实现递归方面的应用。
- 介绍队列 ADT 及其在操作系统和算法设计中的应用。

在这一章，我们提供实现两个库类重要子集 ArrayList 和 LinkedList 的代码。

3.1　抽象数据类型

抽象数据类型(abstract data type，ADT)是带有一组操作的一些对象的集合。抽象数据类型是数学的抽象；在 ADT 的定义中没有地方提到关于这组操作是如何实现的任何解释。诸如表、集合、图以及与它们各自的操作一起形成的这些对象都可以被看做是抽象数据类型，这就像整数、实数、布尔数都是数据类型一样。整数、实数和布尔数各自都有与之相关的操作，而抽象数据类型也是如此。对于集合 ADT，可以有像添加(add)、删除(remove)、以及包含(cotains)这样一些操作。当然，也可以只要两种操作并(union)和查找(find)，这两种操作又在这个集合上定义了一种不同的 ADT。

Java 类也考虑到 ADT 的实现，不过适当地隐藏了实现的细节。这样，程序中需要对 ADT 实施操作的任何其他部分可以通过调用适当的方法来进行。如果由于某种原因需要改变实现的细节，那么通过仅仅改变执行这些 ADT 操作的例程应该是很容易做到的。这种改变对于程序的其余部分是完全透明的。

对于每种 ADT 并不存在什么法则来告诉我们必须要有哪些操作，这是一个设计决策。错误处理和结构调整(在适当的地方)一般也取决于程序的设计者。本章中将要讨论的这三种数据结构是 ADT 的最基本的例子。我们将会看到它们中的每一种是如何以多种方法实现的，不过，当它们被正确地实现以后，使用它们的程序却没有必要知道它们是如何实现的。

3.2　表 ADT

我们将处理形如 $A_0, A_1, A_2, \cdots, A_{N-1}$的一般的表。我们说这个表的大小是 N。我们将大小为 0 的特殊的表称为**空表**(empty list)。

对于除空表外的任何表，我们说 A_i 后继 A_{i-1}(或继 A_{i-1}之后，$i < N$)并称 A_{i-1}前驱 A_i($i > 0$)。表中的第一个元素是 A_0，而最后一个元素是 A_{N-1}。我们将不定义 A_0 的前驱元，也不定义 A_{N-1}的后继元。元素 A_i 在表中的位置为 $i+1$。为了简单起见，我们假设表中的元素是整数，但一般说来任意的复元素也是允许的(而且容易由 Java 泛型类处理)。

与这些“定义”相关的是要在表 ADT 上进行操作的集合。printList 和 makeEmpty 是常用的操作，其功能显而易见；find 返回某一项首次出现的位置；insert 和 remove 一般是从表的某个位