```
1  public class MyArrayList<AnyType> implements Iterable<AnyType>
2  {
3      private int theSize;
4      private AnyType [ ] theItems;
5          ...
6      public java.util.Iterator<AnyType> iterator( )
7        { return new ArrayListIterator( ); }
8
9      private class ArrayListIterator implements java.util.Iterator<AnyType>
10     {
11         private int current = 0;
12
13         public boolean hasNext( )
14           { return current < size( ); }
15         public AnyType next( )
16           { return theItems[ current++ ]; }
17         public void remove( )
18           { MyArrayList.this.remove( --current ); }
19     }
20 }
```

图 3-21 迭代器 4 号版本(能够使用)：迭代器是一个内部类并存储当前位置和一个连接到 MyArrayList 的隐式链

3.5 LinkedList 类的实现

本节给出可以使用的 LinkedList 泛型类的实现。和在 ArrayList 类中的情形一样，我们这里的链表类将叫做 MyLinkedList 以避免与库中的类相混。

前面提到，LinkedList 将作为双链表来实现，而且我们还需要保留到该表两端的引用。这样做可以保持每个操作花费常数时间的代价，只要操作发生在已知的位置。这个已知的位置可以是端点，也可以是由迭代器指定的一个位置(不过，我们不实现 ListIterator，因此有些代码留给读者去完成)。

在考虑设计方面，我们将需要提供三个类：

1. MyLinkedList 类本身，它包含到两端的链、表的大小以及一些方法。

2. Node 类，它可能是一个私有的嵌套类。一个节点包含数据以及到前一个节点的链和到下一个节点的链，还有一些适当的构造方法。

3. LinkedListIterator 类，该类抽象了位置的概念，是一个私有类，并实现接口 Iterator。它提供了方法 next、hasNext 和 remove 的实现。

由于这些迭代器类存储“当前节点”的引用，并且终端标记是一个合理的位置，因此它对于在表的终端创建一个额外的节点来表示终端标记是有意义的。更进一步，我们还能够在表的前端创建一个额外的节点，逻辑上代表开始的标记。这些额外的节点有时候就叫做**标记节点**(sentinel node)；特别地，在前端的节点有时候也叫做**头节点**(header node)，而在末端的节点有时候也叫作**尾节点**(tail node)。

使用这些额外节点的优点在于，通过排除许多特殊情形极大地简化了编码。例如，如果我们不是用头节点，那么删除第 1 个节点就变成了一种特殊的情况，因为在删除期间我们必须重新调整链表的到第 1 个节点的链，还因为删除算法一般还要访问被删除节点前面的那个节点(而若无