3.4.2 迭代器、Java 嵌套类和内部类

ArrayListIterator 使用一个复杂 Java 结构，叫做**内部类**(inner class)。显然该类在 MyArrayList 类内部被声明，这是被许多语言支持的特性。然而，Java 中的内部类具有更微妙的性质。

为了了解内部类是如何工作的，图 3-17 描绘了迭代器的思路(不过，代码有缺欠)，使 ArrayListIterator 成为一个顶级类。我们只着重讨论 MyArrayList 的数据域、MyArrayList 中的 iterator 方法以及 ArrayListIterator 类(而不是它的 remove 方法)。

```
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<AnyType>( ); }
8   }
9   class ArrayListIterator<AnyType> 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  }
```

图 3-17　迭代器 1 号版本(但不能使用)：迭代器是一个顶级类并存储当前位置。它不能使用是因为 theItems 和 size()不是 ArrayListIterator 类的一部分

在图 3-17 中，ArrayListIterator 是泛型类，它存储当前位置，程序在 next 方法中试图使用当前位置作为下标访问数组元素然后将当前位置向后推进。注意，如果 arr 是一个数组，则 arr[idx++]对数组使用 idx，然后向后推进 idx。操作 ++ 在此处存在问题。我们这里使用的形式叫做**后缀 ++ 操作**(postfix ++ operator)，此时的 ++ 是在 idx 之后进行的。但在**前缀 ++ 操作**(prefix ++ operator)中，arr[++ idx]先推进 idx 然后再使用新的 idx 作为数组元素的下标。图 3-17 中的问题在于，theItems[current++]是非法的，因为 theItems 不是 ArrayListIterator 的一部分；它是 MyArrayList 的一部分。因此程序根本没有意义。

最简单的解决方案见图 3-18，不过它也有缺点，但是以更微小的方式呈现。在图 3-18 中，我们通过让迭代器存储 MyArrayList 的引用来解决在迭代器中没有数组的问题。这个引用是第二个数据域，是通过 ArrayListIterator 的一个新的单参数构造器而被初始化的。既然有一个 MyArrayList 的引用，那么就可以访问包含于 MyArrayList 中的数组域(还可得到 MyArrayList 的大小，该大小在 hasNext 中是需要的)。

图 3-18 中的问题在于，theItems 是 MyArrayList 中的私有(private)域，而由于 ArrayListIterator 是一个不同的类，因此在 next 方法中访问 theItems 是非法的。最简单的修正办法是改变 theItems 在 MyArrayList 中的可见性，从 private 改成某种稍宽松的可见性(如 public，或默认的可见性，它也被称为**包可见性**(package visibility))。不过，这违反了良好的面向对象编程的基本原则，它要求数据应尽可能地隐蔽。