```
1  public class Employee
2  {
3      public boolean equals( Object rhs )
4        { return rhs instanceof Employee && name.equals( ((Employee)rhs).name ); }
5
6      public int hashCode( )
7        { return name.hashCode( ); }
8
9      private String name;
10     private double salary;
11     private int seniority;
12
13       // Additional fields and methods
14 }
```

图 5-8　可以放在一个散列表中的 Employee 类的例子

```
1       /**
2        * Construct the hash table.
3        */
4       public SeparateChainingHashTable( )
5       {
6           this( DEFAULT_TABLE_SIZE );
7       }
8
9       /**
10       * Construct the hash table.
11       * @param size approximate table size.
12       */
13      public SeparateChainingHashTable( int size )
14      {
15          theLists = new LinkedList[ nextPrime( size ) ];
16          for( int i = 0; i < theLists.length; i++ )
17              theLists[ i ] = new LinkedList<AnyType>( );
18      }
19
20      /**
21       * Make the hash table logically empty.
22       */
23      public void makeEmpty( )
24      {
25          for( int i = 0; i < theLists.length; i++ )
26              theLists[ i ].clear( );
27          theSize = 0;
28      }
```

图 5-9　分离链接散列表的构造方法和 makeEmpty 方法

在插入例程中，如果被插入的项已经存在，那么我们不执行任何操作；否则，我们将其放入链表中。该元素可以被放到链表中的任何位置；在我们的情形下使用 add 方法是最方便的。

除链表外，任何方案都可以解决冲突现象；一棵二叉查找树或甚至另一个散列表都将胜任这个工作，但是，我们期望如果散列表是大的并且散列函数是好的，那么所有的链表都应该是短