建时所设置的原值，并提供一种方法以重新得到该值。包装类也包含不少的静态实用方法。

例如，图 1-7 说明如何能够使用 MemoryCell 来存储整数。

```
1   public class WrapperDemo
2   {
3       public static void main( String [ ] args )
4       {
5           MemoryCell m = new MemoryCell( );
6
7           m.write( new Integer( 37 ) );
8           Integer wrapperVal = (Integer) m.read( );
9           int val = wrapperVal.intValue( );
10          System.out.println( "Contents are: " + val );
11      }
12  }
```

图 1-7 Integer 包装类的一种演示

1.4.3 使用接口类型表示泛型

只有在使用 Object 类中已有的那些方法能够表示所执行的操作的时候，才能使用 Object 作为泛型类型来工作。

例如，考虑在由一些项组成的数组中找出最大项的问题。基本的代码是类型无关的，但是它的确需要一种能力来比较任意两个对象，并确定哪个是大的，哪个是小的。因此，我们不能直接找出 Object 的数组中的最大元素——我们需要更多的信息。最简单的想法就是找出 Comparable 的数组中的最大元。要确定顺序，可以使用 compareTo 方法，我们知道，它对所有的 Comparable 都必然是现成可用的。图 1-8 中的代码做的就是这项工作，它提供一种 main 方法，该方法能够找出 String 或 Shape 数组中的最大元。

现在，提出几个忠告很重要。首先，只有实现 Comparable 接口的那些对象才能够作为 Comparable 数组的元素被传递。仅有 compareTo 方法但并未宣称实现 Comparable 接口的对象不是 Comparable 的，它不具有必需的 IS-A 关系。因为我们也许会比较两个 Shape 的面积，因此假设 Shape 实现 Comparable 接口。这个测试程序还告诉我们，Circle、Square 和 Rectangle 都是 Shape 的子类。

第二，如果 Comparable 数组有两个不相容的对象(例如，一个 String 和一个 Shape)，那么 CompareTo 方法将抛出异常 ClassCastException。这是我们期望的性质。

第三，如前所述，基本类型不能作为 Comparable 传递，但是包装类则可以，因为它们实现了 Comparable 接口。

第四，接口究竟是不是标准的库接口倒不是必需的。

最后，这个方案不是总能够行得通，因为有时宣称一个类实现所需的接口是不可能的。例如，一个类可能是库中的类，而接口却是用户定义的接口。如果一个类是 final 类，那么我们就不可能扩展它以创建一个新的类。1.6 节对这个问题提出了另一个解决方案，即 **function object**。这种**函数对象**(function object)也使用一些接口，它或许是我们在 Java 库中所遇到的核心论题之一。

1.4.4 数组类型的兼容性

语言设计中的困难之一是如何处理集合类型的继承问题。设 Employee *IS-A* Person。那么，这是不是也意味着数组 Employee[ ] *IS-A* Person[ ]呢？换句话说，如果一个例程接受 Person[ ]作为参数，那么我们能不能把 Employee[ ]作为参数来传递呢？