teger 对象的结果赋值给一个 int 型的量，则编译也将产生一个错误信息。图 1-7 中的代码准确地反映出基本类型和引用类型之间的区别，但还没有清楚地表示出程序员把那些 int 存入集合(collection)的意图。

Java 5 矫正了这种情形。如果一个 int 型量被传递到需要一个 Integer 对象的地方，那么，编译器将在幕后插入一个对 Integer 构造方法的调用。这就叫做自动装箱。而如果一个 Integer 对象被放到需要 int 型量的地方，则编译器将在幕后插入一个对 intValue 方法的调用，这就叫做自动拆箱。对于其他 7 对基本类型/包装类型，同样会发生类似的情形。图 1-11 描述了自动装箱和自动拆箱的使用。注意，在 GenericMemoryCell 中引用的那些实体仍然是 Integer 对象；在 GenericMemoryCell 的实例化中，int 不能够代替 Integer。

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

图 1-11　自动装箱和拆箱

1.5.3　带有限制的通配符

图 1-12 显示一个 static 方法，该方法计算一个 Shape 数组的总面积(假设 Shape 是含有 area 方法的类；而 Circle 和 Square 则是继承 Shape 的类)。假设我们想要重写这个计算总面积的方法，使得该方法能够使用 Collection<Shape>这样的参数。Collection 将在第 3 章描述。当前，唯一重要的是它能够存储一些项，而且这些项可以用一个增强的 for 循环来处理。由于是增强的 for 循环，因此代码是相同的，最后的结果如图 1-13 所示。如果传递一个 Collection<Shape>，那么，程序会正常运行。可是，要是传递一个 Collection<Square>会发生什么情况呢？答案依赖于是否 Collection<Square> *IS-A* Collection<Shape>。回顾 1.4.4 节可知，用技术术语来说即是否我们拥有协变性。

```
1       public static double totalArea( Shape [ ] arr )
2       {
3           double total = 0;
4
5           for( Shape s : arr )
6               if( s != null )
7                   total += s.area( );
8
9           return total;
10      }
```

图 1-12　Shape[ ]的 totalArea 方法

我们在 1.4.4 节提到，Java 中的数组是协变的。于是，Square[ ] *IS-A* Shape[ ]。一方面，这种一致性意味着，如果数组是协变的，那么集合也将是协变的。另一方面，我们在 1.4.4 节看到，数组的协变性导致代码得以编译，但此后会产生一个运行时异常(一个 ArrayStoreException)。因为