1.4.1 使用 Object 表示泛型

Java 中的基本思想就是可以通过使用像 Object 这样适当的超类来实现泛型类。在图 1-5 中所示的 MemoryCell 类就是这样一个例子。

```
1   // MemoryCell class
2   //  Object read( )         -->  Returns the stored value
3   //  void write( Object x ) -->  x is stored
4
5   public class MemoryCell
6   {
7           // Public methods
8       public Object read( )         { return storedValue; }
9       public void write( Object x ) { storedValue = x; }
10
11          // Private internal data representation
12      private Object storedValue;
13  }
```

图 1-5　泛型 MemoryCell 类(pre-Java 5)

当我们使用这种策略时，有两个细节必须要考虑。第一个细节在图 1-6 中阐释，它描述一个 main 方法，该方法把串"37"写到 MemoryCell 对象中，然后又从 MemoryCell 对象读出。为了访问这种对象的一个特定方法，必须要强制转换成正确的类型。(当然，在这个例子中，可以不必进行强制转换，因为在程序的第 9 行可以调用 toString()方法，这种调用对任意对象都是能够做到的)。

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

图 1-6　使用泛型 MemoryCell 类(pre-Java 5)

第二个重要的细节是不能使用基本类型。只有引用类型能够与 Object 相容。这个问题的标准工作马上就要讨论。

1.4.2 基本类型的包装

当我们实现算法的时候，常常遇到语言定型问题：我们已有一种类型的对象，可是语言的语法却需要一种不同类型的对象。

这种技巧阐释了**包装类**(wrapper class)的基本主题。一种典型的用法是存储一个基本的类型，并添加一些这种基本类型不支持或不能正确支持的操作。

在 Java 中我们已经看到，虽然每一个引用类型都和 Object 相容，但是，8 种基本类型却不能。于是，Java 为这 8 种基本类型中的每一种都提供了一个包装类。例如，int 类型的包装是 Integer。每一个包装对象都是**不可变的**(就是说它的状态绝不能改变)，它存储一种当该对象被构