是非法的。T由它的限界代替，这可能是Object(或甚至是抽象类)，因此对new的调用没有意义。

泛型数组对象

也不能创建一个泛型的数组。如果T是一个类型变量，则语句

```
T [ ] arr = new T[ 10];   // 右边是非法的
```

是非法的。T将由它的限界代替，这很可能是Object T，于是(由类型擦除产生的)对T[]的类型转换将无法进行，因为Object[] *IS-NOT-A* T[]。由于我们不能创建泛型对象的数组，因此一般说来我们必须创建一个擦除类型的数组，然后使用类型转换。这种类型转换将产生一个关于未检验的类型转换的编译警告。

参数化类型的数组

参数化类型的数组的实例化是非法的。考虑下列代码：

```
1   GenericMemoryCell<String> [ ] arr1 = new GenericMemoryCell<String>[ 10 ];
2   GenericMemoryCell<Double> cell = new GenericMemoryCell<Double>( ); cell.write( 4.5 );
3   Object [ ] arr2 = arr1;
4   arr2[ 0 ] = cell;
5   String s = arr1[ 0 ].read( );
```

正常情况下，我们认为第4行的赋值会生成一个ArrayStoreException，因为赋值的类型有错误。可是，在类型擦除之后，数组的类型为GenericMemoryCell[]，而加到数组中的对象也是GenericMemoryCell，因此不存在ArrayStoreException异常。于是，该段代码没有类型转换，它最终将在第5行产生一个ClassCastException异常，这正是泛型应该避免的情况。

1.6　函数对象

在1.5节我们指出如何编写泛型算法。例如，图1-16中的泛型方法可以用于找出一个数组中的最大项。

然而，这种泛型方法有一个重要的局限：它只对实现Comparable接口的对象有效，因为它使用compareTo作为所有比较决策的基础。在许多情形下，这种处理方式是不可行的。例如，尽管假设Rectangle类实现Comparable接口有些过分，但即使实现了该接口，它所具有的compareTo方法恐怕还不是我们想要的方法。例如，给定一个2×10的矩形和一个5×5的矩形，哪个是更大的矩形呢？答案恐怕依赖于我们是使用面积还是使用长度来决定。或者，如果我们试图通过一个开口构造该矩形，那么或许较大的矩形就是具有较大最小周长的矩形。作为第二个例子，在一个字符串的数组中如果想要找出最大的串(即字典序排在最后的串)，默认的compareTo不忽略字符的大小写，则"ZEBRA"按字典序排在"alligator"之前，这可能不是我们想要的。

上述这些情形的解决方案是重写findMax，使它接受两个参数：一个是对象的数组，另一个是比较函数，该函数解释如何决定两个对象中哪个大哪个小。实际上，这些对象不再知道如何比较它们自己；这些信息从数组的对象中完全去除了。

一种将函数作为参数传递的独创方法是注意到对象既包含数据也包含方法，于是我们可以定义一个没有数据而只有一个方法的类，并传递该类的一个实例。事实上，一个函数通过将其放在一个对象内部而被传递。这样的对象通常叫做**函数对象**(funtion object)。

图1-18显示函数对象想法的最简单的实现。findMax的第二个参数是Comparator类型的对象。接口Comparator在java.util中指定并包含一个compare方法。这个接口在图1-19中指出。

实现接口Comparator<AnyType>类型的任何类都必须要有一个叫做compare的方法，该方法有两个泛型类型(AnyType)的参数并返回一个int型的量，遵守和compareTo相同的一般约定。