```
1   public static <AnyType extends Comparable<? super AnyType>>
2   AnyType findMax( AnyType [ ] arr )
3   {
4       int maxIndex = 0;
5
6       for( int i = 1; i < arr.length; i++ )
7           if( arr[ i ].compareTo( arr[ maxIndex ] ) > 0 )
8               maxIndex = i;
9
10      return arr[ maxIndex ];
11  }
```

图 1-17 在一个数组中找出最大元的泛型 static 方法。以例说明类型参数的限界

1.5.6 类型擦除

泛型在很大程度上是 Java 语言中的成分而不是虚拟机中的结构。泛型类可以由编译器通过所谓的**类型擦除**(type erasure)过程而转变成非泛型类。这样，编译器就生成一种与泛型类同名的**原始类**(raw class)，但是类型参数都被删去了。类型变量由它们的类型限界来代替，当一个具有擦除返回类型的泛型方法被调用的时候，一些特性被自动地插入。如果使用一个泛型类而不带类型参数，那么使用的是原始类。

类型擦除的一个重要推论是，所生成的代码与程序员在泛型之前所写的代码并没有太多的差异，而且事实上运行的也并不快。其显著的优点在于，程序员不必把一些类型转换放到代码中，编译器将进行重要的类型检验。

1.5.7 对于泛型的限制

对于泛型类型有许多的限制。由于类型擦除的原因，这里列出的每一个限制都是必须要遵守的。

基本类型

基本类型不能用做类型参数。因此，GenericMemoryCell＜int＞是非法的。我们必须使用包装类。

instanceof 检测

instanceof 检测和类型转换工作只对原始类型进行。在下列代码中：

```
GenericMemoryCell<Integer> cell1 = new GenericMemoryCell<Integer>( );
cell1.write( 4 );
Object cell = cell1;
GenericMemoryCell<String> cell2 = (GenericMemoryCell<String>) cell;
String s = cell2.read( );
```

这里的类型转换在运行时是成功的，因为所有的类型都是 GenericMemoryCell。但在最后一行，由于对 read 的调用企图返回一个 String 对象从而产生一个运行时错误。结果，类型转换将产生一个警告，而对应的 instanceof 检测是非法的。

static 的语境

在一个泛型类中，static 方法和 static 域均不可引用类的类型变量，因为在类型擦除后类型变量就不存在了。另外，由于实际上只存在一个原始的类，因此 static 域在该类的诸泛型实例之间是共享的。

泛型类型的实例化

不能创建一个泛型类型的实例。如果 T 是一个类型变量，则语句

```
T obj = new T( );          // 右边是非法的
```