```
1  public static <AnyType> boolean contains( AnyType [ ] arr, AnyType x )
2  {
3      for( AnyType val : arr )
4          if( x.equals( val ) )
5              return true;
6
7      return false;
8  }
```

图 1-15　泛型 static 方法搜索数组

泛型方法特别像是泛型类，因为类型参数表使用相同的语法。在泛型方法中的类型参数位于返回类型之前。

### 1.5.5　类型限界

假设我们想要编写一个 findMax 例程。考虑图 1-16 中的代码。由于编译器不能证明在第 6 行上对 compareTo 的调用是合法的，因此，程序不能正常运行；只有在 AnyType 是 Comparable 的情况下才能保证 compareTo 存在。我们可以使用**类型限界**(type bound)解决这个问题。类型限界在尖括号内指定，它指定参数类型必须具有的性质。一种自然的想法是把性质改写成

```
public static <AnyType extends Comparable> ...
```

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

图 1-16　泛型 static 方法查找一个数组中的最大元素，该方法不能正常运行

我们知道，因为 Comparable 接口如今是泛型的，所以这种做法很自然。虽然这个程序能够被编译，但是更好的做法却是

```
public static <AnyType extends Comparable<AnyType>> ...
```

然而，这个做法还是不能令人满意。为了看清这个问题，假设 Shape 实现 Comparable<Shape>，设 Square 继承 Shape。此时，我们所知道的只是 Square 实现 Comparable<Shape>。于是，Square *IS-A* Comparable<Shape>，但它 *IS-NOT-A* Comparable<Square>！

应该说，AnyType *IS-A* Comparable<T>，其中，T 是 AnyType 的父类。由于我们不需要知道准确的类型 T，因此可以使用通配符。最后的结果变成

```
public static <AnyType extends Comparable<? super AnyType>>
```

图 1-17 显示 findMax 的实现。编译器将接受类型 T 的数组，只是使得 T 实现 Comparable<S>接口，其中 T *IS-A* S。当然，限界声明看起来有些混乱。幸运的是，我们不会再看到任何比这种用语更复杂的用语了。