泛型中? super T 和? extends T 的区别

一、前言

经常发现有 List<? super T>、Set<? extends T>的声明，是什么意思呢？<? super T>表示包括 T 在内的任何 T 的父类，<? extends T>表示包括 T 在内的任何 T 的子类，下面我们详细分析一下两种通配符具体的区别。

二、extends

如果 List<? extends Number> foo3 的通配符声明，意味着以下的赋值是合法的：

```
// Number "extends" Number (in this context)

List<? extends Number> foo3 = new ArrayList<? extends Number>();

// Integer extends Number

List<? extends Number> foo3 = new ArrayList<? extends Integer>();

// Double extends Number

List<? extends Number> foo3 = new ArrayList<? extends Double>();
```

- 读取操作通过以上给定的赋值语句，你一定能从 foo3 列表中读取到的元素的类型是什么呢？你可以读取到 Number，因为以上的列表要么包含 Number 元素，要么包含 Number 的类元素。你不能保证读取到 Integer，因为 foo3 可能指向的是 List<Double>。你不能保证读取到 Double，因为 foo3 可能指向的是 List<Integer>。

- 写入操作过以上给定的赋值语句，你能把一个什么类型的元素合法地插入到 foo3 中呢？你不能插入一个 Integer 元素，因为 foo3 可能指向 List<Double>。你不能插入一个 Double 元素，因为 foo3 可能指向 List<Integer>。你不能插入一个 Number 元素，因为 foo3 可能指向 List<Integer>。你不能往 List<? extends T>中插入任何类型的对象，因为你不能保证列表实际指向的类型是什么，你并不能保证列表中实际存储什么类型的对象。唯一可以保证的是，你可以从中读取到 T 或者 T 的子类。

三、super

现在考虑一下 List<? super T>。

List<? super Integer> foo3 的通配符声明，意味着以下赋值是合法的：

```
// Integer is a "superclass" of Integer (in this context)

List<? super Integer> foo3 = new ArrayList<Integer>();

// Number is a superclass of Integer

List<? super Integer> foo3 = new ArrayList<Number>();

// Object is a superclass of Integer

List<? super Integer> foo3 = new ArrayList<Object>();
```

- 读取操作通过以上给定的赋值语句，你一定能从 foo3 列表中读取到的元素的类型是什么呢？你不能保证读取到 Integer，因为 foo3 可能指向 List<Number> 或者 List<Object>。你不能保证读取到 Number，因为 foo3 可能指向 List<Object>。唯一可以保证的是，你可以读取到 Object 或者 Object 子类的对象（你并不知道具体的子类是什么）。

- 写入操作通过以上给定的赋值语句，你能把一个什么类型的元素合法地插入到 foo3 中呢？你可以插入 Integer 对象，因为上述声明的列表都支持 Integer。你可以插入 Integer 的子类的对象，因为 Integer 的子类同时也是 Integer，原因同上。你不能插入 Double 对象，因为 foo3 可能指向 ArrayList<Integer>。你不能插入 Number 对象，因为 foo3 可能指向 ArrayList<Integer>。你不能插入 Object 对象，因为 foo3 可能指向 ArrayList<Integer>。

四、PECS

请记住 PECS 原则：生产者（Producer）使用 extends，消费者（Consumer）使用 super。

生产者使用 extends
如果你需要一个列表提供 T 类型的元素（即你想从列表中读取 T 类型的元素），你需要把这个列表声明成<? extends T>，比如 List<? extends Integer>，因此你不能往该列表中添加任何元素。

消费者使用 super
如果需要一个列表使用 T 类型的元素（即你想把 T 类型的元素加入到列表中），你需要把这个列表声明成<? super T>，比如 List<? super Integer>，因此你不能保证从中读取到的元素的类型。

即是生产者，也是消费者

如果一个列表即要生产，又要消费，你不能使用泛型通配符声明列表，比如 List<Integer>。

参考 java.util.Collections 里的 copy 方法(JDK1.7)：

```
/**
 * Copies all of the elements from one list into another.  After the
 * operation, the index of each copied element in the destination list
 * will be identical to its index in the source list.  The destination
 * list must be at least as long as the source list.  If it is longer, the
 * remaining elements in the destination list are unaffected. <p>
 *
 * This method runs in linear time.
 *
 * @param  dest The destination list.
 * @param  src The source list.
 * @throws IndexOutOfBoundsException if the destination list is too small
 *         to contain the entire source List.
 * @throws UnsupportedOperationException if the destination list's
 *         list-iterator does not support the <tt>set</tt> operation.
 */
public static <T> void copy(List<? super T> dest, List<? extends T> src) {
    int srcSize = src.size();
    if (srcSize > dest.size())
        throw new IndexOutOfBoundsException("Source does not fit in dest");

    if (srcSize < COPY_THRESHOLD ||
        (src instanceof RandomAccess && dest instanceof RandomAccess)) {
        for (int i=0; i<srcSize; i++)
            dest.set(i, src.get(i));
    } else {
        ListIterator<? super T> di=dest.listIterator();
        ListIterator<? extends T> si=src.listIterator();
        for (int i=0; i<srcSize; i++) {
            di.next();
            di.set(si.next());
        }
    }
}
```