# Comparable&Comparator

目录

[Comparable&Comparator 1](#_Toc59728130)

[概览 3](#_Toc59728131)

[Comparable——内部比较器 4](#_Toc59728132)

[编写一个SortByComparable的类 4](#_Toc59728133)

[编写一个测试类：ComparerTest 6](#_Toc59728134)

[CompareTo()和Collections.sort() 10](#_Toc59728135)

[关于Comparable的总结 14](#_Toc59728136)

[Comparator——外部比较器的介绍 15](#_Toc59728137)

[编写一个SortByComparator的类 15](#_Toc59728138)

[compare方法到底怎么用 16](#_Toc59728139)

[Collections.sort(list,Comparator())源码分析 19](#_Toc59728140)

[Comparator总结 21](#_Toc59728141)

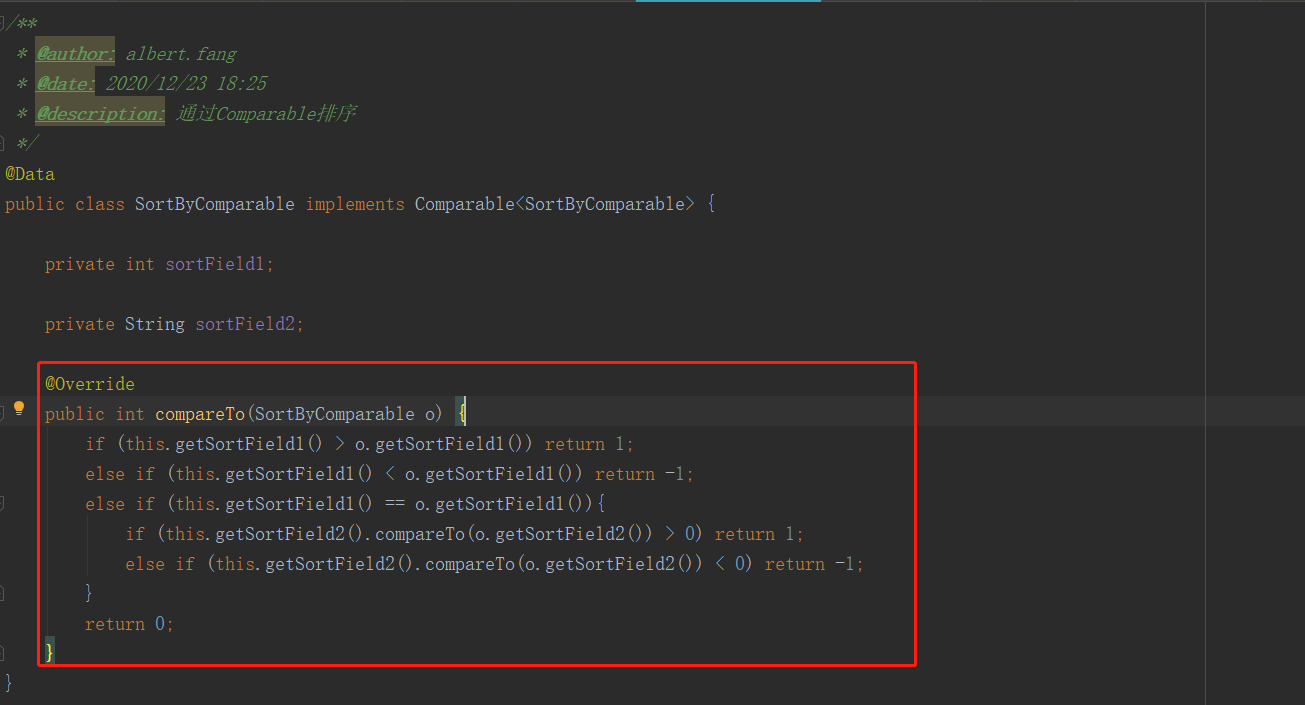
[Comparable & Comparator异同点 22](#_Toc59728142)

## 概览

comparable和comparator都是比较器。都是按照相同的规则比较两个同类型的对象的前后顺序。网上很多人都说comparable是内部比较器；comparator是外部比较器，但是为什么会有这样的区分和命名呢？

## Comparable——内部比较器

### 编写一个SortByComparable的类



这个类很简单，关键部分还是框出来的compareTo方法，这个方法的主要作用是比较当前对象（this）和传入的对象，哪个在前，哪个在后。比较器嘛，自然是比较先后顺序的。

既然是比较，那么自然就有比较的规则，所以compareTo这个方法的作用就是规定两个对象的比较规则。

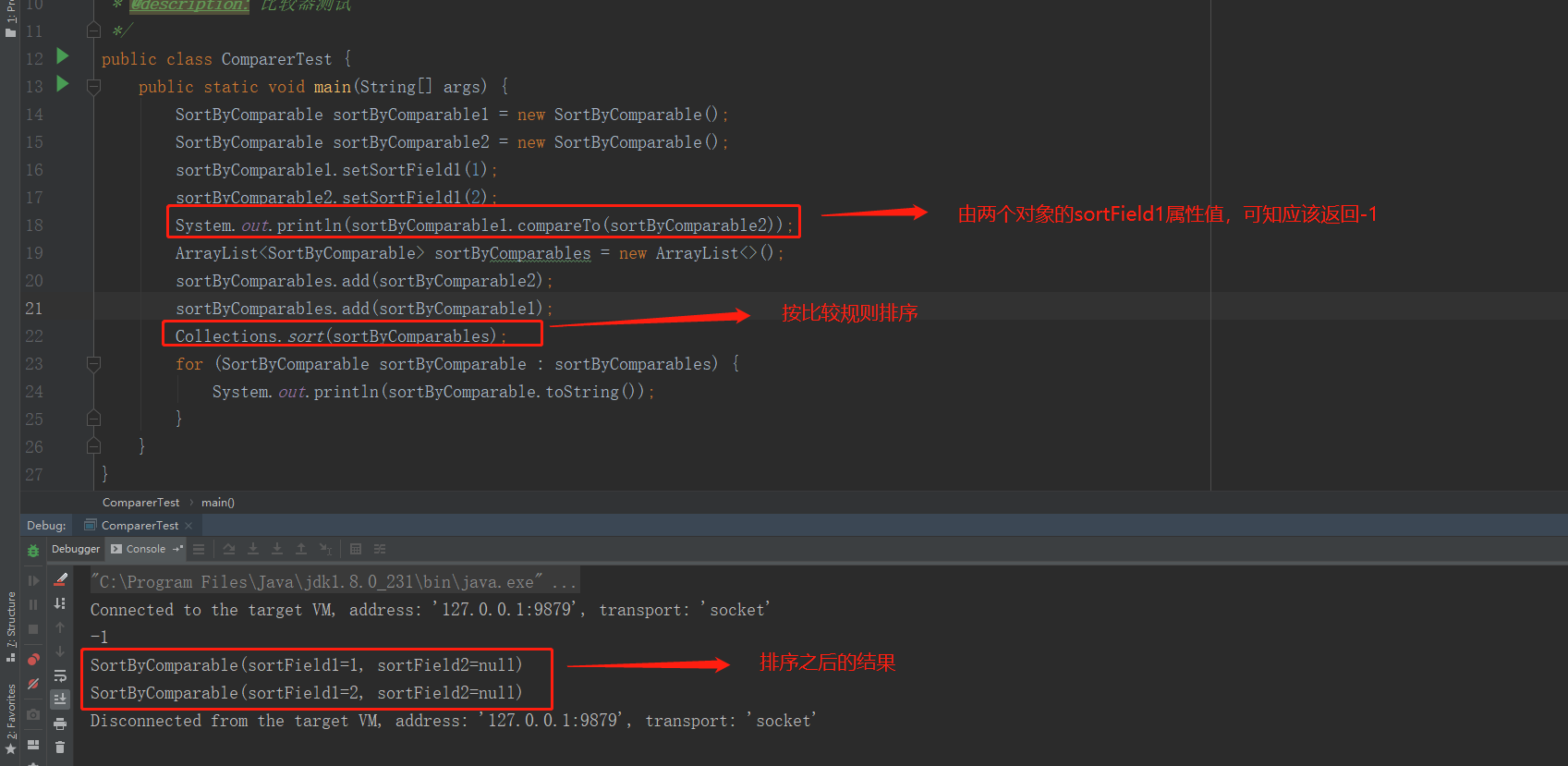
如上所示，这个compareTo方法所定下的比较规则是：

1. 如果当前对象的sortField1字段值比待比较的sortField1字段值大，并且返回大于0的值；那么当前对象排在待比较对象后面（这里网上好多人说是升序，其实也可以这么理解，小的在前大的在后）。
2. 如果当前对象的sortField1字段值比待比较对象的sortField1字段值小，并且返回小于0的值；那么当前对象排在待比较对象的前面。
3. 如果当前对象和待比较对象的字段sortField1值相等，那么就根据sortField2字段进行排序。
   1. 如果当前对象的sortField2字段值比待比较对象的sortField2字段大，并且返回大于0的值；那么当前对象排在待比较对象后面。
   2. 如果当前对象的sortField2字段比待比较对象的sortField2字段小，并且返回小于0的值；那么当前对象排在待比较对象前面。
   3. 其它情况返回0。0表示两个对象按照现有的比较规则并不能比较出哪个对象在前哪个对象在后，所以保持这两个对象的原有顺序不变。

通过上面对该截图中的compareTo方法的排序规则讲解，已经很清楚两个对象的排序规则了，接下来我们测试下是否如我们所想的那样排序。

### 编写一个测试类：ComparerTest

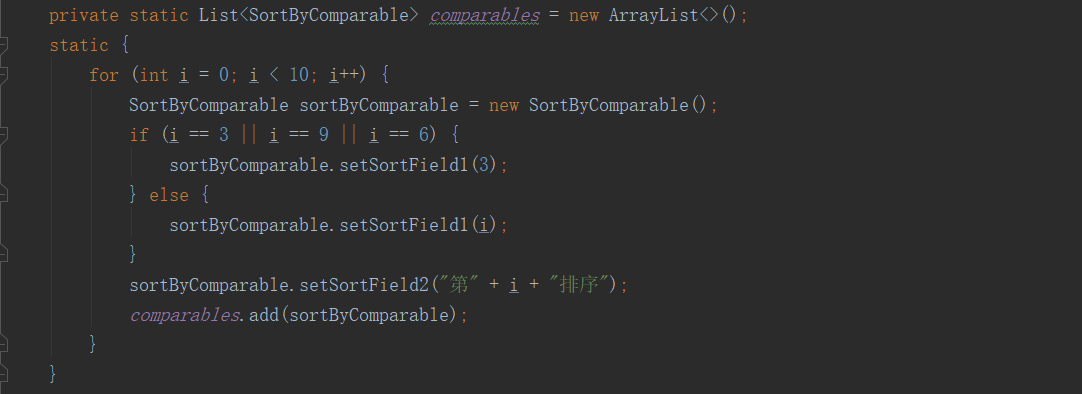
1. 根据sortField1排序。（简单版）



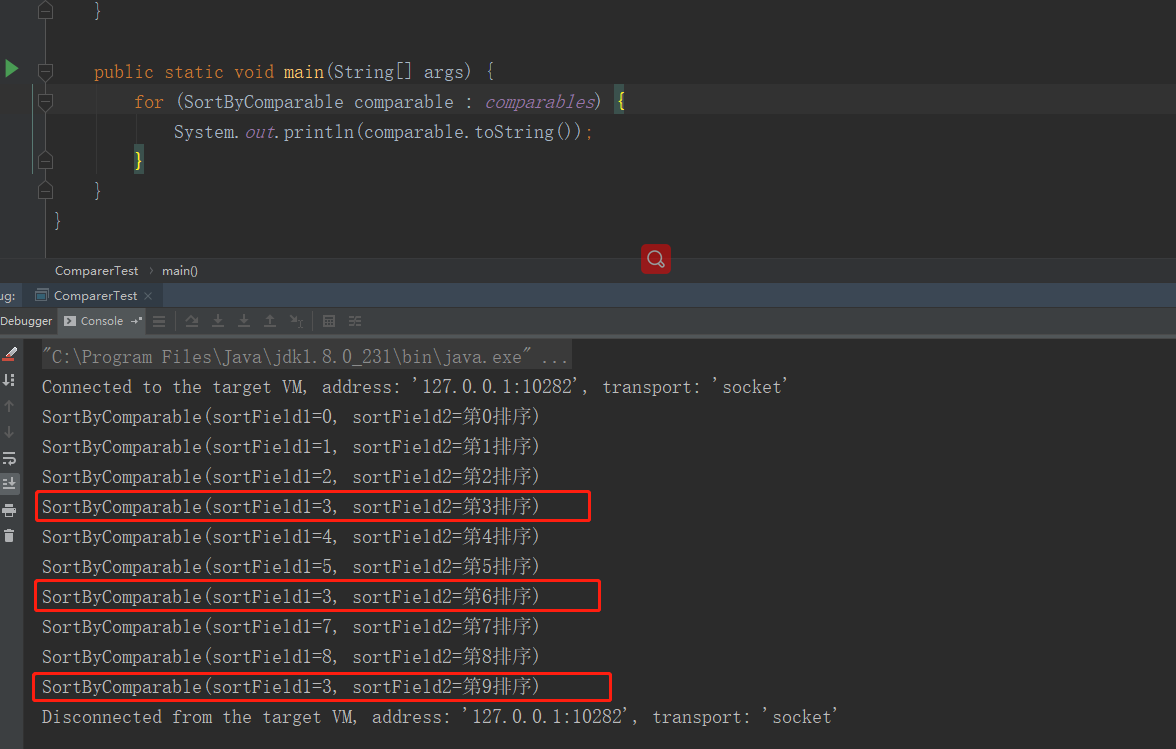
可以看见，控制台输出的结果就是我们理想的结果。

1. 创建10个SortByCompare对象，十个对象放在一个list里。（复杂版）

如图：

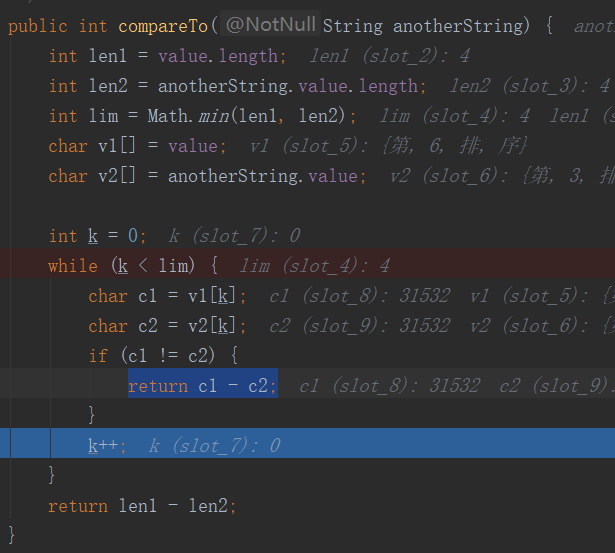


List里的10对象，有3个是sortField1字段值是一致的，我们可以打印出来看看。



这个时候，如果我们调用Collections.sort()方法来对该list排序，那么根据我们刚刚定下的排序规则，肯定不是这样的；首先sortField1字段就得是0，1，2，3，3……,然后根据sortFild2字段值进行排序。SortField1字段是数字所以升序排序容易理解，小的在前，大的在后。但是sortField2是字符串，那么升序排序该如何理解呢？

我们可以先看看String类里的compareTo的比较方法，源代码如下图：



代码很简单，我来简单的翻译下：（其实这个比较规则和我们刚刚自己写的比较规则是类似的，都是通过当前类实现comparable接口，然后重写compareTo方法，对当前对象和传入对象进行制定比较规则。）

（1）得到当前字符串对象的长度以及待比较字符串长度。

（2）取这两个长度中最小的一个。

（3）将当前字符串和待比较字符串转成char数组（String内部已经转好，有一个value属性），进行比较。

（4）迭代两个char数组中的每一个字符，将相应字符转成unicode码，比较大小。如果当前字符串所对应的字符不等于待比较的字符串字符，那么返回两个字符所对应的unicode码的差值。如果差值大于0，则当前字符串比待比较的字符串大；如果差值小于0，则当前字符串比待比较字符串小。

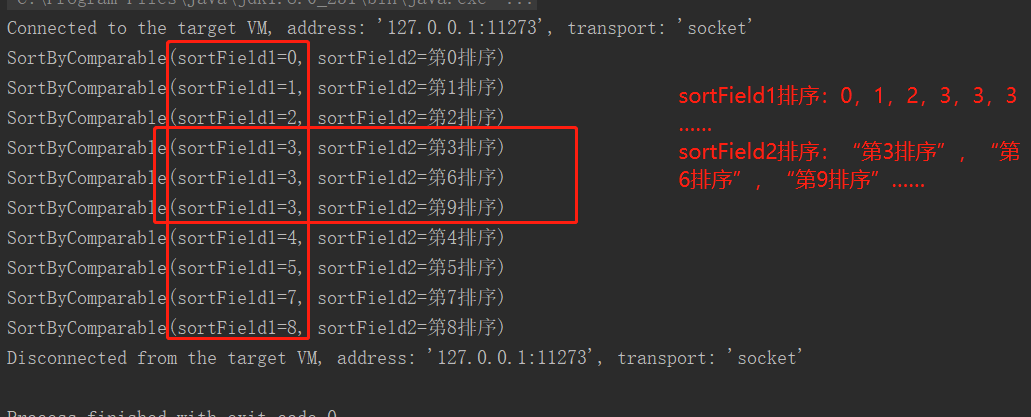
（5）如果遍历完成，都一样的话，那么返回两个字符串的长度差值，如果大于0，也表示当前字符串更大；如果小于0，表示当前字符串更小；如果等于0则表示两个字符串一模一样，不需改变。

（6）String类里的CompareTo方法定制的排序规则是升序排序，小的在前。

我们已经了解清楚字符串的比较规则了，那么对于上面list中的10个对象，排序之后的猜想是：首先按照sortField1的0，1，2，3，3，3，……，然后第二排序是按sortField2处理sortField1值为3，3，3的三个对象。

可以看见这三个对象的sortField2字段值分别为：“第3排序”，“第6排序”，“第9排序”。所以排序之后的结果就是，3-“第3排序”，3-“第6排序”，3-“第9排序”。

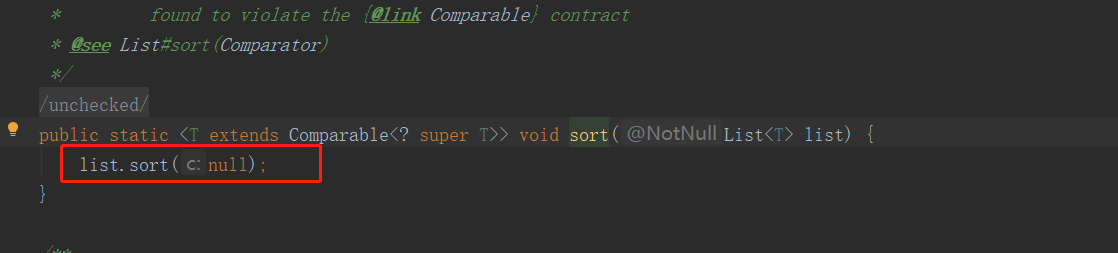
运行代码，看效果：



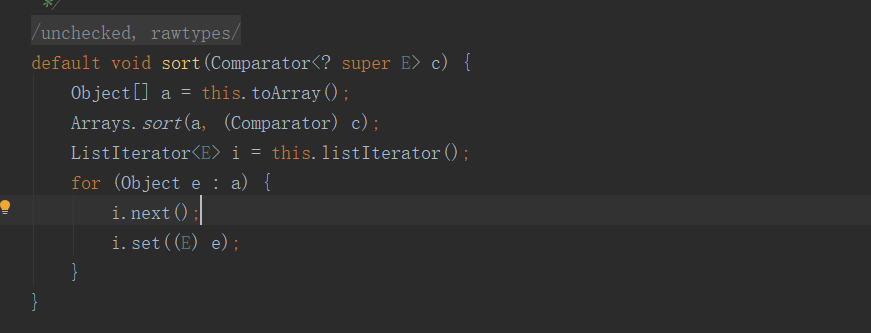
与猜测一样。

### CompareTo()和Collections.sort()

1. CompareTo只是定义了比较规则。而对list集合里的对象进行重组排序是依托于Collections.sort()方法的。
2. 查看Collections.sort()方法，可以了解到，该方法也只是调用了list的sort方法。

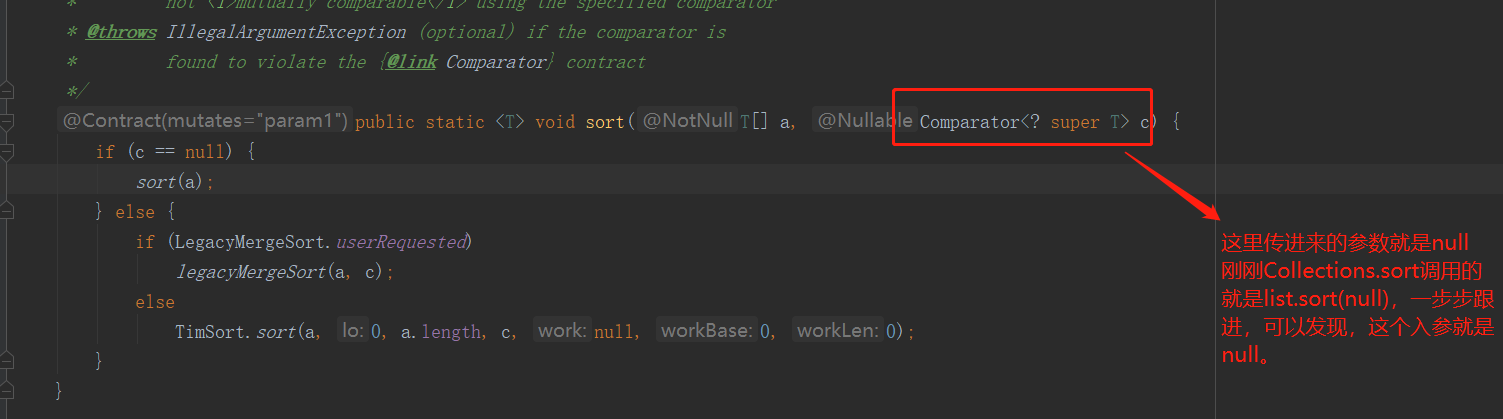


1. 继而往里，查看list的sort方法

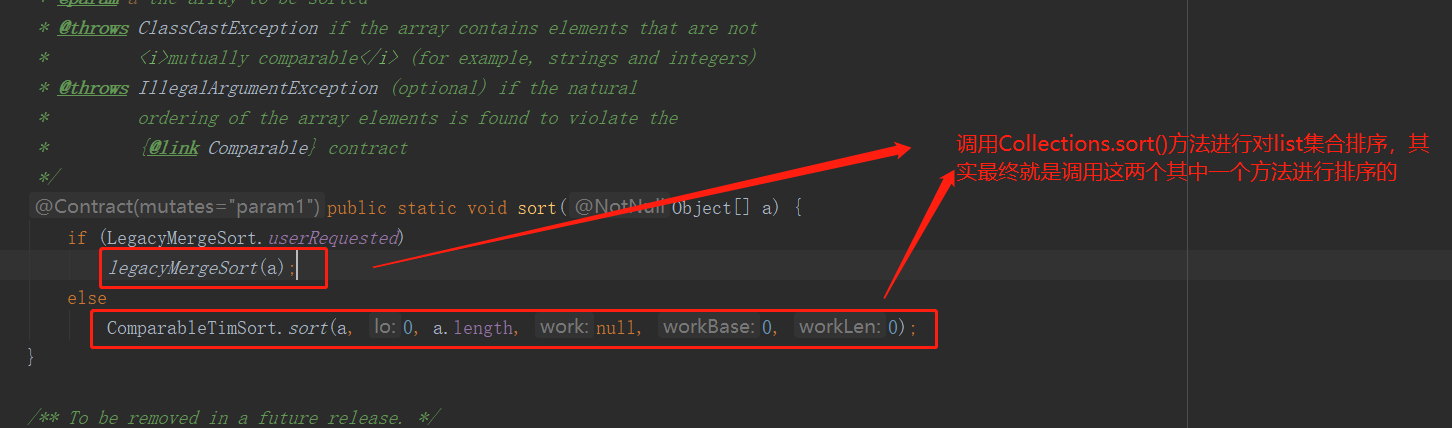


将list转成数组；然后调用的还是Arrays.sort()方法排序。

1. 继续查看Arrays.sort()方法的源码



1. 因为入参是null，所以这里直接调用Arrays类的sort方法



1. 判断条件legacyMergeSort.userRequested是什么意思

该值默认是false，所以默认走的是ComparableTimSort.sort方法。

LegacyMergeSort是list旧的排序方法。

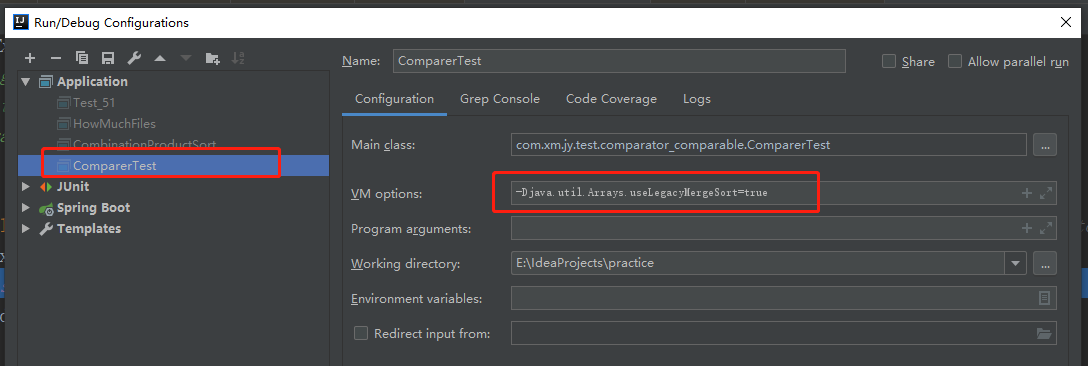
ComparableTimSort.sort是list新的排序方法。

为什么新的出来还要使用旧的呢？这是因为如果通过comparator外部比较器来进行确定比较规则的话，在jdk1.7会出现问题，所以这里需要旧的排序方式。

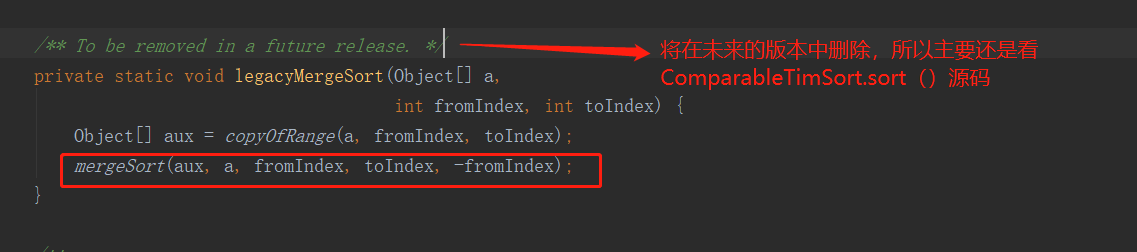
但是因为默认的legacyMergeSort.userRequested是false，那么如何将该值转成true呢？

在IDE里运行该程序的时候添加启动参数：

-Djava.util.Arrays.useLegacyMergeSort=true



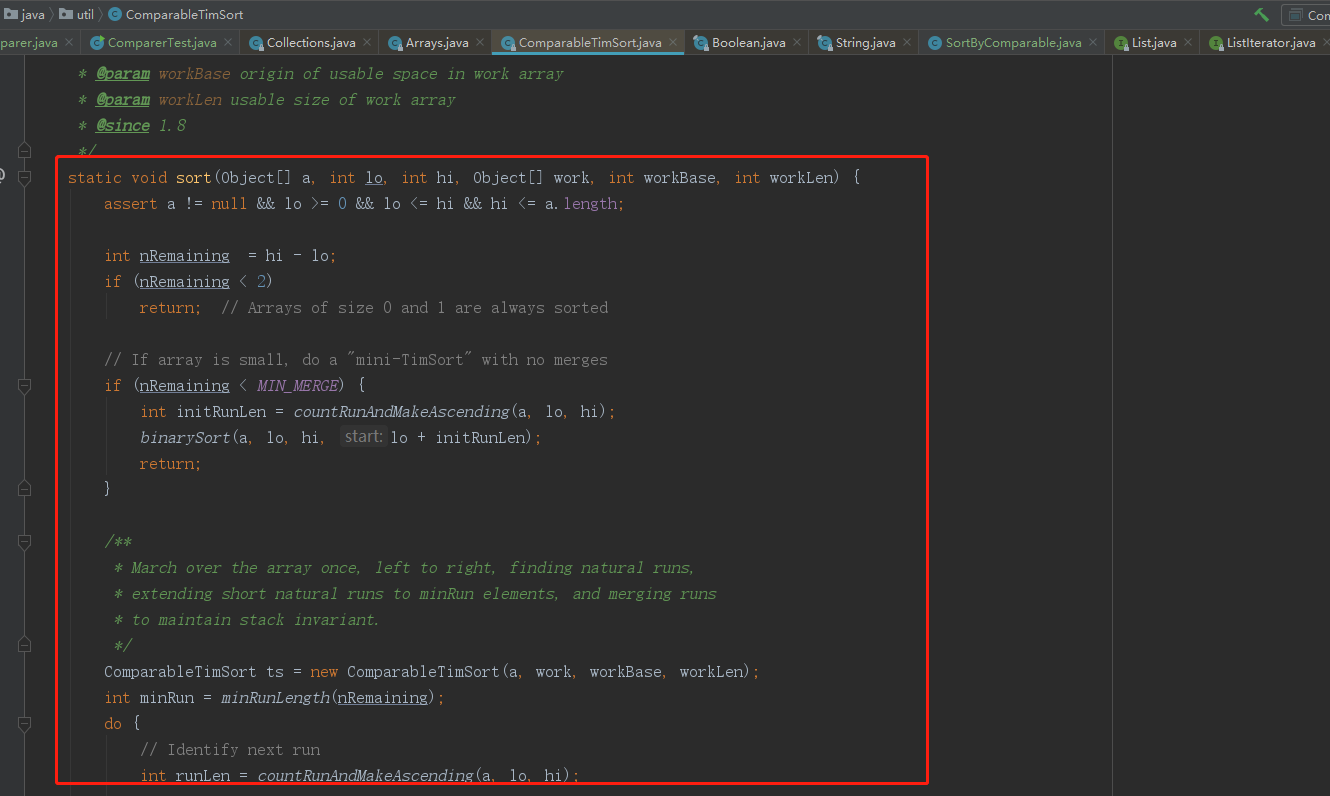
1. 进入其中legacyMergeSort方法，查看源码





该方法主要用到归并排序。

1. 进入其中ComparableTimSort.sort方法，查看源码



该排序是Tim排序（归并排序和插入排序混合排序方式）

### 关于Comparable的总结

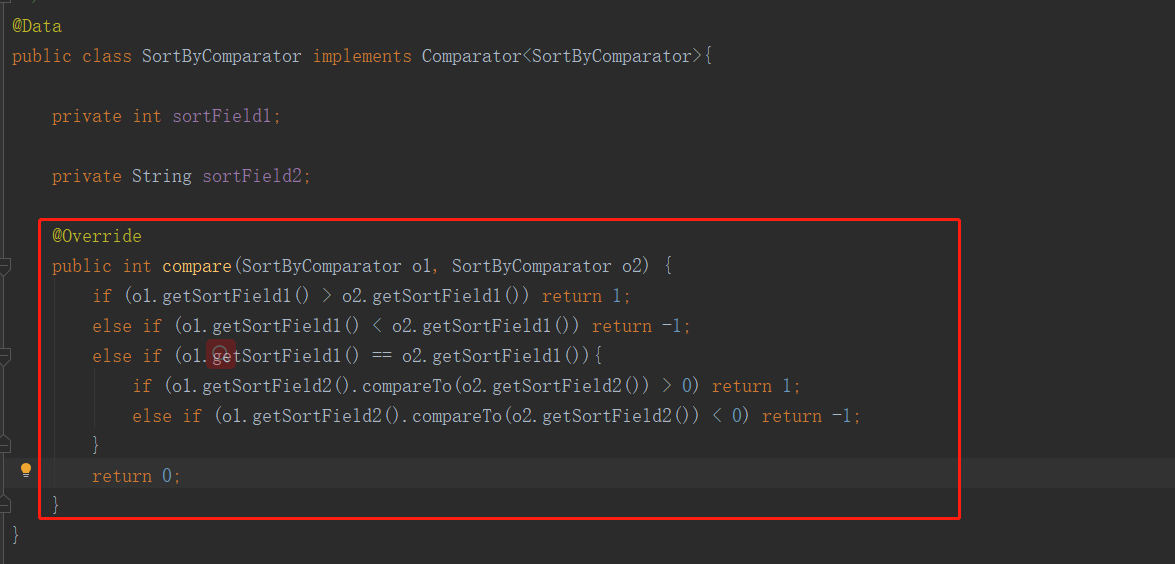
1. 类可以实现comparable接口，从而重写compareTo方法，该方法是制定两个对象之间的比较规则的。
2. 如果一个list里放了多个实现了Comparable接口类的对象，那么可以通过调用Collections.sort()排序，也可以直接调用list.sort(null)进行排序。
3. CompareTo方法的返回正值，负值，0的含义。（升序排序方式）
   * 1. 正值：前一个数据比后一个大。
     2. 负值：前一个数据比后一个小。
     3. 0：前后两个一样大。

要是想倒序排序，反一下就好。

## Comparator——外部比较器的介绍

### 编写一个SortByComparator的类

实现Comparator接口，重写compare方法：



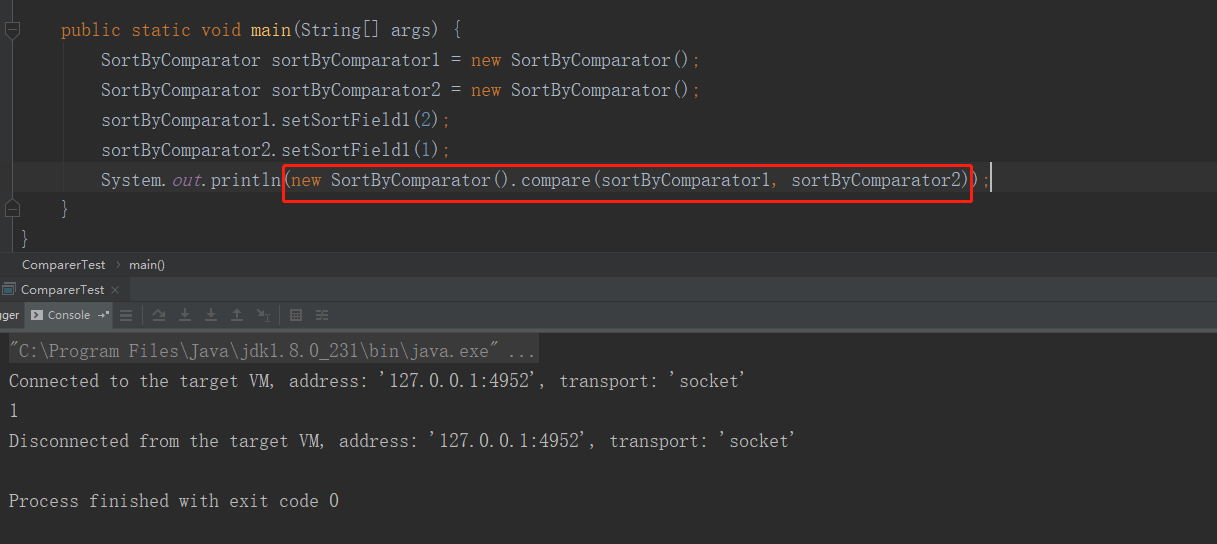
和SortByComparable类相似，不过一个是用compareTo定义两个对象的比较，一个是用compare方法定义两个对象的比较规则。

这里的比较规则就不再介绍了，和compareTo那里的应用是一模一样的，只不过不是当前对象和入参对象进行比较，这里比较的对象是两个入参。把入参o1看成当前对象就和compareTo一模一样了。

### compare方法到底怎么用

1. 比较该类两个对象的前后顺序

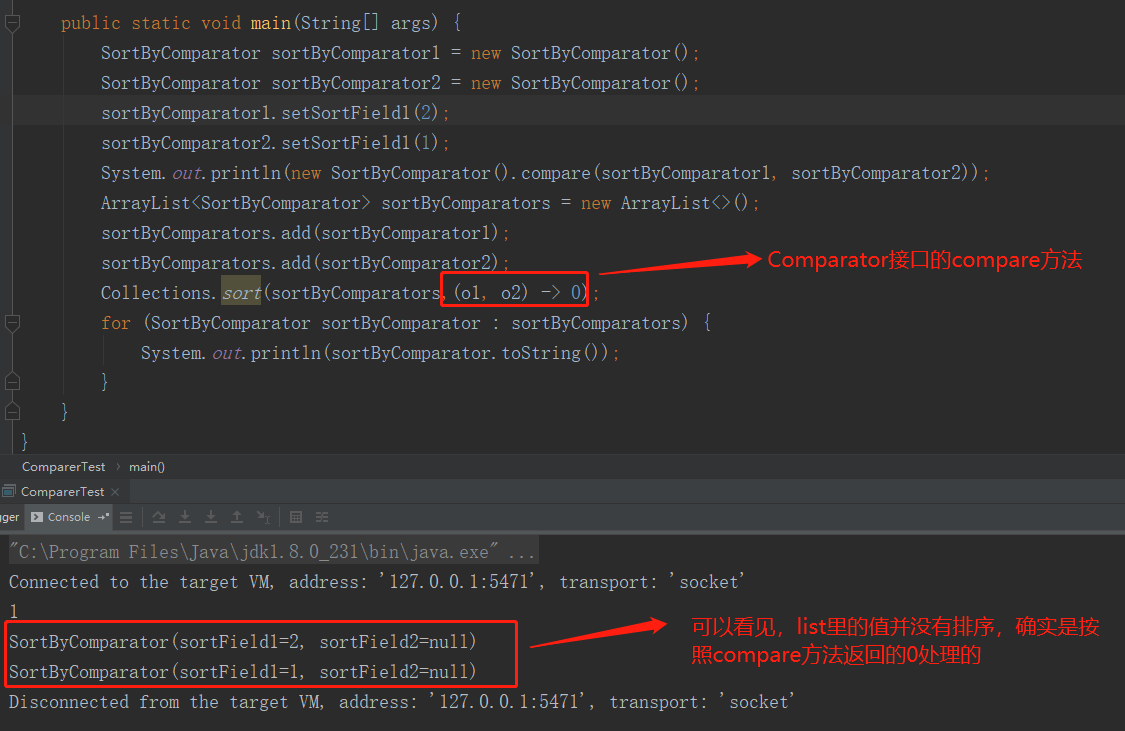
了解到compare方法的方法签名是：public int compare(o1,o2);也就是说这个方法如果要在类之外使用就得通过该类的对象进行调用。那这样的话，岂不是傻傻的；我要创建一个该类新的对象，然后调用这个compare方法，比较该类的两个对象前后顺序。如下图：



好吧，确实好傻。

1. 排序规则的灵活多变（简单性）

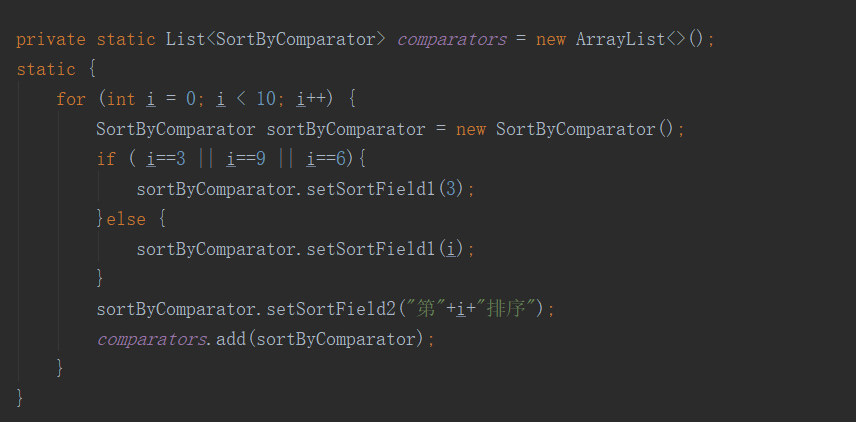
对list里的SortByComparator对象进行排序

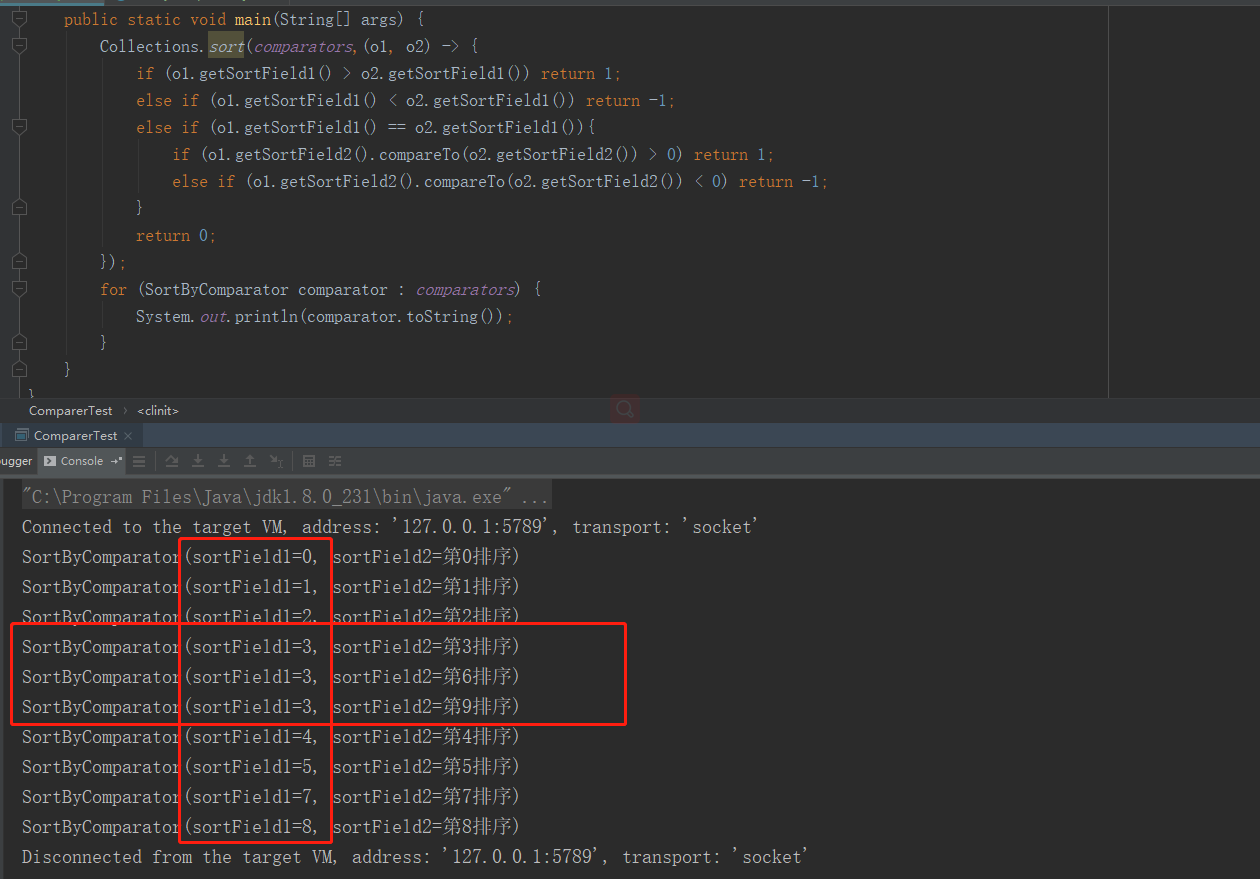


可以看见，这里在SortByComparator类里的compare方法的排序规则并没有生效，而是直接听从lambda表达式的返回0，顺序不进行变动。

从而可以得出结论：在实现了comparator接口的类里compare方法只是起了个比较两个对象的前后顺序作用（并且这种方法还是笨笨）。

1. 排序规则的灵活多变（复杂版本）



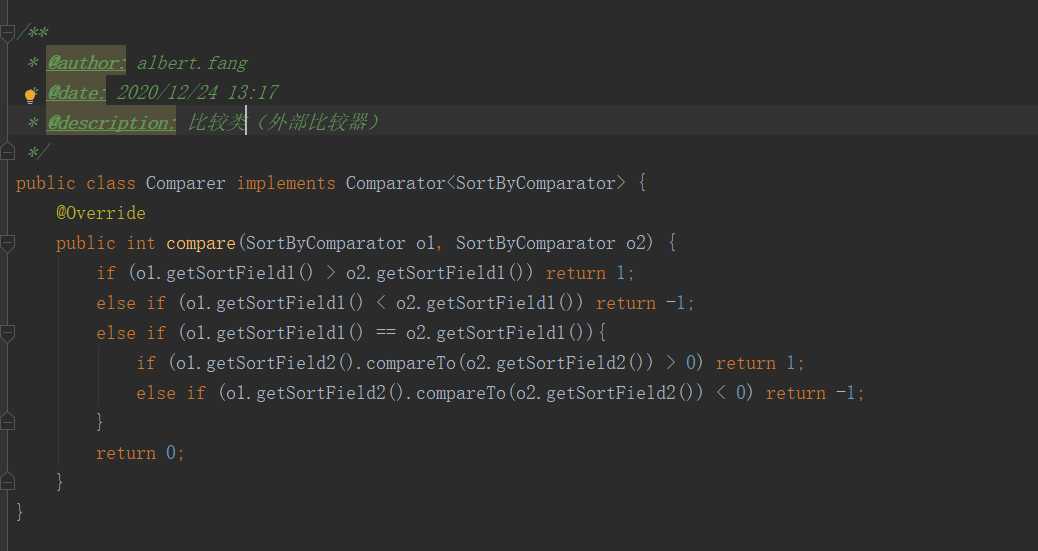


可以看到，每次对list里的对象排序，那么在调用Collections.sort(list,Comparator)方法时，都要自己定义comparator接口里的compare比较规则。这样的排序方式有个很明显的有点——灵活多变。

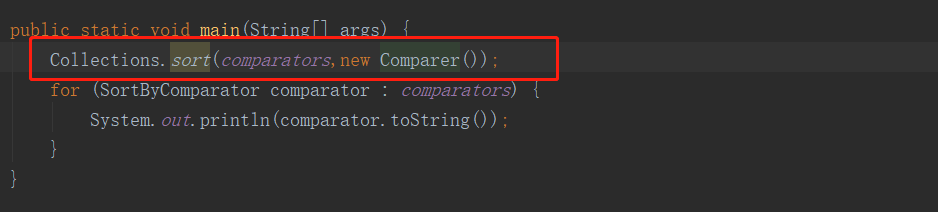
比如说，一个list里面有十个对象，我现在想要通对象的某个属性进行升序排序。在另一处我又想通过这个对象的这个年龄进行倒序排序。在这种场景，comparator这种外部比较器就很好的解决了。

如果对同一个list里的对象，都是同样规则进行排序，但是要在好多处需要用到这些排序规则，那么需要在好几处都写一个排序规则吗？

并不需要，我们可以将这个排序规则提取到一个比较类里，如下图。



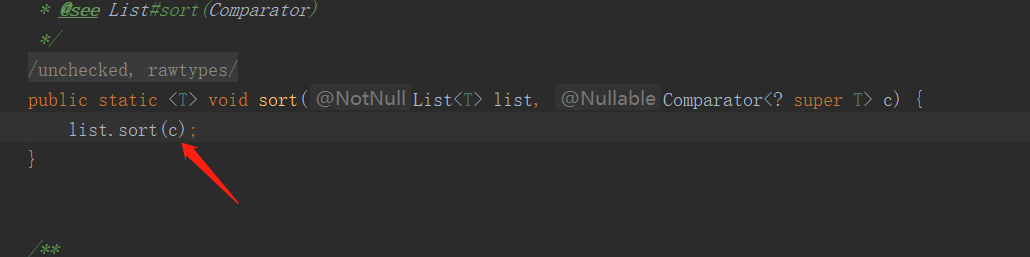
这样的话，我们就可以直接如下图使用该排序规则：



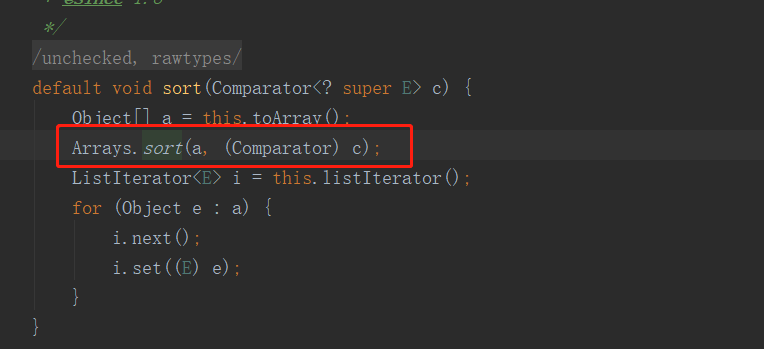
1. 如果只想对list里的对象排序，那么不需要给该对象所对应的类实现Comparator接口，和重写compare接口。

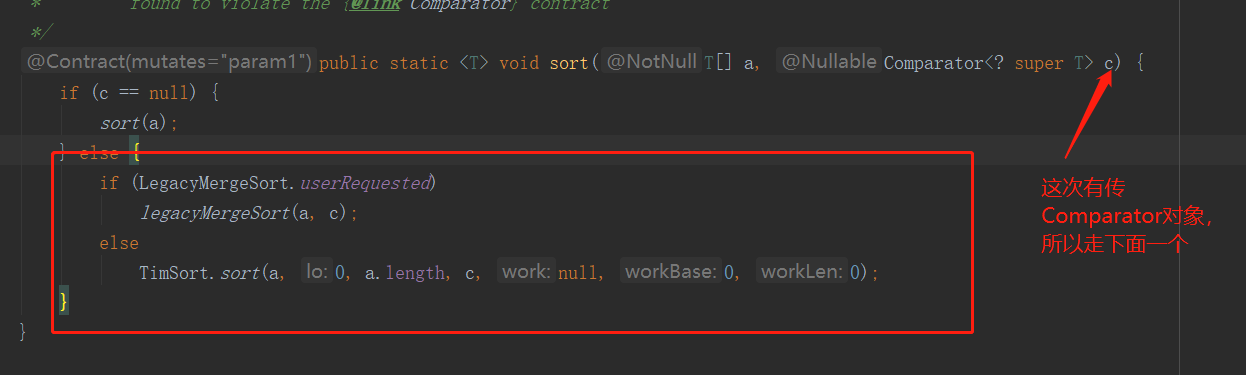
### Collections.sort(list,Comparator())源码分析

1. 可以看见，还是调用了list.sort方法，只不过这次是传了个comparator的比较器进去

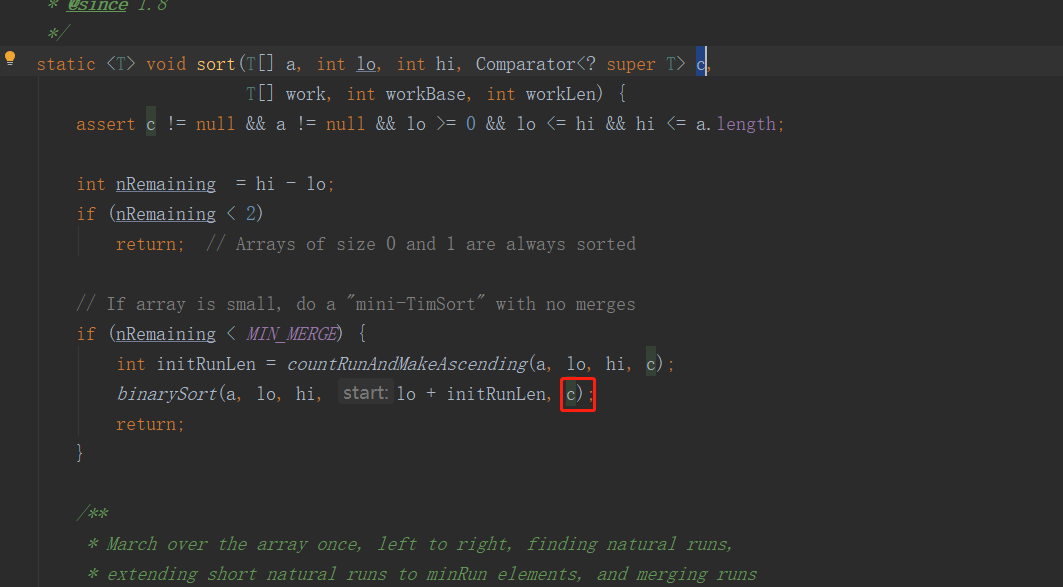


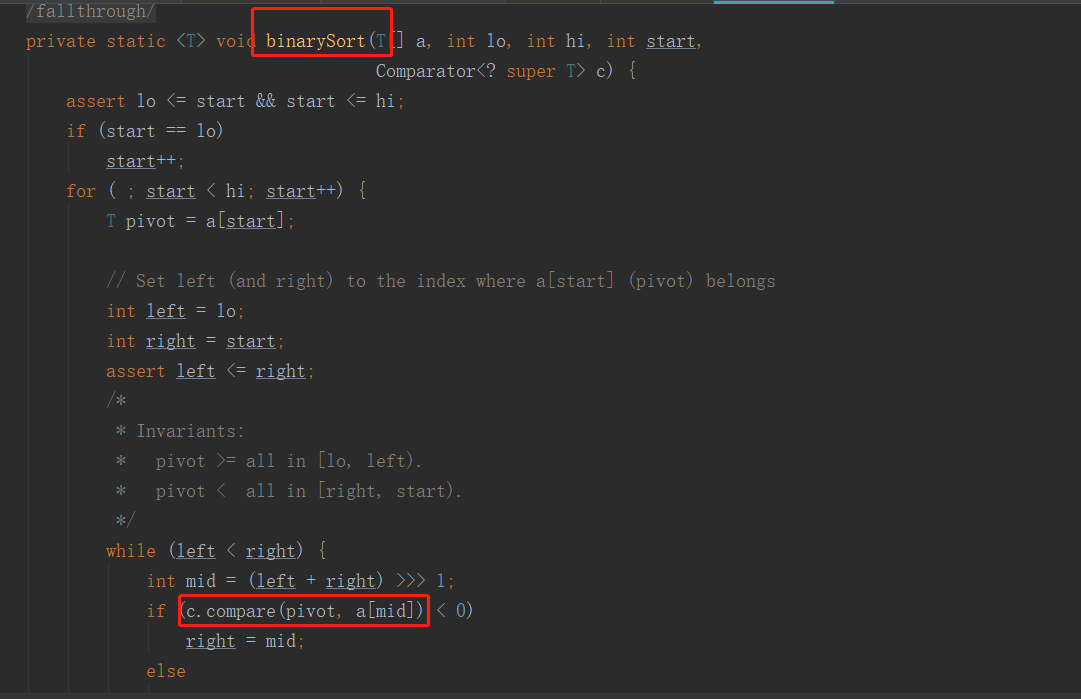
1. 跟进源码





1. 查看TimSort.sort排序





一直到binarySort才使用到传进来的比较器。

### Comparator总结

1. 排序灵活多变。可以使用Collections.sort(list,comparator)也可以直接使用list.sort(comparator);
2. 如果只是比较两个对象的前后顺序，有点笨笨的样子。
3. Compare方法的返回正值，负值，0的含义。（升序排序方式）
   * 1. 正值：前一个数据比后一个大。
     2. 负值：前一个数据比后一个小。
     3. 0：前后两个一样大。

要是想倒序排序，反一下就好。

## Comparable & Comparator异同点

共同点：

1. 都可以用于对list里的对象按照指定规则排序。
2. 可以用于比较两个对象的前后顺序。

不同点：

1. Comparator的排序更加灵活多变。
2. Comparable的对象比较看起来更舒服。

个人觉得comparator比comparable更好用