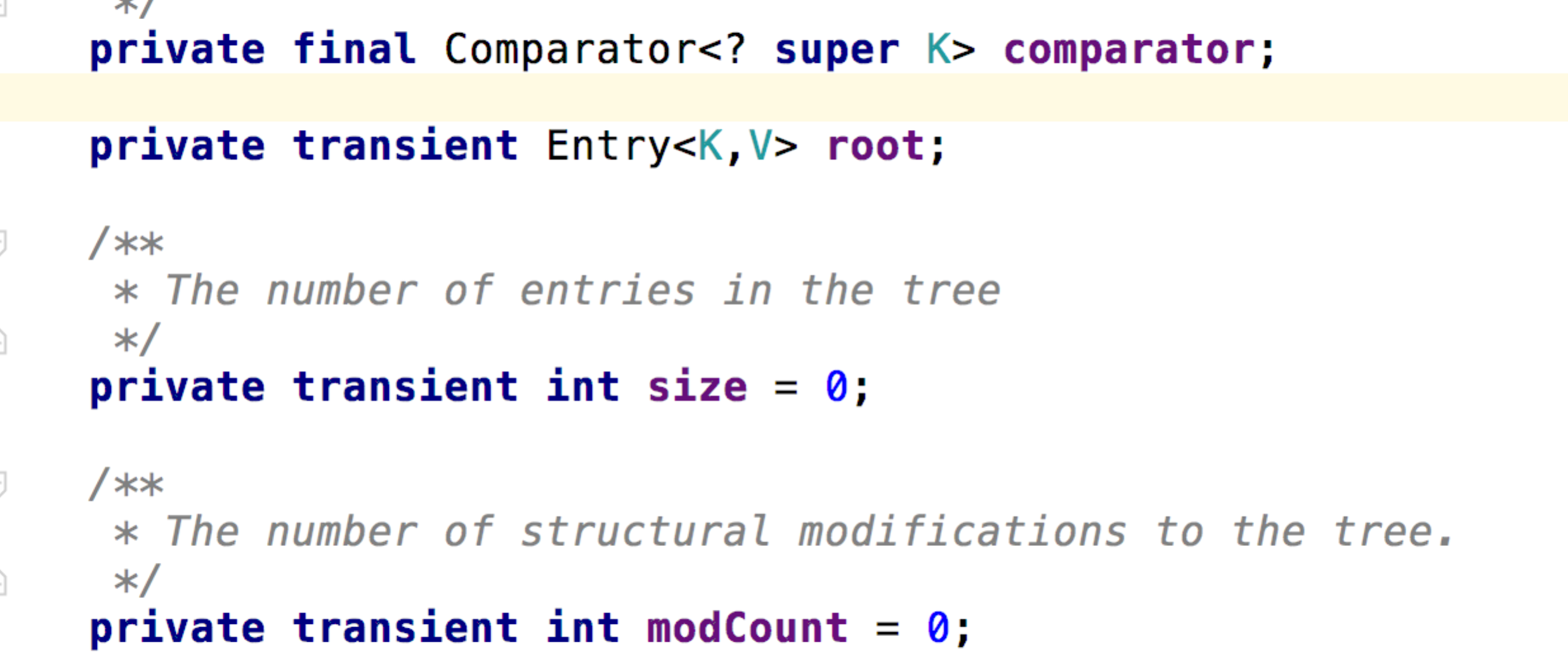
# TreeMap

**一、概述**

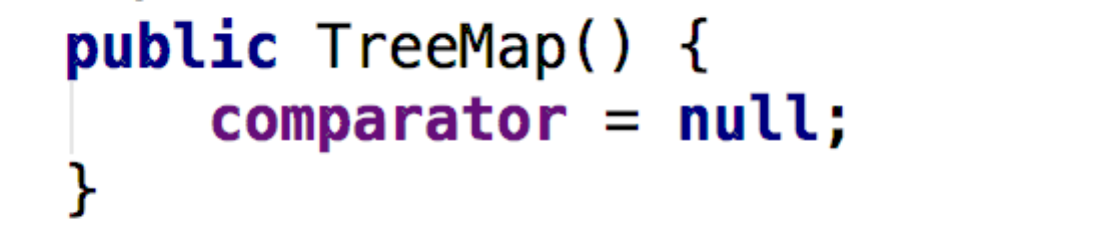
TreeMap按key排序，排序的方法可以配置comparator或实现Comparable接口。提供多种获取子集的方法例如subMap、tailMap，也提供了更丰富的查找方法，例如firstKey、lastKey、floorKey、ceilKey、lowerKey、higherKey。

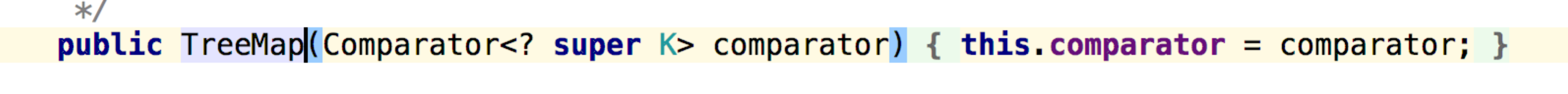
TreeMap底层是红黑树，在Hashmap源码分析中没有详细分析红黑树的实现，TreeMap中更多分析红黑树。

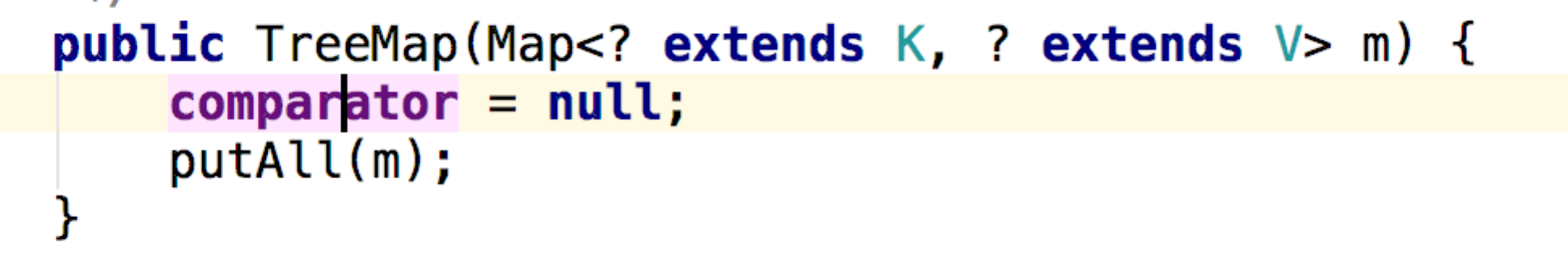
TreeMap可以配置comparator，如果配置则不使用K的compareTo方法。

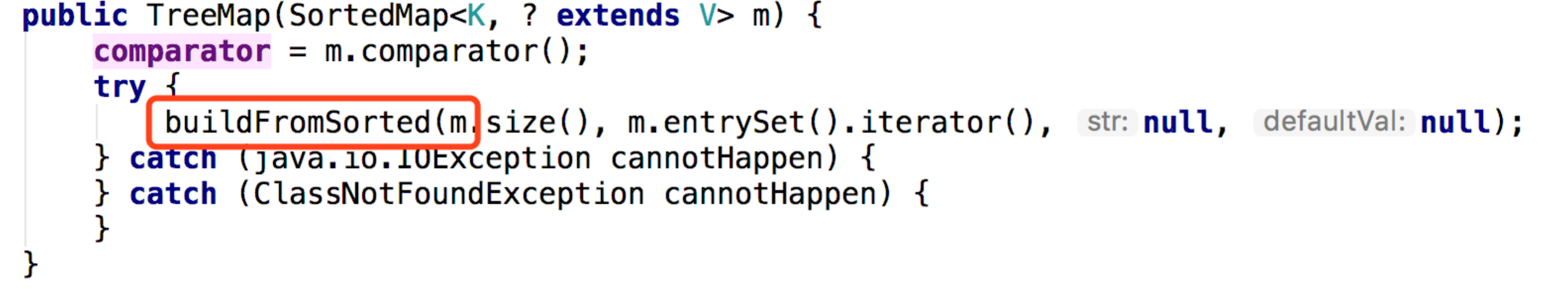


构造方法，TreeMap提供了4种构造方法





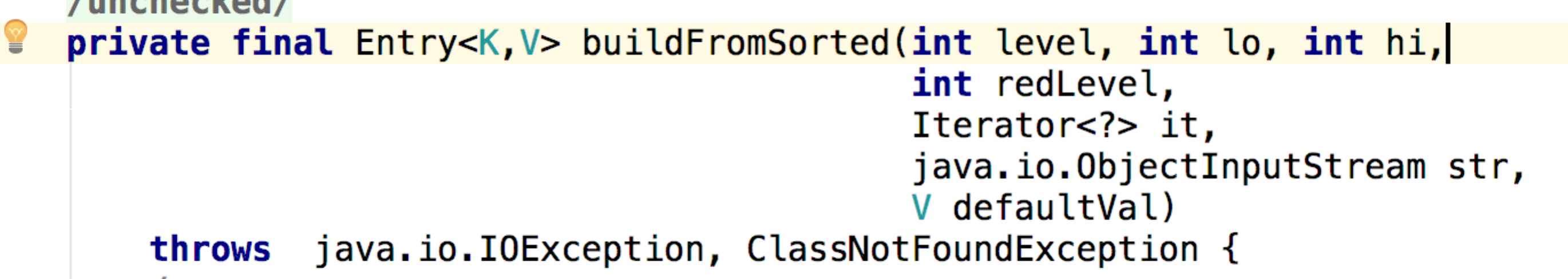




**二、buildFromSorted方法**

构造方法中putAll和buildFromSorted方法，其中putAll会判断map是否为sortedMap类型，如果是，则调用buildFromSorted，如果不是则遍历调用put方法。所以重点放在buildFromSorted和put方法。

由于本身具有有序特性，可以利用有序的特性构造新表。

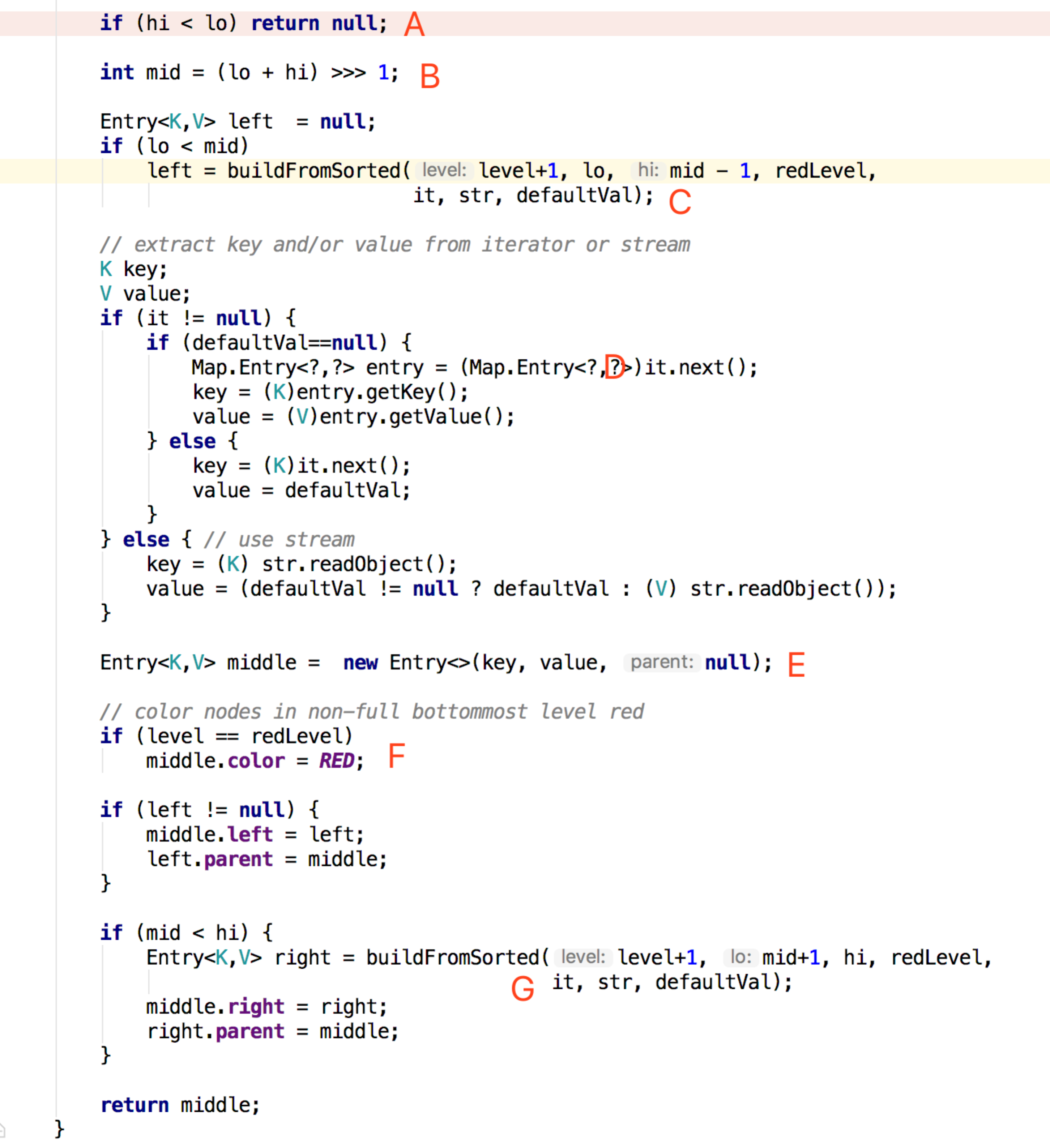


有序可以利用二分查找方法。level指当前构造的对象位于树的第几层，root为0层，依次向下递增。lo指的是二分查找的左边界，hi指的是右边界，如果原表中有25个元素，则lo == 0，hi == 24。redLeve指的是层数为redLevel标记为红色，redLevel是叶子节点的层数。it指的是原表的迭代器。str、defaultValue忽略。

创建时，会先向左递归【代码C】创建当前元素的左子树，接着创建当前节点【代码D】，再向右递归创建右子树【代码G】。

首先创建第一个节点，会一直向左递归，直到lo > hi【代码A】，返回null，现在构建红黑树中左下角的节点，也是最小的节点。迭代器会按顺序迭代元素，则第一个输出的是最小值【代码E】。这里mid的含义是元素的下标，假设25个元素，第一次循环，mid为12，则root为迭代器中第12个元素。

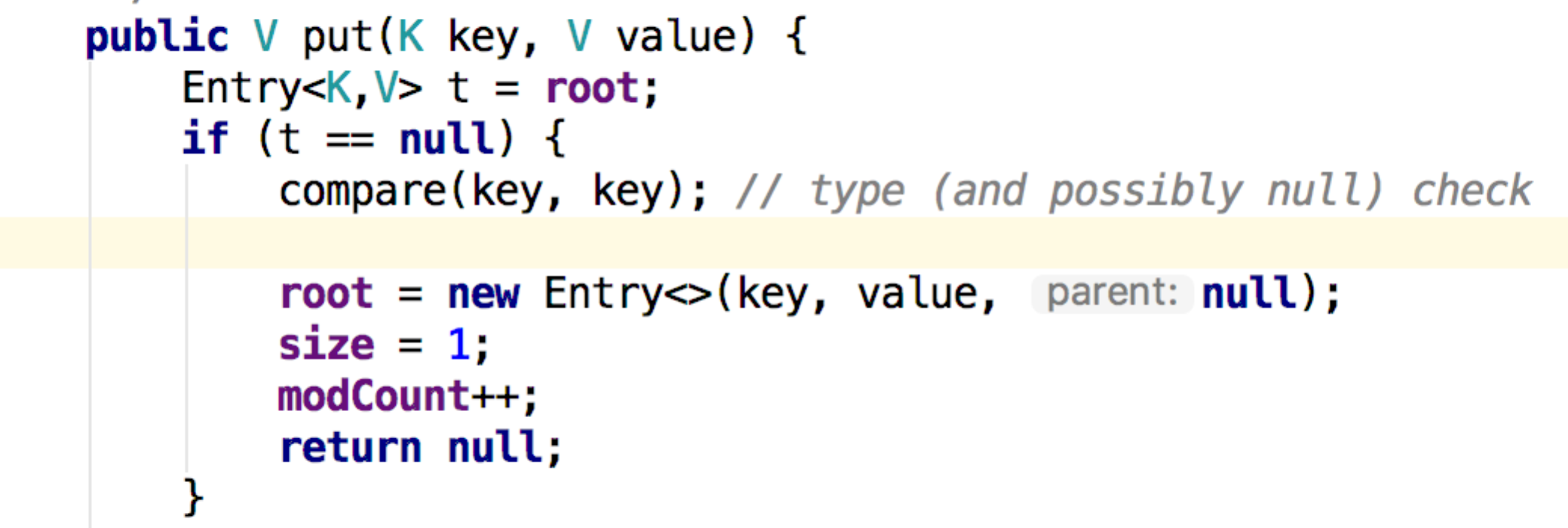
节点创建好了【代码D】，向右也是null，现在最小节点已经创建好了，同时上一层递归的左子树创建好了。接着回到上层递归，输出第二小的值，new出一个新节点，再创建右子树。



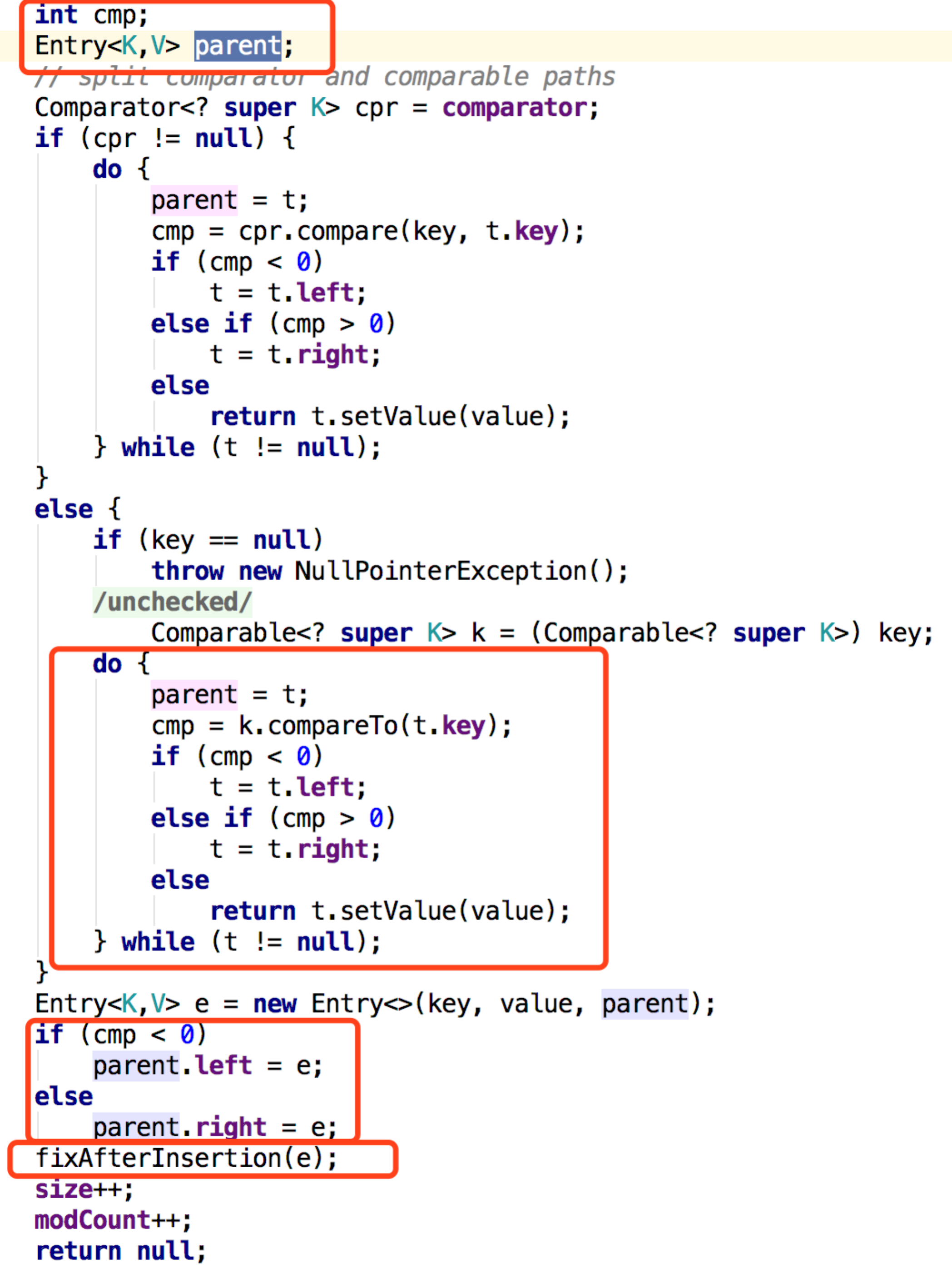
三、put方法

接着看put方法。从HashMap中获得经验，put的目标是找一个父节点，和方向，方向表示向父节点的左或右插入一个孩子。

当root为空是，直接赋值。



其中parent就是要找的父节点，cmp是方向。大体上分为使用comparator还是compareTo比较，逻辑一样。利用红黑树的搜索特性查找，如果发现两个元素相等，则覆盖并返回旧的value。否则小于向左查找，大于向右查找，如第二个红框所示。接着根据cmp决定创建左或右孩子。接着是插入后的平衡，是关注的重点。

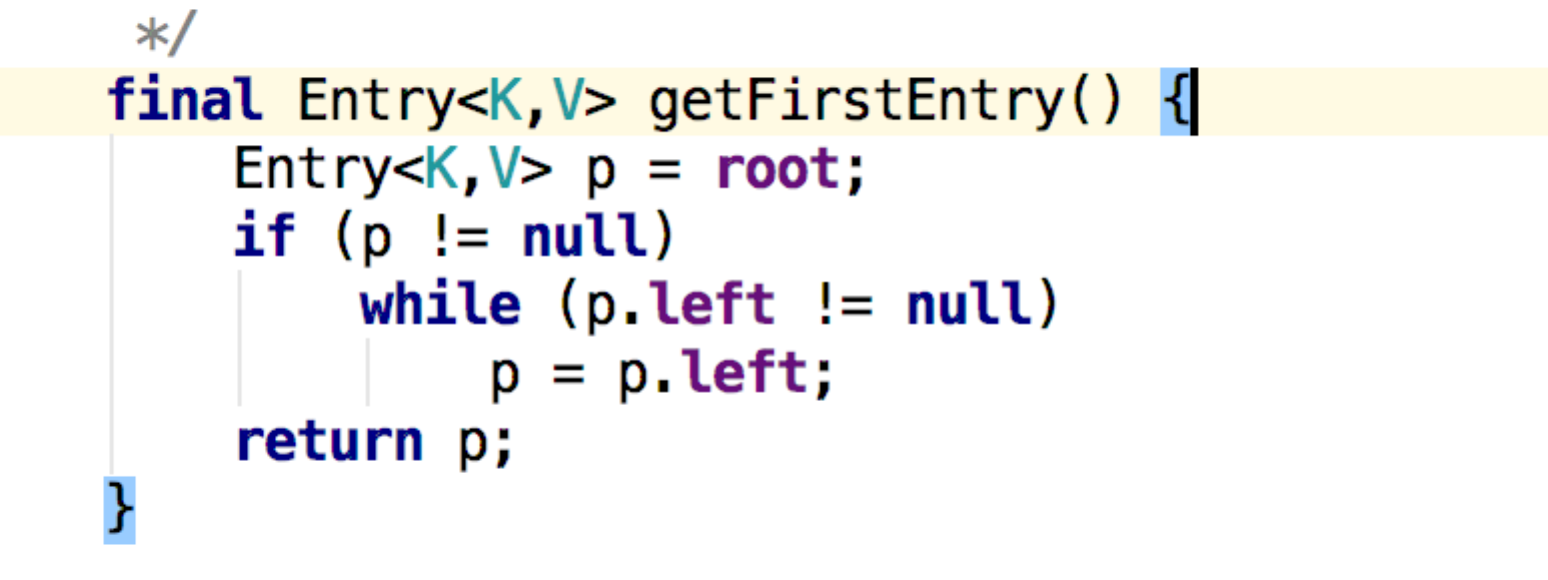


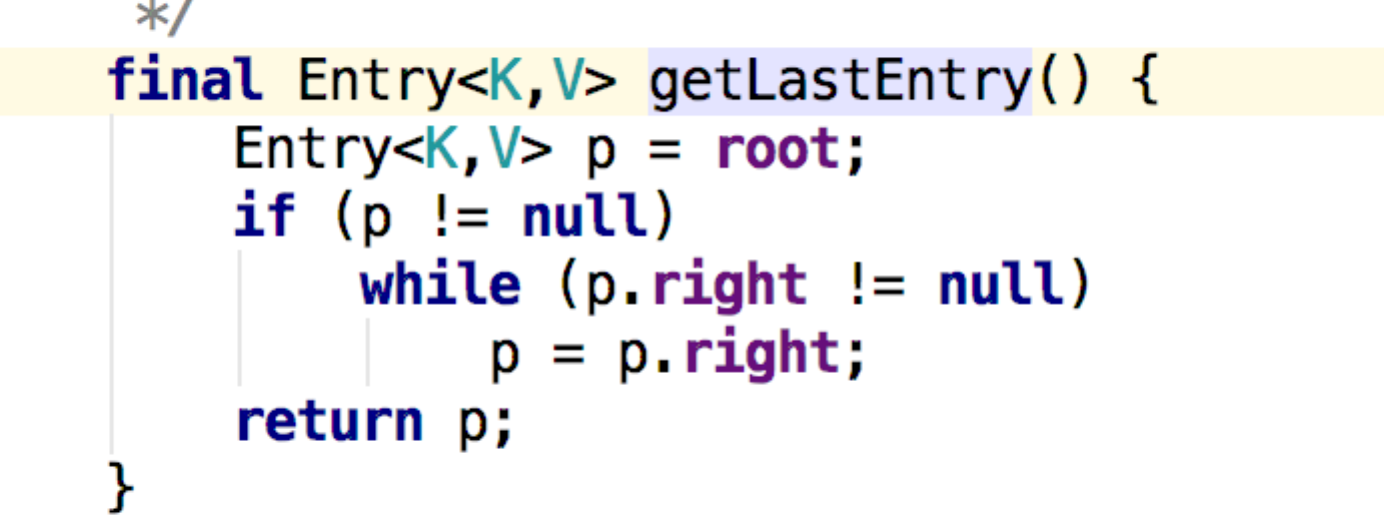
//todo

三、多种get方法

3.1 firstKey、lastKey

一直向左、右迭代





3.2 floorKey(key)、ceilKey(key)、lower(key)、highter(key)

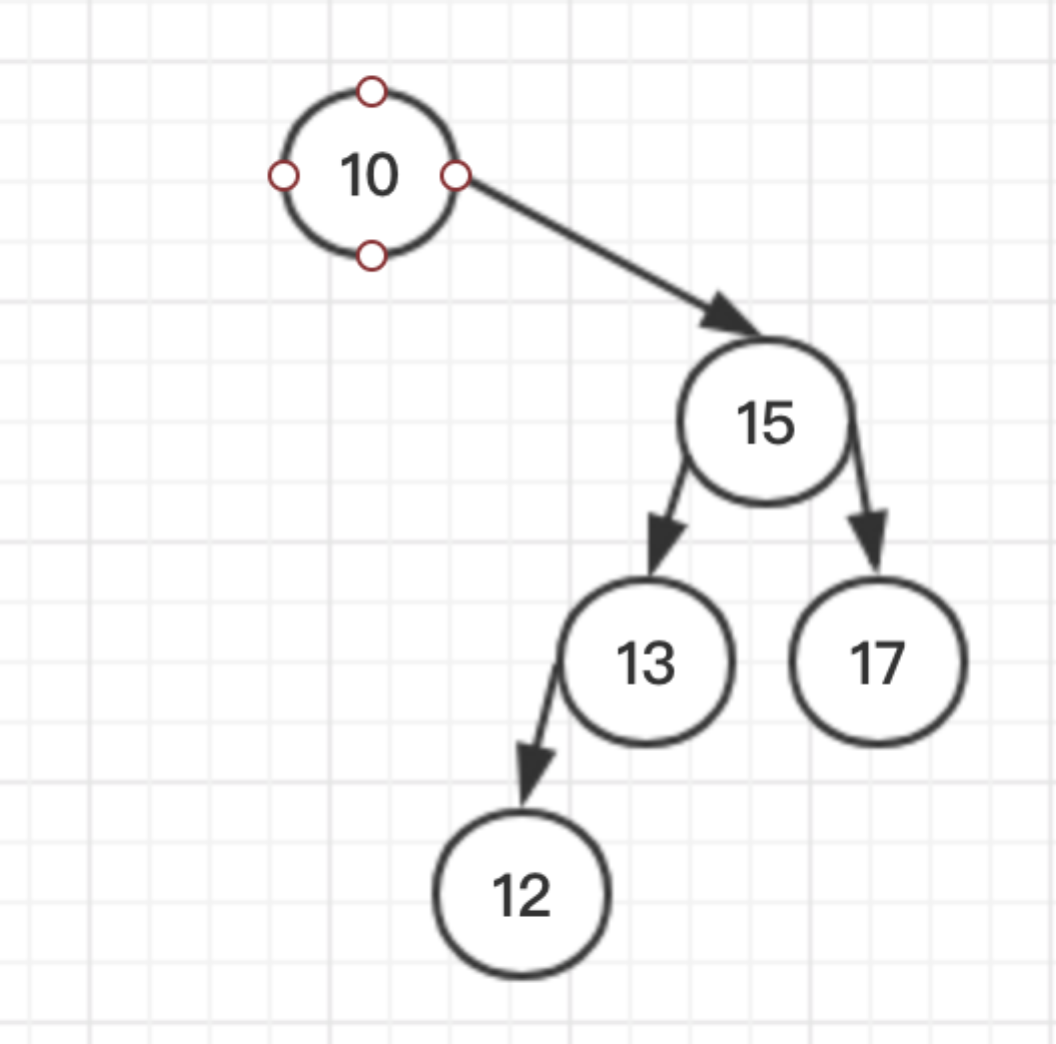
获取刚好不大于目标key的元素。

主要关注两个红框，当向右查找节点为空，则返回当前节点。因为当前节点是刚好小于且最大的元素。

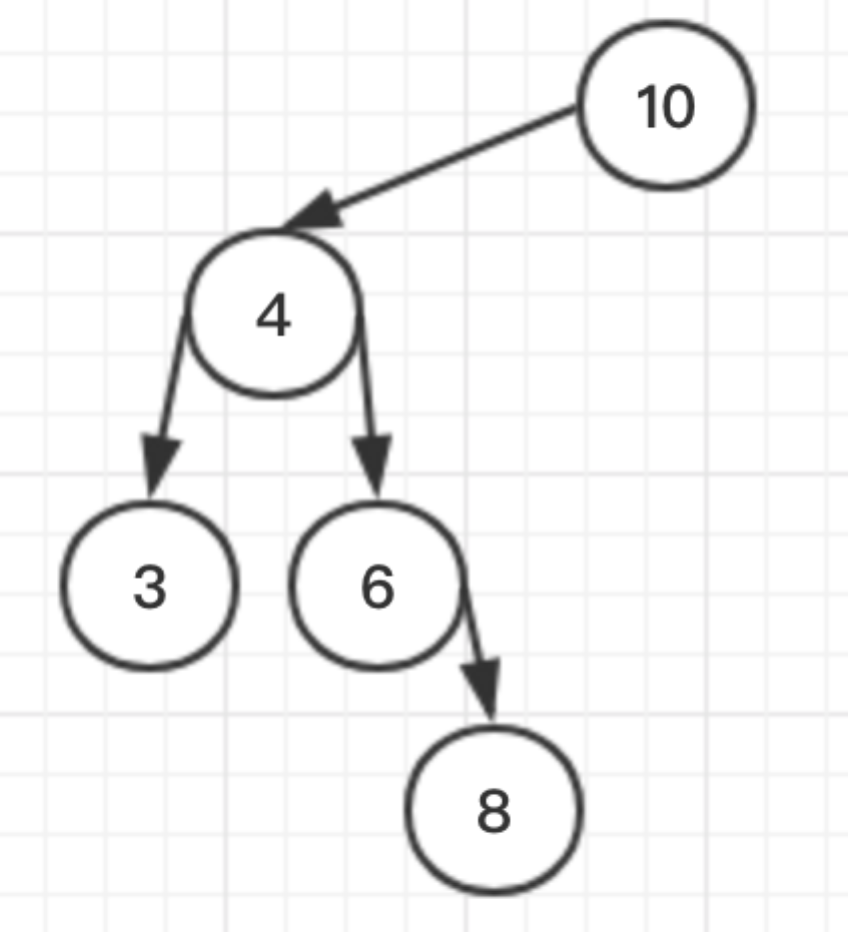
向左查找发现为空时则不同，因为当前节点大于目标节点。需要向上寻找恰好的元素。设想如果当前节点一直都是父亲的左孩子，则父亲会越来越大，不符合要求。又有一个规律，二叉搜索树任意一颗子树左下角元素都刚好大于以这颗子树为右子树的节点值。

如下图所示，如果寻找节点11，会找到12，发现left为空，则依次向上寻找，同时保存两个指针，一个指向当前节点，一个指向当前节点父节点。直到找到当前节点不是父节点的左孩子为止，并返回该父节点。





ceilKey寻找刚好不小于目标key的元素。相同的思路，只不过发现左孩子为空时直接返回；而右孩子为空时，向上寻找，如下图所示，目标值为9。



同理lowerKey和higherKey跟floor和ceil的区别是不返回相等的元素，严格小于或大于。如果发现相等，lowerKey会向左走，寻找左子树的右下角元素，higherKey向右走，寻找右子树左下角元素。这是唯一的区别。