TreeMap in Java is implemented using a Red-Black Tree, a self-balancing binary search tree. This data structure ensures that the keys are always sorted, allowing for efficient operations like searching, insertion, and deletion.

How TreeMap uses the compare() method:

* Natural Ordering:

If you don't provide a custom comparator, TreeMap uses the natural ordering of the keys. This means the keys must implement the Comparable interface, and the compareTo() method of the key class will be used to determine the order.

* Custom Comparator:

If you want to define a custom sorting order, you can provide a Comparator object when creating the TreeMap. This comparator's compare() method will be used to compare the keys.

Example using natural ordering:

Java

import java.util.TreeMap;  
  
public class TreeMapExample {  
 public static void main(String[] args) {  
 TreeMap<Integer, String> treeMap = new TreeMap<>();  
  
 treeMap.put(3, "Three");  
 treeMap.put(1, "One");  
 treeMap.put(2, "Two");  
  
 System.out.println(treeMap); *// Output: {1=One, 2=Two, 3=Three}*  
 }  
}

Example using a custom comparator:

Java

import java.util.Comparator;  
import java.util.TreeMap;  
  
public class TreeMapExample {  
 public static void main(String[] args) {  
 *// Comparator to sort strings in descending order*  
 Comparator<String> reverseComparator = (s1, s2) -> s2.compareTo(s1);  
  
 TreeMap<String, Integer> treeMap = new TreeMap<>(reverseComparator);  
  
 treeMap.put("Apple", 1);  
 treeMap.put("Banana", 2);  
 treeMap.put("Cherry", 3);  
  
 System.out.println(treeMap); *// Output: {Cherry=3, Banana=2, Apple=1}*  
 }  
}