HowToDoInJava

Java TreeMap class

Last Updated: August 30, 2020 By: Lokesh Gupta Java Collections Java Collections, Java Map

TreeMap in Java is used to store key-value pairs very similar to `HashMap` class. Difference is that TreeMap provides an efficient way to **store key/value pairs in sorted order**. It is a **red-Black tree** based `NavigableMap` implementation.

In this **Java TreeMap tutorial**, we will learn about TreeMap class, it's methods, usecases and other important details.

Table of Contents

1. TreeMap Hierarchy

The TreeMap class is declared as following in Java. It **extends AbstractMap** class and **implements NavigableMap** interface. Here `'K'` is the type of keys and `'V'` is the type

of mapped values to keys.

```
TreeMap.java

public class TreeMap<K,V>
    extends AbstractMap<K,V>
    implements NavigableMap<K,V>, Cloneable, java.io.Serializable
{
  //implementation
}
```

2. TreeMap Features

The important points about Java TreeMap class are:

- It stores key-value pairs similar to like HashMap.
- It allows only distinct keys. Duplicate keys are not possible.
- It cannot have `null` key but can have multiple `null` values.
- It stores the keys in sorted order (natural order) or by a `Comparator` provided at map creation time.
- It provides guaranteed **log(n)** time cost for the `containsKey`, `get`, `put` and `remove` operations.
- It is not synchronized. Use `Collections.synchronizedSortedMap(new TreeMap())` to work in concurrent environment.
- The iterators returned by the `iterator` method are **fail-fast**.

3. TreeMap Constructors

The TreeMap has five types of constructors:

1. **TreeMap():** creates a new, empty tree map, using the natural ordering of its keys.
2. **TreeMap(Comparator c):** creates a new, empty tree map, ordered according to the given comparator.

3. **TreeMap(Map map):** creates a new tree map containing the same mappings as the given map, ordered according to the natural ordering of its keys.
4. **TreeMap(SortedMap map):** creates a new tree map containing the same mappings and using the same ordering as the specified sorted map.

4. TreeMap Methods

The important methods we should learn about TreeMap are as follows:

1. **void clear():** It removes all the key-value pairs from the map.
2. **void size():** It returns the number of key-value pairs present in this map.
3. **void isEmpty():** It returns true if this map contains no key-value mappings..
4. **boolean containsKey(Object key):** It returns `'true'` if a specified key is present in the map.
5. **boolean containsValue(Object key):** It returns `'true'` if a specified value is mapped to at least one key in the map.
6. **Object get(Object key):** It retrieves the `value` mapped by the specified `key`, or null if this map contains no mapping for the key.
7. **Object remove(Object key):** It removes the key-value pair for the specified key from the map if present.
8. **Comparator comparator():** It returns the comparator used to order the keys in this map, or null if this map uses the natural ordering of its keys.
9. **Object firstKey():** It returns the first (least) key currently in the tree map.
10. **Object lastKey():** It returns the last (greatest) key currently in the tree map.
11. **Object ceilingKey(Object key):** It returns the least key greater than or equal to the given key, or null if there is no such key.
12. **Object higherKey(Object key):** It returns the least key strictly greater than the specified key.
13. **NavigableMap descendingMap():** It returns a **reverse order view** of the mappings contained in this map.

5. Java TreeMap Example

5.1. TreeMap Example with Natural Ordering

Java program to demonstrate the usages of TreeMap methods with natual ordering.

```
TreeMap examples

import java.util.Iterator;
import java.util.TreeMap;

public class LinkedHashMapExample
{
    public static void main(String[] args)
    {
        //Natual ordering by deafult
        TreeMap<Integer, String> pairs = new TreeMap<>();

        pairs.put(2,  "B");
        pairs.put(1,  "A");
        pairs.put(3,  "C");

        String value = pairs.get(3);    //get method

        System.out.println(value);

        value = pairs.getOrDefault(5, "oops");  //getOrDefault method

        System.out.println(value);

        //Iteration example
        Iterator<Integer> iterator =  pairs.keySet().iterator();

        while(iterator.hasNext()) {
            Integer key = iterator.next();
            System.out.println("Key: " + key + ", Value: " + pairs.get(key));
        }

        //Remove example
        pairs.remove(3);
        System.out.println(pairs);

        System.out.println(pairs.containsKey(1));    //containsKey method

        System.out.println(pairs.containsValue("B"));    //containsValue method

        System.out.println(pairs.ceilingKey(2));
    }
```

```
}
```

Program Output.

```
Console


C
oops
Key: 1, Value: A
Key: 2, Value: B
Key: 3, Value: C
{1=A, 2=B}
true
true
2
```

5.2. TreeMap Example with Custom Ordering using Comparator

```
TreeMap examples


import java.util.Iterator;
import java.util.TreeMap;

public class LinkedHashMapExample
{
    public static void main(String[] args)
    {
      //Sort keys in reverse order
        TreeMap<Integer, String> pairs = new TreeMap<>(Collections.reverseOrder())

        pairs.put(2,  "B" );
        pairs.put(1,  "A");
        pairs.put(3,  "C");

        System.out.println(pairs);
    }
}
```

Program Output.

```
Console
```

```
{3=C, 2=B, 1=A}
```

6. TreeMap Usecases

Whether using default ordering or custom ordering using comparator, TreeMap provides an efficient method to store and retrieve the information contained within in a sorted manner. This makes it excellent tool to be used in scenarios where information needs to displayed in sorted order. For example, employees information based on their age or phone numbers in in any mobile application.

Another useful usecase can be a dictionary where information is recorded and displayed in sorted fashion.

In fact, they are useful any place where the information needs to be sorted and where quick random access is necessary. If random access is not needed then rather use sorted set or list.

7. TreeMap Performance

TreeMap provides the performance of **log(n)** for most operations like add(), remove() and contains(). HashMap performs with constant-time performance O(1) for same operations. In that way, HashMap performs much better than TreeMap.

TreeMap has better performance in memory management as it does not maintain an array internally to store key-value pairs. In HashMap, array size is determined while initialization or resizing which if is often more than needed at the time. It waste the memory. There is no such problem with TreeMap.

8. Concurrency in TreeMap

Both versions of Map, HashMap and TreeMap aren't synchronized and the programmer need to manage concurrent access on the maps.

We can get the synchronized view of the treemap explicitly using **Collections.synchronizedSortedMap(new TreeMap())**.

```
Synchronized view of the TreeMap

Map<Integer, String> syncTreeMap = Collections.synchronizedSortedMap(new TreeMap<I

syncTreeMap.put(1,  "A");
syncTreeMap.put(2,  "B" );
syncTreeMap.put(3,  "C");
```

9. Conclusion

In this tutorial, we learned about **Java TreeMap** class and it's internals. We saw how it stores key-value pairs in sorted manner – either in natural ordering (default) or in some custom ordering of keys (using provided comparator).

We discussed how and when we should use TreeMap in real time applications. We compared the performance of TreeMap with HashMap to bettter understand when to use which version of Map.

Drop me your questions related to working with TreeMap in Java in comments section.

Happy Learning !!

Reference:

TreeMap Java Docs

Was this post helpful?

Let us know if you liked the post. That's the only way we can improve.

Yes

> No

Recommended Reading:

1. Java puzzle – TreeMap put operation
2. [Solved]: javax.xml.bind.JAXBException: class java.util.ArrayList nor any of its super class is known to this context
3. Java TransferQueue – Java LinkedTransferQueue class
4. Java LinkedHashMap class
5. Java PriorityBlockingQueue class
6. Java ArrayBlockingQueue class
7. Java CopyOnWriteArrayList class
8. Java CopyOnWriteArraySet class
9. Java HashSet class
0. Java LinkedHashSet class

Join 7000+ Awesome Developers

Get the latest updates from industry, awesome resources, blog updates and much more.

Email Address

Leave a Comment

Name *

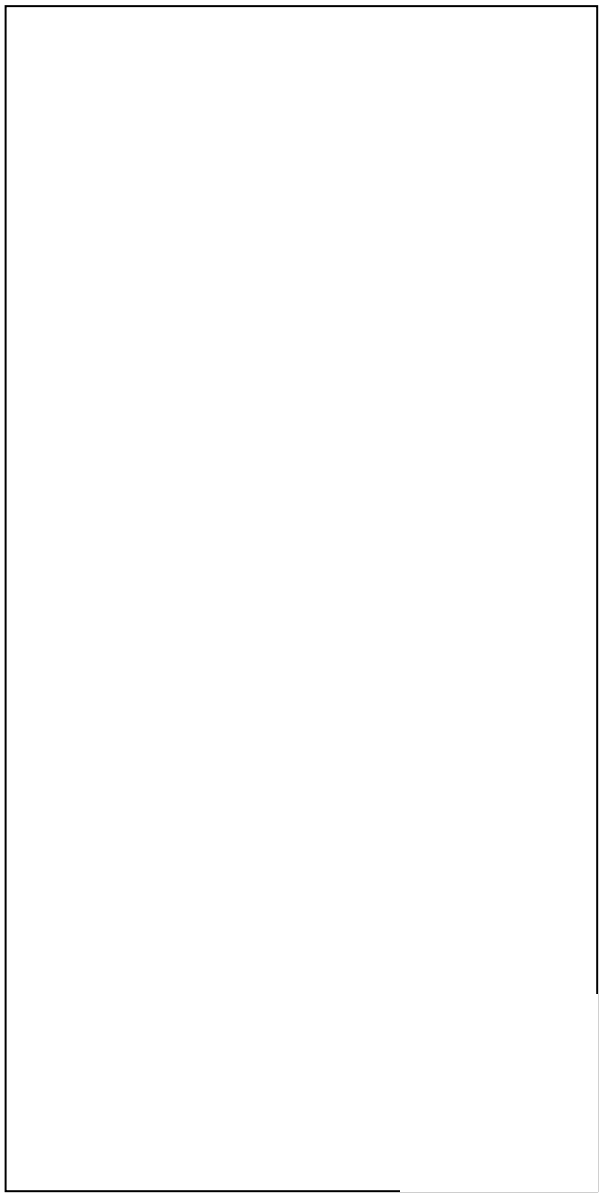
Email *

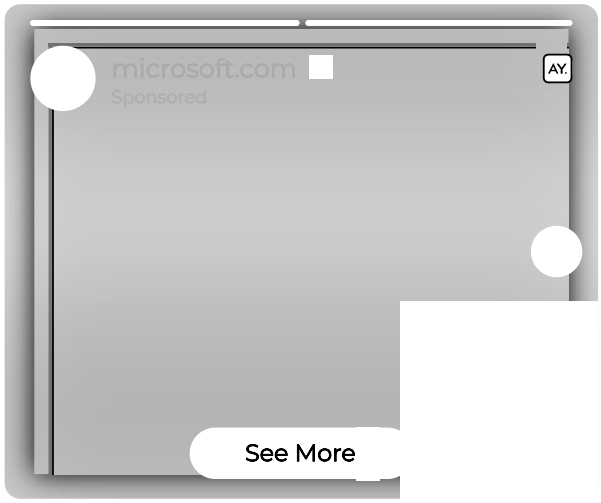
Website

☐ Add me to your newsletter and keep me updated whenever you publish new blog posts

Post Comment

Search ...





HowToDoInJava

A blog about Java and related technologies, the best practices, algorithms, and interview questions.

Meta Links

- About Me
- Contact Us
- Privacy policy
- Advertise
- Guest Posts

Blogs

REST API Tutorial