

Professor Hans Noodles

41 уровень

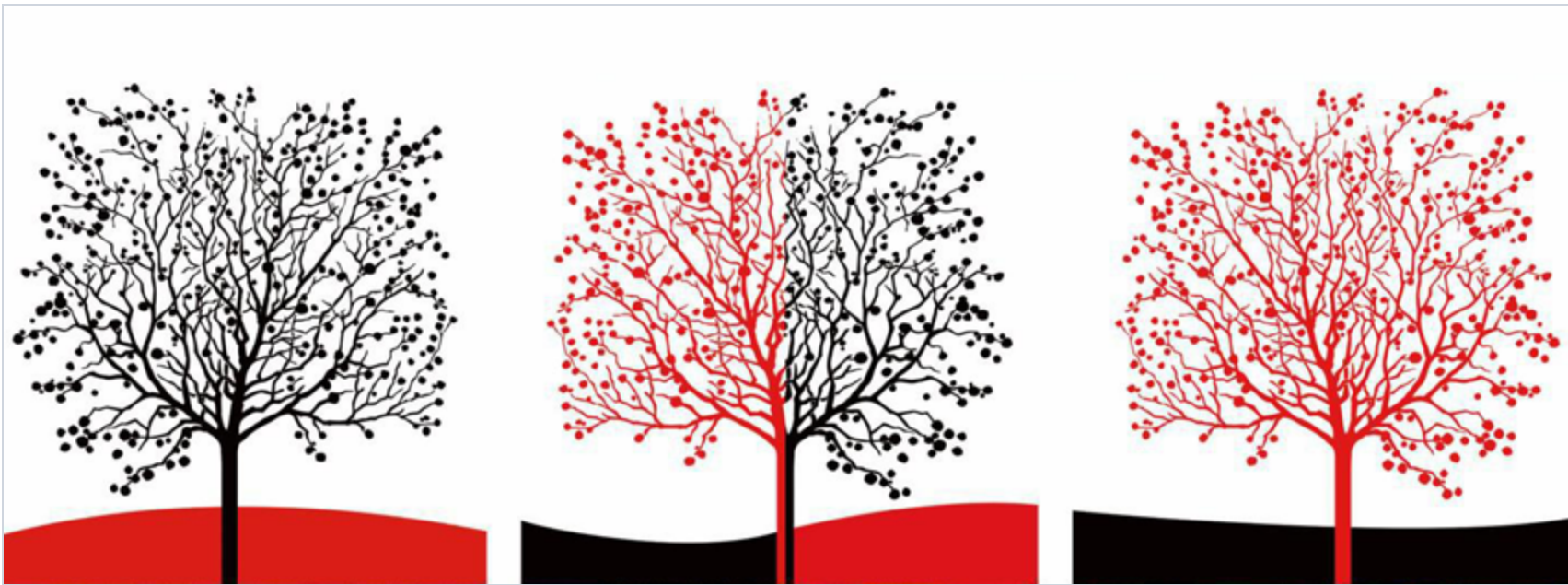
10.03.2020   68584   30

## Особенности TreeMap

Статья из группы Java Developer  
43658 участников

Вы в группе

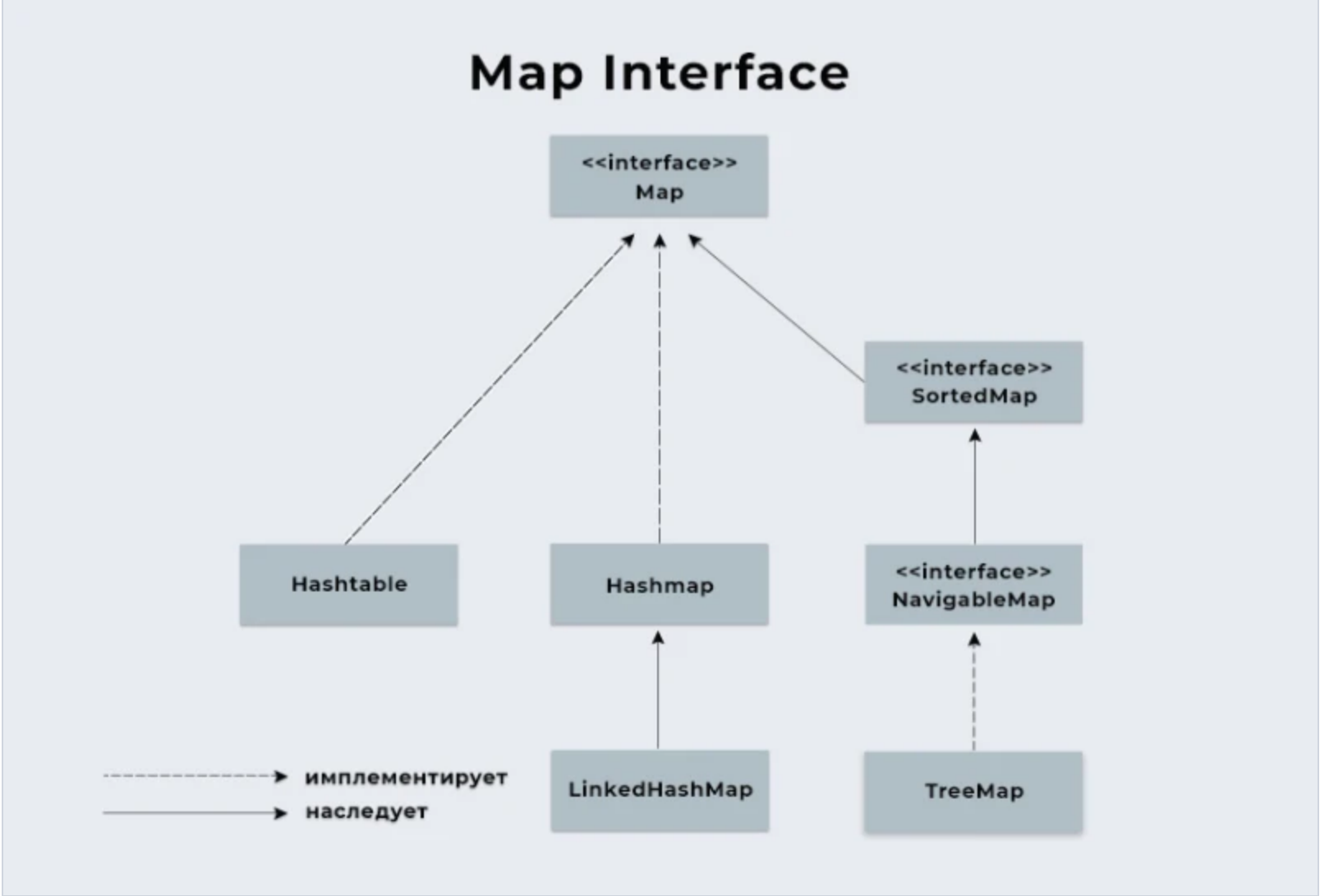
Если ты читаешь эту статью, скорее всего, ты знаком с интерфейсом `Map` и вариантами его применения. Если нет, то тебе сюда. Сегодня мы поговорим об особенностях реализации `TreeMap`, а конкретнее — чем она отличается от `HashMap` и как правильно ее использовать.



### Сравнение TreeMap, HashMap и LinkedHashMap

Наиболее используемая имплементация интерфейса `Map` — это `HashMap`. Она простая в использовании и гарантирует быстрый доступ к данным, поэтому это лучший кандидат для решения большинства задач. Большинство, но не всех. Иногда необходимо хранить данные в структурированном виде с возможностью навигации по ним. В таком случае на помощь приходит другая реализация интерфейса `Map` — `TreeMap`.

`TreeMap` имплементирует интерфейс `NavigableMap`, который наследуется от `SortedMap`, а он, в свою очередь от интерфейса `Map`.



Имплементируя интерфейсы `NavigableMap` и `SortedMap`, `TreeMap` получает дополнительный функционал, которого нет в `HashMap`, но плата за это — производительность.

Существует еще класс `LinkedHashMap`, который тоже позволяет хранить данные в определенном порядке — в порядке добавления.

Чтобы тебе были понятны **различия** между этими тремя классами, посмотри на эту таблицу:

	HashMap	LinkedHashMap	TreeMap
Порядок хранения данных	Случайный. Нет гарантий, что порядок сохранится на протяжении времени	В порядке добавления	В порядке возрастания или исходя из заданного компаратора
Время доступа к элементам	O(1)	O(1)	O(log(n))
Имплементированные интерфейсы	Map	Map	NavigableMap SortedMap Map
Имплементация на основе структуры данных	Корзины (buckets)	Корзины (buckets)	Красно-чёрное дерево (Red-Black Tree)
Возможность работы с null-ключом	Можно	Можно	Можно, если используется компаратор, разрешающий null
Потокобезопасна	Нет	Нет	Нет

Как видишь, в этих классах есть много общего, но и есть несколько отличий. Хоть класс `TreeMap` является самым многофункциональным, он не всегда может хранить `null` в качестве ключа. Кроме этого, время доступа к элементам

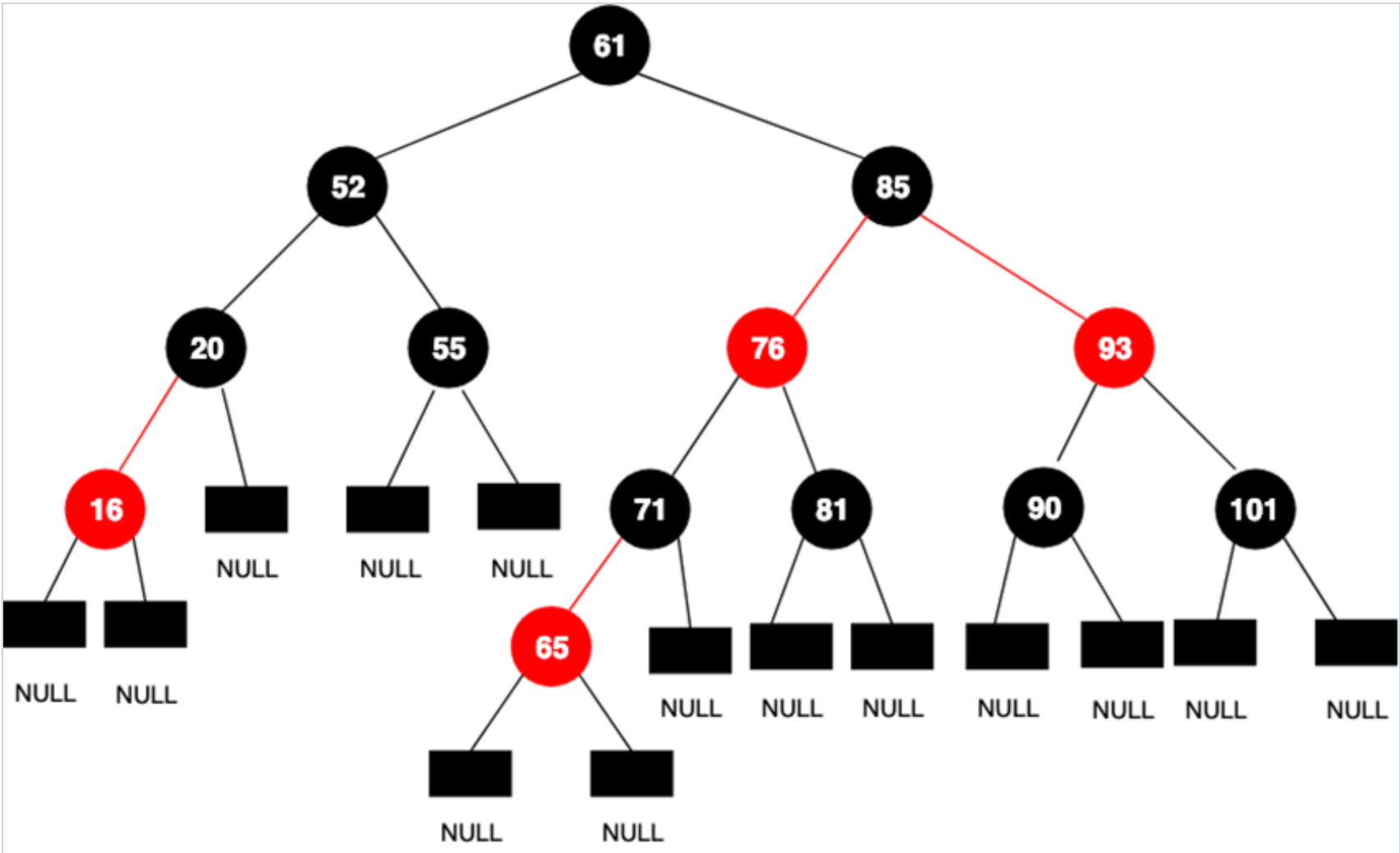
НАЧАТЬ ОБУЧЕНИЕ

Поэтому если тебе не нужно хранить данные в отсортированном виде, лучше используй `HashMap` или `LinkedHashMap`.

## Красно-чёрное дерево

Как ты наверняка заметил, под капотом `TreeMap` использует структуру данных, которая называется красно-чёрное дерево. Именно хранение данных в этой структуре и обеспечивает порядок хранения данных. Что же представляет собой это дерево? Давай разбираться!

Представь, что тебе необходимо хранить пары “Число-Строка”. Числа 16, 20, 52, 55, 61, 65, 71, 76, 81, 85, 90, 93, 101 будут ключами. Если ты хранишь данные в традиционном списке и появляется необходимость найти элемент с ключом 101, нужно будет перебрать все 13 элементов в его поисках. Для 13 элементов это не критично, при работе с миллионом у нас возникнут большие неприятности. Для решения таких проблем программисты используют немного более сложные структуры данных. Поэтому встречай красно-чёрное дерево!



<https://algorithmtutor.com/Data-Structures/Tree/Red-Black-Trees/>

Поиск нужного элемента начинается из корня дерева, в нашем случае это 61. Дальше происходит сравнение с необходимым значением. Если наше значение меньше — отправляемся в левую сторону, если больше — в правую. Так происходит до тех пор, пока не найдем необходимое значение или не упремся в элемент со значением `null` (листок дерева). Красные и черные цвета используются для упрощения навигации по дереву и его балансировки. Существуют правила, которые всегда должны быть соблюдены при постройке красно-черного дерева:

- Корень должен быть окрашен в черный цвет.
- Листья дерева должны быть черного цвета.
- Красный узел должен иметь два черных дочерних узла.
- Черный узел может иметь любые дочерние узлы.
- Путь от узла к его листьям должен содержать одинаковое количество черных узлов.
- Новые узлы добавляются на места листьев.

Если посмотреть на правила 3, 4 и 5 в совокупности, можно понять, как окраска узлов ускоряет навигацию по дереву: путь через черные узлы всегда короче, чем через красные. Поэтому по количеству именно черных узлов и определяется общий размер дерева, и называется этот размер “черная высота”.

Красно-черное дерево реализуют на разных языках программирования. В интернете существует куча реализаций для Java, поэтому не будем на нем останавливаться надолго, а продолжим знакомство с функционалом `TreeMap`.

## Методы, полученные из интерфейсов SortedMap и NavigableMap

Как и `HashMap`, `TreeMap` имплементирует интерфейс `Map`, а это значит, что в `TreeMap` есть все те методы, что и в `HashMap`. Но вдобавок `TreeMap` реализует интерфейсы `SortedMap` и `NavigableMap`, получая дополнительный функционал из них.

`SortedMap` — интерфейс, который расширяет `Map` и добавляет методы, актуальные для отсортированного набора данных:

- `firstKey()`: возвращает ключ первого элемента карты;
- `lastKey()`: возвращает ключ последнего элемента;
- `headMap(K end)`: возвращает карту, которая содержит все элементы текущей, от начала до элемента с ключом `end`;
- `tailMap(K start)`: возвращает карту, которая содержит все элементы текущей, начиная с элемента `start` и до конца;
- `subMap(K start, K end)`: возвращает карту, которая содержит все элементы текущей, начиная с элемента `start` и до элемента с ключом `end`.

`NavigableMap` — интерфейс, который расширяет `SortedMap` и добавляет методы для навигации между элементами карты:

- `firstEntry()`: возвращает первый пару “ключ-значение”;
- `lastEntry()`: возвращает последнюю пару “ключ-значение”;
- `pollFirstEntry()`: возвращает и удаляет первую пару;
- `pollLastEntry()`: возвращает и удаляет последнюю пару;
- `ceilingKey(K obj)`: возвращает наименьший ключ `k`, который больше или равен ключу `obj`. Если такого ключа нет, возвращает `null`;
- `floorKey(K obj)`: возвращает самый большой ключ `k`, который меньше или равен ключу `obj`. Если такого ключа нет, возвращает `null`;
- `lowerKey(K obj)`: возвращает наибольший ключ `k`, который меньше ключа `obj`. Если такого ключа нет, возвращает `null`;
- `higherKey(K obj)`: возвращает наименьший ключ `k`, который больше ключа `obj`. Если такого ключа нет, возвращает `null`;
- `ceilingEntry(K obj)`: аналогичен методу `ceilingKey(K obj)`, только возвращает пару “ключ-значение” (или `null`);
- `floorEntry(K obj)`: аналогичен методу `floorKey(K obj)`, только возвращает пару “ключ-значение” (или `null`);
- `lowerEntry(K obj)`: аналогичен методу `lowerKey(K obj)`, только возвращает пару “ключ-значение” (или `null`);
- `higherEntry(K obj)`: аналогичен методу `higherKey(K obj)`, только возвращает пару “ключ-значение” (или `null`);
- `descendingKeySet()`: возвращает `NavigableSet`, содержащий все ключи, отсортированные в обратном порядке;
- `descendingMap()`: возвращает `NavigableMap`, содержащую все пары, отсортированные в обратном порядке;
- `navigableKeySet()`: возвращает объект `NavigableSet`, содержащий все ключи в порядке хранения;
- `headMap(K upperBound, boolean incl)`: возвращает карту, которая содержит пары от начала и до элемента `upperBound`. Аргумент `incl` указывает, нужно ли включать элемент `upperBound` в возвращаемую карту;
- `tailMap(K lowerBound, boolean incl)`: функционал похож на предыдущий метод, только возвращаются пары от `lowerBound` и до конца;
- `subMap(K lowerBound, boolean lowIncl, K upperBound, boolean highIncl)`: как и в предыдущих методах, возвращаются пары от `lowerBound` и до `upperBound`, аргументы `lowIncl` и `highIncl` указывают, включать ли граничные элементы в новую карту.

В самой же реализации `TreeMap`, кроме привычных нам конструкторов, добавляется еще один, который принимает экземпляр компаратора. Этот компаратор и будет отвечать за порядок хранения элементов.

## Примеры использования TreeMap

Такое изобилие дополнительных методов может показаться ненужным, но очень часто они оказываются куда полезнее, чем казалось изначально.

Давай рассмотрим с тобой вот такой пример. Представь, что мы работаем в маркетинговом отделе большой компании, и у нас



- алгоритм показа рекламы для несовершеннолетних отличается.

Создадим класс `Person`, в котором будет храниться вся доступная нам информация о человеке:

```
1 public class Person {
2     public String firstName;
3     public String lastName;
4     public int age;
5
6     public Person(String firstName, String lastName, int age) {
7         this.firstName = firstName;
8         this.lastName = lastName;
9         this.age = age;
10    }
11 }
```

Логику реализуем в классе `Main`:

```
1 import java.util.Comparator;
2 import java.util.Map;
3 import java.util.TreeMap;
4
5 public class Main {
6     public static void main(String[] args) {
7         TreeMap<Person, Integer> map = new TreeMap<>(Comparator.comparingInt(o -> o.age));
8         map.put(new Person("John", "Smith", 17), 0);
9         map.put(new Person("Ivan", "Petrenko", 65), 0);
10        map.put(new Person("Pedro", "Escobar", 32), 0);
11        map.put(new Person("Radion", "Pyatkin", 14), 0);
12        map.put(new Person("Sergey", "Vashkevich", 19), 0);
13
14        Person firstAdultPerson = map.navigableKeySet().stream().filter(person -> person.age>18).findFirst().get();
15
16        Map<Person, Integer> youngPeopleMap = map.headMap(firstAdultPerson, false);
17        Map<Person, Integer> adultPeopleMap = map.tailMap(firstAdultPerson, true);
18        showAdvertisementToYoung(youngPeopleMap);
19        showAdvertisementToAdult(adultPeopleMap);
20    }
21
22    public static void showAdvertisementToYoung(Map map){}
23    public static void showAdvertisementToAdult(Map map){}
24 }
```

В классе `Main` создаем `TreeMap`, где `key` — это конкретный человек, а `value` — количество показов рекламы в этом месяце. В конструкторе передаем компаратор, который отсортирует людей по возрасту.

Заполняем `map` случайными значениями.

Теперь нам нужно получить ссылку на первого взрослого человека в нашем мини-хранилище данных. Делаем это с помощью Stream API.

После этого получаем две независимые карты, которые передаем в методы, показывающие рекламу.

или подсказками среды разработки.

На этом все! Надеюсь, теперь класс `TreeMap` для тебя понятен, и ты найдешь ему точное применение в решении практических задач.

−

+165

+

Комментарии (30)

популярные

новые

старые

JavaCoder

Введите текст комментария

**Петр Селищев**    Уровень 26, Санкт-Петербург, Россия

2 декабря 2021, 14:49

...

Для лучшего понимания принципов работы класса "TreeMap" рекомендую почитать и посмотреть про "красно-черные деревья" , предварительно погуглив про "бинарные" и "2-3-4  деревья"  
1   <https://www.youtube.com/watch?v=a9EwBVLQ364>  
2   <https://www.youtube.com/watch?v=n7Y2karbxF4&t=6s>  
3   <https://www.happycoders.eu/algorithms/red-black-tree-java/>  
4   <https://www.youtube.com/watch?v=u-ilAwbJWYc>    // детальное объяснение класса TreeMap

Ответить

−

+6

+

**Dmitry Koichev**    QA Automation Engineer в **The Product Engine**

30 марта 2021, 13:46

...

По какому принципу определяется корень дерева ? это первый добавленный в мэпу элемент или нет ?

Ответить

−

+3

+

**Wiktor1**    Уровень 41, Минск

3 апреля 2021, 11:55

...

Да, это первый добавленный элемент. Потом при балансировке дерева корнем может стать другой элемент

Ответить

−

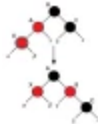
+3

+

**Max Nazaruk**    Уровень 40, Ивано-Франковск, Украина

21 июня 2021, 16:30

...



Ответить

−

+1

+

**Сергей**    Уровень 38

21 января 2021, 22:06

...

При попытке положить в TreeMap null-ключ в любом случае возникает NPE. В лекции написано "Можно, если не используется компаратор" - какая разница, тогда класс ключей должен реализовывать интерфейс Comparable<>.

Ответить

−

0

+

**Prokhorov Artem**    Уровень 24, Москва, Россия

29 августа, 23:11

...

Возможно лекцию подкорректировали, но сейчас указано, что можно использовать null если используется comparator обрабатывающий null

Ответить

−

0

+

**Deniska**    Ученик чародея в **тредевятом царстве**

23 декабря 2020, 11:35

...

а как получить минимальное значение в treeMap не зная его?

Ответить

−

+2

+

**Сергей**    Уровень 38

21 января 2021, 21:51

...

НАЧАТЬ ОБУЧЕНИЕ

Collections.min(treeMap.values());  
treeMap.values().stream().min(String::compareTo);  
и т.д.

Ответить

+4

Regina C

QA Auto Engineer в -

13 декабря 2020, 02:01

все таки в map.put(new Person("Radion", "Pyatkin", 14), 0); - имя Родион, а не Радион)))

Ответить

0

Anonymou**s** #2491313

Java Developer в Росатом

24 апреля 2021, 09:38

В недрах ЗАГСа или паспортного отдела вполне мог появиться и Радион.

Ответить

+3

Regina C

QA Auto Engineer в -

13 декабря 2020, 01:58

кто здесь, потому что на 19 уровне есть задача с ключами и зарплатой?

Ответить

+13

Эдем

Уровень 26

15 декабря 2020, 20:37

На 18 использовал TreeSet в задачке про склеивание частей файлов. Решил про TreeMap глянуть. Ну и на тебя поглядеть, милая :)

Ответить

+1

Эдем

Уровень 26

22 декабря 2020, 02:31

Добрался до этой задачки и снова вернулся, теперь с TreeMap :)

Ответить

0

Рустем

Уровень 24, Уфа, Россия

25 декабря 2020, 22:20

100 %, я из-за этого =)

Ответить

0

Oleksii

Уровень 36, Харьков

11 июня 2021, 15:04

me too

Ответить

0

wan-derer.ru

Уровень 40, Москва, Россия

13 ноября 2020, 19:30

Интересно, но хотелось бы больше примеров этих методов, желательно пока без stream. А то хочу пройти по TreeMap в обратном порядке, а мне IDE красит методы красным и говорит что не понимает чего я хочу.

Ответить

0

Pig Man

Главная свинья в Свинарнике

26 февраля 2021, 13:19

Стримы - это наше все

Ответить

+2

Mikhail Knyazev

Уровень 1

22 октября 2020, 10:18

Схема Map Interface не соответствует описанию в тексте. Вот эта фраза:  
**интерфейс NavigableMap, который наследуется от SortedMap**

Ответить

0

Arjun

Уровень 37, Санкт-Петербург, Россия

14 октября 2020, 16:13

Приведенная картинка красно-черного дерева не соответствует описанным правилам. Вообще логика определения красных и черных узлов не понятна.

Ответить

+7

Pavlo Plynko

Java-разработчик в JavaRush

EXPERT

11 марта 2021, 13:10

Уже исправили картинку? Или какое из правил не соответствует?

Ответить

0

Александр Жарков

Уровень 20, Екатеринбург, Россия

16 июня 2020, 12:06

*Поэтому если тебе не нужно хранить данные в отсортированном виде, лучше используй HashMap или LinkedHashMap.*

**По умолчанию данные в HashMap хранятся далеко не в отсортированном виде.**

НАЧАТЬ ОБУЧЕНИЕ

Евгений

Ведущий инженер в ПАО Сбербанк

EXPERT

15 июля 2020, 21:19

...

Погугли про buckets (вёдра) в NashMap. Ещё вот тут глянь [ссылка](#)

Ответить

0

Владимир Чугуевец

Уровень 50, Краснодар, Россия

27 марта, 16:06

...

Тоже не понятно это противоречие.

Ответить

0

Показать еще комментарии

ОБУЧЕНИЕ

- Курсы программирования
- Курс Java
- Помощь по задачам
- Подписки
- Задачи-игры

СООБЩЕСТВО

- Пользователи
- Статьи
- Форум
- Чат
- Истории успеха
- Активности

КОМПАНИЯ

- О нас
- Контакты
- Отзывы
- FAQ
- Поддержка



JavaRush — это интерактивный онлайн-курс по изучению Java-программирования с нуля. Он содержит 1200 практических задач с проверкой решения в один клик, необходимый минимум теории по основам Java и мотивирующие фишки, которые помогут пройти курс до конца: игры, опросы, интересные проекты и статьи об эффективном обучении и карьере Java-девелопера.

ПОДПИСЫВАЙТЕСЬ

ЯЗЫК ИНТЕРФЕЙСА

Русский

▼

