Keeping Keys Sorted with SortedMap and NavigableMap

Methods Added by SortedMap

The JDK provides two extensions of the `Map` interface: `SortedMap` and `NavigableMap`. `NavigableMap` is an extension of `SortedMap`. Both interfaces are implemented by the same class: `TreeMap`. The `TreeMap` class is a red-black tree, a well-known data structure.

`SortedMap` and `NavigableMap` keep their key/value pairs sorted by key. Just as for `SortedSet` and `NavigableSet`, you need to provide a way to compare these keys. You have two solutions to do this: either the class of your keys implements `Comparable`, or you provide a `Comparator` for your keys when creating your `TreeMap`. If you provide a `Comparator`, it will be used even if your keys are comparable.

If the implementation you chose for your `SortedMap` or `NavigableMap` is `TreeMap`, then you can safely cast the set returned by a call to `keySet()` or `entrySet()` to `SortedSet` or `NavigableSet`. `NavigableMap` has a method, `navigableKeySet()` that returns an instance of `NavigableSet` that you can use instead of the plain `keySet()` method. Both methods return the same object.

The `SortedMap` interface adds the following methods to `Map`:

- `firstKey()` and `lastKey()`: returns the lowest and the greatest key of your map;

- headMap(toKey) and tailMap(fromKey): returns a SortedMap whose keys are strictly less than toKey, or greater than or equal to fromKey;
- subMap(fromKey, toKey): returns a SortedMap whose keys are strictly lesser than toKey, or greater than or equal to fromKey.

These maps are instances of SortedMap and are views backed by this map. Any change made to this map will be seen in these views. These views can be updated, with a restriction: you cannot insert a key outside the boundaries of the map you built.

You can see this behavior on the following example:

```
SortedMap<Integer, String> map = new TreeMap<>();
map.put(1, "one");
map.put(2, "two");
map.put(3, "three");
map.put(5, "five");
map.put(6, "six");

SortedMap<Integer, String> headMap = map.headMap(3);
headMap.put(0, "zero"); // this line is ok
headMap.put(4, "four"); // this line throws an IllegalArgumentException
```

Methods Added by NavigableMap

Accessing to Specific Keys or Entries

The NavigableMap adds more methods to SortedMap. The first set of methods gives you access to specific keys and entries in your map.

- firstKey(), firstEntry(), lastEntry(), and lastKey(): return the lowest or greatest key or entry from this map.
- ceilingKey(key), ceilingEntry(key), higherKey(key), higherEntry(key): return the lowest key or entry greater than the provided key. The ceiling methods may return a key that is equal to the provided key, whereas the key returned by the higher methods is strictly greater.

- `floorKey(key)`, `floorEntry(key)`, `lowerKey(key)`, `lowerEntry(key)`: return the greatest key or entry lesser than the provided key. The `floor` methods may return a key that is equal to the provided key, whereas the key returned by the `higher` methods is strictly lower.

Accessing your Map with Queue-Like Features

The second set gives you queue-like features:

- `pollFirstEntry()`: returns and removes the lowest entry
- `pollLastEntry()`: returns and removes the greatest entry.

Traversing your Map in the Reverse Order

The third set reverses your map, as if it had been built on the reversed comparison logic.

- `navigableKeySet()` is a convenience method that returns a `NavigableSet` so that you do not have to cast the result of `keySet()`
- `descendingKeySet()`: returns a `NavigableSet` backed by the map, on which you can iterate in the descending order
- `descendingMap()`: returns a `NavigableMap` with the same semantic.

Both views support element removal, but you cannot add anything through them.

Here is an example to demonstrate how you can use them.

```java
NavigableMap<Integer, String> map = new TreeMap<>();
map.put(1, "one");
map.put(2, "two");
map.put(3, "three");
map.put(4, "four");
map.put(5, "five");

map.keySet().forEach(key -> System.out.print(key + " "));
System.out.println();

NavigableSet<Integer> descendingKeys = map.descendingKeySet();
descendingKeys.forEach(key -> System.out.print(key + " "));
```

Running this code prints out the following result.

```
1 2 3 4 5
5 4 3 2 1
```

Getting Submap Views

The last set of methods give you access to views on portions of your map.

- `subMap(fromKey, fromInclusive, toKey, toInclusive)`: returns a submap where you can decide to include or not the boundaries
- `headMap(toKey, inclusive)`: same for the head map
- `tailMap(fromKey, inclusive)`: same for the tail map.

These maps are views on this map, which you can update by removing or adding key/value pairs. There is one restriction on adding elements though: you cannot add keys outside the boundaries on which the view has been created.

Last update: *September 14, 2021*

Handling Map Values with Lambda Expressions ← **Keeping Keys Sorted with SortedMap and NavigableMap** → Choosing Immutable Types for Your Key

Home > Tutorials > The Collections Framework > Keeping Keys Sorted with SortedMap and NavigableMap