java.util

Class TreeMap<K,V>

java.lang.Object
  java.util.AbstractMap<K,V>
    java.util.TreeMap<K,V>

Type Parameters:

`K` - the type of keys maintained by this map

`V` - the type of mapped values

All Implemented Interfaces:

Serializable, Cloneable, Map<K,V>, NavigableMap<K,V>, SortedMap<K,V>

```
public class TreeMap<K,V>
extends AbstractMap<K,V>
implements NavigableMap<K,V>, Cloneable, Serializable
```

A Red-Black tree based `NavigableMap` implementation. The map is sorted according to the natural ordering of its keys, or by a `Comparator` provided at map creation time, depending on which constructor is used.

This implementation provides guaranteed log(n) time cost for the `containsKey`, `get`, `put` and `remove` operations. Algorithms are adaptations of those in Cormen, Leiserson, and Rivest's *Introduction to Algorithms*.

Note that the ordering maintained by a tree map, like any sorted map, and whether or not an explicit comparator is provided, must be *consistent with `equals`* if this sorted map is to correctly implement the `Map` interface. (See `Comparable` or `Comparator` for a precise definition of *consistent with equals*.) This is so because the `Map` interface is defined in terms of the `equals` operation, but a sorted map performs all key comparisons using its `compareTo` (or `compare`) method, so two keys that are deemed equal by this method are, from the standpoint of the sorted map, equal. The behavior of a sorted map *is* well-defined even if its ordering is inconsistent with `equals`; it just fails to obey the general contract of the `Map` interface.

Note that this implementation is not synchronized. If multiple threads access a map concurrently, and at least one of the threads modifies the map structurally, it *must* be synchronized externally. (A structural modification is any operation that adds or deletes one or more mappings; merely changing the value associated with an existing key is not a structural modification.) This is typically accomplished by synchronizing on some object that naturally encapsulates the map. If no such object exists, the map should be "wrapped" using the `Collections.synchronizedSortedMap` method. This is best done at creation time, to prevent accidental unsynchronized access to the map:

```
SortedMap m = Collections.synchronizedSortedMap(new TreeMap(...));
```

The iterators returned by the `iterator` method of the collections returned by all of this class's "collection view methods" are *fail-fast*: if the map is structurally modified at any time after the iterator is created, in any way except through the iterator's own `remove` method, the iterator will throw a `ConcurrentModificationException`. Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

Note that the fail-fast behavior of an iterator cannot be guaranteed as it is, generally speaking, impossible to make any hard guarantees in the presence of unsynchronized concurrent modification. Fail-fast iterators throw `ConcurrentModificationException` on a best-effort basis. Therefore, it would be wrong to write a program that depended on this exception for its correctness: *the fail-fast behavior of iterators should be used only to detect bugs.*

All `Map.Entry` pairs returned by methods in this class and its views represent snapshots of mappings at the time they were produced. They do **not** support the `Entry.setValue` method. (Note however that it is possible to change mappings in the associated map using `put`.)

This class is a member of the Java Collections Framework.

Since:

1.2

See Also:

Map, HashMap, Hashtable, Comparable, Comparator, Collection, Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class java.util.AbstractMap

AbstractMap.SimpleEntry<K,V>, AbstractMap.SimpleImmutableEntry<K,V>

Constructor Summary

Constructors

Constructor and Description
TreeMap() Constructs a new, empty tree map, using the natural ordering of its keys.
TreeMap(**Comparator**<? super K> comparator) Constructs a new, empty tree map, ordered according to the given comparator.
TreeMap(**Map**<? extends K,? extends V> m) Constructs a new tree map containing the same mappings as the given map, ordered according to the *natural ordering* of its keys.
TreeMap(**SortedMap**<K,? extends V> m) Constructs a new tree map containing the same mappings and using the same ordering as the specified sorted map.

Method Summary

Methods

Modifier and Type	Method and Description
Map.Entry<K,V>	**ceilingEntry**(K key) Returns a key-value mapping associated with the least key greater than or equal to the given key, or null if there is no such key.
K	**ceilingKey**(K key) Returns the least key greater than or equal to the given key, or null if there is no such key.
void	**clear**() Removes all of the mappings from this map.
Object	**clone**() Returns a shallow copy of this TreeMap instance.
Comparator<? super K>	**comparator**() Returns the comparator used to order the keys in this map, or null if this map uses the **natural ordering** of its keys.
boolean	**containsKey**(Object key) Returns true if this map contains a mapping for the specified key.
boolean	**containsValue**(Object value) Returns true if this map maps one or more keys to the specified value.
NavigableSet<K>	**descendingKeySet**() Returns a reverse order NavigableSet view of the keys contained in this map.

NavigableMap<K,V>	descendingMap() Returns a reverse order view of the mappings contained in this map.
Set<Map.Entry<K,V>>	entrySet() Returns a Set view of the mappings contained in this map.
Map.Entry<K,V>	firstEntry() Returns a key-value mapping associated with the least key in this map, or null if the map is empty.
K	firstKey() Returns the first (lowest) key currently in this map.
Map.Entry<K,V>	floorEntry(K key) Returns a key-value mapping associated with the greatest key less than or equal to the given key, or null if there is no such key.
K	floorKey(K key) Returns the greatest key less than or equal to the given key, or null if there is no such key.
V	get(Object key) Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
SortedMap<K,V>	headMap(K toKey) Returns a view of the portion of this map whose keys are strictly less than toKey.
NavigableMap<K,V>	headMap(K toKey, boolean inclusive) Returns a view of the portion of this map whose keys are less than (or equal to, if inclusive is true) toKey.
Map.Entry<K,V>	higherEntry(K key) Returns a key-value mapping associated with the least key strictly greater than the given key, or null if there is no such key.
K	higherKey(K key) Returns the least key strictly greater than the given key, or null if there is no such key.
Set<K>	keySet() Returns a Set view of the keys contained in this map.
Map.Entry<K,V>	lastEntry() Returns a key-value mapping associated with the greatest key in this map, or null if the map is empty.
K	lastKey() Returns the last (highest) key currently in this map.
Map.Entry<K,V>	lowerEntry(K key) Returns a key-value mapping associated with the greatest key strictly less than the given key, or null if there is no such key.
K	lowerKey(K key) Returns the greatest key strictly less than the given key, or null if there is no such key.
NavigableSet<K>	navigableKeySet() Returns a NavigableSet view of the keys contained in this map.
Map.Entry<K,V>	pollFirstEntry() Removes and returns a key-value mapping associated with the least key in this map, or null if the map is empty.
Map.Entry<K,V>	pollLastEntry() Removes and returns a key-value mapping associated with the greatest key in this map, or null if the map is empty.
V	put(K key, V value) Associates the specified value with the specified key in this map.
void	putAll(Map<? extends K,? extends V> map) Copies all of the mappings from the specified map to this map.
V	remove(Object key) Removes the mapping for this key from this TreeMap if present.
int	size() Returns the number of key-value mappings in this map.

NavigableMap<K,V>	subMap(K fromKey, boolean fromInclusive, K toKey, boolean toInclusive) Returns a view of the portion of this map whose keys range from fromKey to toKey.
SortedMap<K,V>	subMap(K fromKey, K toKey) Returns a view of the portion of this map whose keys range from fromKey, inclusive, to toKey, exclusive.
SortedMap<K,V>	tailMap(K fromKey) Returns a view of the portion of this map whose keys are greater than or equal to fromKey.
NavigableMap<K,V>	tailMap(K fromKey, boolean inclusive) Returns a view of the portion of this map whose keys are greater than (or equal to, if inclusive is true) fromKey.
Collection<V>	values() Returns a Collection view of the values contained in this map.

Methods inherited from class java.util.AbstractMap

equals, hashCode, isEmpty, toString

Methods inherited from class java.lang.Object

finalize, getClass, notify, notifyAll, wait, wait, wait

Methods inherited from interface java.util.Map

equals, hashCode, isEmpty

Constructor Detail

TreeMap

```
public TreeMap()
```

Constructs a new, empty tree map, using the natural ordering of its keys. All keys inserted into the map must implement the Comparable interface. Furthermore, all such keys must be *mutually comparable*: k1.compareTo(k2) must not throw a ClassCastException for any keys k1 and k2 in the map. If the user attempts to put a key into the map that violates this constraint (for example, the user attempts to put a string key into a map whose keys are integers), the put(Object key, Object value) call will throw a ClassCastException.

TreeMap

```
public TreeMap(Comparator<? super K> comparator)
```

Constructs a new, empty tree map, ordered according to the given comparator. All keys inserted into the map must be *mutually comparable* by the given comparator: comparator.compare(k1, k2) must not throw a ClassCastException for any keys k1 and k2 in the map. If the user attempts to put a key into the map that violates this constraint, the put(Object key, Object value) call will throw a ClassCastException.

Parameters:

comparator - the comparator that will be used to order this map. If null, the natural ordering of the keys will be used.

TreeMap

```
public TreeMap(Map<? extends K,? extends V> m)
```

Constructs a new tree map containing the same mappings as the given map, ordered according to the *natural ordering* of its keys. All keys inserted into the new map must implement the `Comparable` interface. Furthermore, all such keys must be *mutually comparable*: `k1.compareTo(k2)` must not throw a `ClassCastException` for any keys `k1` and `k2` in the map. This method runs in n*log(n) time.

Parameters:

`m` - the map whose mappings are to be placed in this map

Throws:

`ClassCastException` - if the keys in m are not `Comparable`, or are not mutually comparable

`NullPointerException` - if the specified map is null

TreeMap

```
public TreeMap(SortedMap<K,? extends V> m)
```

Constructs a new tree map containing the same mappings and using the same ordering as the specified sorted map. This method runs in linear time.

Parameters:

`m` - the sorted map whose mappings are to be placed in this map, and whose comparator is to be used to sort this map

Throws:

`NullPointerException` - if the specified map is null

Method Detail

size

```
public int size()
```

Returns the number of key-value mappings in this map.

Specified by:

`size` in interface `Map<K,V>`

Overrides:

`size` in class `AbstractMap<K,V>`

Returns:

the number of key-value mappings in this map

containsKey

```
public boolean containsKey(Object key)
```

Returns `true` if this map contains a mapping for the specified key.

Specified by:

`containsKey` in interface `Map<K,V>`

Overrides:

containsKey in class AbstractMap<K,V>

Parameters:

key - key whose presence in this map is to be tested

Returns:

true if this map contains a mapping for the specified key

Throws:

ClassCastException - if the specified key cannot be compared with the keys currently in the map

NullPointerException - if the specified key is null and this map uses natural ordering, or its comparator does not permit null keys

containsValue

```
public boolean containsValue(Object value)
```

Returns true if this map maps one or more keys to the specified value. More formally, returns true if and only if this map contains at least one mapping to a value v such that (value==null ? v==null : value.equals(v)). This operation will probably require time linear in the map size for most implementations.

Specified by:

containsValue in interface Map<K,V>

Overrides:

containsValue in class AbstractMap<K,V>

Parameters:

value - value whose presence in this map is to be tested

Returns:

true if a mapping to value exists; false otherwise

Since:

1.2

get

```
public V get(Object key)
```

Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.

More formally, if this map contains a mapping from a key k to a value v such that key compares equal to k according to the map's ordering, then this method returns v; otherwise it returns null. (There can be at most one such mapping.)

A return value of null does not *necessarily* indicate that the map contains no mapping for the key; it's also possible that the map explicitly maps the key to null. The containsKey operation may be used to distinguish these two cases.

Specified by:

get in interface Map<K,V>

Overrides:

get in class AbstractMap<K,V>

Parameters:

key - the key whose associated value is to be returned

Returns:

the value to which the specified key is mapped, or `null` if this map contains no mapping for the key

Throws:

`ClassCastException` - if the specified key cannot be compared with the keys currently in the map

`NullPointerException` - if the specified key is null and this map uses natural ordering, or its comparator does not permit null keys

comparator

```
public Comparator<? super K> comparator()
```

Description copied from interface: `SortedMap`
Returns the comparator used to order the keys in this map, or `null` if this map uses the natural ordering of its keys.

Specified by:

`comparator` in interface `SortedMap<K,V>`

Returns:

the comparator used to order the keys in this map, or `null` if this map uses the natural ordering of its keys

firstKey

```
public K firstKey()
```

Description copied from interface: `SortedMap`
Returns the first (lowest) key currently in this map.

Specified by:

`firstKey` in interface `SortedMap<K,V>`

Returns:

the first (lowest) key currently in this map

Throws:

`NoSuchElementException` - if this map is empty

lastKey

```
public K lastKey()
```

Description copied from interface: `SortedMap`
Returns the last (highest) key currently in this map.

Specified by:

`lastKey` in interface `SortedMap<K,V>`

Returns:

the last (highest) key currently in this map

Throws:

`NoSuchElementException` - if this map is empty

putAll

```
public void putAll(Map<? extends K,? extends V> map)
```

Copies all of the mappings from the specified map to this map. These mappings replace any mappings that this map had for any of the keys currently in the specified map.

Specified by:

`putAll` in interface `Map<K,V>`

Overrides:

`putAll` in class `AbstractMap<K,V>`

Parameters:

`map` - mappings to be stored in this map

Throws:

`ClassCastException` - if the class of a key or value in the specified map prevents it from being stored in this map

`NullPointerException` - if the specified map is null or the specified map contains a null key and this map does not permit null keys

put

```
public V put(K key,
    V value)
```

Associates the specified value with the specified key in this map. If the map previously contained a mapping for the key, the old value is replaced.

Specified by:

`put` in interface `Map<K,V>`

Overrides:

`put` in class `AbstractMap<K,V>`

Parameters:

`key` - key with which the specified value is to be associated

`value` - value to be associated with the specified key

Returns:

the previous value associated with `key`, or `null` if there was no mapping for `key`. (A `null` return can also indicate that the map previously associated `null` with `key`.)

Throws:

`ClassCastException` - if the specified key cannot be compared with the keys currently in the map

`NullPointerException` - if the specified key is null and this map uses natural ordering, or its comparator does not permit null keys

remove

```
public V remove(Object key)
```

Removes the mapping for this key from this TreeMap if present.

Specified by:

`remove` in interface `Map<K,V>`

Overrides:

`remove` in class `AbstractMap<K,V>`

Parameters:

`key` - key for which mapping should be removed

Returns:

the previous value associated with `key`, or `null` if there was no mapping for `key`. (A `null` return can also indicate that the map previously associated `null` with `key`.)

Throws:

`ClassCastException` - if the specified key cannot be compared with the keys currently in the map

`NullPointerException` - if the specified key is null and this map uses natural ordering, or its comparator does not permit null keys

clear

```
public void clear()
```

Removes all of the mappings from this map. The map will be empty after this call returns.

Specified by:

`clear` in interface `Map<K,V>`

Overrides:

`clear` in class `AbstractMap<K,V>`

clone

```
public Object clone()
```

Returns a shallow copy of this `TreeMap` instance. (The keys and values themselves are not cloned.)

Overrides:

`clone` in class `AbstractMap<K,V>`

Returns:

a shallow copy of this map

See Also:

`Cloneable`

firstEntry

```
public Map.Entry<K,V> firstEntry()
```

Description copied from interface: `NavigableMap`
Returns a key-value mapping associated with the least key in this map, or `null` if the map is empty.

Specified by:

`firstEntry` in interface `NavigableMap<K,V>`

Returns:

an entry with the least key, or `null` if this map is empty

Since:

1.6

lastEntry

```
public Map.Entry<K,V> lastEntry()
```

Description copied from interface: NavigableMap
Returns a key-value mapping associated with the greatest key in this map, or `null` if the map is empty.

Specified by:

`lastEntry` in interface `NavigableMap<K,V>`

Returns:

an entry with the greatest key, or `null` if this map is empty

Since:

1.6

pollFirstEntry

```
public Map.Entry<K,V> pollFirstEntry()
```

Description copied from interface: NavigableMap
Removes and returns a key-value mapping associated with the least key in this map, or `null` if the map is empty.

Specified by:

`pollFirstEntry` in interface `NavigableMap<K,V>`

Returns:

the removed first entry of this map, or `null` if this map is empty

Since:

1.6

pollLastEntry

```
public Map.Entry<K,V> pollLastEntry()
```

Description copied from interface: NavigableMap
Removes and returns a key-value mapping associated with the greatest key in this map, or `null` if the map is empty.

Specified by:

`pollLastEntry` in interface `NavigableMap<K,V>`

Returns:

the removed last entry of this map, or `null` if this map is empty

Since:

1.6

lowerEntry

```
public Map.Entry<K,V> lowerEntry(K key)
```

Description copied from interface: `NavigableMap`
Returns a key-value mapping associated with the greatest key strictly less than the given key, or `null` if there is no such key.

Specified by:

`lowerEntry` in interface `NavigableMap<K,V>`

Parameters:

`key` - the key

Returns:

an entry with the greatest key less than `key`, or `null` if there is no such key

Throws:

`ClassCastException` - if the specified key cannot be compared with the keys currently in the map

`NullPointerException` - if the specified key is null and this map uses natural ordering, or its comparator does not permit null keys

Since:

1.6

lowerKey

```
public K lowerKey(K key)
```

Description copied from interface: `NavigableMap`
Returns the greatest key strictly less than the given key, or `null` if there is no such key.

Specified by:

`lowerKey` in interface `NavigableMap<K,V>`

Parameters:

`key` - the key

Returns:

the greatest key less than `key`, or `null` if there is no such key

Throws:

`ClassCastException` - if the specified key cannot be compared with the keys currently in the map

`NullPointerException` - if the specified key is null and this map uses natural ordering, or its comparator does not permit null keys

Since:

1.6

floorEntry

```
public Map.Entry<K,V> floorEntry(K key)
```

Description copied from interface: `NavigableMap`
Returns a key-value mapping associated with the greatest key less than or equal to the given key, or `null` if there is no such key.

Specified by:

`floorEntry` in interface `NavigableMap<K,V>`

Parameters:

`key` - the key

Returns:

an entry with the greatest key less than or equal to `key`, or `null` if there is no such key

Throws:

`ClassCastException` - if the specified key cannot be compared with the keys currently in the map

`NullPointerException` - if the specified key is null and this map uses natural ordering, or its comparator does not permit null keys

Since:

1.6

floorKey

```
public K floorKey(K key)
```

Description copied from interface: `NavigableMap`
Returns the greatest key less than or equal to the given key, or `null` if there is no such key.

Specified by:

`floorKey` in interface `NavigableMap<K,V>`

Parameters:

`key` - the key

Returns:

the greatest key less than or equal to `key`, or `null` if there is no such key

Throws:

`ClassCastException` - if the specified key cannot be compared with the keys currently in the map

`NullPointerException` - if the specified key is null and this map uses natural ordering, or its comparator does not permit null keys

Since:

1.6

ceilingEntry

```
public Map.Entry<K,V> ceilingEntry(K key)
```

Description copied from interface: `NavigableMap`
Returns a key-value mapping associated with the least key greater than or equal to the given key, or `null` if there is no such key.

Specified by:

`ceilingEntry` in interface `NavigableMap<K,V>`

Parameters:

`key` - the key

Returns:

an entry with the least key greater than or equal to `key`, or `null` if there is no such key

Throws:

`ClassCastException` - if the specified key cannot be compared with the keys currently in the map

`NullPointerException` - if the specified key is null and this map uses natural ordering, or its comparator does not permit null keys

Since:

1.6

ceilingKey

```
public K ceilingKey(K key)
```

Description copied from interface: `NavigableMap`
Returns the least key greater than or equal to the given key, or `null` if there is no such key.

Specified by:

`ceilingKey` in interface `NavigableMap<K,V>`

Parameters:

`key` - the key

Returns:

the least key greater than or equal to `key`, or `null` if there is no such key

Throws:

`ClassCastException` - if the specified key cannot be compared with the keys currently in the map

`NullPointerException` - if the specified key is null and this map uses natural ordering, or its comparator does not permit null keys

Since:

1.6

higherEntry

```
public Map.Entry<K,V> higherEntry(K key)
```

Description copied from interface: `NavigableMap`
Returns a key-value mapping associated with the least key strictly greater than the given key, or `null` if there is no such key.

Specified by:

`higherEntry` in interface `NavigableMap<K,V>`

Parameters:

`key` - the key

Returns:

an entry with the least key greater than `key`, or `null` if there is no such key

Throws:

`ClassCastException` - if the specified key cannot be compared with the keys currently in the map

`NullPointerException` - if the specified key is null and this map uses natural ordering, or its comparator does not permit null keys

Since:

1.6

higherKey

```
public K higherKey(K key)
```

Description copied from interface: `NavigableMap`
Returns the least key strictly greater than the given key, or `null` if there is no such key.

Specified by:

`higherKey` in interface `NavigableMap<K,V>`

Parameters:

`key` - the key

Returns:

the least key greater than `key`, or `null` if there is no such key

Throws:

`ClassCastException` - if the specified key cannot be compared with the keys currently in the map

`NullPointerException` - if the specified key is null and this map uses natural ordering, or its comparator does not permit null keys

Since:

1.6

keySet

```
public Set<K> keySet()
```

Returns a `Set` view of the keys contained in this map. The set's iterator returns the keys in ascending order. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. If the map is modified while an iteration over the set is in progress (except through the iterator's own `remove` operation), the results of the iteration are undefined. The set supports element removal, which removes the corresponding mapping from the map, via the `Iterator.remove`, `Set.remove`, `removeAll`, `retainAll`, and `clear` operations. It does not support the `add` or `addAll` operations.

Specified by:

`keySet` in interface `Map<K,V>`

Specified by:

`keySet` in interface `SortedMap<K,V>`

Overrides:

`keySet` in class `AbstractMap<K,V>`

Returns:

a set view of the keys contained in this map

navigableKeySet

```
public NavigableSet<K> navigableKeySet()
```

Description copied from interface: `NavigableMap`

Returns a `NavigableSet` view of the keys contained in this map. The set's iterator returns the keys in ascending order. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. If the map is modified while an iteration over the set is in progress (except through the iterator's own `remove` operation), the results of the iteration are undefined. The set supports element removal, which removes the corresponding mapping from the map, via the `Iterator.remove`, `Set.remove`, `removeAll`, `retainAll`, and `clear` operations. It does not support the `add` or `addAll` operations.

Specified by:

`navigableKeySet` in interface `NavigableMap<K,V>`

Returns:

a navigable set view of the keys in this map

Since:

1.6

descendingKeySet

```
public NavigableSet<K> descendingKeySet()
```

Description copied from interface: `NavigableMap`

Returns a reverse order `NavigableSet` view of the keys contained in this map. The set's iterator returns the keys in descending order. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. If the map is modified while an iteration over the set is in progress (except through the iterator's own `remove` operation), the results of the iteration are undefined. The set supports element removal, which removes the corresponding mapping from the map, via the `Iterator.remove`, `Set.remove`, `removeAll`, `retainAll`, and `clear` operations. It does not support the `add` or `addAll` operations.

Specified by:

`descendingKeySet` in interface `NavigableMap<K,V>`

Returns:

a reverse order navigable set view of the keys in this map

Since:

1.6

values

```
public Collection<V> values()
```

Returns a `Collection` view of the values contained in this map. The collection's iterator returns the values in ascending order of the corresponding keys. The collection is backed by the map, so changes to the map are reflected in the collection, and vice-versa. If the map is modified while an iteration over the collection is in progress (except through the iterator's own `remove` operation), the results of the iteration are undefined. The collection supports element removal, which removes the corresponding mapping from the map, via the `Iterator.remove`, `Collection.remove`, `removeAll`, `retainAll` and `clear` operations. It does not support the `add` or `addAll` operations.

Specified by:

`values` in interface `Map<K,V>`

Specified by:

`values` in interface `SortedMap<K,V>`

Overrides:

`values` in class `AbstractMap<K,V>`

Returns:

a collection view of the values contained in this map

entrySet

```
public Set<Map.Entry<K,V>> entrySet()
```

Returns a `Set` view of the mappings contained in this map. The set's iterator returns the entries in ascending key order. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. If the map is modified while an iteration over the set is in progress (except through the iterator's own `remove` operation, or through the `setValue` operation on a map entry returned by the iterator) the results of the iteration are undefined. The set supports element removal, which removes the corresponding mapping from the map, via the `Iterator.remove`, `Set.remove`, `removeAll`, `retainAll` and `clear` operations. It does not support the `add` or `addAll` operations.

Specified by:

`entrySet` in interface `Map<K,V>`

Specified by:

`entrySet` in interface `SortedMap<K,V>`

Specified by:

`entrySet` in class `AbstractMap<K,V>`

Returns:

a set view of the mappings contained in this map

descendingMap

```
public NavigableMap<K,V> descendingMap()
```

Description copied from interface: `NavigableMap`
Returns a reverse order view of the mappings contained in this map. The descending map is backed by this map, so changes to the map are reflected in the descending map, and vice-versa. If either map is modified while an iteration over a collection view of either map is in progress (except through the iterator's own `remove` operation), the results of the iteration are undefined.

The returned map has an ordering equivalent to `Collections.reverseOrder(comparator())`. The expression `m.descendingMap().descendingMap()` returns a view of `m` essentially equivalent to `m`.

Specified by:

`descendingMap` in interface `NavigableMap<K,V>`

Returns:

a reverse order view of this map

Since:

1.6

subMap

```
public NavigableMap<K,V> subMap(K fromKey,
                                boolean fromInclusive,
                                K toKey,
                                boolean toInclusive)
```

Description copied from interface: `NavigableMap`
Returns a view of the portion of this map whose keys range from `fromKey` to `toKey`. If `fromKey` and `toKey` are equal, the returned map is empty unless `fromInclusive` and `toInclusive` are both true. The returned map is backed by this map,

so changes in the returned map are reflected in this map, and vice-versa. The returned map supports all optional map operations that this map supports.

The returned map will throw an `IllegalArgumentException` on an attempt to insert a key outside of its range, or to construct a submap either of whose endpoints lie outside its range.

Specified by:

`subMap` in interface `NavigableMap<K,V>`

Parameters:

`fromKey` - low endpoint of the keys in the returned map

`fromInclusive` - `true` if the low endpoint is to be included in the returned view

`toKey` - high endpoint of the keys in the returned map

`toInclusive` - `true` if the high endpoint is to be included in the returned view

Returns:

a view of the portion of this map whose keys range from `fromKey` to `toKey`

Throws:

`ClassCastException` - if `fromKey` and `toKey` cannot be compared to one another using this map's comparator (or, if the map has no comparator, using natural ordering). Implementations may, but are not required to, throw this exception if `fromKey` or `toKey` cannot be compared to keys currently in the map.

`NullPointerException` - if `fromKey` or `toKey` is null and this map uses natural ordering, or its comparator does not permit null keys

`IllegalArgumentException` - if `fromKey` is greater than `toKey`; or if this map itself has a restricted range, and `fromKey` or `toKey` lies outside the bounds of the range

Since:

1.6

headMap

```
public NavigableMap<K,V> headMap(K toKey,
                                 boolean inclusive)
```

Description copied from interface: `NavigableMap`

Returns a view of the portion of this map whose keys are less than (or equal to, if `inclusive` is true) `toKey`. The returned map is backed by this map, so changes in the returned map are reflected in this map, and vice-versa. The returned map supports all optional map operations that this map supports.

The returned map will throw an `IllegalArgumentException` on an attempt to insert a key outside its range.

Specified by:

`headMap` in interface `NavigableMap<K,V>`

Parameters:

`toKey` - high endpoint of the keys in the returned map

`inclusive` - `true` if the high endpoint is to be included in the returned view

Returns:

a view of the portion of this map whose keys are less than (or equal to, if `inclusive` is true) `toKey`

Throws:

`ClassCastException` - if `toKey` is not compatible with this map's comparator (or, if the map has no comparator, if `toKey` does not implement `Comparable`). Implementations may, but are not required to, throw this exception if `toKey` cannot be compared to keys currently in the map.

`NullPointerException` - if `toKey` is null and this map uses natural ordering, or its comparator does not permit null keys

`IllegalArgumentException` - if this map itself has a restricted range, and `toKey` lies outside the bounds of the range

Since:

1.6

tailMap

```
public NavigableMap<K,V> tailMap(K fromKey,
                                 boolean inclusive)
```

Description copied from interface: `NavigableMap`

Returns a view of the portion of this map whose keys are greater than (or equal to, if `inclusive` is true) `fromKey`. The returned map is backed by this map, so changes in the returned map are reflected in this map, and vice-versa. The returned map supports all optional map operations that this map supports.

The returned map will throw an `IllegalArgumentException` on an attempt to insert a key outside its range.

Specified by:

`tailMap` in interface `NavigableMap<K,V>`

Parameters:

`fromKey` - low endpoint of the keys in the returned map

`inclusive` - `true` if the low endpoint is to be included in the returned view

Returns:

a view of the portion of this map whose keys are greater than (or equal to, if `inclusive` is true) `fromKey`

Throws:

`ClassCastException` - if `fromKey` is not compatible with this map's comparator (or, if the map has no comparator, if `fromKey` does not implement `Comparable`). Implementations may, but are not required to, throw this exception if `fromKey` cannot be compared to keys currently in the map.

`NullPointerException` - if `fromKey` is null and this map uses natural ordering, or its comparator does not permit null keys

`IllegalArgumentException` - if this map itself has a restricted range, and `fromKey` lies outside the bounds of the range

Since:

1.6

subMap

```
public SortedMap<K,V> subMap(K fromKey,
                             K toKey)
```

Description copied from interface: `NavigableMap`

Returns a view of the portion of this map whose keys range from `fromKey`, inclusive, to `toKey`, exclusive. (If `fromKey` and `toKey` are equal, the returned map is empty.) The returned map is backed by this map, so changes in the returned map are reflected in this map, and vice-versa. The returned map supports all optional map operations that this map supports.

The returned map will throw an `IllegalArgumentException` on an attempt to insert a key outside its range.

Equivalent to `subMap(fromKey, true, toKey, false)`.

Specified by:

`subMap` in interface `NavigableMap<K,V>`

Specified by:

`subMap` in interface `SortedMap<K,V>`

Parameters:

`fromKey` - low endpoint (inclusive) of the keys in the returned map

`toKey` - high endpoint (exclusive) of the keys in the returned map

Returns:

a view of the portion of this map whose keys range from `fromKey`, inclusive, to `toKey`, exclusive

Throws:

`ClassCastException` - if `fromKey` and `toKey` cannot be compared to one another using this map's comparator (or, if the map has no comparator, using natural ordering). Implementations may, but are not required to, throw this exception if `fromKey` or `toKey` cannot be compared to keys currently in the map.

`NullPointerException` - if `fromKey` or `toKey` is null and this map uses natural ordering, or its comparator does not permit null keys

`IllegalArgumentException` - if `fromKey` is greater than `toKey`; or if this map itself has a restricted range, and `fromKey` or `toKey` lies outside the bounds of the range

headMap

```
public SortedMap<K,V> headMap(K toKey)
```

Description copied from interface: `NavigableMap`

Returns a view of the portion of this map whose keys are strictly less than `toKey`. The returned map is backed by this map, so changes in the returned map are reflected in this map, and vice-versa. The returned map supports all optional map operations that this map supports.

The returned map will throw an `IllegalArgumentException` on an attempt to insert a key outside its range.

Equivalent to `headMap(toKey, false)`.

Specified by:

`headMap` in interface `NavigableMap<K,V>`

Specified by:

`headMap` in interface `SortedMap<K,V>`

Parameters:

`toKey` - high endpoint (exclusive) of the keys in the returned map

Returns:

a view of the portion of this map whose keys are strictly less than `toKey`

Throws:

`ClassCastException` - if `toKey` is not compatible with this map's comparator (or, if the map has no comparator, if `toKey` does not implement `Comparable`). Implementations may, but are not required to, throw this exception if `toKey` cannot be compared to keys currently in the map.

`NullPointerException` - if `toKey` is null and this map uses natural ordering, or its comparator does not permit null keys

`IllegalArgumentException` - if this map itself has a restricted range, and `toKey` lies outside the bounds of the range

tailMap

```
public SortedMap<K,V> tailMap(K fromKey)
```

Description copied from interface: `NavigableMap`

Returns a view of the portion of this map whose keys are greater than or equal to `fromKey`. The returned map is backed by this map, so changes in the returned map are reflected in this map, and vice-versa. The returned map supports all optional map operations that this map supports.

The returned map will throw an `IllegalArgumentException` on an attempt to insert a key outside its range.

Equivalent to `tailMap(fromKey, true)`.

Specified by:

`tailMap` in interface `NavigableMap<K,V>`

Specified by:

`tailMap` in interface `SortedMap<K,V>`

Parameters:

`fromKey` - low endpoint (inclusive) of the keys in the returned map

Returns:

a view of the portion of this map whose keys are greater than or equal to `fromKey`

Throws:

`ClassCastException` - if `fromKey` is not compatible with this map's comparator (or, if the map has no comparator, if `fromKey` does not implement `Comparable`). Implementations may, but are not required to, throw this exception if `fromKey` cannot be compared to keys currently in the map.

`NullPointerException` - if `fromKey` is null and this map uses natural ordering, or its comparator does not permit null keys

`IllegalArgumentException` - if this map itself has a restricted range, and `fromKey` lies outside the bounds of the range