compact1, compact2, compact3

java.util

Class HashMap<K,V>

java.lang.Object
    java.util.AbstractMap<K,V>
        java.util.HashMap<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>`

Direct Known Subclasses:

`LinkedHashMap, PrinterStateReasons`

```
public class HashMap<K,V>
extends AbstractMap<K,V>
implements Map<K,V>, Cloneable, Serializable
```

Hash table based implementation of the `Map` interface. This implementation provides all of the optional map operations, and permits `null` values and the `null` key. (The `HashMap` class is roughly equivalent to `Hashtable`, except that it is unsynchronized and permits nulls.) This class makes no guarantees as to the order of the map; in particular, it does not guarantee that the order will remain constant over time.

This implementation provides constant-time performance for the basic operations (`get` and `put`), assuming the hash function disperses the elements properly among the buckets. Iteration over collection views requires time proportional to the "capacity" of the `HashMap` instance (the number of buckets) plus its size (the number of key-value mappings). Thus, it's very important not to set the initial capacity too high (or the load factor too low) if iteration performance is important.

An instance of `HashMap` has two parameters that affect its performance: *initial capacity* and *load factor*. The *capacity* is the number of buckets in the hash table, and the initial capacity is simply the capacity at the time the hash table is created. The *load factor* is a measure of how full the hash table is allowed to get before its capacity is automatically increased. When the number of entries in the hash table exceeds the product of the load factor and the current capacity, the hash table is *rehashed* (that is, internal data structures are rebuilt) so that the hash table has approximately twice the number of buckets.

As a general rule, the default load factor (.75) offers a good tradeoff between time and space costs. Higher values decrease the space overhead but increase the lookup cost (reflected in most of the operations of the `HashMap` class, including `get` and `put`). The expected number of entries in the map and its load factor should be taken into account when setting its initial capacity, so as to minimize the

number of rehash operations. If the initial capacity is greater than the maximum number of entries divided by the load factor, no rehash operations will ever occur.

If many mappings are to be stored in a `HashMap` instance, creating it with a sufficiently large capacity will allow the mappings to be stored more efficiently than letting it perform automatic rehashing as needed to grow the table. Note that using many keys with the same `hashCode()` is a sure way to slow down performance of any hash table. To ameliorate impact, when keys are `Comparable`, this class may use comparison order among keys to help break ties.

Note that this implementation is not synchronized. If multiple threads access a hash 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 a key that an instance already contains 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.synchronizedMap` method. This is best done at creation time, to prevent accidental unsynchronized access to the map:

```
Map m = Collections.synchronizedMap(new HashMap(...));
```

The iterators 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.*

This class is a member of the Java Collections Framework.

Since:

`1.2`

See Also:

`Object.hashCode()`, `Collection`, `Map`, `TreeMap`, `Hashtable`, `Serialized Form`

Nested Class Summary

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

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

Nested classes/interfaces inherited from interface java.util.Map

`Map.Entry<K,V>`

Constructor Summary

Constructors

Constructor and Description
HashMap() Constructs an empty HashMap with the default initial capacity (16) and the default load factor (0.75).
HashMap(int initialCapacity) Constructs an empty HashMap with the specified initial capacity and the default load factor (0.75).
HashMap(int initialCapacity, float loadFactor) Constructs an empty HashMap with the specified initial capacity and load factor.
HashMap(Map<? extends K,? extends V> m) Constructs a new HashMap with the same mappings as the specified Map.

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
void	**clear**() Removes all of the mappings from this map.
Object	**clone**() Returns a shallow copy of this HashMap instance: the keys and values themselves are not cloned.
V	**compute**(**K** key, **BiFunction**<? super **K**,? super **V**,? extends **V**> remappingFunction) Attempts to compute a mapping for the specified key and its current mapped value (or null if there is no current mapping).
V	**computeIfAbsent**(**K** key, **Function**<? super **K**,? extends **V**> mappingFunction) If the specified key is not already associated with a value (or is mapped to null), attempts to compute its value using the given mapping function and enters it into this map unless null.
V	**computeIfPresent**(**K** key, **BiFunction**<? super **K**,? super **V**,? extends **V**> remappingFunction) If the value for the specified key is present and non-null, attempts to compute a new mapping given the key and its current mapped value.
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.

`Set<Map.Entry<K,V>>`	**`entrySet`**`()` Returns a **`Set`** view of the mappings contained in this map.
`void`	**`forEach`**`(`**`BiConsumer`**`<? super K,? super V> action)` Performs the given action for each entry in this map until all entries have been processed or the action throws an exception.
`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.
`V`	**`getOrDefault`**`(`**`Object`**` key, `**`V`**` defaultValue)` Returns the value to which the specified key is mapped, or `defaultValue` if this map contains no mapping for the key.
`boolean`	**`isEmpty`**`()` Returns `true` if this map contains no key-value mappings.
`Set<K>`	**`keySet`**`()` Returns a **`Set`** view of the keys contained in this map.
`V`	**`merge`**`(`**`K`**` key, `**`V`**` value, `**`BiFunction`**`<? super V,? super V,? extends V> remappingFunction)` If the specified key is not already associated with a value or is associated with null, associates it with the given non-null value.
`V`	**`put`**`(`**`K`**` key, `**`V`**` value)` Associates the specified value with the specified key in this map.
`void`	**`putAll`**`(`**`Map`**`<? extends K,? extends V> m)` Copies all of the mappings from the specified map to this map.
`V`	**`putIfAbsent`**`(`**`K`**` key, `**`V`**` value)` If the specified key is not already associated with a value (or is mapped to `null`) associates it with the given value and returns `null`, else returns the current value.
`V`	**`remove`**`(`**`Object`**` key)` Removes the mapping for the specified key from this map if present.
`boolean`	**`remove`**`(`**`Object`**` key, `**`Object`**` value)` Removes the entry for the specified key only if it is currently mapped to the specified value.
`V`	**`replace`**`(`**`K`**` key, `**`V`**` value)` Replaces the entry for the specified key only if it is currently mapped to some value.
`boolean`	**`replace`**`(`**`K`**` key, `**`V`**` oldValue, `**`V`**` newValue)` Replaces the entry for the specified key only if currently mapped to the specified value.
`void`	**`replaceAll`**`(`**`BiFunction`**`<? super K,? super V,? extends V> function)`

	Replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception.
`int`	**`size`**`()` Returns the number of key-value mappings in this map.
`Collection<V>`	**`values`**`()` Returns a **`Collection`** view of the values contained in this map.

Methods inherited from class java.util.AbstractMap

`equals, hashCode, toString`

Methods inherited from class java.lang.Object

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

Methods inherited from interface java.util.Map

`equals, hashCode`

Constructor Detail

HashMap

```
public HashMap(int initialCapacity,
               float loadFactor)
```

Constructs an empty `HashMap` with the specified initial capacity and load factor.

Parameters:

`initialCapacity - the initial capacity`

`loadFactor - the load factor`

Throws:

`IllegalArgumentException - if the initial capacity is negative or the load factor is nonpositive`

HashMap

```
public HashMap(int initialCapacity)
```

Constructs an empty `HashMap` with the specified initial capacity and the default load factor (0.75).

Parameters:

`initialCapacity - the initial capacity.`

Throws:

`IllegalArgumentException` - if the initial capacity is negative.

HashMap

```
public HashMap()
```

Constructs an empty `HashMap` with the default initial capacity (16) and the default load factor (0.75).

HashMap

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

Constructs a new `HashMap` with the same mappings as the specified `Map`. The `HashMap` is created with default load factor (0.75) and an initial capacity sufficient to hold the mappings in the specified `Map`.

Parameters:

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

isEmpty

```
public boolean isEmpty()
```

Returns `true` if this map contains no key-value mappings.

Specified by:

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

Overrides:

isEmpty in class AbstractMap<K,V>

Returns:

true if this map contains no key-value mappings

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==null ? k==null : key.equals(k)), 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

See Also:

put(Object, Object)

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

Returns:

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

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.)`

putAll

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

Copies all of the mappings from the specified map to this map. These mappings will 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:

`m - mappings to be stored in this map`

Throws:

`NullPointerException - if the specified map is null`

remove

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

Removes the mapping for the specified key from this map if present.

Specified by:

`remove in interface Map<K,V>`

Overrides:

`remove in class AbstractMap<K,V>`

Parameters:

key - key whose mapping is to be removed from the map

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.)

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>

containsValue

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

Returns true if this map maps one or more keys to the specified value.

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 this map maps one or more keys to the specified value

keySet

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

Returns a Set view of the keys contained in this map. 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>

Overrides:

keySet in class AbstractMap<K,V>

Returns:

a set view of the keys contained in this map

values

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

Returns a `Collection` view of the values contained in this map. 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>`

Overrides:

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

Returns:

a 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 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 class `AbstractMap<K,V>`

Returns:

a set view of the mappings contained in this map

getOrDefault

```
public V getOrDefault(Object key,
                      V defaultValue)
```

Description copied from interface: `Map`

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

Specified by:

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

Parameters:

`key` - the key whose associated value is to be returned

`defaultValue` - the default mapping of the key

Returns:

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

putIfAbsent

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

Description copied from interface: `Map`

If the specified key is not already associated with a value (or is mapped to `null`) associates it with the given value and returns `null`, else returns the current value.

Specified by:

`putIfAbsent` in interface `Map<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 the specified key, or null if there was no mapping for the key. (A null return can also indicate that the map previously associated null with the key, if the implementation supports null values.)

remove

```
public boolean remove(Object key,
                      Object value)
```

Description copied from interface: `Map`

Removes the entry for the specified key only if it is currently mapped to the specified value.

Specified by:

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

Parameters:

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

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

Returns:

true if the value was removed

replace

```
public boolean replace(K key,
                       V oldValue,
                       V newValue)
```

Description copied from interface: Map

Replaces the entry for the specified key only if currently mapped to the specified value.

Specified by:

replace in interface Map<K,V>

Parameters:

key - key with which the specified value is associated

oldValue - value expected to be associated with the specified key

newValue - value to be associated with the specified key

Returns:

true if the value was replaced

replace

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

Description copied from interface: Map

Replaces the entry for the specified key only if it is currently mapped to some value.

Specified by:

replace in interface Map<K,V>

Parameters:

key - key with which the specified value is associated

value - value to be associated with the specified key

Returns:

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

computeIfAbsent

```
public V computeIfAbsent(K key,
                         Function<? super K,? extends V> mappingFunction)
```

Description copied from interface: Map

If the specified key is not already associated with a value (or is mapped to null), attempts to compute its value using the given mapping function and enters it into this map unless null.

If the function returns `null` no mapping is recorded. If the function itself throws an (unchecked) exception, the exception is rethrown, and no mapping is recorded. The most common usage is to construct a new object serving as an initial mapped value or memoized result, as in:

```
map.computeIfAbsent(key, k -> new Value(f(k)));
```

Or to implement a multi-value map, `Map<K,Collection<V>>`, supporting multiple values per key:

```
map.computeIfAbsent(key, k -> new HashSet<V>()).add(v);
```

Specified by:

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

Parameters:

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

`mappingFunction` - the function to compute a value

Returns:

the current (existing or computed) value associated with the specified key, or null if the computed value is null

computeIfPresent

```
public V computeIfPresent(K key,
                          BiFunction<? super K,? super V,? extends V> remappingFunction)
```

Description copied from interface: `Map`

If the value for the specified key is present and non-null, attempts to compute a new mapping given the key and its current mapped value.

If the function returns `null`, the mapping is removed. If the function itself throws an (unchecked) exception, the exception is rethrown, and the current mapping is left unchanged.

Specified by:

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

Parameters:

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

`remappingFunction` - the function to compute a value

Returns:

the new value associated with the specified key, or null if none

compute

```
public V compute(K key,
                 BiFunction<? super K,? super V,? extends V> remappingFunction)
```

Description copied from interface: `Map`

Attempts to compute a mapping for the specified key and its current mapped value (or `null` if there is no current mapping). For example, to either create or append a `String` msg to a value mapping:

```
map.compute(key, (k, v) -> (v == null) ? msg : v.concat(msg))
```

(Method `merge()` is often simpler to use for such purposes.)

If the function returns `null`, the mapping is removed (or remains absent if initially absent). If the function itself throws an (unchecked) exception, the exception is rethrown, and the current mapping is left unchanged.

Specified by:

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

Parameters:

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

`remappingFunction - the function to compute a value`

Returns:

`the new value associated with the specified key, or null if none`

merge

```
public V merge(K key,
               V value,
               BiFunction<? super V,? super V,? extends V> remappingFunction)
```

Description copied from interface: `Map`

If the specified key is not already associated with a value or is associated with null, associates it with the given non-null value. Otherwise, replaces the associated value with the results of the given remapping function, or removes if the result is `null`. This method may be of use when combining multiple mapped values for a key. For example, to either create or append a `String` `msg` to a value mapping:

```
map.merge(key, msg, String::concat)
```

If the function returns `null` the mapping is removed. If the function itself throws an (unchecked) exception, the exception is rethrown, and the current mapping is left unchanged.

Specified by:

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

Parameters:

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

`value - the non-null value to be merged with the existing value associated with the key or, if no existing value or a null value is associated with the key, to be associated with the key`

`remappingFunction - the function to recompute a value if present`

Returns:

the new value associated with the specified key, or null if no value is associated with the key

forEach

```
public void forEach(BiConsumer<? super K,? super V> action)
```

Description copied from interface: Map

Performs the given action for each entry in this map until all entries have been processed or the action throws an exception. Unless otherwise specified by the implementing class, actions are performed in the order of entry set iteration (if an iteration order is specified.) Exceptions thrown by the action are relayed to the caller.

Specified by:

forEach in interface Map<K,V>

Parameters:

action - The action to be performed for each entry

replaceAll

```
public void replaceAll(BiFunction<? super K,? super V,? extends V> function)
```

Description copied from interface: Map

Replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception. Exceptions thrown by the function are relayed to the caller.

Specified by:

replaceAll in interface Map<K,V>

Parameters:

function - the function to apply to each entry

clone

```
public Object clone()
```

Returns a shallow copy of this HashMap 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

Submit a bug or feature

For further API reference and developer documentation, see Java SE Documentation. That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.